§3 Transformations and Projections

3.1 Coordinate systems

- **Examples:**

  Coordinate systems of objects and of the output device:

  - **Object system:** 3d-coordinate system of the object is aligned with the geometric properties of object, e.g. special directions, symmetries.
  - **Display system:** 2d-coordinate system of the output device or the output window is aligned with the geometry of the device, e.g. origin is the lower left corner, the x- and y-axes are parallel to the device boundaries.

  ► **Coordinate transformations** in $\mathbb{R}^2$ or $\mathbb{R}^3$ transform the object system into the device system:

  ► translations, scalings, rotations, projections.

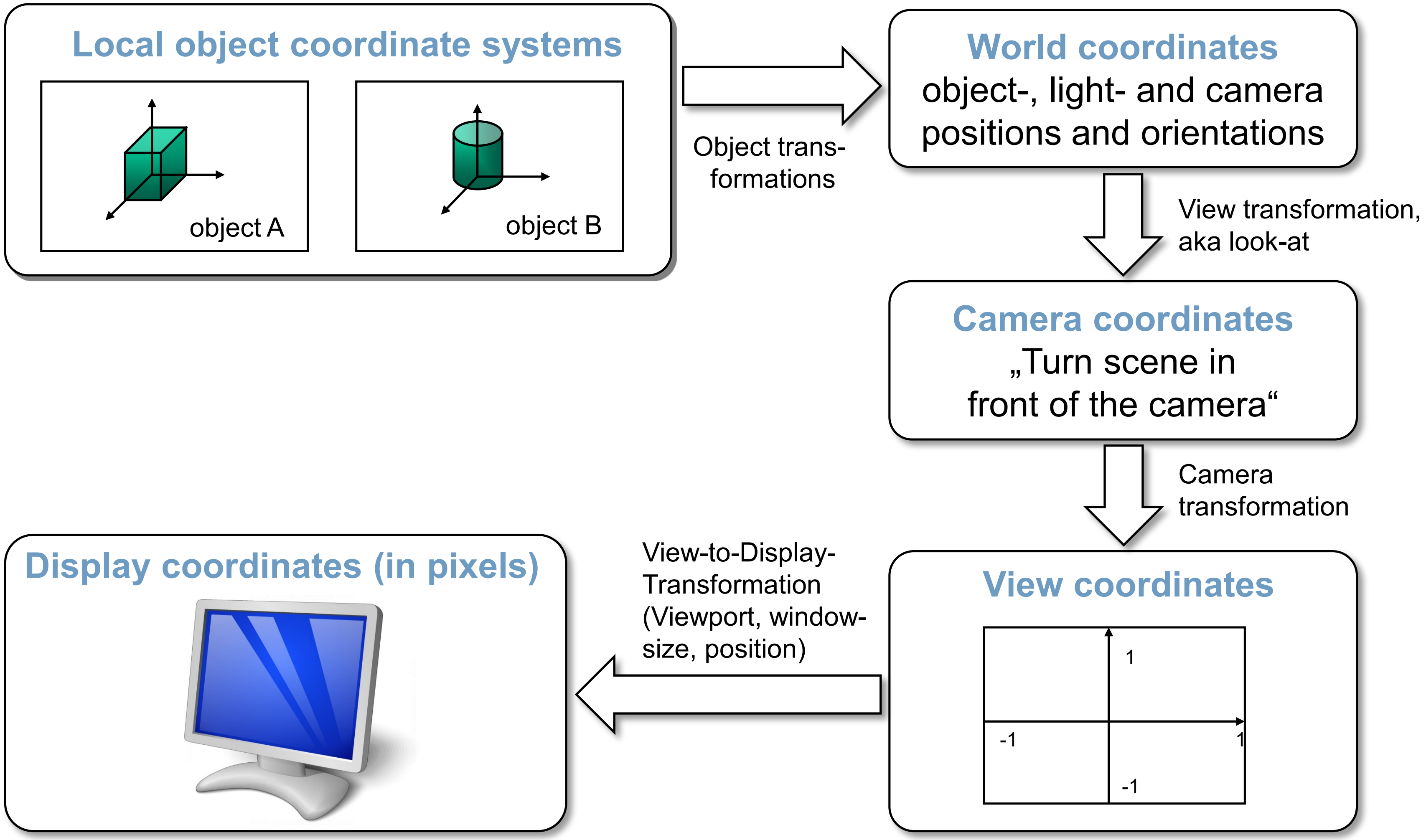
3.1 Coordinate systems

Motivation

Object definition, scene modeling, view definition, animation, rendering, output, ...

- ▶ Use of many different coordinate system.
- ▶ Coordinate transformations in 2d- or 3d-space (translation, scaling, rotation,...) for the definition of position and orientation!
- ▶ Matrix operations
- ▶ Hard- and software-supported transformation between coordinate systems.
  - ▶ These are, except for the translation, basis transformations in $\mathbb{R}^3$.

3.1 Coordinate systems


Local object coordinate systems
object A
object B
Object trans-formations
World coordinates
object-, light- and camera positions and orientations
View transformation, aka look-at
Camera coordinates
„Turn scene in front of the camera“
Camera transformation
View coordinates
1
-1
1
-1
View-to-Display-Transformation (Viewport, window-size, position)
Display coordinates (in pixels)

3.1 Coordinate systems

3.1.1 Object coordinate system
(aka local/modeling coordinate system, object space)

- Individual 3d-coordinate system for every single object.
- Simplifies modeling and animation.
- Often defined via
  - geometric properties of the object, e.g. symmetries or particular directions (e.g. height), or
  - desired behavior of the object, e.g. flight along a given curve with simultaneous rotation.
- Also used for light sources.

3.1 Coordinate systems

3.1.2 World coordinate system

(aka global/scene coordinate system, scene/world space)

- Here „lives“ the complete scene in **3d**.
- Objects (object coordinate systems) are placed in the world via affine transformations.
- For rendering: Transform local object coordinates to global object coordinates using **object transformations**.
  - For animations, i.e. for animated objects: dynamically for each frame.
- Here also: illumination of the scene.

3.1 Coordinate systems

3.1.3 Camera coordinate system (aka eye coordinate system)

- Conceptually: virtual camera in **3d** with its necessary parameters.
- Anchored in the world system (via affine transformations).
- This is the view of the observer of the scene, i.e. here also the visible image area is (view window see page §3-10)!
- For rendering: Usually transformation of global coordinates into camera coordinates („Turn scene in front of the camera“).
  - For animations, i.e. for an animated camera: dynamically in each frame!
- Virtual camera determines also the used type of projection.

3.1 Coordinate systems

3.1.4 Image coordinate system

(aka view/view plane coordinate system, image space)

- Coordinate system in the **2d**-image plane.
- A projection (**camera transformation**) maps 3d-coordinates to 2d-coordinates in the image plane.
- A **window** defines the visible sub-area of the image plane.

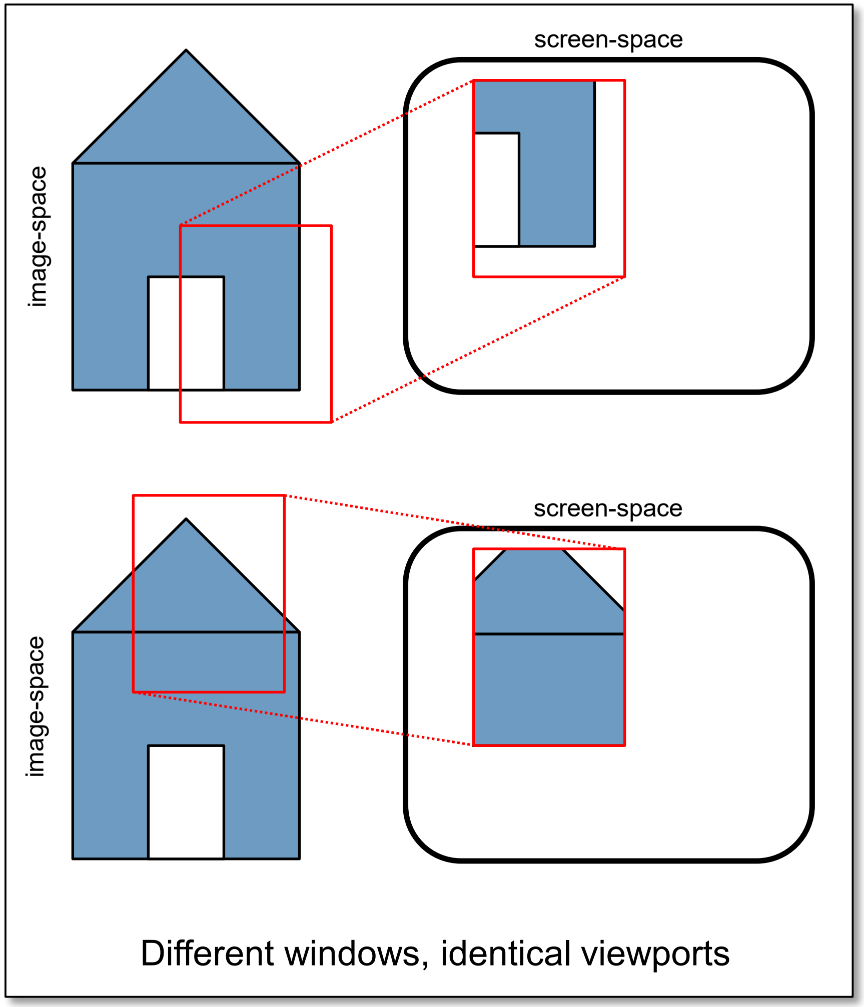
3.1 Coordinate systems

3.1.5 Display coordinate system

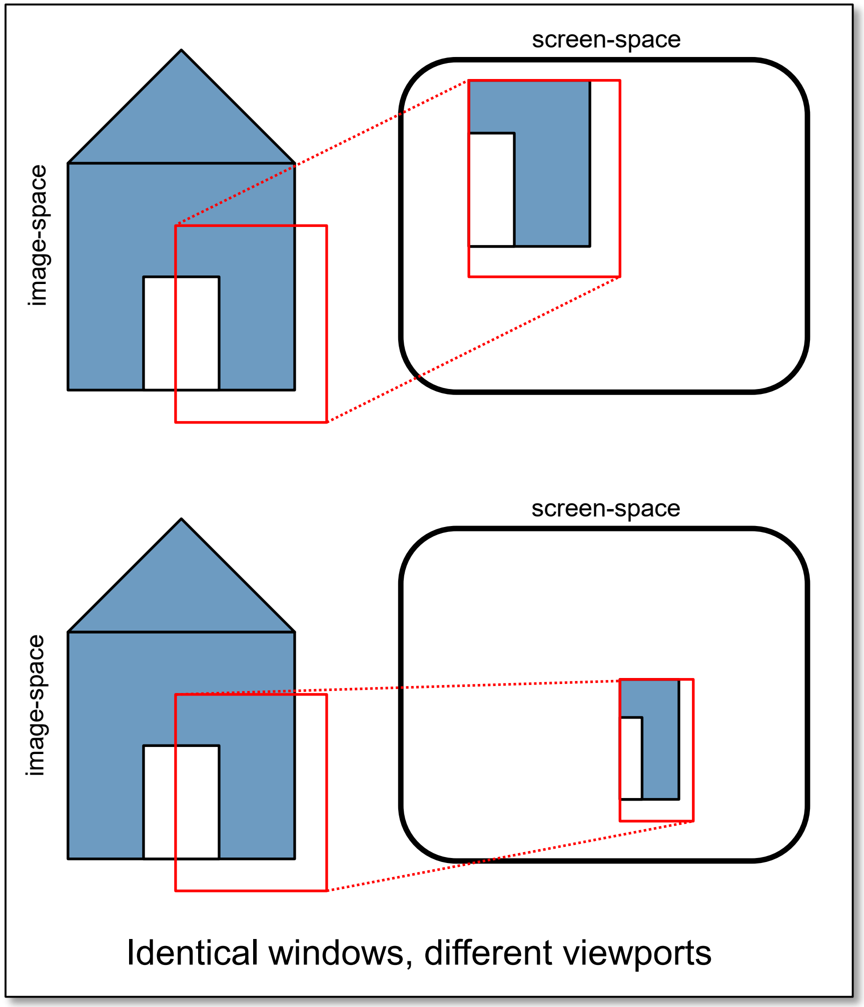
(aka device coordinate system, screen space)

- Defines the **2d**-coordinate system of the display or output device.
- Coordinates are given in **pixels**.
- A **viewport** defines a sub-area of the display, where the content of the window is displayed.
- Window-viewport-transformations (**windowing**) are 2d-2d-transformations.

