

**CSE2215 - Computer Graphics**

# **Introduction to CG Painting by Numbers**

Elmar Eisemann

Delft University of Technology



# Welcome!



**Elmar Eisemann**



**Ricardo Marroquim**



**Mathijs Molenaar**



**Martin Skrodzki**  
**(Linear Algebra Recap)**



**Michael Weinmann**  
**(helps with project)**

Many TAs in the background who you will encounter during the practical sessions!

# Introduction

- Computer Graphics
  - part of computer science

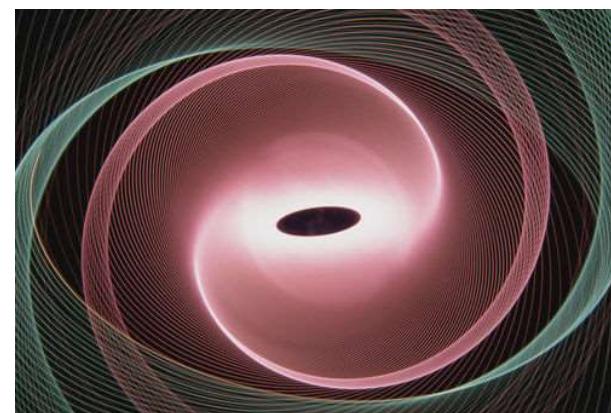


Manipulation, creation, and display of  
visual and geometric information  
with a computer

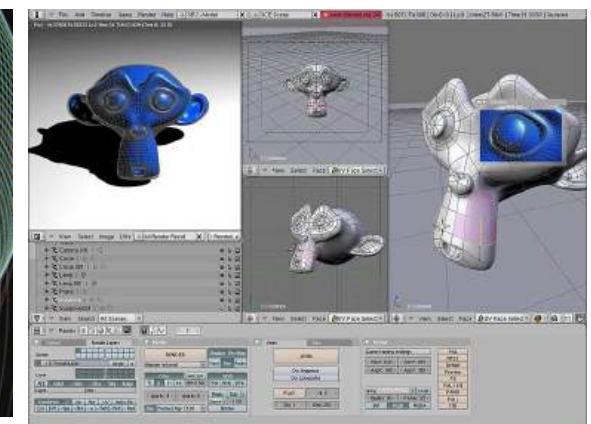
**Not:** using Photoshop

**Instead:** making Photoshop

First hit on Google in 2012...  
don't ask! ;)

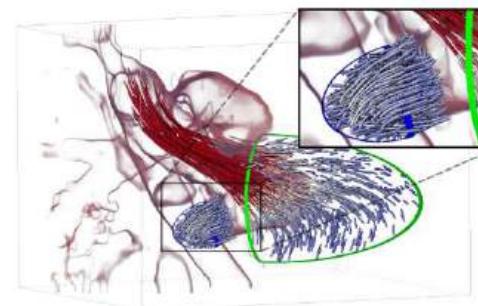
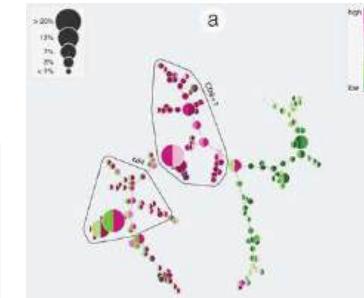


Hit on Google in 2023...  
voila! ;)



# Computer Graphics and Visualization

- CG impacts many domains
  - Big Data Analysis
  - Scientific Visualization
  - Architecture/Design
  - Education
  - Entertainment
  - ...



# Computer Graphics and Visualization



# Computer Graphics and Visualization



# Computer Graphics and Visualization



## What do we cover in this course?

- Key Concepts and basics of CG
- Images
- Geometry
- Transformations/Animation
- Materials and Light
- Textures
- Shadows



# CSE2215 Computer Graphics

- Key Concepts
- Images
- Geometry
- Transformations
- Materials and Light
- Textures
- Shadows
- Advanced Ray Tracing
- Data Structures

relevant for midterm

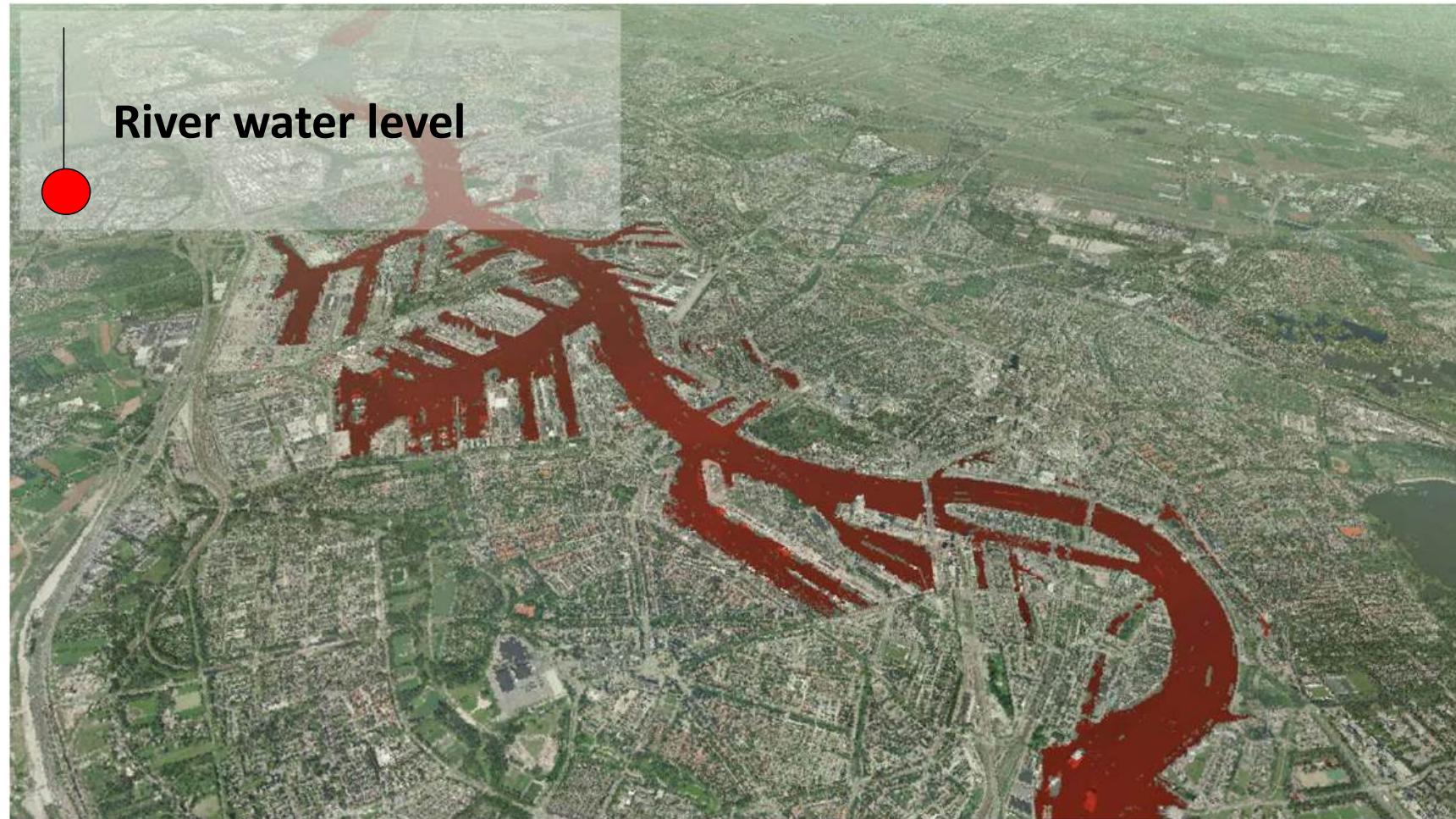


relevant for project

# Large-Scale Visualization



## Models



# Creating the “Omniverse” for Machine Learning

- NVIDIA's vision for future machine learning

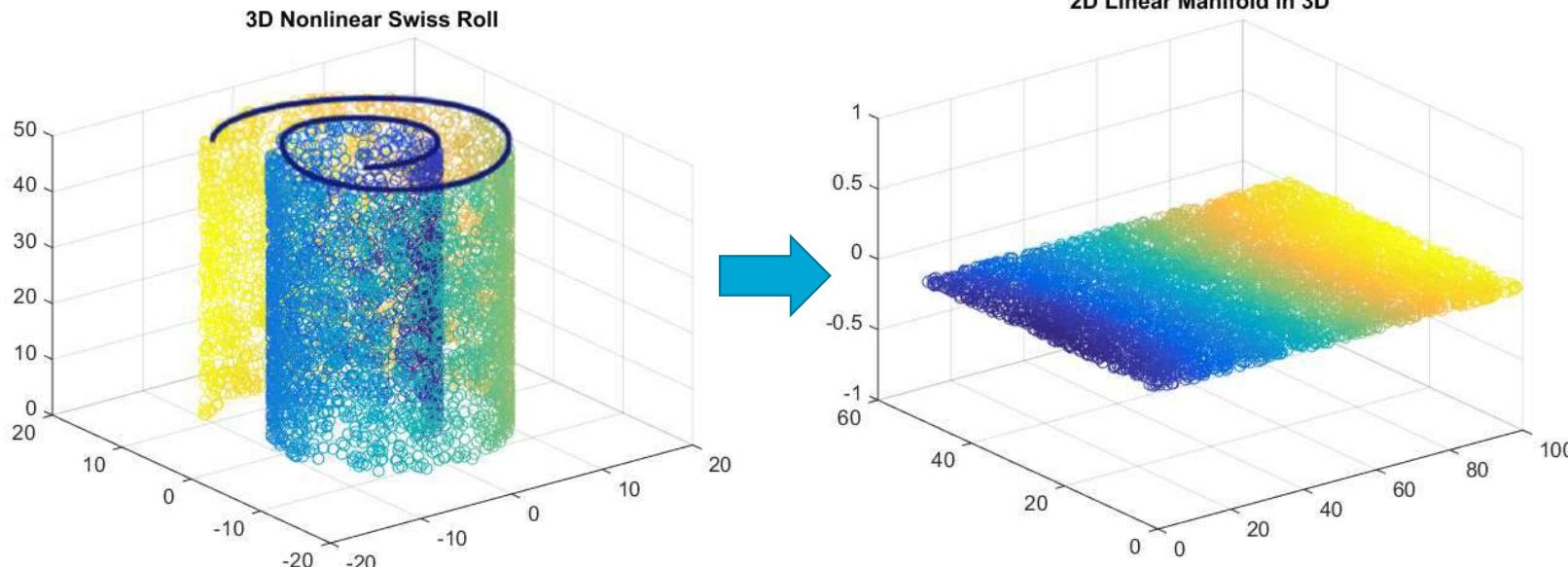




15

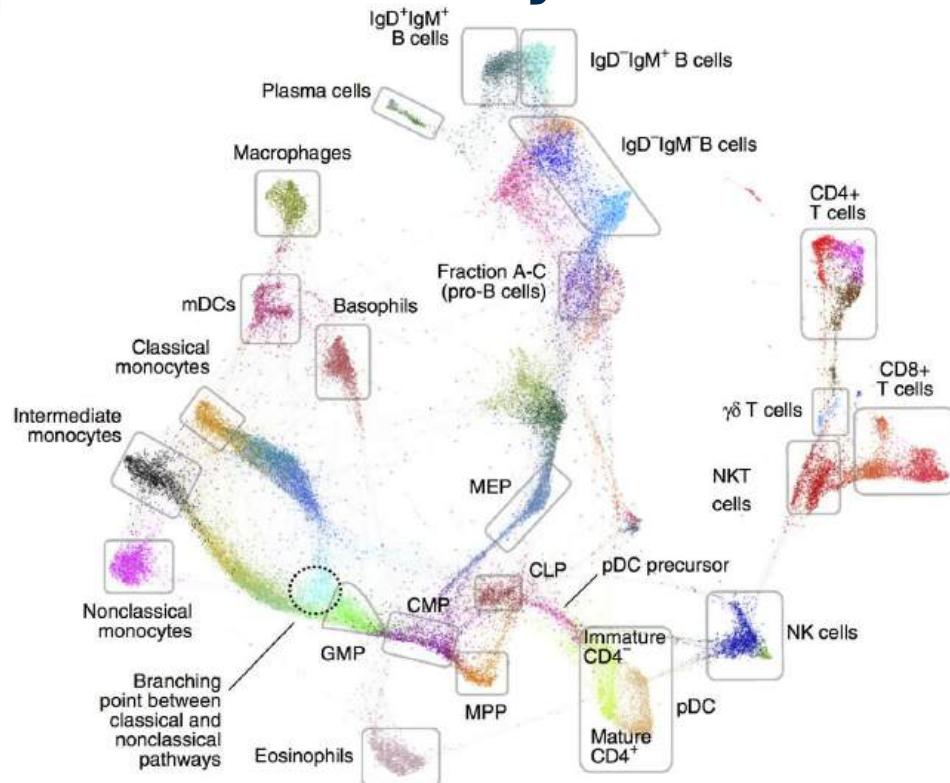


# Acceleration Structures for Dimensionality Reduction



- Original t-SNE:  $O(n^2)$                           2 days(!)
- Barnes-Hut SNE:  $O(n \log(n))$                   8 min
- Linear t-SNE:  $O(n)$                                 10 sec.
- Dual Hierarchy t-SNE:  $O(n)$                         < 2 sec.

# Dimensionality Reduction in the Medical Domain

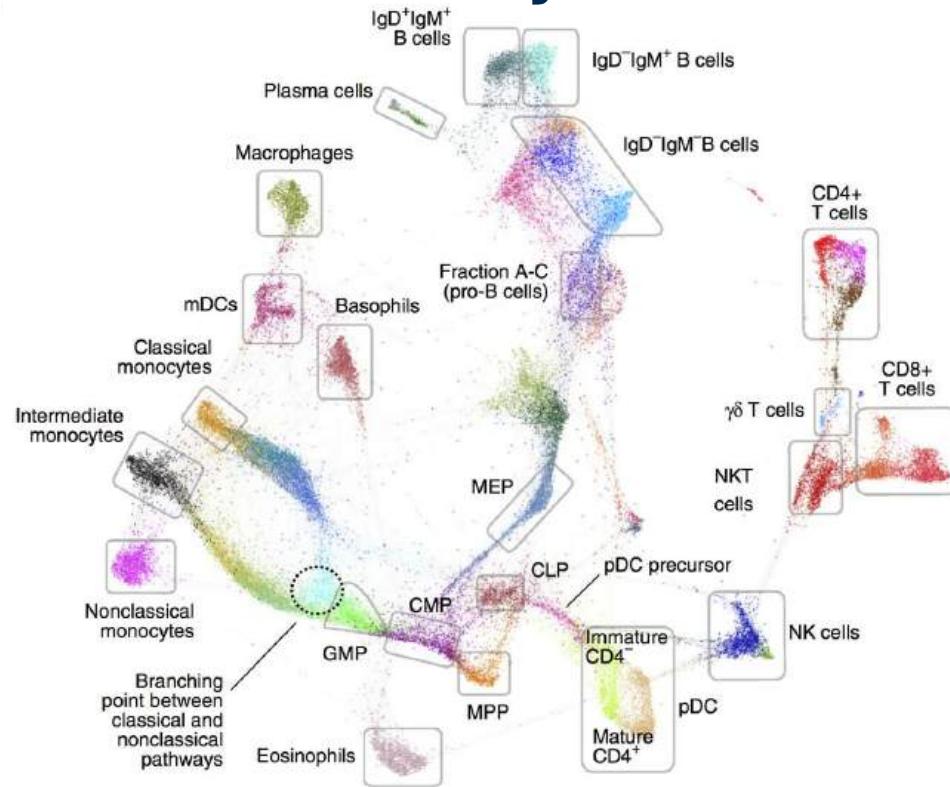


Samusik et al. Nature Methods, 2016

**22 hours**  
30k cells visualized

[van Unen, Höllt, Pezzotti, Li, Reinders, Eisemann, Koning, Vilanova, Lelieveldt: Nature Communications 2017]

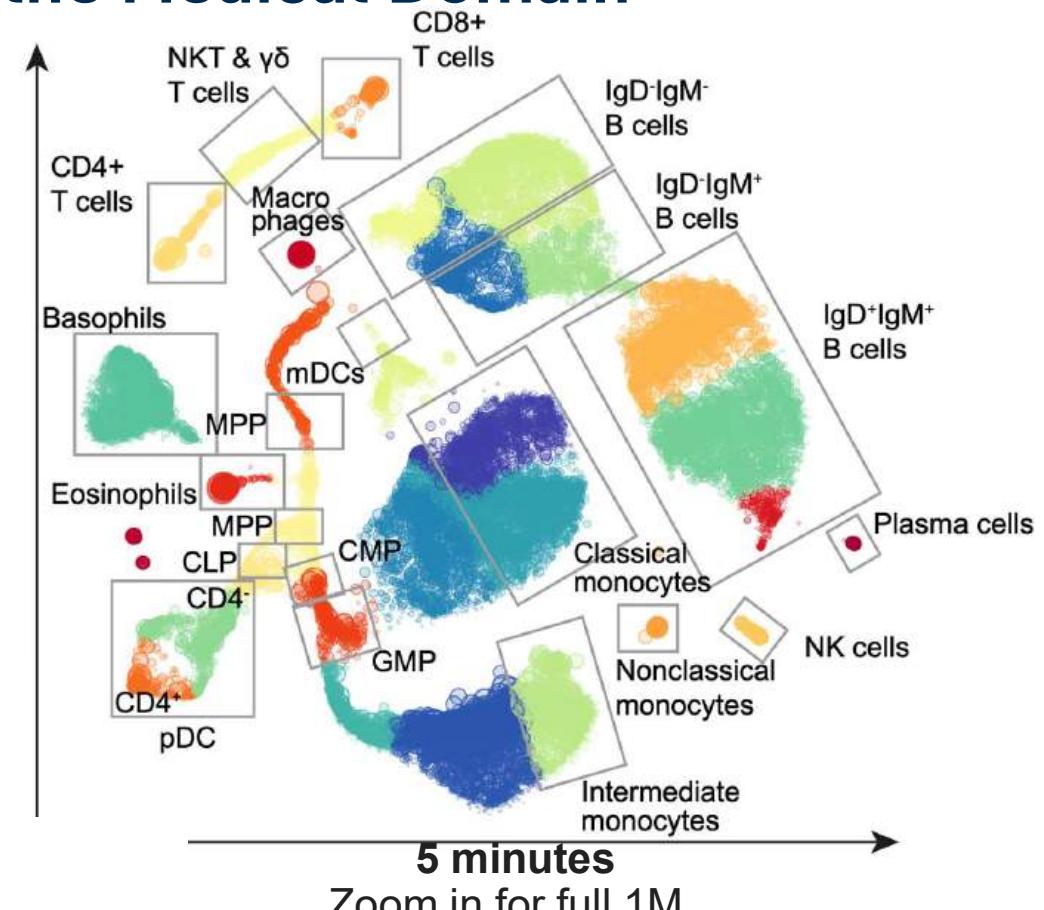
# Dimensionality Reduction in the Medical Domain



Samusik et al. Nature Methods, 2016

**22 hours**  
30k cells visualized

[van Unen, Höllt, Pezzotti, Li, Reinders, Eisemann, Koning, Vilanova, Lelieveldt: Nature Communications 2017]



**5 minutes**  
Zoom in for full 1M

## CSE2215

- The course provides basic (!) knowledge to tackle such challenges.



- An image says more than a thousand words...

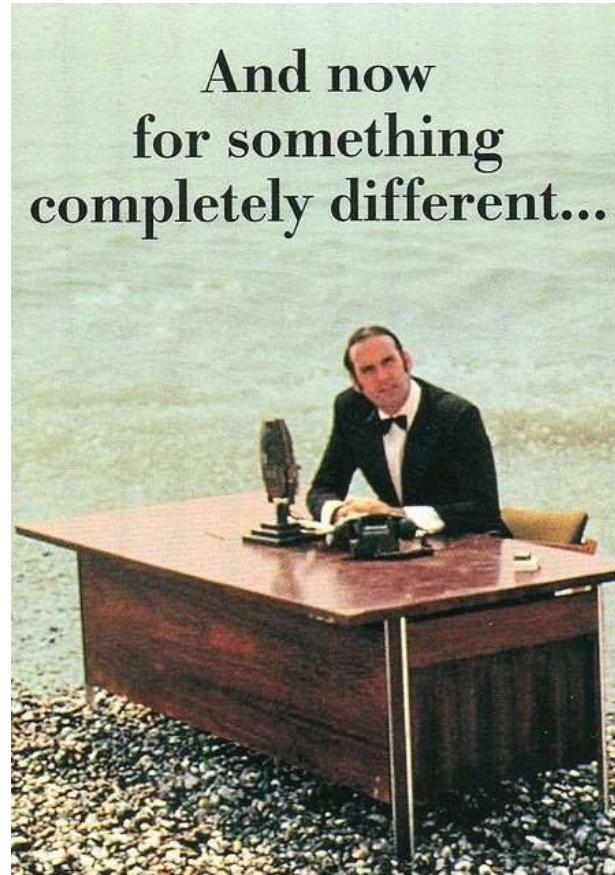
# CSE2215



# CSE2215 Study Goals

- S1- Explain and compare the structure and properties of standard algorithms and data structures linked to Computer Graphics.
- S2- Execute and visualize standard algorithms and data structures.
- S3- Use mathematical methods to analyze, create, apply algorithms and data structures, as well as understanding time and space complexity of image-generation algorithms
- S4- Apply mathematical modeling and theory of geometric computations and transformations, object representations, simulation, and encoding.
- S5- Implement algorithms and data structures using the C++ programming language and OpenGL.
- S6- Apply the knowledge obtained in this course to problems of other fields

## At the moment of suspense...



And now  
for something  
completely different...

Administration...



# CS2215 Computer Graphics

- How to test my knowledge?
- Theory:
  - Exam
  - Use in projects/assignment
- Practical Skills:
  - Tutorials
  - C++ and OpenGL
  - Assignments
  - Projects

# CS2215 Computer Graphics

- **Expected prior knowledge**
  - Object-oriented programming;
  - Algorithms and Data Structures;
  - Linear Algebra;
  - Calculus

# CS2215 Computer Graphics

- **Expected prior knowledge**
  - Object-oriented programming;
  - Algorithms and Data Structures;
  - Linear Algebra;
  - Calculus

# Assessment

- **Practical Grade** (at least 5.0)
  - 5 Assignments (40% of practical grade)
    - referred to as **Assignment Score**  
(individual, first 3 determine project groups)
  - Final Project (60% of practical grade)
    - referred to as **Project Score**  
(group project with individual grades)
- **Exam Grade** (at least 5.0)
- **Final Grade** (at least 6.0)
  - $(\text{Practical Grade} + \text{Exam Grade}) / 2$

## Student Example

- Student A:
  - Scores all 5 Assignments with a 2.5
  - Scores final project with a 10
  - Scores exam with a 4
  - Practical Grade:  $(0.4 * 2.5 + 0.6 * 10) = 1+6 = 7$ 
    - 5 Assignments (40% of practical grade)
    - Final Project (60% of practical grade)
  - Final Grade:  $(\text{Practical} + \text{Exam})/2$ , thus  $(7+6)/2 = 6.5$

# Attention!



# Assessment

- **Practical Grade (at least 5.0)**
  - 5 Assignments (40% of practical grade)
    - referred to as **Assignment Score**  
(individual, first 3 determine project groups)
    - Final Project (60% of practical grade)
      - referred to as **Project Score**  
(group project with individual grades)
  - **Exam Grade (at least 5.0)**
  - **Final Grade (at least 6.0)**
    - $(\text{Practical Grade} + \text{Exam Grade}) / 2$

## Student Examples

- Student A:
  - Scores all 5 Assignments with a 2.5
  - Scores final project with a 10
  - Scores exam with a 6      >5?
  - Practical Grade:  $(0.4 * 2.5 + 0.6 * 10) = 1+6 = 7$       >5?
    - 5 Assignments (40% of practical grade)
    - Final Project (60% of practical grade)
  - Final Grade:  $(\text{Practical} + \text{Exam})/2$ , thus  $7+6/2=6.5$       >6?



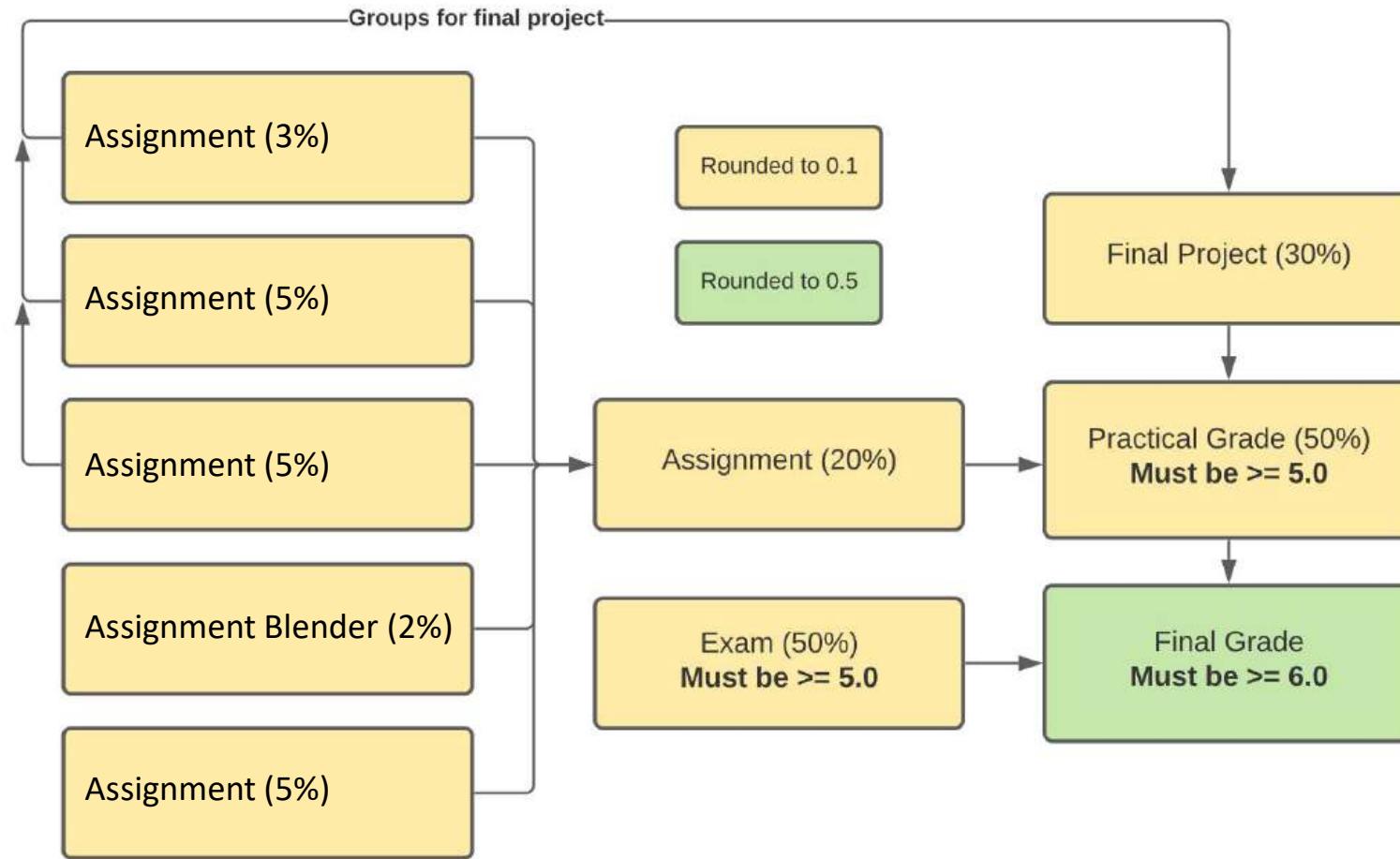
## What to do, if you do not pass?

- **Exam resit:** Open to everyone
- **The repair of the Assignment score** takes form of a multi-component assignment and can involve a computer assessment.
- **The repair of the final project score** implies
  - at least 6.0 on the midterm exam (not resit)
  - at least 6.0 in the assignments (before any repairs)it will take the form of an individual assignment.



Repairs/Resits are expected to take place in the following quarter.  
Following TU Delft regulations, a repair can maximally lead to a 6.0

# Assessment Overview – also available on Brightspace



## How is your work evaluated?

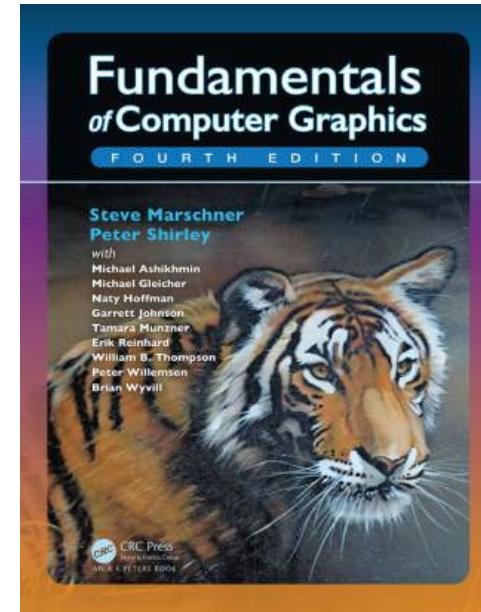
- Assignments
  - Automatic Correction and Feedback
- Exam
  - Mix of multiple choice (MC) and open questions
  - Grading of MC questions: wrong answer=0, skip=1, correct=2
    - If you cannot find your answer, skip to get partial points for your attempt
    - Do not guess, as this is likely to be unsuccessful
  - Passing of the MC part guaranteed if 70% of the points are reached.

## Books

You can get me as  
an e-Book at the  
library!

- **Fundamentals of Computer Graphics**

by Marschner, Shirley



- **Computer Graphics. Principles and Practice**

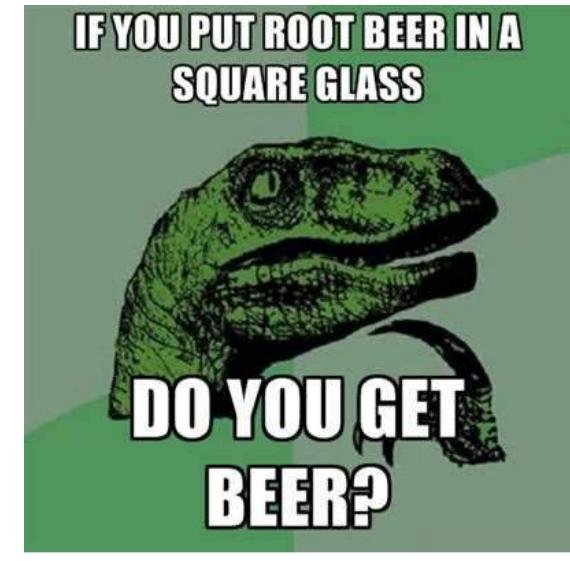
by James D. Foley, Andries VanDam, Steven K. Feiner

- **Real-time Rendering**

by Tomas Akenine-Möller, Eric Haines, Naty Hoffman - Peters, Wellesley

## Questions

- Answers.ewi.tudelft.nl / FAQs on Brightspace
- TAs in practicals
- Teachers in class/break
- **Last resort:** Course email [cg-cs-ewi@tudelft.nl](mailto:cg-cs-ewi@tudelft.nl)
  - Processing time: ~6 workdays
  - **Use TU Delft email and add your student number**



more awesome pictures at [THEMETAPICTURE.COM](http://THEMETAPICTURE.COM)

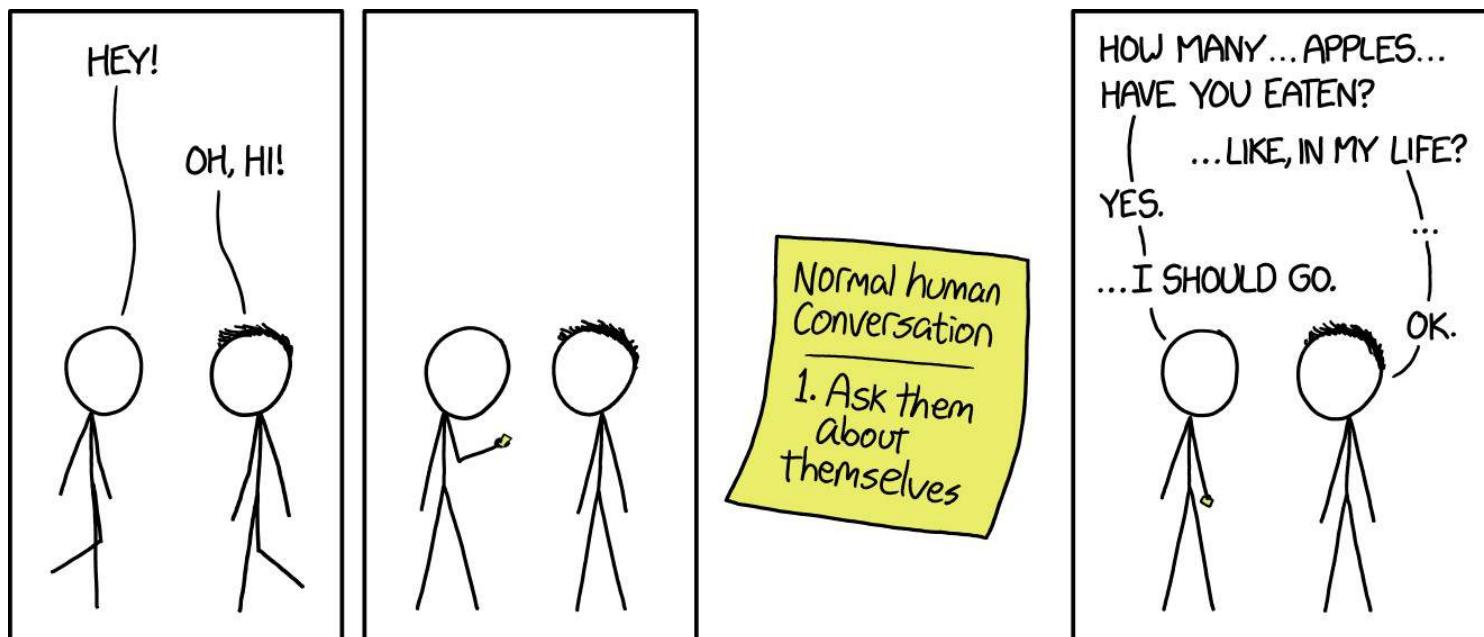
## Information overkill...



## Where do I find all the info?

- On Brightspace
  - Tasks pop up every week
    - (e.g., Lecture 4.5h, readup/revise 3h, exercises 1h, Algebra test 2h, assignments 2h, setup 0.5h “=“ 13h)
  - Content is available after the lectures
  - Assignment download and hand in
  - Assessment explanations
  - Additional exercises
  - Reading instructions
  - FAQs
  - and much more...

# Questions?





Let's get  
started!

## Today's goals

- S1- Explain and compare the structure and properties of standard algorithms and data structures linked to Computer Graphics.
- [S6- Apply the knowledge obtained in this course to problems of other fields]
- Reading material:
  - Introduction (Chapter 1)
  - Color (Pages 493-495)
  - Basic Ray tracing (Pages 69 – 71 up to Section 4.2)

# What is Computer Graphics about?

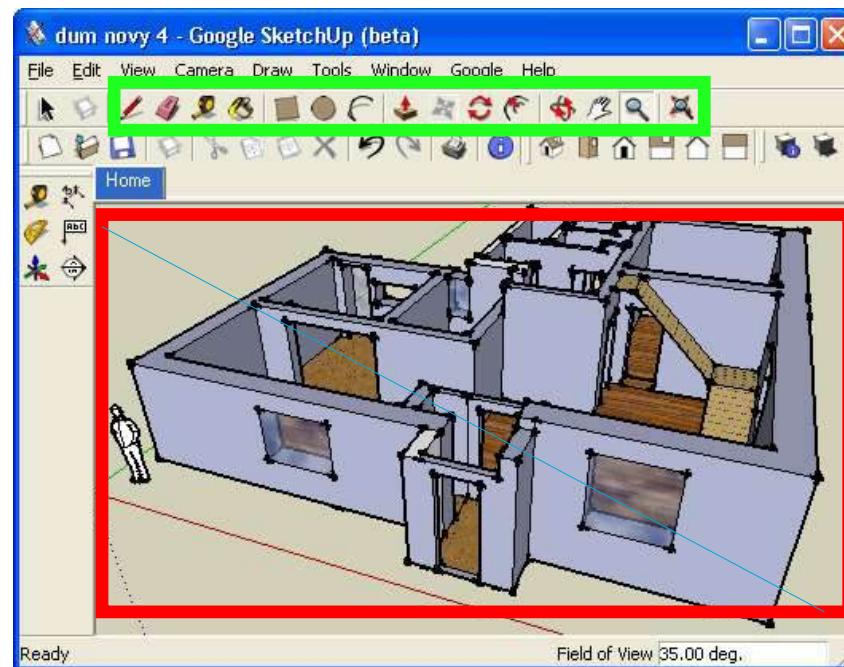
- Modeling - Making content
- Animation - Making movement
- Rendering - Making images

# Modeling

- Create 3D Objects

e.g.,

- Geometry
- Analysis
- Representations



# Subdivision Surfaces



# What is Computer Graphics about?

- Modeling - Making content
- Animation - Making movement
- Rendering - Making images

# Animation

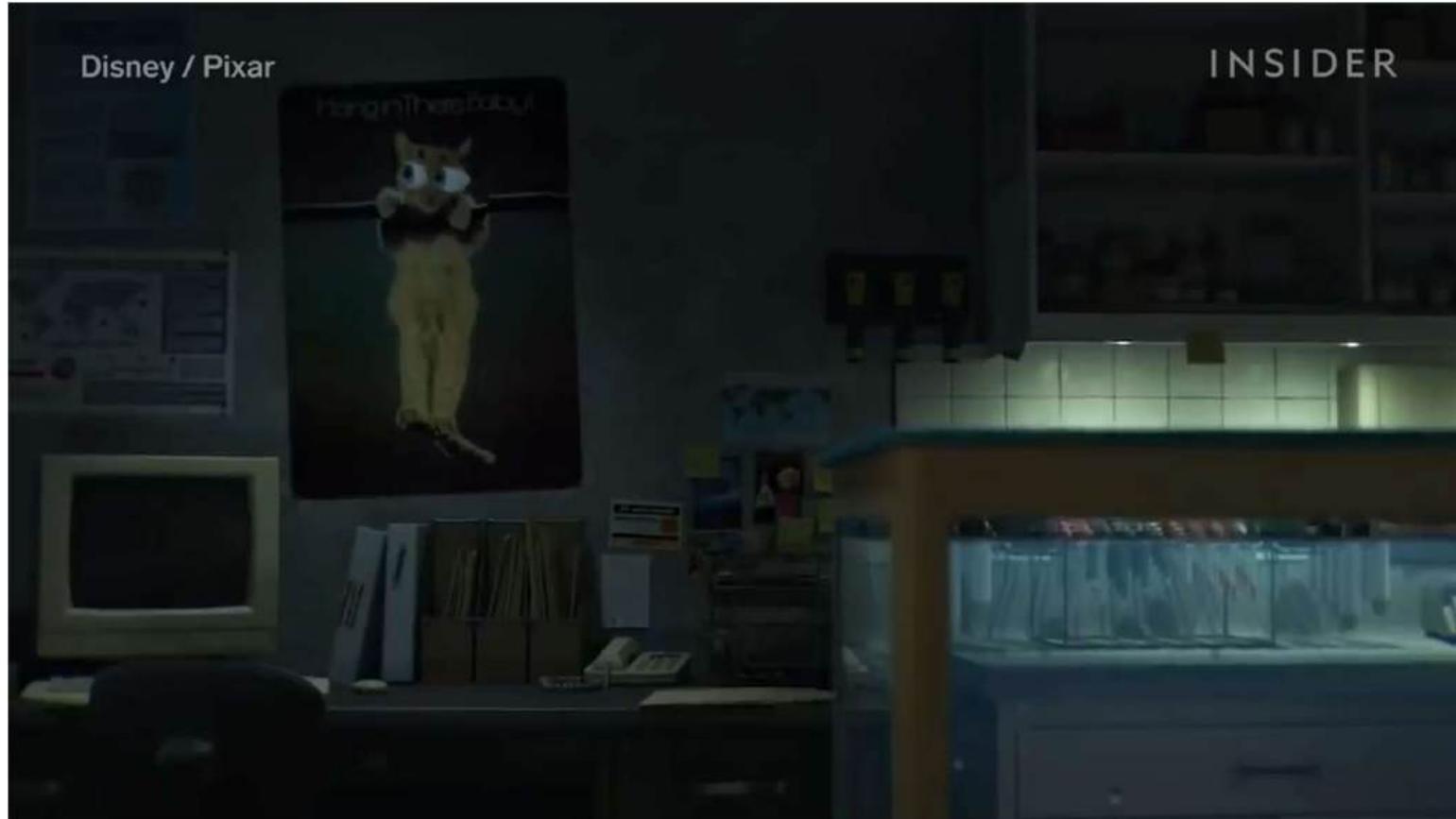
- Synthesize Movement

e.g.,

- Data analysis (e.g., PCA)
- Data Interpolation
- Differential analysis
- Physics



## Extreme Example



Source: Insider  
<https://www.youtube.com/watch?v=qTPKGVrFtQU>

# What is Computer Graphics about?

- Modeling - Making content
- Animation - Making movement
- Rendering - Making images

# Rendering

- Making images

e.g.,

- Physics
- Math
- Perception
- User interfaces
- Electrical Engineering
- ...

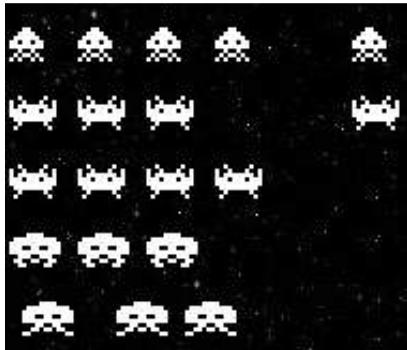
## Rendering



Source: Insider  
<https://www.youtube.com/watch?v=qTPKGVrFtQU>

# Introduction

- Graphics advances at an incredible pace



1978 – Space Invaders



1990 - Loom

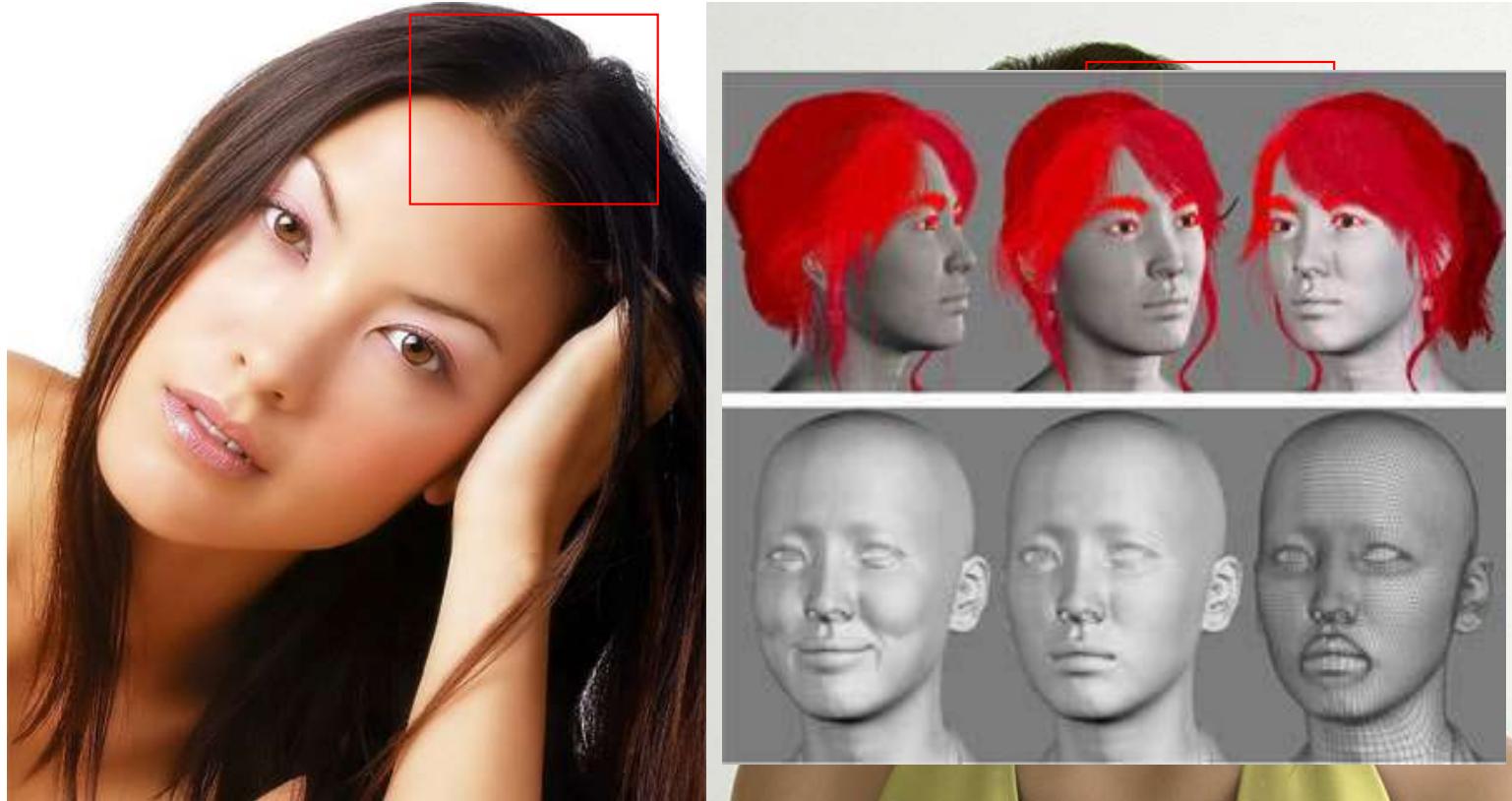


2007 - Unreal



2023 - Starfield

# Photo or Computation?

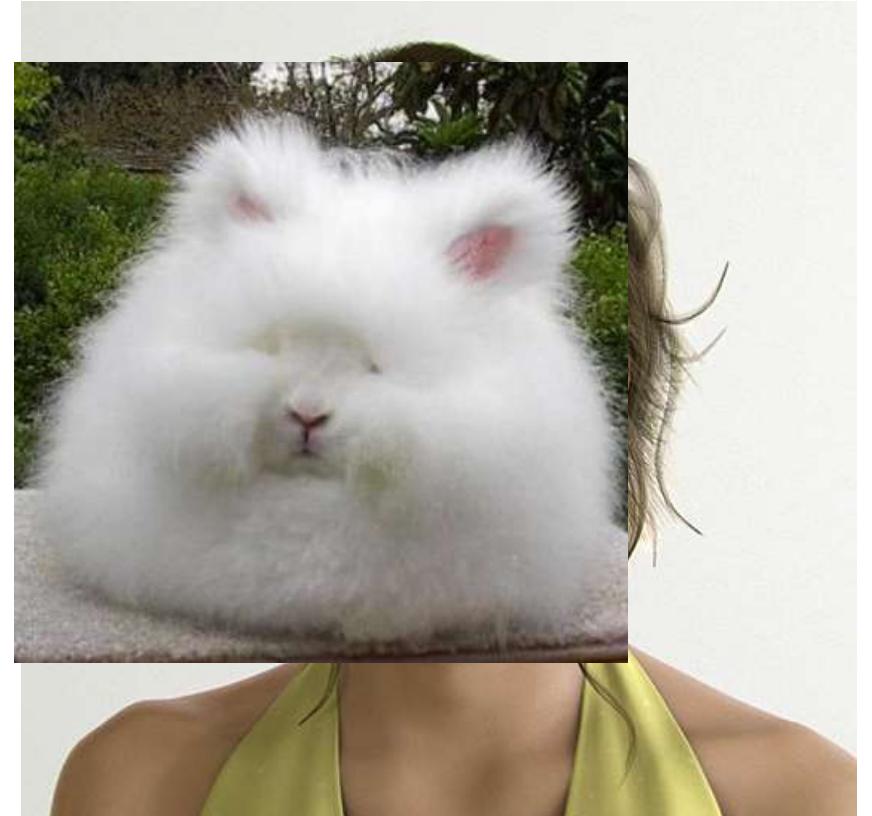


## Can we do better?

- Yes, we can!

It should not...

...take months of work  
...take hours of computation  
...only result in  
one view, pose, and light !



## Intermezzo



## Extreme Computation Times



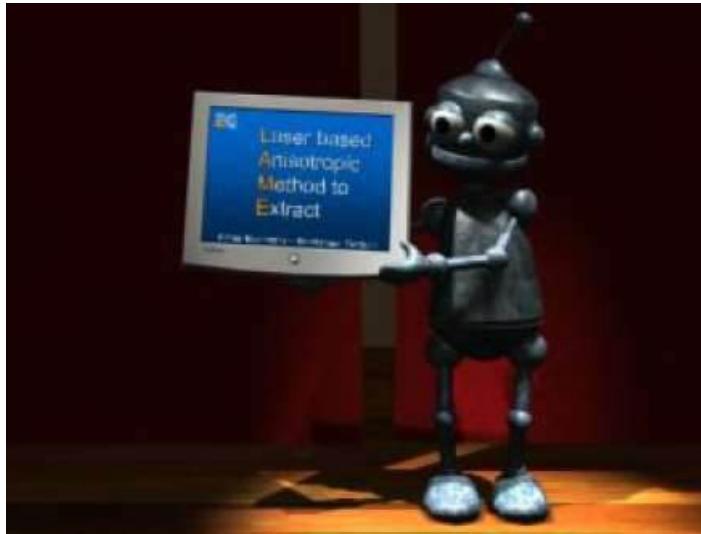
- Big Hero Six – copyright Disney

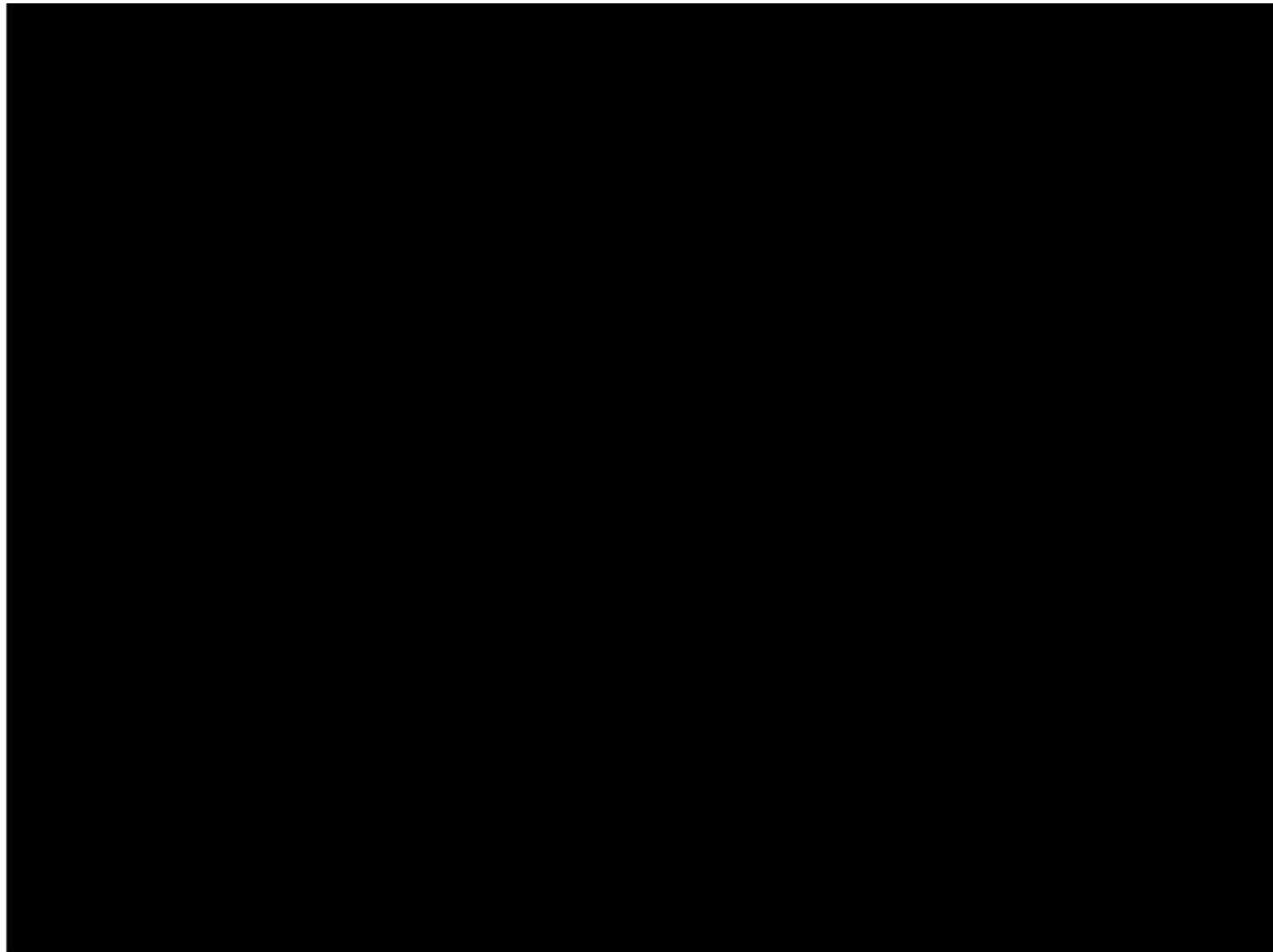
## Financial Facts - Rendering

- Despicable me:  
500.000 € for electricity



## What do we get for it ?





During the practicals:



- Learn to use such tools yourselves!



© Blender Foundation

## How to produce an image?

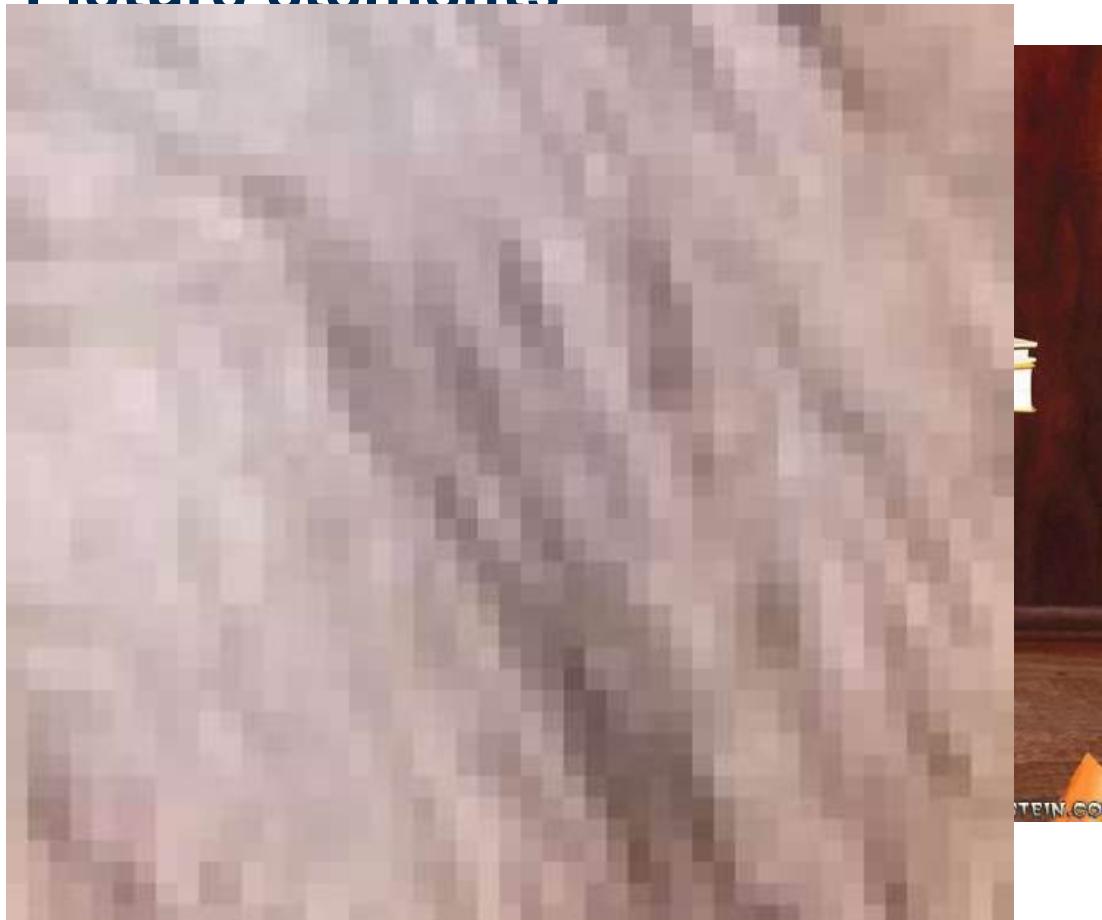
- Computers can calculate...



# Making images with a Computer



## Pixels – Picture elements



# Today

- Give a glimpse on how images are computed
- Explain some of the most-basic principles
- Short outlook on things to come

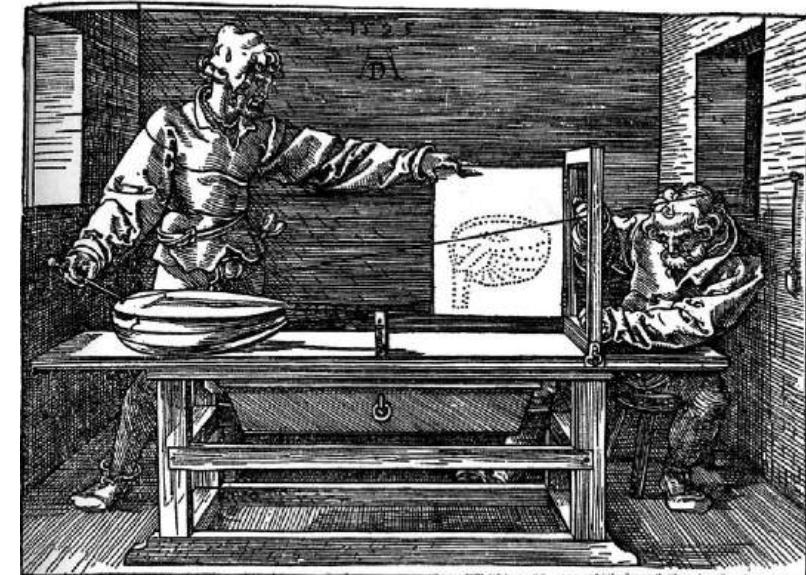
## Producing Images in the Real World

- Albrecht Dürer, 16<sup>th</sup> century



# Producing Images in the Real World

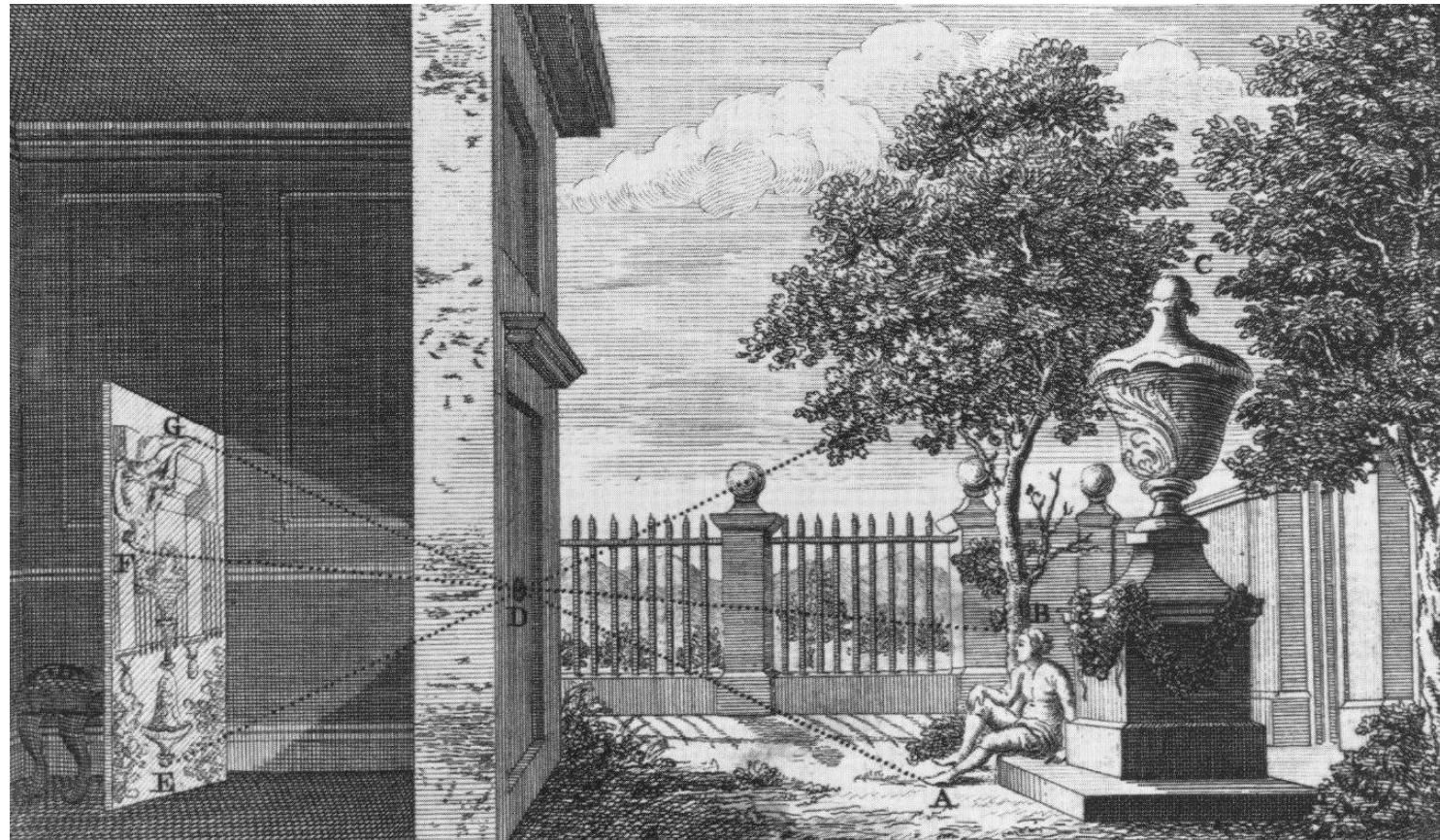
- Albrecht Dürer, 16<sup>th</sup> century



# Producing Images in the Real World



# Producing Images in the Real World



Camera obscura

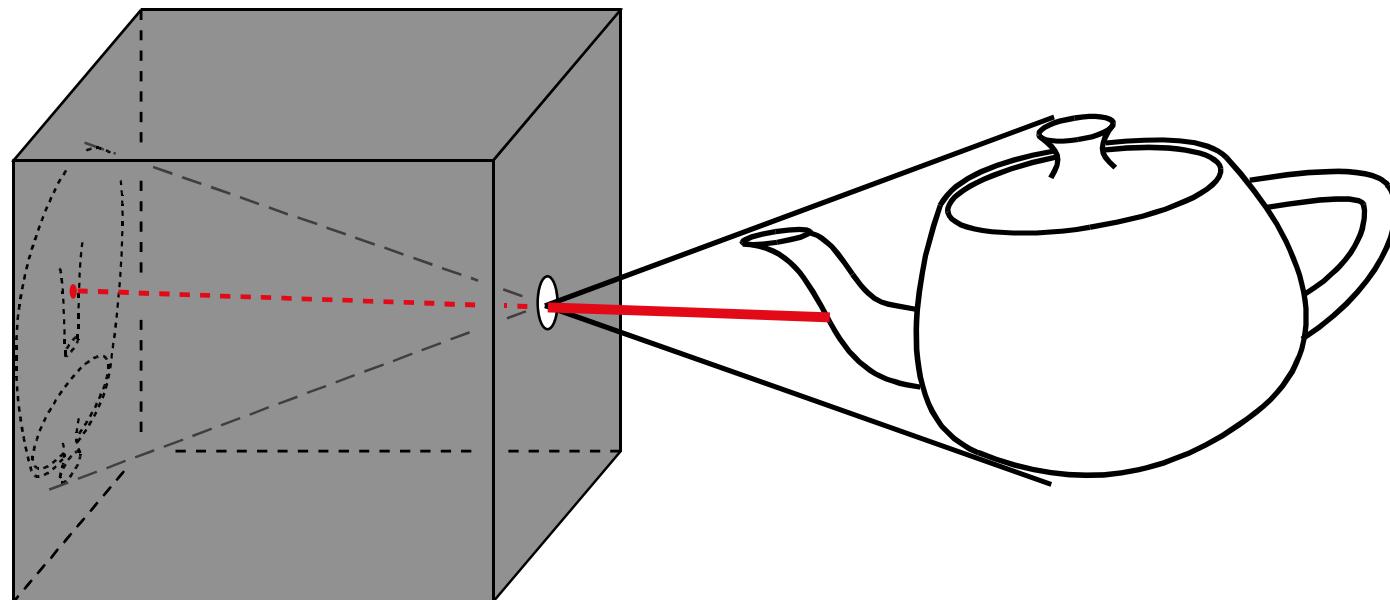
## Producing Images in the Real World

- A photo of such a camera [Abellardo]



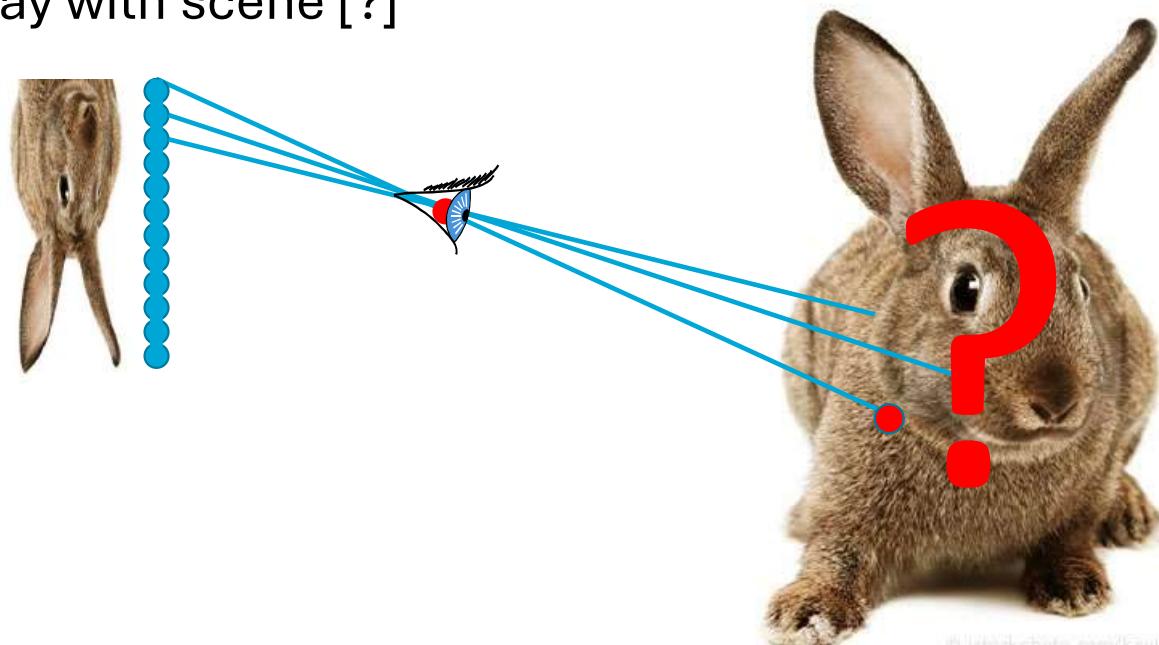
## Pinhole camera

- Box with hole
- Perfect image for “point-sized” hole



## Virtual Camera

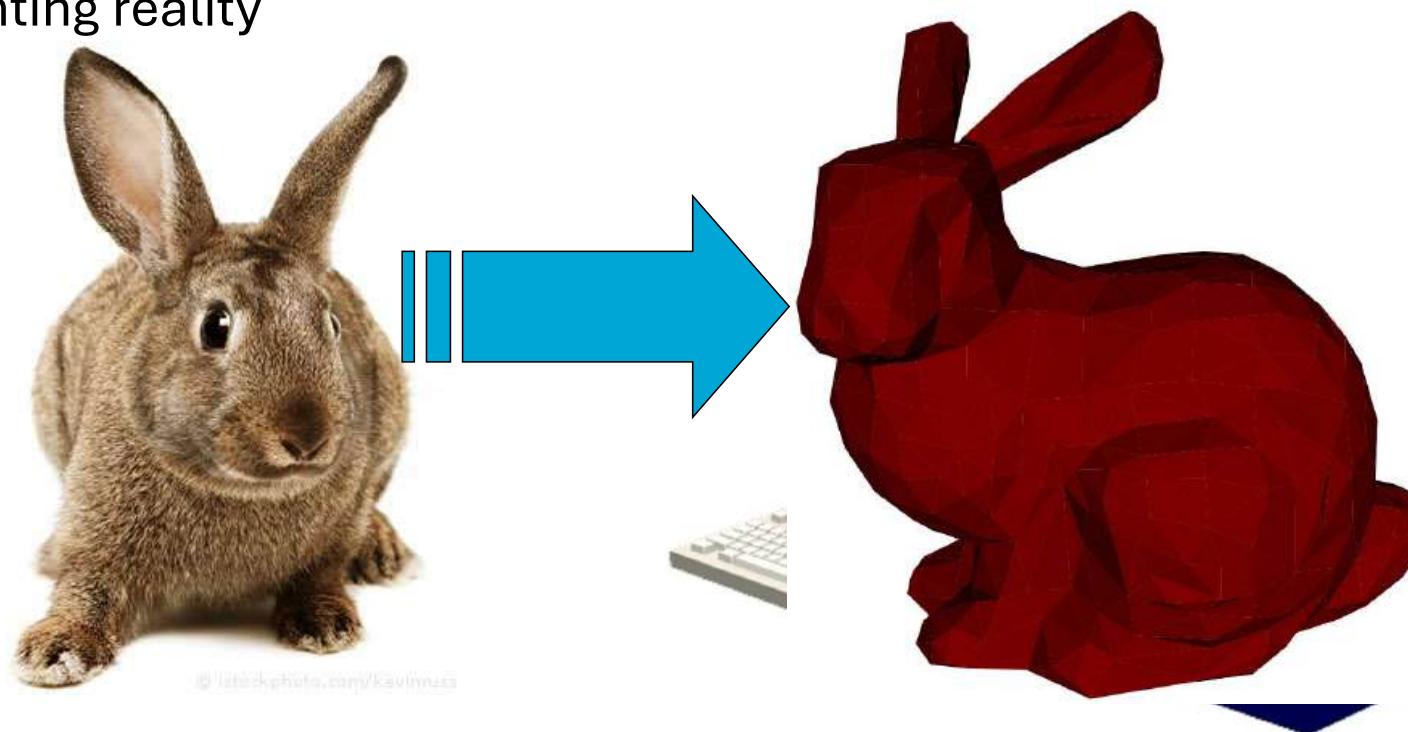
- Take a pixel on the image in the virtual world
- Compute ray through pixel and camera center
- Intersect ray with scene [?]



© iStockphoto.com/kevinzaa

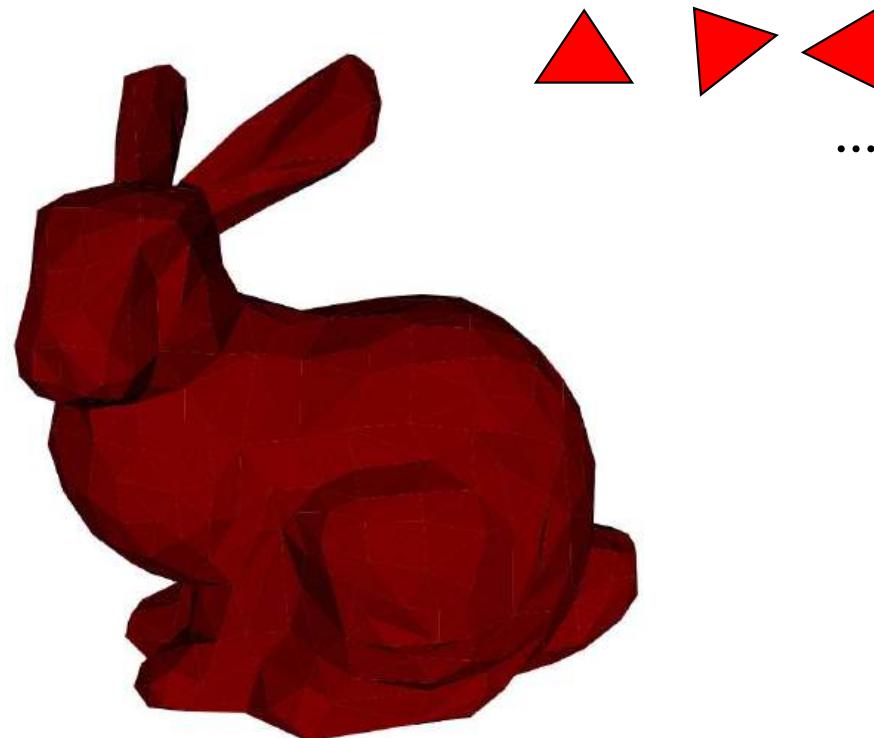
# Models

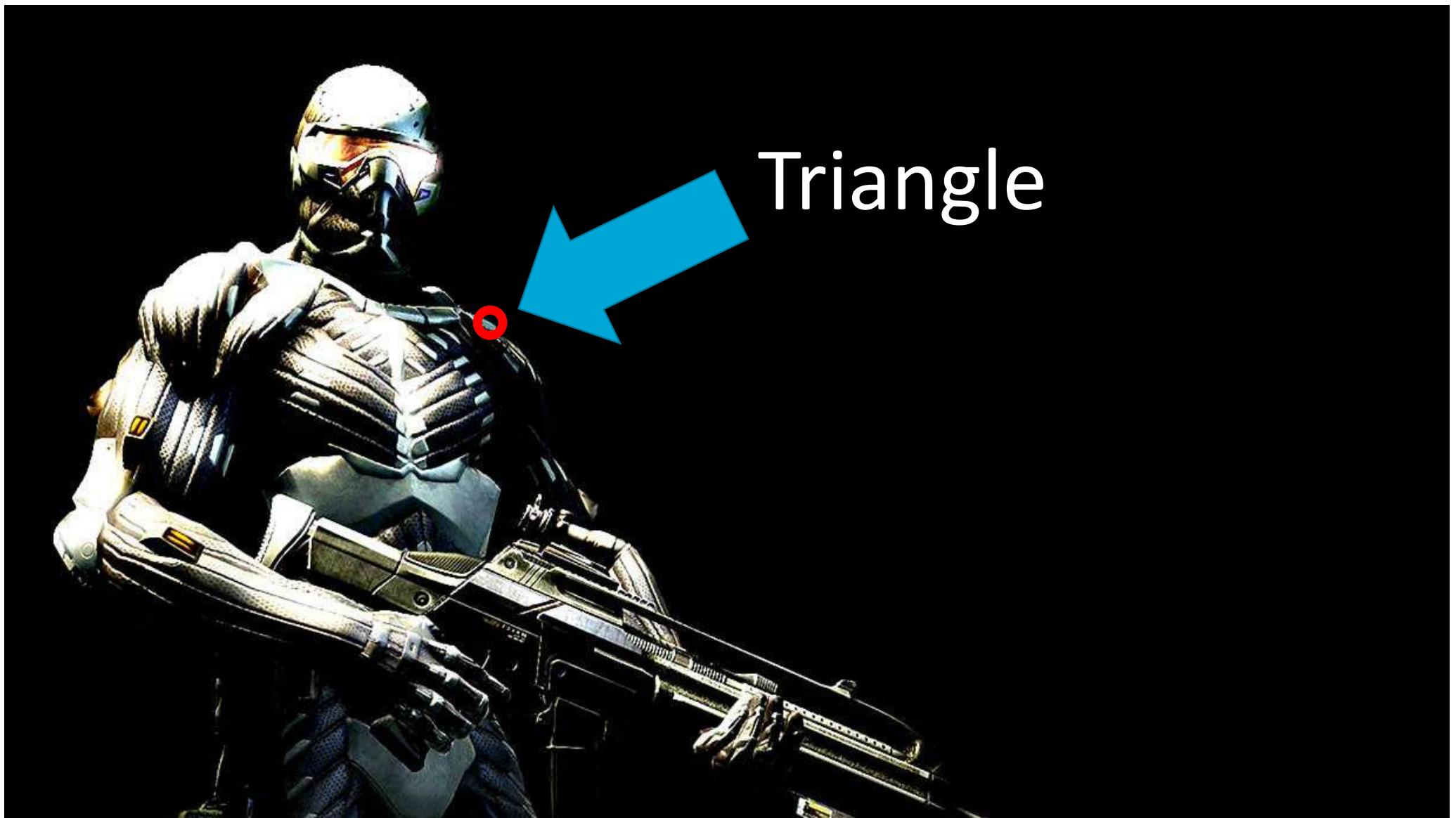
- Representing reality



# Models

- Models are typically lists of triangles

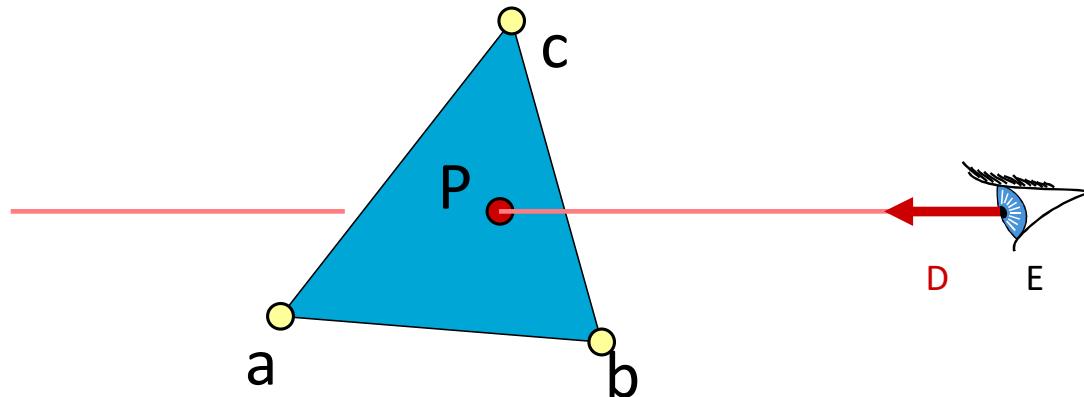




Triangle

## Solve Equations

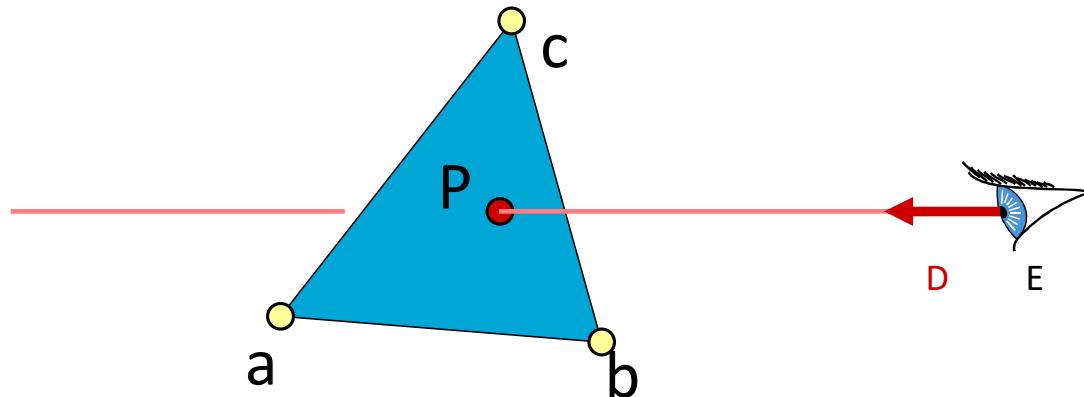
- $E+tD = a + \beta(b-a) + \gamma(c-a)$



## Solve Equations

- $E_x + tD_x = a_x + \beta(b_x - a_x) + \gamma(c_x - a_x)$
- $E_y + tD_y = a_y + \beta(b_y - a_y) + \gamma(c_y - a_y)$
- $E_z + tD_z = a_z + \beta(b_z - a_z) + \gamma(c_z - a_z)$

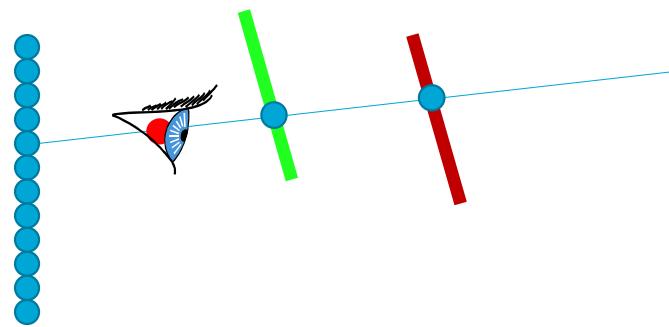
$$\begin{bmatrix} a_x - b_x & a_x - c_x & D_x \\ a_y - b_y & a_y - c_y & D_y \\ a_z - b_z & a_z - c_z & D_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - E_x \\ a_y - E_y \\ a_z - E_z \end{bmatrix}$$



Finally,  
test if P is on  
the triangle or  
outside the  
triangle.  
**Try to do  
this at home!**

## Produce Final Image

- Keep the closest intersection point



## Ray Tracing - Recap

For each pixel

Distance=MAX

Color=0

Ray=computeRay(pixel)

For each triangle

(CurrColor,CurrDistance)=computeIntersection(Ray)

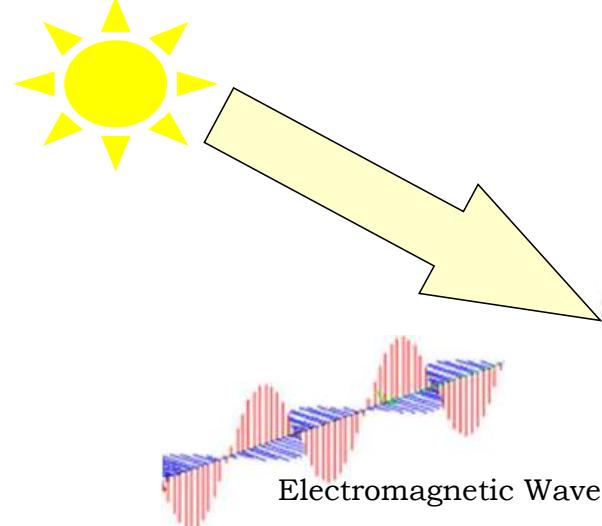
If (CurrDistance<Distance)

    Distance=CurrDistance

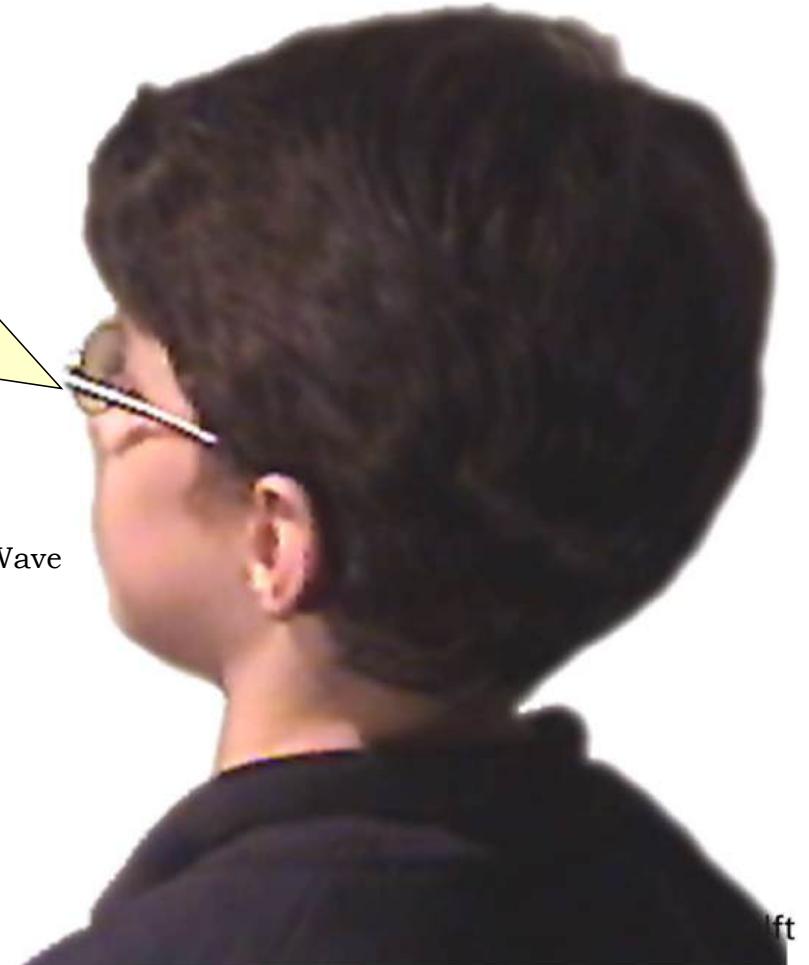
    Color=CurrColor

## What do we see ?

- Light is registered by our eyes... and perceived as color

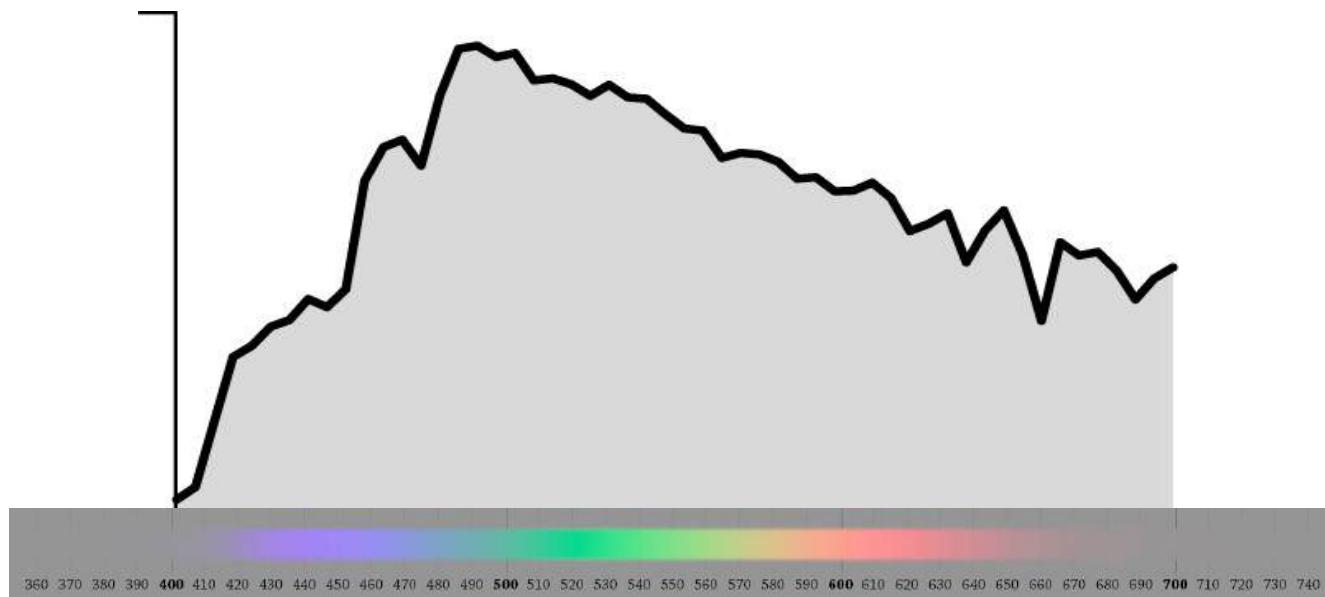


**Observer**



## Physical Definition

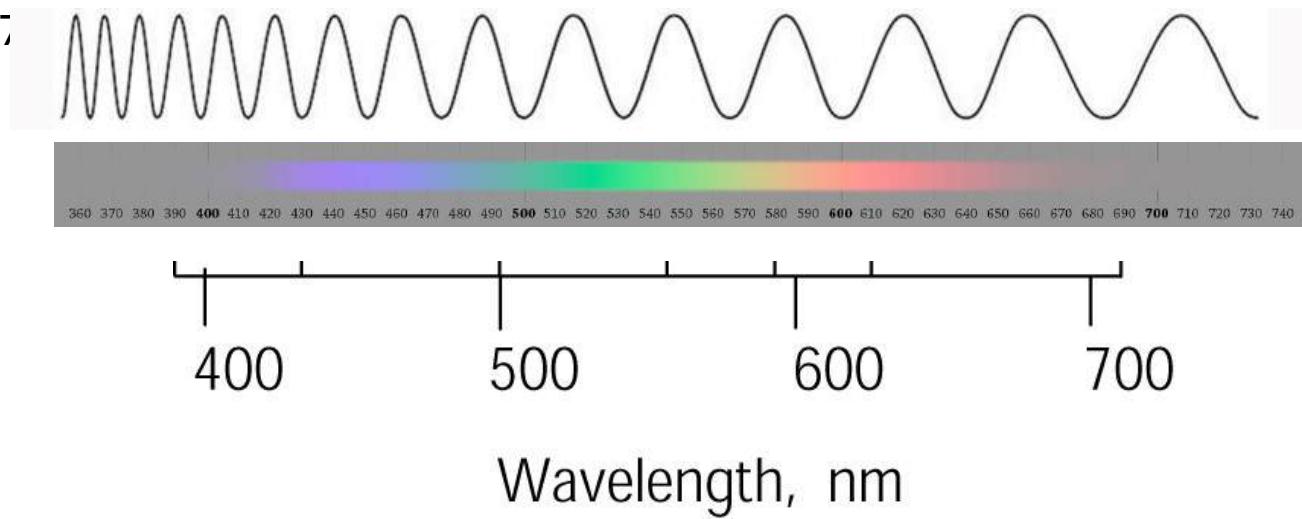
- Light = Distribution of power over a spectrum



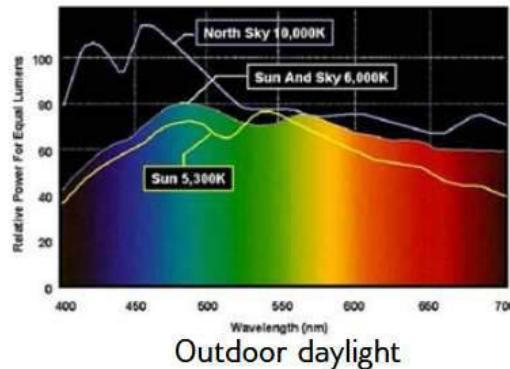
# Light Spectrum

- Visible colors between 380 nm (violet) and 720 nm (red)
- Outside visible range

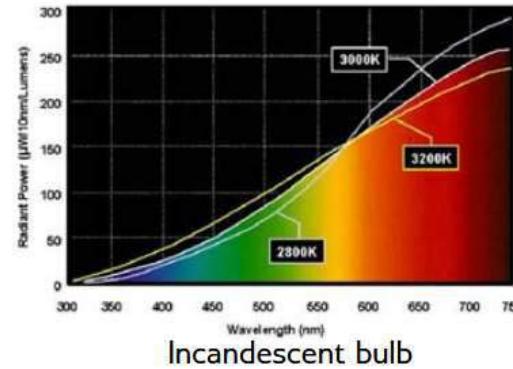
- Below 380 nm : ultra-violet
- Above 720 nm : infrared



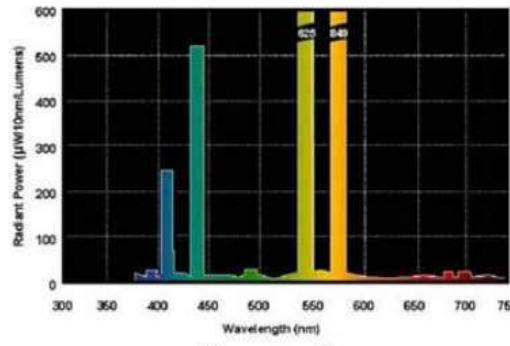
## Examples of Spectra



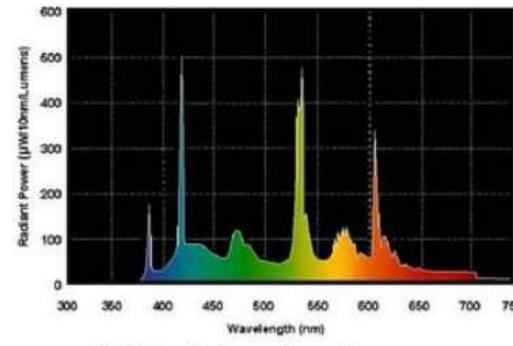
Outdoor daylight



Incandescent bulb



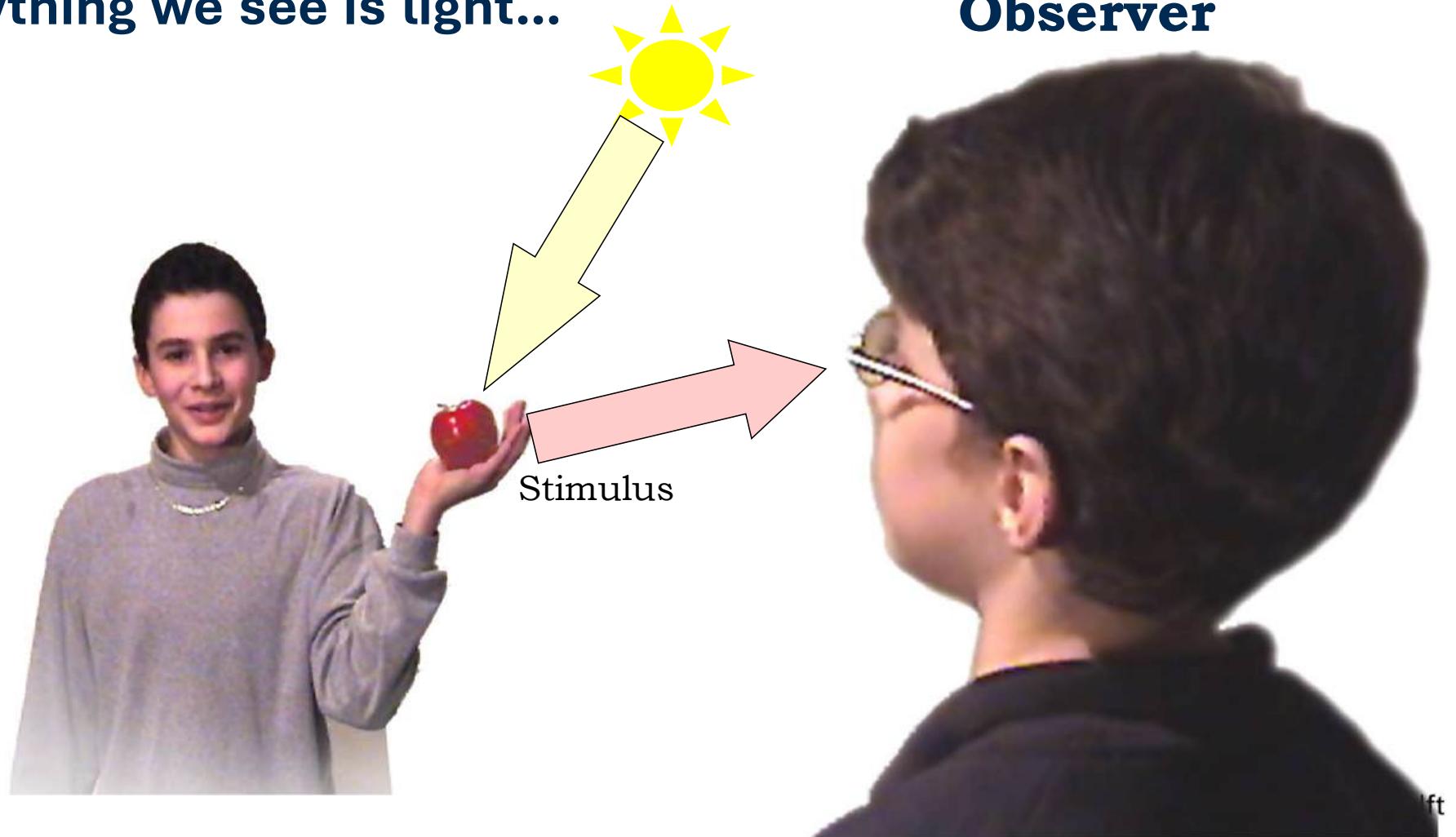
Mercury lamp



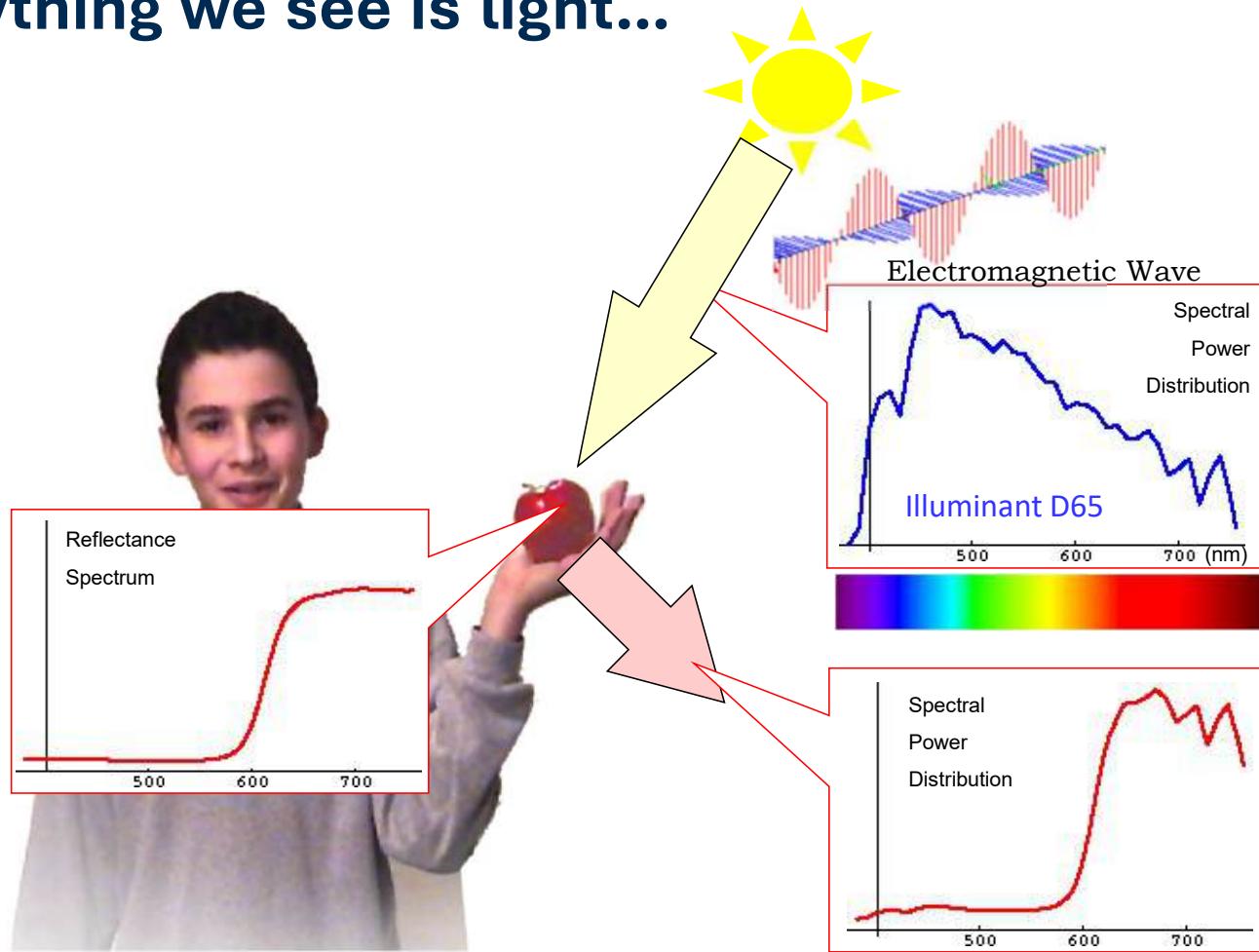
SP65 triphosphor fluorescent

© General Electric Co., 2010

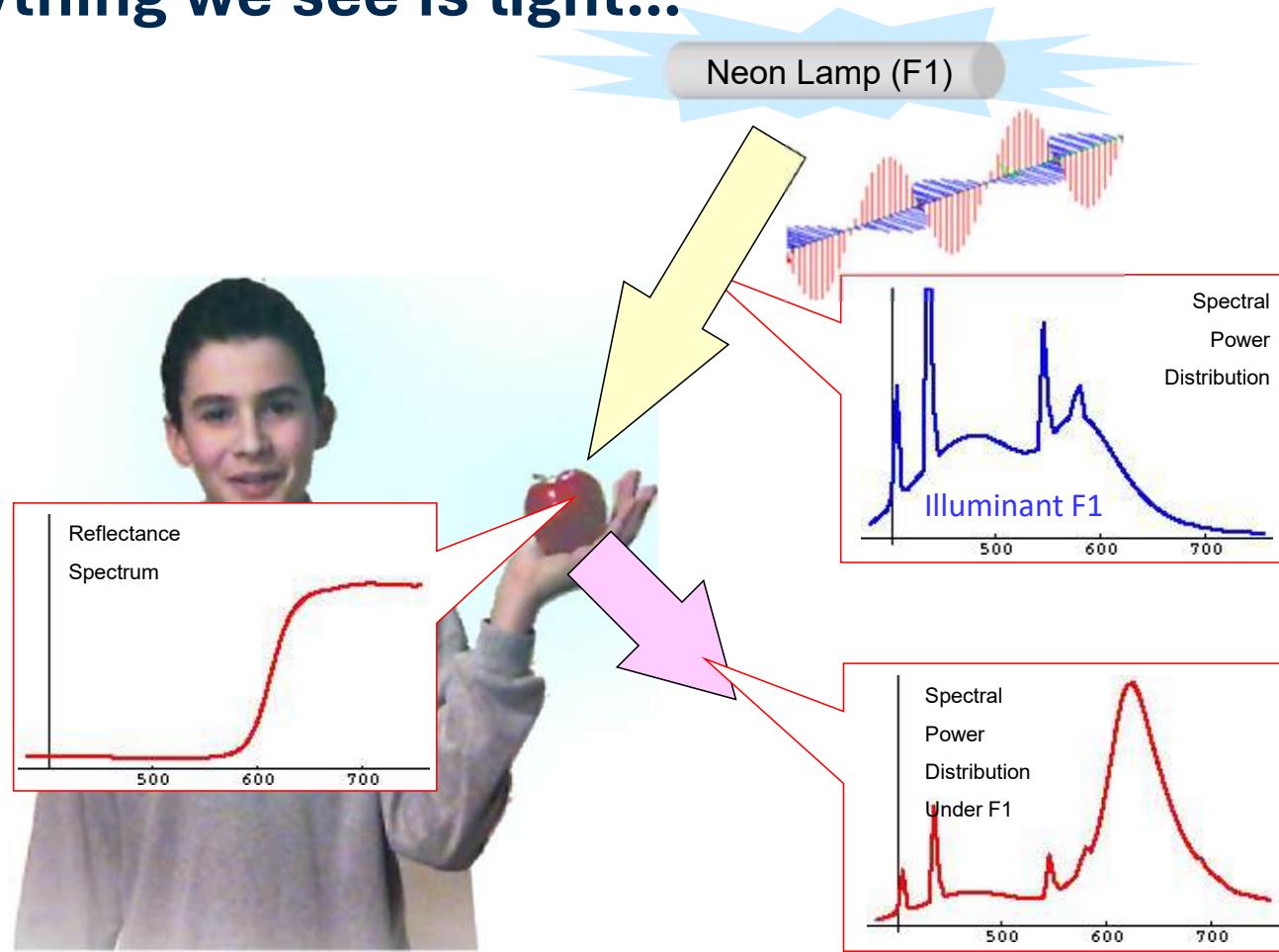
**Everything we see is light...**



# Everything we see is light...

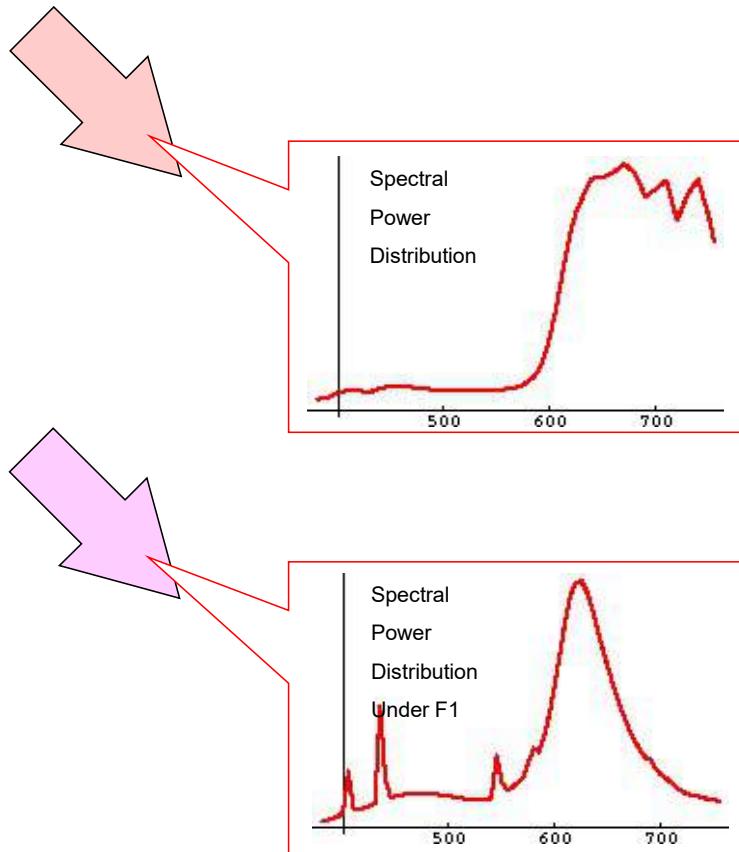


# Everything we see is light...



## Everything we see is light...

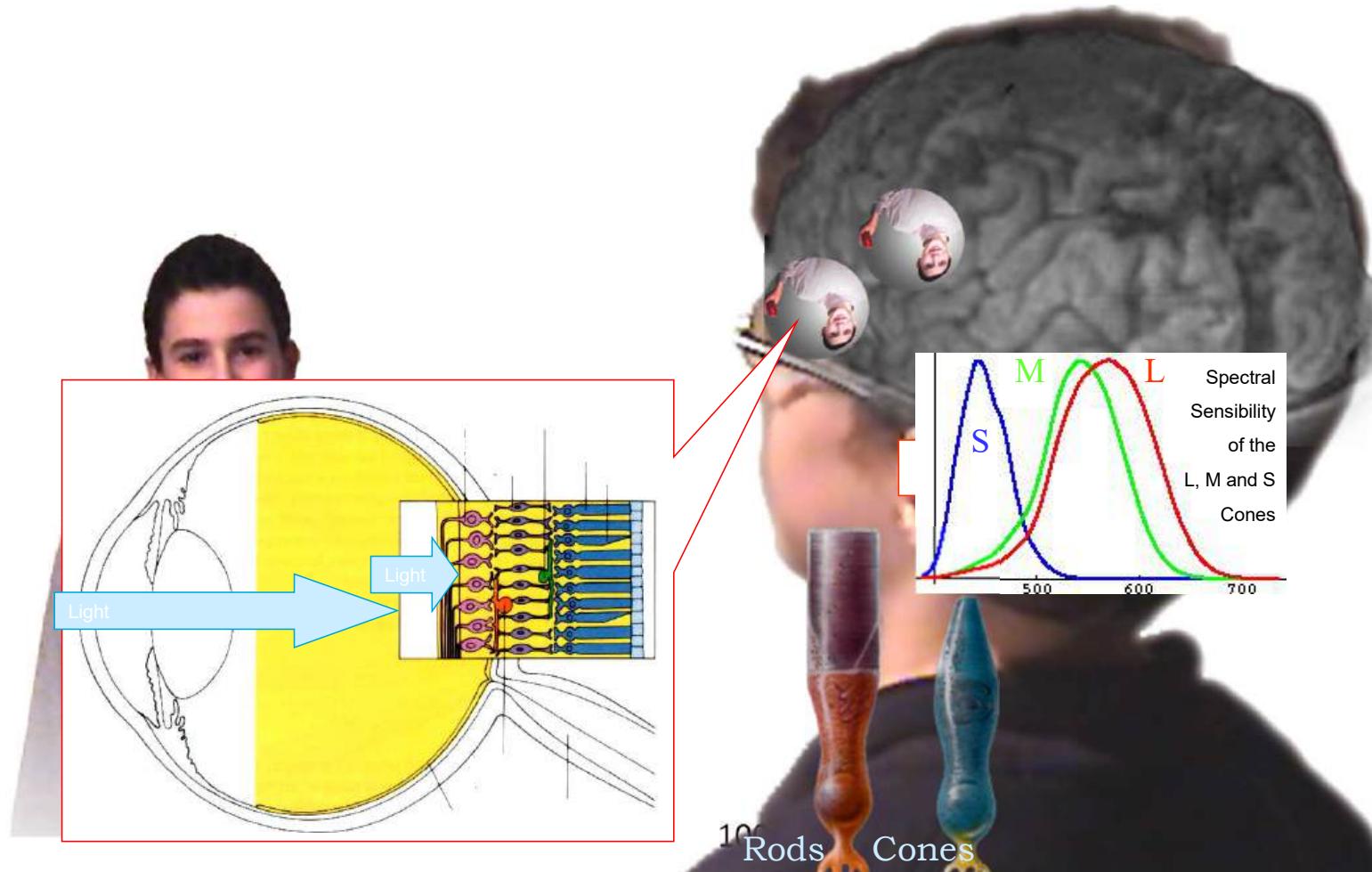
...but we cannot always distinguish it, as our eyes are not perfect.



## Observer



# Eye Biology



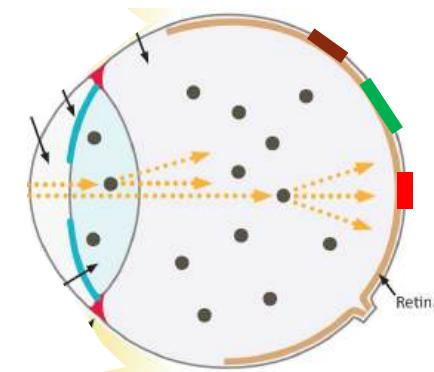
# Eye Biology

- Cones :
  - Chromatic perception (3 types-LMS)
  - Concentrated in center of retina
  - 6 to 7 million in retinal center
    - 3 times full HD
  
- Rods :
  - Achromatic perception
  - Low-light vision

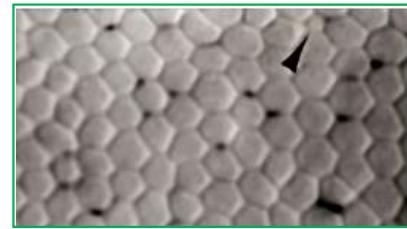


# Eye Biology

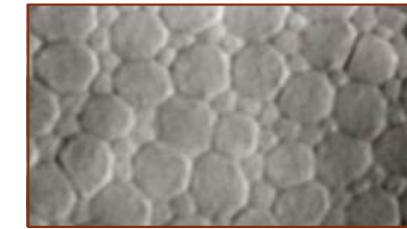
- Retina consists of Cones & Rods



Center – fovea  
**only cones**



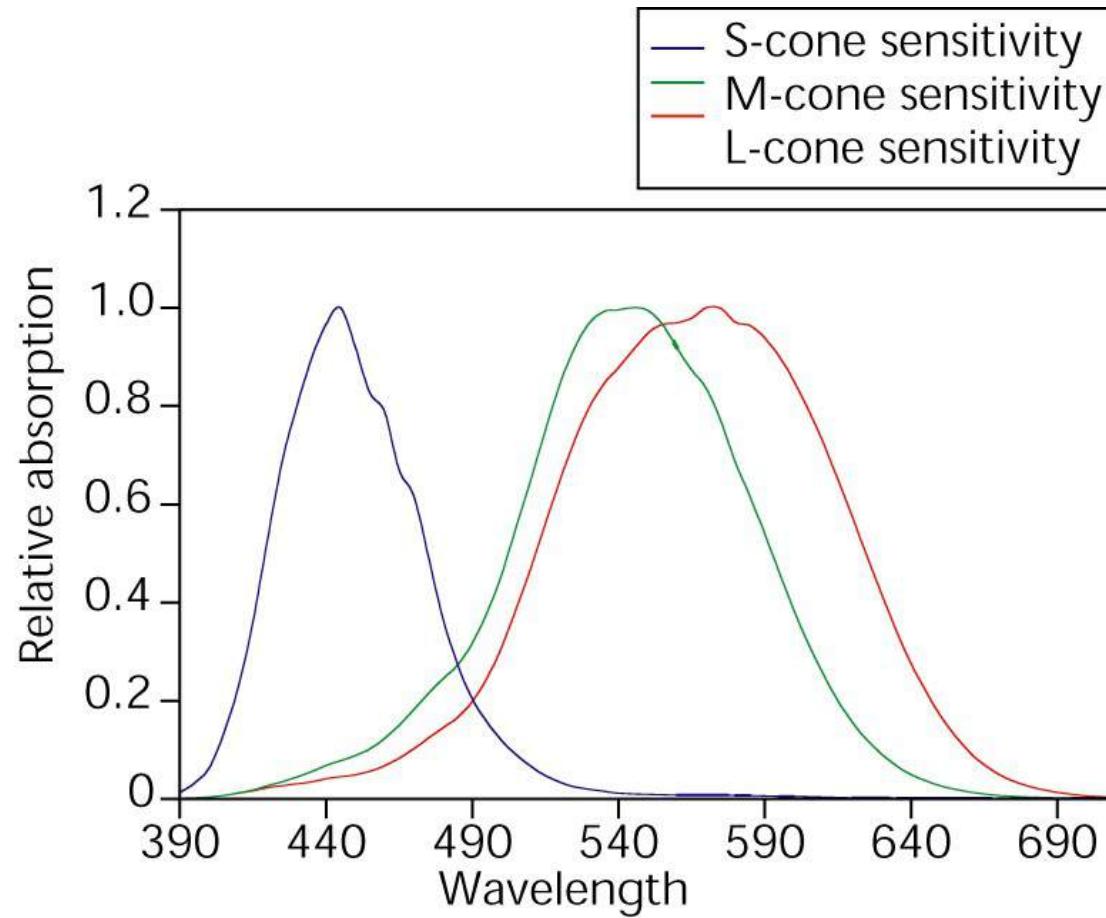
Boundary region  
Mix of both



Periphery  
Mix of both (**more rods**)

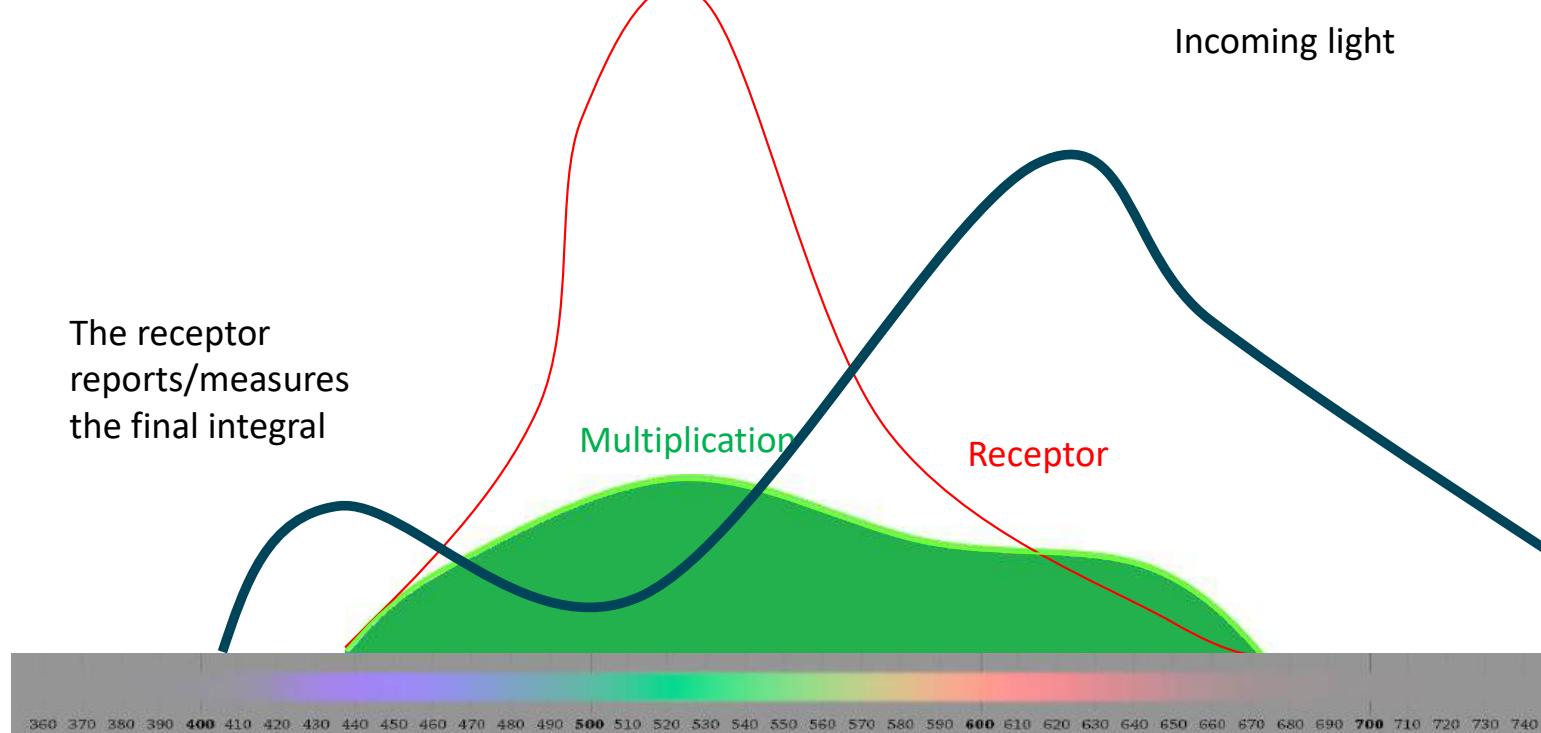
Curcio, C. A., Sloan, K. R., Kalina, R. E., Hendrickson, A. E., 1990. Human photoreceptor topography. J Comp Neurol 292, 497-523

## 3 Cone types



## Simplified Receptor/Light Interaction

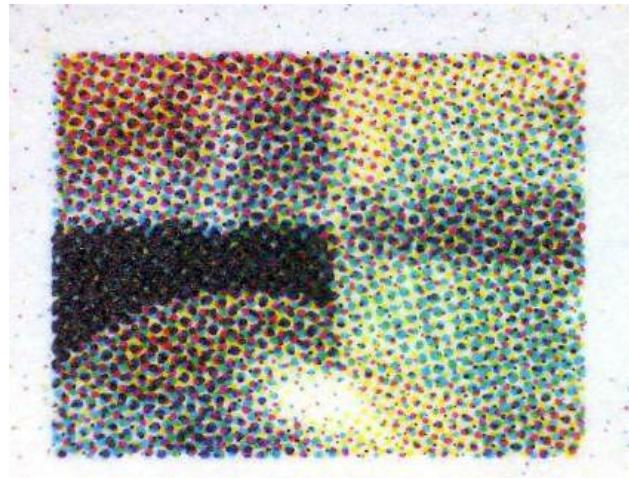
- Multiply incoming light and receptor response curve and integrate the resulting amount



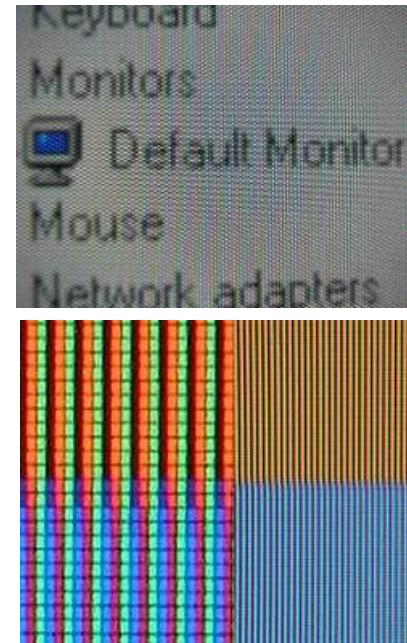
# Receptor and Incoming Light

- How many dimensions?

3 !



Printers work with CMY  
ink

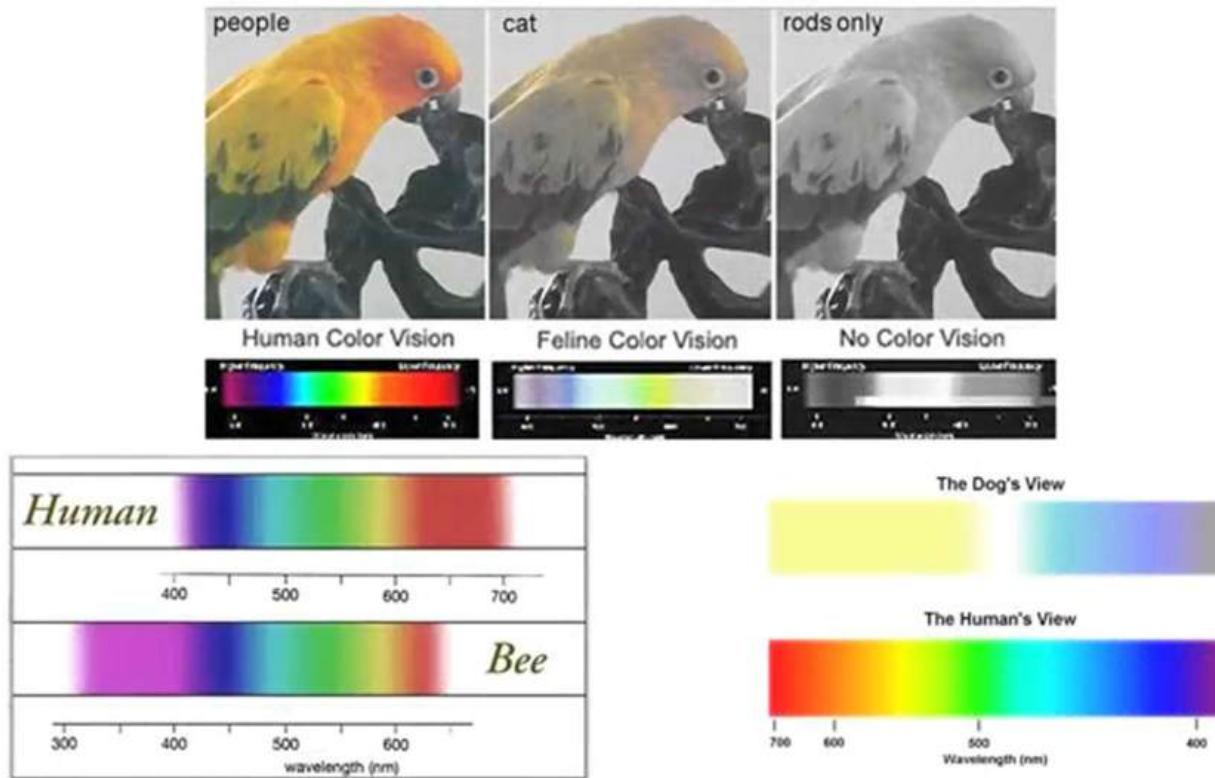


Screens work with RGB  
mask

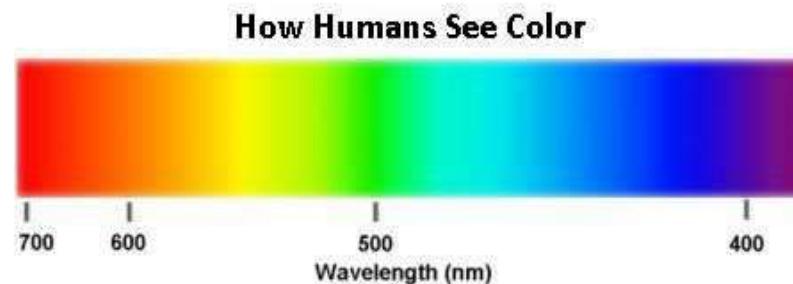
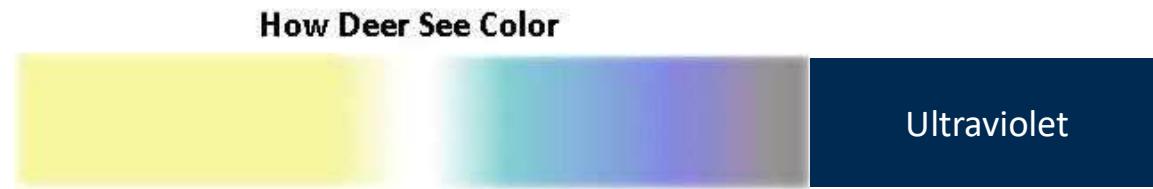
## Metamerisms



## Different Species = Different View of the world



# Different Species = Different View of the world



## Different Species = Different View of the world

Confirmed:

Deer See Ultraviolet, What Does This Mean To Hunters?



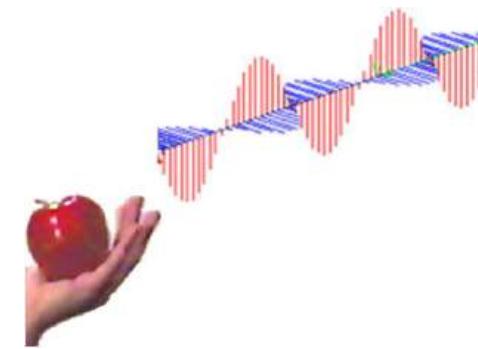
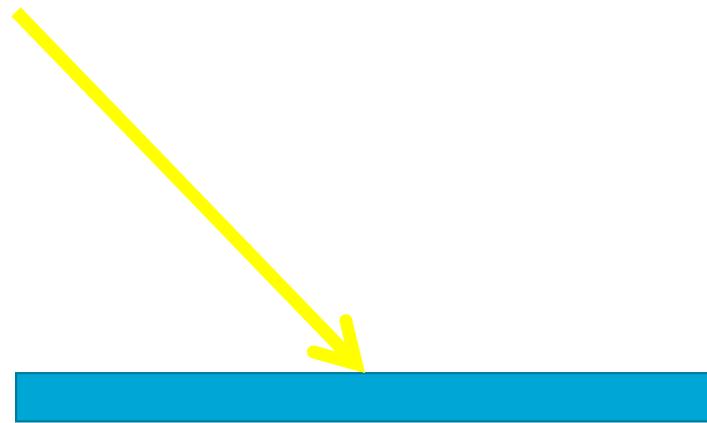
<https://bowhunting.net/2019/02/confirmed-deer-see-ultraviolet-what-does-this-mean-to-hunters/>

**Everything we see is light...**



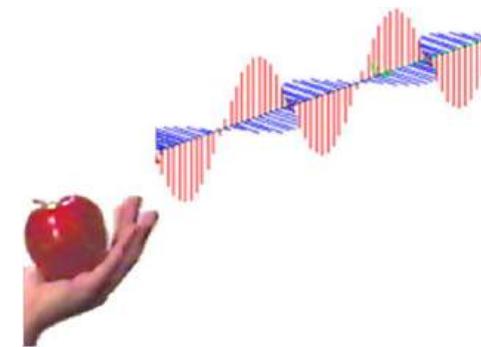
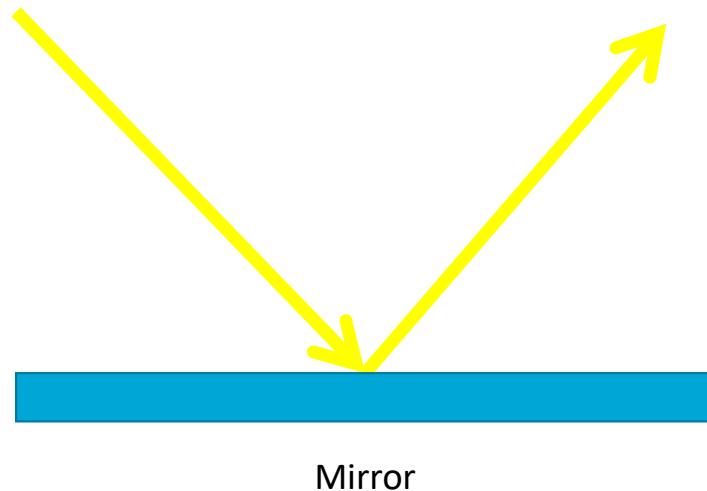
## Reflection

- What happens when the light hits a surface?



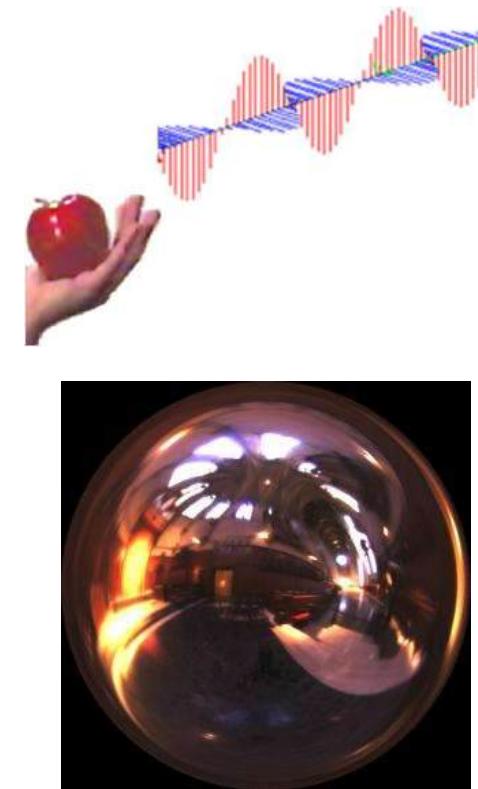
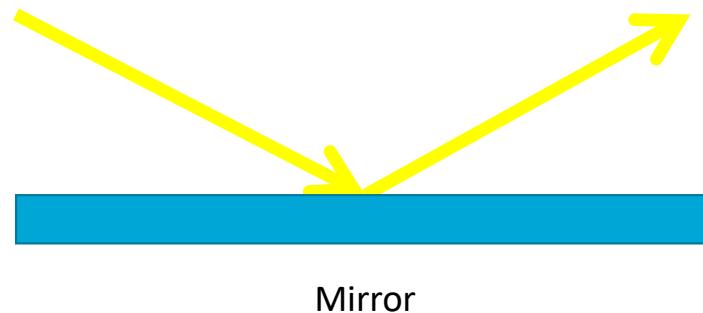
## Reflection

- What happens when the light hits a surface?



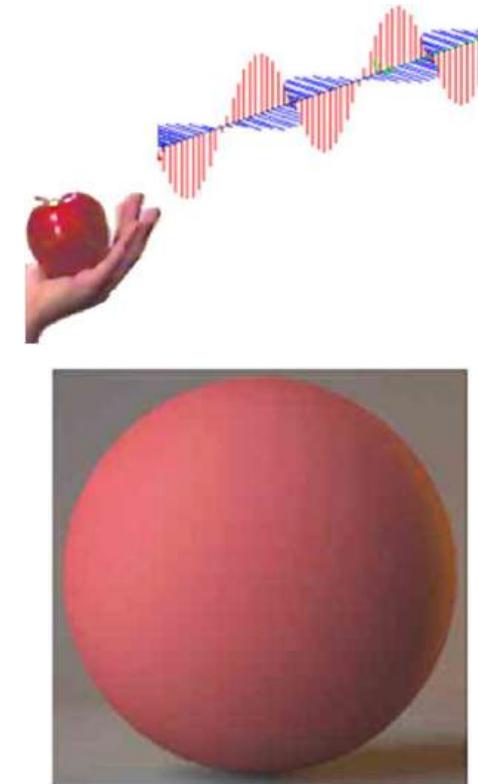
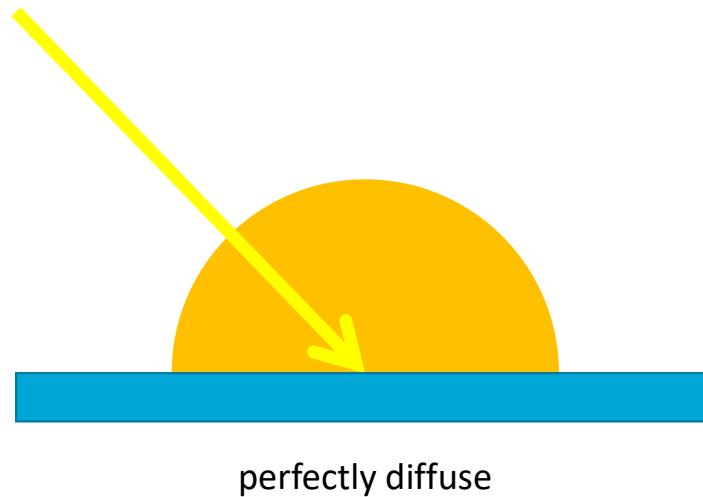
# Reflection

- What happens when the light hits a surface?



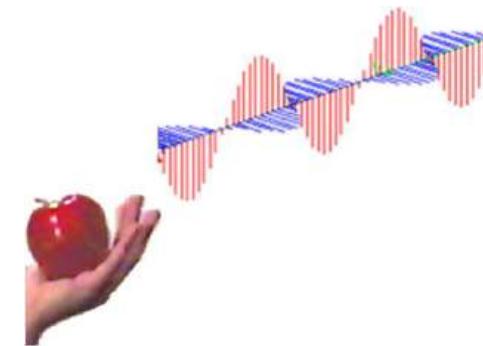
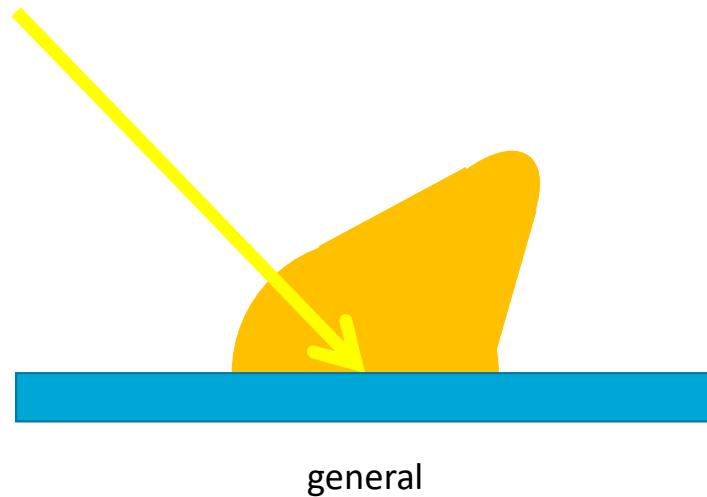
## Reflection

- What happens when the light hits a surface?



## Reflection

- What happens when the light hits a surface?



**Everything we see is light...**



## Ray Tracing - Cost

For each pixel

Distance=MAX

Color=0

R=computeRay(pixel)

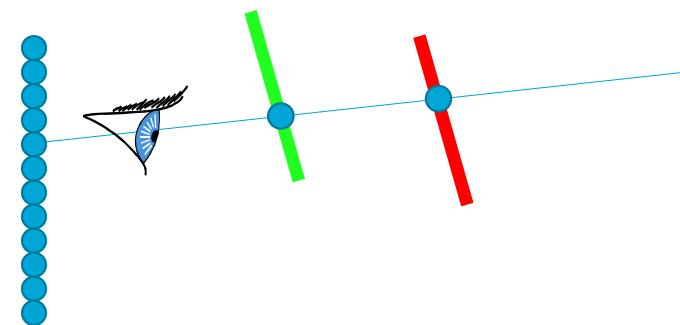
For each triangle

(CurrColor,CurrDistance)=testIntersection(R)

If (CurrDistance<Distance)

Distance=CurrDistance

Color=CurrColor



## Performance Analysis

- **Stupid** implementation:
- Ray Tracing:

Cost = Pixels \* Triangles



e.g., 100.000 triangles and a 1000^2 screen:

Raytracing:  $100.000 \times 1.000.000 = 10^{11}$

## Performance Analysis

- Smart implementation:
- Ray Tracing:

Cost = Pixels \* log(Triangles)  
→ + building a structure



e.g., 100.000 triangles and a  $1000^2$  screen:

Raytracing:  $1.000.000 * 5 + X = 5 * 10^6$

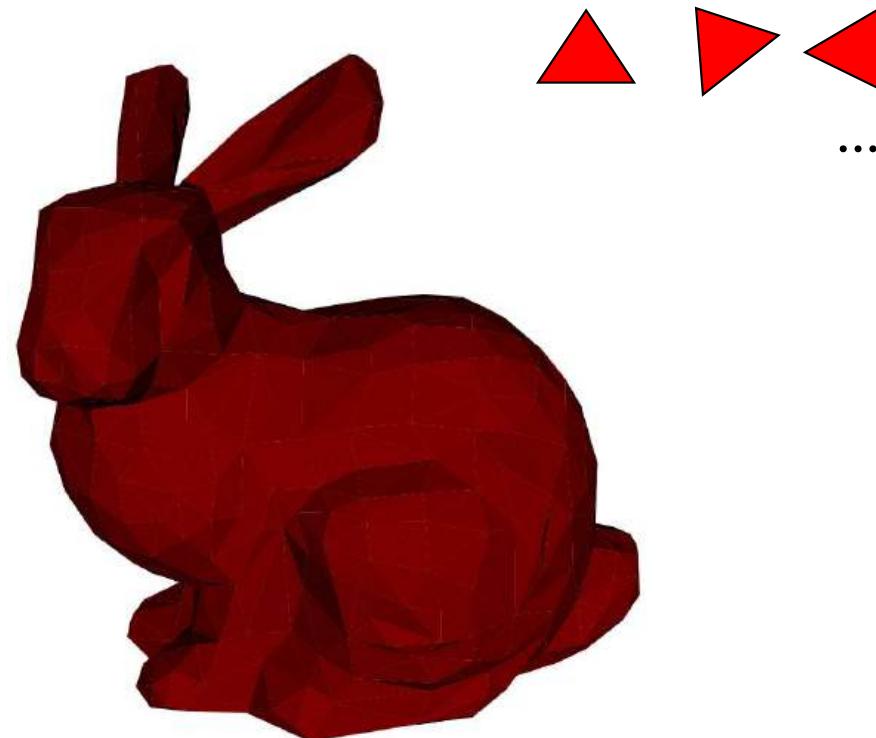
## What is currently used? (30 Images/Sec)

Alternative approach:

Rasterization via the Graphics Pipeline

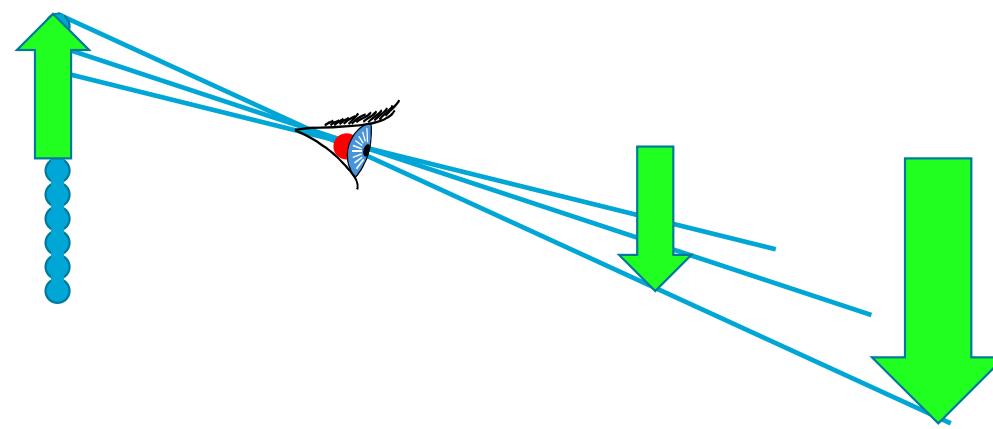
## Simplified Graphics Pipeline

- Models are typically lists of triangles



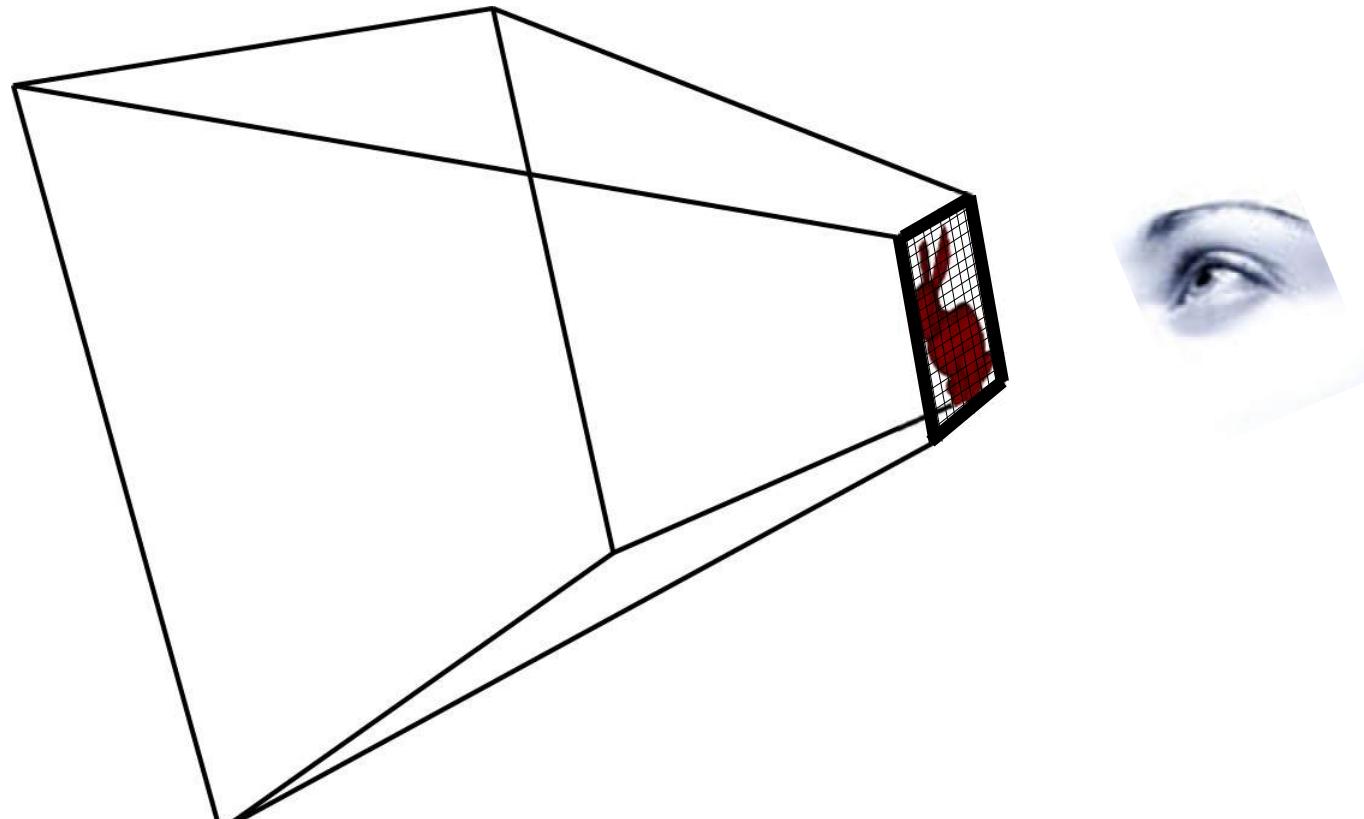
## Virtual Camera

- Camera Plane in front of the eye



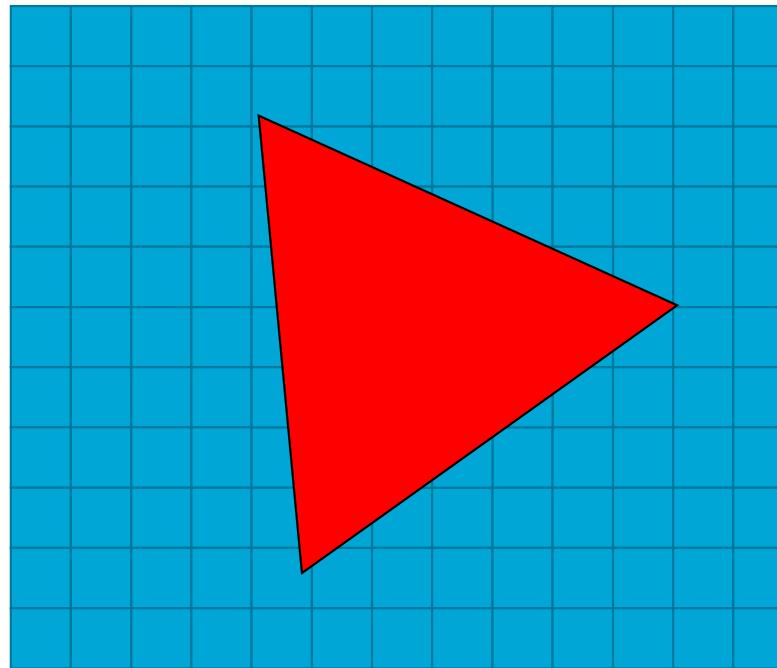
# Simplified Graphics Pipeline

- **Projection:** Transform coordinates to screen



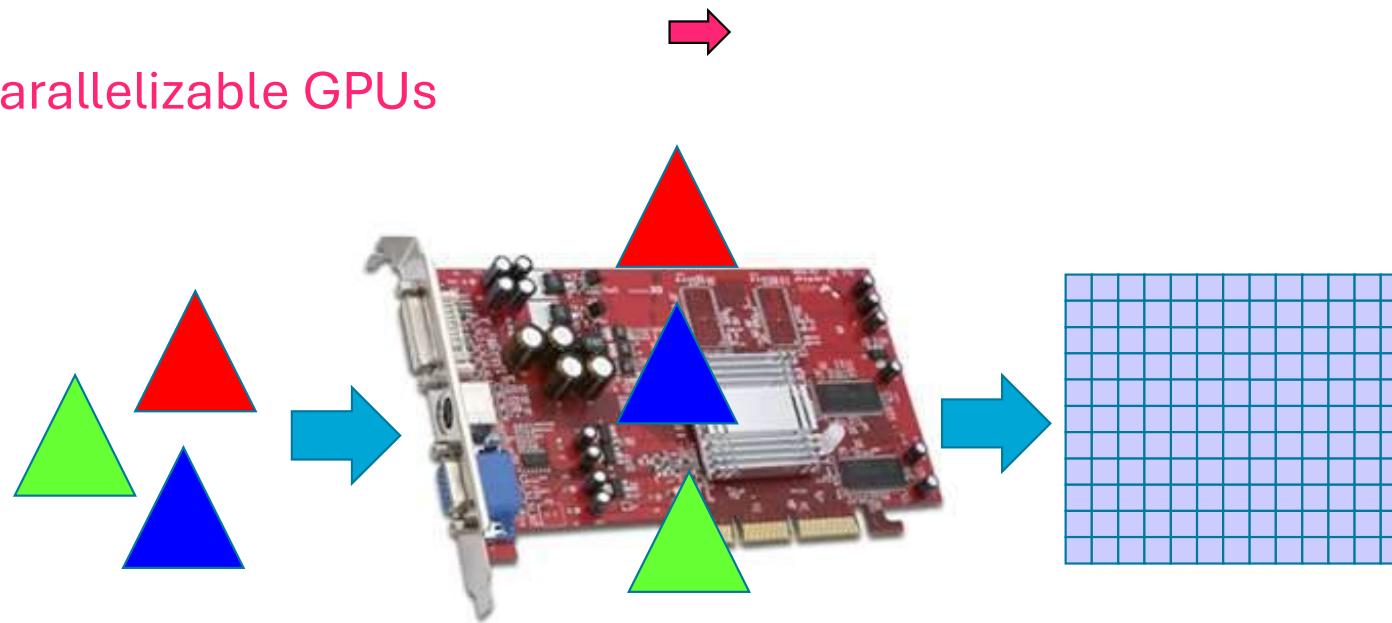
# Simplified Graphics Pipeline

- Rasterization: Fill screen pixels



## Simplified Graphics Pipeline

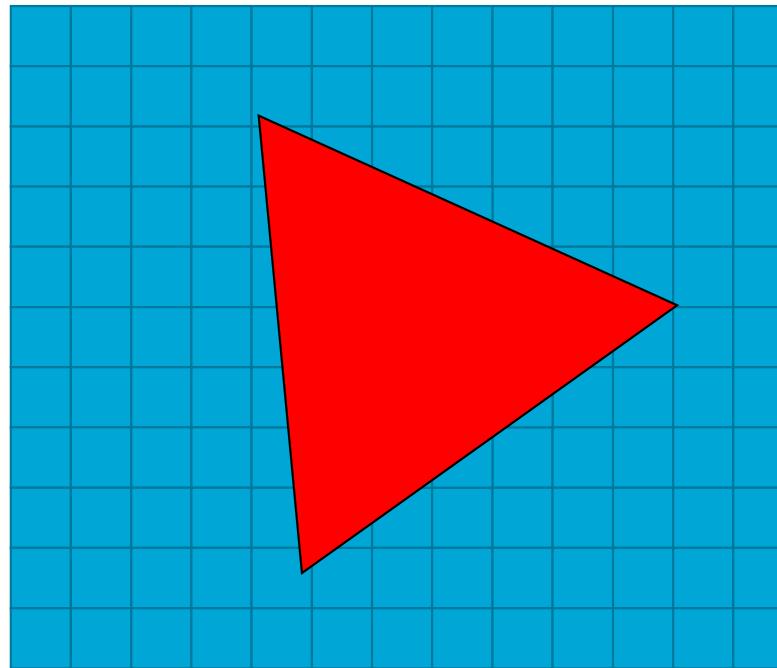
- Highly parallelizable GPUs



...Nvidia 1080 listed with 3584 cores...

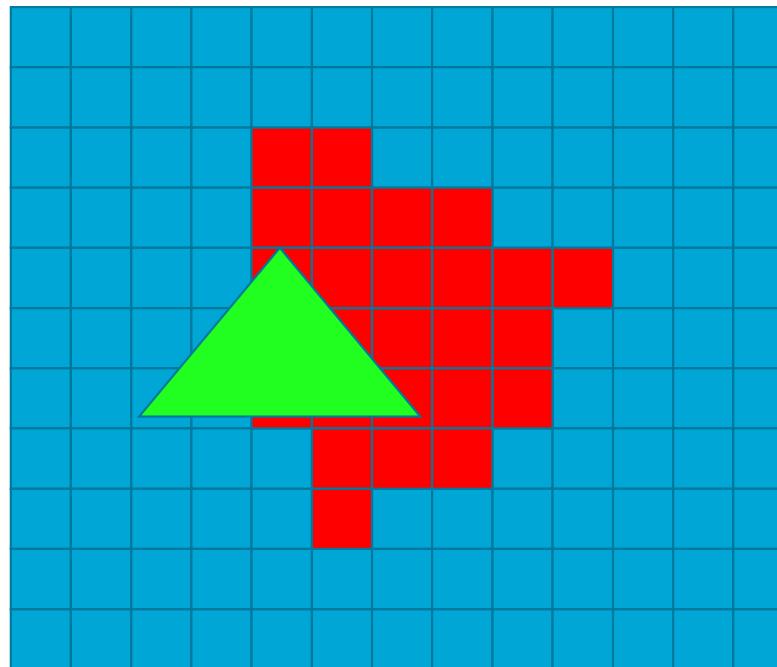
# Simplified Graphics Pipeline

- **Catch:** Let's look at a second triangle...



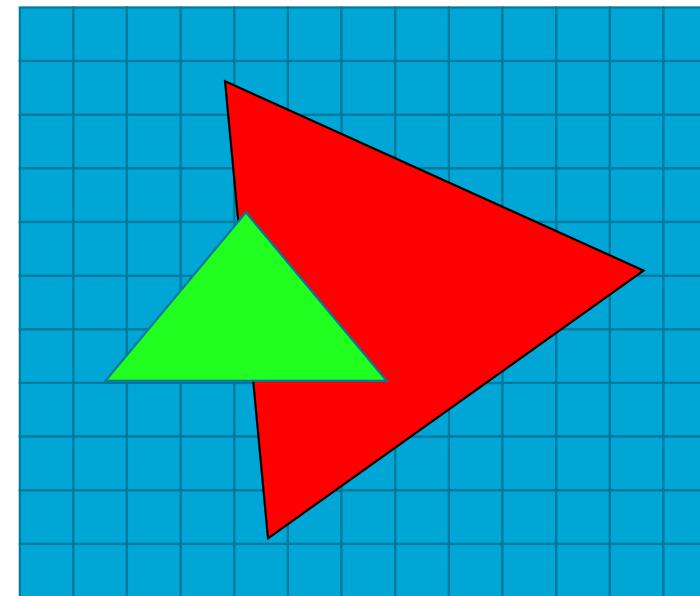
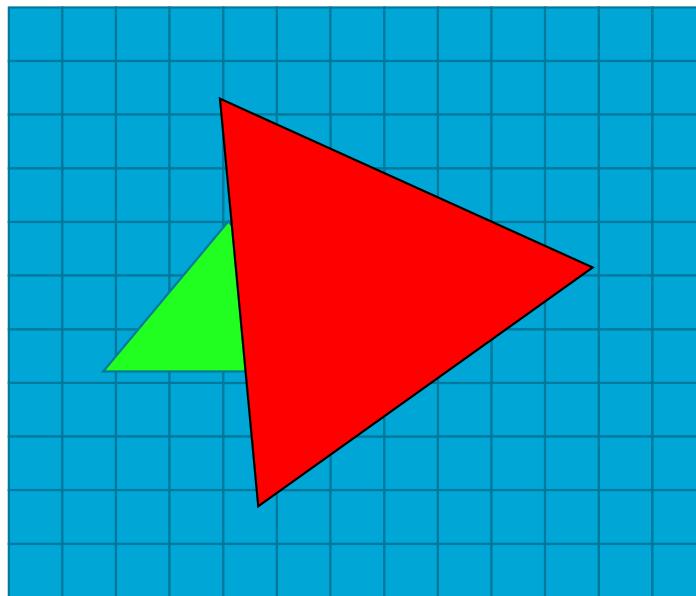
# Simplified Graphics Pipeline

- **Catch:** Let's look at a second triangle...



# Simplified Graphics Pipeline

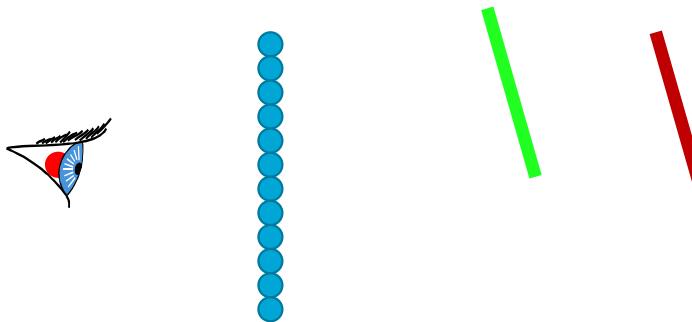
- **Catch:** Triangle drawing order changes result



As for ray tracing: we need to know the closest triangle in a pixel

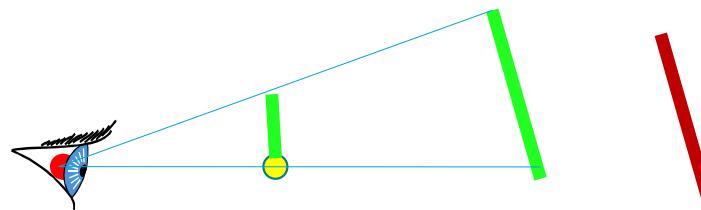
# Simplified Graphics Pipeline

- Depth Test: Avoid sorting!
- Store a depth in each pixel



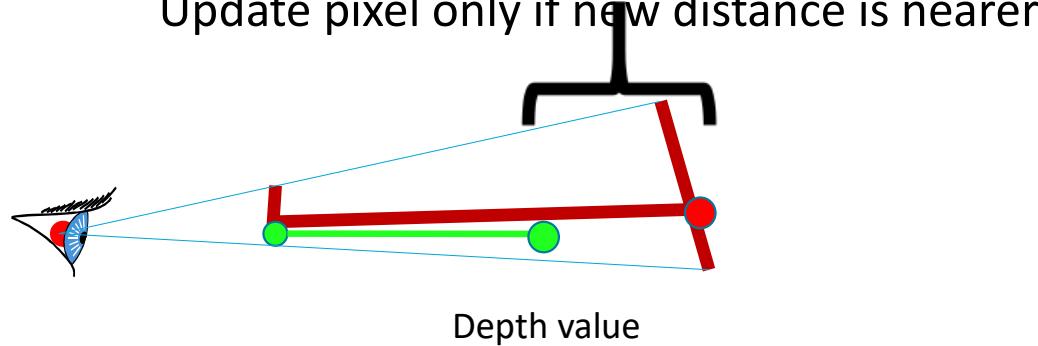
# Simplified Graphics Pipeline

- Depth Test: Avoid sorting!
- Store a depth in each pixel



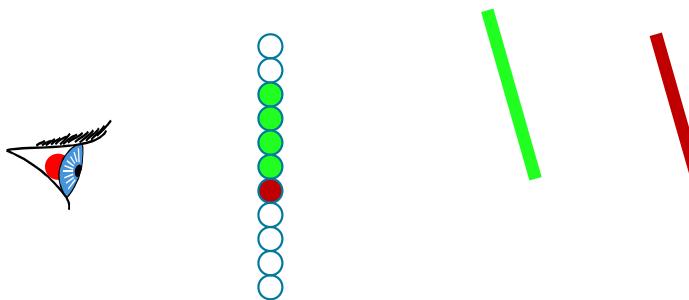
# Simplified Graphics Pipeline

- Depth Test: Avoid sorting!
- Store a depth in each pixel
  - Compare new distance to stored distance
  - Update pixel only if new distance is nearer



# Simplified Graphics Pipeline

- Depth Test: Avoid sorting!
- Store a depth in each pixel



## Cost of Rasterization

Algorithm:

For each triangle

```
projTri=projectTriangle(triangle)
```

```
fillPixels(projTri)
```

Cost = Triangles + “drawn pixels”

## Performance Analysis

Ray Tracing:

$$\text{Cost} = \text{Pixels} * \log(\text{Triangles}) + \text{structure}$$

vs.

Rasterization:

$$\text{Cost} = \text{Triangles} + \text{"drawn pixels"}$$

e.g., 100.000 triangles and a  $1000^2$  screen:

Raytracing:  $X+5 * 1.000.000$

Rasterization:  $100.000 + \text{"drawn pixels"}$

Raytracing/Rasterization :  $\sim 50$

## Performance Analysis – More Geometry...

Ray Tracing:

$$\text{Cost} = \text{Pixels} * \log(\text{Triangles}) + \text{structure}$$

vs.

Rasterization:

$$\text{Cost} = \text{Triangles} + \text{"drawn pixels"}$$

e.g., 100.000.000 triangles and a  $1000^2$  screen:

$$\text{Raytracing: } X+8 * 1.000.000$$

$$\text{Rasterization: } 100.000.000 + \text{"drawn pixels"}$$

$$\text{Raytracing/Rasterization : } \sim 0.1$$

But Rasterization can be made smarter  
with acceleration structures too...

## What complexity do we work with?

- Today's Games:
  - Often around 30K (Gears of War 3 and later)
  - Up to 500.000 triangles (Lamborghini Reventon - 562,786 Forza4)
- Today's Movies
  - Many Billions...





**How many triangles in this shot of Avatar 2 ?**



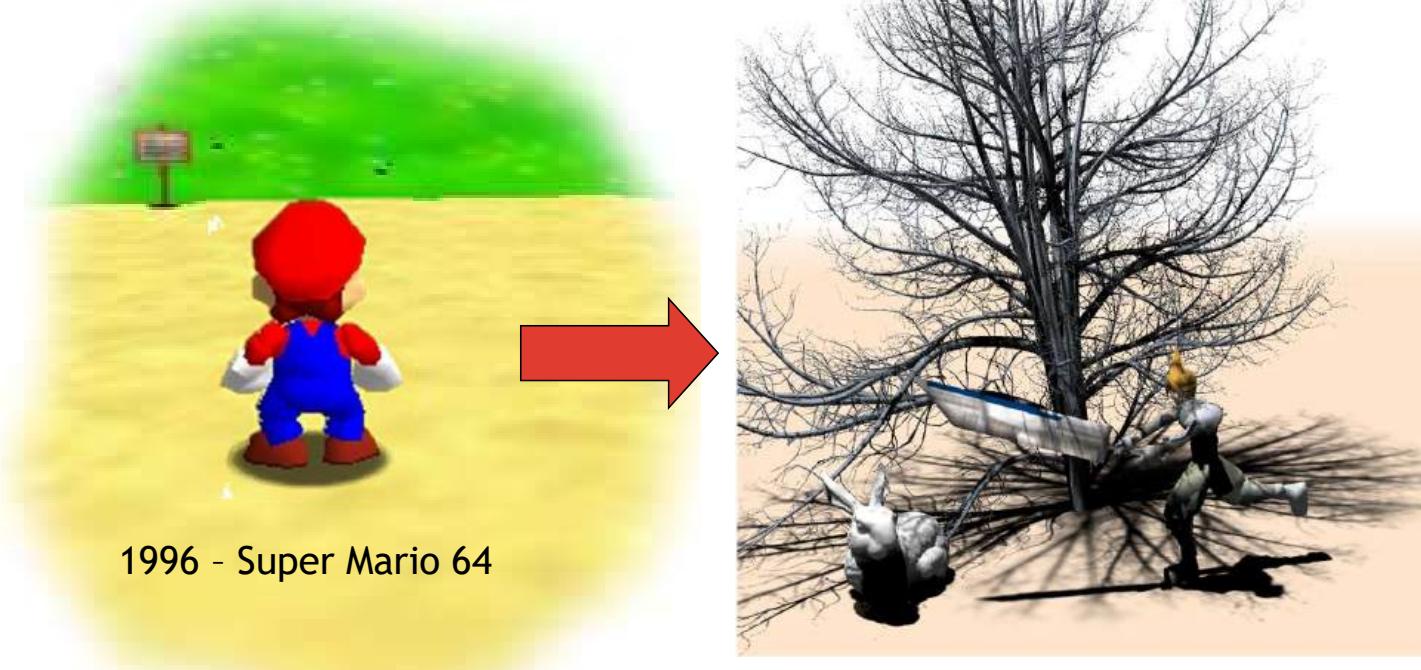
## Local Computations



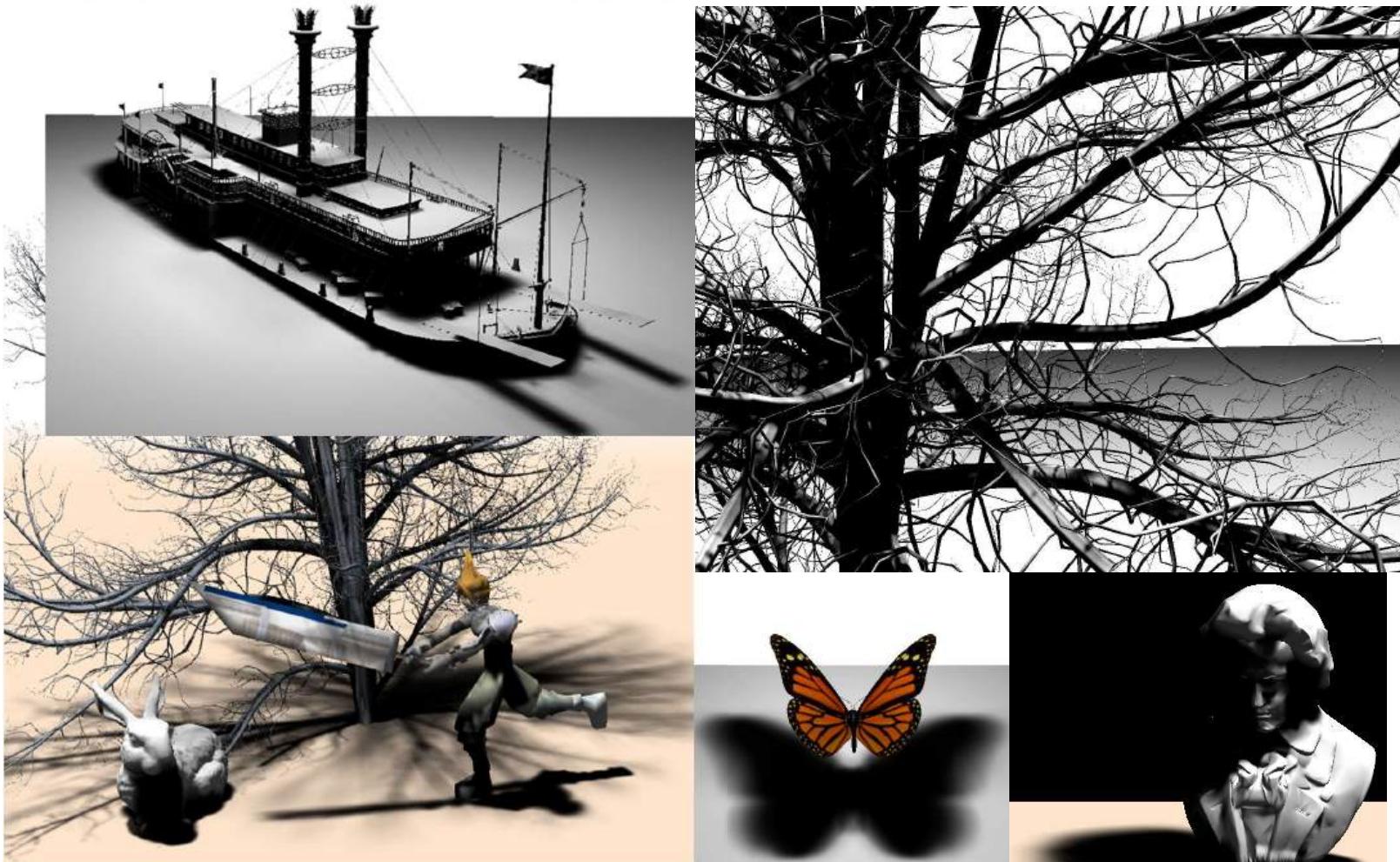
- Processor only knows its *current triangle (project or intersect)*  
generally **NOT enough**

## Non-Local Problems

- Challenges beyond local computations
  - Shadows



## Occlusion Textures for Plausible Soft Shadows



## Non-Local Problems

- Challenges beyond local computations
  - Transmittance

Standard shadow map



Transmittance shadow map



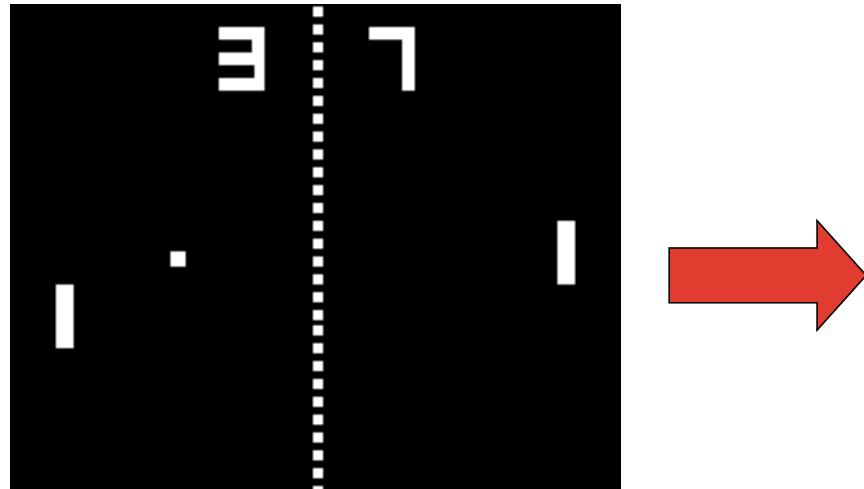
## Non-Local Problems

- Challenges beyond local computations
  - Refraction/Translucency



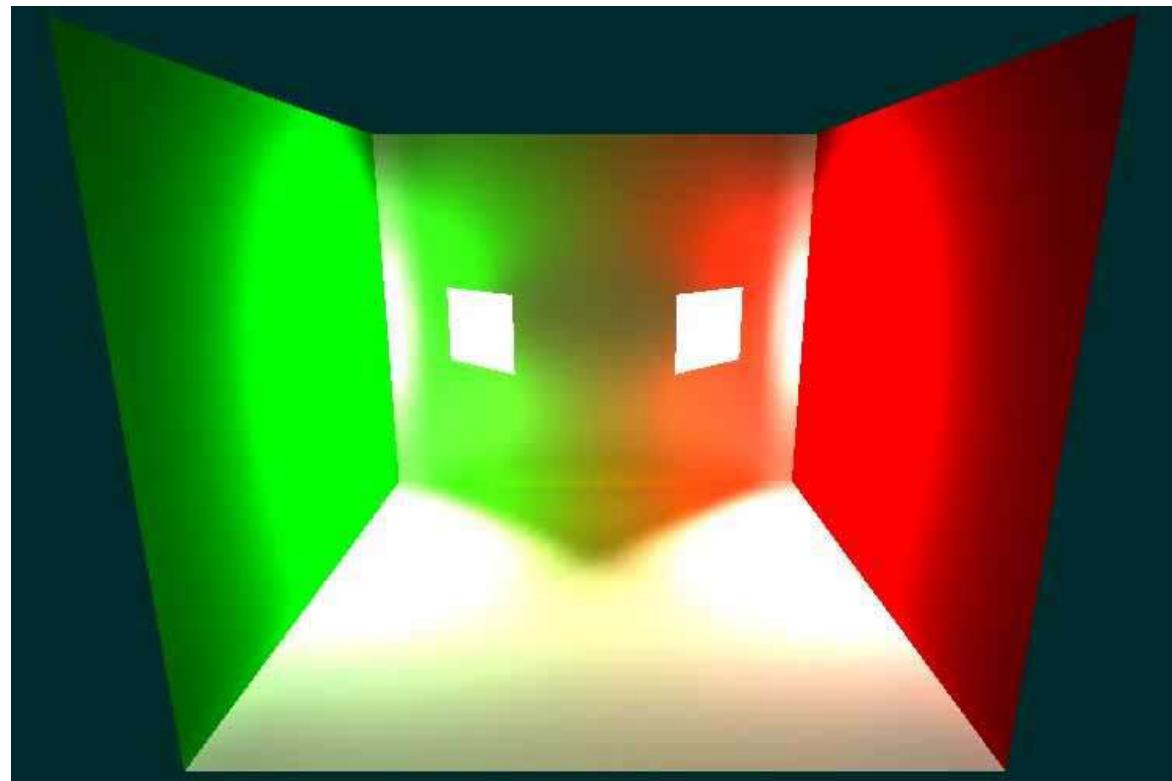
## Non-Local Problems

- Challenges beyond local computations
  - Collision Detection



1972 - Pong

# Global Illumination



In all examples  
SM resolution: 1024x1024  
scattering: < 3 ms

We support dynamic lights and viewpoints.

