LU Factorization + Graphics

Geometric Algorithms
Lecture 14

CAS CS 132

Introduction

Recap Problem

Consider the matrix

$$A = \begin{bmatrix} -1 & -4 & 2 \\ -1 & -4 & 1 \\ 2 & 8 & 1 \end{bmatrix}$$

Find a general form solution for the equation $A\mathbf{x} = \mathbf{0}$.

Answer

$$\begin{bmatrix} 1 & 4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -4 & 2 & 0 \\ -1 & -4 & 1 & 0 \\ 2 & 8 & 1 & 0 \end{bmatrix}$$

step 1: build the augmented
matrix for this equation

Answer

$$\begin{bmatrix} 1 & 4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -4 & 2 & 0 \\ -1 & -4 & 1 & 0 \\ 2 & 8 & 1 & 0 \end{bmatrix}$$

step 2: convert to reduce echelon form

Answer

$$\begin{bmatrix} 1 & 4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -4 & 2 & 0 \\ 0 & 0 & -1 & 0 \\ 2 & 8 & 1 & 0 \end{bmatrix}$$

$$R_2 \leftarrow R_2 - R_1$$

Answer

$$\begin{bmatrix} 1 & 4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -4 & 2 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R_3 \leftarrow R_3 + 2R_1$$

Answer

$$\begin{bmatrix} 1 & 4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -4 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R_1 \leftarrow R_1 + 2R_2$$

Answer

$$\begin{bmatrix} 1 & 4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R_1 \leftarrow -R_1$$
$$R_2 \leftarrow -R_2$$

Answer

$$\begin{bmatrix} 1 & 4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{array}{cccc} x_1 & x_2 & x_3 & \\ \end{array}$$
$$\begin{bmatrix} 1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

`step 3: find the pivot positions and determine what variables to make basic and free`

Answer

$$\begin{bmatrix} 1 & 4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$x_1 = -4x_2$

x_2 is free

$x_3 = 0$

step 4: write down the general form solution by the procedure from Lecture 3

Objectives

1. Finish discussion of LU factorization, with an eye towards performance
2. Look at linear algebraic methods in graphics
3. Briefly discuss Homework 7

Keywords

elementary matrices

LU factorization

wireframe objects

homogeneous coordinates

translation

perspective projections

Recap

Recall: LU Factorization at a High Level

Given a $m \times n$ matrix A, we are going to factorize A as

Echelon form of A

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{U}$$

Recall: A New Perspective on Gaussian Elimination

The forward part of Gaussian elimination <u>is</u> matrix factorization

Recall: Elementary Matrices and Row Operations

Definition. An **elementary matrix** is a matrix obtained by applying a single row operation to the identity matrix I.

Example.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{R_2 \leftarrow R_2 + 3R_3} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

Fact. Any elementary row can be implemented by an elementary matrix.

Recall: Gaussian Elimination and Elementary Matrices

$$A \sim A_1 \sim A_2 \sim \dots \sim A_k$$

Consider a sequence of elementary row operations from A to an echelon form.

Each step can be represent as a **product with an elementary matrix.**

Recall: Gaussian Elimination and Elementary Matrices

$$A \sim E_1 A \sim E_2 E_1 A \sim \ldots \sim E_k E_{k-1} \ldots E_2 E_1 A$$

Recall: Gaussian Elimination and Elementary Matrices

$$A \sim E_1A \sim E_2E_1A \sim \ldots \sim E_kE_{k-1}\ldots E_2E_1A$$

This exactly tells us that if U is the final echelon form we get then

$$U = (E_kE_{k-1}\ldots E_2E_1)A = EA$$

where E implements a <u>sequence</u> of row operations.

Recall: Gaussian Elimination and Elementary Matrices

$$A \sim E_1 A \sim E_2 E_1 A \sim \ldots \sim E_k E_{k-1} \ldots E_2 E_1 A$$

This exactly tells us that if U is the final echelon form we get then

$$U = \underbrace{(E_k E_{k-1} \ldots E_2 E_1)}^{\text{Invertible}} A = EA$$

where E implements a <u>sequence</u> of row operations.

Recall: Gaussian Elimination and Elementary Matrices

$$A \sim E_1 A \sim E_2 E_1 A \sim \ldots \sim E_k E_{k-1} \ldots E_2 E_1 A$$

This exactly tells us that if U is the final echelon form we get then

$$U = \underbrace{(E_k E_{k-1} \ldots E_2 E_1)}^{\text{Invertible}} A = EA$$

where E implements a <u>sequence</u> of row operations.

So

$$A = E^{-1} U = (E_1^{-1} E_2^{-1} \ldots E_{k-1}^{-1} E_k^{-1}) U$$

Recall: Gaussian Elimination and Elementary Matrices

$$A \sim E_1 A \sim E_2 E_1 A \sim \ldots \sim E_k E_{k-1} \ldots E_2 E_1 A$$

This exactly tells us that if U is the final echelon form we get then

$$U = \underbrace{(E_k E_{k-1} \ldots E_2 E_1)}^{\text{Invertible}} A = EA$$

where E implements a <u>sequence</u> of row operations.

So
$$A = E^{-1} U = \overbrace{(E_1^{-1} E_2^{-1} \ldots E_{k-1}^{-1} E_k^{-1})}^{\text{"L" part}} U$$

LU Factorization Algorithm

```
FUNCTION LU_Factorization(A):
    L ← identity matrix
    U ← A
    convert U to an echelon form by GE forward step # without swaps
    FOR each row operation OP in the prev step:
        E ← the matrix implementing OP
        L ← L @ E^{-1}    # note the multiplication on the right
    RETURN (L, U)      this isn't actually how this implemented
```

demo

How To: LU Factorization by hand

Question. Find a LU Factorization for the matrix A (assuming no swaps).

Solution.

» Start with L as the identity matrix.
» Find U by the forward part of GE.
» For each operation $R_i \leftarrow R_i + kR_j$, set L_{ij} to $-k$.

Solving Systems using the LU Factorization

Connecting back to Matrix Equations

$$A\mathbf{x} = \mathbf{b}$$

Question. Solve the above matrix equation (in other words, find a general form solution).

Connecting back to Matrix Equations

$$A\mathbf{x} = \mathbf{b}$$

Question. Solve the above matrix equation (in other words, find a general form solution).

What does the LU factorization give us?

Connecting back to Matrix Equations

$$(LU)\mathbf{x} = \mathbf{b}$$

Question. Solve the above matrix equation (in other words, find a general form solution).

Substitute LU for A

Connecting back to Matrix Equations

$$L(U\mathbf{x}) = \mathbf{b}$$

Question. Solve the above matrix equation (in other words, find a general form solution).

Rearrange matrix-vector multiplications

Connecting back to Matrix Equations

$$U\mathbf{x} = L^{-1}\mathbf{b}$$

Question. Solve the above matrix equation (in other words, find a general form solution).

Multiply by L^{-1} on both sides

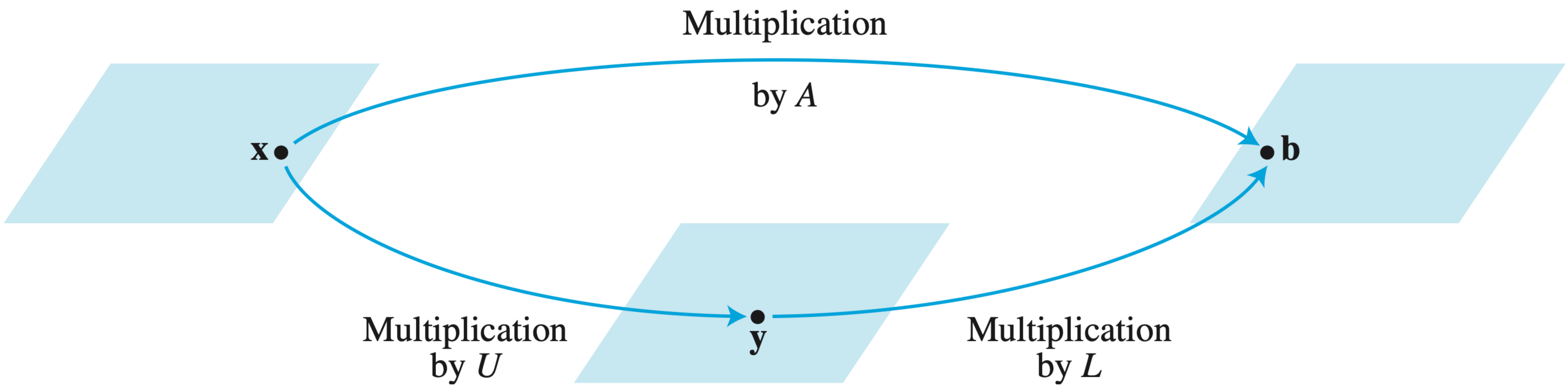
Connecting back to Matrix Equations

$$U\mathbf{x} = L^{-1}\mathbf{b}$$

Question. Solve the above matrix equation (in other words, find a general form solution).

A solution to $A\mathbf{x} = \mathbf{b}$ is the same as a solution to $U\mathbf{x} = L^{-1}\mathbf{b}$

Solving systems with the LU (Pictorially)



$$U\mathbf{x} = \mathbf{y} = L^{-1}\mathbf{b}$$

If A maps $\mathbf{x}$ to $\mathbf{b}$, then U maps $\mathbf{x}$ to some vector $\mathbf{y}$ which is mapped to $\mathbf{b}$ by L.

How To: Solving Systems with the LU

How To: Solving Systems with the LU

Question. Solve the equation $A\mathbf{x} = \mathbf{b}$ given that $A = LU$ is a LU factorization.

How To: Solving Systems with the LU

Question. Solve the equation $A\mathbf{x} = \mathbf{b}$ given that $A = LU$ is a LU factorization.

Solution.

How To: Solving Systems with the LU

Question. Solve the equation $A\mathbf{x} = \mathbf{b}$ given that $A = LU$ is a LU factorization.

Solution.

1. Solve $L\mathbf{x} = \mathbf{b}$ to get the unique solution $\mathbf{v} = L^{-1}\mathbf{b}$.

How To: Solving Systems with the LU

Question. Solve the equation $A\mathbf{x} = \mathbf{b}$ given that $A = LU$ is a LU factorization.

Solution.

1. Solve $L\mathbf{x} = \mathbf{b}$ to get the unique solution $\mathbf{v} = L^{-1}\mathbf{b}$.

2. Solve $U\mathbf{x} = \mathbf{v}$ to get a solution $\mathbf{w}$.

How To: Solving Systems with the LU

Question. Solve the equation $A\mathbf{x} = \mathbf{b}$ given that $A = LU$ is a LU factorization.

Solution.

1. Solve $L\mathbf{x} = \mathbf{b}$ to get the unique solution $\mathbf{v} = L^{-1}\mathbf{b}$.

2. Solve $U\mathbf{x} = \mathbf{v}$ to get a solution $\mathbf{w}$.

$\mathbf{w}$ is a solution to $A\mathbf{x} = \mathbf{b}$

How To: Solving Systems with the LU

Question. Solve the equation $A\mathbf{x} = \mathbf{b}$ given that $A = LU$ is a LU factorization.

Solution.

This is significantly faster than solving $A\mathbf{x} = \mathbf{b}$

1. Solve $L\mathbf{x} = \mathbf{b}$ to get the unique solution $\mathbf{v} = L^{-1}\mathbf{b}$.
2. Solve $U\mathbf{x} = \mathbf{v}$ to get a solution $\mathbf{w}$.

$\mathbf{w}$ is a solution to $A\mathbf{x} = \mathbf{b}$

FLOPs for Gaussian Elimination

Given an $n \times n$ matrix, we have the following FLOP estimates:

» Gaussian Elimination: $\sim \frac{2n^3}{3}$ FLOPS

» GE Forward: $\sim \frac{2n^3}{3}$ FLOPS

» GE Backward: $\sim n^2$ FLOPS

FLOPs for Gaussian Elimination

Given an $n \times n$ matrix, we have the following FLOP estimates:

- Gaussian Elimination: $\sim \dfrac{2n^3}{3}$ FLOPS
- GE Forward: $\boxed{\sim \dfrac{2n^3}{3} \text{ FLOPS}}$ dominant term
- GE Backward: $\sim n^2$ FLOPS

FLOPs for Gaussian Elimination

Given an $n \times n$ matrix, we have the following FLOP estimates:

» Gaussian Elimination: $\sim \dfrac{2n^3}{3}$ FLOPS

» GE Forward: $\boxed{\sim \dfrac{2n^3}{3} \text{ FLOPS}}$ dominant term

» GE Backward: $\sim n^2$ FLOPS

Solving $A\mathbf{x} = \mathbf{b}$ **takes** $\sim \dfrac{2}{3}n^3$ **FLOPS**

FLOPS for $L\mathbf{x} = \mathbf{b}$

L is a **lower triangular** matrix. The system can be solved in $\sim n^2$ FLOPS by forward substitution.

$$\begin{bmatrix} 1 & 0 & 0 \\ a_{21} & 1 & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$x_1 = b_1$$
$$x_2 = b_2 - a_{21}x_1$$
$$x_3 = b_3 - a_{31}x_1 - a_{32}x_2$$

FLOPS for $U\mathbf{x} = \mathbf{v}$

U is in *echelon form*. We only need to perform back substitution, which can be done in $\sim n^2$ FLOPS.

$$\begin{bmatrix} \blacksquare & * & * & * & * & | \\ 0 & \blacksquare & * & * & * & \mathbf{v} \\ 0 & 0 & 0 & \blacksquare & * & \\ 0 & 0 & 0 & 0 & 0 & | \end{bmatrix} \xrightarrow{\text{back substitution}} \begin{bmatrix} 1 & 0 & * & 0 & * & | \\ 0 & 1 & * & 0 & * & \mathbf{w} \\ 0 & 0 & 0 & 1 & * & \\ 0 & 0 & 0 & 0 & 0 & | \end{bmatrix}$$

FLOPS for solving LU systems

» LU Factorization: $\sim \dfrac{2n^3}{3}$ FLOPS

» Solving $L\mathbf{x} = \mathbf{b}$: $\sim n^2$ FLOPS (by "forward" elimination)

» Solving $U\mathbf{x} = \mathbf{c}$: $\sim n^2$ FLOPS (already in echelon form)

LU Factorization: $\sim \dfrac{2n^3}{3}$ FLOPS

FLOPS for solving LU systems

» LU Factorization: $\sim \frac{2n^3}{3}$ FLOPS dominant term

» Solving $L\mathbf{x} = \mathbf{b}$: $\sim n^2$ FLOPS (by "forward" elimination)

» Solving $U\mathbf{x} = \mathbf{c}$: $\sim n^2$ FLOPS (already in echelon form)

LU Factorization: $\sim \frac{2n^3}{3}$ FLOPS

FLOPS for Matrix Inverse

After we find A^{-1}, finding the solution $A^{-1}\mathbf{b}$ is the cost of matrix-vector multiplication.

Matrix Inversion: $\sim 2n^3$ FLOPS

Matrix-Vector Multiplication: $\sim 2n^2$ FLOPS

Matrix inversion: $\sim 2n^3$ FLOPS

FLOPS for Matrix Inverse

After we find A^{-1}, finding the solution $A^{-1}\mathbf{b}$ is the cost of matrix-vector multiplication.

