FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Lecture #06

Going 3D

Computer Graphics

Winter Term 2020/21

Marc Stamminger / Roberto Grosso

Content

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- In this lecture:
  - parallel projection
  - more about homogeneous coordinates
  - points, vectors, and normals
  - projective transformations

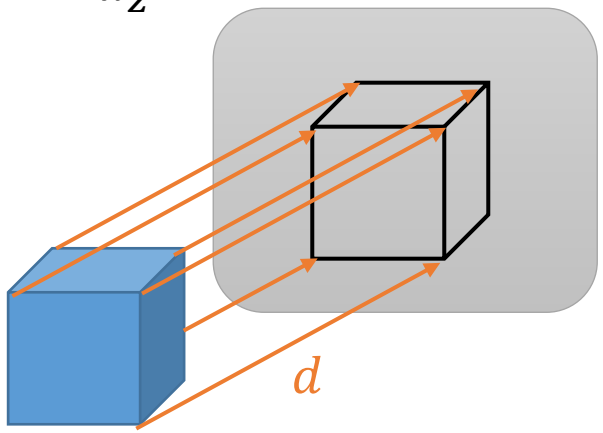
Parallel Projection

- Our world is 3D, images are 2D
- We have to project the world to a 2D image

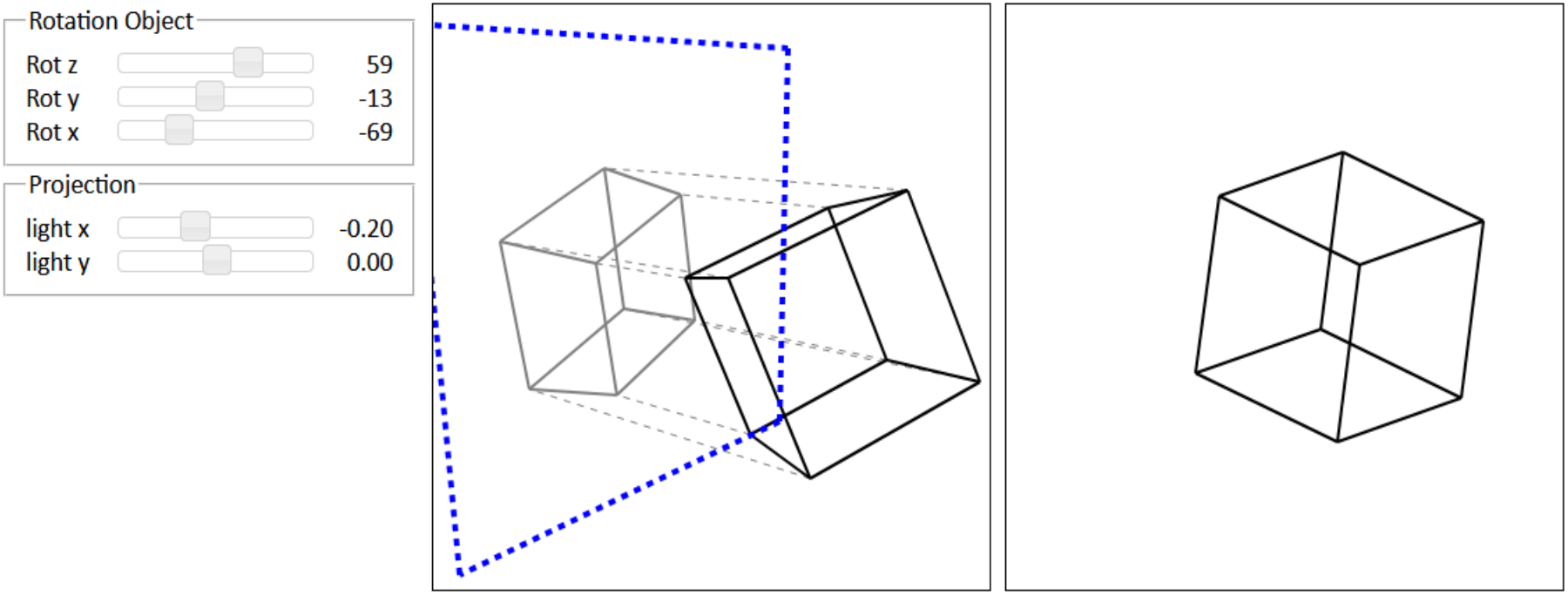
- First attempt:
  project along a given direction $d = (d_1, d_2, d_3)$ to a screen plane, e.g. $z = 0$:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \alpha d = \begin{pmatrix} x' \\ y' \\ 0 \end{pmatrix} \rightarrow \alpha = -\frac{z}{d_z}$$

- $$\rightarrow \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\frac{d_x}{d_z} \\ 0 & 1 & -\frac{d_y}{d_z} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$



Parallel Projection Demo



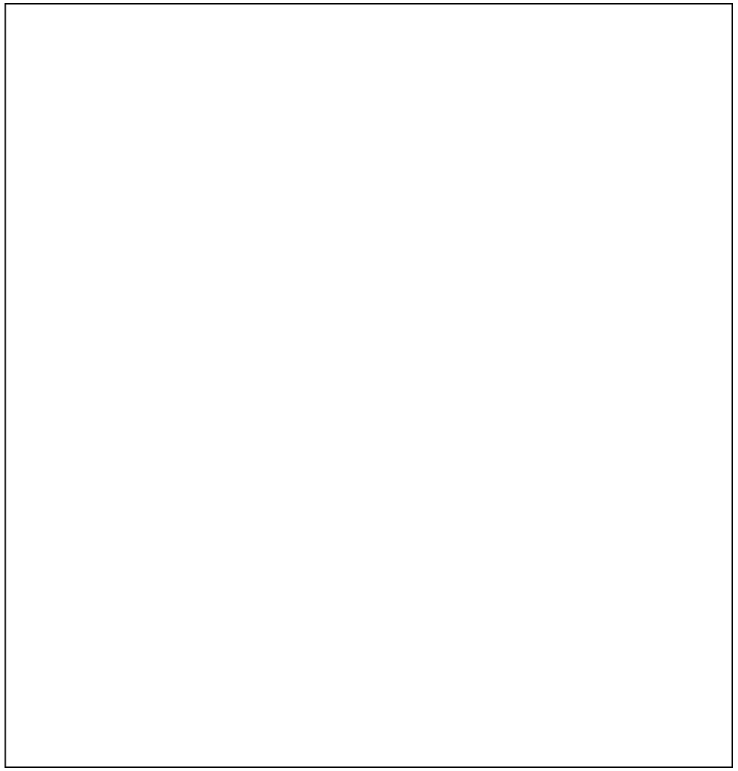
WebGL Parallel Projection

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Source Code | Vertex Shader | Pixel Shader

```
1  // var gl is set to WebGL context
2  var glVersion = gl.getParameter(gl.VERSION);
3  var glslVersion = gl.getParameter(gl.SHADING_LANGUAGE_VERSION);
4  console.log("GL Version: \t" + glVersion);
5  console.log("GLSL Version: \t" + glslVersion);
6
7  var v = [-1,-1,-1,  -1,-1,1, -1,1,-1, -1,1,1,
8           1,-1,-1, 1,-1,1, 1,1,-1, 1,1,1];
9  var i = [0,1, 0,2, 0,4, 1,3, 1,5, 2,3, 2,6, 3,7, 4,5, 4,6, 5,7,
10
11 // parallel projection direction
12 var d = [.2,.3,.8];
13
14 // projection matrix
15 var P = [
16     [1,0,-d[0]/d[2],0],
17     [0,1,-d[1]/d[2],0],
18     [0,0,0,0],
19     [0,0,0,1]
20 ];
21
22
```

>>>>



Mapping 3D to 2D

- We can throw away z to get a 3D→2D mapping:

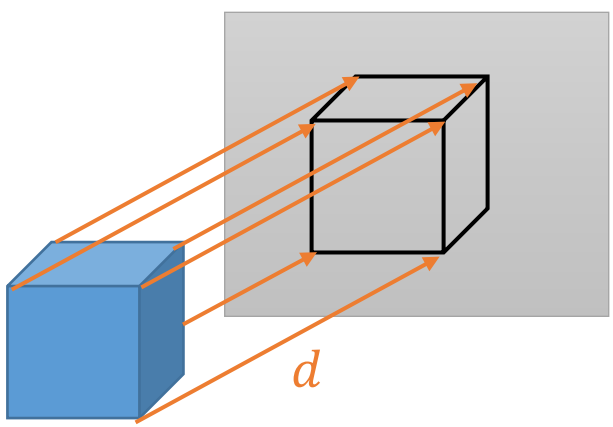
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\frac{d_x}{d_z} \\ 0 & 1 & -\frac{d_y}{d_z} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

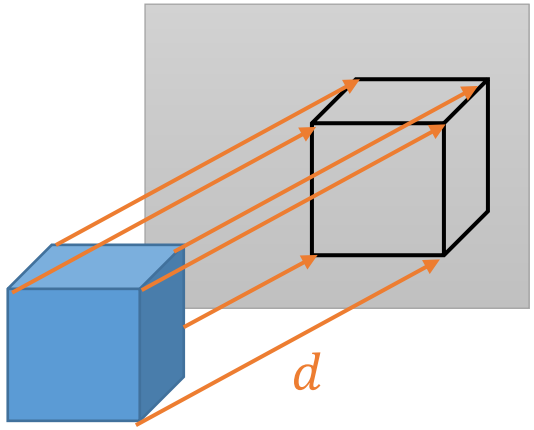
- but it makes sense to keep z, we need it later (e.g. for occlusion):