# Resume

- Introduction
  - Definition of Computer Graphics
  - Basics of how we perceive images/colors
- Creating images on a computer:
  - Raytracing
  - Rasterization (including Depth Buffer)



Thank you very much  
for your attention!

# Computer Graphics – Linear Algebra Recap

Martin Skrodzki

Computer Graphics and Visualization Group  
Delft University of Technology, The  
Netherlands

November 13th, 2024



**Algebra**

**Linear  
Algebra**

# Computer Graphics – Why linear algebra?

<https://vevox.app/#/m/106717265>

# Computer Graphics – Why linear algebra?

<https://vevox.app/#/m/106717265>

According to Google:

Movies



Jurassic Park (1993)



Movie stills from Jurassic Park (1993)



The Matrix (1999)

Motion Capture



Andy Serkis in The Two Towers

Games

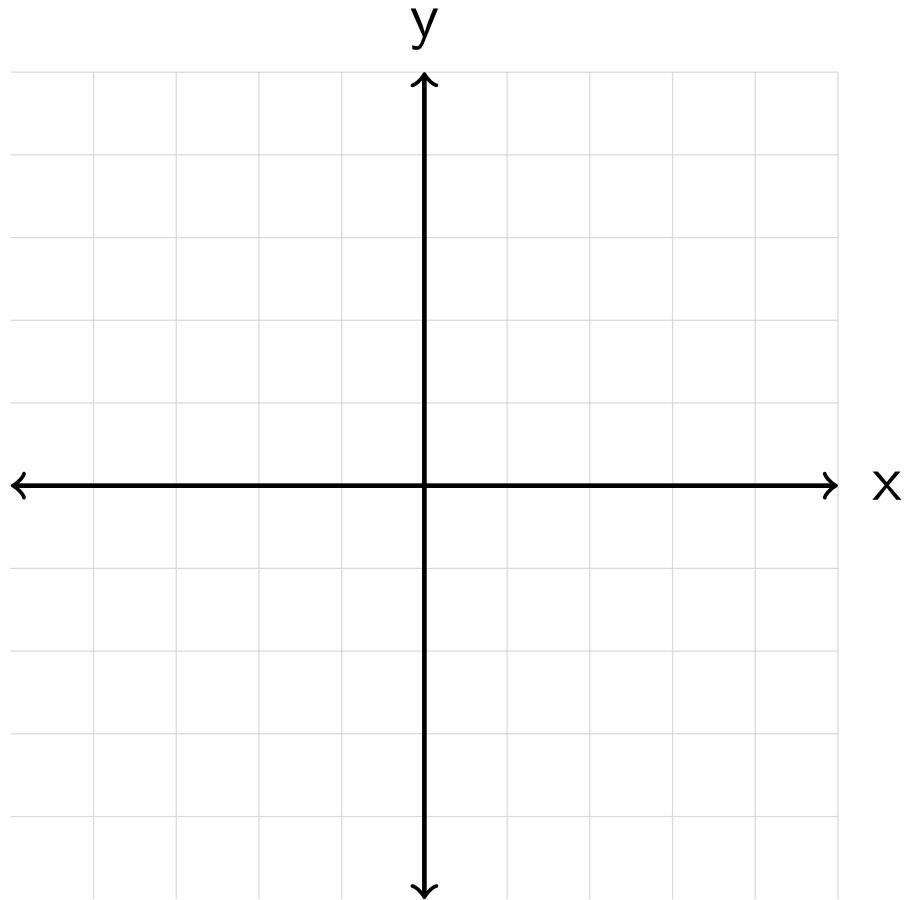


Unreal Engine 5 Demo Realtime in PS5 (2020)

# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265



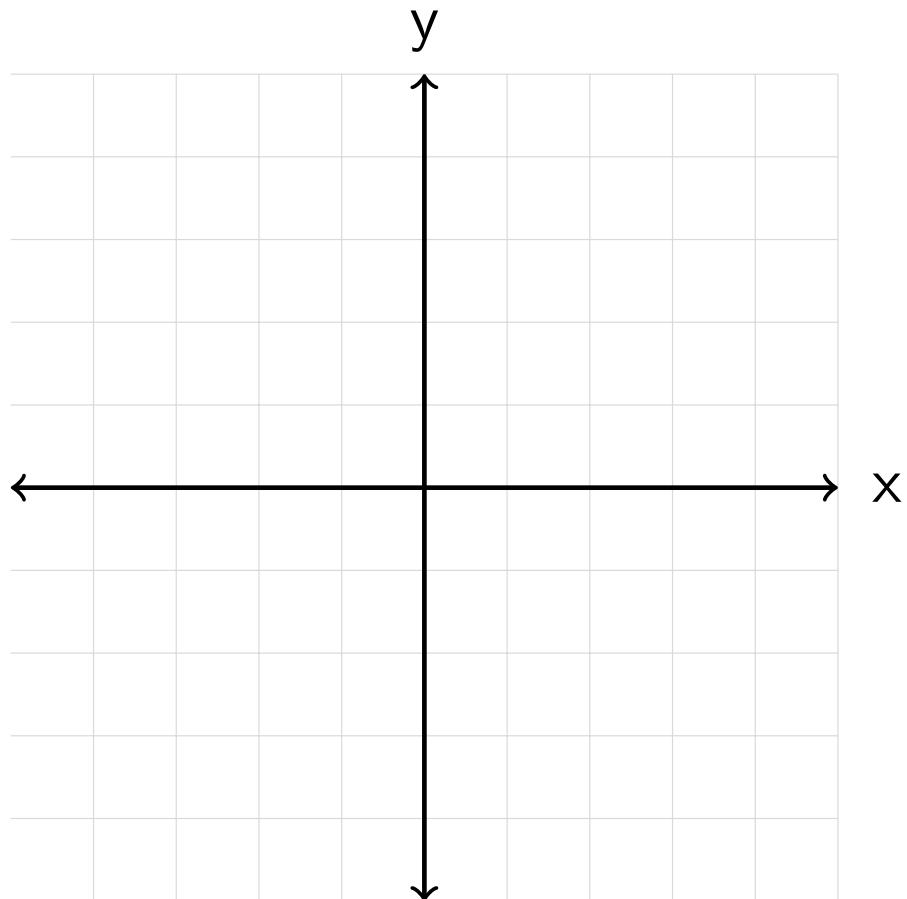
# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$



# Addition and Subtraction of Vectors

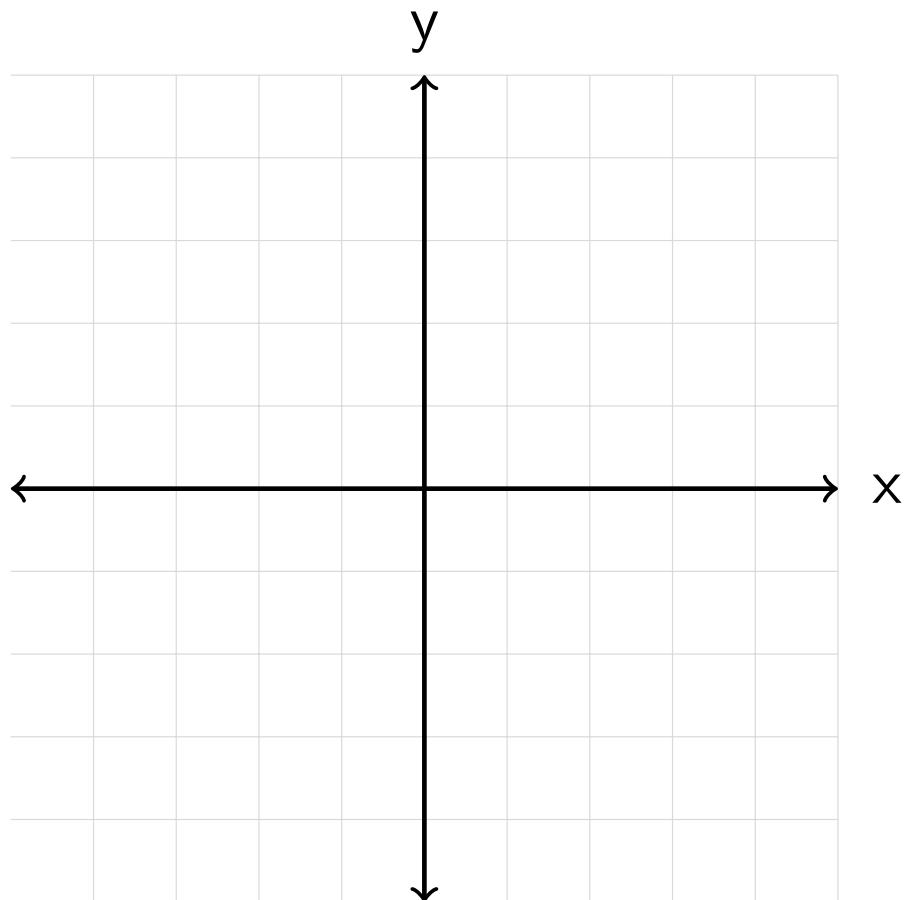
[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$



# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

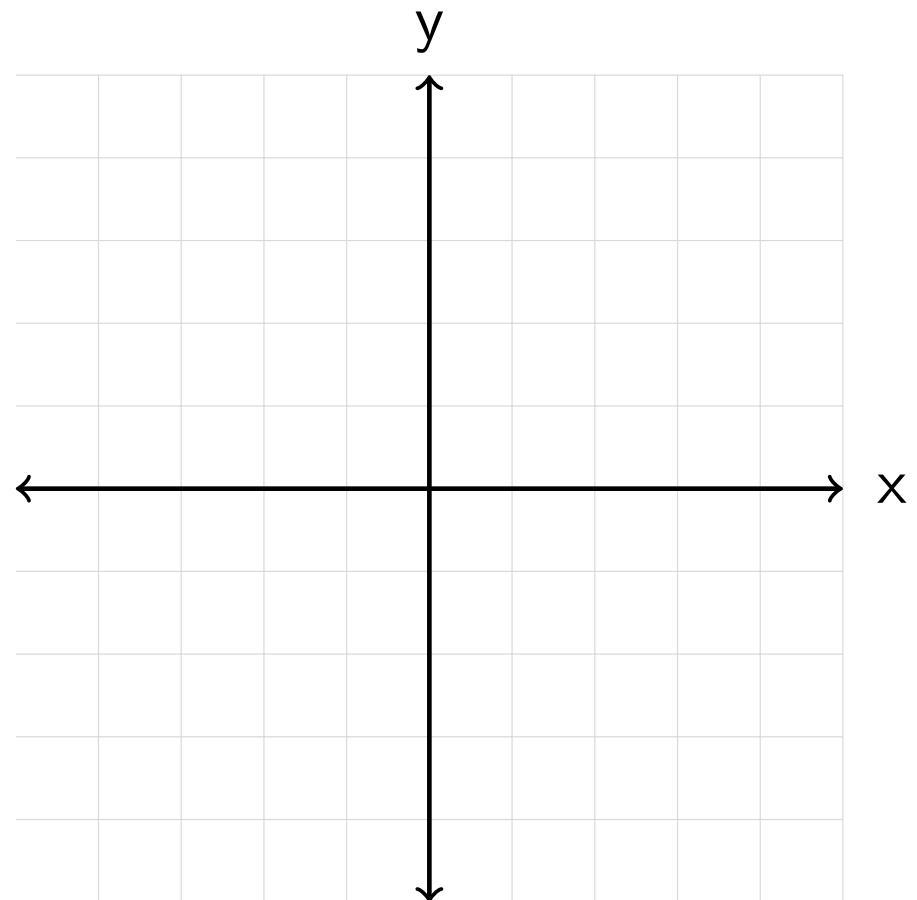
For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$

add them by

$$\mathbf{v} + \mathbf{w} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_3 + w_3 \end{pmatrix}.$$



# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

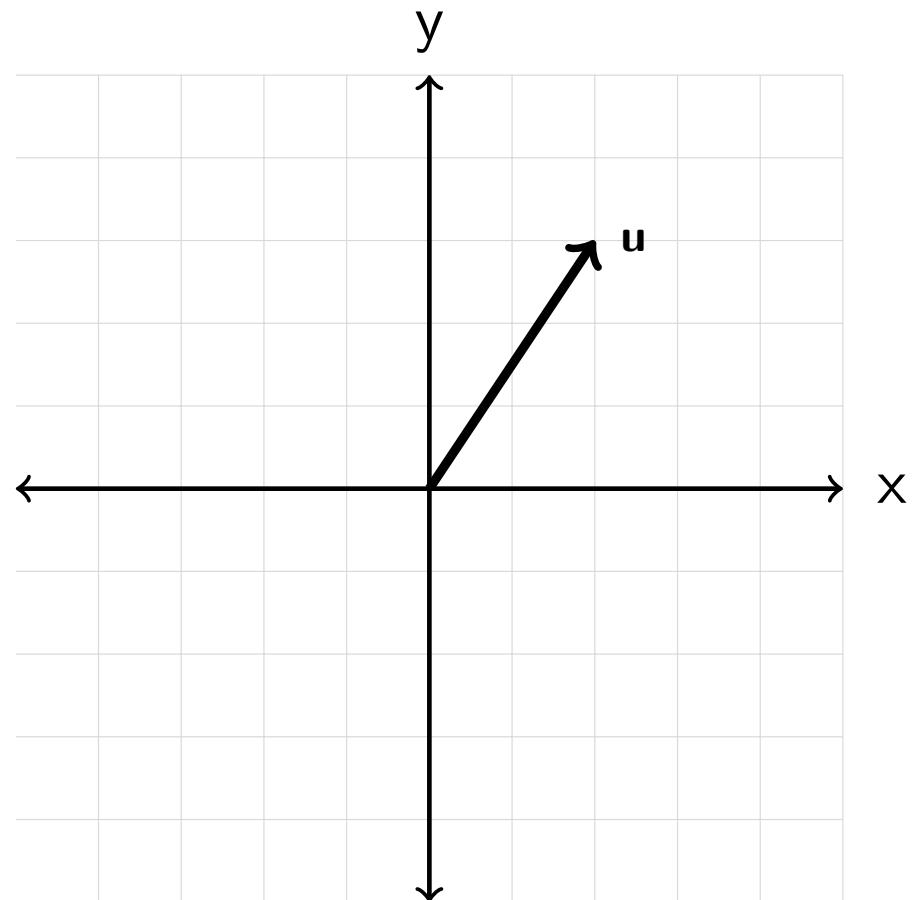
For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$

add them by

$$\mathbf{v} + \mathbf{w} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_3 + w_3 \end{pmatrix}.$$



# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

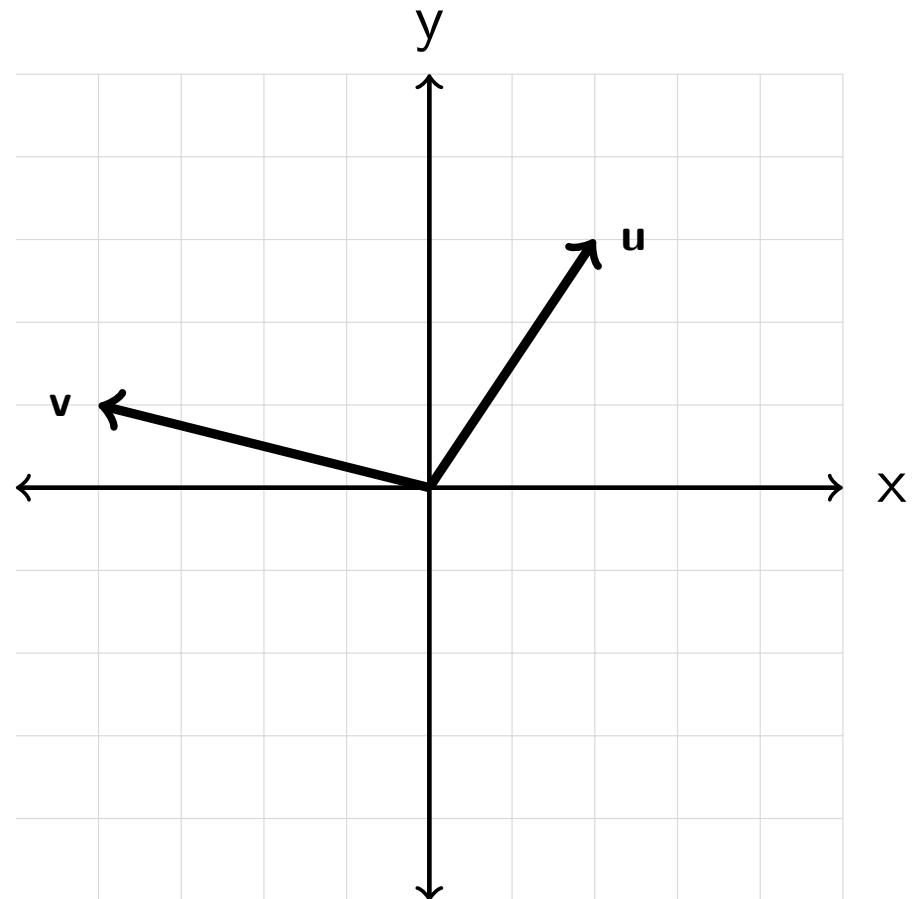
For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$

add them by

$$\mathbf{v} + \mathbf{w} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_3 + w_3 \end{pmatrix}.$$



# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

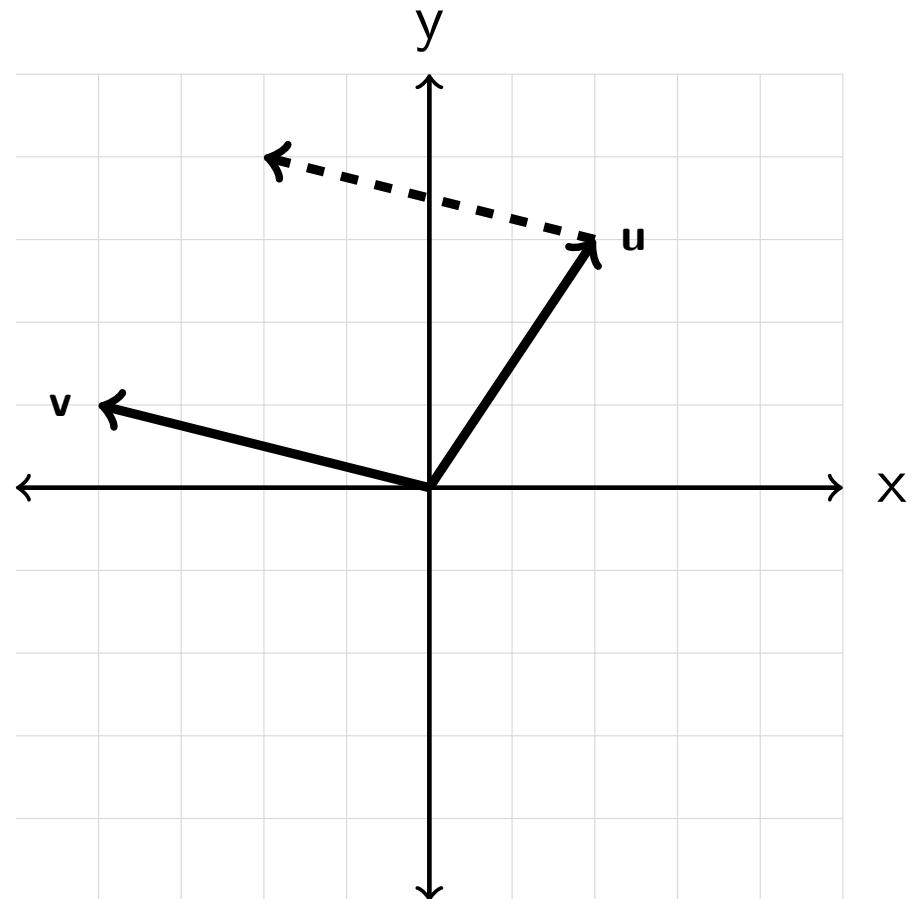
For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$

add them by

$$\mathbf{v} + \mathbf{w} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_3 + w_3 \end{pmatrix}.$$



# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

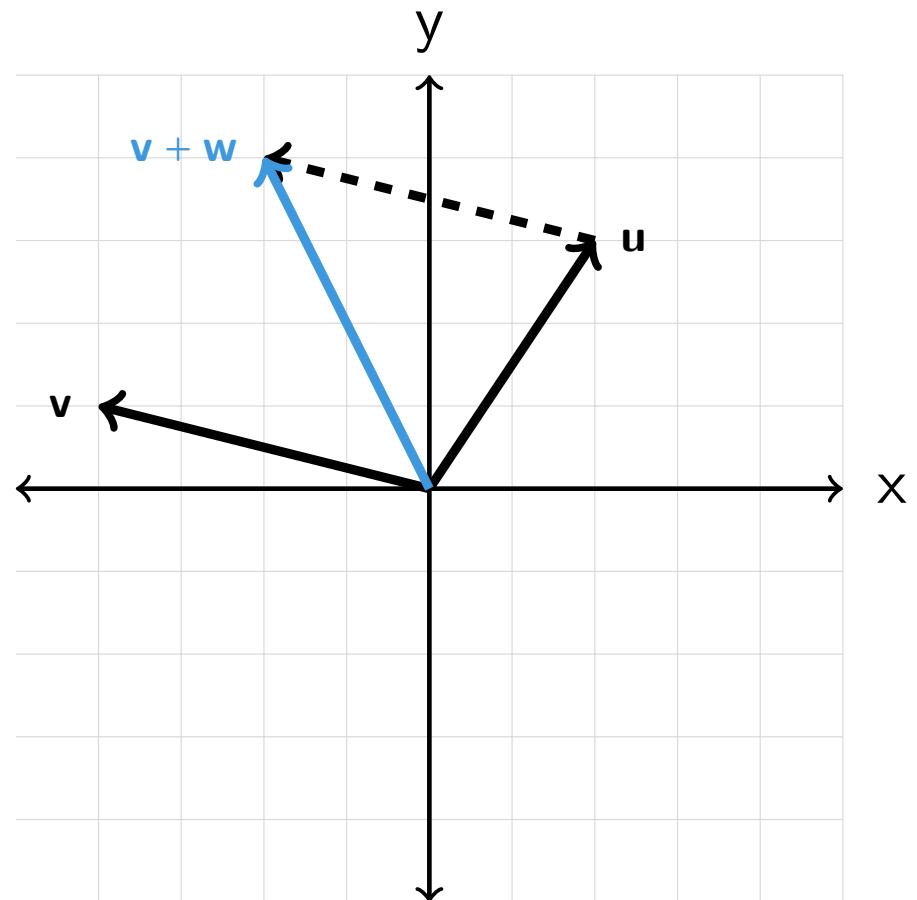
For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$

add them by

$$\mathbf{v} + \mathbf{w} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_3 + w_3 \end{pmatrix}.$$



# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

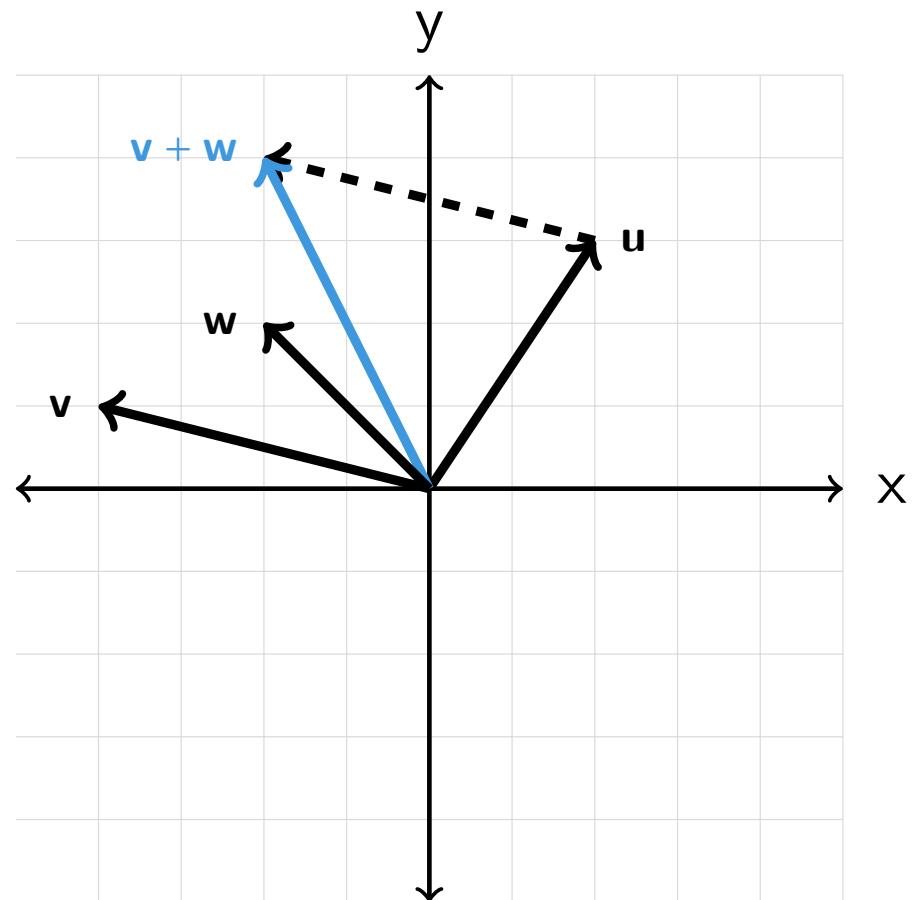
For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$

add them by

$$\mathbf{v} + \mathbf{w} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_3 + w_3 \end{pmatrix}.$$



# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

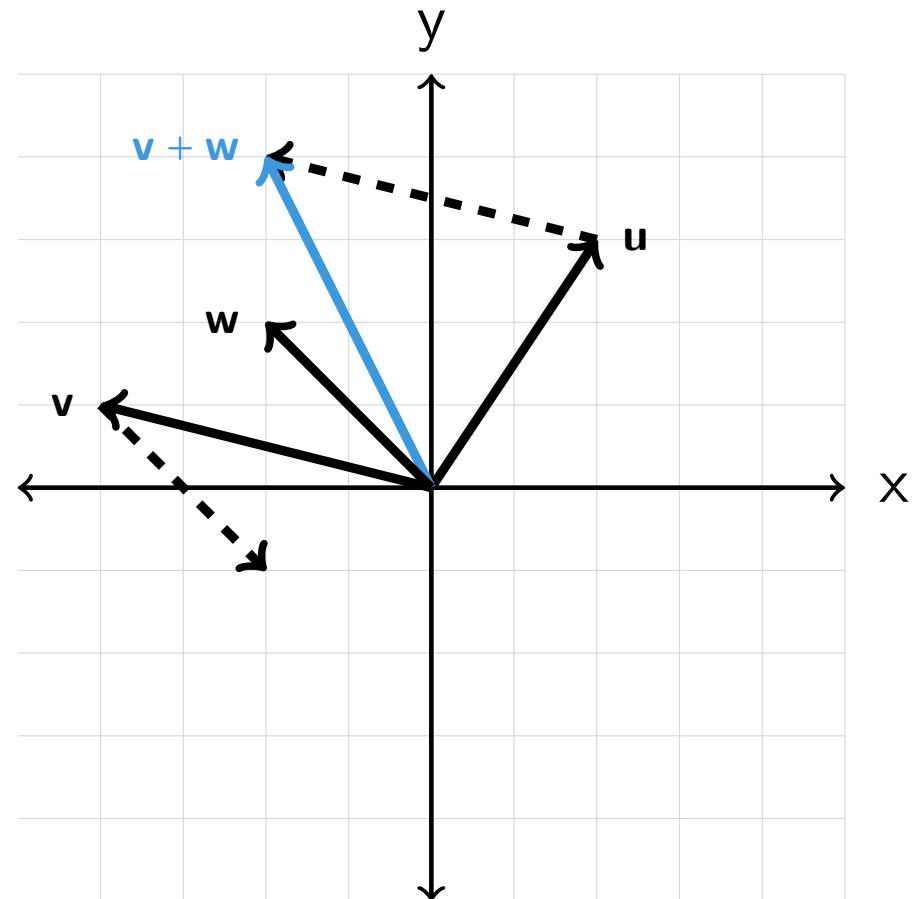
For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$

add them by

$$\mathbf{v} + \mathbf{w} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_3 + w_3 \end{pmatrix}.$$



# Addition and Subtraction of Vectors

[https:](https://vevox.app/#/m/106717265)

//vevox.app/#/m/106717265

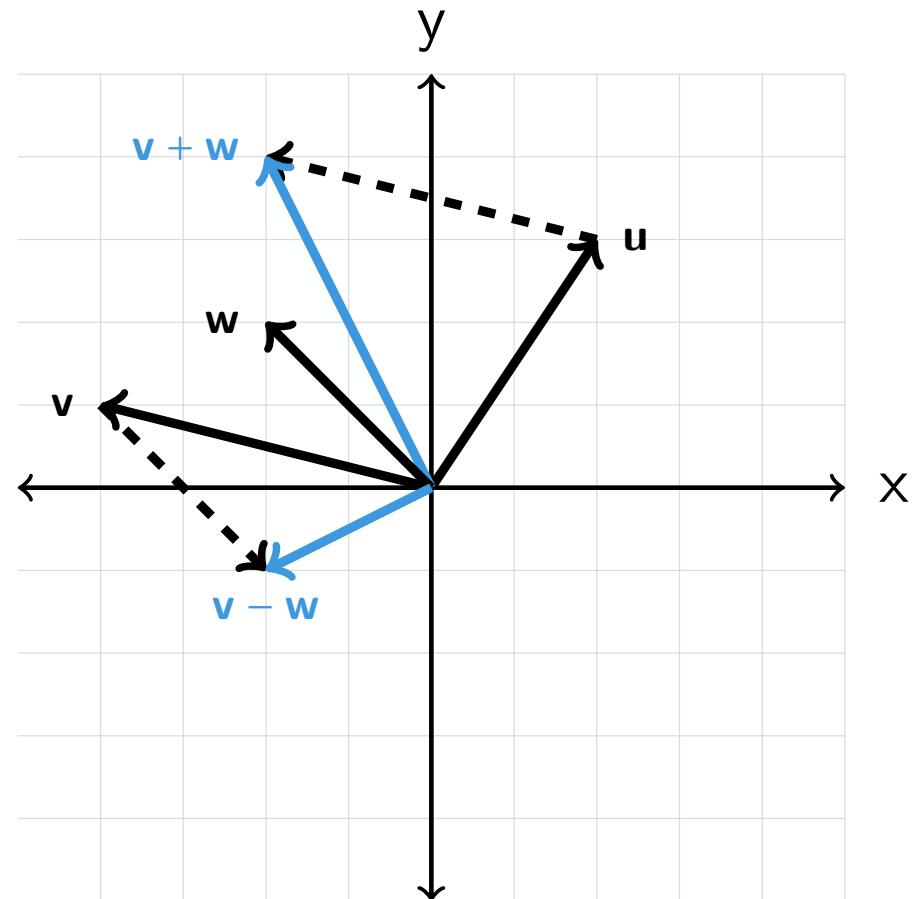
For two vectors

$$\mathbf{v} = (v_1, v_2, v_3)^t = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

$$\mathbf{w} = (w_1, w_2, w_3)^t = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix},$$

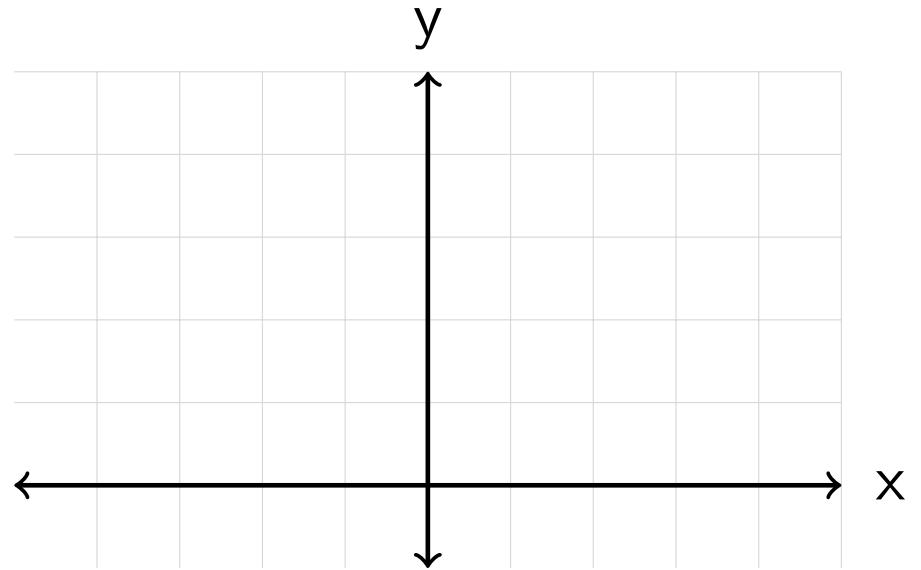
add them by

$$\mathbf{v} + \mathbf{w} = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_3 + w_3 \end{pmatrix}.$$



# Length of a vector

<https://vevox.app/#/m/106717265>



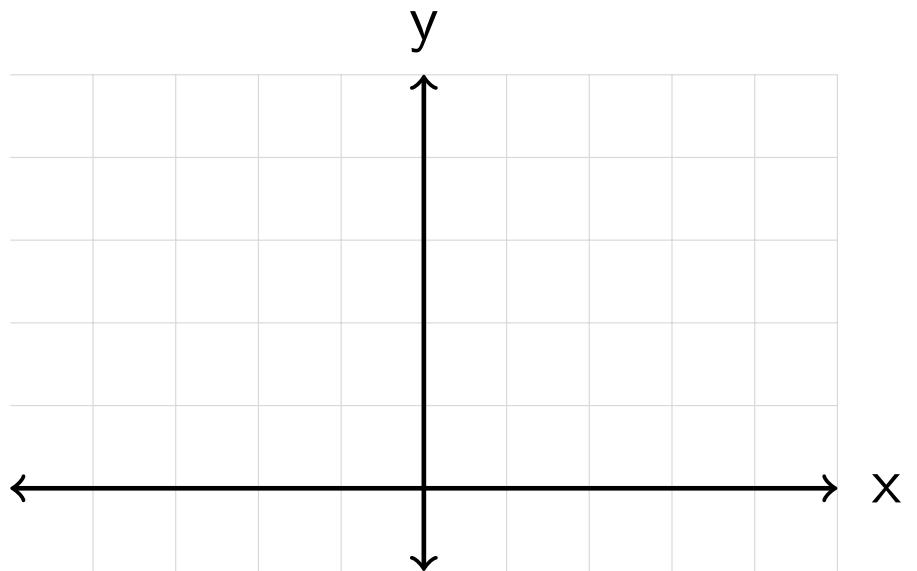
# Length of a vector

<https://vevox.app/#/m/106717265>

aka Pythagoras' Theorem both in  
2D

$$\mathbf{v} = (v_1, v_2)^t \in \mathbb{R}^2,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2},$$



# Length of a vector

<https://vevox.app/#/m/106717265>

aka Pythagoras' Theorem both in  
2D

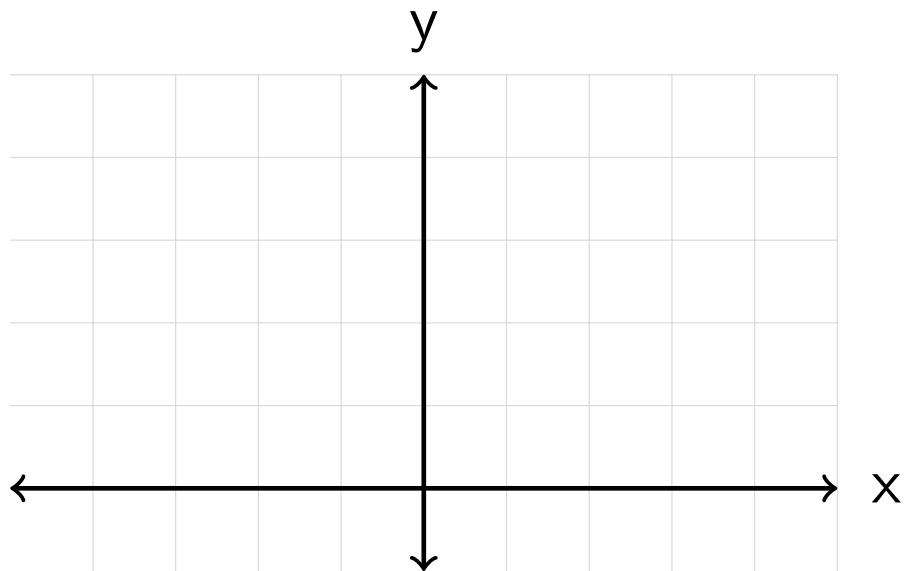
$$\mathbf{v} = (v_1, v_2)^t \in \mathbb{R}^2,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2},$$

and 3D

$$\mathbf{v} = (v_1, v_2, v_3)^t \in \mathbb{R}^3,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2}.$$



# Length of a vector

<https://vevox.app/#/m/106717265>

aka Pythagoras' Theorem both in  
2D

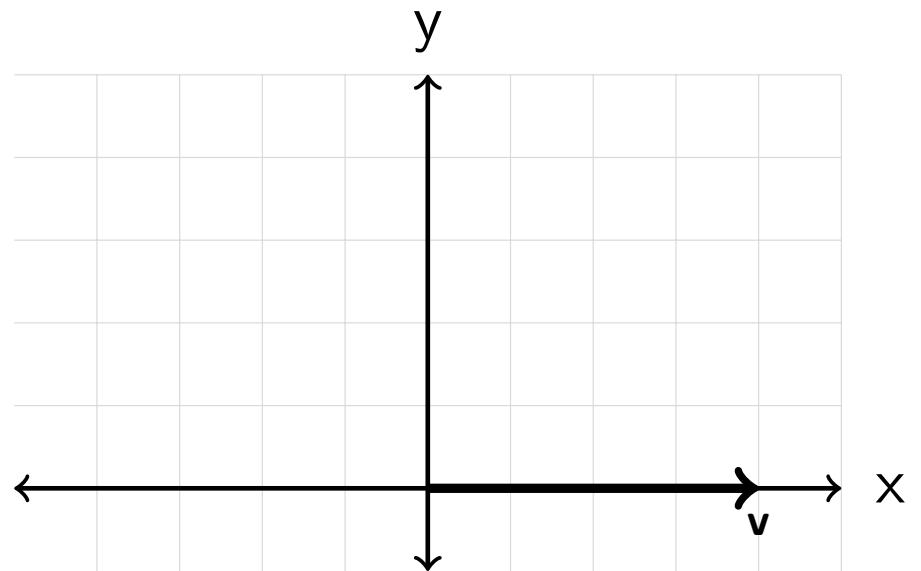
$$\mathbf{v} = (v_1, v_2)^t \in \mathbb{R}^2,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2},$$

and 3D

$$\mathbf{v} = (v_1, v_2, v_3)^t \in \mathbb{R}^3,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2}.$$



# Length of a vector

<https://vevox.app/#/m/106717265>

aka Pythagoras' Theorem both in  
2D

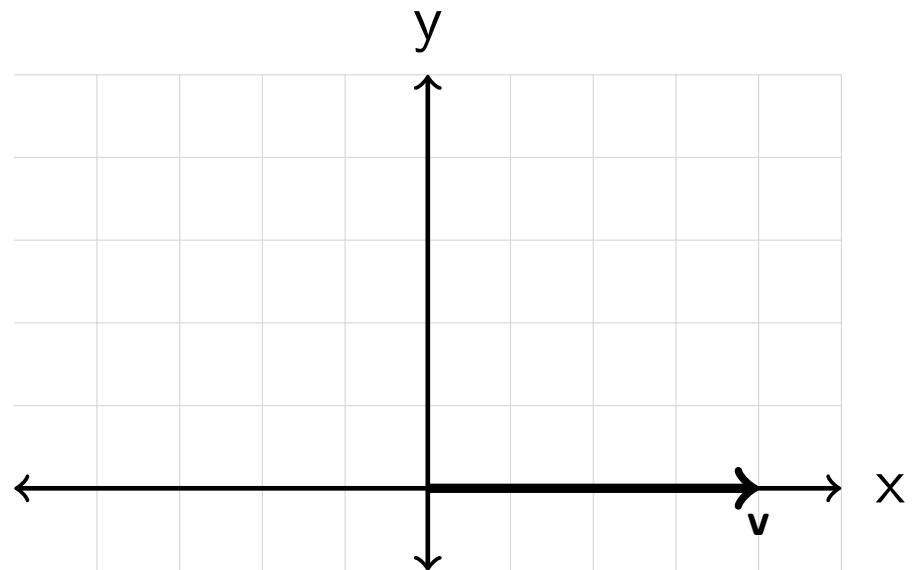
$$\mathbf{v} = (v_1, v_2)^t \in \mathbb{R}^2,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2},$$

and 3D

$$\mathbf{v} = (v_1, v_2, v_3)^t \in \mathbb{R}^3,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2}.$$



$$\|\mathbf{v}\|_2 = \left\| \begin{pmatrix} 2 \\ 0 \end{pmatrix} \right\|_2 = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

# Length of a vector

<https://vevox.app/#/m/106717265>

aka Pythagoras' Theorem both in  
2D

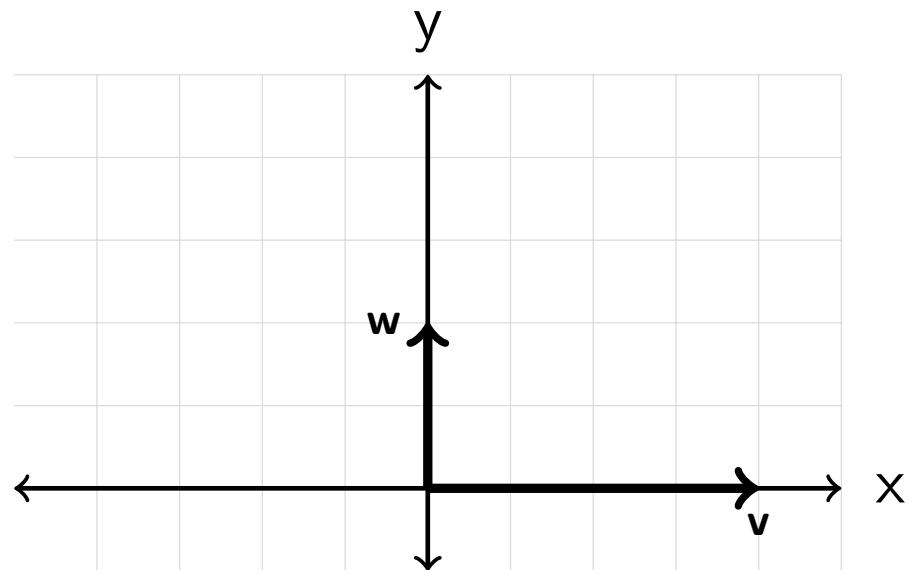
$$\mathbf{v} = (v_1, v_2)^t \in \mathbb{R}^2,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2},$$

and 3D

$$\mathbf{v} = (v_1, v_2, v_3)^t \in \mathbb{R}^3,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2}.$$



$$\|\mathbf{v}\|_2 = \left\| \begin{pmatrix} 2 \\ 0 \end{pmatrix} \right\|_2 = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

# Length of a vector

<https://vevox.app/#/m/106717265>

aka Pythagoras' Theorem both in  
2D

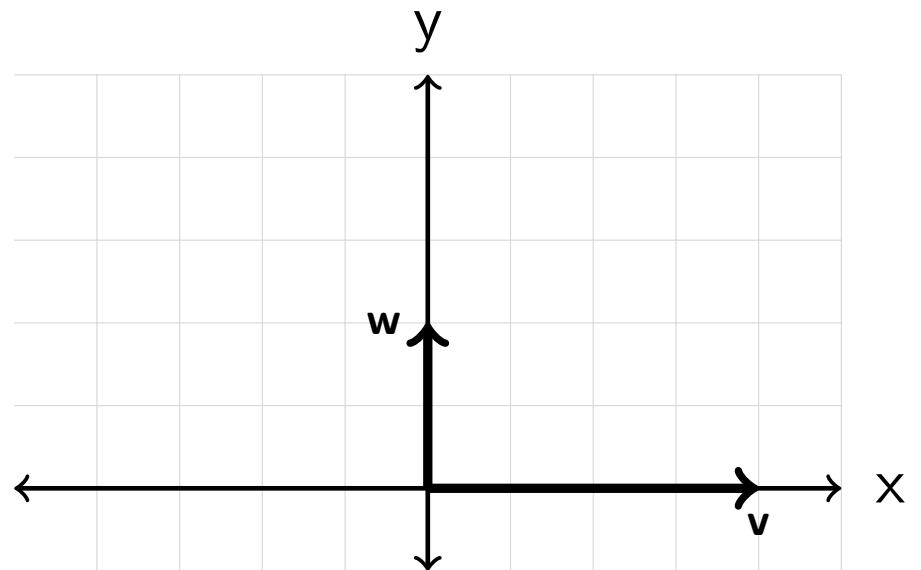
$$\mathbf{v} = (v_1, v_2)^t \in \mathbb{R}^2,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2},$$

and 3D

$$\mathbf{v} = (v_1, v_2, v_3)^t \in \mathbb{R}^3,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2}.$$



$$\|\mathbf{v}\|_2 = \left\| \begin{pmatrix} 2 \\ 0 \end{pmatrix} \right\|_2 = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

$$\|\mathbf{w}\|_2 = \left\| \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\|_2 = \sqrt{0^2 + 1^2} = \sqrt{1} = 1$$

# Length of a vector

<https://vevox.app/#/m/106717265>

aka Pythagoras' Theorem both in  
2D

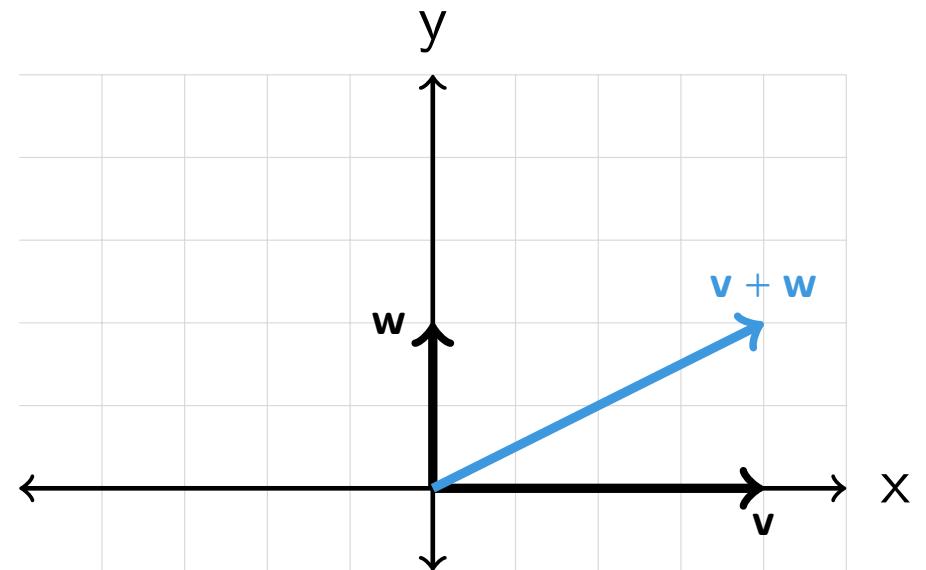
$$\mathbf{v} = (v_1, v_2)^t \in \mathbb{R}^2,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2},$$

and 3D

$$\mathbf{v} = (v_1, v_2, v_3)^t \in \mathbb{R}^3,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2}.$$



$$\|\mathbf{v}\|_2 = \left\| \begin{pmatrix} 2 \\ 0 \end{pmatrix} \right\|_2 = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

$$\|\mathbf{w}\|_2 = \left\| \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\|_2 = \sqrt{0^2 + 1^2} = \sqrt{1} = 1$$

# Length of a vector

<https://vevox.app/#/m/106717265>

aka Pythagoras' Theorem both in  
2D

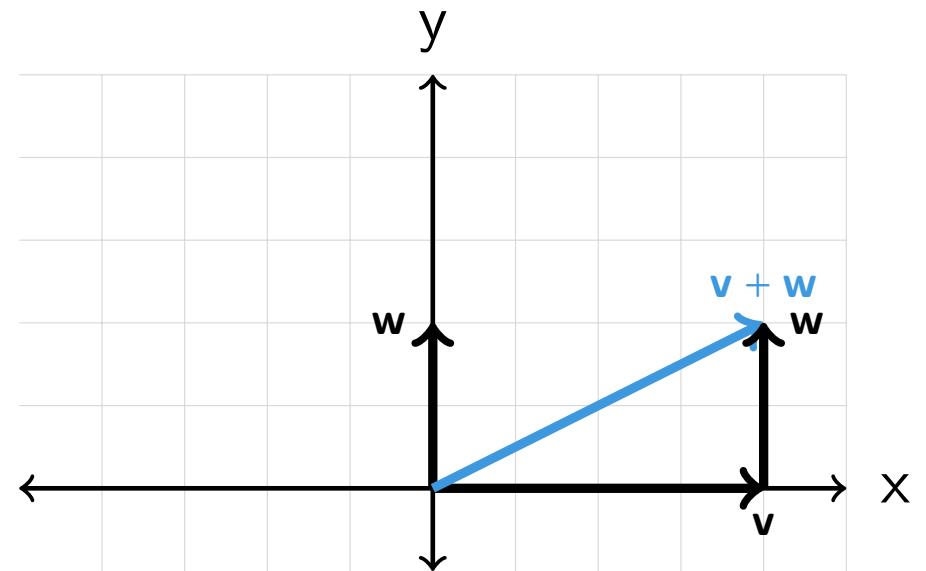
$$\mathbf{v} = (v_1, v_2)^t \in \mathbb{R}^2,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2},$$

and 3D

$$\mathbf{v} = (v_1, v_2, v_3)^t \in \mathbb{R}^3,$$

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2}.$$



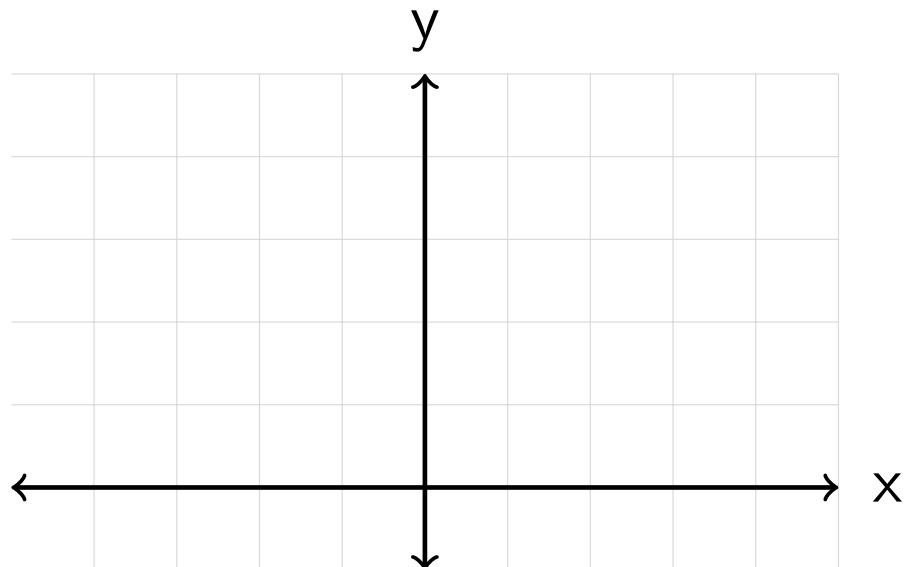
$$\|\mathbf{v}\|_2 = \left\| \begin{pmatrix} 2 \\ 0 \end{pmatrix} \right\|_2 = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

$$\|\mathbf{w}\|_2 = \left\| \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\|_2 = \sqrt{0^2 + 1^2} = \sqrt{1} = 1$$

$$\|\mathbf{v} + \mathbf{w}\|_2 = \left\| \begin{pmatrix} 2 \\ 1 \end{pmatrix} \right\|_2 = \sqrt{2^2 + 1^2} = \sqrt{5}$$

# Combination: Distance between points

<https://vevox.app/#/m/106717265>

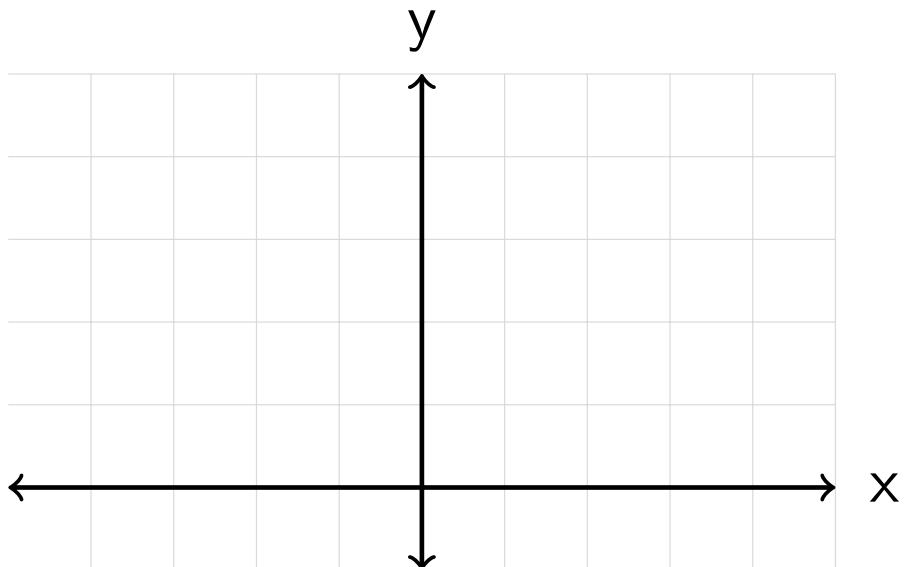


# Combination: Distance between points

<https://vevox.app/#/m/106717265>

Having subtraction and length, can compute the distance between two points  $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$  as

$$\|\mathbf{v} - \mathbf{w}\|_2 = \|\mathbf{w} - \mathbf{v}\|_2.$$

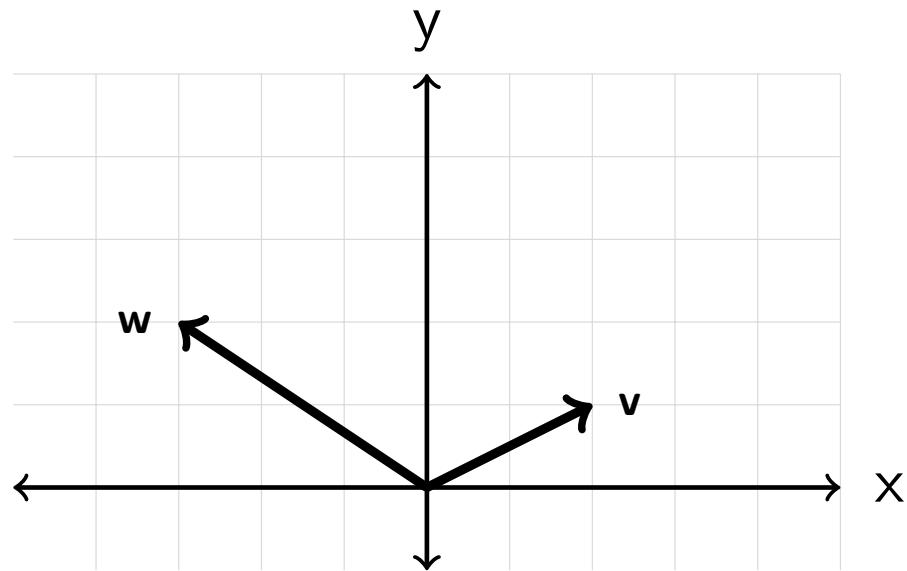


# Combination: Distance between points

<https://vevox.app/#/m/106717265>

Having subtraction and length, can compute the distance between two points  $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$  as

$$\|\mathbf{v} - \mathbf{w}\|_2 = \|\mathbf{w} - \mathbf{v}\|_2.$$

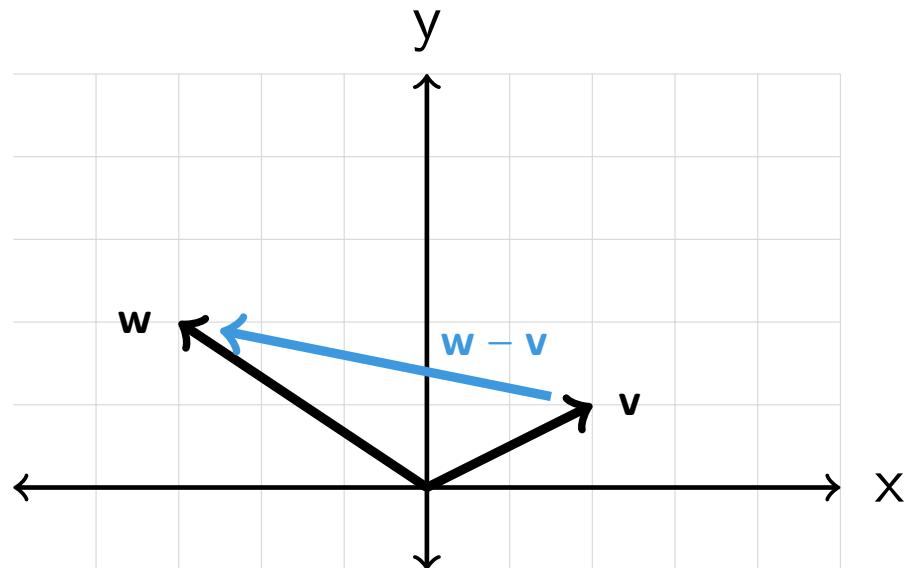


# Combination: Distance between points

<https://vevox.app/#/m/106717265>

Having subtraction and length, can compute the distance between two points  $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$  as

$$\|\mathbf{v} - \mathbf{w}\|_2 = \|\mathbf{w} - \mathbf{v}\|_2.$$

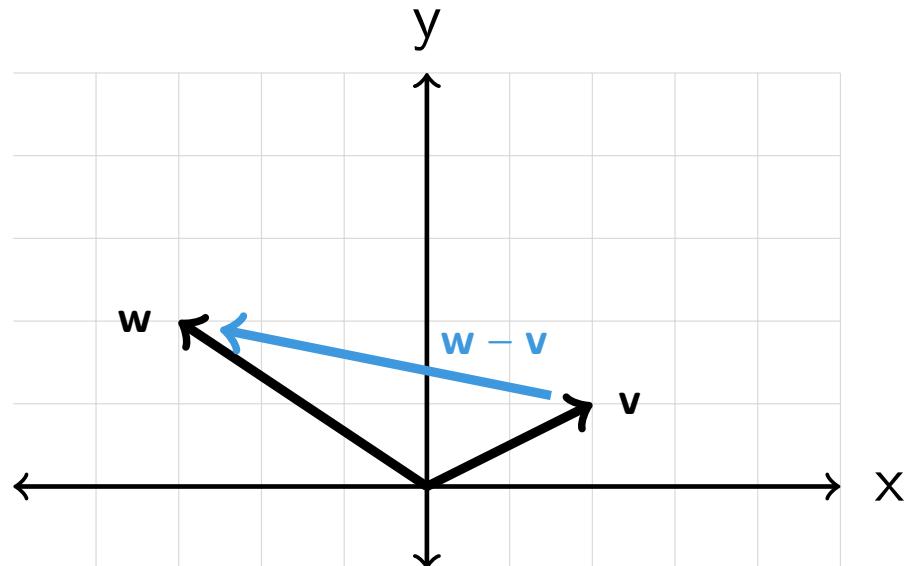


# Combination: Distance between points

<https://vevox.app/#/m/106717265>

Having subtraction and length, can compute the distance between two points  $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$  as

$$\|\mathbf{v} - \mathbf{w}\|_2 = \|\mathbf{w} - \mathbf{v}\|_2.$$



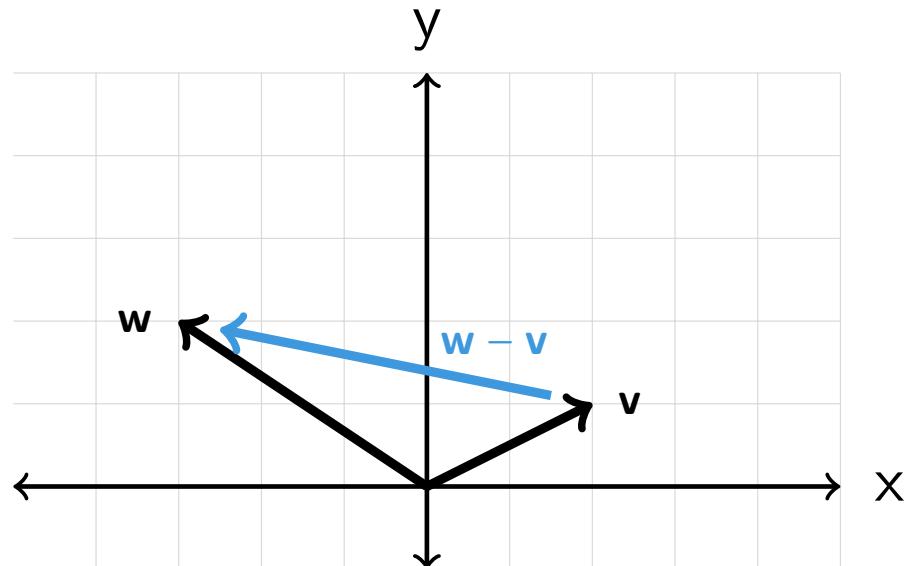
$$\|\mathbf{w} - \mathbf{v}\|_2 = \left\| \begin{pmatrix} -\frac{3}{2} \\ 1 \end{pmatrix} - \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} -\frac{5}{2} \\ \frac{1}{2} \end{pmatrix} \right\|_2$$

# Combination: Distance between points

<https://vevox.app/#/m/106717265>

Having subtraction and length, can compute the distance between two points  $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$  as

$$\|\mathbf{v} - \mathbf{w}\|_2 = \|\mathbf{w} - \mathbf{v}\|_2.$$



$$\begin{aligned}\|\mathbf{w} - \mathbf{v}\|_2 &= \left\| \begin{pmatrix} -\frac{3}{2} \\ 1 \end{pmatrix} - \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} -\frac{5}{2} \\ \frac{1}{2} \end{pmatrix} \right\|_2 \\ &= \sqrt{\left(-\frac{5}{2}\right)^2 + \left(\frac{1}{2}\right)^2} = \sqrt{\frac{26}{4}} \\ &\approx 2.54951\end{aligned}$$

# Dot Product

<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$

# Dot Product

<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$

Analogously for 2D:  
 $\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

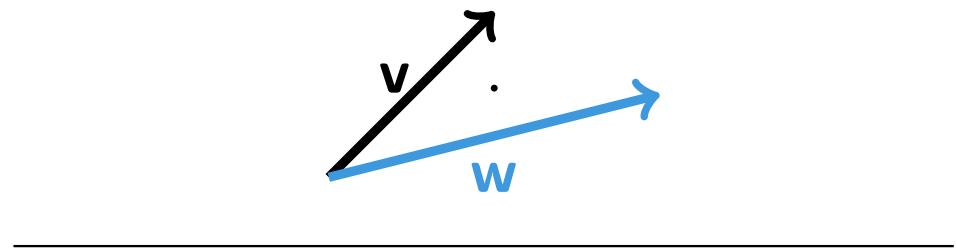
$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$

# Dot Product

<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$



Analogously for 2D:  
 $\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

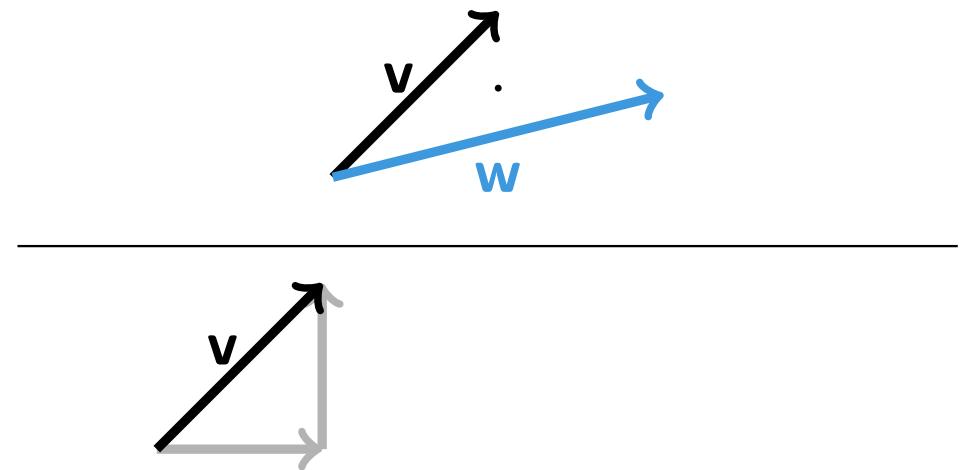
$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$

# Dot Product

<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$



Analogously for 2D:  
 $\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

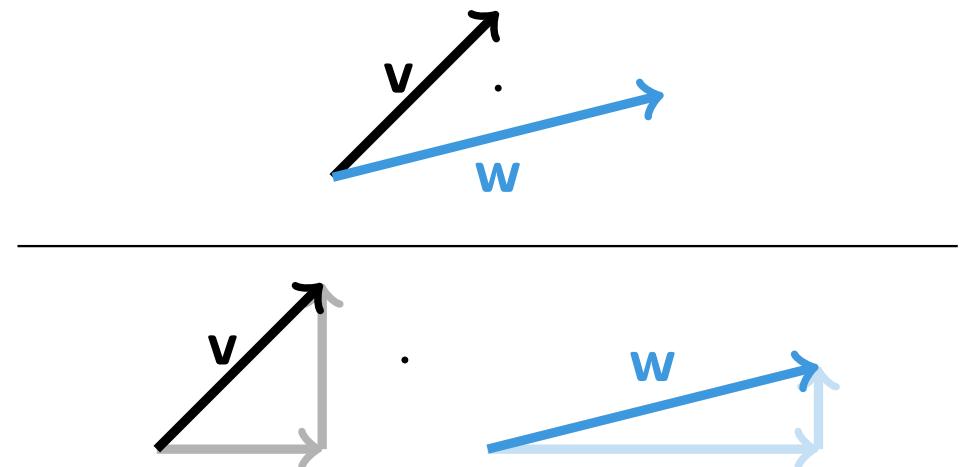
$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$

# Dot Product

<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$



Analogously for 2D:

$\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$

# Dot Product

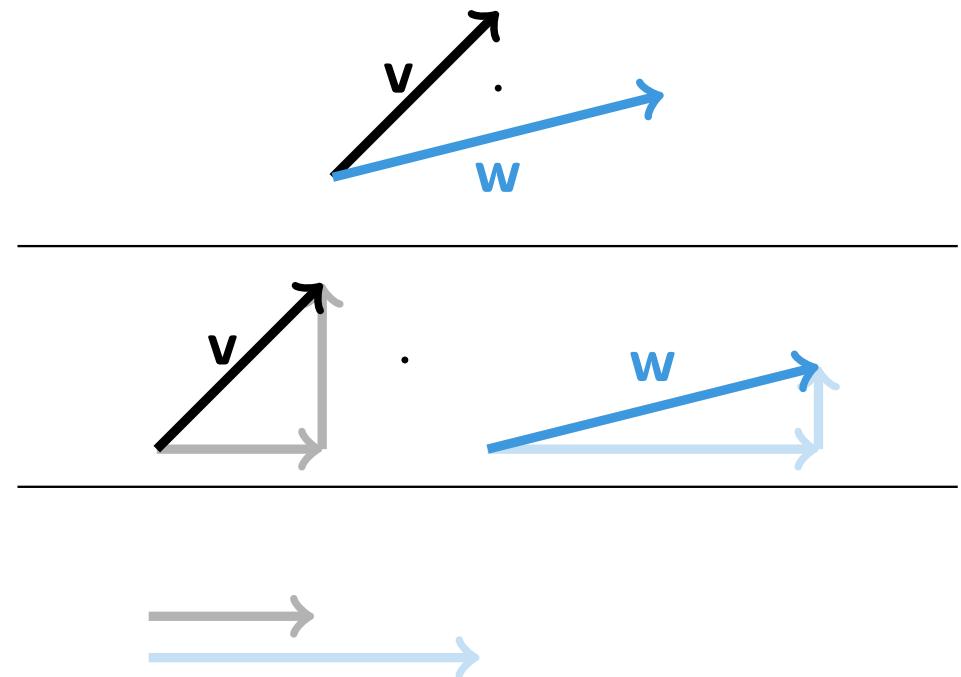
<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$

Analogously for 2D:  
 $\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$



# Dot Product

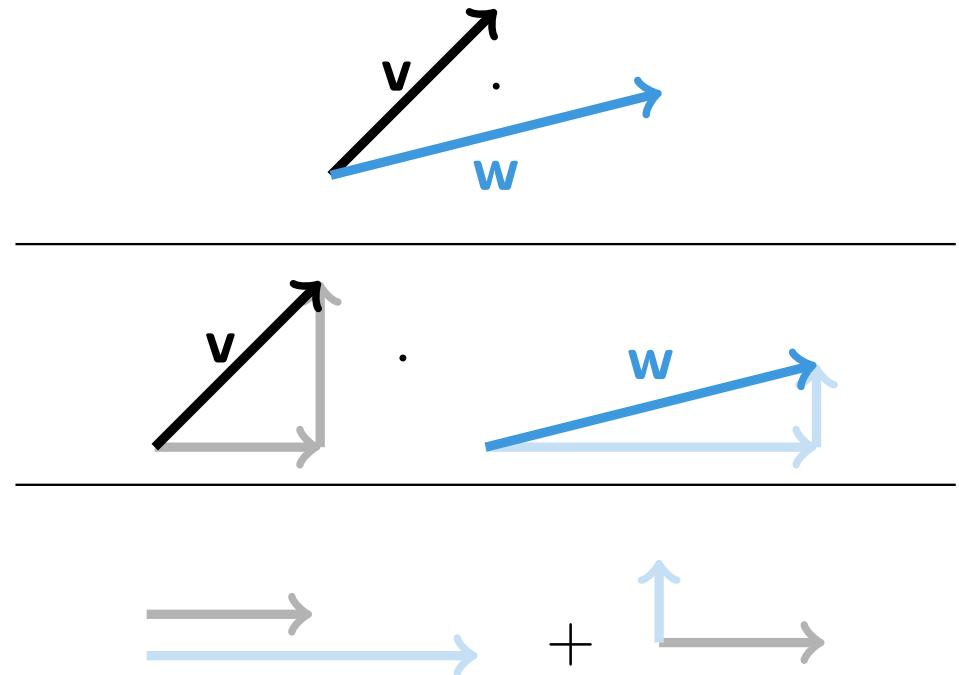
<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$

Analogously for 2D:  
 $\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$



# Dot Product

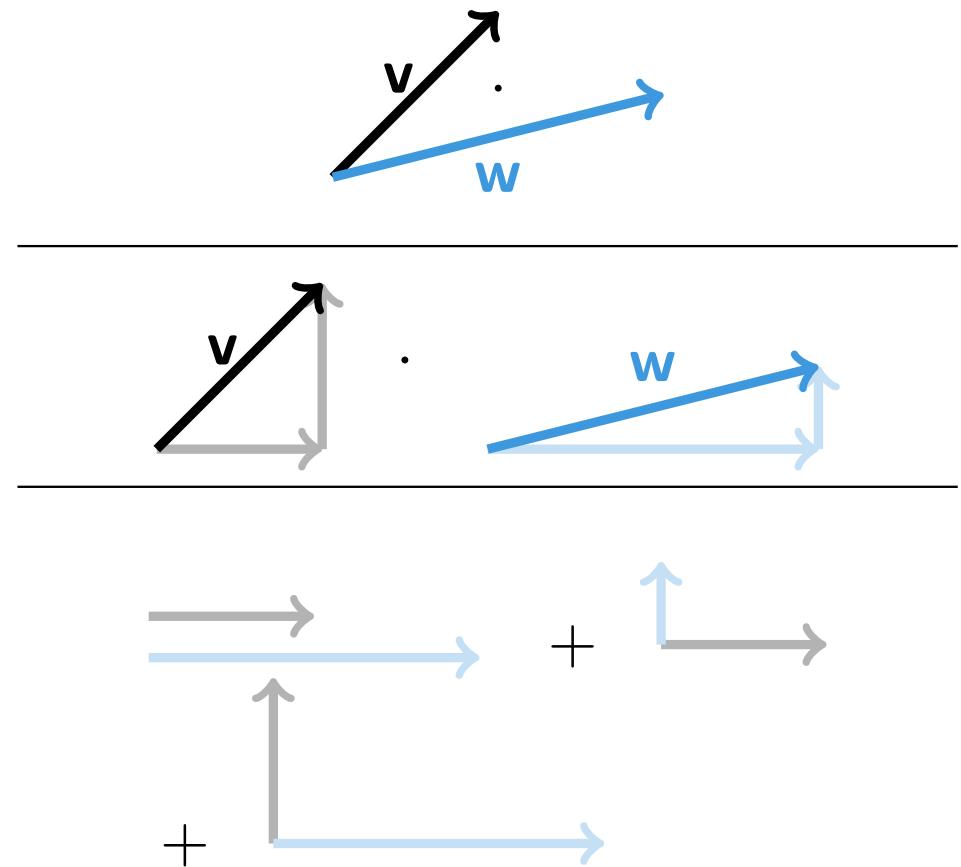
<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$

Analogously for 2D:  
 $\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$



# Dot Product

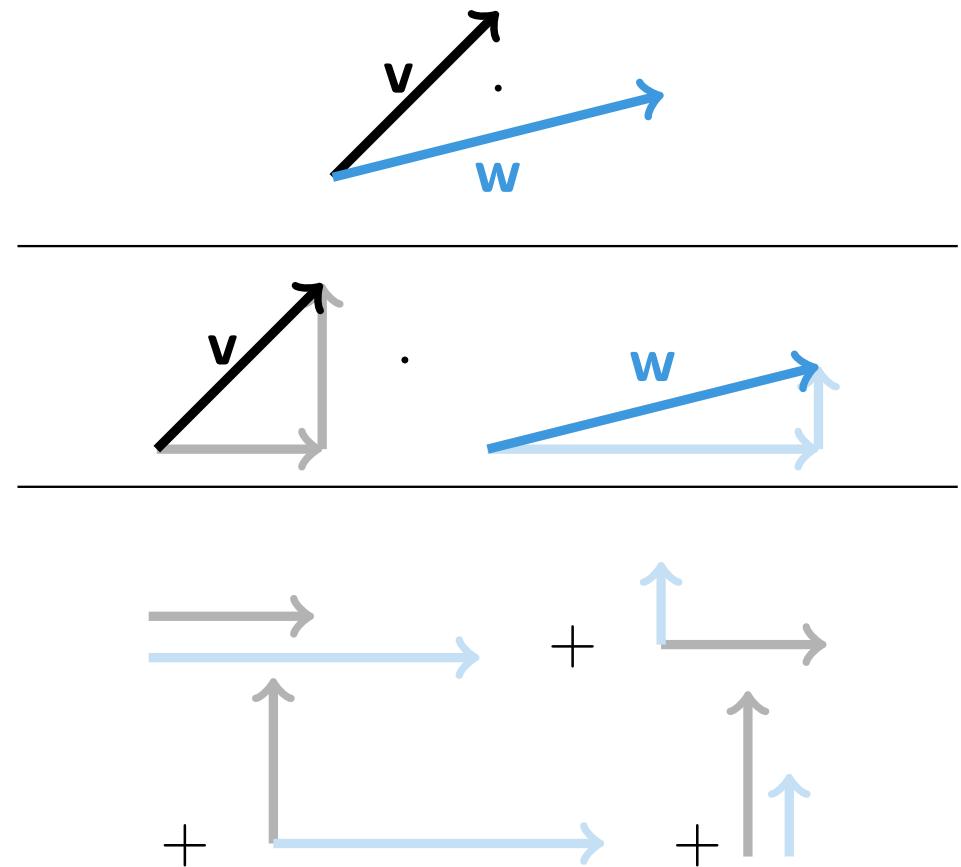
<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$

Analogously for 2D:  
 $\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$



# Dot Product

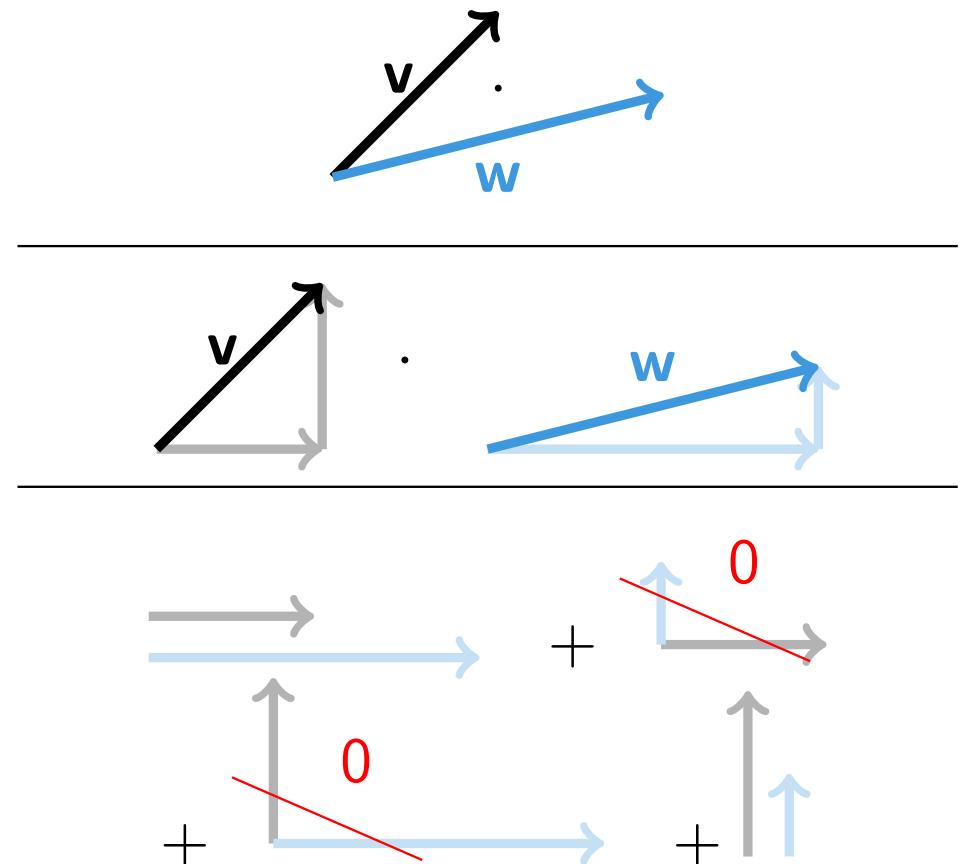
<https://vevox.app/#/m/106717265>

For vectors  $\mathbf{v} = (v_1, v_2, v_3)^t$  and  $\mathbf{w} = (w_1, w_2, w_3)^t$ , the *dot product* is

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3.$$

Analogously for 2D:  
 $\mathbf{v} = (v_1, v_2)^t, \mathbf{w} = (w_1, w_2)^t$ , then

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2.$$



# Angles between Vectors

# Angles between Vectors

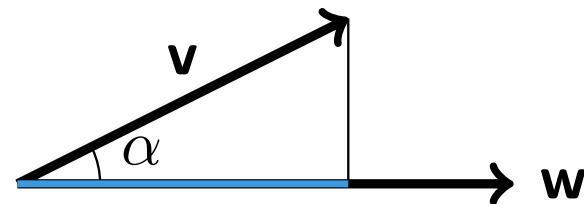
For two vectors  $\mathbf{v}$  and  $\mathbf{w}$ , the angle  $\alpha$  between them can be computed as

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

# Angles between Vectors

For two vectors  $\mathbf{v}$  and  $\mathbf{w}$ , the angle  $\alpha$  between them can be computed as

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$



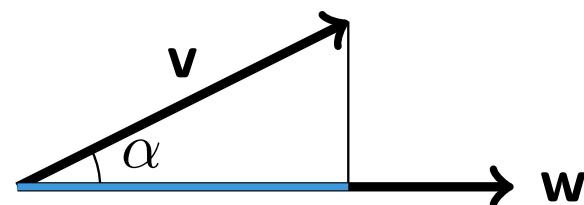
$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

# Angles between Vectors

For two vectors  $\mathbf{v}$  and  $\mathbf{w}$ , the angle  $\alpha$  between them can be computed as

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

$$\Leftrightarrow \alpha = \arccos\left(\frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\|_2 \|\mathbf{w}\|_2}\right).$$



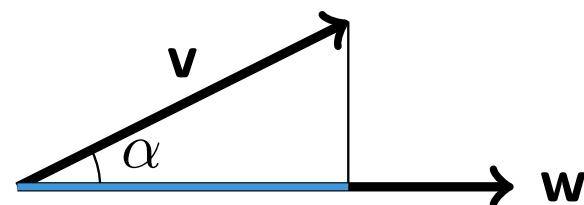
$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

# Angles between Vectors

For two vectors  $\mathbf{v}$  and  $\mathbf{w}$ , the angle  $\alpha$  between them can be computed as

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

$$\Leftrightarrow \alpha = \arccos\left(\frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\|_2 \|\mathbf{w}\|_2}\right).$$



Two vectors that enclose an angle of  $90^\circ$  are called *orthogonal*.

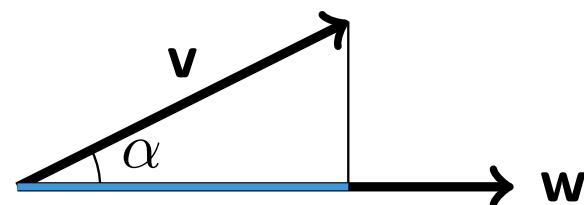
$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

# Angles between Vectors

For two vectors  $\mathbf{v}$  and  $\mathbf{w}$ , the angle  $\alpha$  between them can be computed as

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

$$\Leftrightarrow \alpha = \arccos\left(\frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\|_2 \|\mathbf{w}\|_2}\right).$$



Two vectors that enclose an angle of  $90^\circ$  are called *orthogonal*.

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

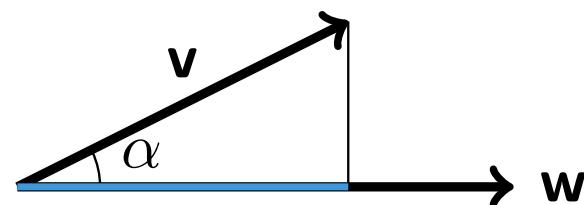
**Question:** Given the formula on the right, what is the dot product of two orthogonal vectors?

# Angles between Vectors

For two vectors  $\mathbf{v}$  and  $\mathbf{w}$ , the angle  $\alpha$  between them can be computed as

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

$$\Leftrightarrow \alpha = \arccos\left(\frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\|_2 \|\mathbf{w}\|_2}\right).$$



Two vectors that enclose an angle of  $90^\circ$  are called *orthogonal*.

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\|_2 \|\mathbf{w}\|_2 \cos(\alpha)$$

**Question:** Given the formula on the right, what is the dot product of two orthogonal vectors?

It's 0, providing a test for orthogonality.

# Find an orthogonal vector in 3D

# Find an orthogonal vector in 3D

Given two vectors  $\mathbf{v} = (1, 0, 1)^t$  and  $\mathbf{w} = (0, 1, 1)^t$ , find a third vector  $\mathbf{u} = (u_1, u_2, u_3)^t \in \mathbb{R}^3$  that is orthogonal to the first two,

# Find an orthogonal vector in 3D

Given two vectors  $\mathbf{v} = (1, 0, 1)^t$  and  $\mathbf{w} = (0, 1, 1)^t$ , find a third vector  $\mathbf{u} = (u_1, u_2, u_3)^t \in \mathbb{R}^3$  that is orthogonal to the first two, by solving the linear system:

$$I \quad \mathbf{v} \cdot \mathbf{u} = u_1 + u_3 = 0$$

$$II \quad \mathbf{w} \cdot \mathbf{u} = u_2 + u_3 = 0$$

# Find an orthogonal vector in 3D

Given two vectors  $\mathbf{v} = (1, 0, 1)^t$  and  $\mathbf{w} = (0, 1, 1)^t$ , find a third vector  $\mathbf{u} = (u_1, u_2, u_3)^t \in \mathbb{R}^3$  that is orthogonal to the first two, by solving the linear system:

$$I \quad \mathbf{v} \cdot \mathbf{u} = u_1 + u_3 = 0$$

$$II \quad \mathbf{w} \cdot \mathbf{u} = u_2 + u_3 = 0$$

Which leads to:

$$\Rightarrow I - II = u_1 - u_2 = 0 \Rightarrow u_1 = u_2$$

# Find an orthogonal vector in 3D

Given two vectors  $\mathbf{v} = (1, 0, 1)^t$  and  $\mathbf{w} = (0, 1, 1)^t$ , find a third vector  $\mathbf{u} = (u_1, u_2, u_3)^t \in \mathbb{R}^3$  that is orthogonal to the first two, by solving the linear system:

$$I \quad \mathbf{v} \cdot \mathbf{u} = u_1 + u_3 = 0$$

$$II \quad \mathbf{w} \cdot \mathbf{u} = u_2 + u_3 = 0$$

Which leads to:

$$\Rightarrow I - II = u_1 - u_2 = 0 \Rightarrow u_1 = u_2$$

Choosing  $u_2 = r$  independent yields  $u_1 = r$  and from  $I$   $u_3 = -r$ , that is the vector is of the form  $\mathbf{u} = r(1, 1, -1)^t$ .

# Find an orthogonal vector in 3D

Given two vectors  $\mathbf{v} = (1, 0, 1)^t$  and  $\mathbf{w} = (0, 1, 1)^t$ , find a third vector  $\mathbf{u} = (u_1, u_2, u_3)^t \in \mathbb{R}^3$  that is orthogonal to the first two, by solving the linear system:

$$I \quad \mathbf{v} \cdot \mathbf{u} = u_1 + u_3 = 0$$

$$II \quad \mathbf{w} \cdot \mathbf{u} = u_2 + u_3 = 0$$

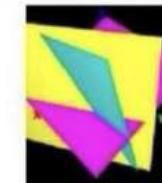
Which leads to:

$$\Rightarrow I - II = u_1 - u_2 = 0 \Rightarrow u_1 = u_2$$

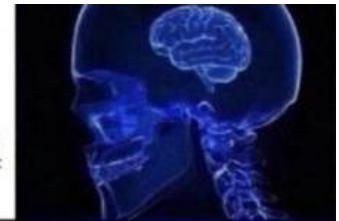
Choosing  $u_2 = r$  independent yields  $u_1 = r$  and from  $I$   $u_3 = -r$ , that is the vector is of the form  $\mathbf{u} = r(1, 1, -1)^t$ .

$$\begin{aligned} 2x + y &= 8 \\ 3x - z &= 10 \\ 2x + y + 4z &= 4 \end{aligned}$$

$$\begin{bmatrix} 2 & 1 & 0 \\ 3 & 0 & -1 \\ 2 & 1 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ 10 \\ 4 \end{bmatrix}$$



$$\begin{aligned} \text{apple} + \text{apple} + \text{watermelon} &= 8 \\ \text{apple} - \text{grapes} &= 10 \\ \text{apple} + \text{grapes} + \text{watermelon} &= 4 \end{aligned}$$



# Cross product

**Show of Hands:** Who has seen  
the cross product before?

# Cross product

**Show of Hands:** Who has seen  
the cross product before?

Given two vectors  $\mathbf{a} = (a_1, a_2, a_3)^t$   
and  $\mathbf{b} = (b_1, b_2, b_3)^t$ , their cross  
product is

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}.$$

# Cross product

**Show of Hands:** Who has seen the cross product before?

Given two vectors  $\mathbf{a} = (a_1, a_2, a_3)^t$  and  $\mathbf{b} = (b_1, b_2, b_3)^t$ , their cross product is

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}.$$

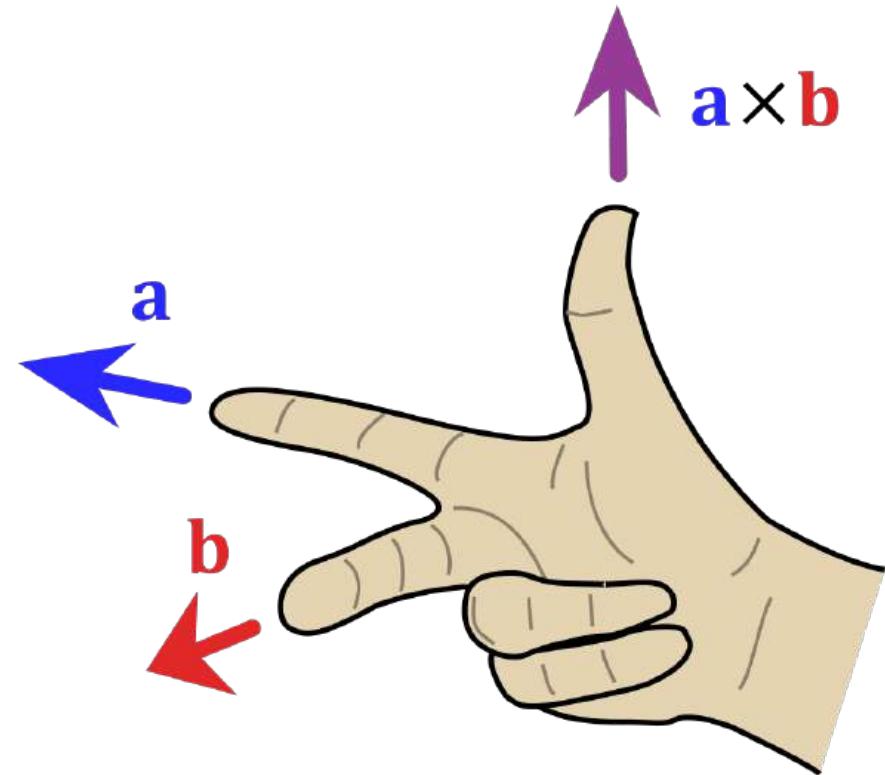
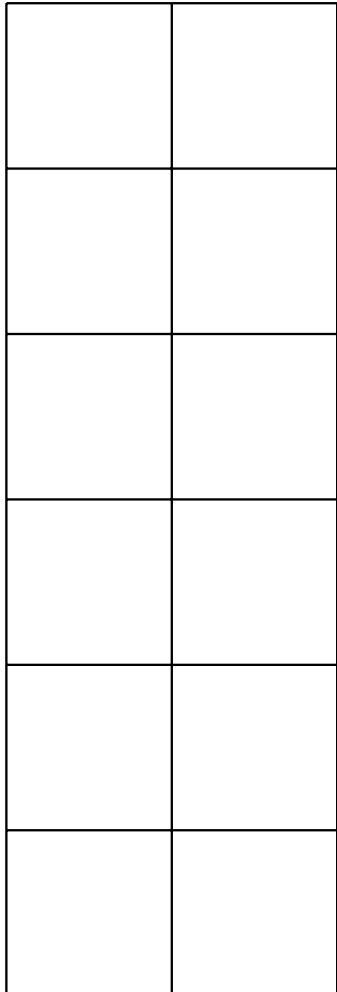


Figure: Right hand rule, Acdx on Wikipedia, (CC BY-SA 3.0).

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :



# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	
2	
3	
-1	
2	
3	

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
3	2
-1	1
2	-1
3	2

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
3	2
-1	1
2	-1
3	2

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
3	2
-1	1
2	-1
3	2

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
<b>3</b>	2
-1	1
2	-1
<b>3</b>	2

$$(2 \cdot 2) - (3 \cdot -1) = 4 + 3 = 7$$

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
<b>  X  </b>	
3	2
<b>  X  </b>	
-1	1
2	-1
<b>  X  </b>	
3	2

$$(2 \cdot 2) - (3 \cdot -1) = 4 + 3 = 7$$

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
<b>×</b>	
3	2
<b>×</b>	
-1	1
2	-1
<b> </b>	
3	2

$$(2 \cdot 2) - (3 \cdot -1) = 4 + 3 = 7$$

$$(3 \cdot 1) - (-1 \cdot 2) = 3 + 2 = 5$$

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
3	2
-1	1
2	-1
3	2

$$(2 \cdot 2) - (3 \cdot -1) = 4 + 3 = 7$$

$$(3 \cdot 1) - (-1 \cdot 2) = 3 + 2 = 5$$

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
3	2
-1	1
2	-1
3	2

$$(2 \cdot 2) - (3 \cdot -1) = 4 + 3 = 7$$

$$(3 \cdot 1) - (-1 \cdot 2) = 3 + 2 = 5$$

$$(-1 \cdot -1) - (2 \cdot 1) = 1 - 2 = -1$$

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
3	2
-1	1
2	-1
3	2

$$(2 \cdot 2) - (3 \cdot -1) = 4 + 3 = 7$$

$$(3 \cdot 1) - (-1 \cdot 2) = 3 + 2 = 5$$

$$(-1 \cdot -1) - (2 \cdot 1) = 1 - 2 = -1$$

$$\begin{pmatrix} 7 \\ 5 \\ -1 \end{pmatrix} = \mathbf{v} \times \mathbf{w}$$

# Cross product

How to remember the cross product rule  $\mathbf{v} = (-1, 2, 3)^t$ ,  $\mathbf{w} = (1, -1, 2)^t$ :

-1	1
2	-1
<b>  X  </b>	
3	2
<b>  X  </b>	
-1	1
<b>  X  </b>	
2	-1
3	2

$$(2 \cdot 2) - (3 \cdot -1) = 4 + 3 = 7$$

$$(3 \cdot 1) - (-1 \cdot 2) = 3 + 2 = 5$$

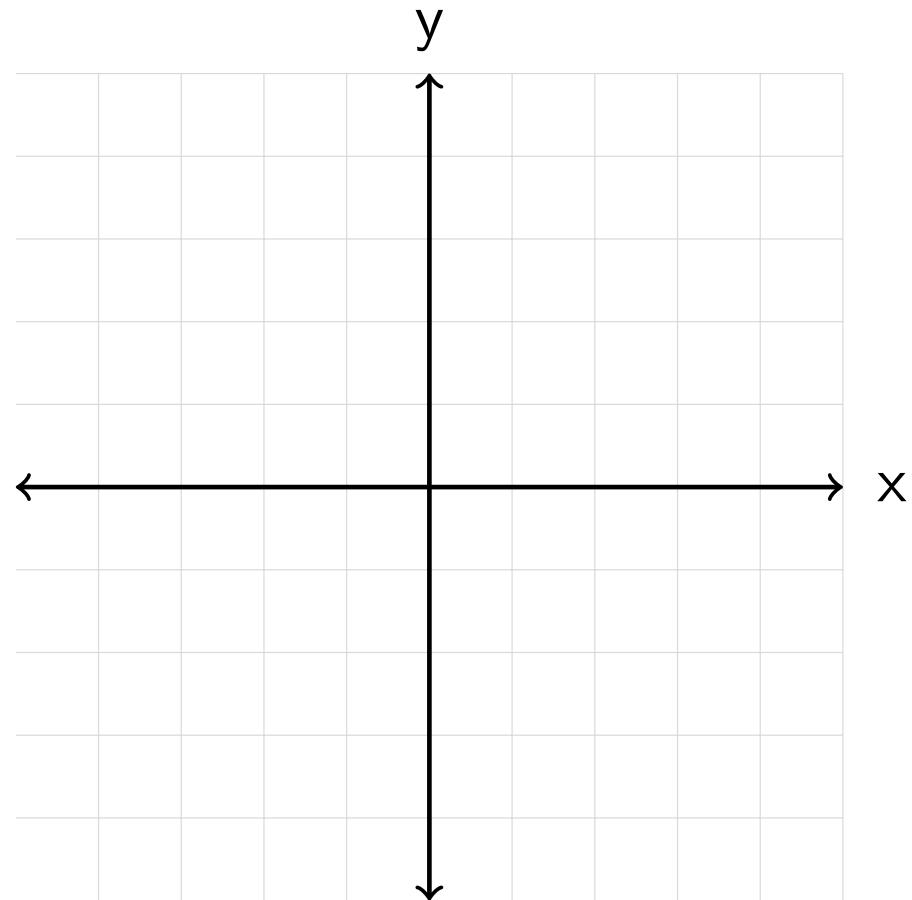
$$(-1 \cdot -1) - (2 \cdot 1) = 1 - 2 = -1$$

$$\begin{pmatrix} 7 \\ 5 \\ -1 \end{pmatrix} = \mathbf{v} \times \mathbf{w}$$

<https://vevox.app/#/m/106717265>

# Linear Independence

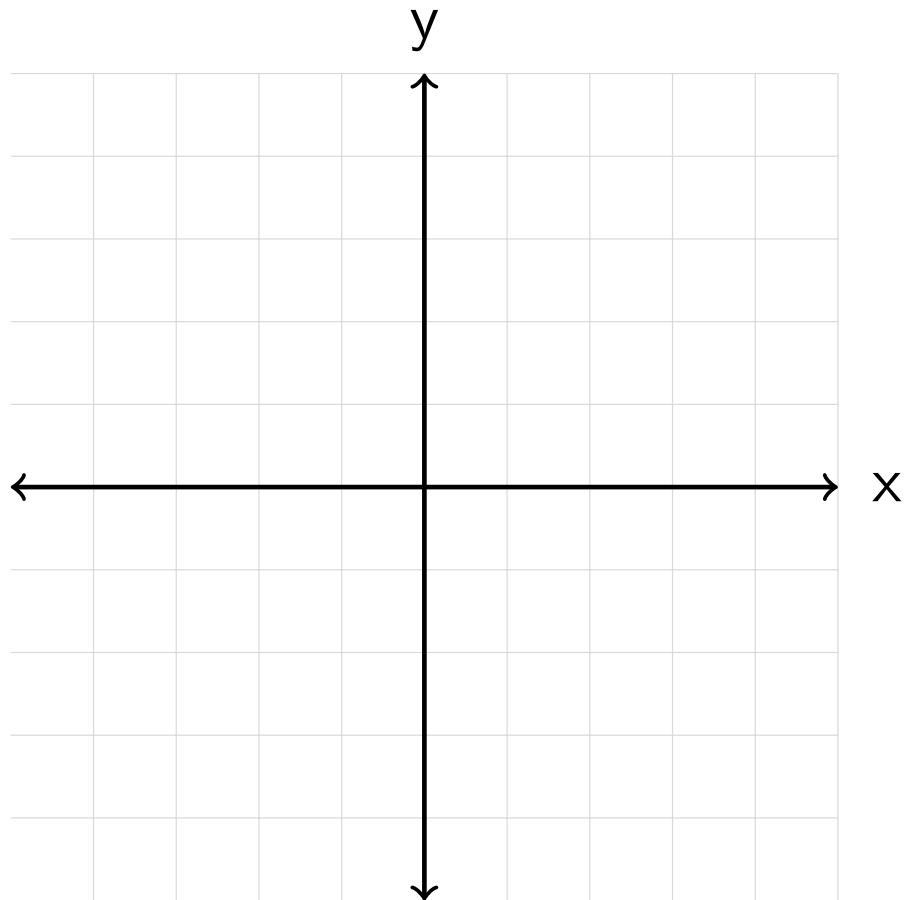
<https://vevox.app/#/m/106717265>



# Linear Independence

<https://vevox.app/#/m/106717265>

A set of vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$



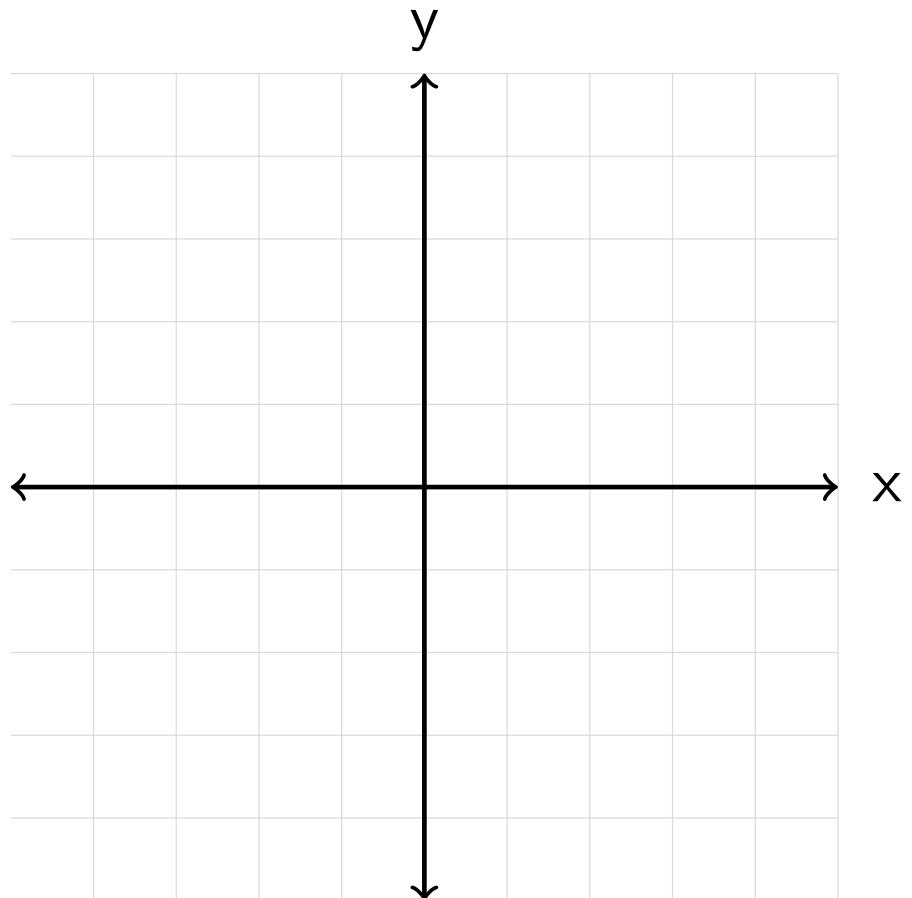
# Linear Independence

<https://vevox.app/#/m/106717265>

A set of vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is *linearly independent* if the only solution to

$$r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k = 0$$

is  $r_1 = \dots = r_k = 0$ .



# Linear Independence

<https://vevox.app/#/m/106717265>

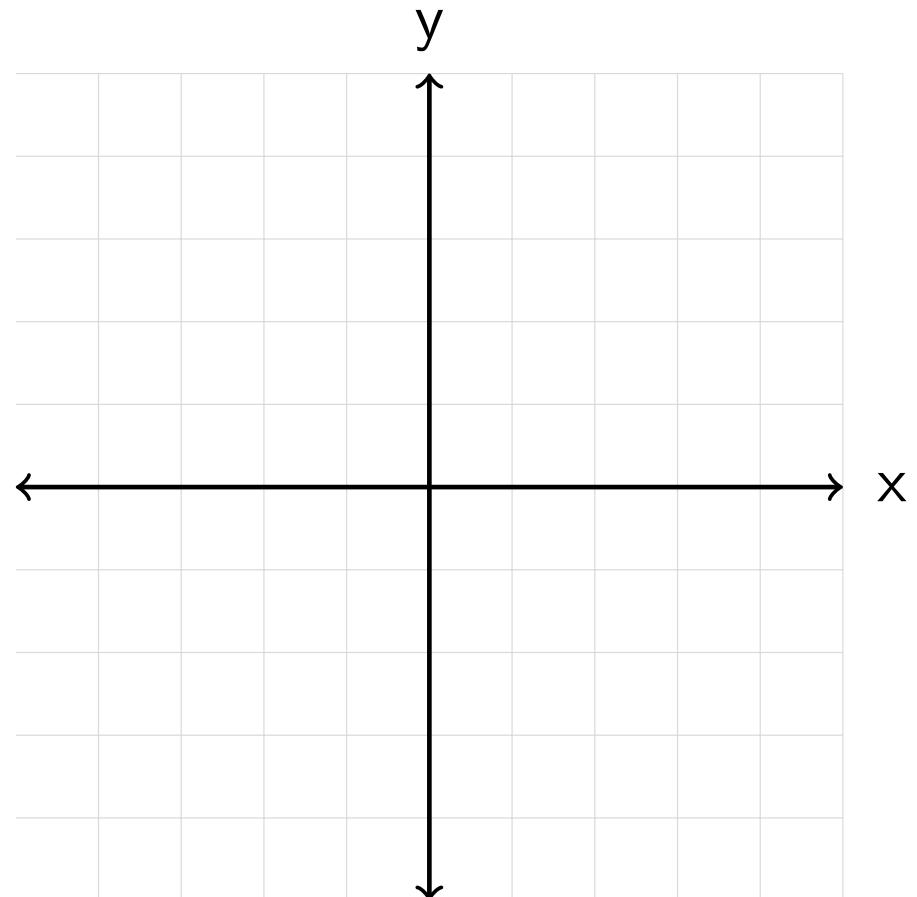
A set of vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is *linearly independent* if the only solution to

$$r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k = 0$$

is  $r_1 = \dots = r_k = 0$ .

Otherwise, they are *linearly dependent*, w.l.g.  $r_1 \neq 0$  and

$$\mathbf{v}_1 = -\frac{r_2}{r_1} \mathbf{v}_2 - \dots - \frac{r_k}{r_1} \mathbf{v}_k.$$



# Linear Independence

<https://vevox.app/#/m/106717265>

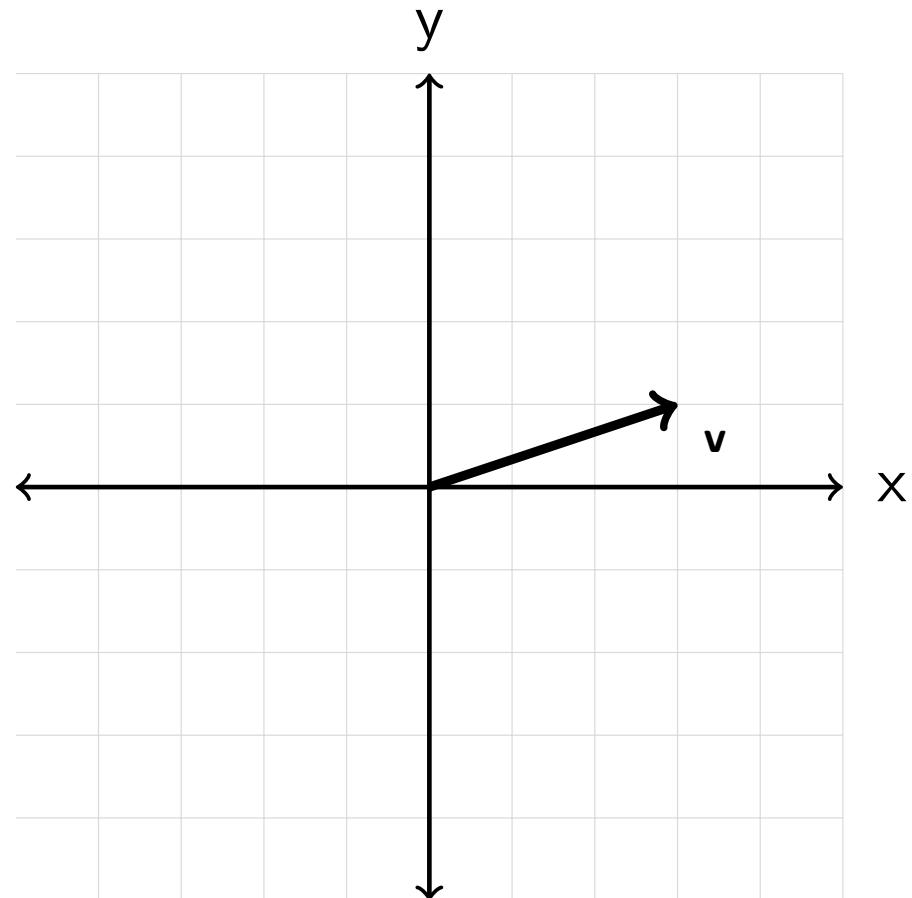
A set of vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is *linearly independent* if the only solution to

$$r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k = 0$$

is  $r_1 = \dots = r_k = 0$ .

Otherwise, they are *linearly dependent*, w.l.g.  $r_1 \neq 0$  and

$$\mathbf{v}_1 = -\frac{r_2}{r_1} \mathbf{v}_2 - \dots - \frac{r_k}{r_1} \mathbf{v}_k.$$



# Linear Independence

<https://vevox.app/#/m/106717265>

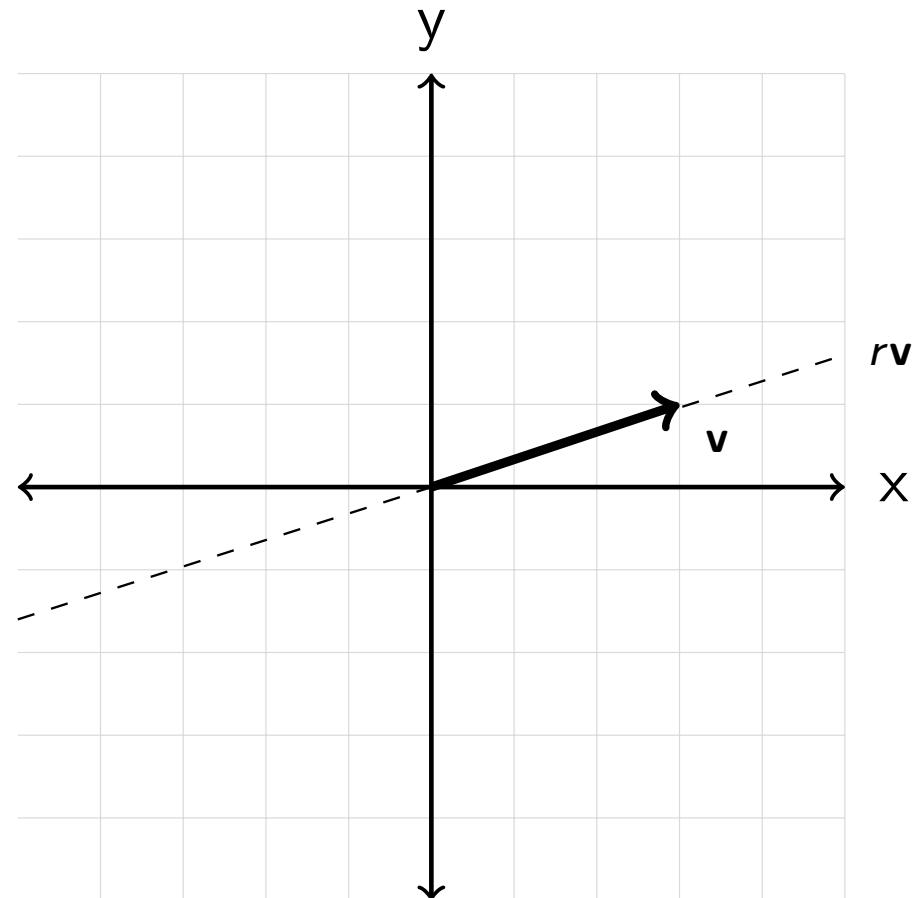
A set of vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is *linearly independent* if the only solution to

$$r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k = 0$$

is  $r_1 = \dots = r_k = 0$ .

Otherwise, they are *linearly dependent*, w.l.g.  $r_1 \neq 0$  and

$$\mathbf{v}_1 = -\frac{r_2}{r_1} \mathbf{v}_2 - \dots - \frac{r_k}{r_1} \mathbf{v}_k.$$



# Linear Independence

<https://vevox.app/#/m/106717265>

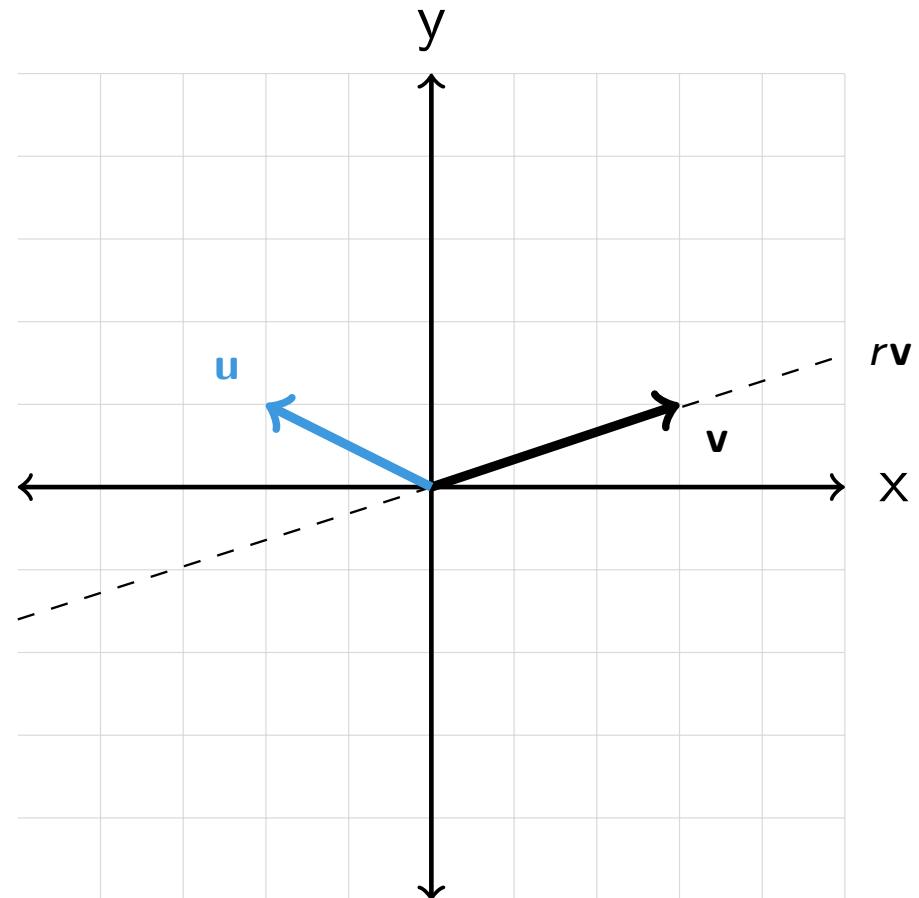
A set of vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is *linearly independent* if the only solution to

$$r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k = 0$$

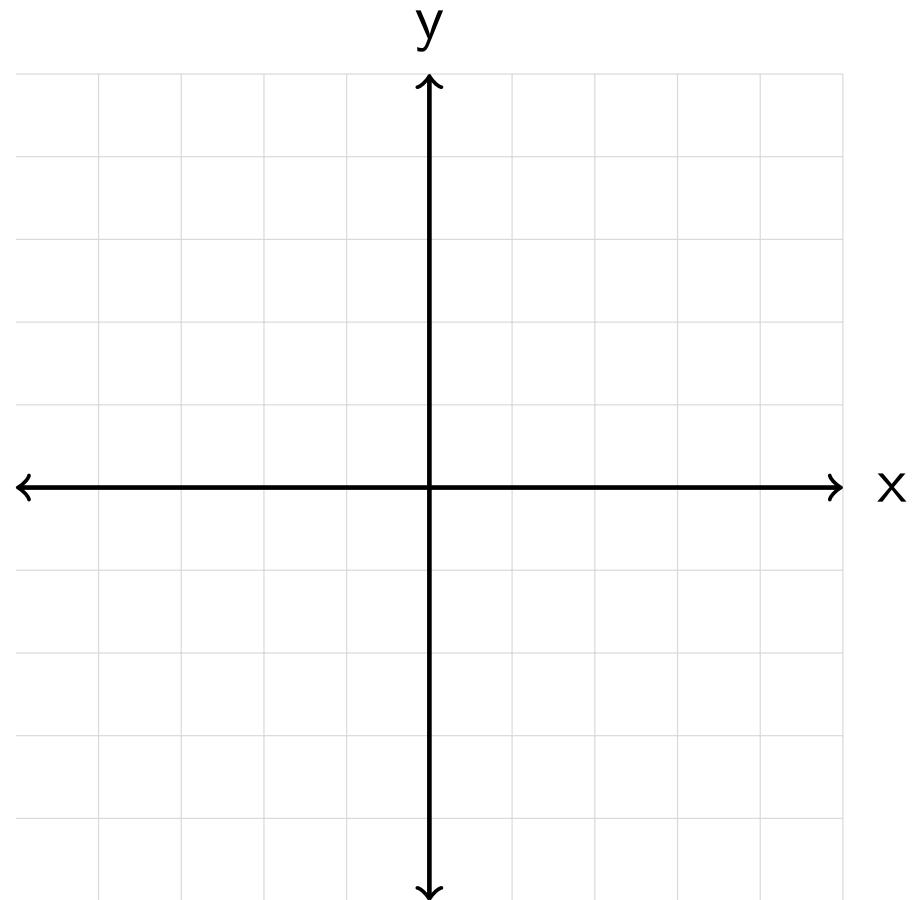
is  $r_1 = \dots = r_k = 0$ .

Otherwise, they are *linearly dependent*, w.l.g.  $r_1 \neq 0$  and

$$\mathbf{v}_1 = -\frac{r_2}{r_1} \mathbf{v}_2 - \dots - \frac{r_k}{r_1} \mathbf{v}_k.$$

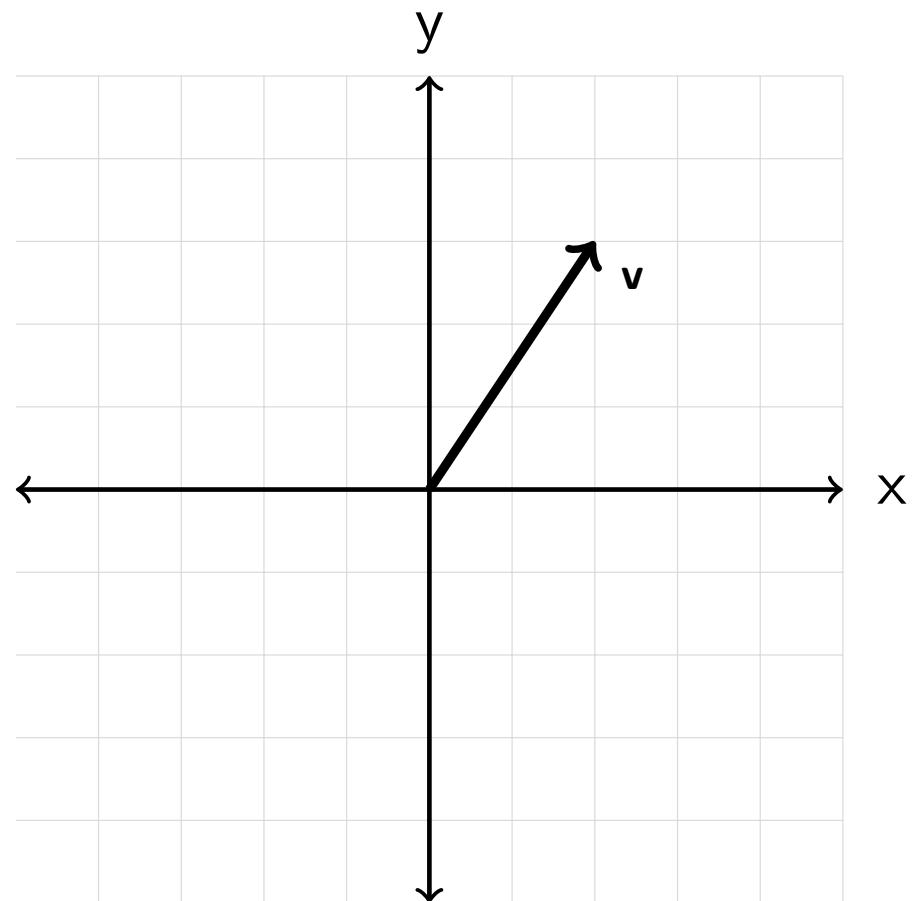


# Finding orthogonal vectors



# Finding orthogonal vectors

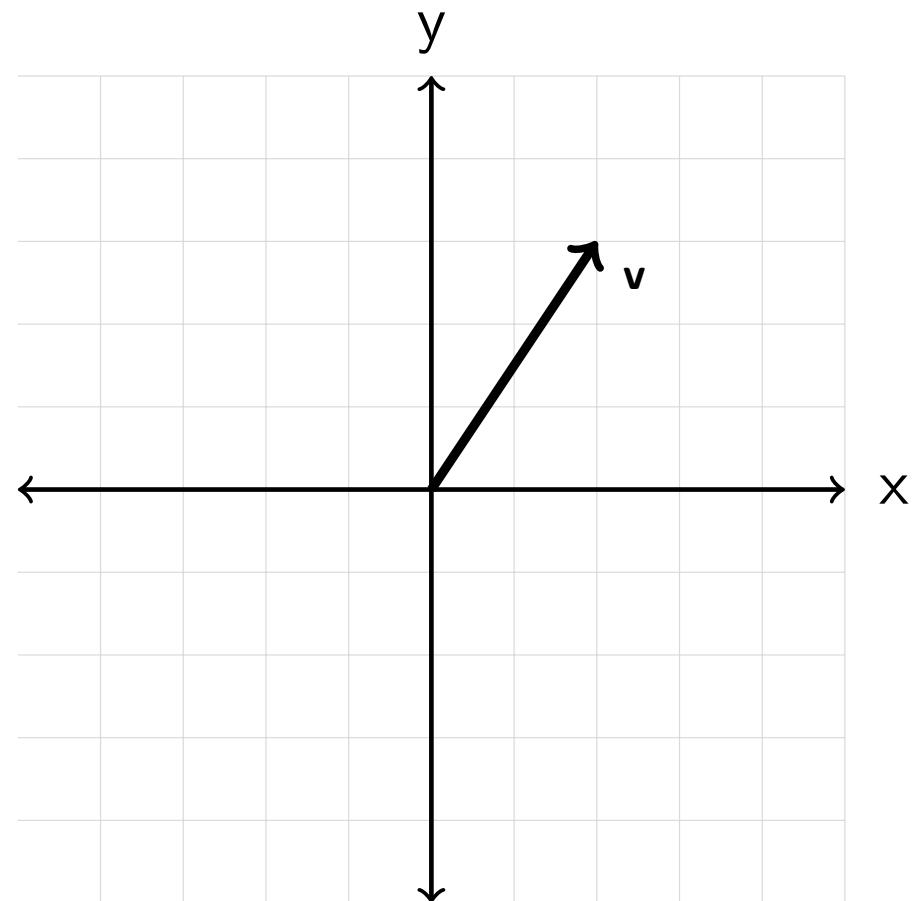
Given a vector  $\mathbf{v} = (2, 3)^t$ , find a vector that is orthogonal to it,



# Finding orthogonal vectors

Given a vector  $\mathbf{v} = (2, 3)^t$ , find a vector that is orthogonal to it, i.e., a vector  $\mathbf{w} = (w_1, w_2)^t$ , such that

$$\mathbf{v} \cdot \mathbf{w} = 2w_1 + 3w_2 = 0$$

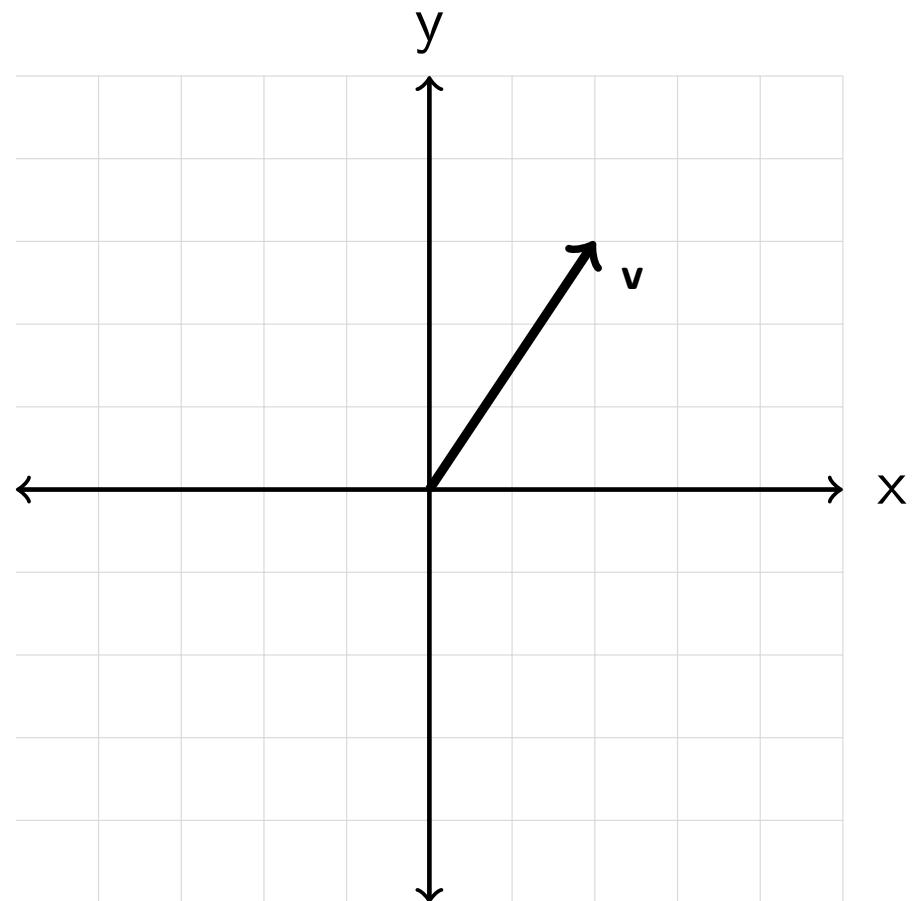


# Finding orthogonal vectors

Given a vector  $\mathbf{v} = (2, 3)^t$ , find a vector that is orthogonal to it, i.e., a vector  $\mathbf{w} = (w_1, w_2)^t$ , such that

$$\mathbf{v} \cdot \mathbf{w} = 2w_1 + 3w_2 = 0$$

$$\Leftrightarrow w_1 = -\frac{3}{2}w_2.$$



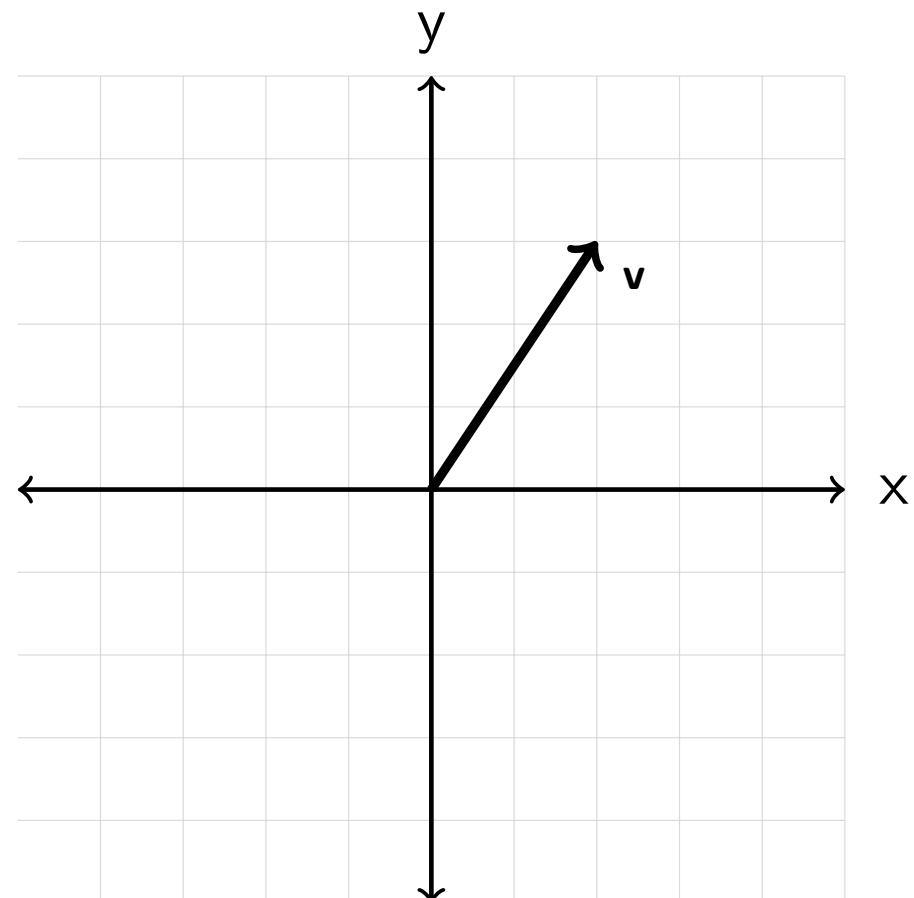
# Finding orthogonal vectors

Given a vector  $\mathbf{v} = (2, 3)^t$ , find a vector that is orthogonal to it, i.e., a vector  $\mathbf{w} = (w_1, w_2)^t$ , such that

$$\mathbf{v} \cdot \mathbf{w} = 2w_1 + 3w_2 = 0$$

$$\Leftrightarrow w_1 = -\frac{3}{2}w_2.$$

That is, all vectors of the form  $\mathbf{w} = r\left(-\frac{3}{2}, 1\right)$  are orthogonal to  $\mathbf{v}$ .



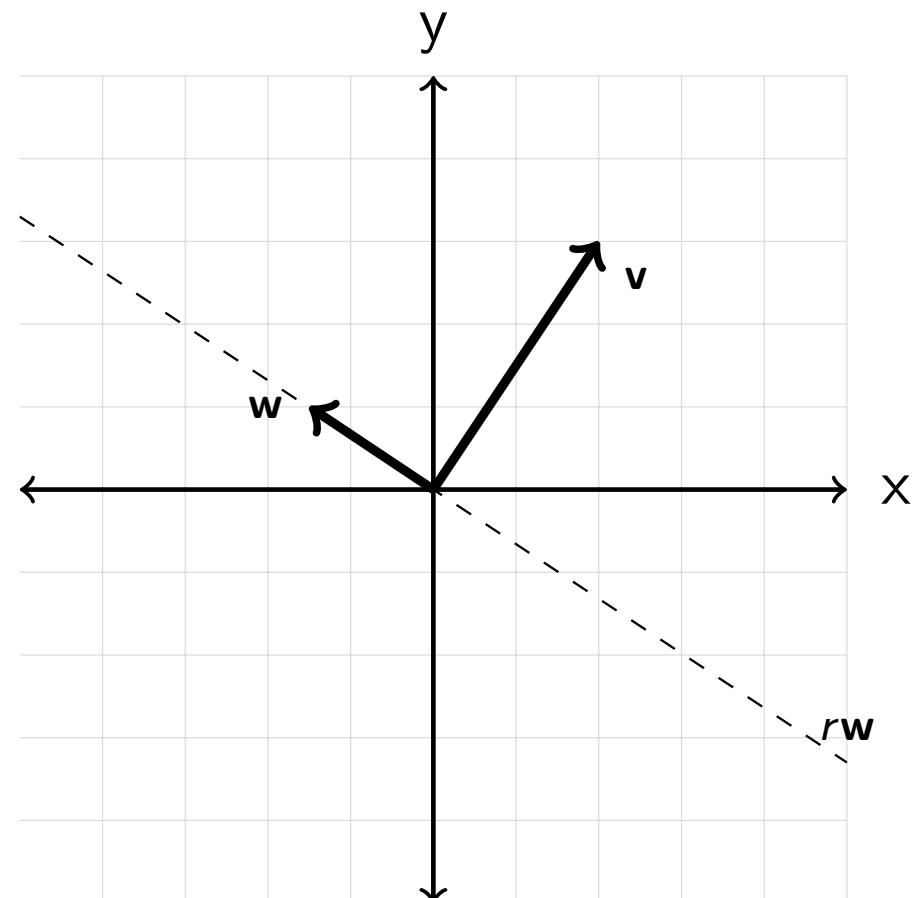
# Finding orthogonal vectors

Given a vector  $\mathbf{v} = (2, 3)^t$ , find a vector that is orthogonal to it, i.e., a vector  $\mathbf{w} = (w_1, w_2)^t$ , such that

$$\mathbf{v} \cdot \mathbf{w} = 2w_1 + 3w_2 = 0$$

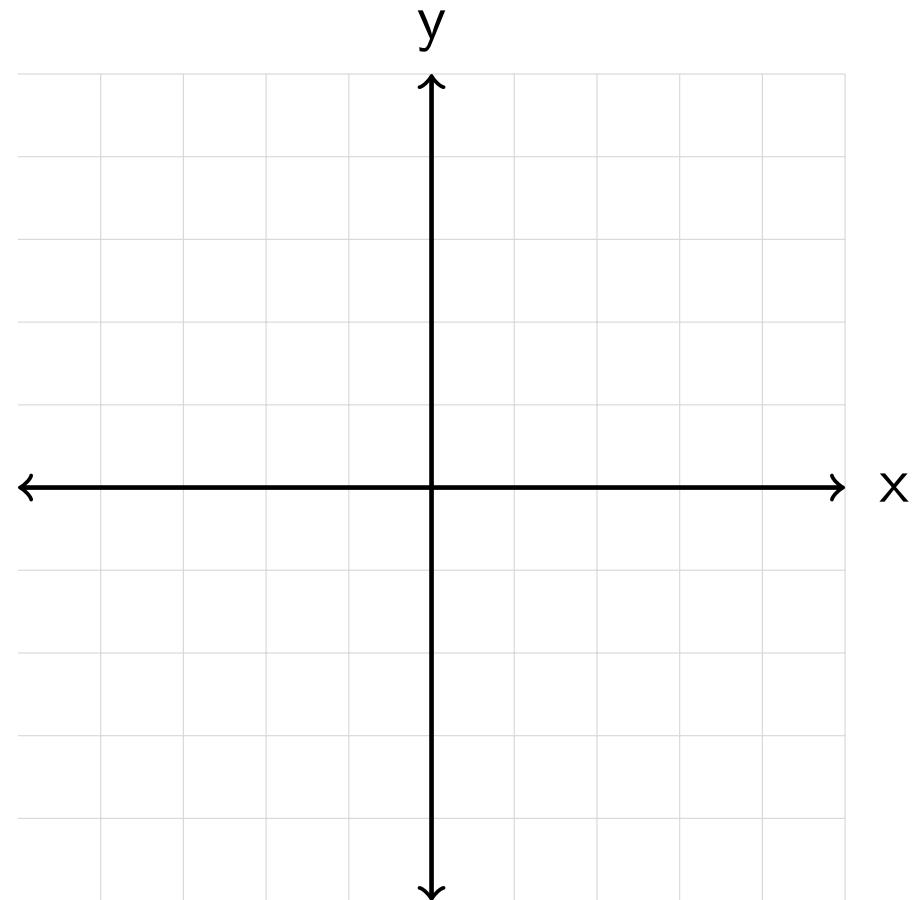
$$\Leftrightarrow w_1 = -\frac{3}{2}w_2.$$

That is, all vectors of the form  $\mathbf{w} = r\left(-\frac{3}{2}, 1\right)$  are orthogonal to  $\mathbf{v}$ .



# Basis

<https://vevox.app/#/m/106717265>

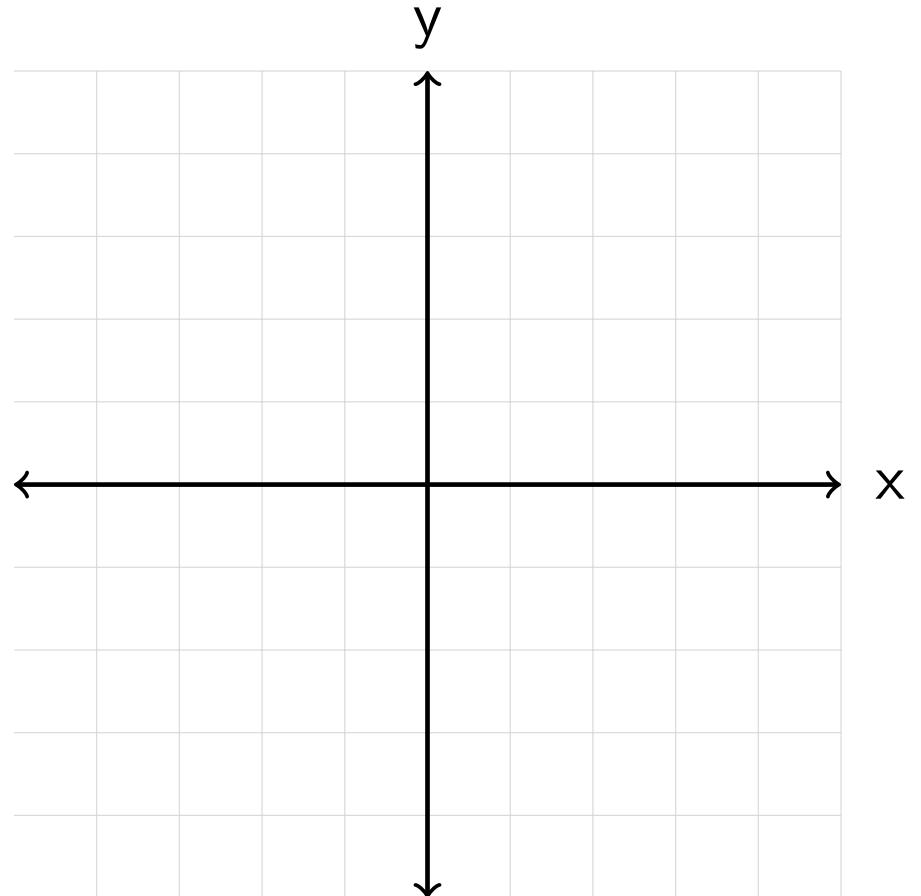


# Basis

<https://vevox.app/#/m/106717265>

A set of linearly independent vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is called a *basis*, if for every vector  $\mathbf{v} \in V$ , there are scalars  $r_1, \dots, r_k$  such that

$$\mathbf{v} = r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k.$$

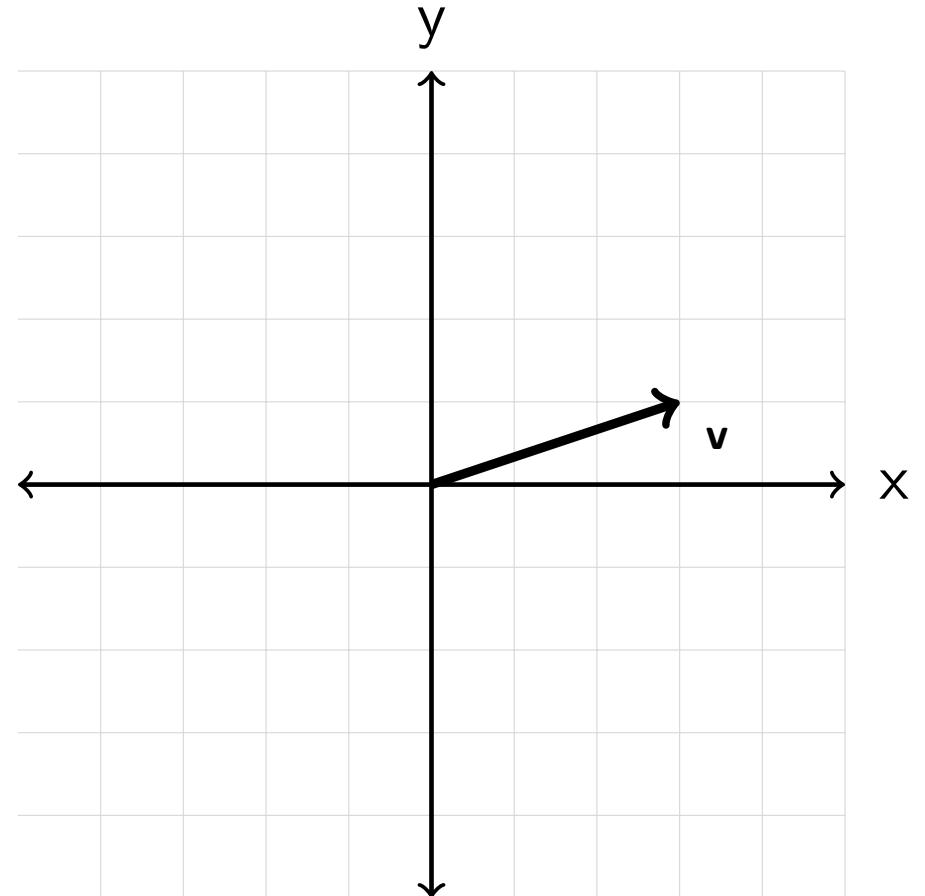


# Basis

<https://vevox.app/#/m/106717265>

A set of linearly independent vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is called a *basis*, if for every vector  $\mathbf{v} \in V$ , there are scalars  $r_1, \dots, r_k$  such that

$$\mathbf{v} = r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k.$$

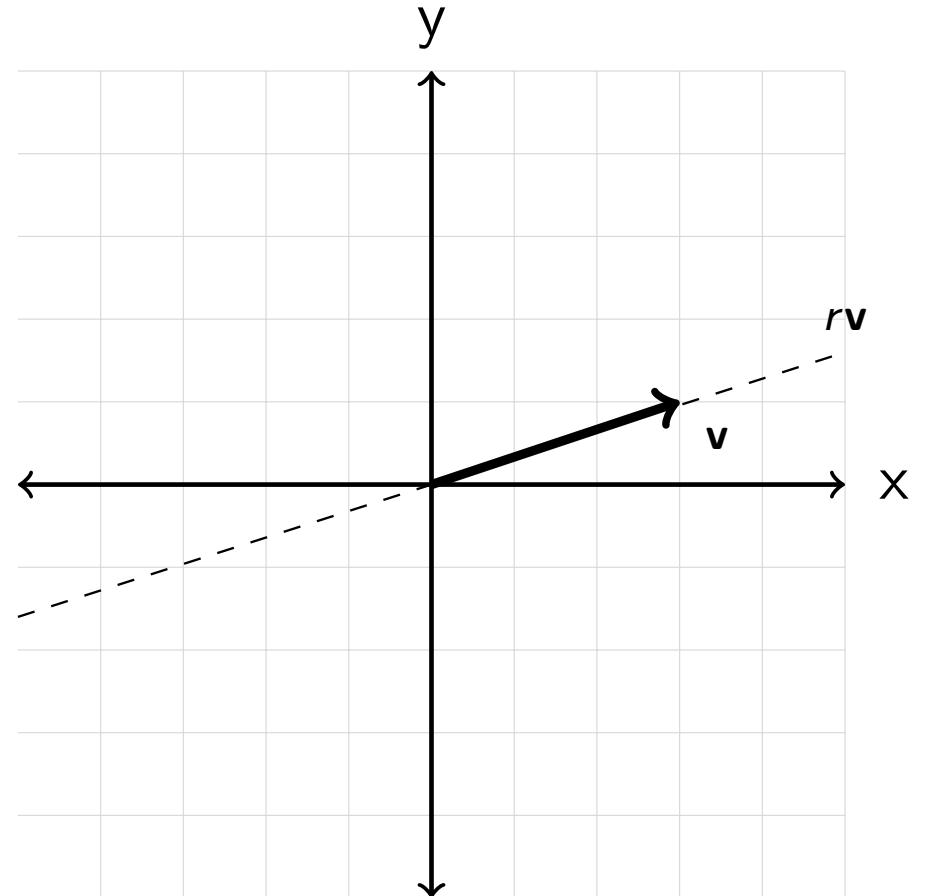


# Basis

<https://vevox.app/#/m/106717265>

A set of linearly independent vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is called a *basis*, if for every vector  $\mathbf{v} \in V$ , there are scalars  $r_1, \dots, r_k$  such that

$$\mathbf{v} = r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k.$$

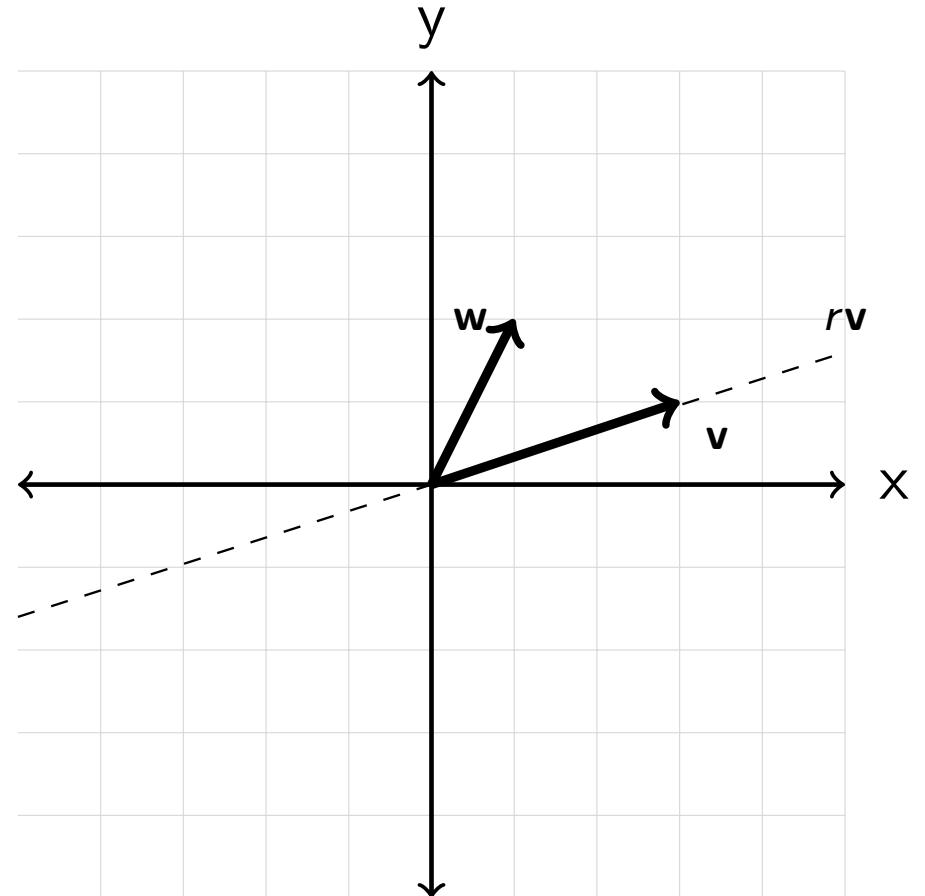


# Basis

<https://vevox.app/#/m/106717265>

A set of linearly independent vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is called a *basis*, if for every vector  $\mathbf{v} \in V$ , there are scalars  $r_1, \dots, r_k$  such that

$$\mathbf{v} = r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k.$$

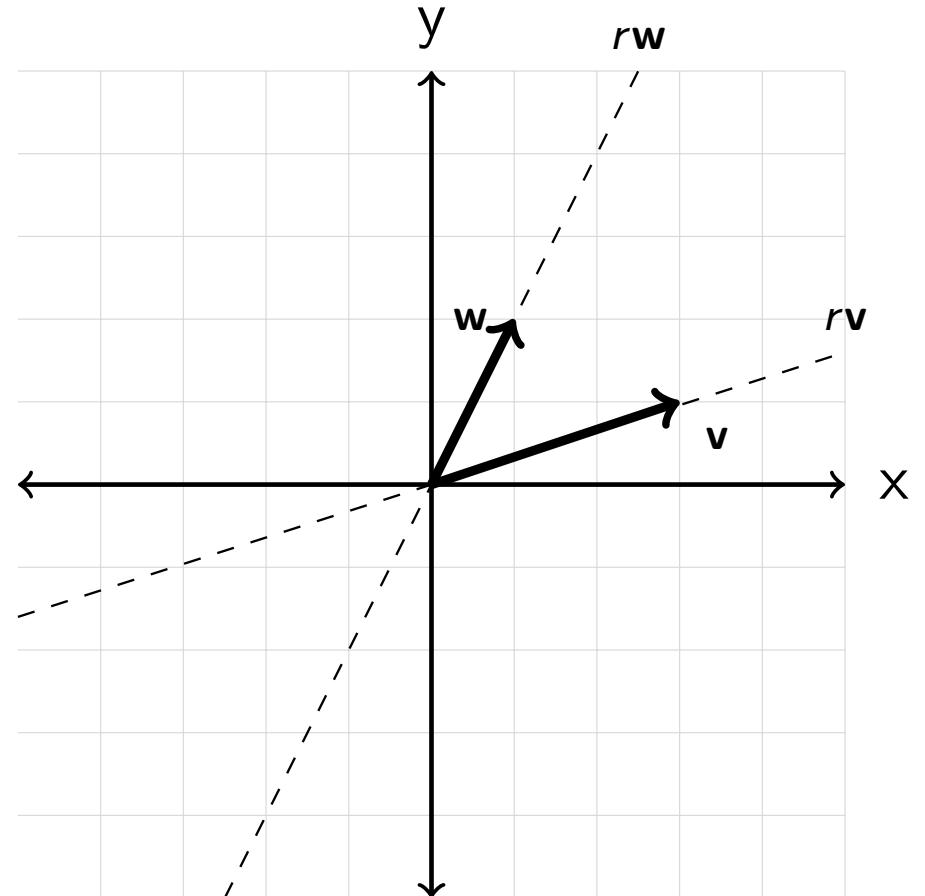


# Basis

<https://vevox.app/#/m/106717265>

A set of linearly independent vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is called a *basis*, if for every vector  $\mathbf{v} \in V$ , there are scalars  $r_1, \dots, r_k$  such that

$$\mathbf{v} = r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k.$$

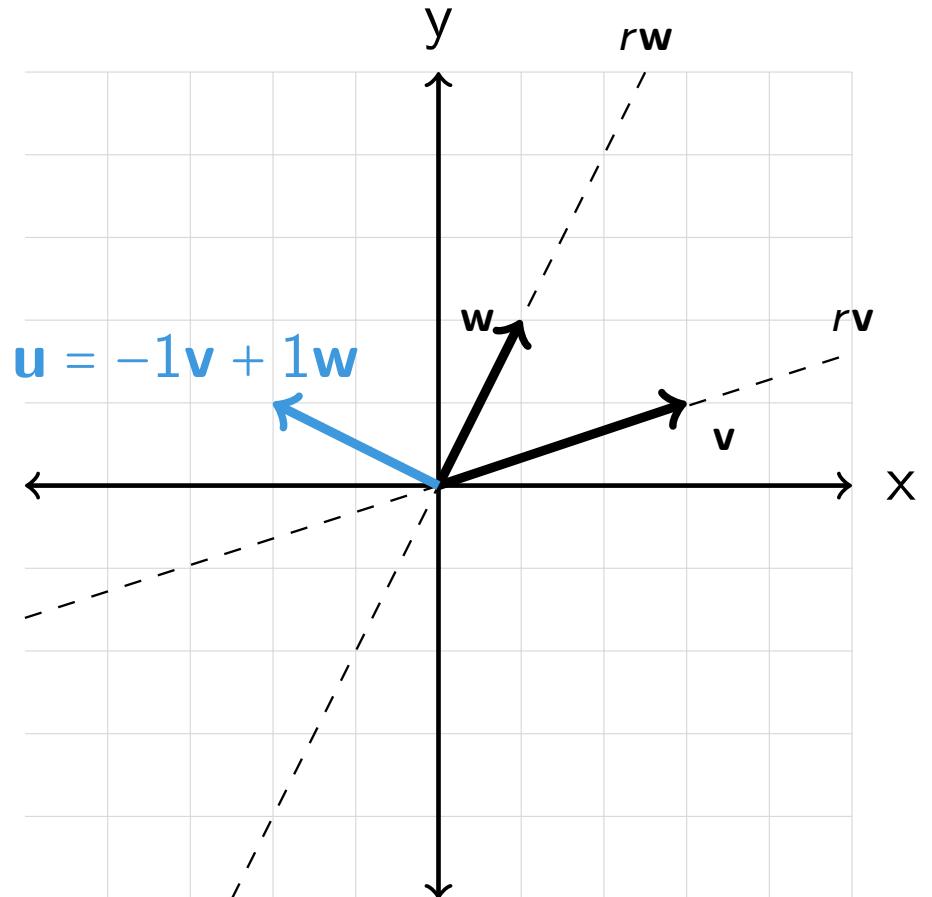


# Basis

<https://vevox.app/#/m/106717265>

A set of linearly independent vectors  $\mathbf{v}_1, \dots, \mathbf{v}_k \in V$  is called a *basis*, if for every vector  $\mathbf{v} \in V$ , there are scalars  $r_1, \dots, r_k$  such that

$$\mathbf{v} = r_1 \mathbf{v}_1 + \dots + r_k \mathbf{v}_k.$$



# Basis Completion

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

First, pick a vector that is *not* linearly dependent, e.g.,  $\mathbf{v} = (1, 0, 0)^t$ .

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

First, pick a vector that is *not* linearly dependent, e.g.,  $\mathbf{v} = (1, 0, 0)^t$ .

How do you know it's not linearly dependent?

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

First, pick a vector that is *not* linearly dependent, e.g.,  $\mathbf{v} = (1, 0, 0)^t$ .

How do you know it's not linearly dependent?

The system  $\mathbf{v} = r\mathbf{u}$  has no solutions.

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

First, pick a vector that is *not* linearly dependent, e.g.,  $\mathbf{v} = (1, 0, 0)^t$ .

How do you know it's not linearly dependent?

The system  $\mathbf{v} = r\mathbf{u}$  has no solutions.

For a basis, need a third vector  $\mathbf{w} = (w_1, w_2, w_3)^t$ ,

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

First, pick a vector that is *not* linearly dependent, e.g.,  $\mathbf{v} = (1, 0, 0)^t$ .

How do you know it's not linearly dependent?

The system  $\mathbf{v} = r\mathbf{u}$  has no solutions.

For a basis, need a third vector  $\mathbf{w} = (w_1, w_2, w_3)^t$ , such that

$$\mathbf{w} = r\mathbf{u} + s\mathbf{v}$$

has no solutions,

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

First, pick a vector that is *not* linearly dependent, e.g.,  $\mathbf{v} = (1, 0, 0)^t$ .

How do you know it's not linearly dependent?

The system  $\mathbf{v} = r\mathbf{u}$  has no solutions.

For a basis, need a third vector  $\mathbf{w} = (w_1, w_2, w_3)^t$ , such that

$$\mathbf{w} = r\mathbf{u} + s\mathbf{v}$$

has no solutions, which happens, e.g., if  $\mathbf{w} \cdot \mathbf{u} = 0$  and  $\mathbf{w} \cdot \mathbf{v} = 0$ .

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

First, pick a vector that is *not* linearly dependent, e.g.,  $\mathbf{v} = (1, 0, 0)^t$ .

How do you know it's not linearly dependent?

The system  $\mathbf{v} = r\mathbf{u}$  has no solutions.

For a basis, need a third vector  $\mathbf{w} = (w_1, w_2, w_3)^t$ , such that

$$\mathbf{w} = r\mathbf{u} + s\mathbf{v}$$

has no solutions, which happens, e.g., if  $\mathbf{w} \cdot \mathbf{u} = 0$  and  $\mathbf{w} \cdot \mathbf{v} = 0$ . Thus

$$\begin{aligned} 0 &= \mathbf{w} \cdot \mathbf{v} = w_1 \\ \Rightarrow 0 &= \mathbf{w} \cdot \mathbf{u} = 2w_2 + 3w_3 \Rightarrow w_2 = -\frac{3}{2}w_3. \end{aligned}$$

Hence, e.g.,  $\mathbf{w} = (0, -\frac{3}{2}, 1)$ .

# Basis Completion

Given a vector  $\mathbf{u} = (1, 2, 3)^t$ , how to complete this to a basis?

First, pick a vector that is *not* linearly dependent, e.g.,  $\mathbf{v} = (1, 0, 0)^t$ .

How do you know it's not linearly dependent?

The system  $\mathbf{v} = r\mathbf{u}$  has no solutions.

For a basis, need a third vector  $\mathbf{w} = (w_1, w_2, w_3)^t$ , such that

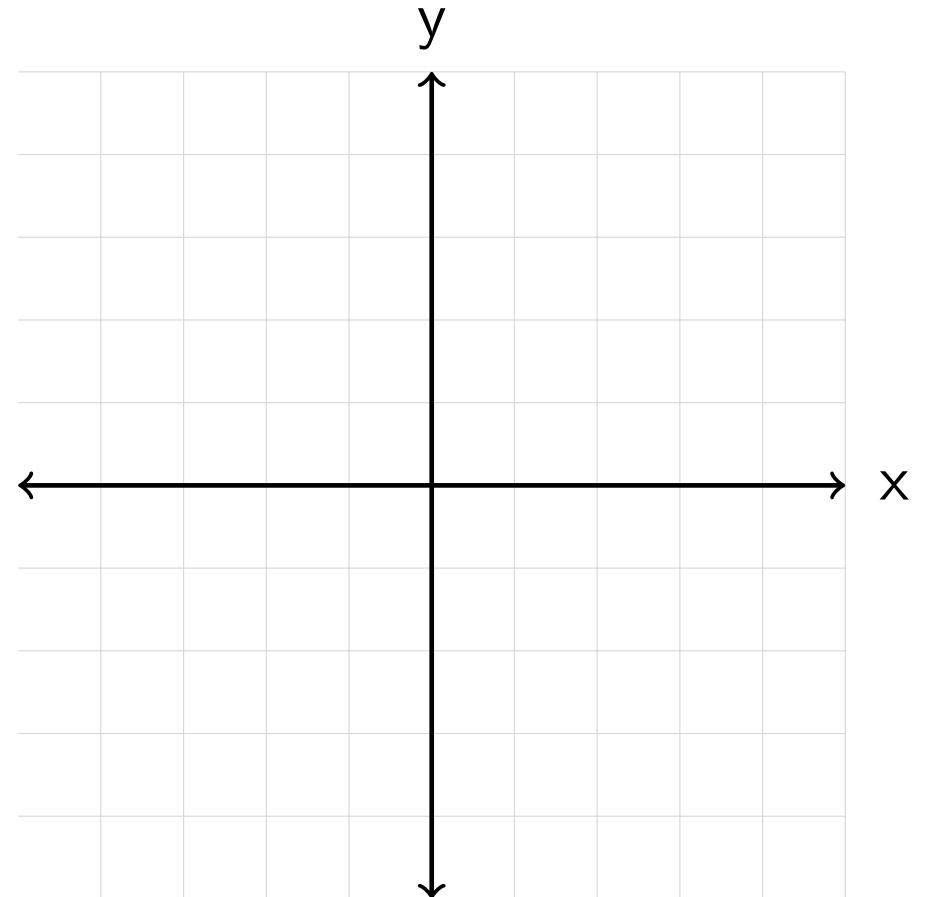
$$\mathbf{w} = r\mathbf{u} + s\mathbf{v}$$

has no solutions, which happens, e.g., if  $\mathbf{w} \cdot \mathbf{u} = 0$  and  $\mathbf{w} \cdot \mathbf{v} = 0$ . Thus

$$\begin{aligned} 0 &= \mathbf{w} \cdot \mathbf{v} = w_1 \\ \Rightarrow 0 &= \mathbf{w} \cdot \mathbf{u} = 2w_2 + 3w_3 \Rightarrow w_2 = -\frac{3}{2}w_3. \end{aligned}$$

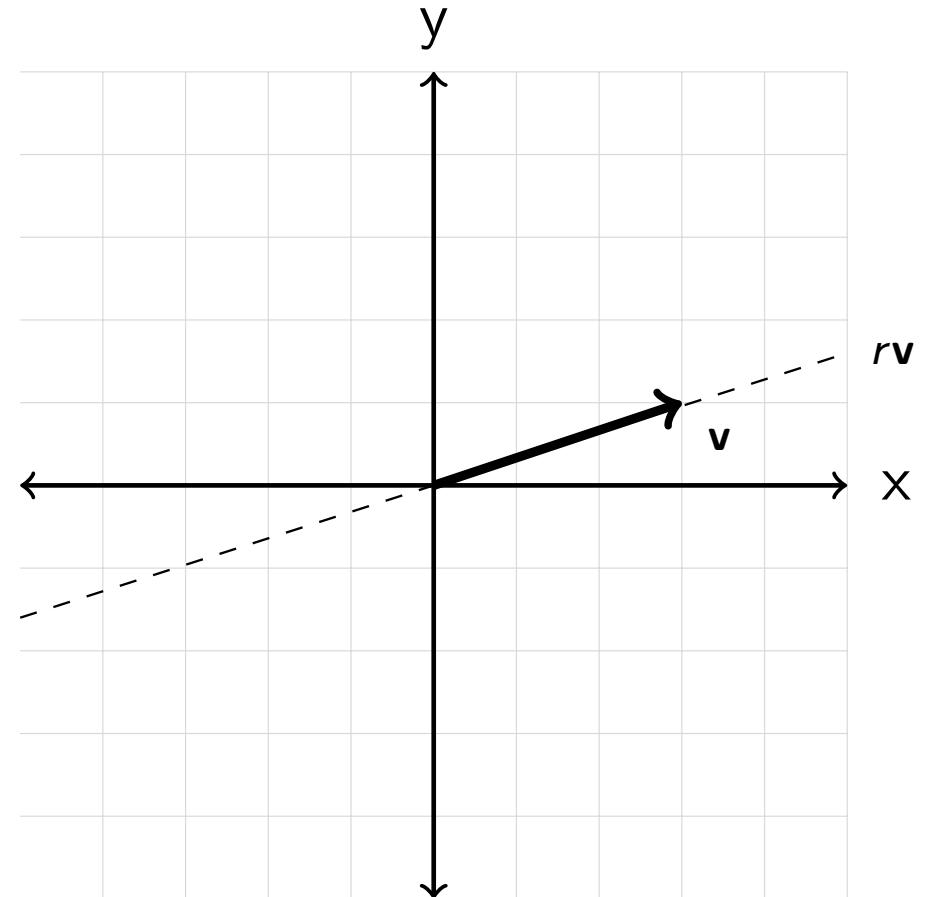
Hence, e.g.,  $\mathbf{w} = (0, -\frac{3}{2}, 1)$ . Or you use the cross-product.

# Computing a basis representation (coordinates)



# Computing a basis representation (coordinates)

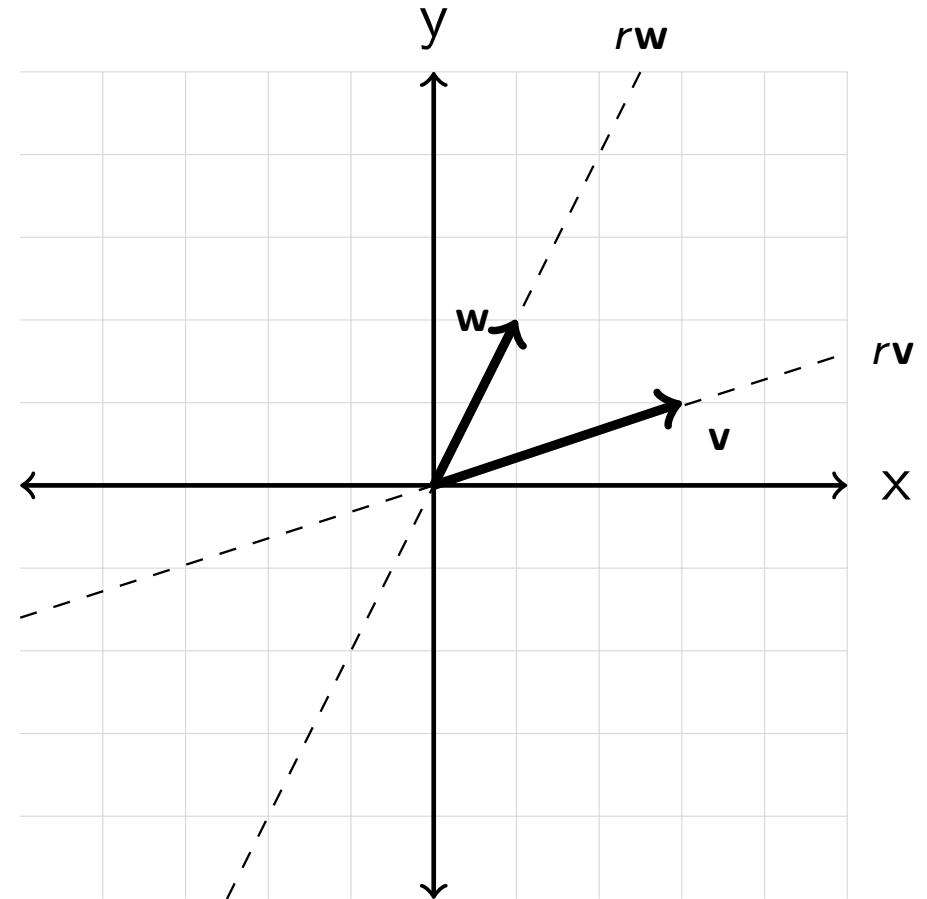
Given a basis  $\mathbf{v} = \left(\frac{3}{2}, \frac{1}{2}\right)^t$ ,



# Computing a basis representation (coordinates)

Given a basis  $\mathbf{v} = \left(\frac{3}{2}, \frac{1}{2}\right)^t$ ,

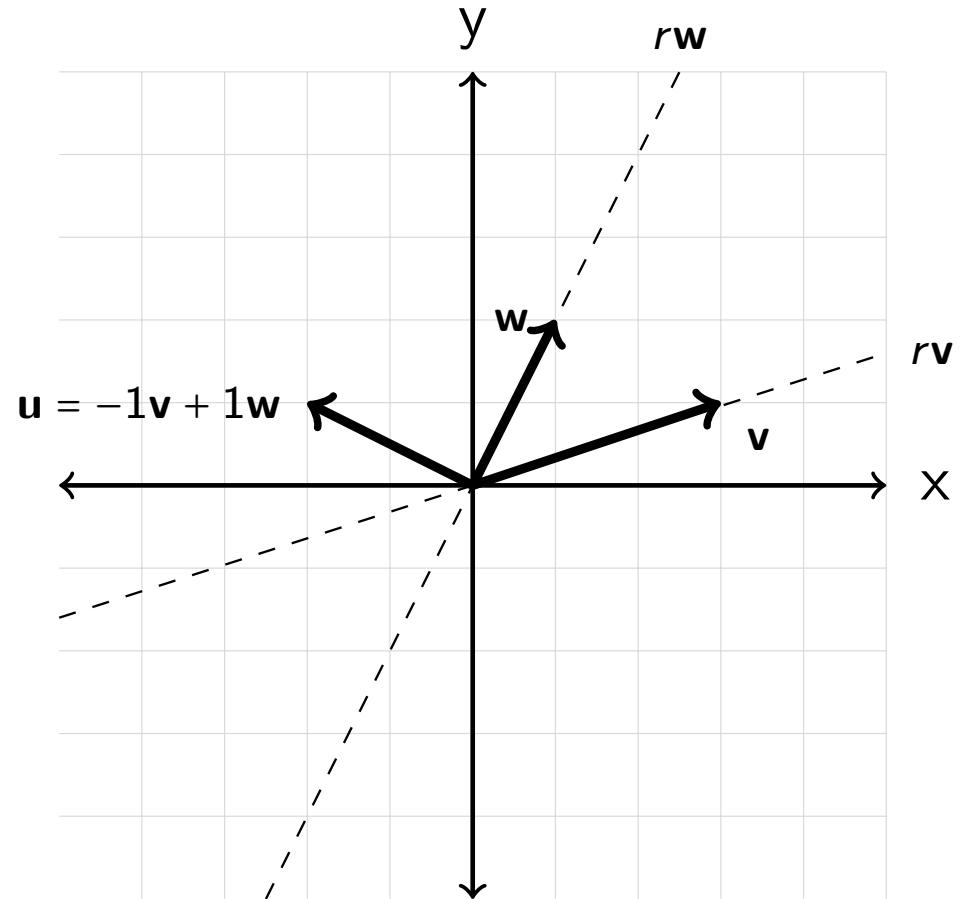
$\mathbf{w} = \left(\frac{1}{2}, 1\right)^t$ ,



# Computing a basis representation (coordinates)

Given a basis  $\mathbf{v} = \left(\frac{3}{2}, \frac{1}{2}\right)^t$ ,

$\mathbf{w} = \left(\frac{1}{2}, 1\right)^t$ ,

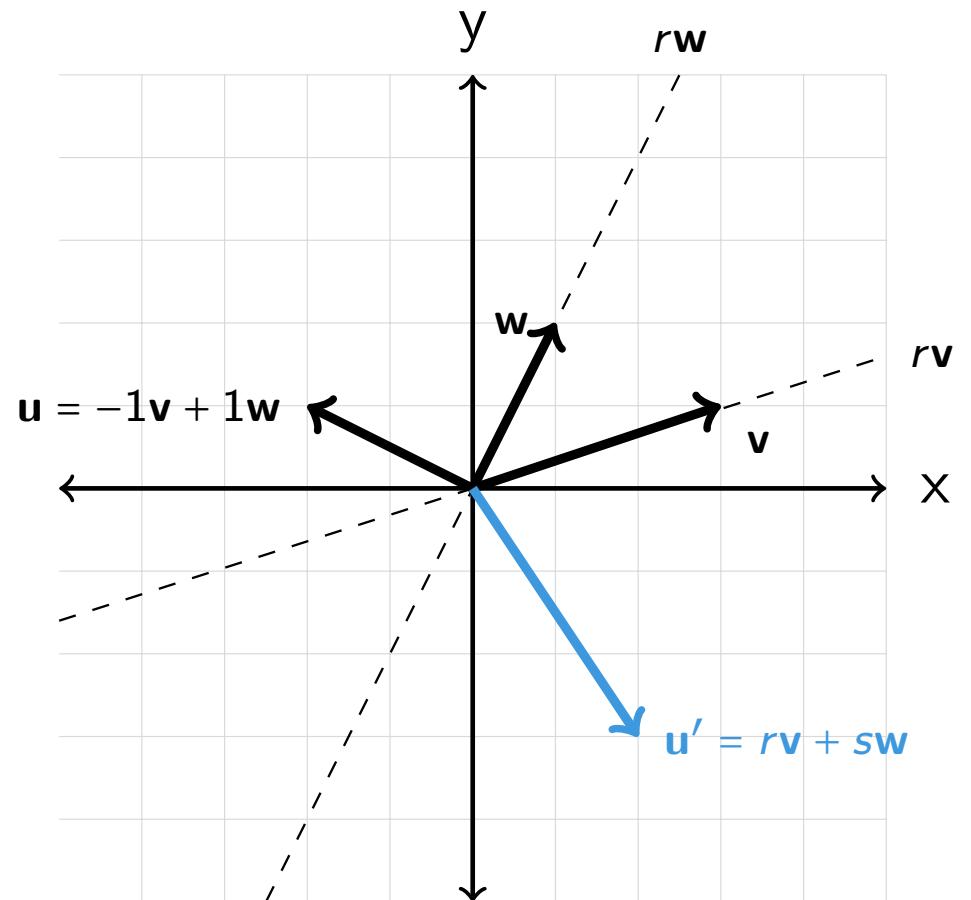


# Computing a basis representation (coordinates)

Given a basis  $\mathbf{v} = \left(\frac{3}{2}, \frac{1}{2}\right)^t$ ,

$\mathbf{w} = \left(\frac{1}{2}, 1\right)^t$ ,

how to find coordinates  $r, s$  to express the vector  $\mathbf{u}' = \left(1, -\frac{3}{2}\right)$  in this basis?