3.1 Coordinate systems



Different windows, identical viewports



Identical windows, different viewports

3.1 Coordinate systems

- We assume in general a
  - **orthonormal:** pair-wise perpendicular and normalized basis vectors,
  - **Cartesian:** orthonormal + right-handed,

  coordinate system.

- But be **aware of confusions!**

For transformations exist several

equivalent interpretations

depending on the

reference system!

3.1 Coordinate systems

3.1.6 Interpretation of transformations

a) „Transformation of points“

- Coordinate system S is fixed.
- Point P changes its position and thus its coordinates in the same system S, i.e. **point transformation**.
- ► In the (global) coordinate system S the transformation acts on the coordinates of P.
- Involved coordinate systems: **one**.

3.1 Coordinate systems

b) „Transformation of coordinate systems“

- Coordinate system S is transformed using the **inverse point transformation**.
  - ▶ Coordinate system S moves.
  - ▶ Point P is fixed, but changes its coordinates, since it is described in the new system.
- Involved coordinate systems: only one, but in a before and after the transformation version.
- „Transformation between coordinate systems.“
  - ▶ Change of basis.

3.1 Coordinate systems

c) „Transformation using coordinate system“

- The local coordinate system S, that contains point P, is moved using a **point transformation** in the global coordinate system S'.
- ► Local coordinate system S moves.
- Place point P (with its „old“ coordinates with respect to S) in the moved local coordinate system.
- ► Point P changes its position, but not its coordinates with respect to S.
- Involved coordinate systems: two (local, global).
- Used for modeling and animation.

3.1 Coordinate systems

We will use simultaneously the two interpretations

„Transformation of coordinate systems“ (Interpretation b)) and

„Transformation of points“ (Interpretation a)).

Input:

- coordinate system S' (e.g. world system) determined by

  $S'\colon\ (O'; x_1', x_2'),$ and

- coordinate system S (e.g. object system) determined by

  $S\colon\ (O; x_1, x_2).$

Output:

- Transformation from S to S' (Interpretation b)) and
- change of point coordinates from S to S' (Interpretation a)).

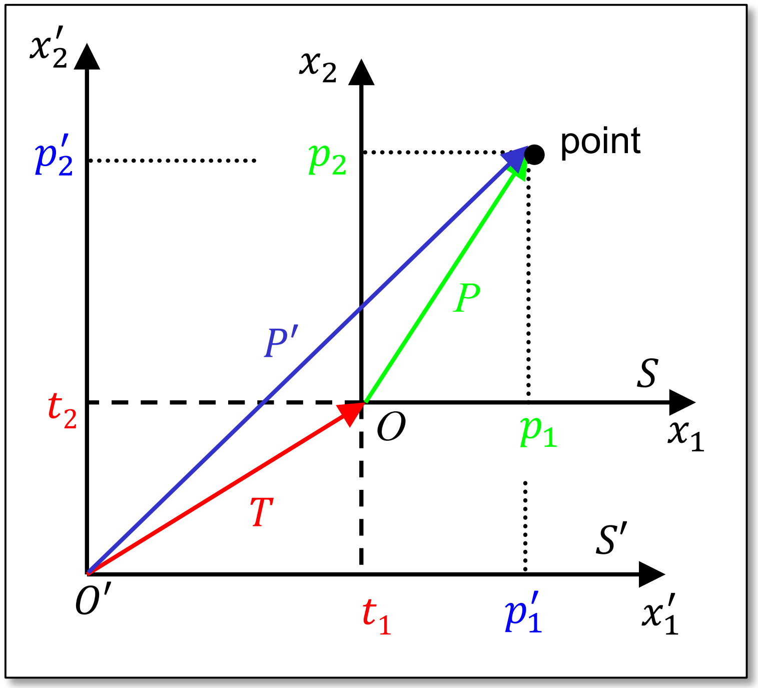
3.2 Affine transformations in 2d

3.2.1 Translation

Assumption: The coordinate axes are parallel to each other.

Interpretation b): For system S' we get

- S' results from S by a translation by $-T$, where $T = (t_1, t_2)^t$ are the coordinates of the origin O of S in the system S'.



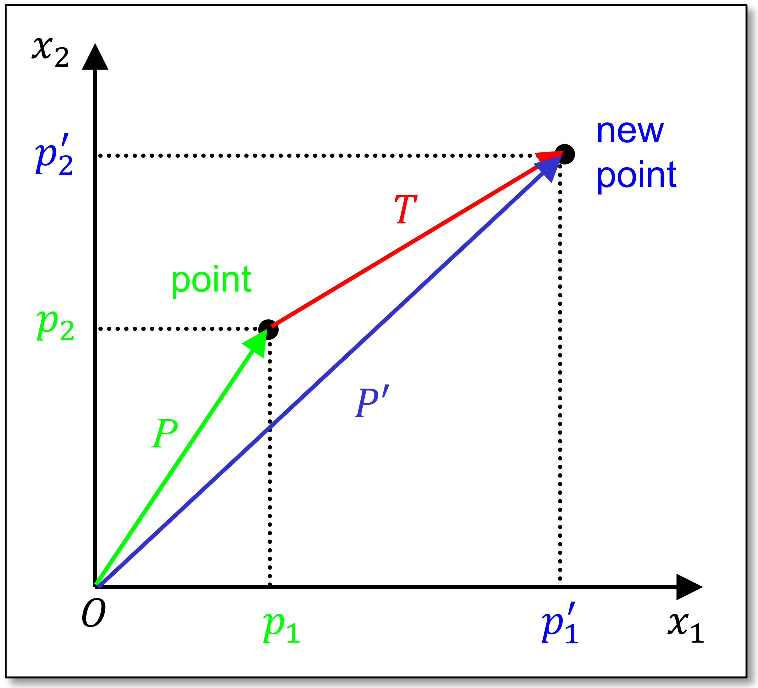
▶ The point has

- in S the coordinates $P = (p_1, p_2)^t$,
- in S' the coordinates $P' = T + P = (t_1, t_2)^t + (p_1, p_2)^t$.

3.2 Affine transformations in 2d

Interpretation a): We get for the point.

- Point with coordinates P is translated by T ...
- ... and we get a new point with coordinates $P' = T + P$.



- Both interpretations can be written in vector form as

$$\begin{pmatrix} p_1' \\ p_2' \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} + \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}.$$

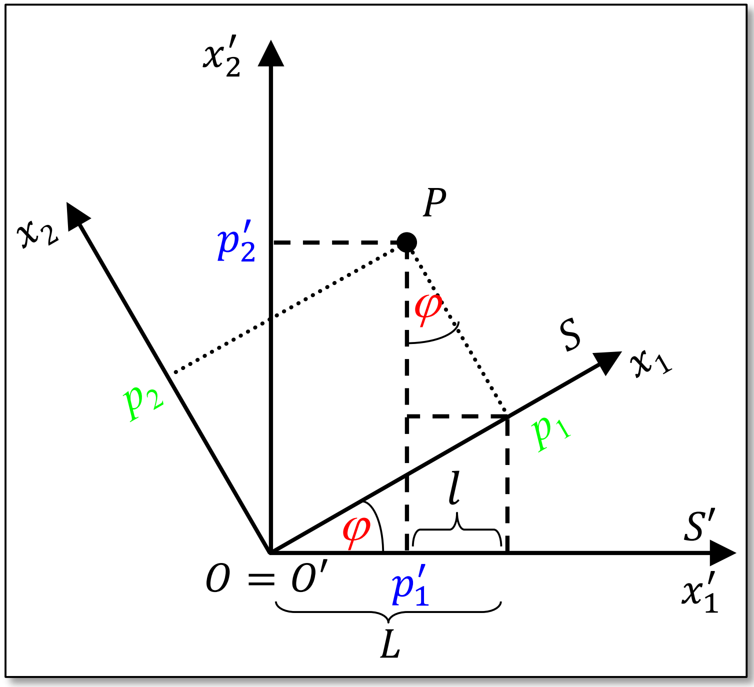
- Short-hand: $P' = T + P$.

3.2 Affine transformations in 2d

3.2.2 Rotation

Assumption: The coordinate systems have the same origin $O = O'$.

Interpretation b): Rotation of system S with respect to S' around angle φ around O yields



- The system S' results from S by a rotation around $-\varphi$.
- Using $l/p_2 = \sin\varphi$ and $L/p_1 = \cos\varphi$,

  yields $p_1' = L - l = p_1 \cos\varphi - p_2 \sin\varphi$.

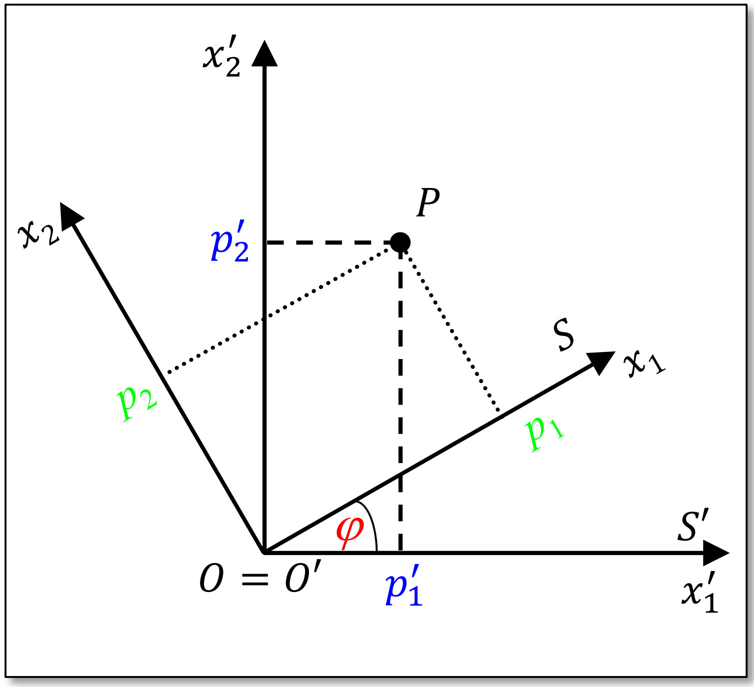
  Analog $p_2' = \;\dots\; = p_1 \sin\varphi + p_2 \cos\varphi$.