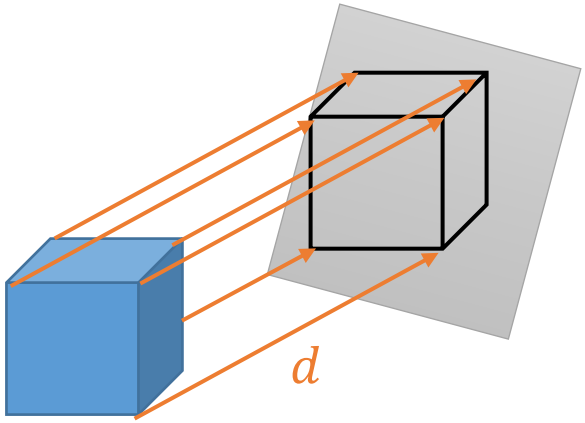
$$\begin{pmatrix} x' \\ y' \\ z' = z \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\frac{d_x}{d_z} \\ 0 & 1 & -\frac{d_y}{d_z} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Parallel Projection

- Further question: which part of the image plane becomes our image ?
  → define rectangle on image plane

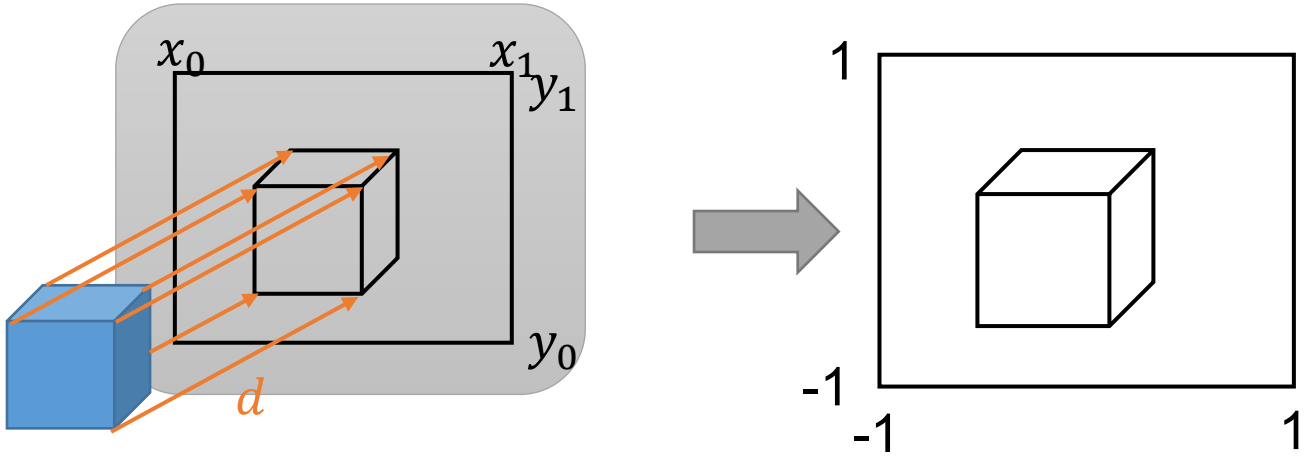






Parallel Projection

- for example, we can use an axis-aligned rectangle $[x_0, x_1] \times [y_0, y_1]$

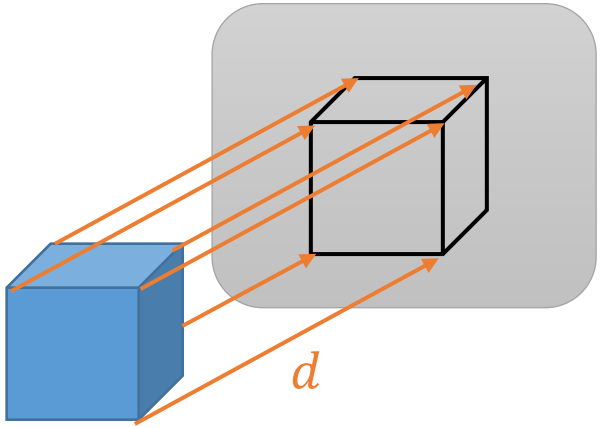


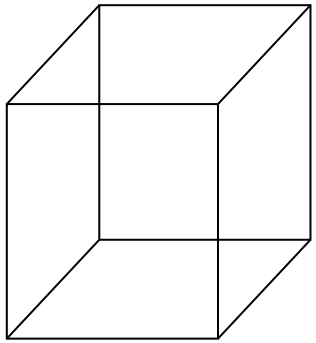
- With homogeneous coordinates:

$$\begin{pmatrix} x' \\ y' \\ z' = z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{x_1 - x_0} & 0 & 0 & \frac{x_0 + x_1}{x_0 - x_1} \\ 0 & \frac{2}{y_1 - y_0} & 0 & \frac{y_0 + y_1}{y_0 - y_1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -\frac{d_x}{d_z} & 0 \\ 0 & 1 & -\frac{d_y}{d_z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Mapping 3D to 2D

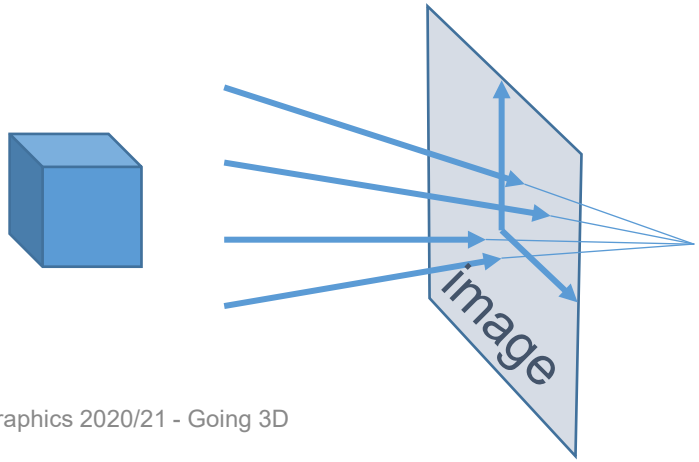
- Simple projection: parallel projection onto plane
- Affine → parallel lines remain parallel

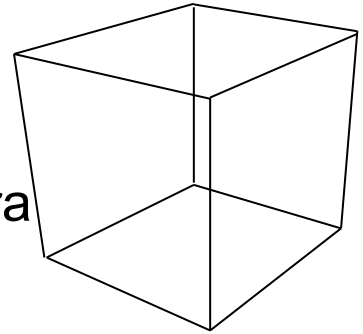




orthographic projection

- real perspective → point projection (later)