# Computing a basis representation (coordinates)

Given a basis  $\mathbf{v} = \left(\frac{3}{2}, \frac{1}{2}\right)^t$ ,

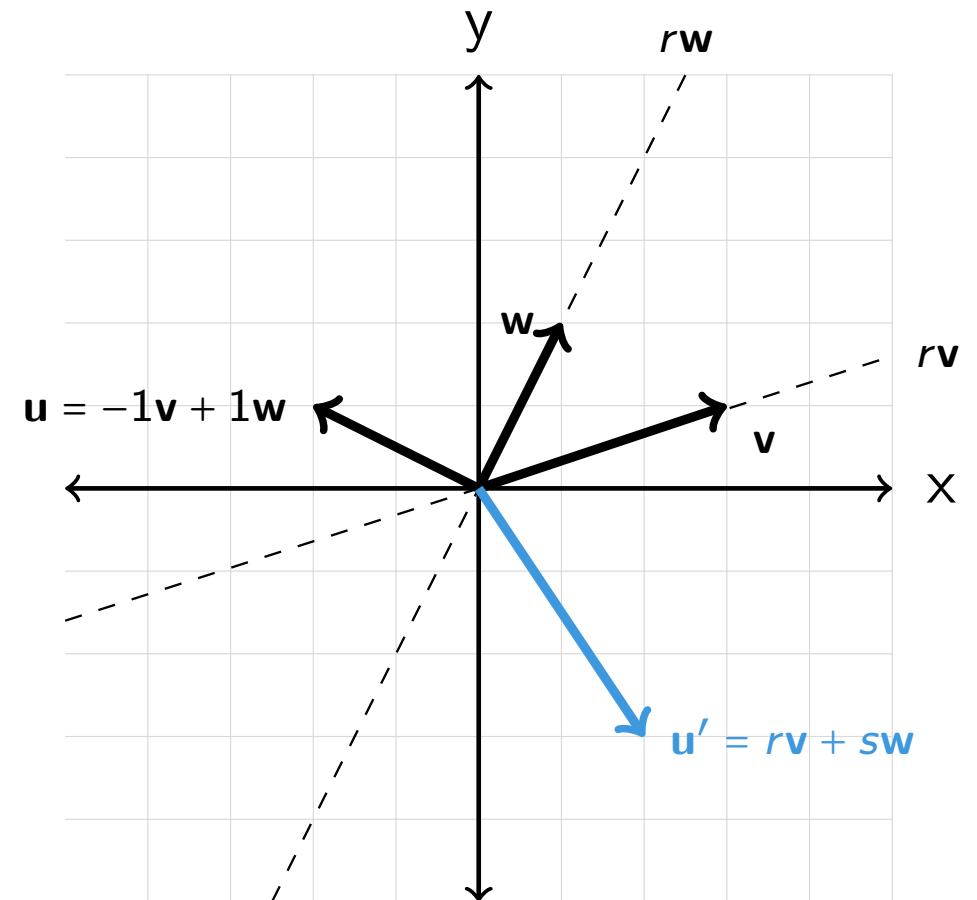
$\mathbf{w} = \left(\frac{1}{2}, 1\right)^t$ ,

how to find coordinates  $r, s$  to express the vector  $\mathbf{u}' = \left(1, -\frac{3}{2}\right)$  in this basis?

Solve the linear system

$$I \quad \frac{3}{2}r + \frac{1}{2}s = 1$$

$$II \quad \frac{1}{2}r + 1s = -\frac{3}{2}$$



# Computing a basis representation (coordinates)

Given a basis  $\mathbf{v} = \left(\frac{3}{2}, \frac{1}{2}\right)^t$ ,

$\mathbf{w} = \left(\frac{1}{2}, 1\right)^t$ ,

how to find coordinates  $r, s$  to express the vector  $\mathbf{u}' = \left(1, -\frac{3}{2}\right)$  in this basis?

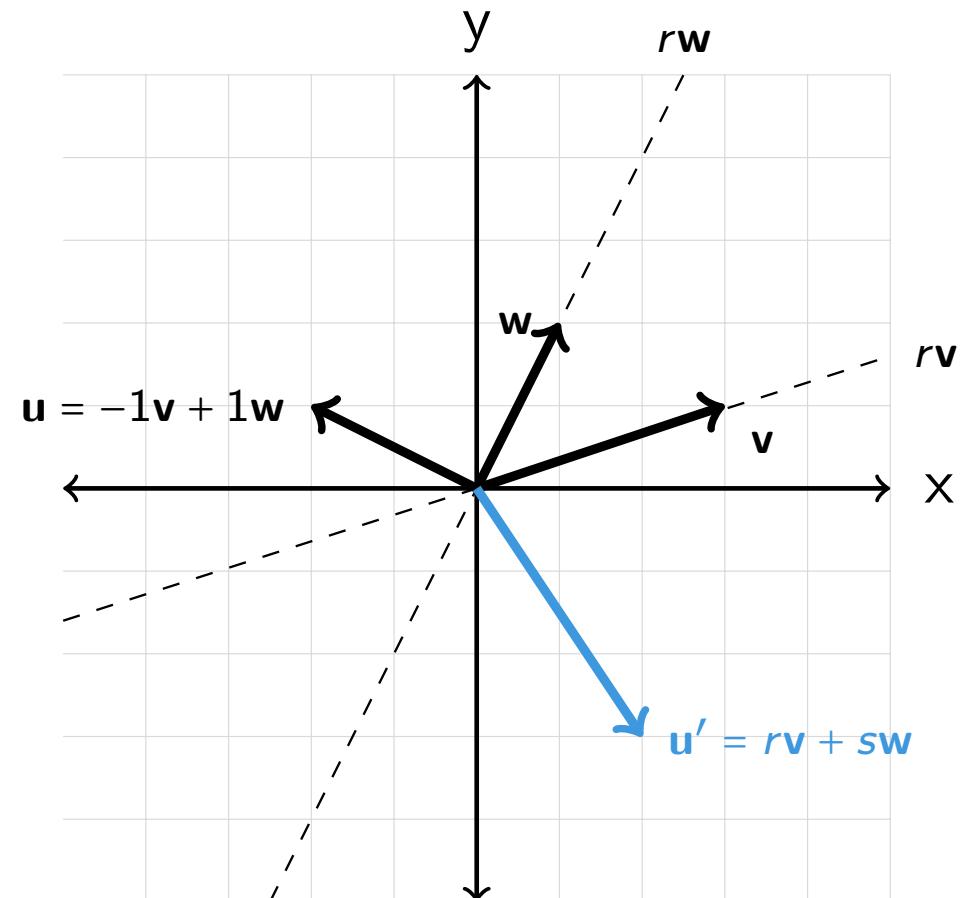
Solve the linear system

$$I \quad \frac{3}{2}r + \frac{1}{2}s = 1$$

$$II \quad \frac{1}{2}r + 1s = -\frac{3}{2}$$

e.g., by computing

$$II - 2I = -\frac{5}{2}r = -\frac{7}{2} \Rightarrow r = \frac{14}{10}$$



# Computing a basis representation (coordinates)

Given a basis  $\mathbf{v} = \left(\frac{3}{2}, \frac{1}{2}\right)^t$ ,

$\mathbf{w} = \left(\frac{1}{2}, 1\right)^t$ ,

how to find coordinates  $r, s$  to express the vector  $\mathbf{u}' = \left(1, -\frac{3}{2}\right)$  in this basis?

Solve the linear system

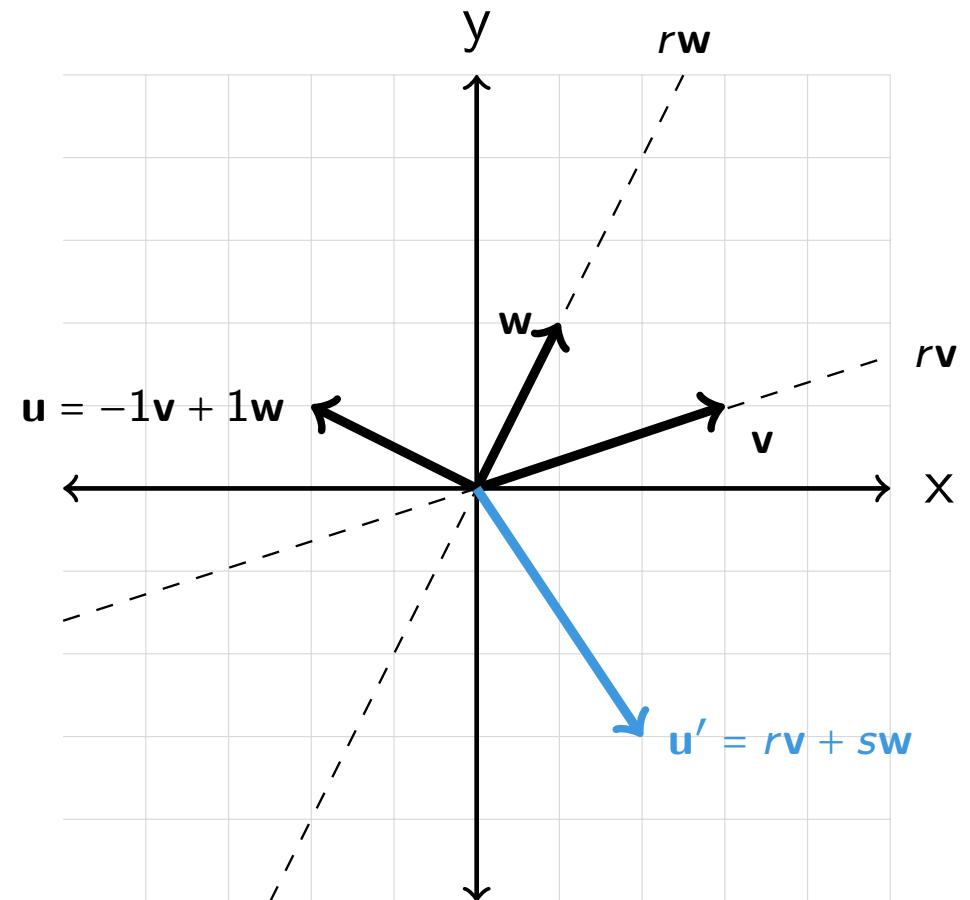
$$I \quad \frac{3}{2}r + \frac{1}{2}s = 1$$

$$II \quad \frac{1}{2}r + 1s = -\frac{3}{2}$$

e.g., by computing

$$II - 2I = -\frac{5}{2}r = -\frac{7}{2} \Rightarrow r = \frac{14}{10}$$

$$\Rightarrow \frac{7}{10} + s = -\frac{3}{2} \Rightarrow s = -\frac{22}{10}.$$



# Definition Matrix

<https://vevox.app/#/m/106717265>

# Definition Matrix

In  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , all linear maps can be written as *matrices*, i.e.,

<https://vevox.app/#/m/106717265>

# Definition Matrix

In  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , all linear maps can be written as *matrices*, i.e.,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

<https://vevox.app/#/m/106717265>

# Definition Matrix

In  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , all linear maps can be written as *matrices*, i.e.,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

<https://vevox.app/#/m/106717265>

# Definition Matrix

In  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , all linear maps can be written as *matrices*, i.e.,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

To multiply two matrices  $A, B$ , we compute

$$\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$
$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

<https://vevox.app/#/m/106717265>

## Definition Matrix

In  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , all linear maps can be written as *matrices*, i.e.,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

To multiply two matrices  $A, B$ , we compute

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}.$$

<https://vevox.app/#/m/106717265>

# Definition Matrix

In  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , all linear maps can be written as *matrices*, i.e.,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

To multiply two matrices  $A, B$ , we compute

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}.$$

But, generally,  $AB \neq BA$ , e.g.,

<https://vevox.app/#/m/106717265>

## Definition Matrix

In  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , all linear maps can be written as *matrices*, i.e.,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

To multiply two matrices  $A, B$ , we compute

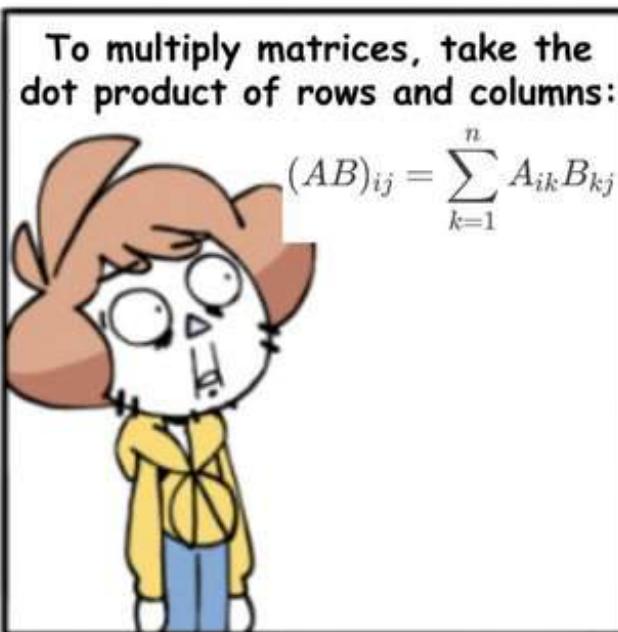
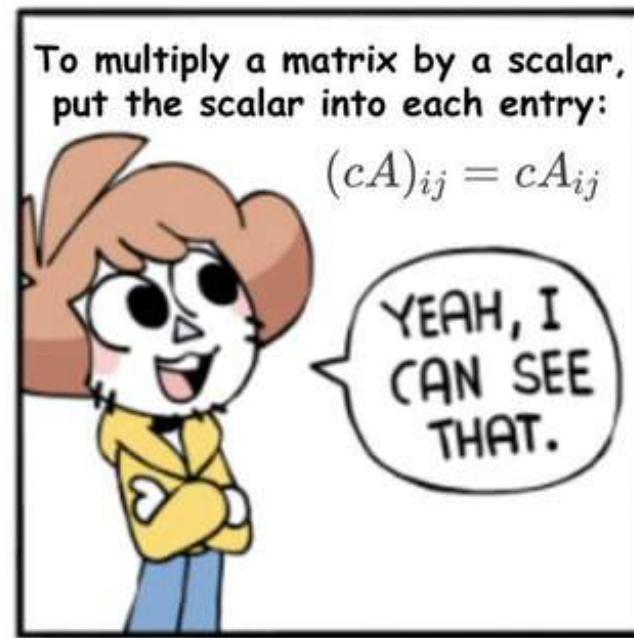
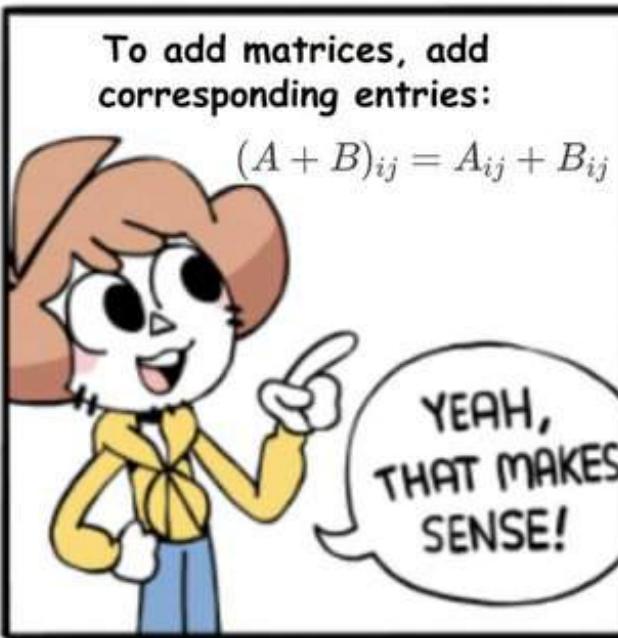
$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}.$$

But, generally,  $AB \neq BA$ , e.g.,

$$\begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 3 & 7 \\ 4 & 8 \end{pmatrix}, \text{ but } \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 7 \\ 2 & 10 \end{pmatrix}.$$

<https://vevox.app/#/m/106717265>

# Matrix Multiplication



# Determinant

# Determinant

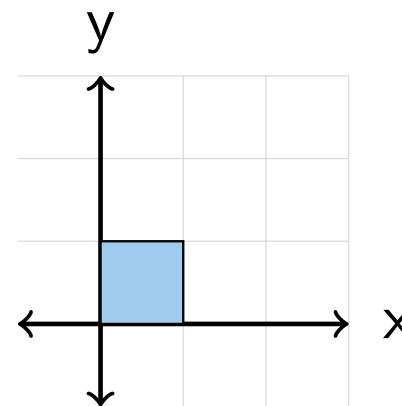
Given a matrix  $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ , its determinant is given by

$$\det(A) = a_{11}a_{22} - a_{21}a_{12}.$$

# Determinant

Given a matrix  $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ , its determinant is given by

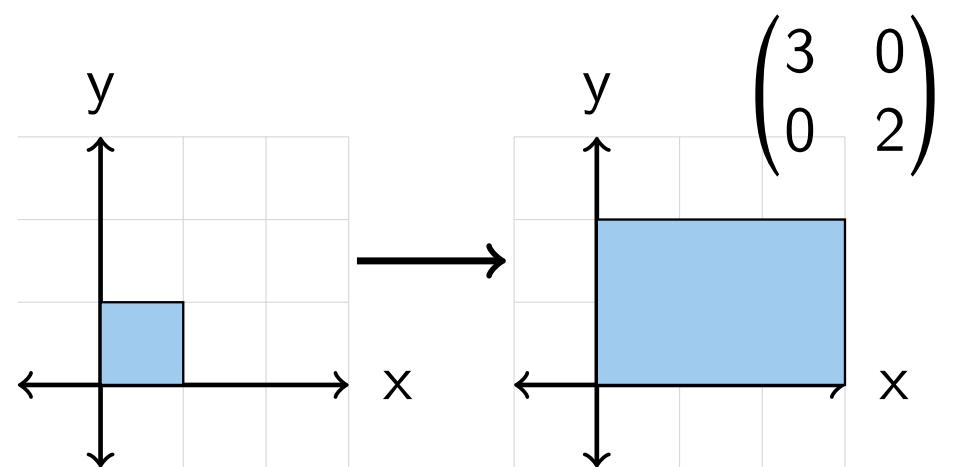
$$\det(A) = a_{11}a_{22} - a_{21}a_{12}.$$



# Determinant

Given a matrix  $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ , its determinant is given by

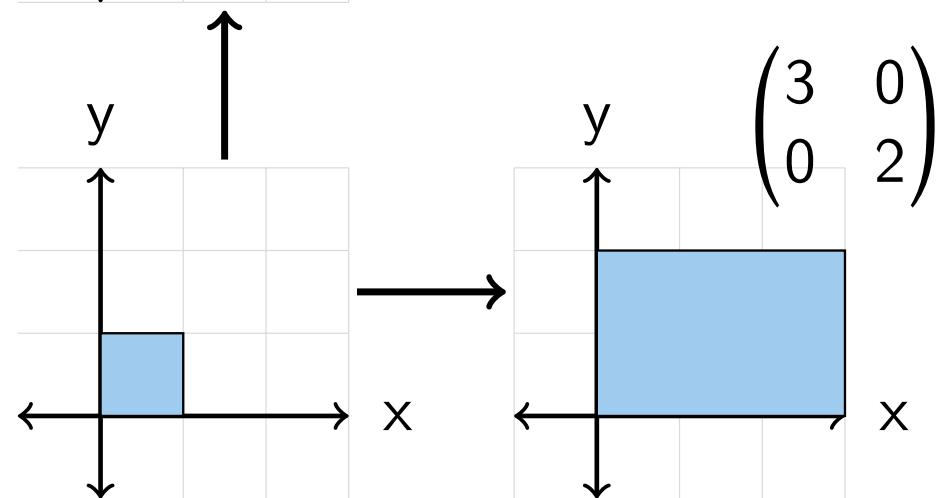
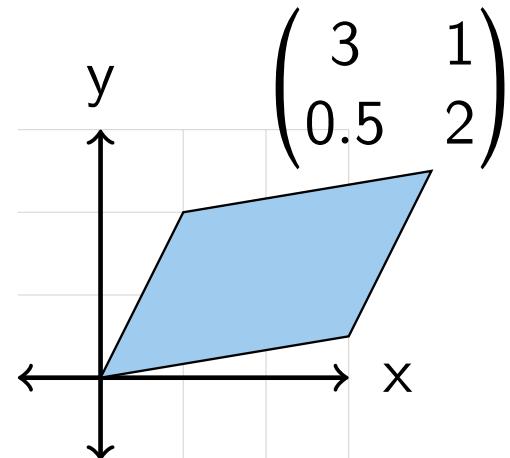
$$\det(A) = a_{11}a_{22} - a_{21}a_{12}.$$



# Determinant

Given a matrix  $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ , its determinant is given by

$$\det(A) = a_{11}a_{22} - a_{21}a_{12}.$$

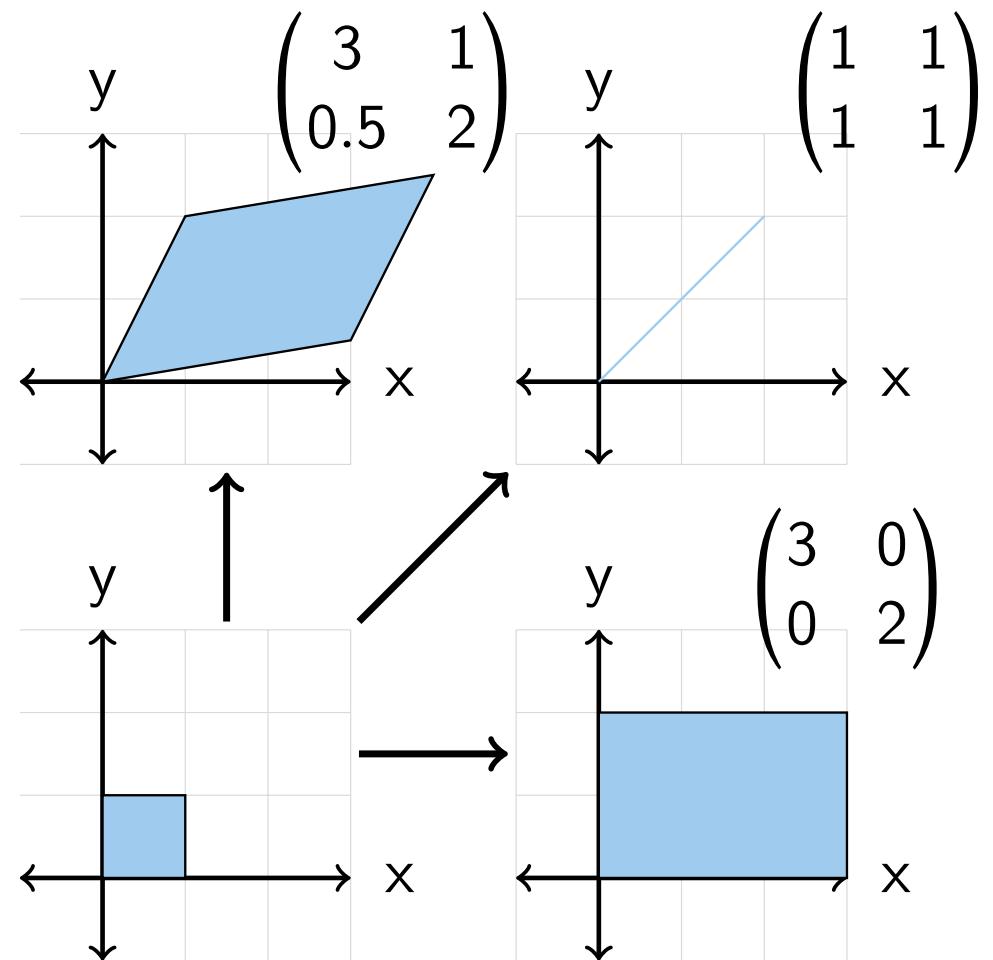


# Determinant

Given a matrix  $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ , its determinant is given by

$$\det(A) = a_{11}a_{22} - a_{21}a_{12}.$$

**Question:** What happens in the degenerate case?



# Determinant

Given a matrix  $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ , its determinant is given by

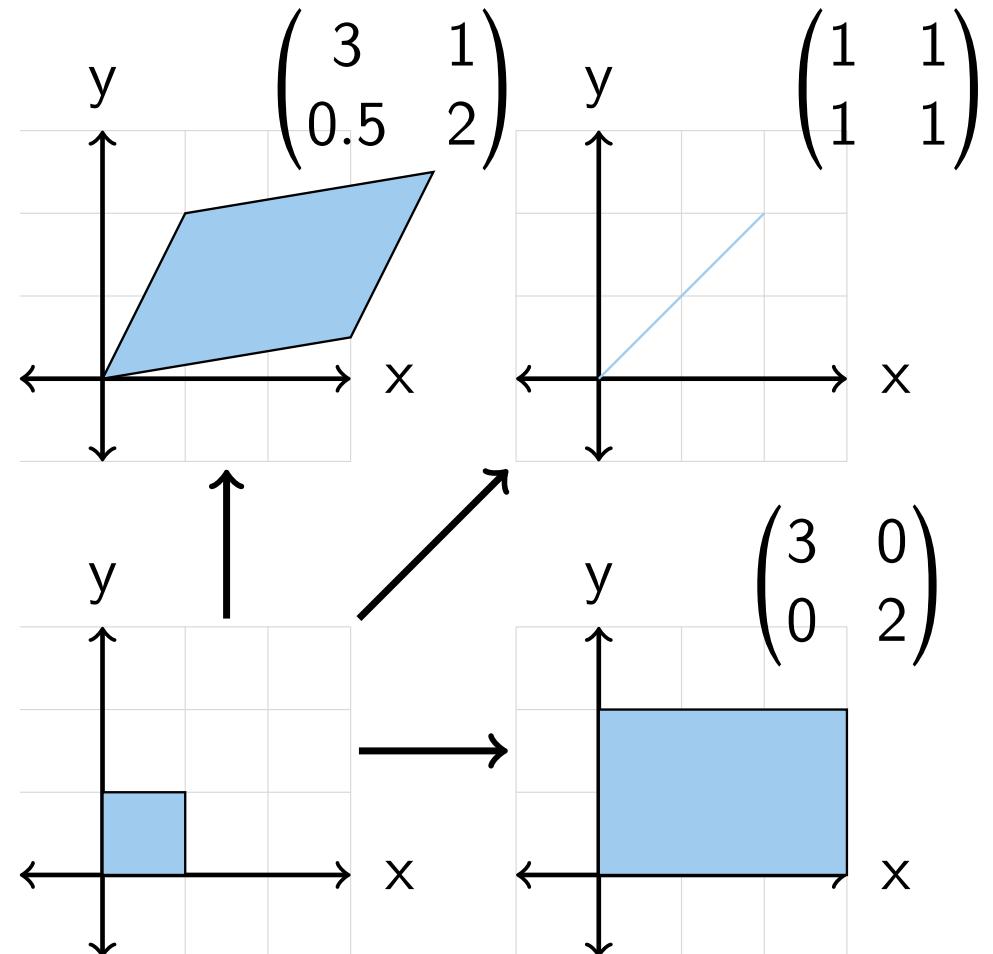
$$\det(A) = a_{11}a_{22} - a_{21}a_{12}.$$

**Question:** What happens in the degenerate case?

For a matrix  $A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$

it is:

$$a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}.$$



# Determinant

# Determinant

$a_{11}$	$a_{12}$	$a_{13}$
$a_{21}$	$a_{22}$	$a_{23}$
$a_{31}$	$a_{32}$	$a_{33}$

# Determinant

$a_{11}$	$a_{12}$	$a_{13}$	$a_{11}$	$a_{12}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{21}$	$a_{22}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{31}$	$a_{32}$

# Determinant

$$a_{11} a_{22} a_{33}$$

$a_{11}$	$a_{12}$	$a_{13}$	$a_{11}$	$a_{12}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{21}$	$a_{22}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{31}$	$a_{32}$

# Determinant

$$a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31}$$

$a_{11}$	$a_{12}$	$a_{13}$	$a_{11}$	$a_{12}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{21}$	$a_{22}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{31}$	$a_{32}$

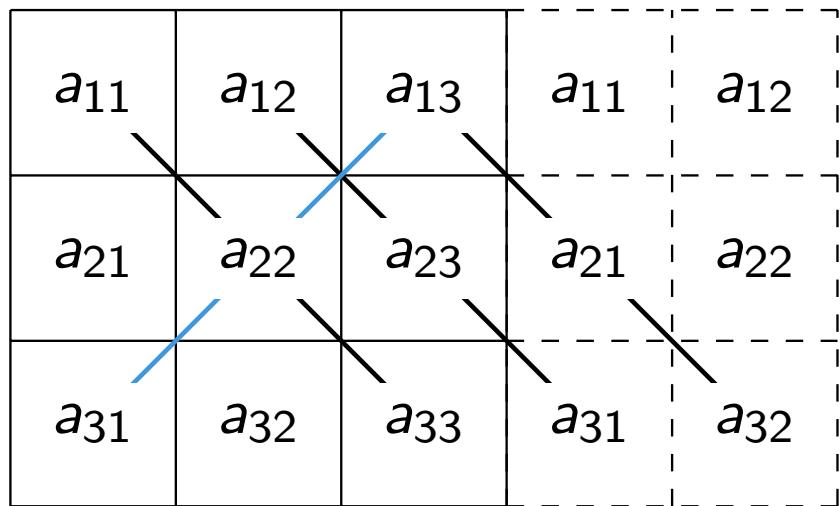
# Determinant

$$a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32}$$

$a_{11}$	$a_{12}$	$a_{13}$	$a_{11}$	$a_{12}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{21}$	$a_{22}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{31}$	$a_{32}$

# Determinant

$$a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13}$$



# Determinant

$$a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11}$$

$a_{11}$	$a_{12}$	$a_{13}$	$a_{11}$	$a_{12}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{21}$	$a_{22}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{31}$	$a_{32}$

# Determinant

$$a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}.$$

$a_{11}$	$a_{12}$	$a_{13}$	$a_{11}$	$a_{12}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{21}$	$a_{22}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{31}$	$a_{32}$

The diagram shows a 3x3 matrix with elements  $a_{ij}$ . Blue lines connect the first column elements ( $a_{11}, a_{21}, a_{31}$ ) to the second column, and black lines connect them to the third column. The second row elements ( $a_{22}, a_{23}$ ) are also connected by blue lines to the third column. The third row element ( $a_{32}$ ) is connected by a black line to the second column. This visualizes the expansion of the determinant along the first column.

# Determinant

$$a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}.$$

$a_{11}$	$a_{12}$	$a_{13}$	$a_{11}$	$a_{12}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{21}$	$a_{22}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{31}$	$a_{32}$

**Cofactors**

**Definition:**  
The  $(i, j)$ -cofactor of a matrix  $A$  is  $C_{ij}$  and is given by

$$C_{ij} = (-1)^{i+j} \det(A_{ij}).$$

$A = \begin{bmatrix} 1 & 5 & 0 \\ 2 & 4 & -1 \\ 0 & -2 & 0 \end{bmatrix}$        $\begin{bmatrix} + & - & + & - & \dots \\ - & + & - & + & \dots \\ + & - & + & - & \dots \\ - & + & - & + & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$

$$C_{23} = (-1)^5 \begin{vmatrix} 1 & 5 \\ 0 & -2 \end{vmatrix} = 2$$

Figure: [https://ocw.tudelft.nl/  
course-lectures/determinants/](https://ocw.tudelft.nl/course-lectures/determinants/)

# Matrix Inverse

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case?)

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case? – Check that determinant  $\neq 0$ ), it can be inverted i.e., there is a matrix  $A^{-1}$  such that  $AA^{-1} = \mathcal{I}$ , with  $\mathcal{I}$  being the *identity matrix*.

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case? – Check that determinant  $\neq 0$ ), it can be inverted i.e., there is a matrix  $A^{-1}$  such that  $AA^{-1} = \mathcal{I}$ , with  $\mathcal{I}$  being the *identity matrix*. Compute the inverse of a matrix  $A$  by solving a linear system,

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case? – Check that determinant  $\neq 0$ ), it can be inverted i.e., there is a matrix  $A^{-1}$  such that  $AA^{-1} = \mathcal{I}$ , with  $\mathcal{I}$  being the *identity matrix*. Compute the inverse of a matrix  $A$  by solving a linear system, e.g.,

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case? – Check that determinant  $\neq 0$ ), it can be inverted i.e., there is a matrix  $A^{-1}$  such that  $AA^{-1} = \mathcal{I}$ , with  $\mathcal{I}$  being the *identity matrix*. Compute the inverse of a matrix  $A$  by solving a linear system, e.g.,

$$\begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \begin{array}{ll} I & a_1 + 2a_3 = 1 \\ II & a_1 + 3a_3 = 0 \\ III & a_2 + 2a_4 = 0 \\ IV & a_2 + 3a_4 = 1 \end{array}$$

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case? – Check that determinant  $\neq 0$ ), it can be inverted i.e., there is a matrix  $A^{-1}$  such that  $AA^{-1} = \mathcal{I}$ , with  $\mathcal{I}$  being the *identity matrix*. Compute the inverse of a matrix  $A$  by solving a linear system, e.g.,

$$\begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \begin{array}{ll} I & a_1 + 2a_3 = 1 \\ II & a_1 + 3a_3 = 0 \\ III & a_2 + 2a_4 = 0 \\ IV & a_2 + 3a_4 = 1 \end{array}$$
$$\Rightarrow II - I = a_3 = -1 \Rightarrow a_1 = 3$$

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case? – Check that determinant  $\neq 0$ ), it can be inverted i.e., there is a matrix  $A^{-1}$  such that  $AA^{-1} = \mathcal{I}$ , with  $\mathcal{I}$  being the *identity matrix*. Compute the inverse of a matrix  $A$  by solving a linear system, e.g.,

$$\begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \begin{array}{ll} I & a_1 + 2a_3 = 1 \\ II & a_1 + 3a_3 = 0 \\ III & a_2 + 2a_4 = 0 \\ IV & a_2 + 3a_4 = 1 \end{array}$$
$$\Rightarrow II - I = a_3 = -1 \Rightarrow a_1 = 3$$
$$\Rightarrow IV - III = a_4 = 1 \Rightarrow a_2 = -2$$

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case? – Check that determinant  $\neq 0$ ), it can be inverted i.e., there is a matrix  $A^{-1}$  such that  $AA^{-1} = \mathcal{I}$ , with  $\mathcal{I}$  being the *identity matrix*. Compute the inverse of a matrix  $A$  by solving a linear system, e.g.,

$$\begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \begin{array}{ll} I & a_1 + 2a_3 = 1 \\ II & a_1 + 3a_3 = 0 \\ III & a_2 + 2a_4 = 0 \\ IV & a_2 + 3a_4 = 1 \end{array}$$
$$\Rightarrow II - I = a_3 = -1 \Rightarrow a_1 = 3$$
$$\Rightarrow IV - III = a_4 = 1 \Rightarrow a_2 = -2$$

The inverse to  $A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix}$  is  $A^{-1} = \begin{pmatrix} 3 & -2 \\ -1 & 1 \end{pmatrix}$ .

# Matrix Inverse

If a matrix  $A$  has *full rank* (**Question:** When is that the case? – Check that determinant  $\neq 0$ ), it can be inverted i.e., there is a matrix  $A^{-1}$  such that  $AA^{-1} = \mathcal{I}$ , with  $\mathcal{I}$  being the *identity matrix*. Compute the inverse of a matrix  $A$  by solving a linear system, e.g.,

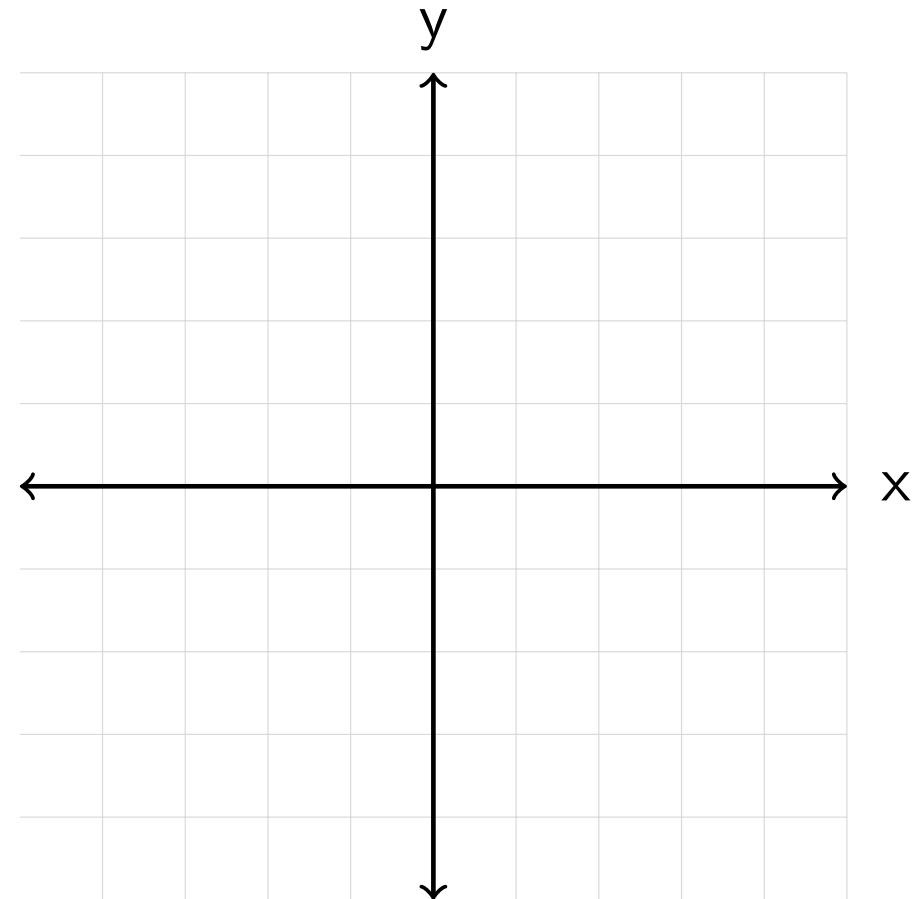
$$\begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \begin{array}{ll} I & a_1 + 2a_3 = 1 \\ II & a_1 + 3a_3 = 0 \\ III & a_2 + 2a_4 = 0 \\ IV & a_2 + 3a_4 = 1 \end{array}$$
$$\Rightarrow II - I = a_3 = -1 \Rightarrow a_1 = 3$$
$$\Rightarrow IV - III = a_4 = 1 \Rightarrow a_2 = -2$$

The inverse to  $A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix}$  is  $A^{-1} = \begin{pmatrix} 3 & -2 \\ -1 & 1 \end{pmatrix}$ .

Different option in the exercise!

# Definition Line

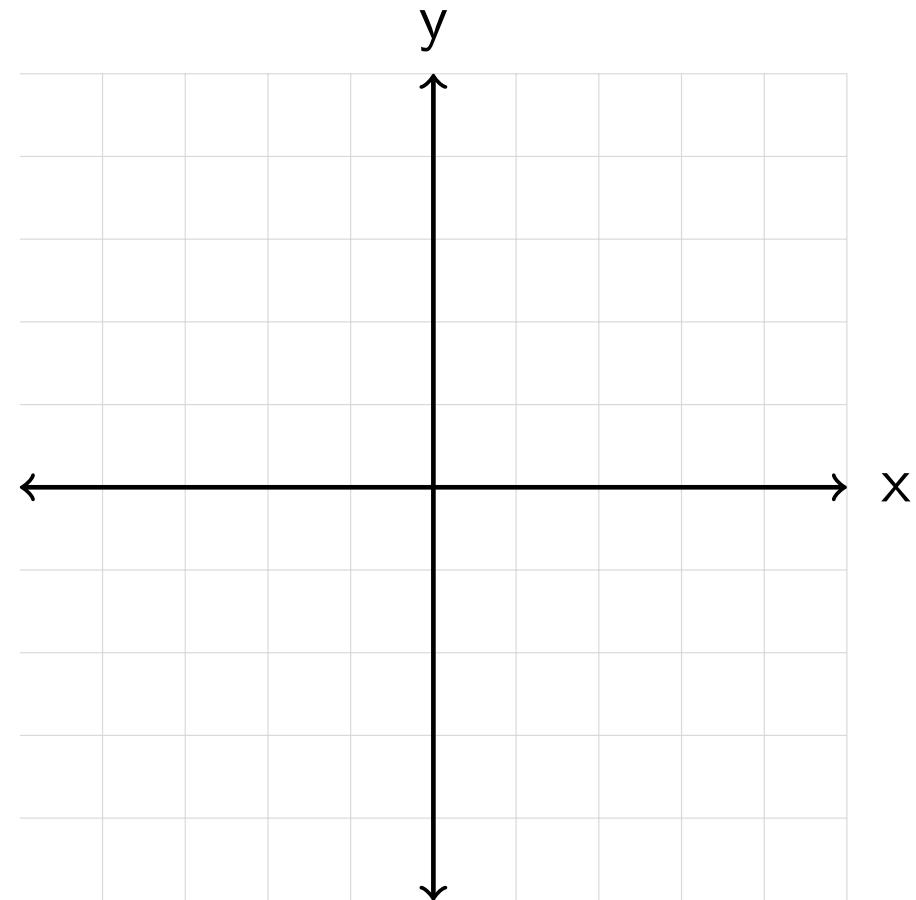
<https://vevox.app/#/m/106717265>



# Definition Line

<https://vevox.app/#/m/106717265>

A *line* in  $\mathbb{R}^2$  can be represented via the formula  $y = mx + b$  for some scalars  $m, b \in \mathbb{R}$ .

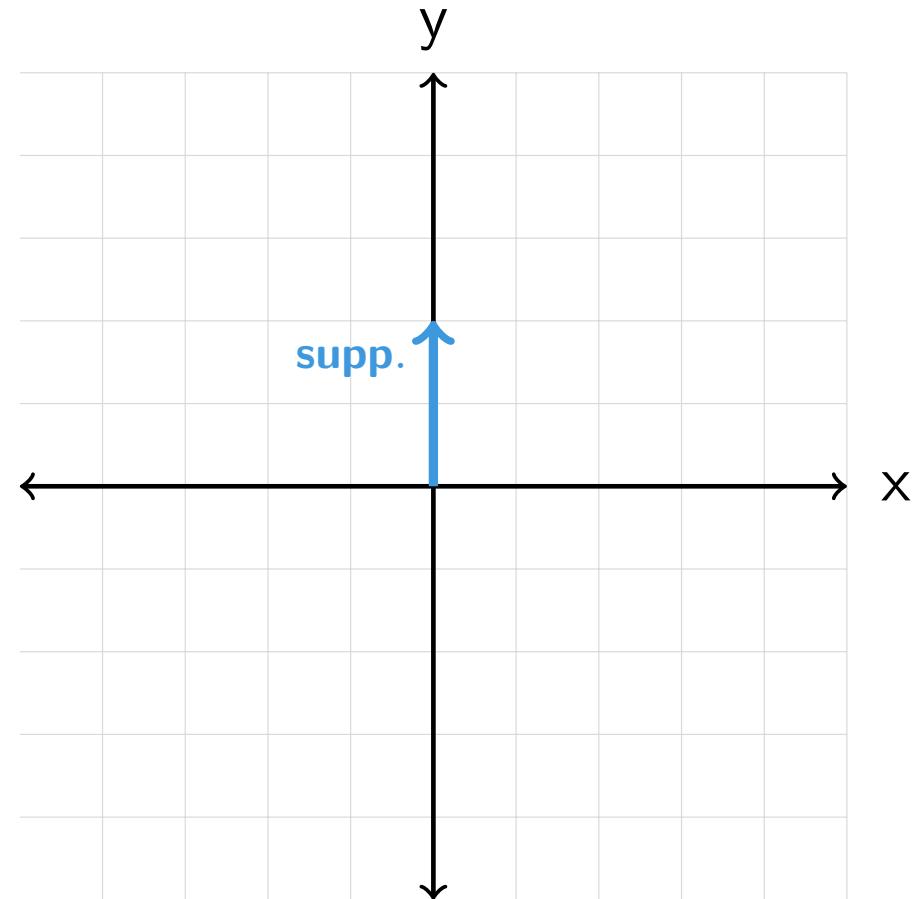


# Definition Line

<https://vevox.app/#/m/106717265>

A *line* in  $\mathbb{R}^2$  can be represented via the formula  $y = mx + b$  for some scalars  $m, b \in \mathbb{R}$ .

However, we can also write it as a composition of a *support vector* (some point on the line)

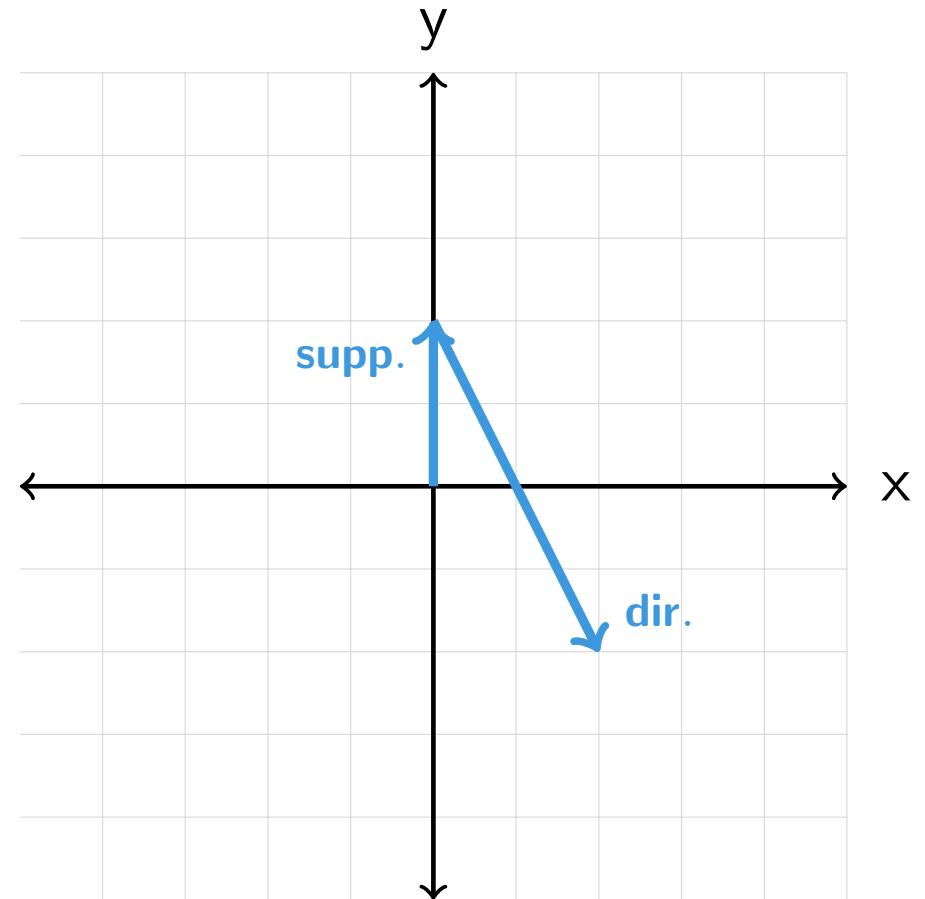


# Definition Line

<https://vevox.app/#/m/106717265>

A *line* in  $\mathbb{R}^2$  can be represented via the formula  $y = mx + b$  for some scalars  $m, b \in \mathbb{R}$ .

However, we can also write it as a composition of a *support vector* (some point on the line) and a direction vector (the direction of the line).



# Definition Line

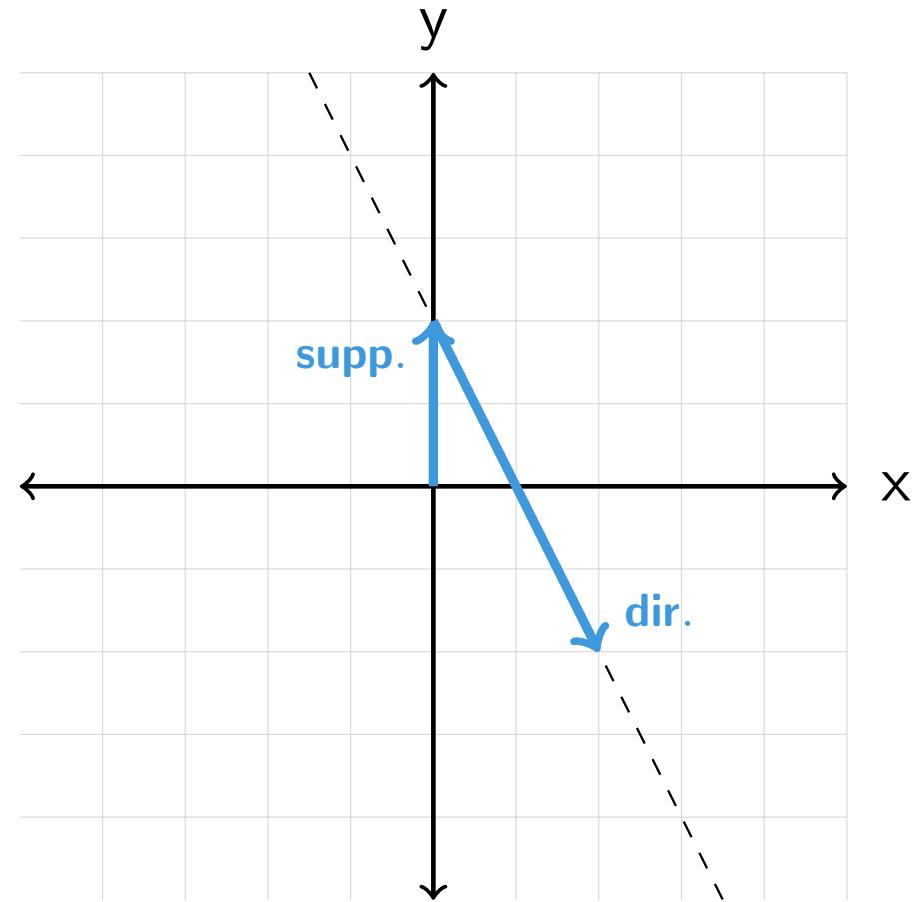
<https://vevox.app/#/m/106717265>

A *line* in  $\mathbb{R}^2$  can be represented via the formula  $y = mx + b$  for some scalars  $m, b \in \mathbb{R}$ .

However, we can also write it as a composition of a *support vector* (some point on the line) and a direction vector (the direction of the line). For instance:

$$y = -2x + 1$$

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} + r \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$



# Points on a Line

For a given line  $(1, 0, 0)^t + r(-1, 2, 1)^t$ , with  $r \in \mathbb{R}$ ,

# Points on a Line

For a given line  $(1, 0, 0)^t + r(-1, 2, 1)^t$ , with  $r \in \mathbb{R}$ , to check whether a point, e.g.,  $\mathbf{v} = (0, 2, 1)^t$  is on that line,

# Points on a Line

For a given line  $(1, 0, 0)^t + r(-1, 2, 1)^t$ , with  $r \in \mathbb{R}$ , to check whether a point, e.g.,  $\mathbf{v} = (0, 2, 1)^t$  is on that line, solve the linear system

$$(1, 0, 0)^t + r(-1, 2, 1)^t = (0, 2, 1)$$

for  $r$

# Points on a Line

For a given line  $(1, 0, 0)^t + r(-1, 2, 1)^t$ , with  $r \in \mathbb{R}$ , to check whether a point, e.g.,  $\mathbf{v} = (0, 2, 1)^t$  is on that line, solve the linear system

$$(1, 0, 0)^t + r(-1, 2, 1)^t = (0, 2, 1)$$

for  $r$  and see that  $r = 1$  is a solution,

# Points on a Line

For a given line  $(1, 0, 0)^t + r(-1, 2, 1)^t$ , with  $r \in \mathbb{R}$ , to check whether a point, e.g.,  $\mathbf{v} = (0, 2, 1)^t$  is on that line, solve the linear system

$$(1, 0, 0)^t + r(-1, 2, 1)^t = (0, 2, 1)$$

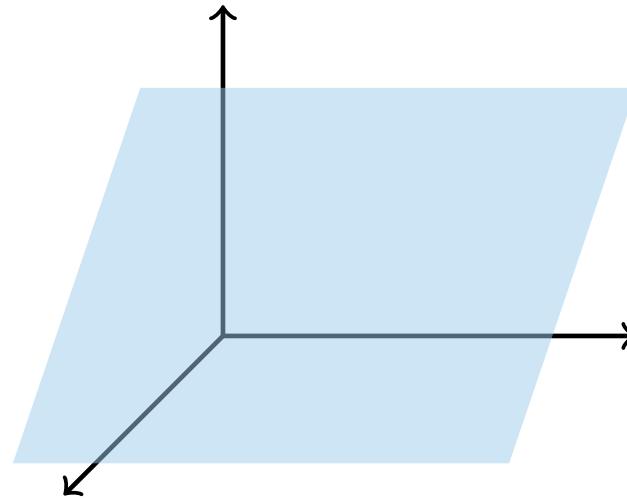
for  $r$  and see that  $r = 1$  is a solution, while for the point  $\mathbf{w} = (0, 1, 1)$  is not on the line as the linear system

$$(1, 0, 0)^t + r(-1, 2, 1)^t = (0, 1, 1)$$

does not have any solution.

# Definition Plane

**Question:** How to define a plane?

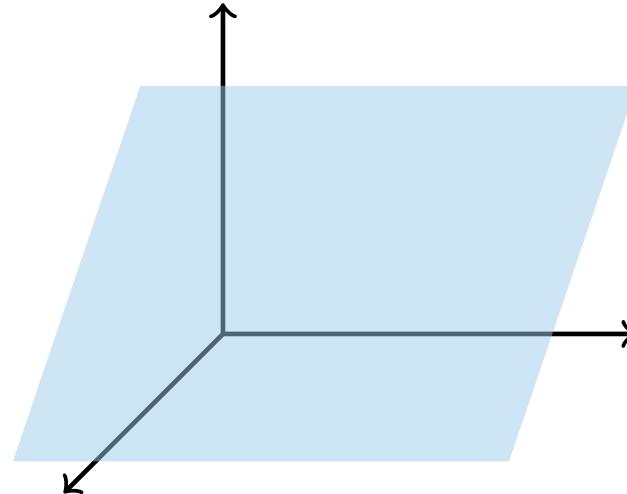


## Definition Plane

**Question:** How to define a plane?

Analog to spanning a line, in  $\mathbb{R}^3$ , we can span a plane by two spanning vectors, i.e., it is of the form

$$\mathbf{u} + r\mathbf{v} + s\mathbf{w}$$



with support vector  $\mathbf{u}$  and spanning vectors  $\mathbf{v}, \mathbf{w}$ .

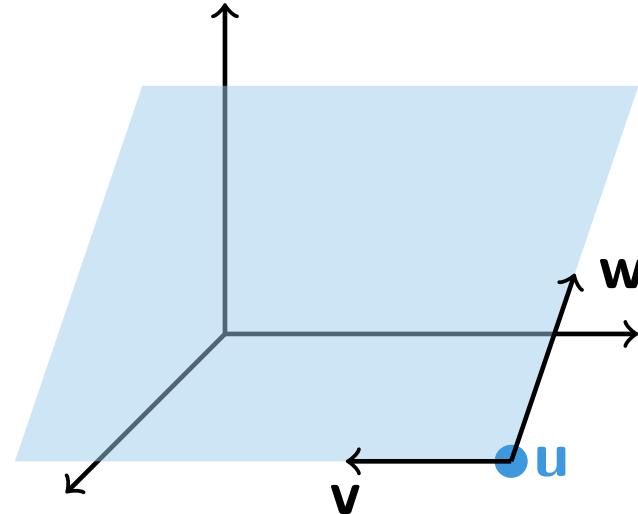
## Definition Plane

**Question:** How to define a plane?

Analog to spanning a line, in  $\mathbb{R}^3$ , we can span a plane by two spanning vectors, i.e., it is of the form

$$\mathbf{u} + r\mathbf{v} + s\mathbf{w}$$

with support vector  $\mathbf{u}$  and spanning vectors  $\mathbf{v}, \mathbf{w}$ .



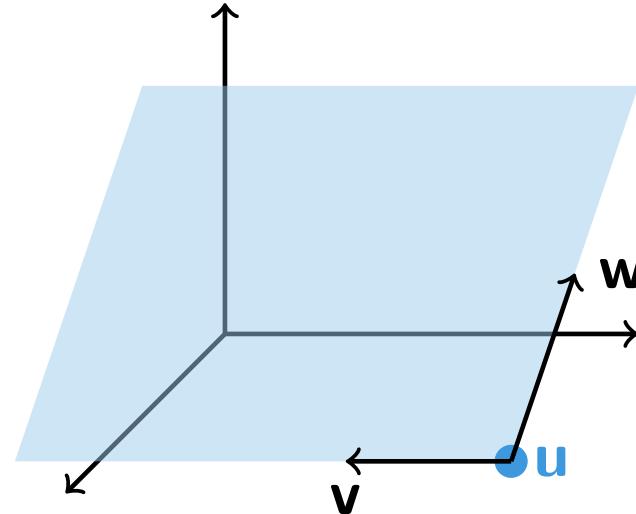
## Definition Plane

**Question:** How to define a plane?

Analog to spanning a line, in  $\mathbb{R}^3$ , we can span a plane by two spanning vectors, i.e., it is of the form

$$\mathbf{u} + r\mathbf{v} + s\mathbf{w}$$

with support vector  $\mathbf{u}$  and spanning vectors  $\mathbf{v}, \mathbf{w}$ .



Alternatively, find a *normal* vector  $\mathbf{n}$ , which is orthogonal to both  $\mathbf{v}$  and  $\mathbf{w}$ . Then, any point  $\mathbf{p}$  on the plane satisfies

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{u}) = 0.$$

# Definition Plane

**Question:** How to define a plane?

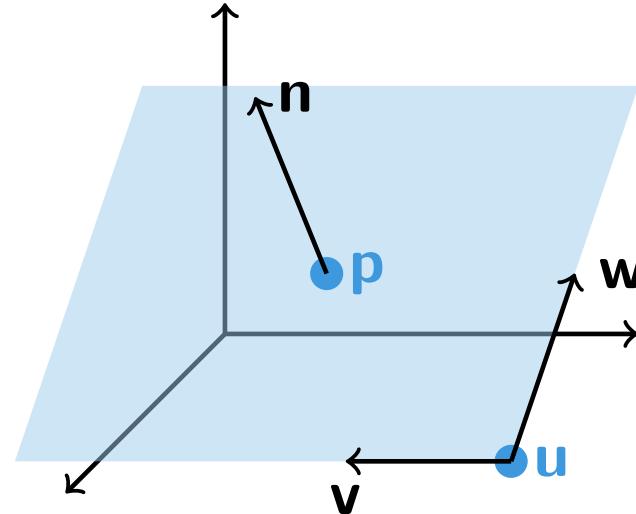
Analog to spanning a line, in  $\mathbb{R}^3$ , we can span a plane by two spanning vectors, i.e., it is of the form

$$\mathbf{u} + r\mathbf{v} + s\mathbf{w}$$

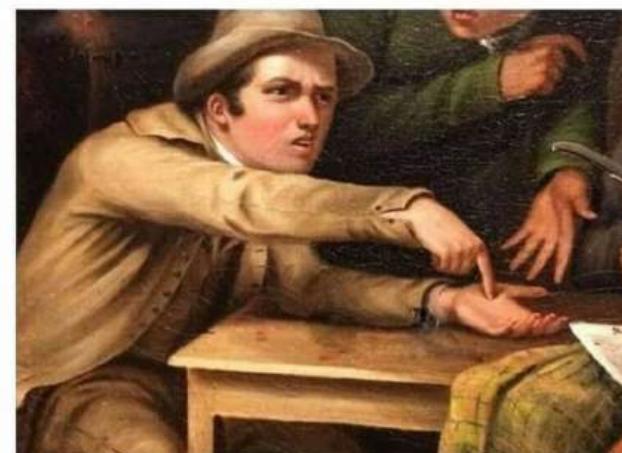
with support vector  $\mathbf{u}$  and spanning vectors  $\mathbf{v}, \mathbf{w}$ .

Alternatively, find a *normal* vector  $\mathbf{n}$ , which is orthogonal to both  $\mathbf{v}$  and  $\mathbf{w}$ . Then, any point  $\mathbf{p}$  on the plane satisfies

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{u}) = 0.$$



When your friend asks what the normal vector to a plane looks like



# Going from one plane representation to the other

# Going from one plane representation to the other

Given two spanning vectors you already know how to compute the normal (find a vector orthogonal to them via solving a linear system).

# Going from one plane representation to the other

Given two spanning vectors you already know how to compute the normal (find a vector orthogonal to them via solving a linear system).

Given the normal form of a plane, you can write down the spanning form by finding two vectors orthogonal to the normal.

# Going from one plane representation to the other

Given two spanning vectors you already know how to compute the normal (find a vector orthogonal to them via solving a linear system).

Given the normal form of a plane, you can write down the spanning form by finding two vectors orthogonal to the normal.

Consider, e.g., the plane given by  $(1, 0, 1)^t \cdot (v - (0, 1, 0)) = 0$ .

# Going from one plane representation to the other

Given two spanning vectors you already know how to compute the normal (find a vector orthogonal to them via solving a linear system).

Given the normal form of a plane, you can write down the spanning form by finding two vectors orthogonal to the normal.

Consider, e.g., the plane given by  $(1, 0, 1)^t \cdot (v - (0, 1, 0)) = 0$ . Then the vectors  $(0, 1, 0)^t$  and  $(1, 0, -1)^t$  are orthogonal to the normal  $(1, 0, 1)^t$ .

# Going from one plane representation to the other

Given two spanning vectors you already know how to compute the normal (find a vector orthogonal to them via solving a linear system).

Given the normal form of a plane, you can write down the spanning form by finding two vectors orthogonal to the normal.

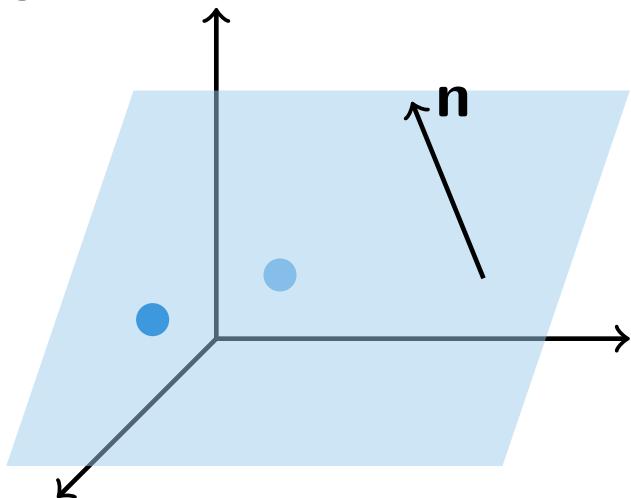
Consider, e.g., the plane given by  $(1, 0, 1)^t \cdot (v - (0, 1, 0)) = 0$ . Then the vectors  $(0, 1, 0)^t$  and  $(1, 0, -1)^t$  are orthogonal to the normal  $(1, 0, 1)^t$  and the spanning form is therefore

$$(0, 1, 0)^t + r(0, 1, 0)^t + s(1, 0, -1)^t.$$

# Points on sides of a plane

For points,  $v, w$ , you can see how they are ordered along a line spanned by  $u$  by looking at the dot product.

Hence, comparing the dot product of the normal with points on the plane and outside of the plane gives directions.



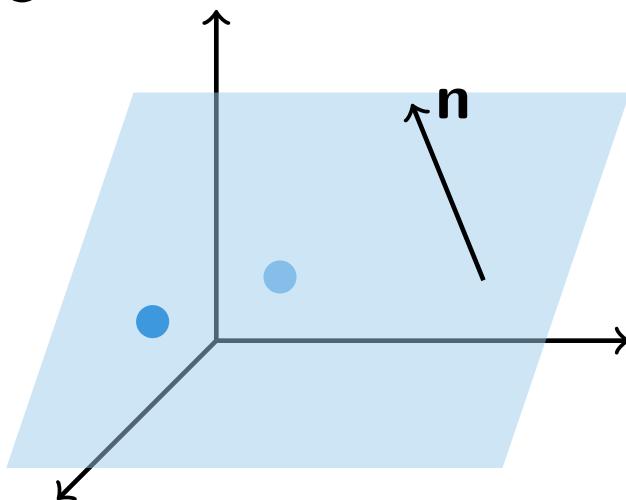
# Points on sides of a plane

For points,  $\mathbf{v}, \mathbf{w}$ , you can see how they are ordered along a line spanned by  $\mathbf{u}$  by looking at the dot product.

Hence, comparing the dot product of the normal with points on the plane and outside of the plane gives directions.

Consider the plane

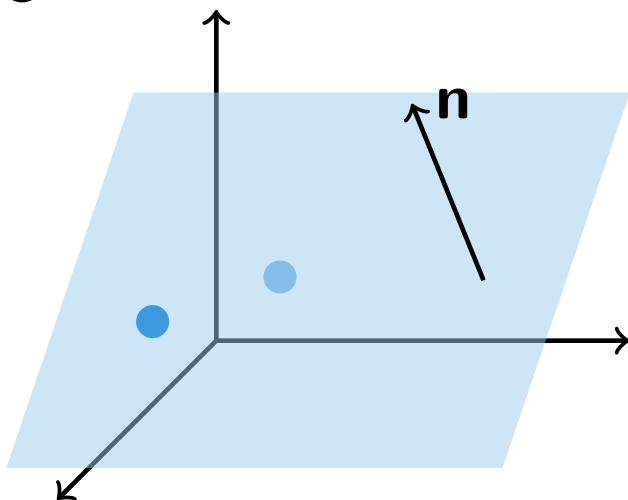
$$(0, 1, 1)^t \cdot (\mathbf{v} - (0, 0, 3)^t) = 0.$$



# Points on sides of a plane

For points,  $\mathbf{v}, \mathbf{w}$ , you can see how they are ordered along a line spanned by  $\mathbf{u}$  by looking at the dot product.

Hence, comparing the dot product of the normal with points on the plane and outside of the plane gives directions.



Consider the plane

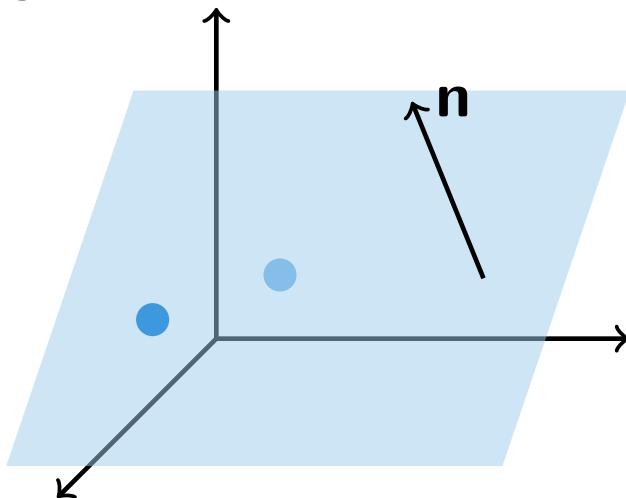
$$(0, 1, 1)^t \cdot (\mathbf{v} - (0, 0, 3)^t) = 0.$$

Then the point  $(0, 0, 3)^t$  on the plane yields  $(0, 1, 1)^t \cdot (0, 0, 3)^t = 3$ .

# Points on sides of a plane

For points,  $v, w$ , you can see how they are ordered along a line spanned by  $u$  by looking at the dot product.

Hence, comparing the dot product of the normal with points on the plane and outside of the plane gives directions.



Consider the plane

$$(0, 1, 1)^t \cdot (v - (0, 0, 3)^t) = 0.$$

Then the point  $(0, 0, 3)^t$  on the plane yields  $(0, 1, 1)^t \cdot (0, 0, 3)^t = 3$ .

While the points  $(1, 0, 1)$  and  $(0, 2, 2)$  yield

$$(0, 1, 1)^t \cdot (1, 0, 1)^t = 1 < 3$$

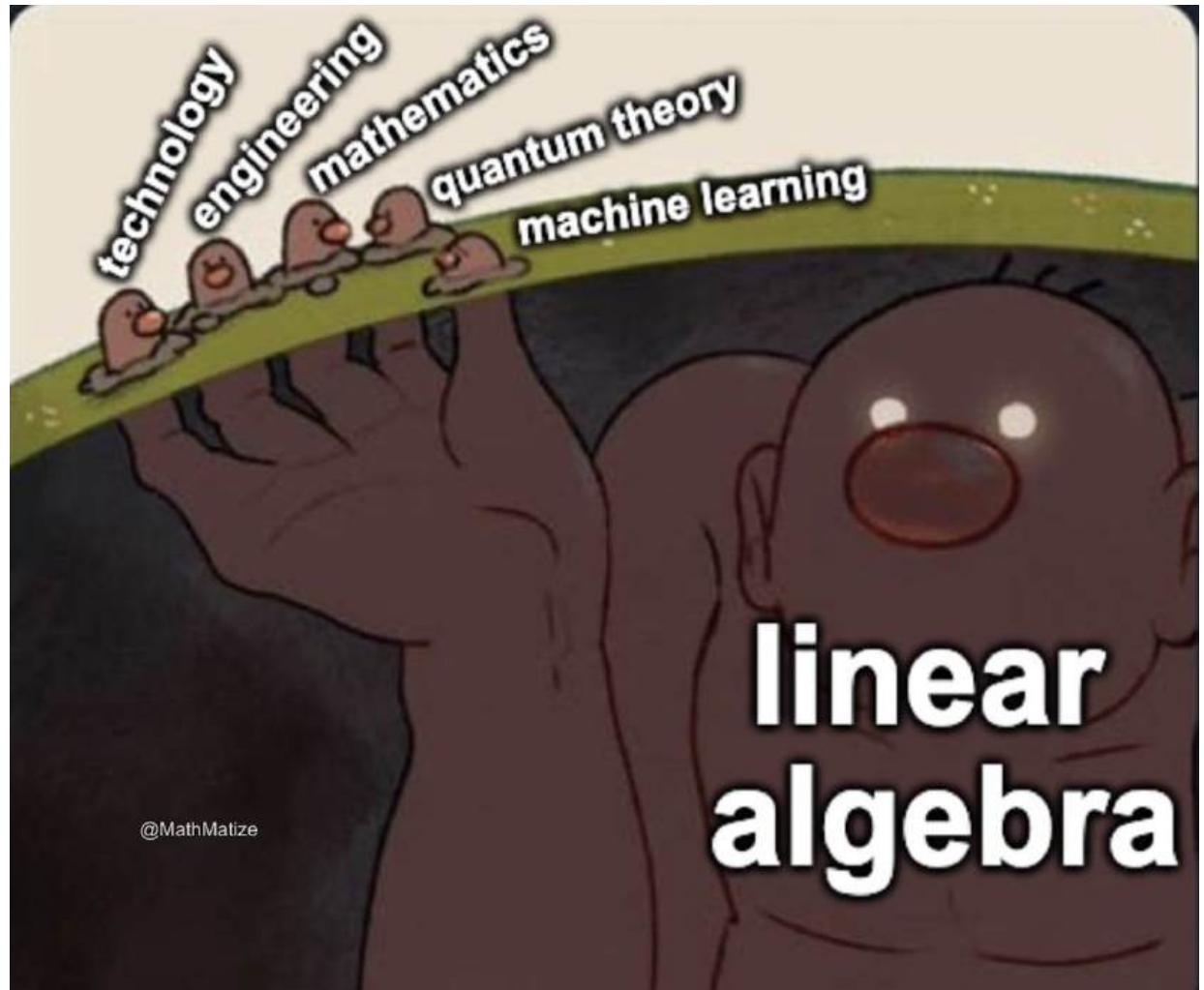
$$(0, 1, 1)^t \cdot (0, 2, 2)^t = 4 > 3$$

hence they lie on opposite sites of the plane.

# Summary

How Computer Graphics continues:

- Setup assignment (1a) due tomorrow.
- Do the WebLab assignment (1b).
- Do the Programming assignment (1c).
- Ask questions you have at this point.



Don't worry – it will all be on Brightspace.

# CSE2215 - Computer Graphics

## Images and Algebra Taking care of your image

Elmar Eisemann

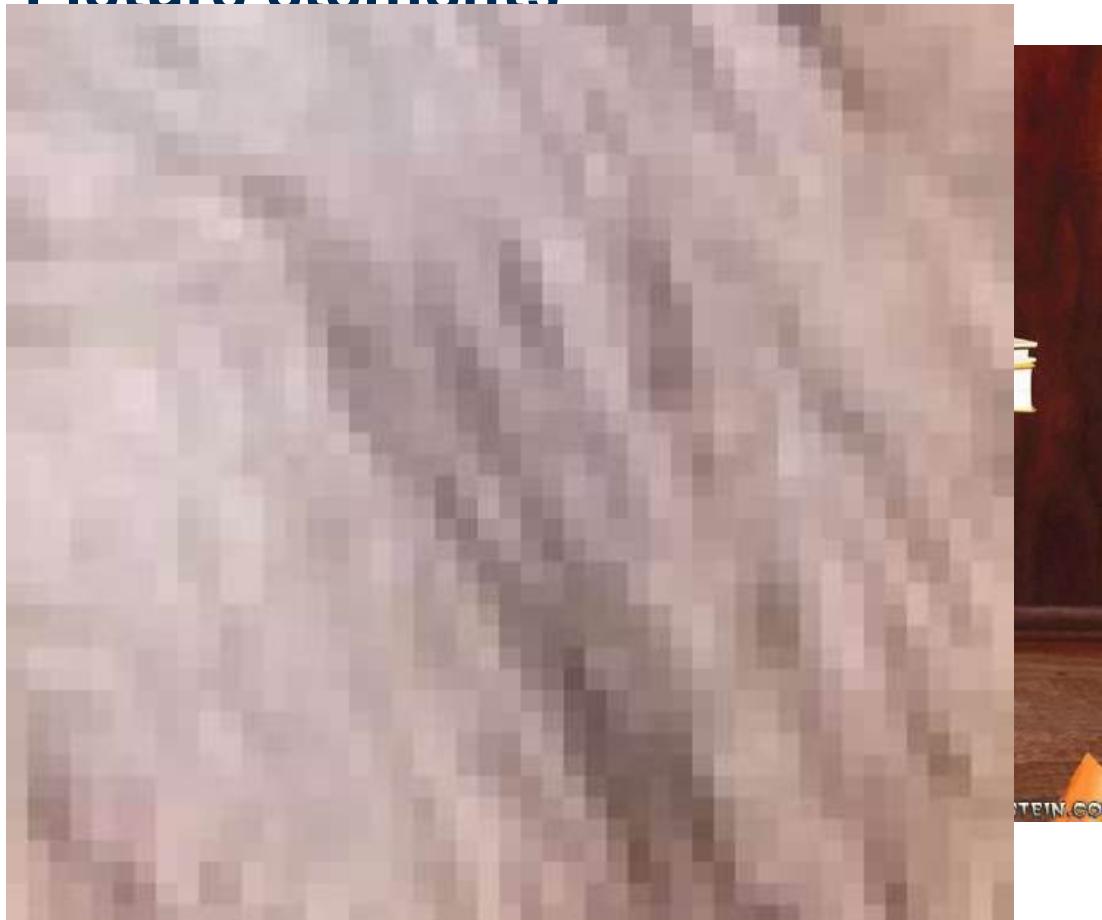
Delft University of Technology



# Making images with a Computer

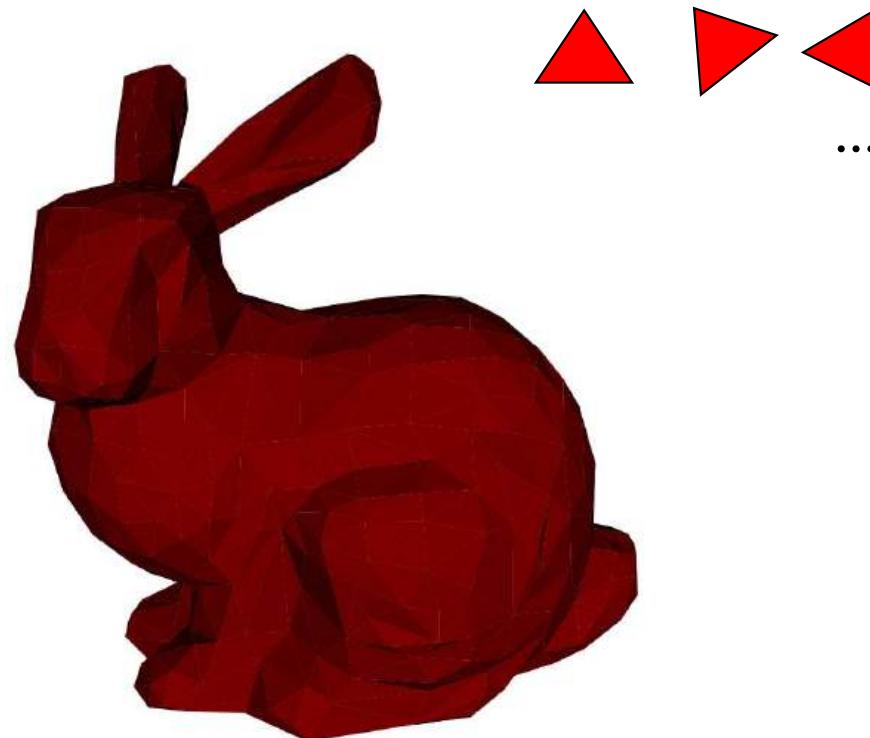


## Pixels – Picture elements



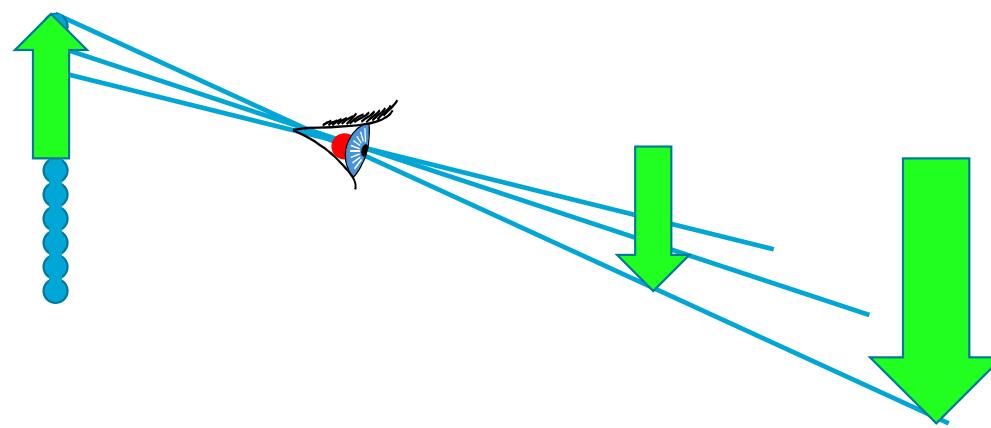
## Simplified Graphics Pipeline

- Models are typically lists of triangles



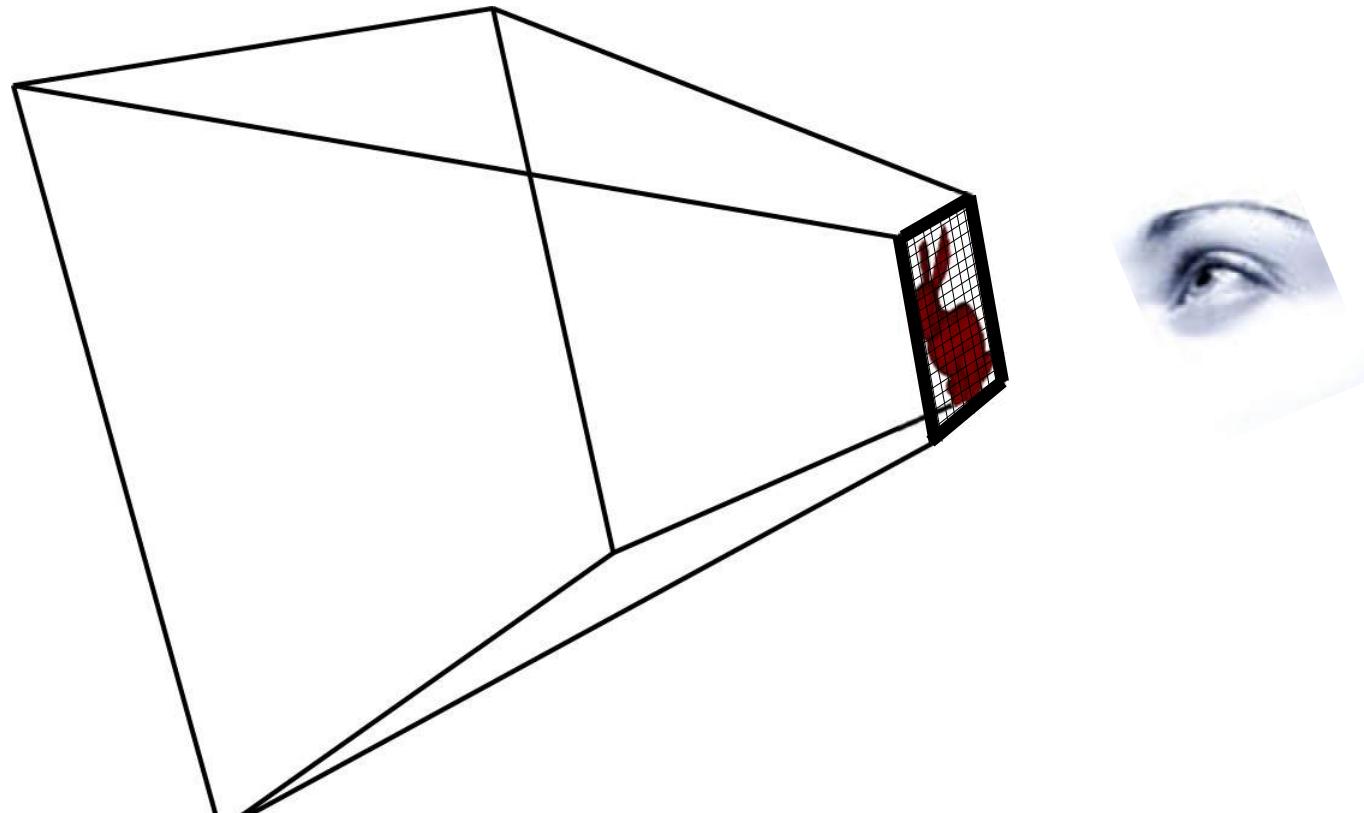
## Virtual Camera

- Camera Plane in front of the eye



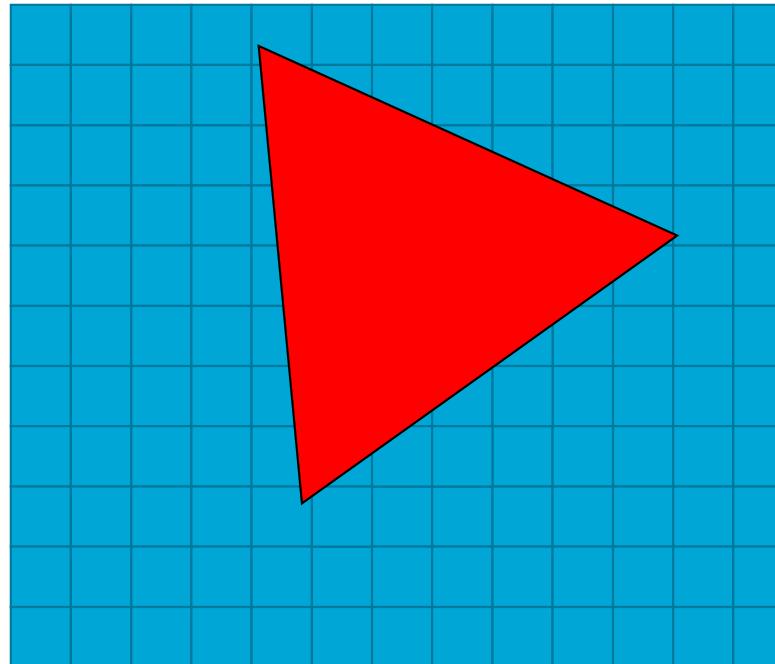
# Simplified Graphics Pipeline

- **Projection:** Transform coordinates to screen



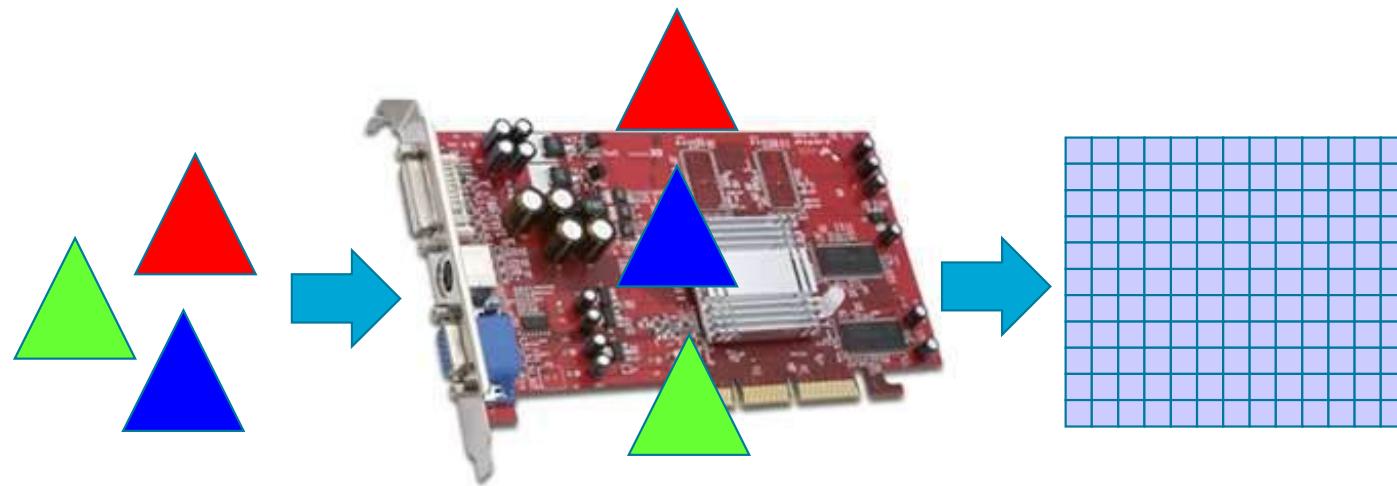
# Simplified Graphics Pipeline

- Rasterization: Fill screen pixels



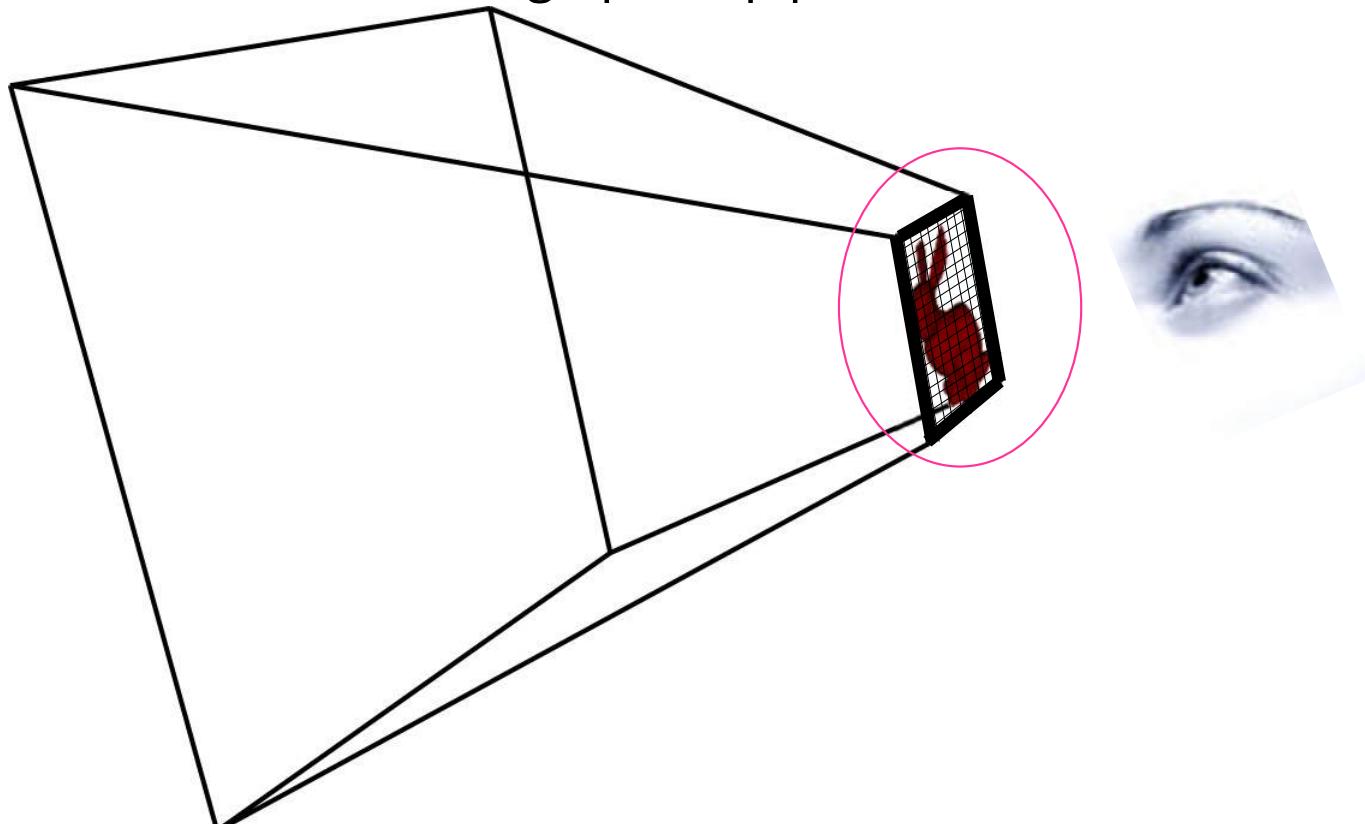
## Simplified Graphics Pipeline

- Highly parallelizable → GPUs



# Today

- We will work towards the graphics pipeline...



## Relevant Study Goals for Today

- S4 Apply mathematical modeling and theory of geometric computations and transformations, object representations, simulation, and encoding.
  - We learn about the mathematical basics for complex object representations
- S5 Implement algorithms and data structures using the C++ programming language
  - We see some rudimentary code to implement basic image operations
- S6 Apply the knowledge obtained in this course to problems of other fields
  - We will see several application examples beyond computer graphics.

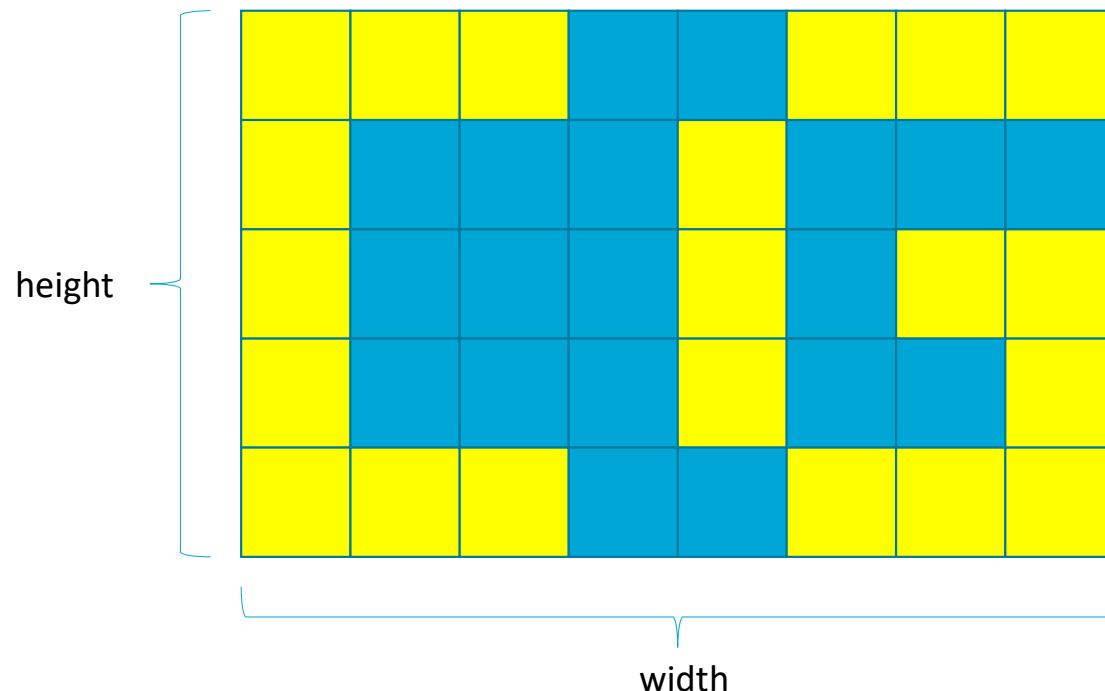
# Images

# Images

- What is an image?
- How to represent it in memory?
- How to access individual pixels?
- How can we process images?
- How are images stored?

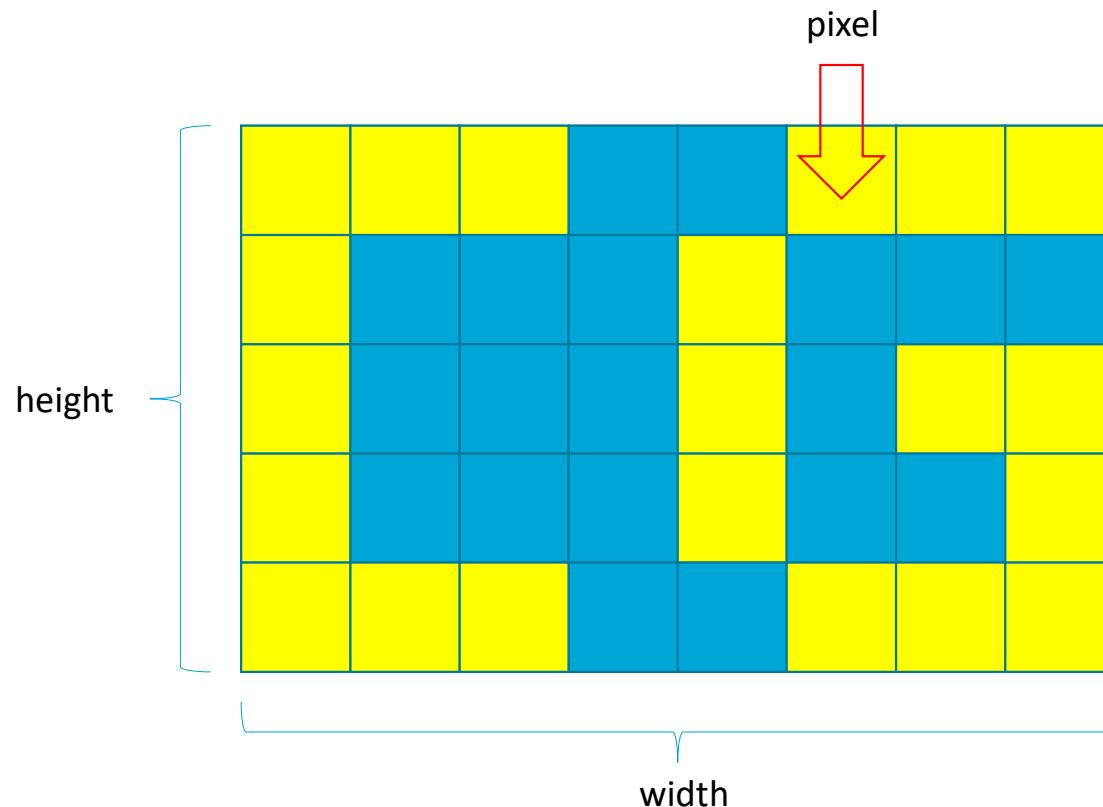
# Representation

- Image

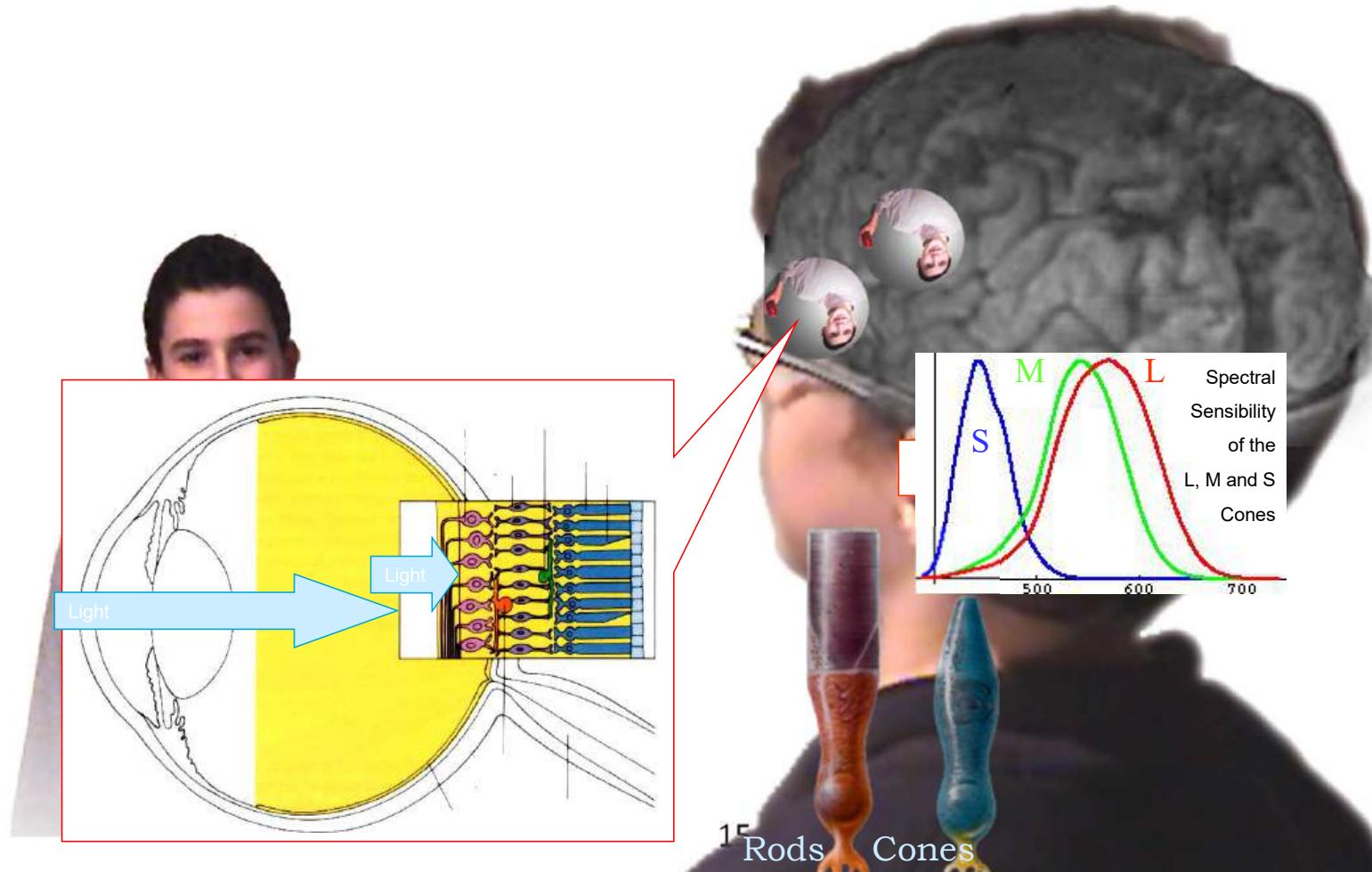


# Representation

- Image

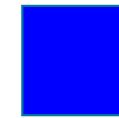
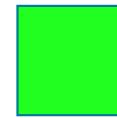
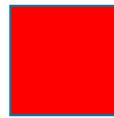


# Eye Biology



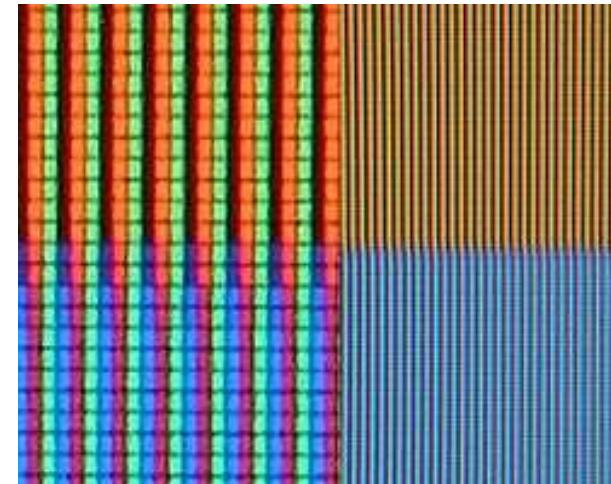
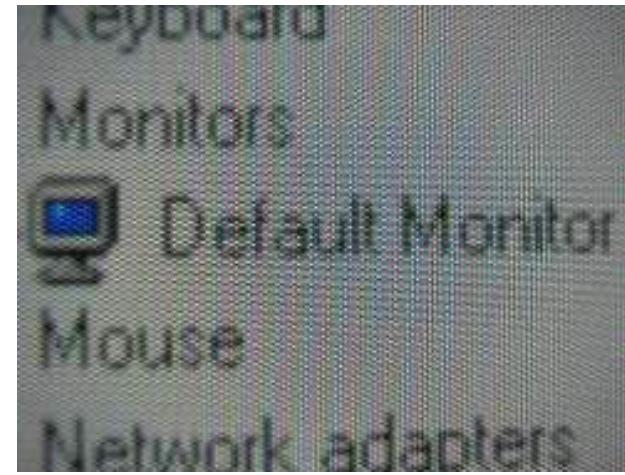
## Representation

- Pixel:
- Contains 3 components Red, Green, Blue



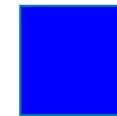
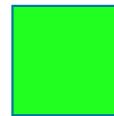
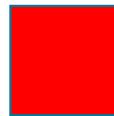
## Representation

- Standard Screens work with RGB
- A value of 1 in a color channel activates the channel as strongly as possible
- For example (1,0,0) is the strongest red



## Representation

- Pixel:
- Contains 3 components Red, Green, Blue



- Typically, continuous values in [0,1].
- E.g., (1,0.5,0)



Grab a paint program  
and test some values for yourself!  
Often values between 0,255 instead of 0,1

# Images

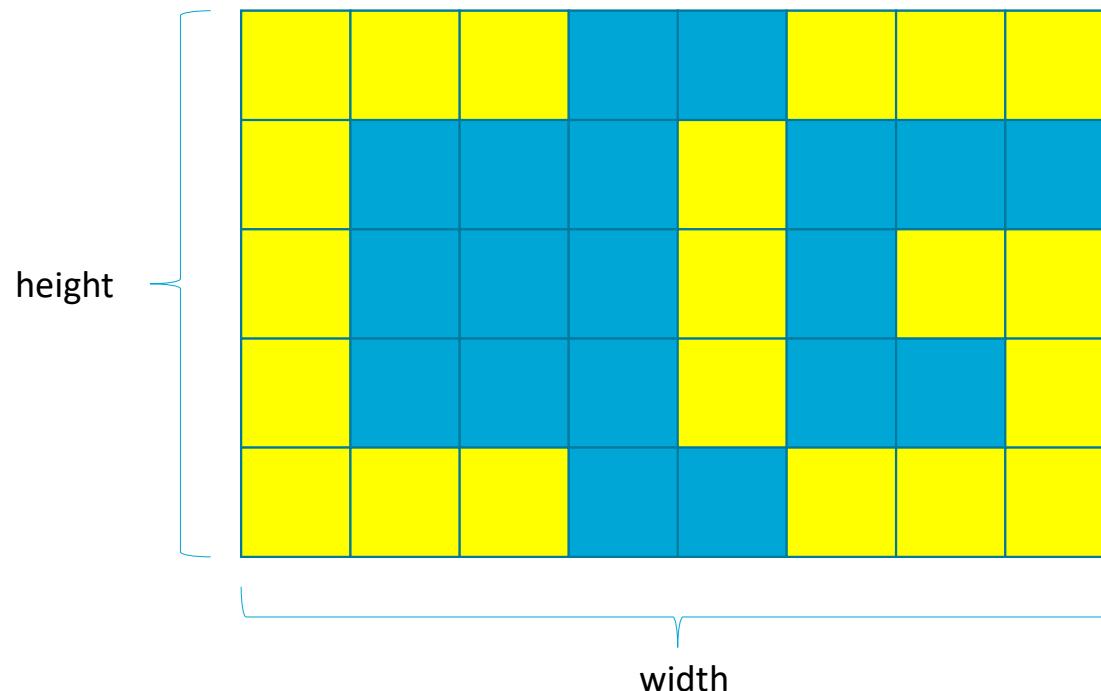
- What is an image?
- How to represent it in memory?
- How to access individual pixels?
- How can we process images?
- How are images stored?

## Representation

- Color values are discretized/quantized:
  - most common for image storage and display output:  
8 bit per color channel (256 values)
  - conversion of float  $v$  in  $[0,1]$  to 8bit:  $v * 255$ ;

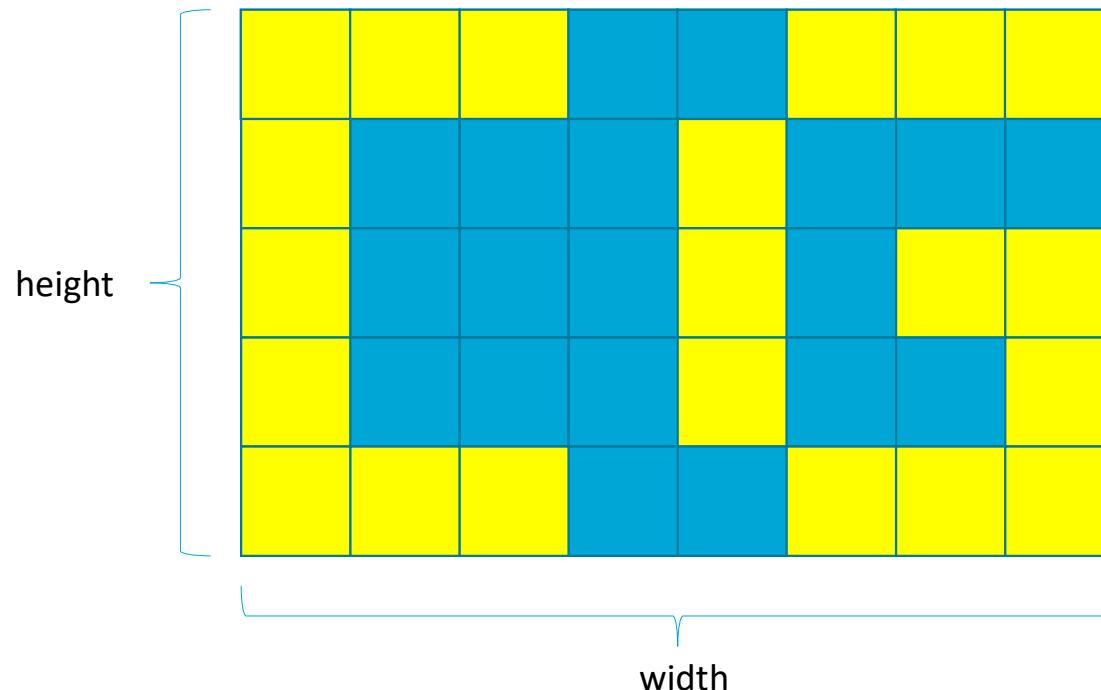
# Representation

- Image representation in memory?



# Representation

- Image representation in memory?

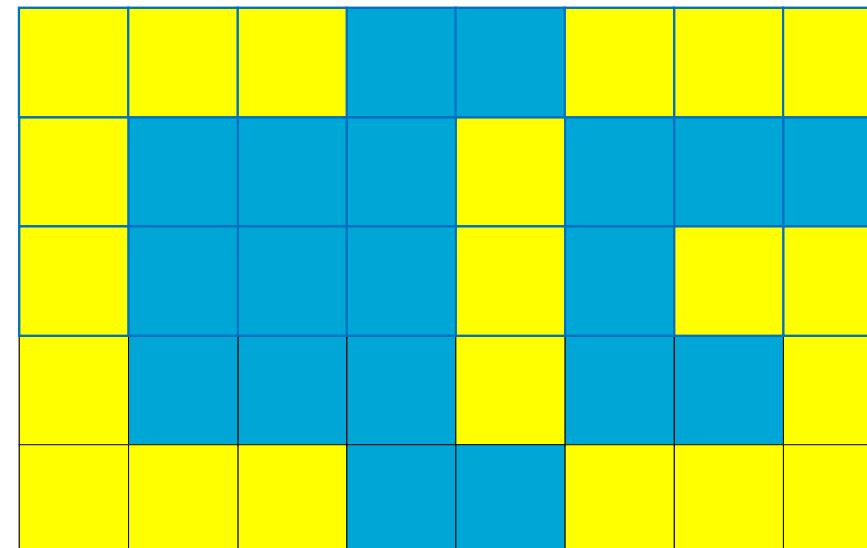


# Representation

- Image representation in memory?

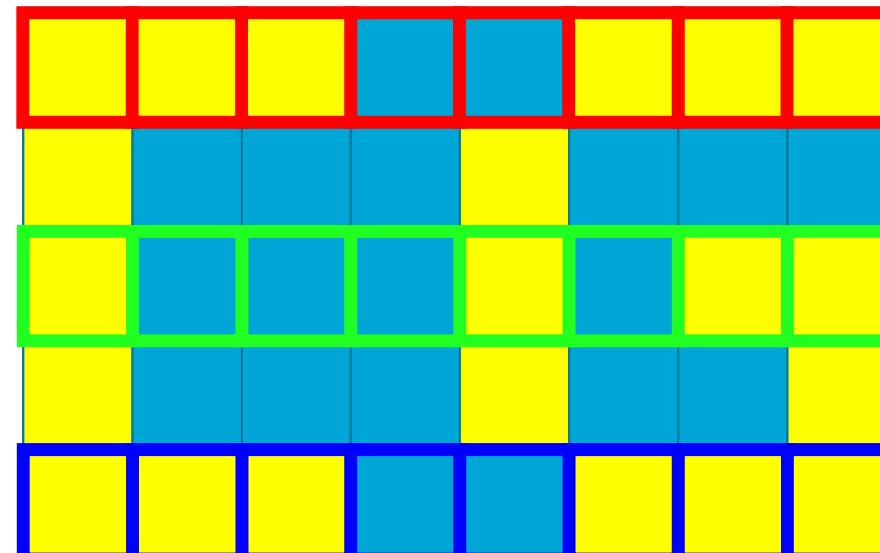
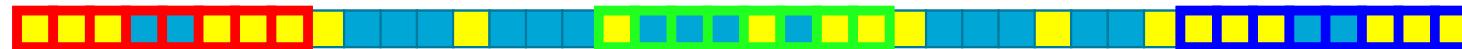


3 values per pixel  
(Red, Green, Blue)



# Representation

- Image representation in memory?



# Images

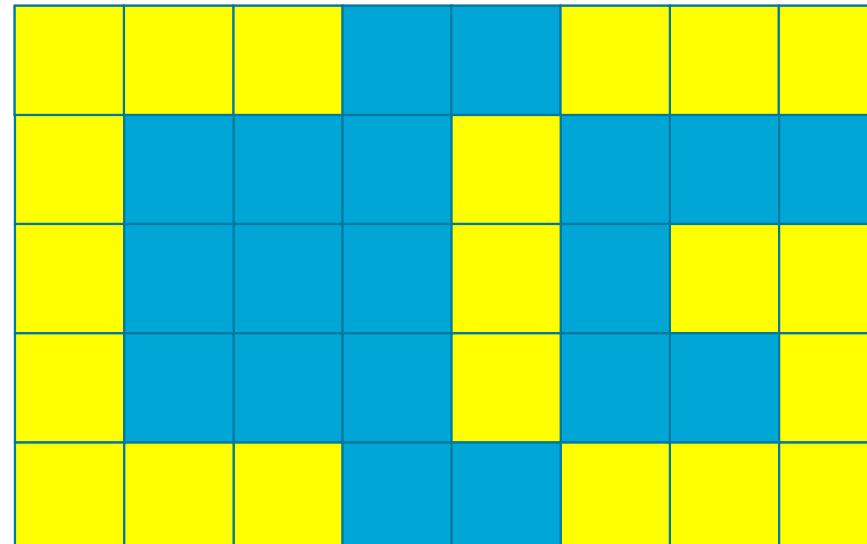
- What is an image?
- How to represent it in memory?
- How to access individual pixels?
- How can we process images?
- How are images stored?

## Representation

- How to access pixel  $(i,j)$ ?
- Find index to access corresponding memory
- Solution:  $3*(j*width+i)$     3 for the 3 values per pixel (RGB)
  - Need to know the width!  
Typically stored in an image file
  - What about height?  
Usually as well but could be derived from data.

## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?



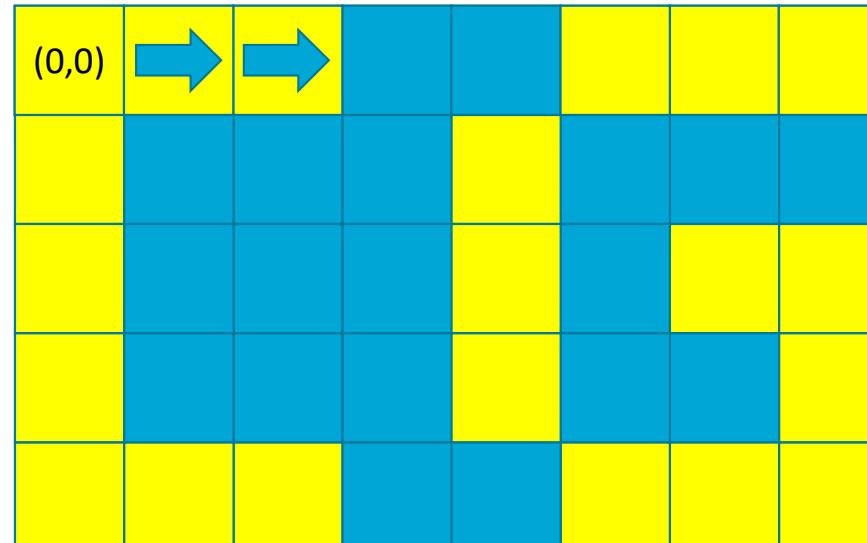
## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?

(0,0)							

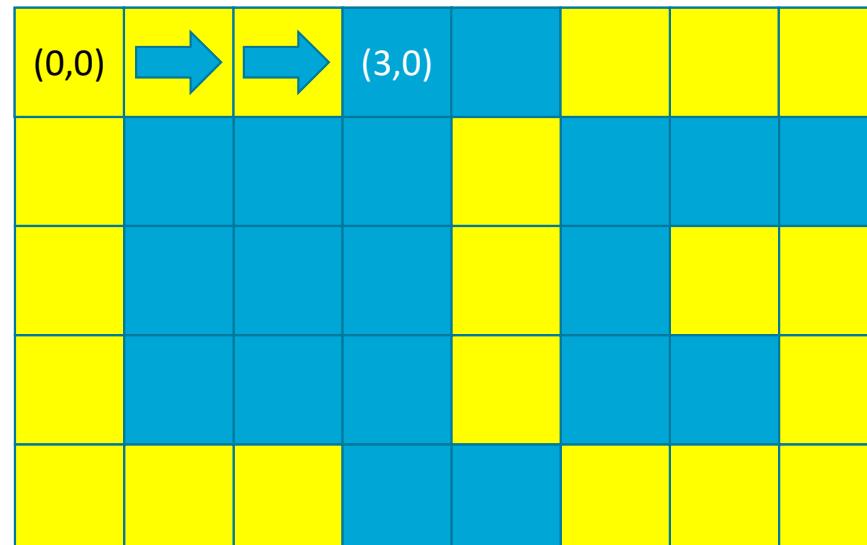
## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?



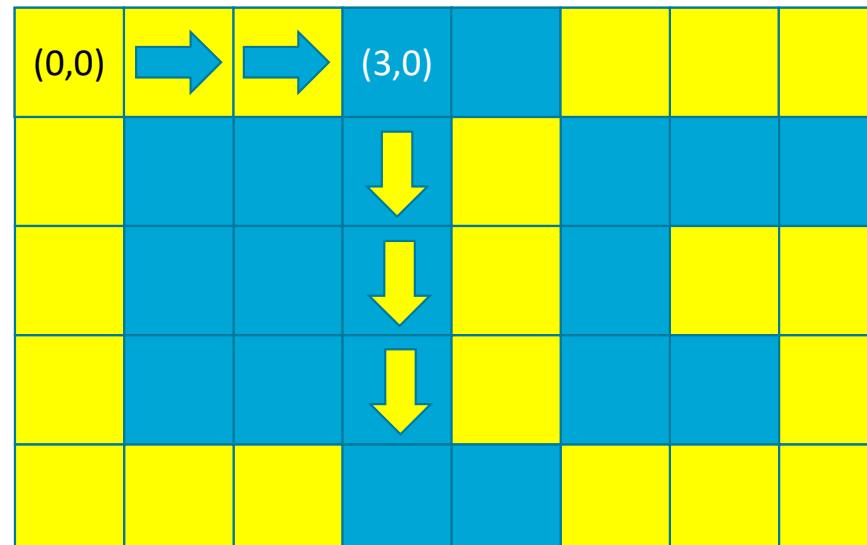
## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?



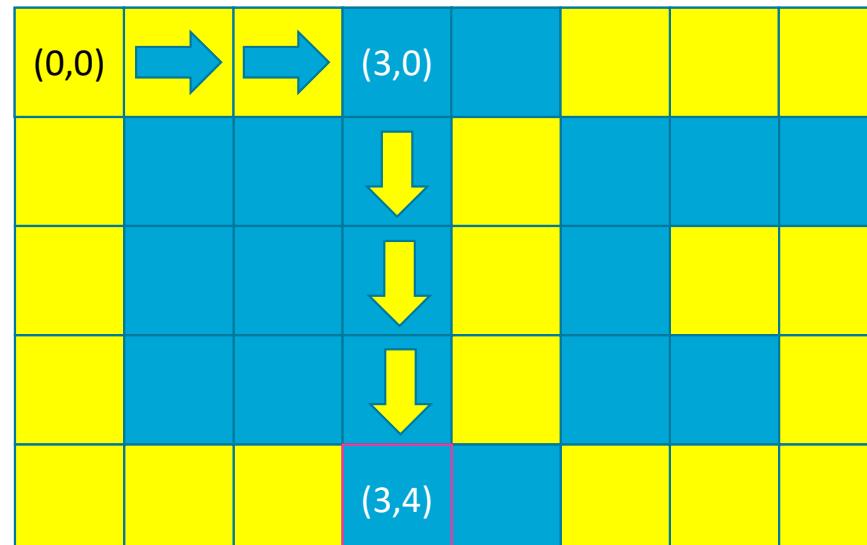
## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?



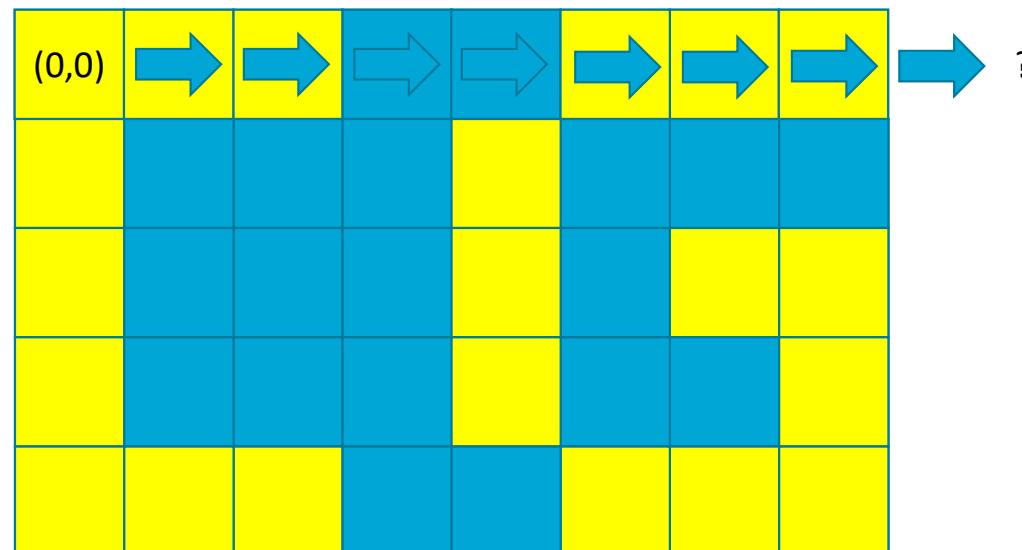
## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?



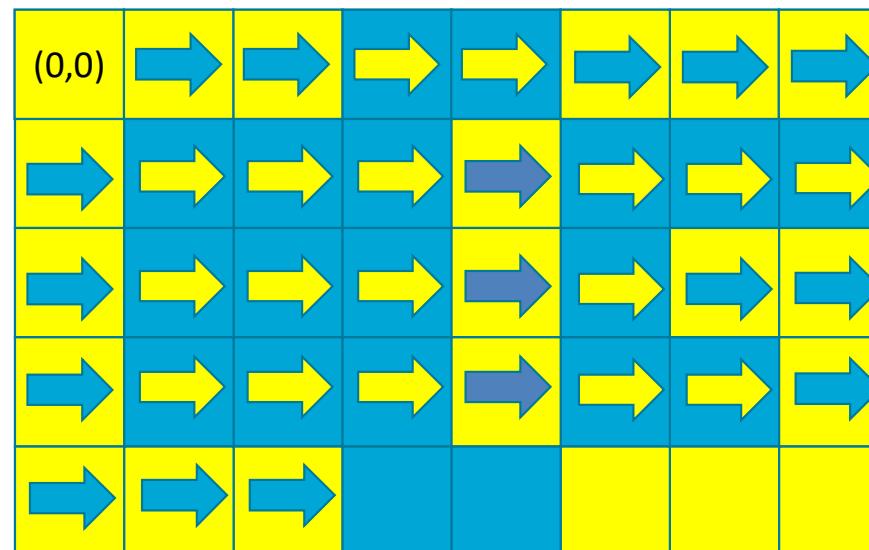
## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?  $\cancel{?} = j * \text{width} + i = 4 * 8 + 3 = 35$



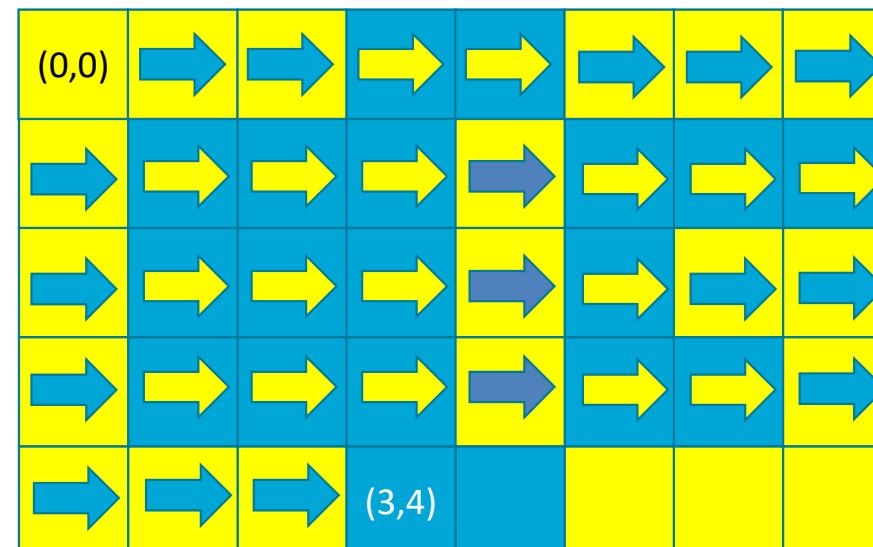
## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?  $j * \text{width} + i$   $= 4 * 8 + 3 = 35$



## Representation

- Example:
- One channel (grayscale) image (resolution 8x5)
- What index accesses pixel (3,4)?  $(j * \text{width} + i) = 4 * 8 + 3 = 35$

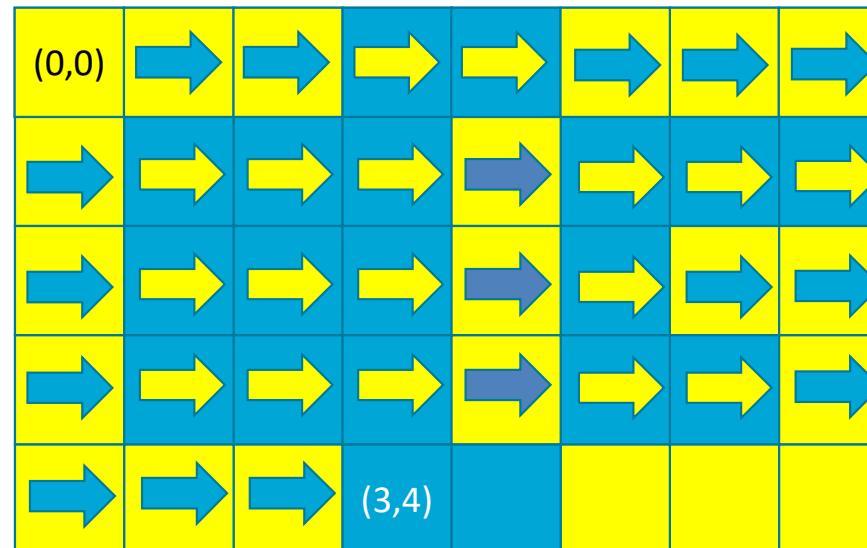


When you start counting  
from 0 instead of 1



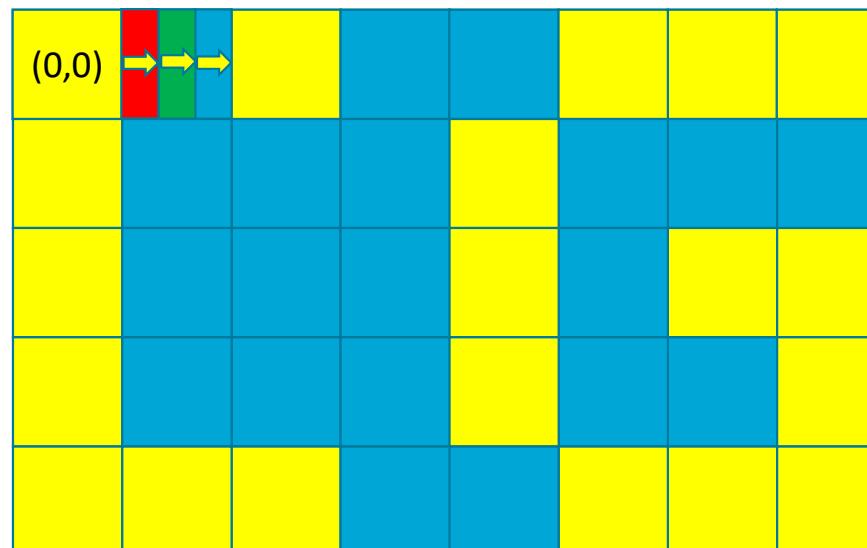
## Representation

- Example:
- **Three-channel (RGB) image (resolution 8x5)**
- What index accesses pixel (3,4)?



## Representation

- Example:
- **Three-channel (RGB) image** (resolution 8x5)
- What index accesses pixel (3,4)?  $3*(j*width+i) = 3*(4*8+3)$   
 $=105$



## Representation

- How to find a pixel  $(i,j)$  given index  $l$  for a grayscale image?
- Solution:

```
i= l%width      //modulo = remainder of division  
j= l / width
```

**(ATTENTION:** Integer division! E.g.,  $5/3=1$ )

## Representation

- Let's test it:
- Grayscale Image (resolution 3x3), index  $l=8\dots$
- Solution:

$$i = l \% \text{width}$$

$$i = 8 \% 3 = 2 \quad // \ 8 = 2 * 3 + 2 = 8$$

$$j = l / \text{width}$$

$$j = 8 / 3 = 2 \quad // , \text{ thus pixel } (2,2)$$

**(ATTENTION:** Integer division! E.g.,  $5/3=1$ )

Why does it work?

$l/\text{width}$  are the full rows that fit into  $l$

$l \% \text{width}$  is the remainder, thus the offset in the row.

# Simple Image Class

```
class Image
```

```
{public:
```

```
    std::vector<float> data;
```

```
    int w, h;
```

```
};
```

**Disclaimer:** the following code is not a coding paradigm, e.g., you should use **const** qualifiers and private variables. The goal here is to reduce the amount of text...

# Simple Image Class

```
class Image

{public:

    std::vector<float> data;

    int w, h;

    Image(int wl, int hl){w=wl; h=hl; data.resize(3*w*h);}

    Image(const Image & i){w=i.w; h=i.h, data=i.data;};

};
```

# Simple Image Class

```
class Image

{public:

    std::vector<float> data;

    int w, h;

    Image(int wl, int hl){w=wl; h=hl; data.resize(3*w*h);}

    Image(const Image & i){w=i.w; h=i.h, data=i.data;};

    float & pixel(int i, int j, int col){

        return data[3*(i+j*w)+col]; //Problem?

    };

};
```

# Simple Image Class

```
class Image

{public:

    std::vector<float> data; float border=0;

    int w, h;

    Image(int wl, int hl){w=wl; h=hl; data.resize(3*w*h);}

    Image(const Image & i){w=i.w; h=i.h, data=i.data;};

    float & pixel(int i, int j, int col){

        if (i<0 || i>=w || j<0 || j>=h || col<0 || col>2)

            return border;

        else return data[3*(i+j*w)+col];

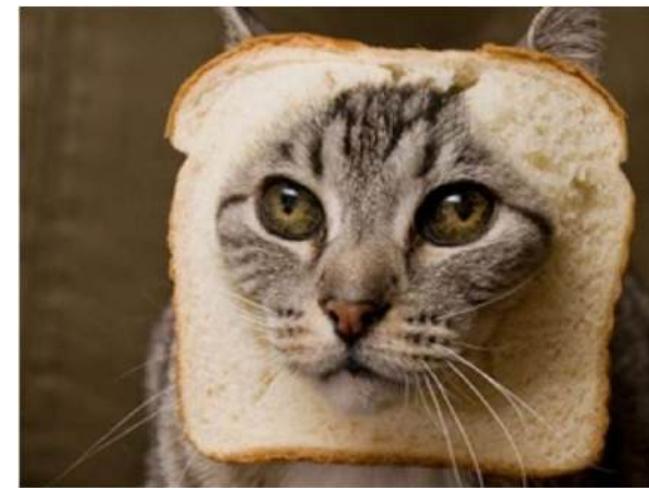
    };

};
```

# Images

- What is an image?
- How to represent it in memory?
- How to access individual pixels?
- How can we process images?
- How are images stored?

# Filter

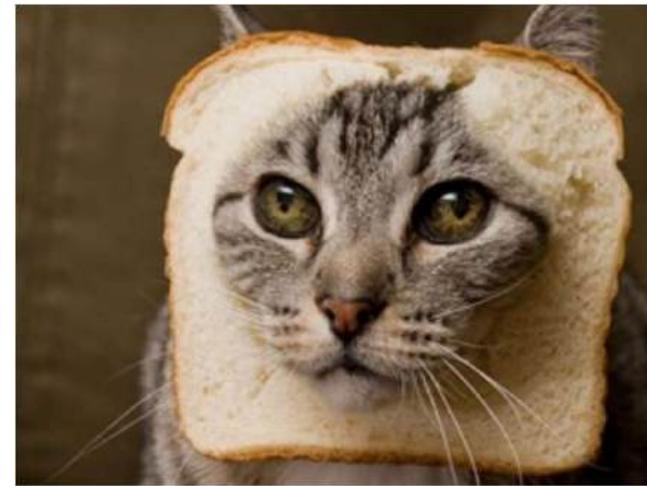
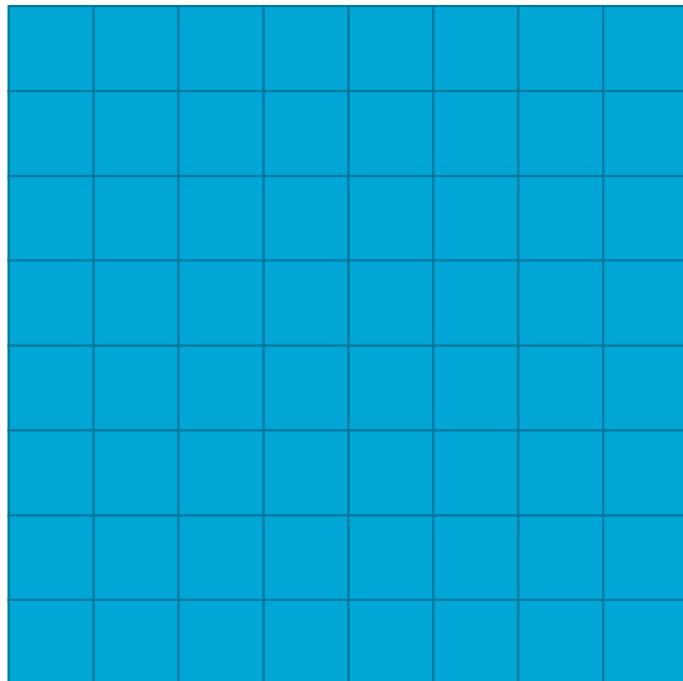


# Filter



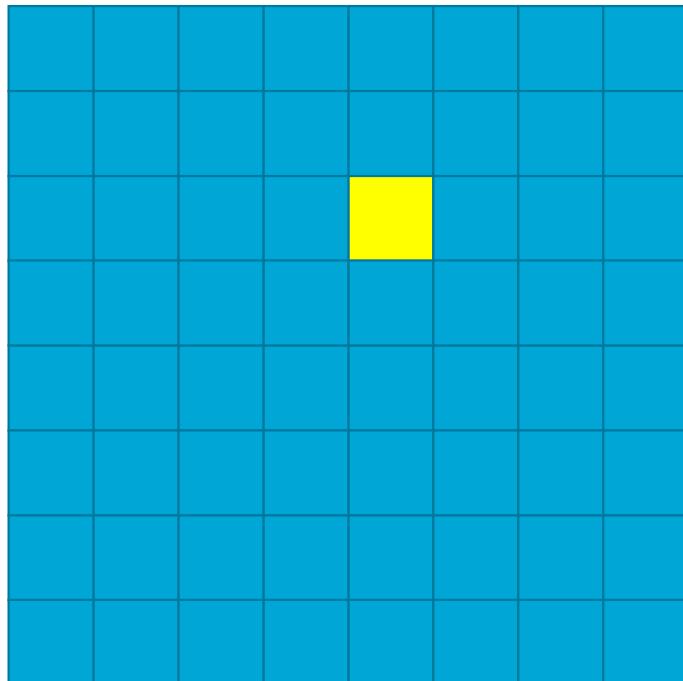
# Filter

- Example Box Filter 3x3



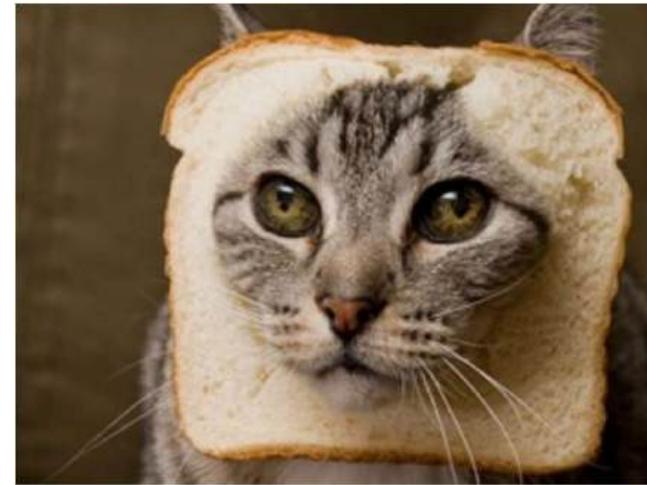
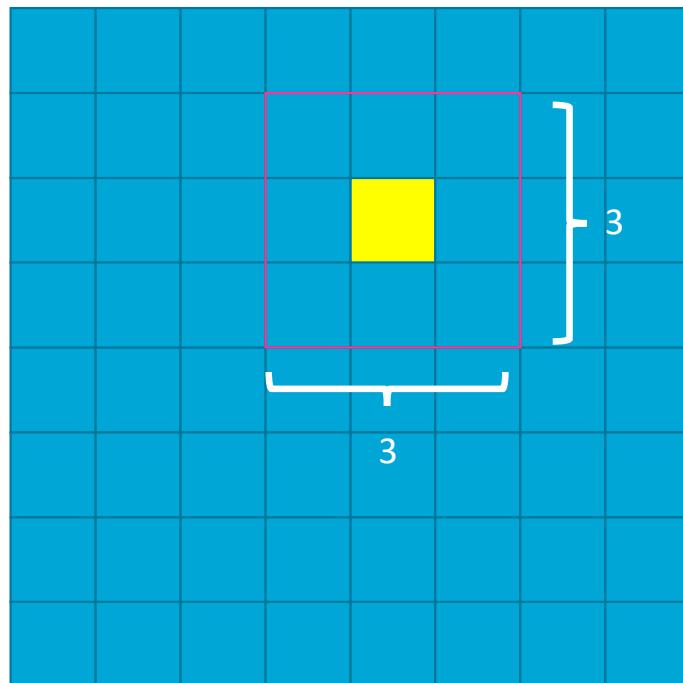
# Filter

- Example Box Filter 3x3



# Filter

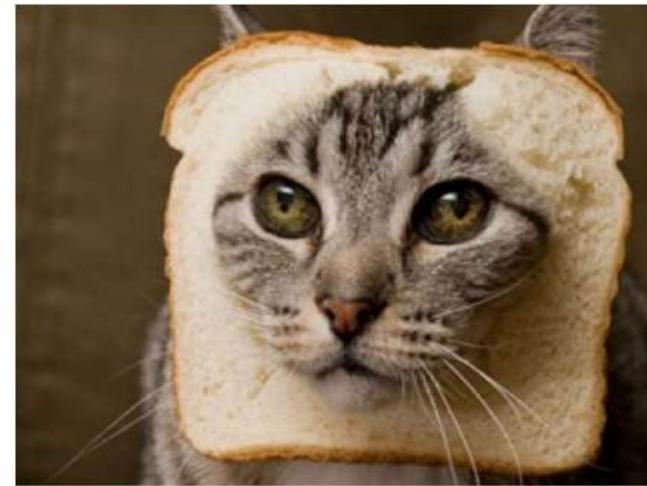
- Example Box Filter 3x3



# Filter

- Example Box Filter 3x3

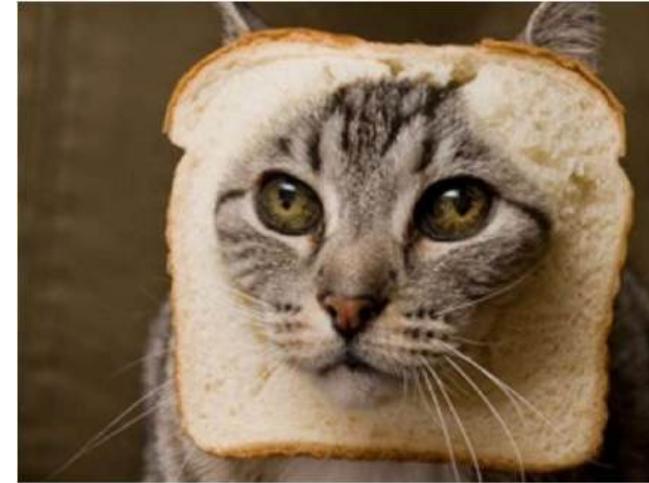
0.5	1	0
1	0	0.5
1	1	1



## Filter

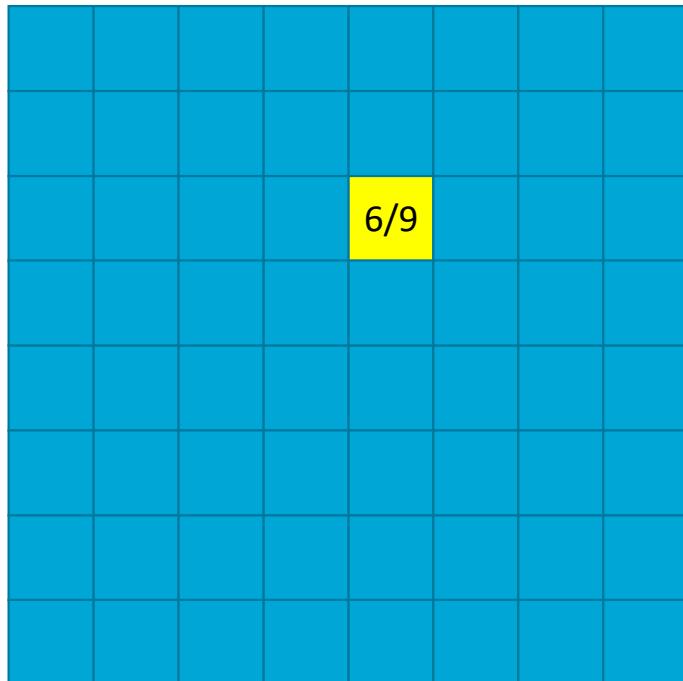
- Example Box Filter 3x3

0.5	1	0
1	0	0.5
1	1	1



# Filter

- Example Box Filter 3x3



## Box Filter Code 1/2

```
Image FilterImage(Image & source, int filterSize)
{
    // we create a result image
    Image result(source);

    //and process every channel of every pixel independently
    for (int i=0;i<source.w;++i)
        for (int j=0;j<source.h;++j) // for each pixel
            for (int col=0;col<3;++col) // for each color channel
                result.pixel(i,j,col)=boxFilter(source, i, j, col, filterSize);

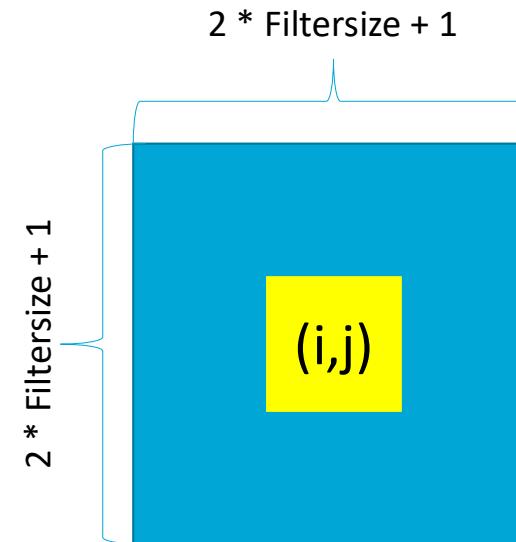
    return result;
};
```

## Box Filter Code 2/2

```

float boxFilter (Image & source, int i, int j, int col, int filterSize)
{
    filterSize=max(1,filterSize);
    float sum=0;
    //Average pixels in the box-filter region
    for (int x=-filterSize;x<filterSize+1;++x)
        for (int y=-filterSize;y<filterSize+1;++y)
            sum+=source.pixel(i+x,j+y,col);
    sum/=(2*filterSize+1)*(2*filterSize+1);
    return sum;
};

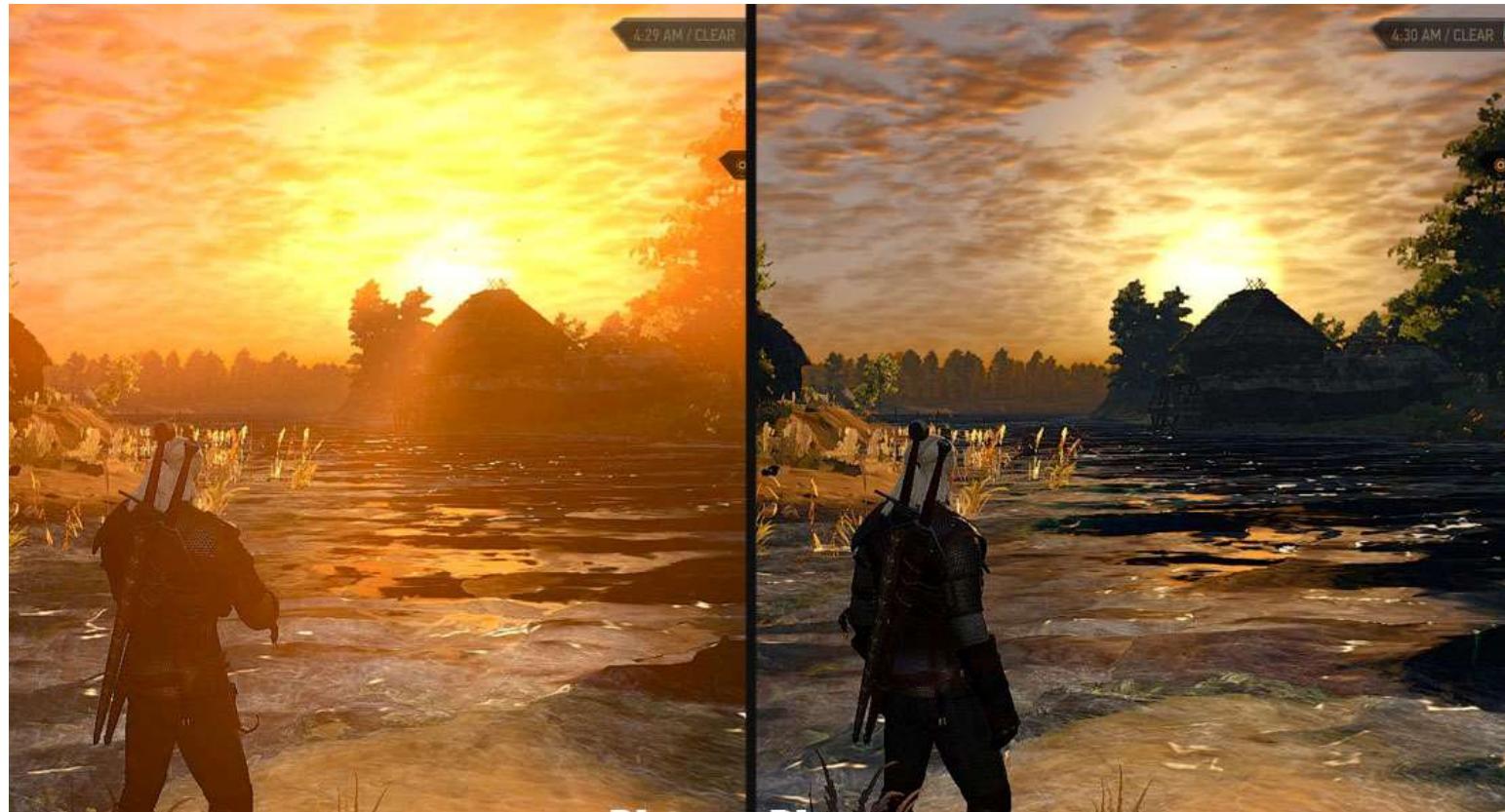
```



Note: 3x3 Box Filter  
means Filtersize=1

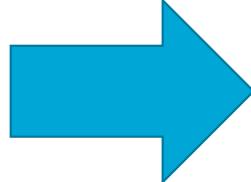
## Example Application

- Bloom effects (here Witcher 3)



## Example Application

- Bloom effects



[https://nl.freepik.com/premium-vector/dark-fire-ranger-geometry-style\\_3799192.htm](https://nl.freepik.com/premium-vector/dark-fire-ranger-geometry-style_3799192.htm)

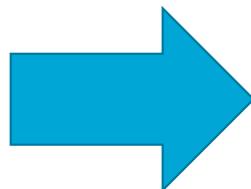
## Example Application

- Bloom effects



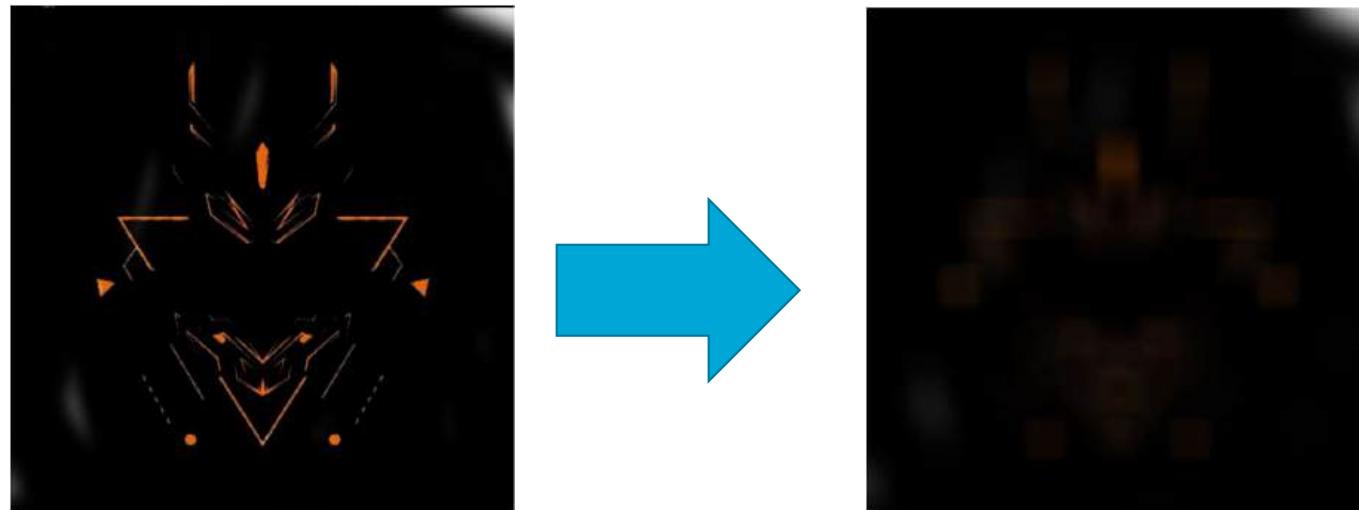
## Example Application

- 1) Threshold – only keep large values (e.g.,  $>0.9$ )



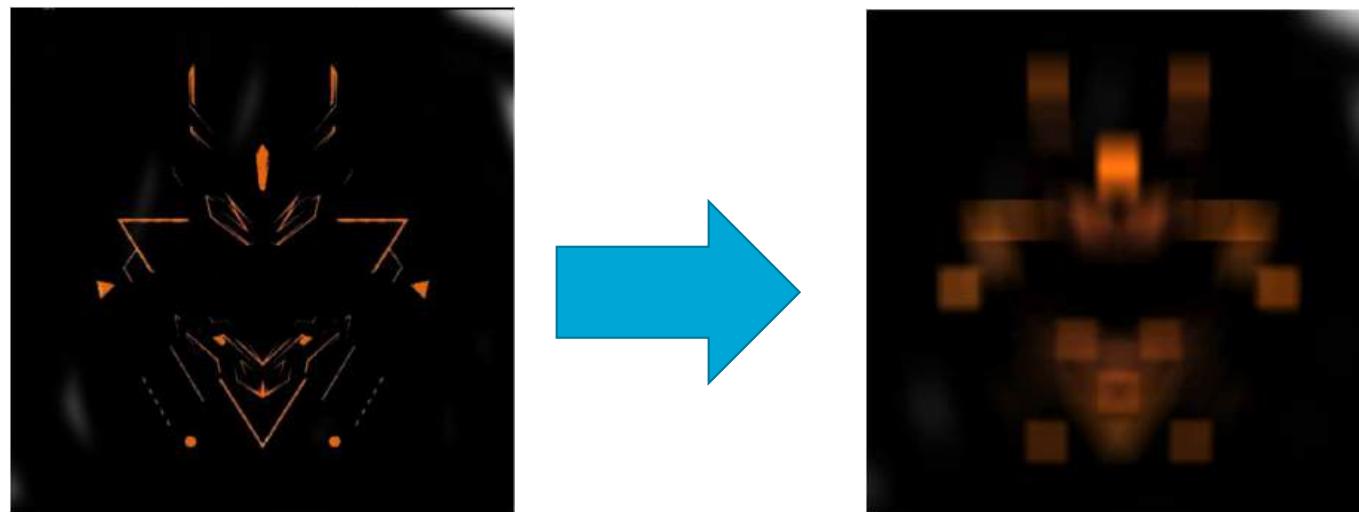
## Example Application

- 1) Threshold – only keep large values (e.g.,  $>0.9$ )
- 2) Box filter on thresholded image



## Example Application

- 1) Threshold – only keep large values (e.g.,  $>0.9$ )
- 2) Box filter on thresholded image and scale



## Example Application

- 1) Threshold – only keep large values (e.g.,  $>0.9$ )
- 2) Box filter on thresholded image and scale
- 3) Add to the original



## General Filter Code

```
float filteredPixel(Image & source, Image & filter
                    int i, int j, int col)
{
    float sum=0;
    for (int x=0;x<filter.w;++x)
        for (int y=0;y<filter.h;++y)
            sum+=
                filter.pixel(x,y,col)
                * source.pixel(i+x-filter.w/2,j+y-filter.h/2,col); //Place center of the filter at (i,j)
    return sum;
};
```

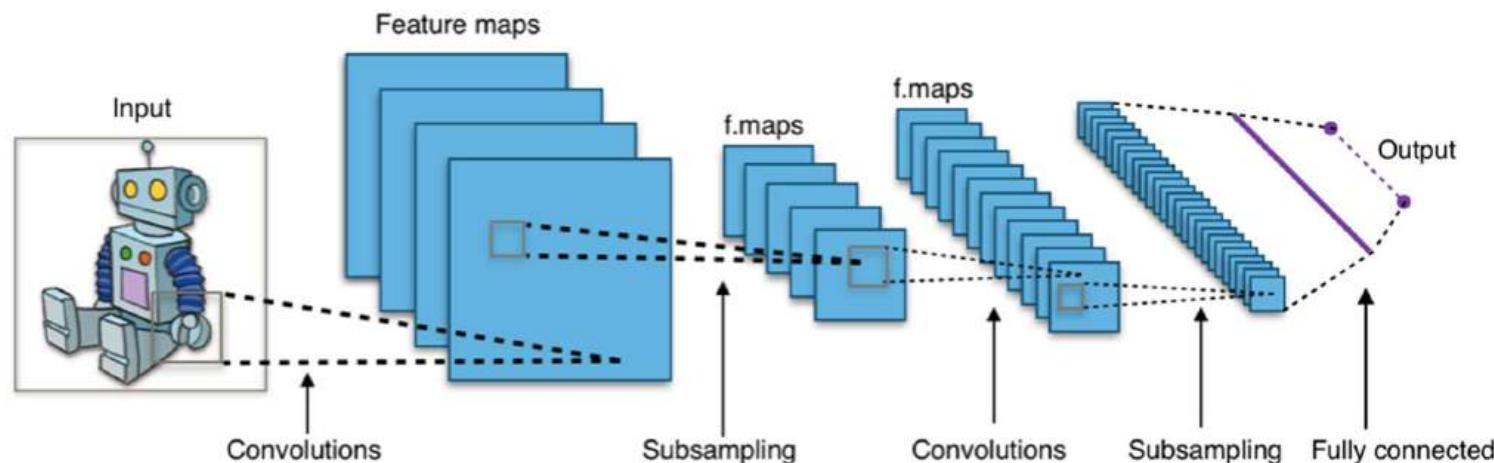
If you want to perform a box filtering,  
the image “filter” should contain  
 $1.0f/(filter.w*filter.h)$  in all pixels.  
**Homework: verify this claim**

## Example Application 2

Machine Learning:

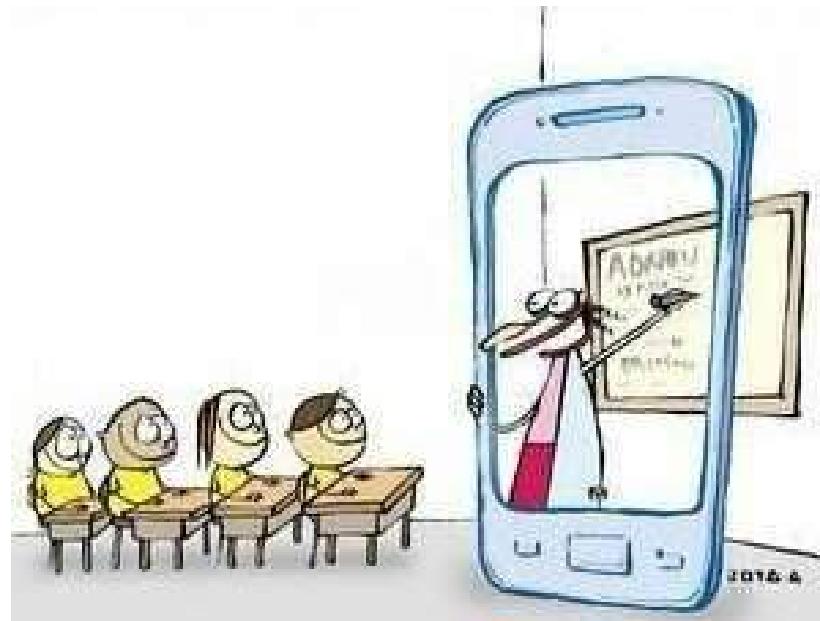
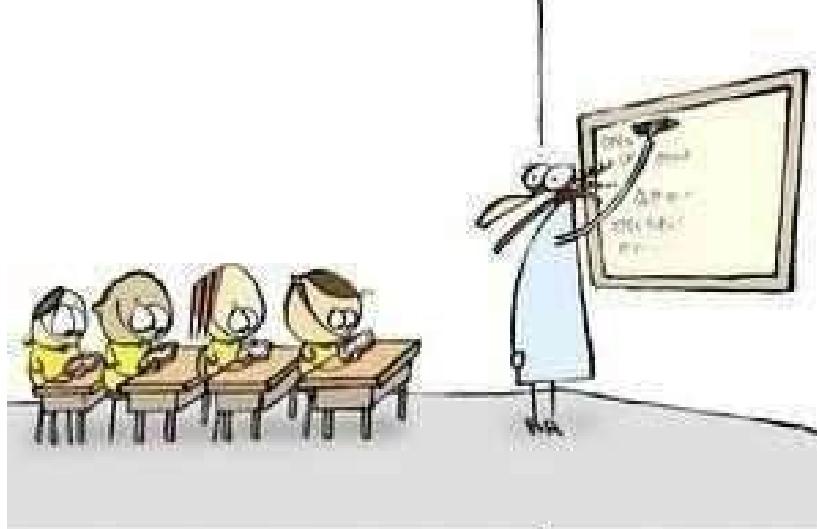
Convolutional Neural Networks Successive filtering and thresholding.

“Filters” are optimized during training.



[https://upload.wikimedia.org/wikipedia/commons/6/63/Typical\\_cnn.png](https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png)

# Questions?



# Images

- What is an image?
- How to represent it in memory?
- How to access individual pixels?
- How can we process images?
- How are images stored?

## Storing an image

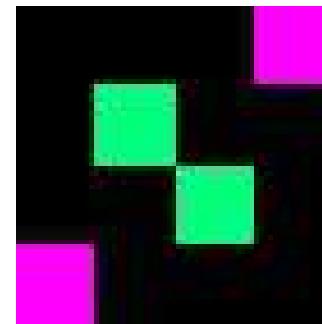
- Several formats for images
  - Some compress data but lose details
  - Others are lossless but require more space
- Many options, we will only cover two:
  - PPM – very simple, big file size, lossless
  - JPEG – complex, small file size, lossy

# Simple Image Format: PPM

Each PPM image consists of a **header** and **image data**

**Example file opened in Text Editor:**

```
P3  
4 4  
15  
0 0 0 0 0 0 0 0 15 0 15  
0 0 0 0 15 7 0 0 0 0 0 0  
0 0 0 0 0 0 0 15 7 0 0 0  
15 0 15 0 0 0 0 0 0 0 0 0
```



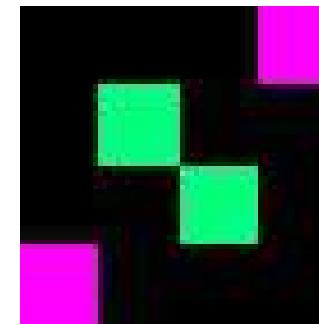
# Simple Image Format: PPM

Each PPM image consists of a **header** and image data. Header contains:

- "magic number", e.g., "P3" for human-readable pixel values in RGB format
- Image width <Whitespace> Image height <Whitespace>
- Maximum color value between [0,65535] <Whitespace> (usually a newline).

**Example file opened in Text Editor:**

```
P3
4 4
15
0 0 0 0 0 0 0 0 15 0 15
0 0 0 0 15 7 0 0 0 0 0 0
0 0 0 0 0 0 0 15 7 0 0 0
15 0 15 0 0 0 0 0 0 0 0 0
```



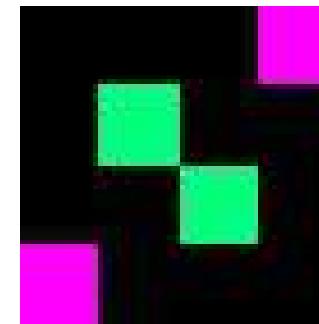
# Simple Image Format: PPM

Each PPM image consists of a **header** and image data. Header contains:

- "magic number", e.g., "P3" for human-readable pixel values in RGB format
- Image width <Whitespace> Image height <Whitespace>
- Maximum color value between [0,65535] <Whitespace> (usually a newline).

**Example file opened in Text Editor:**

```
P3
4 4
15
0 0 0 0 0 0 0 0 15 0 15
0 0 0 0 15 7 0 0 0 0 0 0
0 0 0 0 0 0 0 15 7 0 0 0
15 0 15 0 0 0 0 0 0 0 0 0
```



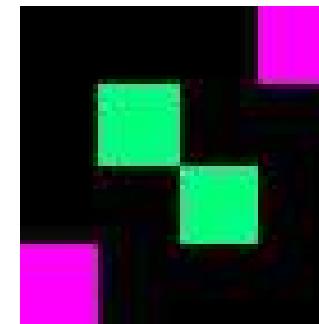
# Simple Image Format: PPM

Each PPM image consists of a **header** and image data. Header contains:

- "magic number", e.g., "P3" for human-readable pixel values in RGB format
- **Image width <Whitespace> Image height <Whitespace>**
- Maximum color value between [0,65535] <Whitespace> (usually a newline).

**Example file opened in Text Editor:**

```
P3
4 4
15
0 0 0 0 0 0 0 0 15 0 15
0 0 0 0 15 7 0 0 0 0 0 0
0 0 0 0 0 0 0 15 7 0 0 0
15 0 15 0 0 0 0 0 0 0 0 0
```



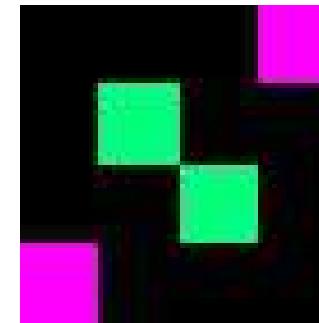
# Simple Image Format: PPM

Each PPM image consists of a **header** and image data. Header contains:

- "magic number", e.g., "P3" for human-readable pixel values in RGB format
- Image width <Whitespace> Image height <Whitespace>
- **Maximum color value between [0,65535] <Whitespace> (usually a newline).**

**Example file opened in Text Editor:**

```
P3
4 4
15
0 0 0 0 0 0 0 0 15 0 15
0 0 0 0 15 7 0 0 0 0 0 0
0 0 0 0 0 0 0 15 7 0 0 0
15 0 15 0 0 0 0 0 0 0 0 0
```



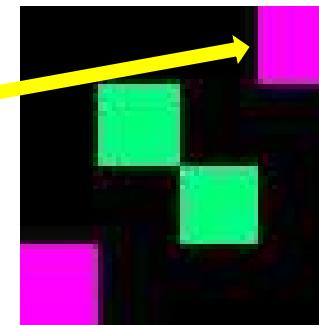
# Simple Image Format: PPM

Each PPM image consists of a header and **image data**. Header contains:

- "magic number", e.g., "P3" for human-readable pixel values in RGB format
- Image width <Whitespace> Image height <Whitespace>
- Maximum color value between [0,65535] <Whitespace> (usually a newline).

**Example file opened in Text Editor:**

```
P3
4 4
15
0 0 0 0 0 0 0 0 15 0 15
0 0 0 0 15 7 0 0 0 0 0 0
0 0 0 0 0 0 0 15 7 0 0 0
15 0 15 0 0 0 0 0 0 0 0 0
```



## Simple Image Format: PPM

- Simple but inefficient in terms of storage
- Filesize directly related to
  - bytes per color channel (if not human-readable, otherwise worse...)
  - resolution

## Storing an image

- Many options, we will only cover two:
  - PPM – very simple, big file size, lossless
  - **JPEG – complex, small file size, lossy**

### DISCLAIMER:

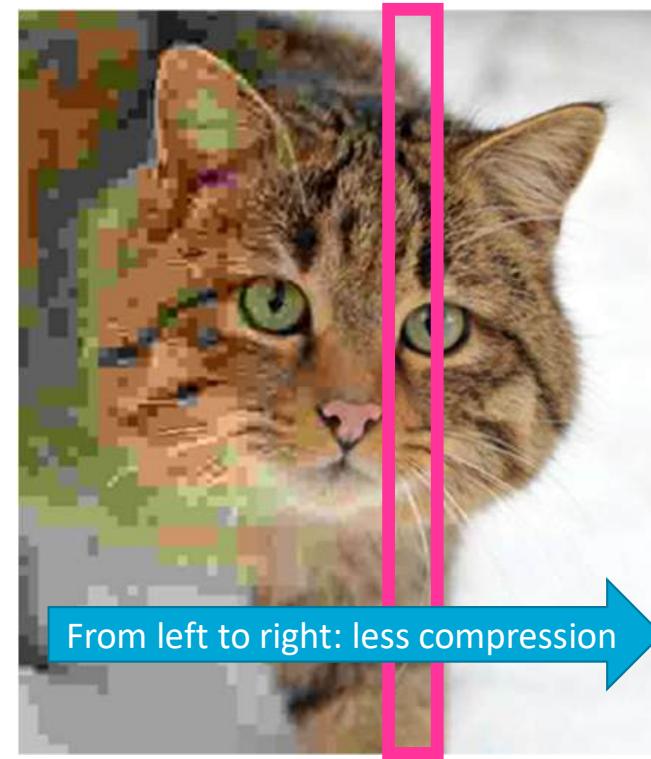
Method is complicated!

You will NOT be asked to reproduce  
or fully understand the details of JPEG compression!