3.2 Affine transformations in 2d

- Short-hand:

$$P' = \begin{pmatrix} p_1' \\ p_2' \end{pmatrix} = \begin{pmatrix} p_1 \cos\varphi - p_2 \sin\varphi \\ p_1 \sin\varphi + p_2 \cos\varphi \end{pmatrix}$$

$$= \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$



- In Vector-matrix-notation:

$$P' = R(\varphi) \cdot P,$$

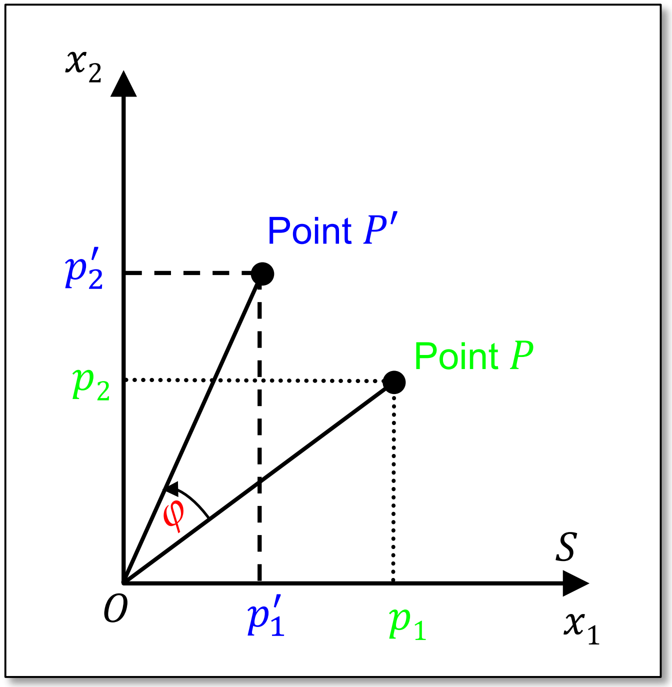
with the orthonormal rotation matrix: $R(\varphi) = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}$

Remark: R is orthonormal, iff $R^{-1} = R^t$.

3.2 Affine transformations in 2d

Interpretation a)

Matrix $R(\varphi)$ rotates a point around the angle φ in positive direction around the origin O of a fixed coordinate system.



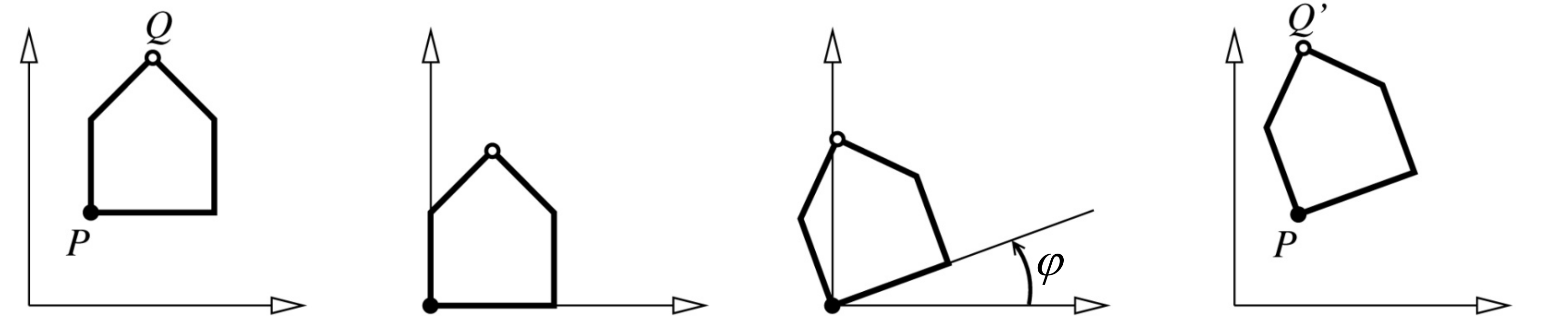
3.2 Affine transformations in 2d

Rotation around an arbitrary point P:

(1) translation of P into the origin,

(2) rotation around the origin around the angle φ,

(3) translation of P to its original position.



Original house | After translation | After rotation | After back-translation

Remark: Matrix multiplication is not commutative.

► The order of matrices must be the same as the order of rotations .

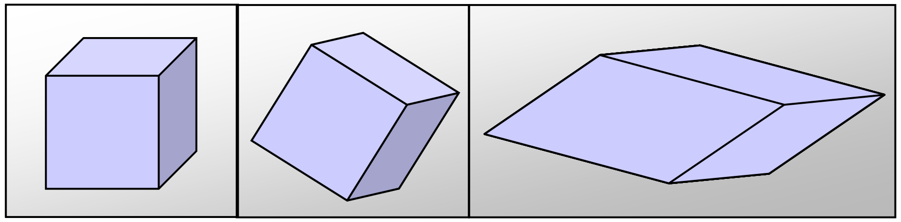
3.2 Affine transformations in 2d

3.2.3 Scaling

- To enlarge or shorten the system S, a scaling is used:

$$p_1' = \lambda_1 \cdot p_1$$
$$p_2' = \lambda_2 \cdot p_2$$

- In vector-matrix-notation: $P' = S \cdot P$ with $S = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$.
- **Attention!** Scalings can change lengths and angles, i.e. orthogonality (angles), orthonormality (length), right-handedness (orientation)!



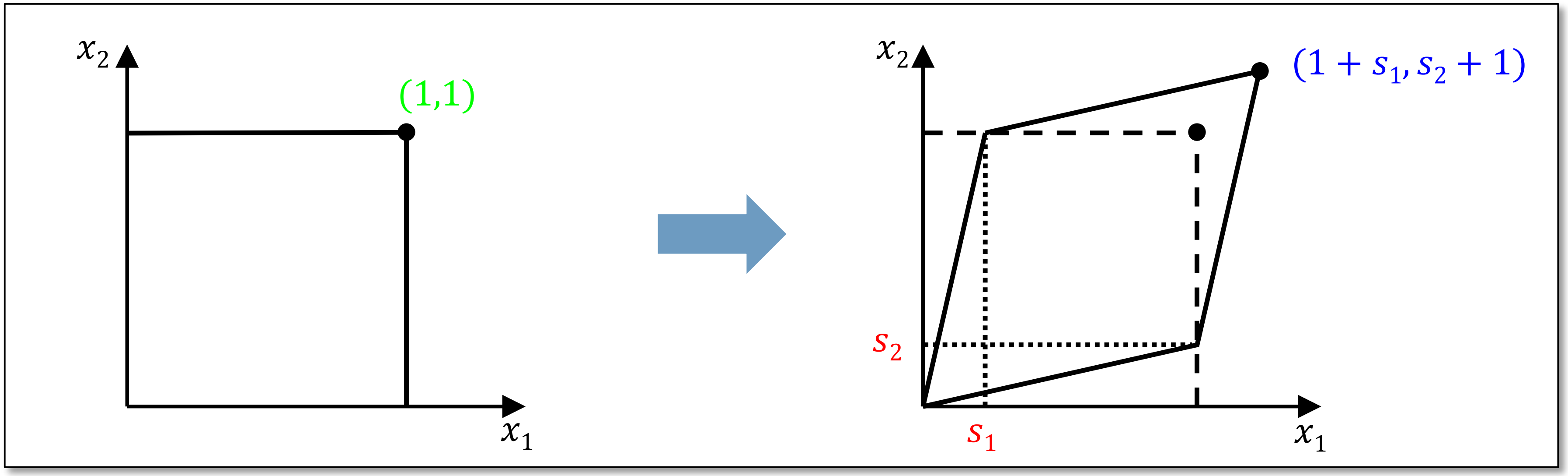
3.2 Affine transformations in 2d

3.2.4 Shearing

- A Shearing is given by:

$$\begin{aligned} p_1' &= p_1 + s_1 \cdot p_2 \\ p_2' &= s_2 \cdot p_1 + p_2 \end{aligned}$$

- In Vector-matrix-notation: $P' = S \cdot P$ with $S = \begin{pmatrix} 1 & s_1 \\ s_2 & 1 \end{pmatrix}$.



3.2 Affine transformations in 2d

3.2.5 Affine transformations

- Affine transformations can be represented as a combination of a linear map A and a translation T

$$P' = A \cdot P + T.$$

- All transformations seen so far (translation, rotation, scaling, shearing) are affine transformations.

3.2 Affine transformations in 2d

Affine invariance of ratios:

For an affine transformation F and points P and Q we always get

$$F(\lambda P + (1-\lambda)Q) = \lambda \cdot F(P) + (1-\lambda) \cdot F(Q)$$

for $0 \leq \lambda \leq 1$

and for general points P_i

$$F\left(\sum_{i=0}^{n} \alpha_i P_i\right) = \sum_{i=0}^{n} \alpha_i F(P_i)$$

iff

$$\sum_{i=0}^{n} \alpha_i = 1 .$$

3.2 Affine transformations in 2d

Properties of affine transformations:

1. An affine transformation F does not change ratios $\lambda : (1-\lambda)$.
2. The image of a line segment from Q to P is again a line segment under an affine transformation F.
3. It suffices to affinely transform the end points Q and P of a line segment; intermediate points can be computed by linear interpolation of $F(Q)$ and $F(P)$.
4. Under an affine transformation parallel lines remain parallel.

- **BUT:** Angles, lengths, and orientations can change!

3.2 Affine transformations in 2d

Further affine transformations:

- Reflection at the line $y = x$: $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.
- Reflection at the line $y = -x$: $A = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$.
- Reflection at the x-axis: $A = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$.
- Reflection at the y-axis: $A = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$.
- Reflection at the origin: $A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$.

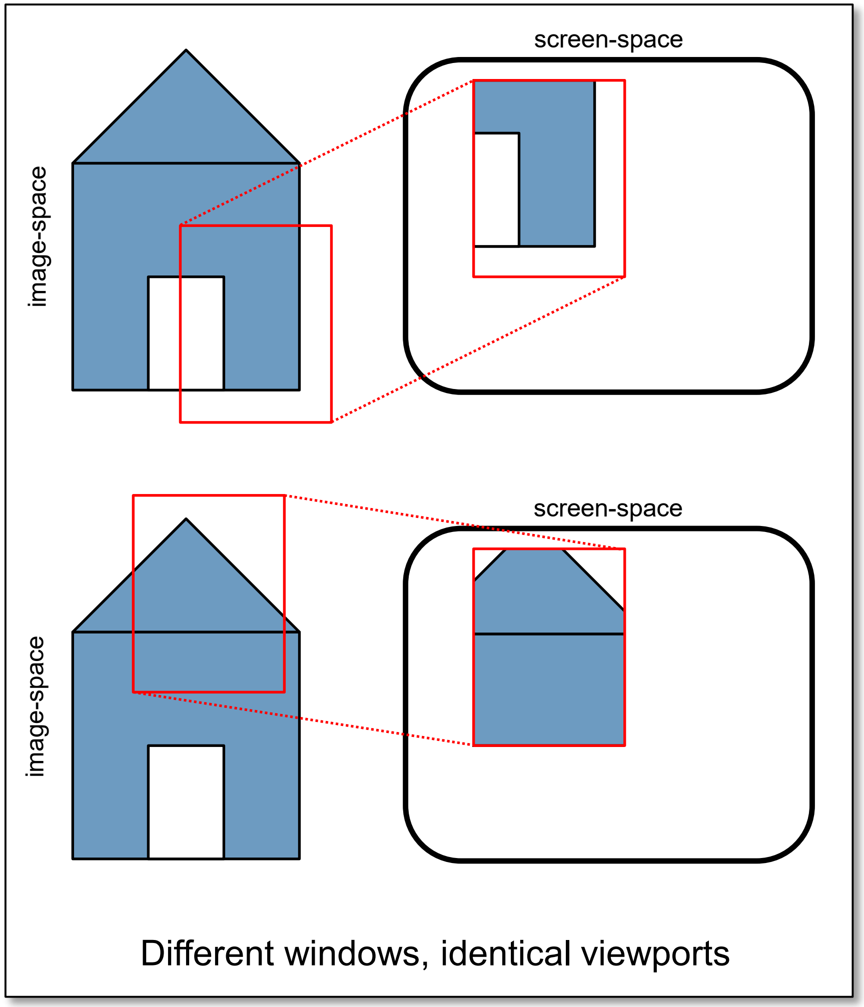
3.3 Windowing

Window:
- Also „view window“
- Defines window in the image plane.
- Defines, which sub-area of the scene is to be rendered.