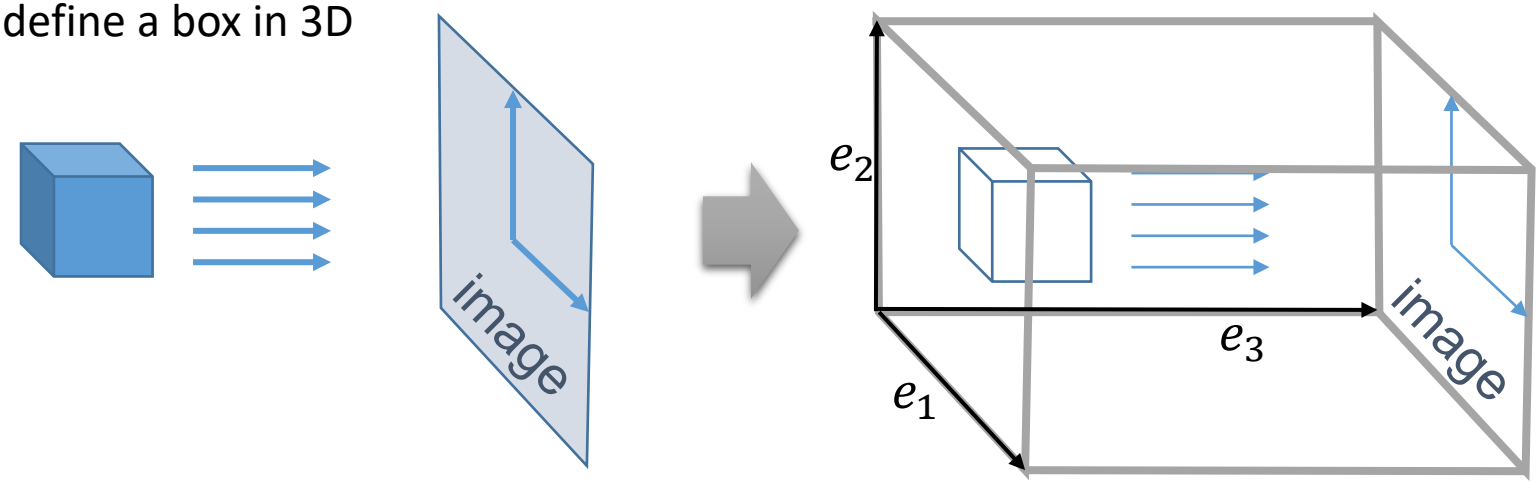




“real” perspective projection

Orthographic Projection

- Alternative interpretation:
  - define a box in 3D

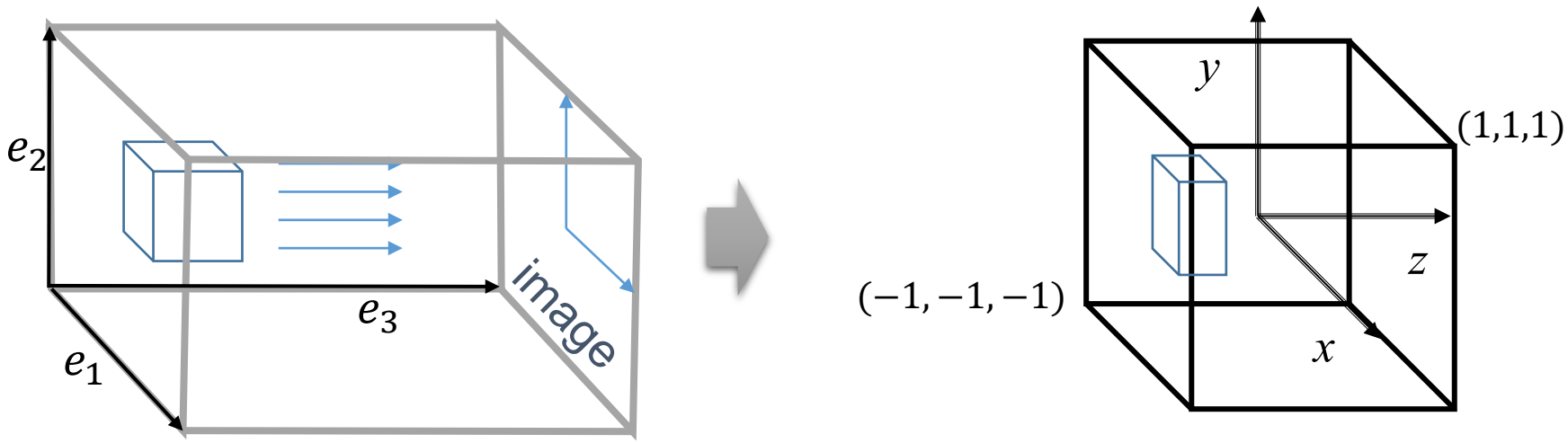


  - e_3 is the projection direction
  - e_1 and e_2 span the image plane and define the image window

  - The scene inside the box is projected to the box's backside
  - This can generate all parallel projections !
    → Non-perpendicular projection only results in non-uniform scaling

Orthographic Projection

- Simple way to compute this:
  - transform box to unit cube $[-1,1]^3$
  - → (x, y) are image coordinates ∈ $[-1,1]^2$
    → z is „normalized depth“ ∈ $[-1,1]$
  - usually box is chosen deep enough to contain entire scene
    → all depth values in the range $[-1,1]$

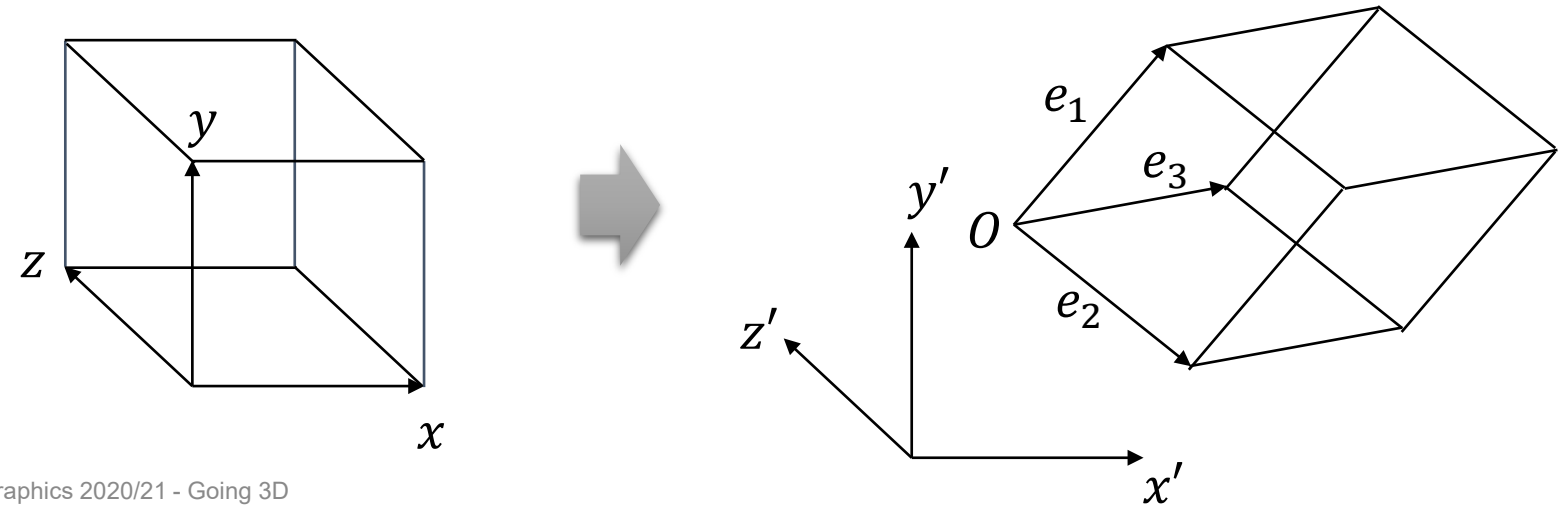


3D Affine Transformations

- Affine Transformations in 3D as in 2D:
  - Coordinate system changes:

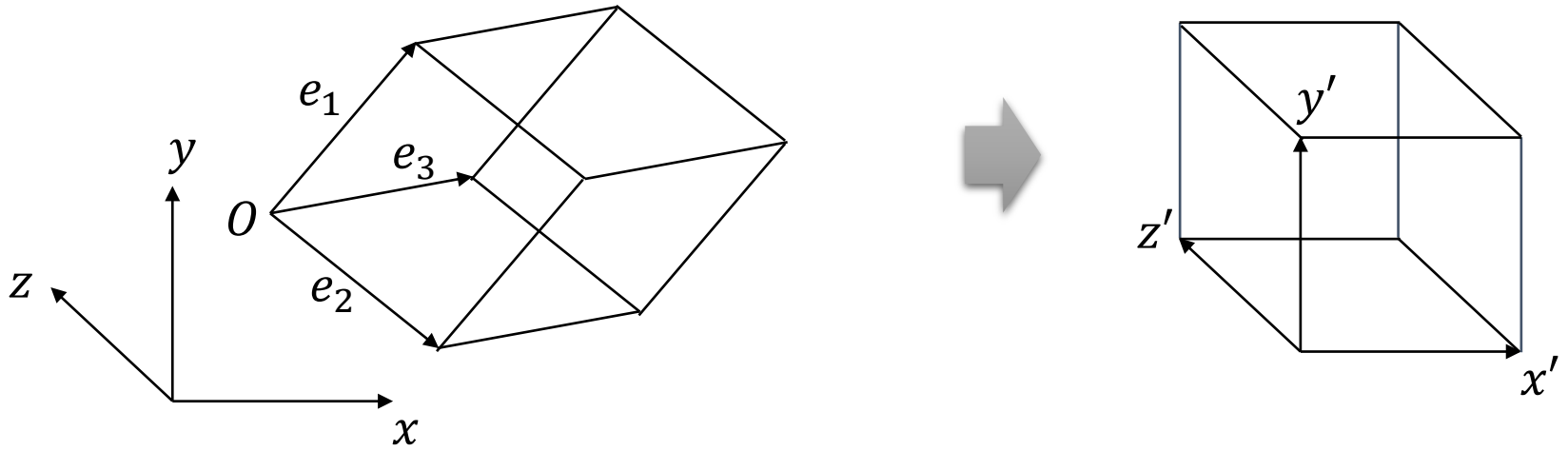
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} \vdots & \vdots & \vdots & O_1 \\ e_1 & e_2 & e_3 & O_2 \\ \vdots & \vdots & \vdots & O_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

  - Maps the unit cube to a parallelepiped



3D Affine Transformations

- Other direction: maps any parallelepiped to the unit cube



- use the inverse matrix:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} \vdots & \vdots & \vdots & O_1 \\ e_1 & e_2 & e_3 & O_2 \\ \vdots & \vdots & \vdots & O_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

3D Affine Transformations

- Scalings, Translation, Shearings, Reflections are in 3D as in 2D
  - only three values / skew directions / reflection planes instead of 2
  - Translation: $translate(d_x, d_y, d_z) = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - Scaling: $scale(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - Z-Shear: $shear_z(d_x, d_y) = \begin{pmatrix} 1 & 0 & d_x & 0 \\ 0 & 1 & d_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

3D Affine Transformations

- Rotation around the x-, y- and z-axis
  - $Rot_x(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - $Rot_y(\phi) = \begin{pmatrix} \cos\phi & 0 & \sin\phi\, 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
  - $Rot_z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- But how do we describe arbitrary rotations ?