Matrix Inversion: $\sim 2n^3$ FLOPS dominant term

Matrix-Vector Multiplication: $\sim 2n^2$ FLOPS

Matrix inversion: $\sim 2n^3$ FLOPS

FLOP Comparison

	Preprocessing	Solving
Gaussian Elimination	0	$\sim \frac{2}{3}n^3$
Matrix Inversion	$\sim 2n^3$	$\sim 2n^2$
LU Factorization	$\sim \frac{2}{3}n^3$	$\sim 2n^2$

If you solve several matrix equations for the same matrix, **LU factorization** is <u>faster</u> than **matrix inversion** on the *first* equation, and the same (in the worst case) in later equation.

Another Consideration: Density

Another Consideration: Density

A matrix is **sparse** if it has mostly zeros.

Another Consideration: Density

A matrix is **sparse** if it has mostly zeros.

If A is sparse, then L and U probably are too.

Another Consideration: Density

A matrix is **sparse** if it has mostly zeros.

If A is sparse, then L and U probably are too.

But A^{-1} may have ***many*** nonzero entries (in other words, A^{-1} is **dense**)

Another Consideration: Density

A matrix is **sparse** if it has mostly zeros.

If A is sparse, then L and U probably are too.

But A^{-1} may have ***many*** nonzero entries (in other words, A^{-1} is **dense**)

Sparse matrices are faster to compute with and better with respect to storage.

(switching gears...)

Graphics

Disclaimer

`I am not an expert in this field.`

Motivation (or Pretty Pictures)

Graphics doesn't need much motivation.

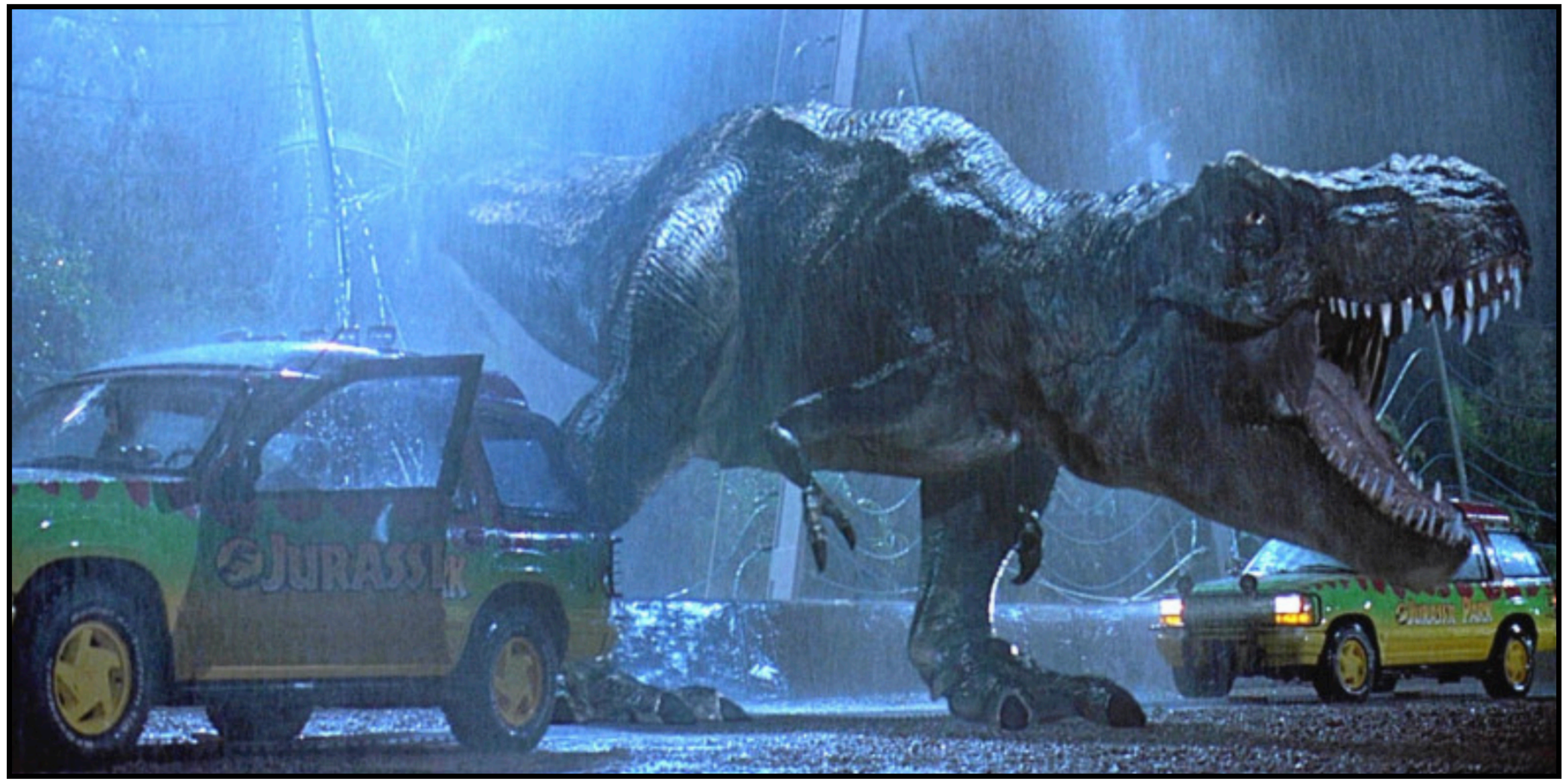
We spend so much time interacting graphics in one form or another.

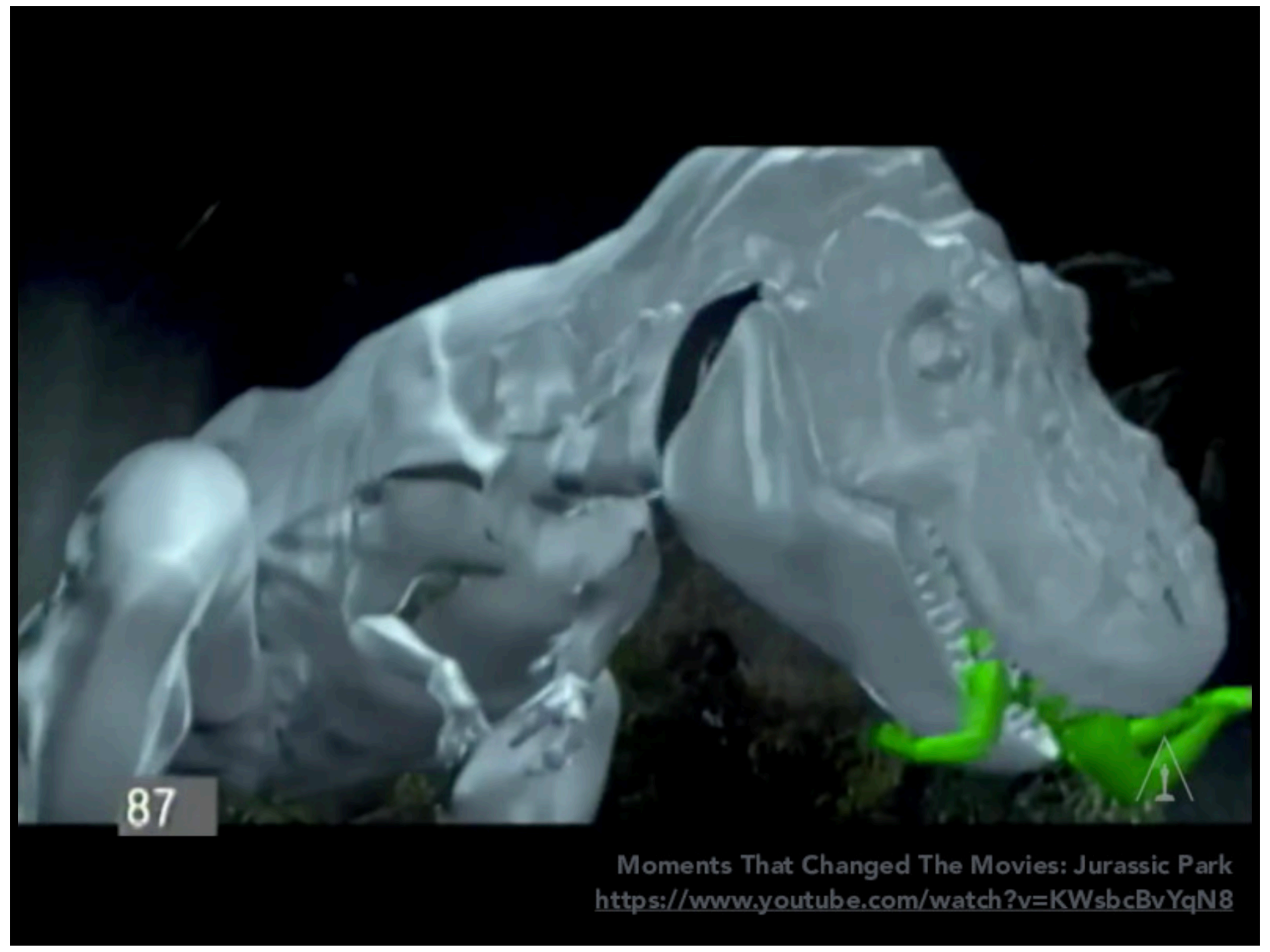
But in case you haven't thought too much about it, some examples...

source: CS184 Lecture Slides, UC Berkeley, Ng Ren

Movies

Jurassic Park (1993)





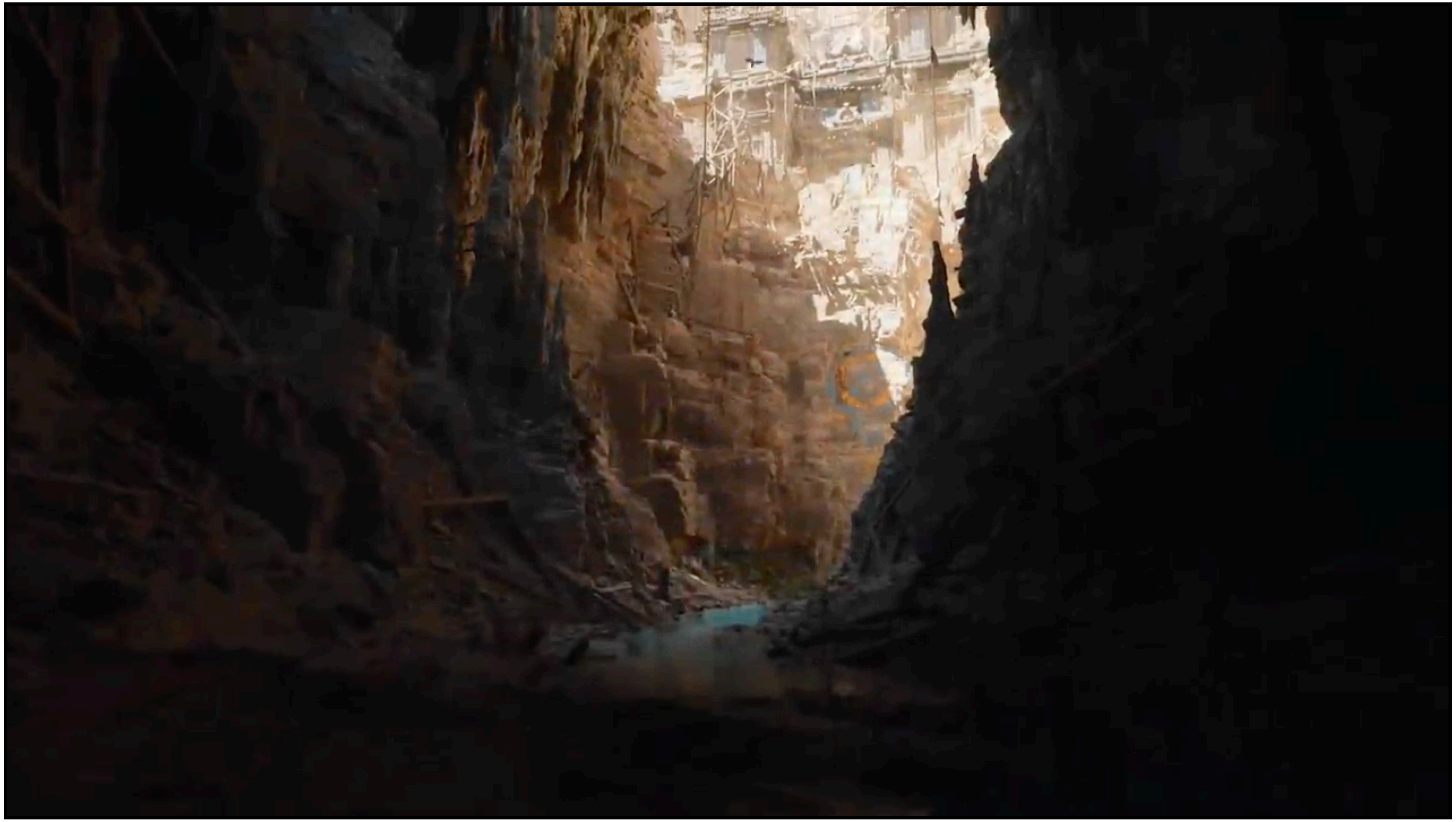
Alice in Wonderland (2010)

Motion Capture

Two Towers (2002)

Video Games

Unreal Engine 5 (2020)

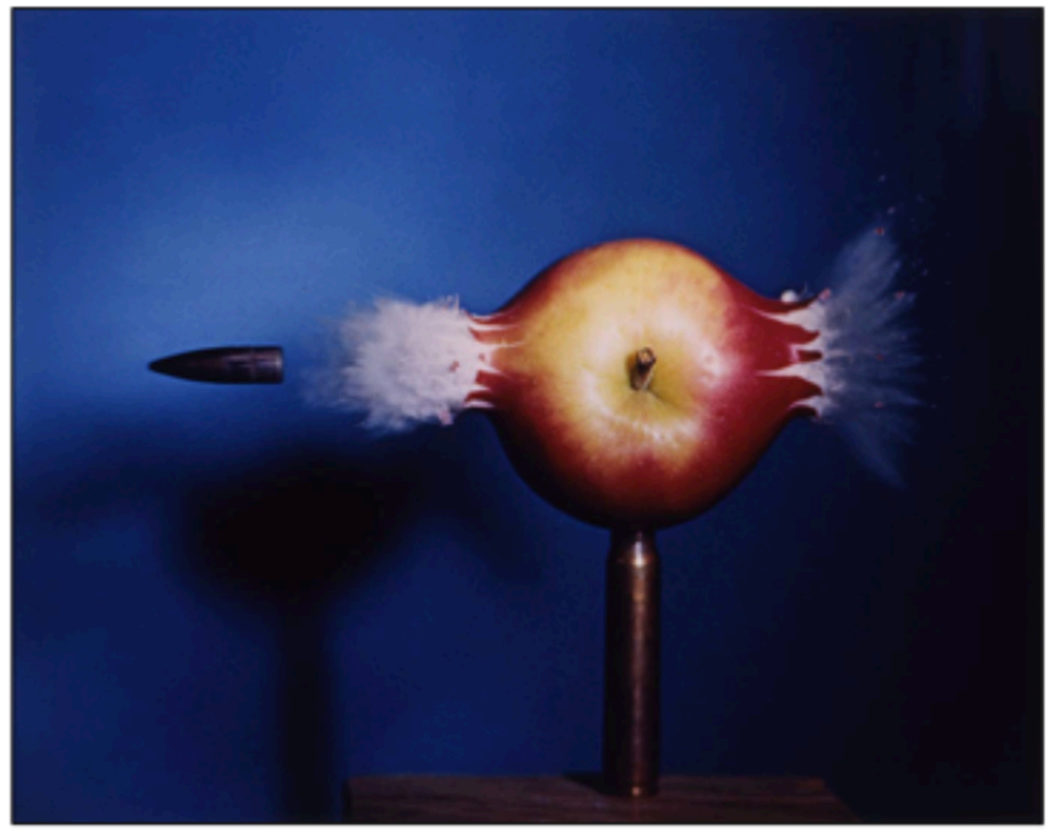


Scientific Visualization

First image of a black hole (2022)