The topic will, for example, come back in **Signal Processing**

## Complex Image Format: JPEG

- Compression by reducing quality (lossy)
- 1:10 compression still has high quality



## Complex Image Format: JPEG

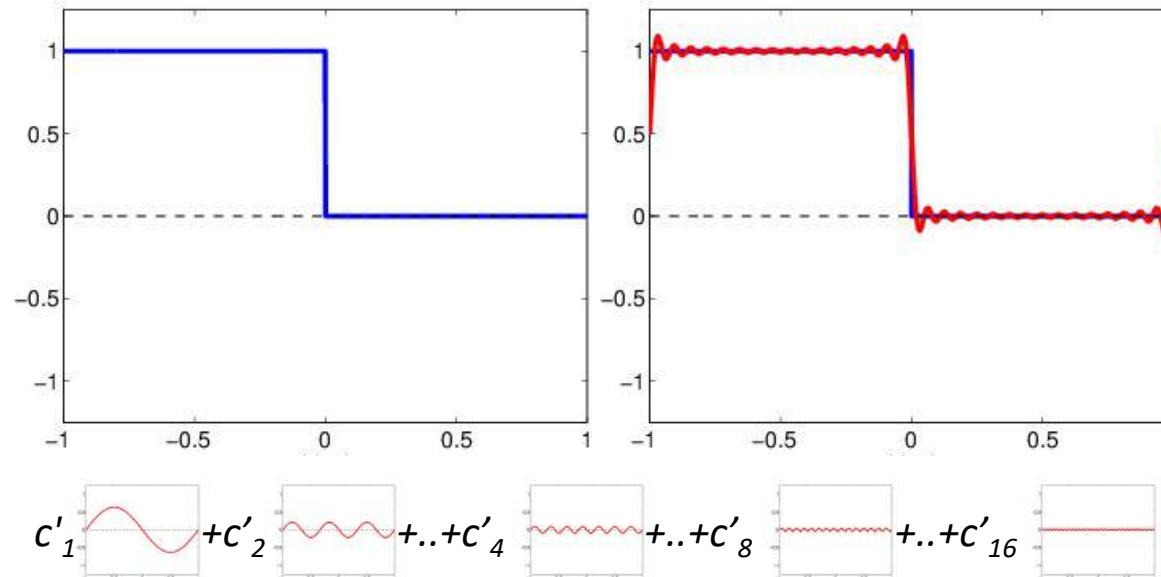
Main idea:

- Use frequency decomposition
- Remove high frequencies first

## Complex Image Format: JPEG

Slightly simplified, we can say that:

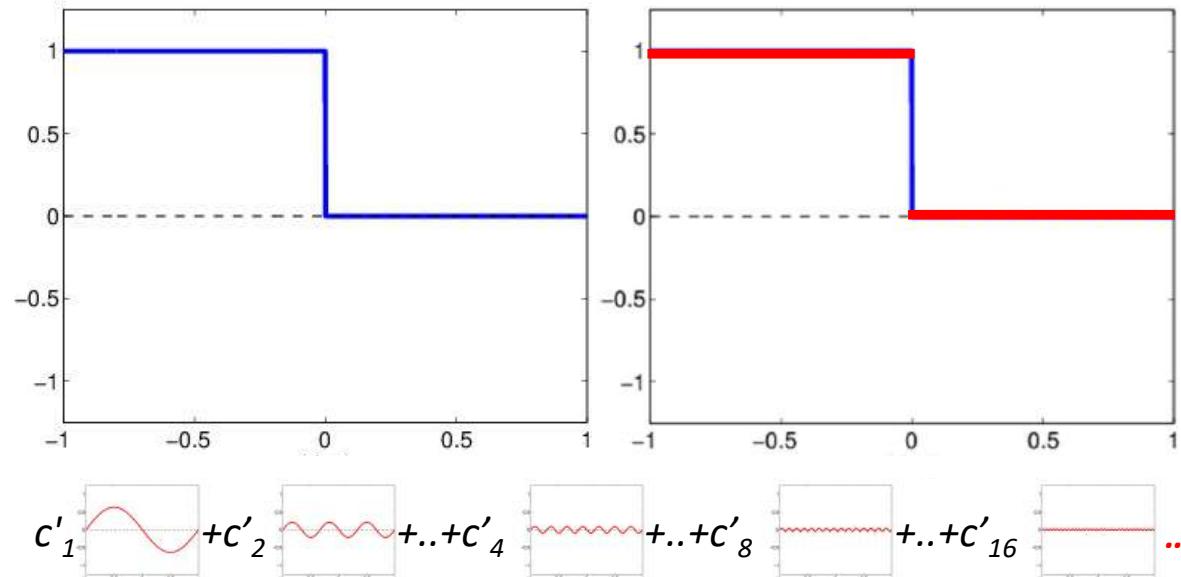
- Any (Riemann) integrable function has a converging Fourier series



## Complex Image Format: JPEG

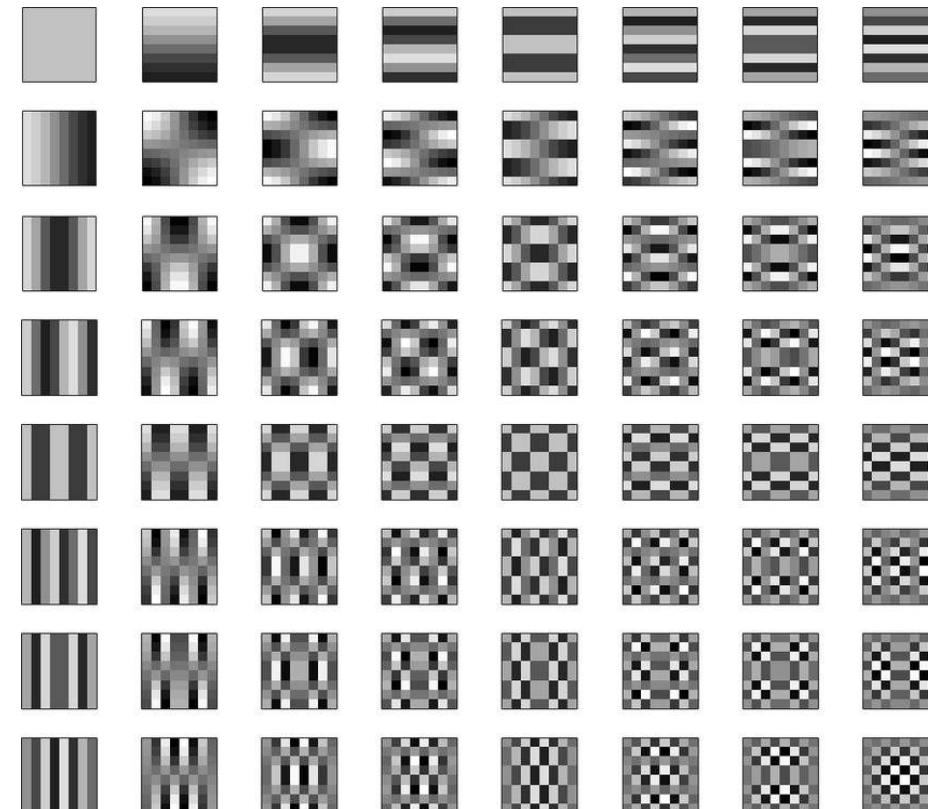
Slightly simplified, we can say that:

- Any (Riemann) integrable function has a converging Fourier series



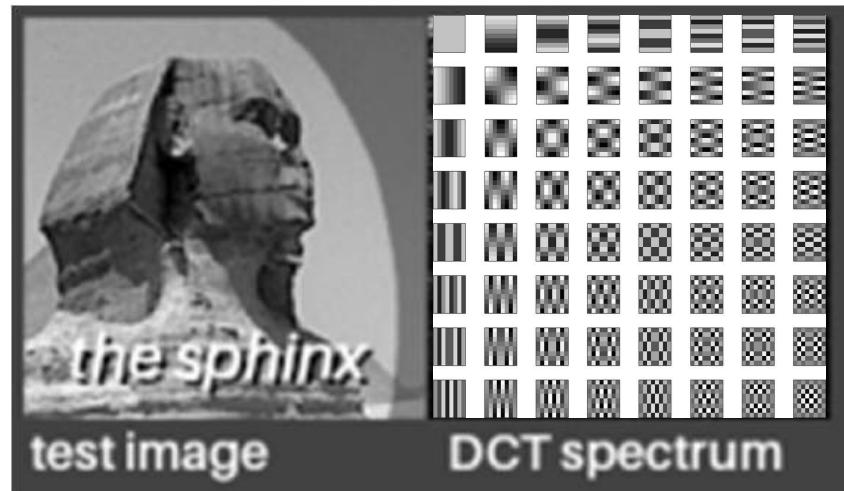
## Complex Image Format: JPEG

- Discrete Cosine Transform
- These are the base functions that are linearly combined to reproduce the image



## Complex Image Format: JPEG

- Discrete Cosine Transform



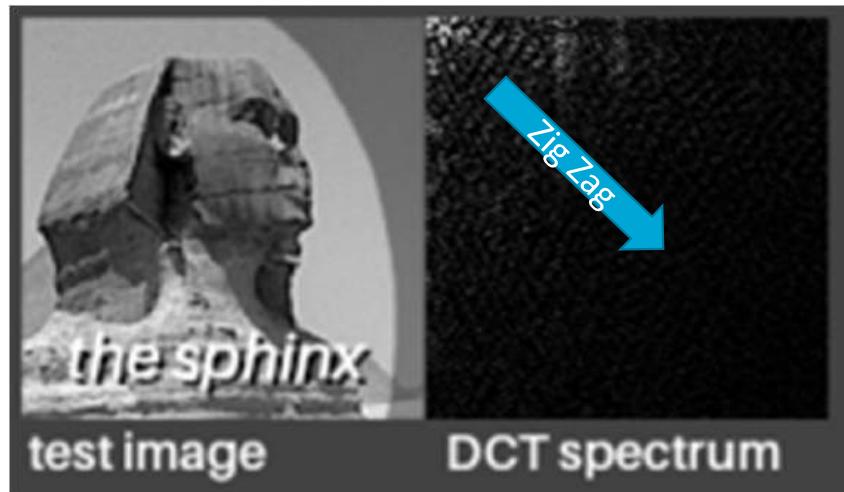
Wikimedia

Observation:

Coefficients of high frequencies are low (bottom right is mostly black), this holds generally for natural images.

## Complex Image Format: JPEG

- Discrete Cosine Transform



Wikimedia

Observation:

Coefficients of high frequencies are low (bottom right is mostly black), this holds generally for natural images.

Idea:

Quantize values (e.g., transform to bytes)

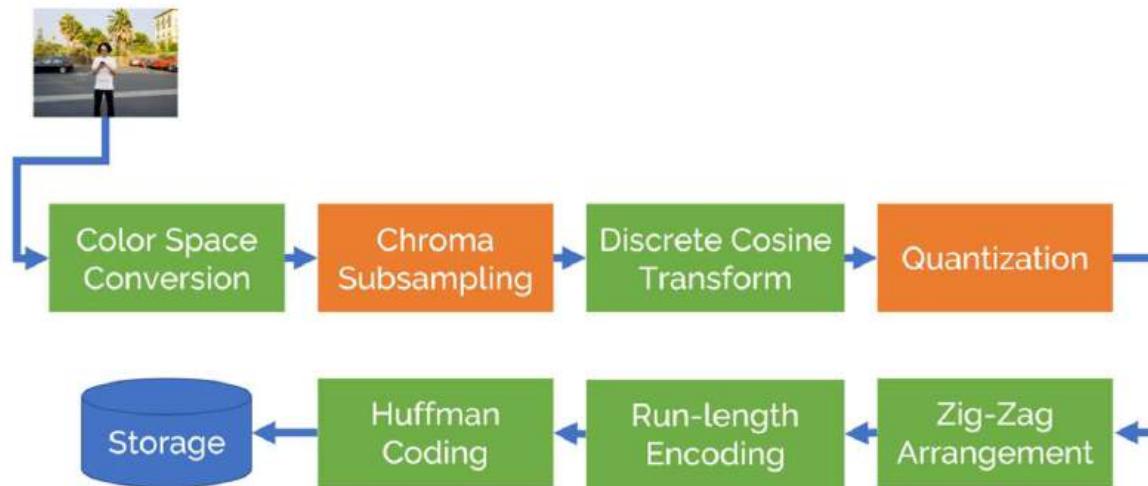
Move from top left to bottom right (zigzag)

Efficient encoding (e.g., count consecutive 0s)

1	3	6
2	5	8
4	7	9

## Want to know more about JPEG?

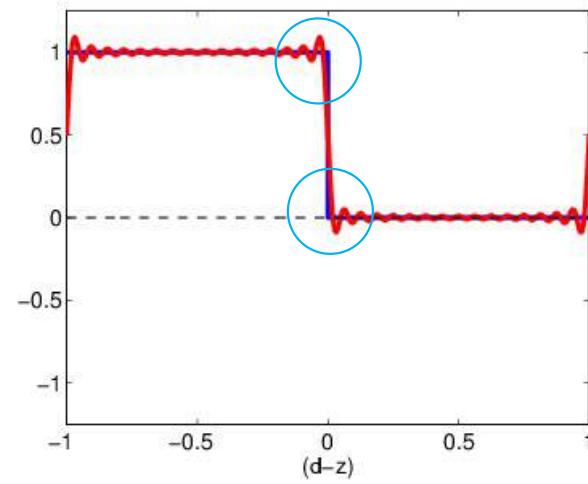
- Principle of JPEG compression
- Leo Isikdogan: <https://www.youtube.com/watch?v=Ba89cl9elg8>



Details on Signal Processing are in the book as well Chapter 9 – NOT mandatory to read

# JPEG Compression

- Very efficient
- Widespread use (e.g., internet and cameras)
- Quality is slightly reduced



## Image Storage != Image in use

- When working with images (modifications, display, conversion...) the image is usually in the decompressed form
- JPEG requires more memory when the image is loaded in memory compared to its file size

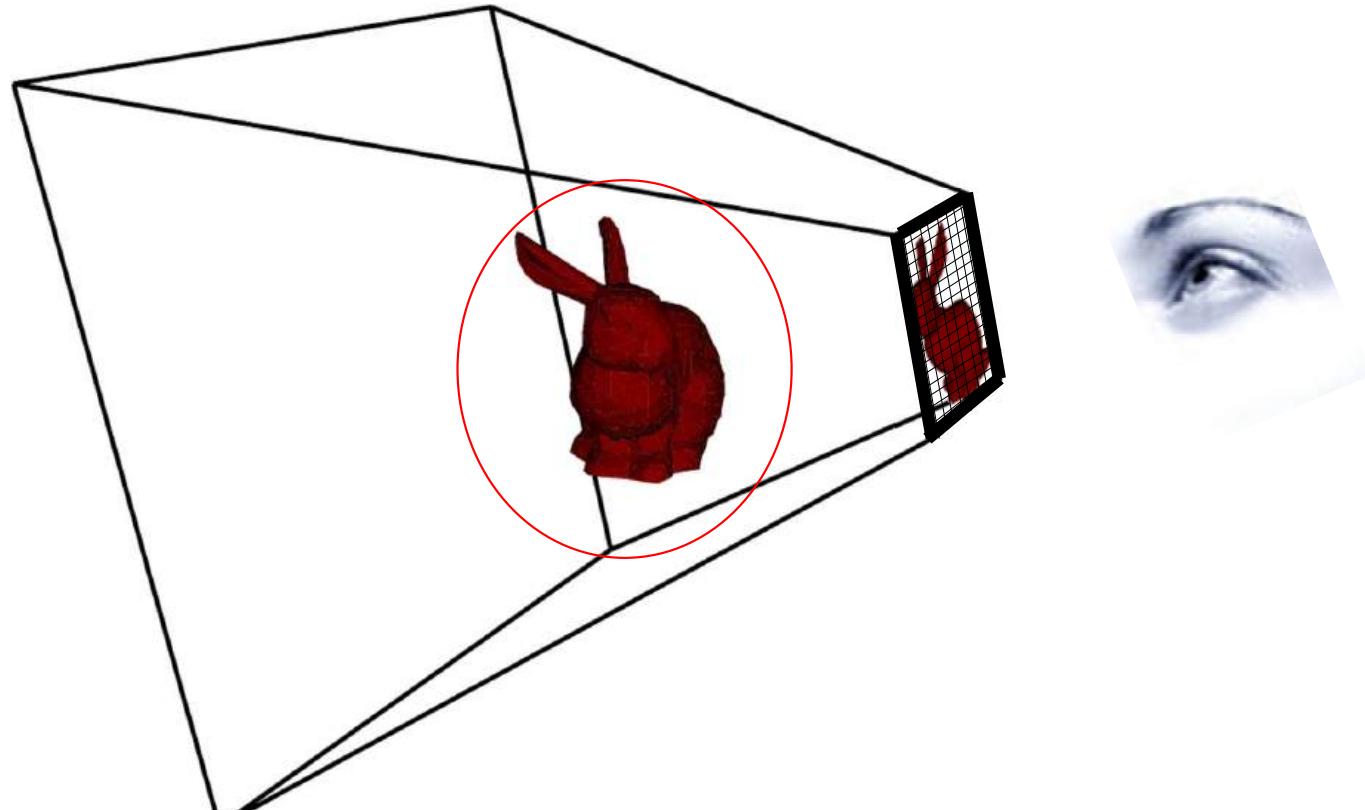
# Images

- What is an image?
- How to represent it in memory?
- How to access individual pixels?
- How can we process images?
- How are images stored?

## BREAK



## Today's Part 2 - Geometry



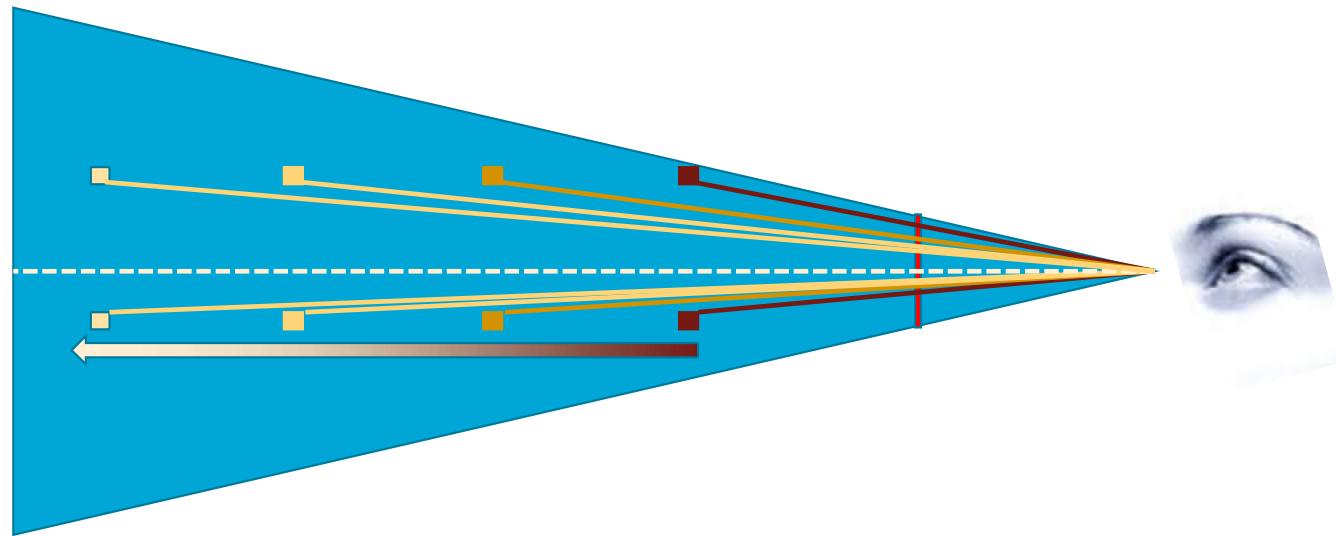
# Linking Algebra and GPUs

What has algebra to do with Graphics Cards (GPUs)?



How to draw with accurate perspective?

# Linear Perspective

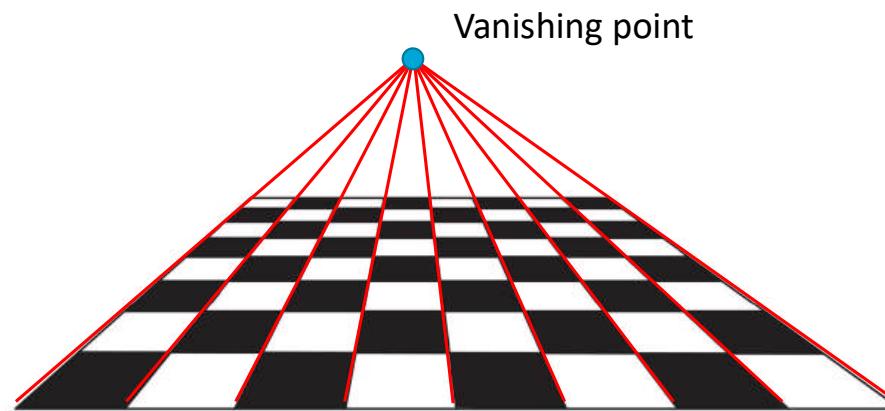


# Linear Perspective



# Linear Perspective

- Central Perspective





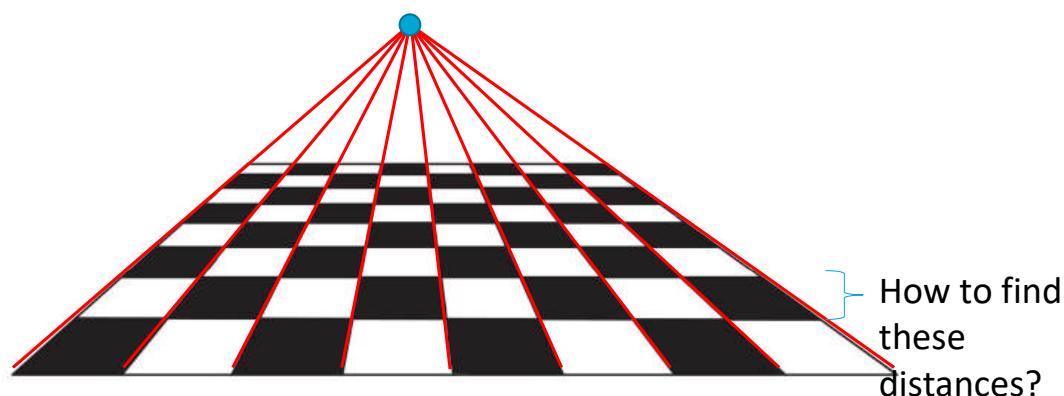
Johannes Vermeer, *The Milkmaid*, Rijksmuseum



Detail of an x-ray  
of *The Milkmaid*  
(courtesy Rijksmuseum)

# Linear Perspective

- Central Perspective

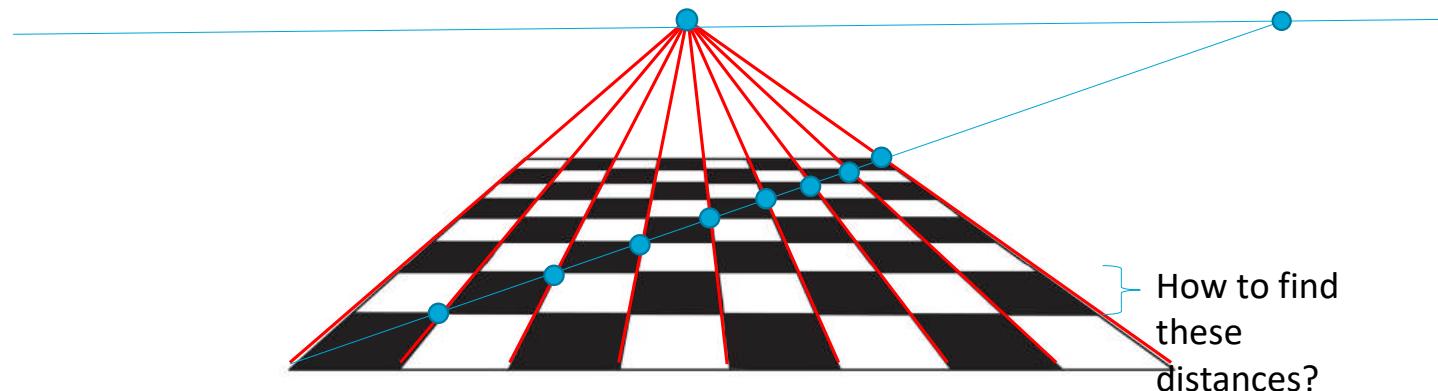




97

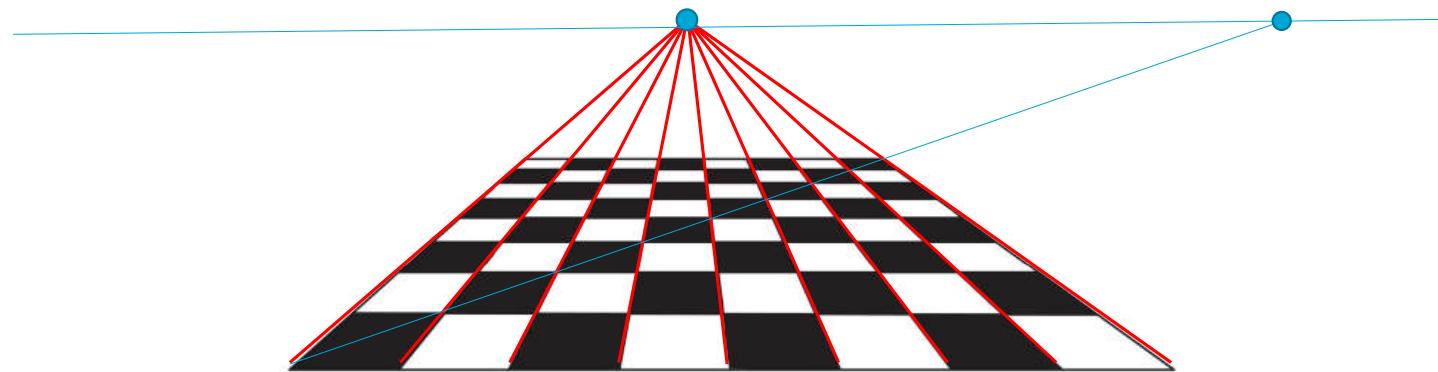
# Linear Perspective

- Central Perspective



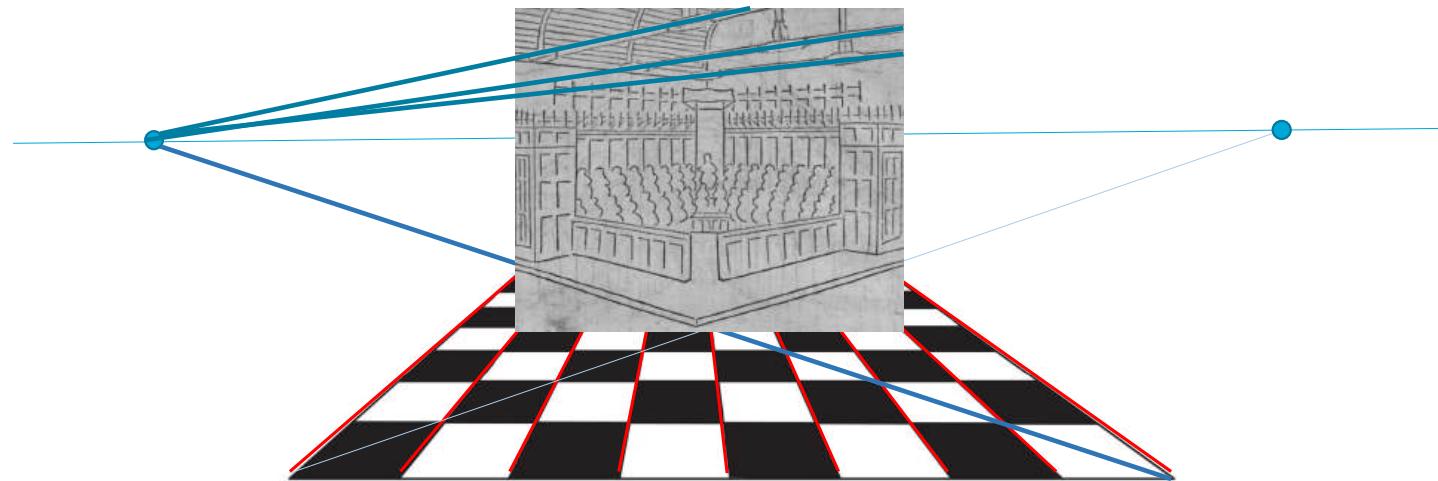
# Linear Perspective

- Central Perspective

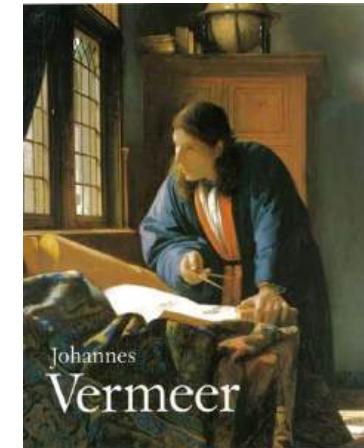
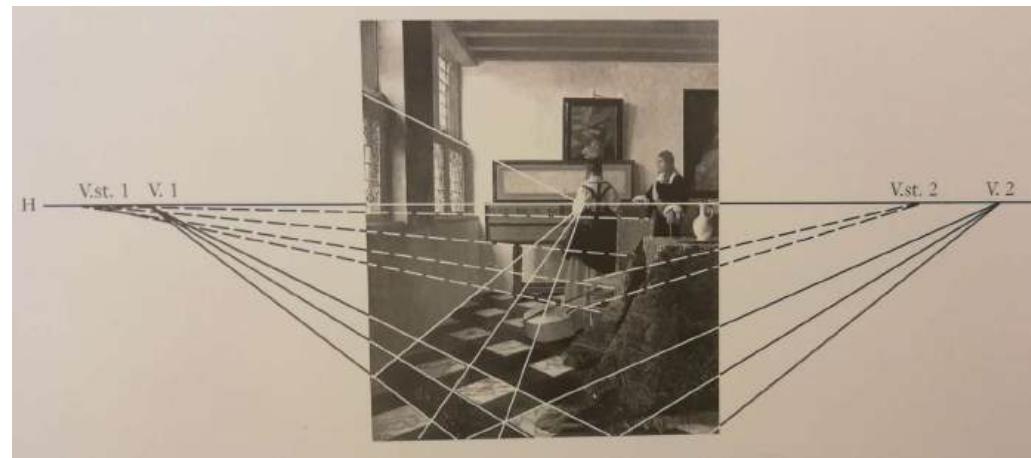


# Linear Perspective

- Two Point Perspective



Johannes Vermeer, *The Music Lesson*, ca. 1662-1665, Royal Collection Trust



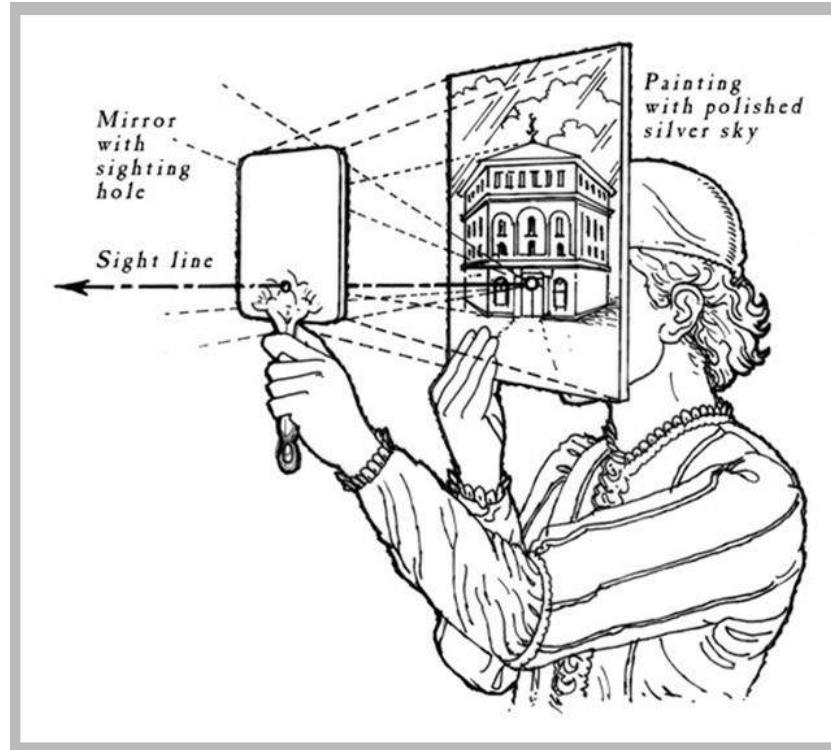
How to find the correct location  
for these vanishing points?

# Linear Perspective

$$\begin{aligned}
 & \left\{ \emptyset : \bar{F} = -1 < \int_{-1}^{\infty} \bigcup_{A \in A} \cos^{-1} (\|A\|) d\Xi'' \right\} \\
 &= \prod_{Z,g=1}^0 \mathfrak{t} \left( -\infty, \dots, \frac{1}{1} \right) - \dots \pm 2X \\
 &\neq \bigcap_{\mathcal{S}=1}^2 \mathbf{x}(d, \dots, 1 \times e) \vee S(r_i, \mathcal{X}(\Phi'), \dots, L(\mathcal{W}_{G,\Psi}) \cap -1) \\
 &\geq \frac{\varphi^{(\epsilon)}}{-O} - \log(-c_Z(\mathfrak{s}')).
 \end{aligned}$$

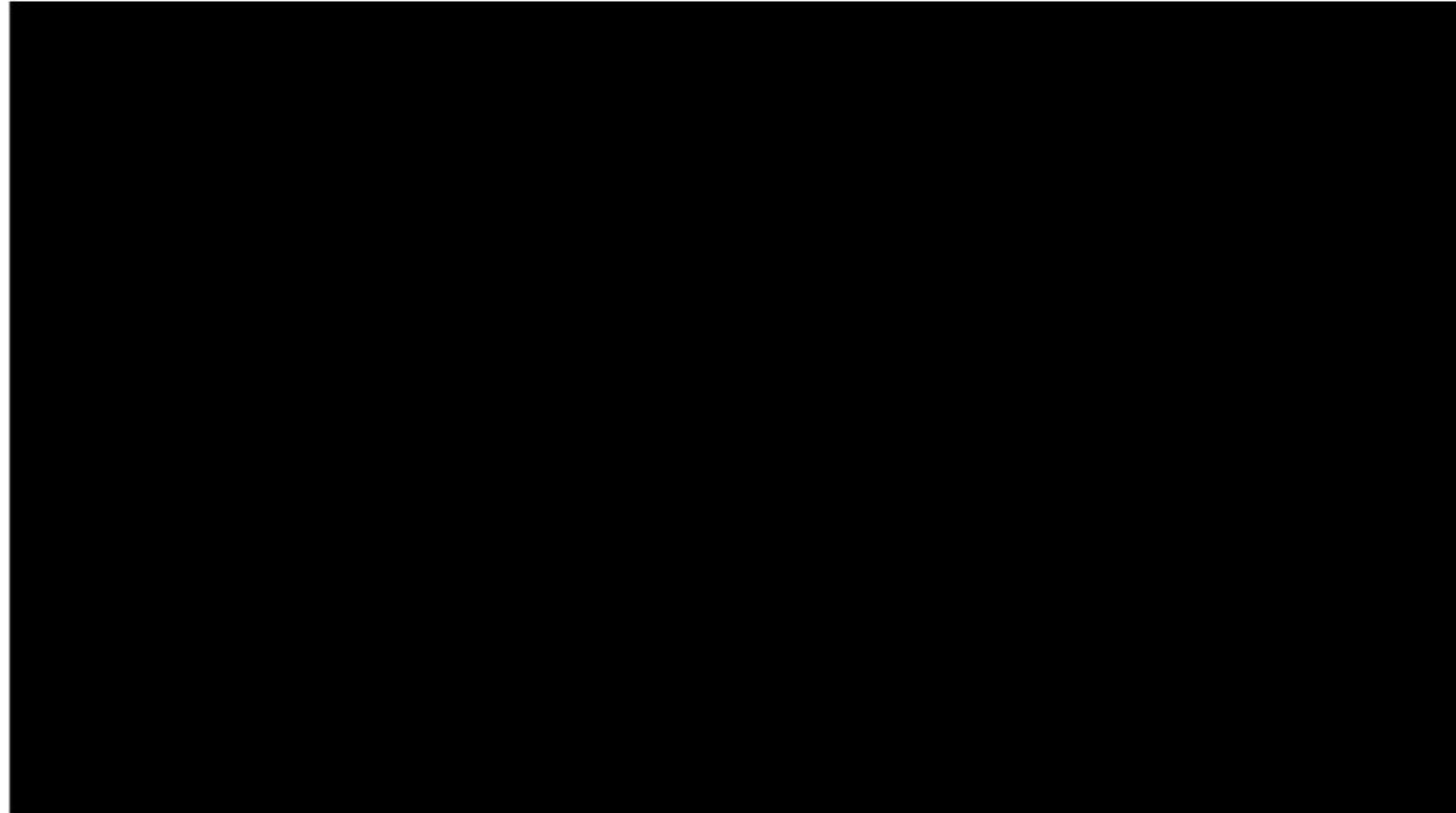


# Linear Perspective



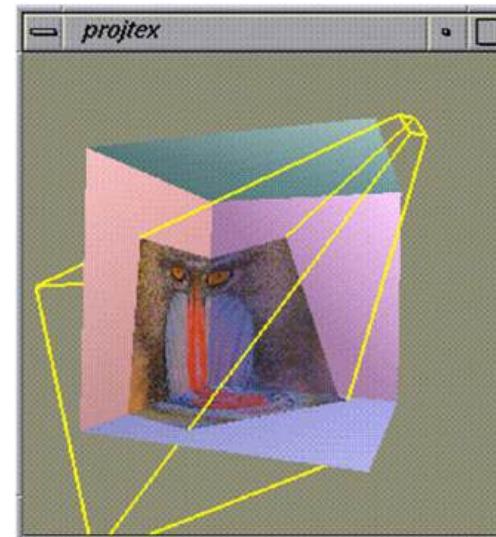
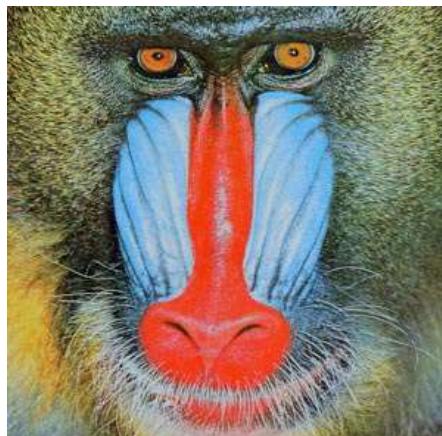
Filippo Brunelleschi – 1377–1446

## How to convince in a modern age?



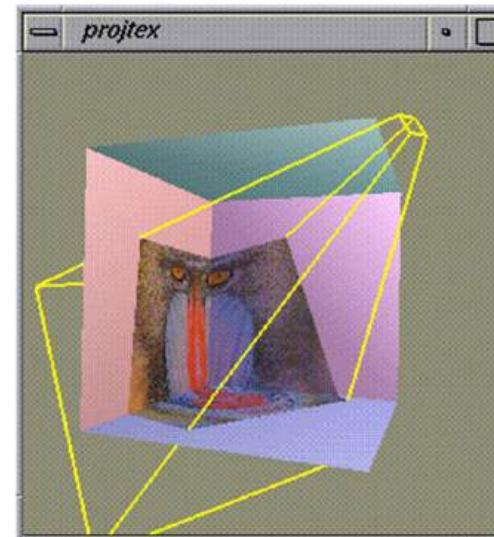
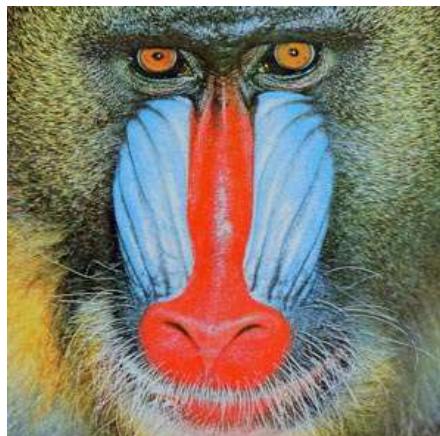
## Spotlight Technology

- Create virtual image for camera
- Project on screen and film with real camera



## Spotlight Technology

- Create virtual image for virtual camera
- Project on screen and film with real camera



# Linear Perspective

$$\begin{aligned}
 & \left\{ \emptyset : \bar{F} = -1 < \int_{-1}^{\infty} \bigcup_{A \in A} \cos^{-1}(\|A\|) d\Xi'' \right\} \\
 &= \prod_{Z,g=1}^0 \mathfrak{t}\left(-\infty, \dots, \frac{1}{1}\right) - \dots \pm 2X \\
 &\neq \bigcap_{\mathcal{S}=1}^2 \mathbf{x}(d, \dots, 1 \times e) \vee S(r_i, \mathcal{X}(\Phi'), \dots, L(\mathcal{W}_{G,\Psi}) \cap -1) \\
 &\geq \frac{\varphi^{(\epsilon)}}{-O} - \log(-c_Z(\mathfrak{s}')).
 \end{aligned}$$

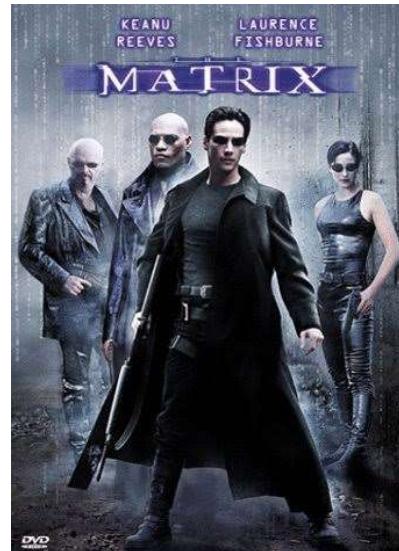


# Linear Perspective

Mathematical description:

- **Easy** because with **projective geometry**  
everything is **linear** using **homogeneous coordinates**

$$P = \begin{pmatrix} a_x \\ 0 \\ 0 \end{pmatrix}$$



$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \end{pmatrix}$$

Justified reaction  
when your teacher says  
something like this...

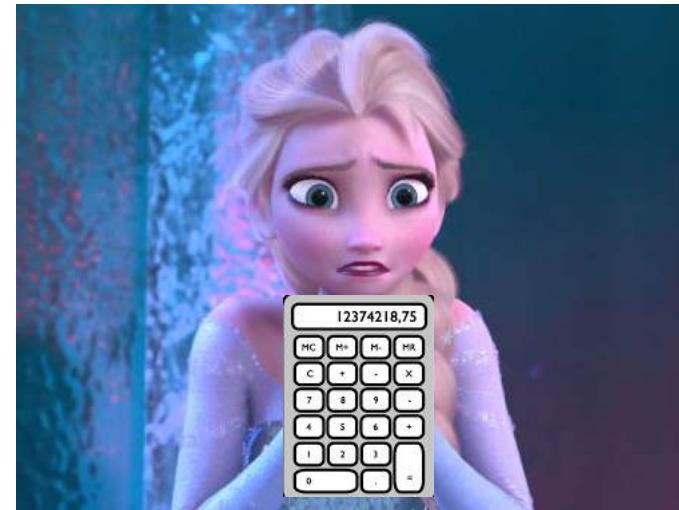


## Today

- How to build a virtual camera?
- How can projective geometry help us?  
What are homogeneous coordinates?
- How to transform objects using projective geometry?
- Next time: Full projective camera model  
Complex transformations

# Do you wanna build a camera...?

...mathematically? 😊

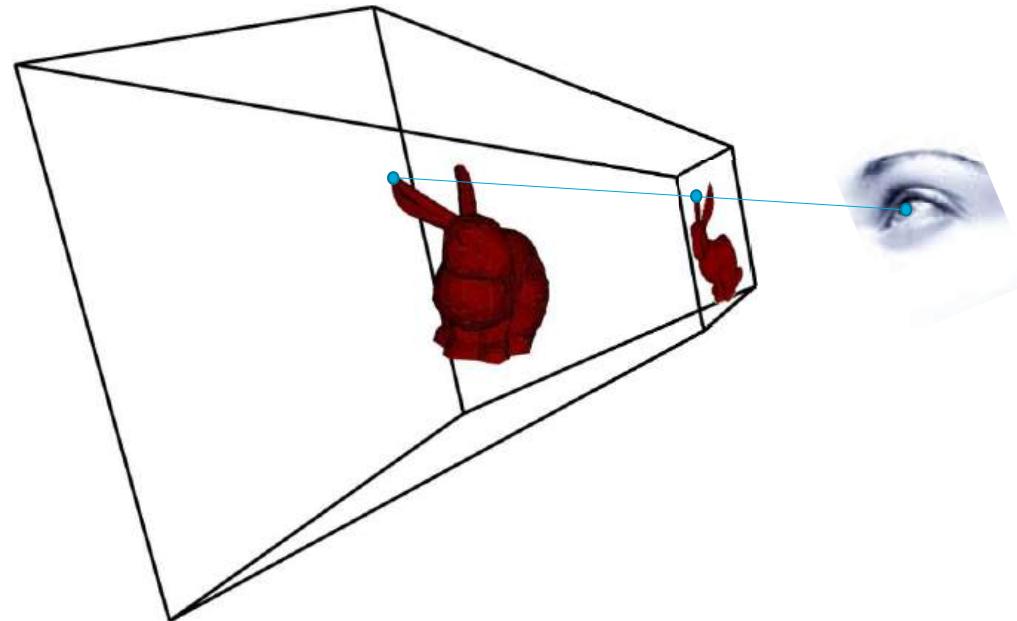


## What does a virtual camera do?

- Given a 3D point, we should find a function that results in the point's projection in the photo.

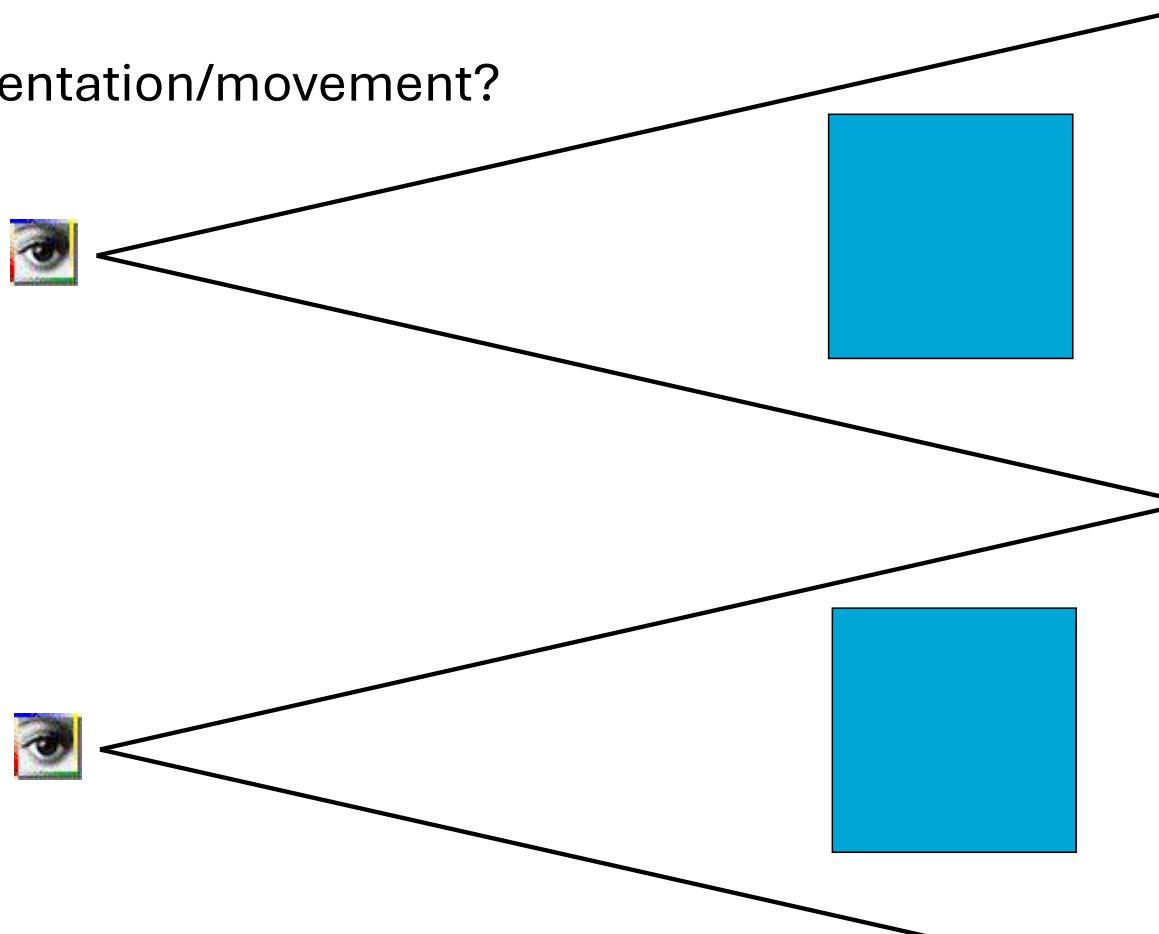


## Perspective projection



## Static Virtual Camera Model

- Camera orientation/movement?



**Idea:**  
Keep camera in one spot and modify the scene around it to indirectly move the camera and perform its projection operation.

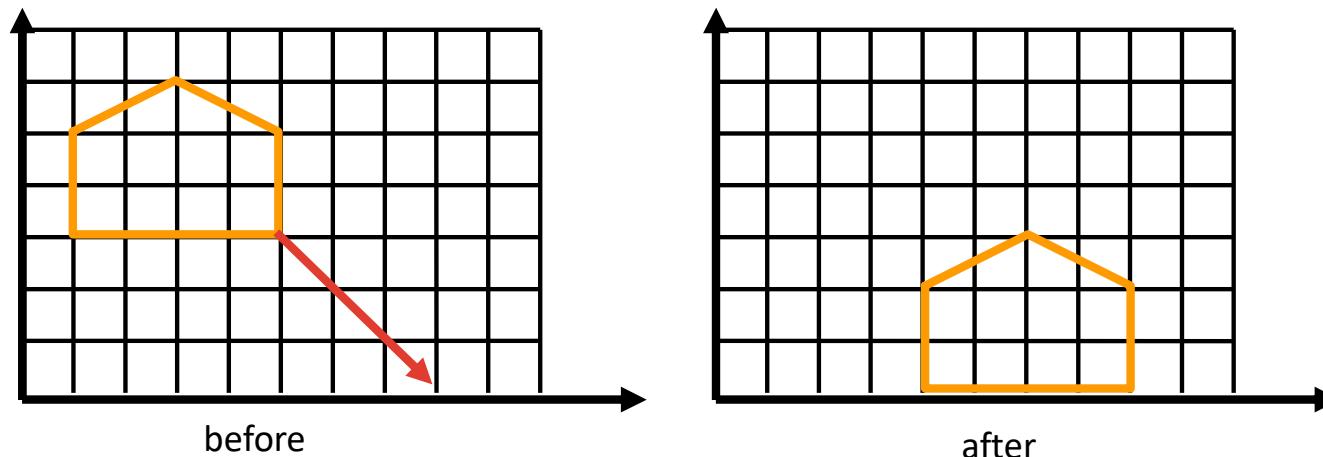
# Movement and Orientation in 2D

- Starting in 2D
  - Simpler to represent

## Translations

- Simple Modification :

- $x' = x + t_x$
- $y' = y + t_y$

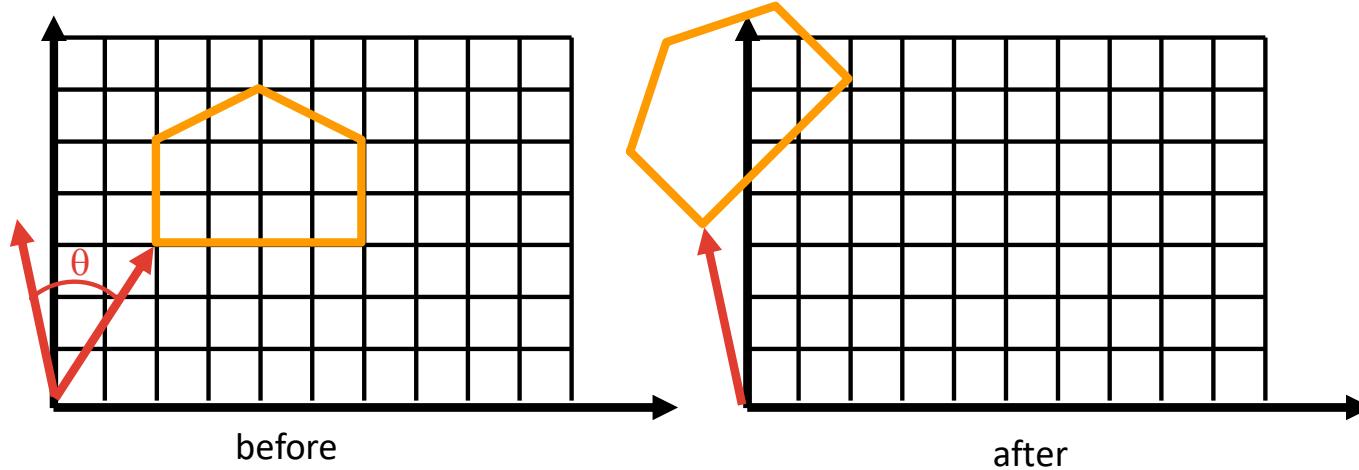


## Translation

- Is a sum of vectors:  $P' = P + T$

# Rotation

- Rotation in 2D :
  - $x' = \cos\theta x - \sin\theta y$
  - $y' = \sin\theta x + \cos\theta y$

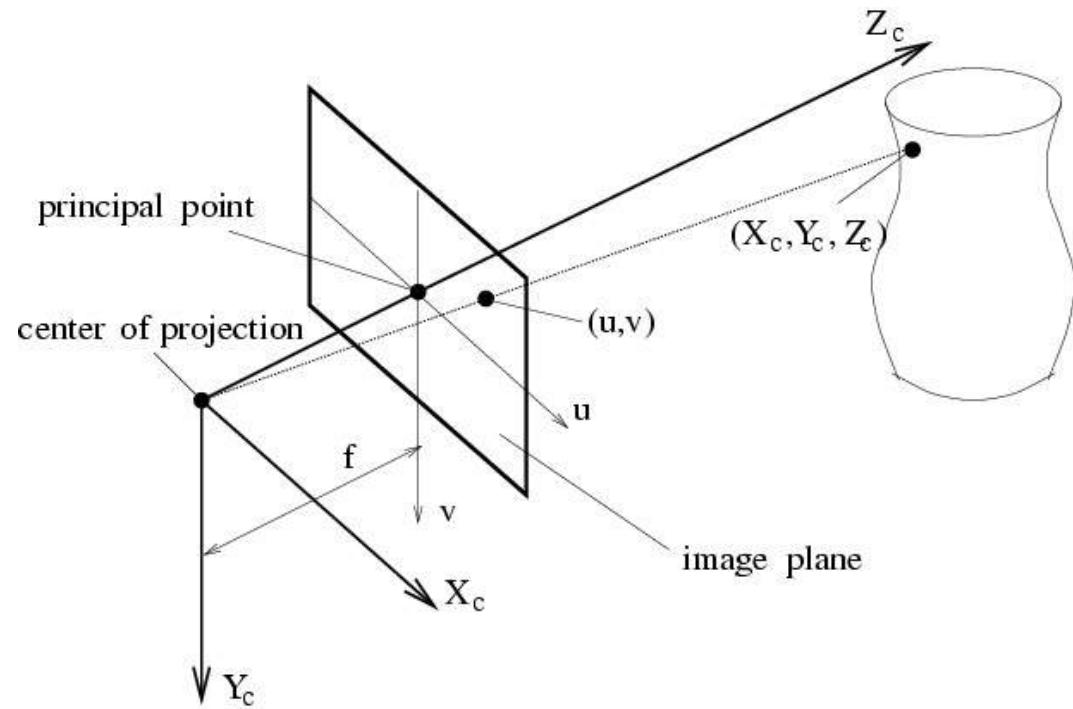


## Rotation

- Is a matrix multiplication:

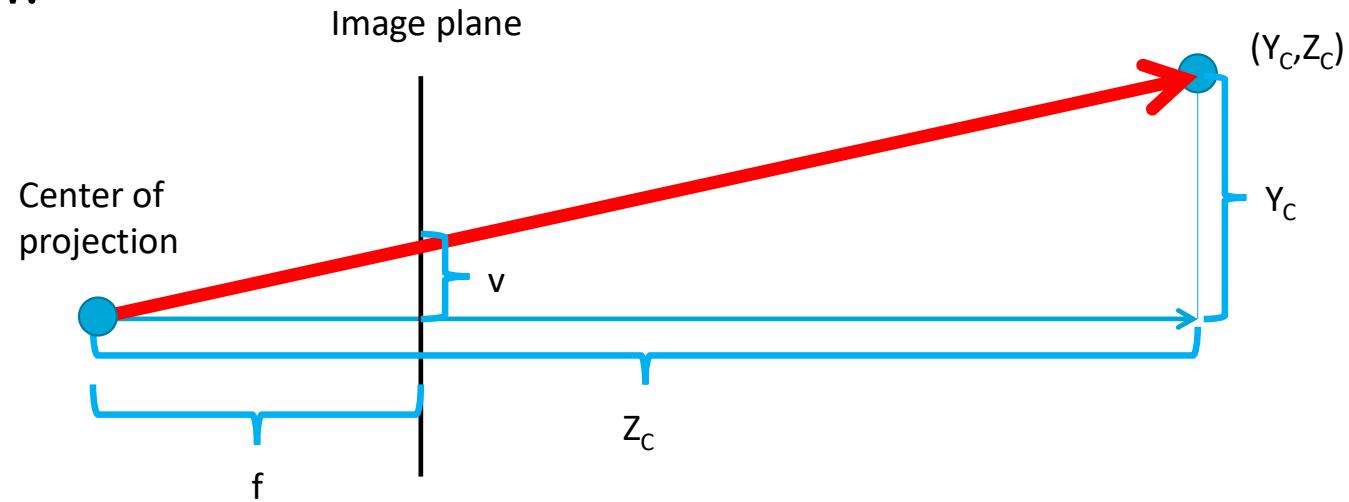
$$\mathbf{P}' = \mathbf{R}\mathbf{P}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



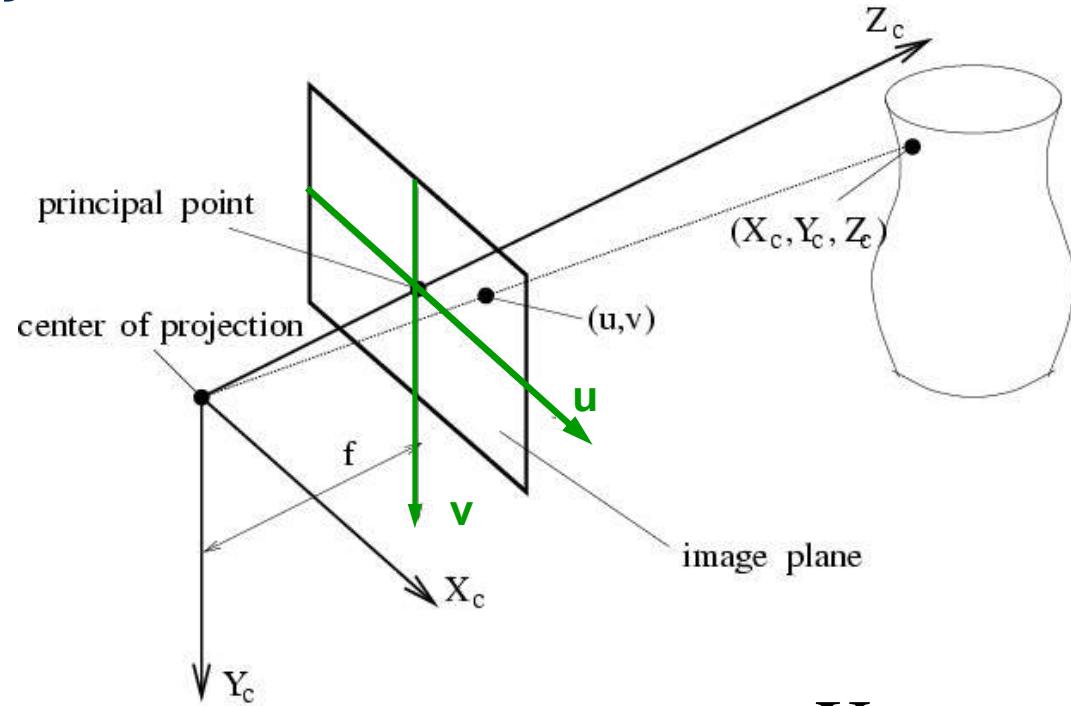
# Perspective Projection

- sideview:



$$\text{Similar triangles: } v / f = Y_C / Z_C$$

# Perspective Projection



$$u = f \frac{X_c}{Z_c} \quad v = f \frac{Y_c}{Z_c}$$

## Virtual Camera Model

Projecting a scene point with the camera:

- Apply camera position (adding an offset)
- Apply rotation (matrix multiplication)
- Apply projection (non-linear scaling)

Our camera starts to become complicated  
and not well adapted to a hardware solution...

There has to be a better way...



## What we want:

- Simple, concise notation
- Unification
  - Translation, rotation, projection

And if I am allowed to dream:

**Do everything with a matrix**



Dreams can  
come true!

## Is that really possible?

- Imagine a point  $X$  and a matrix  $T$  that describes a translation...
- $T(X) = X+t$
- $T(X+0)$

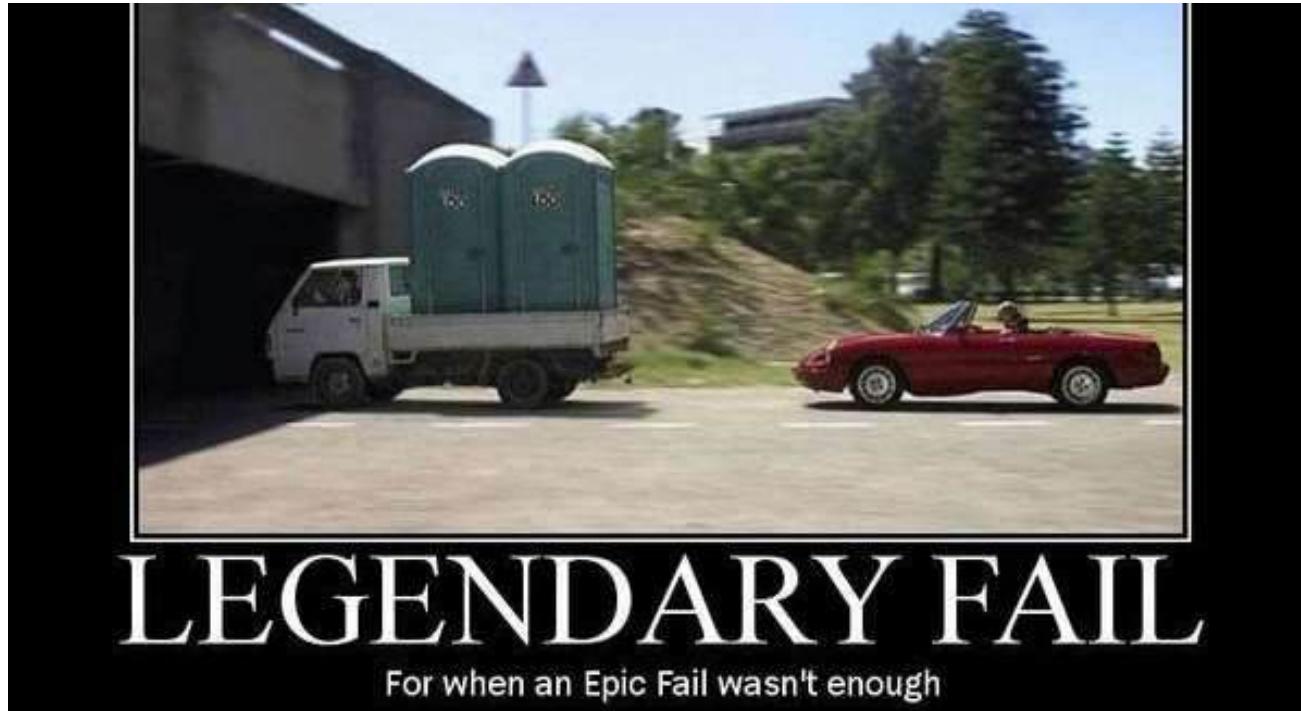
$$=T(X)+T(0)$$

$$=X+t+0+t = X+2t$$



## Fail...

- Translations are not linear (nor are projections),  
thus, cannot be represented by a matrix ...



# Today

- How to build a virtual camera?
- **How can projective geometry help us?  
What are homogeneous coordinates?**
- How to transform objects using projective geometry?
- Next time: Full projective camera model  
Complex transformations

# Projective Geometry

With homogeneous coordinates...

- Translations and rotations are matrices
- A camera projection is a matrix
- Combining matrices will prove very powerful and will allow us to define complex hierarchical dependencies (earth rotating around the sun, a hand moving with the arm...)



# Homogenous Coordinates - Definition

- $N$ -D projective space  $P^n$  is represented by  $N+1$  coordinates, has no null vector, but a special equivalence relation:

Two points  $p, q$  are **equal**

iff (if and only if)

exists  $a \neq 0$  such that  $p^*a = q$

Examples in a 2D projective space  $P^2$ :

$$(2,2,2) = (3,3,3) = (4,4,4) = (\pi, \pi, \pi)$$

$$(2,2,2) \neq (3,1,3)$$

$$(0,1,0) = (0,2,0)$$

(0,0,0) not part of the space

## Homogeneous Coordinates

To embed a standard vector space  $R^n$  in an n-D projective space  $P^n$ , we can map:  
 $(x_0, x_1, \dots, x_{n-1})$  in  $R^n$  to  $(x_0, x_1, \dots, x_{n-1}, 1)$  in  $P^n$

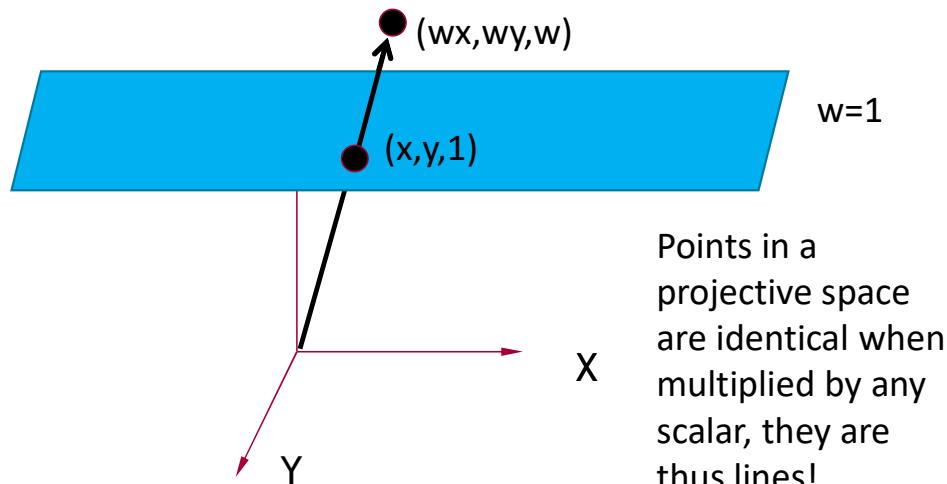
Typically, the last coordinate in a  
projective space is denoted with w.



## Homogeneous Coordinates

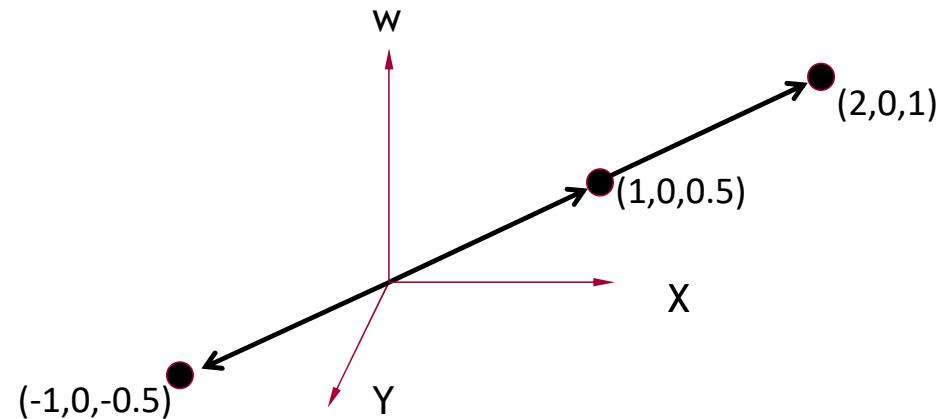
- A point  $(x,y)$  in  $R^2$  embedded in a projective space corresponds to  $(x,y,1)$ .  
All points  $(x,y,1)$  form a plane (referred to as *affine plane*)

The points in this plane correspond to points in our standard vector space  $R^2$



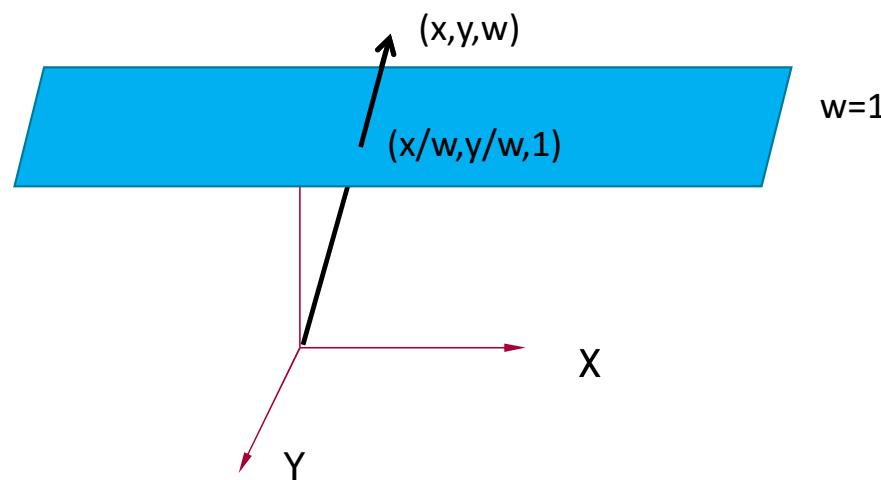
## Homogeneous Coordinates

- Example:  
This is the same point!



## Homogeneous Coordinates

- Thus, we can go back to  $R^2$  by dividing the coordinates by the last entry.  $(x,y,w)$  in  $P^2$  corresponds to  $(x/w,y/w)$  in  $R^2$ .



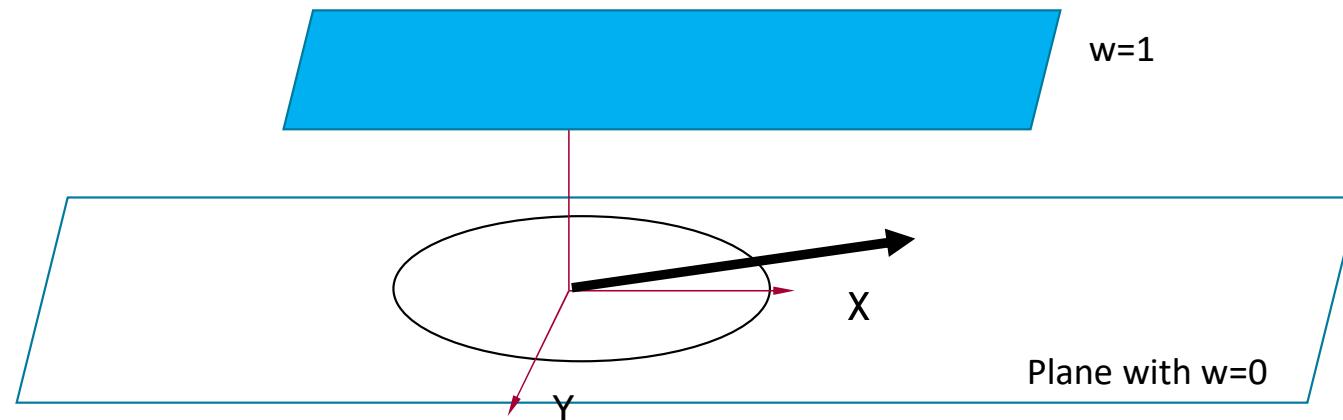
## Homogeneous Coordinates

From projective to standard vector space:

- For  $(x_0, x_1, \dots, x_{n-1}, w)$  with  $w \neq 0$ , the corresponding point in  $\mathbb{R}^n$  is  $(x_0/w, x_1/w, \dots, x_{n-1}/w)$
- $(x_0, x_1, \dots, x_{n-1}, 0)$  has no correspondence in  $\mathbb{R}^n$ !

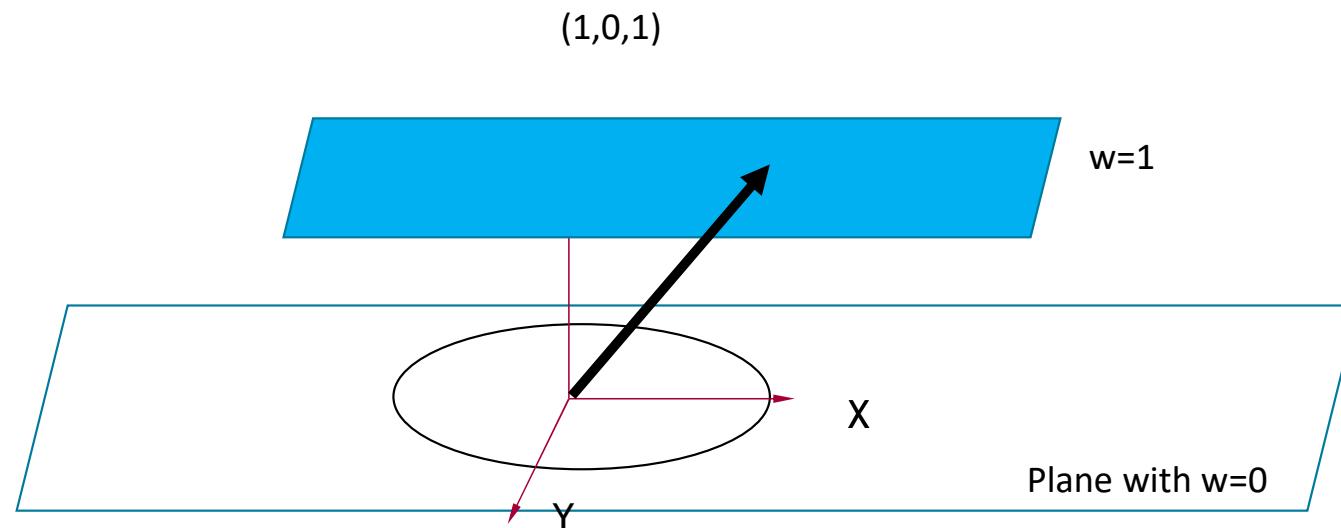
## Homogeneous Coordinates

- What about the points with  $w=0$ ?



## Homogeneous Coordinates

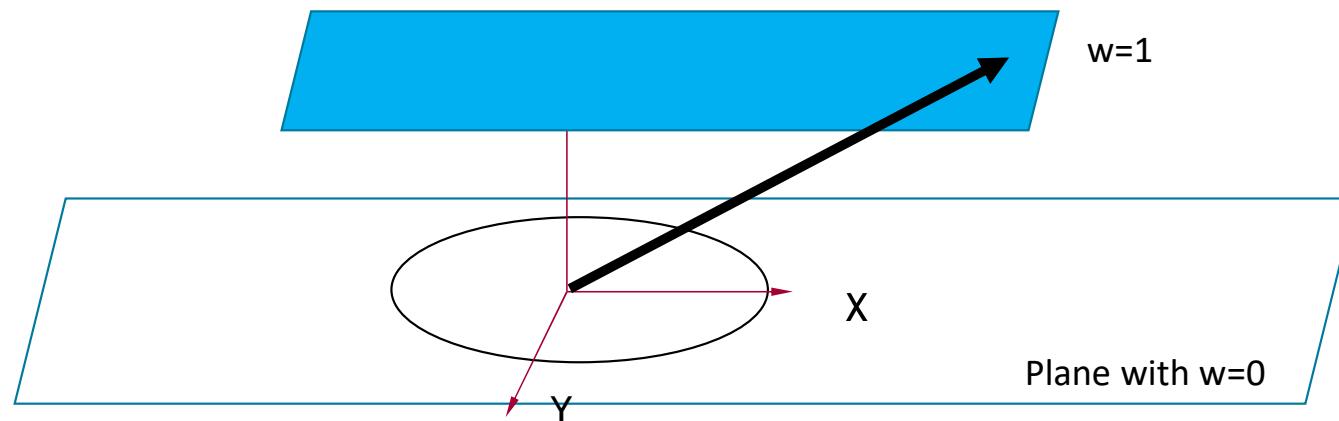
- What about the points with  $w=0$ ?
- Let's try it out with a point  $(1,0,w)$  and we decrease  $w$ ...



## Homogeneous Coordinates

- What about the points with  $w=0$ ?
- Let's try it out with a point  $(1,0,w)$  and we decrease  $w$ ...

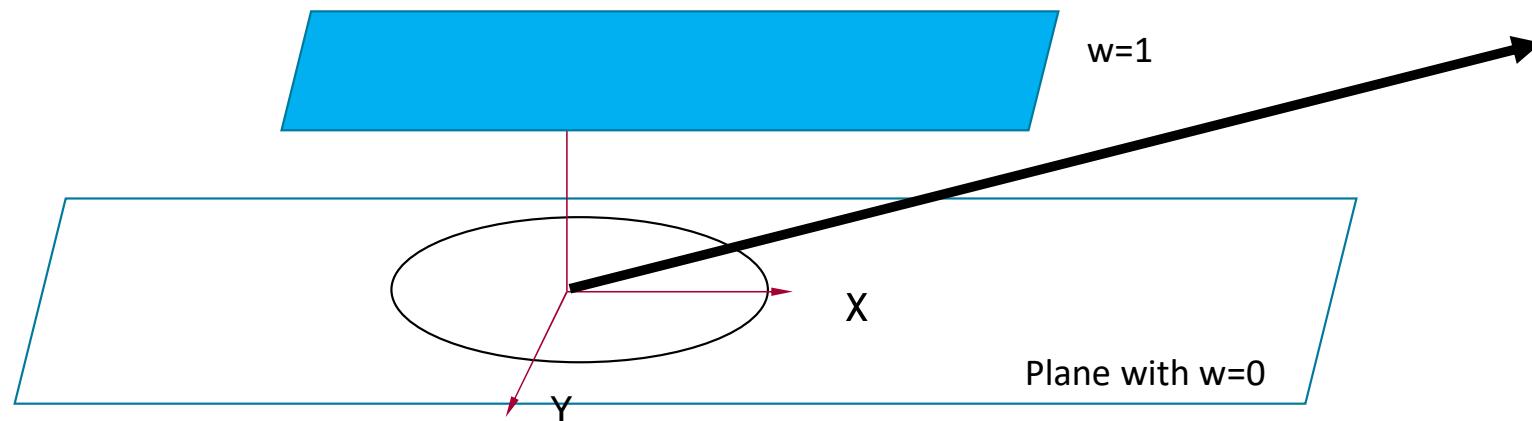
$$(1,0,0.5)=(2,0,1)$$



## Homogeneous Coordinates

- What about the points with  $w=0$ ?
- Let's try it out with a point  $(1,0,w)$  and we decrease  $w$ ...

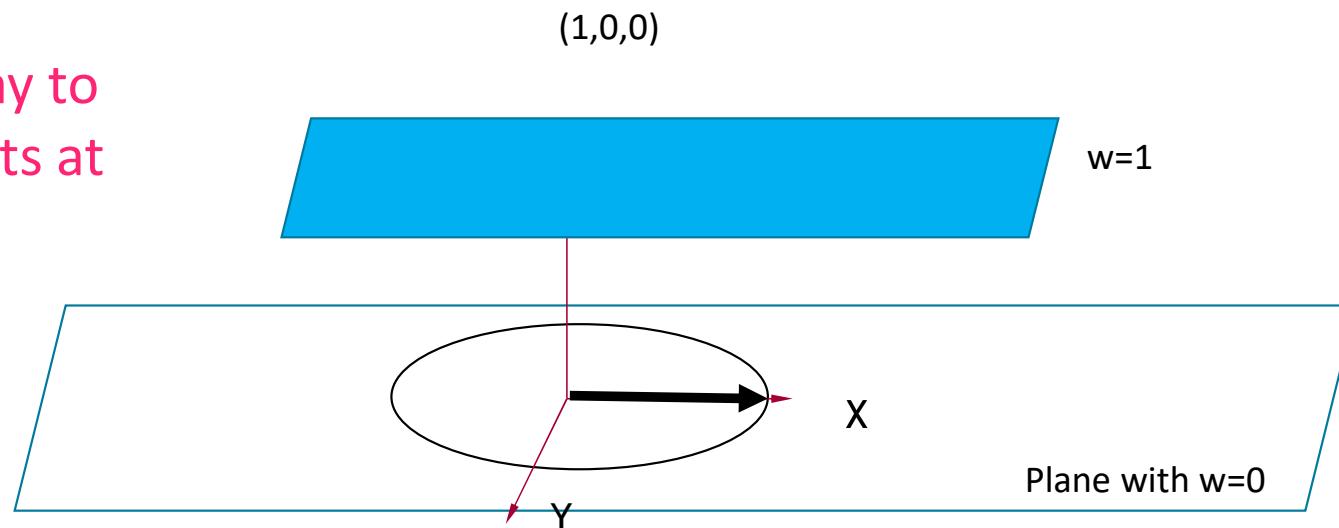
$$(1,0,0.25) = (4,0,1)$$



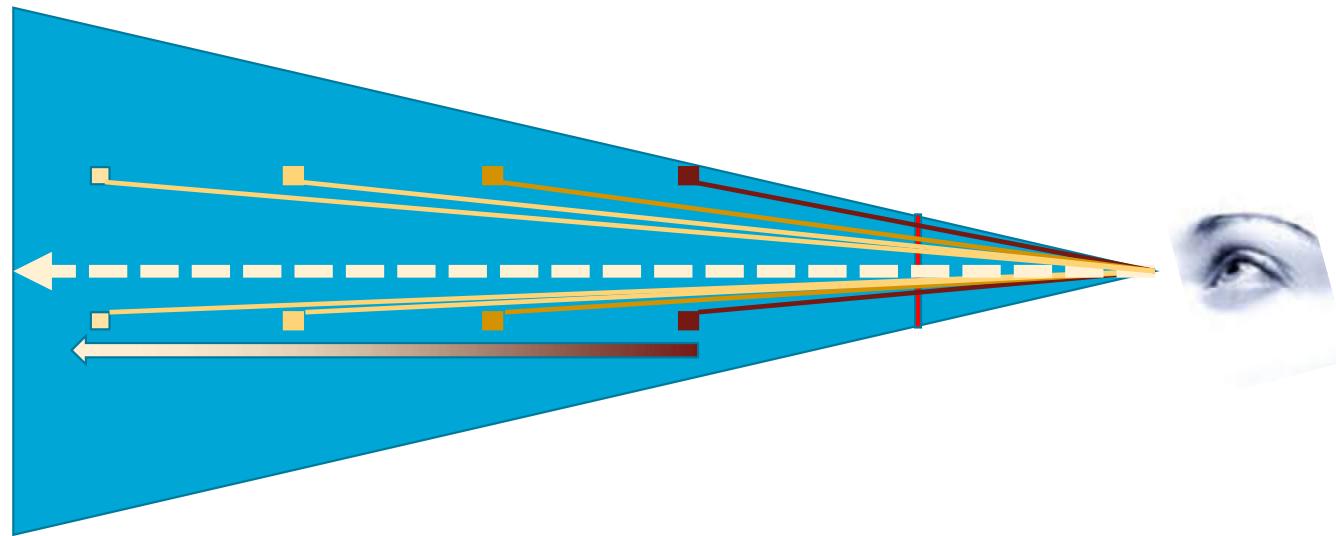
## Homogeneous Coordinates

- What about the points with  $w=0$ ?
- Let's try it out with a point  $(1,0,w)$  and we decrease  $w$ ...

We have a way to  
describe points at  
INFINITY!



# Linear Perspective



# Linear Perspective



# Homogeneous Coordinates

We will see:

Easy transformations = matrices

Translations, rotations, scaling, projection are matrices

Concatenating transformations is trivial!  
simple matrix multiplications

## Translations in $\mathbf{R}^2$

- To translate vector  $(x,y)$ ,  
we add  $(t_x, t_y)$  to obtain:  $(x+t_x, y+t_y)$

In a projective space,  
we would like to have a matrix  $M$ , such that

$$M(x,y,1) = (x+t_x, y+t_y, 1)$$

## Translations in homog. coordinates

- $\mathbb{R}^2$

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

---

- $\mathbb{P}^2$

## Translations in homog. coordinates

- $\mathbb{R}^2$

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

- $\mathbb{P}^2$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$

## Translations in homog. coordinates

- $\mathbb{R}^2$

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

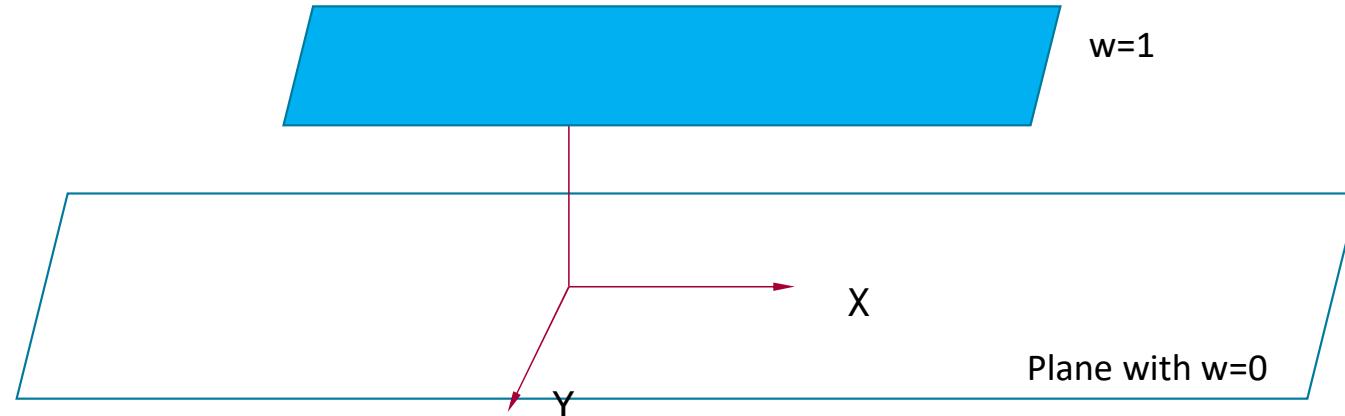


- $\mathbb{P}^2$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$

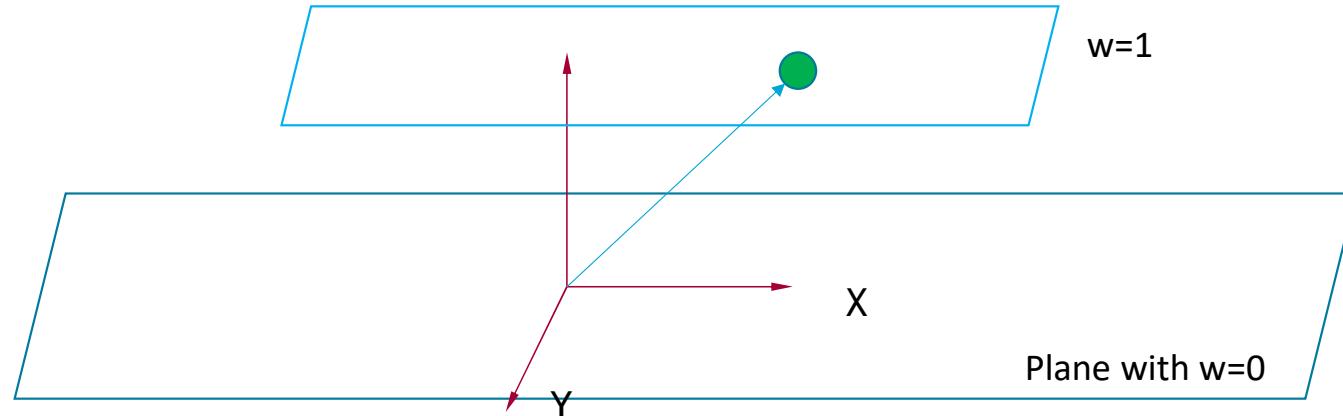
## How does this work?

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$



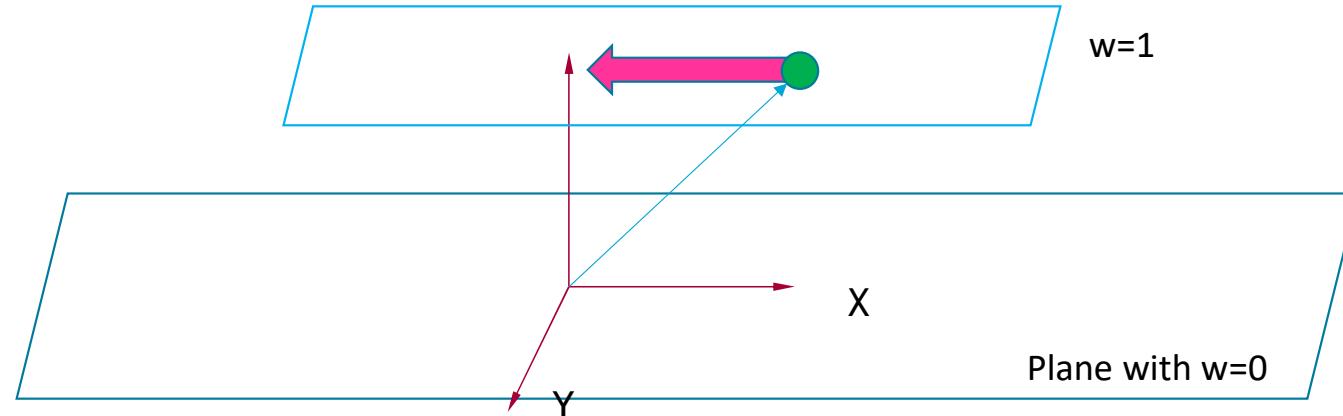
## How does this work?

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left[ \begin{array}{c|c} 1 \\ 0 \\ 1 \end{array} \right]$$



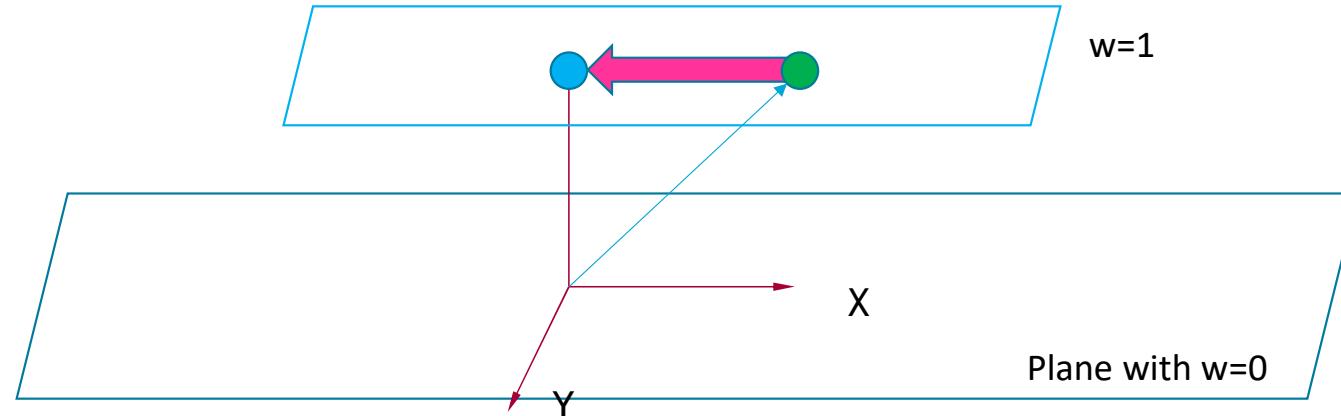
## How does this work?

$$\left[ \begin{array}{ccc|c} 1 & 0 & -1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{array} \right]$$



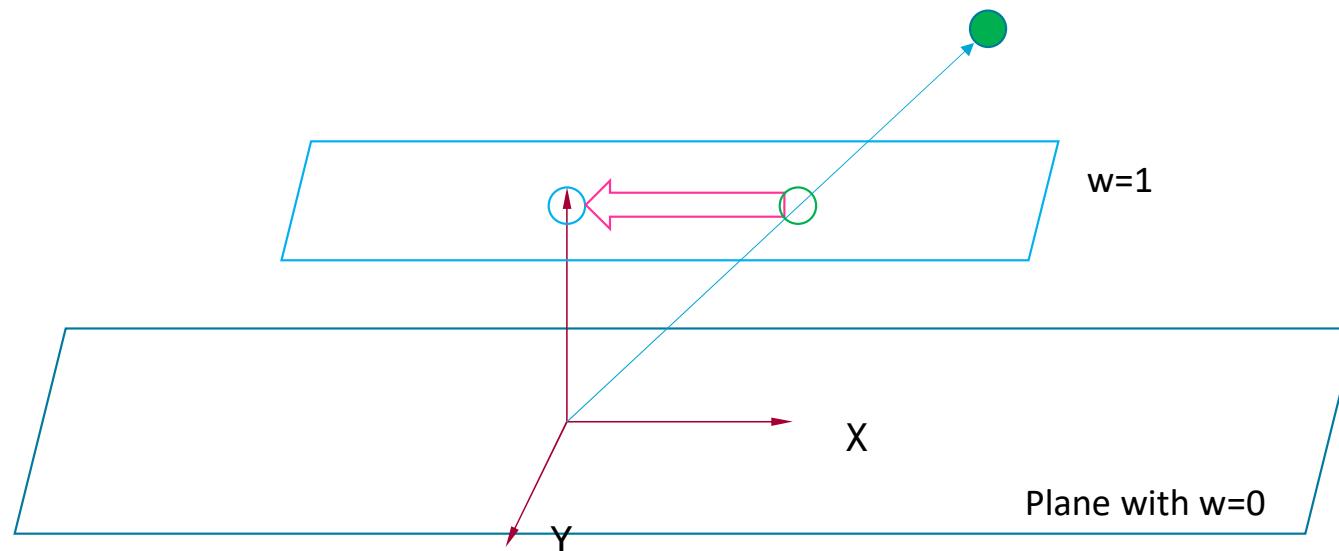
## How does this work?

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 - 1 \\ 0 + 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



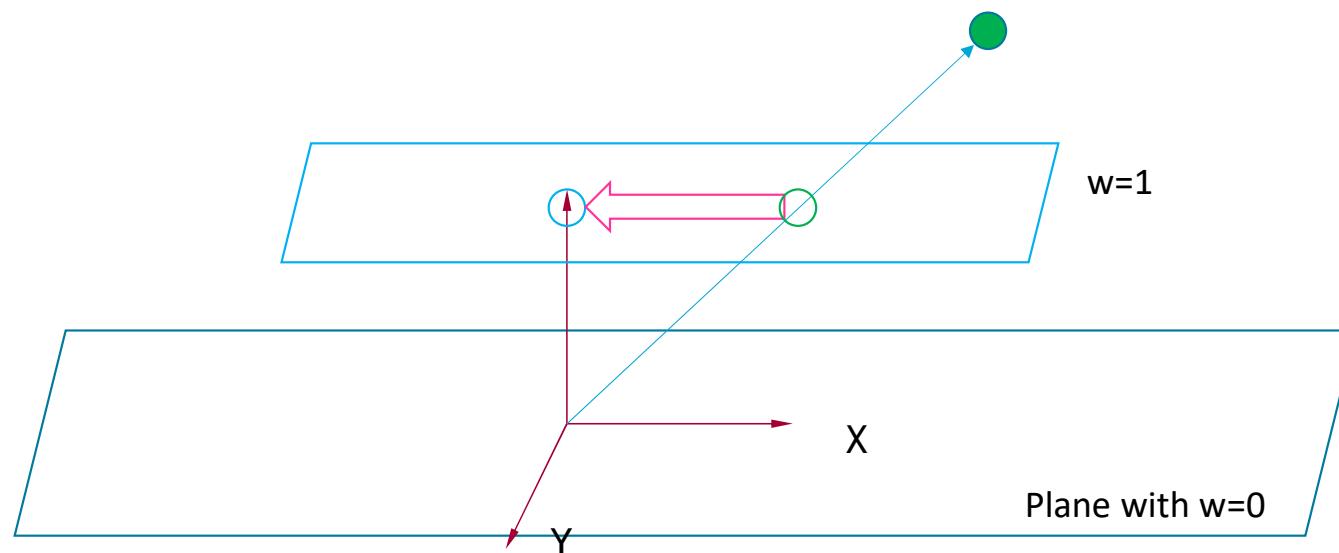
## How does this work?

$$\left[ \begin{array}{ccc|c} 1 & 0 & -1 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{array} \right]$$



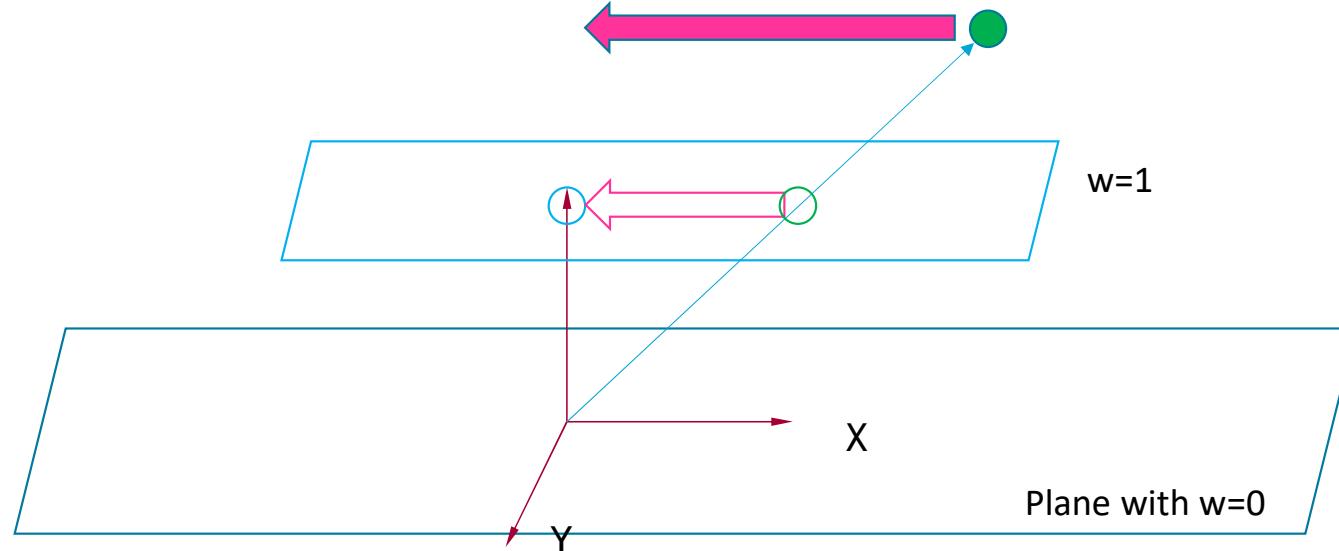
## How does this work?

$$\left[ \begin{array}{ccc|c} 1 & 0 & -1 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{array} \right]$$



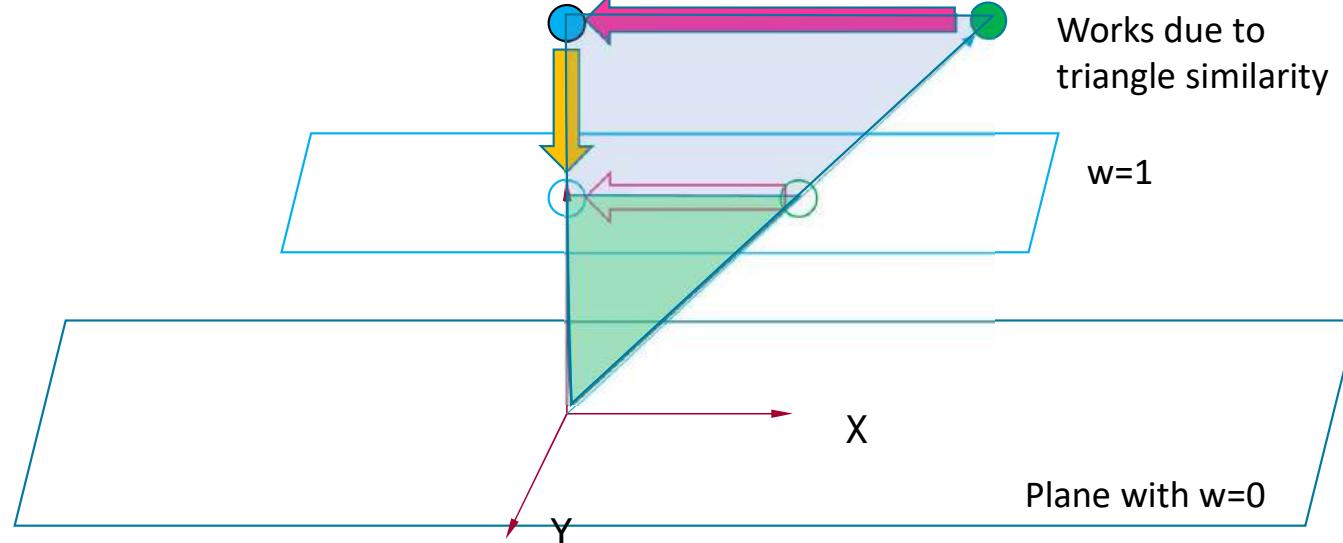
## How does this work?

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 - 2 \\ 0 + 0 \\ 2 \end{bmatrix}$$



## How does this work?

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 - 2 \\ 0 + 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



## Translations in homog. coordinates

- $\mathbb{R}^2$

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

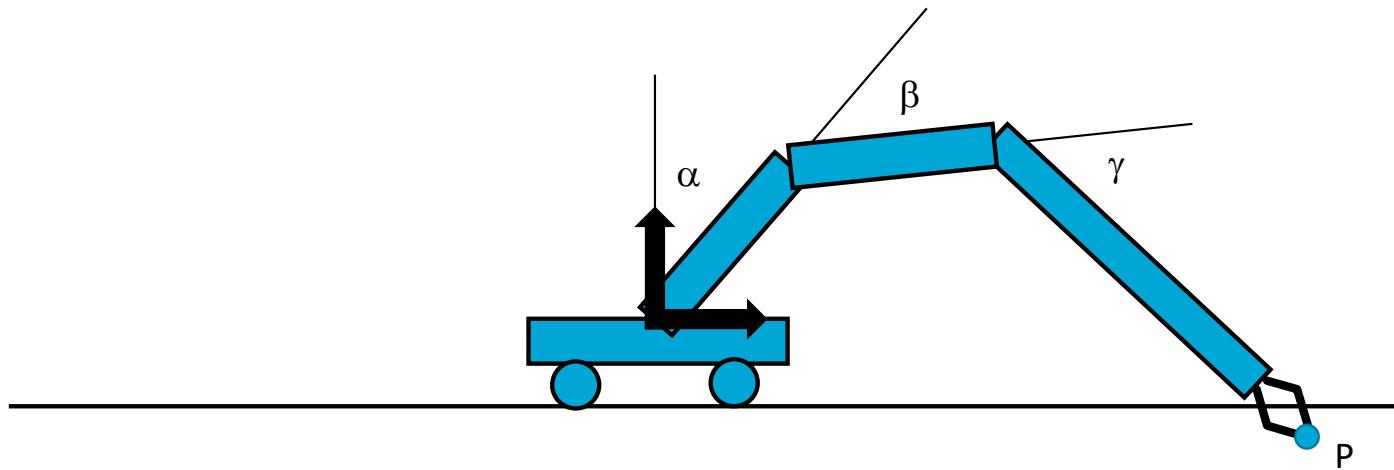


- $\mathbb{P}^2$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$

## Next time:

Change angle  $\alpha$  and calculate the new position of point P  
with **one** matrix multiplication!



# Thank you very much for your attention!

When the lecture ends before all math is explained...

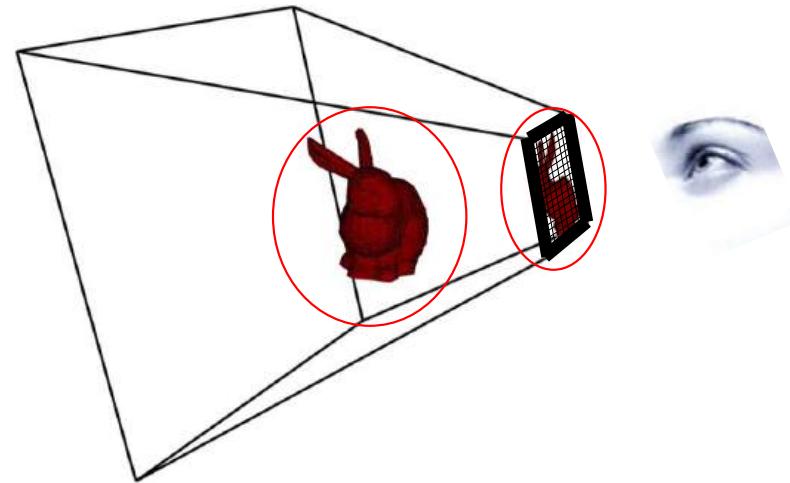


## Practice Questions

- Given index  $I$  and a two-color-channel image, how can we identify the corresponding pixel?
- Verify the claim regarding how to formulate box filtering as a general filtering.
- How would you define a matrix  $M'$  in the projective space  $P^2$  that represents the operation  $f(x,y)=(ax, by)^t$  of  $R^2$ ?

# Conclusion

- Covered a lot of ground today!
- Images
  - Representation and Processing
- Geometry
  - Introduction of Projective Geometry



**CSE2215 - Computer Graphics**

# **Geometry Pipeline**

## **Enter the Matrix**

Elmar Eisemann

Delft University of Technology



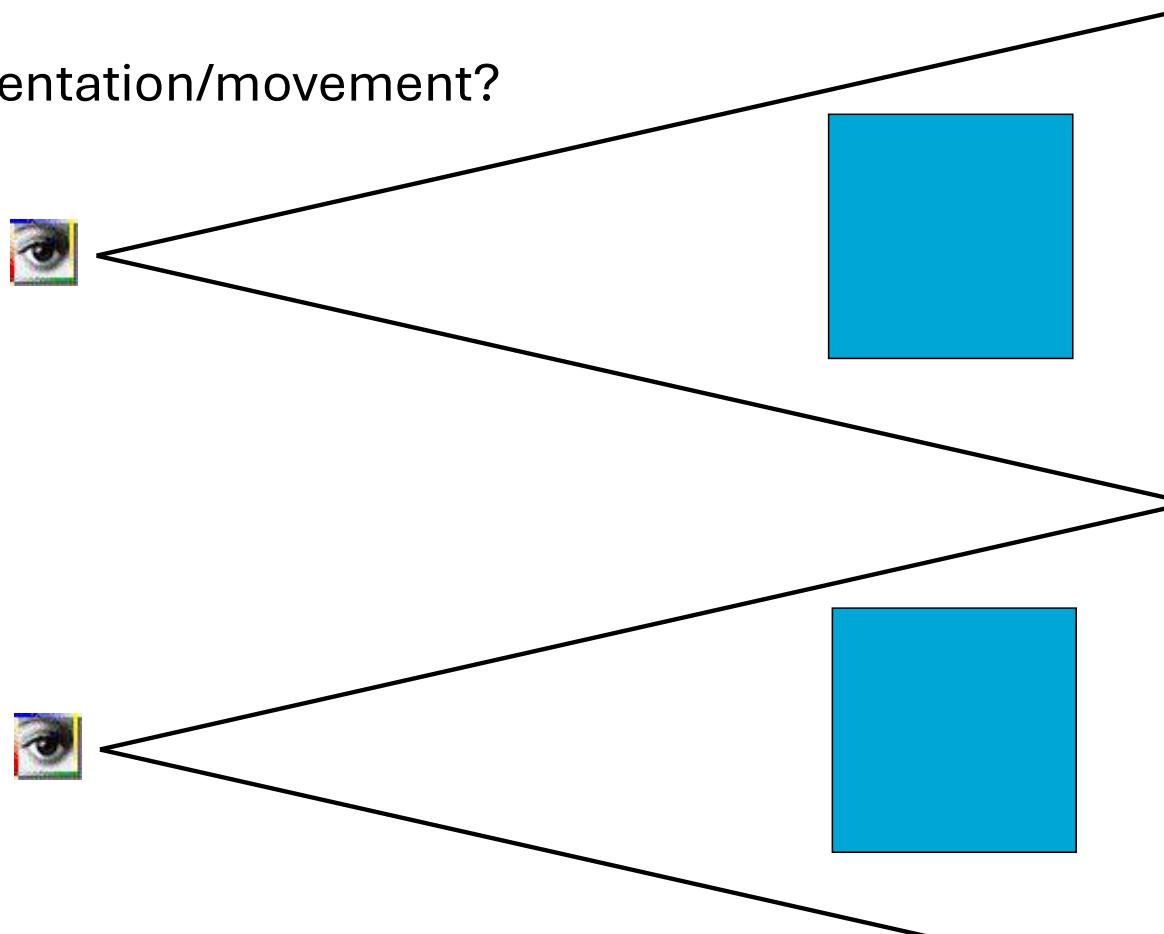
## Virtual Camera Model

- Given 3D point  $P$ , we want a function  $M$ ,  
such that  $M(P)$  is the point's projection in the photo.



## Virtual Camera Model

- Camera orientation/movement?

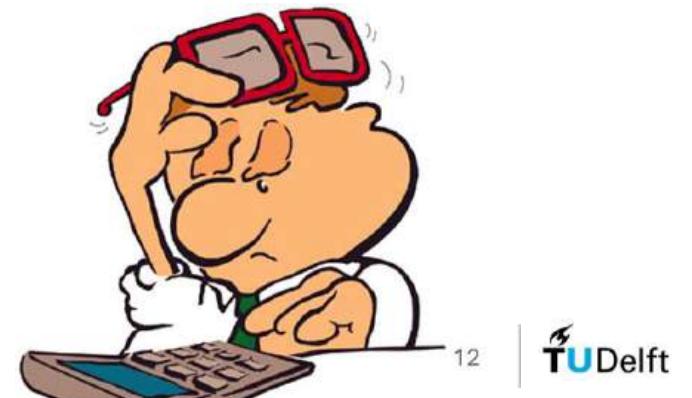


## Virtual Camera Model

Projecting a scene point with the camera:

- Apply camera position (adding an offset)
- Apply rotation (matrix multiplication)
- Apply projection (non-linear scaling)

There has to be a better way...



## What we want:

- Simple, concise notation
- Unification
  - Translation, rotation, projection...

And if I am allowed to dream:

**Do everything with a matrix**



Dreams can  
come true!

# Homogenous Coordinates - Definition

- $N$ -D projective space  $P^n$  is represented by  $N+1$  coordinates, has no null vector, but a special equivalence relation:

Two points  $p, q$  are **equal**

iff (if and only if)

exists  $a \neq 0$  such that  $p^*a = q$

Examples in a 2D projective space  $P^2$ :

$$(2,2,2) = (3,3,3) = (4,4,4) = (\pi, \pi, \pi)$$

$$(2,2,2) \neq (3,1,3)$$

$$(0,1,0) = (0,2,0)$$

(0,0,0) not part of the space

## Homogeneous Coordinates

To embed a standard vector space  $R^n$  in an n-D projective space  $P^n$ , we can map:

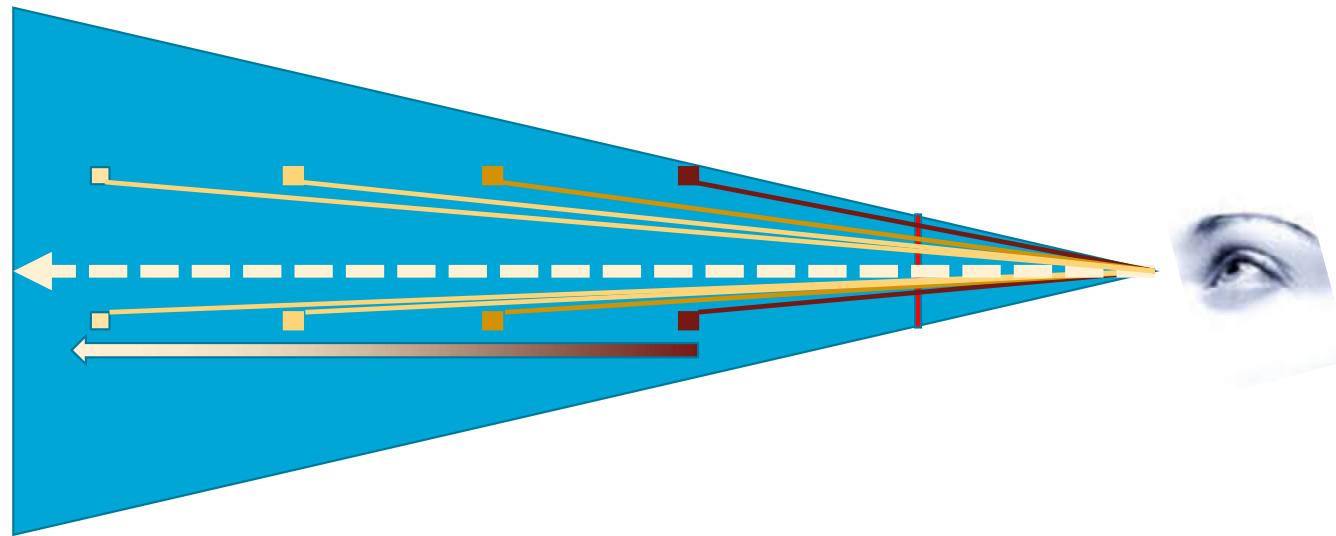
$(x_0, x_1, \dots, x_{n-1})$  in  $R^n$  to  $(x_0, x_1, \dots, x_{n-1}, 1)$  in  $P^n$



Typically, the last coordinate in a projective space is denoted with w.

What are those points in  $P^n$  with the last coordinate equal to 0?

# Linear Perspective



# Linear Perspective



# Today

- How to build a virtual camera?
- How can projective geometry help us?  
What are homogeneous coordinates?

- **How to transform objects using projective geometry?**
- Full projective camera model  
Complex transformations

## Translations in homog. coordinates

- $\mathbb{R}^2$

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

---

- $\mathbb{P}^2$

## Translations in homog. coordinates

- $\mathbb{R}^2$

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

- $\mathbb{P}^2$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ 0 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

## Translations in homog. coordinates

- $\mathbb{R}^2$

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

- $\mathbb{P}^2$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$

## Translations in homog. coordinates

What about points at infinity?

- $\mathbb{P}^2$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x + 0 \\ y + 0 \\ 0 \end{bmatrix}$$

## Rotation in homog. coordinates

- $\mathbb{R}^2$

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\theta)x - \sin(\theta)y \\ \sin(\theta)x + \cos(\theta)y \end{bmatrix}$$


---

- $\mathbb{P}^2$

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta)x - \sin(\theta)y \\ \sin(\theta)x + \cos(\theta)y \\ 1 \end{bmatrix}$$

## Rotation in homog. coordinates

What about points at infinity?

$$\bullet \mathbb{P}^2 \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\theta)x - \sin(\theta)y \\ \sin(\theta)x + \cos(\theta)y \\ 0 \end{bmatrix}$$

## Rotation around point Q

- Rotation around point Q:

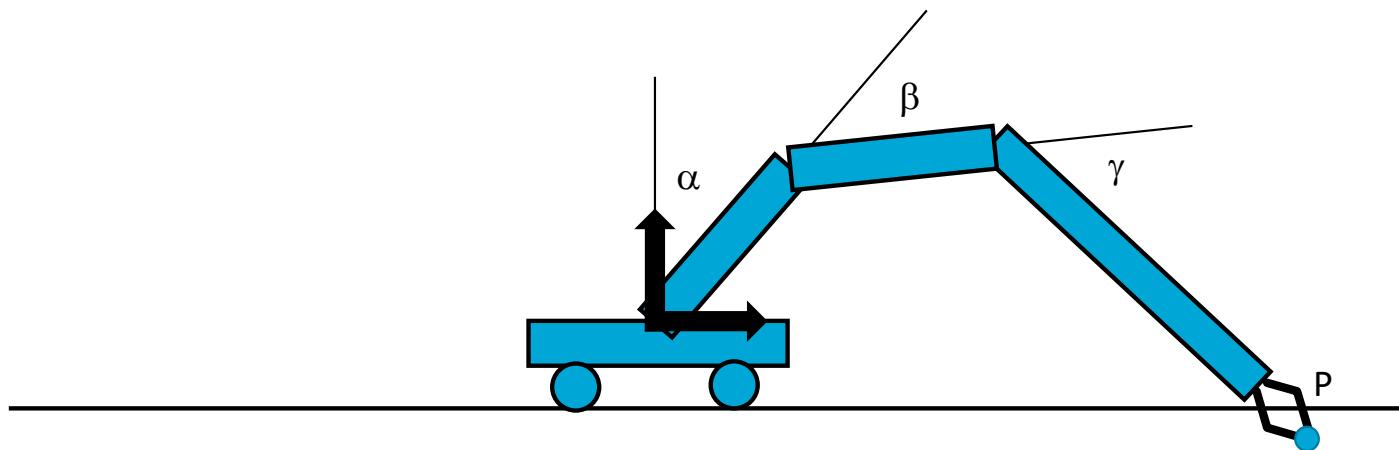
- Translate Q to origin( $T_Q$ ),
- Rotate around origin ( $R_\Theta$ )
- Translate back to Q ( $T_{-Q}$ ).

$$\longrightarrow P' = (T_{-Q}) R_\Theta T_Q P$$

Exercise: Construct an example yourself with a 45 degree rotation and test the resulting matrix to see if it worked.

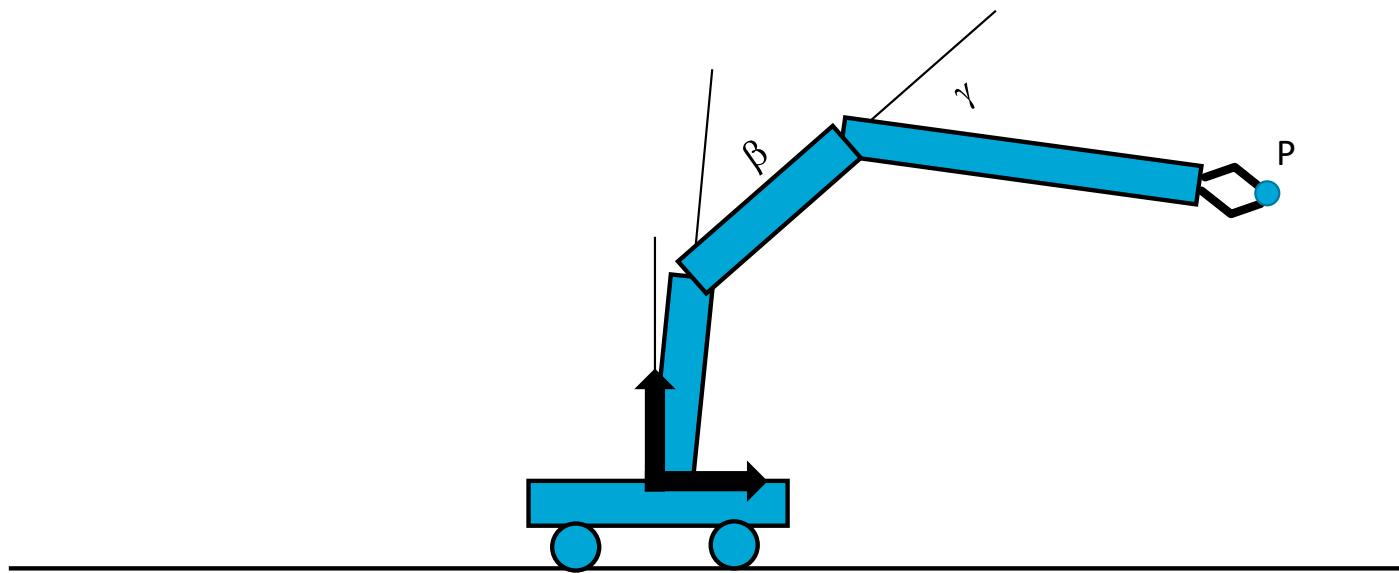
## Today

Change angle  $\alpha$  and calculate the new position of point P with **one** matrix multiplication!

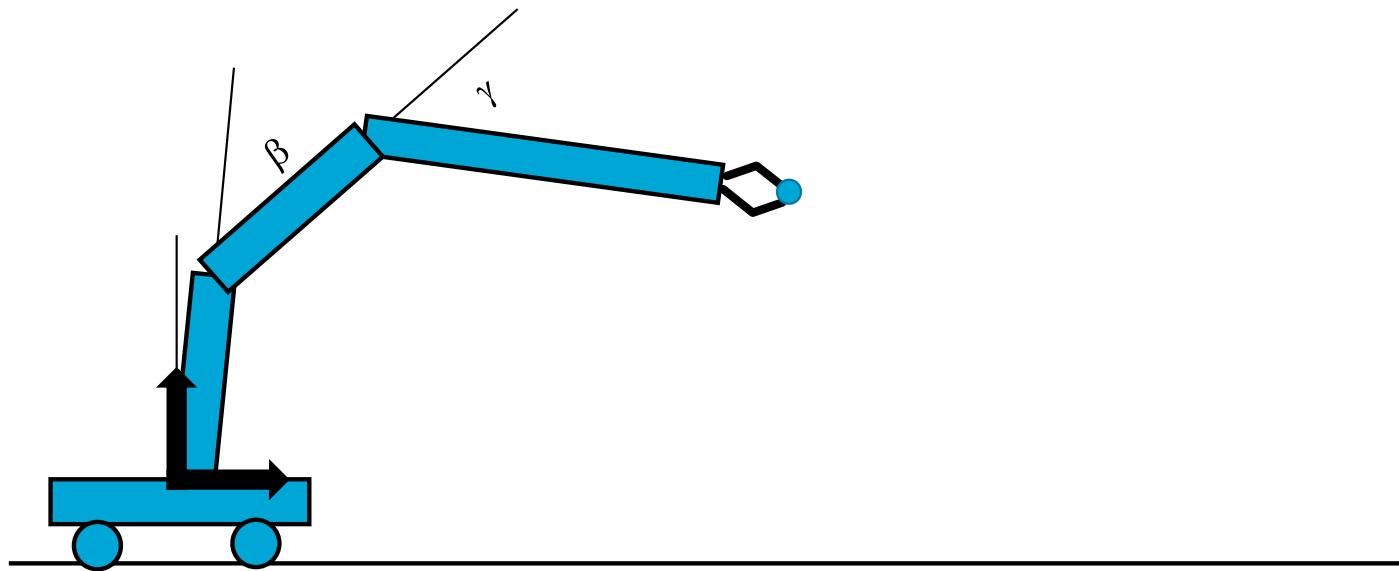


## Today

Change angle  $\alpha$  and calculate the new position of point P with **one** matrix multiplication!

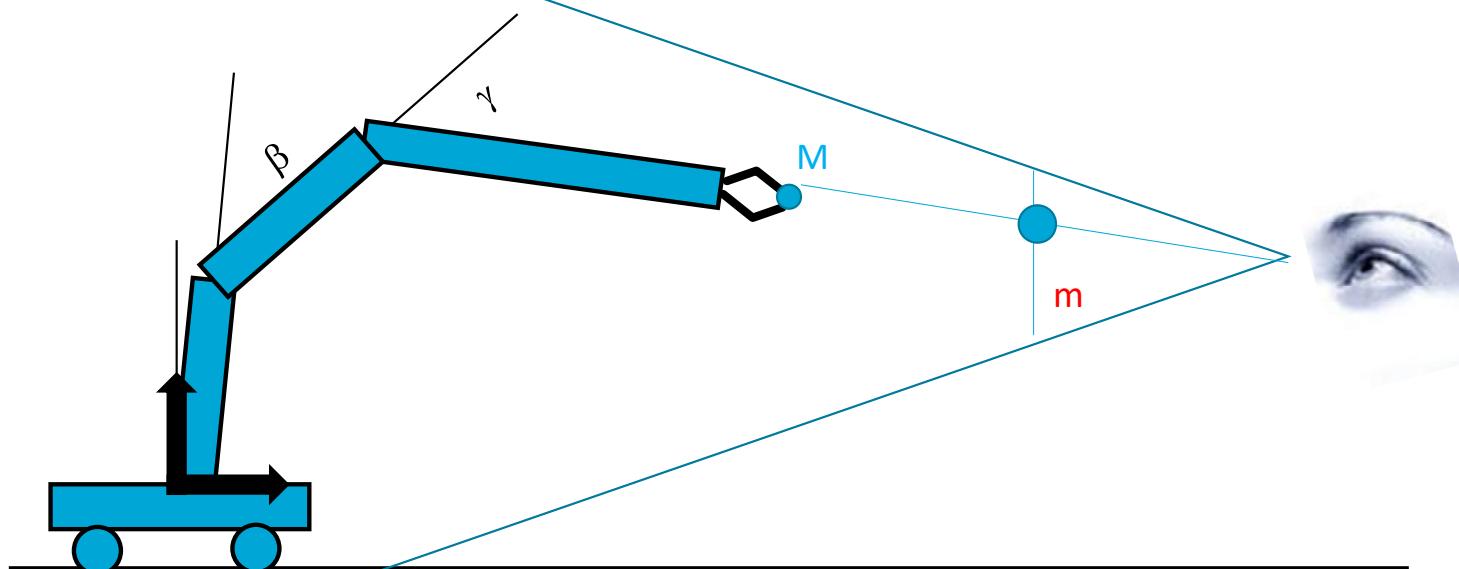


# Today



# Today

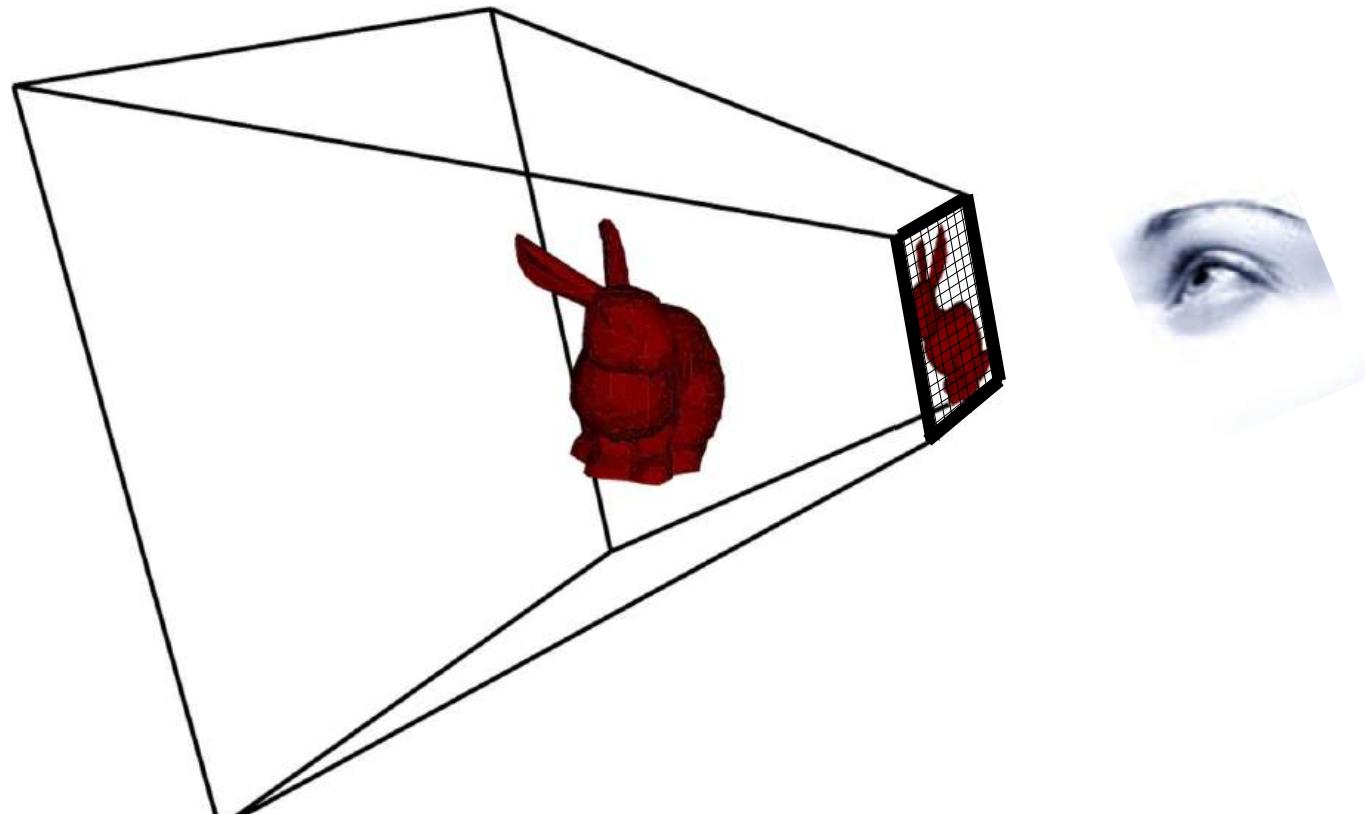
Find matrix  $P$  such that the projected pixel position  $m$  of point  $M$  is  $PM$ .



## Relevant Study Goals for Today

- S1- Explain and compare the structure and properties of standard algorithms and data structures linked to Computer Graphics.
  - We learn about the Matrix Stack for scene/object representations
- S4 Apply mathematical modeling and theory of geometric computations and transformations, object representations, simulation, and encoding.
  - We look at the math behind articulated objects
- S6 Apply the knowledge obtained in this course to problems of other fields
  - We see several application examples in Robotics and Vision

**Can we extend our derivations in 2D also to 3D?**



## 3 Dimensions

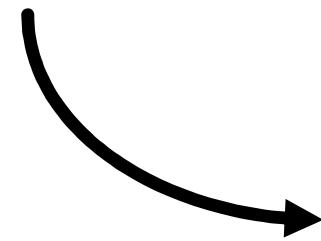
- The same!
- Add a fourth coordinate,  $w$
- All transformations are 4x4 matrices

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

# Translations in 3D

## Translations in 3D

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$



$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ z' = z + wt_z \\ w' = w \end{cases}$$

## Rotations in 3D

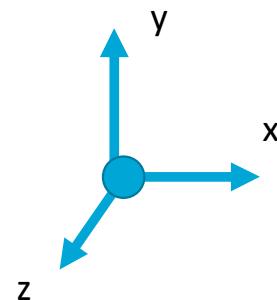
- Rotation = axis and angle
- Rotations around x,y,z axis are simple matrices
  - All axes can be achieved by combining these  
(it is a bit cumbersome though...)

## Rotation about z-axis

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

*Axis z not changing*

**Quick verification :** rotation of  $\pi/2$   
Should change  $x$  in  $y$ , and  $y$  in  $-x$



$$R_z\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Rotation about x-axis

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

*Axis x not changing*

**Quick verification :** rotation of  $\pi/2$   
Should change  $y$  in  $z$ , and  $z$  in  $-y$

$$R_x\left(\frac{\pi}{2}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Rotation about y-axis

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

*Axis y not changing*

**Quick verification :** rotation of  $\pi/2$   
 Should change  $z$  in  $x$ , and  $x$  in  $-z$

$$R_y\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

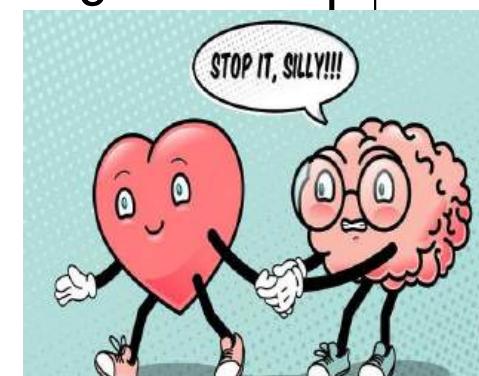
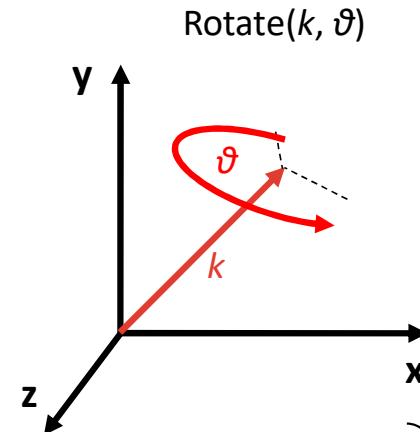
## Rotation around ( $k_x, k_y, k_z$ )

- Rodrigues Formula

$$\begin{pmatrix}
 k_xk_x(1-c)+c & k_zk_x(1-c)-k_zs & k_xk_z(1-c)+k_ys & 0 \\
 k_yk_x(1-c)+k_zs & k_zk_x(1-c)+c & k_yk_z(1-c)-k_xs & 0 \\
 k_zk_x(1-c)-k_ys & k_zk_x(1-c)-k_xs & k_zk_z(1-c)+c & 0 \\
 0 & 0 & 0 & 1
 \end{pmatrix}$$

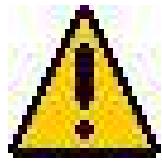
DO NOT LEARN THIS BY HEART!

where  $c = \cos \vartheta$  &  $s = \sin \vartheta$

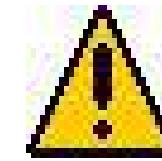


## Attention !

- Matrix Multiplication is not commutative
- The order of transformations is important
  - Rotation then translation != transl. then rotation

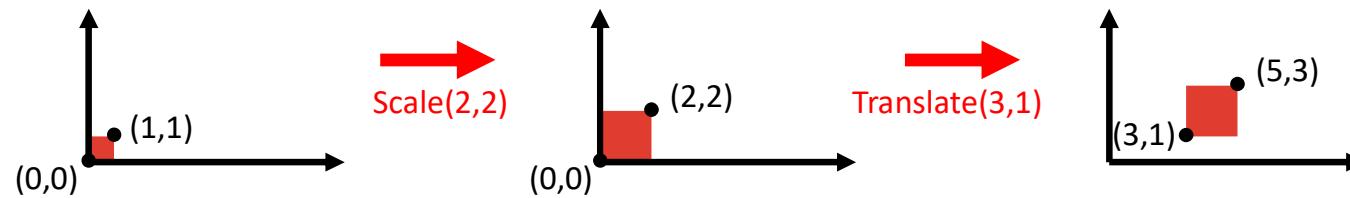


Source of bugs...



## Example

Scaling + Translation

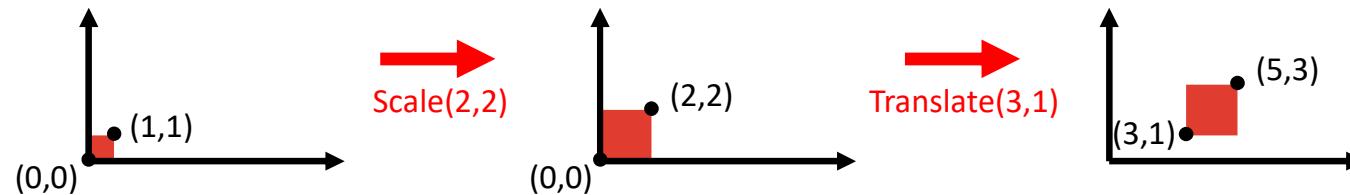


Matrix combination:  $p' = T(S p) = TS p$

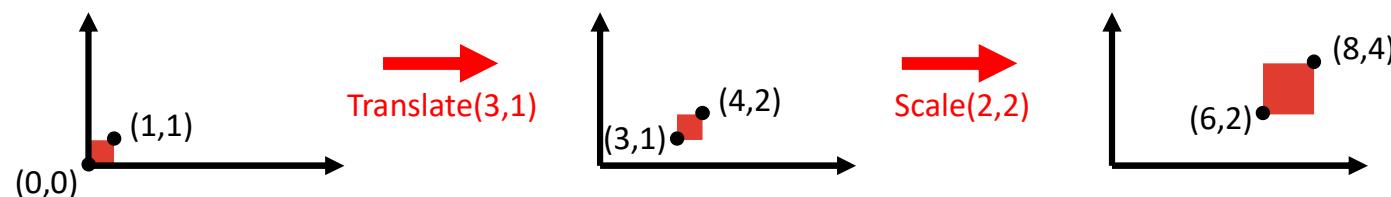
$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

## NOT commutative

Scaling + translation:  $p' = T(S p) = TS p$



Translation + scaling:  $p' = S(T p) = ST p$



## NOT commutative

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

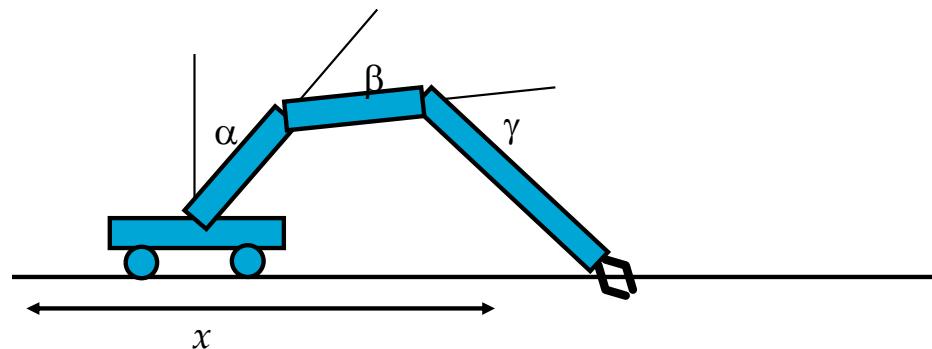
## NOT commutative

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

## Complex Objects

- Objects are often defined via many components
  - E.g., wheels of cars, fingers on hands on arms ...
- Concatenate matrices to place objects
  - Changing one matrix affects everyone hereafter



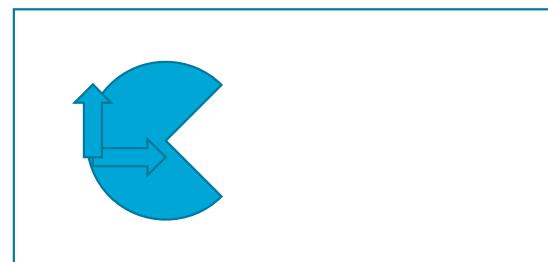
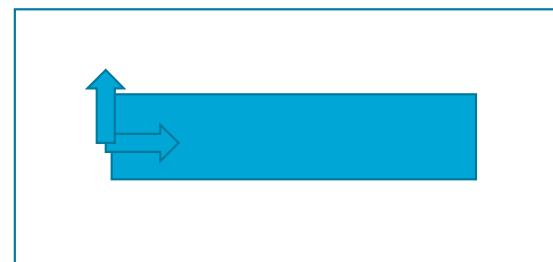
# Complex Objects

Example:

Robot arm consisting of two parts:

The arm itself and a hand

Both are designed independently  
and are at the origin (shown below)



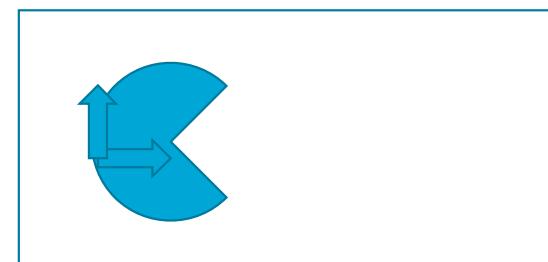
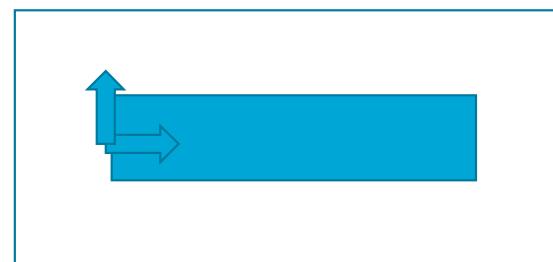
# Why concatenate operations?

Example:

Robot arm consisting of two parts:

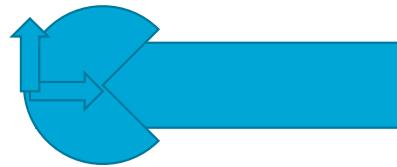
The arm itself and a hand

Both are designed independently  
and are at the origin (shown below)



## Why concatenate operations?

- Drawing first the arm, then the hand, you get:



- That does not look right!
- Instead: Produce matrix that when applied to the object shifts it to the wanted location

## Why concatenate operations?

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)



## Why concatenate operations?

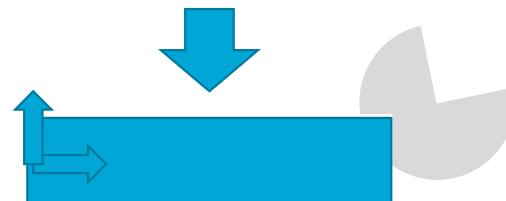
- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)



## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**Shoulder**)
  - Apply  $S$  to all vertices of arm

Resulting positions  
After applying the object  
vertices to matrix  $S$



## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**Shoulder**)
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **Joint of the hand**)
  - $J=S\ T$



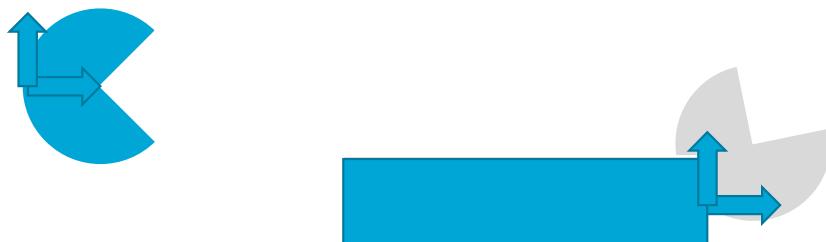
## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**Shoulder**)
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **Joint of the hand**)
  - $J=S\ T$



## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**Shoulder**)
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **Joint of the hand**)
  - $J=S\ T$



## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**Shoulder**)
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **Joint of the hand**)
  - $J=S\ T$



## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**S**houlder)
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **J**oint of the hand)
  - $J=S T$
  - $R$ = Rotation (for **H**and)
  - $H:=J R$



## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**Shoulder**)
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **Joint of the hand**)
  - $J=S T$
  - $R$ = Rotation (for **Hand**)
  - $H:=J R$
  - Apply  $H$  to all vertices of the hand



## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**Shoulder**)
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **Joint of the hand**)
  - $J=S T$
  - $R$ = Rotation (for **Hand**)
  - $H:=J R$
  - Apply  $H$  to all vertices of the hand



## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**S**houlder)
  - **$S:=SN$ , with  $N$  a new rotation matrix**
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **J**oint of the hand)
  - $J=S T$
  - $R$ = Rotation (for **H**and)
  - $H:=J R$
  - Apply  $H$  to all vertices of the hand



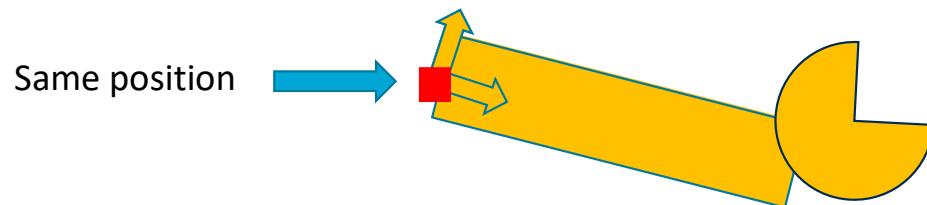
## Why concatenate operations?

- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**S**houlder)
  - **$S:=SN$ , with  $N$  a new rotation matrix**
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **J**oint of the hand)
  - $J=S T$
  - $R$ = Rotation (for **H**and)
  - $H:=J R$
  - Apply  $H$  to all vertices of the hand

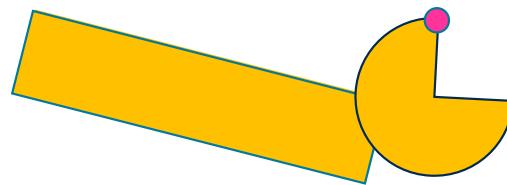


## Why concatenate operations?

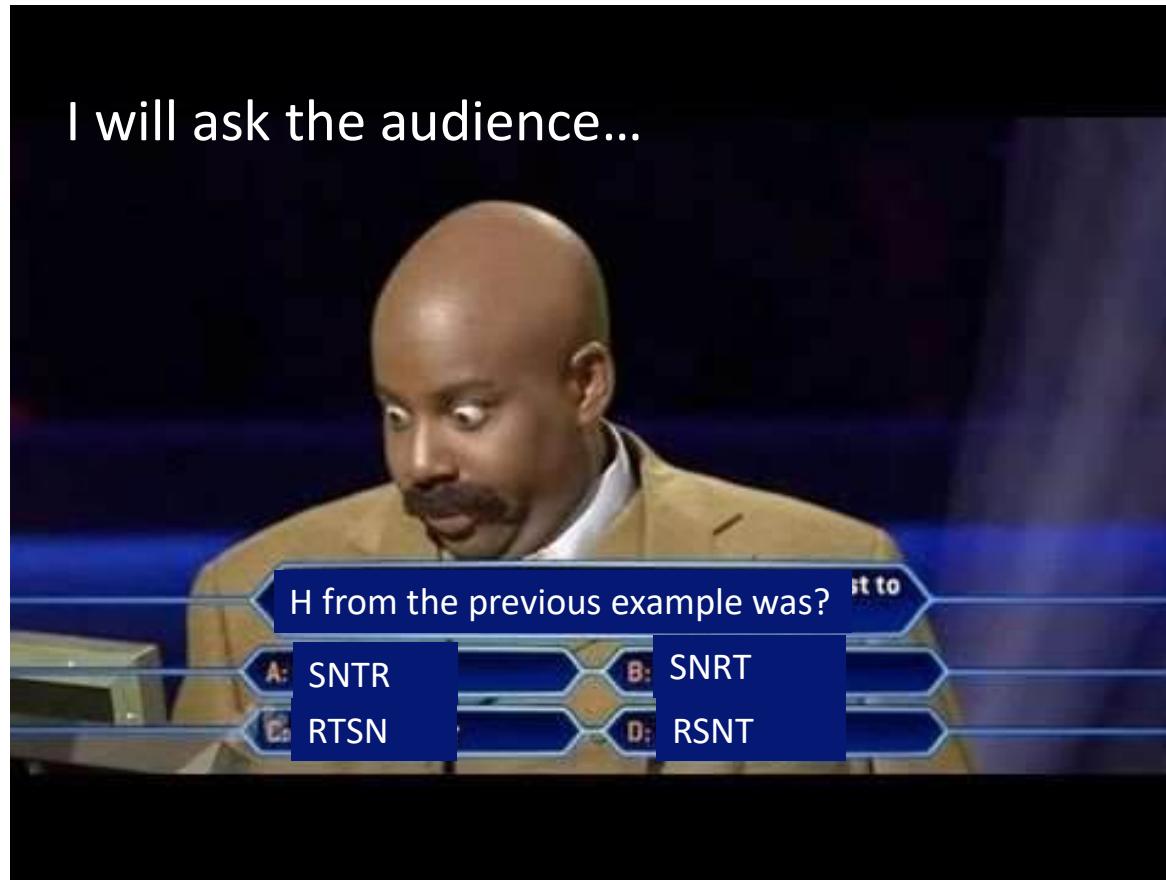
- Concatenate and apply matrices
  - $S$ :=Translation matrix to position of arm (**S**houlder)
  - **$S:=SN$ , with  $N$  a new rotation matrix**
  - Apply  $S$  to all vertices of arm
  - $T$ :=translation along arm (to the **J**oint of the hand)
  - $J=S T$
  - $R$ = Rotation (for **H**and)
  - $H:=J R$
  - Apply  $H$  to all vertices of the hand



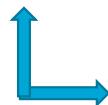
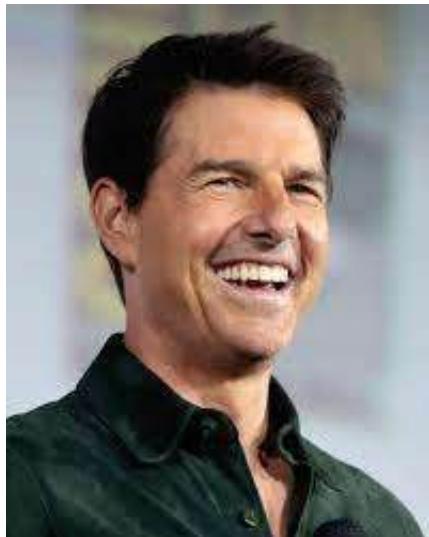
## How to remember the order?



## How to remember the order?



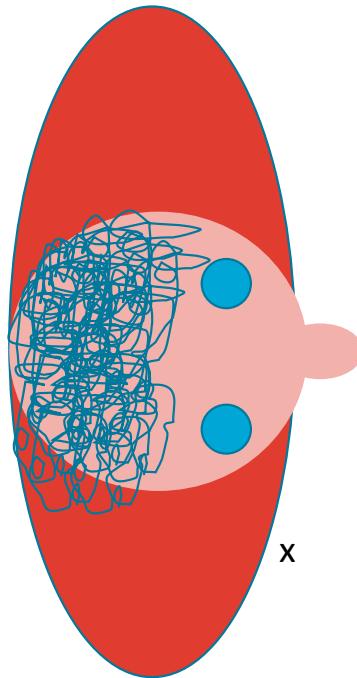
## How to remember the order?



Tom/Penelope “Cross”

wikipedia

## How to remember the order?

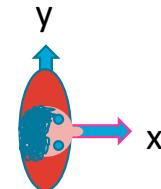


Tom/Penelope “Cross”

## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

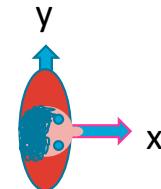


Tom/Penelope “Cross”

## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.



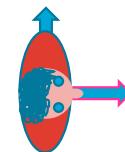
Tom/Penelope “Cross”

## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

T=Translate(2,0)



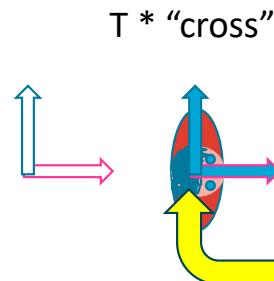
Tom/Penelope “Cross”

## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

T=Translate(2,0)



For example, if we apply the point  $(0,0,1)^t$  to T, we end up here.

## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

$T * \text{"cross"}$

T=Translate(2,0)



## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

$T * \text{"cross"}$

$T = \text{Translate}(2,0)$

$R = \text{Rotate}(45)$



## How to remember the order?

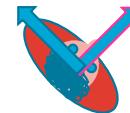
Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

$T * R * \text{"cross"}$

T=Translate(2,0)

R=Rotate(45)



## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

T=Translate(2,0)

R=Rotate(45)



## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

T=Translate(2,0)

R=Rotate(45)



Let's apply both operations to a point:

$$T^*R^*(0,2,1)^t$$

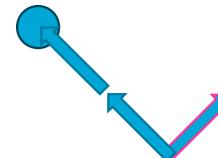
## How to remember the order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.

If we depict the origin by “cross”, matrix operations will move “cross” (and in turn the origin) around.

T=Translate(2,0)

R=Rotate(45)



Let's apply both operations to a point:

$$T^*R^*(0,2,1)^t$$

The point lands in the location that we would expect, if the new frame described the origin of our space

## How to remember the order?

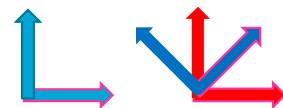
T=translate (2,0)



## How to remember the order?

T=translate (2,0)

R=rotate(45)

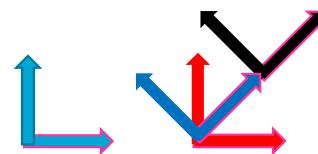


## How to remember the order?

T=translate (2,0)

R=rotate(45)

T=translate (1,0)



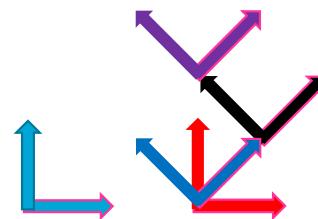
## How to remember the order?

T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)



## How to remember the order?

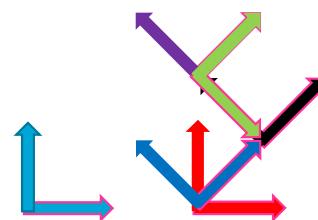
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



## How to remember the order?

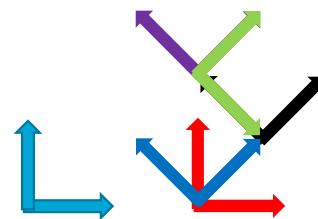
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTTR

Multiply left to right in the “playthrough” order

## How to remember the order?

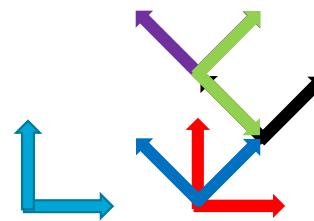
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRT<sub>1</sub>T<sub>2</sub>R

Multiply left to right in the “playthrough” order

## How to remember the order?

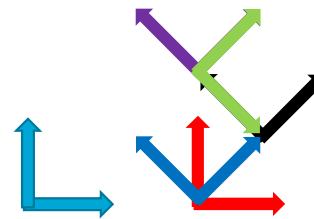
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTTR

Where is TRTTTR  $(0,0,1)^t$  ?

## How to remember the order?

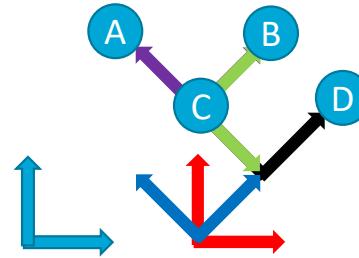
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be:  $\textcolor{red}{T}\textcolor{blue}{R}\textcolor{teal}{T}\textcolor{magenta}{T}\textcolor{green}{R}$

Where is  $\textcolor{red}{T}\textcolor{blue}{R}\textcolor{teal}{T}\textcolor{magenta}{T}\textcolor{green}{R} (0,0,1)^t$  ?

## How to remember the order?

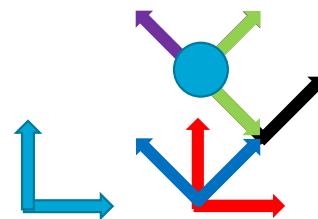
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTTR

Where is TRTTTR  $(0,0,1)^t$  ?

## How to remember the order?

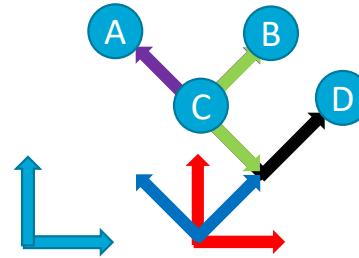
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be:  $\textcolor{red}{T}\textcolor{blue}{R}\textcolor{teal}{T}\textcolor{magenta}{T}\textcolor{green}{R}$

Where is  $\textcolor{red}{T}\textcolor{blue}{R}\textcolor{teal}{T}\textcolor{magenta}{T}\textcolor{green}{R} (0,1,1)^t$  ?

## How to remember the order?

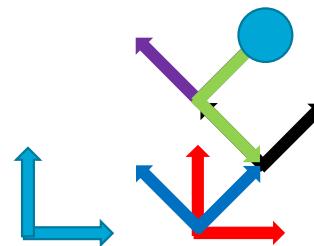
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTTR

Where is TRTTTR  $(0,1,1)^t$  ?

## How to remember the order?

T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)

(**TRTTTR**) P

Multiply left to right in the “playthrough” order

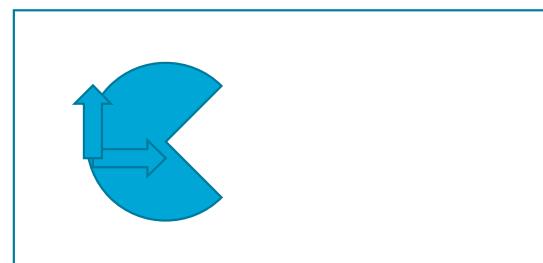
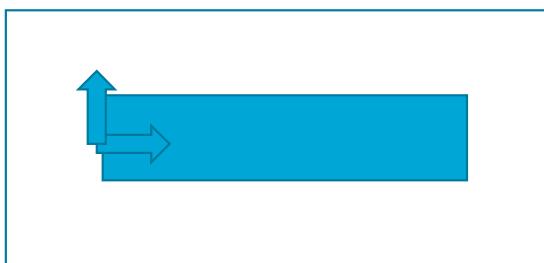
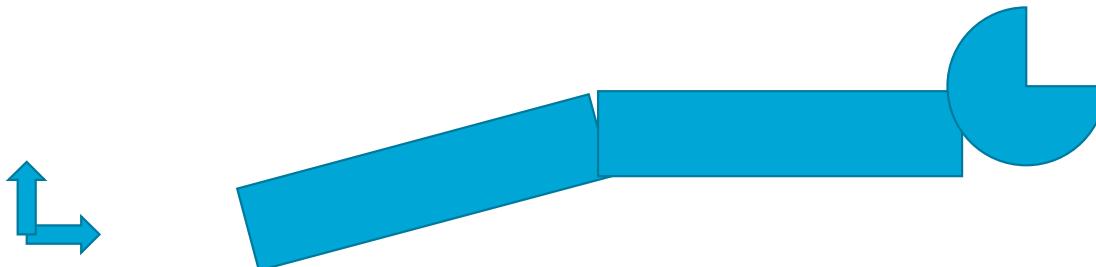
Why does this work? P multiplies mathematically from the right!

## Local vs. Global interpretation

- Two interpretations of the same mathematics:
  - Points multiplied from the right:  
This transforms the points with respect to a GLOBAL coordinate system.
  - Multiplying matrices together on the left:  
This changes the origin/coordinate system, i.e., a LOCAL coordinate system is established.

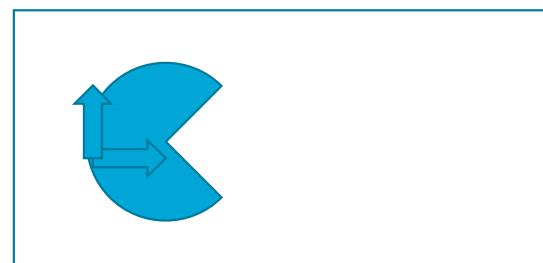
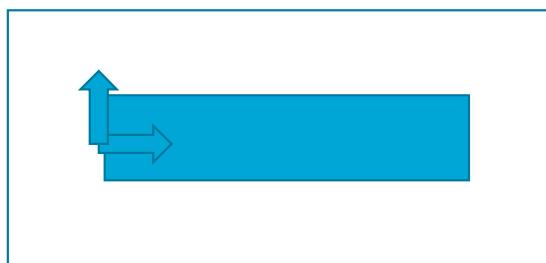
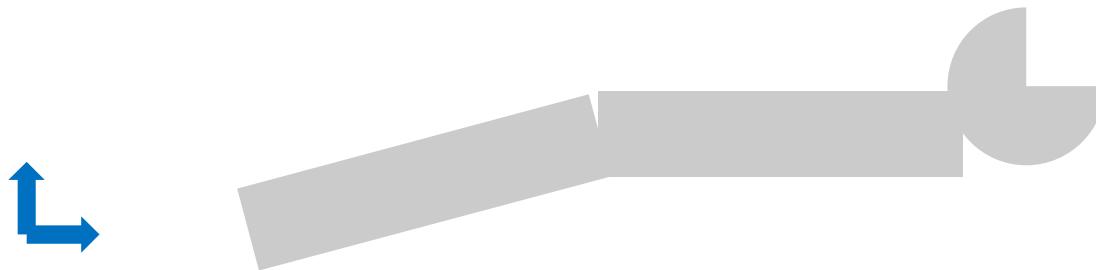
## Local Interpretation

- Build complex object dependencies



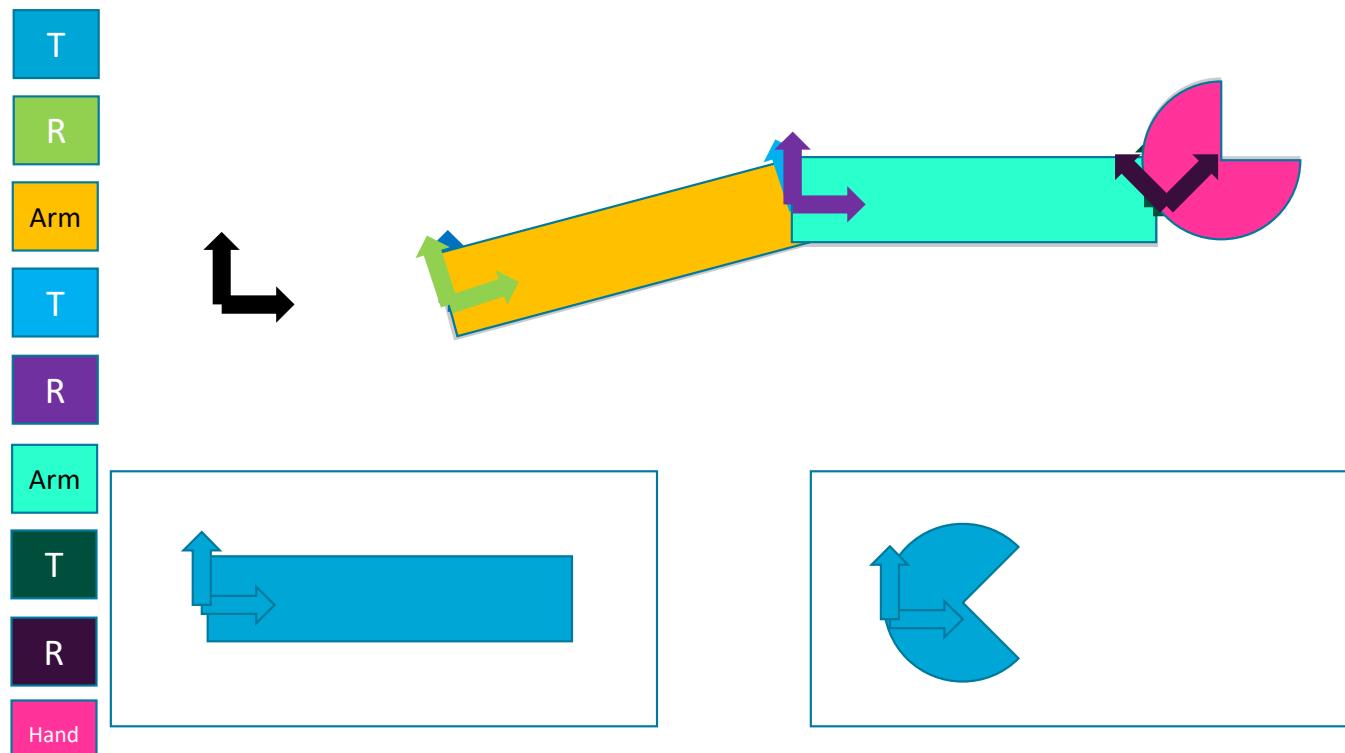
## Local Interpretation

- Build complex object dependencies



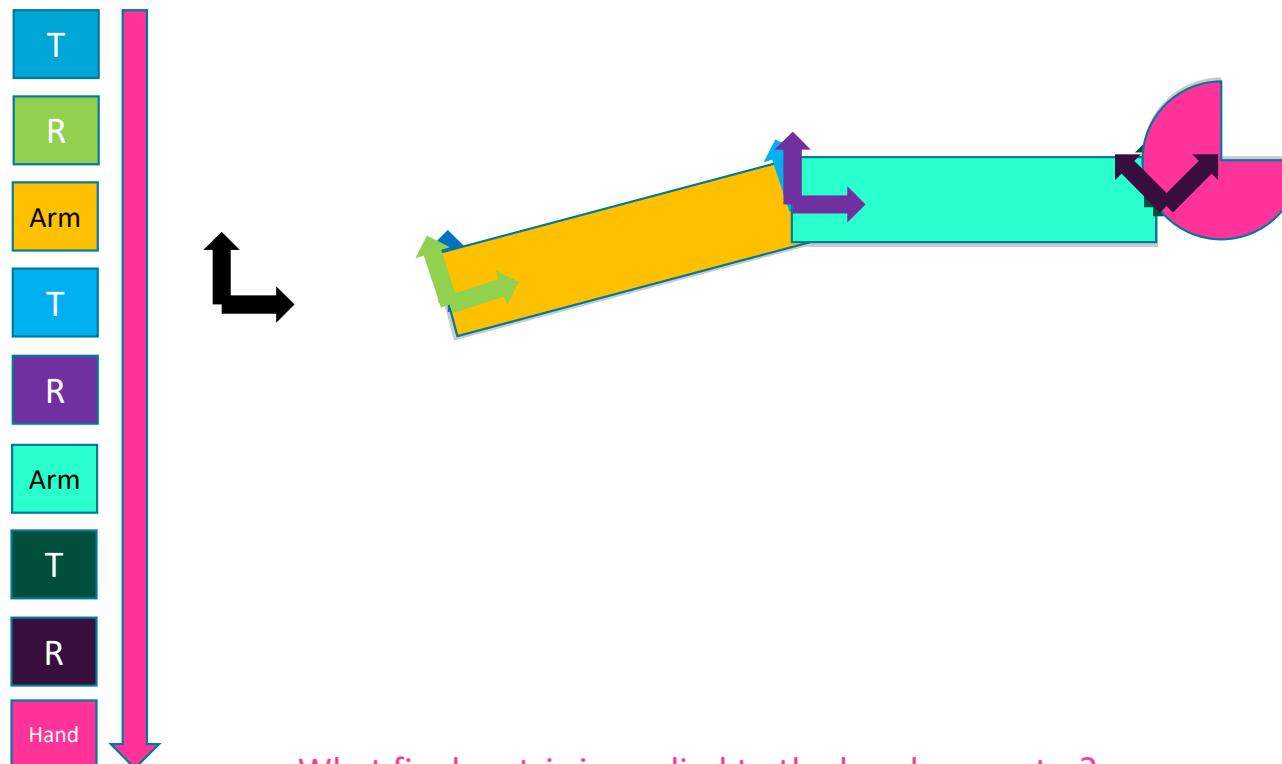
## Local Interpretation

- Build complex object dependencies



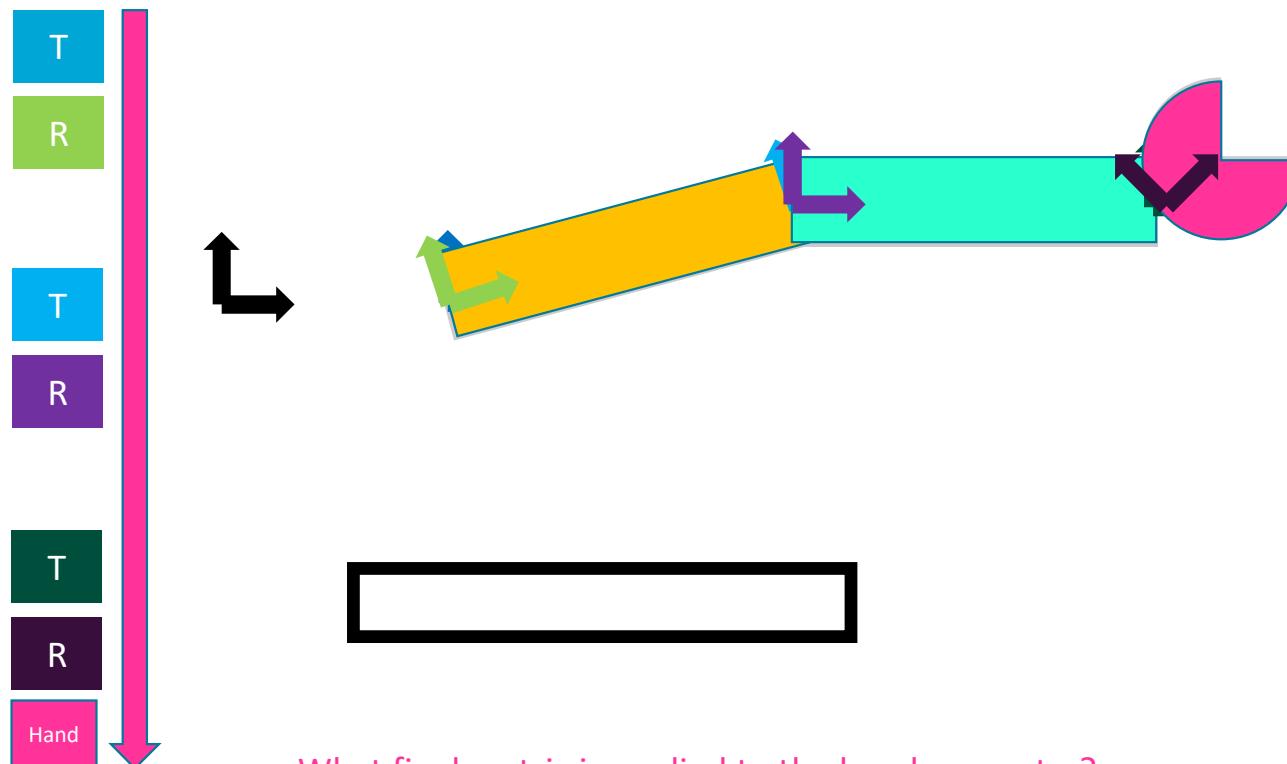
## Local Interpretation

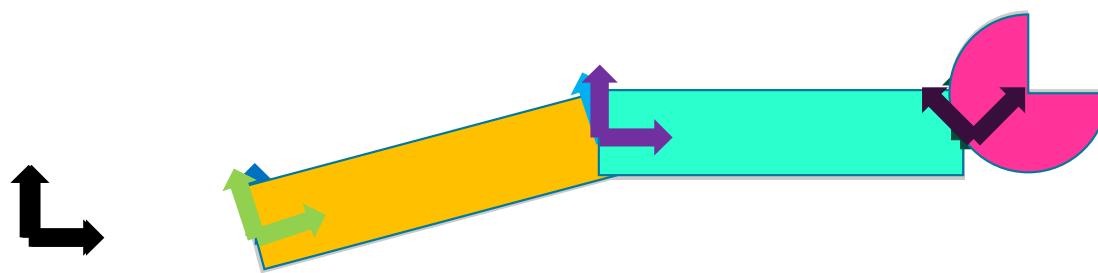
- Build complex object dependencies



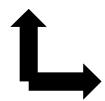
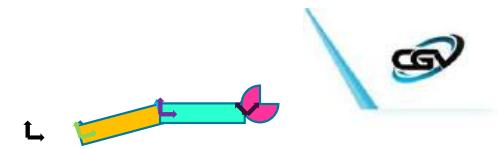
## Local Interpretation

- Build complex object dependencies

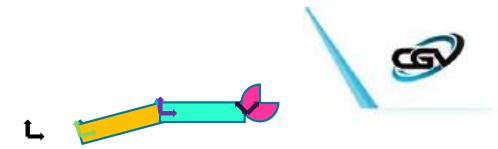




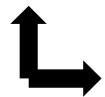
T	R	T	R	T	R
---	---	---	---	---	---



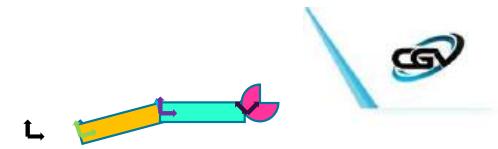
T R T R T R



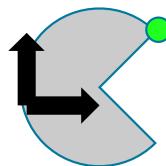
## Global Interpretation

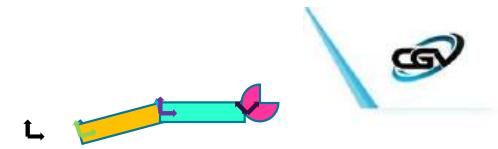


T R T R T R

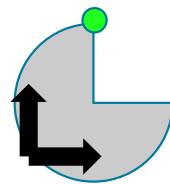


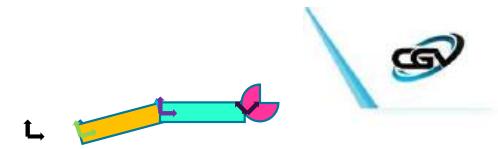
# Global Interpretation



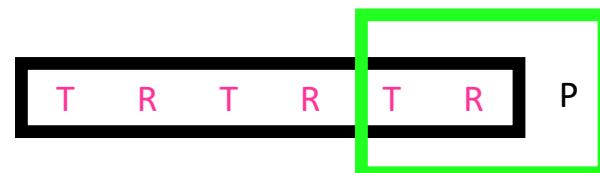


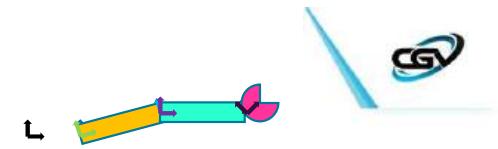
# Global Interpretation



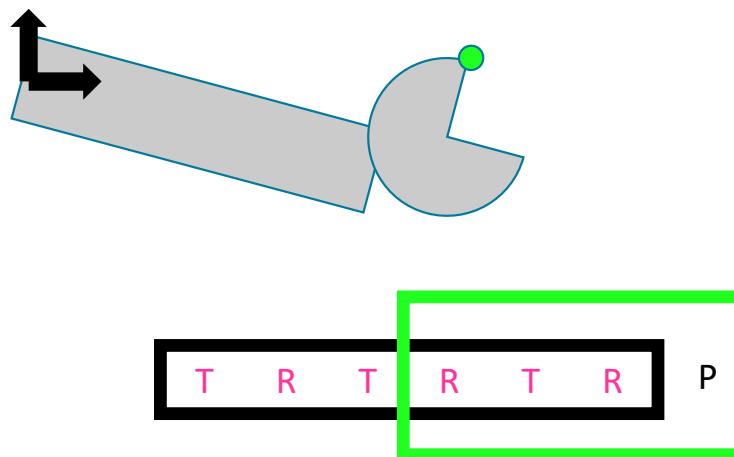


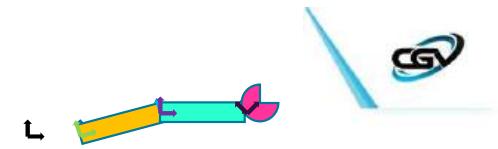
## Global Interpretation



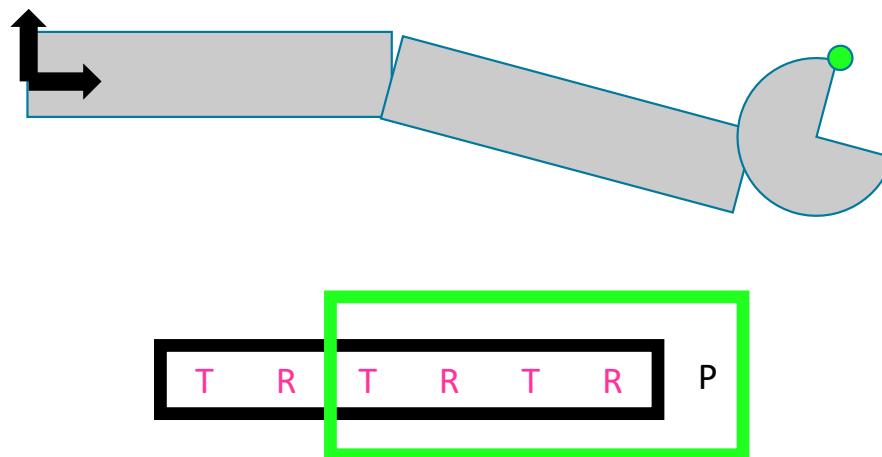


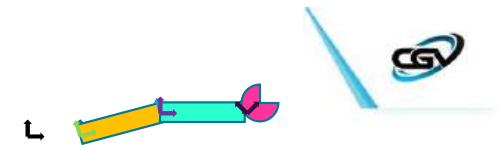
## Global Interpretation



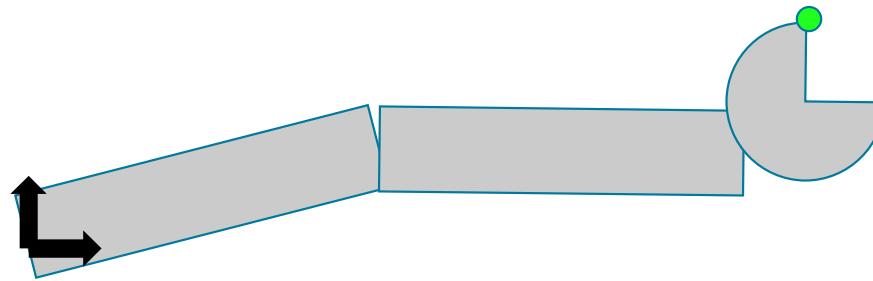


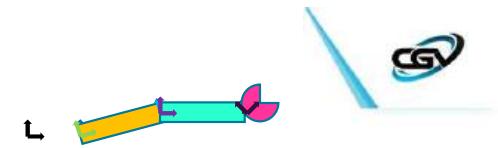
## Global Interpretation



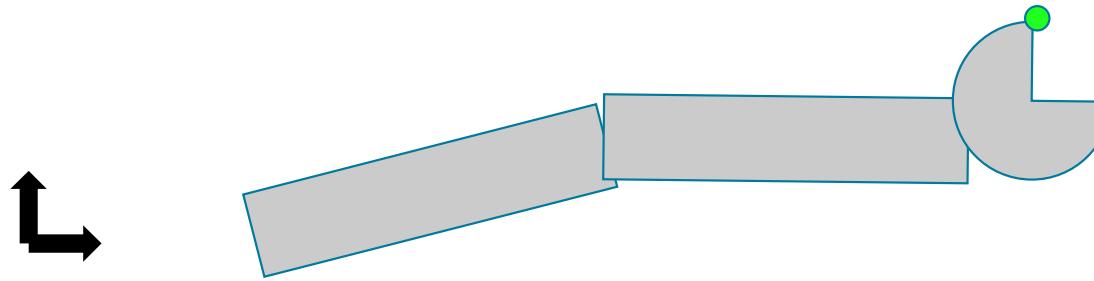


## Global Interpretation

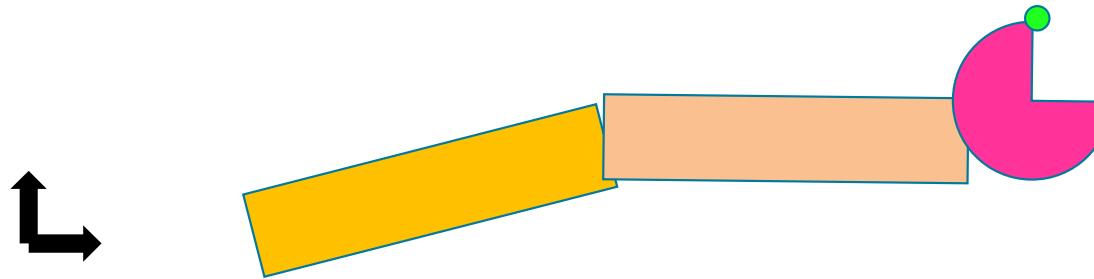




## Global Interpretation



## Global Interpretation



The point does end up in the  
expected location

Please note: this is ONE matrix, which makes it very efficient to apply to all vertices of the hand.