Rotations in 3D

- For now, we only rotate around the origin
  → a 3x3 matrix is sufficient
- The columns of a rotation matrix are the unit vectors after rotation:

$$R\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Here u, v, w are the main axes after the rotation

- For rotation matrices, the inverse is simply the transpose:

$$R^{-1} = R^T$$

3D Affine Transformations

- Special case: orthonormal e_1, e_2, e_3:
  $$e_1 \circ e_2 = e_1 \circ e_3 = e_2 \circ e_3 = 0, \qquad e_1 \circ e_1 = e_2 \circ e_2 = e_3 \circ e_3 = 1$$

- = Rigid Transformation: rotation + translation

- The inverse of rotation = transposed (for linear part)

$$\begin{pmatrix} \vdots & \vdots & \vdots & O_1 \\ e_1 & e_2 & e_3 & O_2 \\ \vdots & \vdots & \vdots & O_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} & R & & \vdots \\ & & & t \\ & & & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} & R^T & & \vdots \\ & & & -R^T t \\ & & & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D Affine Transformations

- Caveat:
  - a matrix with orthonormal rows (or columns) is called “orthogonal”
  - a matrix with orthogonal rows (or columns) is called “matrix” ☺

Rotations in 3D

- The description of 3D rotations is a core problem in computer graphics
  - Positioning objects in the world
  - Animating objects (= interpolating rotations)
  - Modeling camera animations
  - …
- Two important questions:
  - how to describe a rotation ?
  - how to interpolate rotations ?
    → Some representations result in awkward interpolation

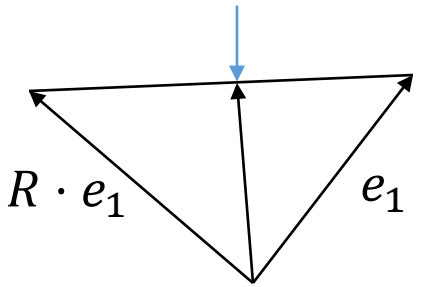
Rotations in 3D

- How to specify rotations in 3D
  - Orthogonal matrices
  - 3 Euler rotations, e.g.
    - Rotz $\rightarrow$ Rotx $\rightarrow$ Rotz
    - Rotz $\rightarrow$ Roty $\rightarrow$ Rotz
    - Rotx $\rightarrow$ Roty $\rightarrow$ Rotz
  - Axis of rotation and angle
  - Quaternions
- Etc, e.g. 2 (planar) reflections

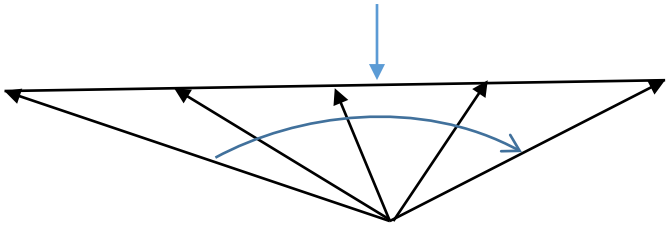
Rotations in 3D

- Orthonormal matrices
  - 9 degrees of freedom for matrix, 6 of which are fixed by constraints
  - Not very intuitive (user interface?)
  - Interpolation
    - Linear interpolation
      → interpolation of unit vectors
    - Requires renormalization
    - Non-uniform animation
      → see later: slerp-interpolation
    - Impossible for 180°rotations

$$\alpha R \cdot e_1 + (1-\alpha) \cdot e_1$$



linear interpolation on this line



non-uniform in angular space

Euler Rotations

- Euler angles
  - Any rotation can be given by three rotations about the main axes, e.g. X, Y, and Z (Leonhard Euler 1707 – 1783)
  - If the rotations angles about Z, Y, and Z are ψ, θ, and ϕ respectively, then the rotation matrix is:

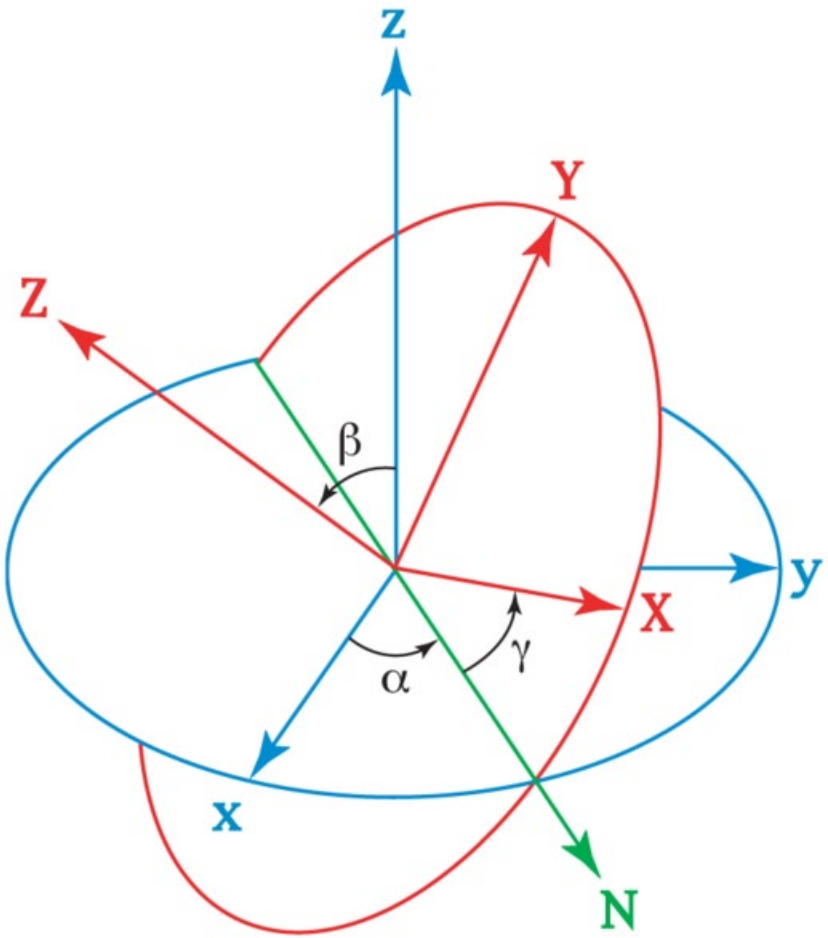
$$R = R_z(\phi)R_y(\theta)R_z(\psi)$$

$$= \begin{pmatrix} \cos\theta\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \cos\theta\sin\phi & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi \\ -\sin\theta & \sin\psi\cos\theta & \cos\psi\cos\theta \end{pmatrix}$$

- The angles ψ, θ, and ϕ are called **Euler angles**.

Euler Rotations

- Euler angles for the Euler rotation $z - x - z$ with angles $\alpha - \beta - \gamma$
- For given angles, the matrix can be computed as $R_z(\gamma)R_x(\beta)R_z(\alpha)$



Euler Rotations

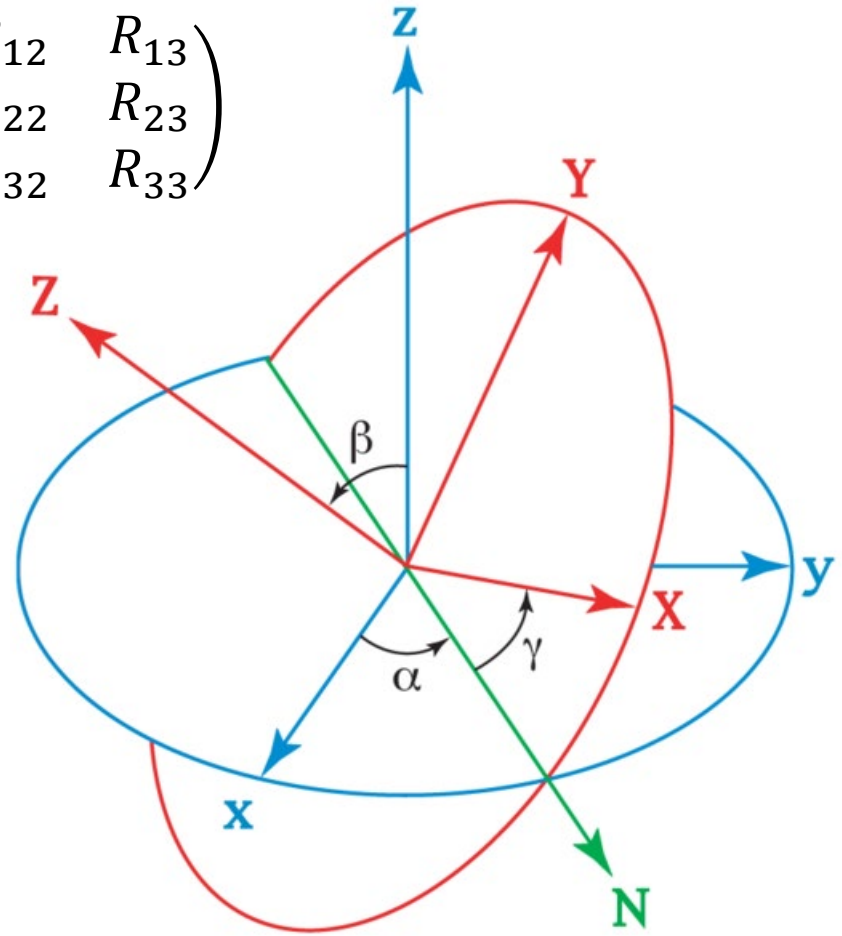
- If the rotation matrix is given as:

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$$

the angles can be determined accordingly:

- $-\sin\beta = R_{31}$
- $\tan\alpha = R_{32}/R_{33}$
- $\tan\gamma = R_{21}/R_{11}$