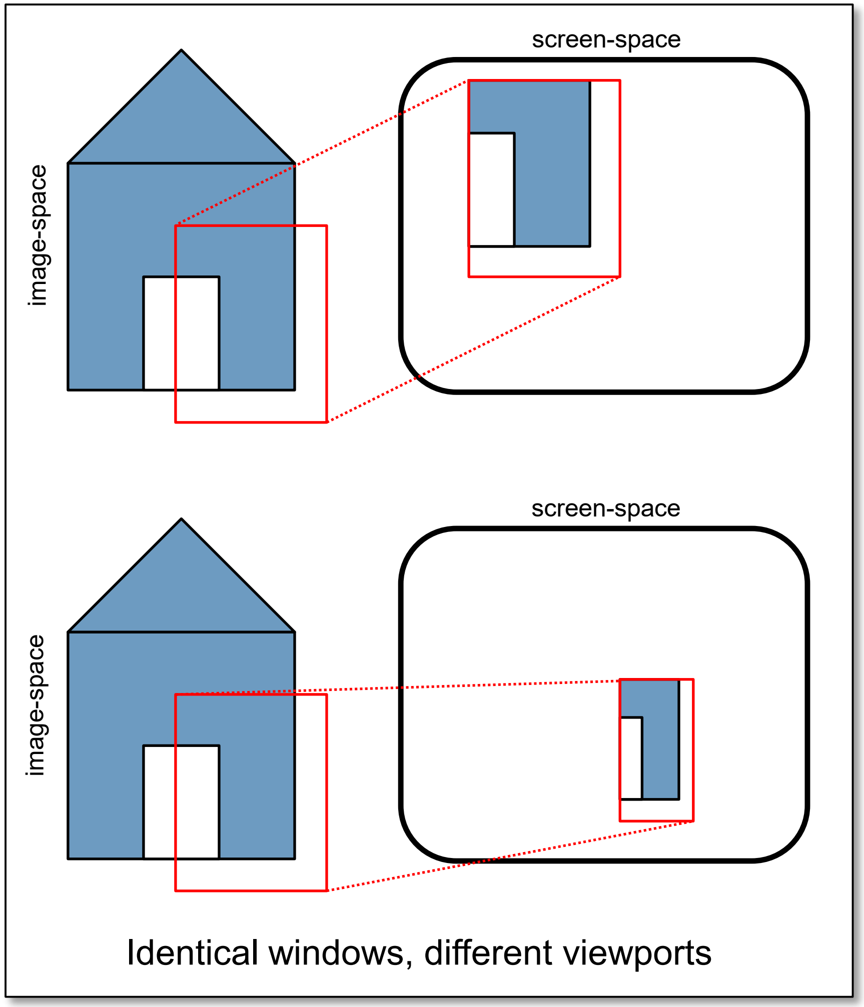
Viewport:
- Defines sub-area of the display, where the content of the window is visualized.

- Usually the window and the viewport are axis-aligned rectangular areas.
- The windowing-operations (window-viewport-transformation) are a combination of elementary translations and scalings.

3.3 Windowing

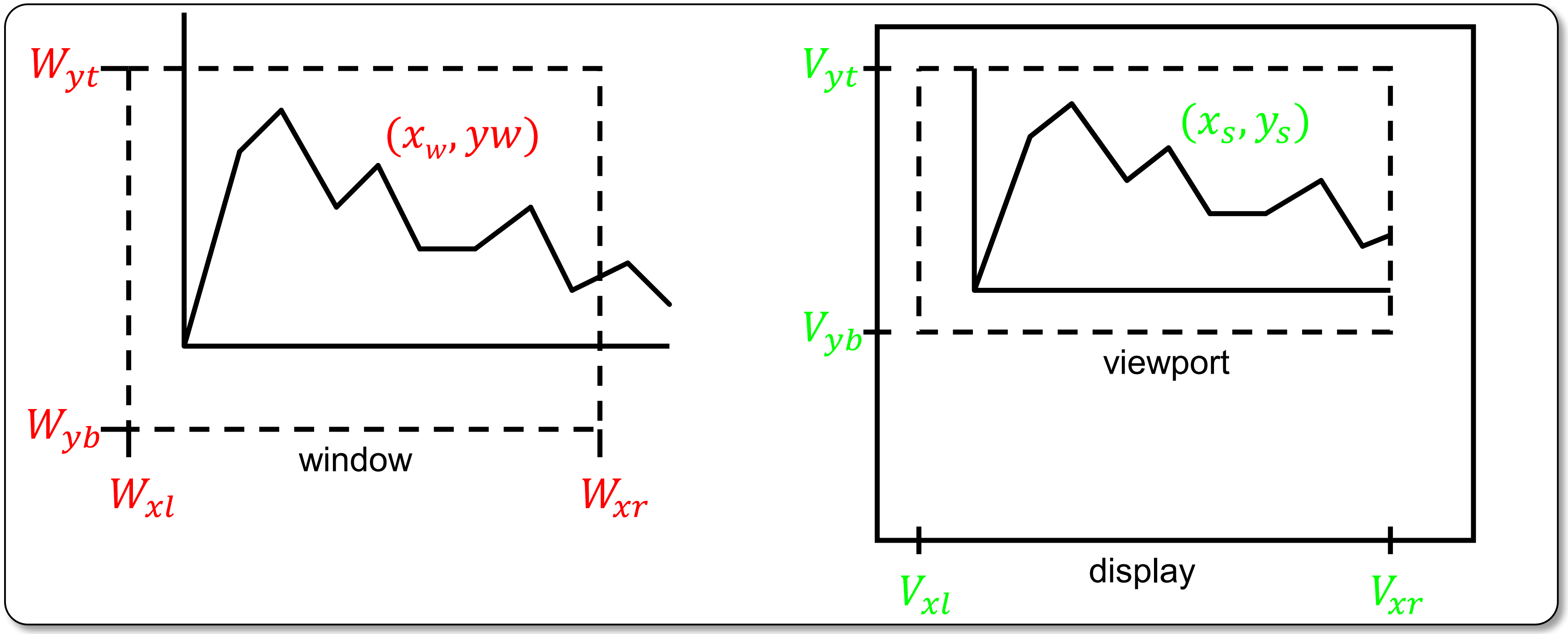


Different windows, identical viewports



Identical windows, different viewports

3.3 Windowing



x_w, yw	point coordinates in window
x_s, ys	point coordinates on the display
$W_{xl}, W_{xr}, W_{yb}, Y_{yt}$	coordinates of the window
$V_{xl}, V_{xr}\ V_{yb}, V_{yt}$	coordinates of the viewport in display coordinate system

3.3 Windowing

Transformation in 3 steps

1) Translation of image into the origin

$$x' = x_w - W_{xl},$$

$$y' = y_w - W_{yb}.$$

2) Scaling to size of viewport

$$x'' = x' \cdot (V_{xr} - V_{xl})/(W_{xr} - W_{xl}),$$

$$y'' = y' \cdot (V_{yt} - V_{yb})/(W_{yt} - W_{yl}).$$

3) Translation of image to viewport position

$$x_s = x'' + V_{xl},$$

$$y_s = y'' + V_{yb}.$$

3.3 Windowing

Condensed to

$$x_s = a \cdot x_w + b,$$

$$y_s = b \cdot y_w + d,$$

with $a = (V_{xr} - V_{xl})/(W_{xr} - W_{xl}),$ $\quad b = V_{xl} - a \cdot W_{xl},$

$c = (V_{yt} - V_{yb})/(W_{yt} - W_{yb}),$ $\quad d = V_{yb} - c \cdot W_{yb}.$

- Point transformation uses only two multiplications and two addition.

3.4 Homogeneous coordinates

- The serial application of a rotation R, translation T, and scaling S yields the equation

$$P' = S \cdot (T + R \cdot P).$$

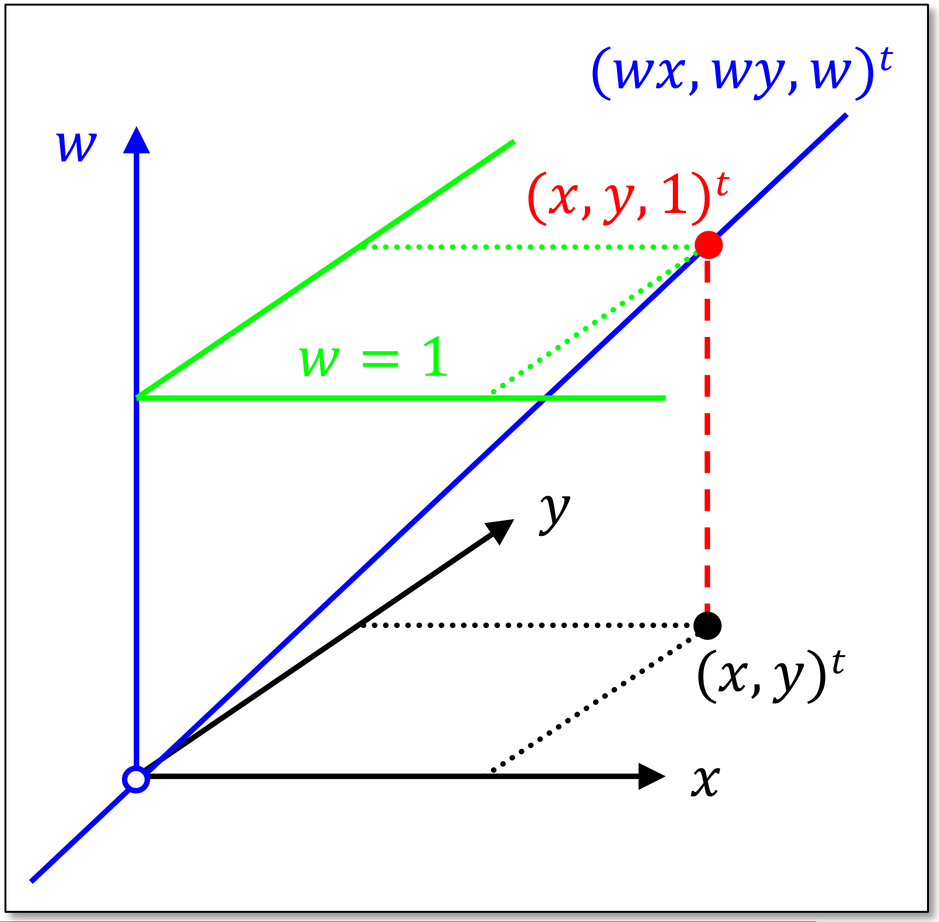
- If multiple of these transformation have to be computed, the addition in the above equation is annoying.
- In particular, today's GPUs implement matrix multiplications in hardware.
- It would be much better, if all affine transformations were represented as matrix multiplications

$$P' = M_n \cdots M_3 \cdot M_2 \ \cdot M_1 \cdot P.$$

3.4 Homogeneous coordinates

Lifting the problem to the next higher dimension:

- The triple $(wx, wy, w)^t$, $w \neq 0$, represents the homogenous coordinates of the point $(x, y)^t \in \mathbb{R}^2$.
- There are infinitely many such representations for the same point.
- Use the so-called standard representation $w = 1$.
- Now, a point $P = (x, y)^t \in \mathbb{R}^2$ has the unique homo-genous coordinates $(x, y, 1)^t$.
- Analogously for points in $\mathbb{R}^3$.