Photography



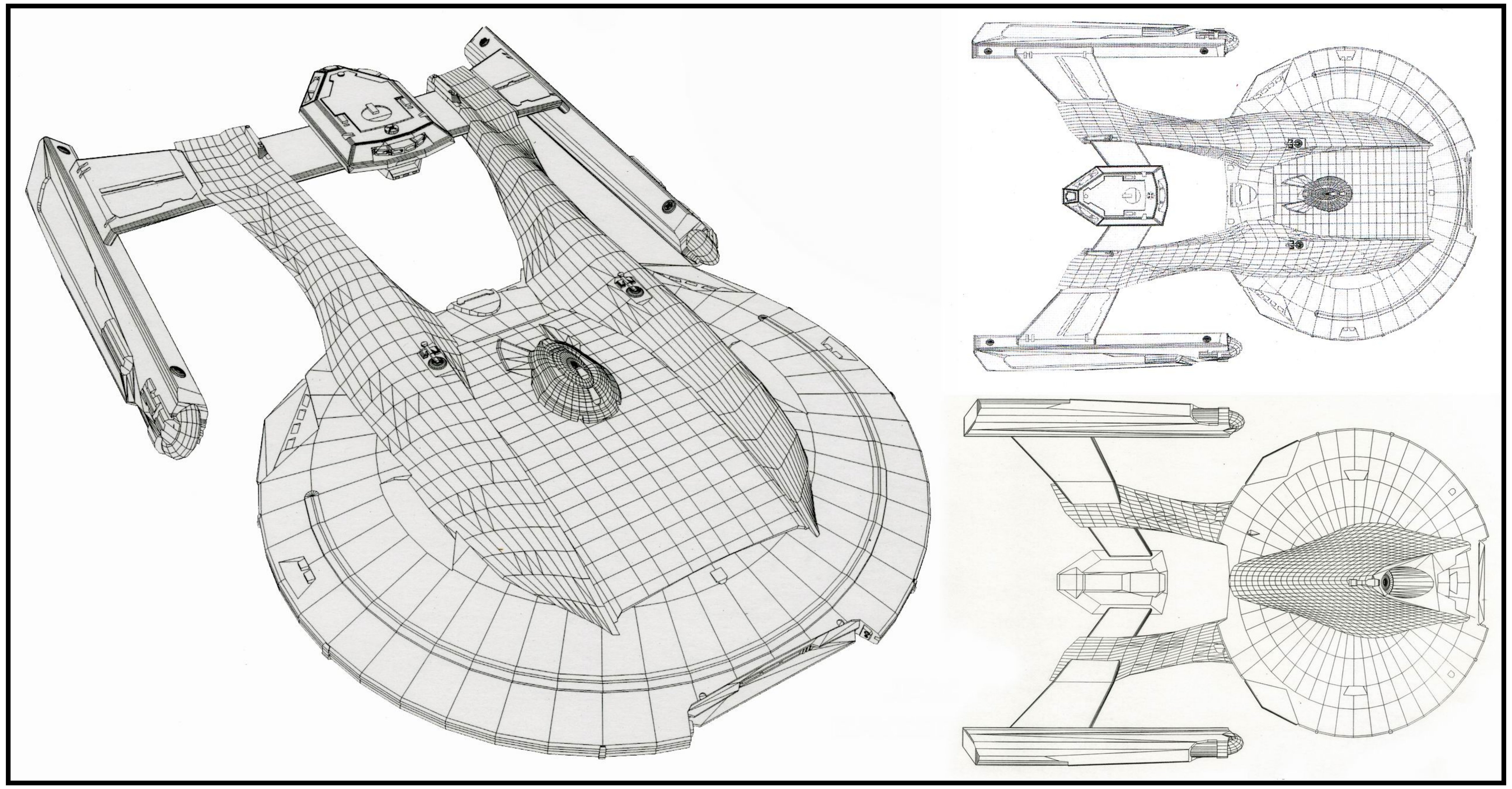
NASA | Walter Iooss | Steve McCurry
Harold Edgerton | NASA | National Geographic

Graphics and Linear Algebra

3D Graphics

There are many facets of computer graphics, but we will be focusing on one problem today:

Manipulating and Transforming 3D objects and rendering them on a screen.



3D Graphics Pipeline

3D Graphics Pipeline

1. Create a 3D model of objects + scene.

3D Graphics Pipeline

1. Create a 3D model of objects + scene.

2. Convert the surfaces of the objects in the model into approximations called **wire frames** or **tessellations** built out of a massive number of polygons (often triangles).

3D Graphics Pipeline

1. Create a 3D model of objects + scene.

2. Convert the surfaces of the objects in the model into approximations called **wire frames** or **tessellations** built out of a massive number of polygons (often triangles).

3. Manipulate the polygons via ***linear*** transformations and then ***linearly*** render it in 2D (in a way that preserves perspective).

3D Graphics Pipeline

1. Create a 3D model of objects + scene.

2. Convert the surfaces of the objects in the model into approximations called **wire frames** or **tessellations** built out of a massive number of polygons (often triangles).

3. Manipulate the polygons via ***linear*** transformations and then ***linearly*** render it in 2D (in a way that preserves perspective).

Today

Wire Frames

A **wire frame** is representation of a surface as a collection of polygons and line segments.

Transformations on line segments and polygons are **linear**.

https://commons.wikimedia.org/wiki/File:Wireframe_Render_of_Digital_Clothing_Bathingrobe_3D_Model.jpg

Transformations

We've seen many 2D transformations

» Reflections
» Expansion
» Shearing
» Projection

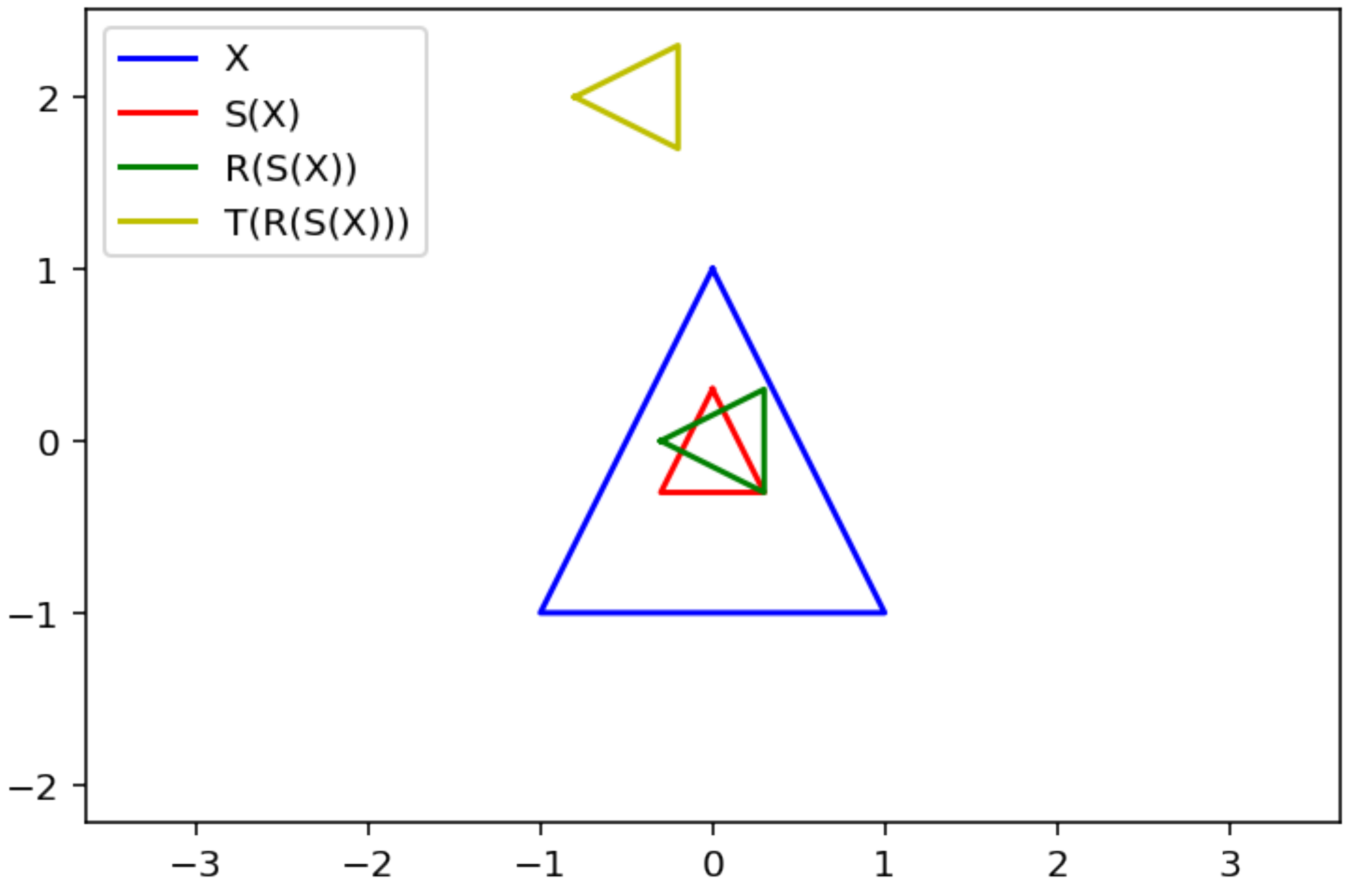
We've seen some 3D transformations

» Rotations
» Projections

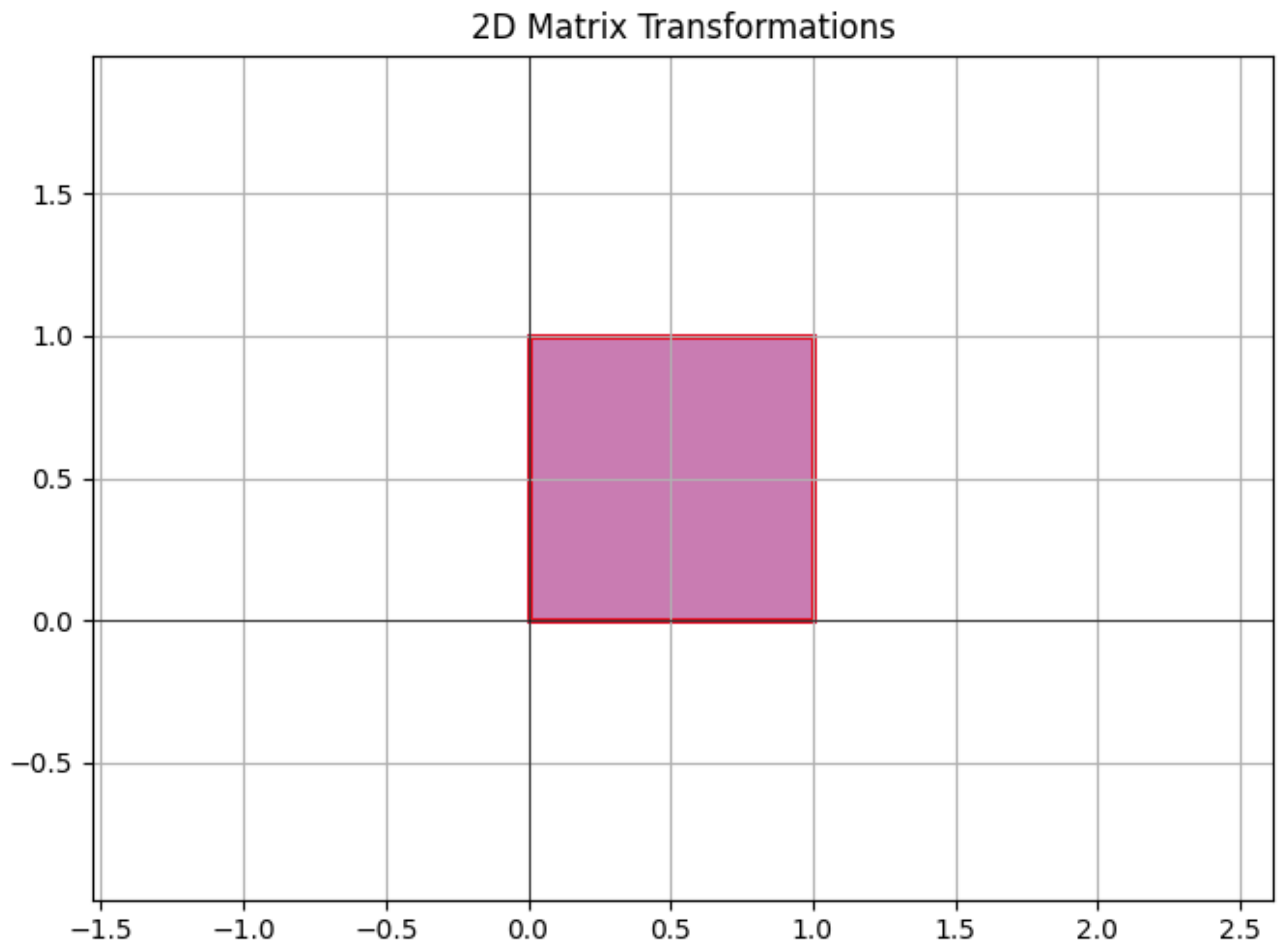
Composing Transformations

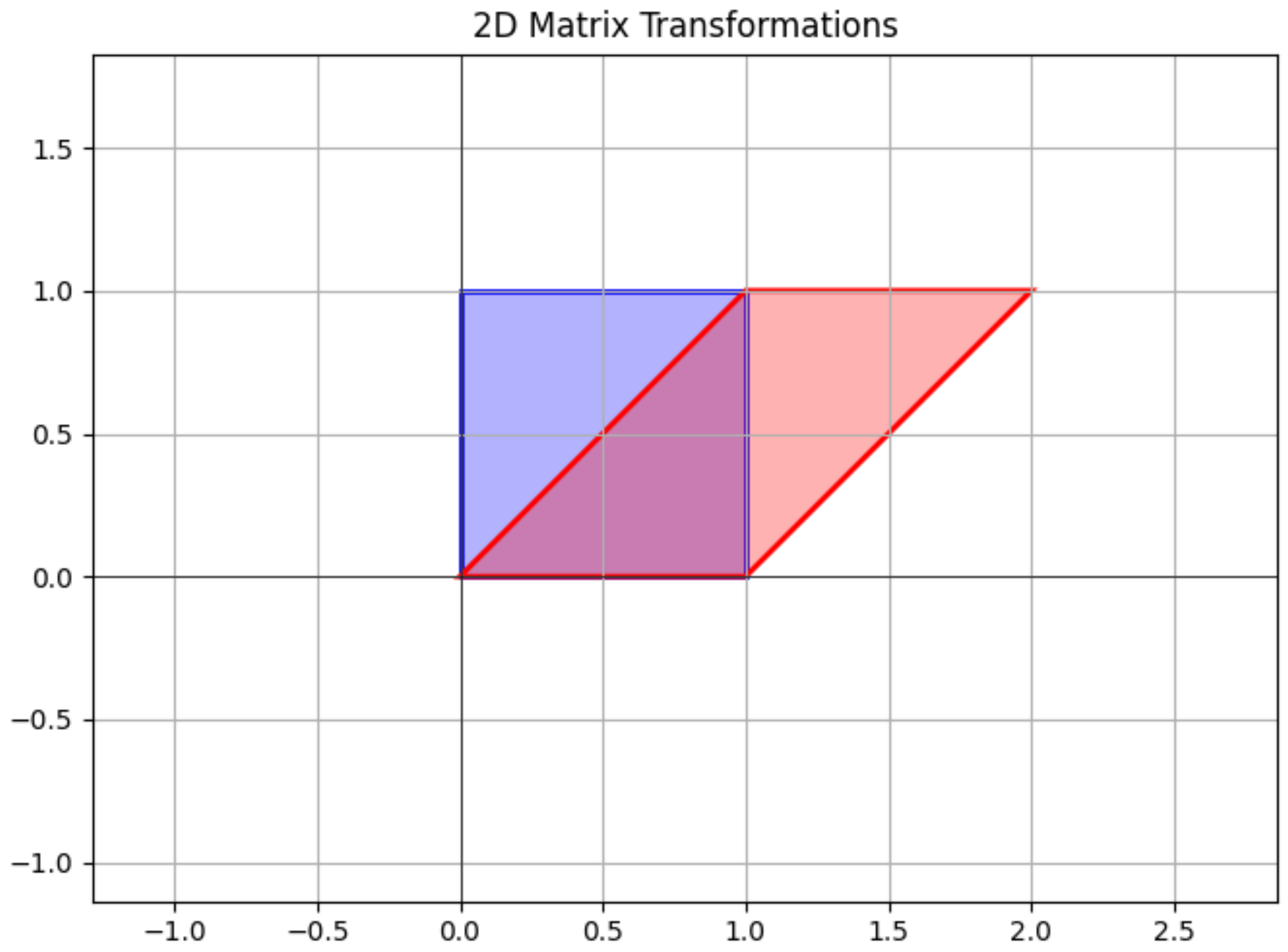
Recall. Multiplying matrices **composes** their associated transformations.