- for $R = R_z(\gamma)R_x(\beta)R_z(\alpha)$



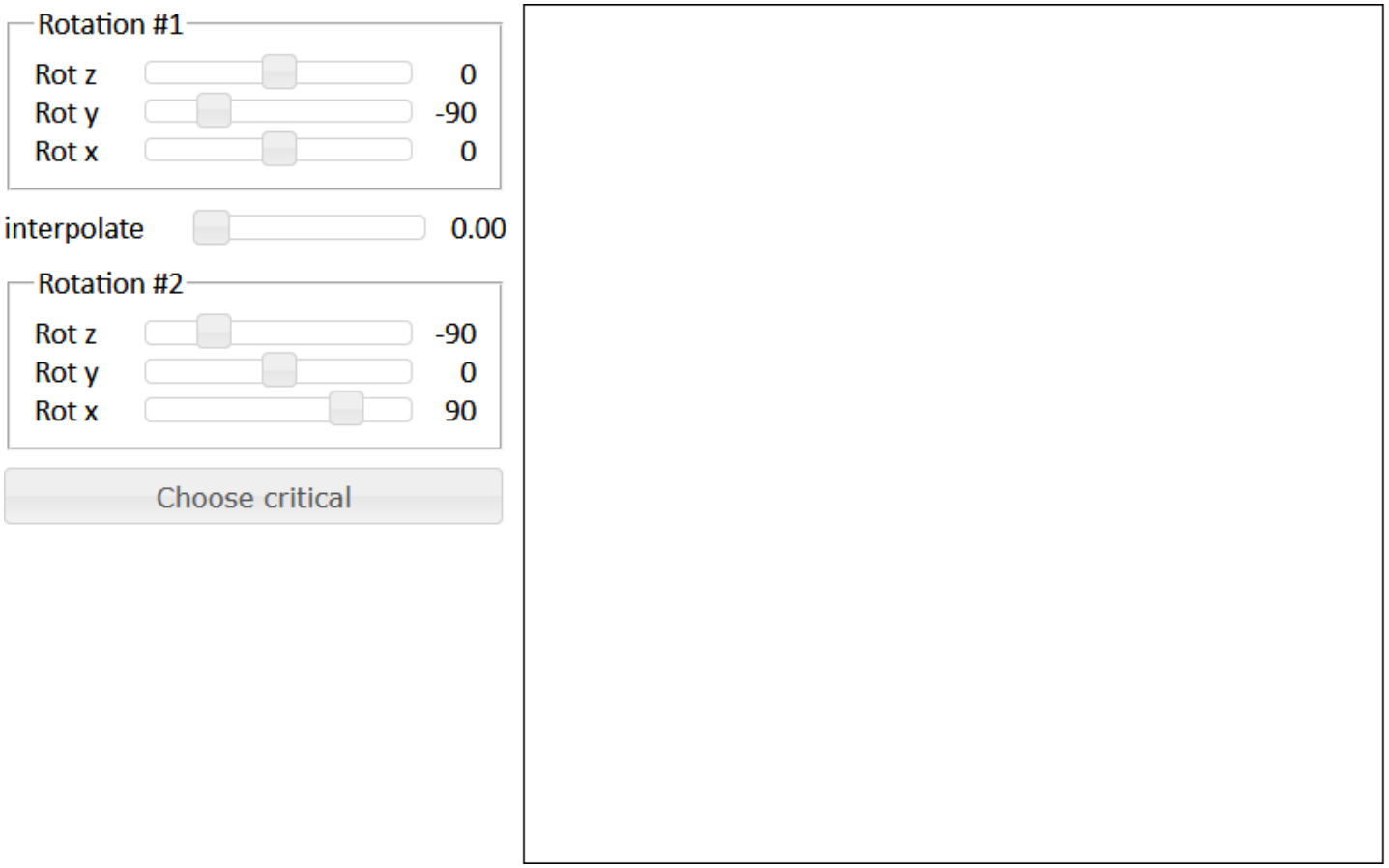
Euler Rotations

- compact
- sort of intuitive (more than matrices)

- not suited for interpolation: **Gimbal Lock problem**

- Watch this great explanatory video: https://www.youtube.com/watch?v=zc8b2Jo7mno

Euler Angles and Gimbal Lock Demo



Axis and Angle

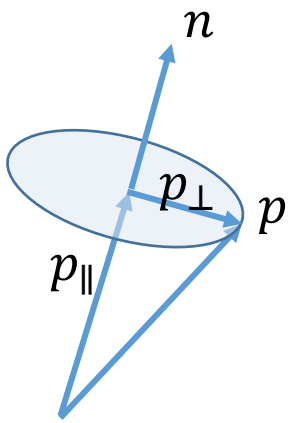
- Specify rotation by an axis $n, \|n\| = 1,$ and a rotation angle ω

- Derivation of corresponding matrix
  → transform some point p
  - decompose p into parallel (to n) and orthogonal components:
    $p = p_{\parallel} + p_{\perp}$, where $p_{\parallel} = (n \circ p)n$ and $p_{\perp} = p - p_{\parallel}$
  - Create local coordinate system $\{p_{\perp}, n \times p, n\}$, where $n \times p = n \times p_{\perp}$
  - Rotate $p_{\perp}$ about n

$$rot(p_{\perp}) = p_{\perp} \cos\omega + (n \times p) \sin\omega$$

  - add the parallel component

$$rot(p) = p_{\perp} \cos\omega + (n \times p) \sin\omega + p_{\parallel}$$



Axis and Angle

- Can we express this in a matrix?
- Rodrigues formula
  - $p_{\parallel} = (p \circ n)n = \cdots = \begin{pmatrix} n_x^2 & n_x n_y & n_x n_z \\ n_y n_x & n_y^2 & n_y n_z \\ n_z n_x & n_z n_y & n_z^2 \end{pmatrix} p = (n \cdot n^T)p = Pp$
    - $n \cdot n^T$ is called “outer product”
  - $n \times p$ can also be written as matrix:

$$n \times p = \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix} p = Qp$$

    - Q is called “skew symmetric” form of n

Axis and Angle

- $rot(p) = p_{\perp} \cos\omega + (n \times p) \sin\omega + p_{\parallel}$
  $= (p - p_{\parallel})cos\omega + \mathrm{Q\ p\ sin}\omega + p_{\parallel}$
  $= (I - P) \cos\omega \cdot p + Q \sin\omega \cdot p + P \cdot p$

- Then
  $$R(\omega, n) = P + \cos\omega(1 - P) + \sin\omega\, Q$$

- Often used definition: scale rotation axis by rotation angle
  - Define rotation with an arbitrary vector w
  - Length of w is rotation angle, normalized w is axis

Rotations in 3D: Quaternions

- Remember: complex numbers add further, imaginary component to real number:

$$(\mathrm{x}, \mathrm{y}) = x + iy$$

- Addition, multiplication etc. can be defined on these such that they form a field (Körper), and we can use them almost like real numbers
- Multiplication with a unit length complex number $(\cos\omega, \sin\omega)$ is equivalent to a rotation by ω

Rotations in 3D: Quaternions

- Quaternions carry this idea further and add three imaginary components:

$$(\mathrm{x}, \mathrm{y}) = x + i_1 y_1 + i_2 y_2 + i_3 y_3$$

  - Note: y is a 3D-vector
- Computing with quaternions:

$$q + q' = (a, b) + (c, d) = (a + c, b + d)$$
$$k \cdot q = k \cdot (a, b) = (ka, kb)$$
$$q \cdot q' = (a, b) \cdot (c, d) = (ac - b \circ d, ad + bc + b \times d)$$
$$q \circ q' = (a, b) \circ (c, d) = ac + b \circ d$$

- Further operations:
  - Conjugate of a quaternion q: $q^* = (a, b)^* = (a, -b)$
  - Norm of a quaternion q: $\|q\|_2 = \sqrt{qq^*} = \sqrt{q \circ q}$
  - Inverse of a quaternion q: $q^{-1} = \frac{\mathrm{q}^*}{\|q\|_2^2}$

Rotations in 3D: Quaternions

- Quaternions can be used to describe rotations in 3D, like complex numbers describe rotations in 2D
- Consider a unit length quaternion q
- A rotation can be applied to a vector $v \in \mathbb{R}\text{^}3$
  - Transform v to a quaternion $v \rightarrow (0, v)$
  - Rotate using $q \cdot (0, v) \cdot q^{-1}$
  - Result will have real part 0, imaginary part is rotated vector!

$$v \rightarrow (0, v) \rightarrow q \cdot v \cdot q^{-1} \rightarrow \big(0, rot(v)\big) \rightarrow rot(v)$$