3.4 Homogeneous coordinates

Representation of affine transformations in homogenous coordinates:

- **Translation** of point$(x,y)^t$ by vector $(t_1,t_2)^t$:

$$\begin{pmatrix} 1 & 0 & t_1 \\ 0 & 1 & t_2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_1 \\ y + t_2 \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$$

- **Rotation** of point$(x,y)^t$ around angle φ arand the origin:

$$\begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x \cdot \cos\varphi - y \cdot \sin\varphi \\ x \cdot \sin\varphi + y \cdot \cos\varphi \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$$

3.4 Homogeneous coordinates

- **Scaling** of point $(x, y)^t$ by factors λ_1 and λ_2:

$$\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda_1 \cdot x \\ \lambda_2 \cdot y \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}.$$

- **Rotation** of point $(x, y)^t$ **around a point** $P = (px, py)^t$ around angle φ:

$$\begin{pmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$$

3.5 Affine transformations in 3d

3.5.1 Translation

The translation of a point $(x, y, z)^t$ along the translation vector $(t_x, t_y, t_z)^t$ yields the point $(x', y', z')^t$ with

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{=T(t_x,t_y,t_z)} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}.$$

3.5 Affine transformations in 3d

3.5.2 Scaling

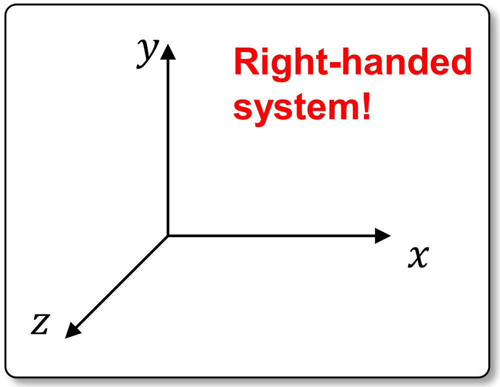
A scaling by the factors λ_1, λ_2, and λ_3 in all three axis directions yields

$$\begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda_1 \cdot x \\ \lambda_2 \cdot y \\ \lambda_3 \cdot z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}.$$

3.5 Affine transformations in 3d

3.5.3 Rotation

- All rotations in mathematically positive orientation (i.e. counter-clockwise).
  - The observer sits on the axis of rotations and looks towards the origin of the coordinate system.



- First only rotations around a single coordinate axis around the angle φ.

► Transformation matrices $R_x(\varphi), R_y(\varphi), R_z(\varphi)$.

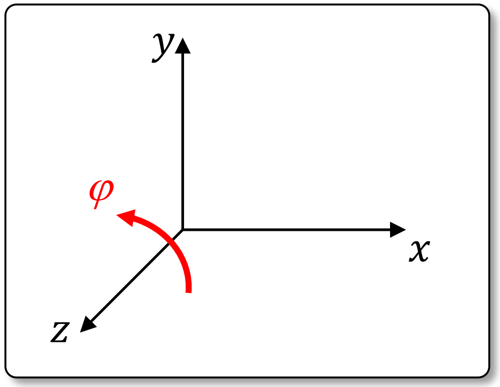
- We use interpretation a):

„global fixed coordinate system;
point is transformed (rotated)“

3.5 Affine transformations in 3d

Rotation around the z-axis

The rotation of a point $(x, y, z)^t$ around the z-axis around the angle φ yields point $(x', y', z')^t$ with



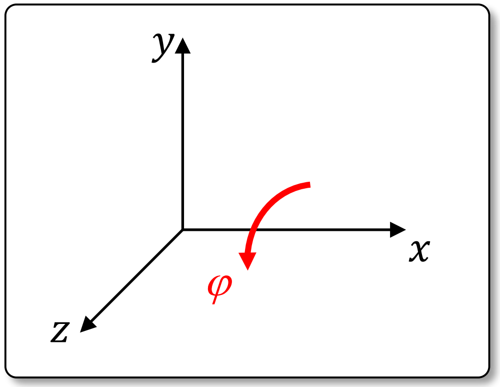
$$\underbrace{\begin{pmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{=R_z(\varphi)} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \cdot \cos\varphi - y \cdot \sin\varphi \\ x \cdot \sin\varphi + y \cdot \cos\varphi \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

Note: Rotation around the angle φ around the z-axis corresponds to the 2d-case, where the z-coordinate is constant!

3.5 Affine transformations in 3d

Rotation around the x-axis

The rotation of a point $(x, y, z)^t$ around the x-axis around the angle φ yields point $(x', y', z')^t$ with



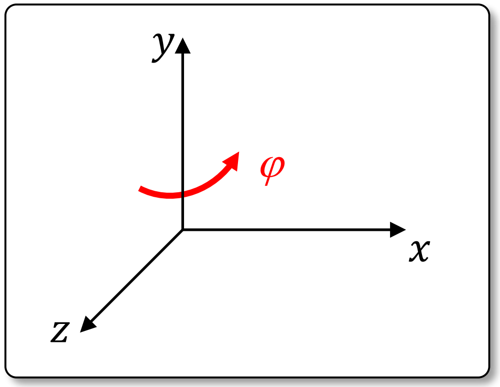
$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{=R_x(\varphi)} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \cdot \cos\varphi - z \cdot \sin\varphi \\ y \cdot \sin\varphi + z \cdot \cos\varphi \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

Note: Rotation around the angle φ around the x-axis corresponds to the 2d-case, where the x-coordinate is constant!

3.5 Affine transformations in 3d

Rotation around the y-axis

The rotation of a point $(x,y,z)^t$ around the y-axis around the angle φ yields point $(x',y',z')^t$ with



$$\underbrace{\begin{pmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{=R_y(\varphi)} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x\cdot\cos\varphi + z\cdot\sin\varphi \\ y \\ -x\cdot\sin\varphi + z\cdot\cos\varphi \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

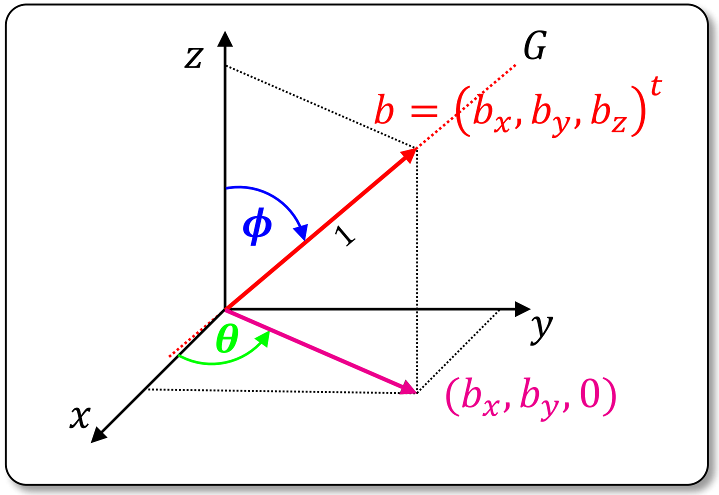
Note: Rotation around the angle φ around the y-axis corresponds to the 2d-case, where the y-coordinate is constant!

3.5 Affine transformations in 3d

Rotation around arbitrary axis G

- **Goal**: Rotation $R_G(\psi)$ of a point P around an arbitrarily oriented axis G in space around an angle ψ.
- **Special case:** Axis of rotation G passes through the origin and is spanned by vector $b = (b_x, b_y, b_z)^t$ with $||b|| = 1$:

$$G = \{\lambda \cdot b : \lambda \in \mathbb{R}\}.$$



$$\Rightarrow \begin{cases} b_x = \sin\phi \cdot \cos\theta \\ b_y = \sin\phi \cdot \sin\theta \\ b_z = \cos\phi \end{cases}$$

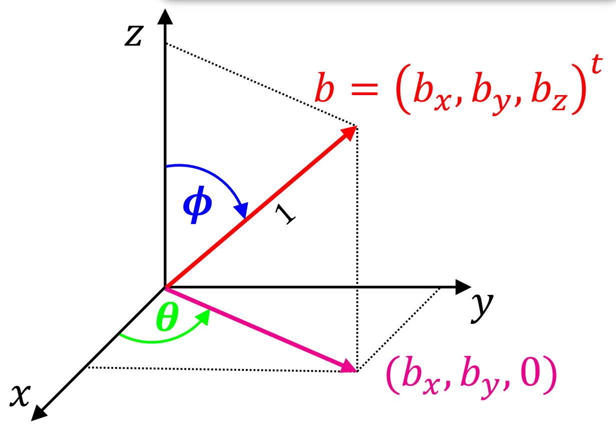
Remark: right-handed system, here only rotated.

3.5 Affine transformations in 3d

Rotation around arbitrary axis G

- Every rotation around an arbitrary axis can be combined from three rotations around individual coordinates axis.
  - *Intrinsic representation:* angles with respect to rotated axis.
  - *Extrinsic representation:* angles with respect to a fixed world system.

Rem.: Always right-handed system, here only rotated.



$$b_x = \sin\phi \cdot \cos\theta$$
$$b_y = \sin\phi \cdot \sin\theta$$
$$b_z = \cos\phi$$

- Choice of axis:
  - *Proper Euler angles:* x-y-x, x-z-x, y-z-y, y-x-y, z-x-z, **z-y-z**
  - *Tait-Bryan angles* (Cardan angles): x-y-z, x-z-y, y-x-z, y-z-x, z-x-y, z-y-x

▶ There are 24 different representations in terms of three angles, which are all equivalent.

▶ We will use in the sequel the extrinsic z-y-z representation.

3.5 Affine transformations in 3d

Wanted: Coordinates of a point P after a rotation around the axis G around the angle ψ.

- Extrinsic z-y-z approach:
  (1) Transform point P such that the axis of rotation coincides with the z-axis.
  (2) Then, use the rotation matrix $R_z(\psi)$ for the rotation around ψ.
  (3) Finally, invert the auxiliary transformation from step (1).
  - If G is already the z-axis, the auxiliary transformations are obsolete.

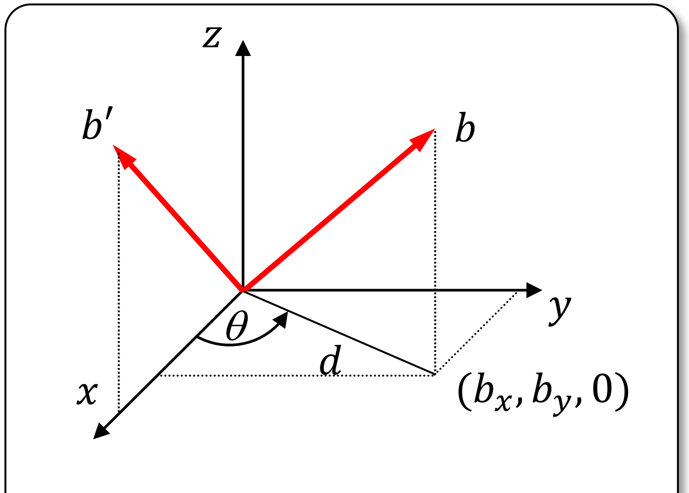
3.5 Affine transformations in 3d

Step 1:

- Rotate vector b into the (z, x)-plane: $b \to b'$.
- P is mapped to P' with $P' = R_z(-\theta) \cdot P$, with

$$R_z(-\theta) = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \frac{1}{d} \begin{pmatrix} b_x & b_y & 0 & 0 \\ -b_y & b_x & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 0 & d \end{pmatrix}.$$



$$d^2 = b_x^2 + b_y^2$$

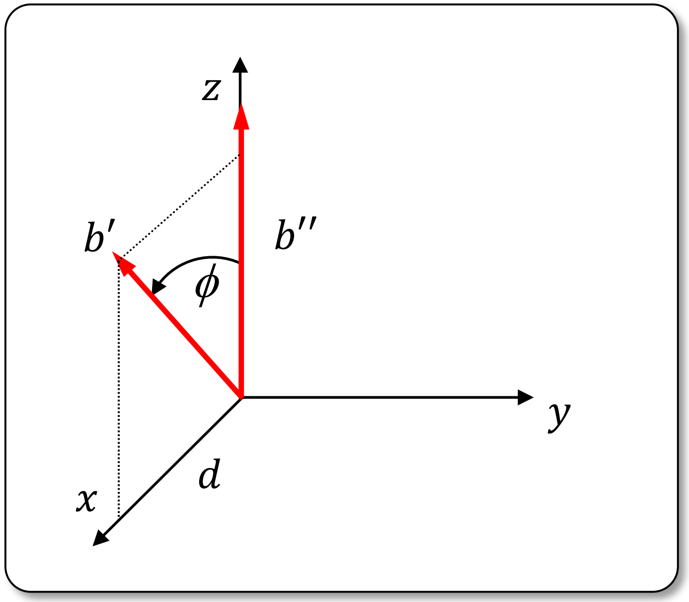
3.5 Affine transformations in 3d

Step 2:

- Rotate vector b' onto the z-axis: $b' \to b''$.
- P' is mapped to P'' with $P'' = R_y(-\phi) \cdot P'$, with

$$R_y(-\phi) = \begin{pmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} b_z & 0 & -d & 0 \\ 0 & 1 & 0 & 0 \\ d & 0 & b_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

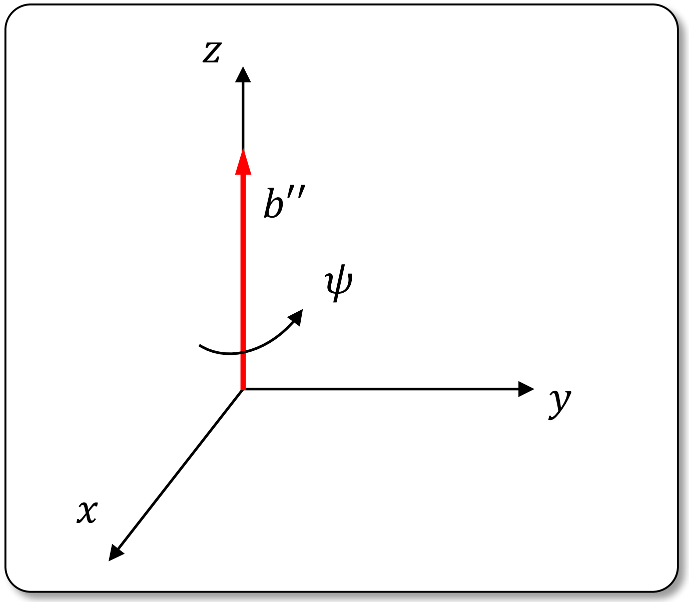


3.5 Affine transformations in 3d

Step 3:

- Rotate around the angle ψ around the z-axis.
- P'' mapped to P''' with $P''' = R_z(\psi) \cdot P''$, with

$$R_z(\alpha) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$



3.5 Affine transformations in 3d

Steps 4 and 5:

- Invert the rotations from steps 1 and 2 in reverse order.
- P''' is mapped the wanted rotated point Q of the original point P by $Q = R_z(\theta)R_y(\phi) \cdot P'''$, with

$$R_y(\phi) = \begin{pmatrix} b_z & 0 & d & 0 \\ 0 & 1 & 0 & 0 \\ -d & 0 & b_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } R_z(\theta) = \frac{1}{d}\begin{pmatrix} b_x & -b_y & 0 & 0 \\ b_y & b_x & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 0 & d \end{pmatrix}.$$

3.5 Affine transformations in 3d

Result:

The total transformation can be realized by the concatenation (multiplication) of the respective individual transformations

$$M_b(\psi) = R_z(\theta)R_y(\phi)R_z(\psi)R_y(-\phi)R_z(-\theta).$$

General case:

If the axis of rotation is an arbitrary line

$$G = \{a + \lambda \cdot b : \lambda \in \mathbb{R}, a = \left(a_x, a_y, a_z\right)^t, \|b\| = 1\},$$

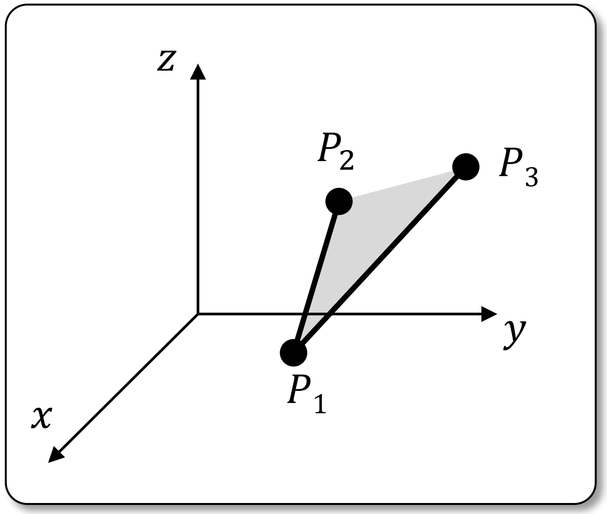
before the first step 1 and after the last step 5 a translation has to be applied, i.e.

$$M_b(\psi) = T(a)\,R_z(\theta)\,R_y(\phi)\,R_z(\psi)\,R_y(-\phi)\,R_z(-\theta)\,T(-a).$$

3.5 Affine transformations in 3d

3.5.4 Transformation of coordinate systems (orientation)

- For the definition of a coordinate system in 3d three (non-collinear) points P_1, P_2, P_3 suffice.
- How do we determine the transformation matrix, that maps
  - P_1 to the origin,
  - P_1P_2 onto the z-axis and
  - P_2P_3 into the (y, z)-plane with positive y-coordinate?



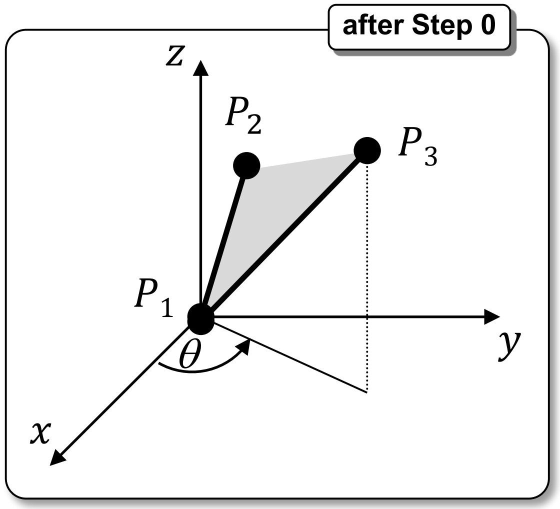
3.5 Affine transformations in 3d

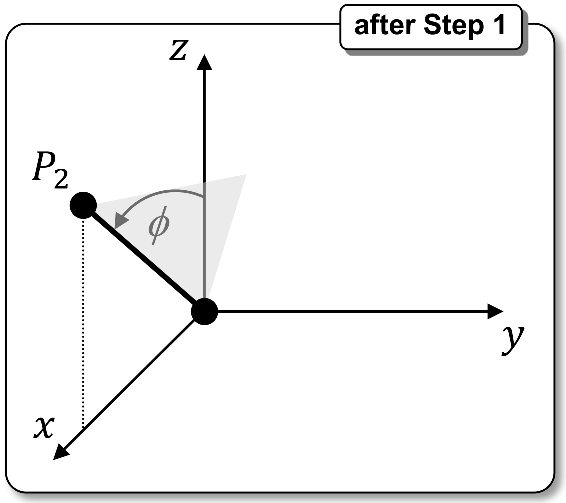
Step 0: Translation with $T(-P_1)$.

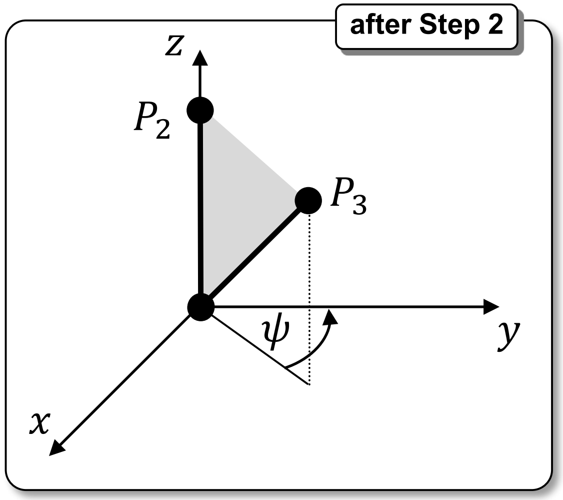
Step 1: Rotation with $R_z(-\theta)$.

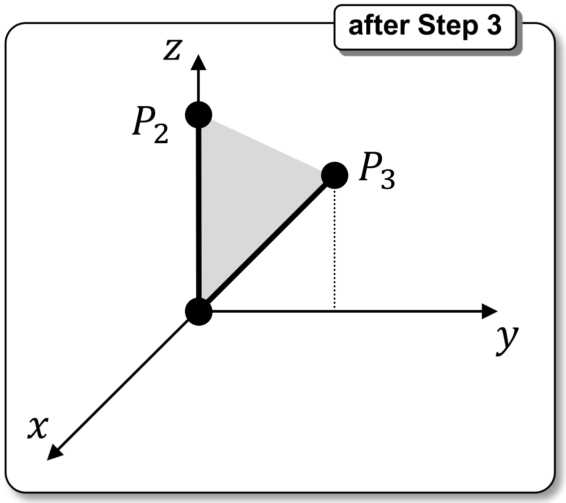
Step 2: Rotation with $R_y(-\phi)$.

Step 3: Rotation with $R_z(\psi)$.









3.5 Affine transformations in 3d

The combined transformation matrix is of the form

$$A_{l\to g} = R_z(\psi)\cdot R_y(-\phi)\cdot R_z(-\theta)\cdot T(-P_1) = \begin{pmatrix} R & -P_1 \\ 0 & 1 \end{pmatrix}.$$

with an orthonormal 3×3 matrix R.

- The rows R_1, R_2, R_3 of R span an orthonormal basis

  $$R\cdot R_1^t = (1,0,0)^t,$$
  $$R\cdot R_2^t = (0,1,0)^t,$$
  $$R\cdot R_3^t = (0,0,1)^t,$$

  i.e. they are mapped by R onto the axes of the coordinate systems (x, y, z).

3.5 Affine transformations in 3d

From these relations the row of R can be determined:

Row 3: P_1P_2 is mapped by R onto the z-axis, i.e.

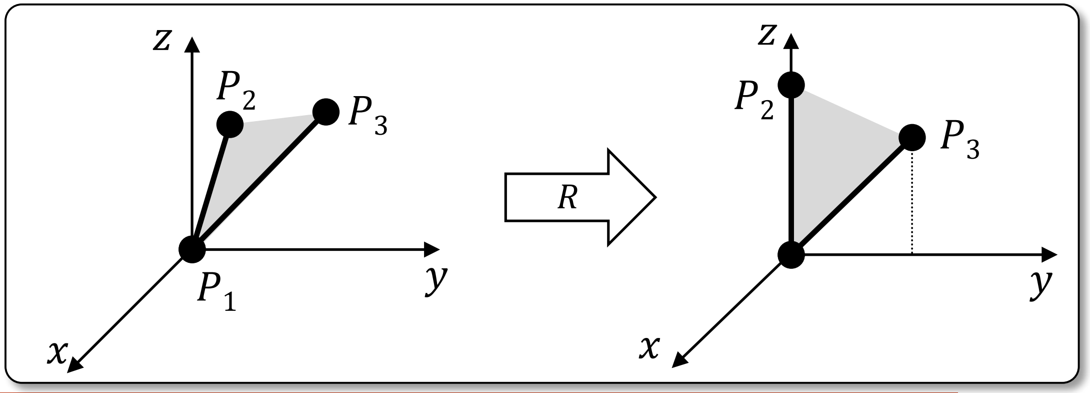
$$R_3^t = (P_2 - P_1)/||P_2 - P_1||.$$

Row 1: The (normalized) normal vector of the plane $P_1P_2P_3$ is mapped by R onto the positive x-axis, i.e.

$$R_1^t = (P_3 - P_1) \times (P_2 - P_1)/||(P_3 - P_1) \times (P_2 - P_1)||.$$

Row 2: The second row is orthonormal to R_1, R_3, i.e.

$$R_2^t = R_3^t \times R_1^t.$$



3.5 Affine transformations in 3d

Remark

1. R maps a local to a global system.
   - Also $A_{l\to g}$ maps a local to a global system.
2. For the inverse transformation of a global to a local system the inverse matrix of

$$A_{l\to g} = \begin{pmatrix} R & -P_1 \\ 0 & 1 \end{pmatrix}$$

needs to be determined.