## Recap Video

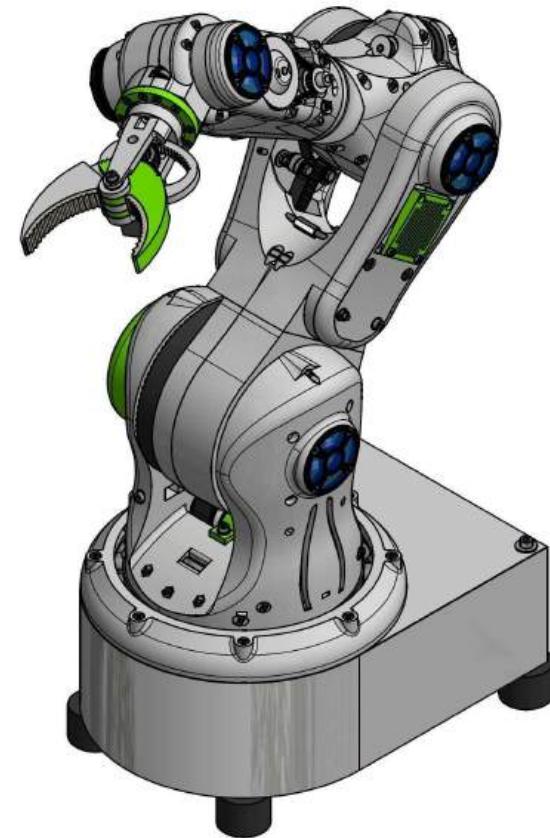
Thanks to Tim Huisman

CSE2215 Computer Graphics

# Right-To-Left or Left-To-Right?

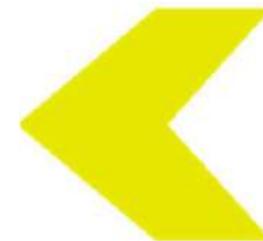
## Let's try it out!

- Professional Robot Arm:

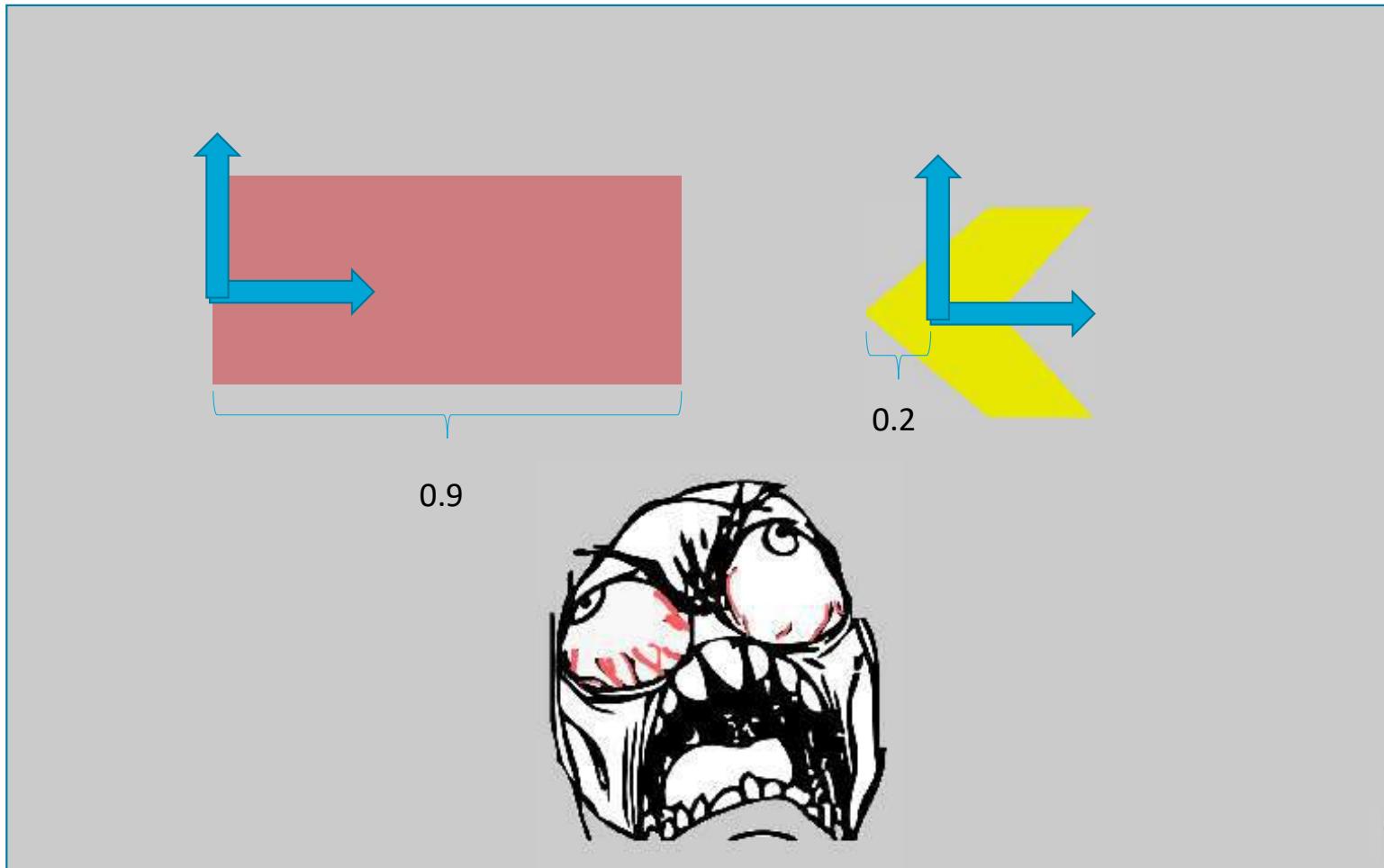


## Let's try it out!

- Professional Robot Arm:



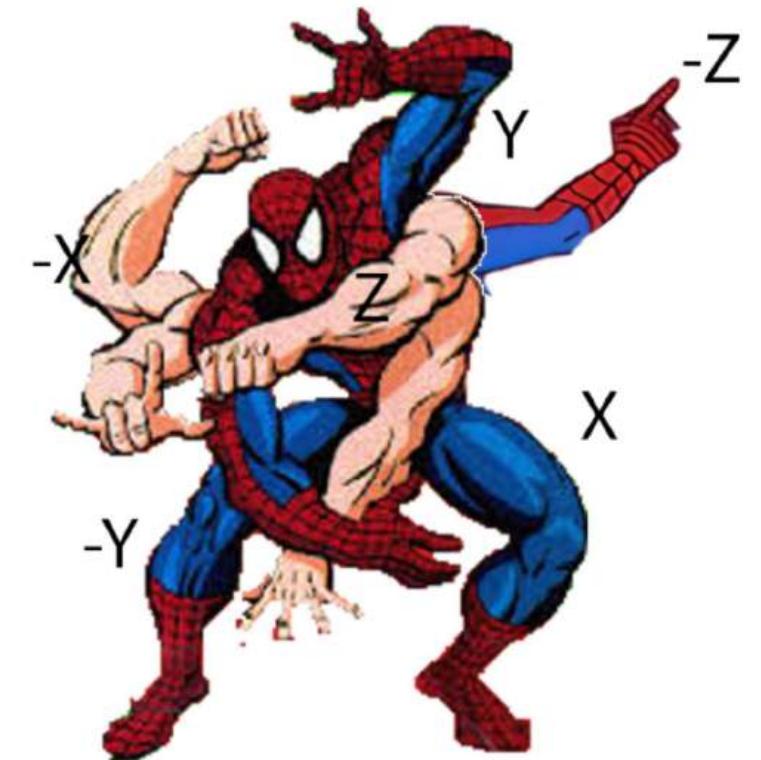
## Let's try it out!



## Questions?

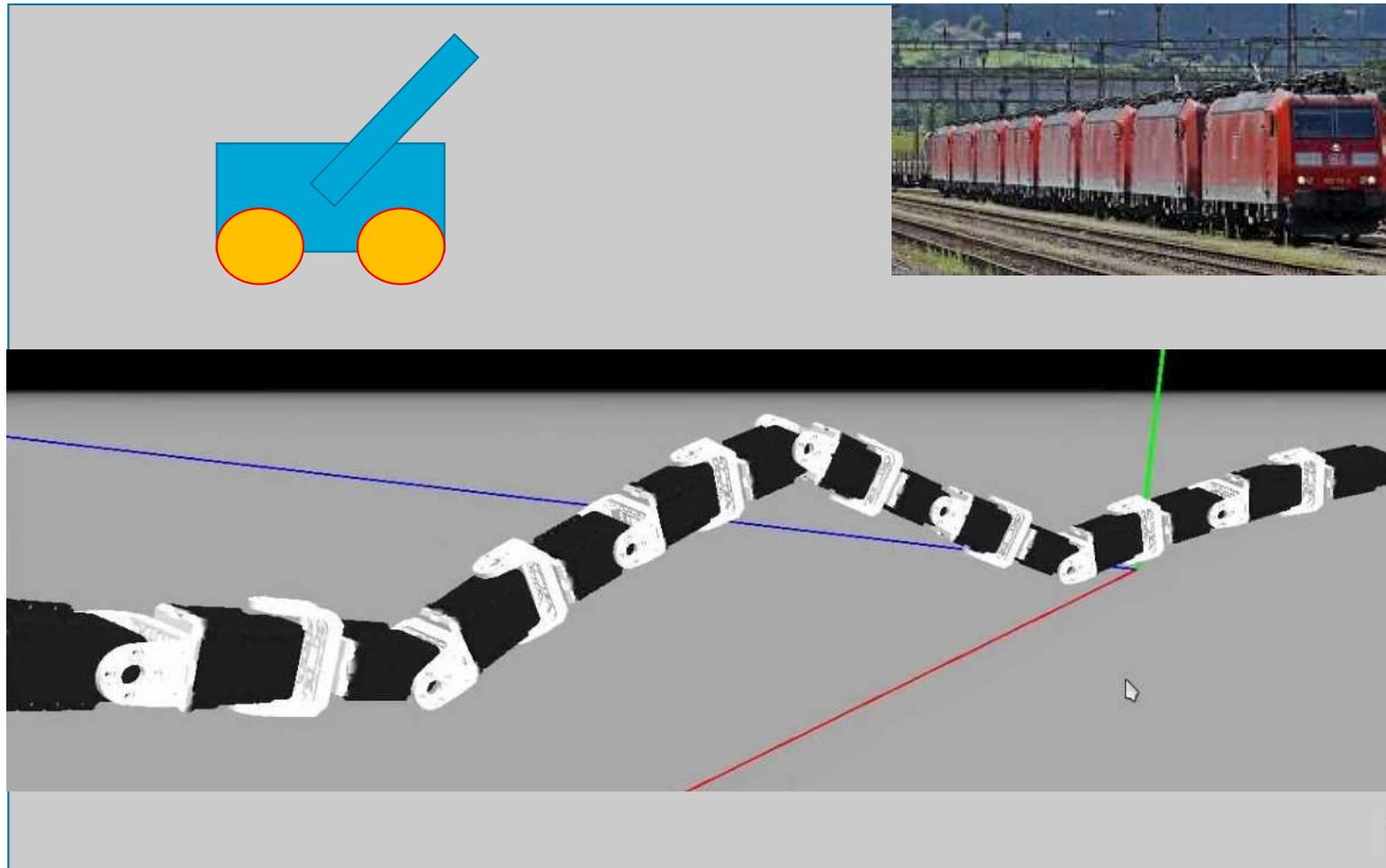


Before CSE2215

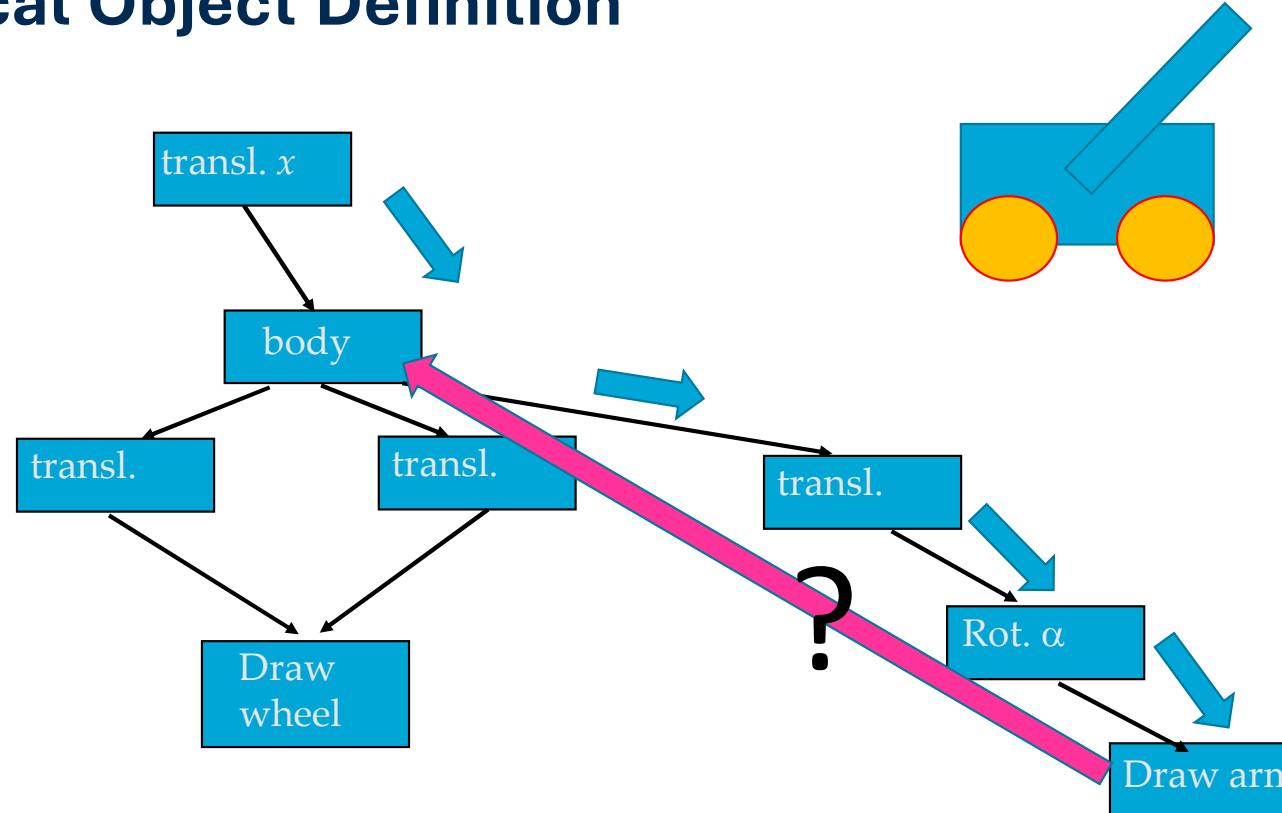


After CSE2215

## Other examples?



## Hierarchical Object Definition



Common representation of objects in form of a tree  
 Geometry is reused (e.g., only one wheel is stored)

## Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
```

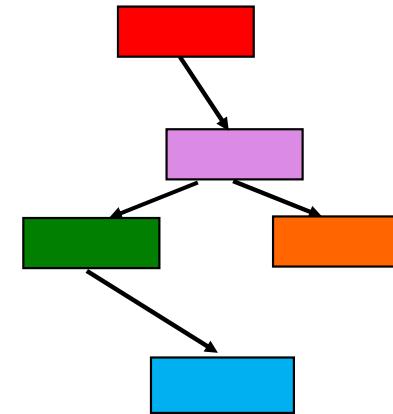
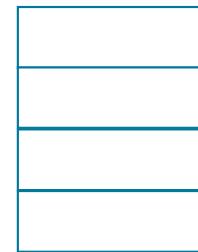
```
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```

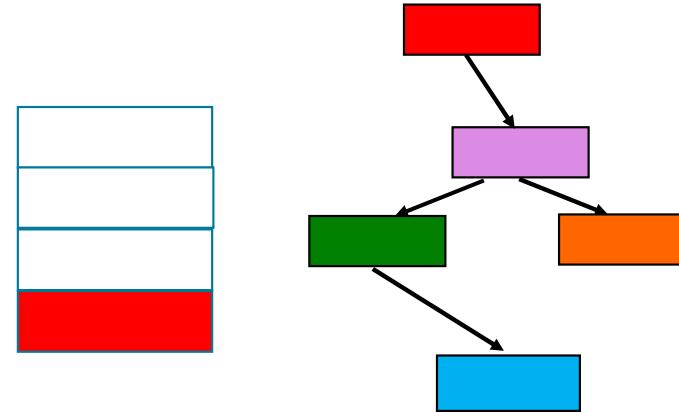


## Excursion: Walking over a tree

Depth-first tree traversal  
using a stack:

stack.push(root)

```
while (!stack.empty())
{
    node=stack.pop()
    process(node)//do what you need to do
    if (node.hasChildren()) stack.push(node.children())
}
```



## Excursion: Walking over a tree

Depth-first tree traversal  
using a stack:

```
stack.push(root)
```

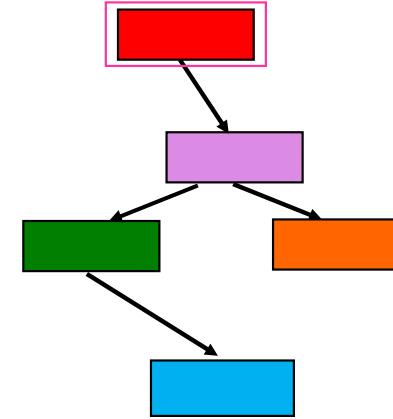
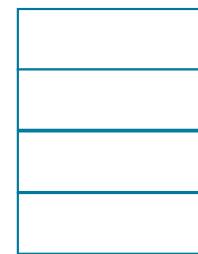
```
while (!stack.empty())  
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```



## Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
```

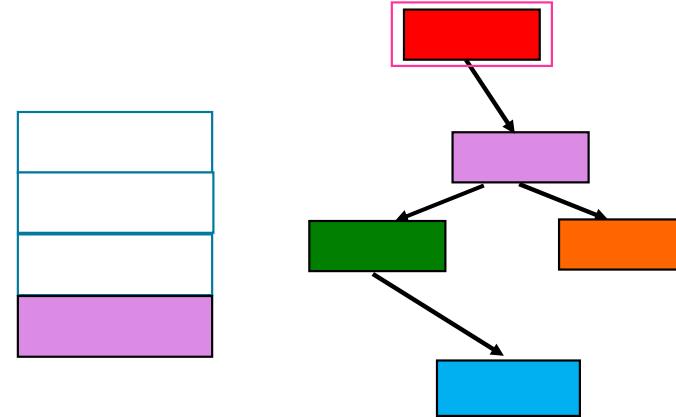
```
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```

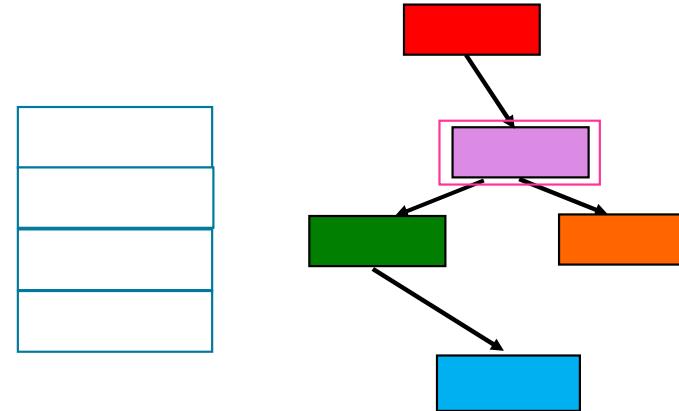


## Excursion: Walking over a tree

Depth-first tree traversal  
using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
{
    node=stack.pop()
    process(node)//do what you need to do
    if (node.hasChildren()) stack.push(node.children())
}
```



## Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
```

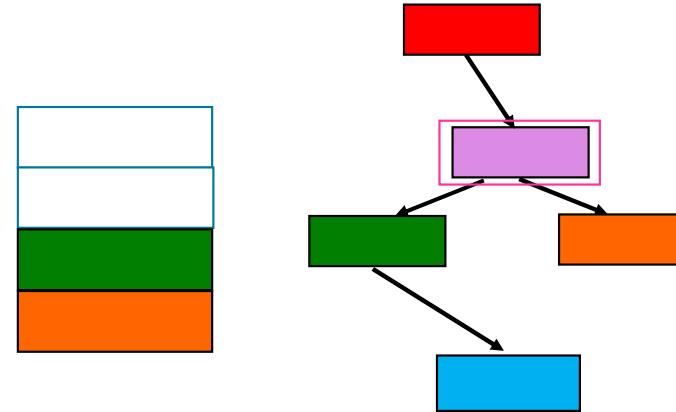
```
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```



## Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
```

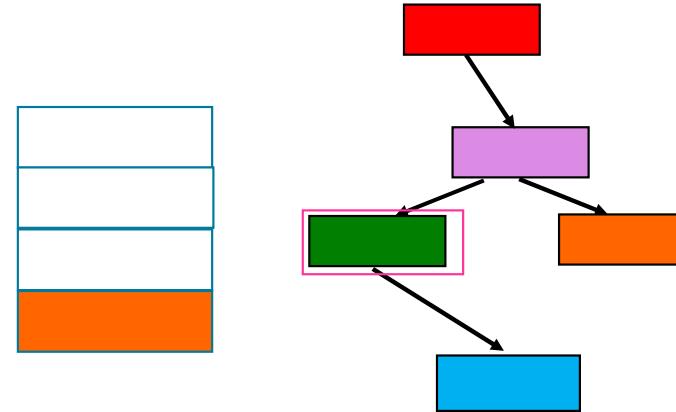
```
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```



## Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
```

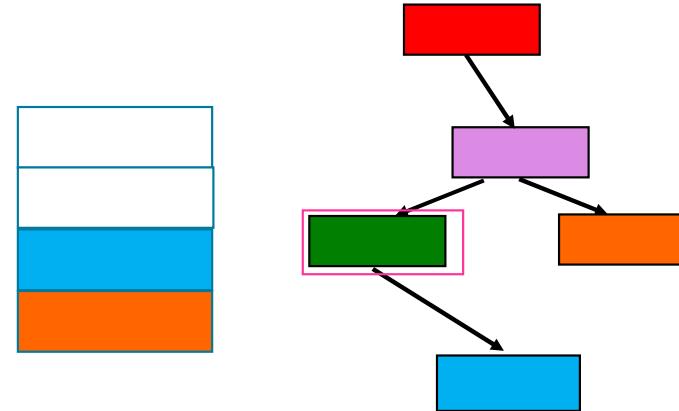
```
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```



## Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
```

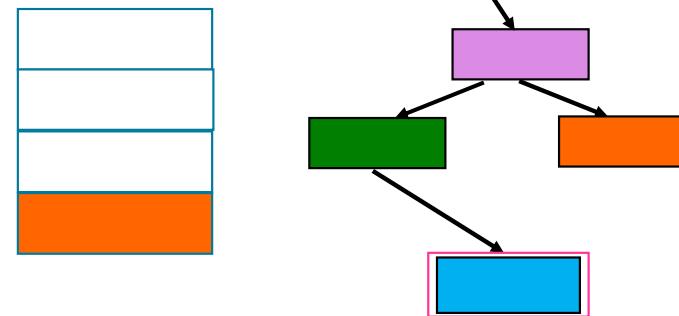
```
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```



## Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
```

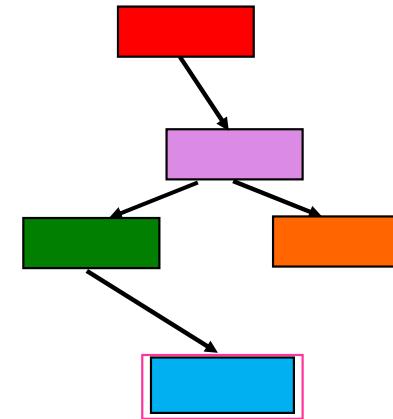
```
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```



## Excursion: Walking over a tree

Depth-first tree traversal  
using a stack:

```
stack.push(root)
```

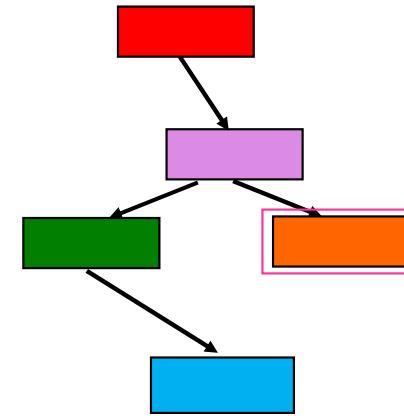
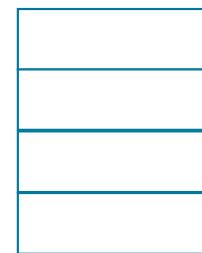
```
while (!stack.empty())  
{
```

```
    node=stack.pop()
```

```
    process(node)//do what you need to do
```

```
    if (node.hasChildren()) stack.push(node.children())
```

```
}
```

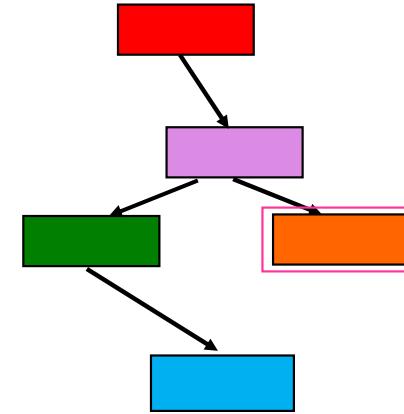
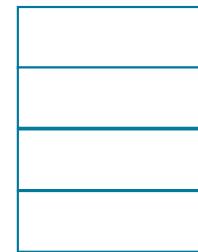


## Excursion: Walking over a tree

Depth-first tree traversal  
using a stack:

```
stack.push(root)
```

```
while (!stack.empty())
{
    node=stack.pop()
    process(node)//do what you need to do
    if (node.hasChildren()) stack.push(node.children())
}
```

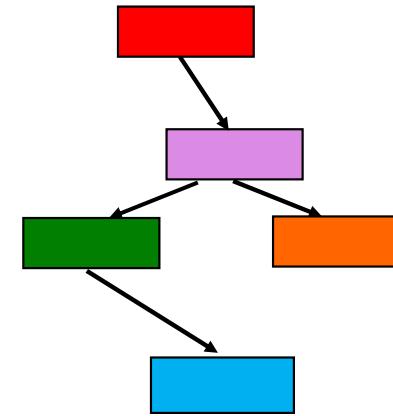
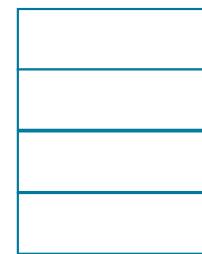


## Excursion: Walking over a tree

Depth-first tree traversal  
using a stack:

stack.push(root)

```
while (!stack.empty())
{
    node=stack.pop()
    process(node)//do what you need to do
    if (node.hasChildren()) stack.push(node.children())
}
```

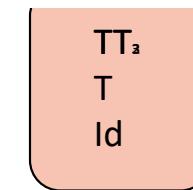
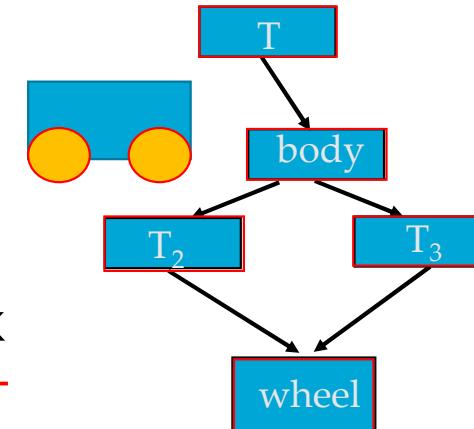
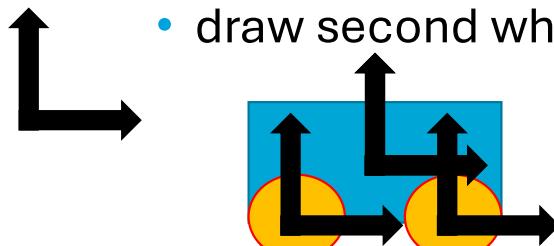


## Hierarchical Objects

- Walk over tree using depth-first traversal
- Keep a “matrix” stack that maintains the concatenations (multiplications) of matrices along the way

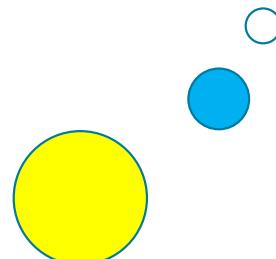
## Matrix Stack

- Parallel to walking over the tree:
- Keep previous matrices on a stack
  - $T$  (translation by  $x$ ) **pushMatrix idT=T**
  - draw robot body
  - $T_2$  (translation to center of 1st wheel) **pushMatrix TT<sub>2</sub>**
  - draw first wheel as circle of center (0,0)
  - return to  $T$ : **popMatrix**
  - $T_3$  ( $T_3$  translation to center of 2nd wheel) **pushMatrix TT<sub>3</sub>**
  - draw second wheel as circle of center (0,0)



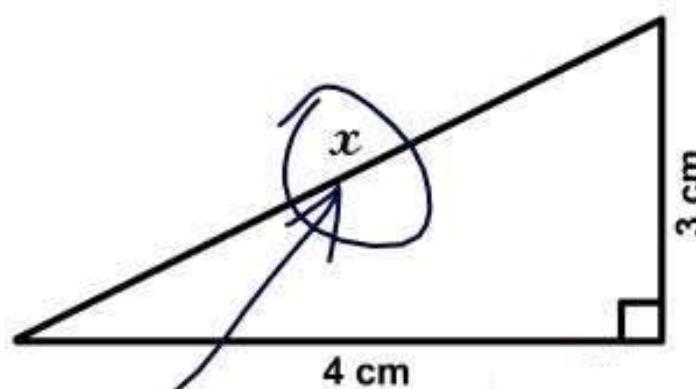
## In the practical...

- Define the hierarchy for a solar system:
- Earth rotating around sun
- Moon rotating around earth



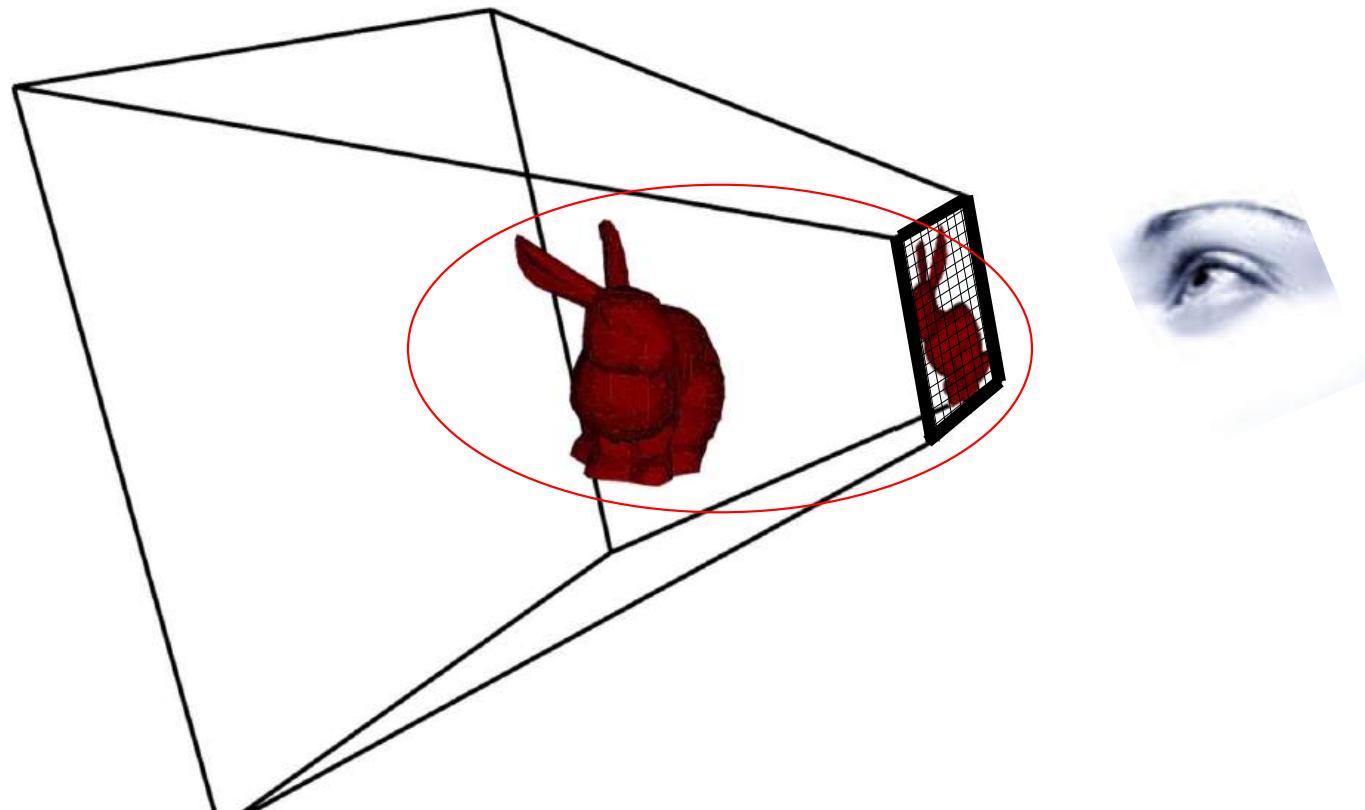
## Questions?

Find x.



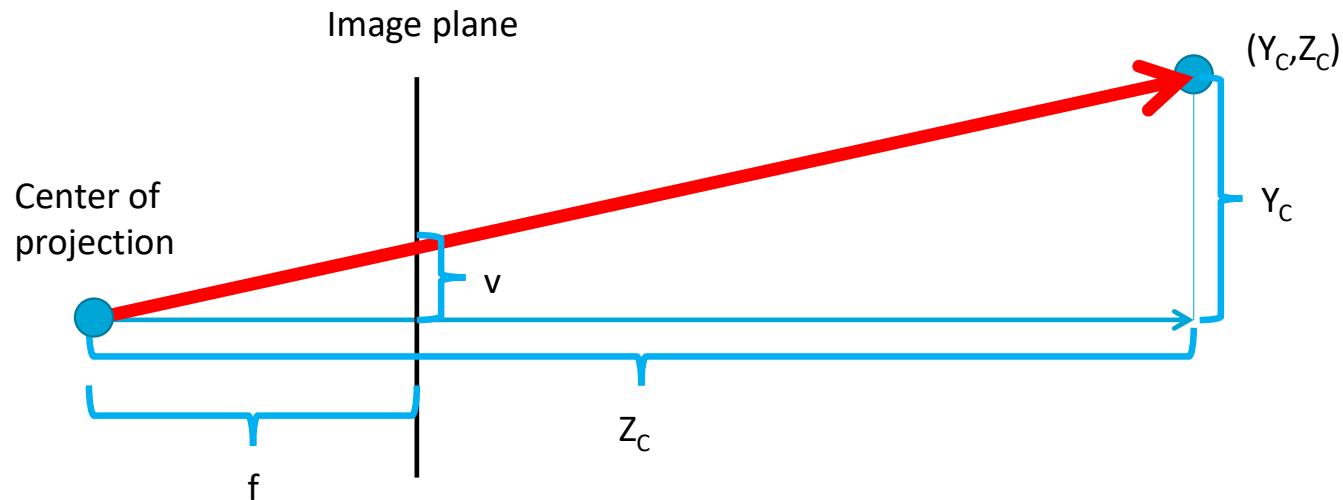
Here it is

**One pipeline step is still missing...**



## Perspective Projection

- sideview: Formula is simple if camera at origin



Similar triangles:  $v / f = Y_c / Z_c$

# Perspective Projection

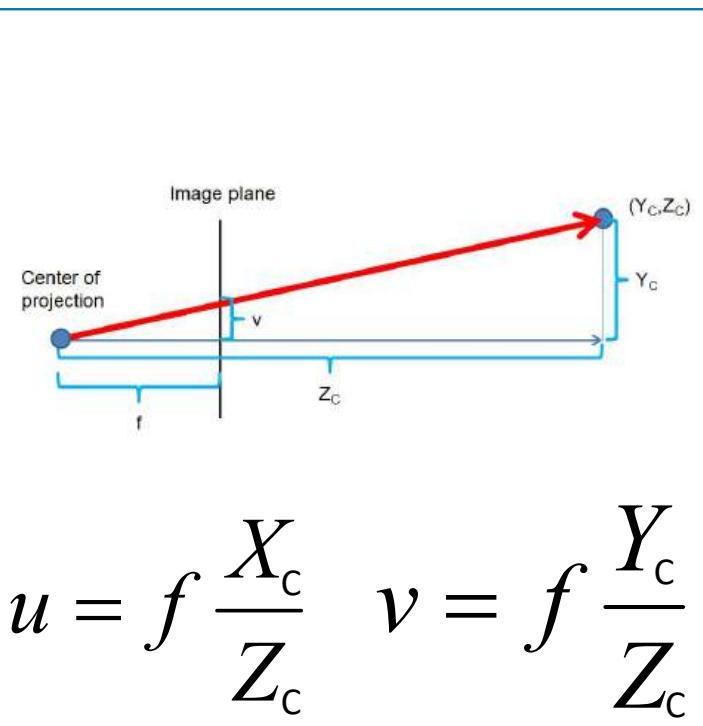
- Embed the point P in the projective space

$$P = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \Rightarrow \tilde{P} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- Look for M such that:

$$M\tilde{P} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix}$$

Impossible in standard  $\mathbb{R}^n$



$$u = f \frac{X_c}{Z_c} \quad v = f \frac{Y_c}{Z_c}$$

## What is the problem?

$$M\tilde{P} = \begin{bmatrix} a & b & c & d \\ e & q & g & h \\ i & j & m & n \\ k & l & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

## What is the problem?

$$M\tilde{P} = \begin{bmatrix} a & b & c & d \\ e & q & g & h \\ i & j & m & n \\ k & l & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax + by + cz + dw \\ \dots \end{bmatrix}$$

## What is the problem?

$$M\tilde{P} = \begin{bmatrix} a & b & c & d \\ e & q & g & h \\ i & j & m & n \\ k & l & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax + by + cz + dw \\ \dots \end{bmatrix} \xrightarrow{\text{?}} \frac{fx}{z}$$

$$u = f \frac{X_c}{Z_c}$$

# Perspective Projection

- Hint: Think projective!

$$M\tilde{P} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix}$$

## Perspective Projection

- Hint: Think projective!

$$M\tilde{P} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix}$$

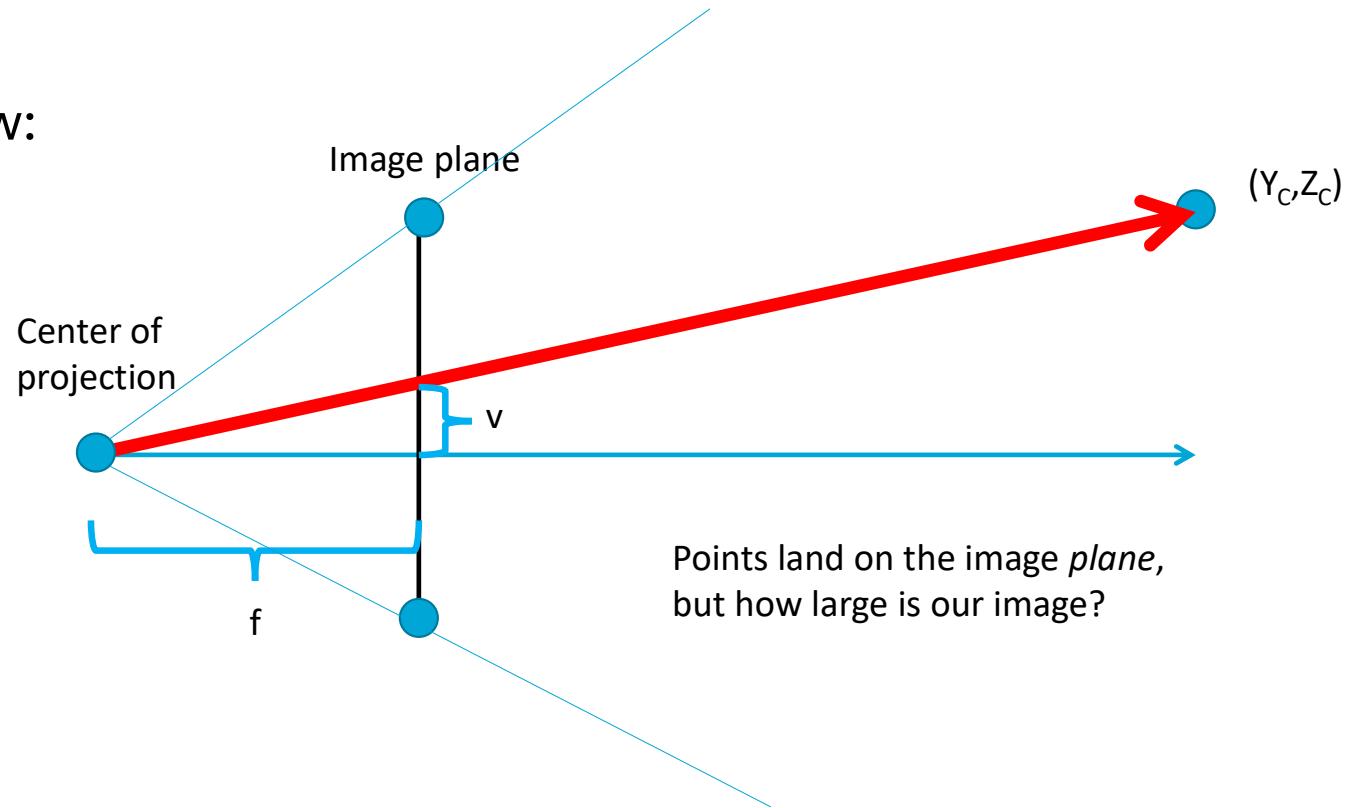
## Perspective Projection

- Solution:

$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

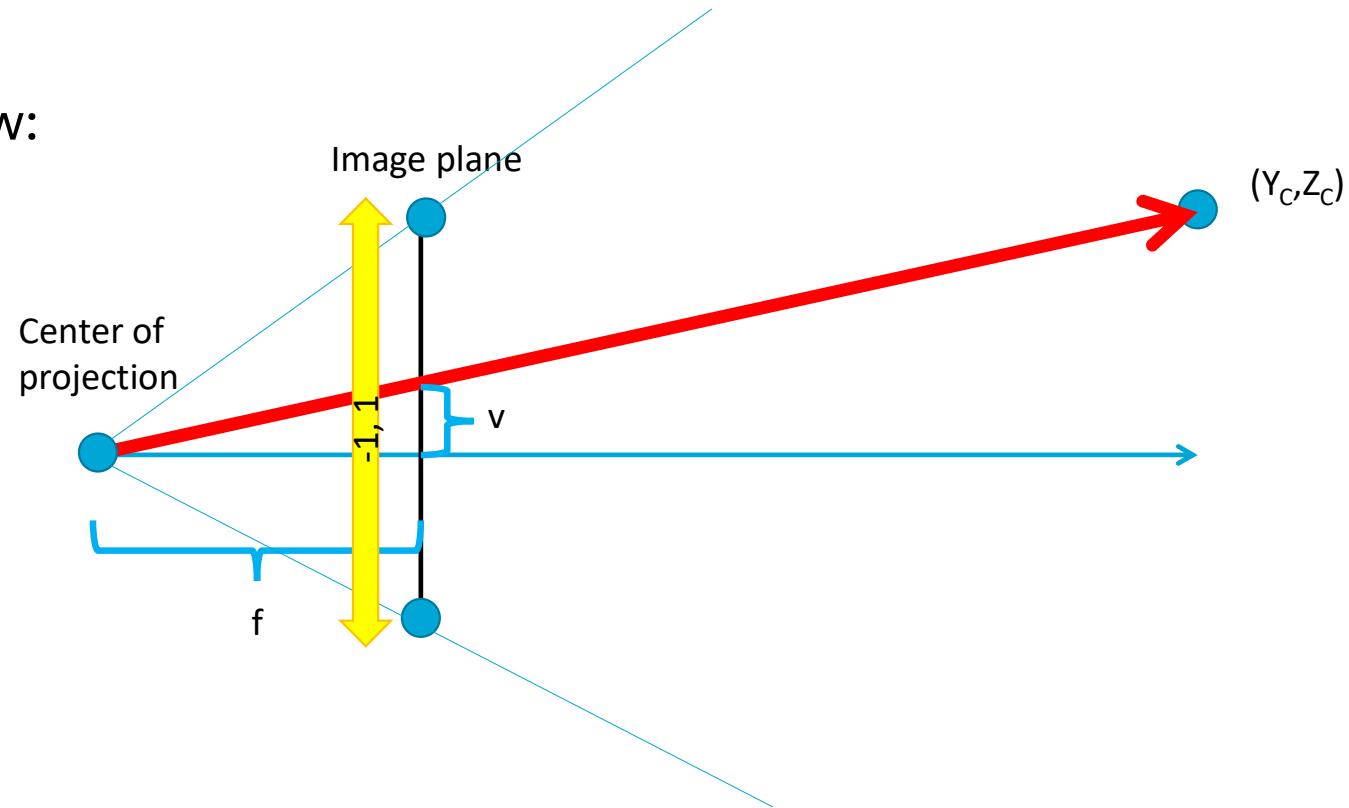
# Virtual Camera Model

- sideview:



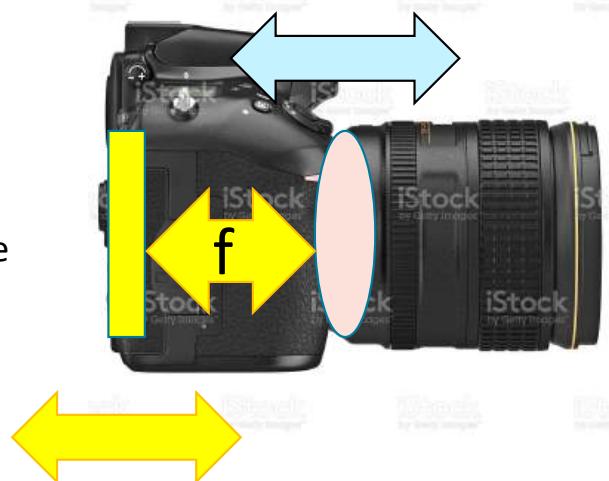
## Virtual Camera Model

- sideview:



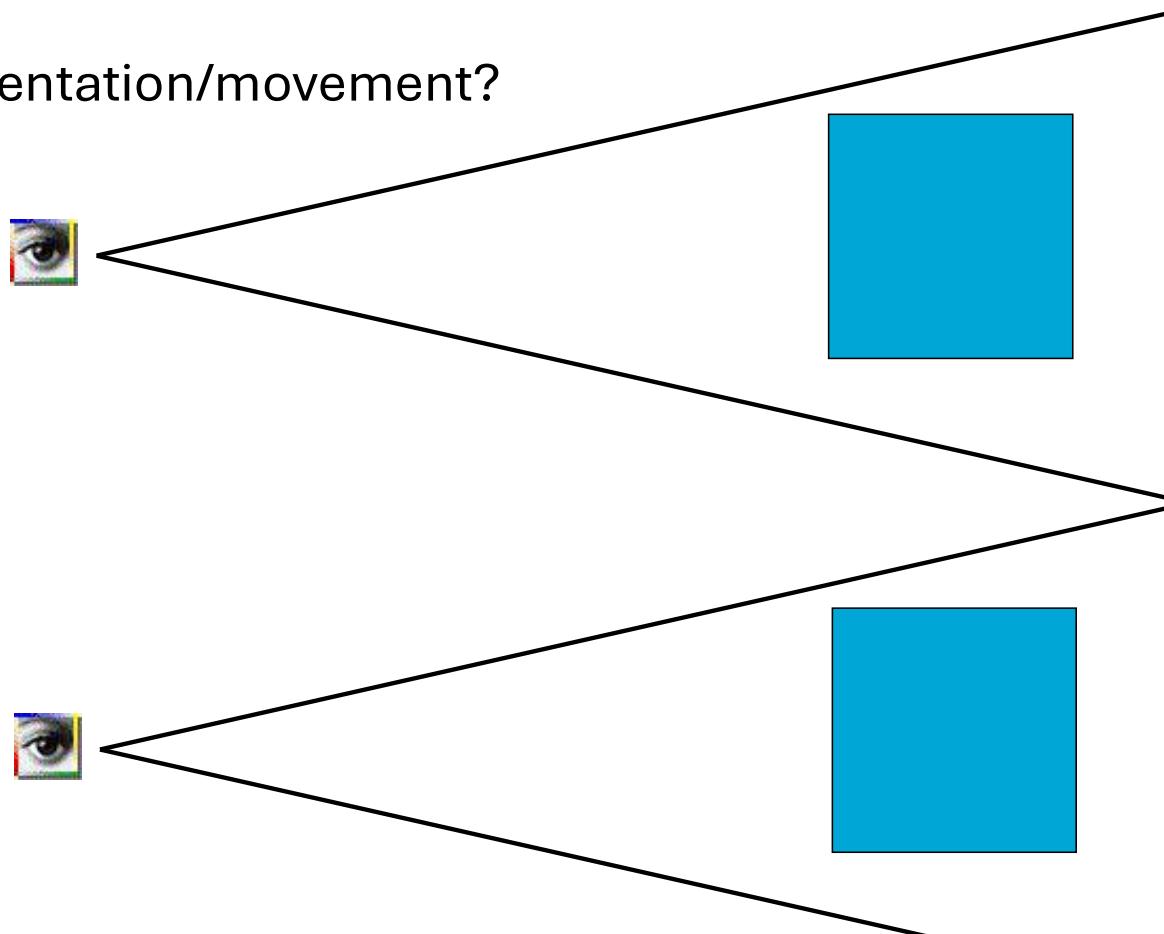
## Real camera

Sensor has a fixed size  
“ $[-1,1]$ ”



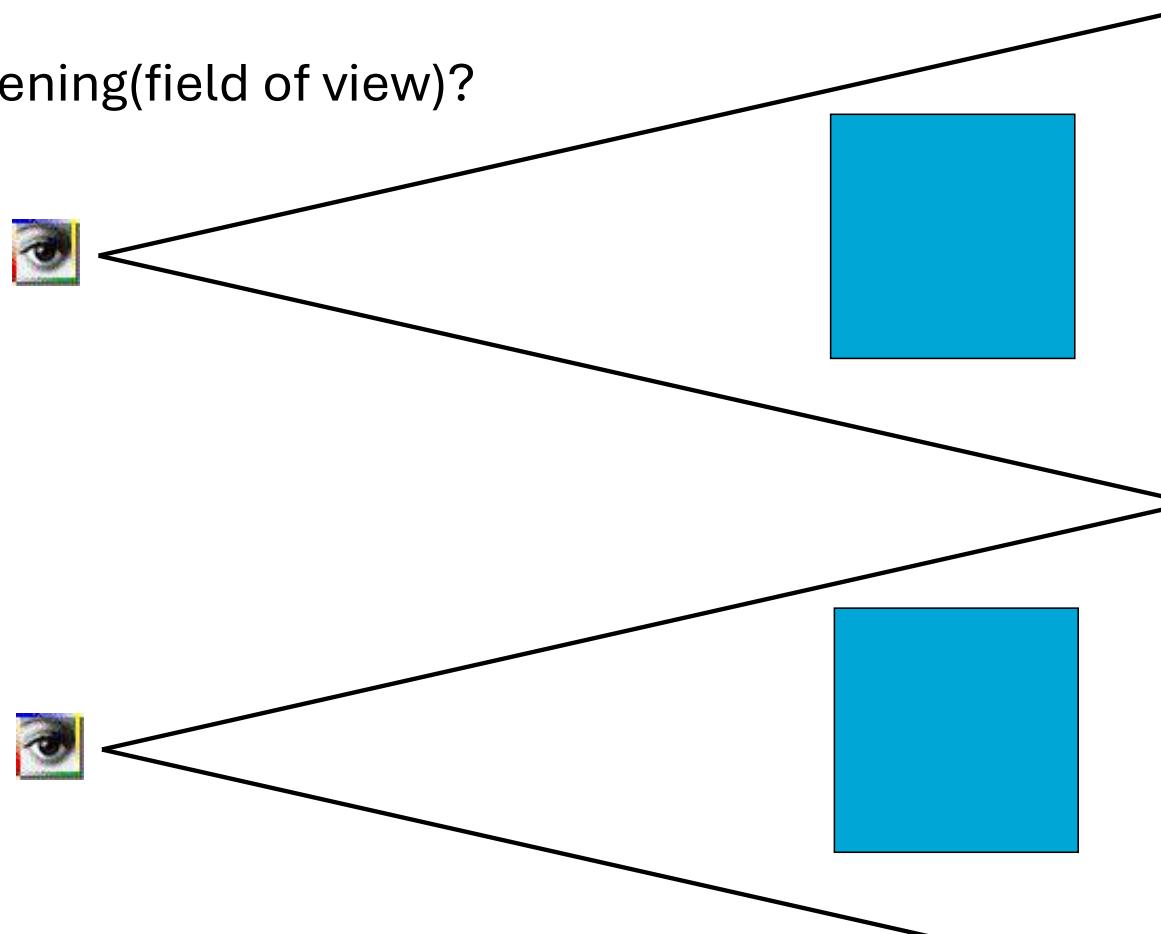
## Virtual Camera Model

- Camera orientation/movement?



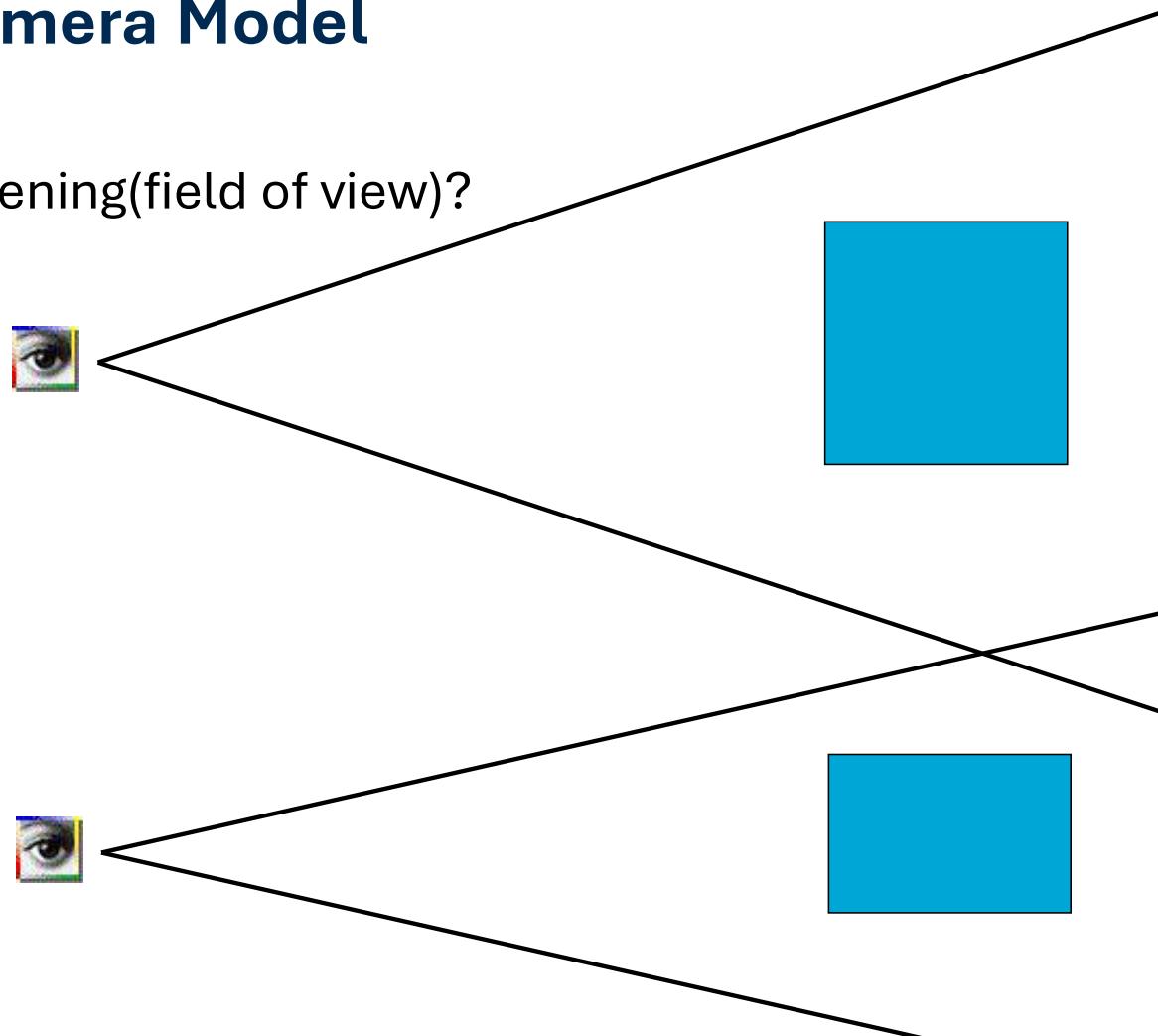
## Virtual Camera Model

- Camera opening(field of view)?



## Virtual Camera Model

- Camera opening(field of view)?

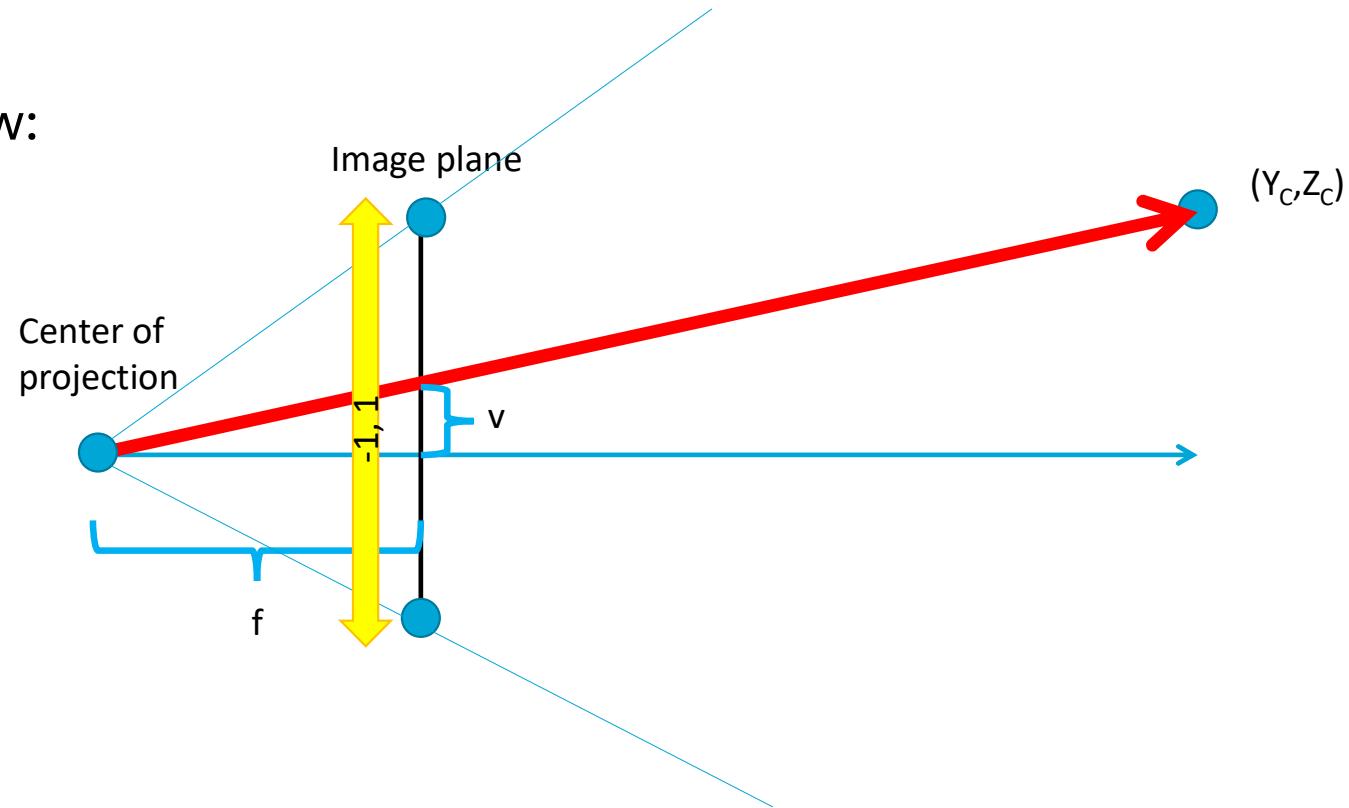


## Virtual Camera Model

- The world literally revolves around the camera!
- Just deform the scene in the “right” way and we can always assume that
  - Camera centre (eye) is at the origin
  - Camera is oriented along the z-axis
  - Image aligned with x-y and has unit size  $[-1,1]^2$
  - Only keep parameter  $f$  – focal length

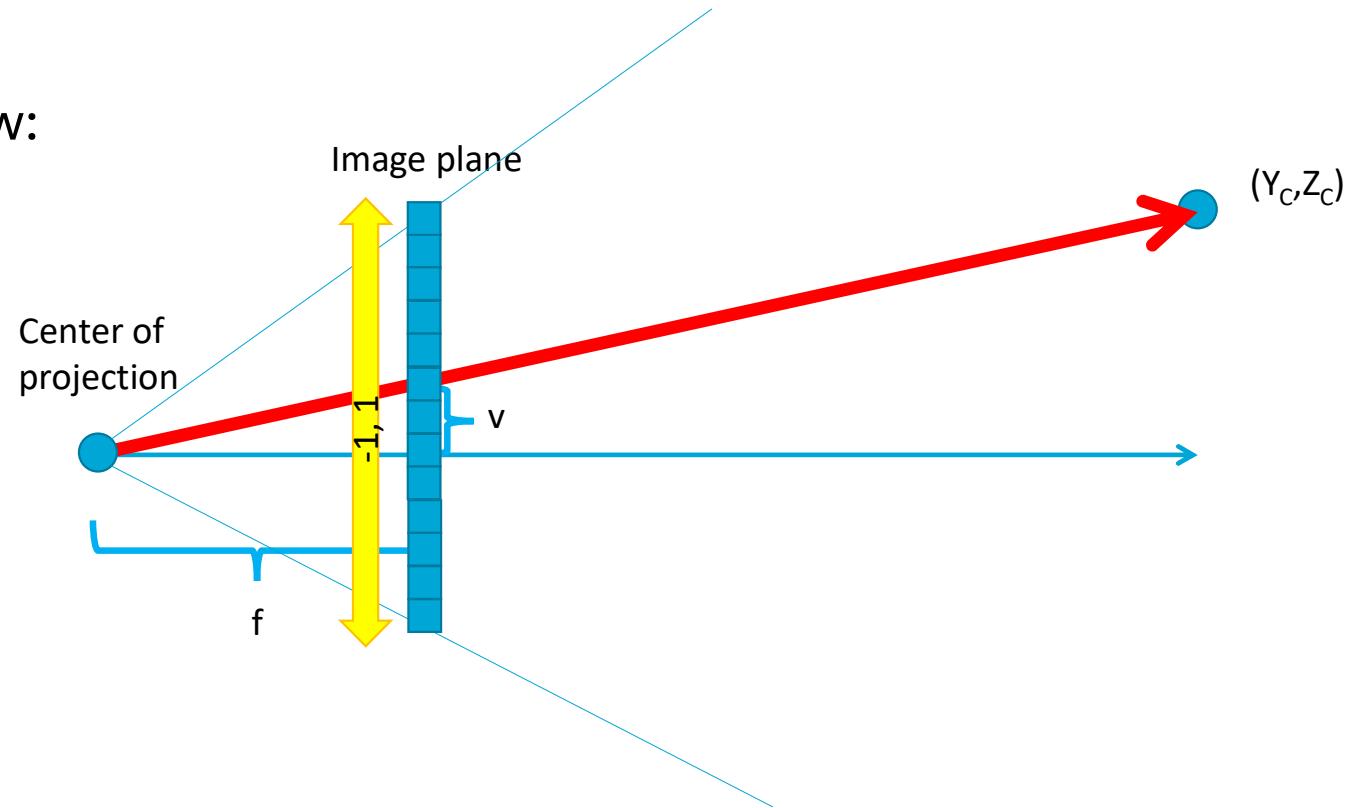
## Virtual Camera Model

- sideview:



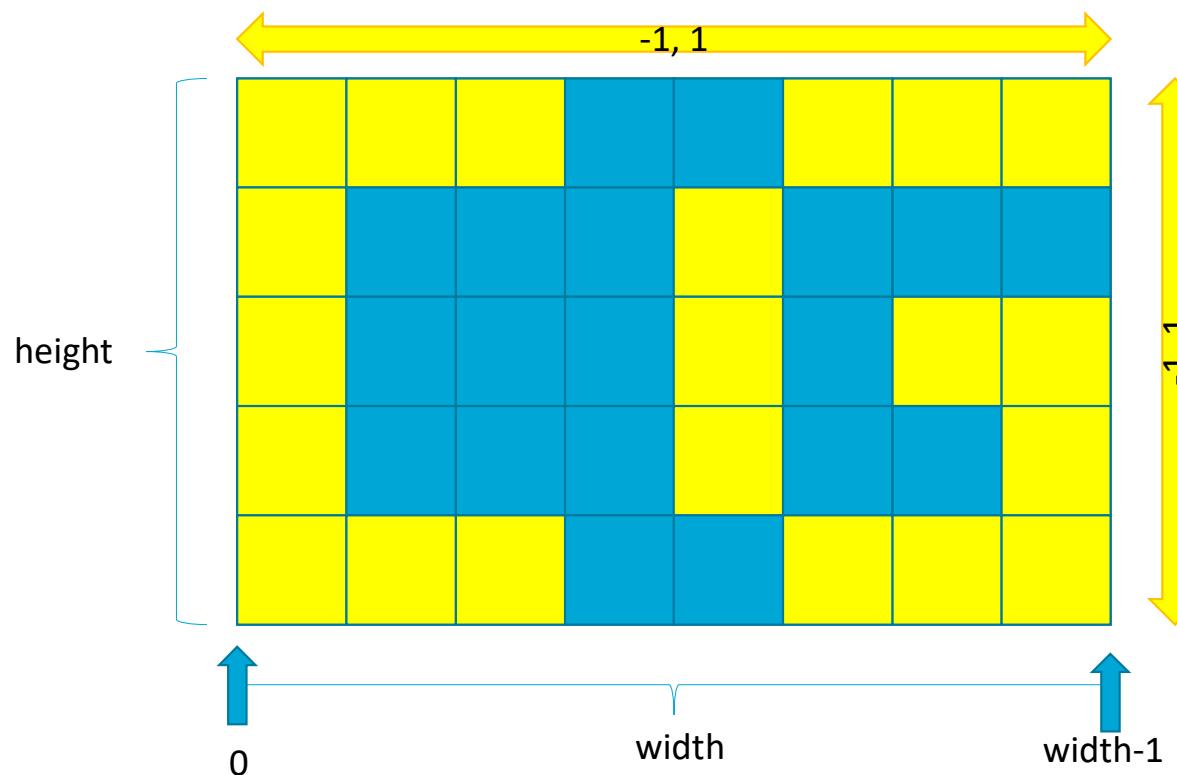
## Virtual Camera Model

- sideview:



# Viewport Transformation

- $(-1,1) \times (-1,1) \rightarrow [0, width-1] \times [0, height-1]$



Homework:  
Find a matrix  
to do this  
mapping

Hint:  
it has the form

$$\begin{pmatrix} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

# Complete Camera Model

- Finally:

$$\begin{array}{c}
 \text{Viewport} \\
 \left( \begin{array}{ccc} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{array} \right) \left( \begin{array}{cccc} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right) \left( \begin{array}{cccc} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{array} \right)
 \end{array}$$



Pixel mapping

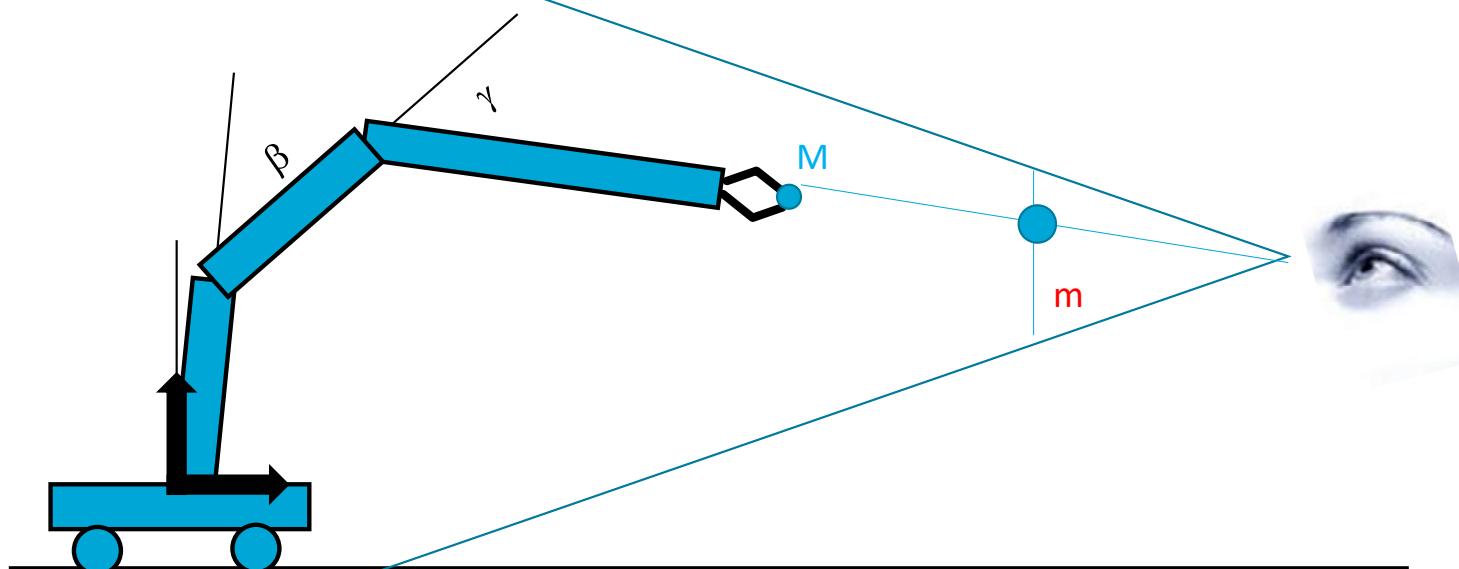


“standard camera” Deforms scene so that a “standard projection camera” can be used



# Today

Find matrix  $P$  such that the projected pixel position  $m$  of point  $M$  is  $PM$ .



# Conclusion

- In this lecture, you have seen how to:
  - Transformations with homogeneous coordinates
  - Creation of complex objects via local frames
  - A virtual camera expressed as a matrix

... (almost) all ingredients for the geometric graphics pipeline

## Reading

- Chapter 7 in the book
  - Derivation is in a slightly different order:
    - First scene deformation with respect to a camera frustum
    - Optional: As preparation for next time
      - viewport matrix
      - projection matrix

## Exercises

- Define a 45 degree rotation around a point and verify the matrix is correct with examples
- Check the given rotation matrices on the slides

**Thank you very much!**



**CSE2215 - Computer Graphics**

# **Materials & Shading**

## **All that glitters is not gold**

Elmar Eisemann

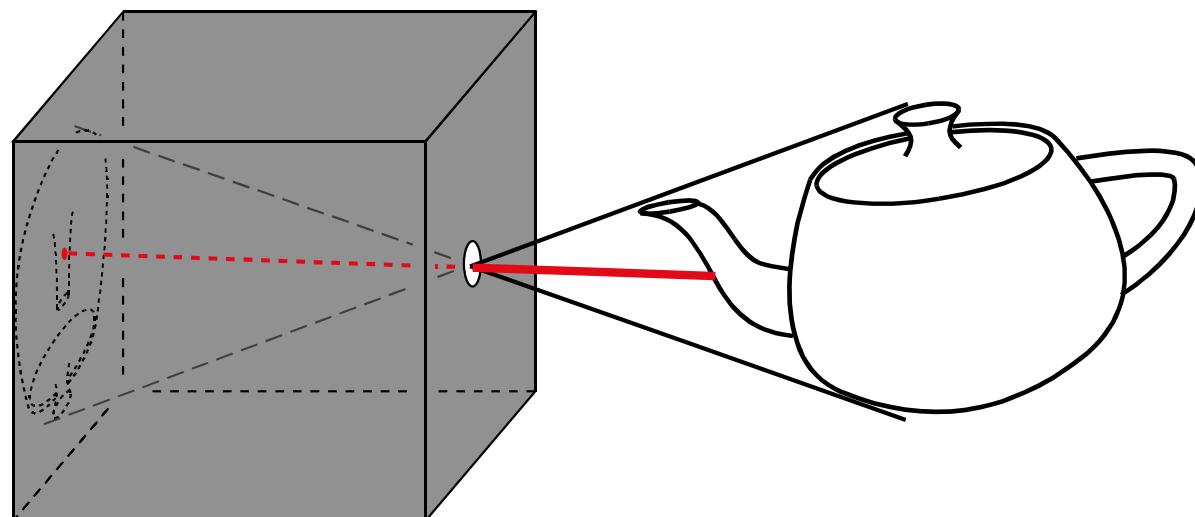
Delft University of Technology



The last week in a memory far,  
far away....

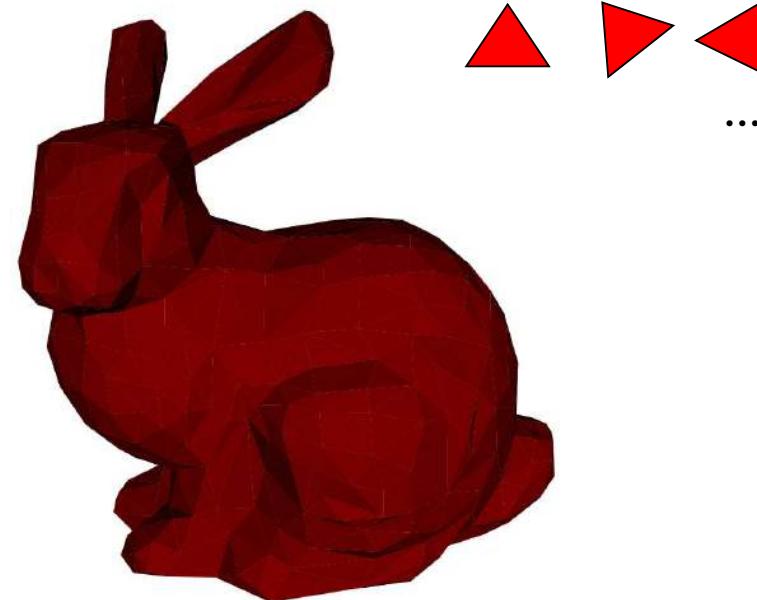
## Pinhole camera

- Box with hole
- Perfect image for “point-sized” hole



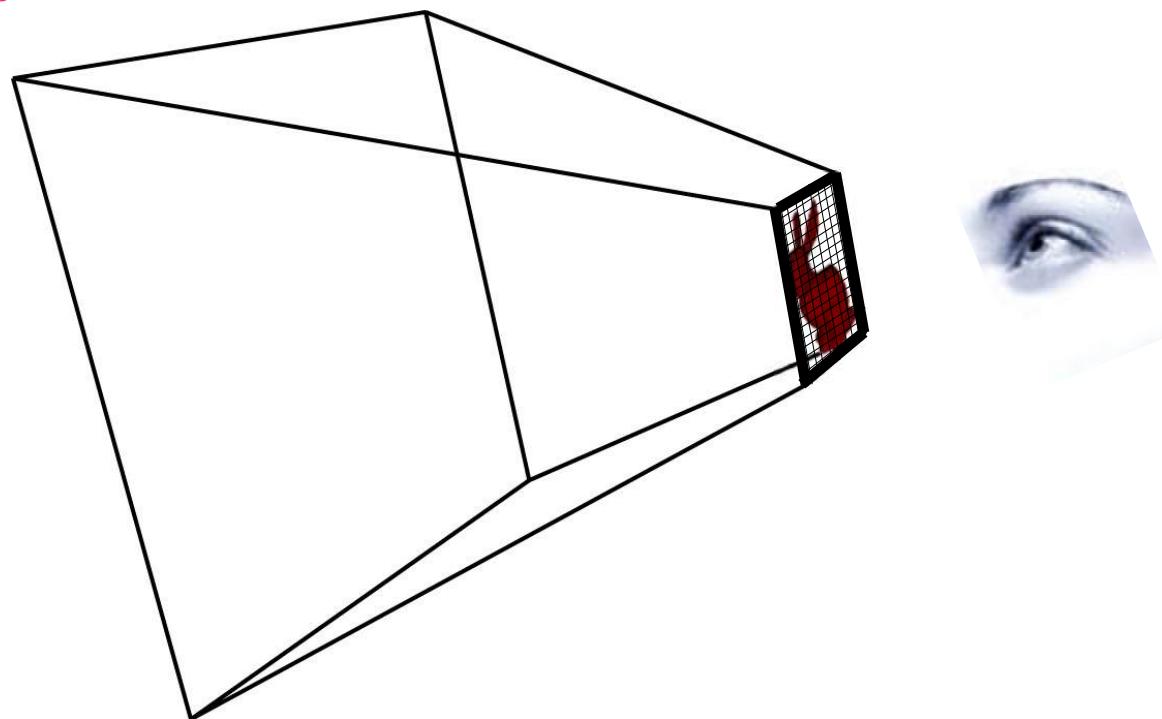
## Triangle Mesh

- Models are typically lists of triangles



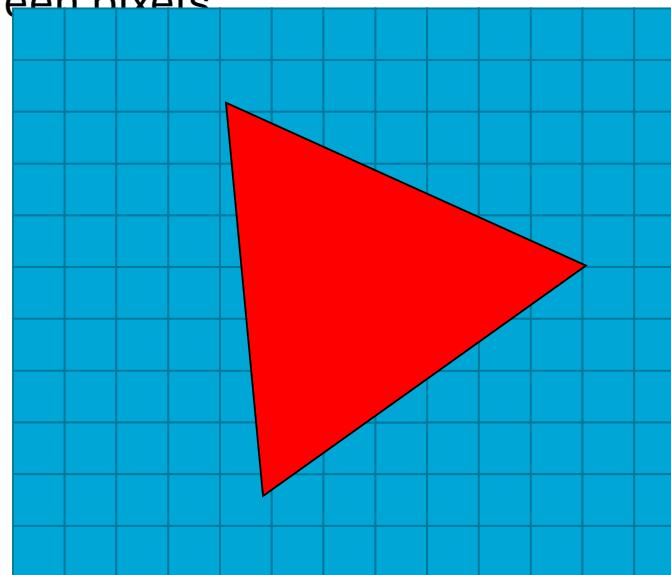
## Simplified Graphics Pipeline

- **Projection:** Transform coordinates to screen



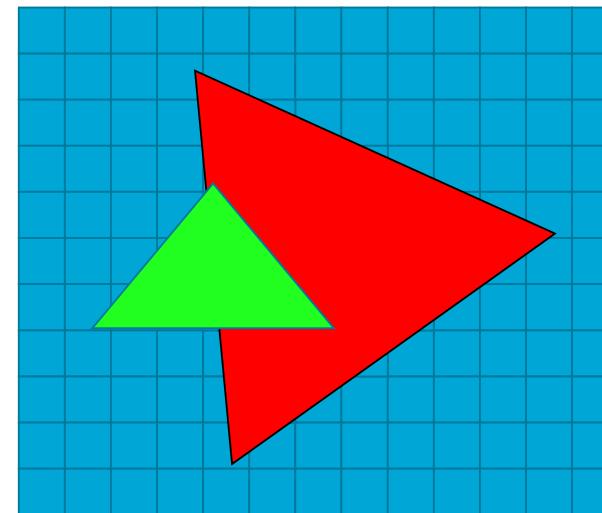
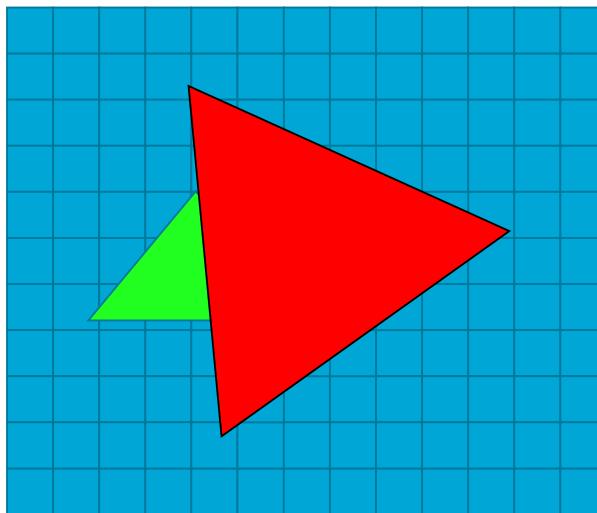
# Simplified Graphics Pipeline

- Rasterization: Fill screen pixels



## Simplified Graphics Pipeline

- **Catch:** Triangle order would change result

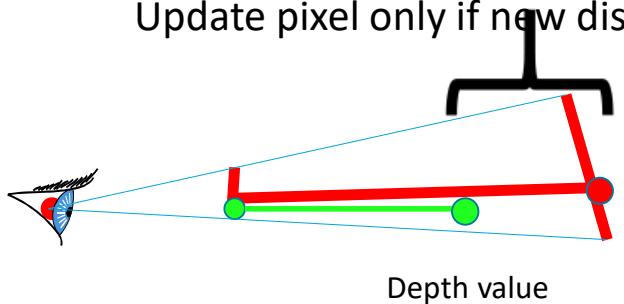


Need to keep the pixel of the closest triangle

## Simplified Graphics Pipeline

- Solution to maintain drawing order:  
Compare Z values!

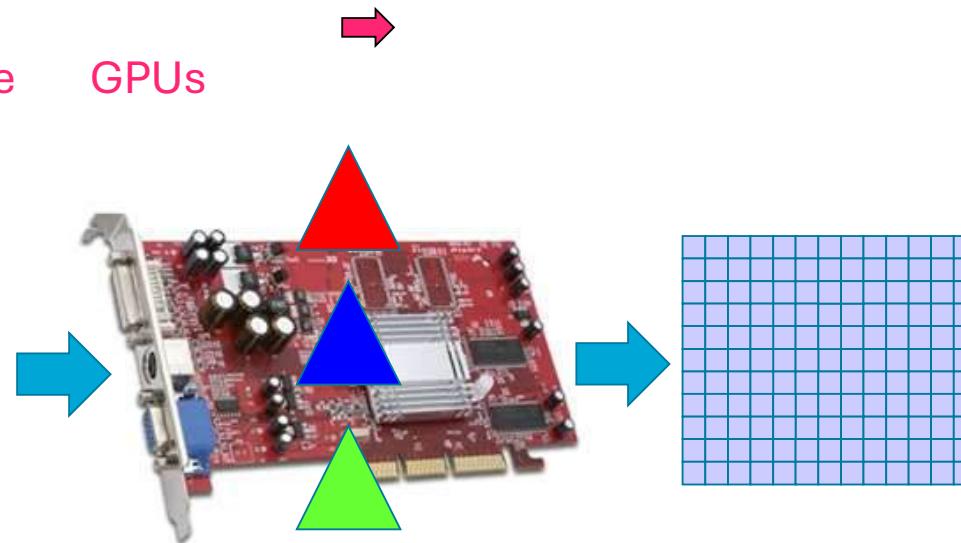
Compare new distance to stored distance  
Update pixel only if new distance is nearer



- We need a “depth”!

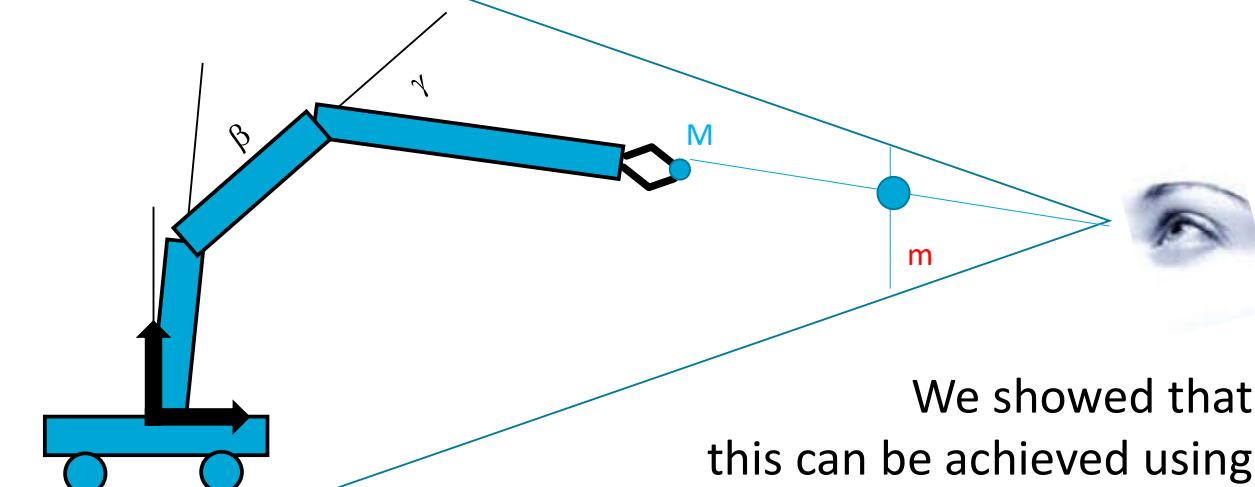
## Graphics Pipeline

- Highly parallelizable GPUs



Last time:

Find matrix  $P$  such that the projected pixel position  $m$  of point  $M$  is  $PM$ .



We showed that  
this can be achieved using  
homogeneous coordinates

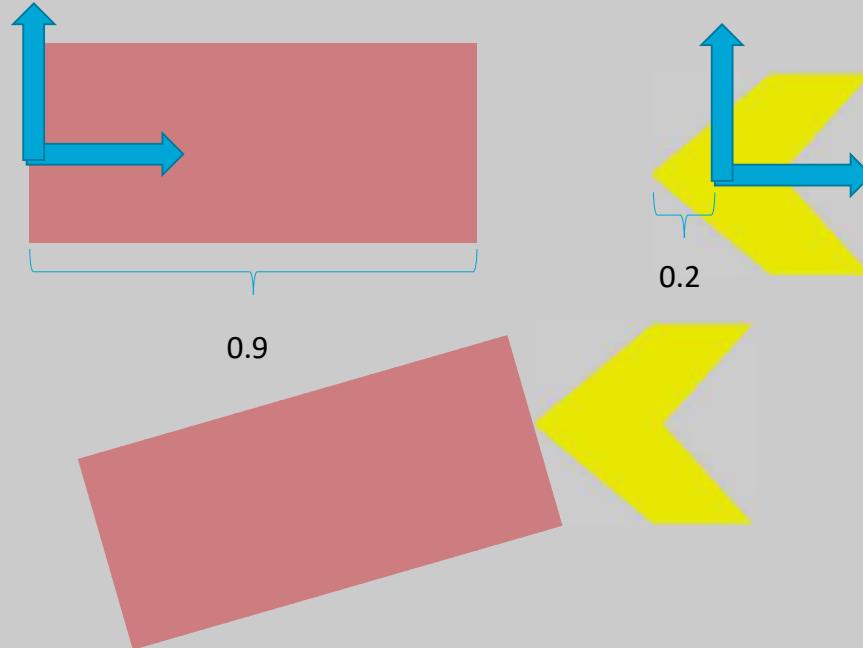
## Last time:

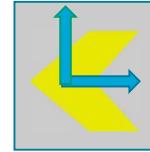
- This matrix  $M$  is usually constructed as the multiplication of two matrices:
- A matrix to place the vertices of an object in the right location in 3D space
- A matrix that performs the projection from 3D via the virtual camera

## Last time:

- This matrix  $M$  is usually constructed as the multiplication of two matrices:
- **A matrix to place the vertices of an object in the right location in 3D space**
- A matrix that performs the projection from 3D via the virtual camera

## Object Placement

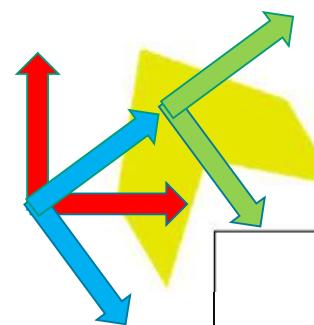
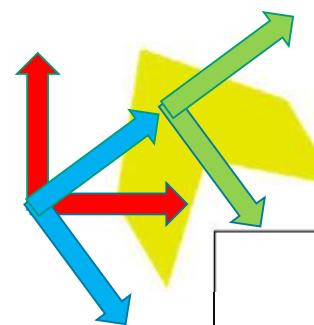
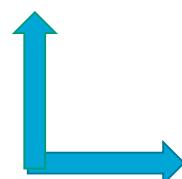




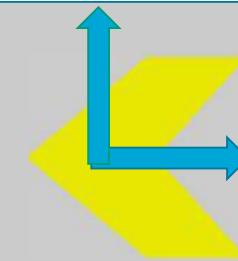
## Object placement

- The matrix transformation to place the object can be derived by imagining a local frame that is moved by the matrices
- Imagine the following instructions:

- $T_1$
- $R$
- $T_2$



$T_1 R T_2$



14

TU Delft

## Last time:

- This matrix  $M$  is usually constructed as the multiplication of two matrices:
- A matrix to place the vertices of an object in the right location in 3D space
- **A matrix that performs the projection from 3D via the virtual camera**

## Camera Model

$$\begin{array}{c} \text{Viewport} \\ \left( \begin{array}{ccc} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{array} \right) \end{array} \begin{array}{c} \text{Projection} \\ \left( \begin{array}{cccc} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right) \end{array} \begin{array}{c} \text{Camera Matrix} \\ \left( \begin{array}{cccc} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{array} \right) \end{array}$$



Pixel mapping



“standard camera”  
projection



Deforms scene so that a  
“standard camera”  
can be used

## Last time:

- Use of homogeneous coordinates

$$P = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \Rightarrow \tilde{P} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

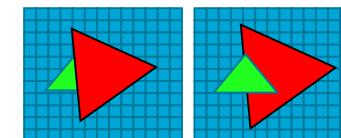
- Complex Object transformations
- Camera movement/orientation
- Projection
- Mapping to pixels

Single  
Matrix

## Something is missing...

$$P = \begin{pmatrix} \text{image} & & \\ \begin{pmatrix} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} \text{projection} & & \\ \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} & \begin{pmatrix} \text{orientation/location} & & \\ \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{pmatrix} \end{pmatrix}$$

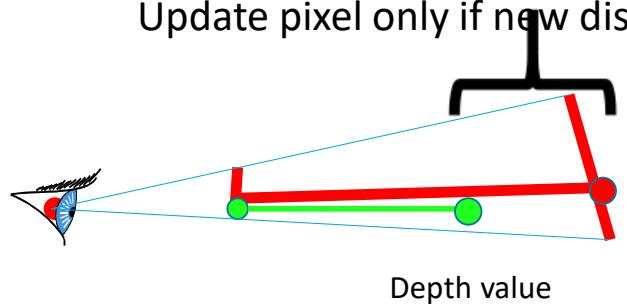
- What is the problem of this matrix for the Graphics Pipeline?



## The Depth Test misses the depth...

- We need to keep a Z coordinate!

Compare new distance to stored distance  
Update pixel only if new distance is nearer



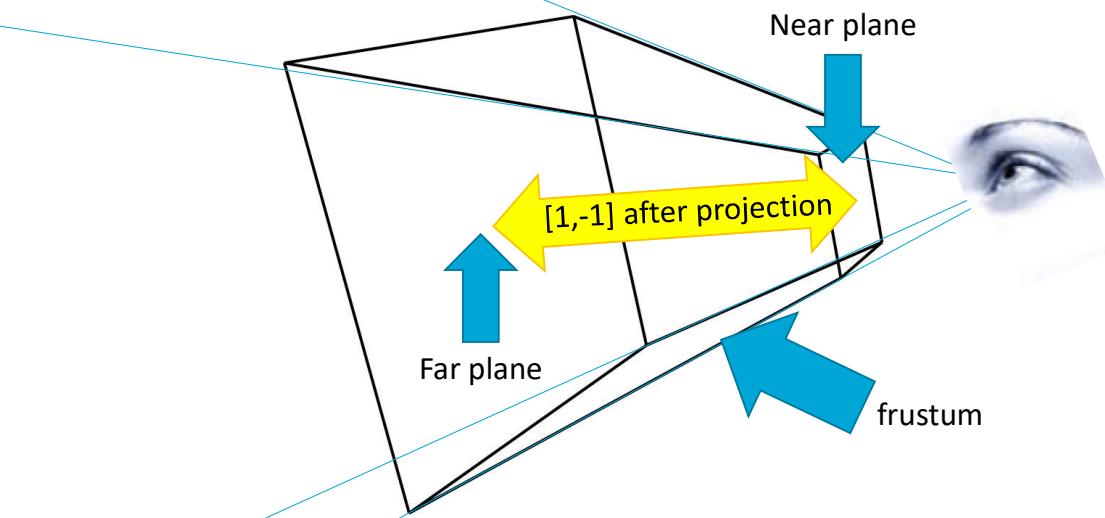
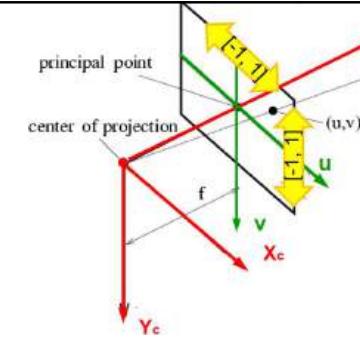
## Something is missing... the depth

$$P = \begin{pmatrix} \text{image} & & & \\ \left( \begin{array}{ccc} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{array} \right) & \boxed{\begin{array}{cccc} \text{projection} & & & \\ \left( \begin{array}{cccc} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right) & & & \end{array}} & \text{orientation/location} & \\ & & \left( \begin{array}{cccc} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{array} \right) & \end{pmatrix}$$

- It was eliminated in this projection matrix...
- To get it back, we need to extend this matrix

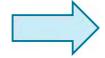
## Defining a Depth

- A 3D scene is infinite...
- How do we represent Z?
- Solution add a **near and far clipping plane!**



## The OpenGL Projection Matrix

- Our projection matrix:

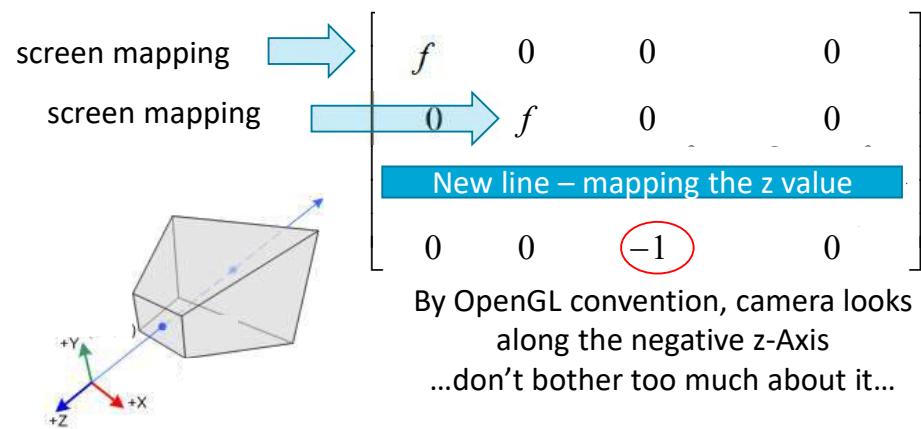
screen mapping   $f \quad 0 \quad 0 \quad 0$

screen mapping   $0 \quad f \quad 0 \quad 0$

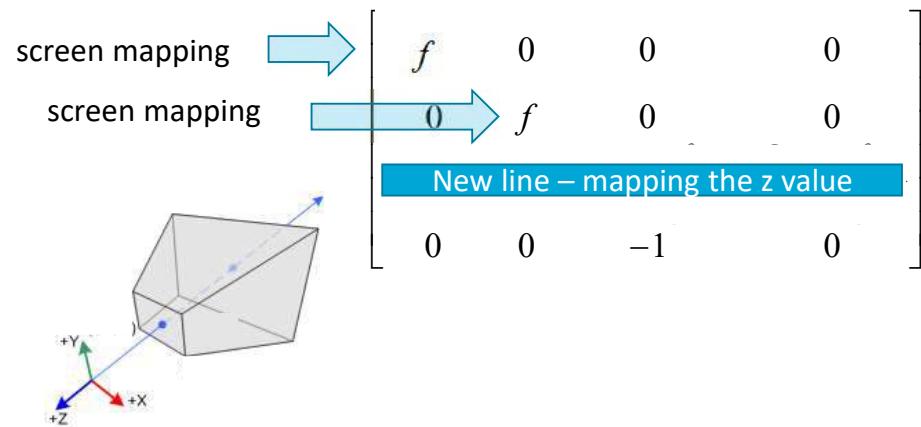
New line – mapping the z value

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

## The OpenGL Projection Matrix



## The OpenGL Projection Matrix



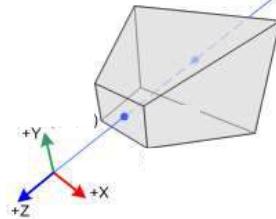
# The OpenGL Projection Matrix

- Aspect ratio for non-square displays:

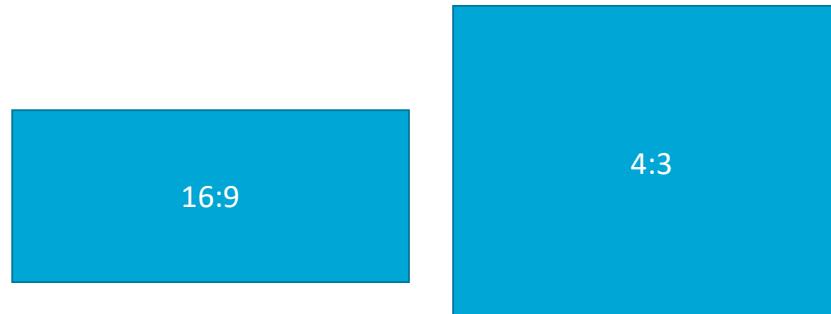
screen mapping  $\rightarrow$

$$\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ \text{New line - mapping the z value} & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Make a drawing:  
How does a  
projection to  
 $[-1,1]^2$  for a 16:9  
image look like



In other words:  
The scene is scaled such that  
the square image appears  
correctly stretched



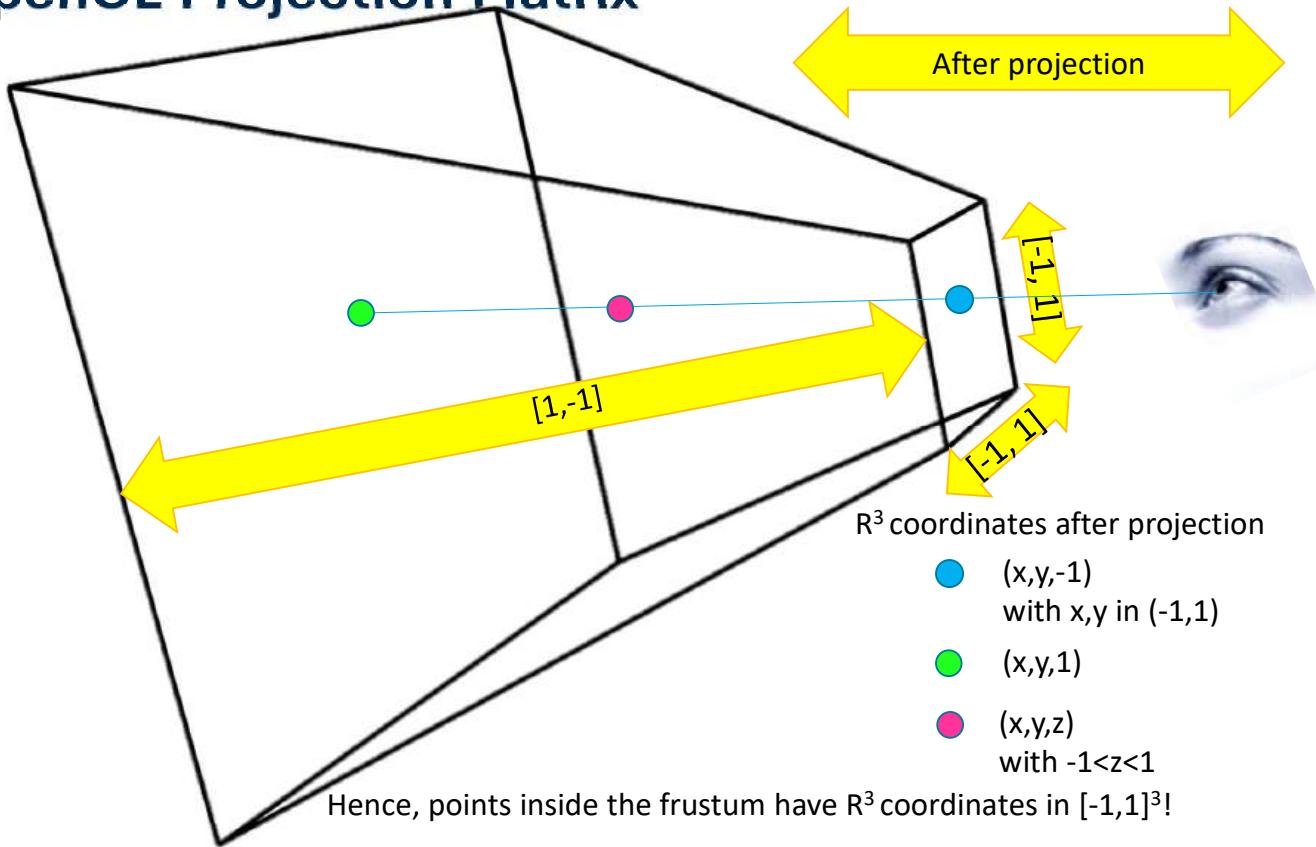
## The OpenGL Projection Matrix

- Definition maps near and far to [-1,1]:

$$\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near} + \text{far}}{\text{near} - \text{far}} & \frac{2\text{near}\text{far}}{\text{near} - \text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

, where near and far are the distances of the planes to the origin.

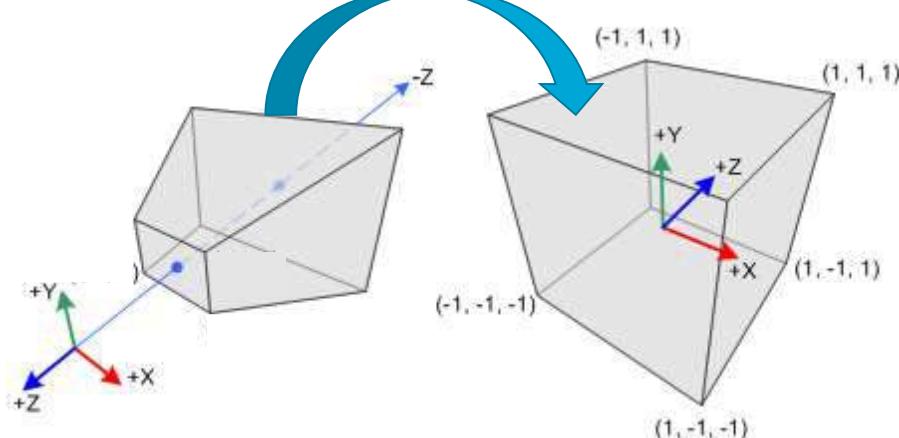
## The OpenGL Projection Matrix



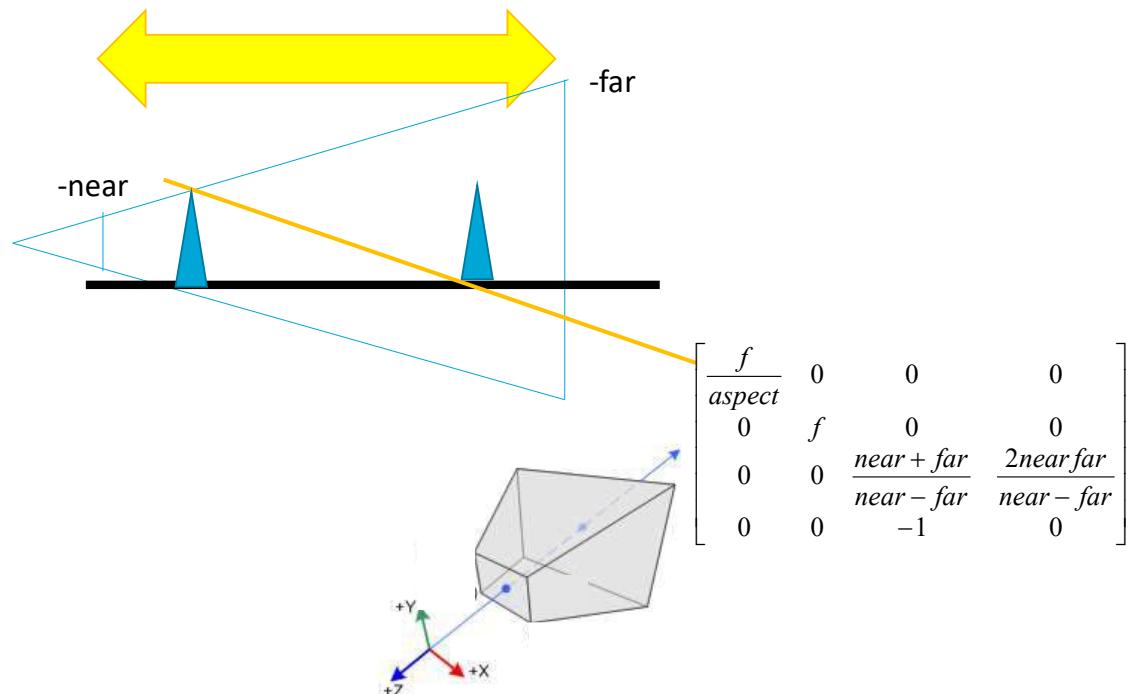
## Camera Space

- Content of Frustum is mapped inside a cube

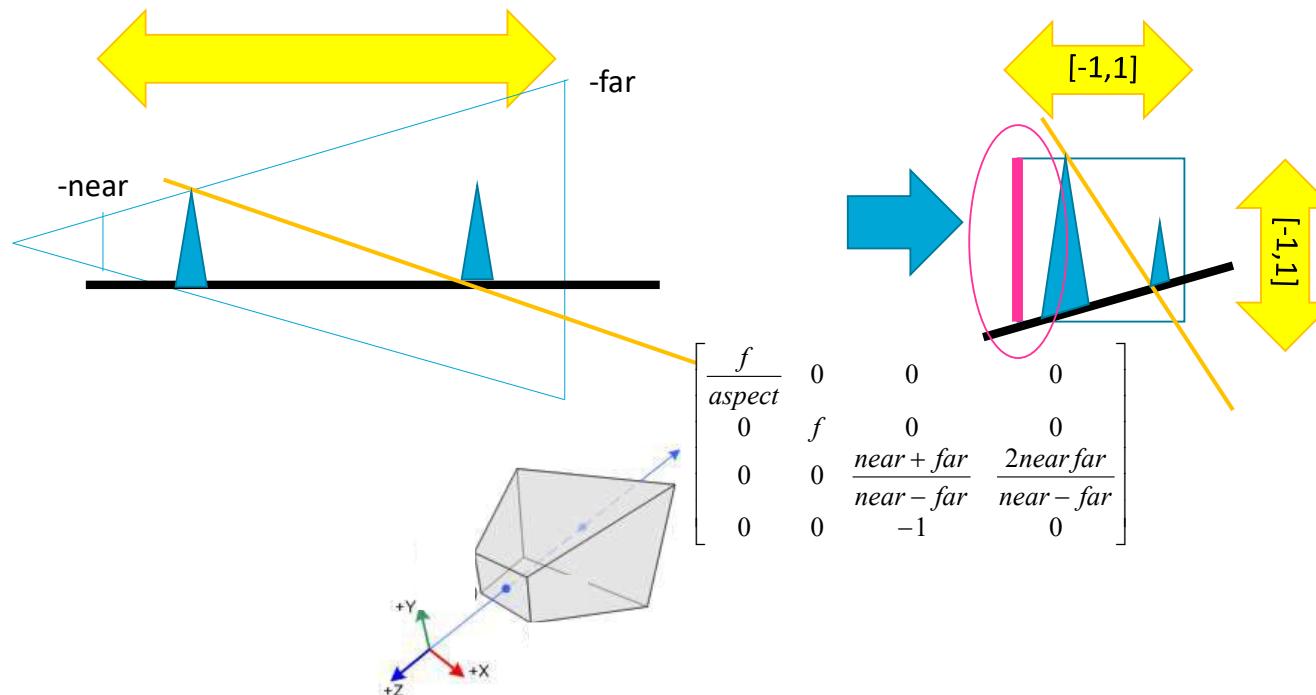
$$\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near} + \text{far}}{\text{near} - \text{far}} & \frac{2\text{near}\text{far}}{\text{near} - \text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



## Illustration of the Camera Mapping

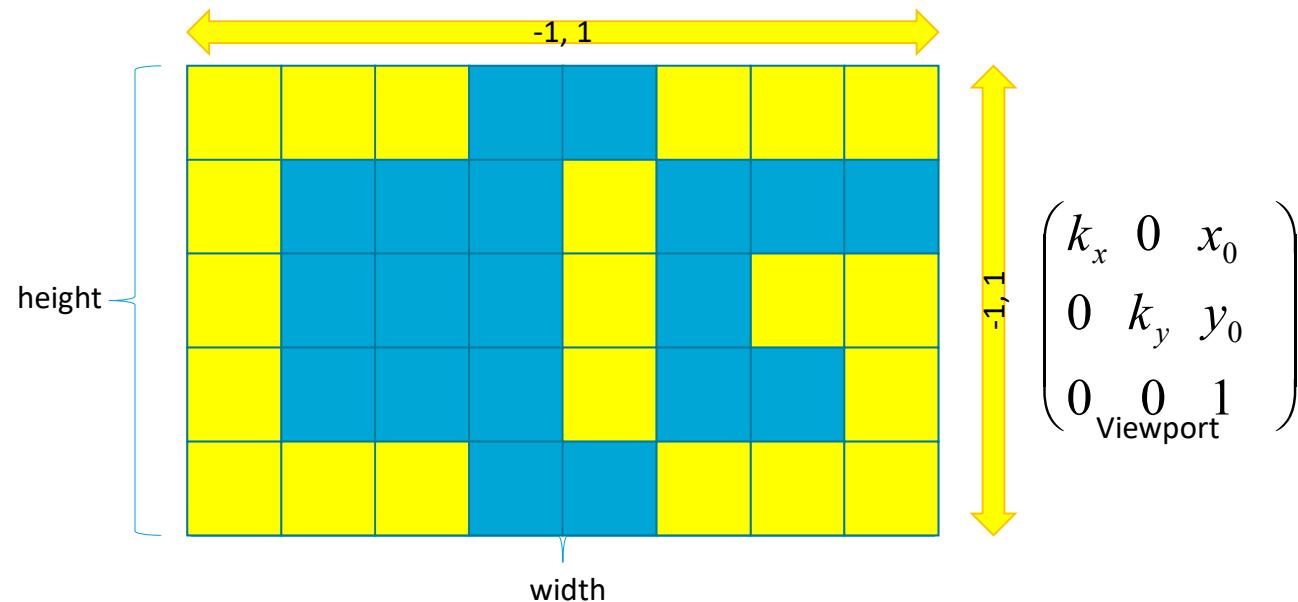


## Illustration of the Camera Mapping



## Simplified Graphics Pipeline

- Map the projection onto pixels
- $(-1,1) \times (-1,1) \rightarrow [0, \text{width}-1] \times [0, \text{height}-1]$



# Simplified Graphics Pipeline

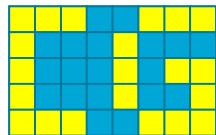
Traditional OpenGL Camera

can be described as a single 4x4 camera matrix!

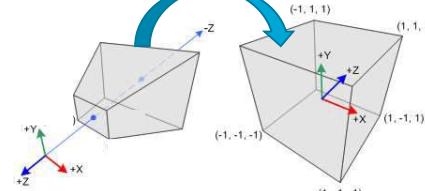
$$\begin{bmatrix} k_x & 0 & 0 & x_0 \\ 0 & k_y & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near} + \text{far}}{\text{near} - \text{far}} & \frac{2\text{near}\text{far}}{\text{near} - \text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

4x4  
ModelView Matrix

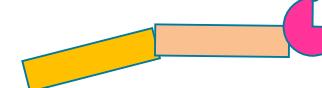
Viewport  
(including depth)



Projection



ModelView



T R T R T R

## Questions?



*Episode IV*

## ***MATERIALS AND SHADING***

*Transformations have taken over.  
Homogeneous coordinates rule  
the 3D space. Only a few more  
questions remain...*

I promise, this is the last point...

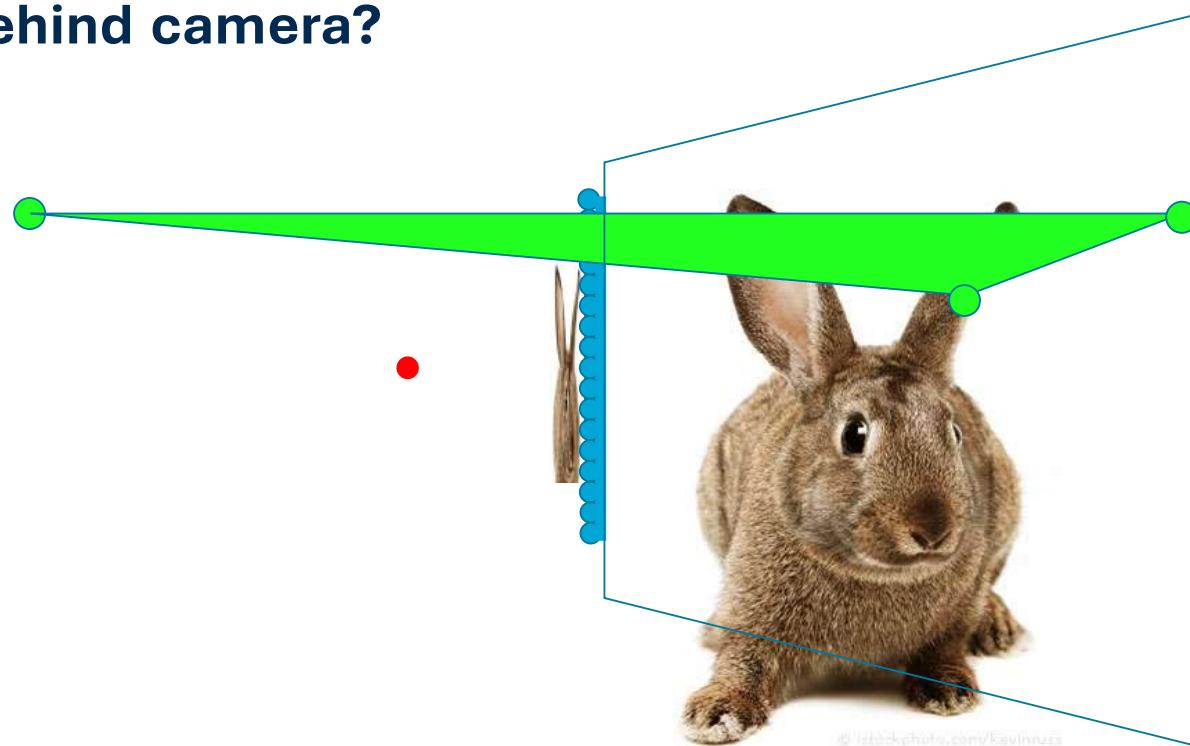


## I promise, this is the last point...

- While all the math works out for individual points, it fails for triangles...
- Unfortunately, we need an additional step:

**Clipping**

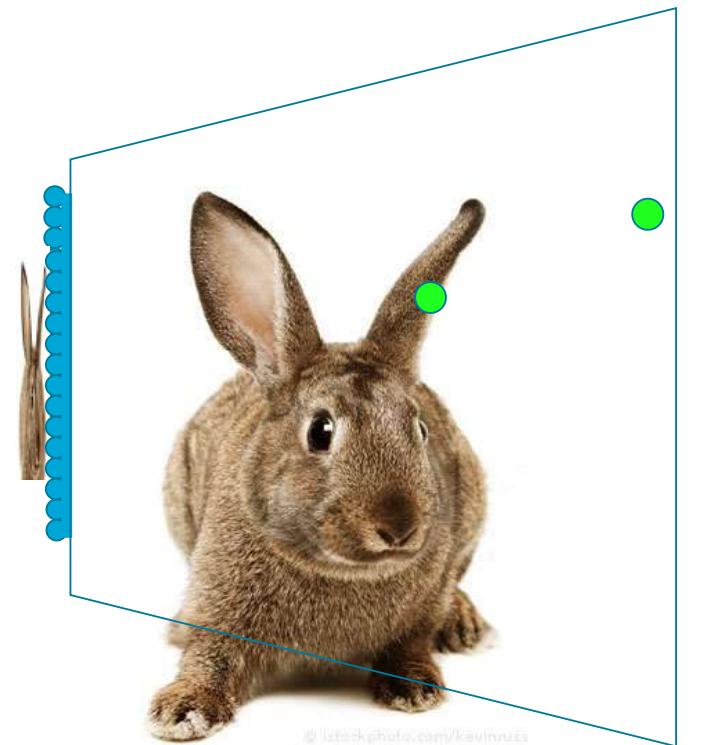
## Points behind camera?



© iStockphoto.com/kevinmusa

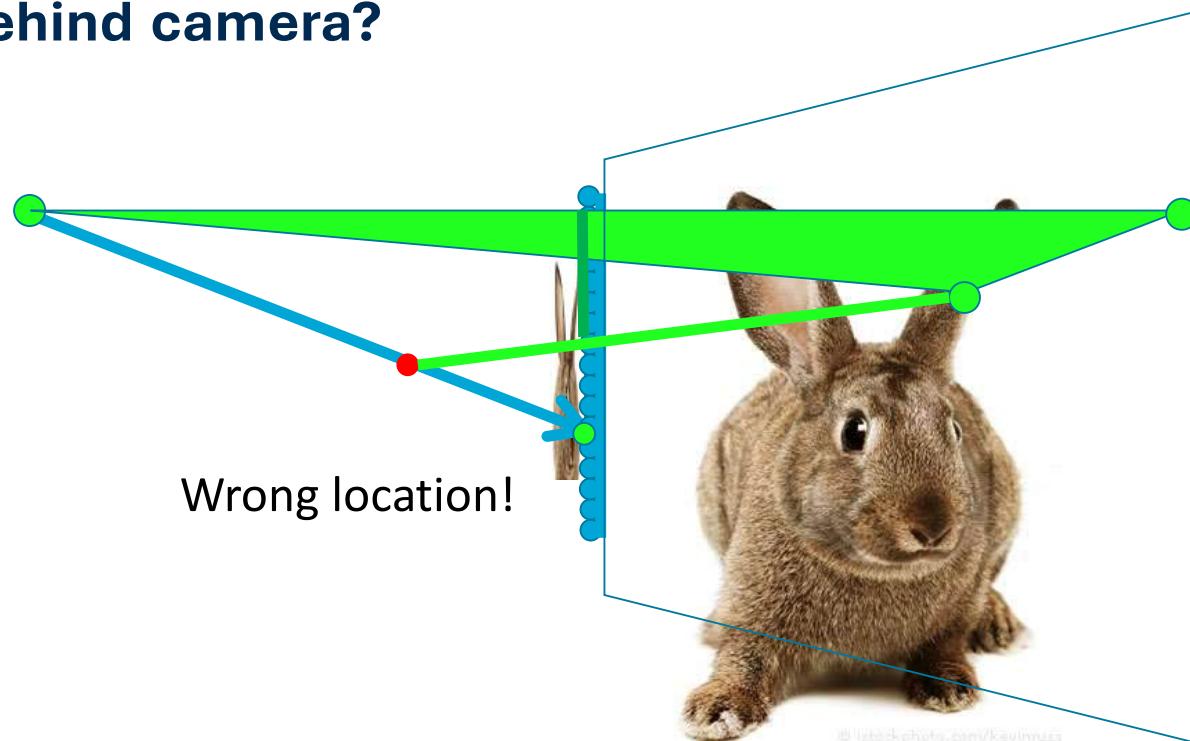
## Points behind camera?

- Delete? Triangle is gone!!!



## Points behind camera?

- Project?

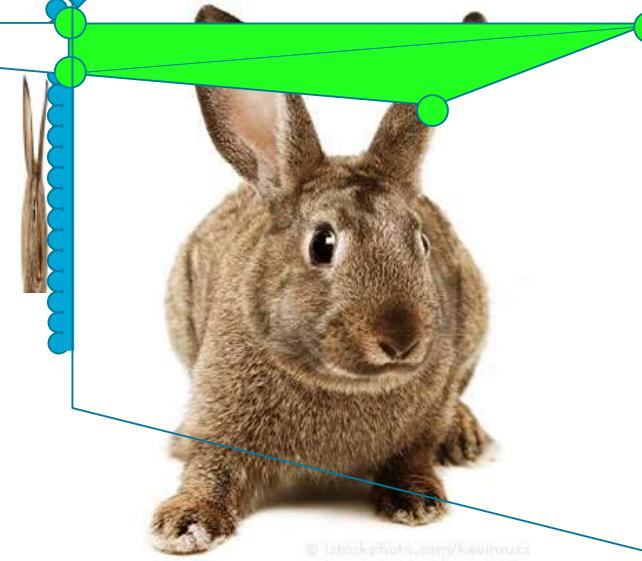


© iStockphoto.com/kevinnuss

## Points behind camera?

- What to do?

We should cut here!

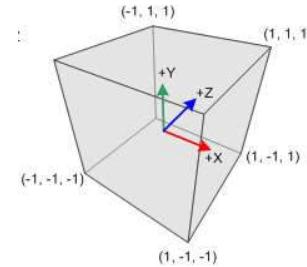


© iStockphoto.com/kevinnuss

- Test triangle and **clip** if it crosses frustum!

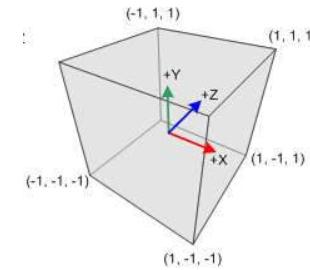
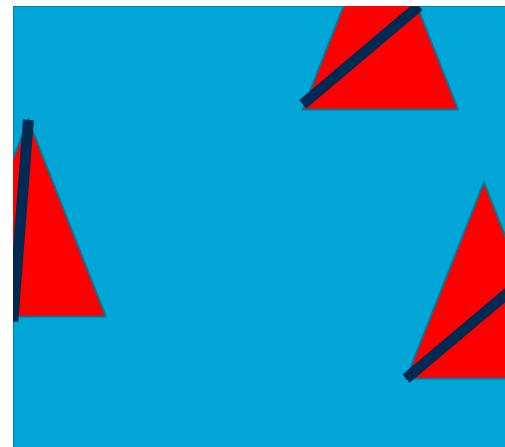
## Efficient Clipping

- Clip triangles against cube  
after applying projection matrix



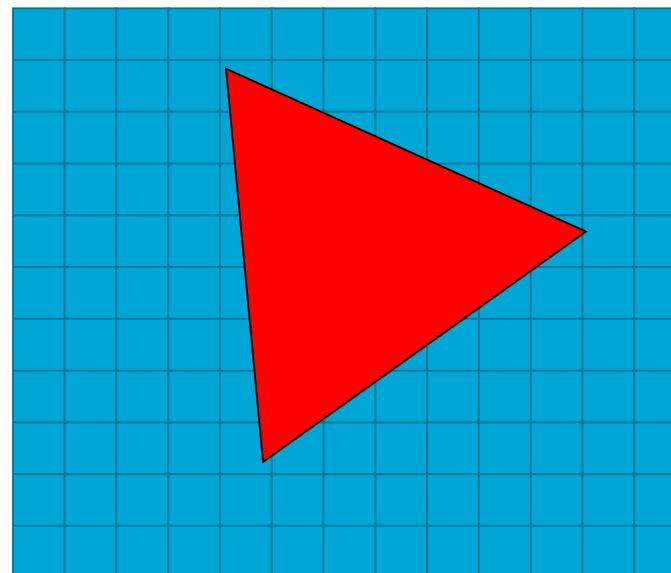
## Efficient Clipping

- Clip triangles against cube  
after applying projection matrix



## Simplified Graphics Pipeline

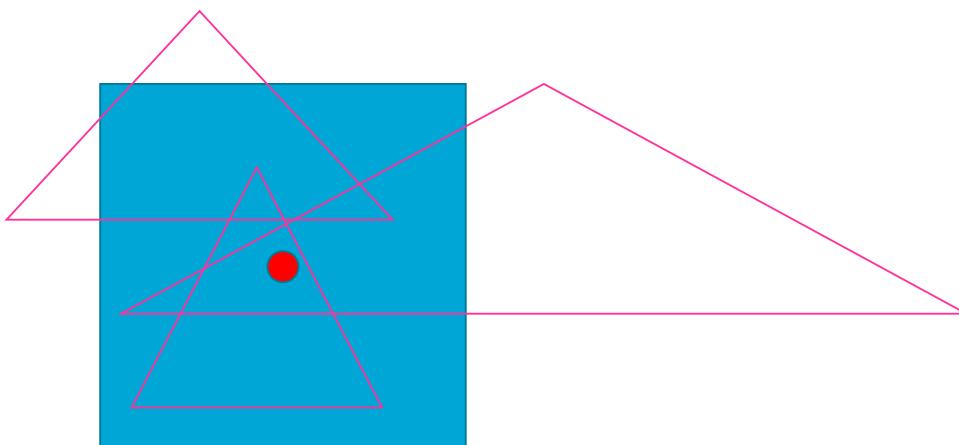
- Rasterization: Fill screen pixels



+ Depth Test

## Simplified Graphics Pipeline

- **Rasterization:** Fill screen pixels
- Pixels are filled if the center is covered



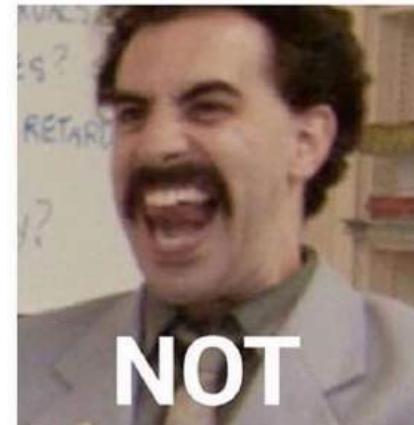
+ Depth Test

# Simplified Graphics Pipeline

Geometric Transformations: Completed



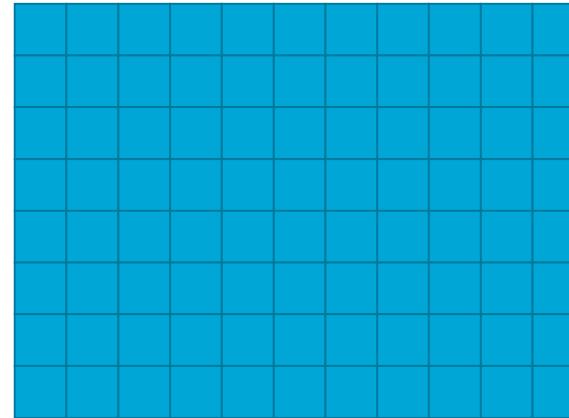
We can make beautiful pictures!



**A first step towards beauty...**

## Rasterization

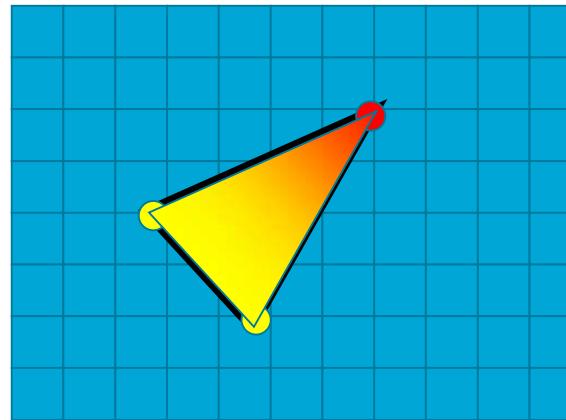
- Triangles can have different colors on vertices



- Values are extracted at pixel centers

## Rasterization

- Triangles can have different colors on vertices



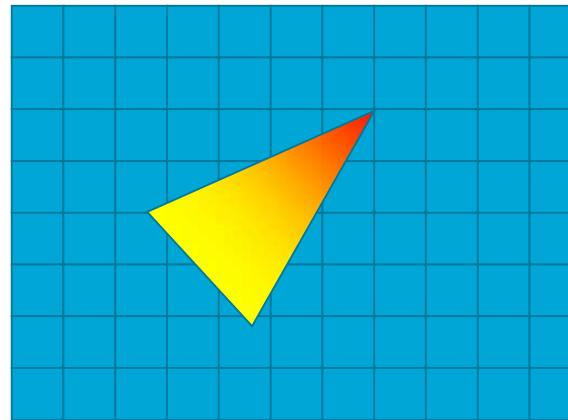
Two yellow,  
one red vertex

Colors are  
interpolated over  
triangle

- Values are extracted at pixel centers

## Rasterization

- Triangles can have different colors on vertices

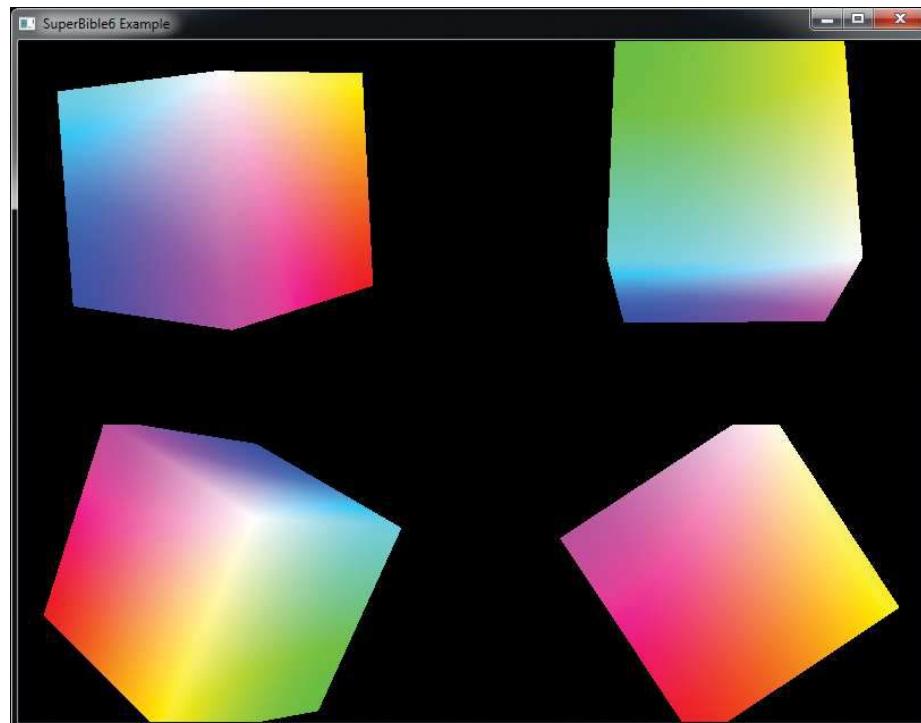


Two yellow,  
one red vertex

Colors are  
interpolated over  
triangle

- Values are extracted at pixel centers

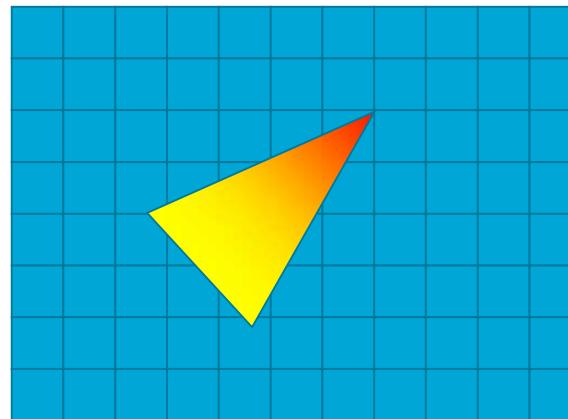
A first step towards beauty...



[OpenGL SuperBible: Comprehensive Tutorial and Reference, 6th Edition](#)

## Rasterization

- Triangles can have different colors on vertices



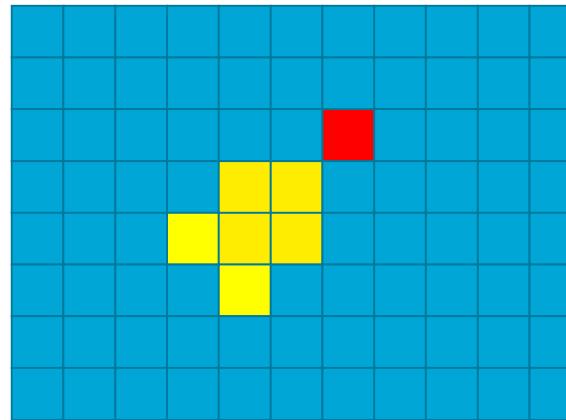
Two yellow,  
one red vertex

Colors are  
interpolated over  
triangle

- Values are extracted at pixel centers

## Rasterization

- Triangles can have different colors on vertices



Colors sampled  
at pixel centers

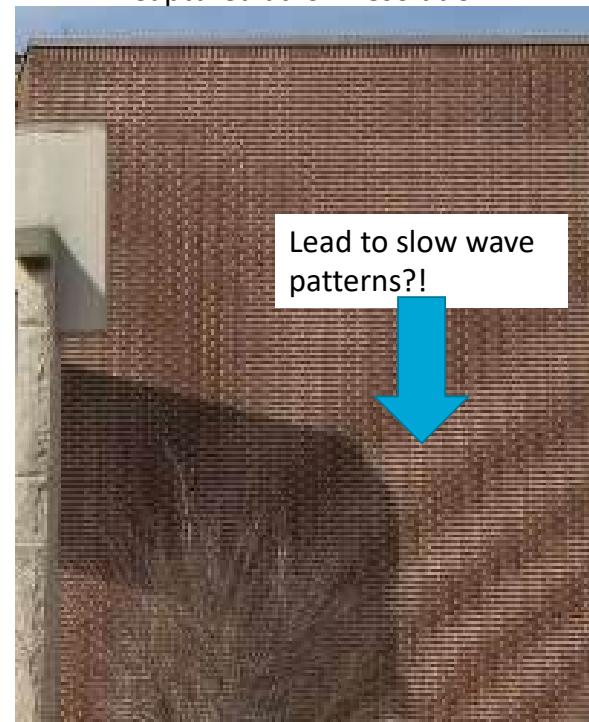
- Values are extracted at pixel centers
- Blocky appearance is referred to as “aliasing”

## Excursion: Aliasing

Many details

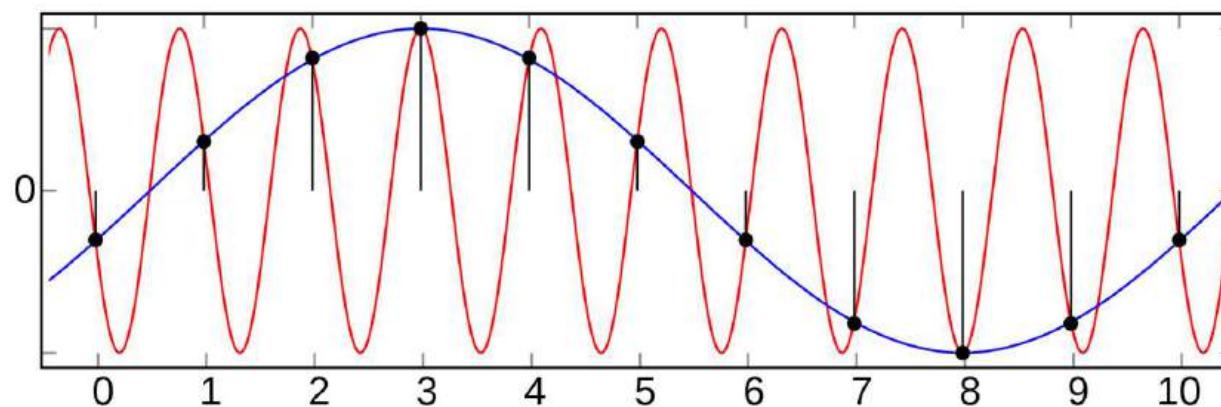


Captured at low resolution



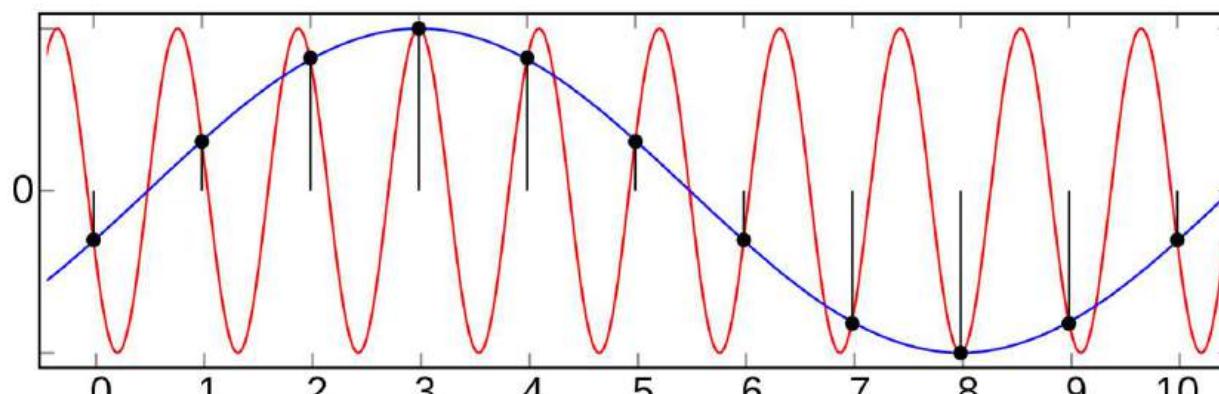
## Excursion: Aliasing

- How would something with lot of details look like a low-frequency variation?



## Excursion: Aliasing

- Two functions can have the same result after sampling



- Too high frequency creates artifacts...

## Excursion: Aliasing

- If pixels are partially covered “aliasing” can occur.



Broken wires because  
their triangles are too small



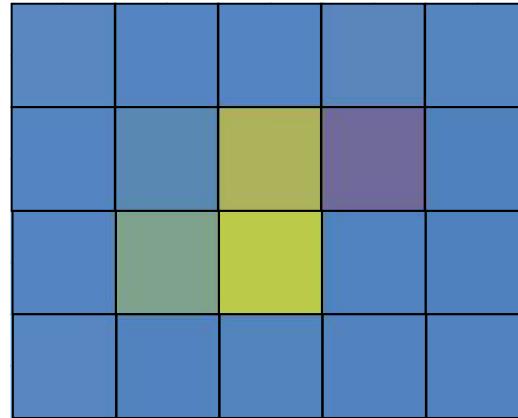
Dirty fix: render at higher  
resolution then reduce  
resolution by averaging pixels  
It is called super sampling.

Images: <http://www.humus.name/>

61

## Excursion: Aliasing

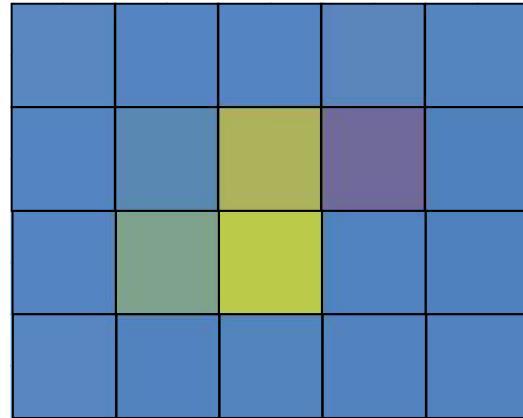
- Antialiasing via supersampling



- Reduce effective resolution by averaging neighboring pixels

## Excursion: Aliasing

- Antialiasing via supersampling



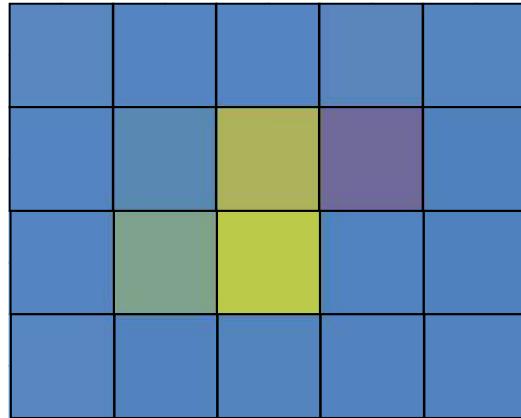
For this we **can** use:

- a) Kung-fu Filter
- b) Karate Filter
- c) Box Filter

- Reduce effective resolution by averaging neighboring pixels

## Excursion: Aliasing

- Antialiasing via supersampling



- Reduce effective resolution by averaging neighboring pixels

For this we **can** use:

- a) Kung-fu Filter
- b) Karate Filter
- c) Box Filter

**Although later in your studies, you will see that better filters exist**

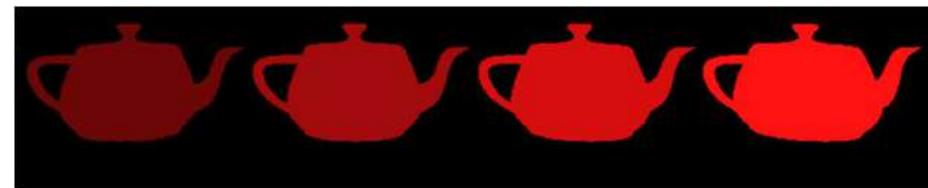
...

Images are not very exciting yet...



# Shading

How to transform

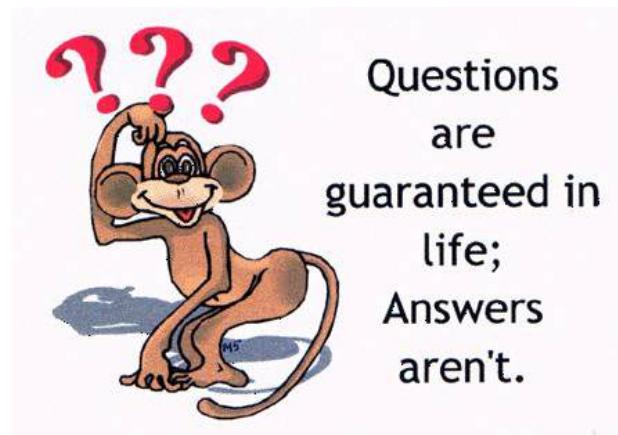


in



?

## Questions?



Questions  
are  
guaranteed in  
life;  
Answers  
aren't.

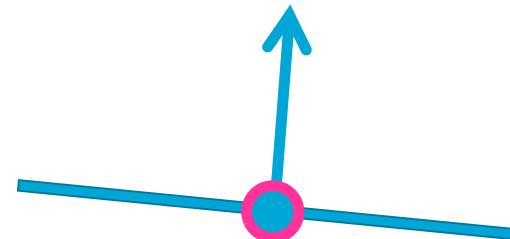
## Today's Learning Goals

- S4- Apply mathematical modeling and theory of geometric computations and transformations, object representations, **simulation**, and encoding.
  - We have seen a transformation recap
  - You will learn how to derive and calculate a simple physically-oriented material model.
- S3- Use mathematical methods to analyze, create, apply algorithms and **data structures, as well as understanding time** and space **complexity** of image-generation algorithms
  - You will see various ways of calculating the effect of light on a mesh

## Making beautiful pictures...

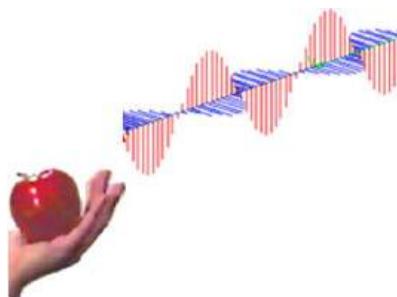
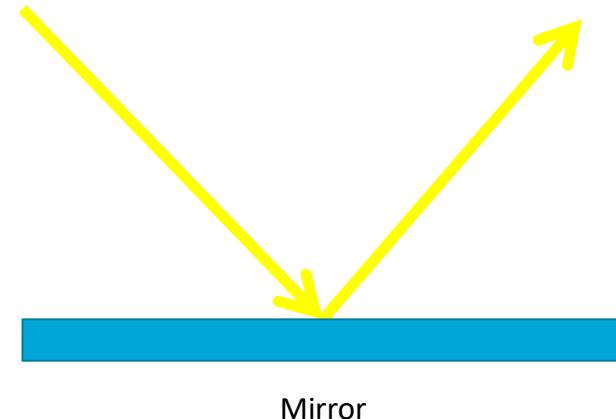
TODAY'S QUESTION:

- Given a surface point  
(position, its normal, and potential attributes)  
and a point emitting light...
- How to compute a realistic color???



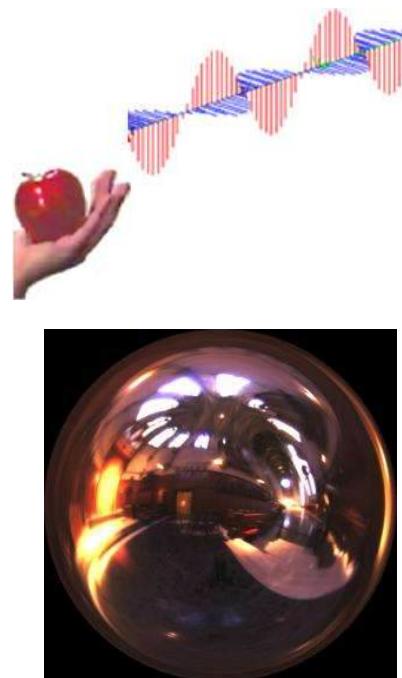
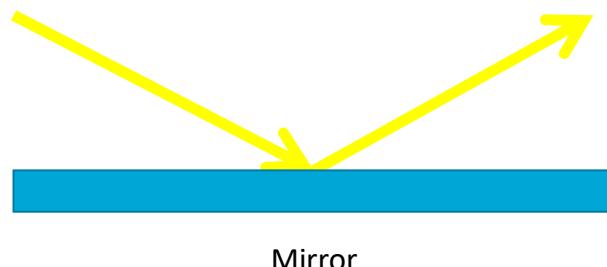
## Reflection

- What happens when the light hits a surface?



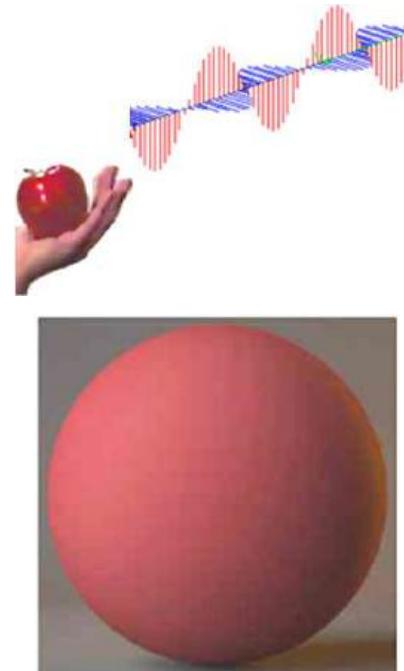
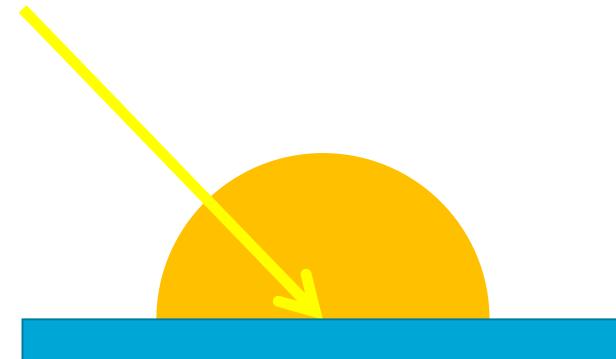
## Reflection

- What happens when the light hits a surface?



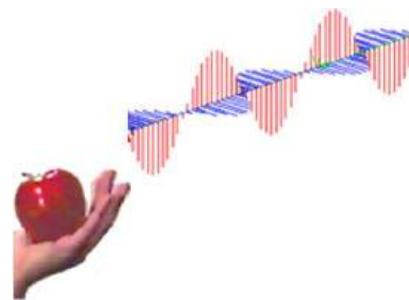
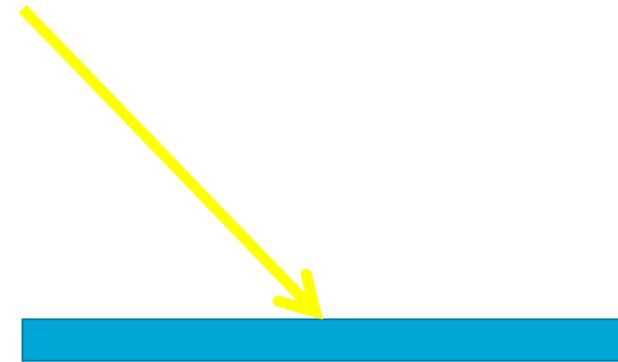
## Reflection

- What happens when the light hits a surface?



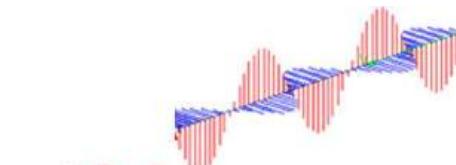
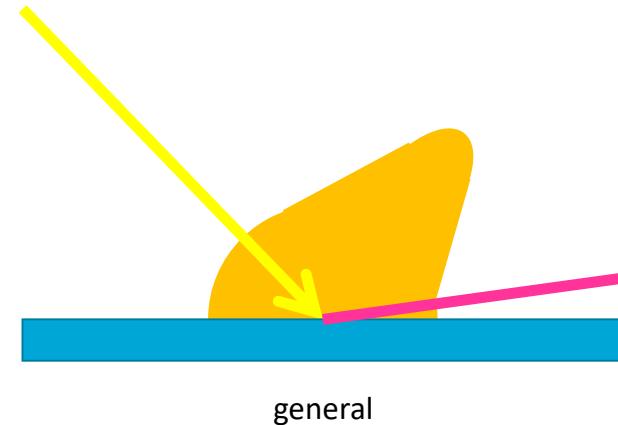
## Reflection

- What happens when the light hits a surface?



## Reflection

- What happens when the light hits a surface?



Today:  
Mostly interested in the  
direction towards the  
camera!

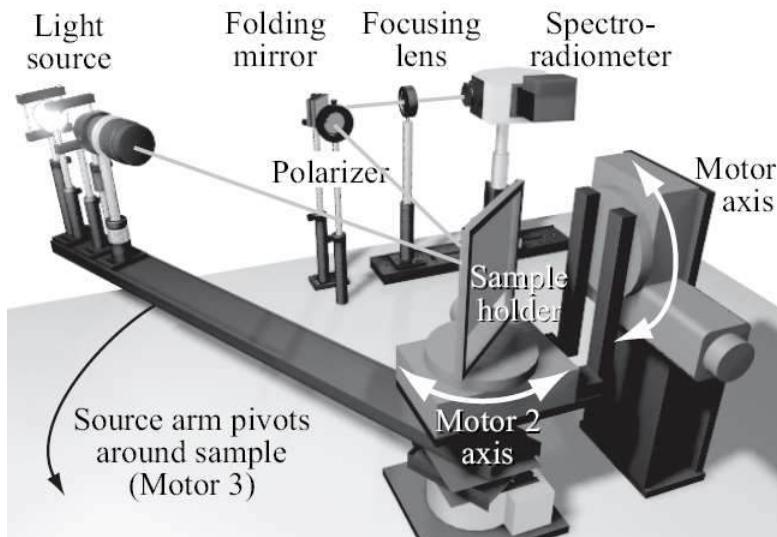
## Acquired Materials

- Big databases



## Material Acquisition

- Use a gonioreflectometer – yes, that is the name...



<http://www.graphics.cornell.edu/~westin/>



## Acquired Materials

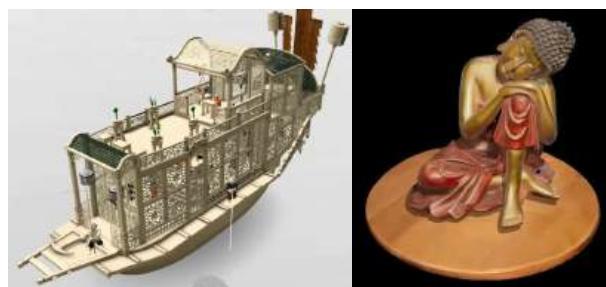
- Big databases

Often costly, much data

Hard to use for artists

e.g., “Can you make the blue darker?”

Very important when  
acquiring real-world objects



## Mathematical Models

- Describe light interaction as a function
- Usually more lightweight
- Has parameters to control appearance
- Acquired materials can be approximated



Published recently:  
Effect of the additional glaze layer  
(left) that Vermeer placed over the  
black background.  
[graphics.tudelft.nl/WebPearl/](http://graphics.tudelft.nl/WebPearl/)

## **Girl With A Pearl Earring**

Exhibition in the Mauritshuis

until 8<sup>th</sup> of January 24

World's largest 3D print

Time travel to the creation of the girl



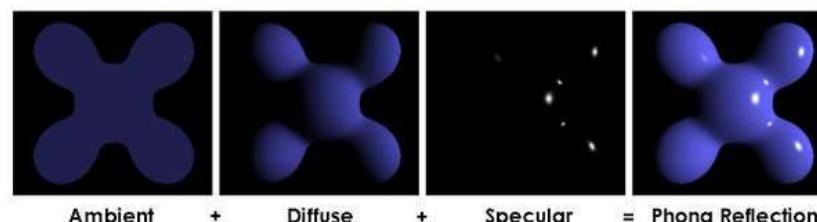
vlnr: Elmar Eisemann (TUD), Ruben Wiersma (TUD), Lucie Oude Luttikhuis (TUD),  
Liselore Tissen (TUD/UL) Mane van Veldhuizen (UVA), Abbie Vandivere (MH),  
Mathijs Lefferts (TUD), Emilien Leonhardt (Hirox), Clemens Weijkamp (Canon),  
Camil Rejhons (Canon)

## Simplified Models

- Mathematically describe Material Properties

- Phong Model: Sum of 3 terms

- Ambient
- Diffuse
- Specular



## Color - Recap

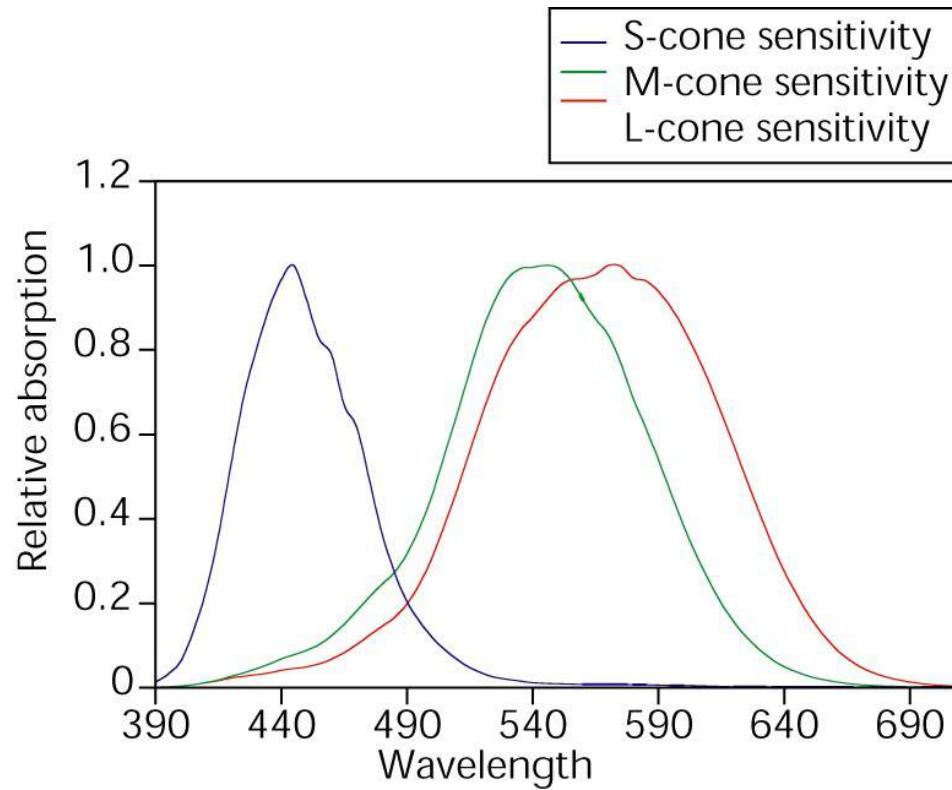
- Remember:

Visual system uses 3 cone types for color

In our model, we will treat wavelengths separately (in practice: **Red, Green, Blue**).

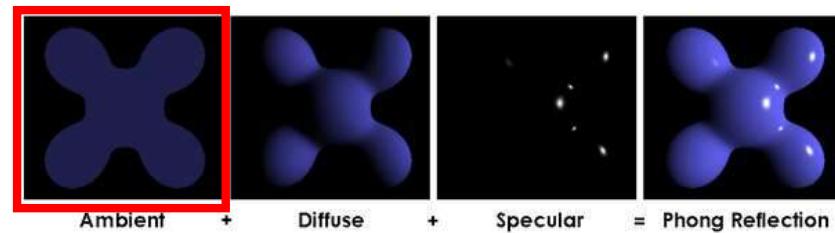
In the following, we usually describe the model for a single wavelength/color channel  
(do it 3 times for red, green, blue...)

## 3 Cone types



## Phong Model

- Sum of 3 terms
  - Ambient
  - Diffuse
  - Specular



## Ambient Term

- Is supposed to mimic “scene light”:
  - Skylight
  - Reflections from neighboring surfaces

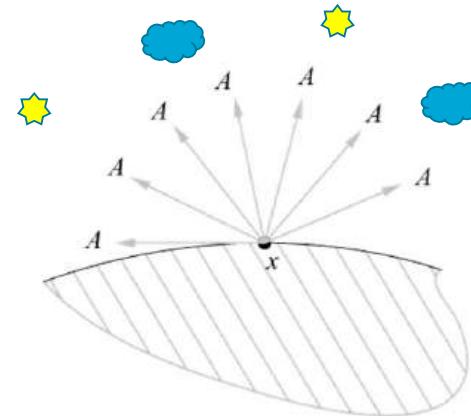
- Model:

- Very simple:

Formula is used for Red,  
Green & Blue.

$$A = I_a K_a$$

Light property  
↓  
 $A$   
↑  
Surface property



$$A = I_a K_a$$



## Ambient Term



$K_a$  increasing

$$A = I_a K_a$$



## Ambient Term

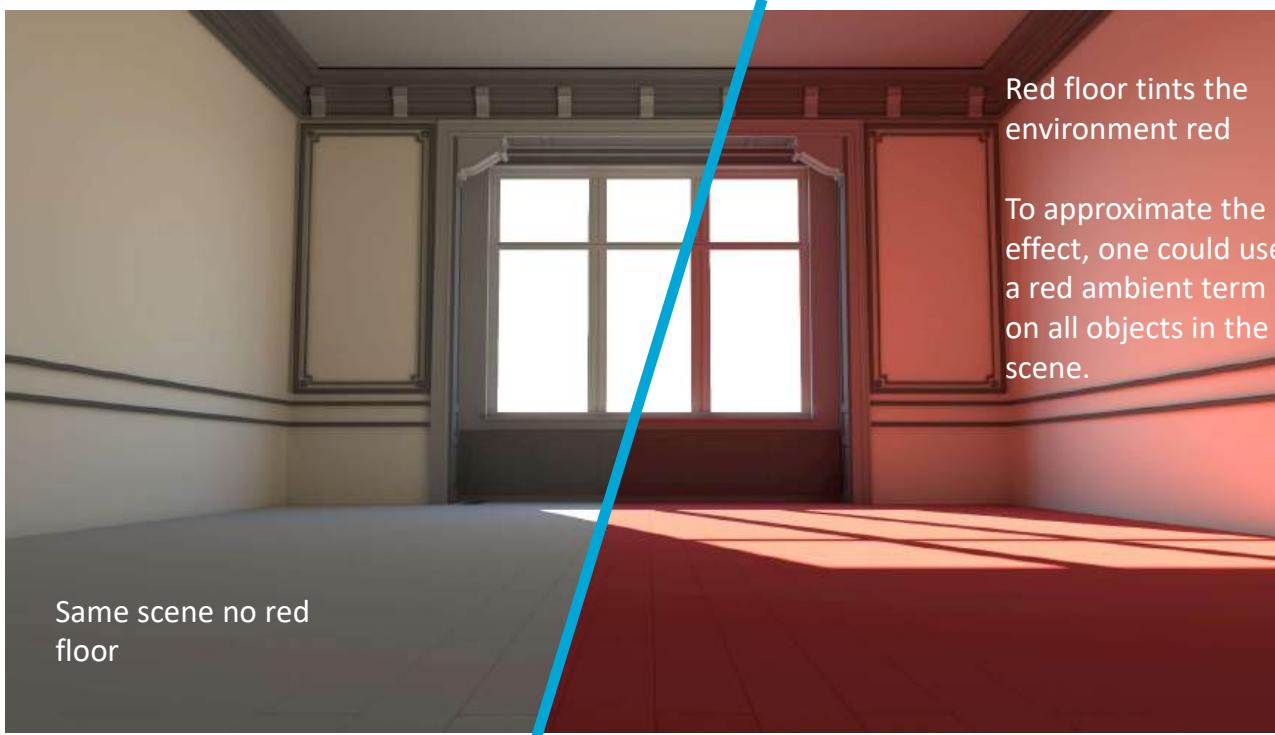
- Very simplistic
  - No indications on the shape of an object!



$K_a$  increasing

- Used often in practice as a strong approximation of indirect light

## Example of Indirect Light



$$A = I_a K_a$$



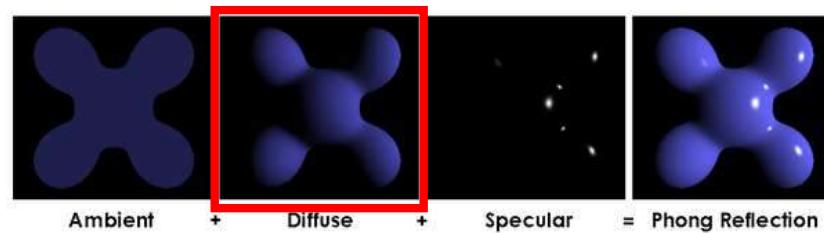
## Example Ambient

- Typically, values between [0,1] are used
- $I_a = (0.9,0.9,0.9)$  “white light”  
(Red, green and blue are close to one)
- $K_a = (0.5,0,0)$  the surface is “dark red”
- The ambient term is?

$$A = I_a K_a = (0.9,0.9,0.9) * (0.5,0,0) = (0.45,0,0) \quad \text{...also dark red.}$$

## Phong Model

- Sum of 3 terms
  - Ambient
  - Diffuse
  - Specular

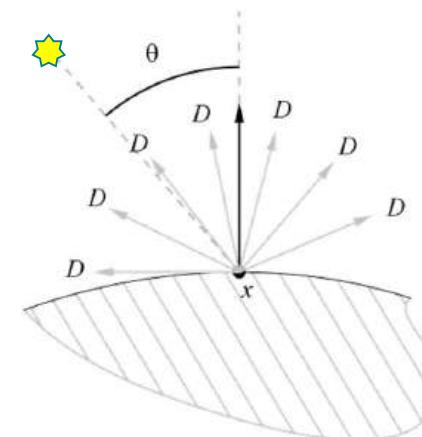


## Diffuse Term

- Lambert Surfaces
  - Light is reflected uniformly in all directions
- Model:
  - Uses local surface orientation

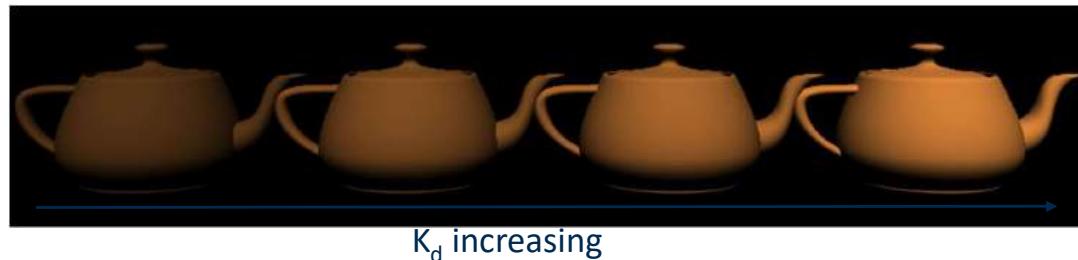
$$D = I_d K_d \cos \theta$$

Light property (RGB)      Surface property (RGB)



## Diffuse Term

$$D = I_d K_d \cos \theta$$



## Diffuse Term

$$D = I_d K_d \cos \theta$$

- Shading varies along surface
  - Gives information about shape

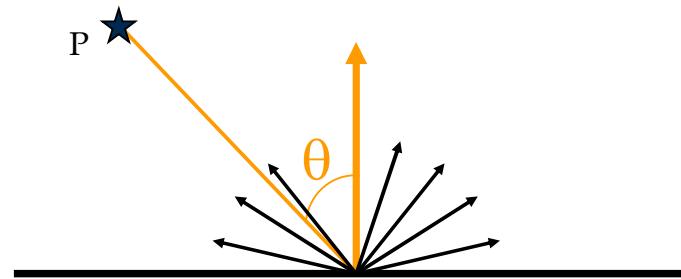


- Does not depend on observer position  
(looks the same from all positions)
- **Careful:**  
the light should always come from above the surface, otherwise, it should stay black.  
**What does this mean for the angle  $\theta$ ?**

## Diffuse Term

- Where does the cosine come from?