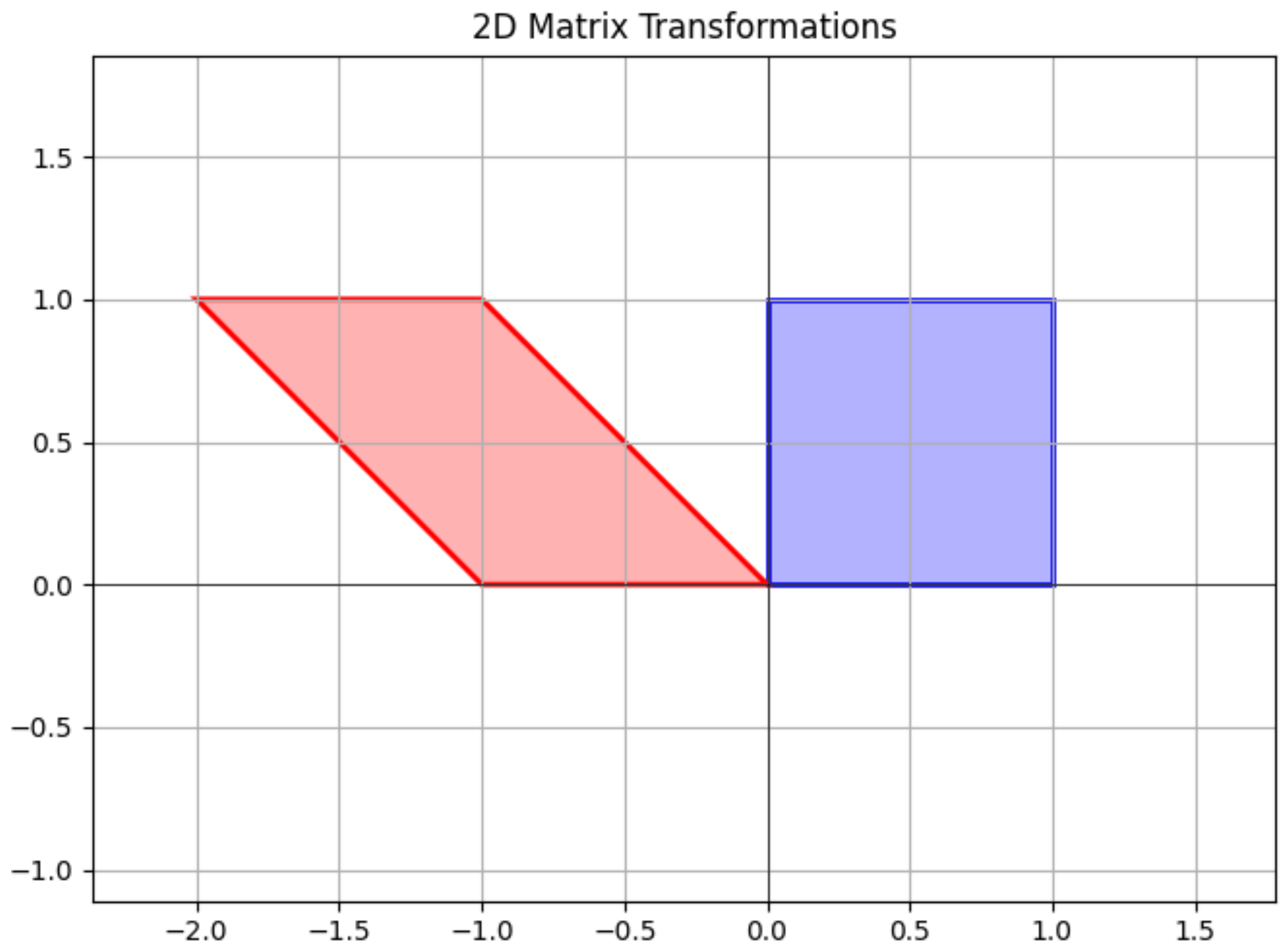
So complex graphical transformations can be combined into a single matrix.



Shearing and Reflecting (Geometrically)







shear

reflect

More Transformations

What we're adding today:

» More on rotations
» translations
» perspective projections

More Transformations

What we're adding today:

» More on rotations
» translations
» perspective projections

These aren't linear, but they are incredibly important so we have to address them.

3D Rotation Matrices

$$R_x^\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad R_y^\theta = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad R_z^\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3D Rotation Matrices

$$R_x^\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad R_y^\theta = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad R_z^\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

These are the matrices for counterclockwise rotation around x, y, and z axes.

3D Rotation Matrices

$$R_x^\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y^\theta = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z^\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

These are the matrices for counterclockwise rotation around x, y, and z axes.

(note the change in sign for y)

3D Rotation Matrices

$$R_x^\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad R_y^\theta = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad R_z^\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

These are the matrices for counterclockwise rotation around x, y, and z axes.

(note the change in sign for y)

Fact. Any rotation can be done by some matrix of the form

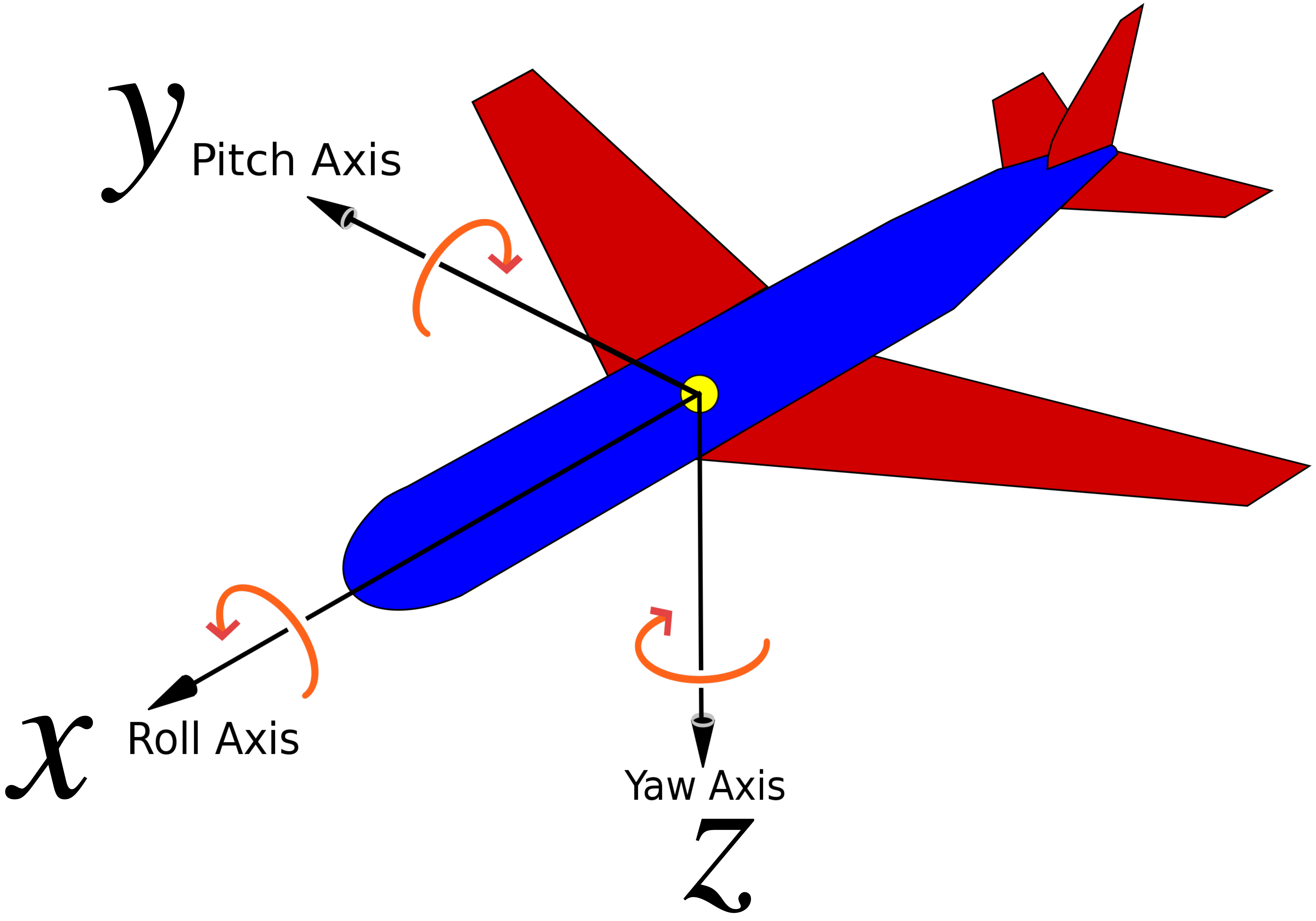
$$R_z^\theta R_y^\gamma R_x^\eta$$

Roll, Pitch and Yaw

roll changes the side-to-side tilt

pitch changes the up-down tilt

yaw changes direction



https://commons.wikimedia.org/wiki/File:Yaw_Axis_Corrected.svg

General Rotations

$$R_z^{\theta} R_y^{\gamma} R_x^{\eta}$$

yaw pitch roll

General Rotations

$$R_z^{\theta} R_y^{\gamma} R_x^{\eta}$$

yaw pitch roll

Exactly what rotation you get is not obvious (this a hard problem in control theory).

General Rotations

$$R_z^{\theta} R_y^{\gamma} R_x^{\eta}$$

yaw pitch roll

Exactly what rotation you get is not obvious (this a hard problem in control theory).

Remember. !!Matrix multiplication does not commute!!

General Rotations

$$\underbrace{R_z^\theta}_{\text{yaw}} \underbrace{R_y^\gamma}_{\text{pitch}} \underbrace{R_x^\eta}_{\text{roll}}$$

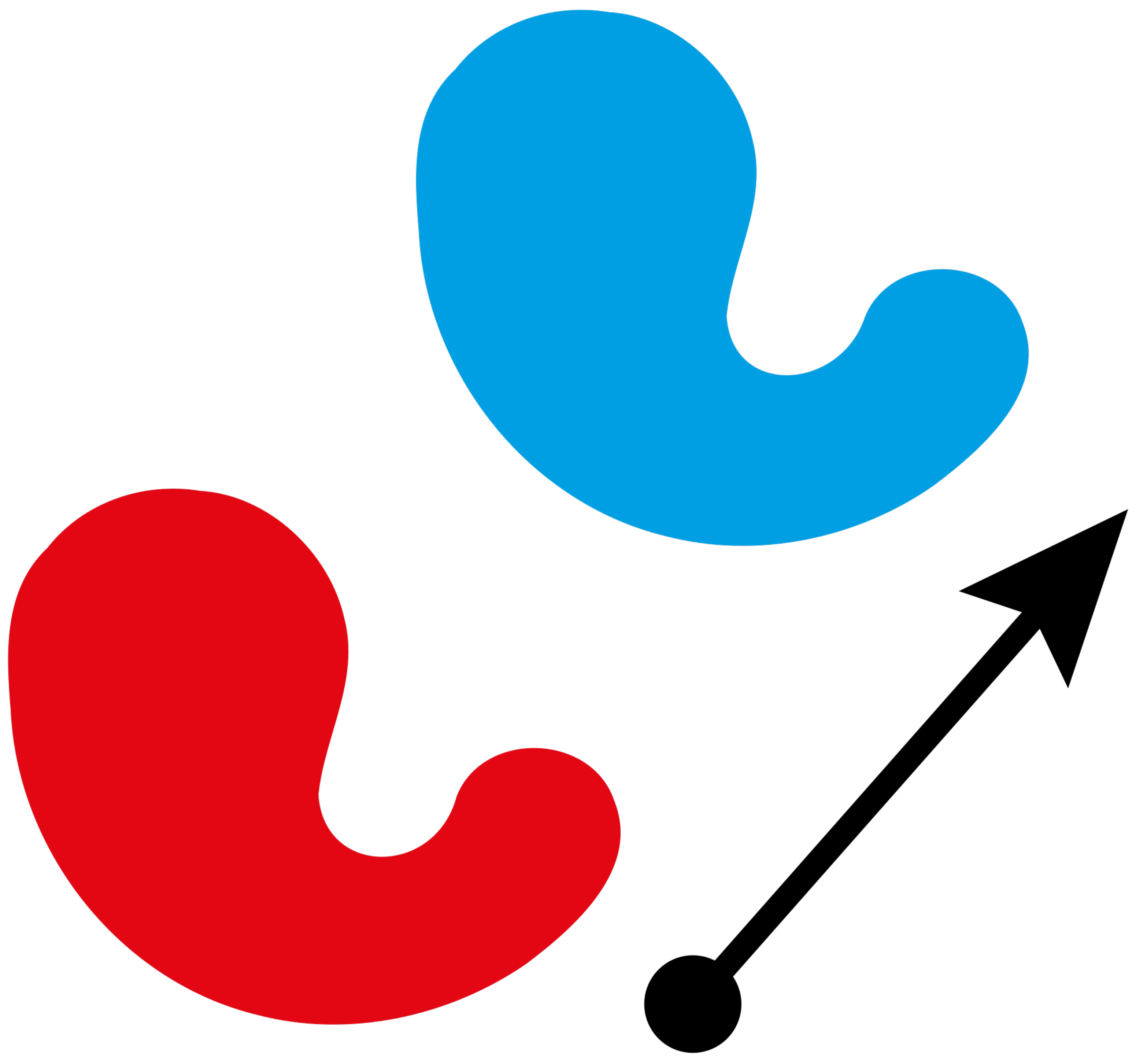
Exactly what rotation you get is not obvious (this a hard problem in control theory).