It can be computed as

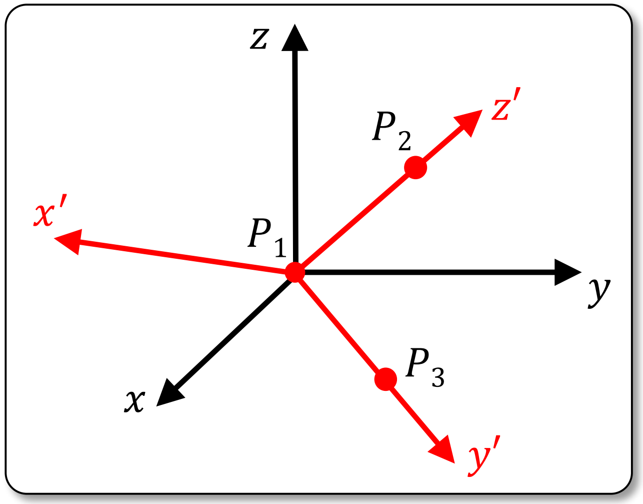
$$A_{g\to l} = A_{l\to g}^{-1} = \begin{pmatrix} R^t & R^t P_1 \\ 0 & 1 \end{pmatrix}.$$

3.5 Affine transformations in 3d

Special case: If the coordinate system spanned by P_1, P_2, P_3 is already Cartesian, R is determined by

$$R = \begin{pmatrix} x'^t \\ y'^t \\ z'^t \end{pmatrix}$$

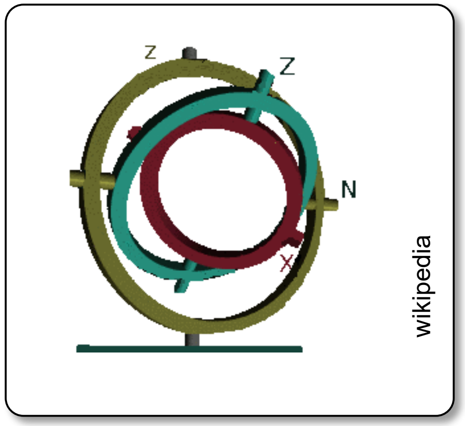
in coordinates with respect to the coordinate system (x, y, z).

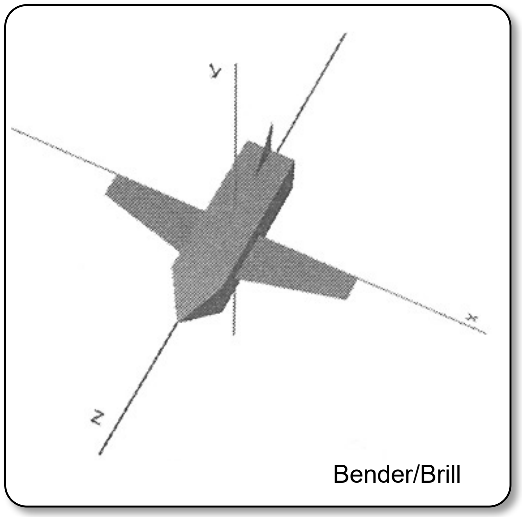


3.5 Affine transformations in 3d

Attention: If P_1P_2 and the z-axis are parallel, the angles θ and ψ are not well defined.

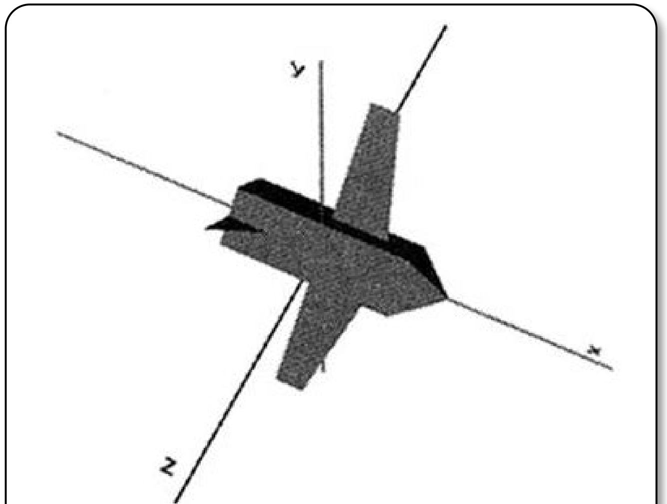
- In this case only $\theta + \psi$ is well defined.
- This effect is called **gimbal-lock**.
- It is more apparent in a different order of rotations (pilot-view):
  - x-rotation around pitch angle φ_x,
  - y-rotation around yaw angle φ_y,
  - z-rotation around roll angle φ_z.
  - The orientation of a coordinate system is represented by the triple $(\varphi_x \, \varphi_y \, \varphi_z)$.



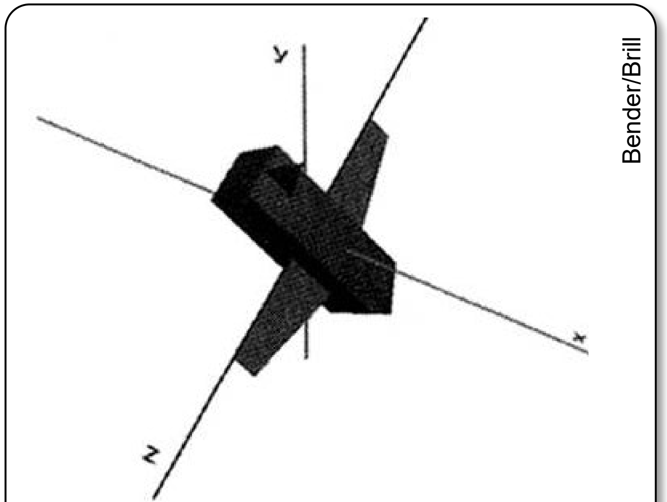


3.5 Affine transformations in 3d

- A change from the orientation from $(0°, 0°, 0°)$ to $(0°, 90°, 45°)$ does not yield the desired result:
  - After the rotation around the y-axis the third rotations axis coincides with the x-axis.



Expected position.



Reached position.

- One degree of freedom is lost!

3.5 Affine transformations in 3d

3.5.5 Quaternions

- Avoid the Gimbal-Locks for Euler-angles.
- Interpolation of rotations for animations.

Idea: Representation of rotations without coordinate systems!

- Rotation axis G passes through the origin and is spanned by a vector $b = \left(b_x, b_y, b_z\right)^t$ with $||b|| = 1$:

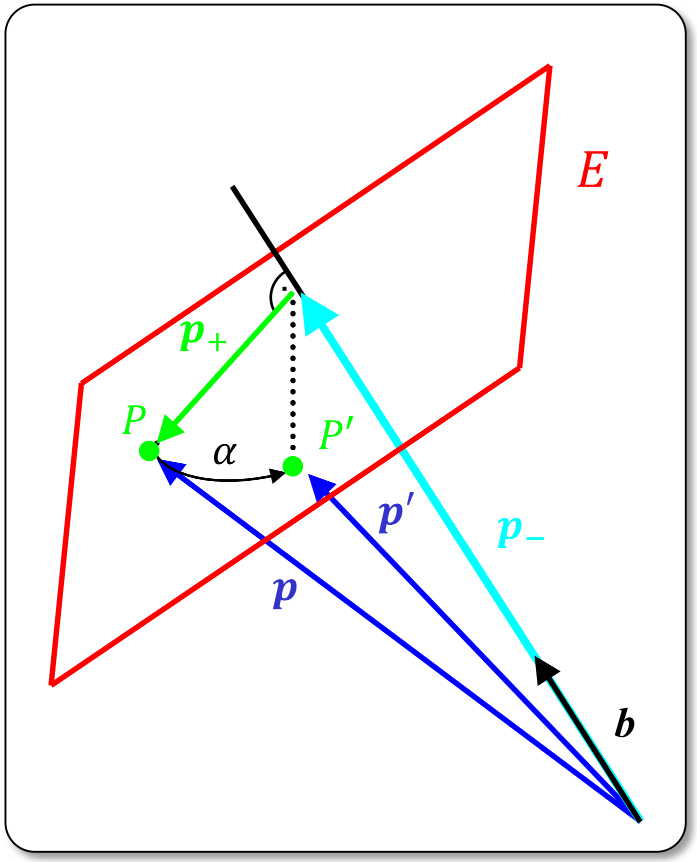
$$G = \{\lambda \cdot b\colon \lambda \in \mathbb{R}\}.$$

- Rotate the point P around the oriented axis G in space around the angle α.

3.5 Affine transformations in 3d

- Consider plane E through P perpendicular to $\boldsymbol{b}$.
- P is rotated in the plane E to P'.
  - The vector $\boldsymbol{p}$ is rotated to $\boldsymbol{p}'$.
  - This rotation has a component
    - $\boldsymbol{p}_- = \boldsymbol{b}\,(\boldsymbol{p}\,\boldsymbol{b})$ parallel to and
    - $\boldsymbol{p}_+ = \boldsymbol{p} - \boldsymbol{b}\,(\boldsymbol{p}\,\boldsymbol{b})$ perpendicular to

  the plane normal $\boldsymbol{b}$.



3.5 Affine transformations in 3d

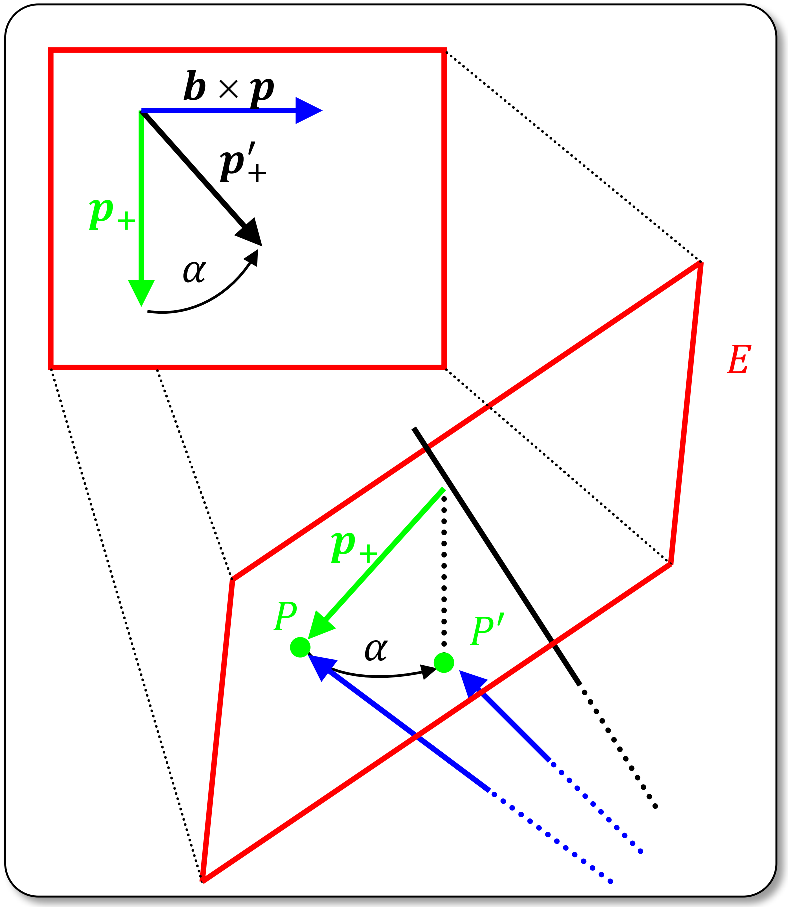
- In plane E, $\boldsymbol{p}_+$ is rotated to

$$\boldsymbol{p}'_+ = \cos\alpha \cdot \boldsymbol{p}_+ + \sin\alpha \cdot \boldsymbol{b} \times \boldsymbol{p}.$$

- ▶ The total rotation is represented as

$$\begin{aligned}\boldsymbol{p}' = {} & \cos\alpha \cdot \boldsymbol{p} \\ & + (1-\cos\alpha)\cdot \boldsymbol{b}\cdot(\boldsymbol{b}\cdot\boldsymbol{p}) \\ & + \sin\alpha \cdot \boldsymbol{b}\times\boldsymbol{p}.\end{aligned}$$

- Is there an elegant formulation for this last expression?