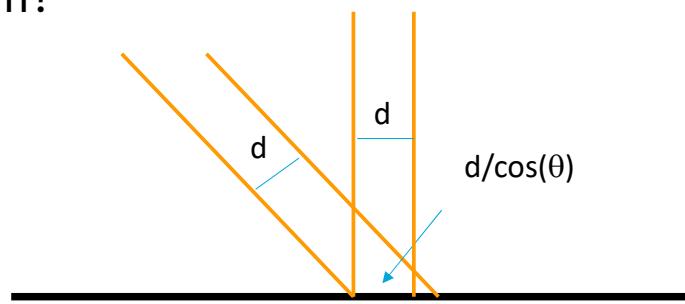
$$D = I_d K_d \cos \theta$$



## Diffuse Term

- Where does the cosine come from?

$$D = I_d K_d \cos \theta$$



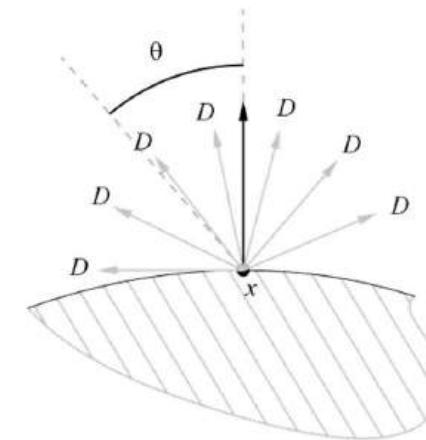
- What do you observe when tilting a flashlight?
- Imagine light arriving as parallel rays

## Example

$$A = I_a K_a$$

$$D = I_d K_d \cos \theta$$

- $K_d = (0.9, 0, 0)$
  - $I_d = (0.9, 0.5, 1.0)$
  - $K_a = (0, 0, 0.1)$
  - $I_a = (1, 1, 0.1)$
- 
- Normal at x:  $(0, 0, 1)$
  - Position x:  $(0, 0, 0)$
  - light position :  $(0, 0, 10)$
  - Resulting color:

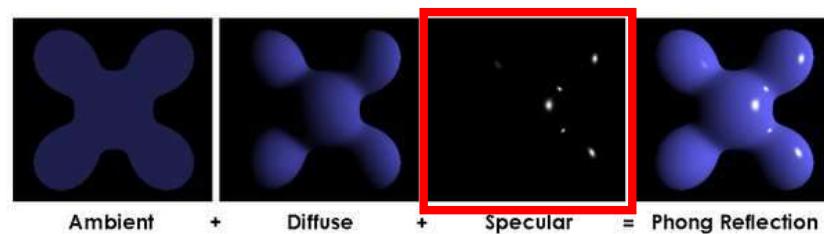


$(0.81, 0, 0.01)$



## Phong Model

- Sum of 3 terms
  - Ambient
  - Diffuse
  - Specular

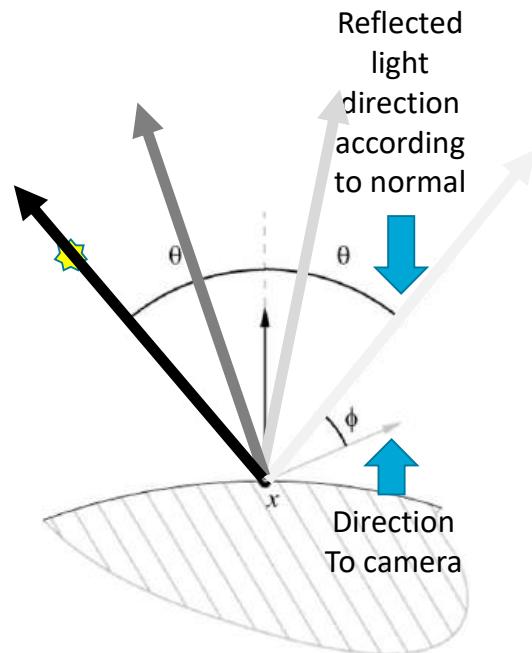


## Specular Term

- Represent glossy surfaces
  - Ideal case: mirror
- Model:
  - reflection around mirrored ray
  - Falloff around perfect reflection

$$S(\phi) = I_s K_s (\cos \phi)^n$$

*n : shininess*

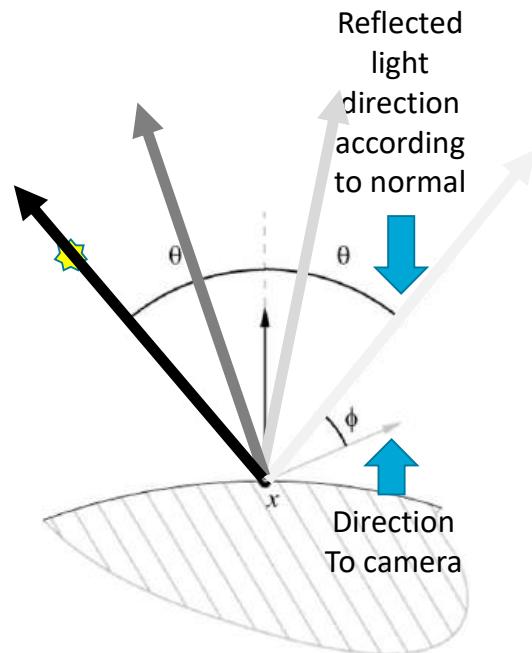


## Specular Term

- Represent glossy surfaces
  - Ideal case: mirror
- Model:
  - reflection around mirrored ray
  - Falloff around perfect reflection

$$S(\phi) = I_s K_s (\cos \phi)^n$$

*n : shininess*

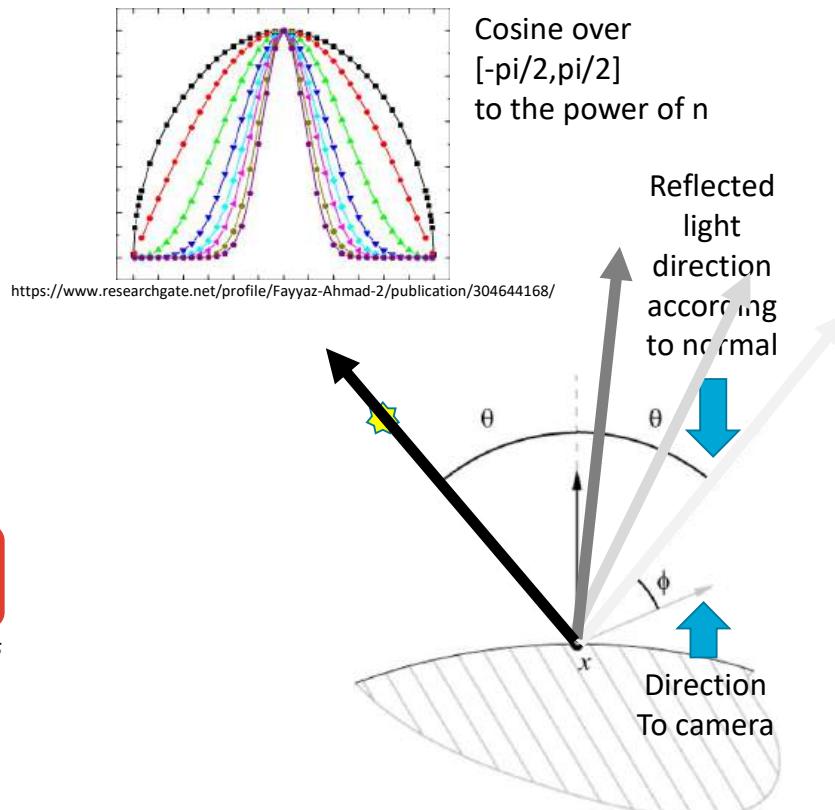


## Specular Term

- Represent glossy surfaces
  - Ideal case: mirror
- Model:
  - reflection around mirrored ray
  - Falloff around perfect reflection

$$S(\phi) = I_s K_s (\cos \phi)^n$$

*n : shininess*

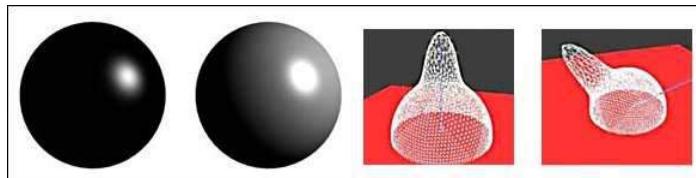


## Specular Term

- Represent glossy surfaces
  - Ideal case: mirror
- Model:
  - reflection around mirrored ray
  - Falloff around perfect reflection

$$S(\phi) = I_s K_s (\cos \phi)^n$$

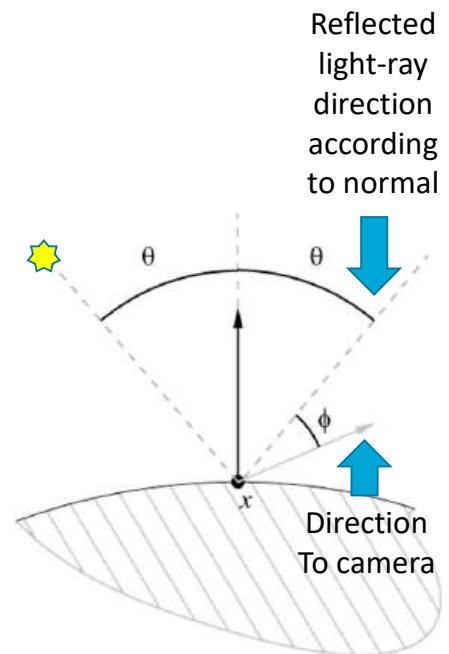
*n : shininess*



specular

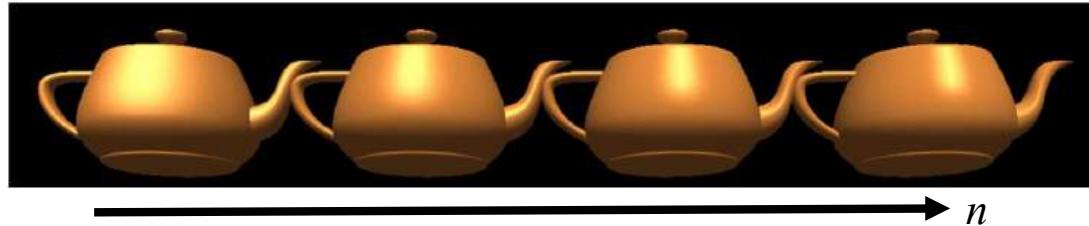
specular+diffuse

$S(\phi)$ , called the *lobe*



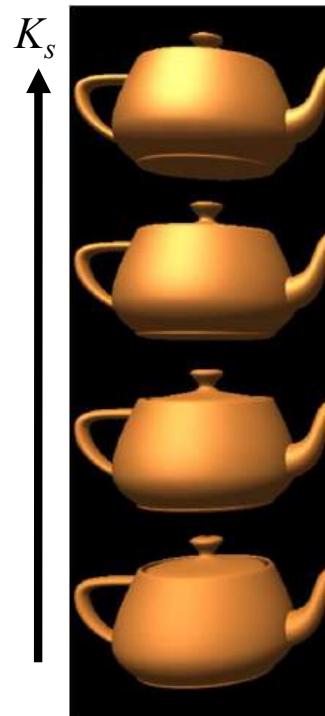
## Specular Term

$$S(\phi) = I_s K_s (\cos \phi)^n$$



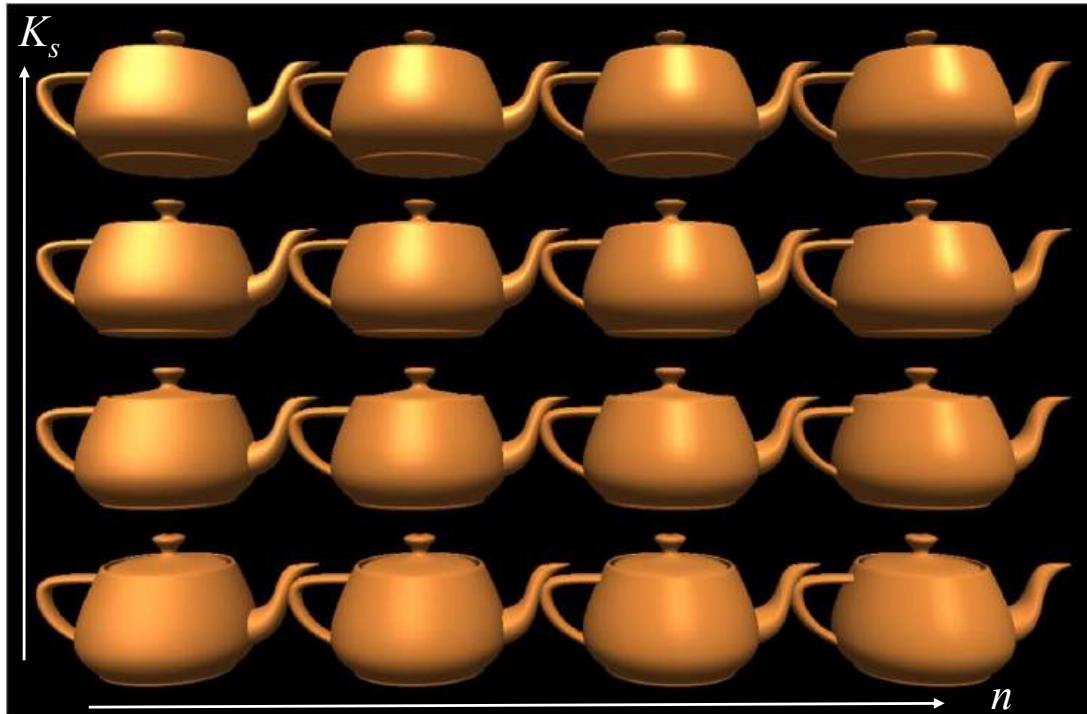
## Specular Term

$$S(\phi) = I_s K_s (\cos \phi)^n$$



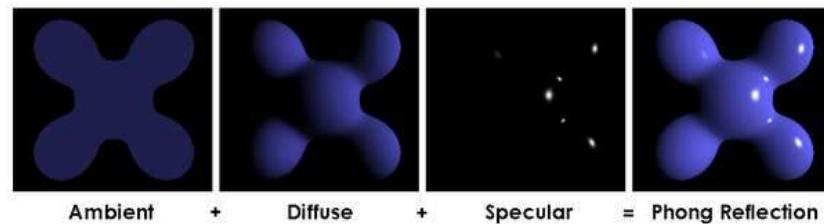
## Specular Term

$$S(\phi) = I_s K_s (\cos \phi)^n$$



## Phong Model

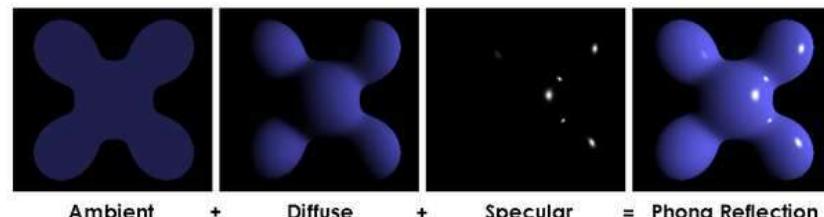
- Sum of 3 terms
  - Ambient
  - Diffuse
  - Specular



## Phong Model

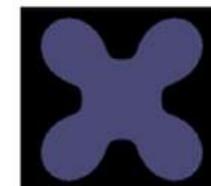
- Sum of 3 terms

- Ambient
- Diffuse
- Specular



- Optional Extension:

- Emission = Ambient with a Light set to 1
  - Idea: object is emitting light (e.g., hot glowing metal)

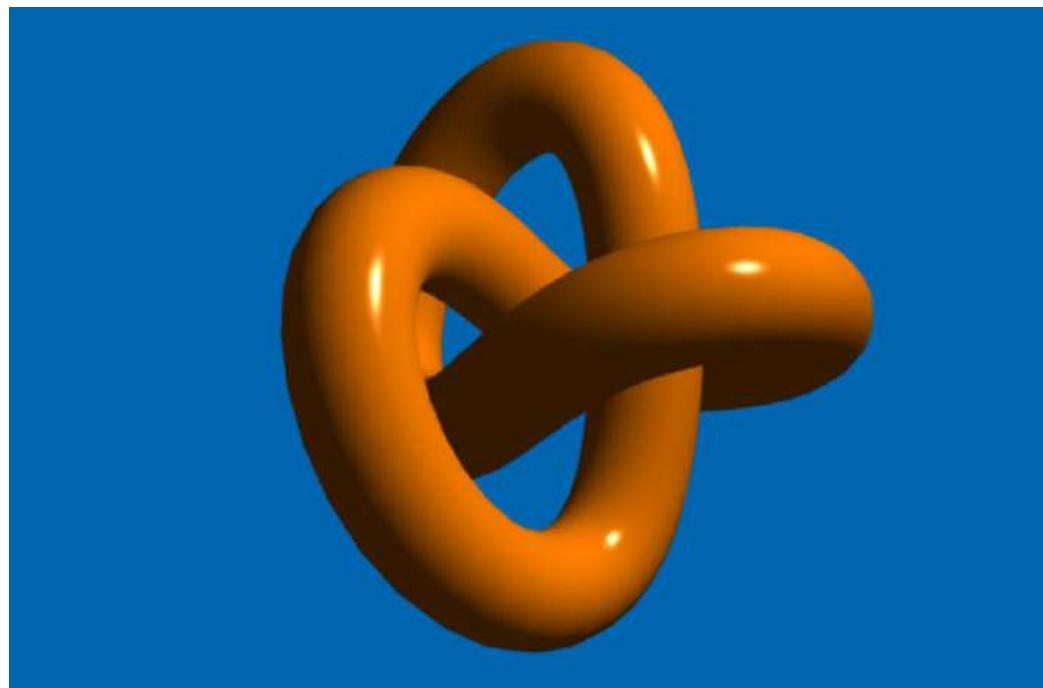


## Demo Time



## Demo Time

- <http://multivis.net/lecture/phong.html>

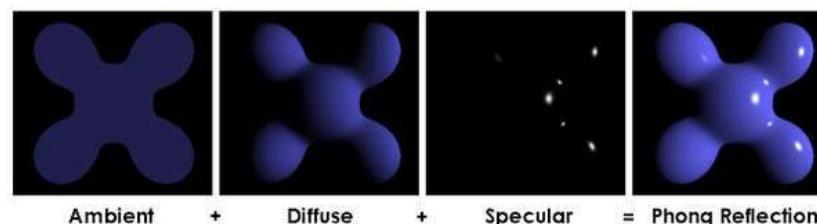


## Mathematical Models

- Mathematically describe Material Properties

- Phong Model: Sum of 3 terms

- Ambient
- Diffuse
- Specular



- In the literature: Many more material models

Tradeoff between efficiency and accuracy

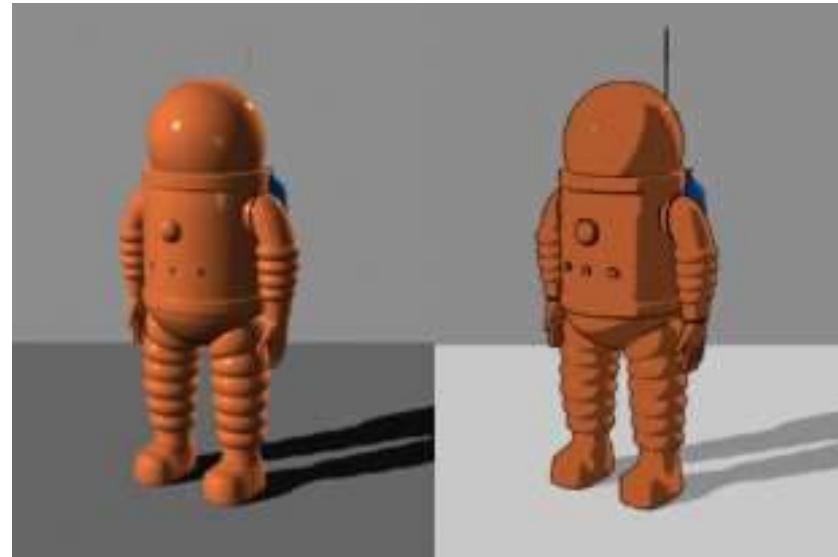
## Advanced Material Models



<https://blog.playcanvas.com/physically-based-rendering-comes-to-webgl/>

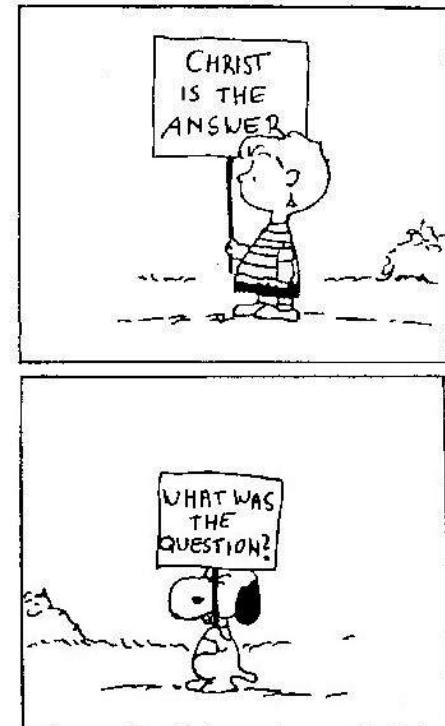
## Extreme Example...

- Non-photorealistic materials



- Cel-shading: threshold on the diffuse shading

## Questions?



## How to apply Phong Model ?

- We know how to compute shading of a point,  
but how is it applied on a triangle mesh?

## Shading

- Early days - compute color per face:  
*Flat shading* produces “facets”

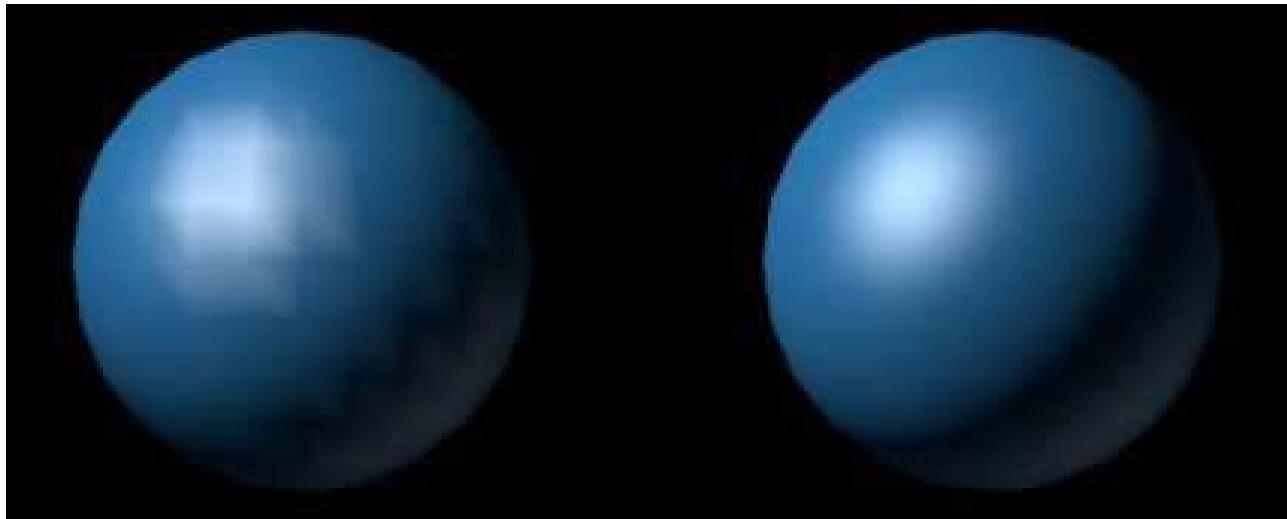


- Later – compute color per vertex:  
*Gouraud Shading* produces a smooth look



## Phong shading

- Today: compute result per pixel
- Phong Shading leads to smooth specularities



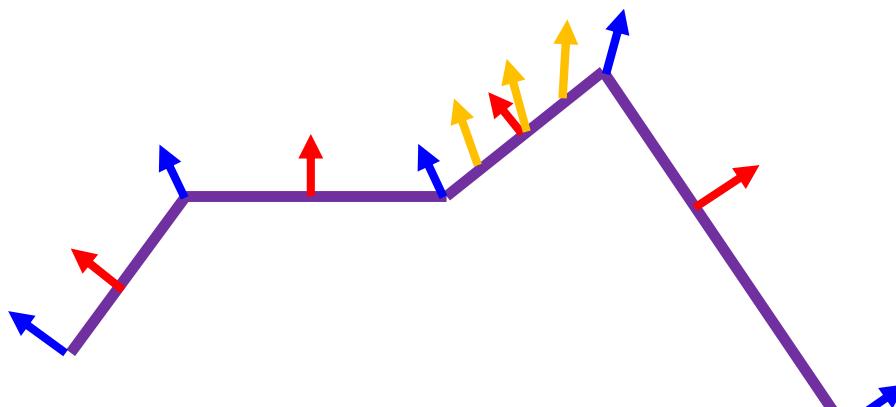
Diccan.com

- Phong interpolates normals from vertices over pixels

124

## Normals on Meshes

- Face normals (normal of the plane containing triangle)
- Vertex normals (e.g., average neighboring face normal)
- Interpolated normal (interpolate vertex normals over triangle)



*Per vertex*

*Per pixel*

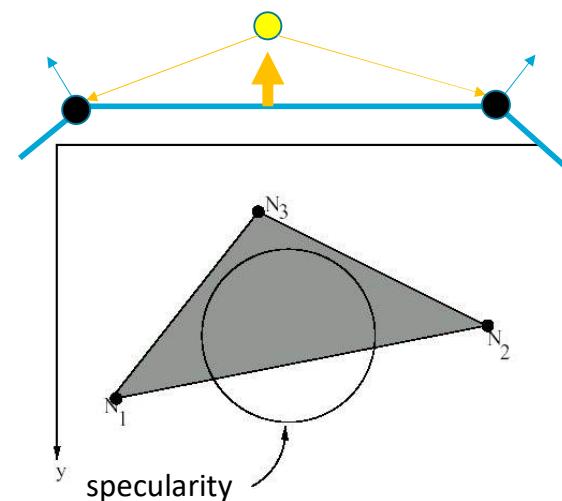


## Gouraud vs. Phong shading

- Phong usually more expensive than Gouraud
  - Because there are often more pixels than vertices
- Phong is more beautiful and minimal standard
  - Captures specularities between faces



Diccan.com



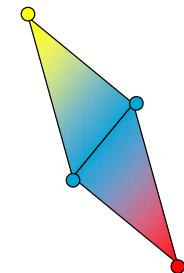
126

TU Delft

## On the practical side: Shading types

- How are the three different types computed?

- *Flat shading*
  - Applies Phong Model to produce a color per face
- *Gouraud shading*
  - Applies Phong to produce a color per vertex
  - Interpolate color from vertices over triangle
- *Phong shading*
  - Interpolate parameters of Phong model
  - Applies Phong to produce a color per pixel



2 MEANINGS!!!

**Questions?** when your lecturer asks if you have any questions



Jemima Skelley / BuzzFeed

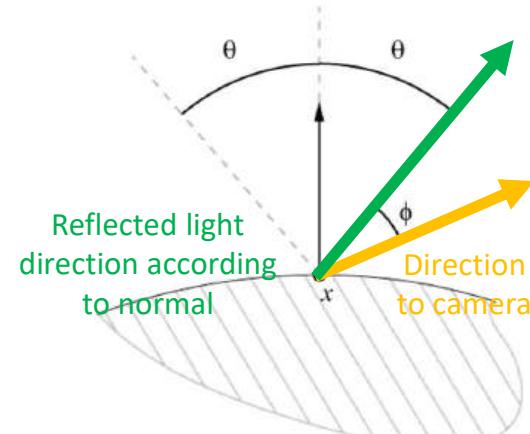
128

## Summary

- Graphics Pipeline Core:
  - Geometry
    - Transformation and Projection
  - Shading
    - Material model (Ambient, Diffuse, Specular)
    - Shading interpolation (Flat, Gouraud, Phong)

## Additional Exercises

$$S(\phi) = I_s K_s (\cos \phi)^n$$

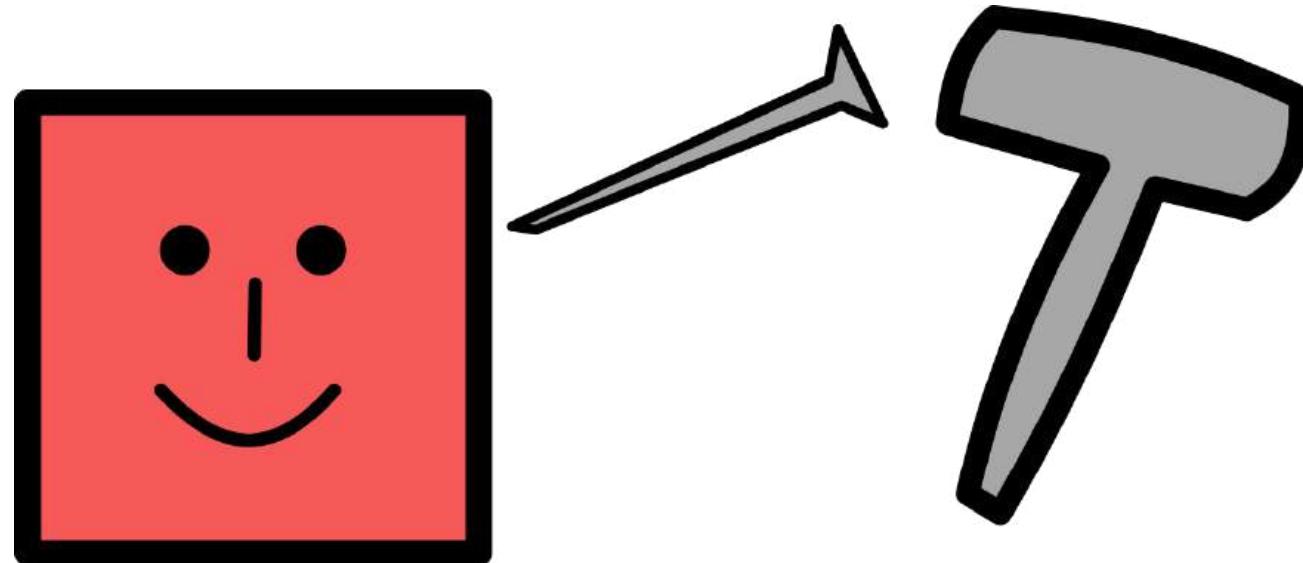


- The specular part of the Phong model uses the angle between the reflected light direction (according to the normal) and the direction to the camera.
- Calculate the reflected light direction.

**Thank you very much  
for your attention!**

# OpenGL : how to make a pixel

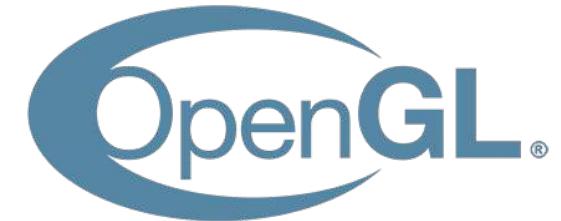
CSE2215 Computer Graphics



Ricardo Marroquim

Delft University of Technology (TU Delft)

today



- **Graphics Pipeline with OpenGL**

- model representation
- transformation
- rasterization
- interpolation

**last lectures in a nutshell!**

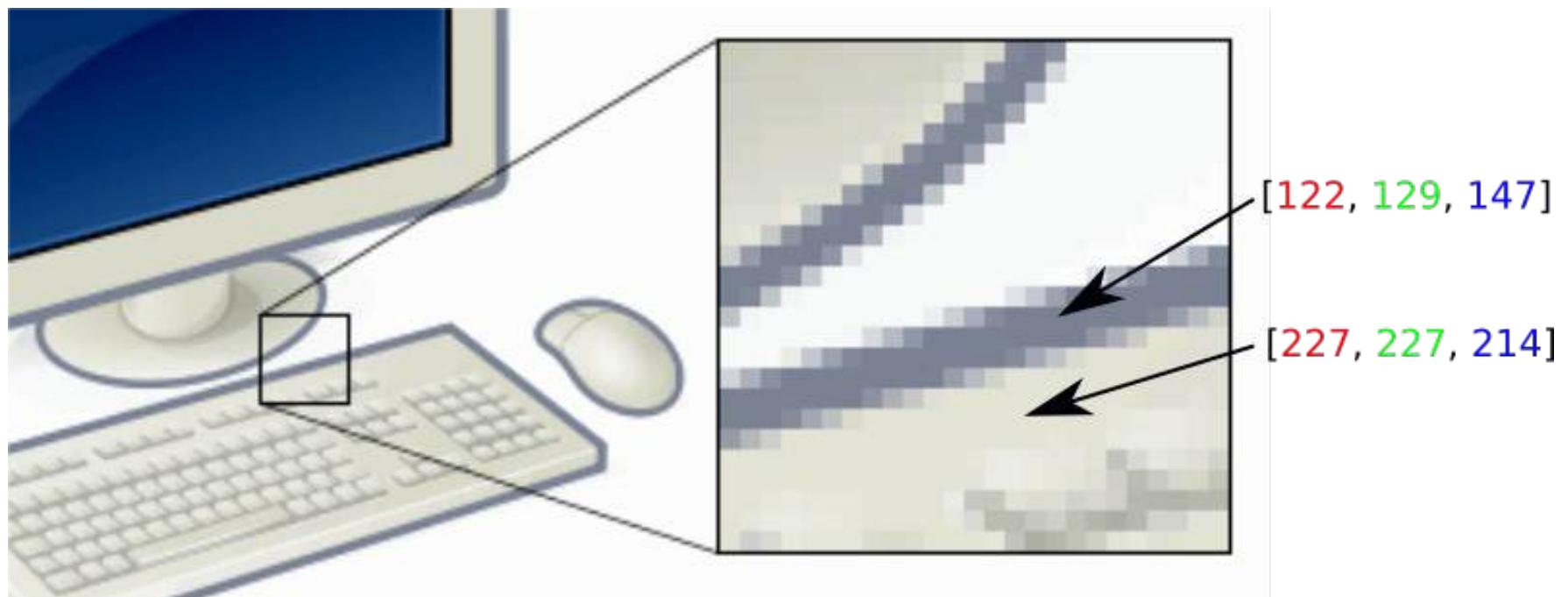


# study goals

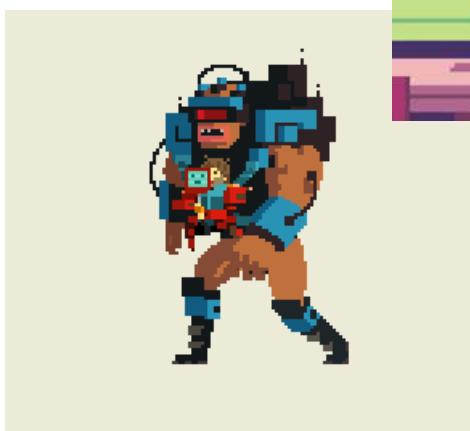
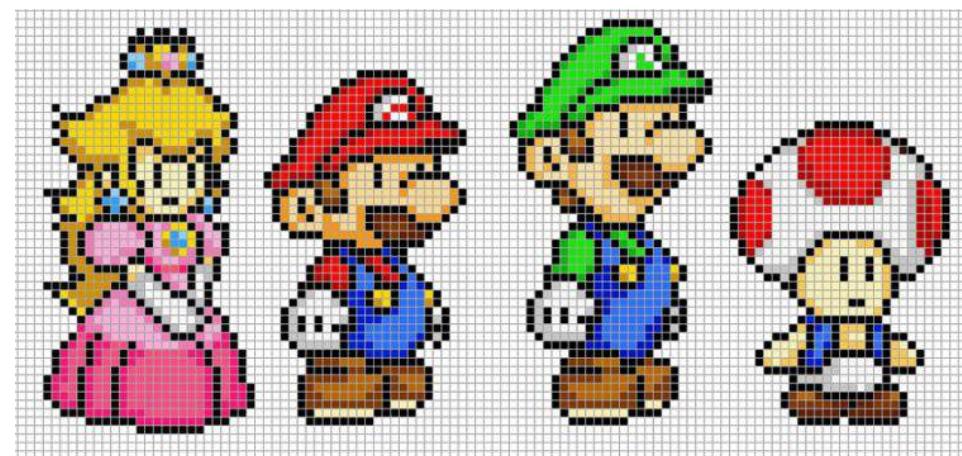
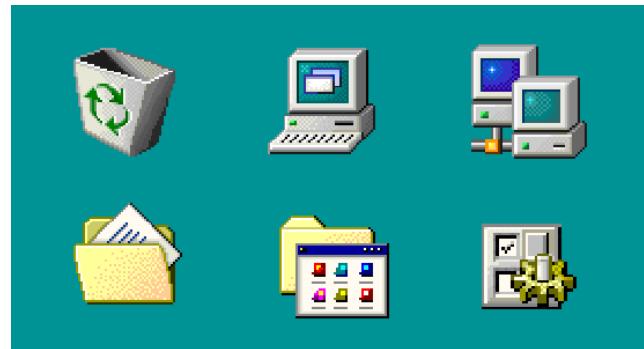
- S4- Apply mathematical modeling and theory of geometric computations and transformations, object representations, simulation, and encoding.
- S5- Implement algorithms and data structures using the C++ programming language and OpenGL.

# in its simplest form

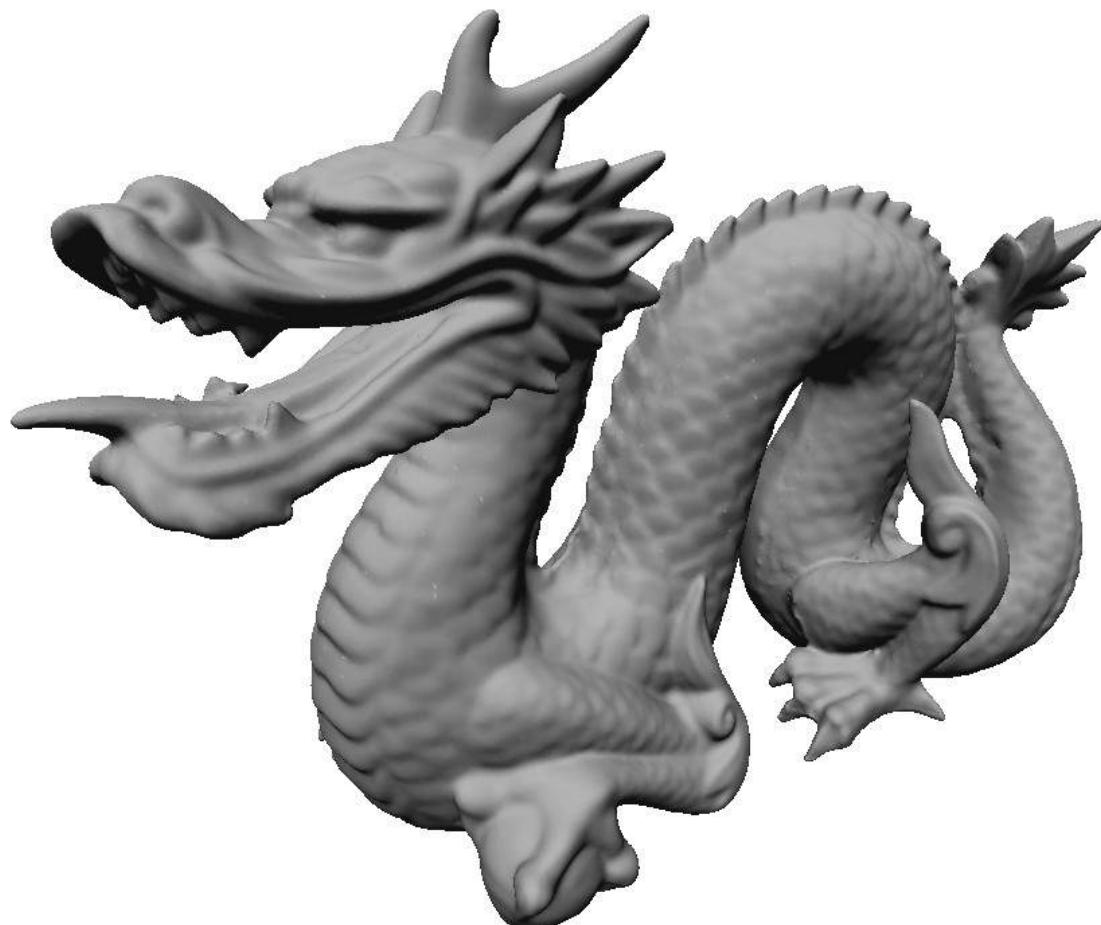
- our task is to color pixels RGB [0,255]



# pixel art

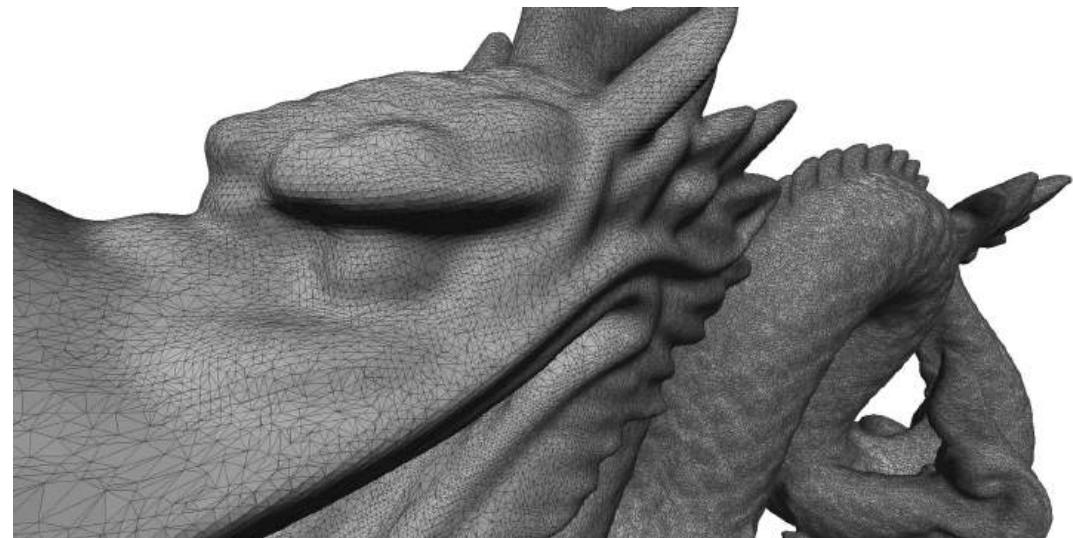
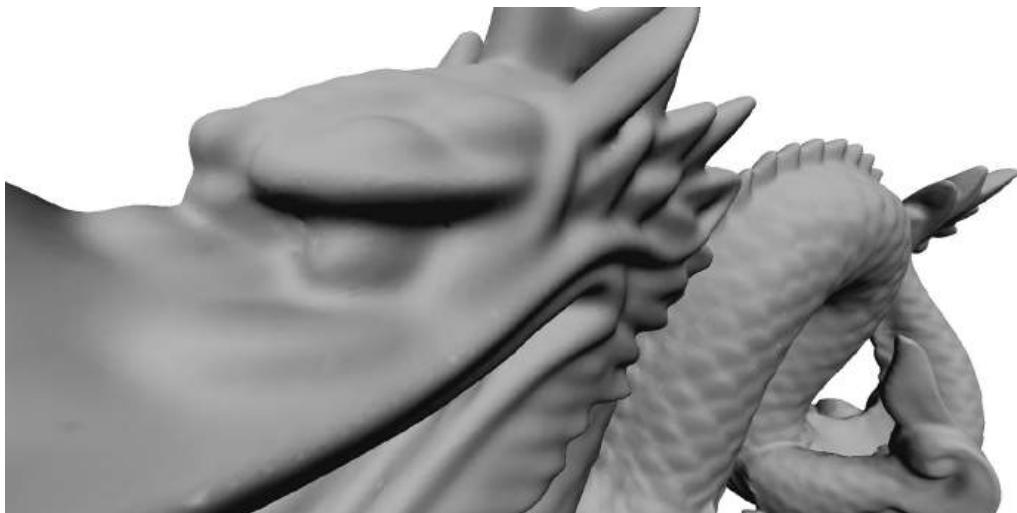


# models

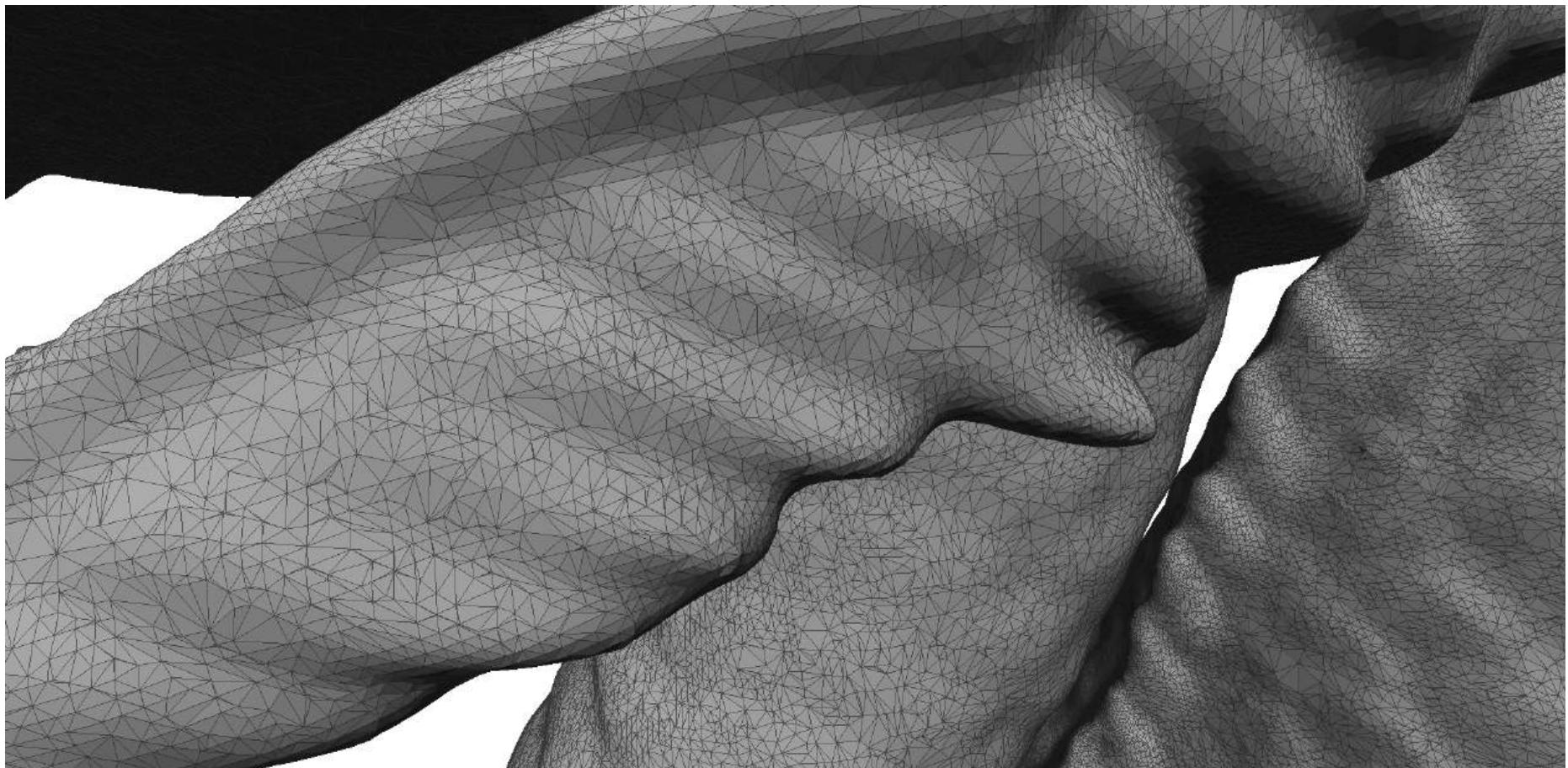


# triangles

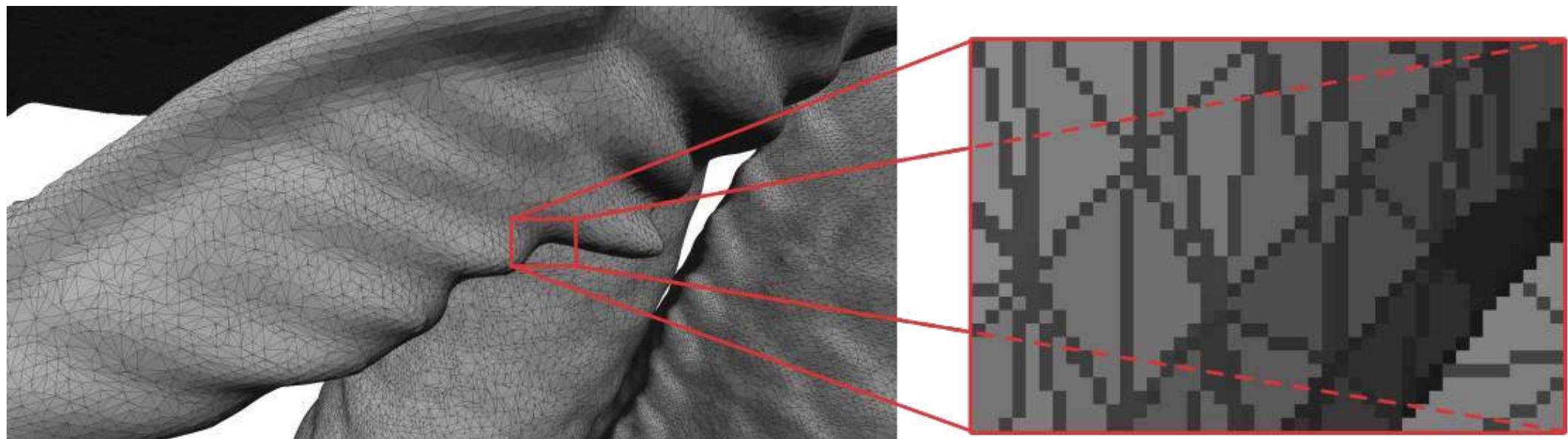
- **most** common representation

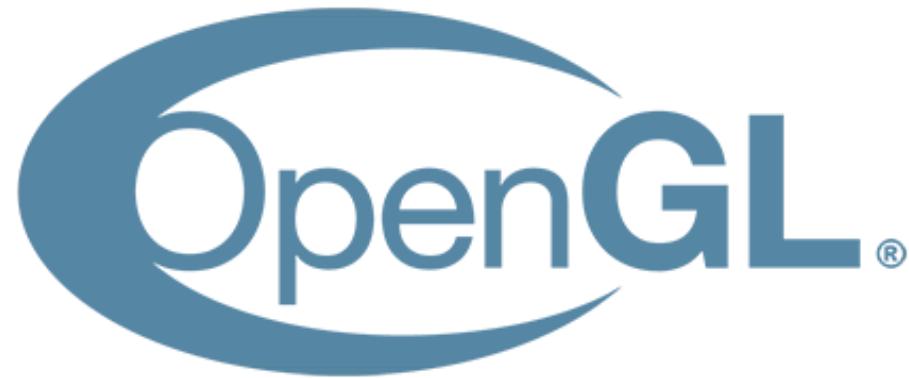


# triangles

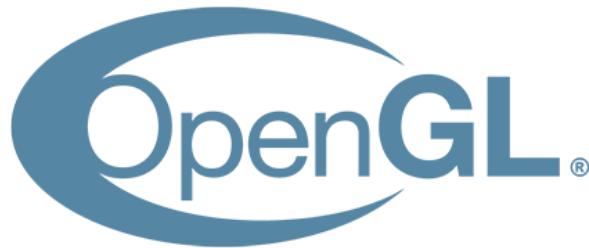


# triangles





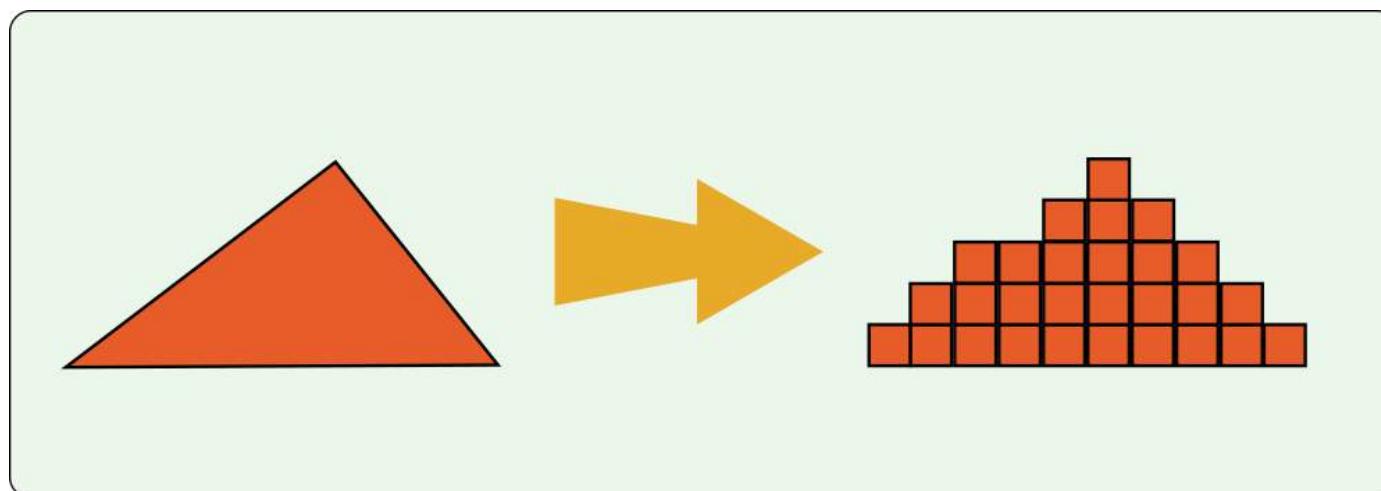
- **Open Graphics Library**
  - **rendering** API : draw primitives on the screen (e.g. triangles)
  - actually it is an **abstract** API (specification)
    - usually implemented by hardware driver
    - OpenGL 1.0 (1992) → OpenGL 4.6 (2017)
  - does not handle your windows, needs libs (e.g. GLFW)



- **Open Graphics Library**
  - around 1999 the first “programmable” GPU by NVIDIA
  - with new generations → more “programmability”
  - “modern” OpenGL: after 3.0 (2008)
    - more access to hardware features
  - but we stick to old school for now

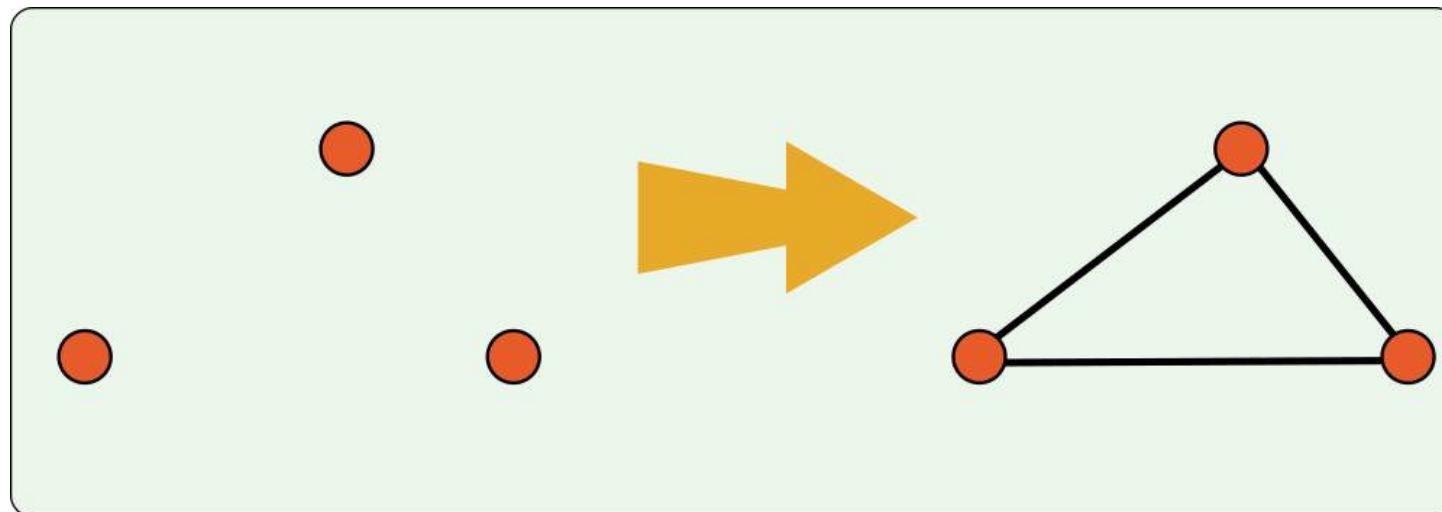
# basic idea

- transform triangle to pixels
- **rasterization**



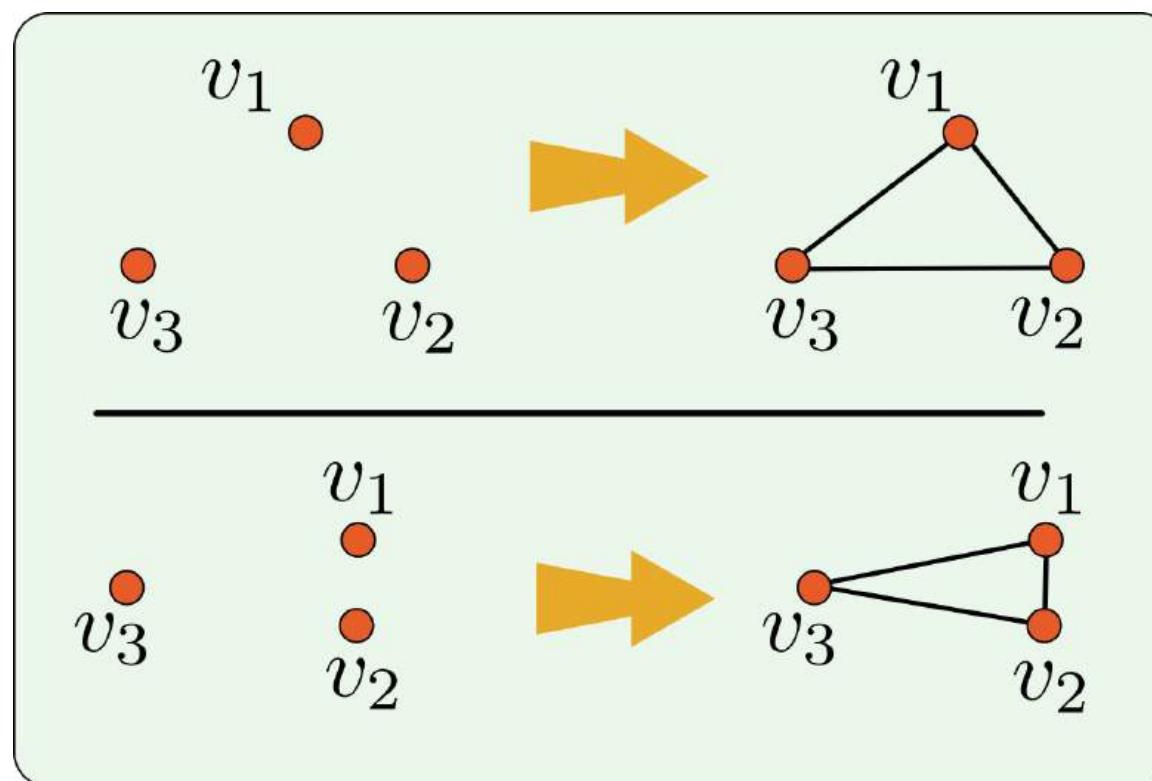
# first point

- basic element is a **vertex**
  - triangles are just 3 connected vertices

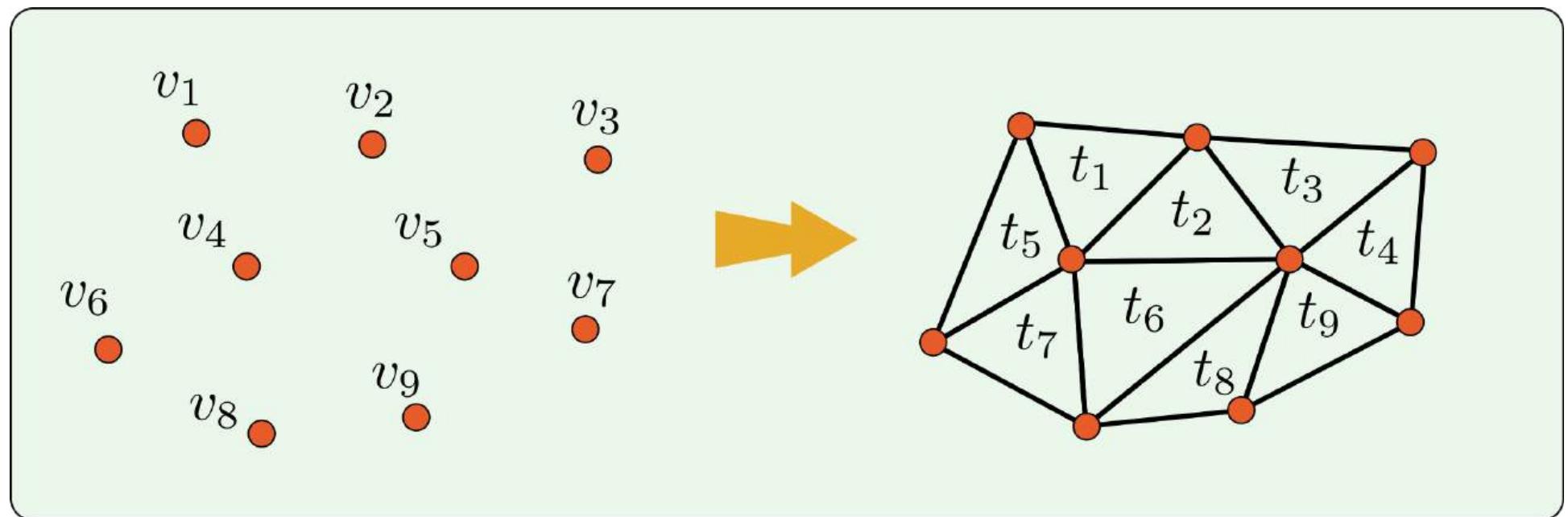


# first point

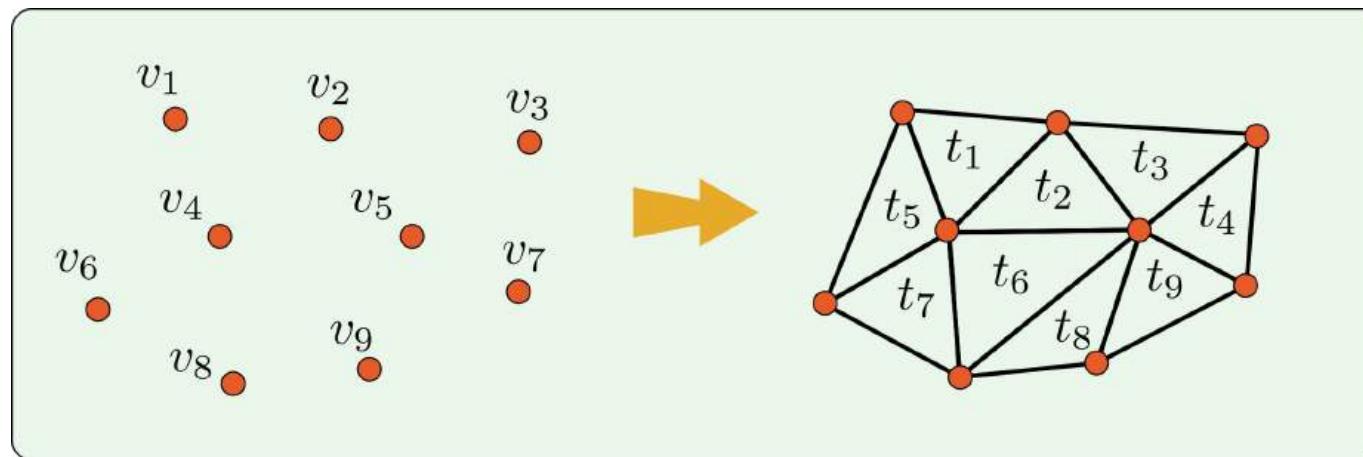
- modify vertices → new triangle is formed



# basic structure



# basic structure



vertices

$$v_1 = \{x_1, y_1, z_1\}$$

$$v_2 = \{x_2, y_2, z_2\}$$

$\vdots$

$$v_n = \{x_n, y_n, z_n\}$$

triangles

$$t_1 = \{v_1, v_2, v_4\}$$

$$t_2 = \{v_2, v_5, v_4\}$$

$$t_3 = \{v_2, v_3, v_5\}$$

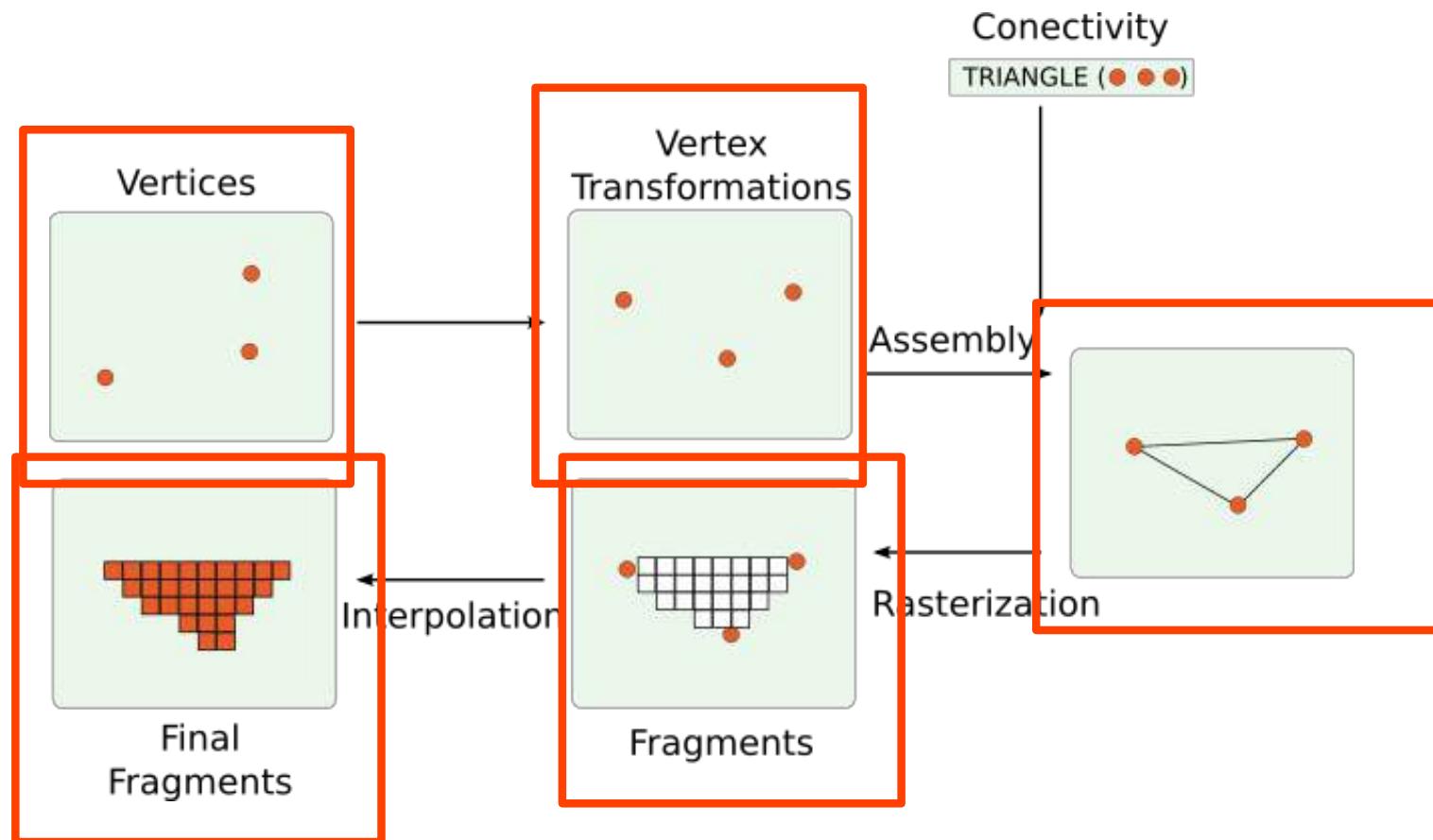
$\vdots$

# OBJ file

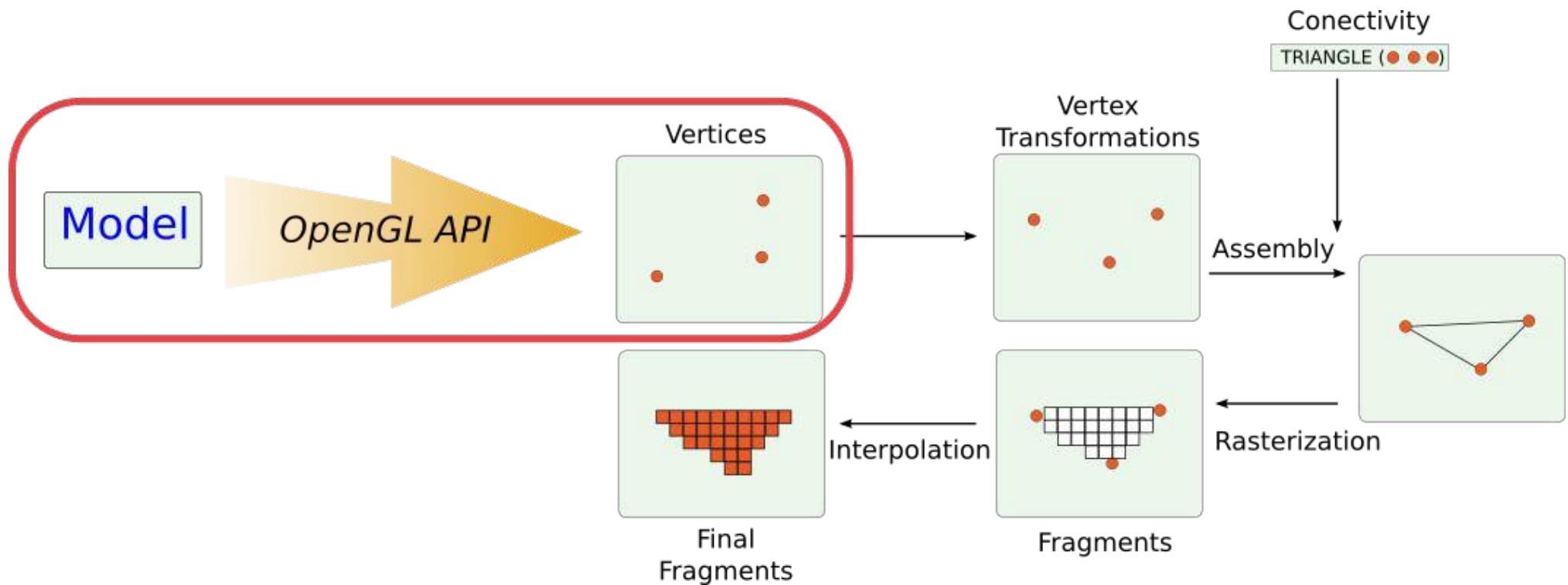
```
1 # triangle.obj
2 #
3
4 v 0.0 0.0 0.0
5 v 0.0 0.0 1.0
6 v 0.0 1.0 0.0
7
8 f 1 2 3
```

```
1 # Blender v2.78 (sub 0) OBJ File: ''
2 # www.blender.org
3 o root
4 v 0.207195 -0.263764 0.214438
5 v 0.206990 -0.264675 0.211045
6 v 0.207187 -0.260728 0.213590
7 v 0.206078 -0.264410 0.217782
8 v 0.207410 -0.260020 0.216926
9 v 0.206488 -0.259670 0.220876
10 v 0.206412 -0.267725 0.208776
11 v 0.205247 -0.269258 0.207019
12 v 0.205058 -0.267527 0.205096
13 v 0.204569 -0.270174 0.210299
14 v 0.205506 -0.267026 0.214885
15 v 0.206529 -0.265218 0.208719
16 v 0.205088 -0.265985 0.205913
17 v 0.203961 -0.261230 0.207378
18 v 0.203722 -0.262400 0.222970
19 v 0.203432 -0.259140 0.226576
20 v 0.204938 -0.257517 0.211903
21 v 0.205295 -0.255813 0.226379
22 v 0.205062 -0.254126 0.216151
23 v 0.206667 -0.256820 0.218732
24 v 0.206375 -0.256206 0.222311
25 v 0.205382 -0.251990 0.221828
```

# graphics pipeline (simplified)

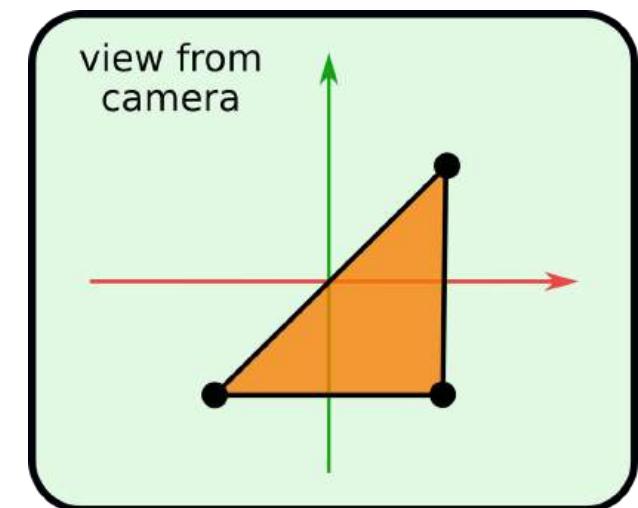
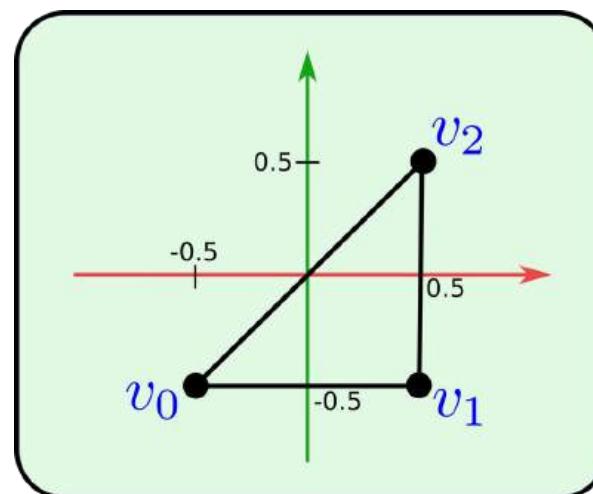


# graphics pipeline (simplified)



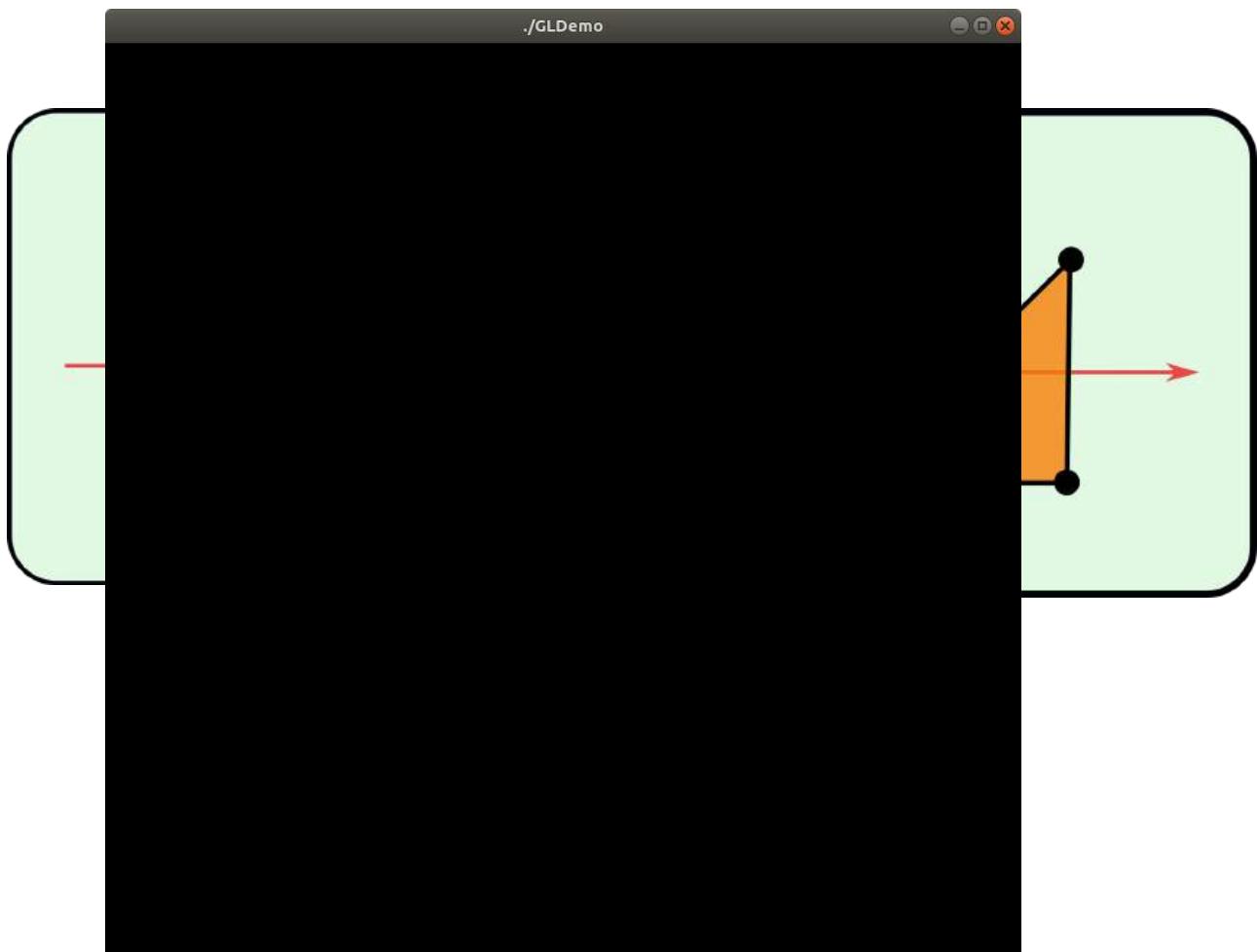
# simple triangle

```
1 void drawTriangle()
2 {
3     glBegin(GL_TRIANGLES);
4     glColor3f(1.0, 0.5, 0.0);
5     glVertex3f (-0.5, -0.5, 0.0);
6     glVertex3f ( 0.5, -0.5, 0.0);
7     glVertex3f ( 0.5,  0.5, 0.0);
8
9     glEnd();
10 }
```



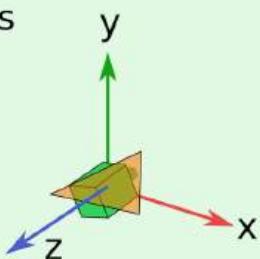
# simple triangle

```
1
2 void drawTriangle()
3 {
4     glBegin(GL_TRIANGLES);
5     glColor3f(1.0, 0.5, 0.0);
6     glVertex3f (-0.5, -0.5, 0.0);
7     glVertex3f ( 0.5, -0.5, 0.0);
8     glVertex3f ( 0.5,  0.5, 0.0);
9     glEnd();
10 }
```

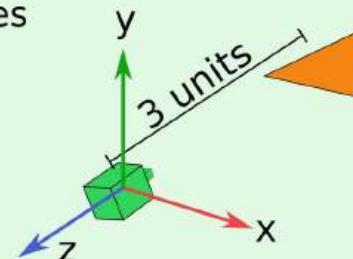


# simple triangle

world  
coordinates

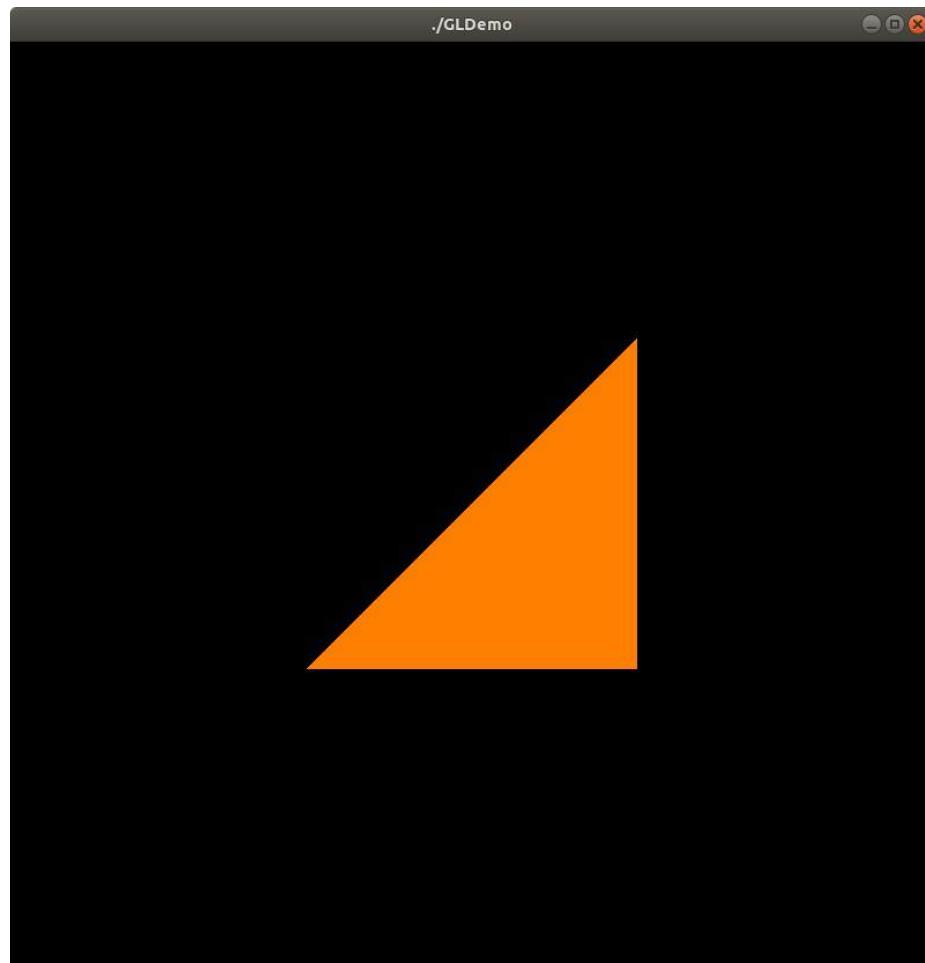
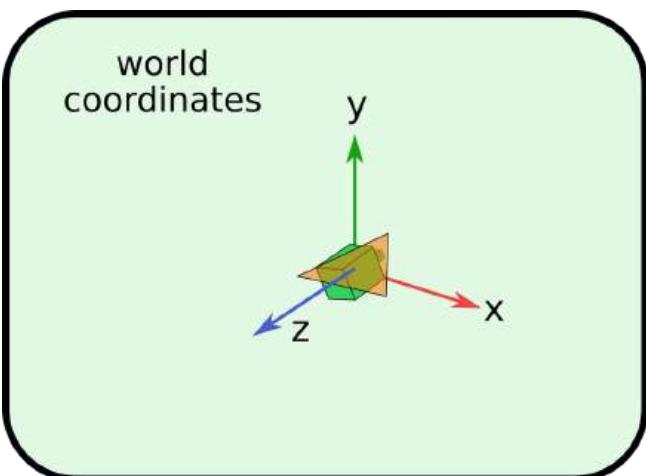


world  
coordinates



```
1 void drawTriangle()
2 {
3     glBegin(GL_TRIANGLES);
4     glColor3f(1.0, 0.5, 0.0);
5     glVertex3f (-0.5, -0.5, -3.0);
6     glVertex3f ( 0.5, -0.5, -3.0);
7     glVertex3f ( 0.5,  0.5, -3.0);
8
9 }
10 }
```

# simple triangle



# color per vertex

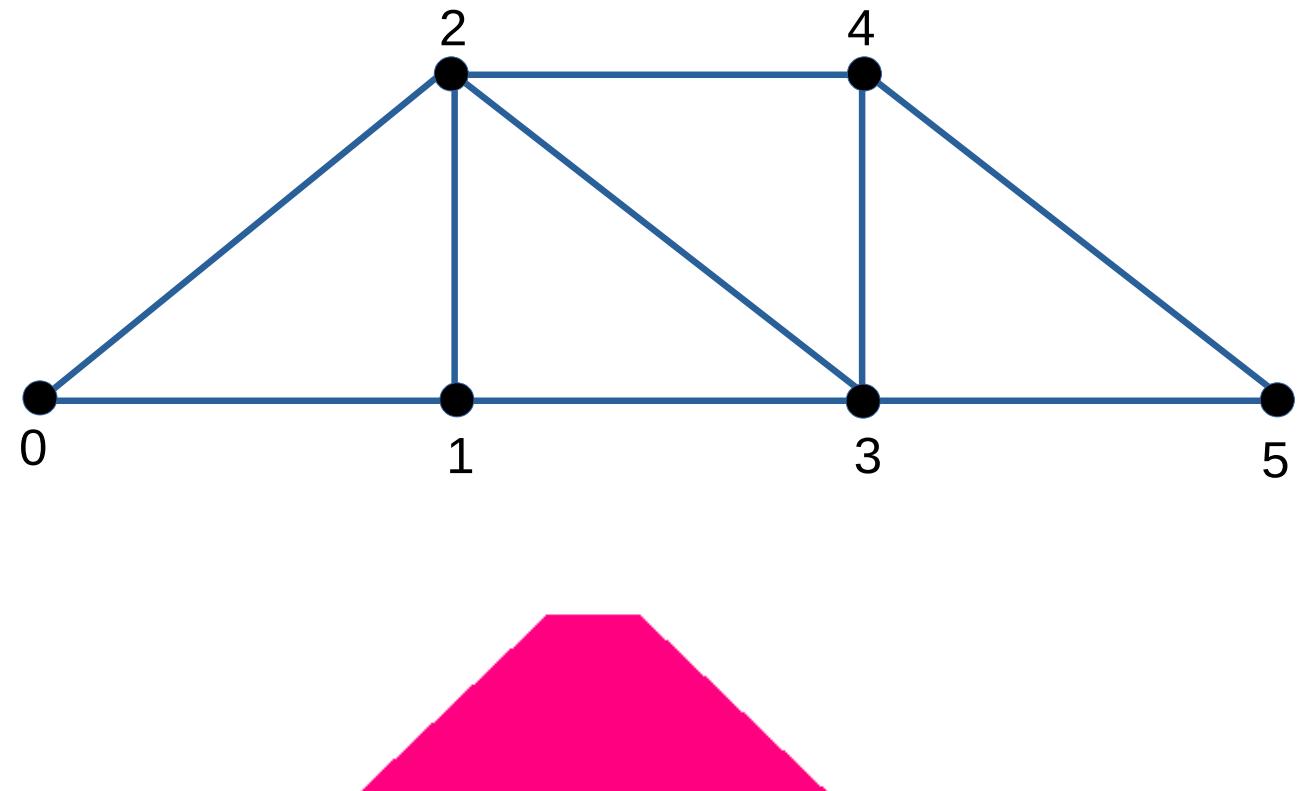


```
1
2 void drawTriangle()
3 {
4     glBegin(GL_TRIANGLES);
5     glColor3f(1.0, 0.5, 0.0);
6     glVertex3f (-0.5, -0.5, -3.0);
7     glVertex3f ( 0.5, -0.5, -3.0);
8     glVertex3f ( 0.5,  0.5, -3.0);
9     glEnd();
10 }
```

```
1 void drawTriangle()
2 {
3     glBegin(GL_TRIANGLES);
4
5     glColor3f(1.0f, 0.5f, 0.0f);
6     glVertex3f (-0.5f, -0.5f, -3.0f);
7
8     glColor3f(1.0f, 0.0f, 1.0f);
9     glVertex3f ( 0.5f, -0.5f, -3.0f);
10
11    glColor3f(0.0f, 1.0f, 0.5f);
12    glVertex3f ( 0.5f,  0.5f, -3.0f);
13
14    glEnd();
15 }
```



# triangle strips



```
glBegin(GL_TRIANGLE_STRIP);
glColor3f(1.0f, 0.0f, 0.5f);
glVertex3f (-0.5f, -0.5f, 0.0f);
glVertex3f ( 0.5f, -0.5f, 0.0f);
glVertex3f ( 0.5f,  0.5f, 0.0f);
glVertex3f ( 1.0f, -0.5f, 0.0f);
glVertex3f ( 1.0f,  0.5f, 0.0f);
glVertex3f ( 2.0f, -0.5f, 0.0f);

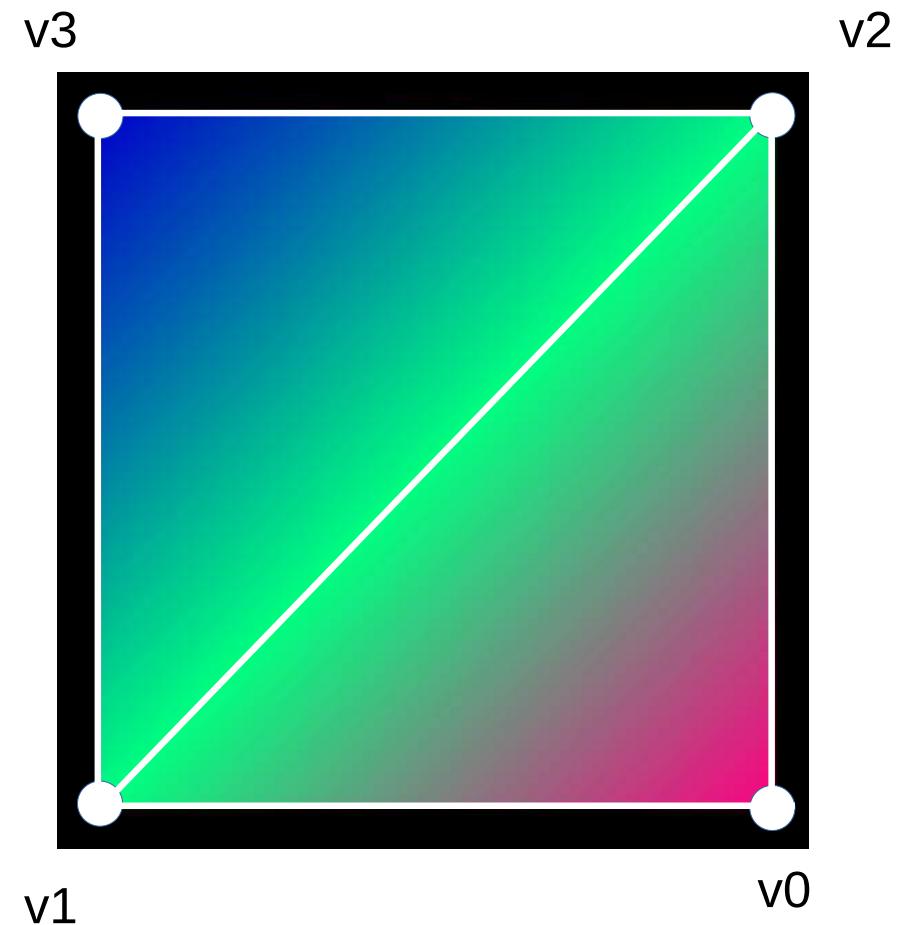
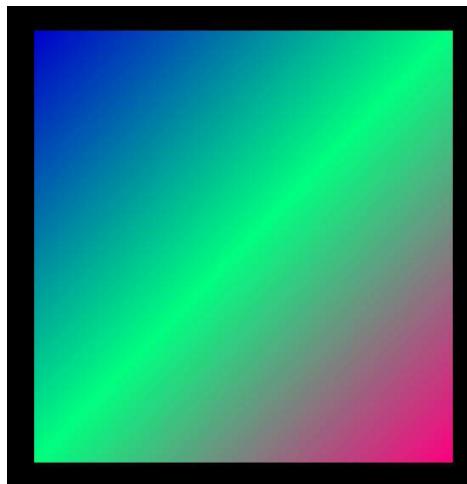
glEnd();
```

# triangle strip

```
glBegin(GL_TRIANGLE_STRIP);
glColor3f(1.0f, 0.0f, 0.5f);
glVertex3f ( 0.5f, -0.5f, 0.0f);

glColor3f(0.0f, 1.0f, 0.5f);
glVertex3f (-0.5f, -0.5f, 0.0f);
glVertex3f ( 0.5f, 0.5f, 0.0f);

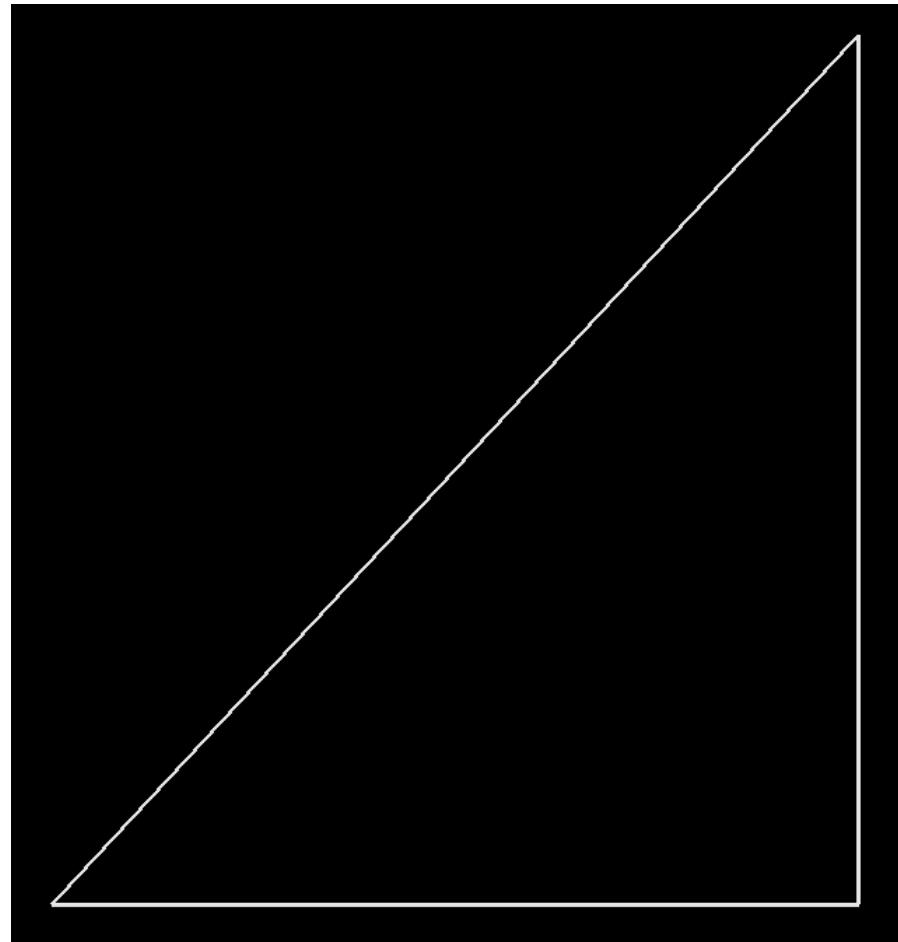
glColor3f(0.0f, 0.0f, 0.8f);
glVertex3f (-0.5f, 0.5f, 0.0f);
glEnd();
```



# lines

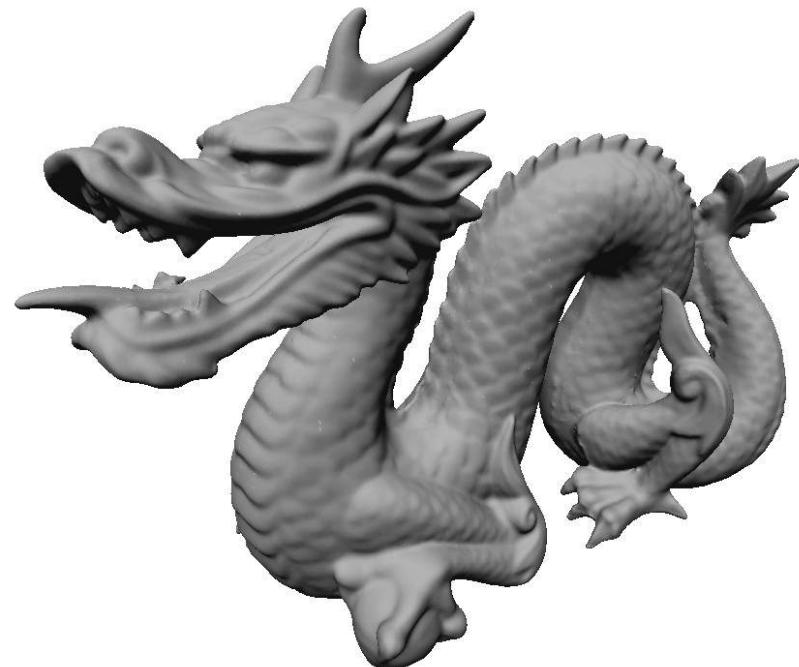
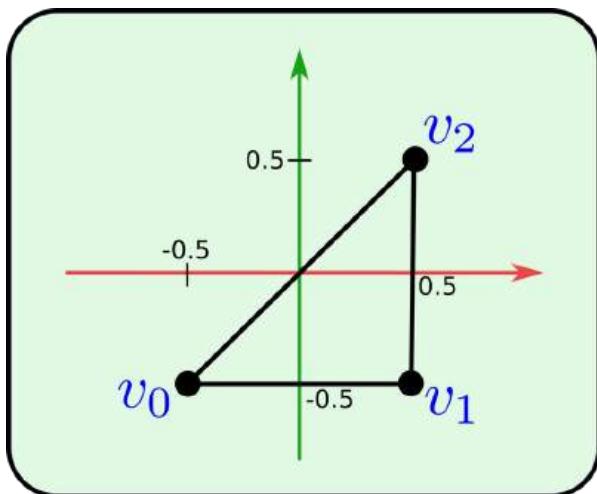
```
void drawTriangleWireframe()
{
    glLineWidth(3.0f);
    glBegin(GL_LINES);
    glColor3f(0.9f, 0.9f, 0.9f);
    glVertex3f (-0.5f, -0.5f, -3.0f);
    glVertex3f ( 0.5f, -0.5f, -3.0f);
    glVertex3f ( 0.5f, -0.5f, -3.0f);
    glVertex3f ( 0.5f,  0.5f, -3.0f);
    glVertex3f ( 0.5f,  0.5f, -3.0f);
    glVertex3f (-0.5f, -0.5f, -3.0f);
    glEnd();
}
```

```
void drawTriangleWireframe()
{
    glLineWidth(3.0f);
    glBegin(GL_LINE_STRIP);
    glColor3f(0.9f, 0.9f, 0.9f);
    glVertex3f (-0.5f, -0.5f, -3.0f);
    glVertex3f ( 0.5f, -0.5f, -3.0f);
    glVertex3f ( 0.5f,  0.5f, -3.0f);
    glVertex3f (-0.5f, -0.5f, -3.0f);
    glEnd();
}
```



# transformations

```
1 void drawTriangle()
2 {
3     glBegin(GL_TRIANGLES);
4     glColor3f(1.0, 0.5, 0.0);
5     glVertex3f (-0.5, -0.5, -3.0);
6     glVertex3f ( 0.5, -0.5, -3.0);
7     glVertex3f ( 0.5, 0.5, -3.0);
8     glEnd();
9 }
```



changing the values of  
thousands of vertices is costly!!

# transformations



# transformations

remember the camera model?

$$\underbrace{\begin{bmatrix} k_x & 0 & 0 & x_0 \\ 0 & k_y & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{image} \underbrace{\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{projection} \underbrace{\begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{modelview}$$

OpenGL uses the same model!

# ModelView

$$\begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

rotation, scale, shear

translation

eye coordinates

object coordinates

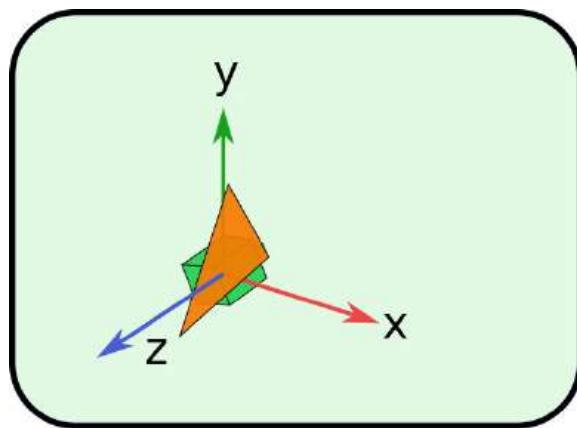
# ModelView

translate vertex -3 units  
along the z axis?

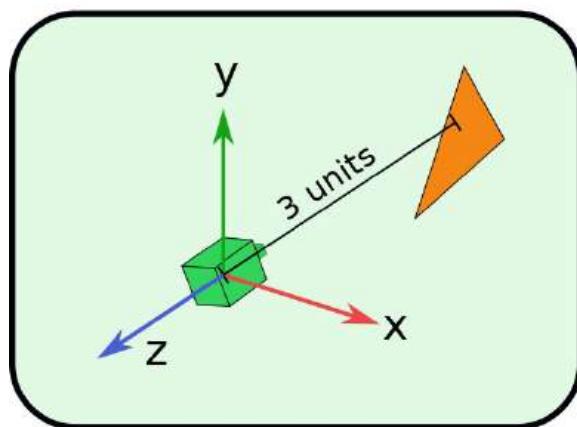
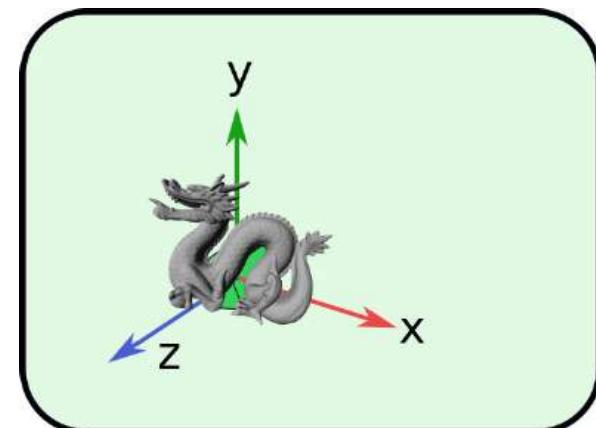
$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

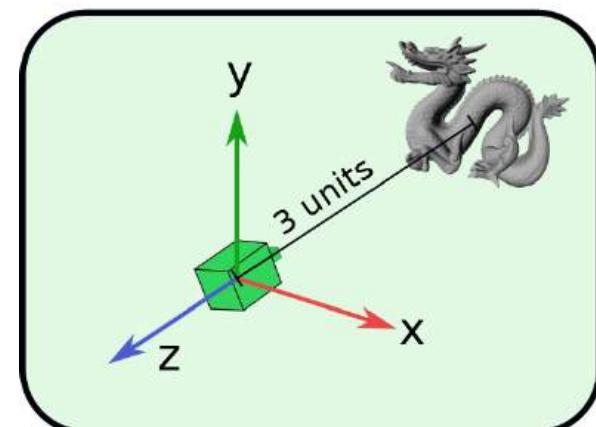
# ModelView



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# ModelViewMatrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
1 void drawTriangle()
2 {
3     glBegin(GL_TRIANGLES);
4     glColor3f(1.0, 0.5, 0.0);
5     glVertex3f (-0.5, -0.5, -3.0);
6     glVertex3f ( 0.5, -0.5, -3.0);
7     glVertex3f ( 0.5,  0.5, -3.0);
8     glEnd();
9 }
```

```
1 void drawTriangle()
2 {
3     glMatrixMode(GL_MODELVIEW);
4     float mm [16] = {1, 0, 0, 0,
5                       0, 1, 0, 0,
6                       0, 0, 1, 0,
7                       0, 0, -3, 1};
8     glLoadMatrixf(mm);
9
10    glBegin(GL_TRIANGLES);
11    glColor3f(1.0, 0.5, 0.0);
12    glVertex3f (-0.5, -0.5, 0.0);
13    glVertex3f ( 0.5, -0.5, 0.0);
14    glVertex3f ( 0.5,  0.5, 0.0);
15    glEnd();
16 }
```

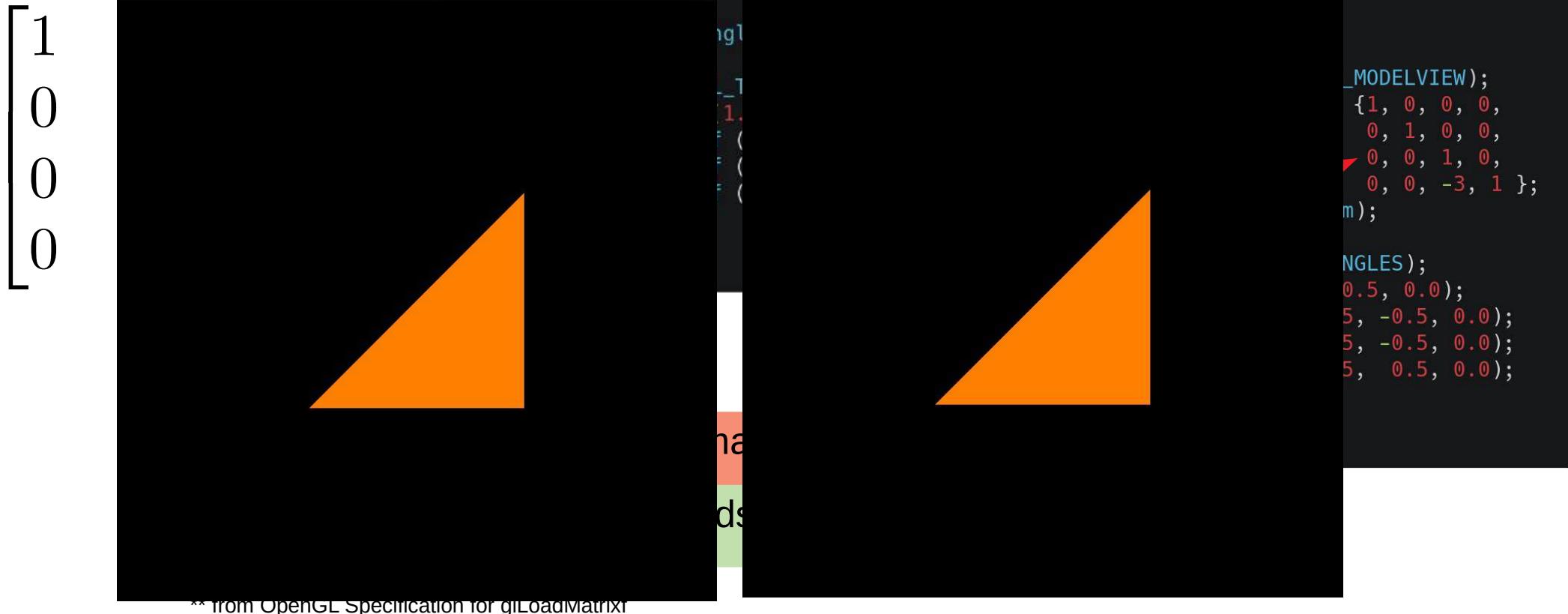
why is the matrix transposed?

OpenGL reads in column order!

\*\* from OpenGL Specification for `glLoadMatrixf`

Specifies a pointer to 16 consecutive values, which are used as the elements of a  $4 \times 4$  column-major matrix.

# ModelViewMatrix



Specifies a pointer to 16 consecutive values, which are used as the elements of a  $4 \times 4$  column-major matrix.

# OpenGL API

**glMatrixMode (mode)**  
mode = GL\_MODELVIEW, GL\_PROJECTION ...

**State Machine**  
any operation after  
glMatrixMode affects only the  
current *mode*

**matrix stack (for each mode)**  
save and restore matrix

**glPushMatrix()**

**glPopMatrix()**

**glMultMatrix (\*m)**

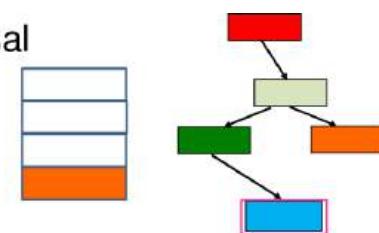
**glLoadMatrix (\*m)**

**glLoadIdentity ()**

**you can also write your own stack!**

Depth-first tree traversal  
using a stack:

```
stack.push(root)
while (!stack.empty())
{
```



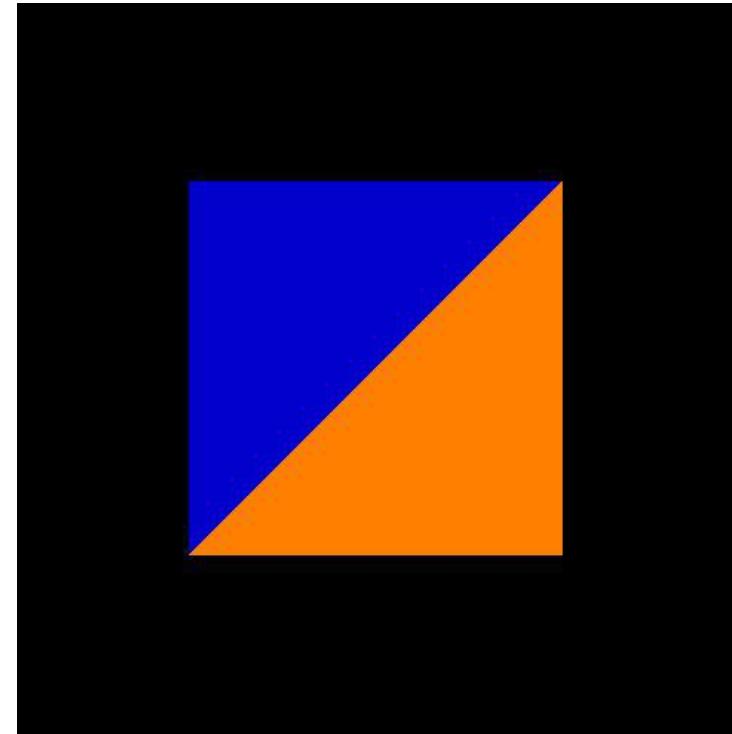
# ModelView

```
1 void drawTriangle()
2 {
3     glMatrixMode(GL_MODELVIEW);
4     float mm [16] = {1, 0, 0, 0,
5                         0, 1, 0, 0,
6                         0, 0, 1, 0,
7                         0, 0, -3, 1 };
8     glLoadMatrixf(mm);
9
10    glBegin(GL_TRIANGLES);
11    glColor3f(1.0, 0.5, 0.0);
12    glVertex3f (-0.5, -0.5, 0.0);
13    glVertex3f ( 0.5, -0.5, 0.0);
14    glVertex3f ( 0.5,  0.5, 0.0);
15    glEnd();
16 }
```

```
1 void drawTriangle()
2 {
3     glMatrixMode(GL_MODELVIEW);
4     glm::mat4 mm (1, 0, 0, 0,
5                   0, 1, 0, 0,
6                   0, 0, 1, 0,
7                   0, 0, -3, 1 );
8     glLoadMatrixf(glm::value_ptr(mm));
9
10    glBegin(GL_TRIANGLES);
11    glColor3f(1.0, 0.5, 0.0);
12    glVertex3f (-0.5, -0.5, 0.0);
13    glVertex3f ( 0.5, -0.5, 0.0);
14    glVertex3f ( 0.5,  0.5, 0.0);
15    glEnd();
16 }
```

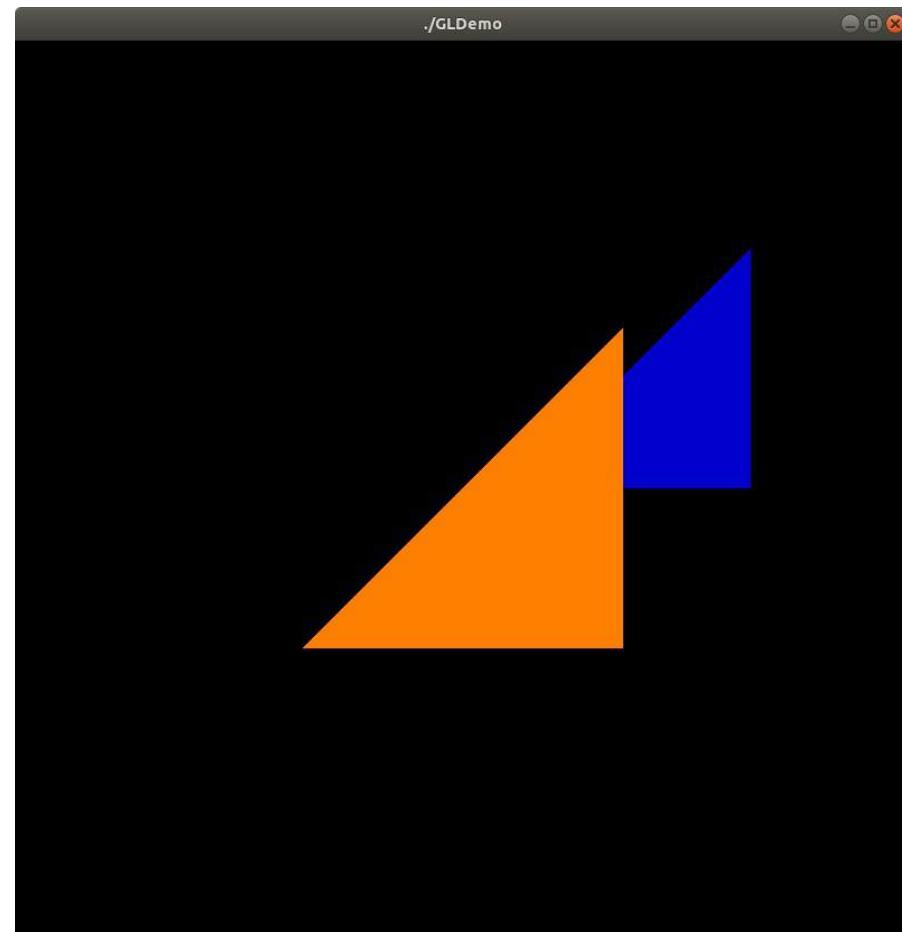
using glm

```
1 void drawTriangle( )
2 {
3     glMatrixMode(GL_MODELVIEW);
4     glm::mat4 mm (1, 0, 0, 0,
5                   0, 1, 0, 0,
6                   0, 0, 1, 0,
7                   0, 0, -3, 1 );
8     glMultMatrixf(glm::value_ptr(mm));
9
10    glBegin(GL_TRIANGLES);
11    glColor3f(1.0, 0.5, 0.0);
12    glVertex3f (-0.5, -0.5, 0.0);
13    glVertex3f ( 0.5, -0.5, 0.0);
14    glVertex3f ( 0.5,  0.5, 0.0);
15    glColor3f(0.0, 0.0, 0.8);
16    glVertex3f (-0.5, -0.5, 0.0);
17    glVertex3f ( 0.5,  0.5, 0.0);
18    glVertex3f (-0.5,  0.5, 0.0);
19    glEnd();
20 }
```



# ModelView

```
1 void drawTriangle()
2 {
3     glMatrixMode(GL_MODELVIEW);
4     glm::mat4 mm ( 1, 0, 0, 0,
5                     0, 1, 0, 0,
6                     0, 0, 1, 0,
7                     0, 0, -3, 1 );
8     glLoadMatrixf(glm::value_ptr(mm));
9
10    glBegin(GL_TRIANGLES);
11    glColor3f(1.0, 0.5, 0.0);
12    glVertex3f (-0.5, -0.5, 0.0);
13    glVertex3f ( 0.5, -0.5, 0.0);
14    glVertex3f ( 0.5, 0.5, 0.0);
15    glEnd();
16
17
18    mm = glm::mat4(1, 0, 0, 0,
19                    0, 1, 0, 0,
20                    0, 0, 1, 0,
21                    0.7, 0.5, -4, 1 );
22    glLoadMatrixf(glm::value_ptr(mm));
23
24    glBegin(GL_TRIANGLES);
25    glColor3f(0.0, 0.0, 0.8);
26    glVertex3f (-0.5, -0.5, 0.0);
27    glVertex3f ( 0.5, -0.5, 0.0);
28    glVertex3f ( 0.5, 0.5, 0.0);
29    glEnd();
30 }
```



# OpenGL API

you can load your  
matrices directly

`glLoadMatrix (*m)`

`glLoadIdentity ()`

`glMultMatrix (*m)`

or use helper functions that  
act on top of stack

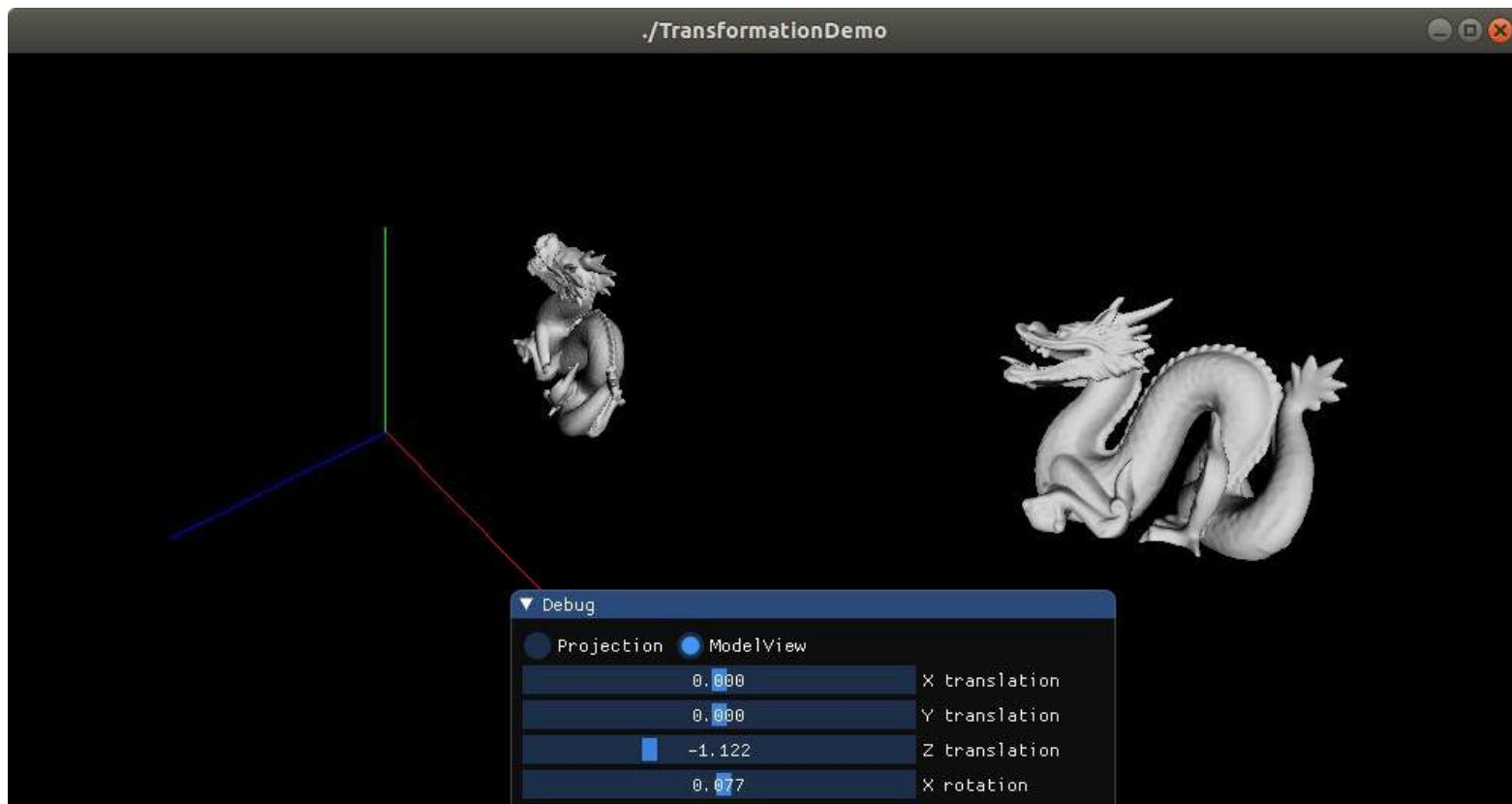
~~`glTranslate (x, y, z)`~~

~~`glRotate (angle, x, y, z)`~~

~~`glScale (x, y, z)`~~

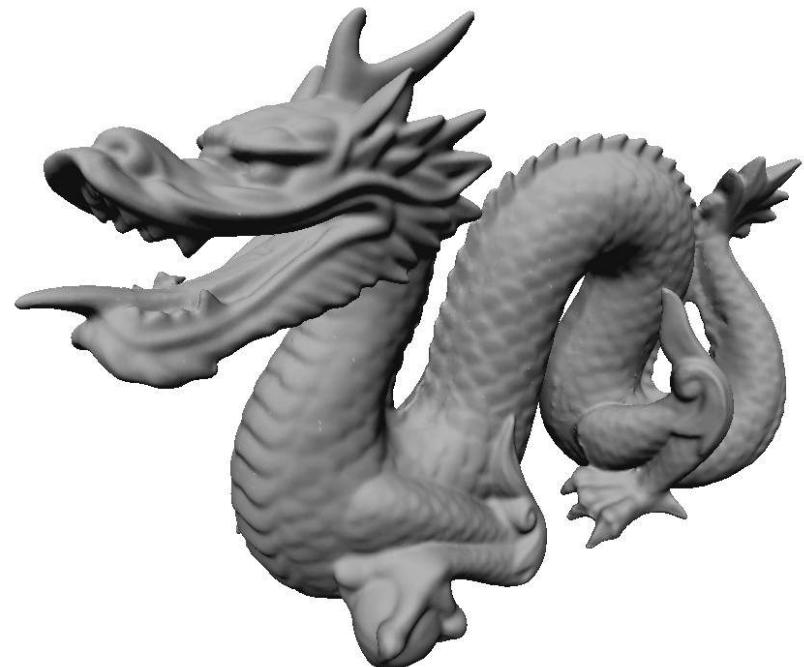
you don't need this, you KNOW  
how to create the transformation  
you want!

# demo



# note: normals

```
1 void drawTriangle()
2 {
3     glBegin(GL_TRIANGLES);
4     glColor3f (1.0f, 0.5f, 0.0f);
5
6     glNormal3f ( 0.0f, 0.0, 1.0);
7     glVertex3f (-0.5f, -0.5f, -3.0f);
8
9     glNormal3f ( 1.0f, 0.0, 0.0);
10    glVertex3f ( 0.5f, -0.5f, -3.0f);
11
12    glNormal3f ( 0.0f, 1.0, 0.0);
13    glVertex3f ( 0.5f, 0.5f, -3.0f);
14
15 }
```



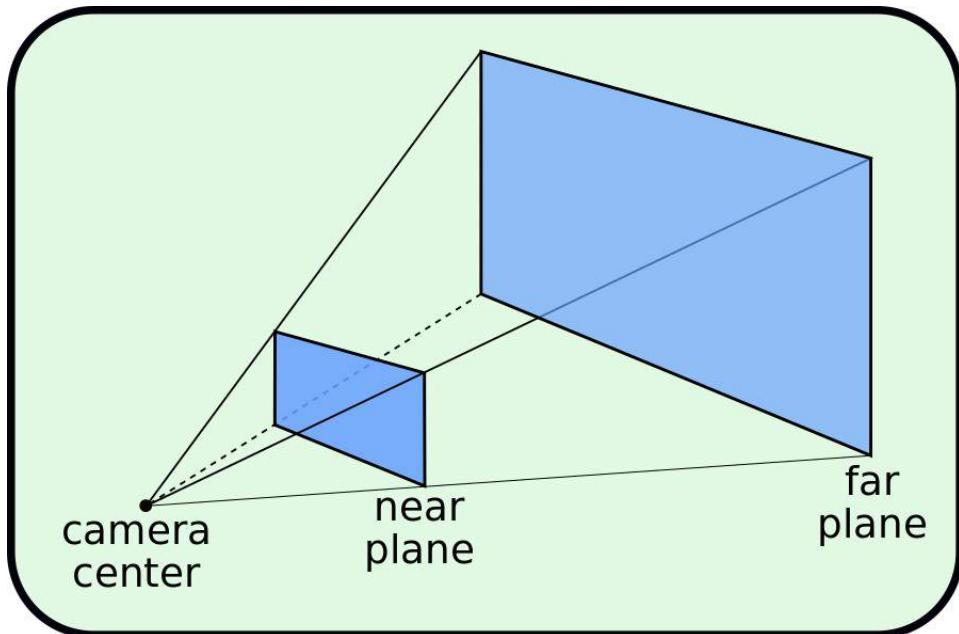
# OpenGL transformations

$$\underbrace{\begin{bmatrix} k_x & 0 & 0 & x_0 \\ 0 & k_y & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{image} \underbrace{\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{projection} \underbrace{\begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{modelview}$$

$$\begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

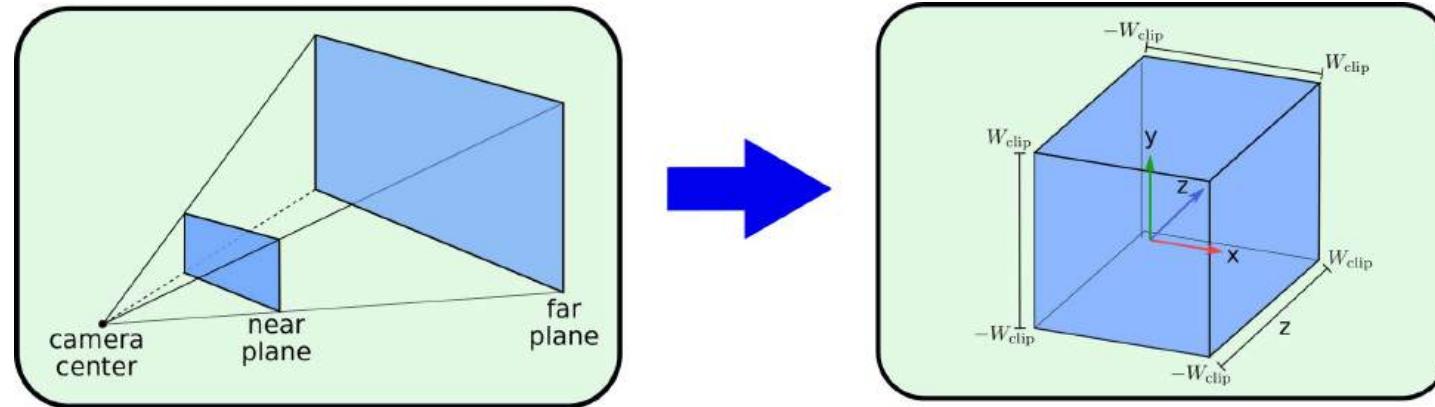


# Projection



$$\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

# Projection



we do not actually project to a plane in order to keep the z value

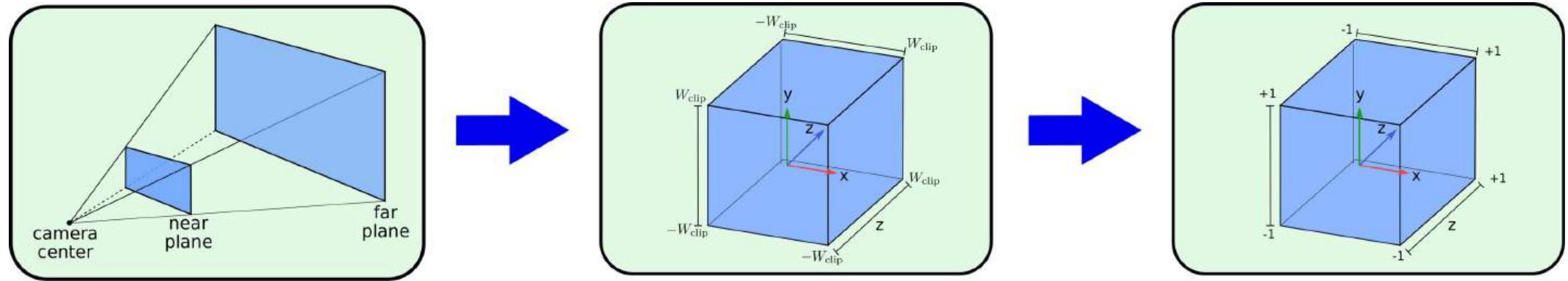
$$\begin{bmatrix} X_{clip} \\ Y_{clip} \\ Z_{clip} \\ W_{clip} \end{bmatrix} = \begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix}$$

clip coordinates

OpenGL clip space is in range  
 $[-W_{clip}, +W_{clip}]$

eye coordinates

# Perspective division



$$\begin{bmatrix} X_{clip} \\ Y_{clip} \\ Z_{clip} \\ W_{clip} \end{bmatrix} = \begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix}$$

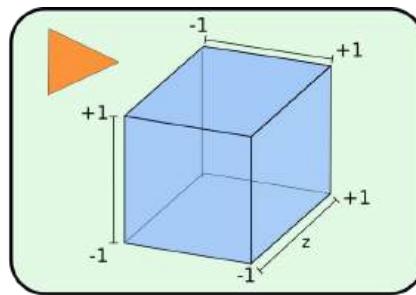
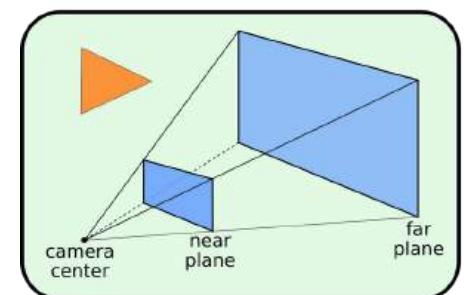
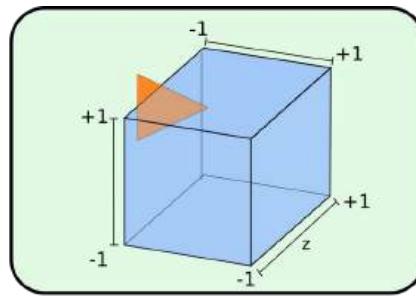
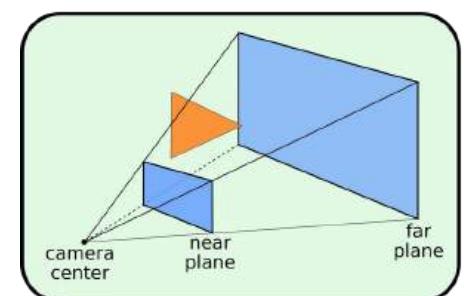
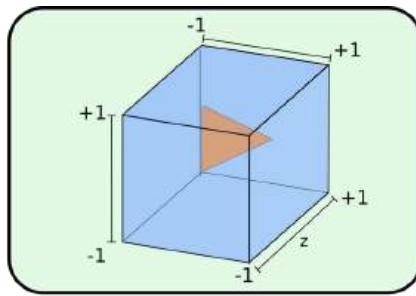
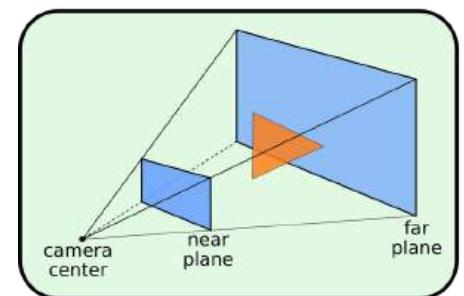
clip coordinates

$$\begin{bmatrix} X_{ndc} \\ Y_{ndc} \\ Z_{ndc} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{X_{clip}}{W_{clip}} \\ \frac{Y_{clip}}{W_{clip}} \\ \frac{Z_{clip}}{W_{clip}} \\ \frac{W_{clip}}{W_{clip}} \end{bmatrix}$$

eye coordinates

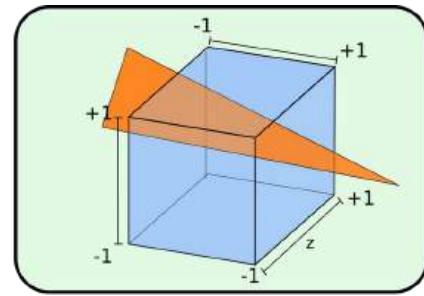
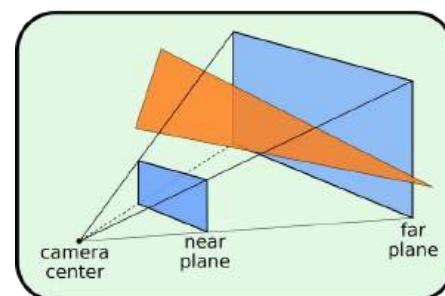
normalized device  
coordinates

# Clipping

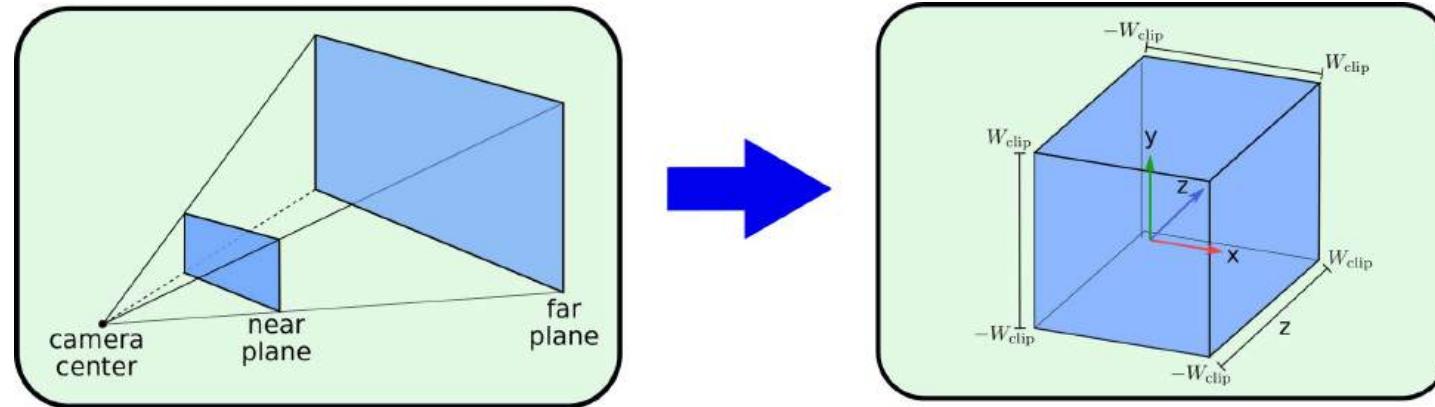


can we discard triangles with all vertices outside the cube?

no!



# Projection



**Question:**  
OpenGL performs  
clipping in Clip  
Space, not DNC.  
**Why??**

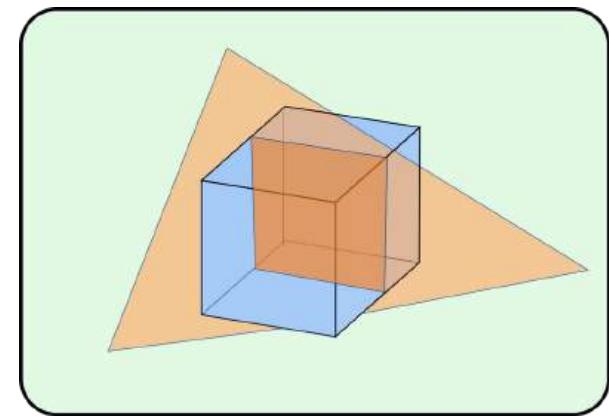
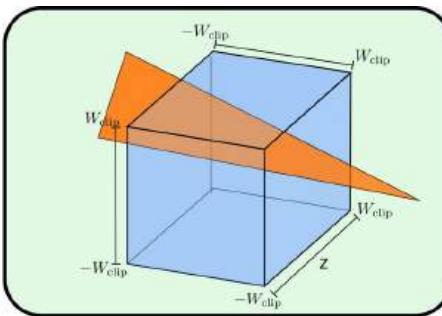
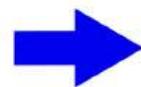
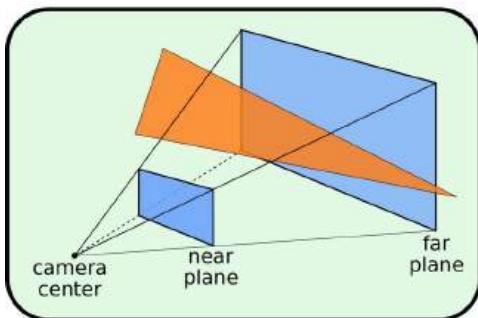
$$\begin{bmatrix} X_{clip} \\ Y_{clip} \\ Z_{clip} \\ W_{clip} \end{bmatrix} = \begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix}$$

clip coordinates

OpenGL clips primitives  
using the range  
 $[-W_{clip}, +W_{clip}]$

eye coordinates

# question



- 1) with the knowledge so far, how would you implement a function to test if a triangle needs to be clipped?
- 2) where would you clip the triangle?

obs1: work in normalized device coordinates.

obs2: It does not have to be efficient

obs3: think about the first lecture

bonus case: can your method detect a triangle that intersects the cube, but no triangle edge intersects the cube?  
Would you need another test for this case?

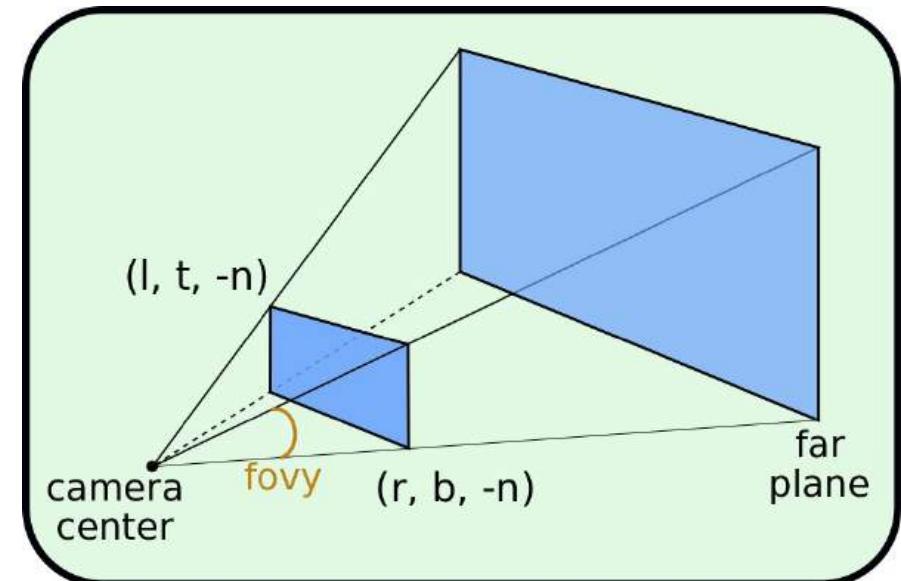
# OpenGL API

**glFrustum (left, right, bottom, top, near , far)**

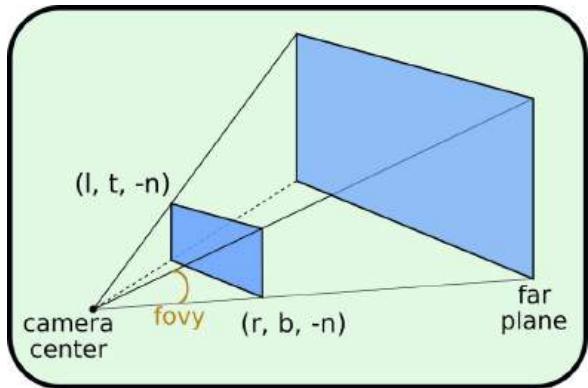
**gluPerspective (fovy, aspect, near , far)**

**glm::perspective (fovy, aspect, near , far)**

$$\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

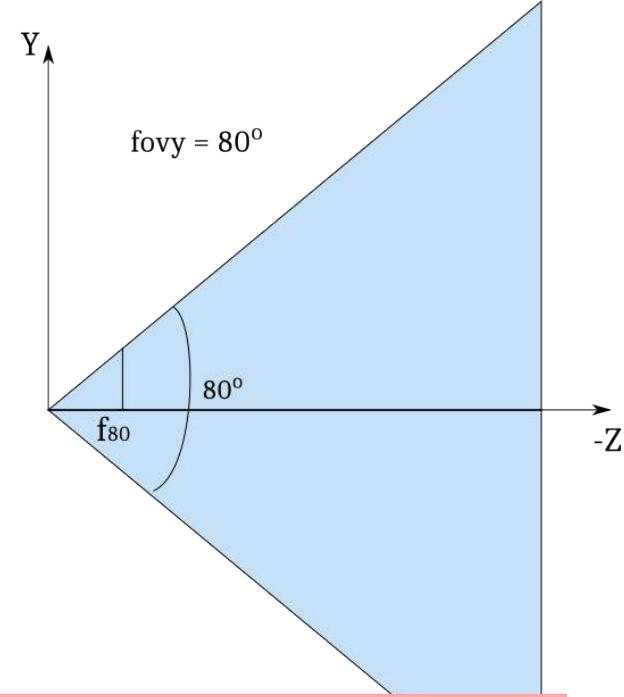
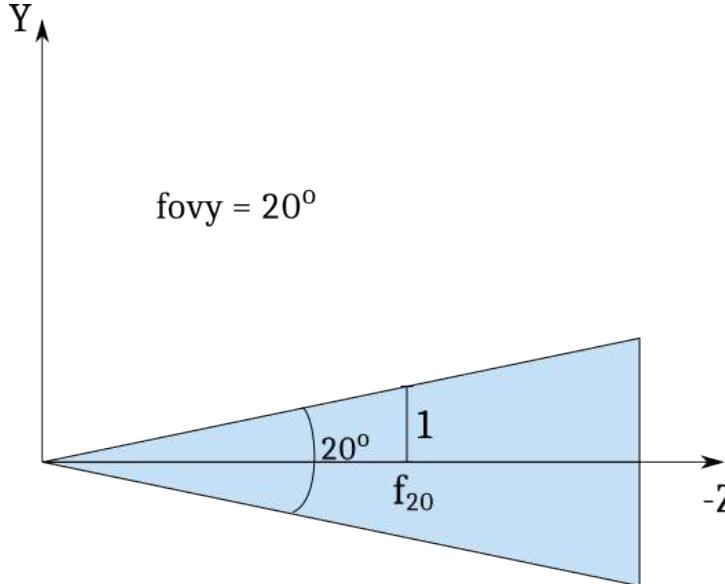
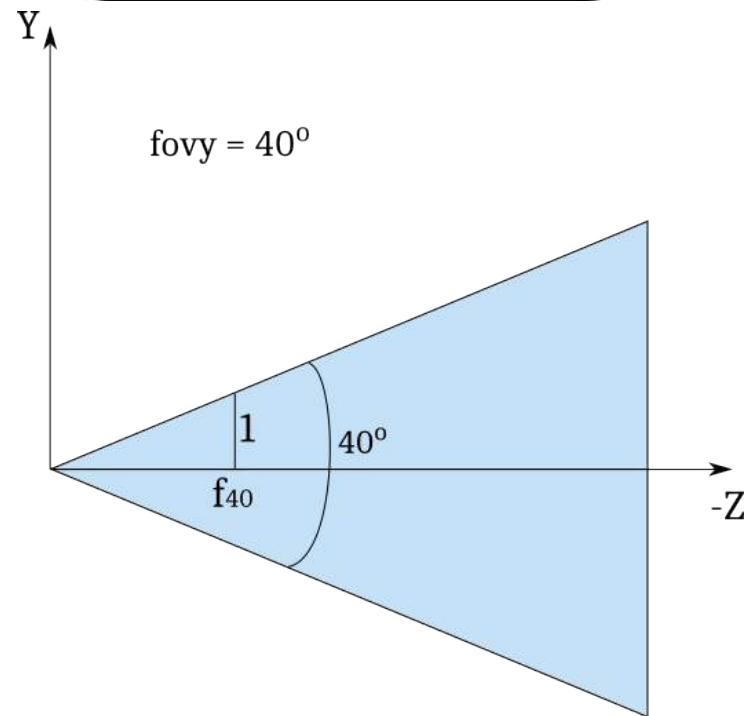


**attention: specify ||near|| and ||far||**



# FOV

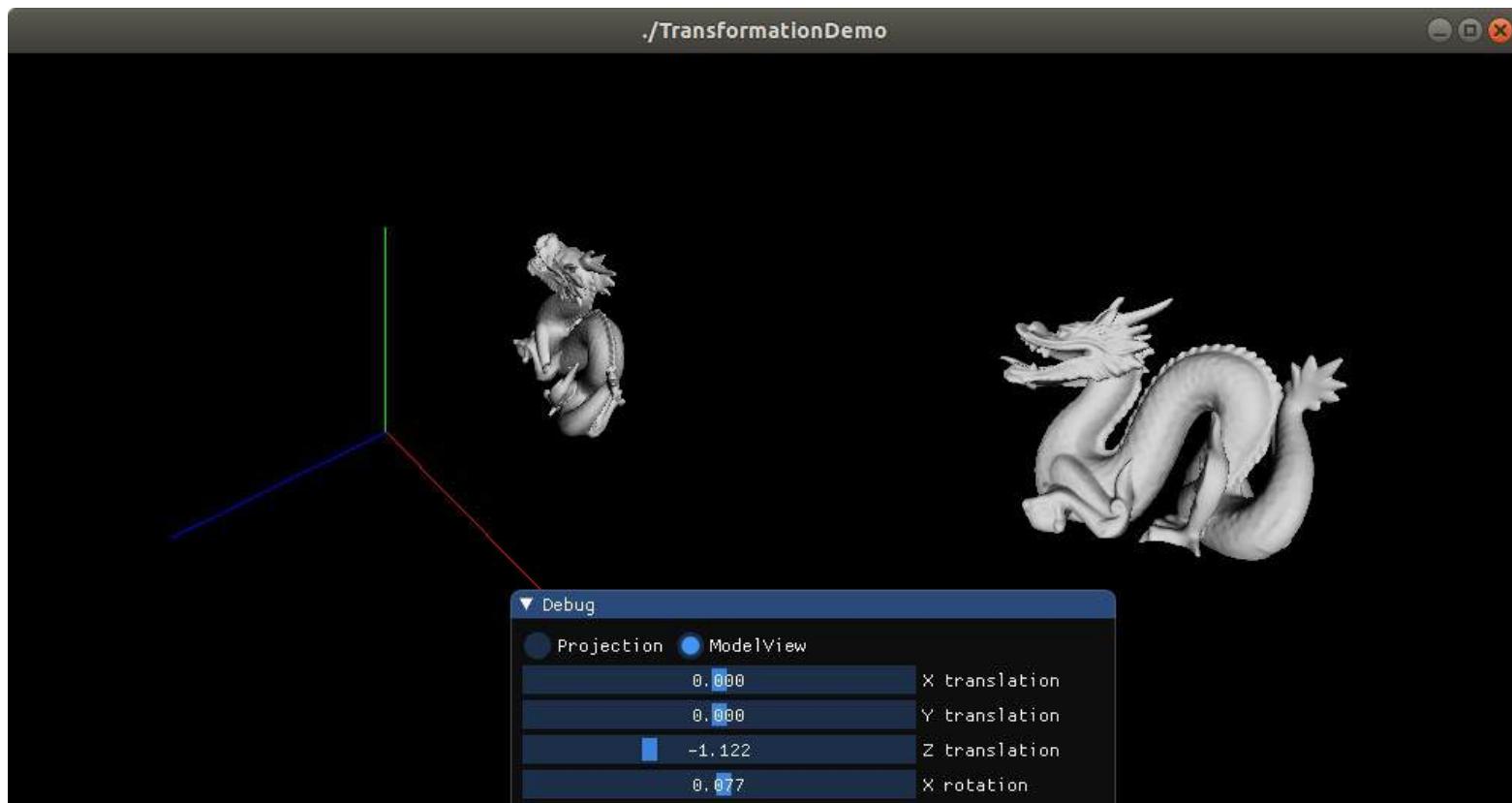
## Field of View (section 7.5)



$$f_{80} < f_{40} < f_{20}$$

**focal length (f) has inverse relation with field of view (fov)**

# demo



# question

$$\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



**can we simulate a fisheye lens with this matrix? Why or why not?**



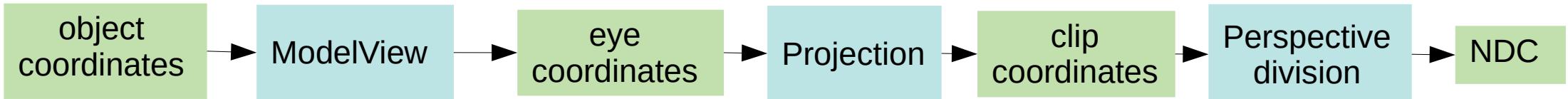
# OpenGL transformations

$$\underbrace{\begin{bmatrix} k_x & 0 & 0 & x_0 \\ 0 & k_y & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{image} \underbrace{\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{projection} \underbrace{\begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{modelview}$$

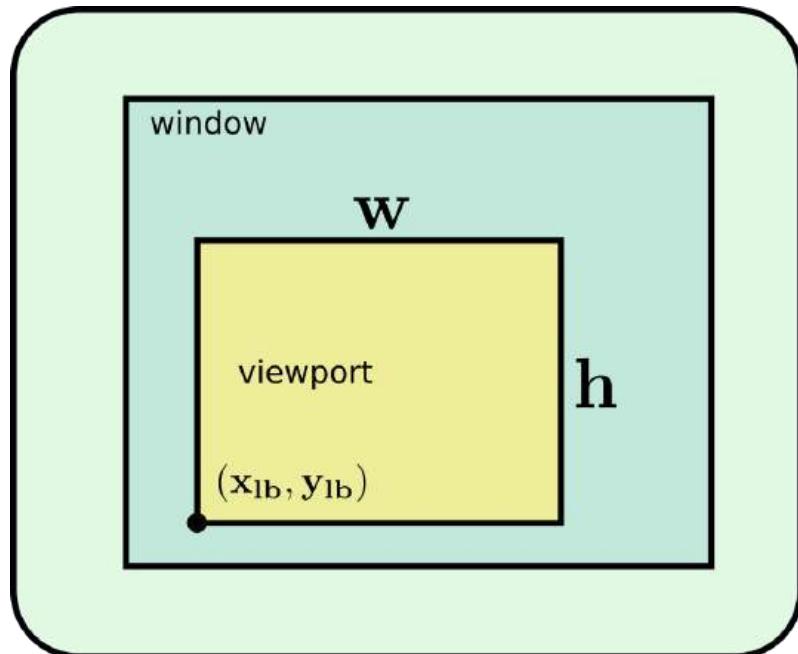
$$\begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{clip} \\ Y_{clip} \\ Z_{clip} \\ W_{clip} \end{bmatrix} = \begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{ndc} \\ Y_{ndc} \\ Z_{ndc} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{X_{clip}}{W_{clip}} \\ \frac{Y_{clip}}{W_{clip}} \\ \frac{Z_{clip}}{W_{clip}} \\ \frac{1}{W_{clip}} \end{bmatrix}$$



# Viewport



defines part of the window to use  
(for most applications, use entire window)

## windows position (pixels)

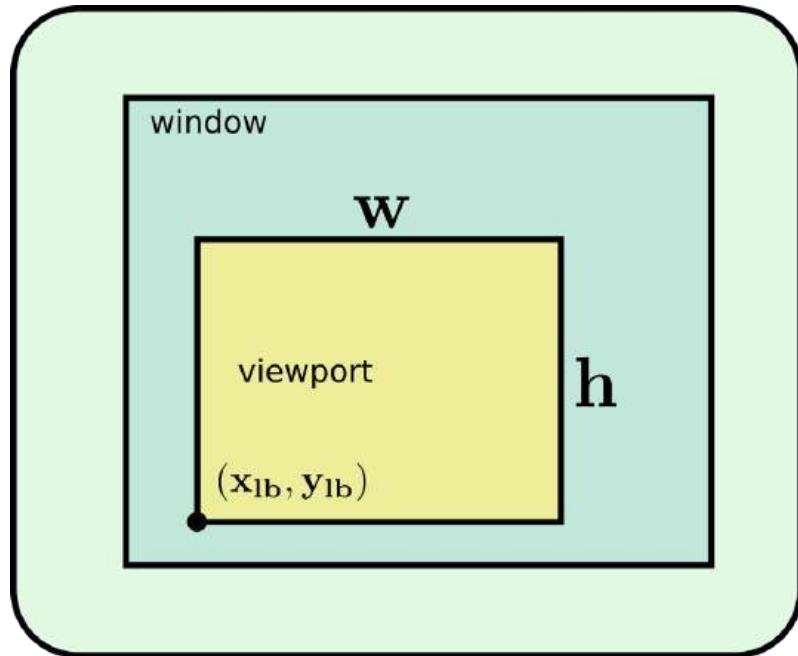
$$X_{\text{window}} = \left(\frac{X_{\text{ndc}}+1}{2}\right)w + x_{\text{lb}}$$

$$Y_{\text{window}} = \left(\frac{Y_{\text{ndc}}+1}{2}\right)h + y_{\text{lb}}$$

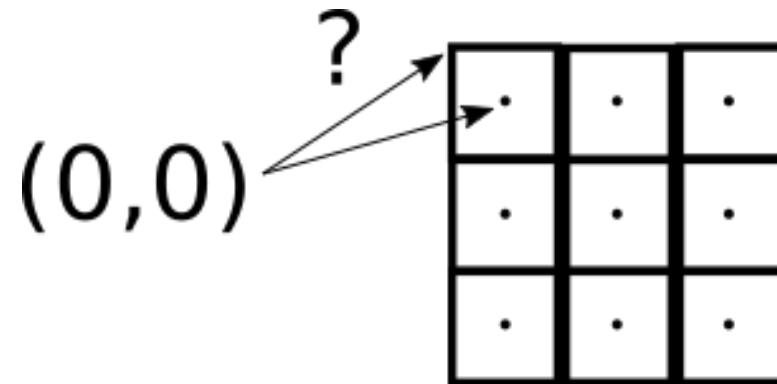
$$x_{\text{lb}} = 0, y_{\text{lb}} = 0$$

$$X_{\text{ndc}} = +1 \rightarrow X_{\text{window}} = ?$$

# Viewport



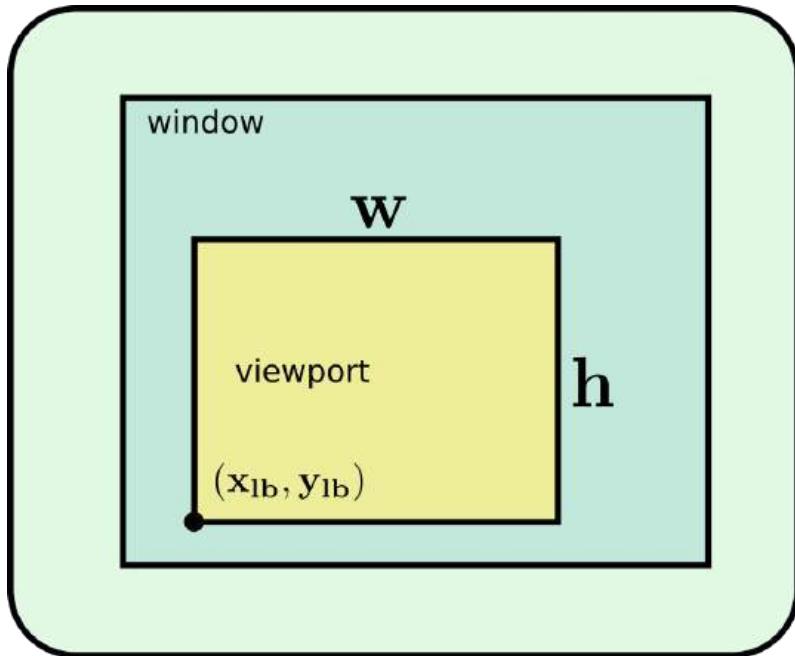
defines part of the window to use  
(for most applications, use entire window)



- if top-left corner is  $(0,0)$  the range is  $[0,w]$
- if pixel center is  $(0,0)$  the range is  $[-0.5,w-0.5]$

**note: different implementation choices  
(here vs book)**

# Viewport



defines part of the window to  
use  
(for most applications, use  
entire window)

**void glViewport(x, y, width, height);**

**Defines lower-left corner and size in pixels**

# OpenGL transformations

$$\begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} X_{clip} \\ Y_{clip} \\ Z_{clip} \\ W_{clip} \end{bmatrix} = \begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix}$$

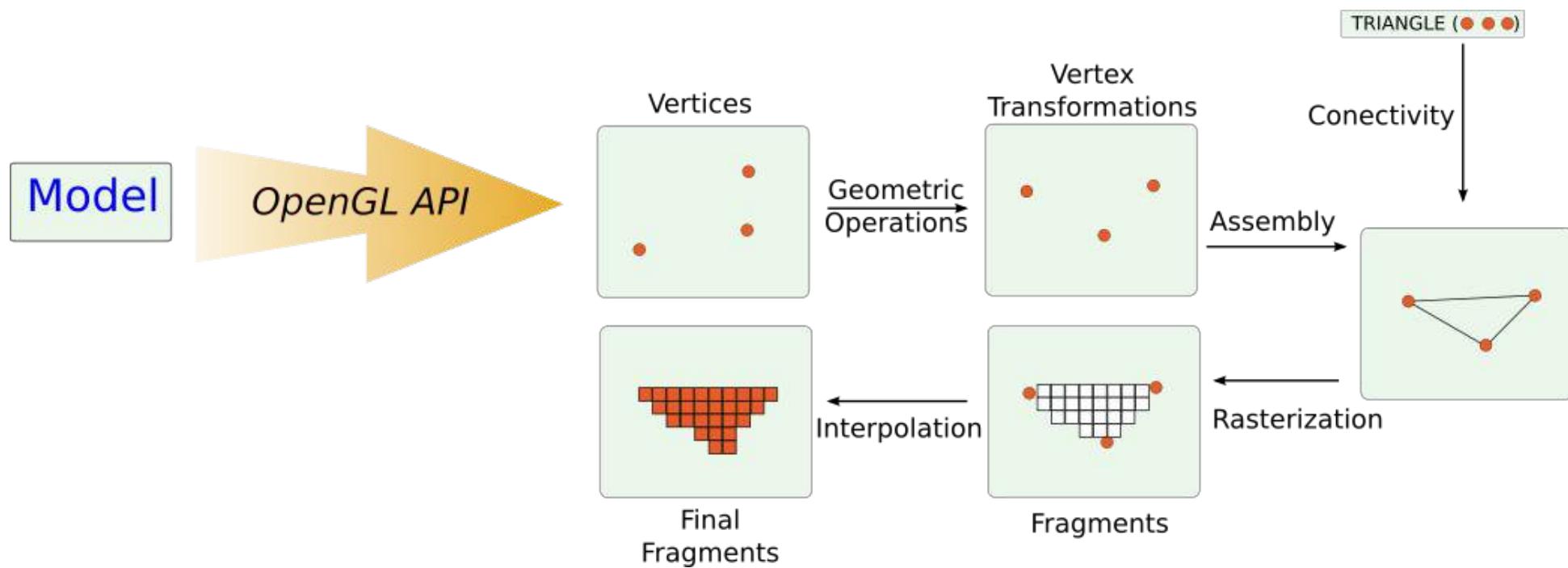


$$\begin{bmatrix} X_{ndc} \\ Y_{ndc} \\ Z_{ndc} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{X_{clip}}{W_{clip}} \\ \frac{Y_{clip}}{W_{clip}} \\ \frac{Z_{clip}}{W_{clip}} \\ \frac{W_{clip}}{W_{clip}} \end{bmatrix}$$

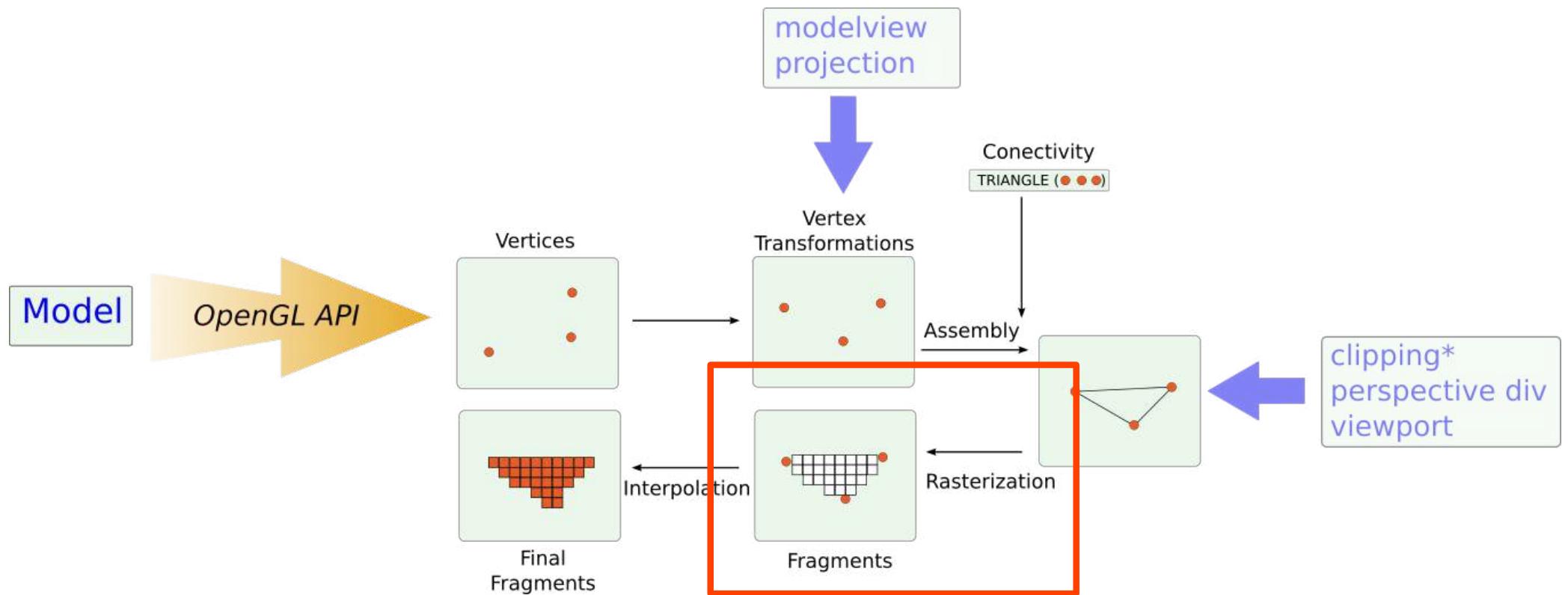
$$X_{window} = \left( \frac{X_{ndc}+1}{2} \right) \mathbf{w} + \mathbf{x}_{lb}$$
$$Y_{window} = \left( \frac{Y_{ndc}+1}{2} \right) \mathbf{h} + \mathbf{y}_{lb}$$



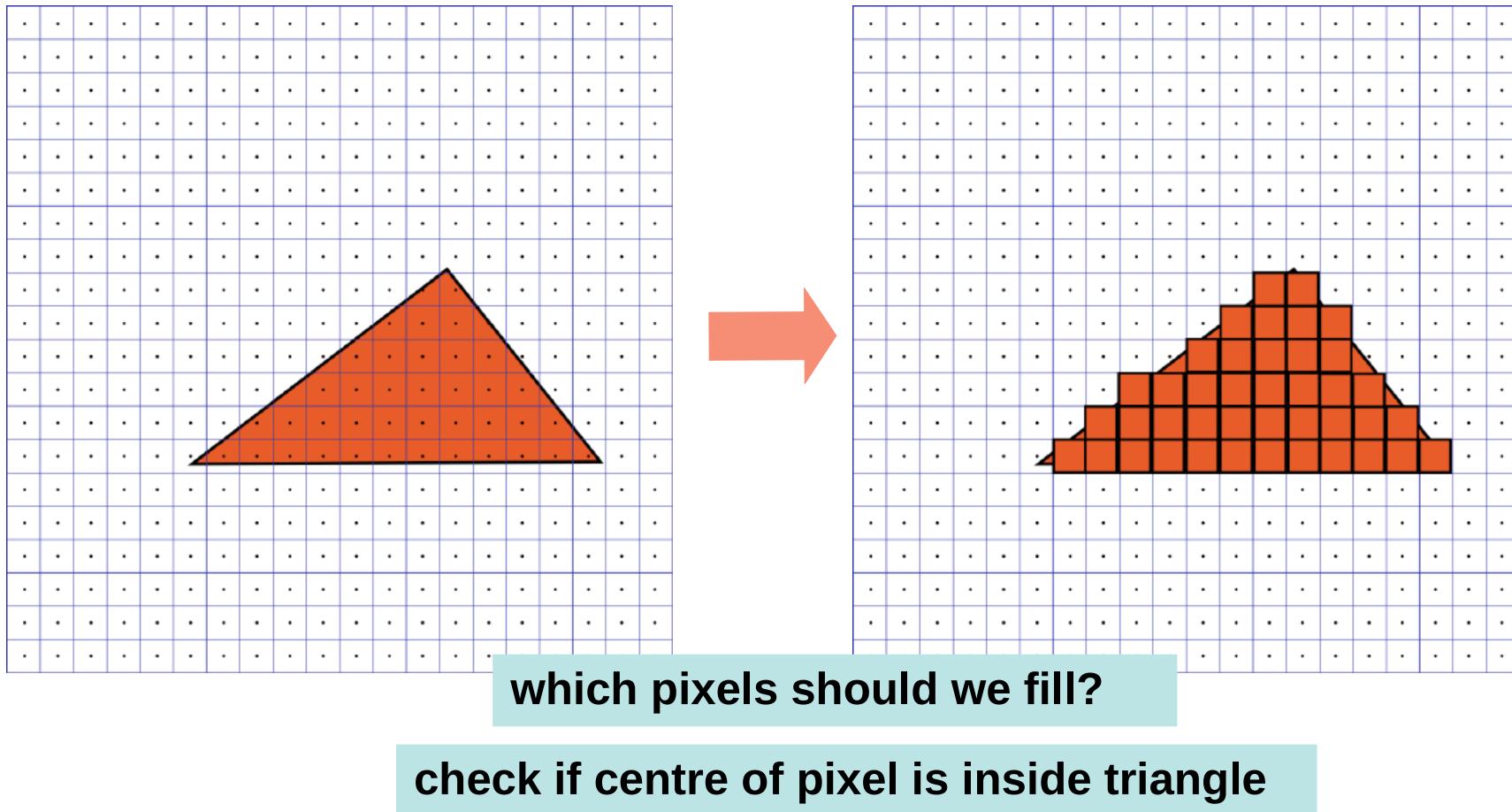
# graphics pipeline



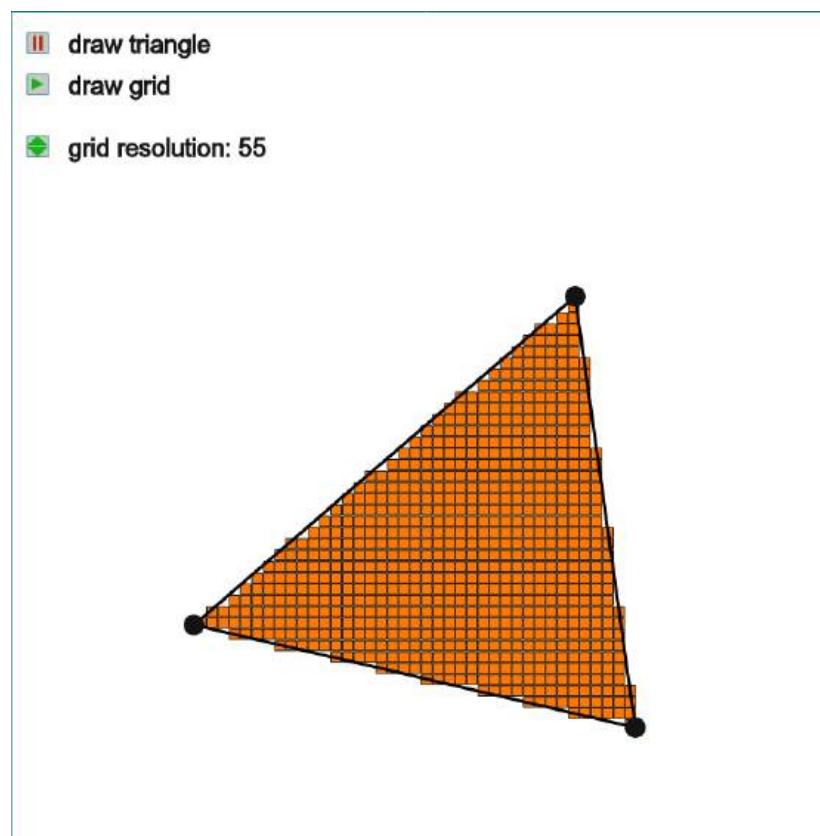
# graphics pipeline: transformations



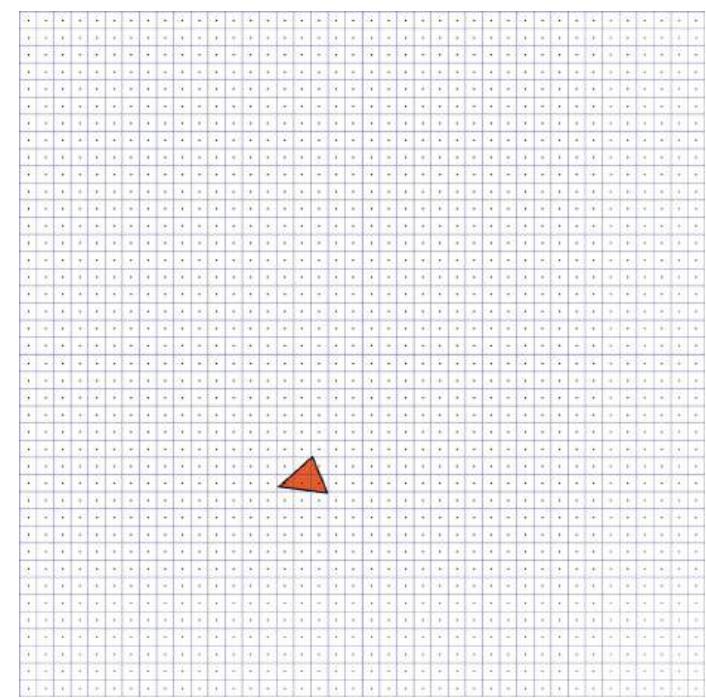
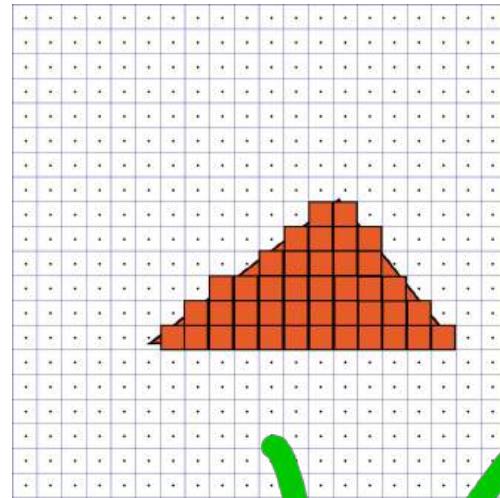
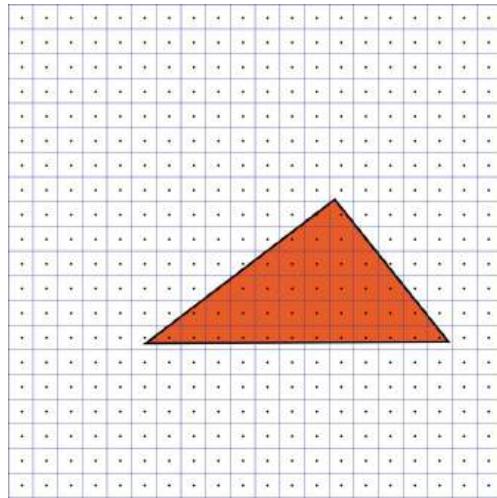
# rasterization



# demo



# rasterization



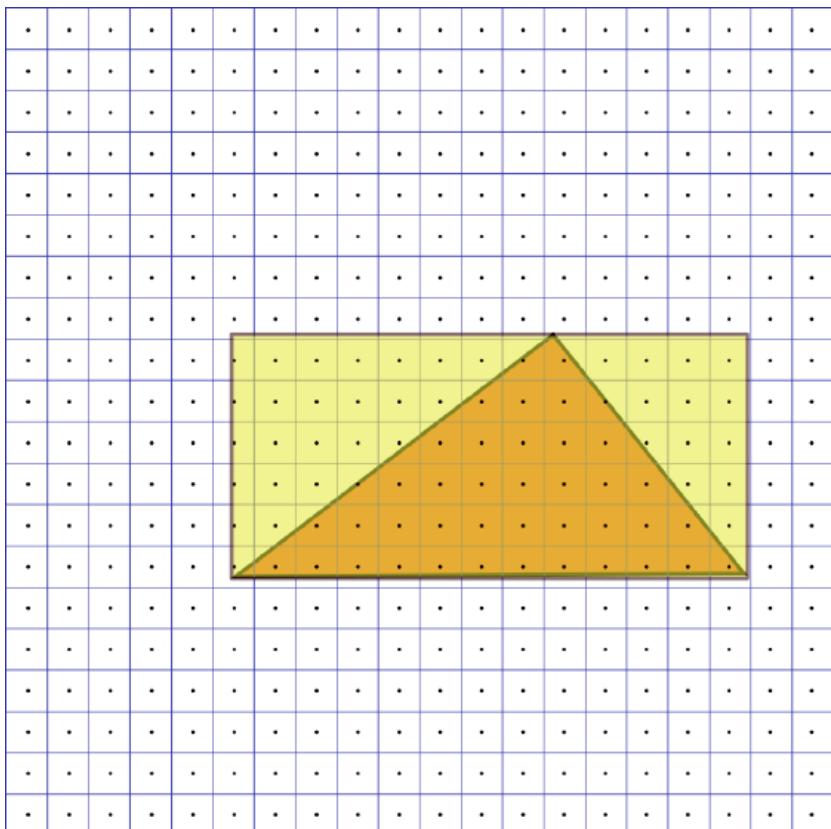
which pixels should we fill?

test if each pixel centre is  
inside/outside triangle

issues?

check all pixels for  
each triangle is not  
a good idea!

# question

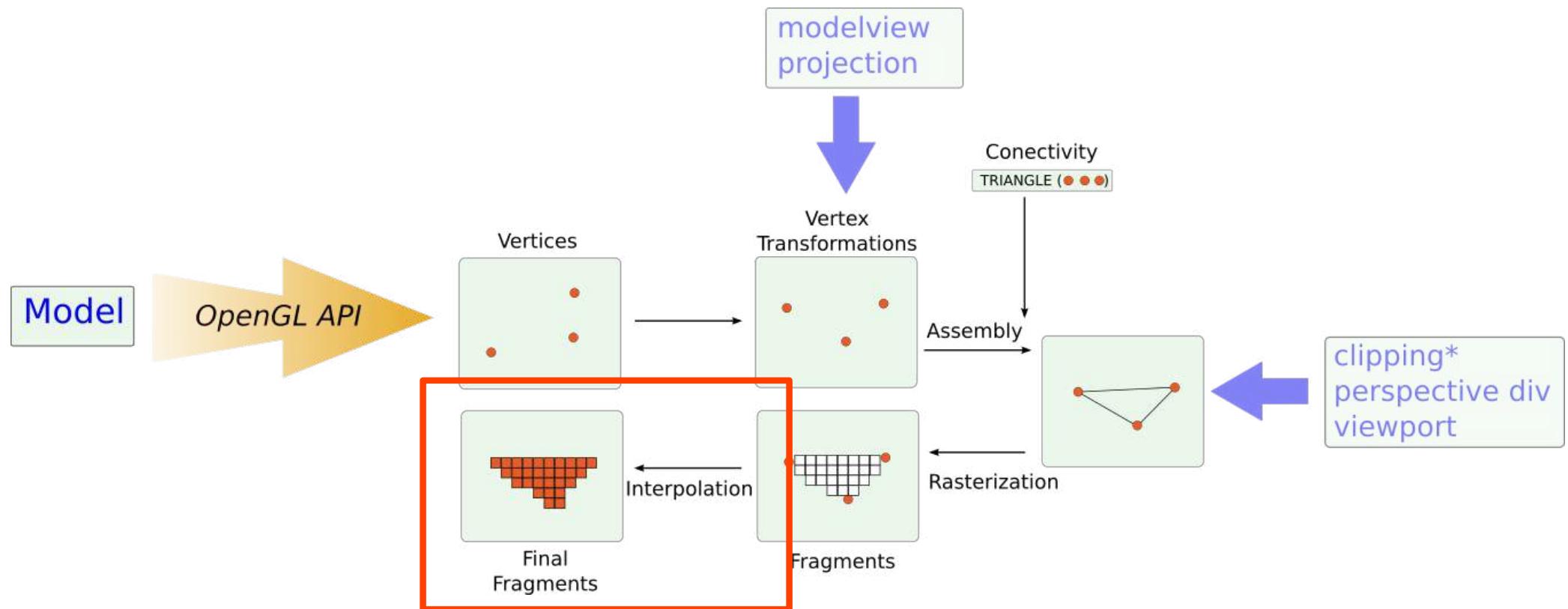


Axis-Aligned Bounding Box (AABB)

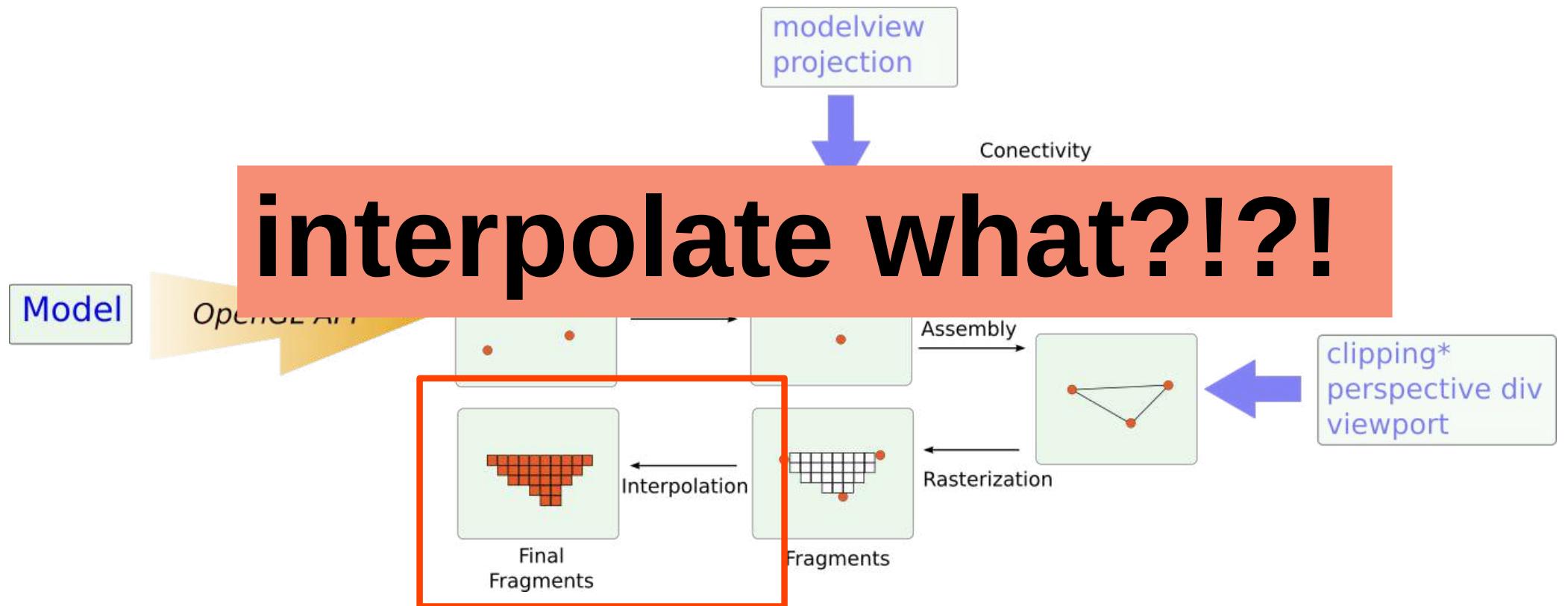
if pixel inside triangle → inside box

given the window coordinates (x,y) of the three vertices, how do we find the AABB?  
With the AABB which pixels do we need to check (design a loop to iterate over the pixels)

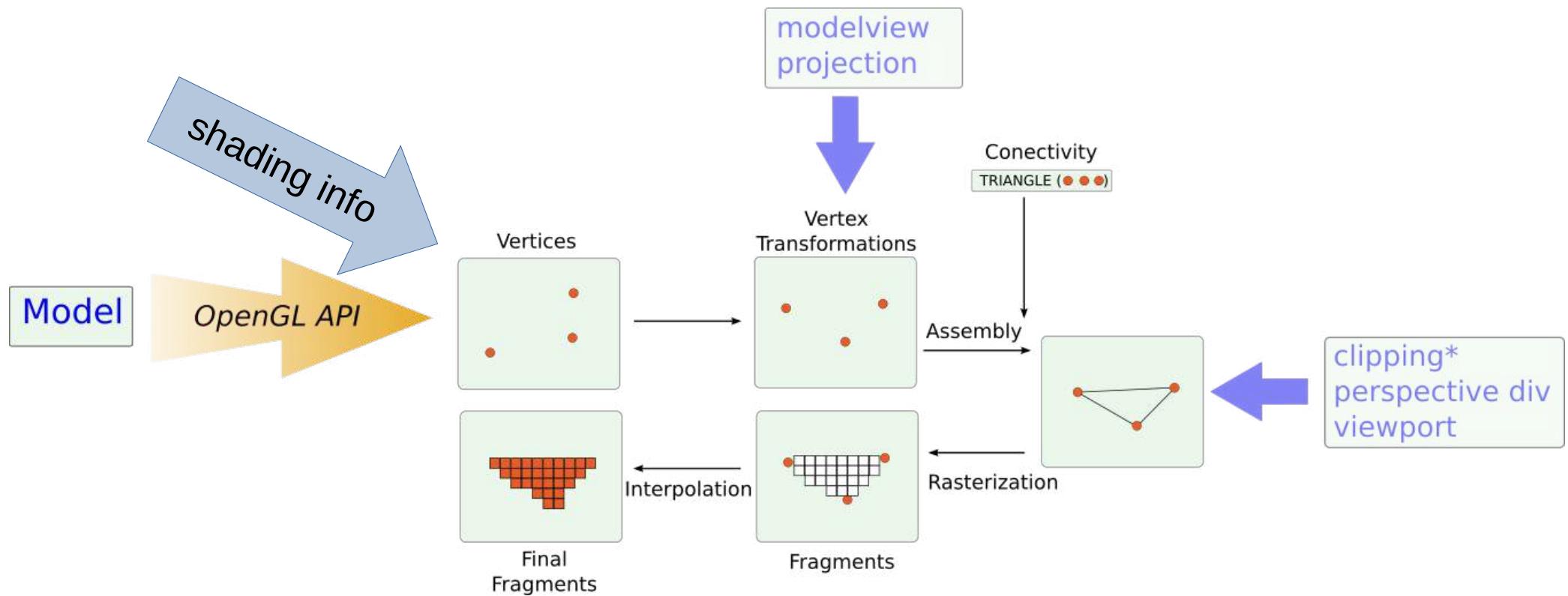
# graphics pipeline



# graphics pipeline



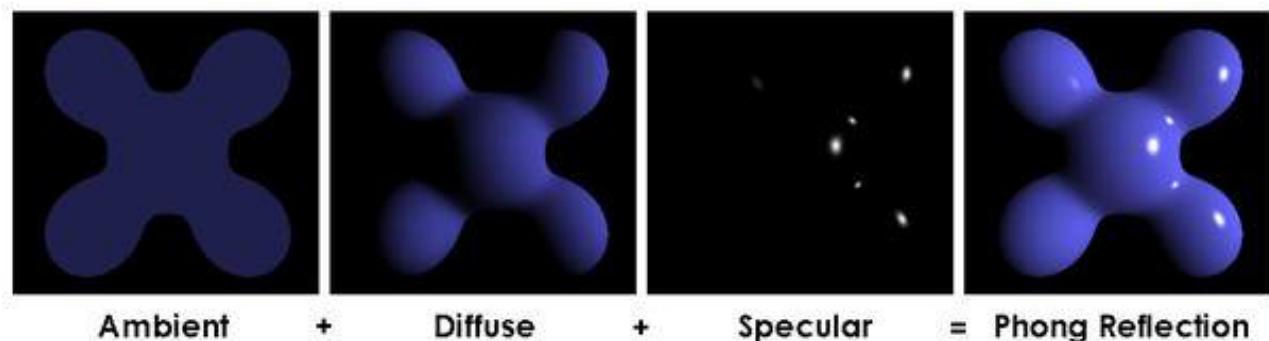
# e.g. colours!!!



# shading (Phong model)

Phong Model: Sum of 3 terms

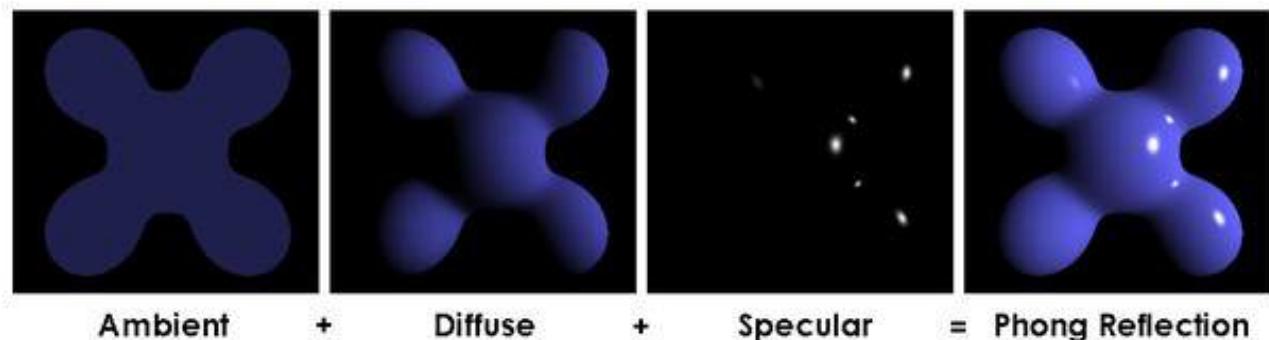
- Ambient
- Diffuse
- Specular



# shading (Phong model)

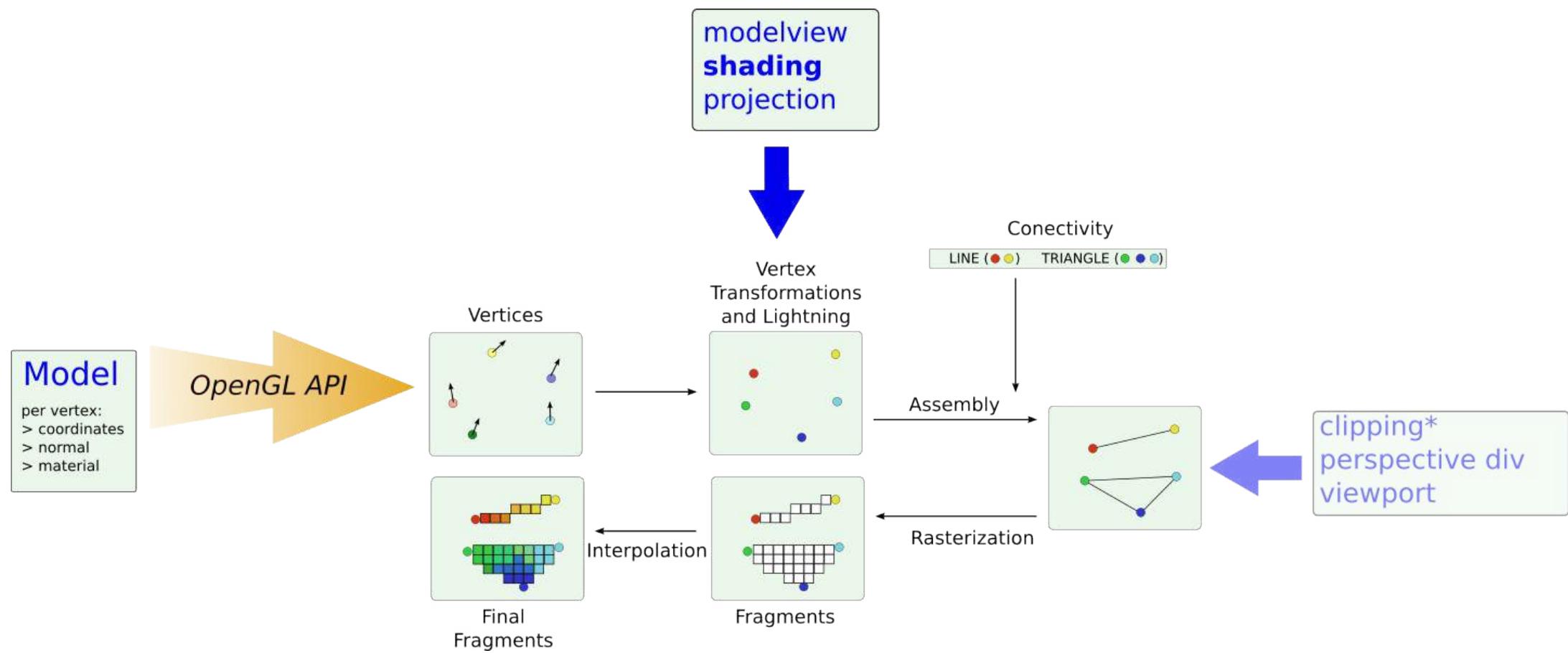
Phong Model: Sum of 3 terms

- Ambient
- Diffuse
- Specular



- light and view info
- surface material ( $ka$ ,  $kd$ ,  $ks$ )
- normal

# e.g., shading information



# lighting (Phong Model)

define each light source

glEnable ( GL\_LIGHTING )

glDisable ( GL\_LIGHTING )