Remember. !!Matrix multiplication does not commute!!

So changing η above doesn't just rotate the object around the x–axis (that axis might be tilted along the pitch axis, for example).

demo

Translation

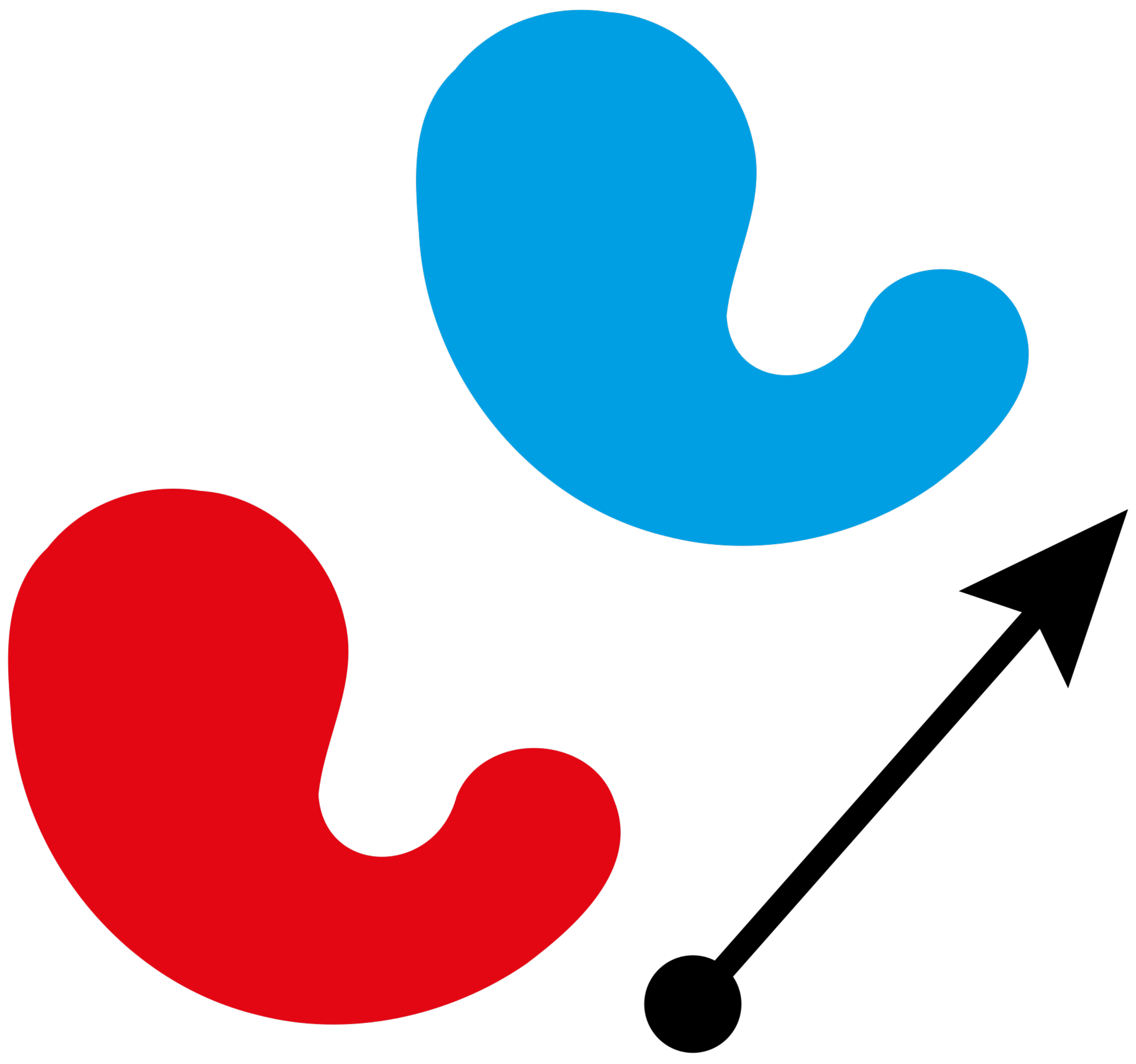


In 2D

https://commons.wikimedia.org/wiki/File:Traslazione_OK.svg

Translation

Given a vector $\mathbf{t}$ a **translation** is the transformation



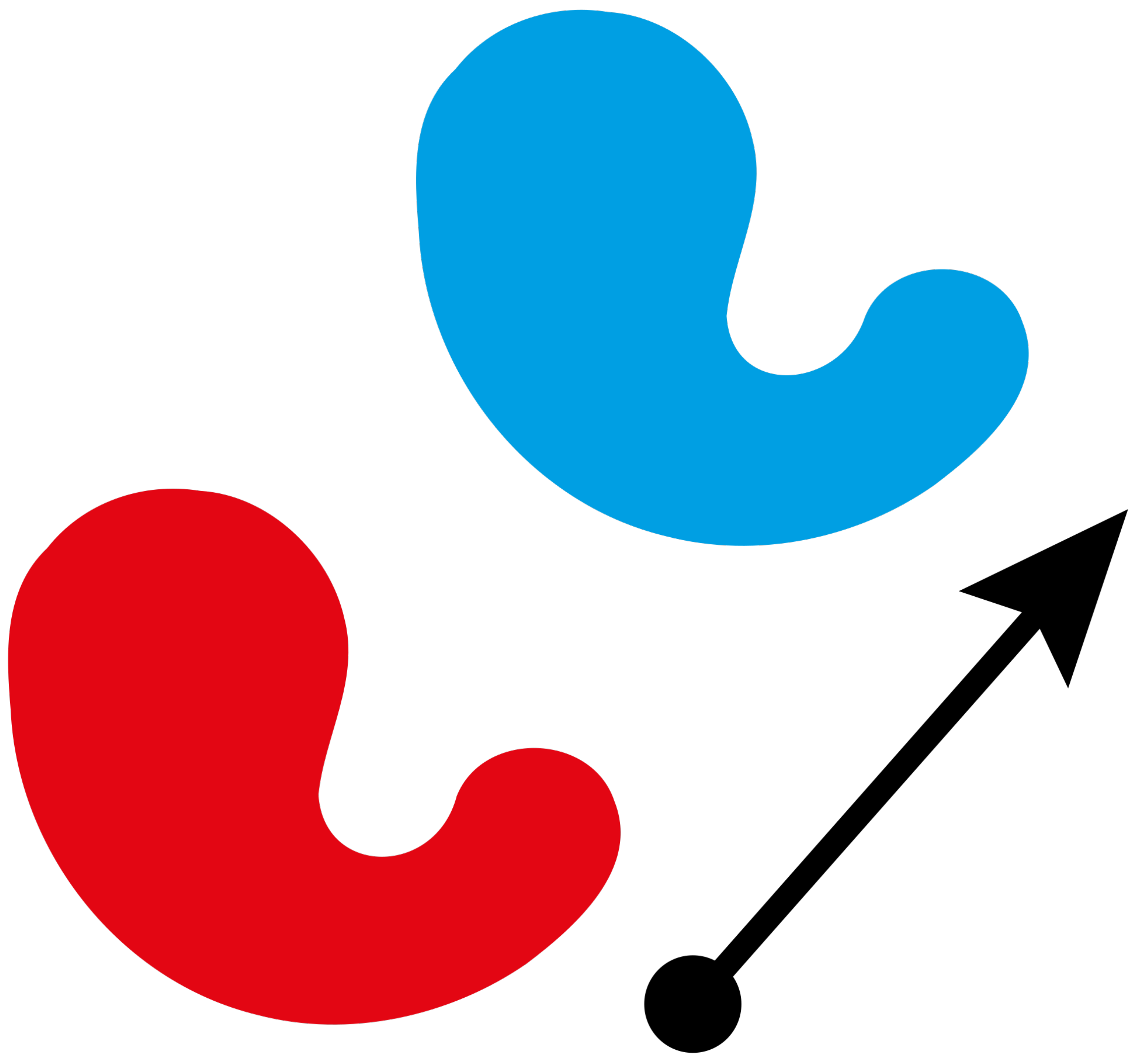
In 2D

https://commons.wikimedia.org/wiki/File:Traslazione_OK.svg

Translation

Given a vector $\mathbf{t}$ a **translation** is the transformation

$$T(\mathbf{x}) = \mathbf{x} + \mathbf{t}$$



In 2D

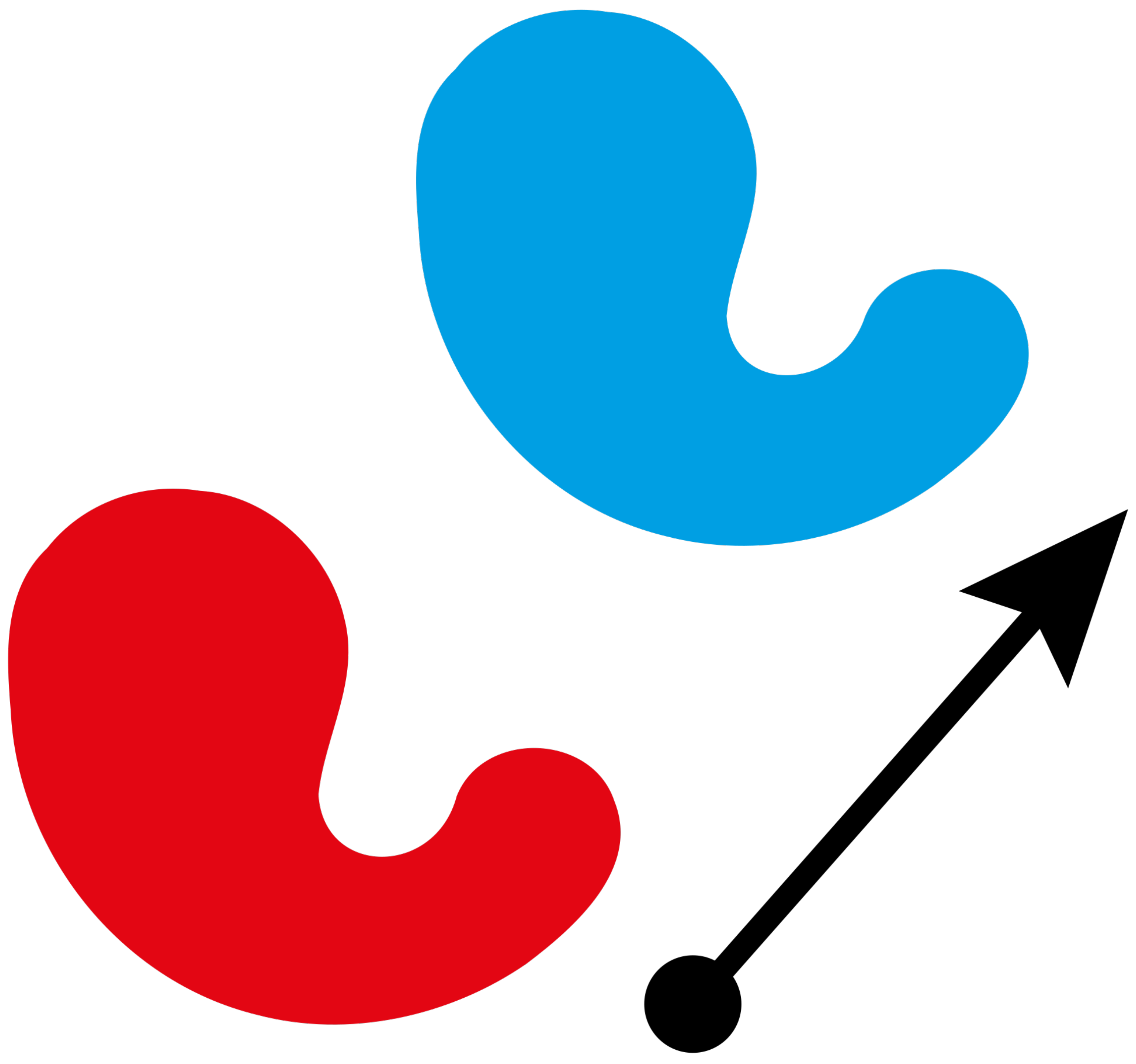
https://commons.wikimedia.org/wiki/File:Traslazione_OK.svg