  - Every rotation can be described by a quaternion and vice versa !
- Concatenation of two rotations q and r:

$$v \rightarrow q \cdot v \cdot q^{-1} \rightarrow r \cdot q \cdot v \cdot q^{-1} \cdot r^{-1} = (rq) \cdot v \cdot (rq)^{-1}$$

  is simple product

Rotations in 3D: Quaternions

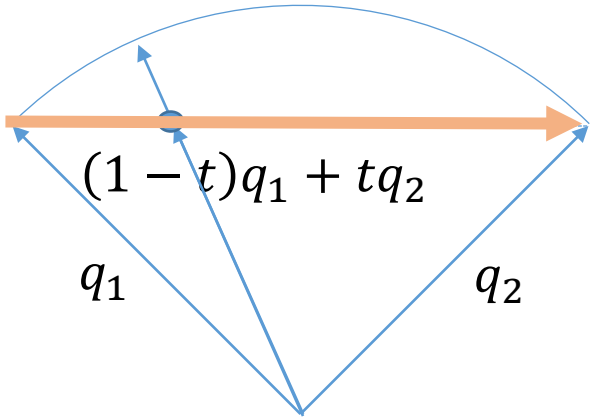
- Rotation about axis n by angle ω is expressed by the quaternion
  $q = (\cos\frac{\omega}{2}, \mathrm{n} \cdot \sin\frac{\omega}{2})$
- Scaling doesn't change the rotation
  → we usually look at unit quaternions only
- q and $-q$ describe the same rotation, otherwise the mapping is unique
  - Every rotation is represented by exactly two unit quaternions
  - Every unit quaternion describes a rotation

Rotations in 3D: Quaternions

- Application: Interpolation of rotations
  - Assume you have an object under rotation R_1 and want to animate it to a new rotation R_2
  - Interpolating matrices fails
  - Interpolating Euler angles will result in very weird movements
  - Interpolating quaternions works better…
  - … when using **spherical interpolation**

Rotations in 3D: Quaternions

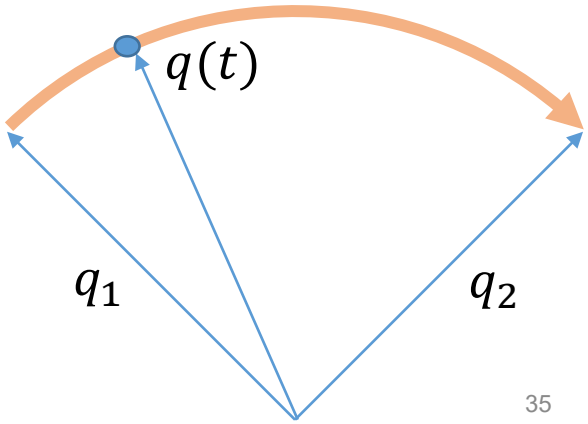
- Linear interpolation of unit vectors requires renormalization
- Velocity is not uniform



- Instead, we should directly interpolate on the sphere
  → spherical interpolation:

$$q(t) = \frac{\sin\big((1-t)\Omega\big)}{\sin\Omega} q_1 + \frac{\sin(t\Omega)}{\sin\Omega} q_2$$

$$\cos(\Omega) = q_1 \circ q_2$$



often called “slerp”

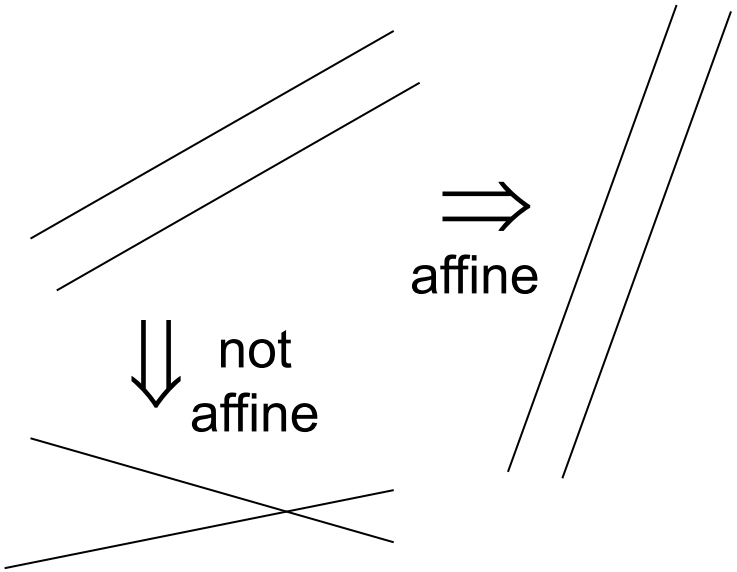
Affine Transformations

- Affine = combination of linear transformation and translation

$$x \rightarrow Ax + b$$

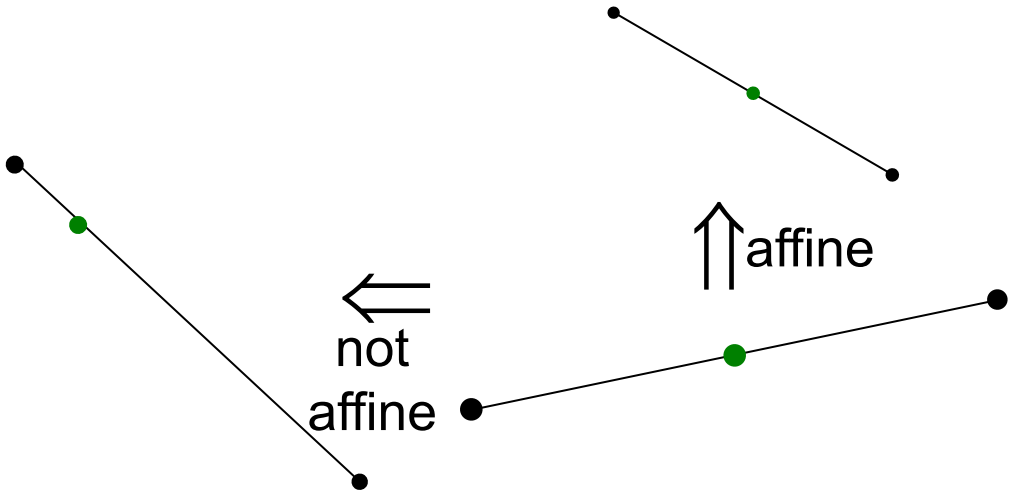
- Abstract characterization
  - Maps lines to lines
  - Parallel lines will be mapped to parallel lines
  - Division ratios are preserved
  - Angles are not preserved
  - Examples: rotations, translation, scaling, shears

Affine Transformations



parallel lines remain parallel

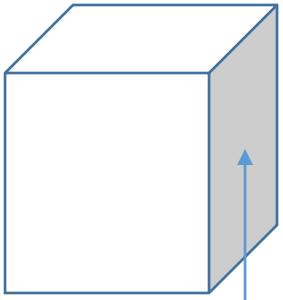
relative position of points on lines remain: ratios are maintained



Perspective ?

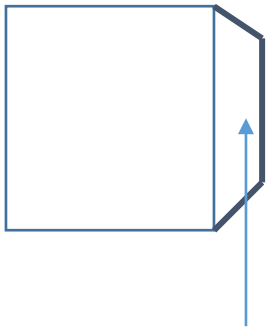
- Look at the image of a cube

parallel projection:
affine mapping



shearing

real perspective projection:
projective mapping



not affine

- → we need **projective mappings** to describe perspective
- → we need to reconsider homogeneous coordinates

Homogeneous Coordinates

i. simplify geometric computations
ii. unify points and vectors
iii. allow us to describe **projective transformations**

Homogeneous Coordinates – Fun Facts

- Implicit equation of a line:

$$s \cdot x + t \cdot y + u = 0$$

  → (s, t) is normal to line, i.e. line direction is $(-t, s)$ or $(t, -s)$

- We can thus represented this line by

$$l = (s, t, u)$$

  → a point $p = (x, y, 1)$ is on the line l if

$$p \circ l = 0$$

- The line through two points p and q can be computed as

$$l = p \times q$$

- The intersection point between two lines l and m is

$$p = l \times m$$

Points and Vectors

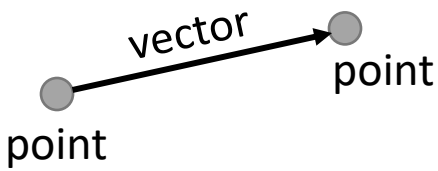
- Difference of **Points** and **Vectors (= Directions)**

Points

- Position in space
- Elements of **Euclidean space E**
- No algebraic structure
  - No addition
  - no multiplication with scalars, etc.

Vectors

- Difference between two points
- Elements of a **vector space** $\mathcal{V}$
- Algebraic structure
  - Addition of vectors
  - Multiplication with scalars
  - Linear combination of vectors
  - etc.