```
glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient)  
glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse)  
glLightfv(GL_LIGHT0, GL_SPECULAR, light0_specular)  
glLightfv(GL_LIGHT0, GL_POSITION, light0_position)  
  
glEnable(GL_LIGHT0)
```

instead of glColor define the complete material

obs: Normal is needed for shading

```
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular)  
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess)  
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse)
```

use only RGB to fake material

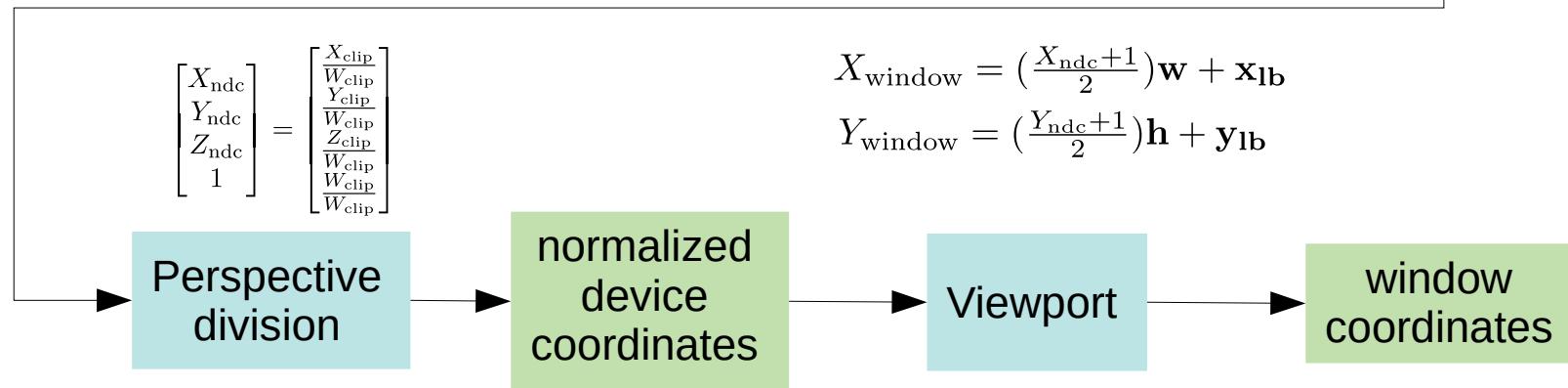
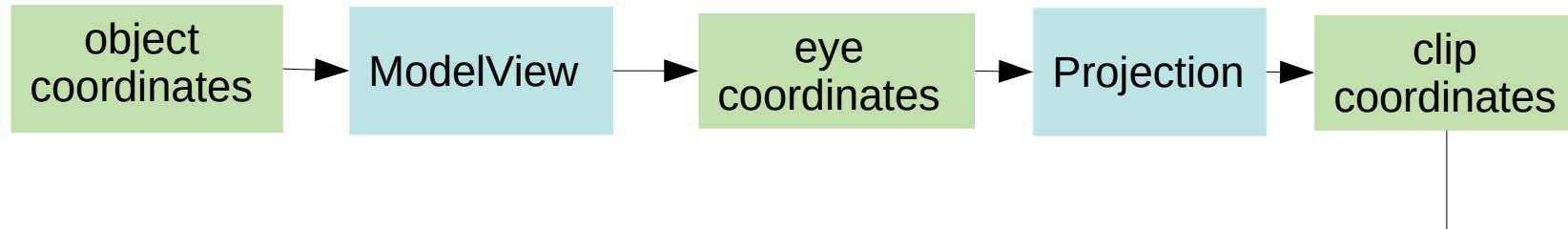
```
glEnable ( GL_COLOR_MATERIAL )
```

this is for completeness!

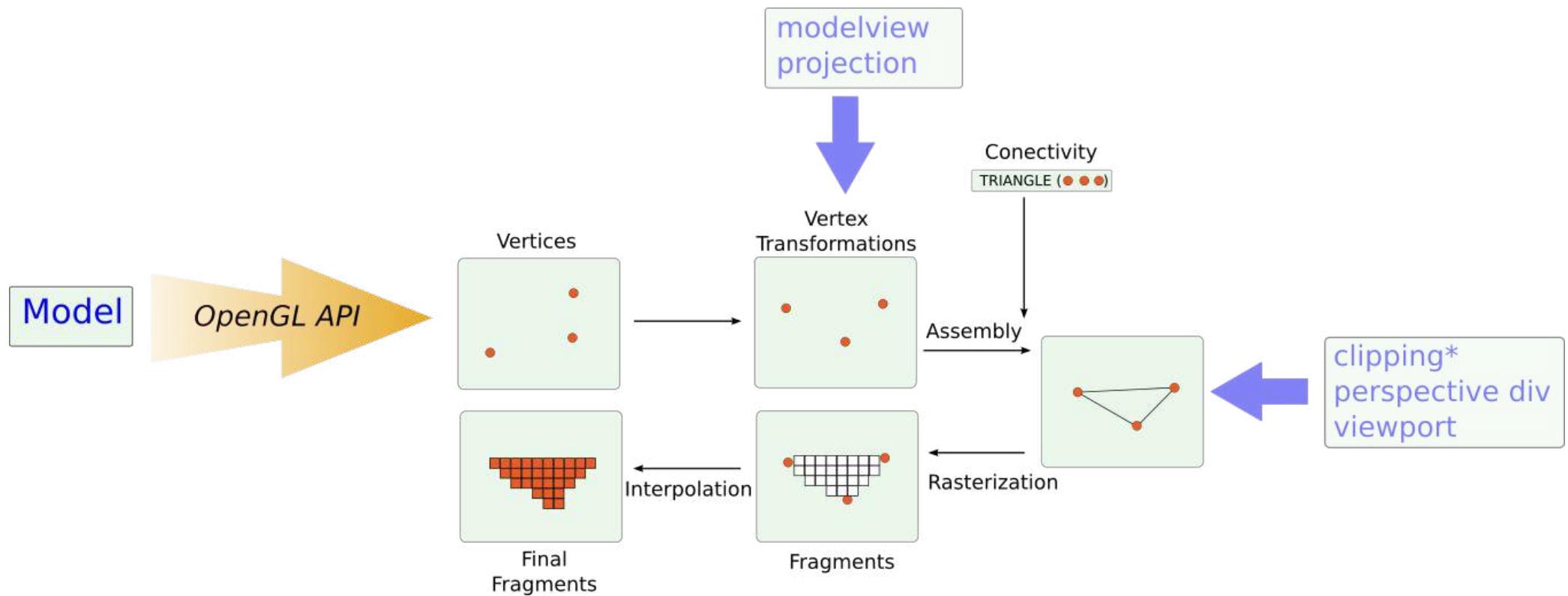
For Visual Debug  
you can mostly  
use Color Material

# today: OpenGL transformation pipeline

$$\begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} X_{clip} \\ Y_{clip} \\ Z_{clip} \\ W_{clip} \end{bmatrix} = \begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{near}+\text{far}}{\text{near}-\text{far}} & \frac{2*\text{near}*\text{far}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ 1 \end{bmatrix}$$

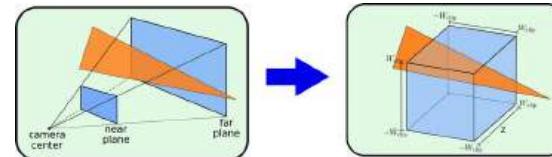


# graphics pipeline



# questions

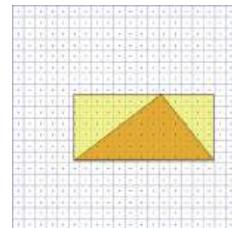
- design a simple clipping algorithm



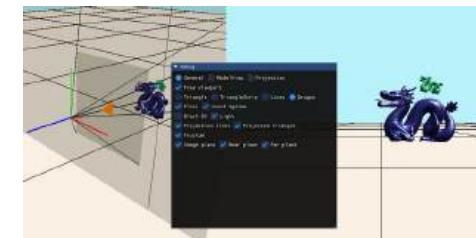
- explain if fisheye lens is possible with our camera model



- find triangle's AABB

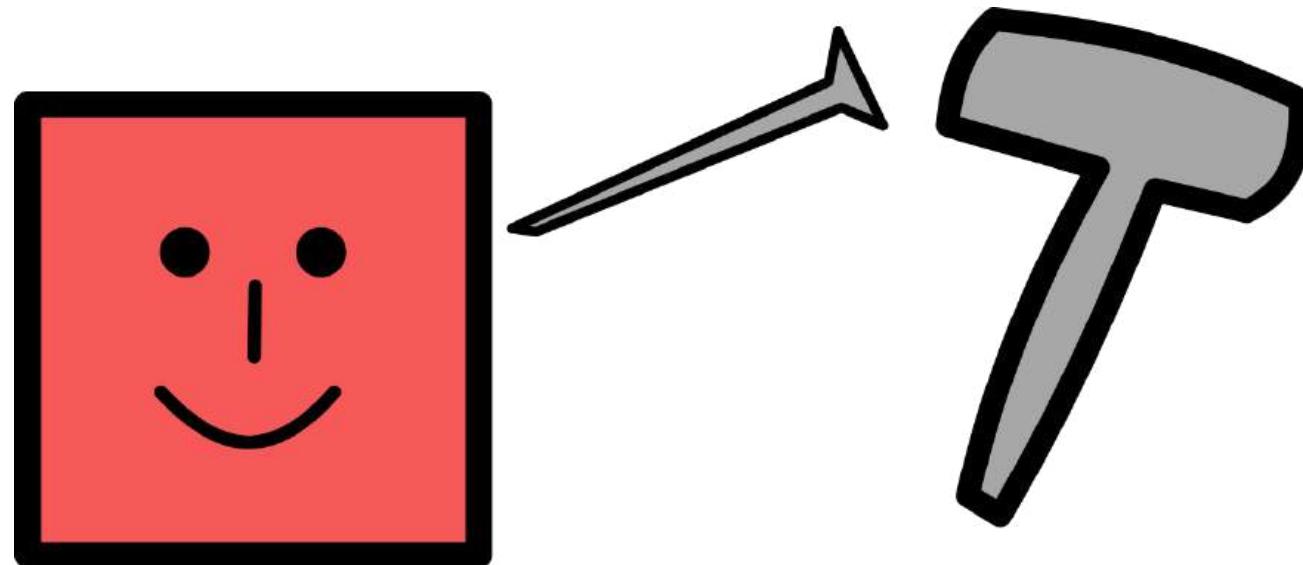


- Rebuild the demo and introduce new features



# how to make a pixel

CSE2215 Computer Graphics



Ricardo Marroquim

Delft University of Technology (TU Delft)

# CSE2215 - Computer Graphics

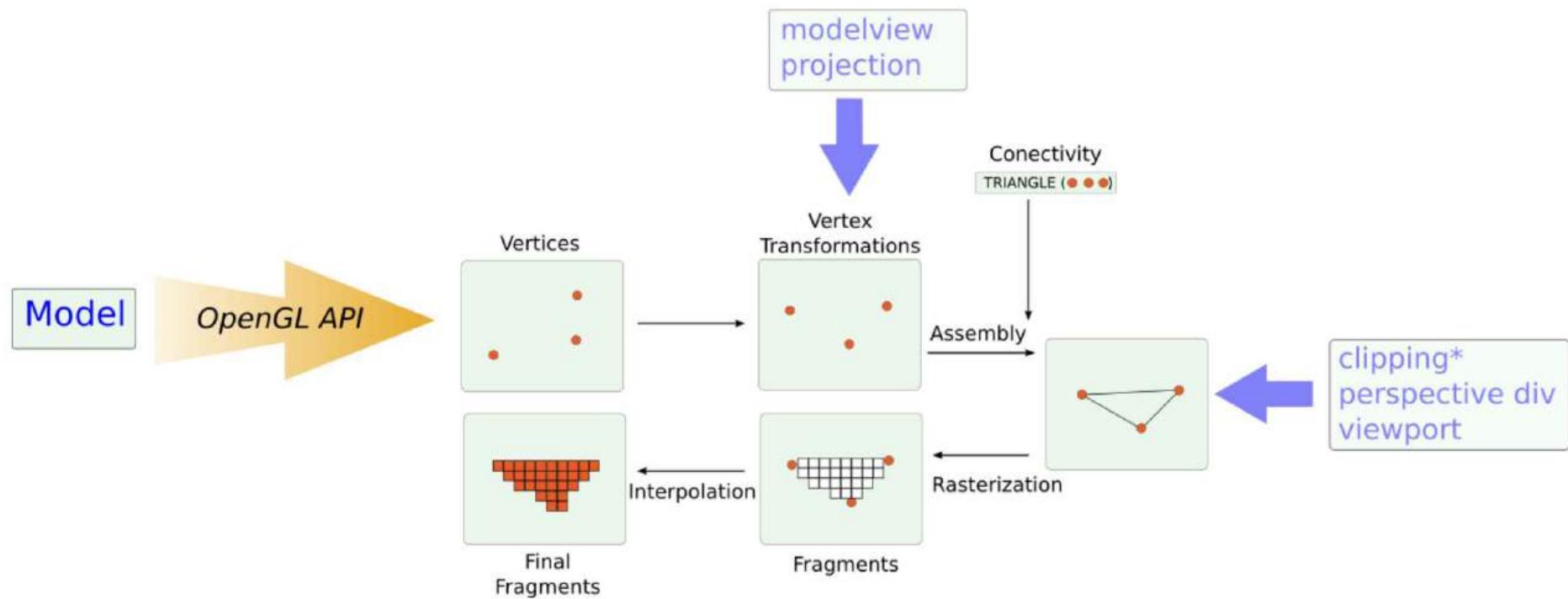
## Textures An image says more than...

Elmar Eisemann

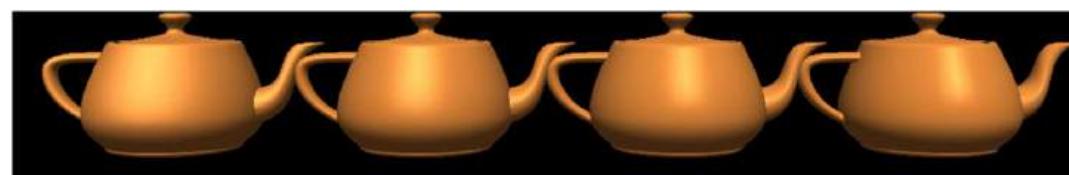
Delft University of Technology



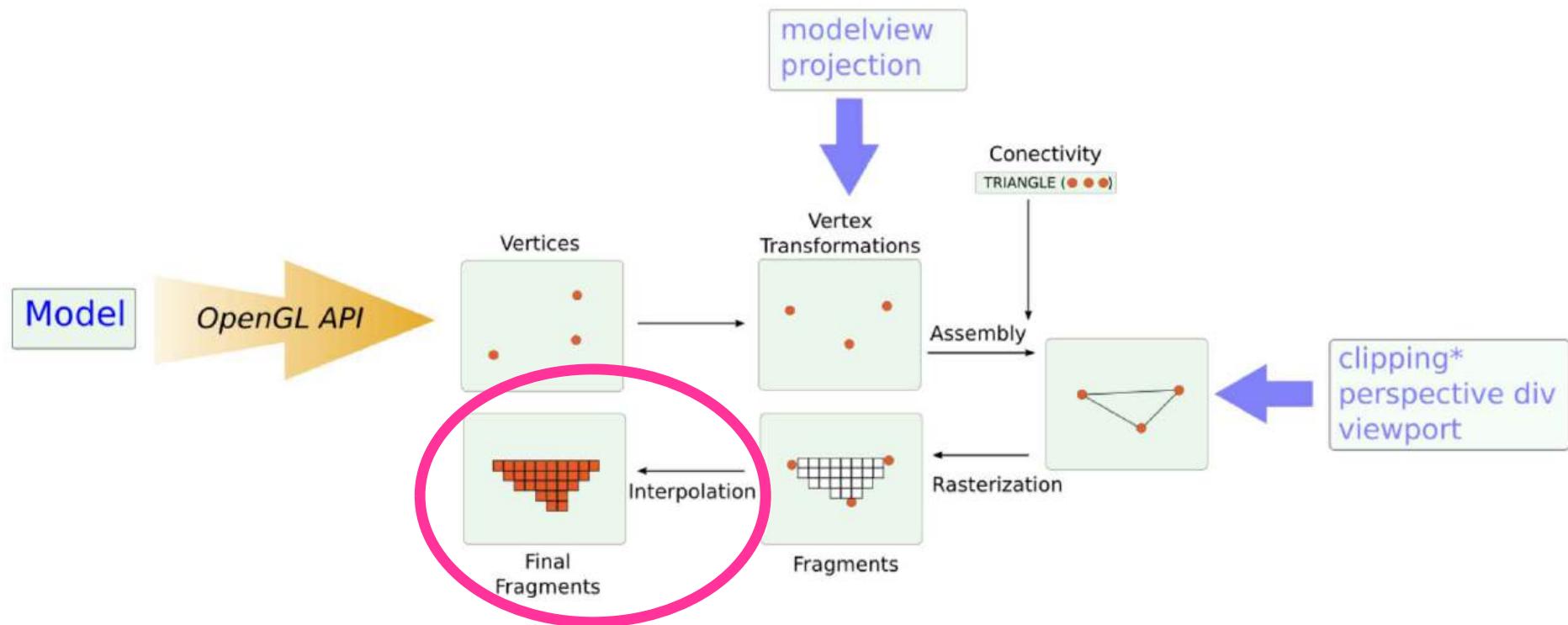
# Graphics Pipeline



## Shading



# Graphics Pipeline



## Reminder: How to apply Phong Model ?

- We know how to compute shading of a point,  
but how is it applied on a triangle mesh?

# Shading

- *Early days - compute color per face:*  
*Flat shading* produces “facets”



- Later – compute color per vertex:  
produces *Gouraud Shading* produces a smooth look



## Phong shading

- Today: compute result per pixel
- Phong Shading leads to smooth specularities

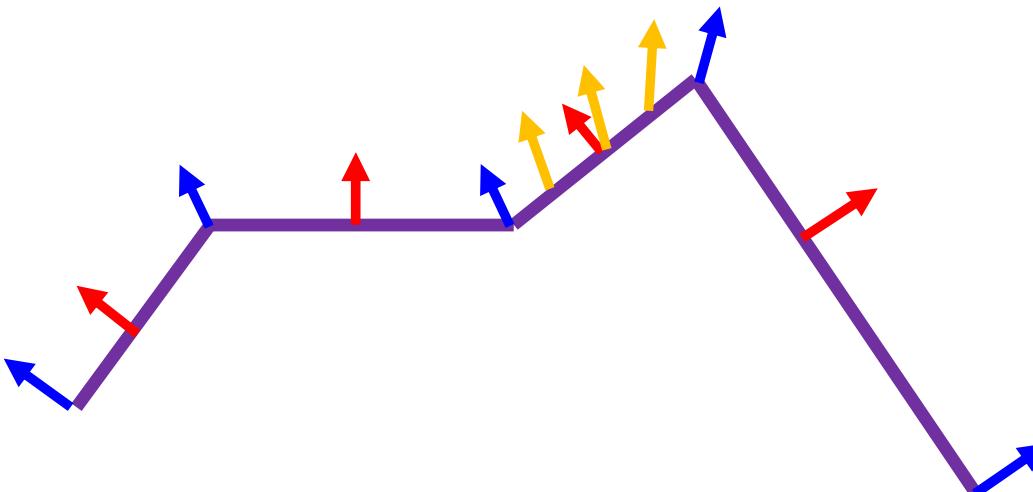


Diccan.com

- Phong interpolates normals from vertices over pixels

## Normals on Meshes

- Face normals (normal of the plane containing triangle)
- Vertex normals (e.g., average neighboring face normal)
- Interpolated normal (interpolate vertex normals over triangle)

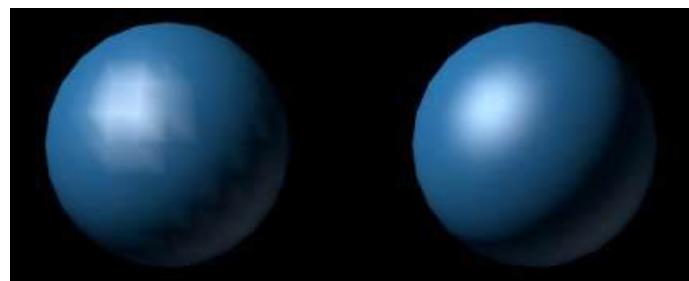


## Gouraud vs. Phong shading

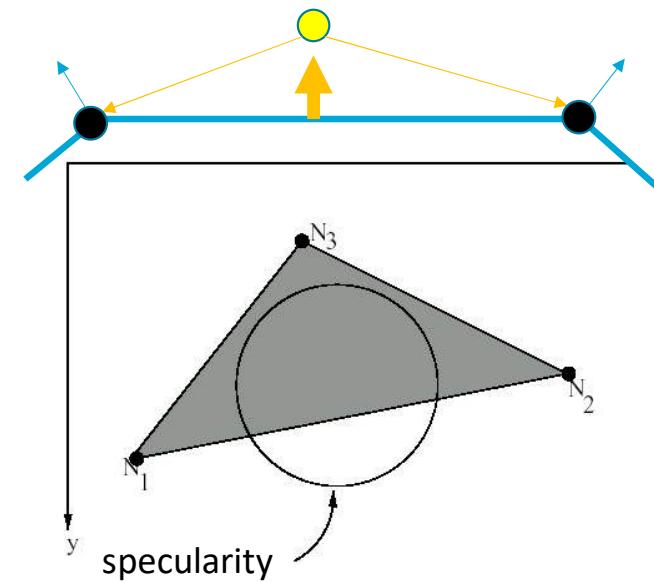
*Per vertex*

*Per pixel*

- Phong usually more expensive than Gouraud
  - Because there are often more pixels than vertices
- Phong is more beautiful and minimal standard
  - Captures specularities between faces



Diccan.com



## On the practical side: Shading types

- How are the three different types computed?

- *Flat shading*

- Applies Phong Model to produce a color per face

- *Gouraud shading*

- Applies Phong to produce a color per vertex

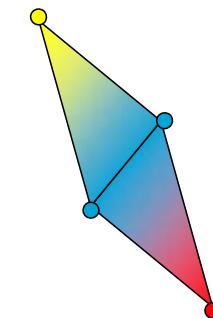
- Interpolate color from vertices over triangle

- *Phong shading*

2 MEANINGS!!!

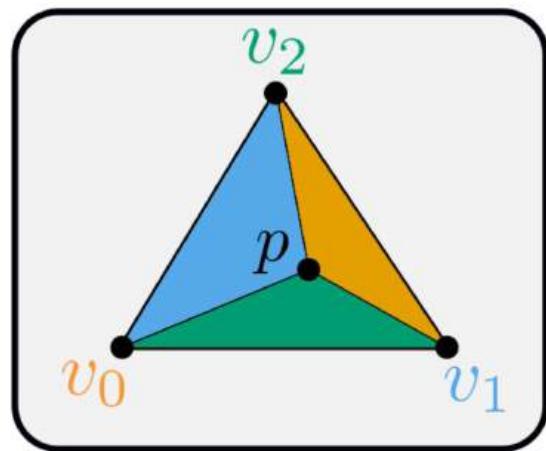
- Interpolate parameters of Phong model

- Applies Phong to produce a color per pixel



## Interpolation on a Triangle

- Last time you went through barycentric coordinates



weights are related to areas

$$p = \alpha v_0 + \beta v_1 + \gamma v_2$$

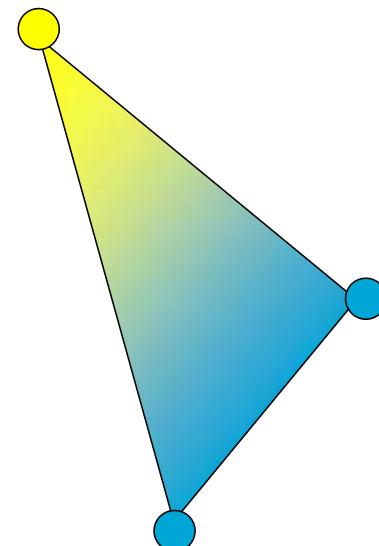
$$\alpha = \frac{A(pv_1v_2)}{A(v_0v_1v_2)}$$

$$\beta = \frac{A(pv_0v_2)}{A(v_0v_1v_2)}$$

$$\gamma = \frac{A(pv_0v_1)}{A(v_0v_1v_2)}$$

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

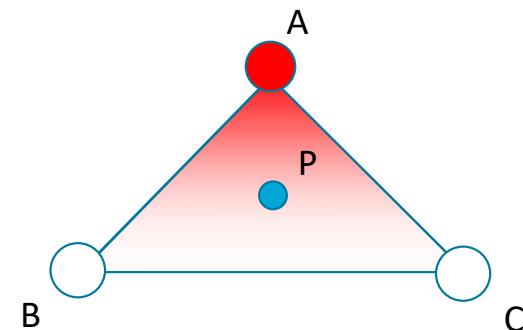
$$\alpha, \beta, \gamma > 0$$



## Interpolation on a Triangle

- Imagine  $P = 1/3 A + 1/3 B + 1/3 C$
- If the colors at A, B, C are red (1,0,0), white (1,1,1), white (1,1,1) respectively
- The interpolated color at P would be:  
 $(1, 2/3, 2/3)$  pink

Do you know  
how this  
color is  
called?



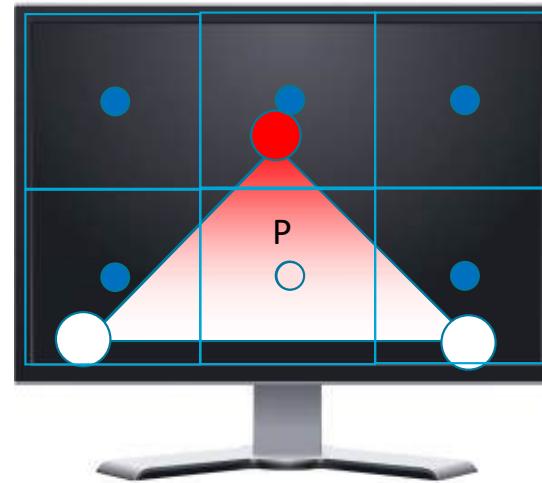
## Rasterization

- Interpolated values are used for rasterization. Here is an amazing 6x2 pixel display...



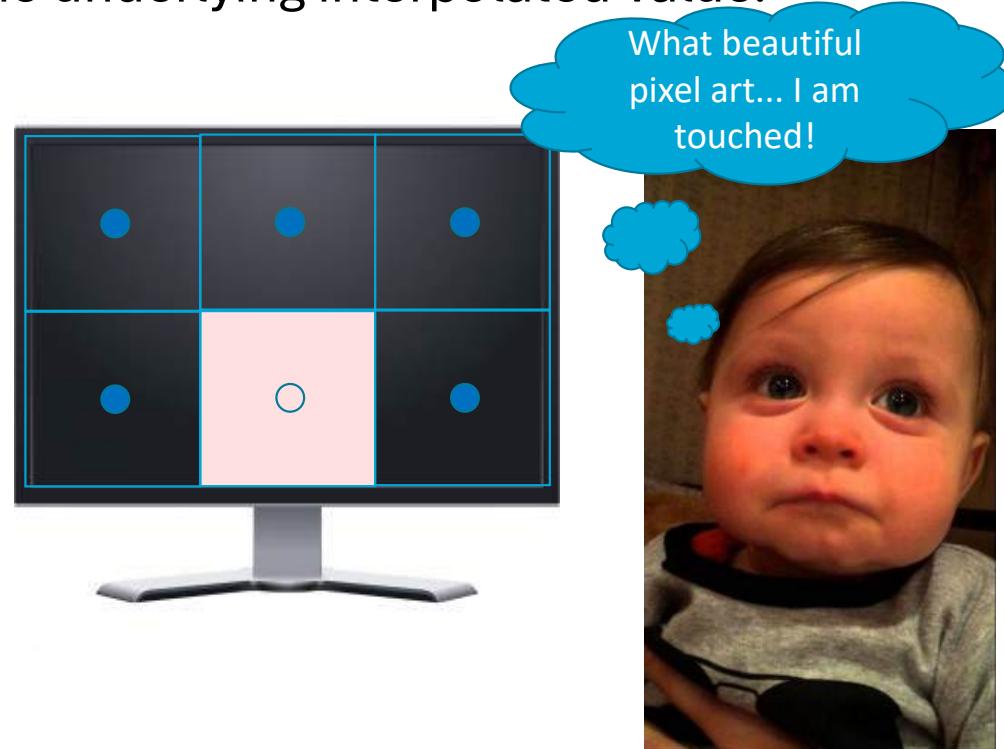
## Rasterization

- For the pixel center, find the underlying interpolated value.



# Rasterization

- For the pixel center, find the underlying interpolated value.



## Joke aside...

- There is some truth to it...
- For now, everything looks “kind of clean”...



## Textures: Adding small scale details

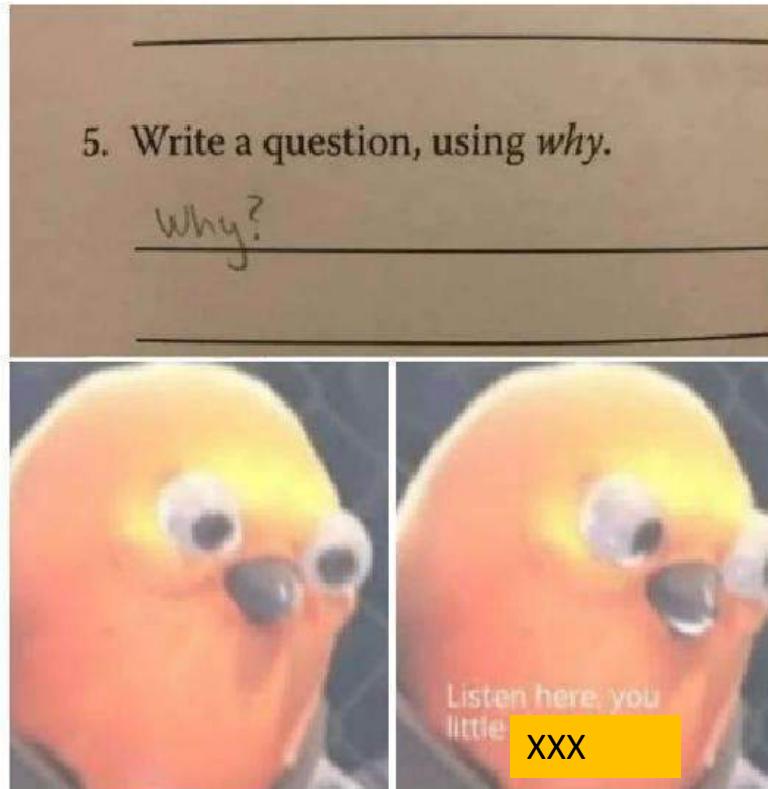
- No textures



- Textures



## Questions?



## Study Goals

- S1- Explain and compare the structure and properties of standard algorithms and data structures linked to Computer Graphics.
- S3- Use mathematical methods to analyze, create, apply algorithms and data structures, as well as understanding time and space complexity of image-generation algorithms



You could represent all color changes with triangles... but it is too expensive because it generates a lot of geometry...  
Textures avoid this problem!

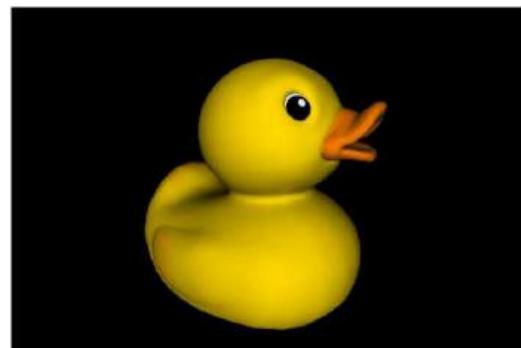
# Textures

- Mapping an image onto a surface

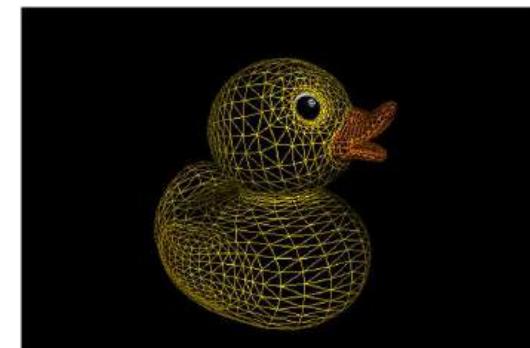
Texture



Texture mapped triangles



Wireframe of triangles

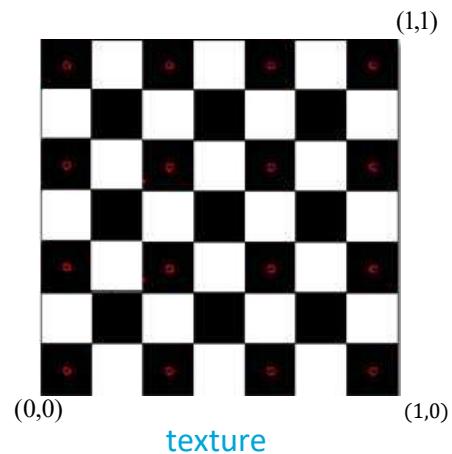


consisting of *texels*  
(texture pixels)\*

\* Some say **texture elements**

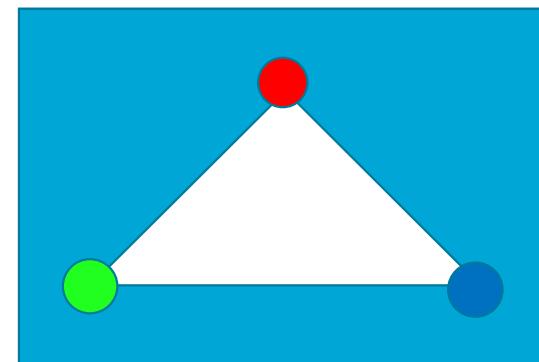
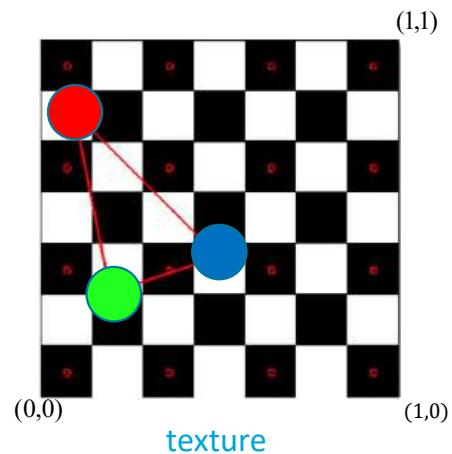
# Textures

- Map image via texture coordinates



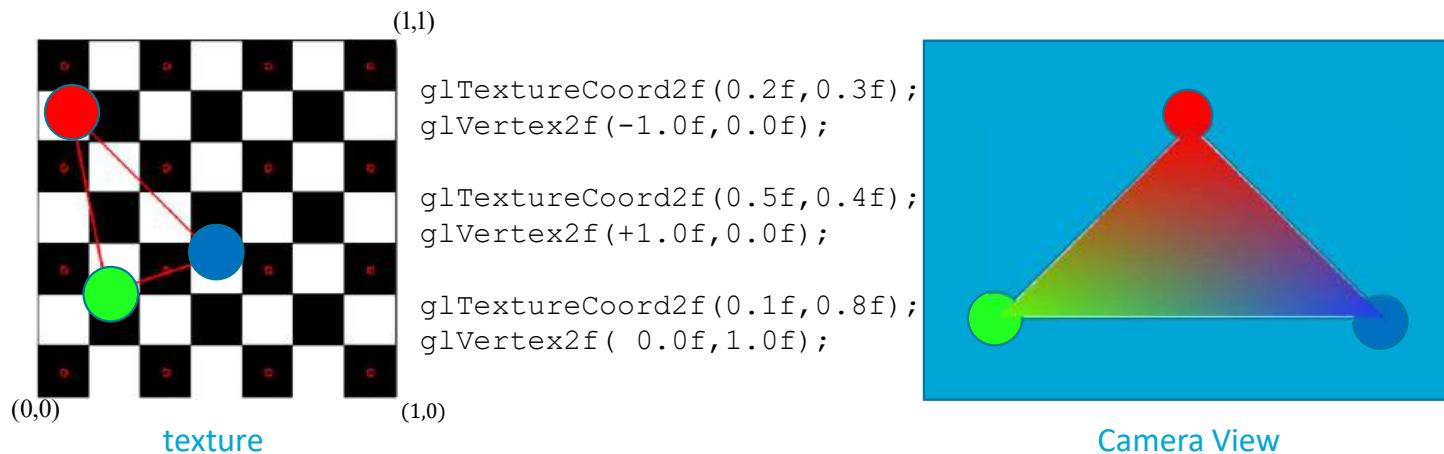
# Textures

- Map image via texture coordinates
  - Specify a texture coordinate at each vertex



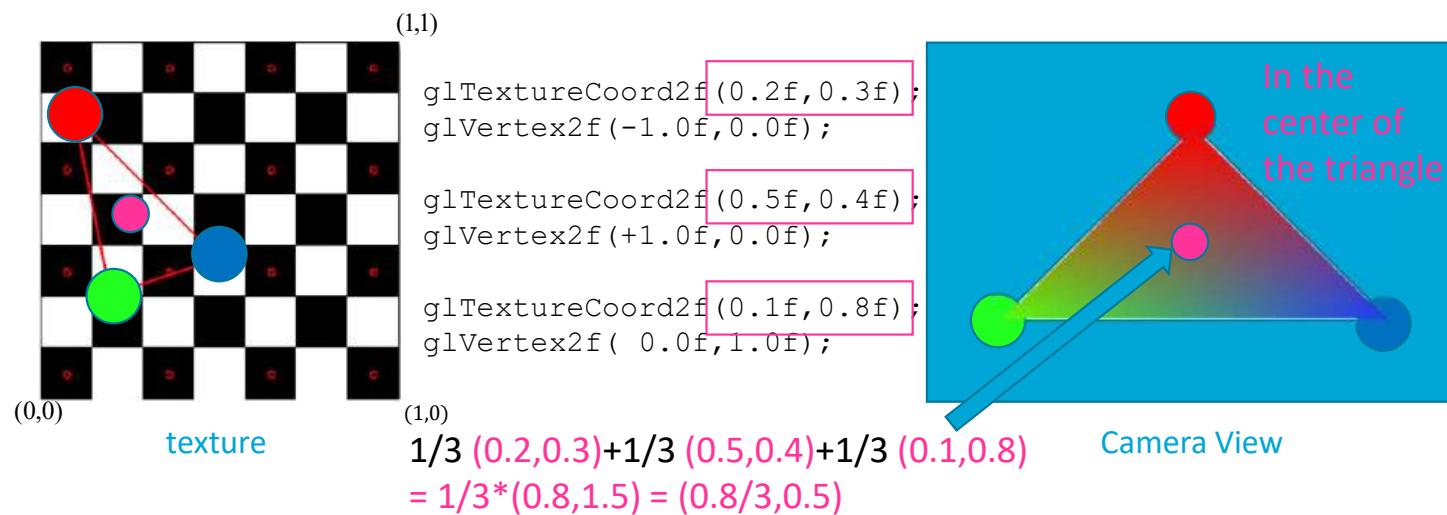
# Textures

- Map image via texture coordinates
  - Specify a texture coordinate at each vertex
  - Vertex texture coordinates are interpolated over triangle



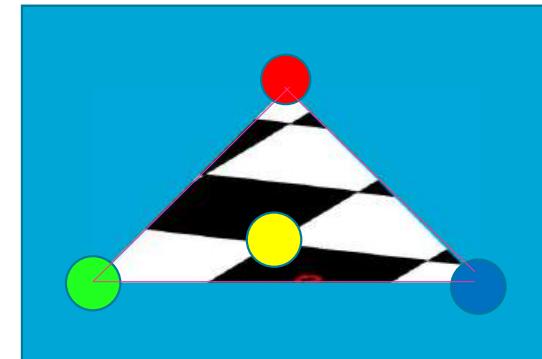
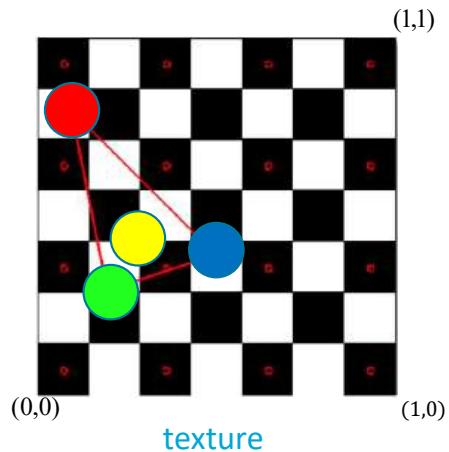
# Textures

- Map image via texture coordinates
  - Specify a texture coordinate at each vertex
  - Vertex texture coordinates are interpolated over triangle
  - Drawn pixels use interpolated coordinates to retrieve texel values



# Textures

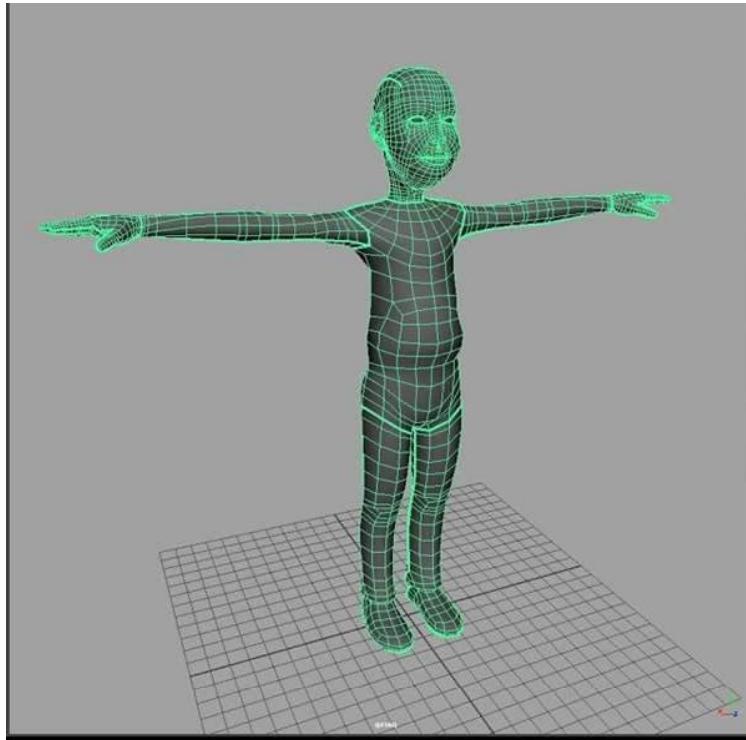
- Map image via texture coordinates
  - Specify a texture coordinate at each vertex
  - Vertex texture coordinates are interpolated over triangle
  - Drawn pixels use interpolated coordinates to retrieve texel values



Camera View

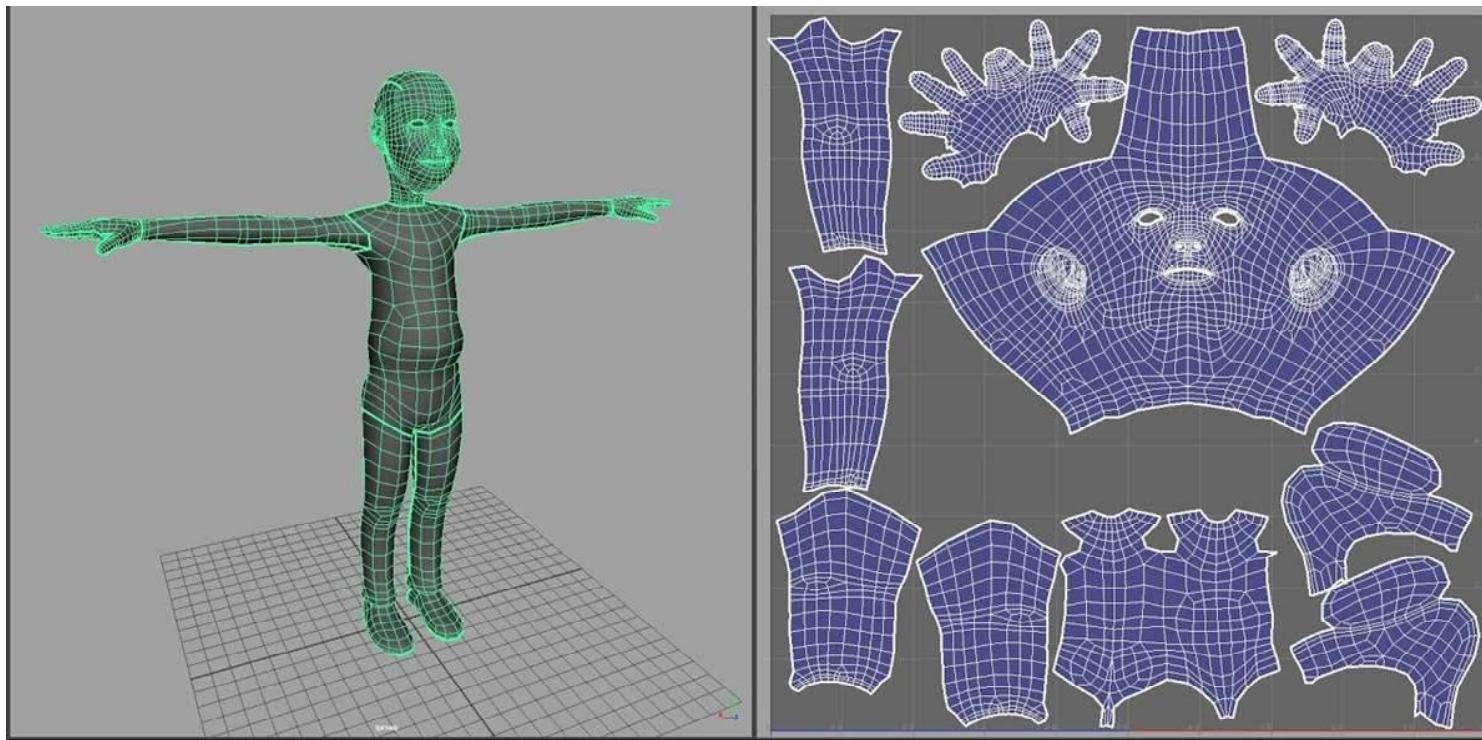
## How to define Texture Coordinates?

- Common start: Mesh Unwrapping



## How to define Texture Coordinates?

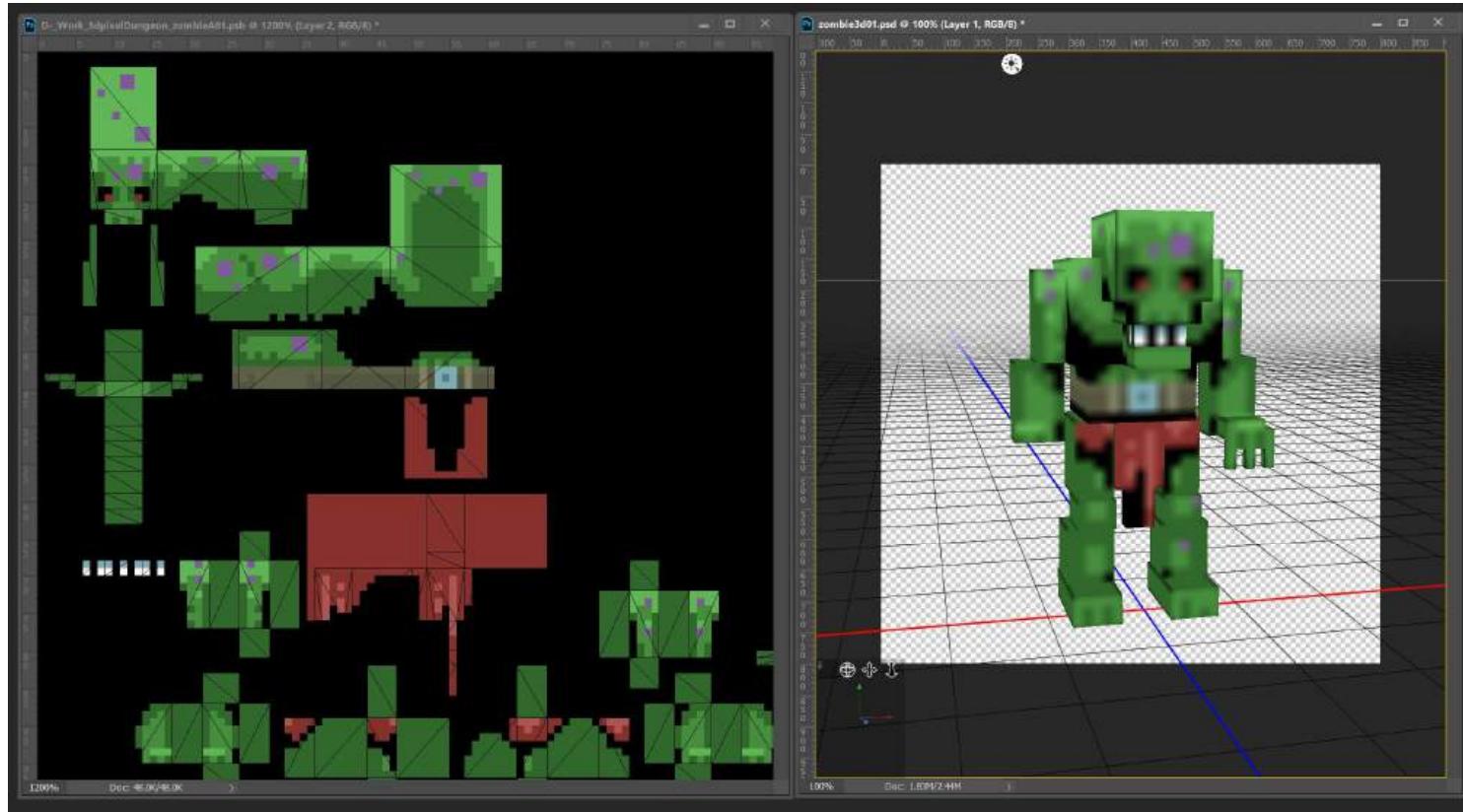
- Common start: Mesh Unwrapping



## Specialized Software

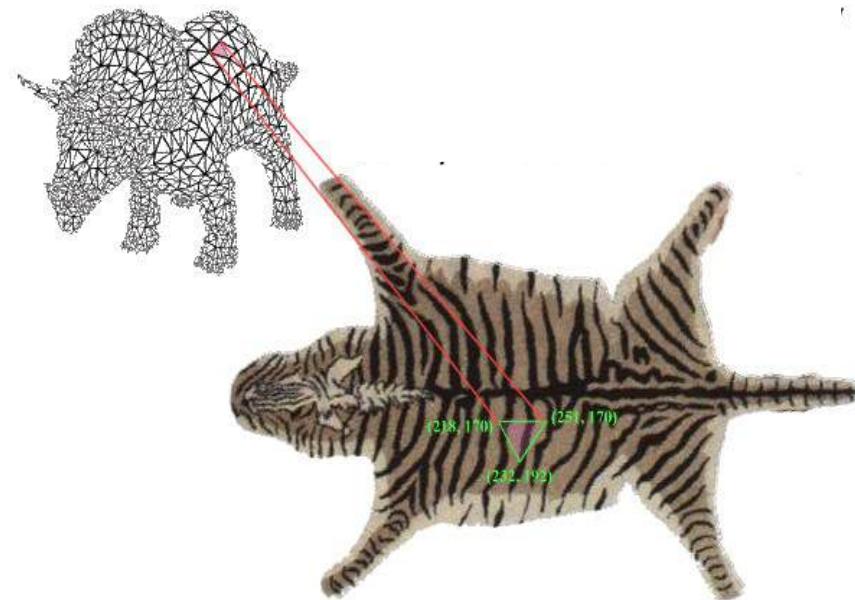
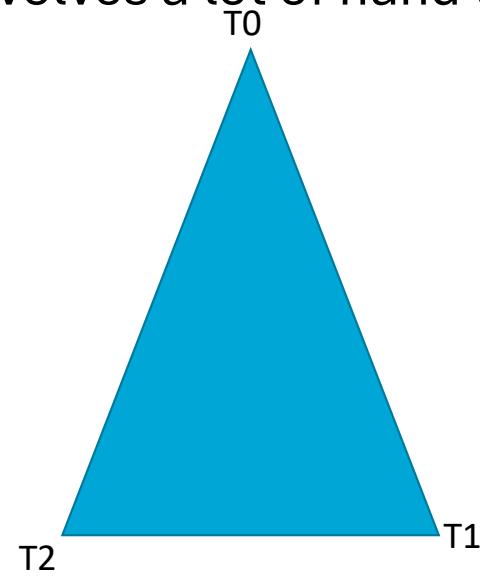


## Specialized Software



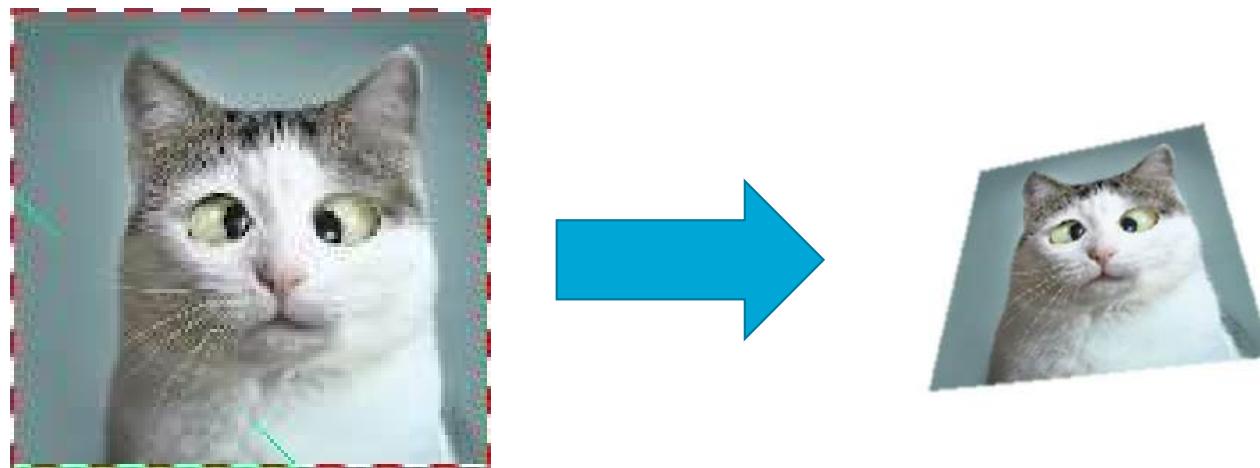
## How to define Texture Coordinates?

- Often involves a lot of hand tweaking!



Texture coordinates need to be defined for each vertex but are then interpolated over the triangle

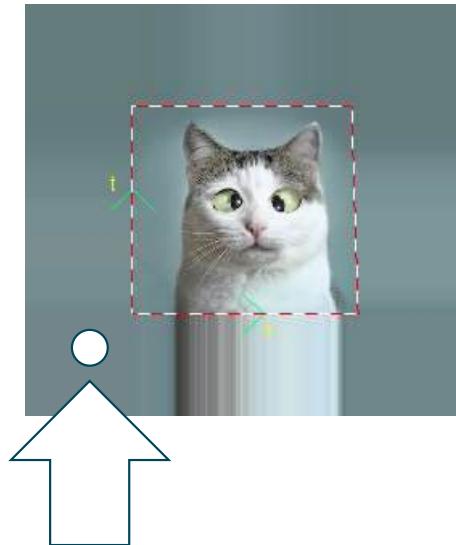
## Example of a mapped texture



## What happens outside the texture?

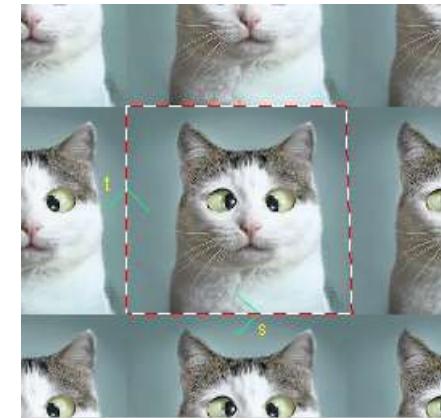
- Different modes can be defined **per axis**:

**Border** = constant color



Texture coordinate?  
(-0.25, -0.25)

**Clamp** = keep texel value on the border

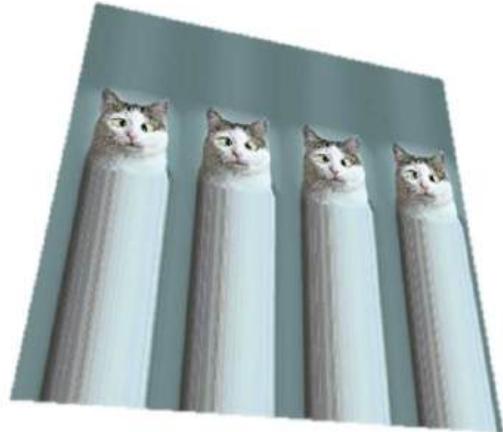


**Repeat** = repeat at borders

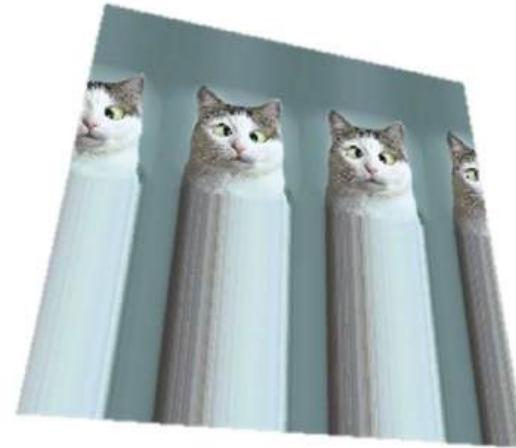


# Demo Time!!!

## What happened here?



- Vertex texture coordinates are  $(-2,-2), (-2,2), (2,-2), (2,2)$
- X axis: repeat
- Y axis: clamp



- Vertex texture coordinates: same with -1.5 and 1.5
- X axis: repeat
- Y axis: clamp

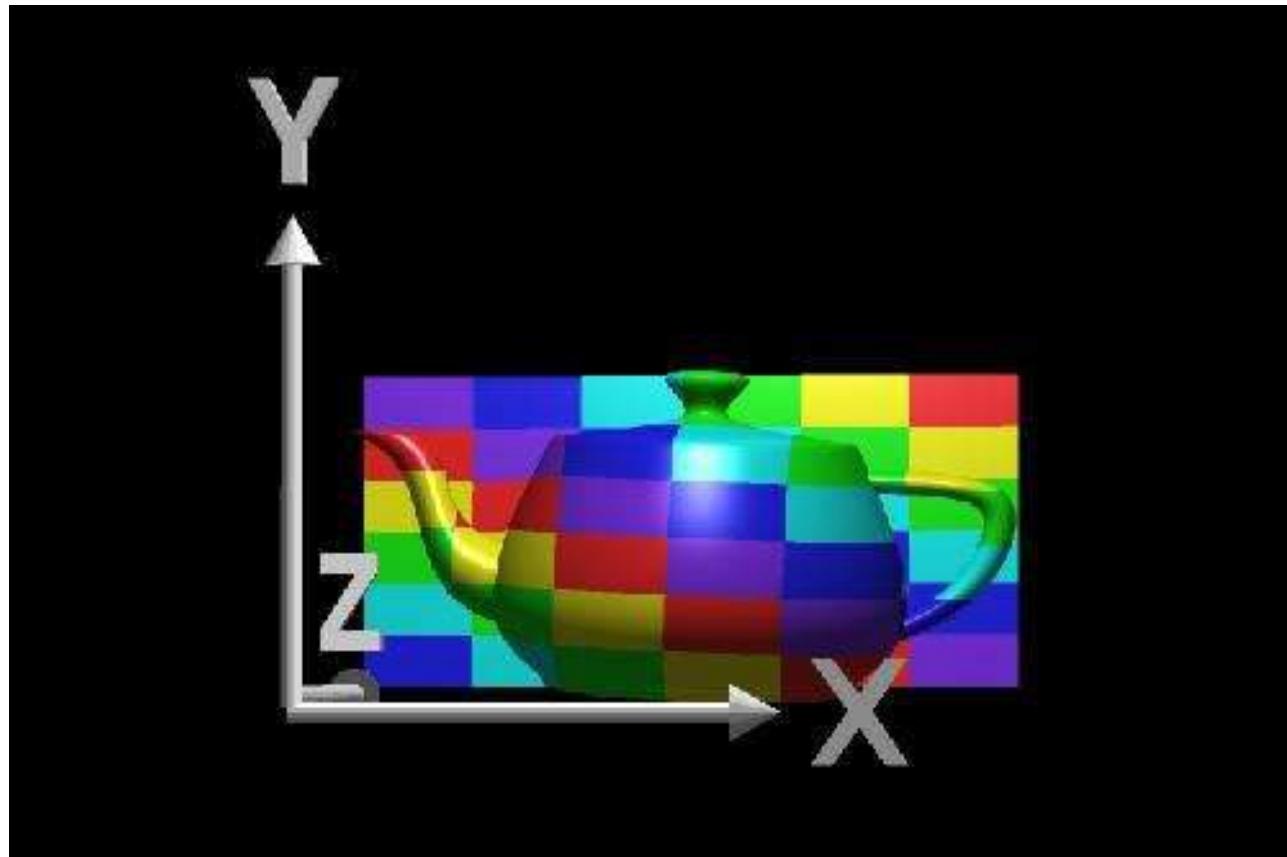
## How else to define Tex Coords?

- Geometric Texture Mapping:
- Define a function  $T$  that takes a vertex position  $(x,y,z)$  in  $R^3$  and maps it to texture coordinates  $(u,v)=T(x,y,z)$
- For example:

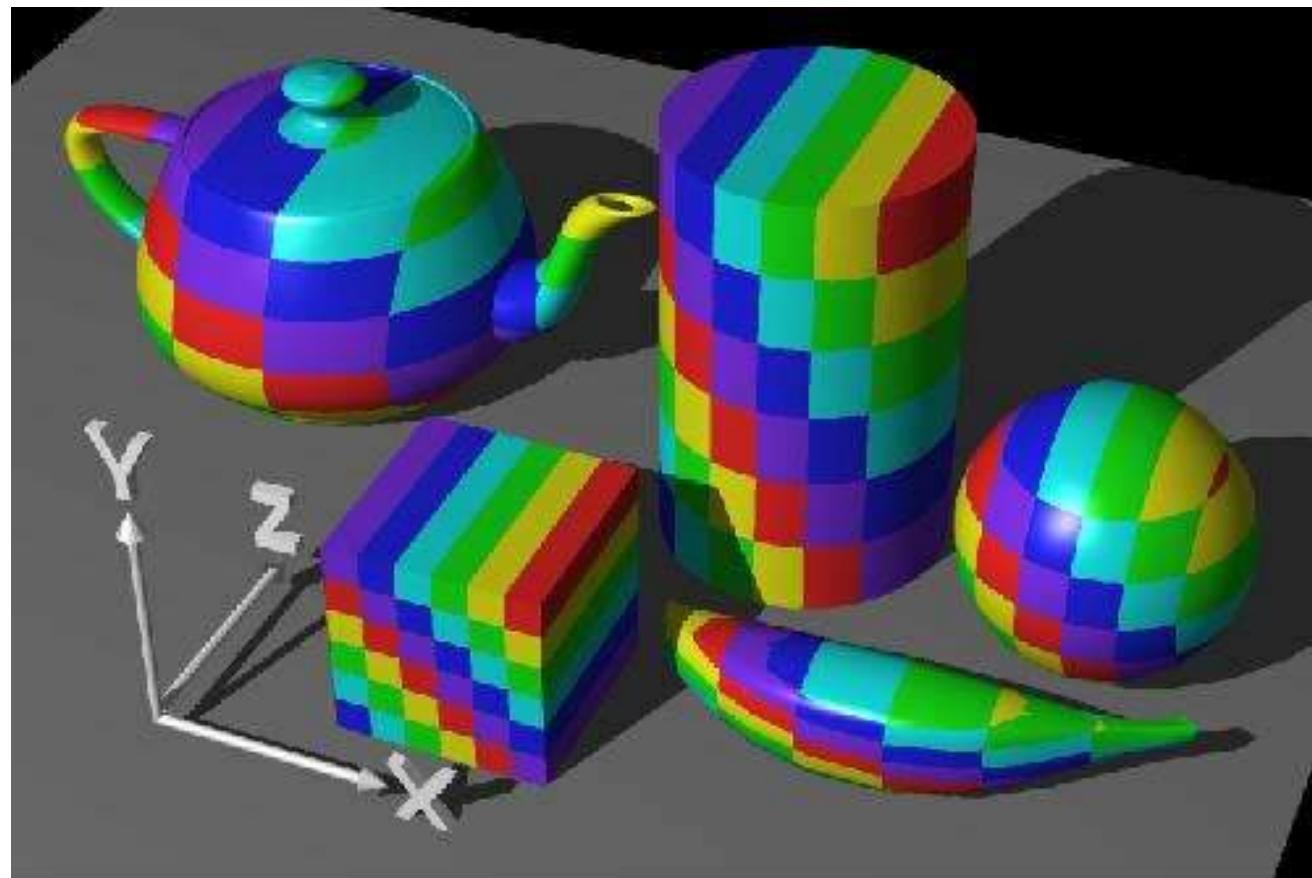
$$T(x,y,z)=(x,y)$$



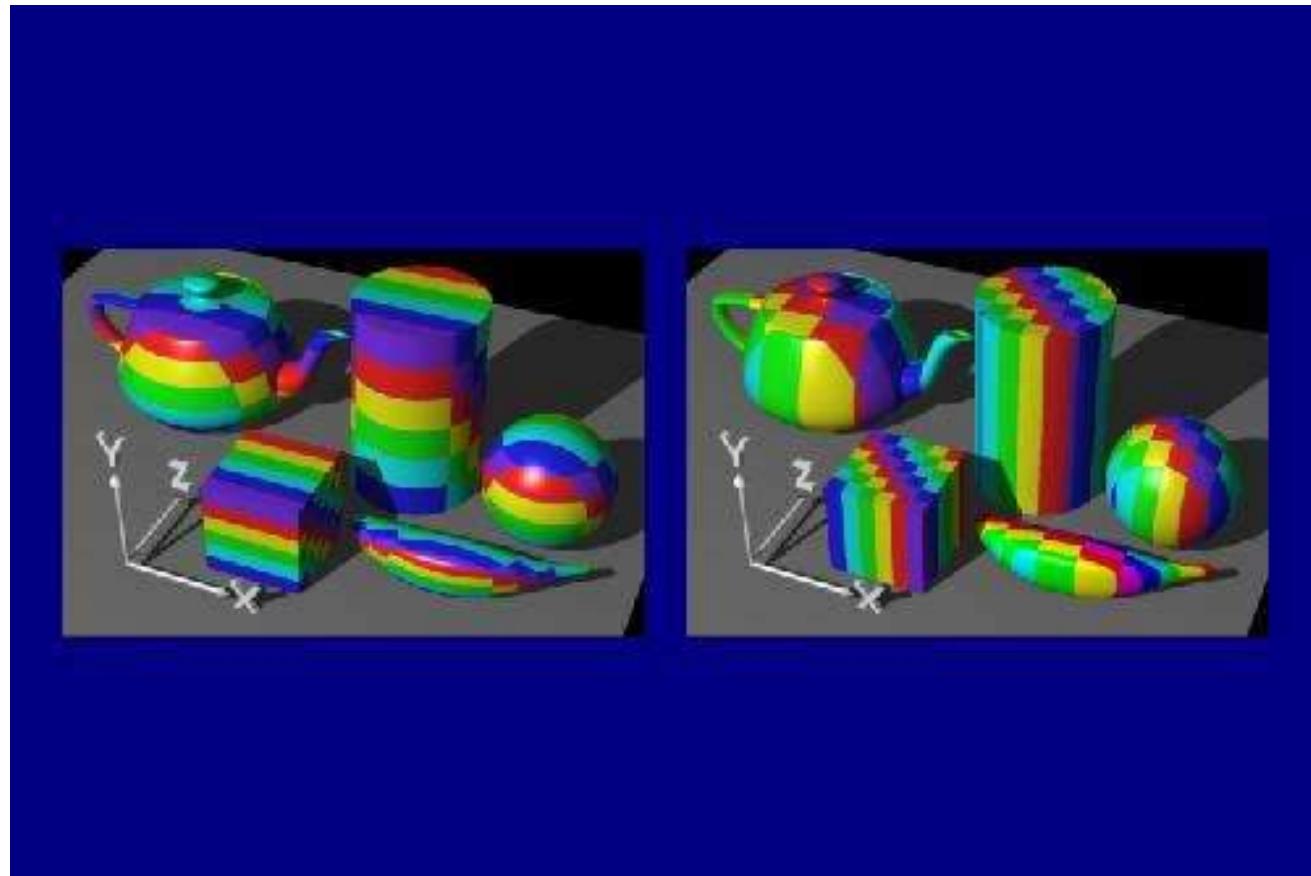
XY



XY

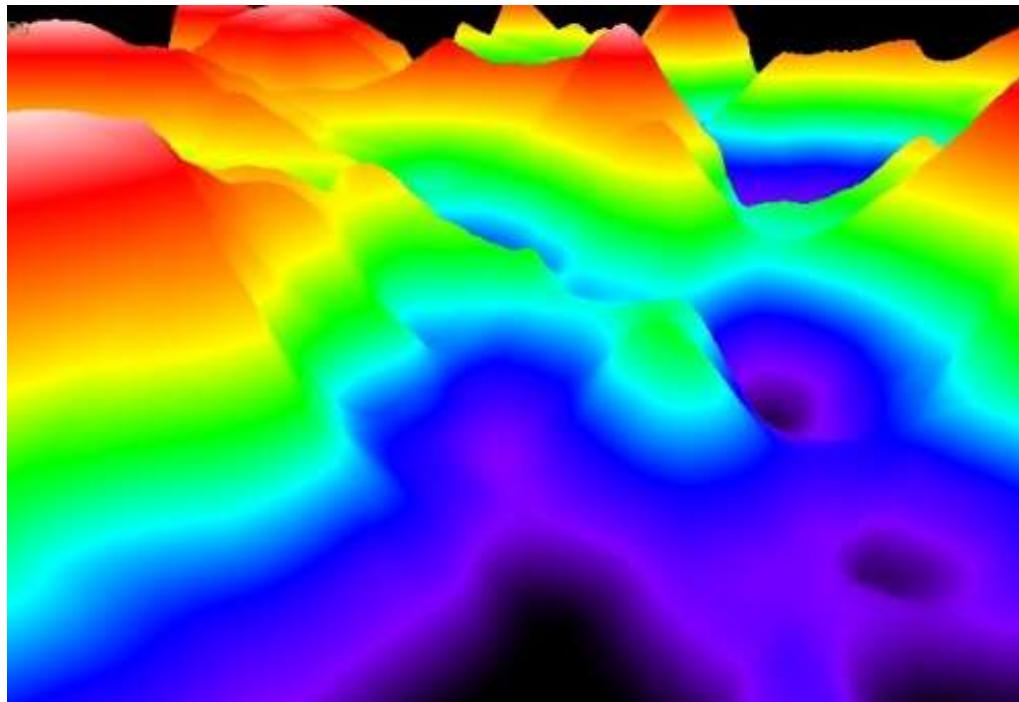


YZ/ZX



## How could you do this?

- E.g., map to  $(0, z)$

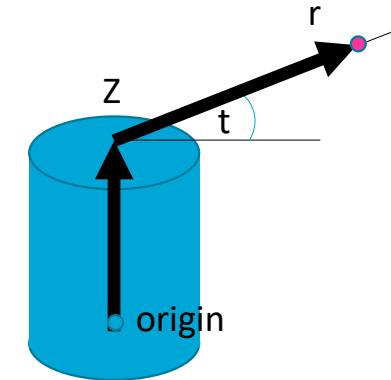


# Cylinder

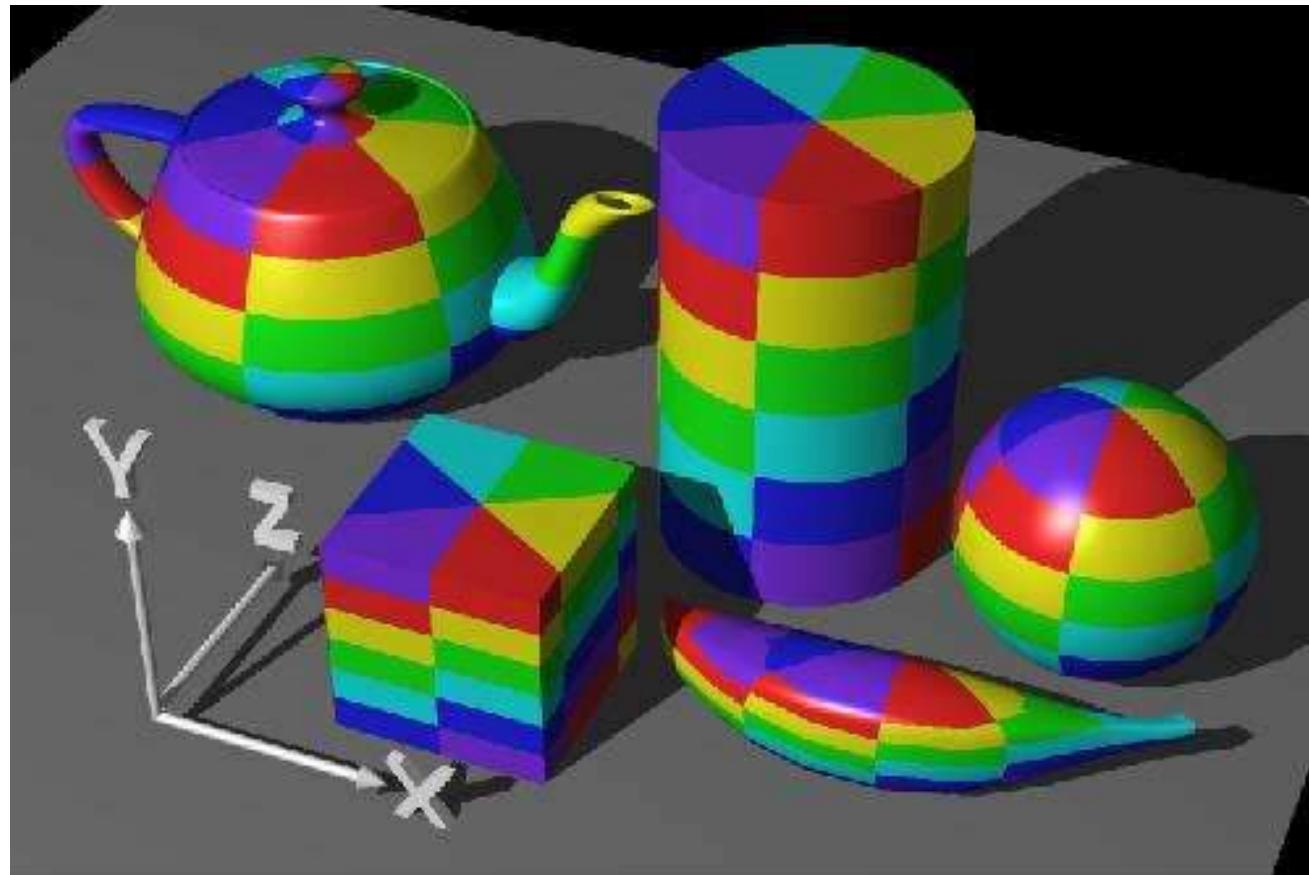


# Cylinder

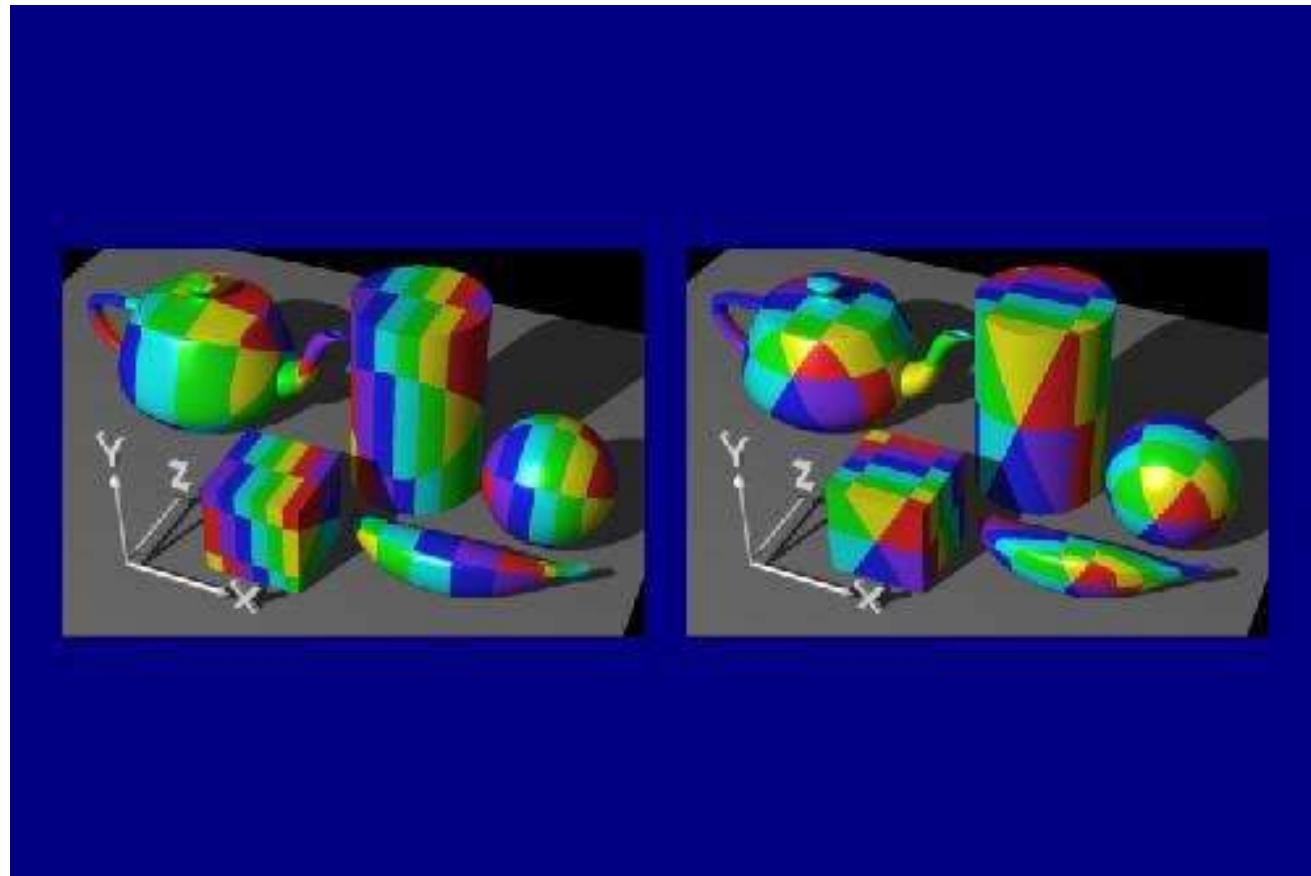
- A bit harder to find T:
- Let position  
 $(x,y,z) = (r \cos(t), r \sin(t), z)$   
for suitable  $r, t, z$   
(you can write any 3D position like this)
- Then  $T(r \cos(t), r \sin(t), z) = (t/2\pi, z)$



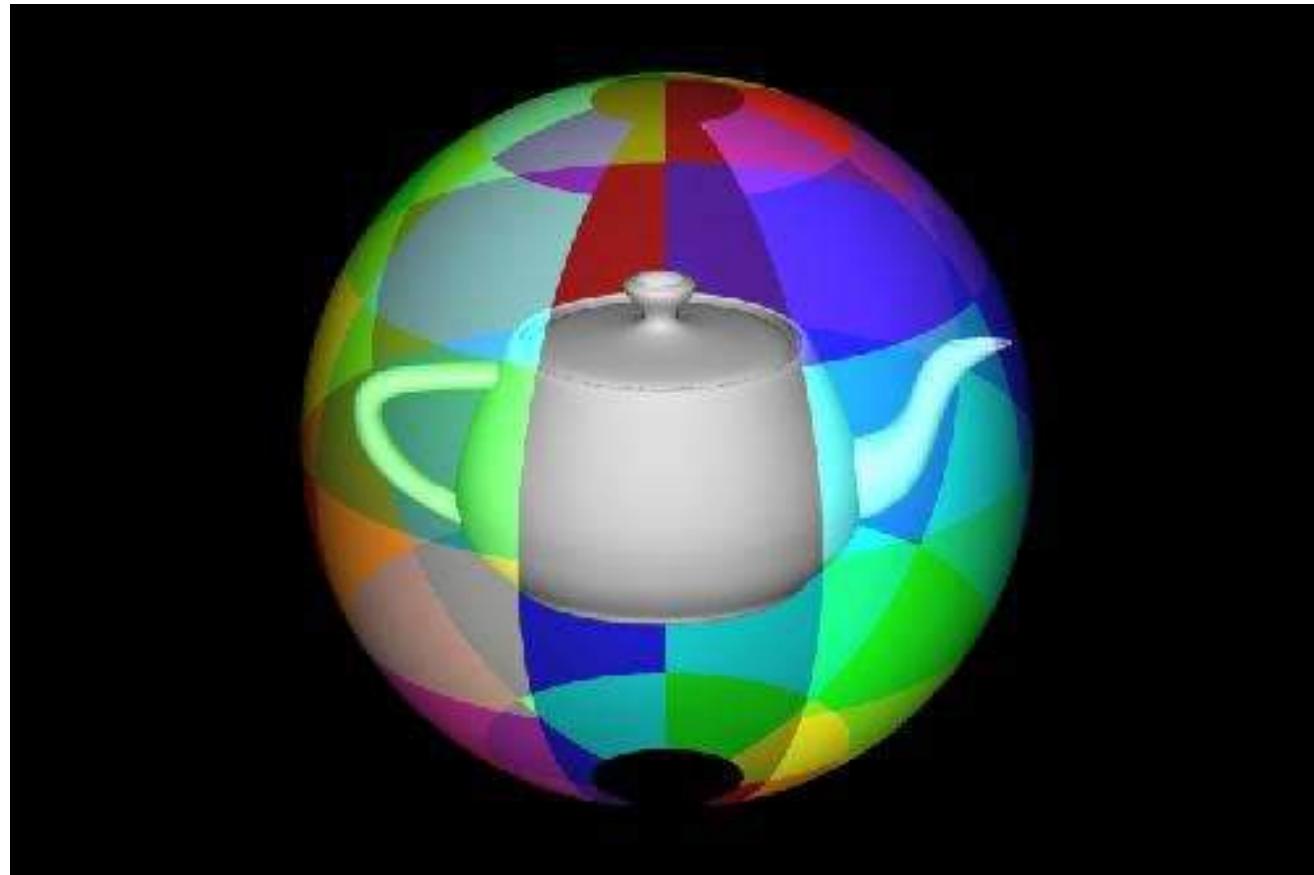
# Cylinder



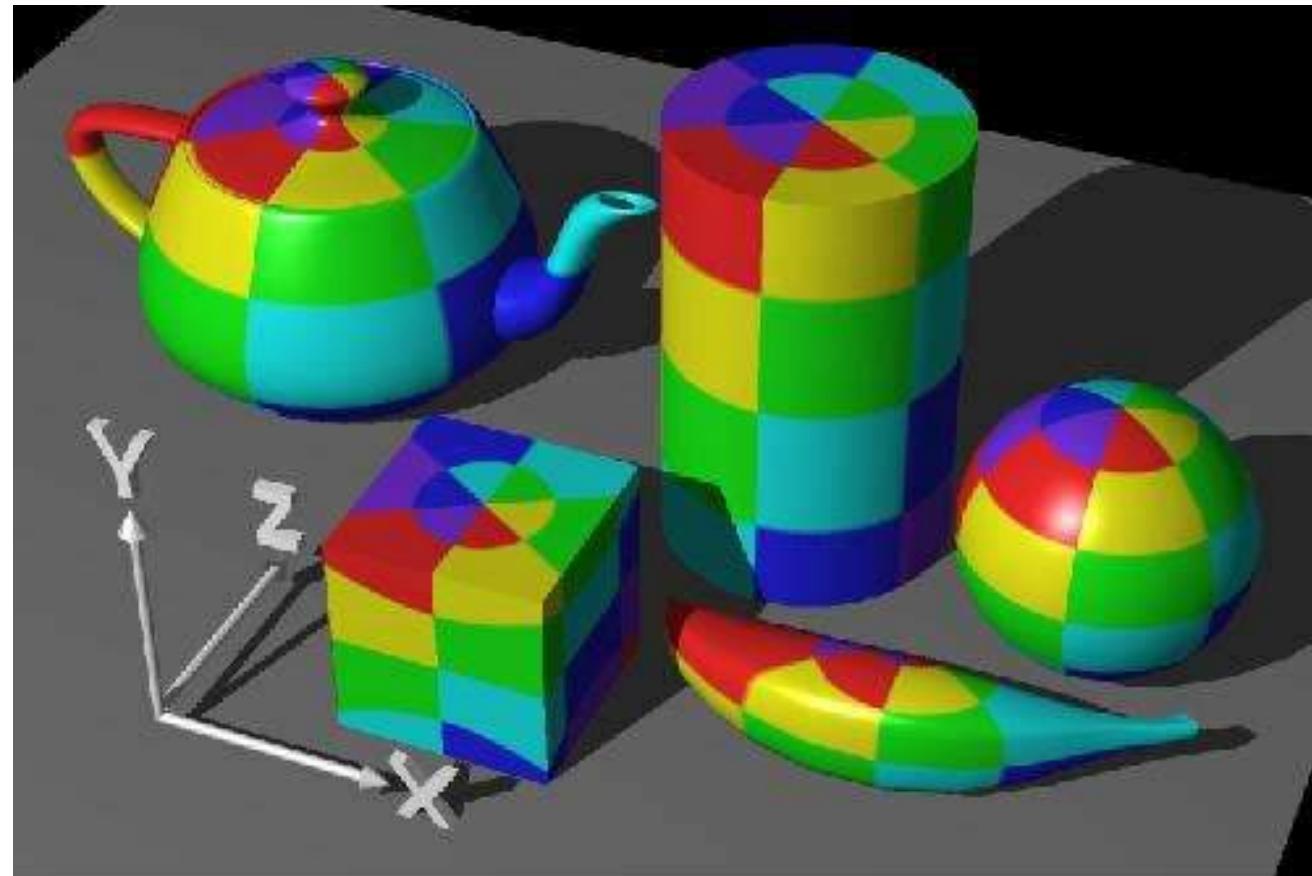
# Cylinder



# Sphere

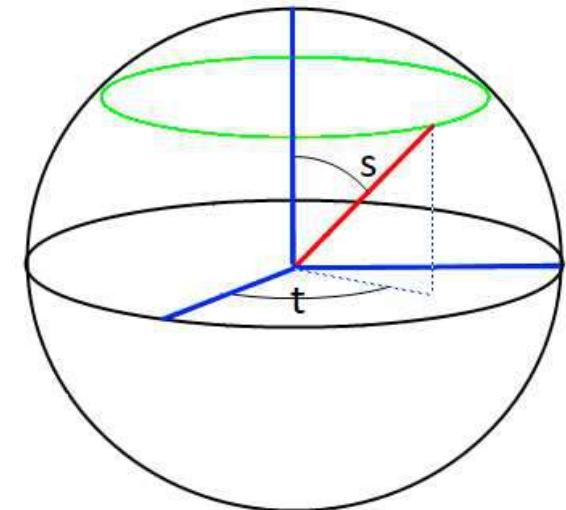


# Sphere



# Sphere

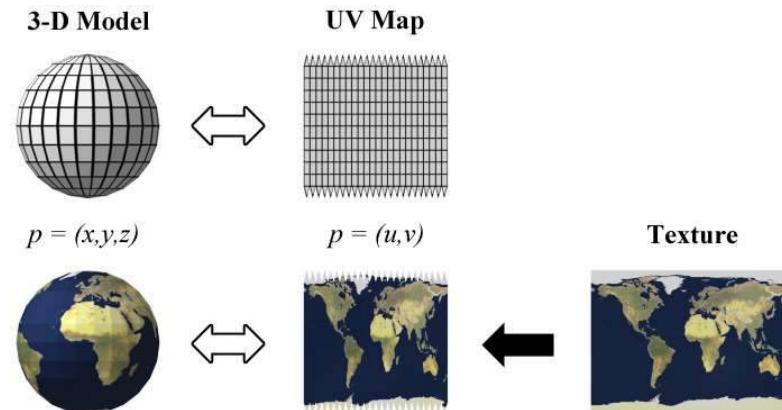
- A bit harder to find T:
- Let position  
 $(x,y,z) = (r \cos(t) \sin(s), r \sin(t) \sin(s), r \cos(s))$   
for suitable  $r$  in  $[0, \infty)$ ,  $t$  in  $[0, 2\pi]$ ,  $s$  in  $[0, \pi]$   
(you can write any 3D position like this,  
if  $r$  is constant, points are on a sphere)
- Then  $T((r \cos(t) \sin(s), r \sin(t) \sin(s), r \cos(s)))$   
 $= (t/2\pi, s/\pi)$



# Sphere is useful for Environment Mapping

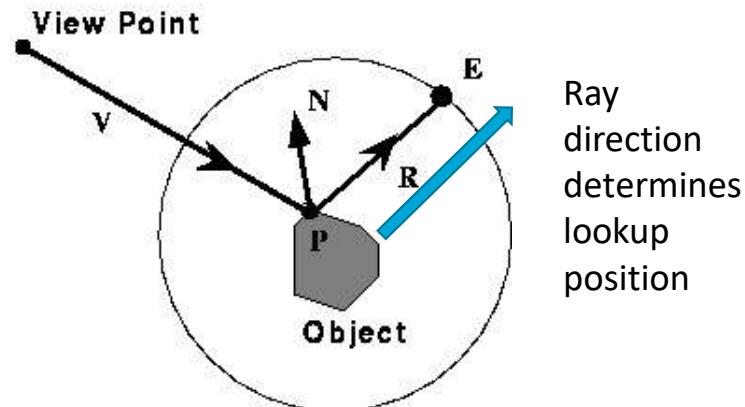
- Textures can encode an environment  
An environment map (approximation of scene)

Spherical mapping



# Environment Mapping

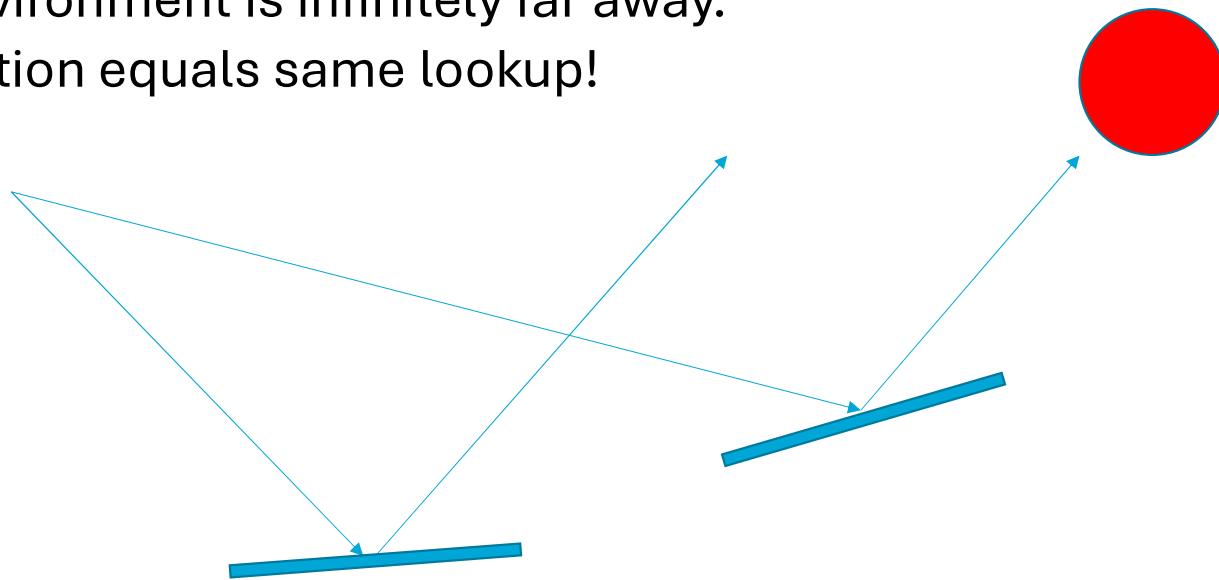
- Textures can encode an environment  
An environment map (approximation of scene)  
is useful for, e.g., reflections (texcoords = ray)



- This *environment mapping* is an approximation!

## Environment Mapping

- Typical approximation:  
Assume environment is infinitely far away.  
Same direction equals same lookup!



- Red ball reflection seen by both or none.

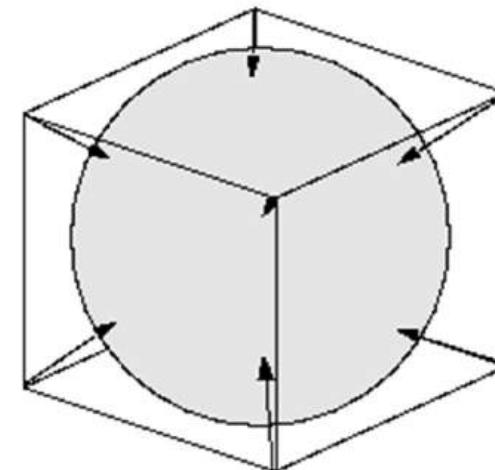
# Environment Mapping



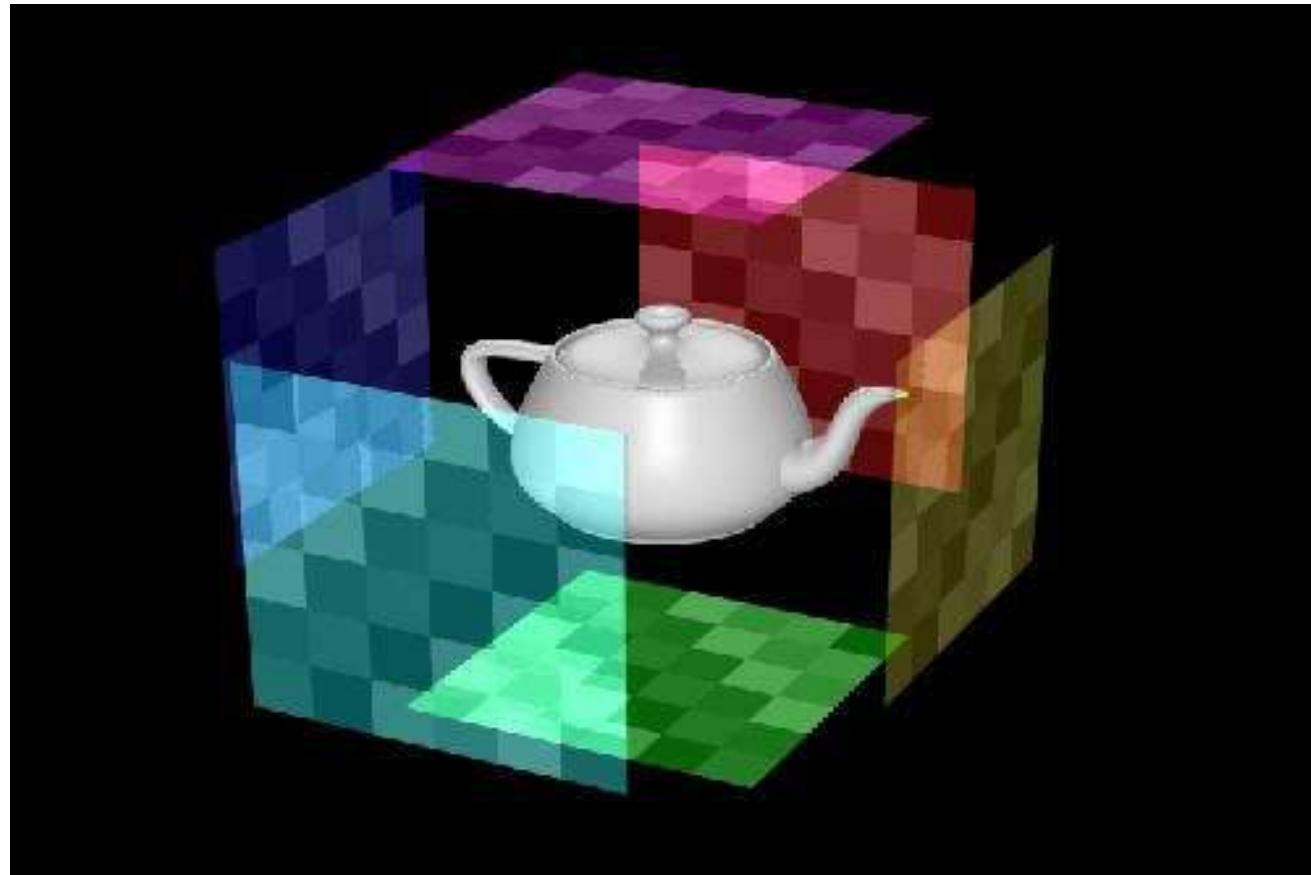
# Environment Mapping

- Alternatively, a sphere can be mapped to a cube
- Less distortions if images are used for the cube faces

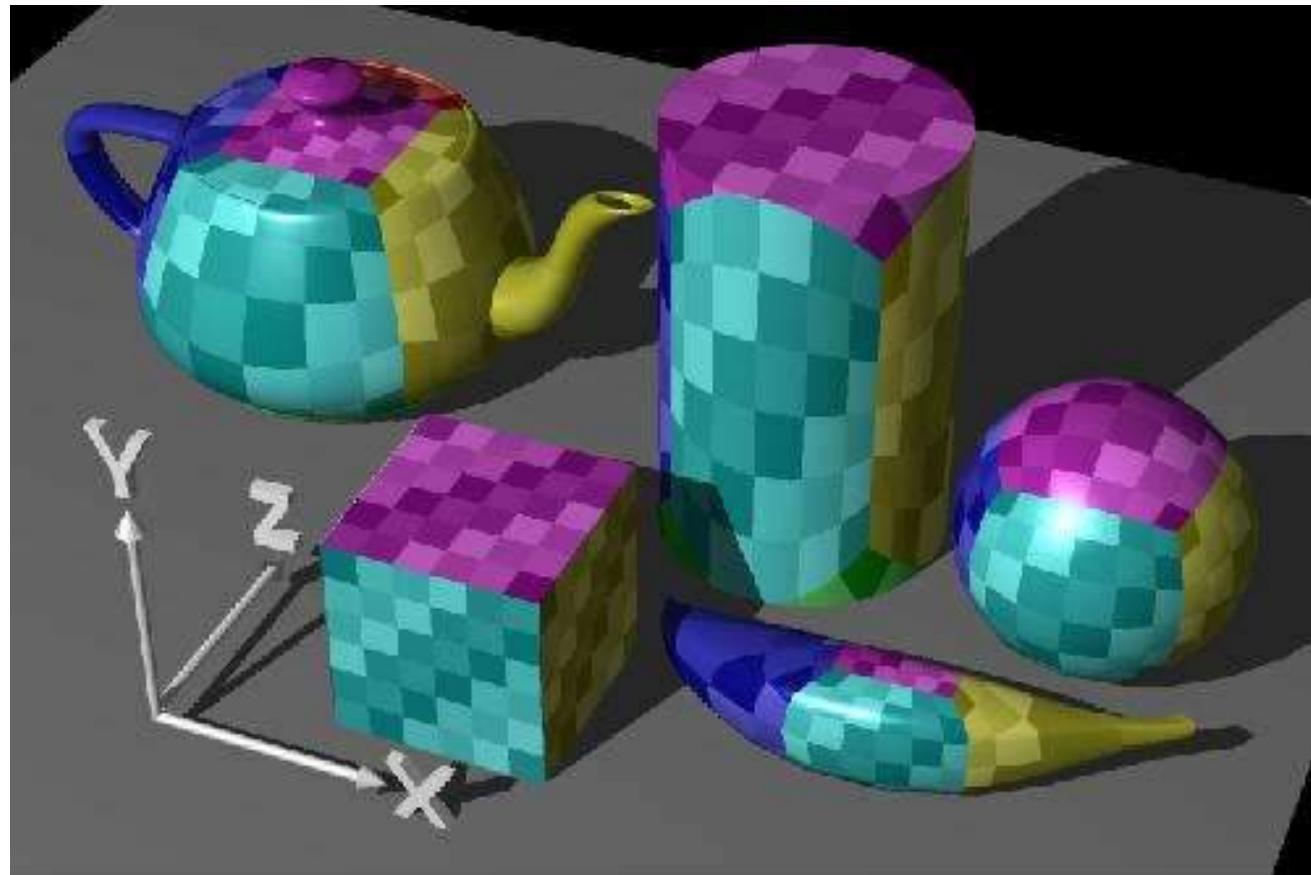
Cube mapping



## Cube Map



## Cube Map



# Questions?



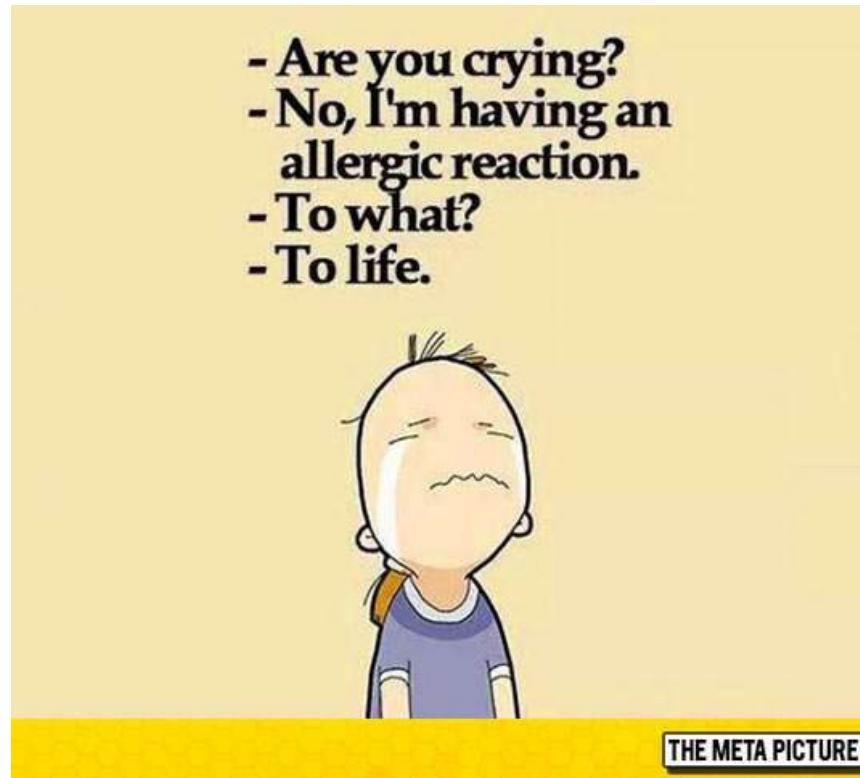
# Textures

- Image content mapped on surfaces
- Increase detail level without geometric cost
- Many applications for color textures



## Texture Issues?

- What are limitations of textures?



## Texture issues: Aliasing

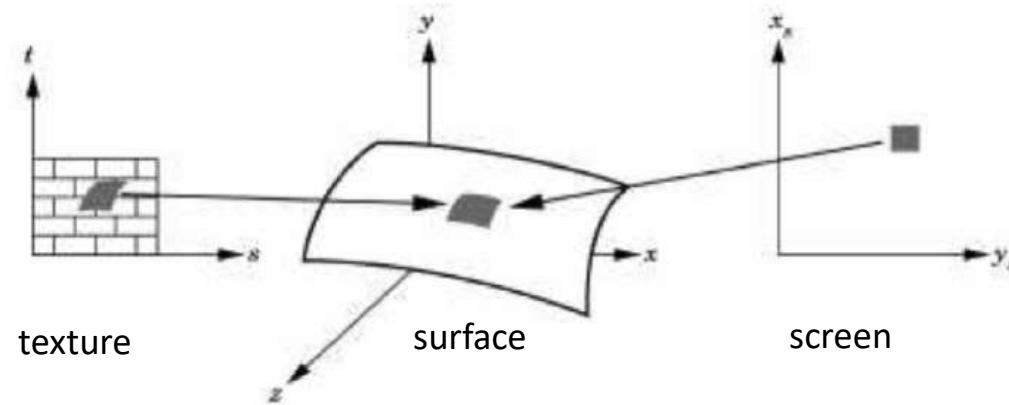
- Not always completely beautiful...



# Aliasing

- From screen to texture:

Not a one-to-one pixel mapping!



## Texture problems: Aliasing

- Two types of aliasing!
- Oversampling
- Undersampling

# 1. Oversampling

- Due to limited texture resolution

Minecraft

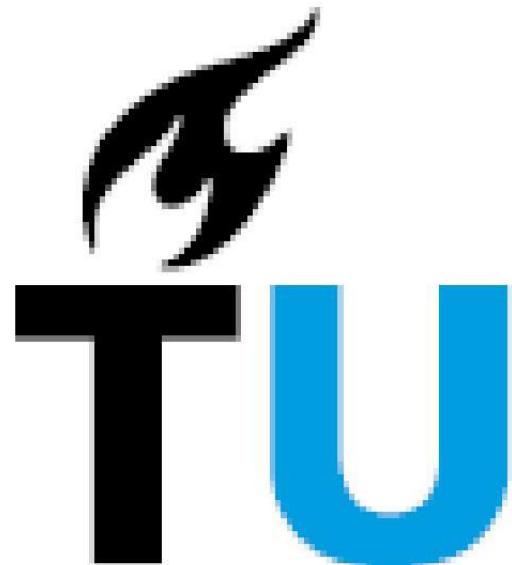


Resident Evil 4

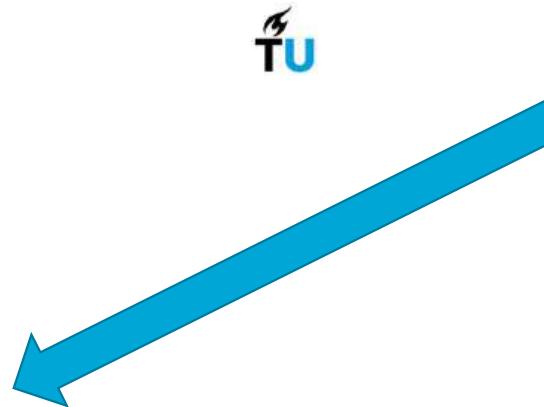


## 1. Oversampling

- Pixel smaller than texel



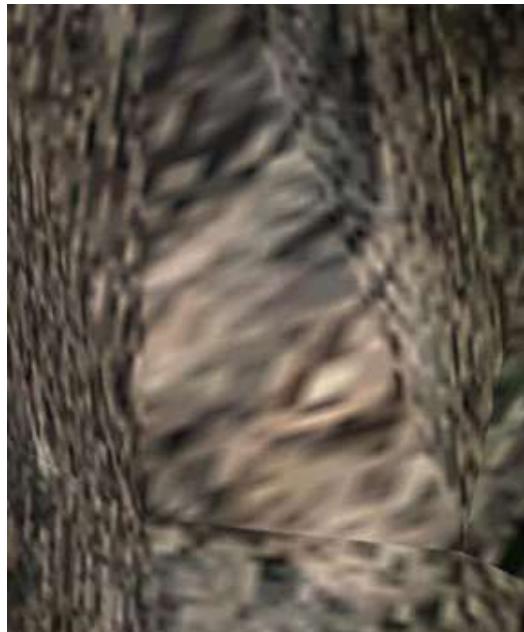
Nearest Neighbor



## 1. Oversampling

- In practice:

Often less blocky but washed out (most applications use texel interpolation)

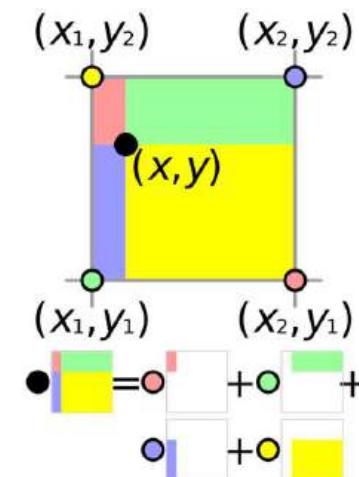
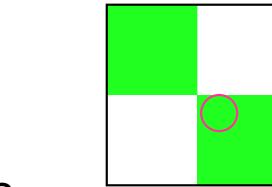
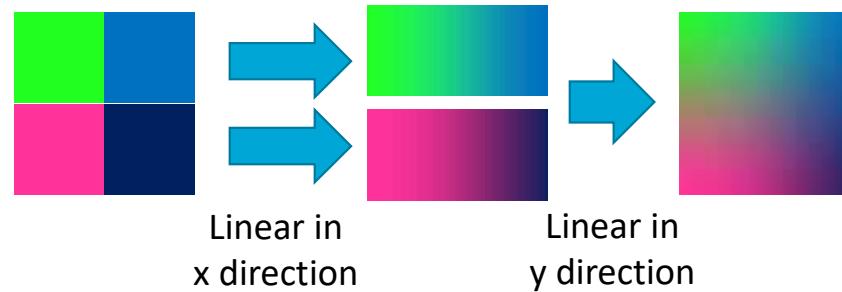


# 1. Oversampling

- Bilinear interpolation
  - Linear interpolation:  $(1-\alpha) * \text{col1} + \alpha * \text{col2}$

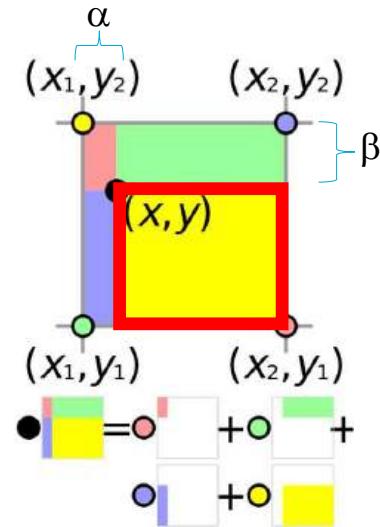


- Bilinear interpolation:



# 1. Oversampling

- Why does this work?



$$\begin{aligned}
 & ((1-\alpha) * \text{yellow} + \alpha * \text{blue}) * (1-\beta) + \\
 & ((1-\alpha) * \text{green} + \alpha * \text{red}) * \beta
 \end{aligned}$$



## Recap Video – Bilinear Interpolation

CSE2215 Computer Graphics

# Bilinear Interpolation

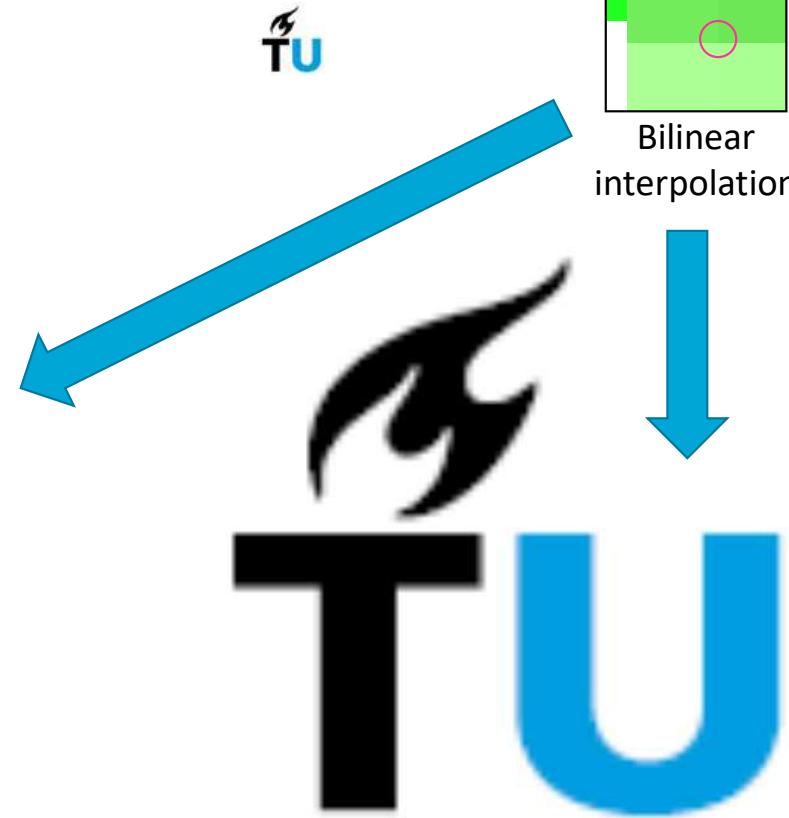
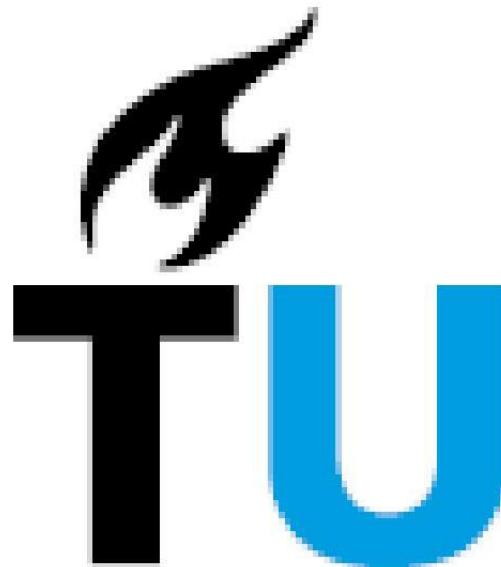
Tim Huisman

77



## 1. Oversampling

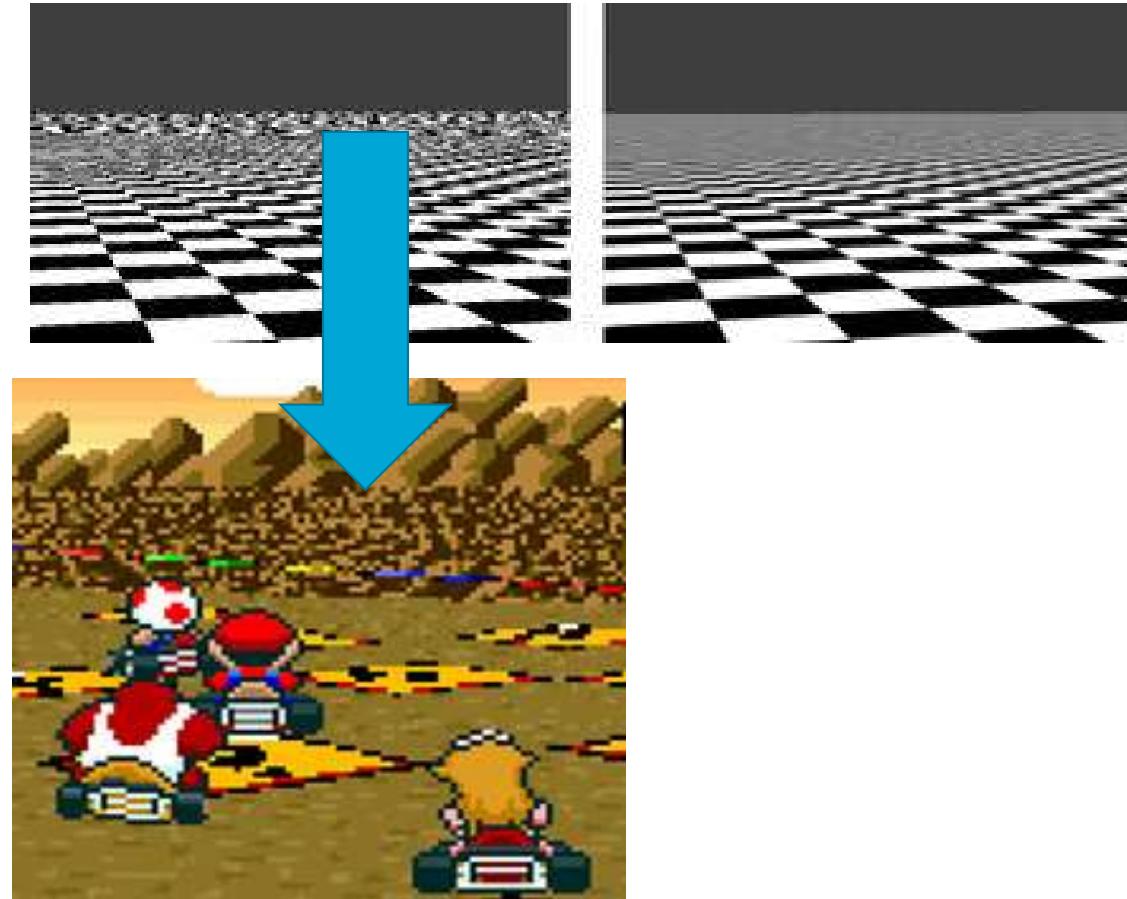
- Pixel smaller than texel



Nearest Neighbor

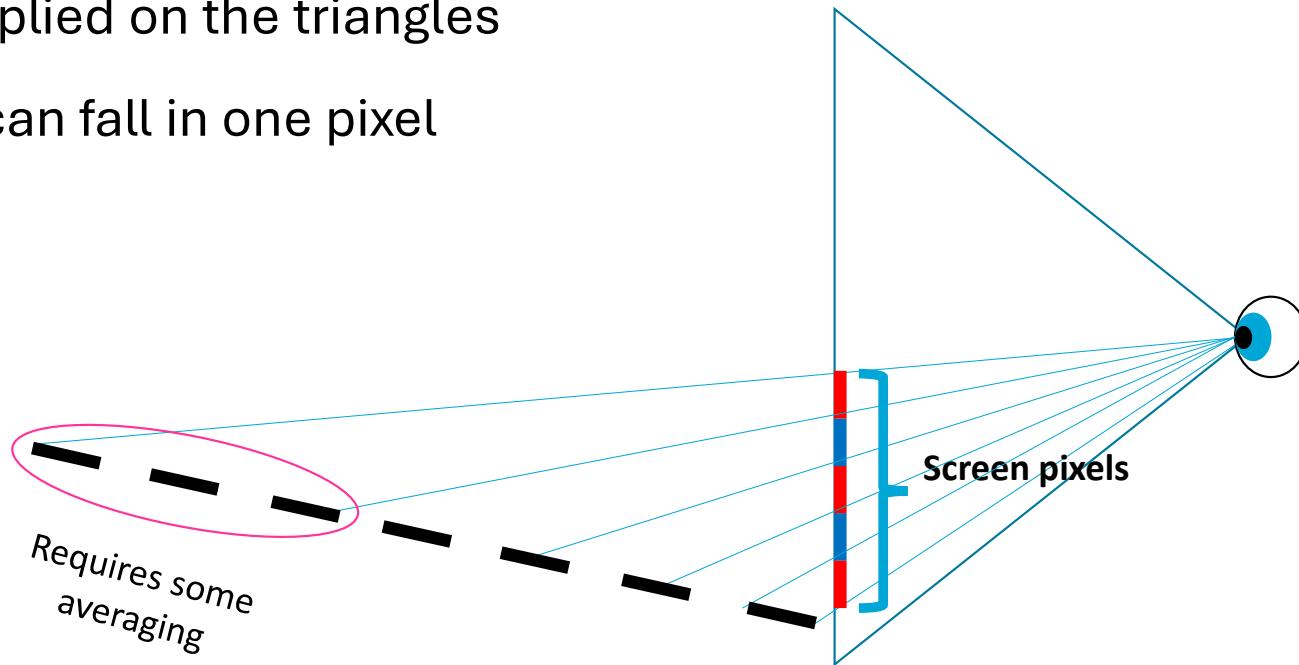
Bilinear Interpolation

## 2. Undersampling



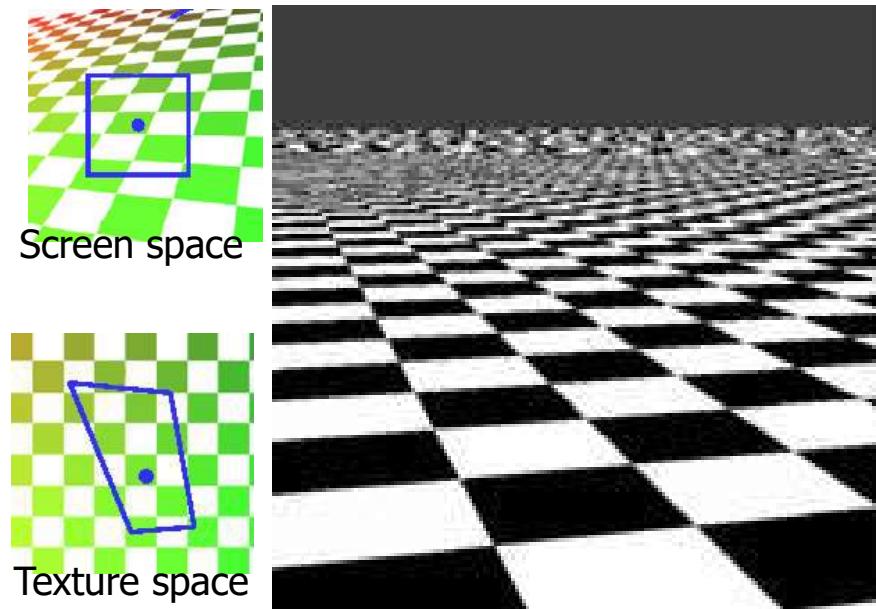
## 2. Undersampling

- Textures are applied on the triangles
- Several texels can fall in one pixel



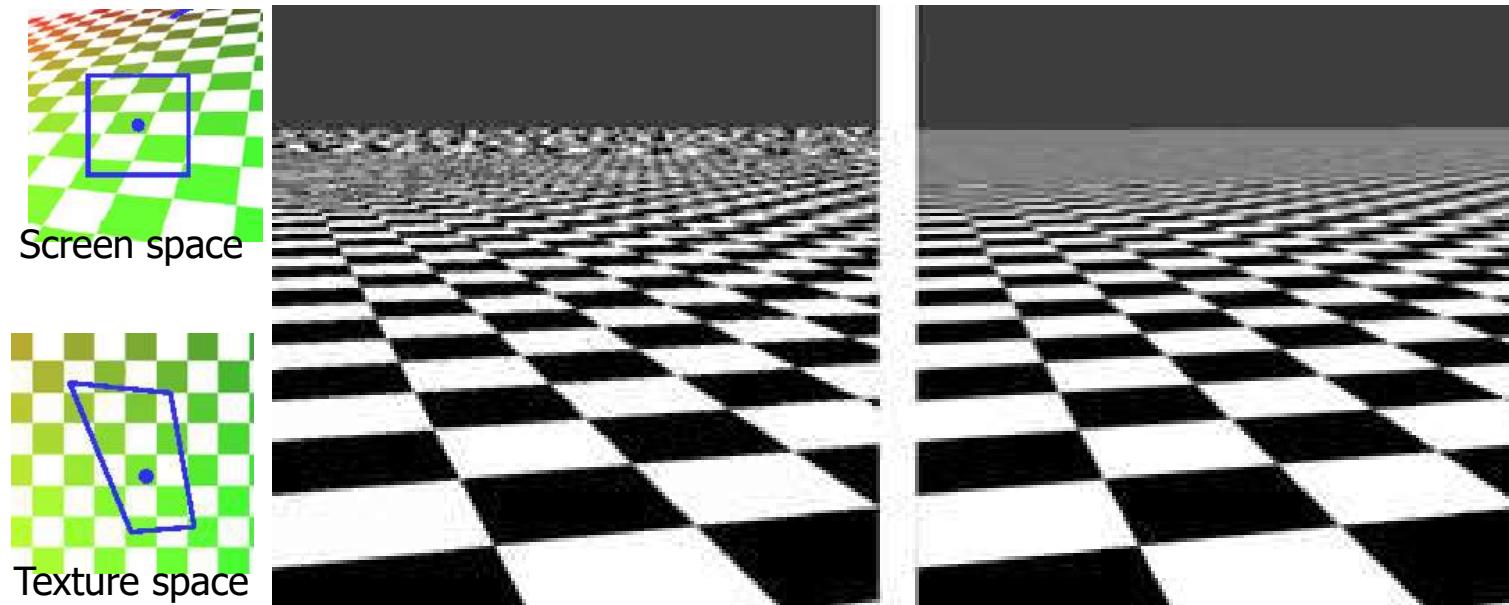
## 2. Undersampling

- One screen pixel does not necessarily correspond to one texel



## 2. Undersampling

- One screen pixel does not necessarily correspond to one texel



## 2. Undersampling

- Naïve solution:  
Render at high resolution  
then average the result



- Very costly....

## MipMapping: Approximate Filtering

- *Mip = multum in parvo*

- meaning "much in little"



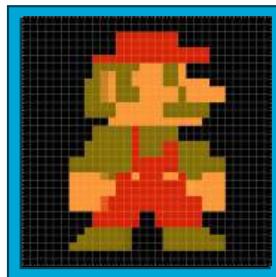
# MipMapping: Approximate Filtering

- Idea: Compute a filtered texture beforehand

The figure displays a series of six images illustrating a multi-scale feature map of Mario's head. The images are arranged horizontally, representing different levels of resolution and fusion:

- 32x32 level 0: The original pixelated image of Mario's head.
- 16x16 level 1: A coarser representation where Mario's features are partially merged.
- 8x8 level 2: Further merging of features, with distinct colors for different parts of the head.
- 4x4 level 3: Features are becoming more integrated across the grid.
- 2x2 level 4: Most features are now fully fused into a single color.
- 1x1 level 5: The final, fully fused feature map where all details are lost.

Imagine  
A 32x32  
screen

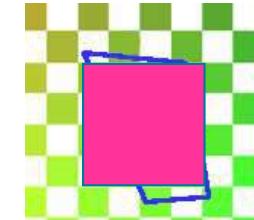
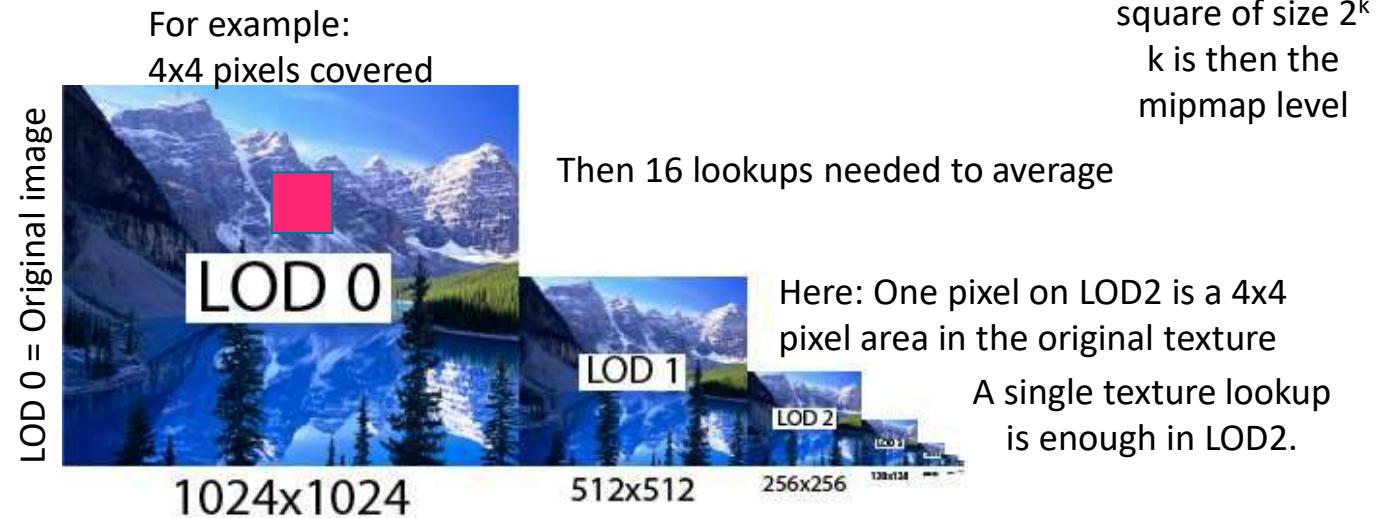


Choose the correct level depending on pixel-to-texel matching

e.g., 4 texels per pixel

# MipMapping: Approximate Filtering

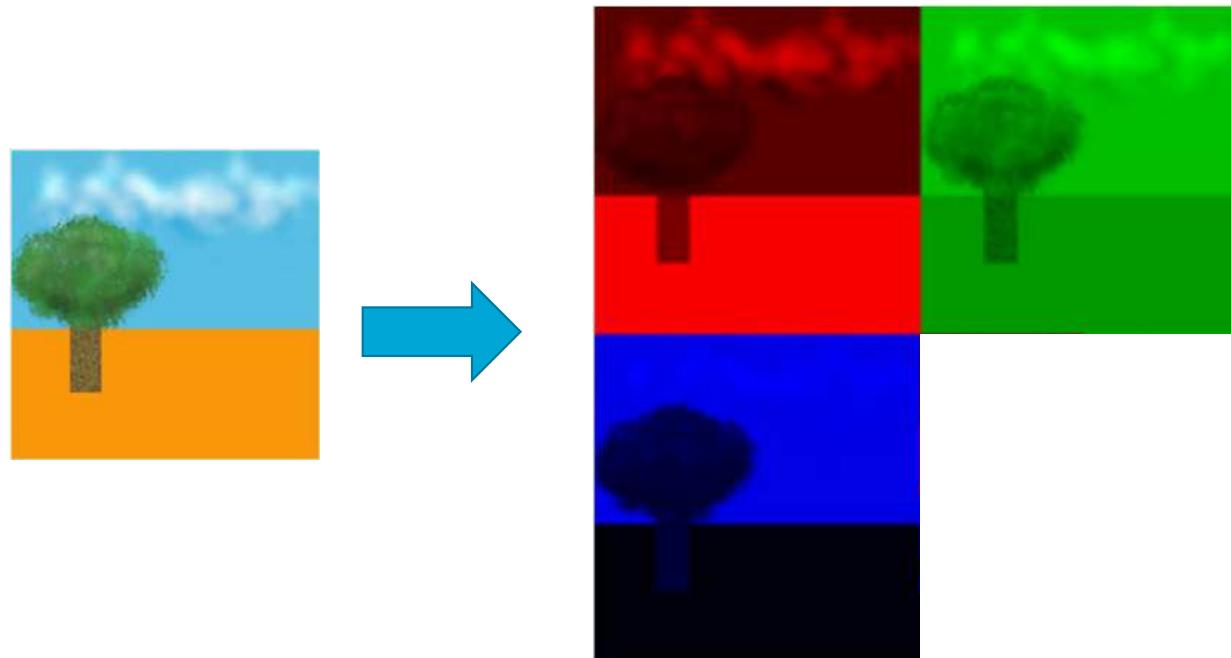
- Precompute *Mipmap pyramid*:
  - *Hierarchical texture*
  - *Reduce resolution by 2x2 on each level*



Texture space  
Per pixel,  
approximate  
region with a  
square of size  $2^k$   
 $k$  is then the  
mipmap level

## MipMapping: Memory Requirements

- Visual Proof that it is <1/3 extra memory



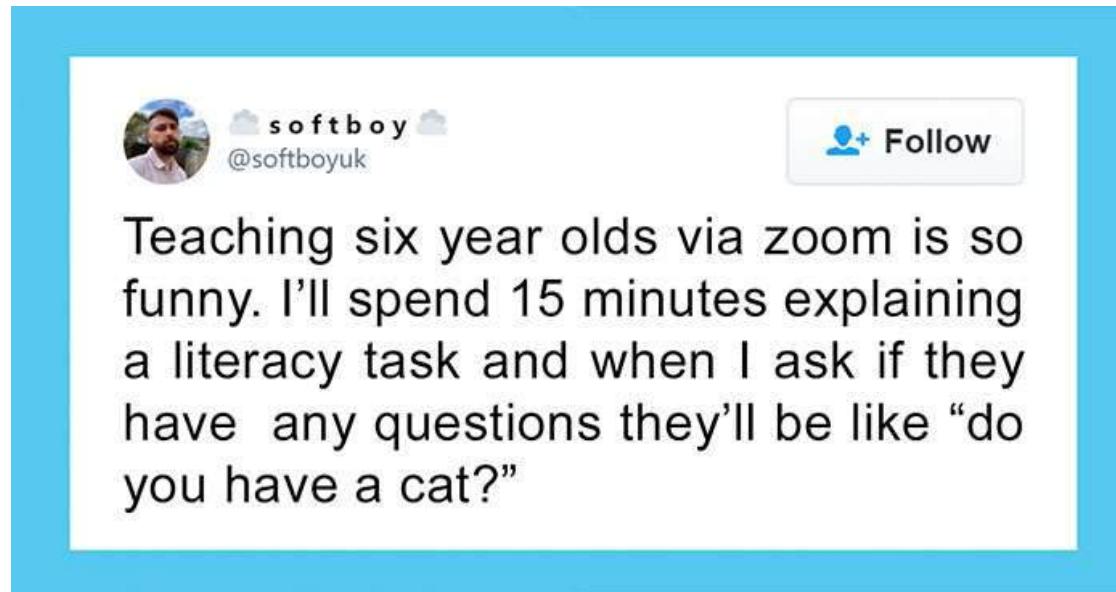
## MipMapping: Memory Requirements

- We assume memory of the original texture is 1
- Texture resolution  $2^n \times 2^n$
- Then the memory cost for all mipmap levels is:

$$\begin{aligned} & \sum_{i=0}^n \left(\frac{1}{4}\right)^i \\ &= \frac{1 - 1/4^{n+1}}{1 - 1/4} \\ &= \frac{4}{3}(1 - 1/4^{n+1}) \end{aligned}$$

In the limit the cost is 1/3 higher.

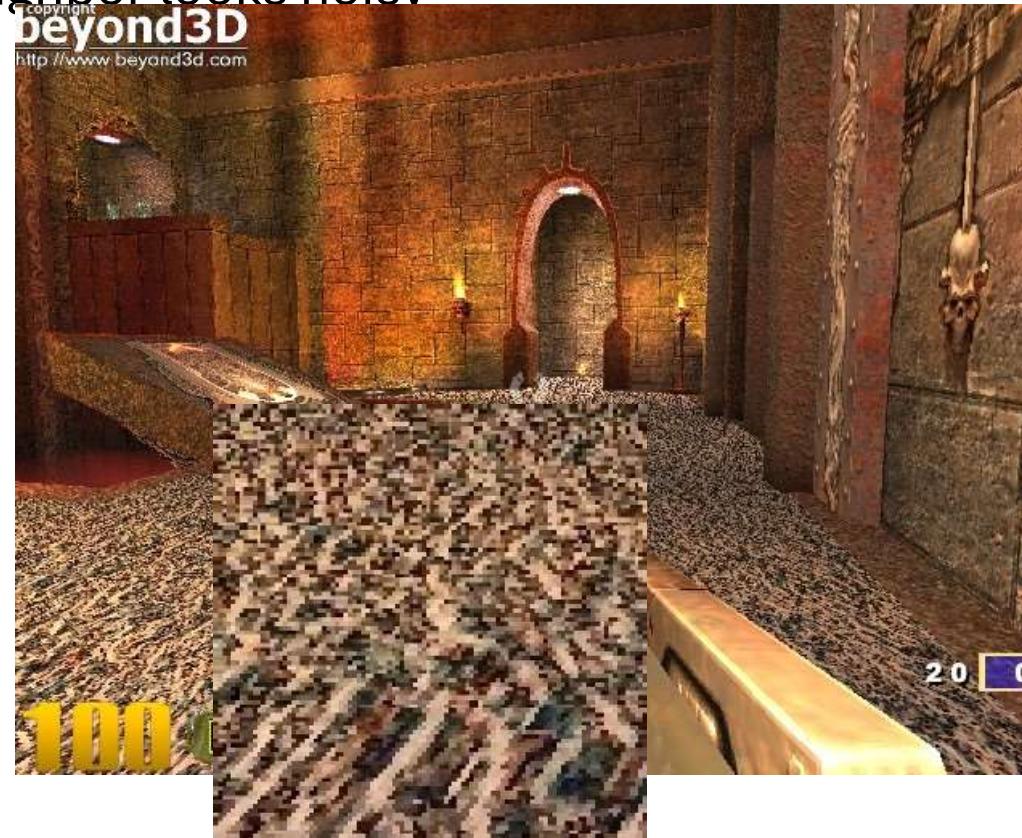
## Questions?



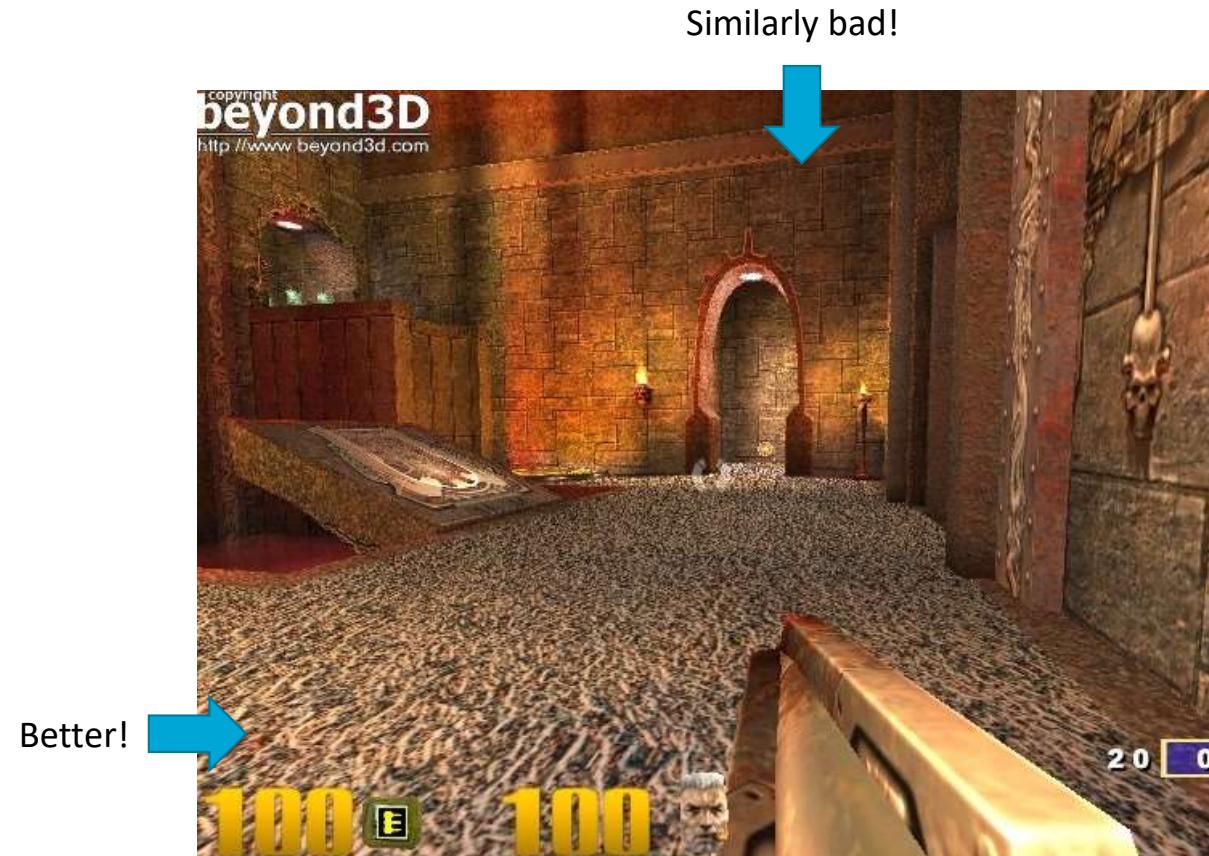
**Let's look at an example...**

## MipMapping: OFF

- Just Nearest Neighbor looks noisy



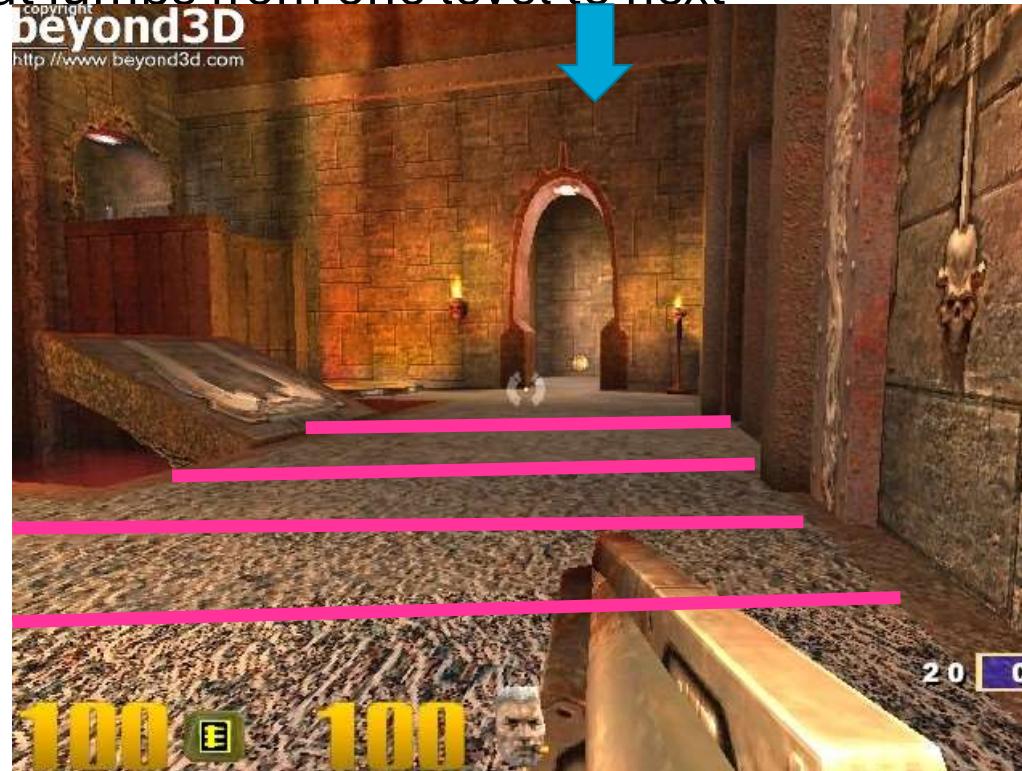
## MipMapping: Off + Linear Filtering



## MipMapping On (use nearest level)

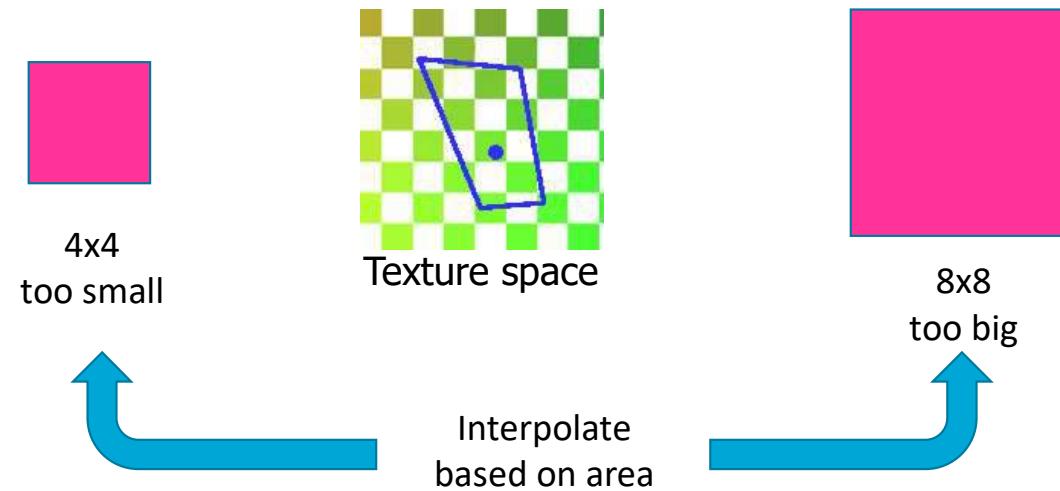
- Discontinuities at jumps from one level to next

Bad: Different levels visible!



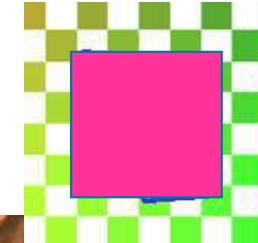
# MipMapping

- Discontinuities come from changing region size



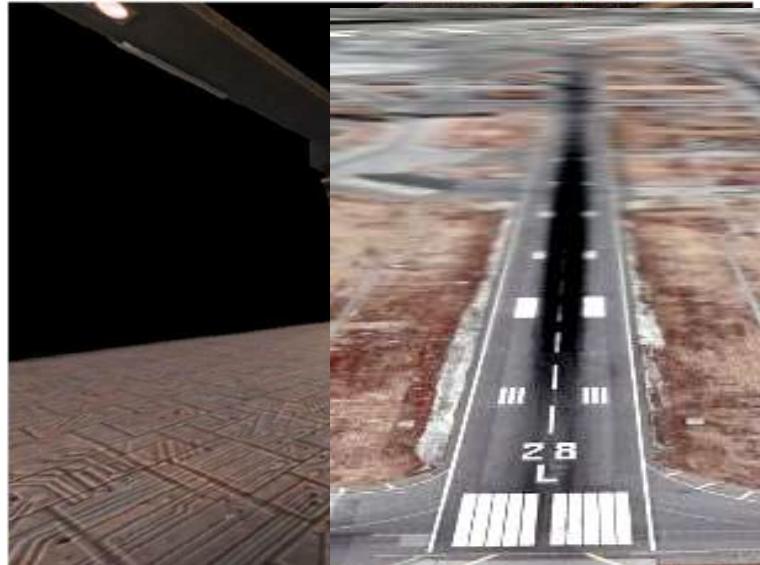
## MipMapping: TriLinear Filtering

- Blend (mix) between different levels

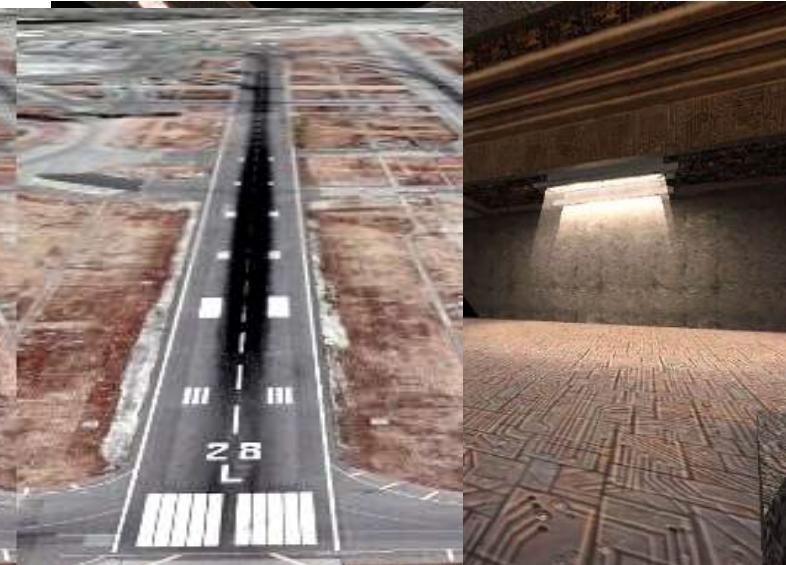


## Anisotropic: Comparison

MipMapping

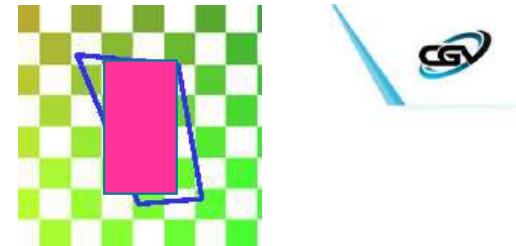


Anisotropic



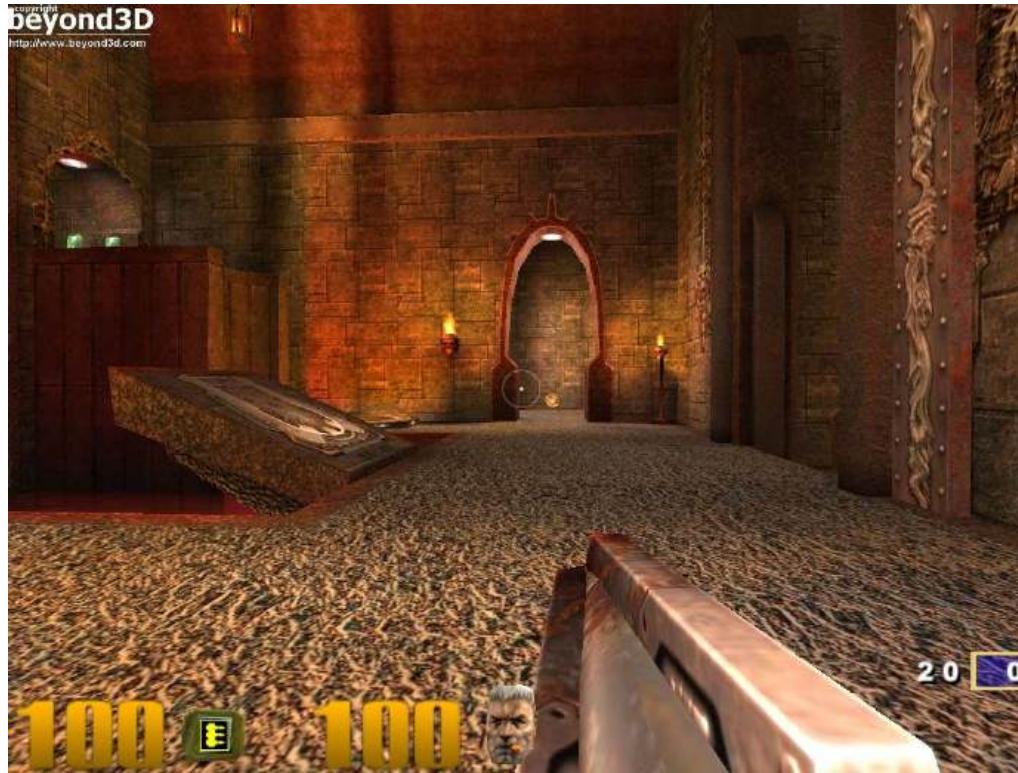
- Better approximates  
pixel projection in texture

# Anisotropic Filtering



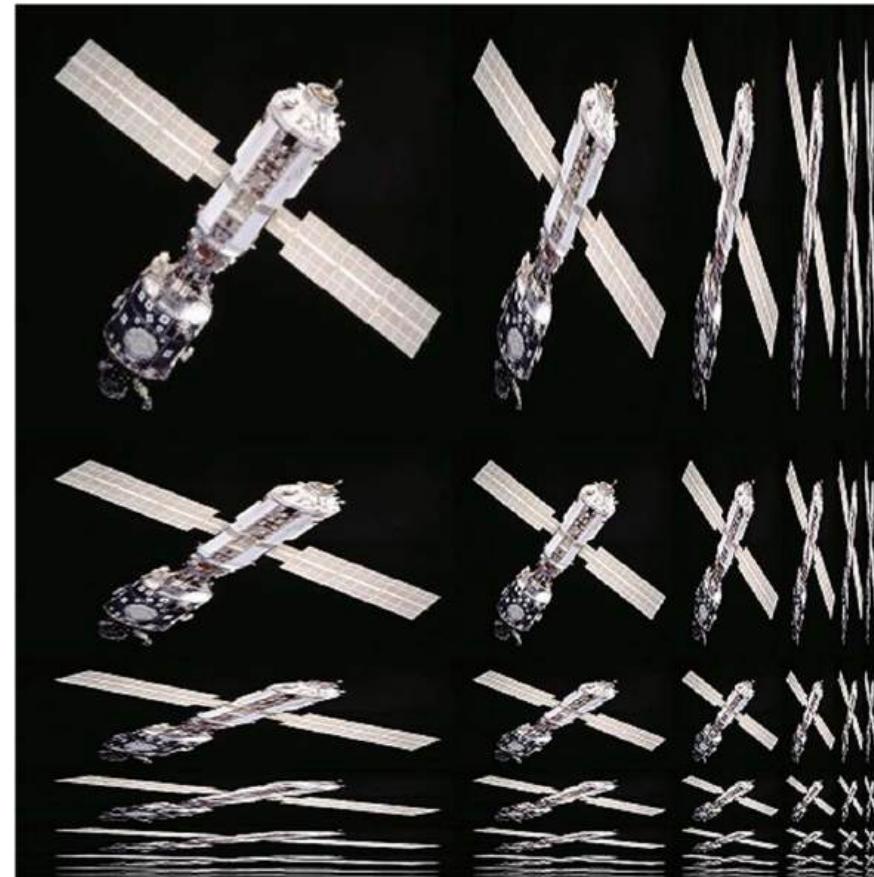
- Better approximate real region

Beautiful!



## Anisotropic MipMaps

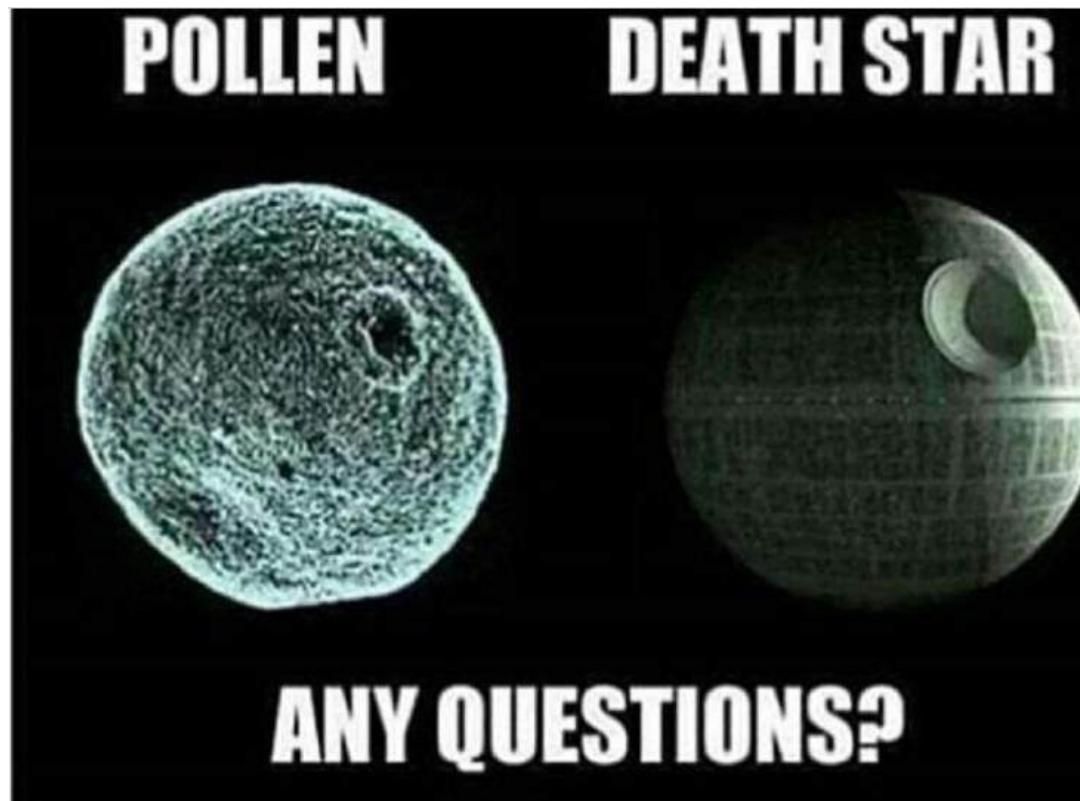
- Storage:
  - Power of two scaling on each axis
  - Fits into 4 times the original memory



# MipMaps

- Absolute standard!
- Memory cost relatively low
- Computational cost very low

## Questions



## **Dissection of a professional example:**

**You can store more than color in a texture!**

-NOT EXAM RELEVANT-

## Textures as Lookup Table

- Approximate complex computations via texture lookups

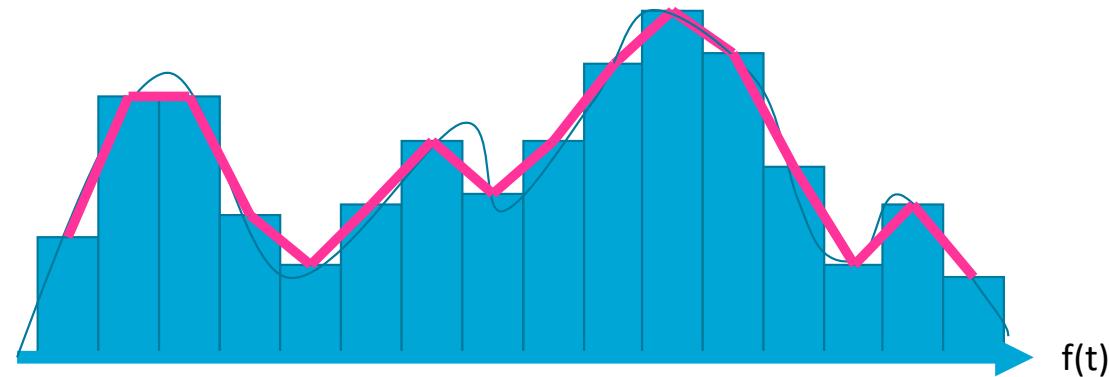


Replace  
computation  
by a lookup  
in a texture



## Textures as Lookup Table

- Activating linear texture interpolation leads to a piecewise linear approximation



- This type of representation is very common in high-performance computing and simulations

## Textures to the top!

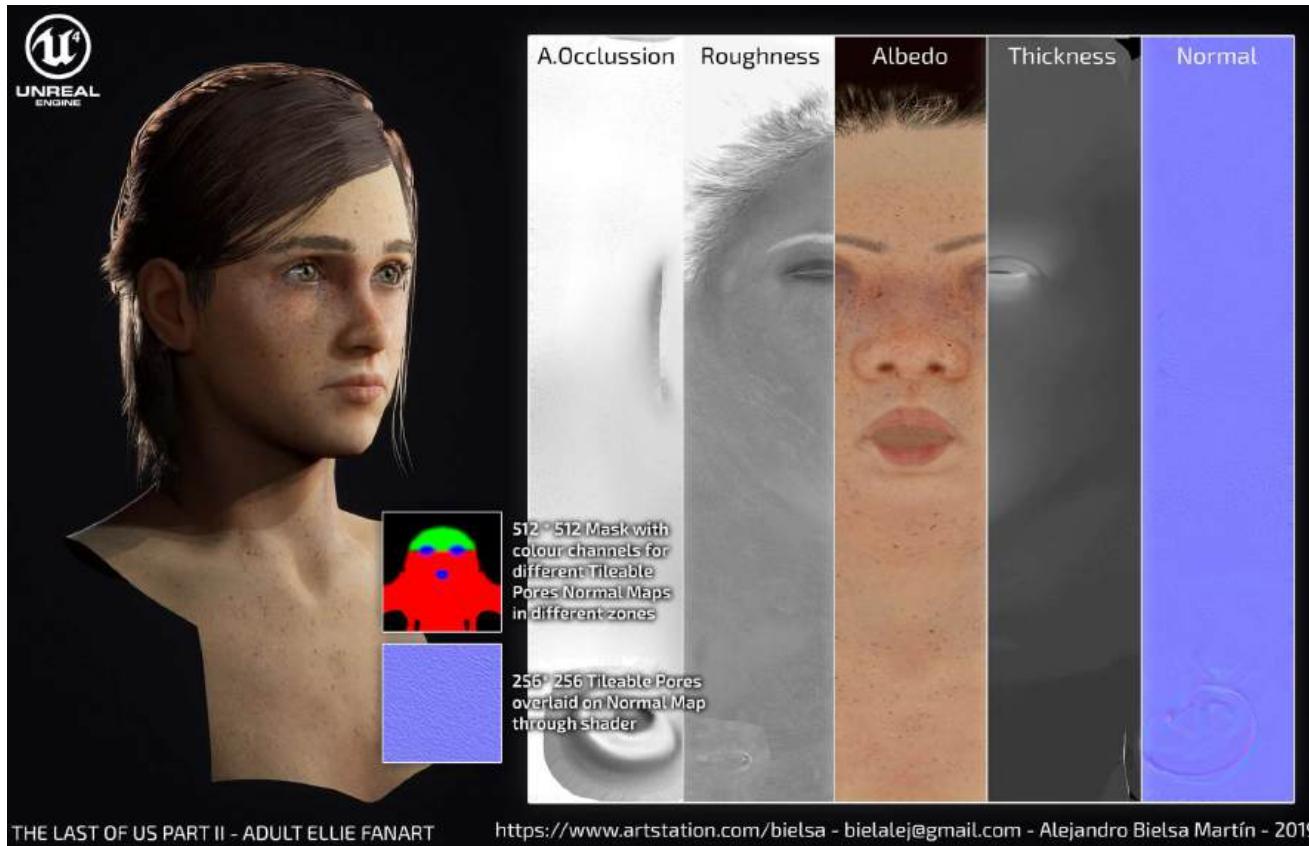


Last of us

## Last of Us on PS1



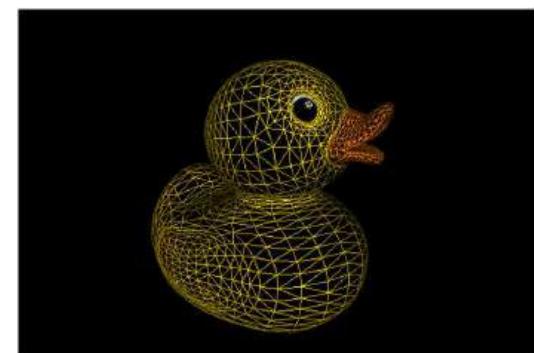
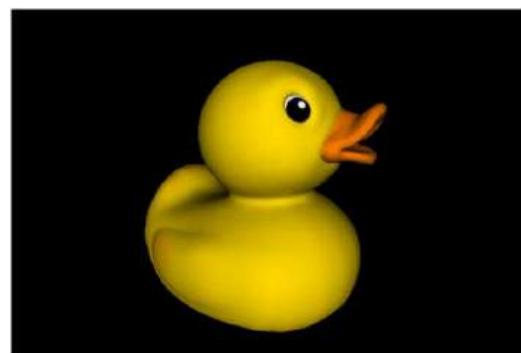
## Last of Us – Professional Example



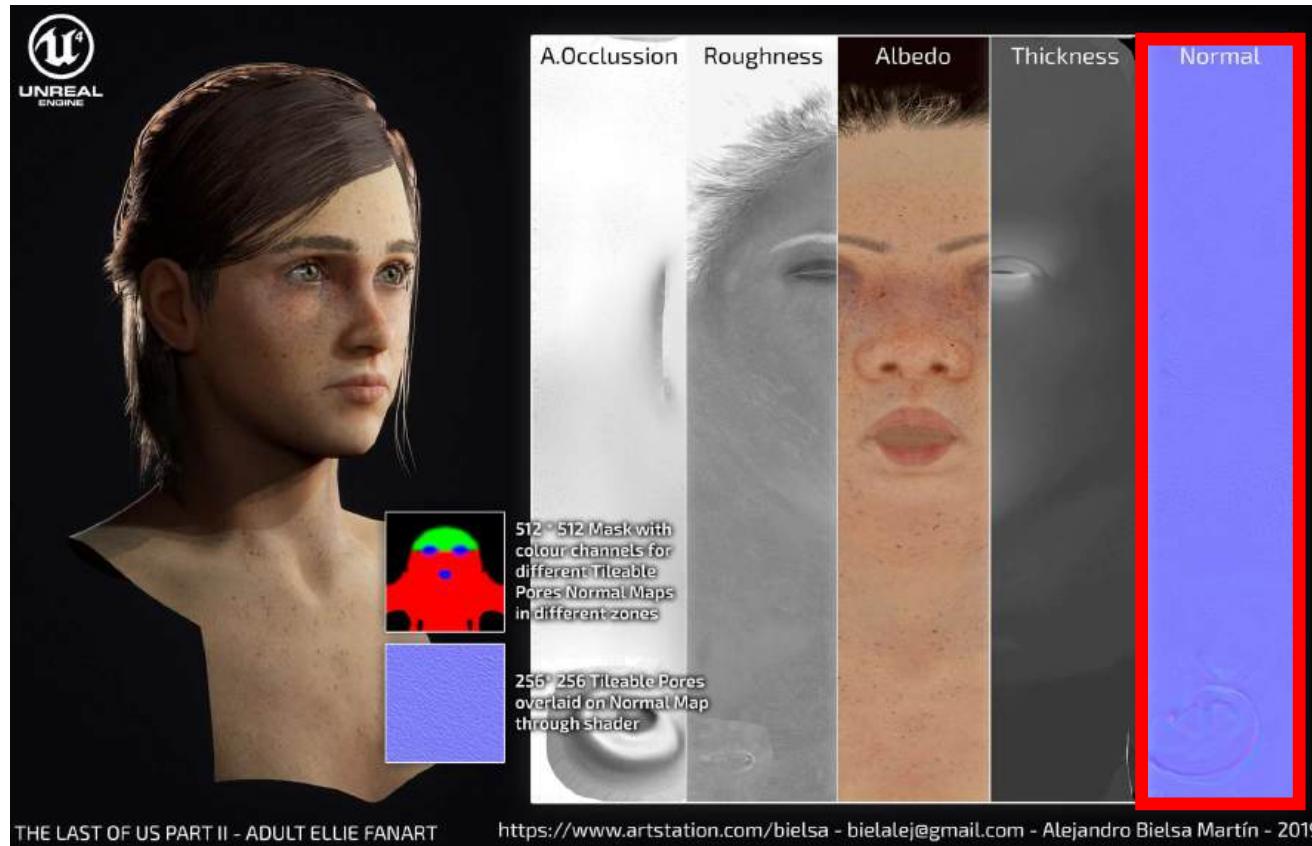


## Textures: Materials

- Can be used to provide, e.g.,
  - parameters for material models  
(ambient, diffuse...)

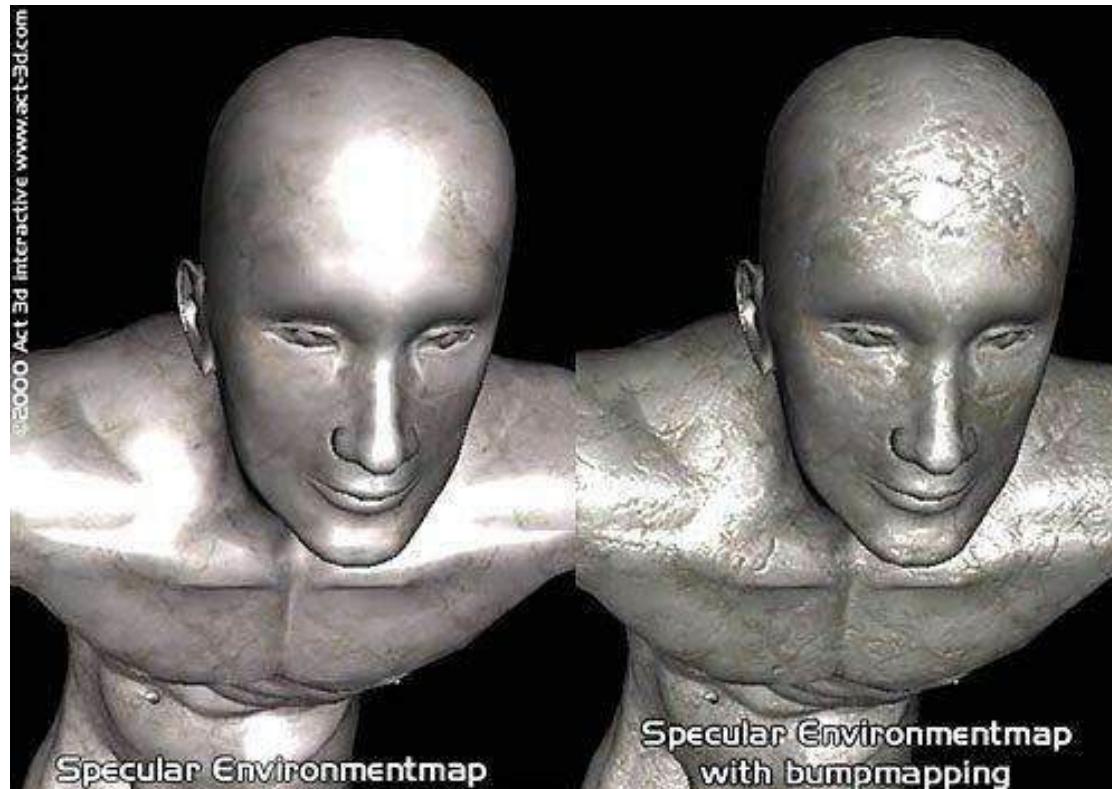


## Last of Us – Professional Example



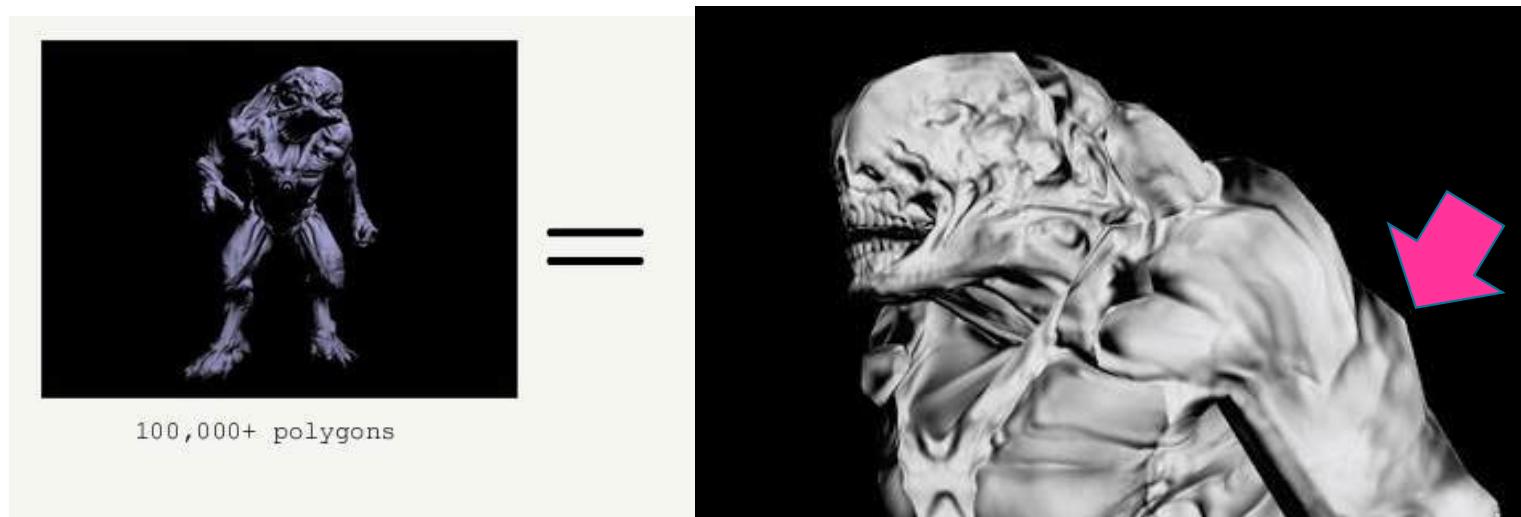
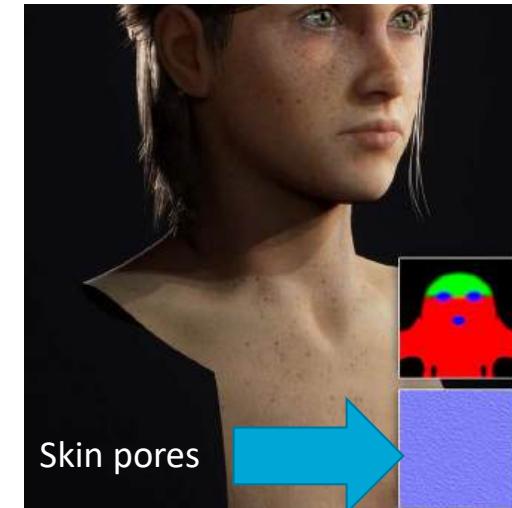
## Textures: Normal Mapping

- Encode Normals (*bump/normal mapping*)



# Textures

- Normal Mapping



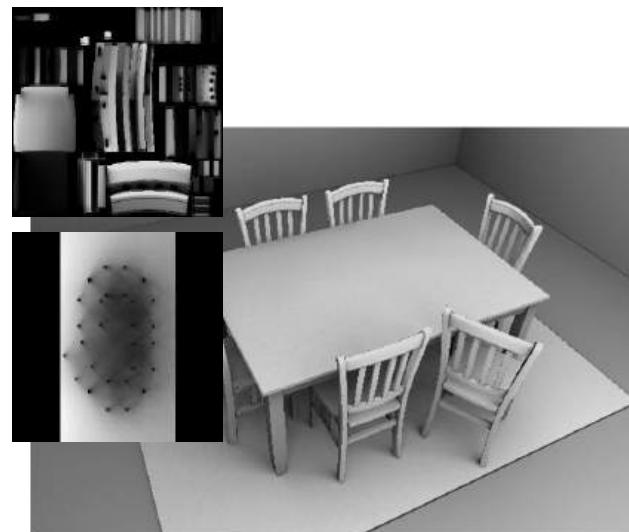
100,000+ polygons

# Textures: Light Maps

- Precomputed illumination (*light map*)



Diffuse mapped on scene



Lightmap mapped on scene



Combined

# Textures: Light Maps

- Precomputed illumination (*light map*)

Quake – id Software



Materials

\*



Light Map

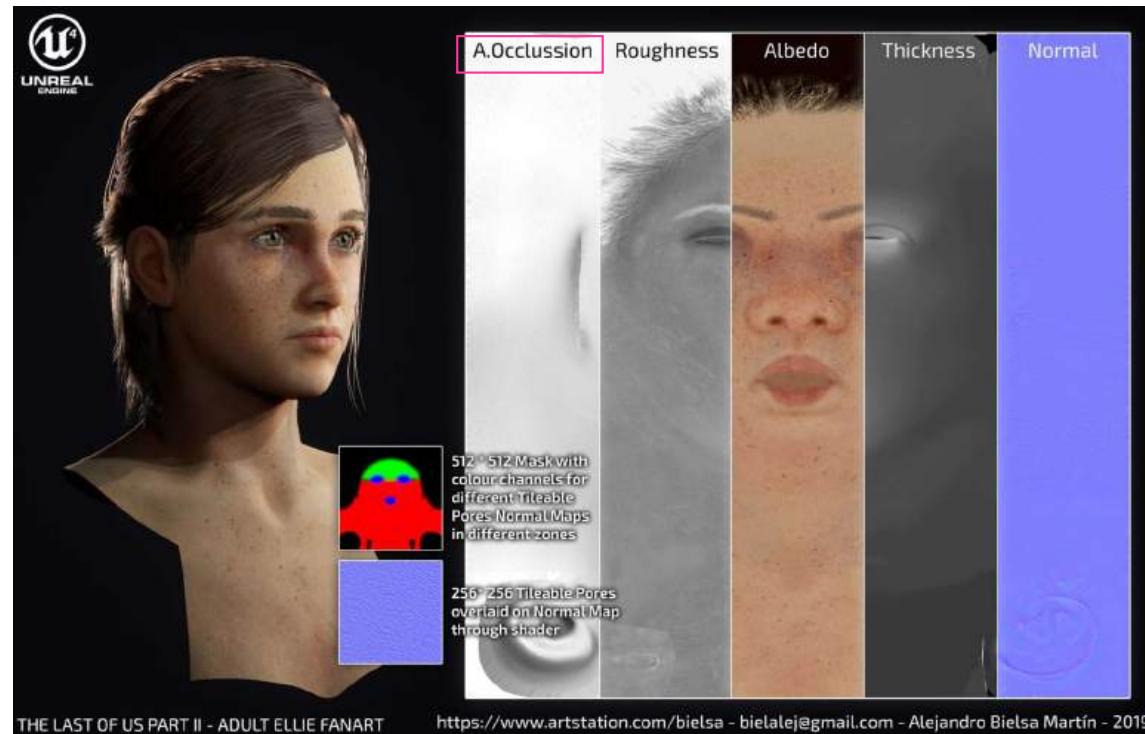
=



Final result

# Textures: Light Maps

- Precomputed illumination (*light map*)



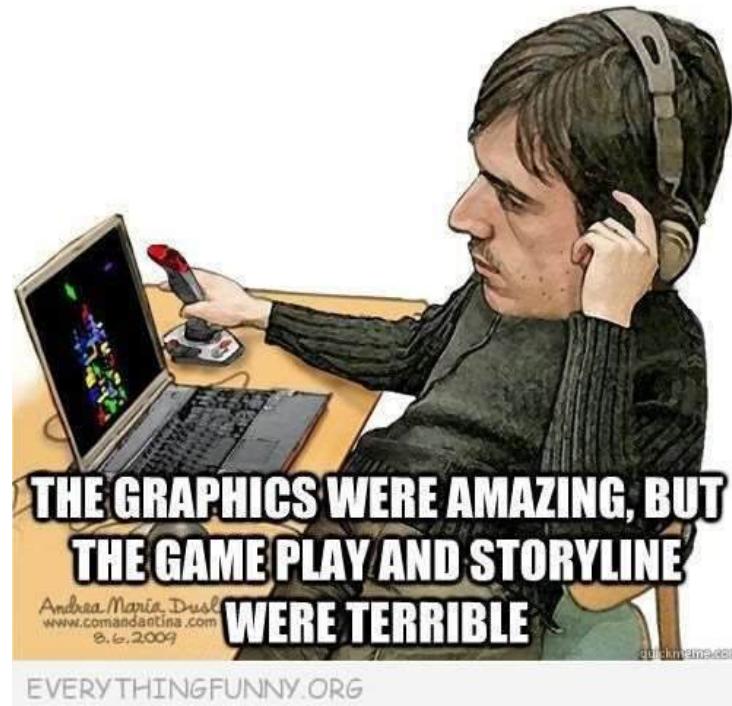
## Texture Summary

- Textures are used to attach data to scene points
  - Mapping performed via Interpolation
  - Filtering is crucial for high quality