Translation

Given a vector $\mathbf{t}$ a **translation** is the transformation

$$T(\mathbf{x}) = \mathbf{x} + \mathbf{t}$$

As we've seen, **translation is not linear:**



In 2D

https://commons.wikimedia.org/wiki/File:Traslazione_OK.svg

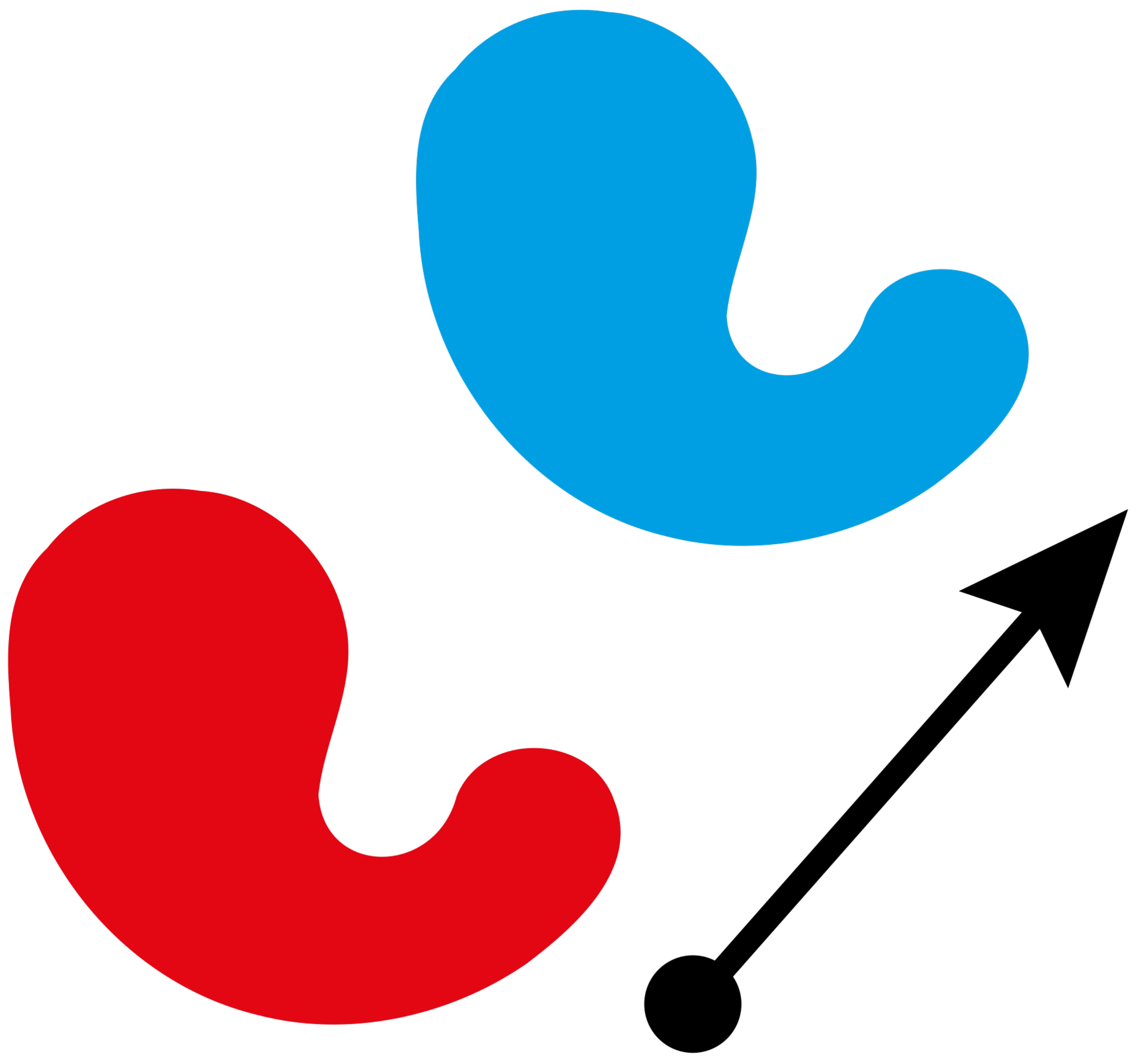
Translation

Given a vector $\mathbf{t}$ a **translation** is the transformation

$$T(\mathbf{x}) = \mathbf{x} + \mathbf{t}$$

As we've seen, **translation is not linear:**

$$T(\mathbf{0}) = \mathbf{t}$$



In 2D

https://commons.wikimedia.org/wiki/File:Traslazione_OK.svg

Translation

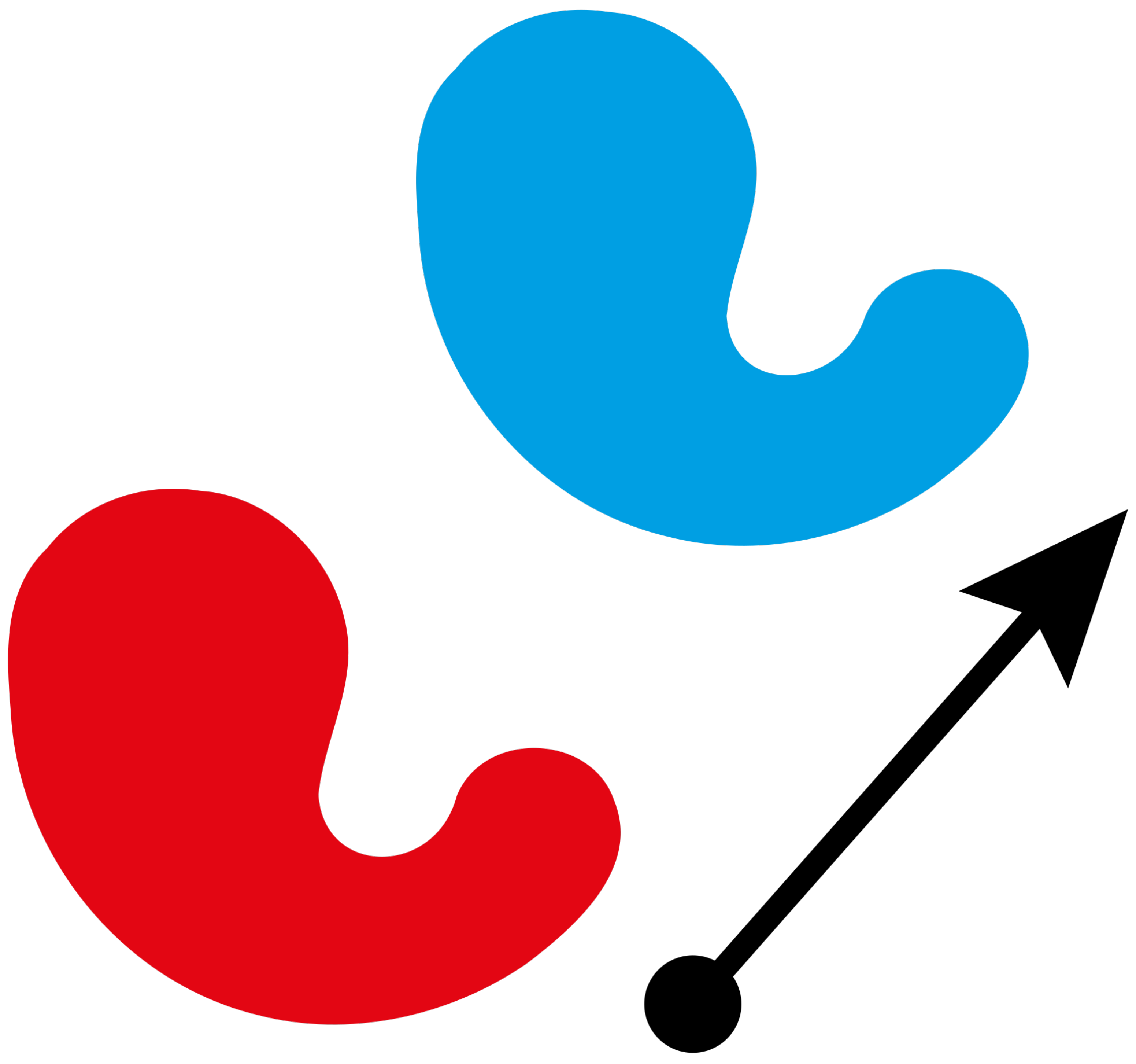
Given a vector $\mathbf{t}$ a **translation** is the transformation

$$T(\mathbf{x}) = \mathbf{x} + \mathbf{t}$$

As we've seen, **translation is not linear:**

$$T(\mathbf{0}) = \boxed{\mathbf{t}}$$

For this to be interesting
$\mathbf{t}$ will be nonzero



In 2D

https://commons.wikimedia.org/wiki/File:Traslazione_OK.svg

Translation (3D)

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \mapsto \begin{bmatrix} x + a \\ y + b \\ z + c \end{bmatrix}$$

Observation. This would be linear if we had another variable.

Translation (3D)

$$\begin{bmatrix} x \\ y \\ z \\ q \end{bmatrix} \mapsto \begin{bmatrix} x + aq \\ y + bq \\ z + cq \\ q \end{bmatrix}$$

Observation. This would be linear if we had another variable.

Translation (3D)

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \mapsto \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \qquad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \mapsto \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \qquad \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \mapsto \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \qquad \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \mapsto \begin{bmatrix}a\\b\\c\\1\end{bmatrix}$$

Observation. This would be linear if we had another variable.

Translation (3D)

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Observation. This would be linear if we had another variable.

So if we are willing to keep around an extra entry, we can do translation **linearly.**

Homogeneous Coordinates

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

For initializing to homogeneous coordinates, we set this to 1

Cartesian to homogeneous

The **homogeneous coordinate** for vector in $\mathbb{R}^3$ is the same except "sheared" into the 4th dimension.

We use the extra entry to perform simple nonlinear transformations in a linear setting.

Translation (3D)

Definition. The **3D translation matrix** for homogeneous coordinates which translates by $(a, b, c)^T$ is the following.

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example. $\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \begin{bmatrix} x+2 \\ y+2 \\ z+2 \\ 1 \end{bmatrix}$

Matrix Transformations for Homogeneous Coordinates

$$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \longrightarrow \begin{bmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix Transformations for Homogeneous Coordinates

$$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \longrightarrow \begin{bmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now all our transformations need to be 4×4 matrices.

Matrix Transformations for Homogeneous Coordinates

$$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \longrightarrow \begin{bmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now all our transformations need to be 4×4 matrices.

But it's easy make 3×3 matrices work for homogeneous coordinates.

Matrix Transformations for Homogeneous Coordinates

$$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \longrightarrow \begin{bmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now all our transformations need to be 4×4 matrices.

But it's easy make 3×3 matrices work for homogeneous coordinates.

If a transformation is linear, it doesn't need the extra coordinate.

Example: Homogeneous Rotation

Rotating counterclockwise about the x–axis in homogeneous coordinates is given by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projections

Vanishing Points

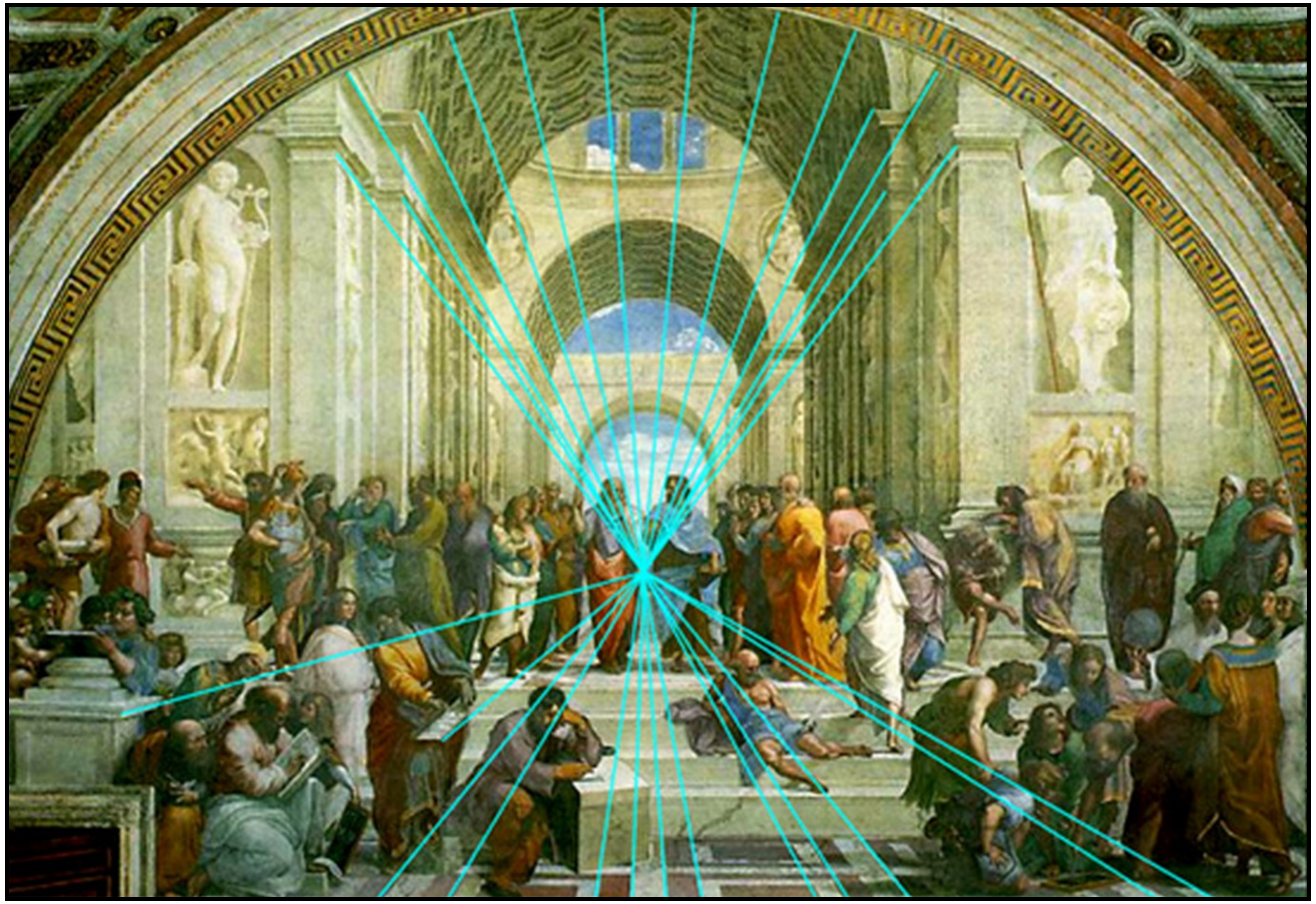
Parallel lines in space don't necessarily look parallel at a distance, they angle towards a point in the distance.

This is a side effect of **perspective projection**.

https://commons.wikimedia.org/wiki/File:Railroad-Tracks-Perspective.jpg

Vanishing Point

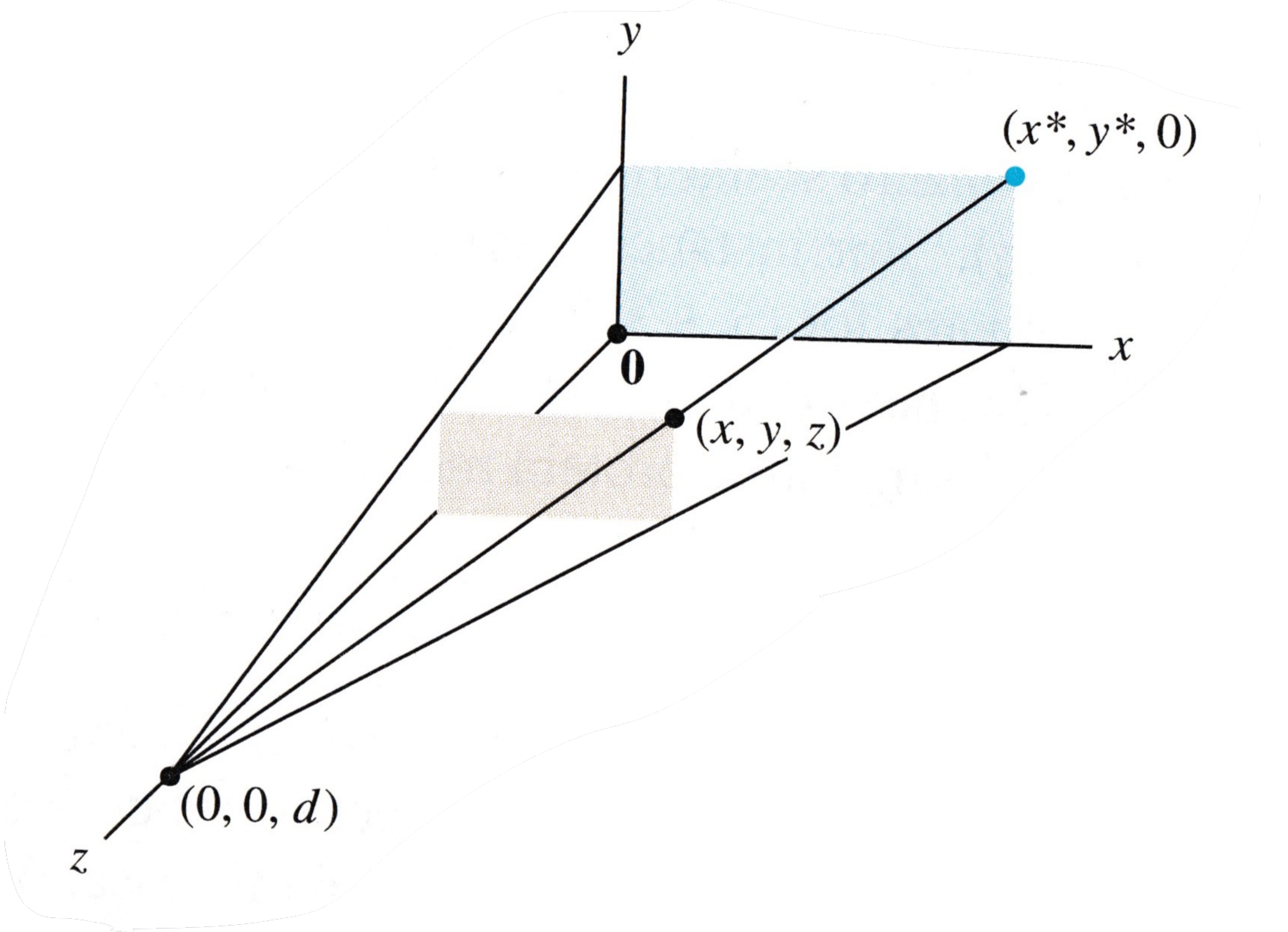
The School of Athens (~1510)