Points and Vectors

- Affine space: points and vectors
  - Given two points $a, b \in E$ there exists a unique vector in V: $\boldsymbol{ab} = \mathrm{b} - \mathrm{a}$
  - Algebraic structure:
    - **ab** = b – a difference of two points is vector
    - b = a + **ab** point + vector = point
    - **ab + bc = ac** vector + vector = vector

- Often, Euclidean space and vector space are mixed
- Point is then considered as a vector from origin to point

Points and Vectors

- Some special linear combinations of points p_i are valid operations:

- Barycenter, affine combination → barycentric coordinates
  - $p = \sum_i \lambda_i p_i$ is a point if $\sum_i \lambda_i = 1$
  - Example.: $0.5\ a\ +\ 0.5\ b\ =\ c\ =$ midpoint of a and b

- Generalized Difference:
  - $v = \sum_i \lambda_i p_i$ is a vector if $\sum_i \lambda_i = 0$
  - Example.: $-a\ +\ b\ =\ b - a\ =$ vector from a to b

Points and Vectors

- With homogeneous coordinates:

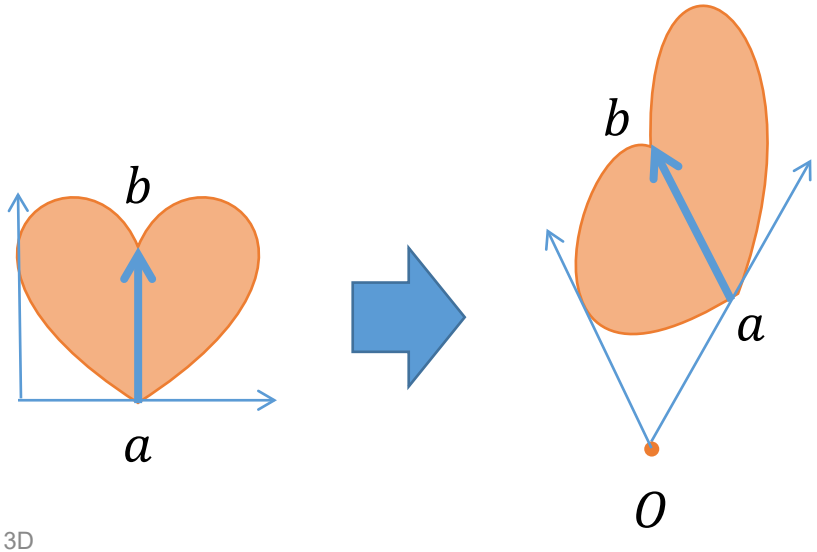
- Points $p = (x, y, 1)$
- Vectors $v = (x, y, 0)$

Affine Transformations + Hom. Coord.

- Remember: affine transformation of points

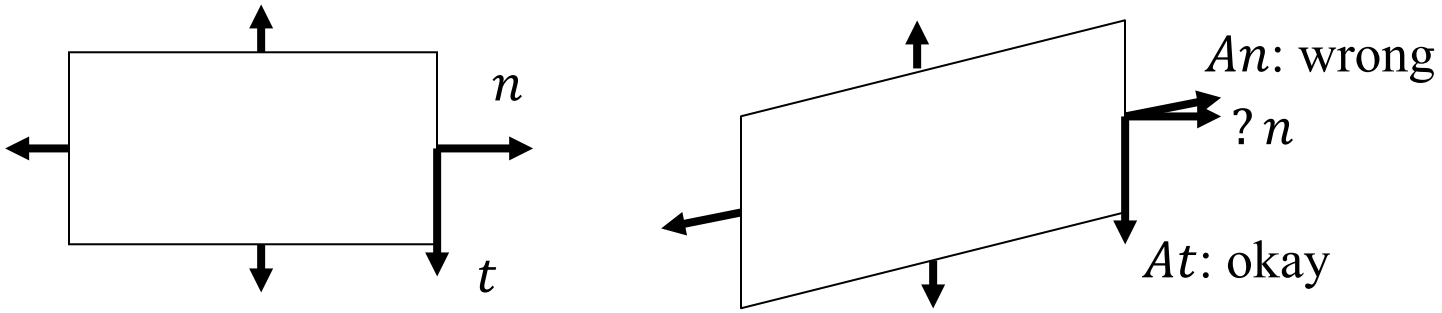
$$x \rightarrow Ax + t$$

- but vectors?
  - $v = b - a$
  - $v \rightarrow Ab + t - (Aa + t) = A(b - a) = Av$
  - becomes linear transformation with A, without translation t
- Directly works with vectors in the form $(x, y, 0)$!



Affine Normal Transformations

- More tricky: transforming normal vectors
  - Normals are perpendicular to tangent plane of a surface
  - Transformation with matrix A will differ from transformation of underlying surface



- We will need this later for lighting !

Affine Normal Transformations

- Matrix N for transformation of normal n
  - Perpendicular to tangent vector $n^T \cdot t = 0$
  - Transformed normal n' should also be perpendicular to transformed tangent $t' = At$
  - $n^T \cdot t = n^T \cdot I \cdot t$
    $= n^T \cdot A^{-1}A \cdot t$
    $= (n^T A^{-1}) \cdot \mathrm{t}' =^! 0$
  - $\rightarrow n' = (n^T A^{-1})^T = (A^{-1})^T n = A^{-T} n$
- Result: transform normal vectors by matrix $\boldsymbol{A^{-T}}$
- Remark: if M is orthogonal (rigid transformation) then $A^{-T} = A$
- Remark: length of normal vector can change!

Homogeneous Coordinates

- More general: lift 2D point to homogeneous coordinates:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} xw \\ yw \\ w \end{pmatrix}$$

- "Dehomogenize" → map back to 2D:

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} \rightarrow \begin{pmatrix} x/w \\ y/w \end{pmatrix}$$

- w is a "common denominator"

- $w = 0$ → point at infinity → direction → vector

Homogeneous Coordinates

- Transformations on homogeneous coordinates:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \overset{\cdot A}{\rightarrow} \begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} \rightarrow \begin{pmatrix} \frac{x'}{w'} \\ \frac{y'}{w'} \end{pmatrix}$$

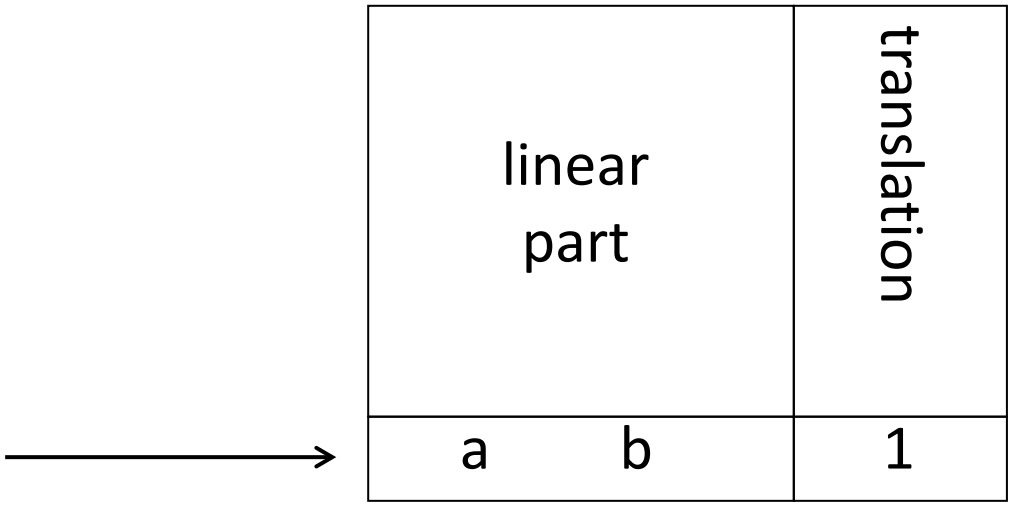
- For a 3x3 matrix $A = (a_{ij})$:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \frac{1}{a_{31}x + a_{32}y + a_{33}} \begin{pmatrix} a_{11}x + a_{12}y + a_{13} \\ a_{21}x + a_{22}y + a_{23} \end{pmatrix}$$

Homogeneous Coordinates

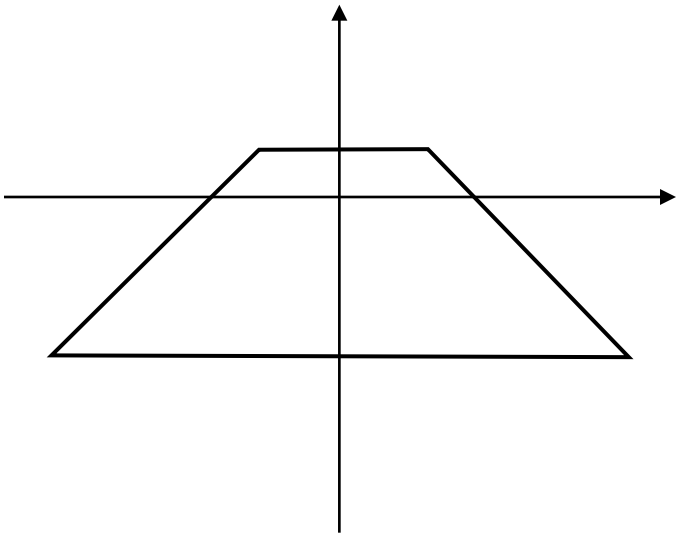
- When we use the last line of a matrix and homogeneous coordinates, we enter a new class of transformations:

  projective transformations



Projective Transformations

- Example $M = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 2 \end{pmatrix}$
- $(1,1) \rightarrow \left(\frac{2}{3}, \frac{1}{3}\right)$
- $(-1,1) \rightarrow \left(-\frac{2}{3}, \frac{1}{3}\right)$
- $(-1,-1) \rightarrow (-2,-1)$
- $(1,-1) \rightarrow (2,-1)$



looks like a perspective image of a square, right?

Projection and Perspective

- For each convex quadrilateral there is a unique projective transformation that maps the unit square to the quadrilateral

- Perspective is a projective mapping!