- 
- Broad range of applications
    - Actually many more things  
... a little “foreshadowing” ;)



Thank you very much!

I WENT OUTSIDE ONCE



## Exercise: Texture Coordinates in Screen Space

- Example application:  
Add paper grain over your rendered image



How would you define the texture mapping  $T$  to obtain screen-space texture coordinates?  
Hint: Use the projected vertex position!

# CSE2215 - Computer Graphics

## Shadows Separate Light & Darkness

Elmar Eisemann

Delft University of Technology



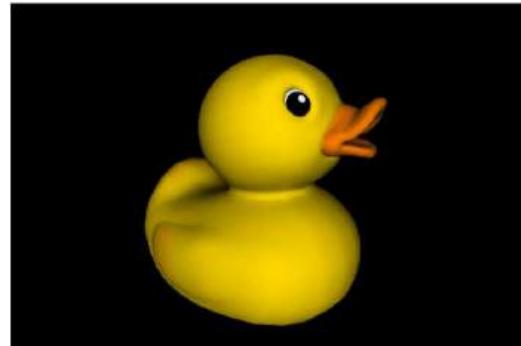
# Textures

- Mapping an image onto a surface

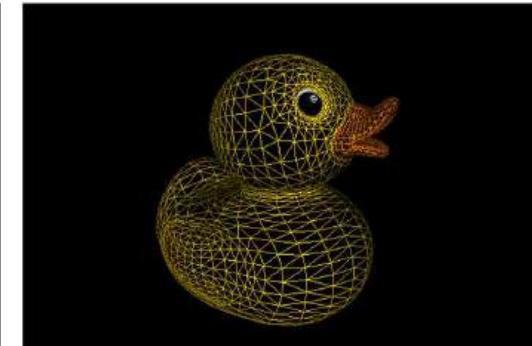
Texture



Texture mapped triangles



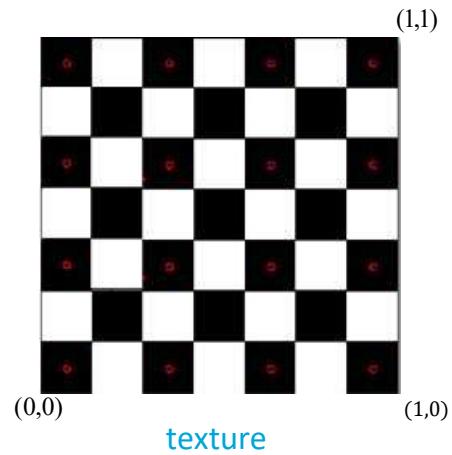
Wireframe of triangles



consisting of *texels*  
(texture elements)

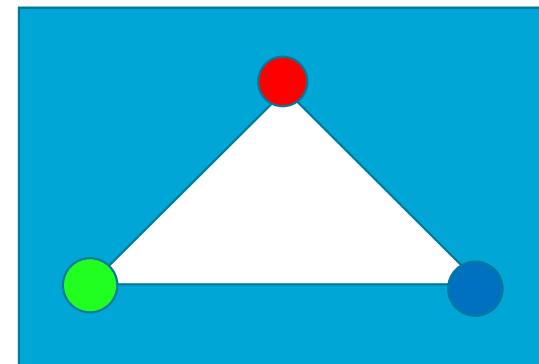
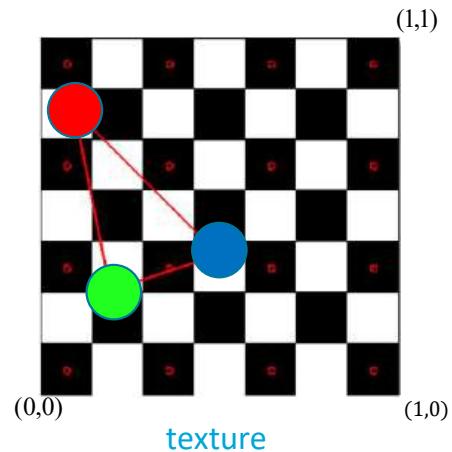
# Textures

- Map image via texture coordinates



# Textures

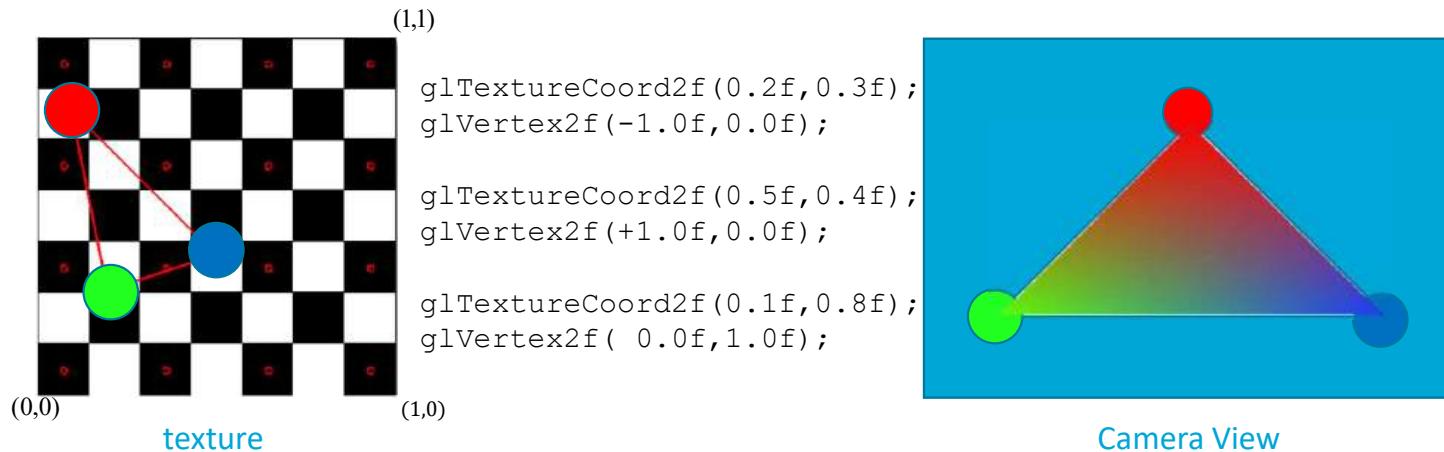
- Map image via texture coordinates
  - Specify a texture coordinate at each vertex



Camera View

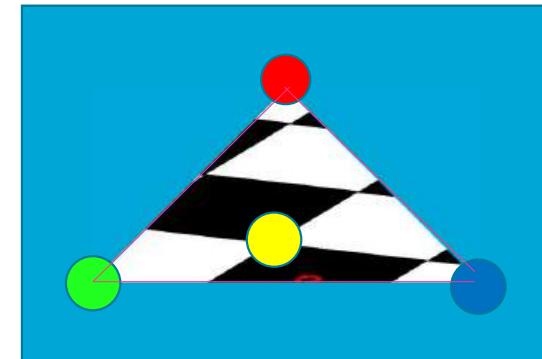
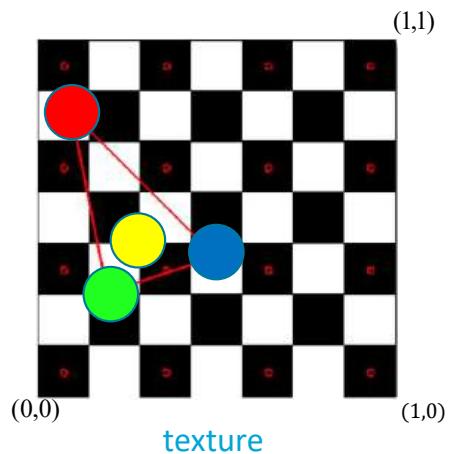
# Textures

- Map image via texture coordinates
  - Specify a texture coordinate at each vertex
  - Vertex texture coordinates are interpolated over triangle



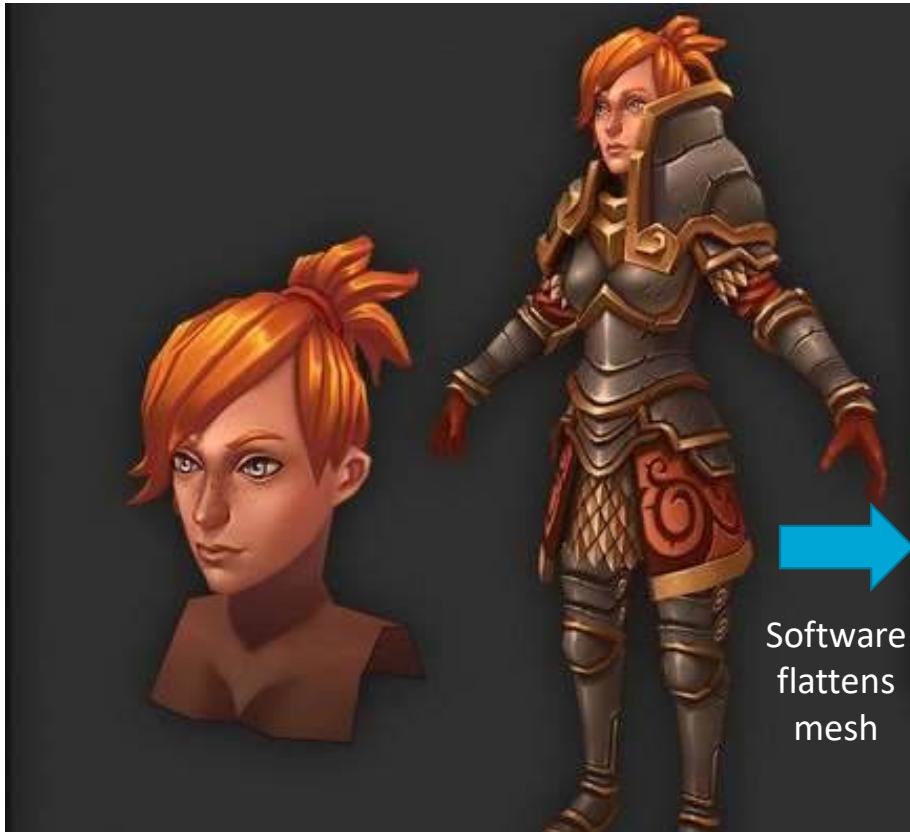
# Textures

- Map image via texture coordinates
  - Specify a texture coordinate at each vertex
  - Vertex texture coordinates are interpolated over triangle
  - Drawn pixels use interpolated coordinates to retrieve texel values



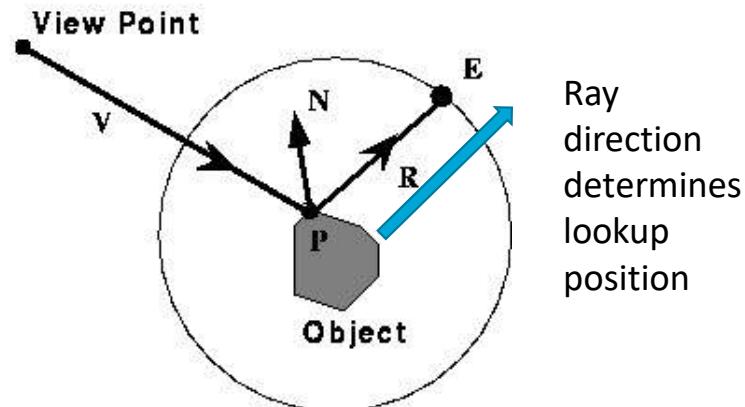
Camera View

## Texture Mapping: Special Software



# Environment Mapping

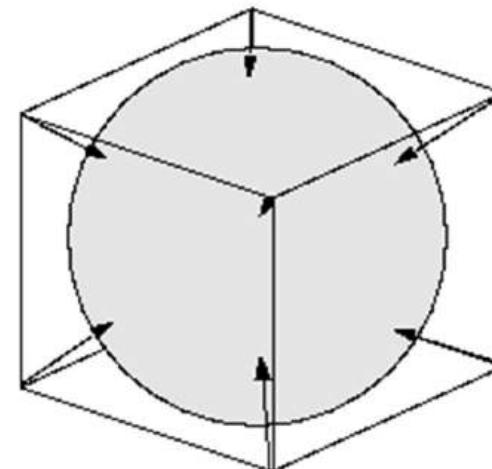
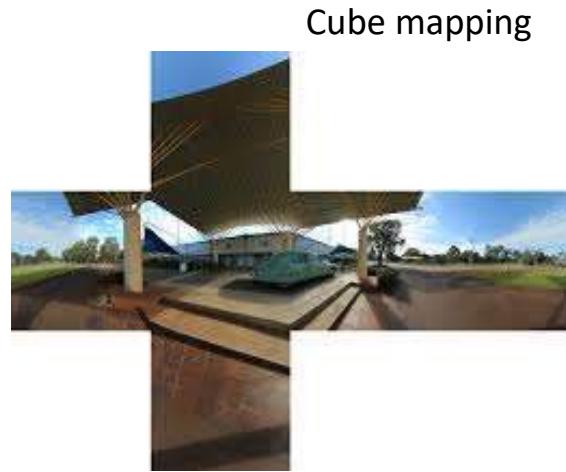
- Textures can encode an environment  
An environment map (approximation of scene)  
is useful for, e.g., reflections (texcoords = ray)



- This *environment mapping* is an approximation!

## Environment Mapping

- Alternatively, a sphere can be mapped to a cube
- Less distortions if images are used for the cube faces



It is costly, but you can even update such textures in every frame!  
Render an image (or here, it would be 6) and use it as a texture.

## Textures

- Image content mapped on surfaces
- Increase detail level without geometric cost
- Many applications for color textures



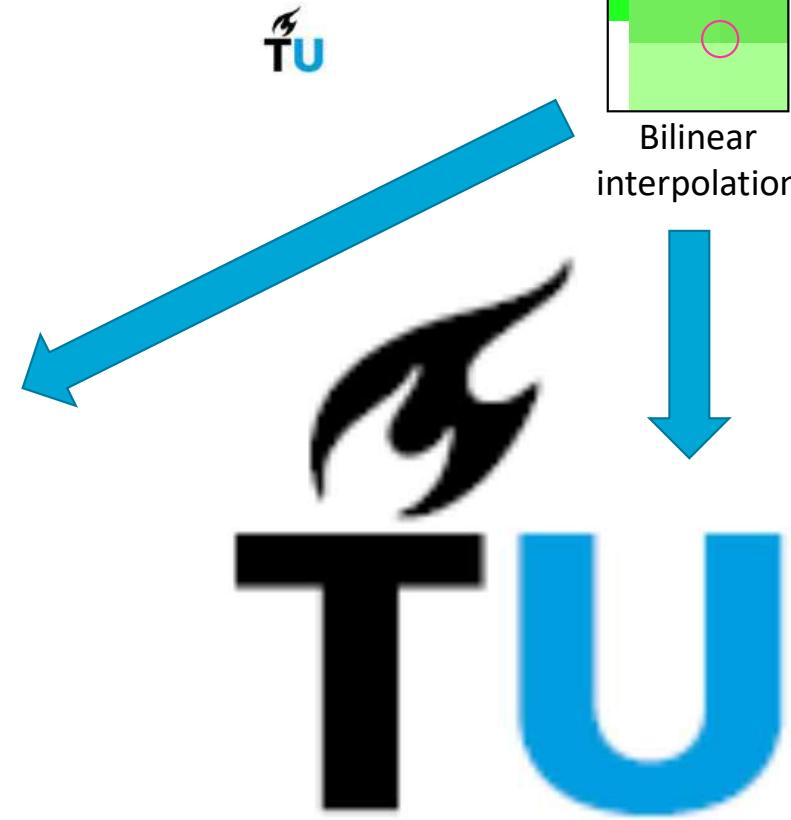
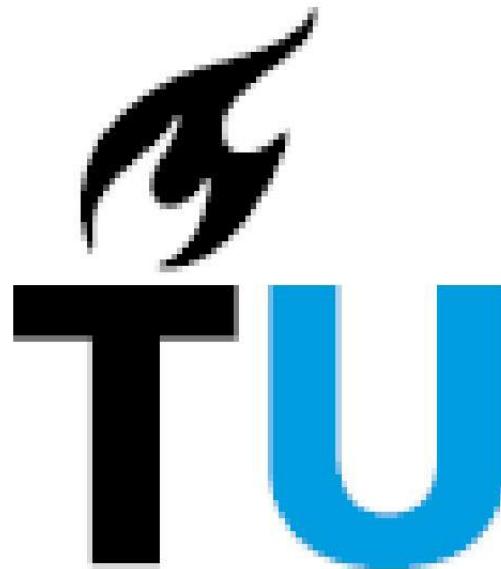
## Texture issues: Aliasing

- Not always completely beautiful...



## 1. Oversampling

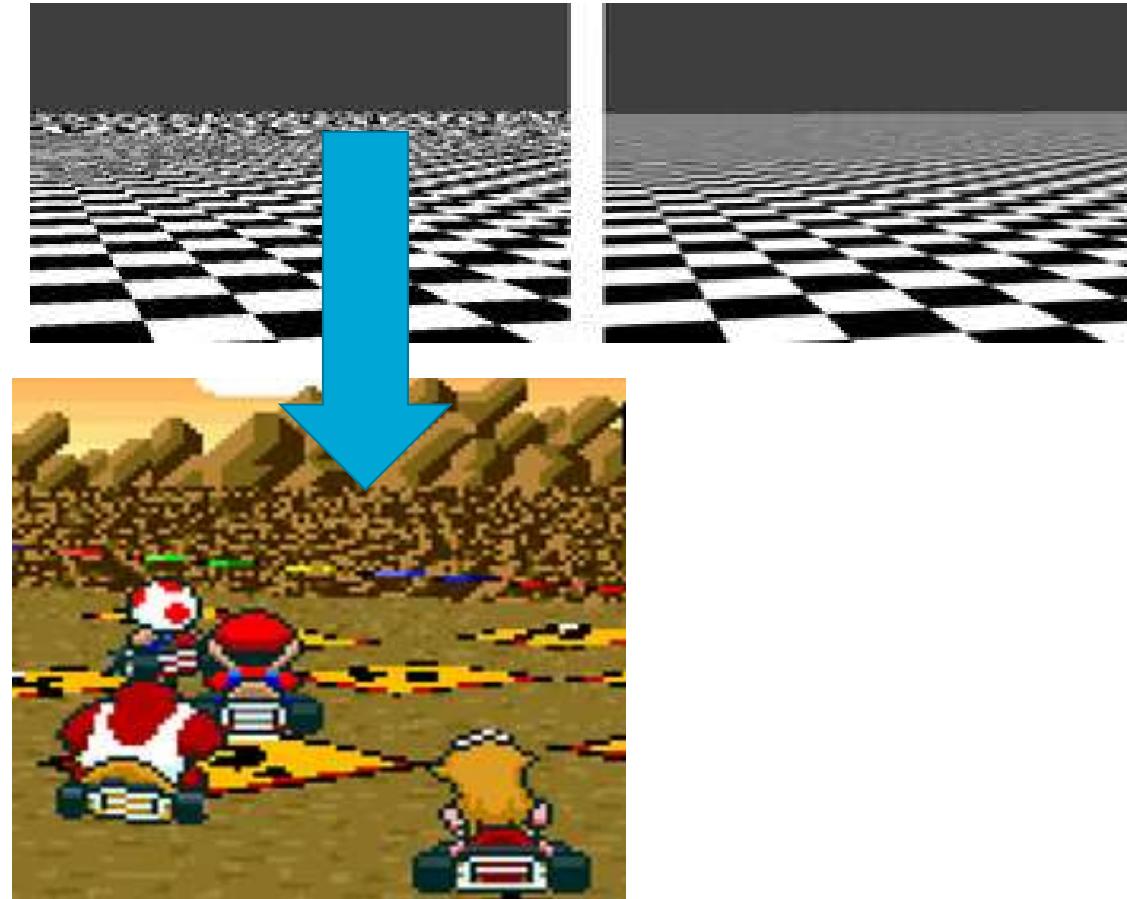
- Pixel smaller than texel



Nearest Neighbor

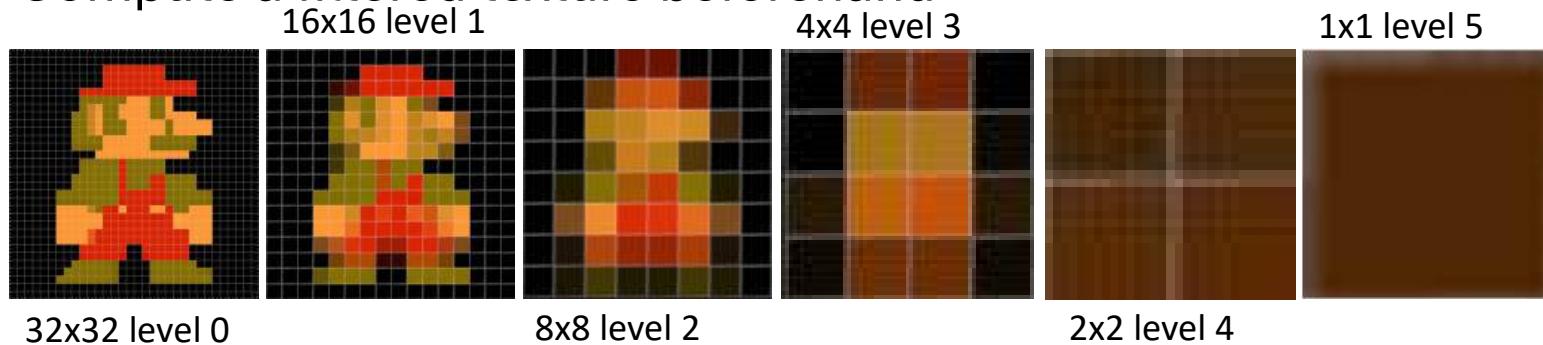
Bilinear Interpolation

## 2. Undersampling

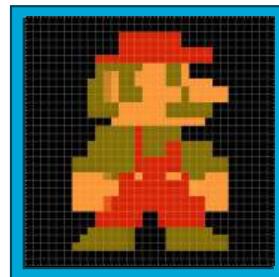


# MipMapping: Approximate Filtering

- Idea: Compute a filtered texture beforehand



Imagine  
A 32x32  
screen



Choose the correct level depending on pixel-to-texel matching

e.g., 4 texels per pixel

## Texture filtering: Off



## Texture filtering: On



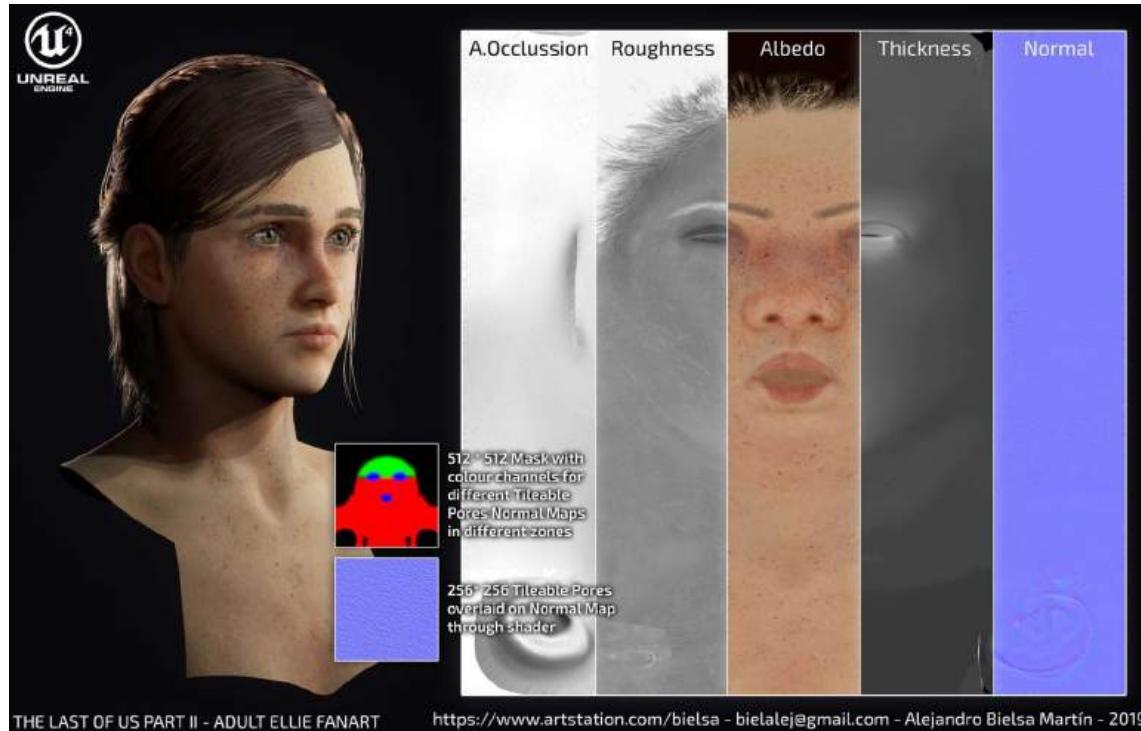
## Advanced Texture Representations

- Texture Data can represent more than just colors of an image
- Modern definition:  
A texture stores values associated to a domain

## Advanced Texture Representations

- Texture Data can represent more than just colors of an image
- Modern definition:  
A texture stores **values** associated to a domain

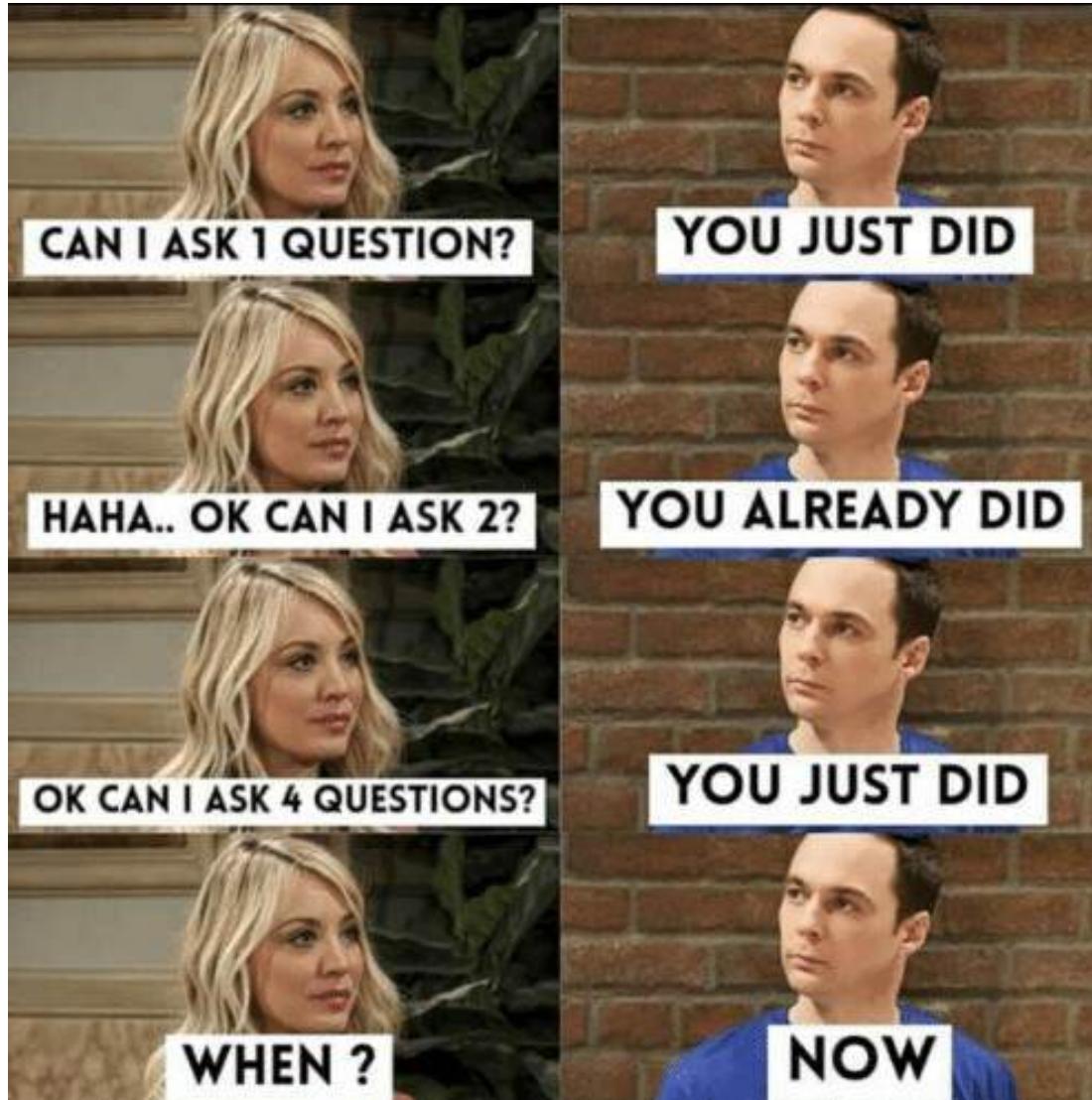
# Multiple Texture Maps store Surface Appearance





## Demo Time...

## Questions?



## Today's Study Goals

- S1- Explain and compare the structure and properties of **standard algorithms and data structures linked to Computer Graphics**.
  - 1D and 3D textures, and Shadow Mapping.
- S4- **Apply mathematical modeling** and theory of geometric computations and transformations, object representations, **simulation**, and encoding.
  - We discuss appearance, derive the formulas, and compute shadows.

## Advanced Texture Representations

- Texture Data can represent more than just colors of an image
- Modern definition:  
A texture stores values associated to a domain

## Advanced Texture Representations

- Texture Data can represent more than just colors of an image
- Modern definition:  
A texture stores **values** associated to a domain

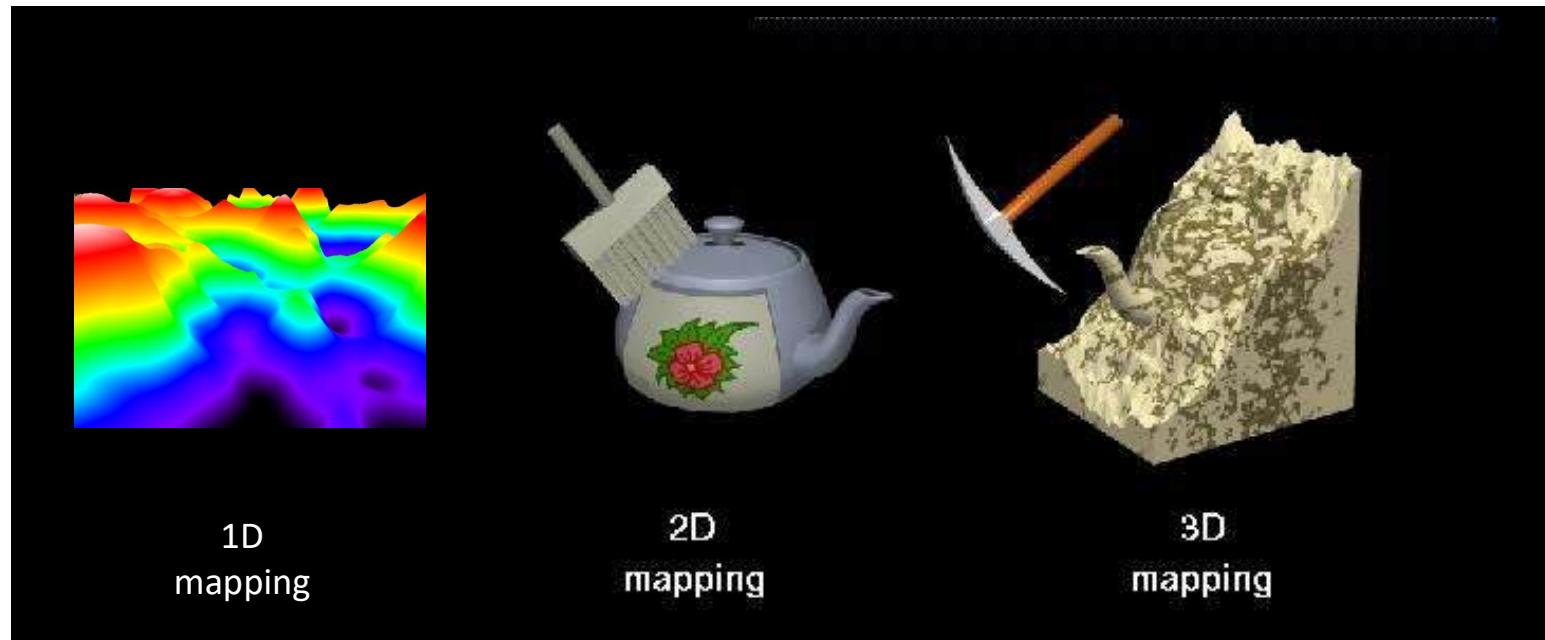
## Advanced Texture Representations

- Texture Data can represent more than just colors of an image
- Modern definition:  
A texture stores values associated to a **domain**

domain is accessed via the texture coordinates,  
so far, we had 2 coordinates to access a 2D image.

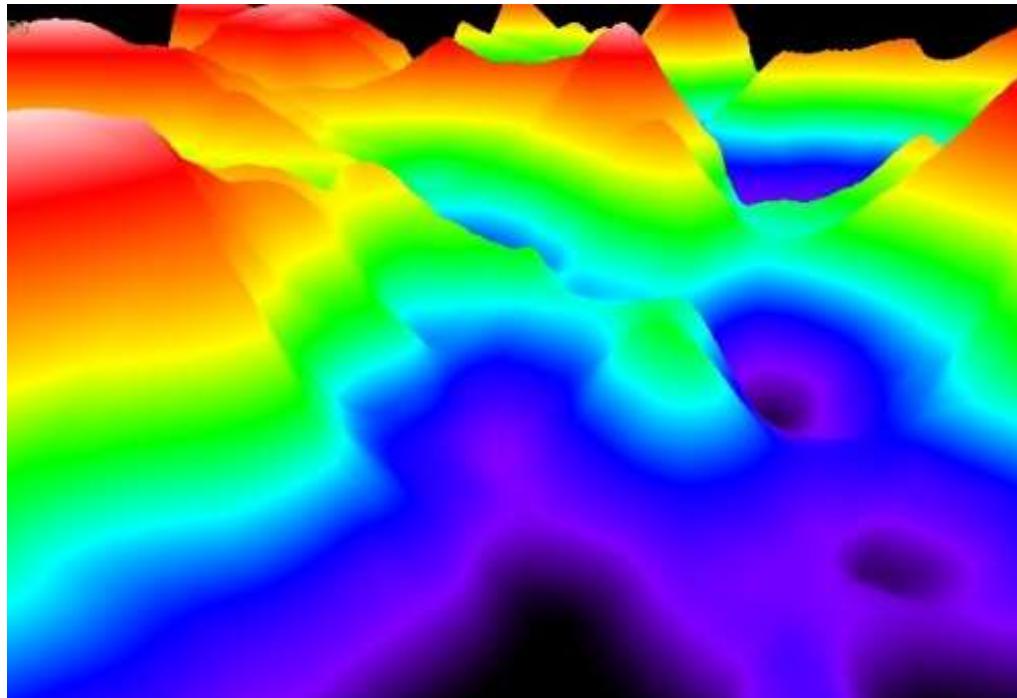
## Advanced Texture Representations

- Textures can be 1D, 2D, 3D... using `glTexCoord1f`, `glTexCoord2f`, `glTexCoord3f`. GPUs provide native support up to 3D



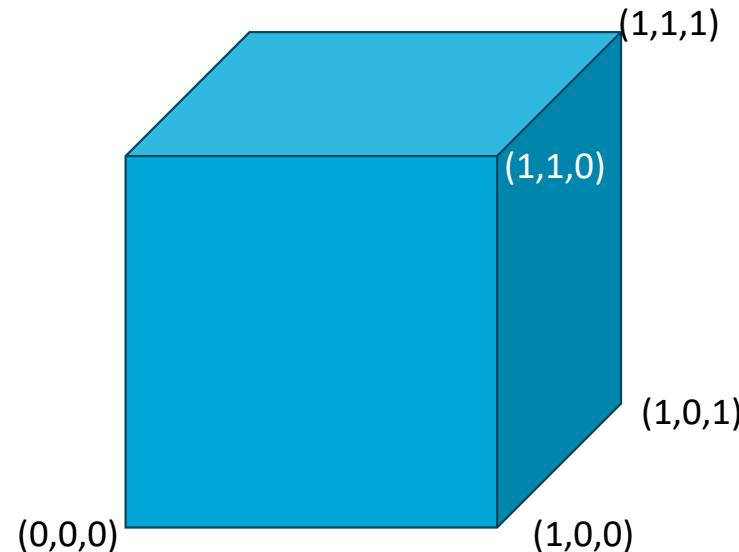
## We already saw one example of a 1D Texture

- We can use a 1-pixel wide 2D Texture accessed with `glTexCoord2f(0,z)`.
- A 1D Texture always has 1-pixel width. Access with `glTexCoord1f(z)`.



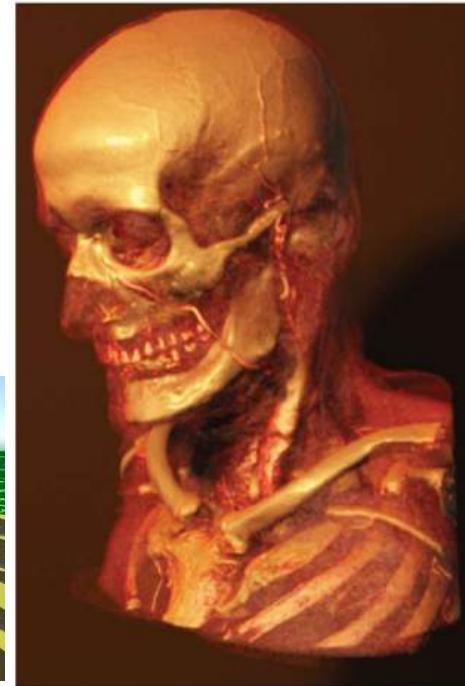
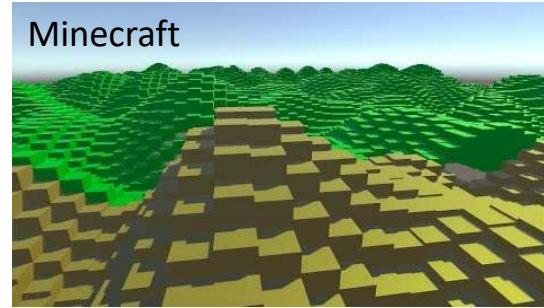
## What are 3D Textures?

- They are accessed with 3 texture coordinates ( $u,v,w$ )
- Bilinear interpolation in 2D textures becomes trilinear with 3D textures



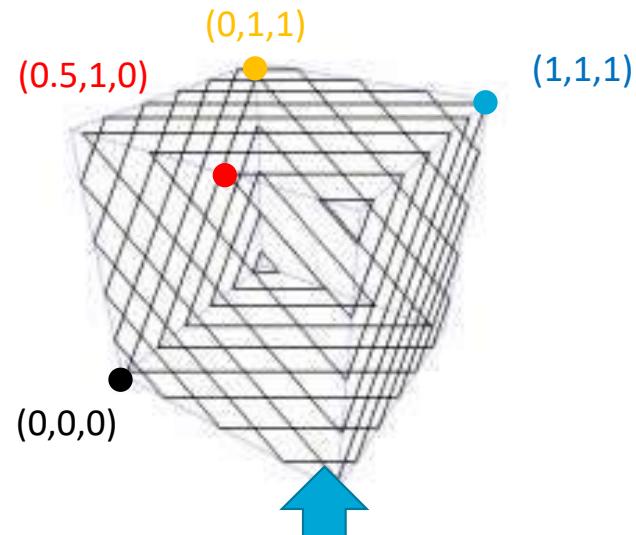
## 3D Textures are Volumes

- The equivalent of a texels is a small cubic volume (**volume element=voxel**)



# Volume Rendering

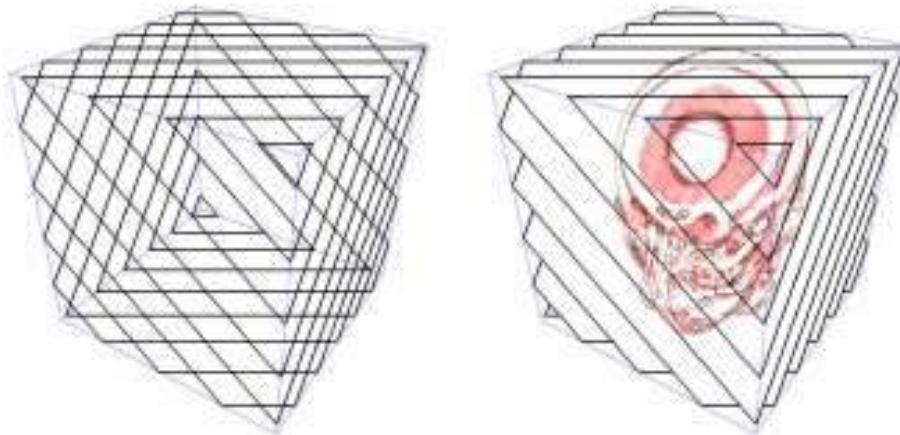
- How to render volumes?
  - One often-employed method is slicing



Draw slices with 3D texture coordinates  
corresponding to the position of the  
vertices in the volume

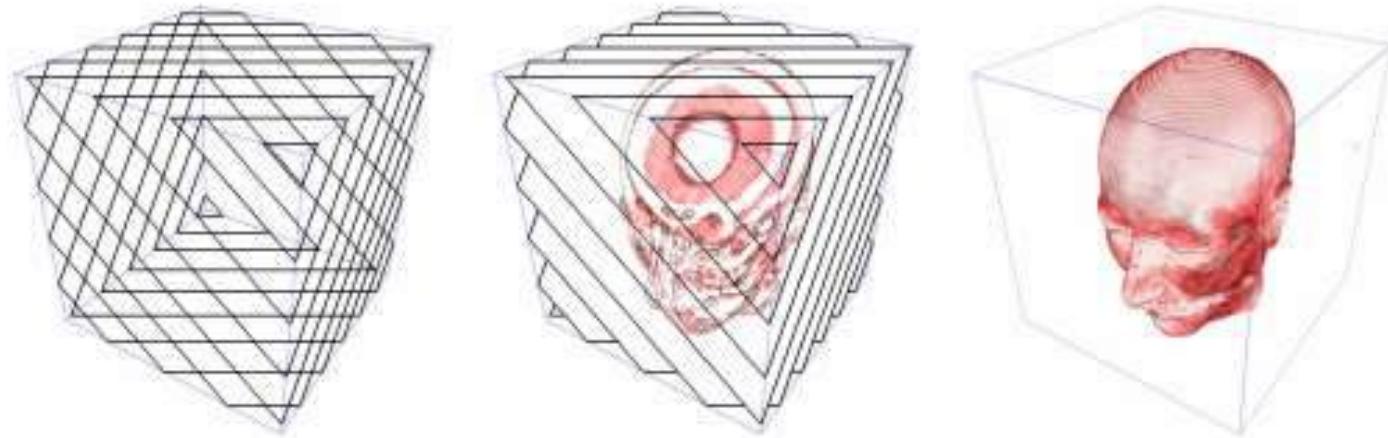
# Volume Rendering

- How to render volumes?
  - One often-employed method is slicing



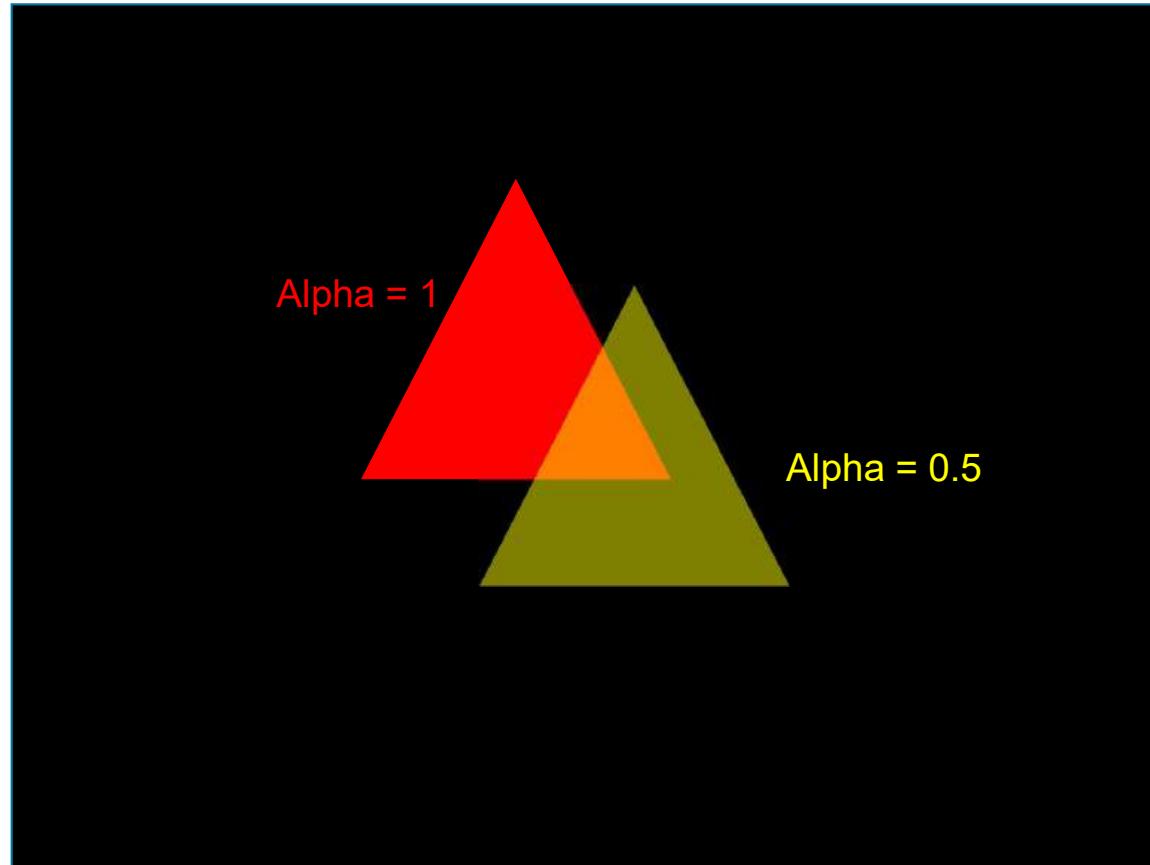
# Volume Rendering

- How to render volumes?
  - One often-employed method is slicing



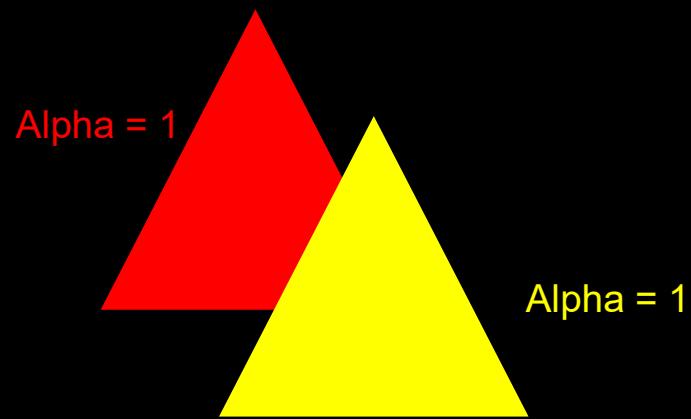
But wait? Why are there parts that are transparent?

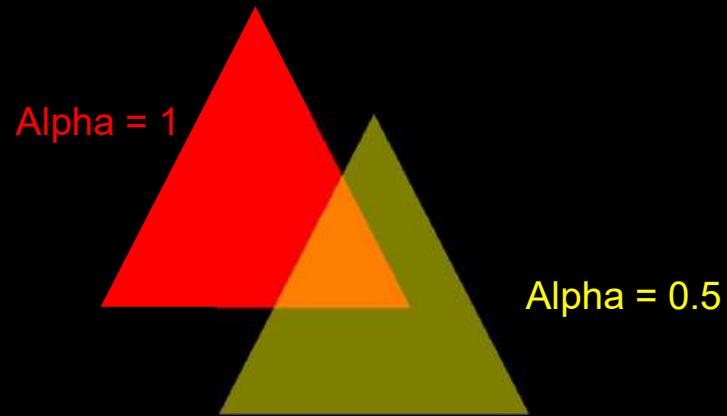
# Alpha Blending

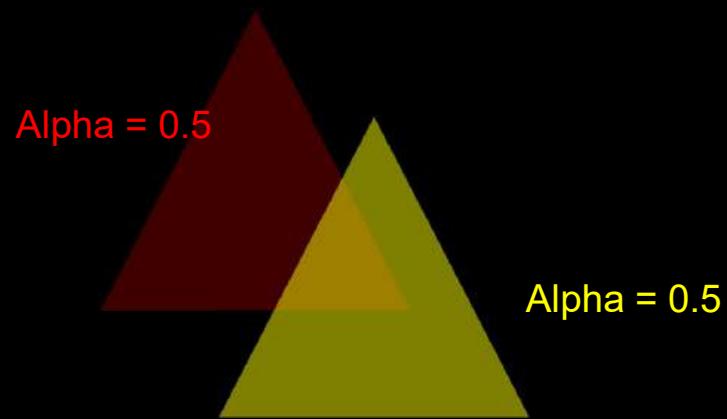


## Standard Alpha Blending

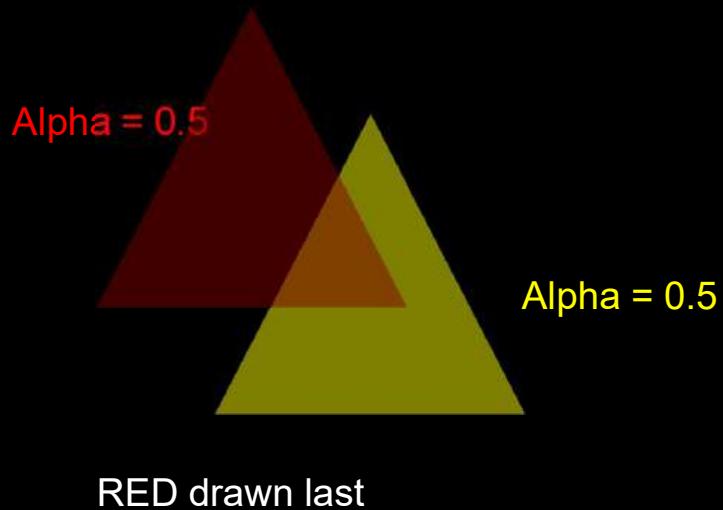
- Color with four components: RGB and an alpha value
- Alpha describes the “opacity”
  - 0 : object is completely transparent
  - 1 : object is completely opaque
  - 0.5: 50% of the stored color and 50% background
- In general, drawn  $(R, G, B, A)$  and in a pixel with color  $(R_B, G_B, B_B)$  results in:  
$$A * RGB + (1 - A) * R_B G_B B_B$$



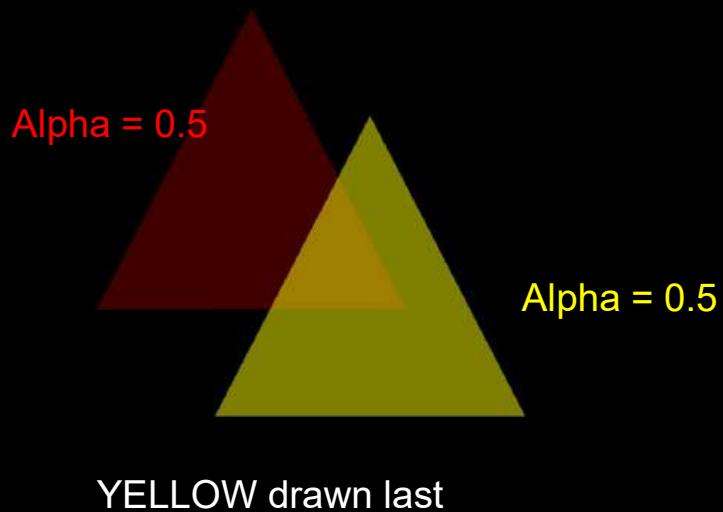




Attention: ORDER is important



Attention: ORDER is important



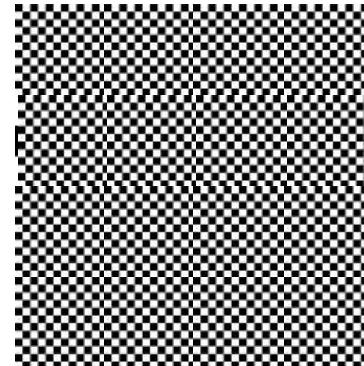
## Why this formula for Alpha Blending?

- Alpha is the opacity of the object
- Imagine the object to have tiny holes with probability Alpha
- 0.5 could be imagined as a checkerboard



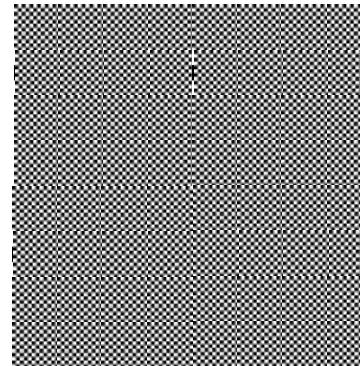
## Why this formula for Alpha Blending?

- Alpha is the opacity of the object
- Imagine the object to have tiny holes with probability Alpha
- 0.5 could be imagined as a checkerboard



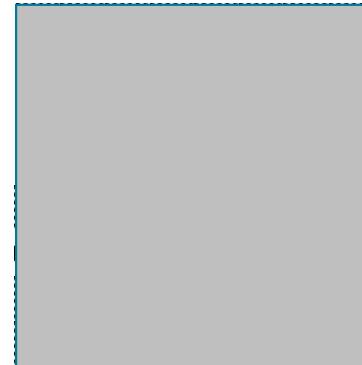
## Why this formula for Alpha Blending?

- Alpha is the opacity of the object
- Imagine the object to have tiny holes with probability Alpha
- 0.5 could be imagined as a checkerboard



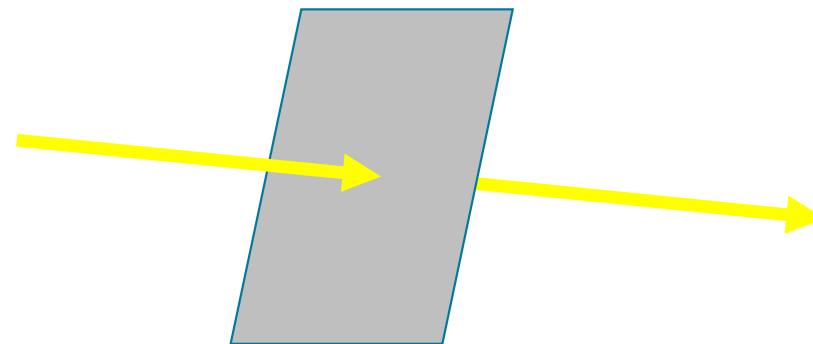
## Why this formula for Alpha Blending?

- Alpha is the opacity of the object
- Imagine the object to have tiny holes with probability Alpha
- 0.5 could be imagined as a checkerboard



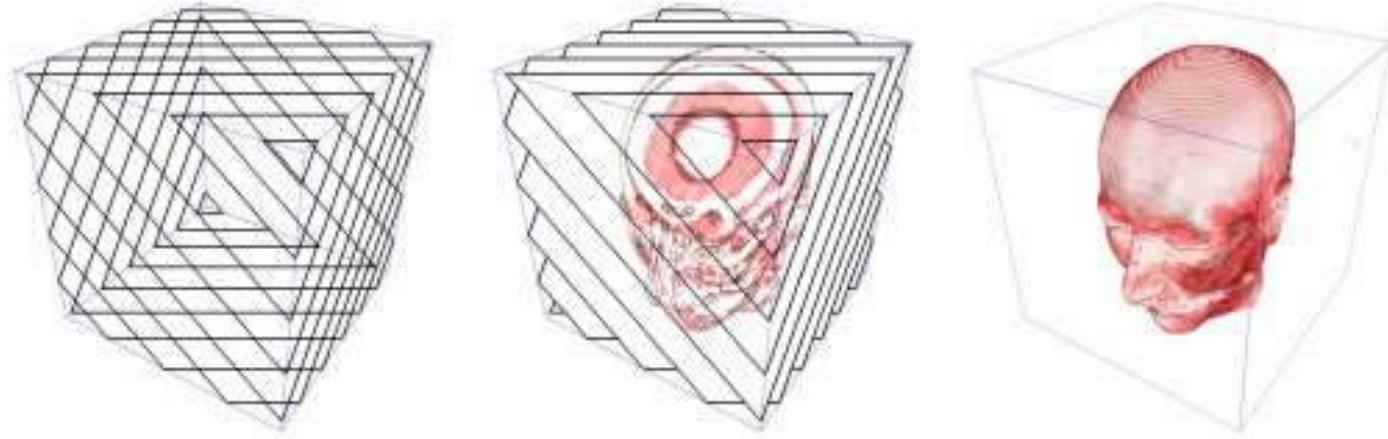
## Why this formula for Alpha Blending?

- A light ray hits with probability  $A$  and passes with probability  $(1-A)$ , hence,  $A * \text{color} + (1-A) * \text{Background}$



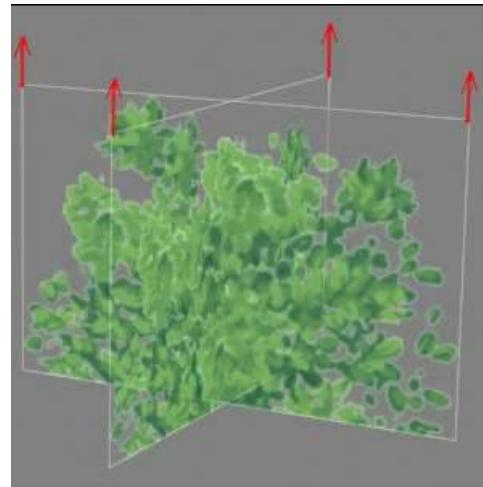
# Volume Rendering

- One often-employed method is slicing



## Alpha Blending

- Used to create transparency effects
- Textures can have (R,G,B,A) values:
  - If  $A=0$  : texel is considered transparent – not drawn
  - If  $A=1$  : texel is drawn



## Alpha Blending - Stacked Polygons

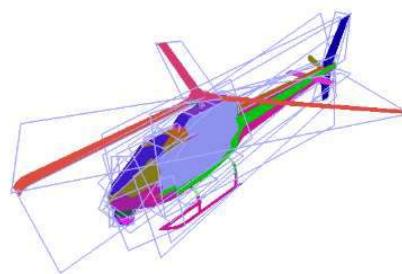


- Source: irrlicht engine, artplants.blogspot.com

# Billboard Clouds



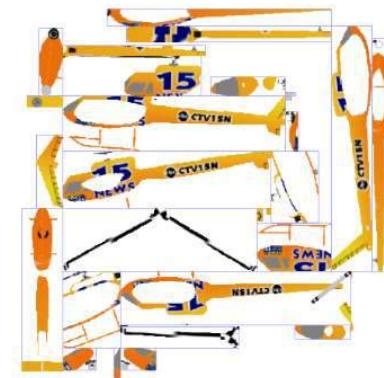
(a)



(b)



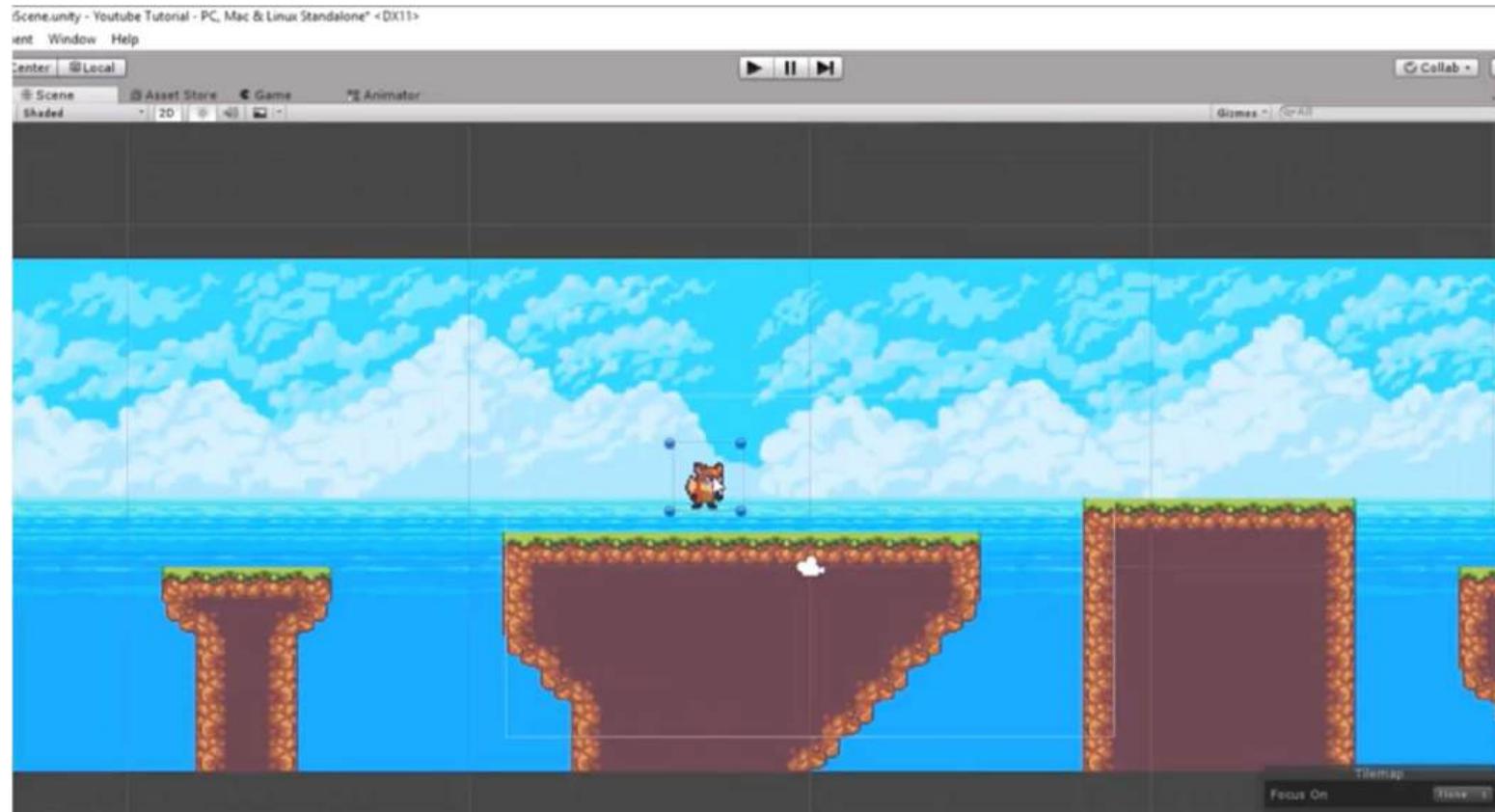
(c)



(d)

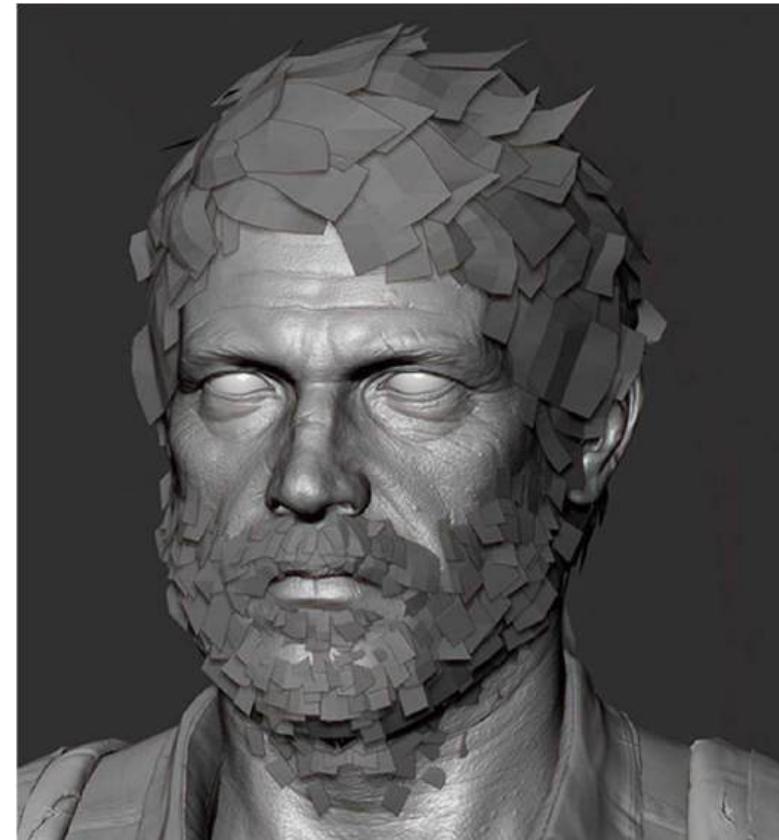
Figure 1: Example of a billboard cloud: (a) Original model (5,138 polygons) (b) false-color rendering using one color per billboard to show the faces that were grouped on each (c) View of the (automatically generated) 32 textured billboards (d) the billboards side by side.

## Alpha Blending - Sprites (here Unity)



Use pixel discard based on alpha to make the object background transparent

## Alpha Blending - Hair



## Alpha Blending - Hair



Last of us

## Alpha Blending - Water Particles

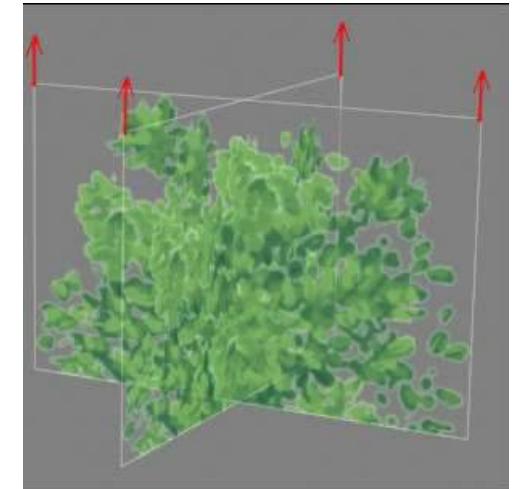
- Textured quads used as water particles



# Alpha Blending

Pros:

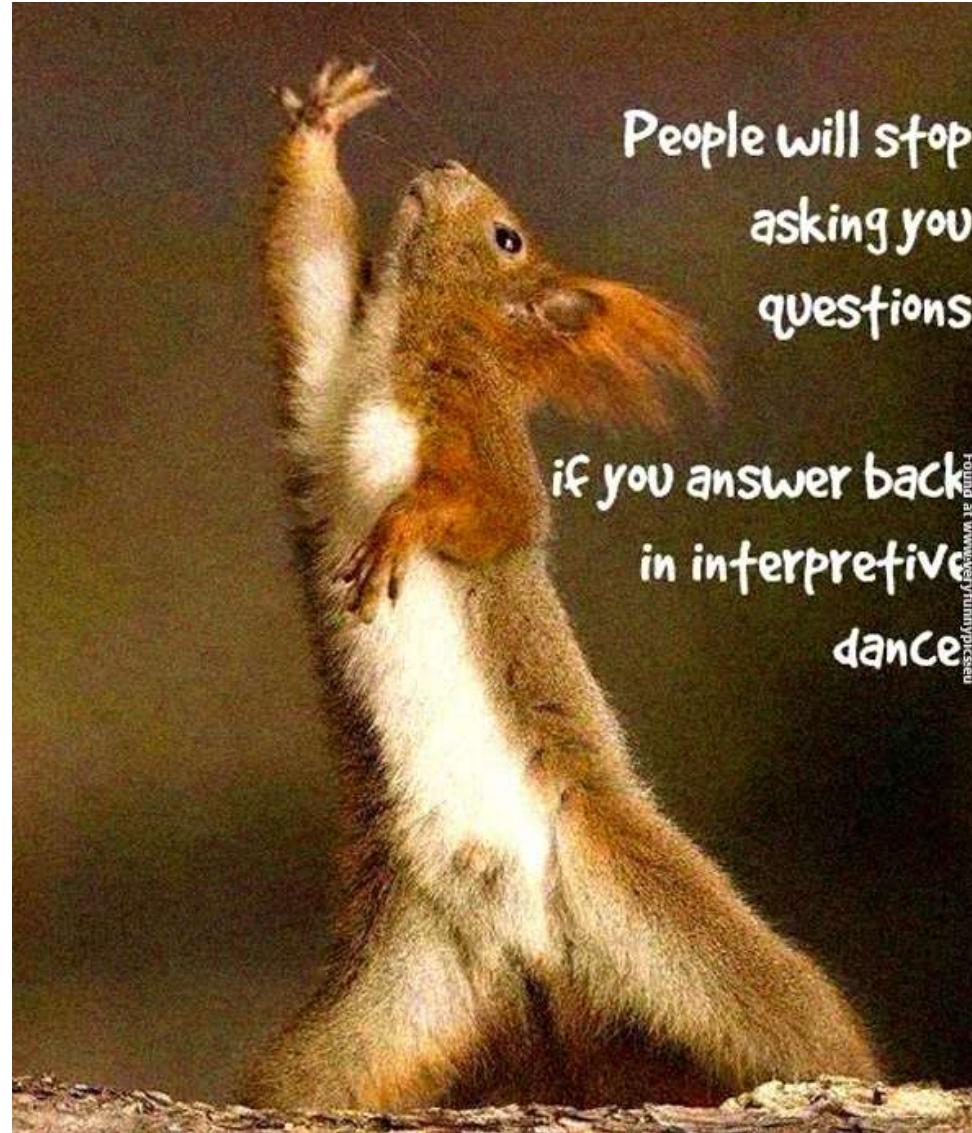
- Give the illusion of very complex geometry
- Can mimic transparent materials (glas, smoke, ...)



Cons:

- Additional cost:
  - Generally need to sort triangles
  - If only binary alpha values (0 and 1), sorting can be omitted

# Questions?



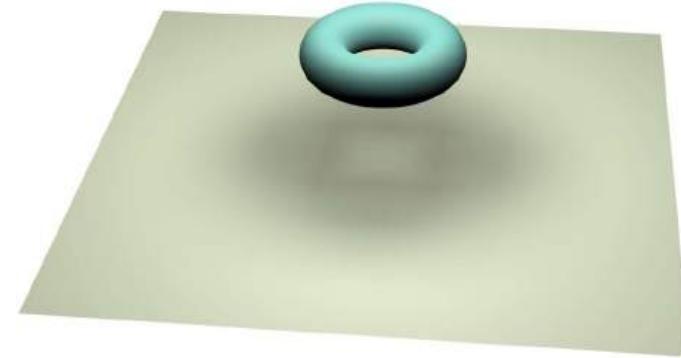
## What is a Shadow?

- WordNet:

*Shade within clear boundaries*

*or*

*An unilluminated area.*

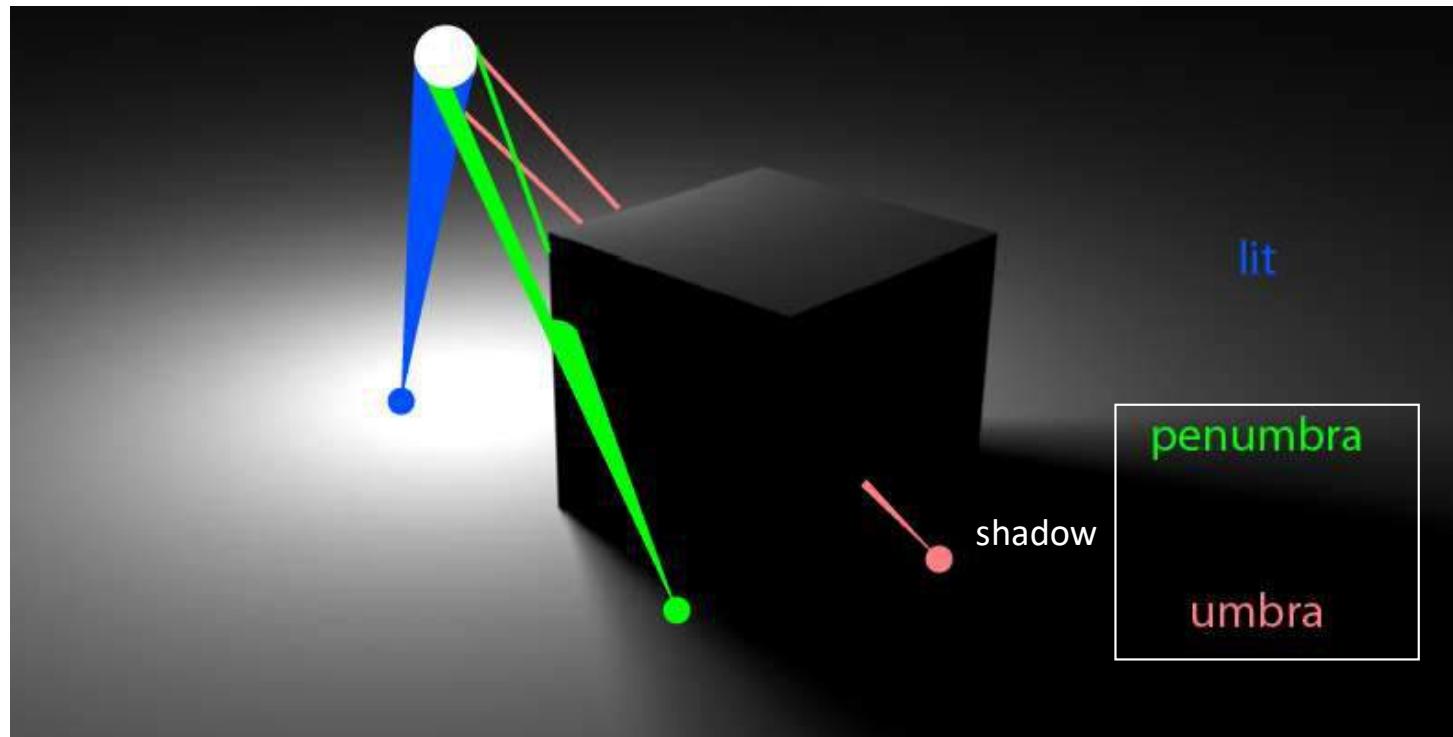


## What is a Shadow?

- Hasenfratz et al. [2003]:

*Shadow [is] the region of space  
for which at least one point of the light source is occluded.*

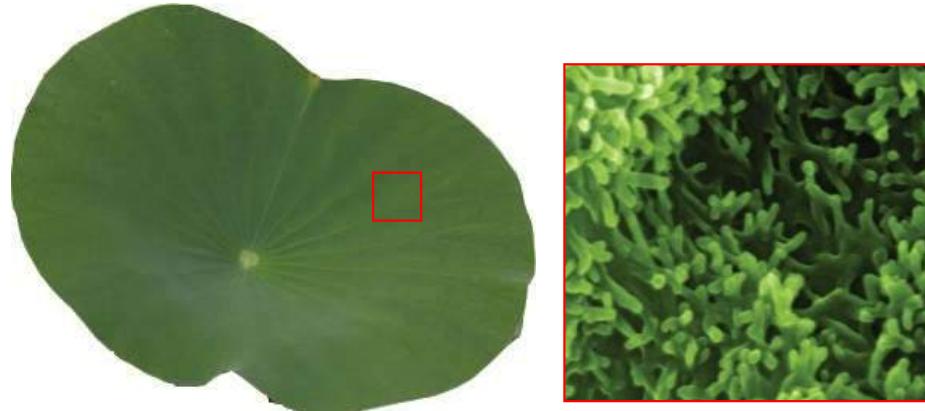
# What is a Shadow?



## What is a Shadow?

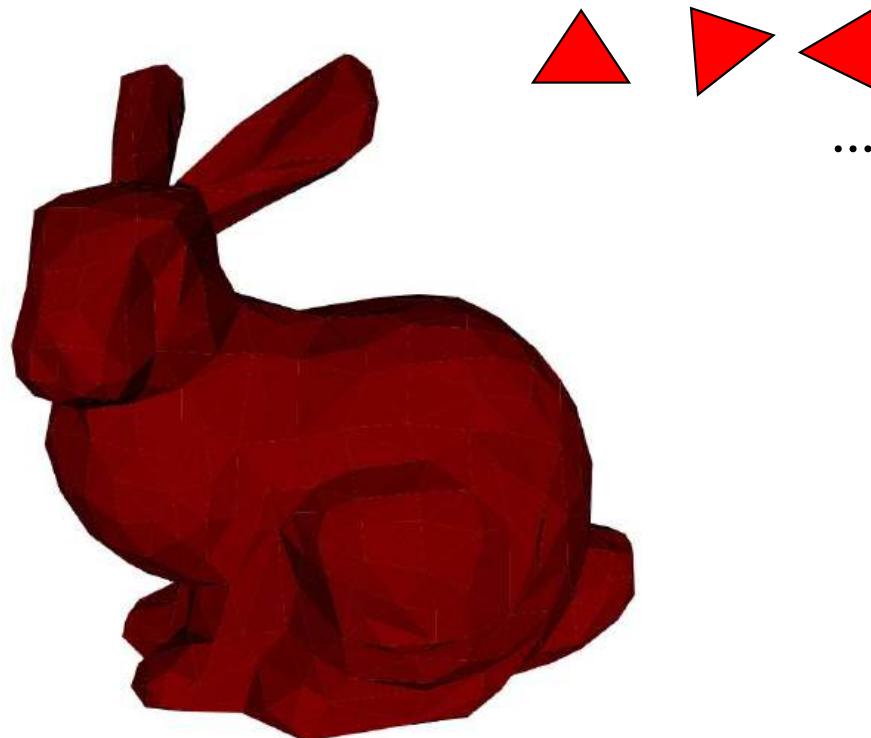
- Hasenfratz et al. [2003]:

*Shadow [is] the region of space  
for which at least one point of the light source is occluded.*



## Our models are simple...

- Typically a list of triangles



## What is a Shadow?

- Hasenfratz et al. [2003]:

*Shadow [is] the region of space  
for which at least one point of the light source is occluded.*

# How to draw shadows?

- Artists know how to draw shadows!

Or not?



Fra Carnevale  
(1467)

## How to draw shadows?

- Artists know how to draw shadows!

Or not?



Signorelli  
(1488)

62

## How to draw shadows?

- Artists know how to draw shadows!

Or not?



Moore  
(1893)

## How to draw shadows?

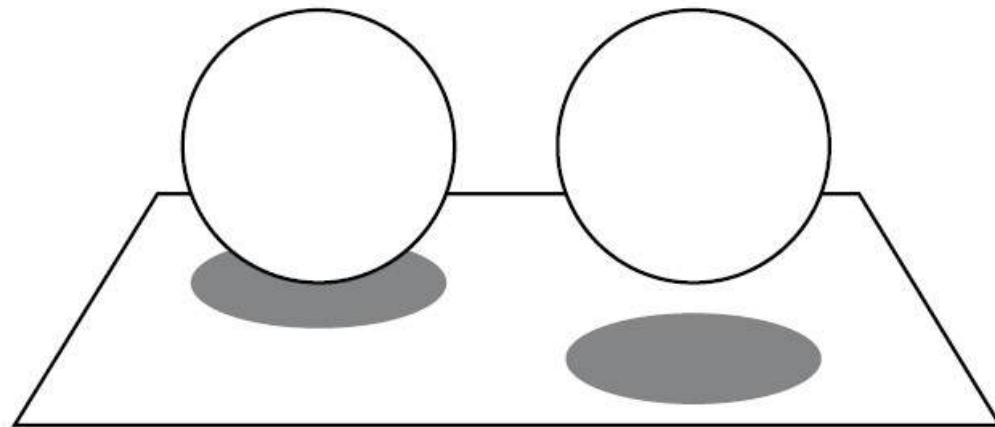
- Drawing shadows is apparently difficult...

## So why not just ignore shadows?

- Shadow of the Colossus, Sony



## So why not just ignore shadows?



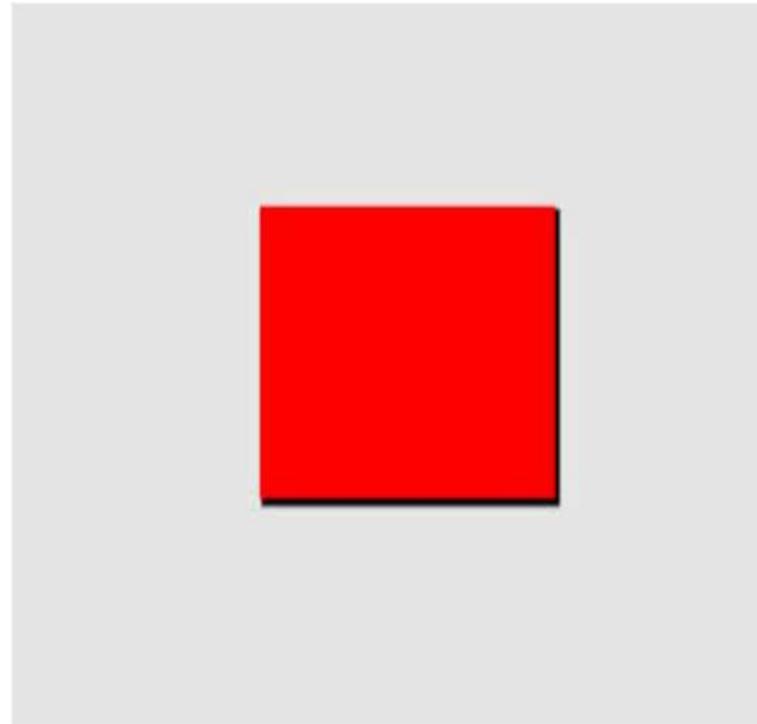
## So why not just ignore shadows?

- Shadow of the Colossus, Sony



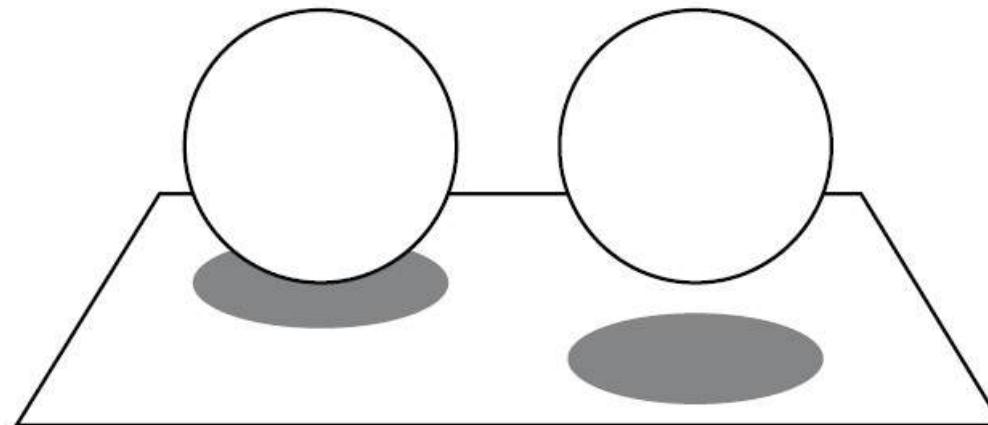
# Psychophysical-Experiments

[Kersten et al. 96]

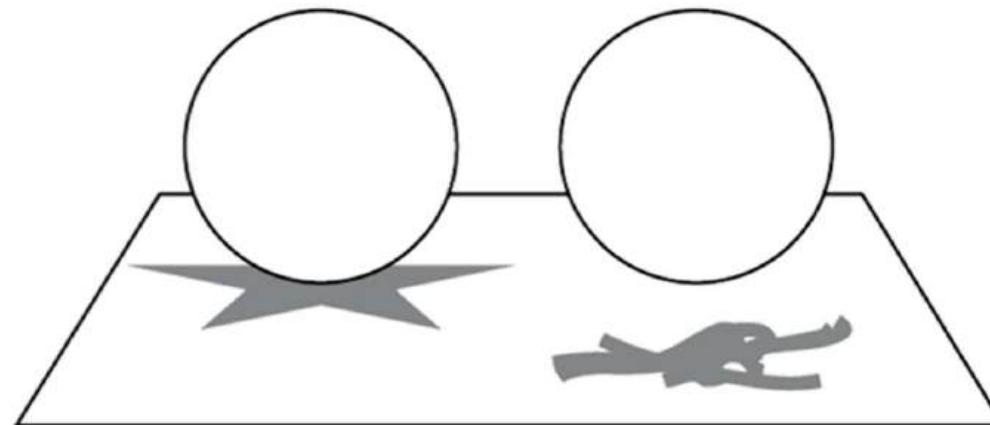


## So why not just ignore shadows?

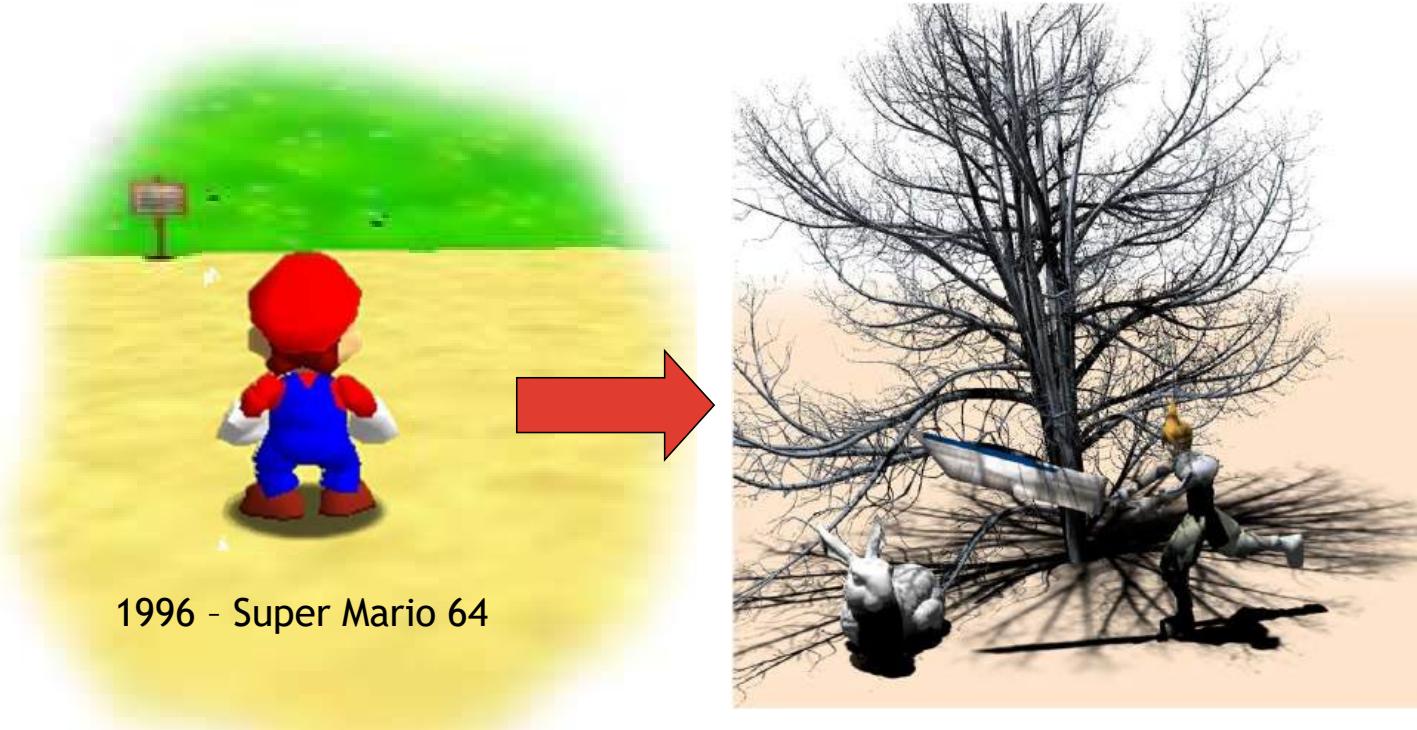
- But this is not a good argument for realistic shadows...



## So why not just ignore shadows?

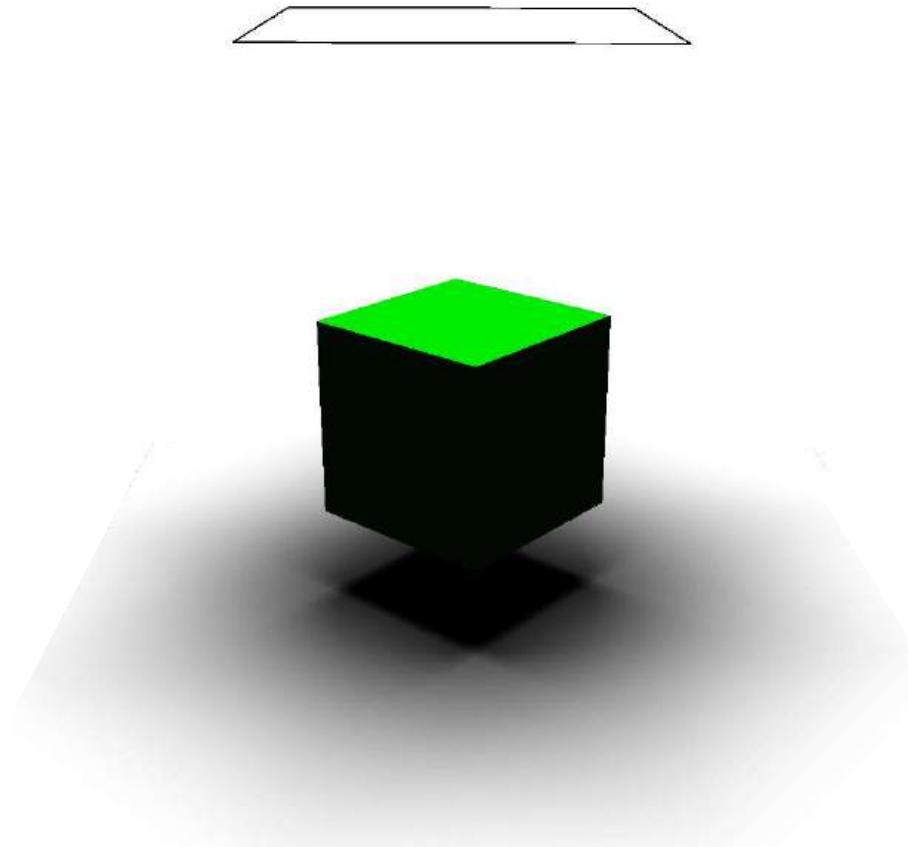


## Simple shadows can be sufficient



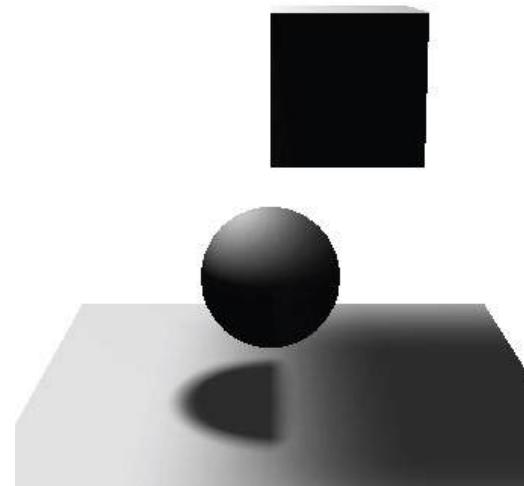
1996 - Super Mario 64

## Plausible shadows



## Plausible shadows

- Attention: “plausible” can often fail



## Realistic shadows are important



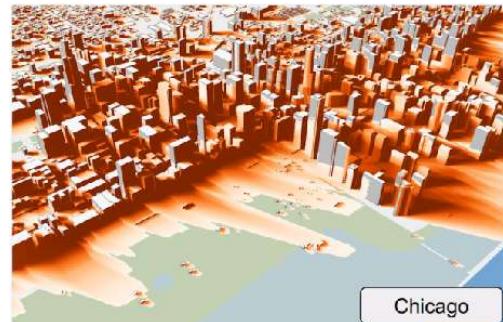
## Shadows can give information!

- Courtesy of Hasenfratz et al.



## Realistic shadows are important

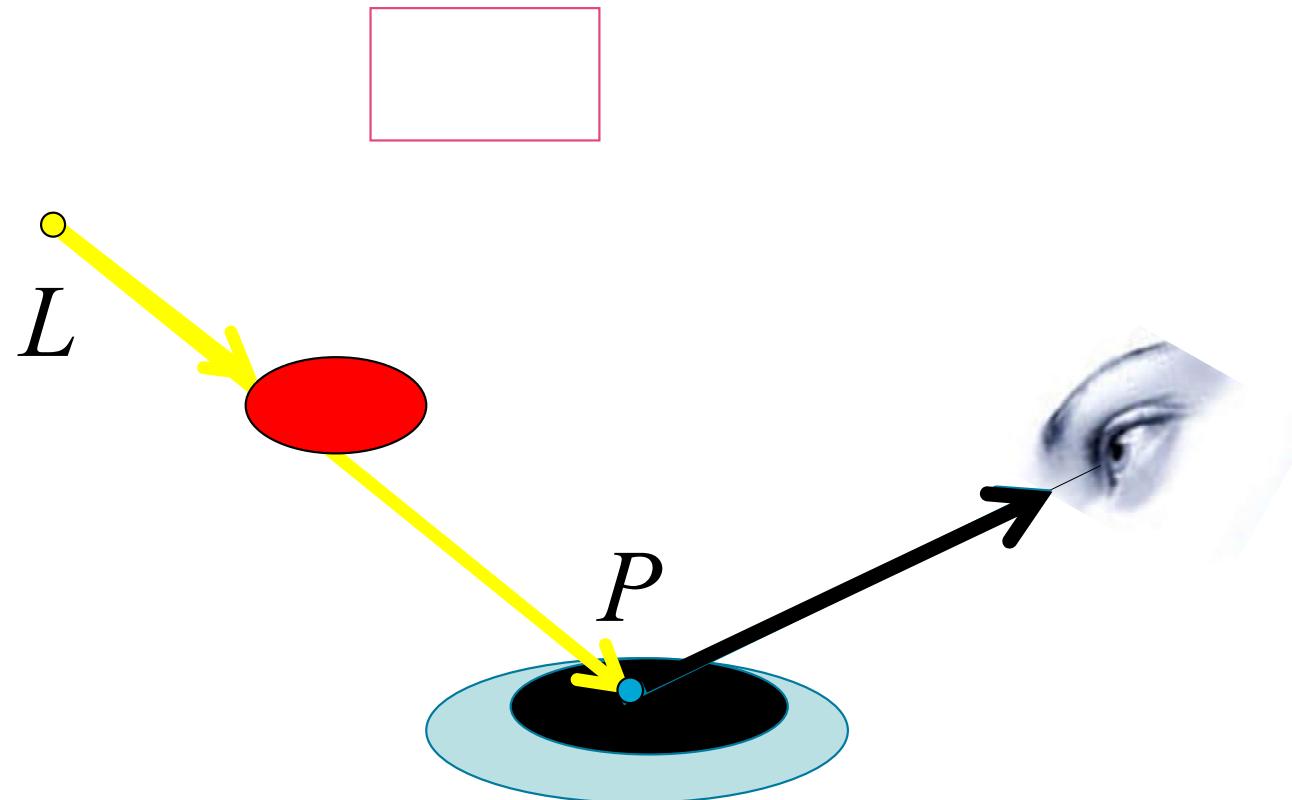
- Many contexts in which accuracy is needed:
  - Architecture
  - Simulation
  - Movies
- ...



Desert villa by Studio Aiko

Miranda et al. TVCG2019

## How to compute shadows?



## Hard Shadows



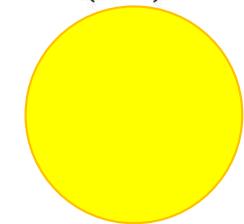
Point lights do not create penumbras

## Soft Shadows

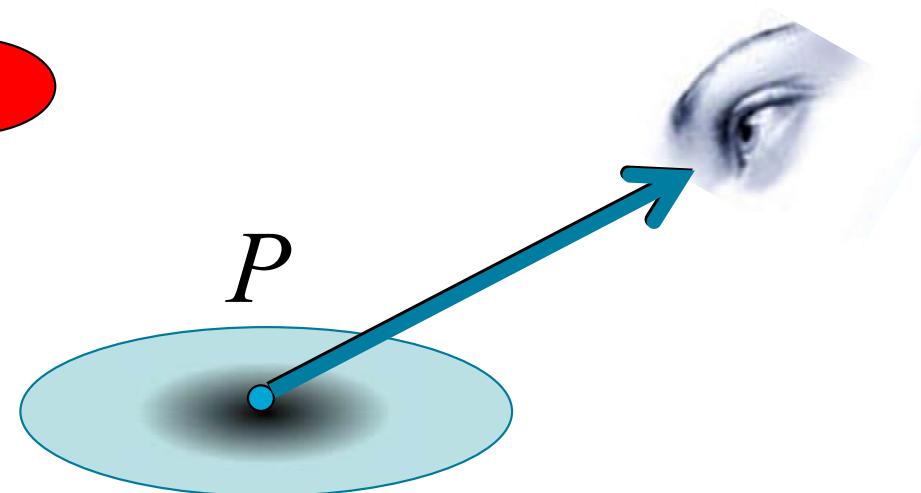
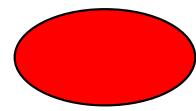


## What is a Shadow ? – Part II

$$B(P) = E(L) \nu(P,L) \text{ Transfer}(P,L)$$

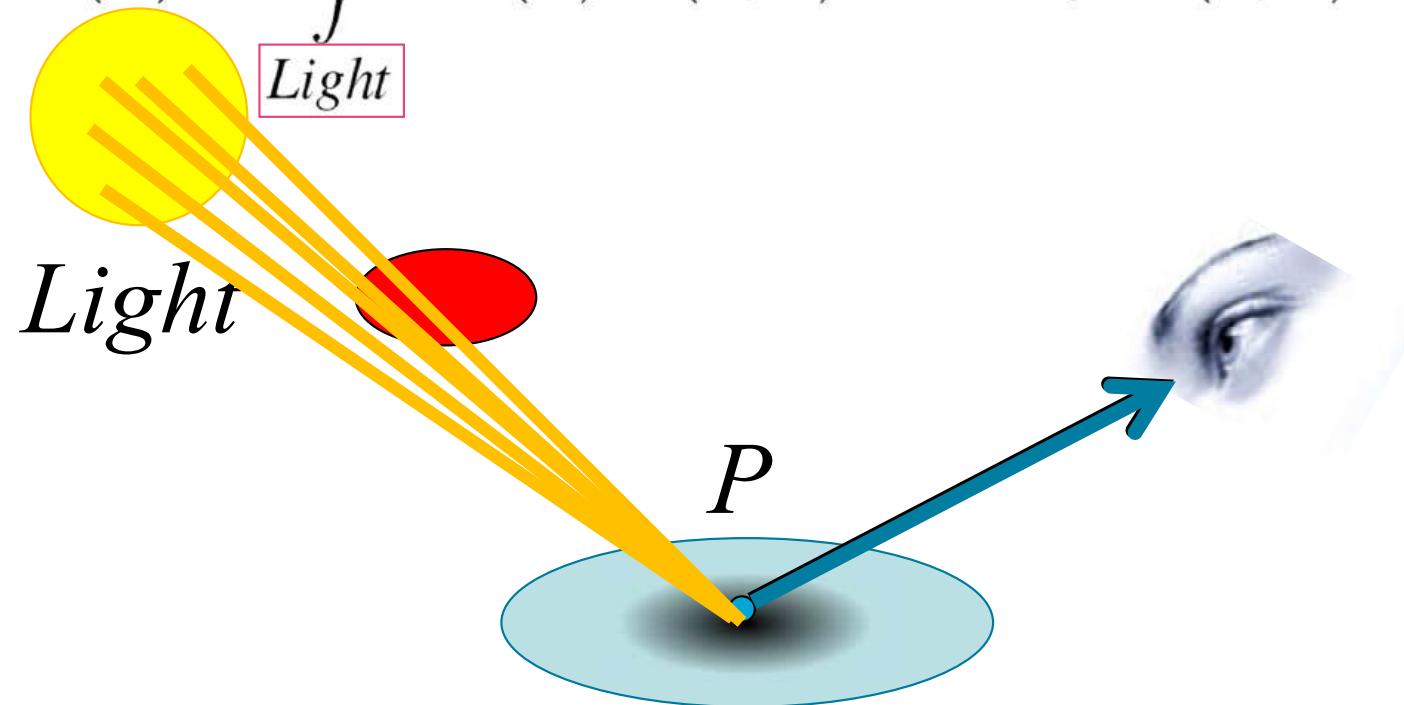


*Light*

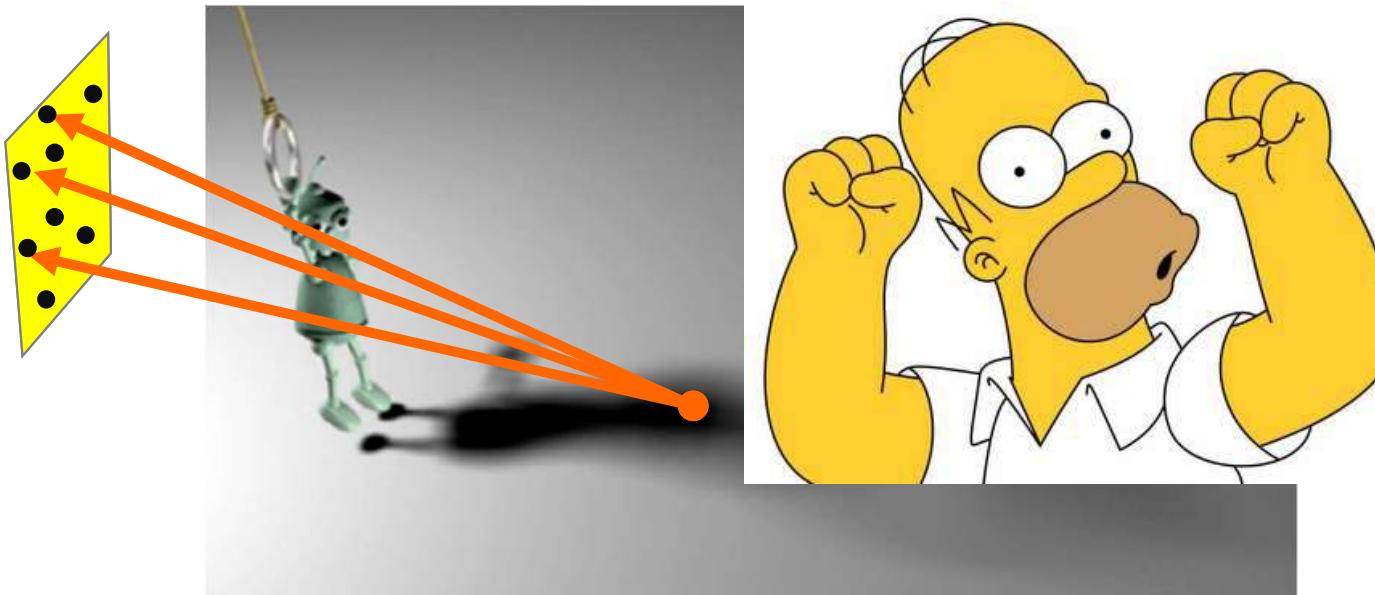


## What is a Shadow ? – Part II

$$B(P) = \int E(L) v(P,L) \text{ Transfer}(P,L) dL$$



## Lecture 1? Ray Tracing?



- Physically correct (...enough)
- Robust
- Easy to implement

# Ray Tracing?

1 sample



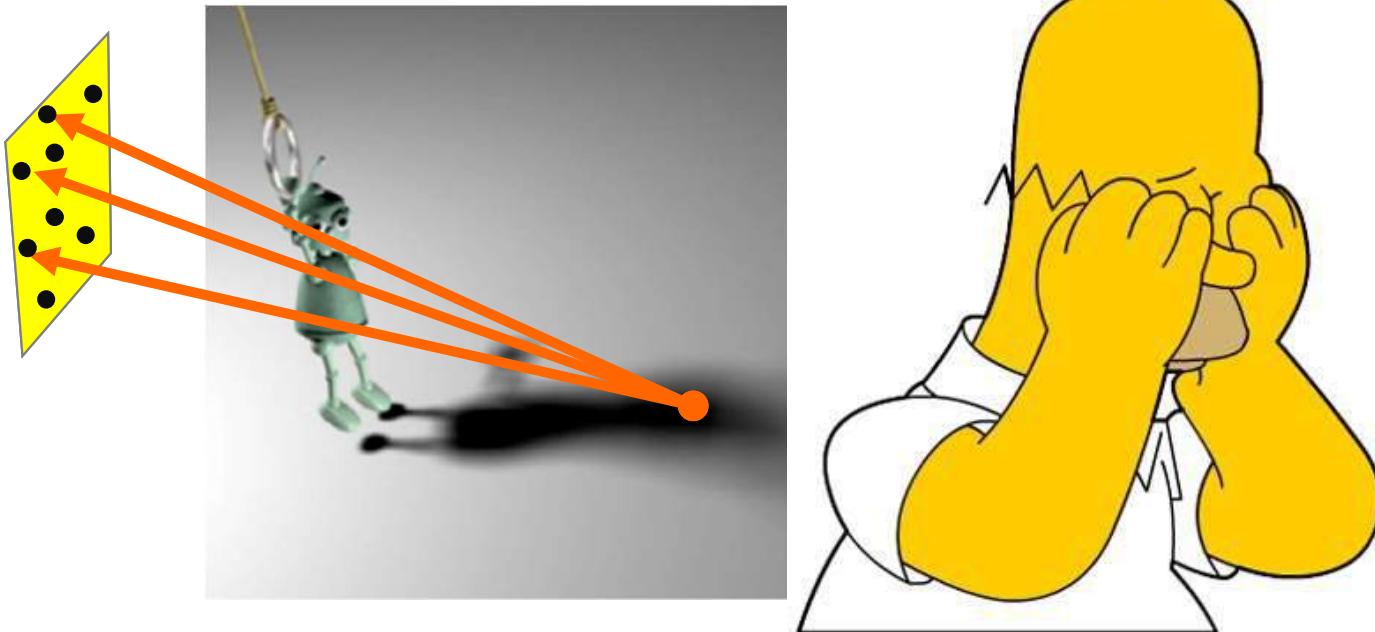
32 samples



512 samples



## Ray Tracing



Even on modern GPUs (RTX) relatively costly...

**SLOW!**

## Today: Hard Shadows within the Rendering Pipeline

- Meaning: light source  $L$  is a point

$L$

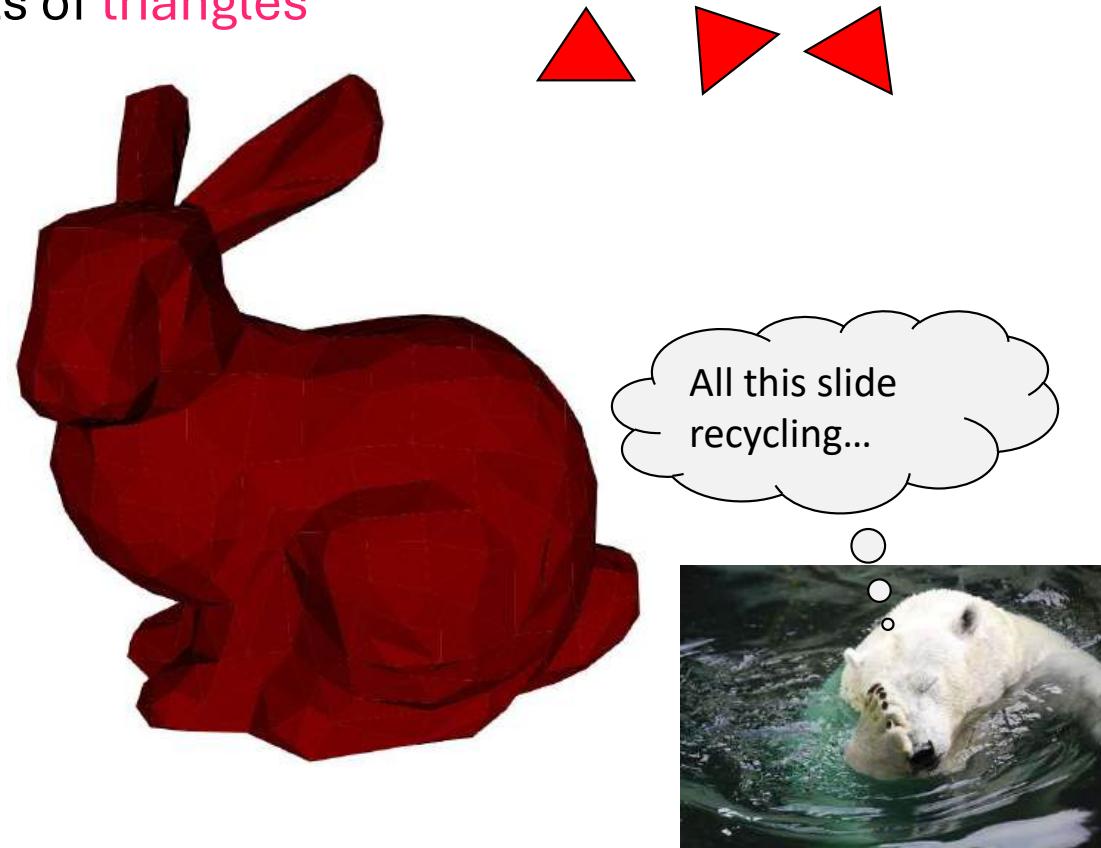


## How to accelerate the process?



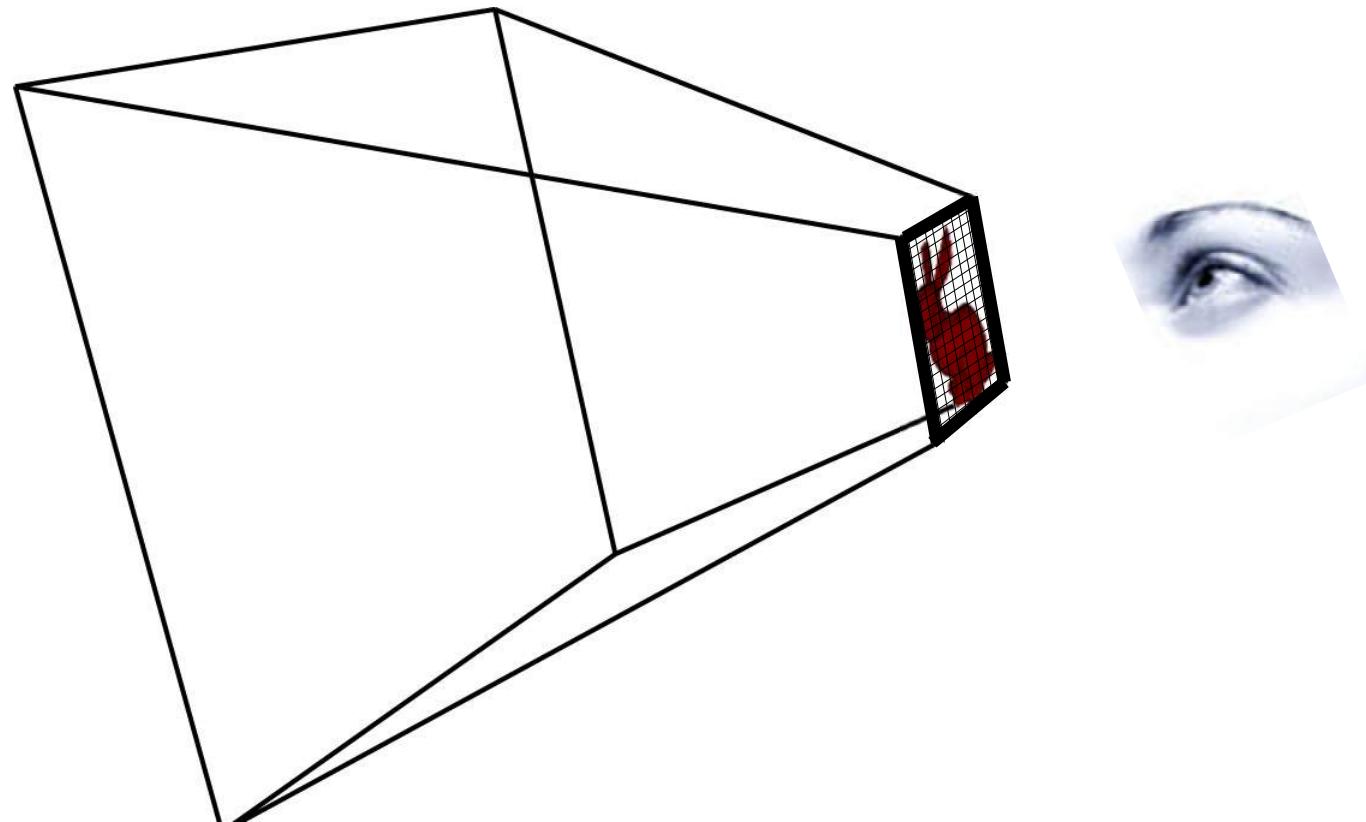
## Simplified Graphics Pipeline

- Models are typically lists of triangles



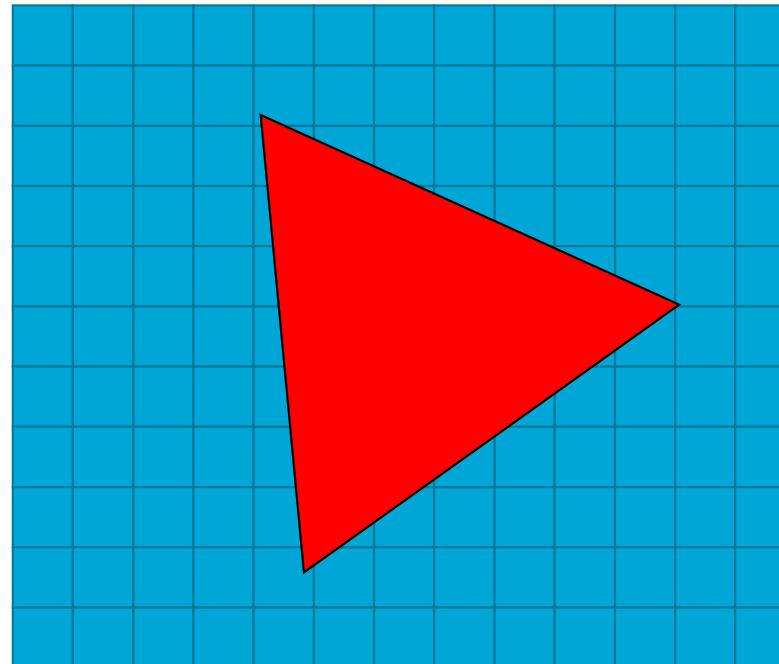
# Simplified Graphics Pipeline

- **Projection:** Transform coordinates to screen



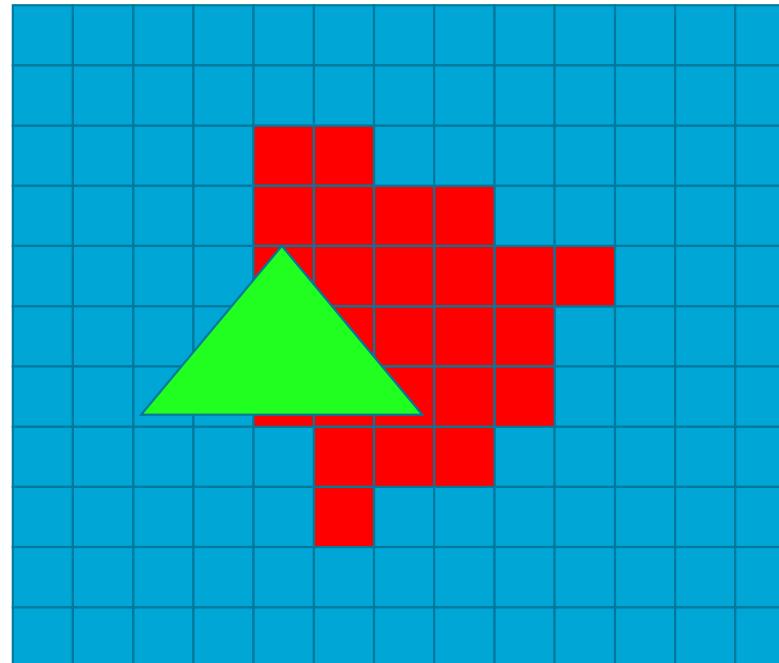
## Simplified Graphics Pipeline

- Rasterization: Fill screen pixels



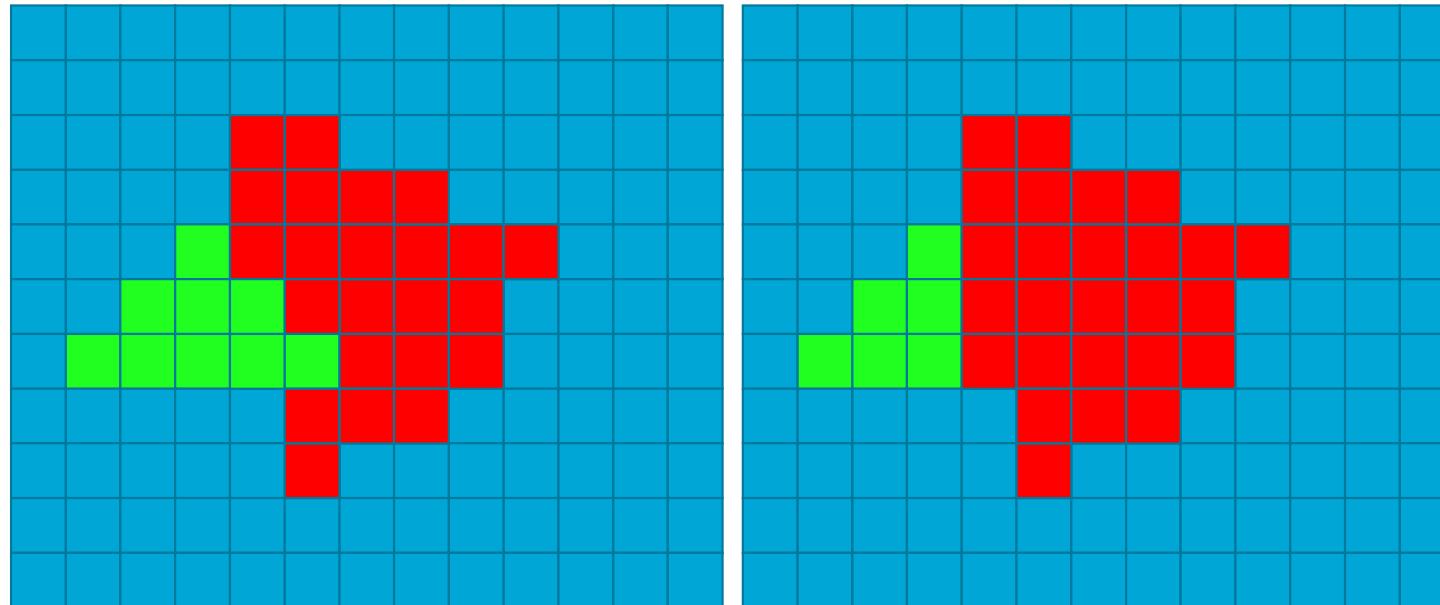
# Simplified Graphics Pipeline

- **Catch:** Let's look at a second triangle...



# Simplified Graphics Pipeline

- **Catch:** Drawing order changes result



Need to **keep nearest** pixels

# Simplified Graphics Pipeline

[Catmull74] , [Strasser74]

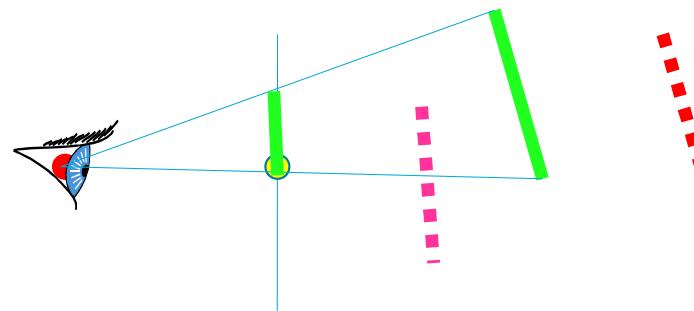
- Depth Buffer: Avoid sorting triangles!
- Store a color and depth in each pixel



# Simplified Graphics Pipeline

[Catmull74] , [Strasser74]

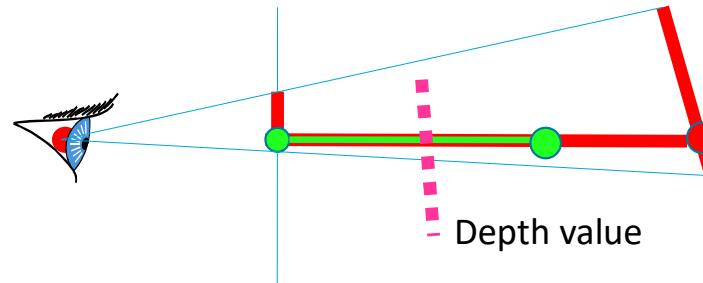
- Depth Buffer: Avoid sorting triangles!
- Store a color and depth in each pixel



# Simplified Graphics Pipeline

[Catmull74] , [Strasser74]

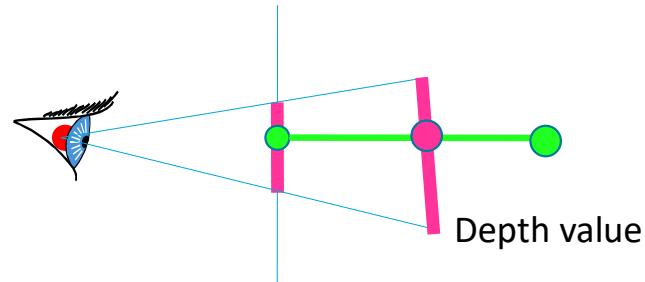
- Depth Buffer: Avoid sorting triangles!
- Store a color and depth in each pixel



# Simplified Graphics Pipeline

[Catmull74] , [Strasser74]

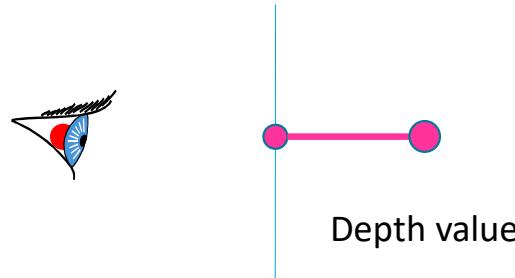
- Depth Buffer: Avoid sorting triangles!
- Store a color and depth in each pixel



# Simplified Graphics Pipeline

[Catmull74] , [Strasser74]

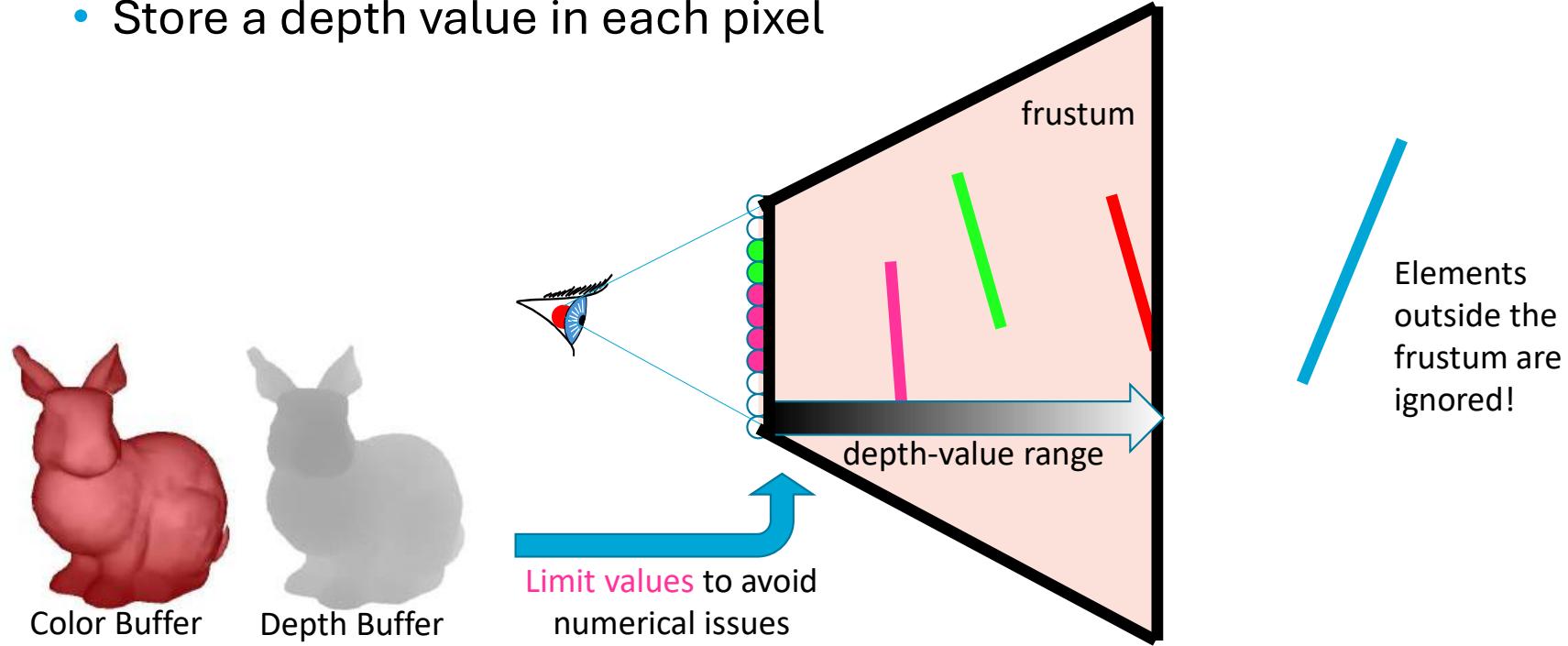
- Depth Buffer: Avoid sorting triangles!
- Store a color and depth in each pixel



# Simplified Graphics Pipeline

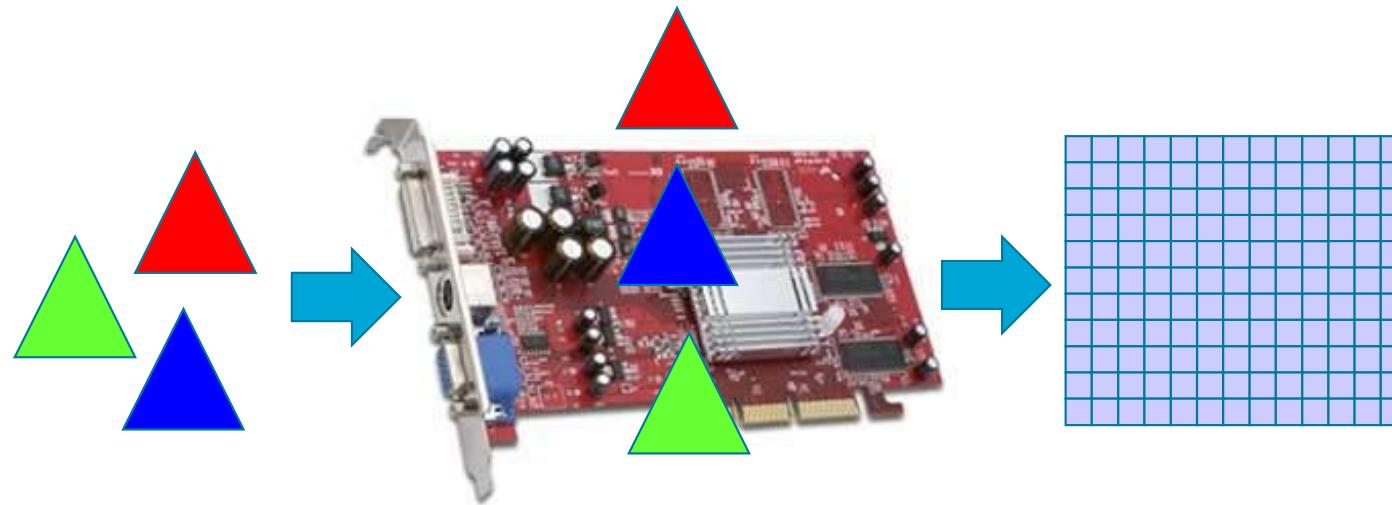
[Catmull74] , [Strasser74]

- Depth Buffer: Avoid sorting triangles!
- Store a depth value in each pixel



## Simplified Graphics Pipeline

- Highly parallelizable  
→ Graphics Processing Units (GPUs)

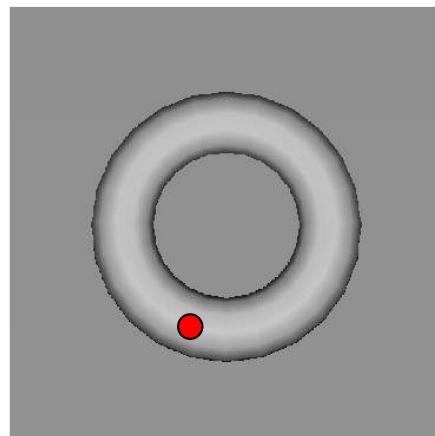
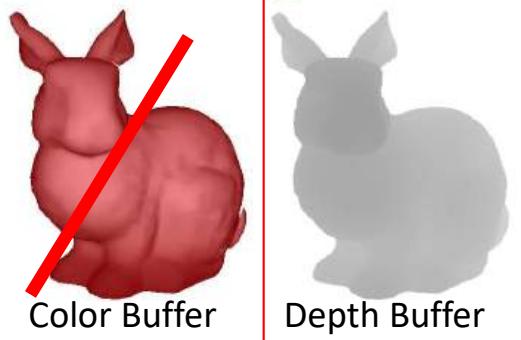


## Simplified Graphics Pipeline

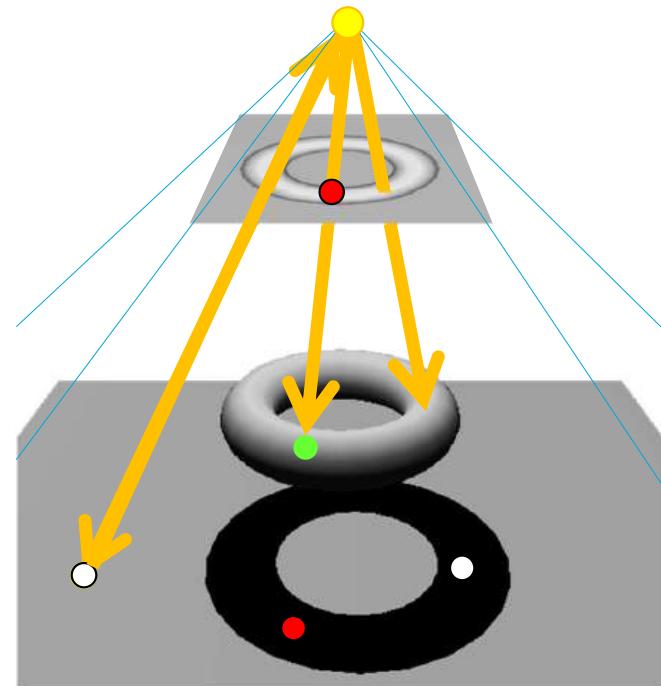


- Local computations:
- Processor only knows its current triangle  
this is NOT enough for shadows

## Shadow Mapping [Williams78]



1. render view from light  
use depth buffer



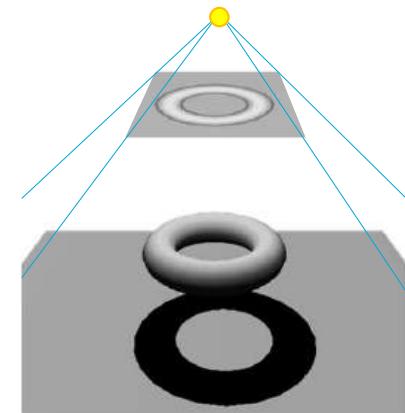
2. render from viewpoint

## Shadow Mapping [Williams78]

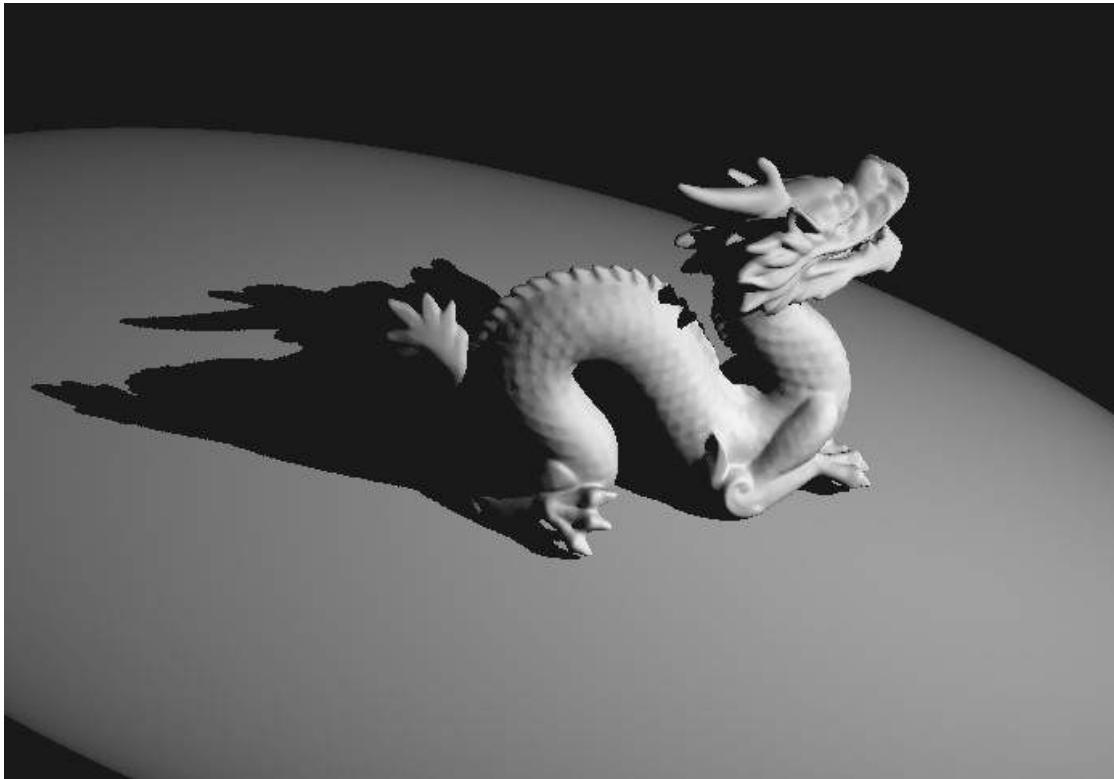
1. Render depth buffer from light (**Shadow Map**)
  - Store result in a texture

## 2. Render from viewpoint

- For each drawn pixel:  
Compare its depth in the light's view  
to the stored depth at this texel location  
in the Shadow Map
  - Equal: pixel is lit
  - Farther: pixel is in shadow



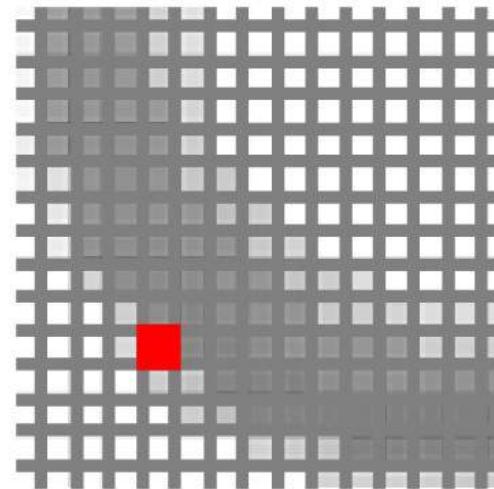
## Demo



## Problem 1: Discretization



from viewpoint



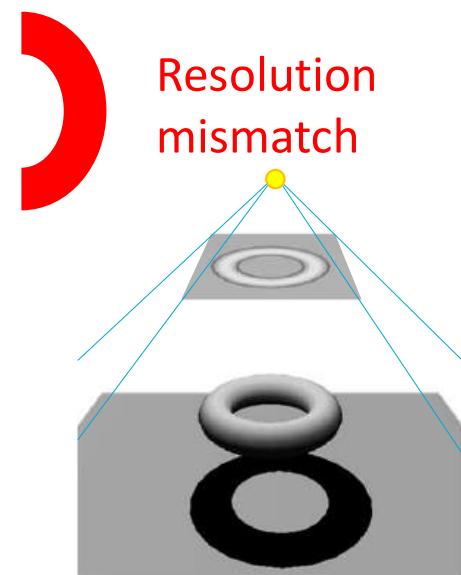
from light

## Shadow Mapping [Williams78]

1. Render depth buffer from light (**Shadow Map**)
  - Store result in a texture

### 2. Render from **viewpoint**

- For each drawn pixel:  
Compare its depth in the **light view**  
to the stored depth at the texel location  
in the Shadow Map
  - Equal: pixel is lit
  - Farther: pixel is in shadow



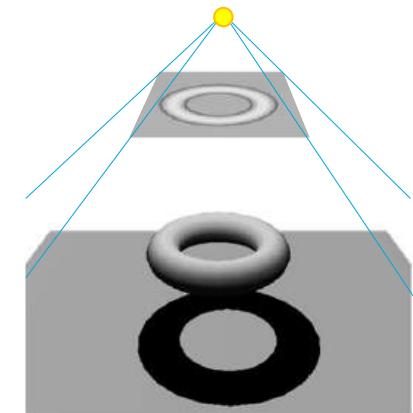
CGV

## Shadow Mapping [Williams78]

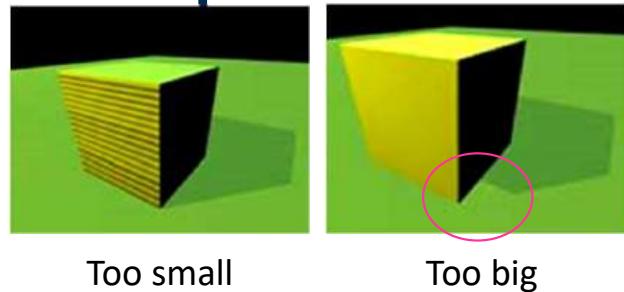
1. Render depth buffer from light (**Shadow Map**)
  - Store result in a texture

## 2. Render from viewpoint

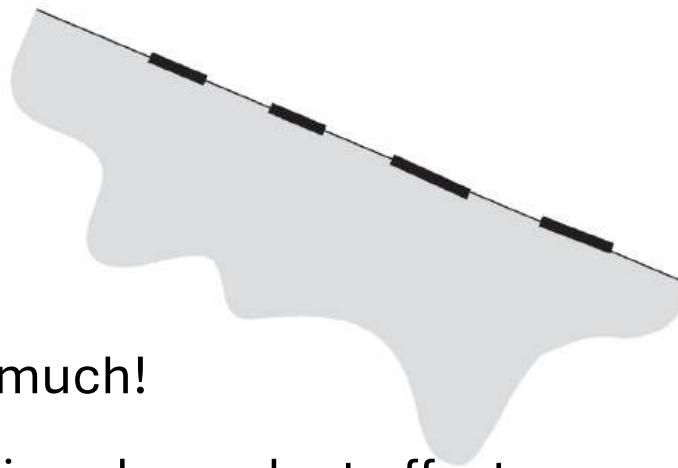
2. Render from viewpoint
  - For each drawn pixel:  
Compare its depth in the light view  
to the stored depth at the texel location  
in the Shadow Map
    - Equal: pixel is lit
    - Farther: pixel is in shadow



## Problem 2: Depth Bias



- Self-shadowing
  - Discretisation
  - Limited precision
- Solution:  
add an offset, but not too much!
- OpenGL supports orientation-dependent offsets



## Problem 2: Depth Bias

- “Real-world” example in Crysis by Crytek

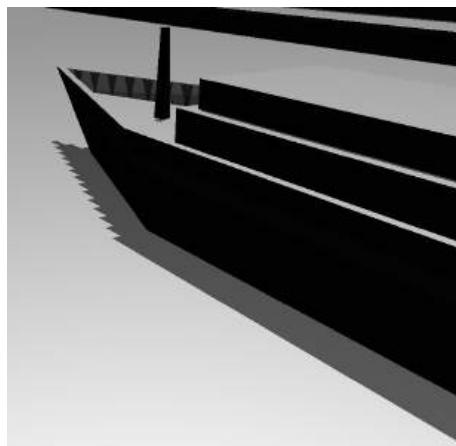


## Problem 3:

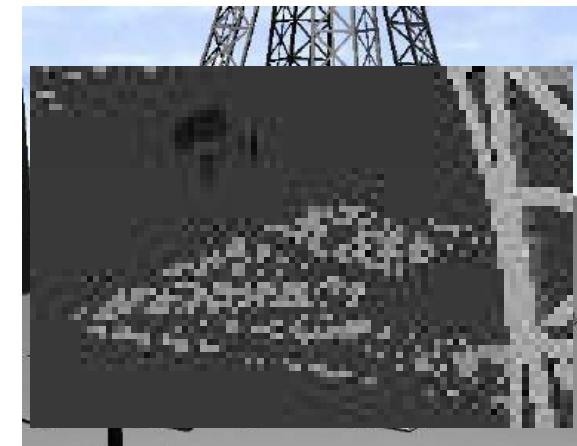
- Does this ring a bell?  
Unfortunately, no easy solution...

Reconstruction

- Staircase artifacts

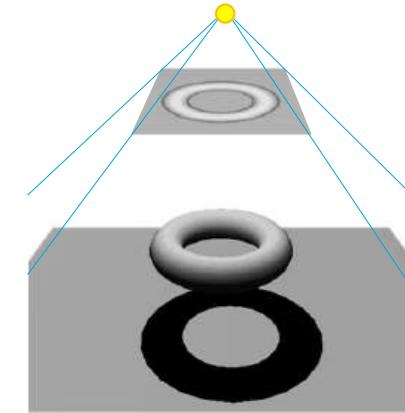


Undersampling  
of the shadow map



## Conclusion: Shadow Mapping [Williams78]

- Simple, efficient, and easy to implement
- Compatible to most object representations
- Additional hardware support
- Variants are common
  - (games, movies...)



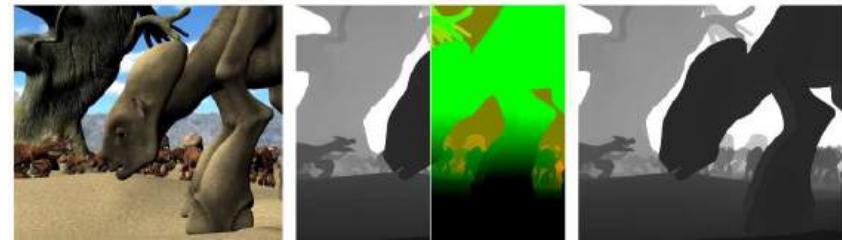
# Shadow Maps have a widespread use!

## Light Design



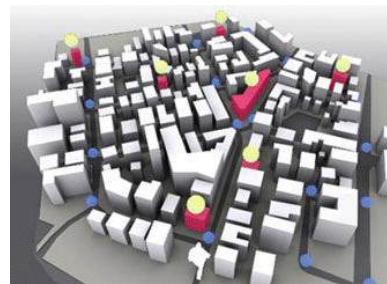
Schwaerzler et al. VIS2020

## Real-time Visibility Testing of Objects



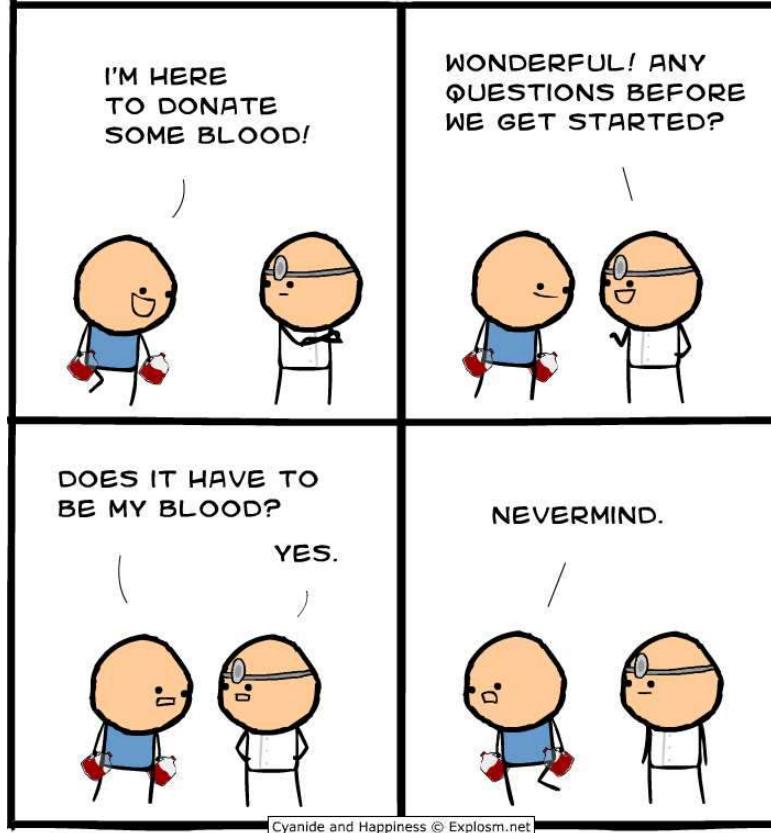
Lee et al. ToG2019

## Urban Design



many many more...

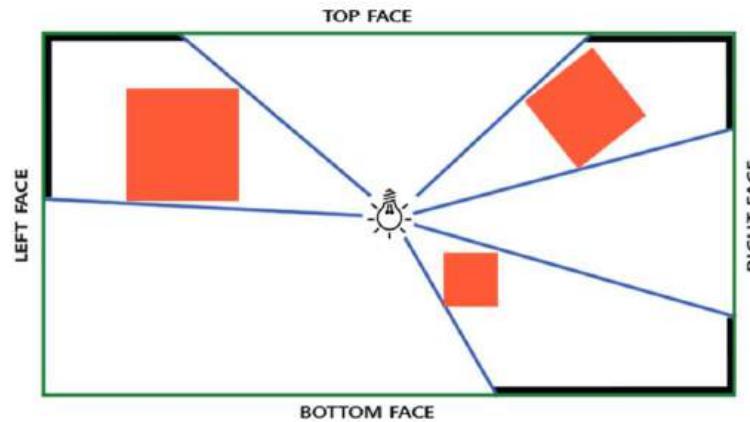
## Questions?



Talking about questions:

Consider submitting a potential exam question! See Brightspace for details! | 

## Exercise: How to make an omnidirectional Shadow Map?



## Answer: Environment Mapping

- Render a shadow map for each view.
- Could use Geometry Shader to avoid 6 loops

Cube mapping