Computing Perspective

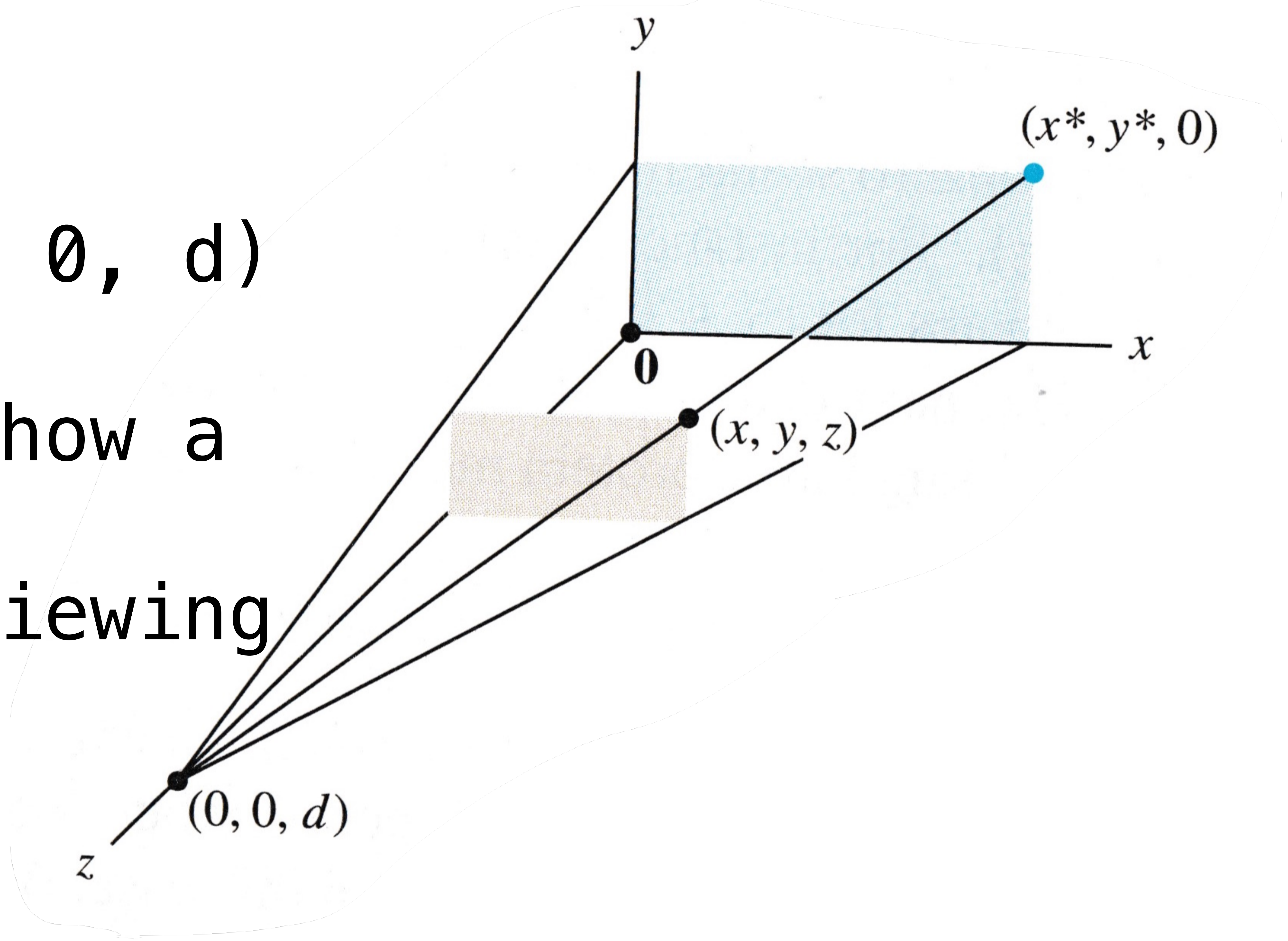
Light enters our eyes (or camera) at a single point from all directions.

Closer things "appear bigger" in our field of vision.

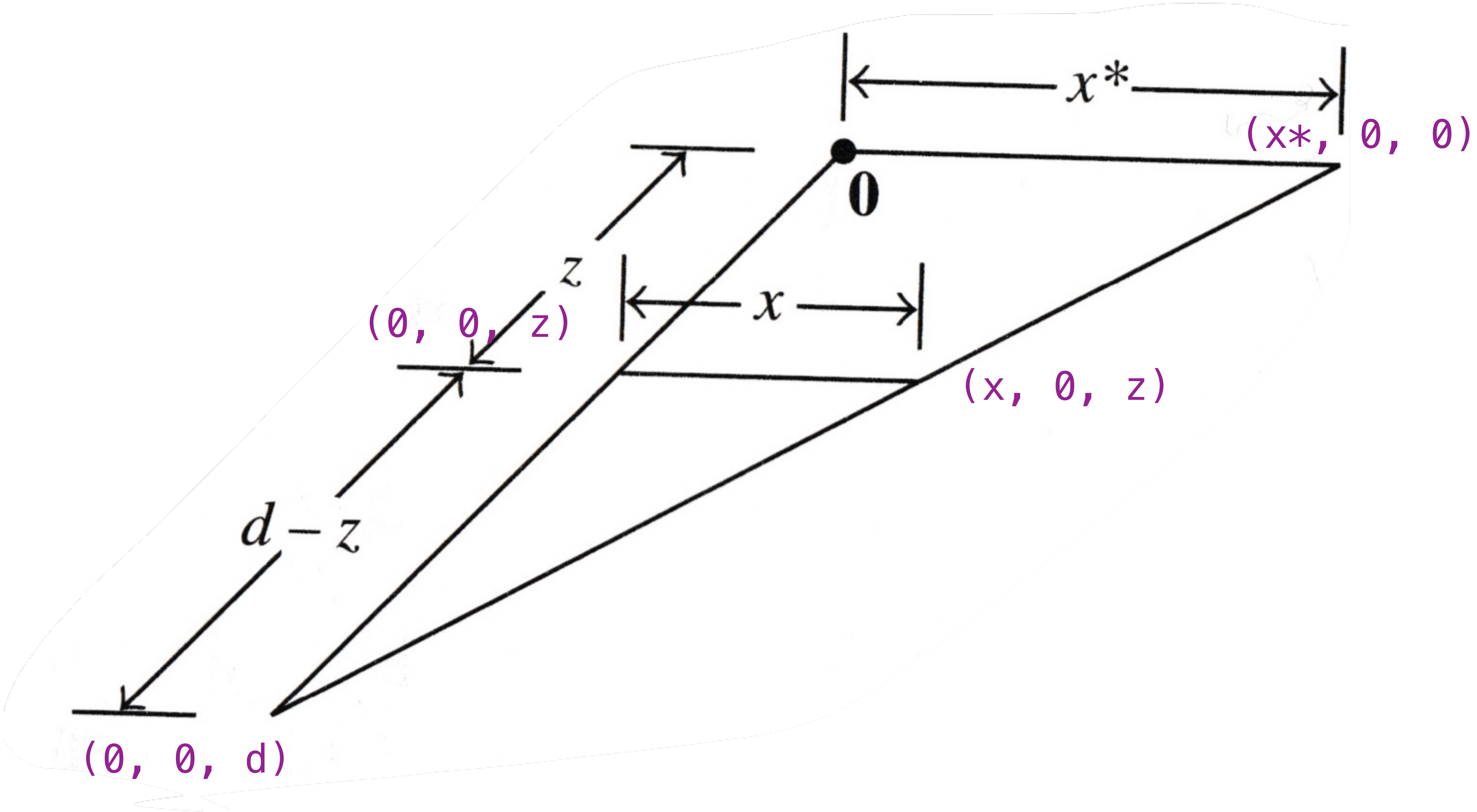


Computing Perspective

Problem. Given a **viewing position** (0, 0, d) and a **viewing plane** (xy-axis) determine how a point (x, y, z) is *projected* onto the viewing plane.

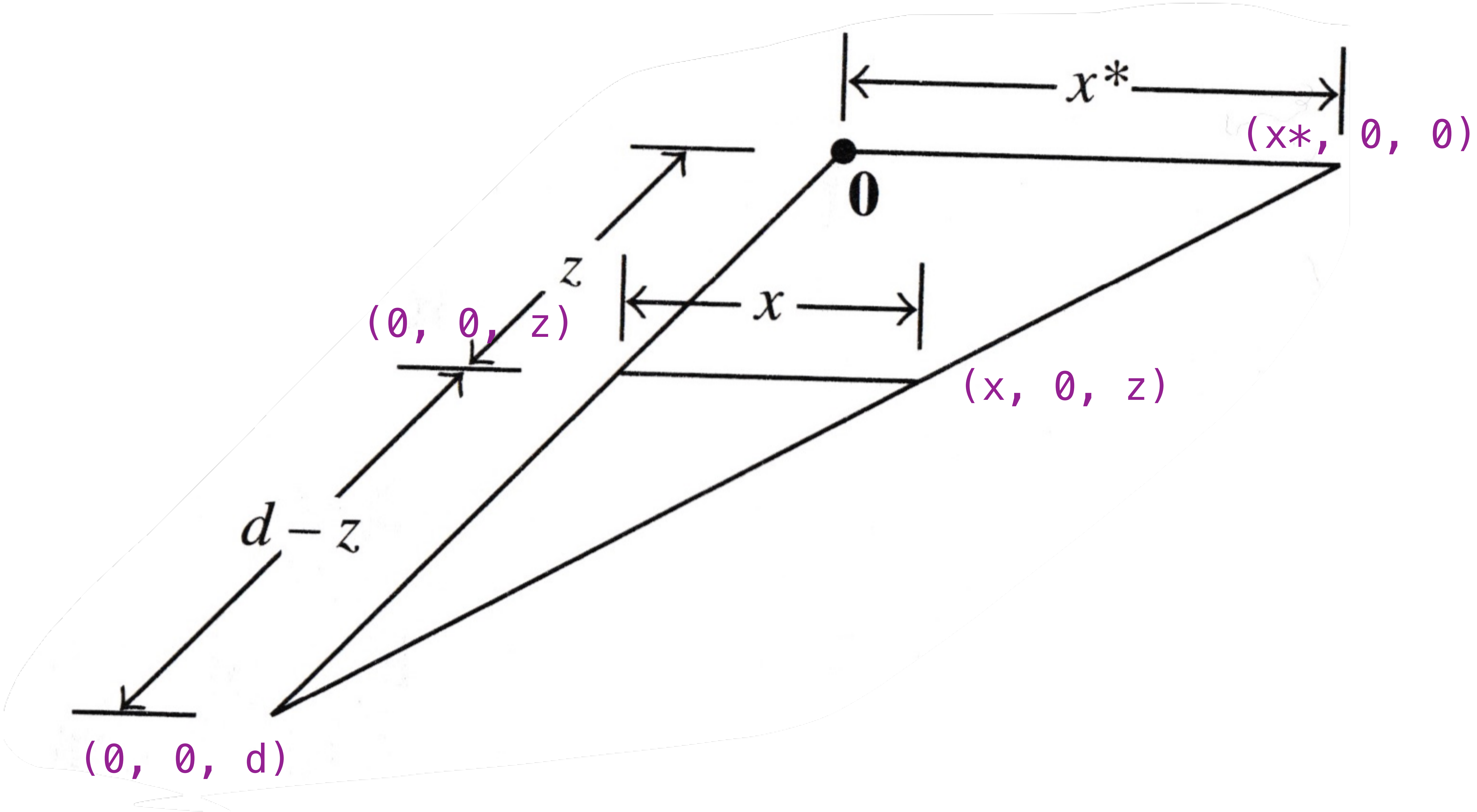


Similar Triangles



Similar Triangles

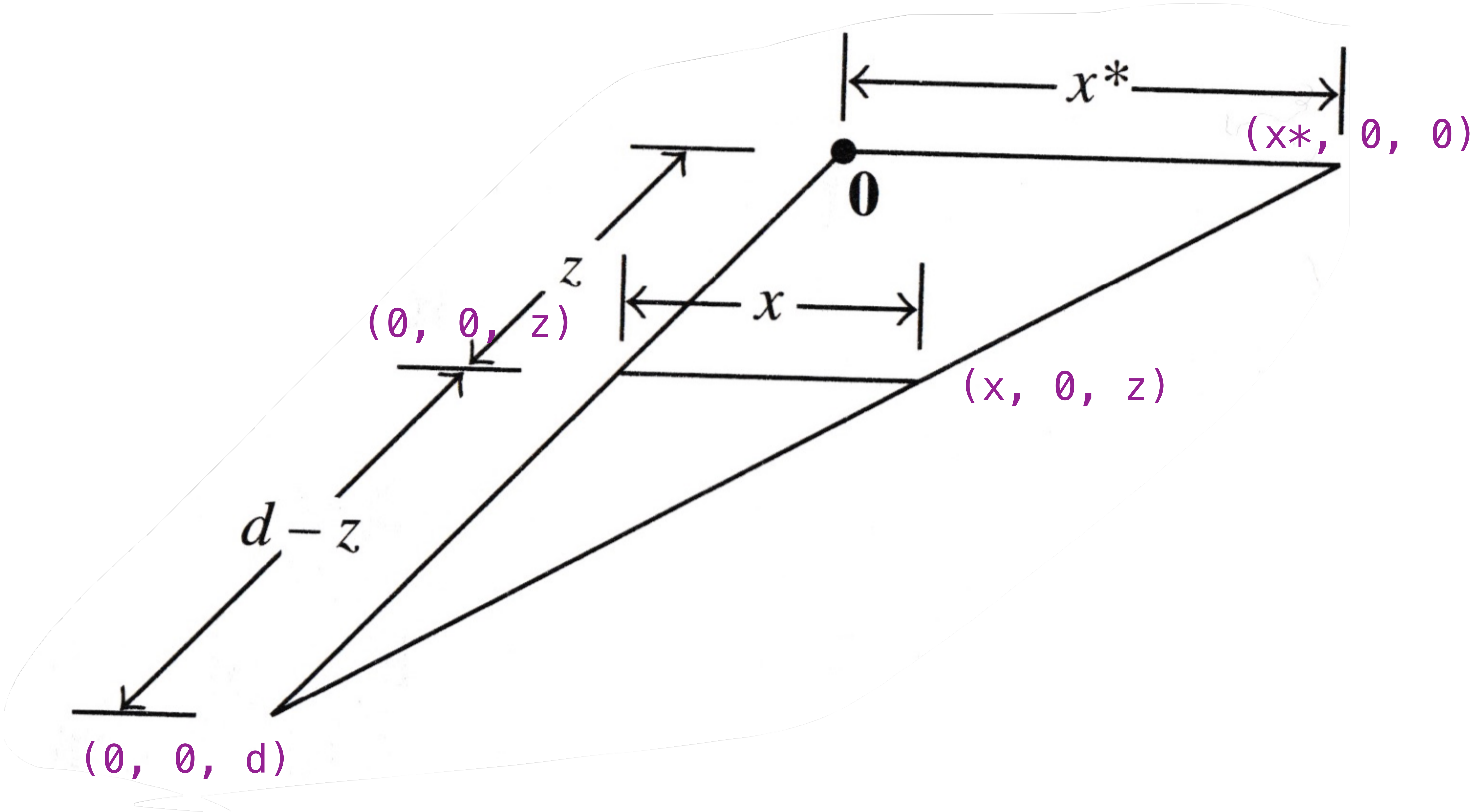
Similar triangles are triangles with the same angles (in the same order).