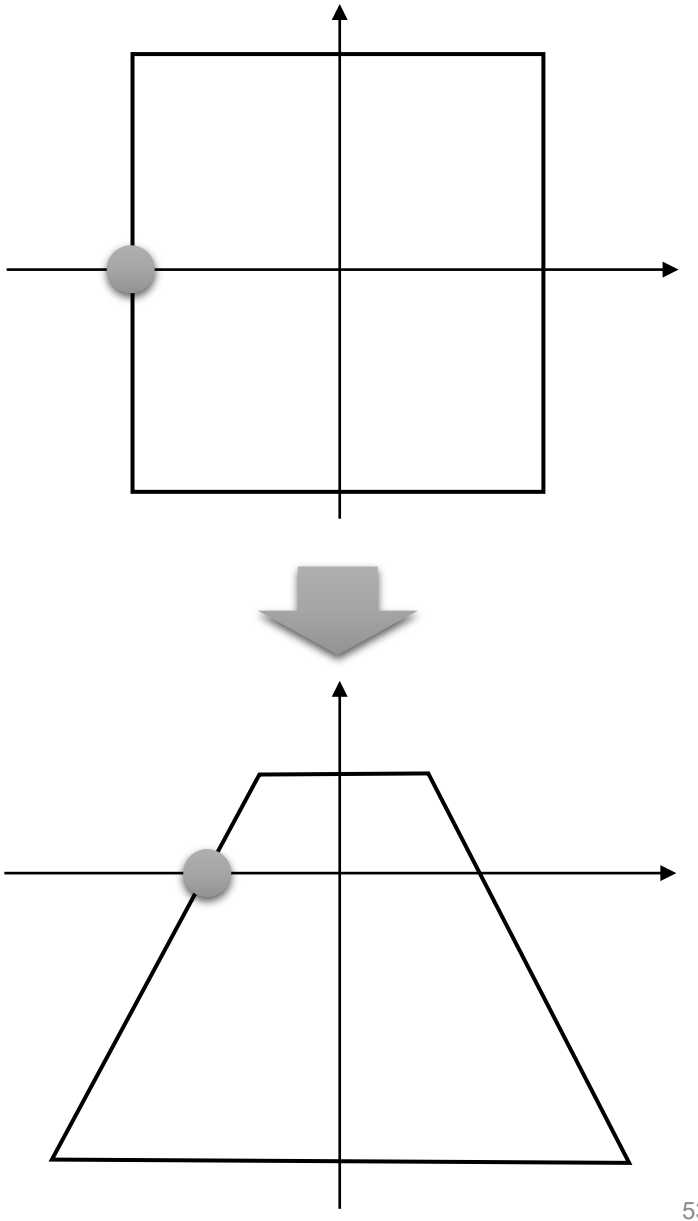
ikea.de

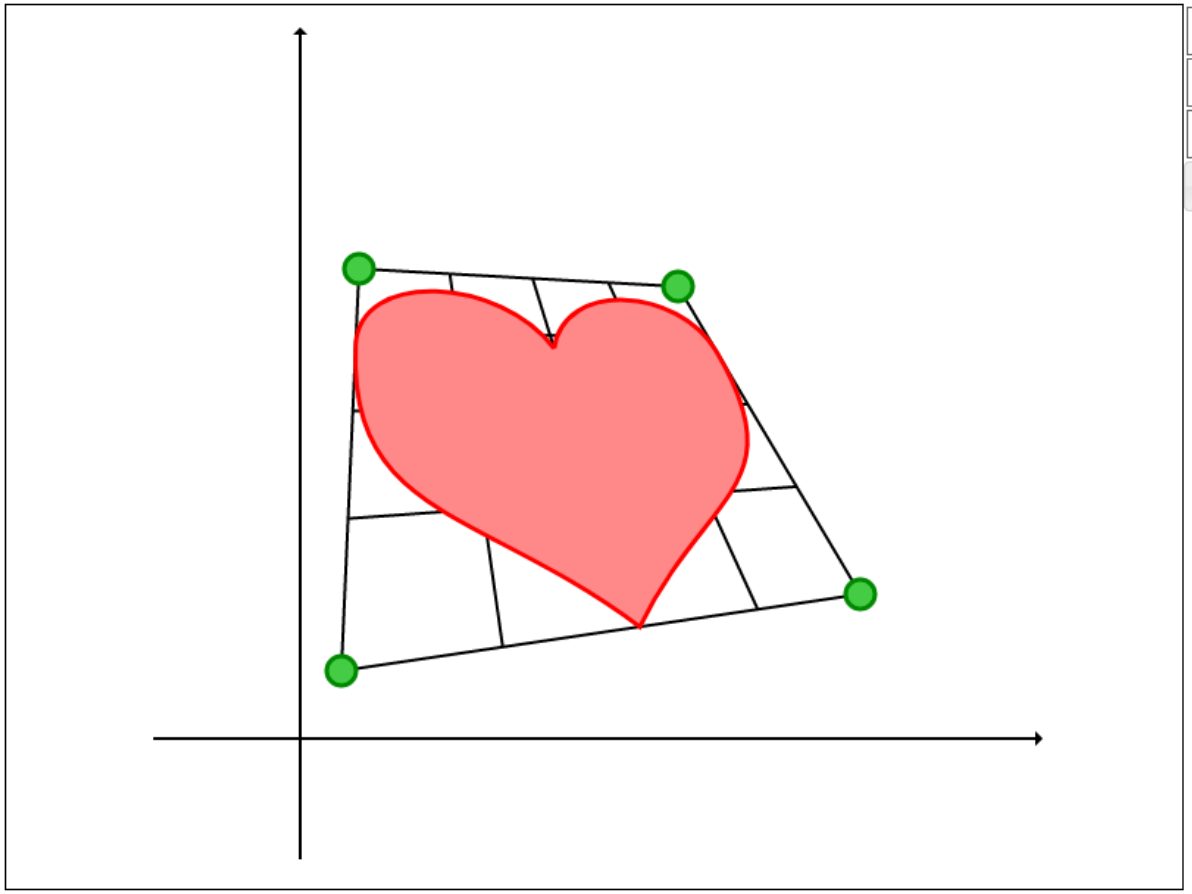
Projective Transformations

- Properties

- **Parallel lines do not remain parallel**
  → look at vertical lines
- **Lines are mapped to lines**
  → can be shown easily
- **Ratios are not preserved**
  → look at marked point
-

  → try it:



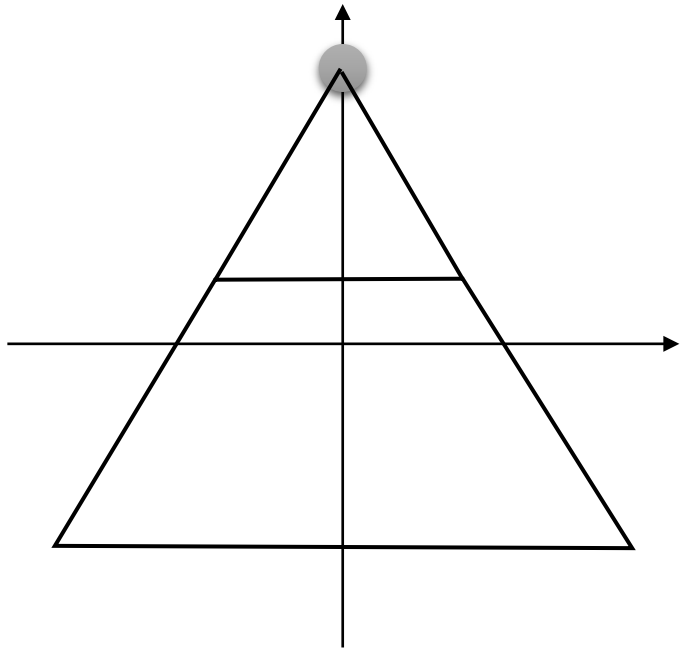


1.22	0.11	0.07
0.22	1.35	0.12
0.35	0.83	1.00

make affine

Projective Transformations

- points can be mapped to points at infinity and vice versa
- Example from before $M = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 2 \end{pmatrix}$
- $\lim_{y \to \infty} M \begin{pmatrix} 0 \\ y \end{pmatrix} = (0,1)$



Side Remark
Felix Klein's "Erlanger Programm"

Aus Wikipedia „Erlanger Programm"

- Die elementare euklidische Geometrie oder Kongruenzgeometrie ist die Geometrie des Anschauungsraumes, deren Transformationsgruppe die Gruppe der Bewegungen (also der Translationen, Drehungen oder Spiegelungen) ist, die alle **längen- und winkeltreue Abbildungen** sind.
- Verzichtet man bei den zugelassenen Transformationen auf die Längentreue und lässt auch Punktstreckungen zu, so erhält man die äquiforme Gruppe der Transformationen, die die **Ähnlichkeits- oder äquiforme Geometrie** kennzeichnet.
- Verzichtet man auch auf die Winkeltreue, so gelangt man zur Transformationsgruppe der bei Koordinatendarstellung linearen Transformationen, d.h. der Kollineationen, die das Teilverhältnis je dreier Punkte erhalten. Sie kennzeichnen die **affine Geometrie**.
- Fügt man schließlich zum Anschauungsraum noch unendlich ferne oder uneigentliche Punkte als Schnittpunkte von Parallelen hinzu, so lassen die Kollineationen in diesem Raum das Doppelverhältnis von je vier Punkten invariant und bilden die Gruppe der projektiven Transformationen, deren zugehörige Geometrie die **projektive Geometrie** ist.

Side side remark

- Felix Klein is also the inventor of the **Klein Bottle**
- We will come back to this later in this lecture

From Futurama

Next Lecture

- Real Perspective !