Similar Triangles

Similar triangles are triangles with the same angles (in the same order).

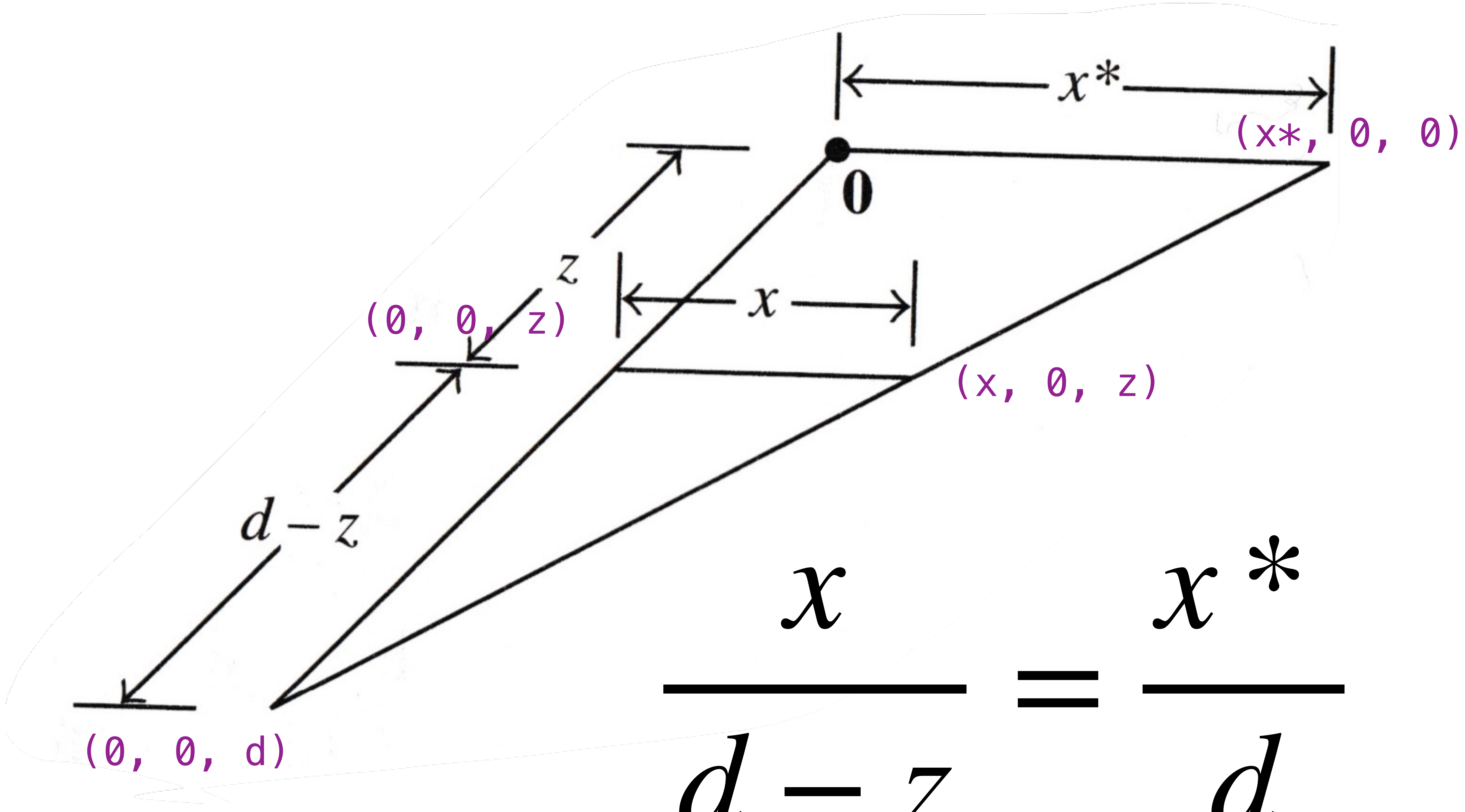
Similar triangles preserve side ratios.



Similar Triangles

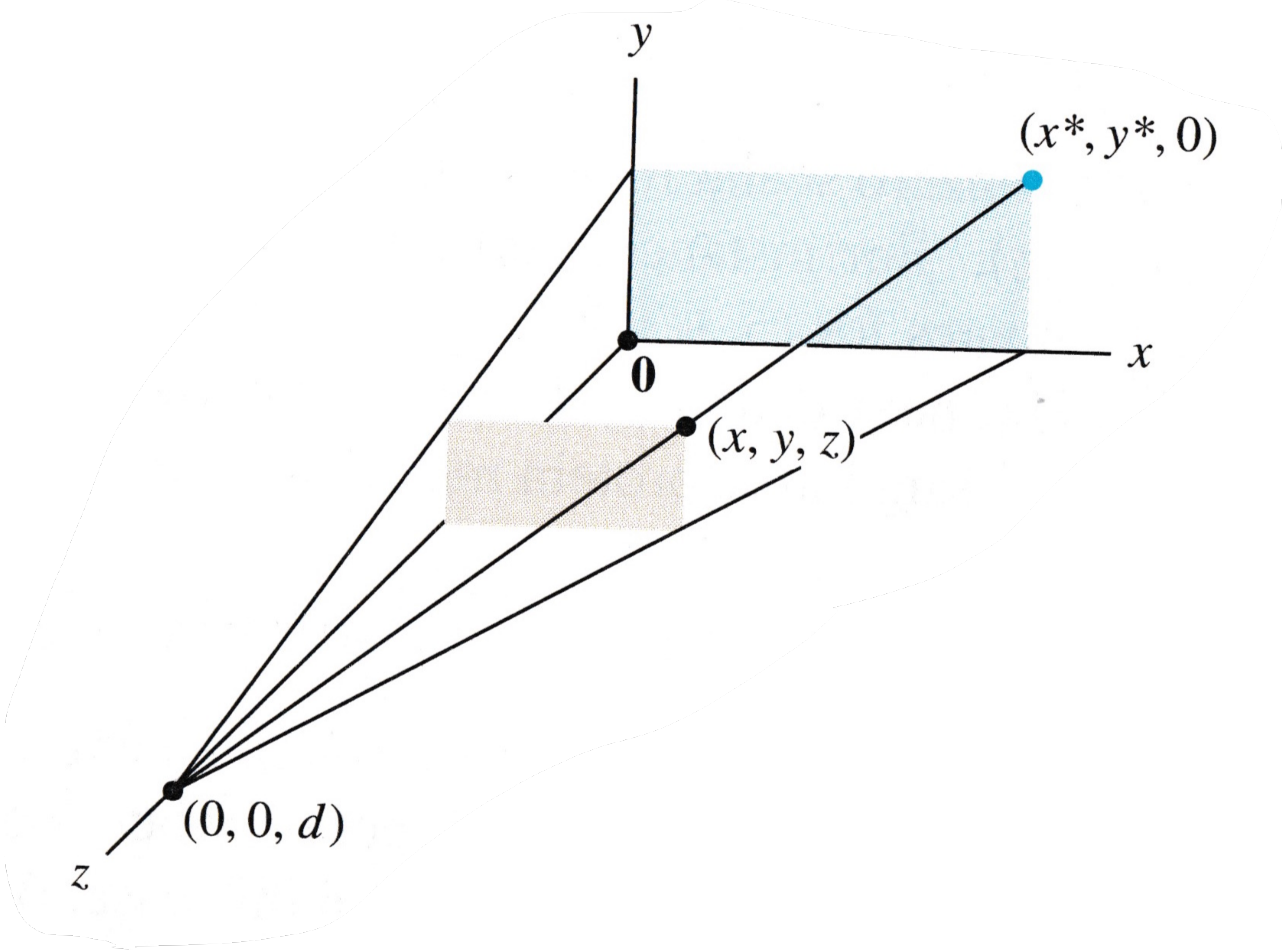
Similar triangles are triangles with the same angles (in the same order).

Similar triangles preserve side ratios.



$$\frac{x}{d-z} = \frac{x^*}{d}$$

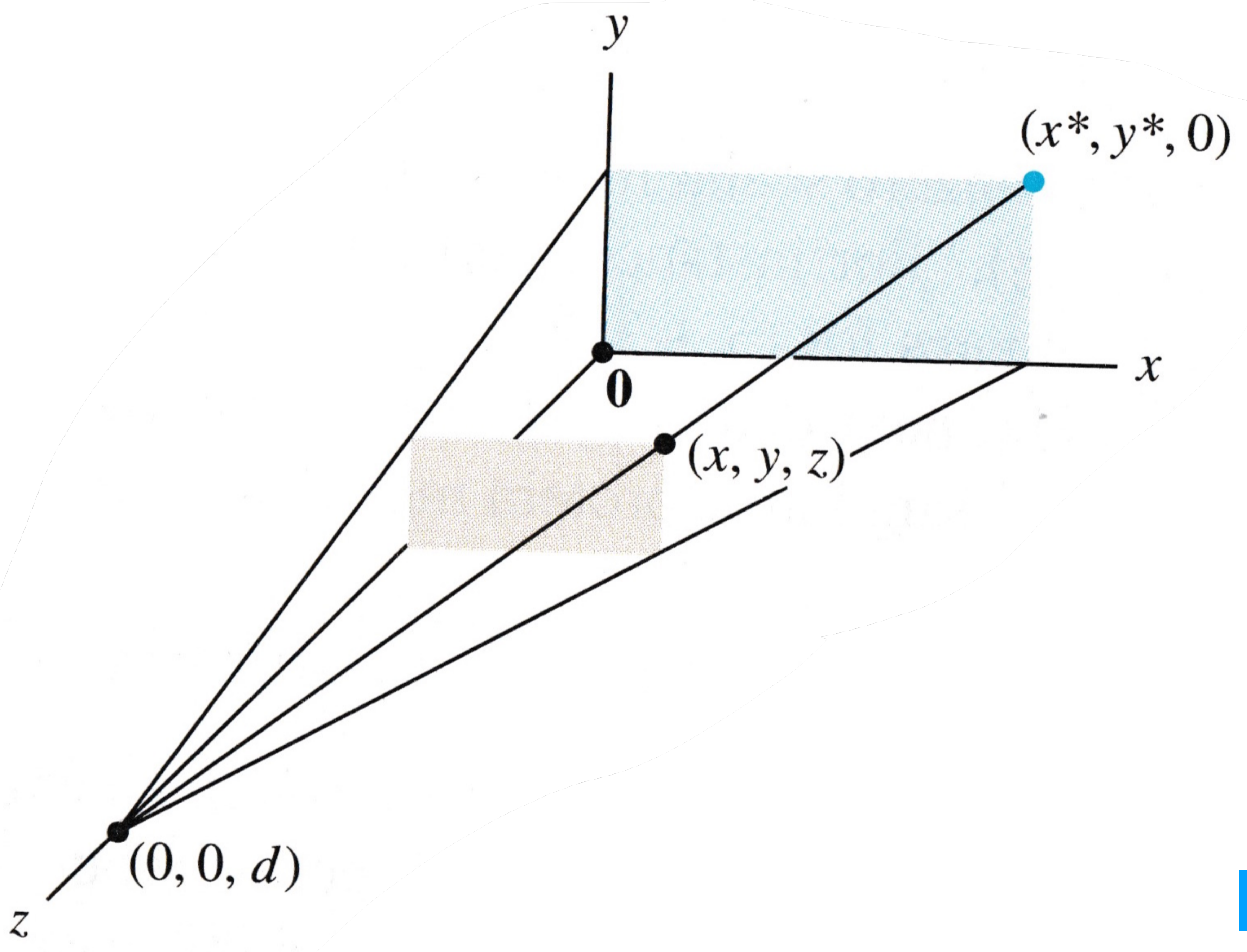
The Transformation



$$x^* = \frac{dx}{d - z} = \frac{x}{1 - z/d}$$

$$y^* = \frac{dy}{d - z} = \frac{y}{1 - z/d}$$

The Transformation



$$x^* = \frac{dx}{d-z} = \boxed{\frac{x}{1 - z/d}}$$

$$y^* = \frac{dy}{d-z} = \boxed{\frac{y}{1 - z/d}}$$

Not linear, But we will homogeneous coordinates to address this

A Trick with Homogeneous Coordinates

$$\begin{bmatrix} x \\ y \\ z \\ h \end{bmatrix} \mapsto \begin{bmatrix} x/h \\ y/h \\ z/h \end{bmatrix}$$

homogeneous to Cartesian

A Trick with Homogeneous Coordinates

$$\begin{bmatrix} x \\ y \\ z \\ h \end{bmatrix} \mapsto \begin{bmatrix} x/h \\ y/h \\ z/h \end{bmatrix}$$

homogeneous to Cartesian

We can compute perspective using homogeneous coordinates if we allow the extra entry to **vary.**

A Trick with Homogeneous Coordinates

$$\begin{bmatrix} x \\ y \\ z \\ h \end{bmatrix} \mapsto \begin{bmatrix} x/h \\ y/h \\ z/h \end{bmatrix}$$

homogeneous to Cartesian

We can compute perspective using homogeneous coordinates if we allow the extra entry to **vary.**

When we convert back to normal coordinates, we divide by the extra entry (this is consistent with before).

Perspective Projection

Definition. The **perspective projection** (and matrix) is given by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 - z/d \end{bmatrix}$$

When we convert back to usual coordinates, we divide by $1 - z/d$ as desired.

Homework 7

The Rough Outline

The Rough Outline

1. Take in a wire frame, represented as a collection of m line segments (pairs of points in $\mathbb{R}^3$).

The Rough Outline

1. Take in a wire frame, represented as a collection of m line segments (pairs of points in $\mathbb{R}^3$).

2. Convert these points into a $4 \times 2m$ matrix D, one column for each endpoint, in homogeneous coordinates.

The Rough Outline

1. Take in a wire frame, represented as a collection of m line segments (pairs of points in $\mathbb{R}^3$).
2. Convert these points into a $4 \times 2m$ matrix D, one column for each endpoint, in homogeneous coordinates.
3. Build a transformation matrix A to manipulate the wireframe and project it onto a viewing plane.

The Rough Outline

1. Take in a wire frame, represented as a collection of m line segments (pairs of points in $\mathbb{R}^3$).

2. Convert these points into a $4 \times 2m$ matrix D, one column for each endpoint, in homogeneous coordinates.

3. Build a transformation matrix A to manipulate the wireframe and project it onto a viewing plane.

4. Convert the columns of D into points in $\mathbb{R}^2$, and then pair them back up into endpoints of line segments.

The Rough Outline

1. Take in a wire frame, represented as a collection of m line segments (pairs of points in $\mathbb{R}^3$).

2. Convert these points into a $4 \times 2m$ matrix D, one column for each endpoint, in homogeneous coordinates.

3. Build a transformation matrix A to manipulate the wireframe and project it onto a viewing plane.

4. Convert the columns of D into points in $\mathbb{R}^2$, and then pair them back up into endpoints of line segments.

5. Draw the resulting image on the screen.

demo

A Couple Words of Warning

Check your system early. Make sure you can run matplotlib widgets.

Post on piazza if there seems to be a platform dependent issue.