3.5 Affine transformations in 3d

... yes, with quaternions…

Sir William Hamilton, 1805-1865.

Definition

A quaternion q is defined by four real numbers (s, a, b, c) and

$$q = s + a \cdot \boldsymbol{i} + b \cdot \boldsymbol{j} + c \cdot \boldsymbol{k} \qquad \text{with} \qquad \boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1.$$

The imaginary units $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ satisfy additionally

$$\boldsymbol{i} \cdot \boldsymbol{j} = -\boldsymbol{j} \cdot \boldsymbol{i} = \boldsymbol{k}; \qquad \boldsymbol{j} \cdot \boldsymbol{k} = -\boldsymbol{k} \cdot \boldsymbol{j} = \boldsymbol{i}; \qquad \boldsymbol{k} \cdot \boldsymbol{i} = -\boldsymbol{i} \cdot \boldsymbol{k} = \boldsymbol{j}.$$

3.5 Affine transformations in 3d

Addition and multiplication are defied as

$$
\begin{aligned}
(s_1, a_1, b_1, c_1) + (s_2, a_2, b_2, c_2) &= (s_1 + s_2, a_1 + a_2, b_1 + b_2, c_1 + c_2) \\
&= s_1 + s_2 + (a_1 + a_2)\boldsymbol{i} \\
&\quad + (b_1 + b_2)\boldsymbol{j} + (c_1 + c_2)\boldsymbol{k}, \\
(s_1, a_1, b_1, c_1) \cdot (s_2, a_2, b_2, c_2) &= s_1 s_2 - a_1 a_2 - b_1 b_2 - c_1 c_2 \\
&\quad + (s_1 a_2 + s_2 a_1 + b_1 c_2 - b_2 c_1)\, \boldsymbol{i} \\
&\quad + (s_1 b_2 + s_2 b_1 + c_1 a_2 - c_2 a_1)\, \boldsymbol{j} \\
&\quad + (s_1 c_2 + s_2 c_1 + a_1 b_2 - a_2 b_1)\, \boldsymbol{k}
\end{aligned}
$$

A quaternion q can be represented as $q = (s, \boldsymbol{x})$ with $s \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^3$. This implies

$$(s_1, \boldsymbol{x}_1) \cdot (s_2, \boldsymbol{x}_2) = (s_1 s_2 - \boldsymbol{x}_1 \cdot \boldsymbol{x}_2, s_1 \boldsymbol{x}_2 + s_2 \boldsymbol{x}_1 + \boldsymbol{x}_1 \times \boldsymbol{x}_2).$$

3.5 Affine transformations in 3d

- The multiplication for quaternions induces a norm for quaternions:

  $$(q \cdot q')^{1/2} = ||q|| = (s^2 + ||\boldsymbol{x}||^2)^{1/2} \text{ with } q' = (s, -\boldsymbol{x}).$$

  Here q' is the **conjugate quaternion**.

► For a **normalized quaternion** q, i.e. $||q|| = 1$, the conjugate quaternion is its inverse q^{-1}, i.e. $q \cdot q' = (1,0,0,0)$.

Theorem

Every normalized quaternion q can be represented as

$$q = (\cos(\theta/2), \sin(\theta/2)\, \boldsymbol{n})$$

for a $\theta \in [0, \pi]$ and $\boldsymbol{n} \in \mathbb{R}^3$ with $||\boldsymbol{n}|| = 1$.

3.5 Affine transformations in 3d

- Identifying the vector $\boldsymbol{p}$ on page §3-54 with the quaternion $p = (0, \boldsymbol{p})$, the multiplication with a normalized quaternion q from left and the conjugate quaternion q' from right yields the quaternion

$$q \cdot p \cdot q' = \left(\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\boldsymbol{n}\right) \cdot (0, \boldsymbol{p}) \cdot \left(\cos\left(\frac{\theta}{2}\right), -\sin\left(\frac{\theta}{2}\right)\boldsymbol{n}\right)$$
$$= (0, \cos\theta \cdot \boldsymbol{p} + (1 - \cos\theta) \cdot \boldsymbol{n} \cdot (\boldsymbol{n} \cdot \boldsymbol{p}) + \sin\theta \cdot \boldsymbol{n} \times \boldsymbol{p}).$$

- ► This is exactly the same formula as on page §3-55 for $\boldsymbol{p}'$ with $\boldsymbol{b} = \boldsymbol{n}$ and $\alpha = \theta$.
- ► Multiplication of a quaternion with a normalized quaternion q from left and with the conjugate quaternion q' from right is a rotation around the axis $\boldsymbol{n}$ with angle θ.

3.5 Affine transformations in 3d

Rotation with quaternions of a $\boldsymbol{p}$ around the axis $\boldsymbol{n}$ with angle θ:

1. Construct the quaternion $q = \left(\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right) \cdot \boldsymbol{n}\right)$.
2. Compute $q \cdot (0, \boldsymbol{p}) \cdot q'$.
3. Construct from the result a vector in $\mathbb{R}^3$.

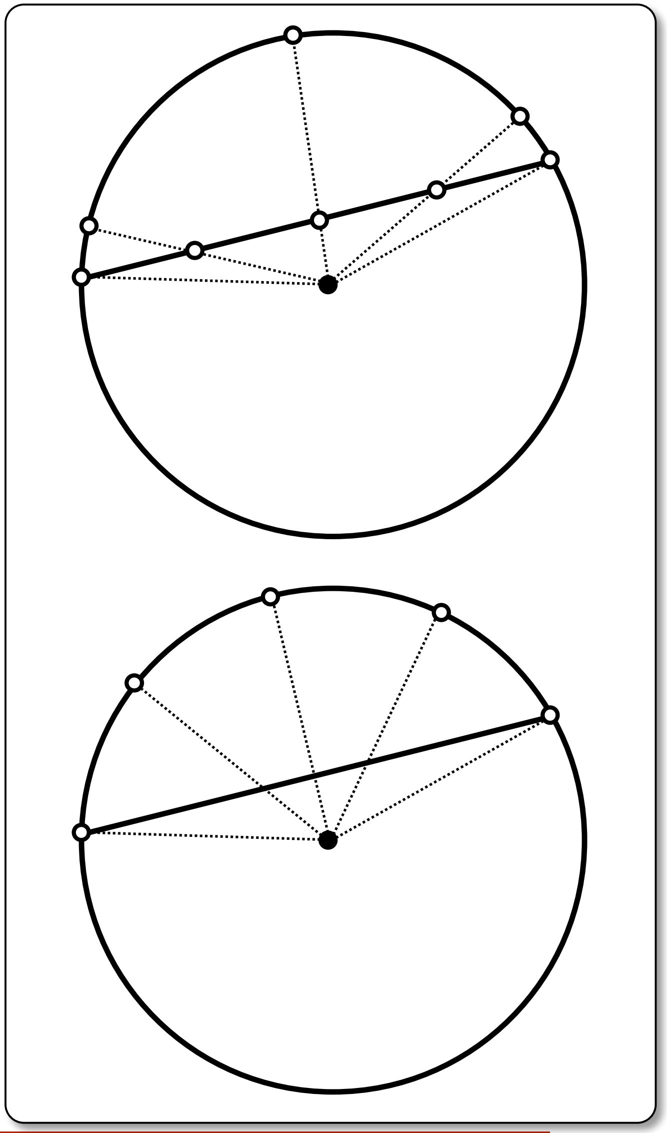
- Because the product of two normalized quaternions yields again a normalized quaternion, multiple rotations around different axes can be computed subsequently.

▶ **Attention:** The quaternion multiplication is not commutative!

3.5 Affine transformations in 3d

Properties:

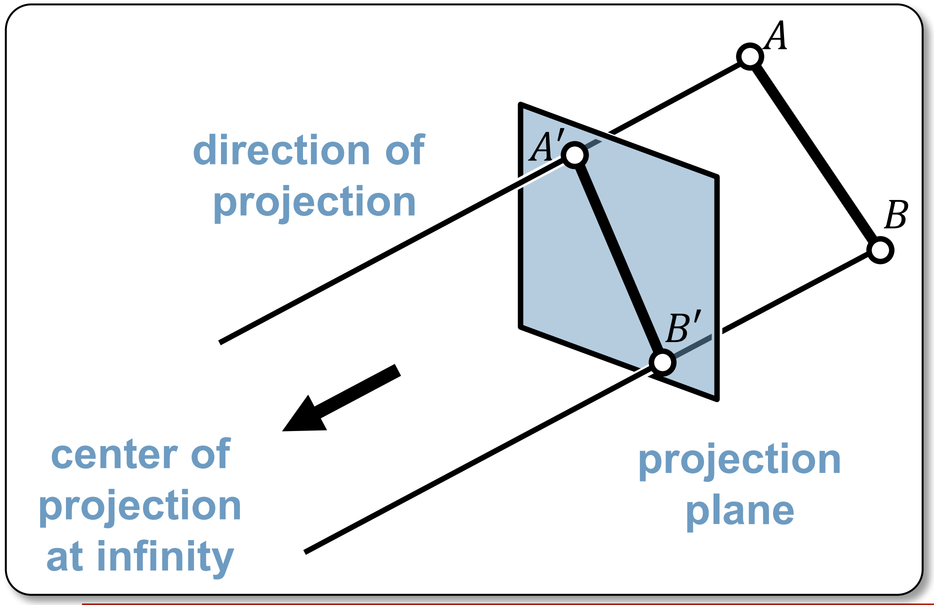
- The quaternion multiplication is not commutative!
- There is no gimbal-lock, because the order of coordinate axes is irrelevant.
- Interpolation:
  - Linear interpolation yields varying angular velocities, top image.
  - **S**pherical **L**in**E**a**R** inter**P**olation (SLERP) [Shoemeker85], bottom image.

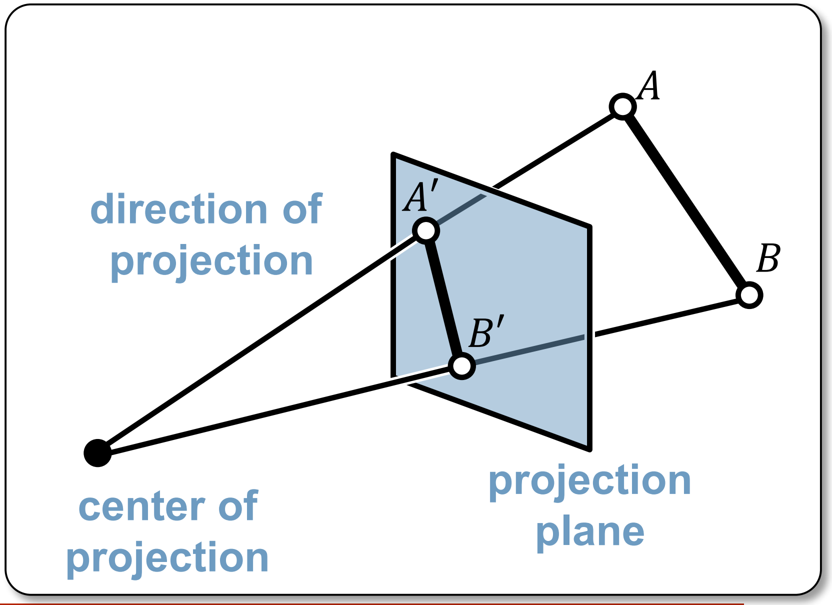


3.6 Projections

Definition: A **projection** is a mapping, that maps a space of dimension n to a space of dimension $< n$.

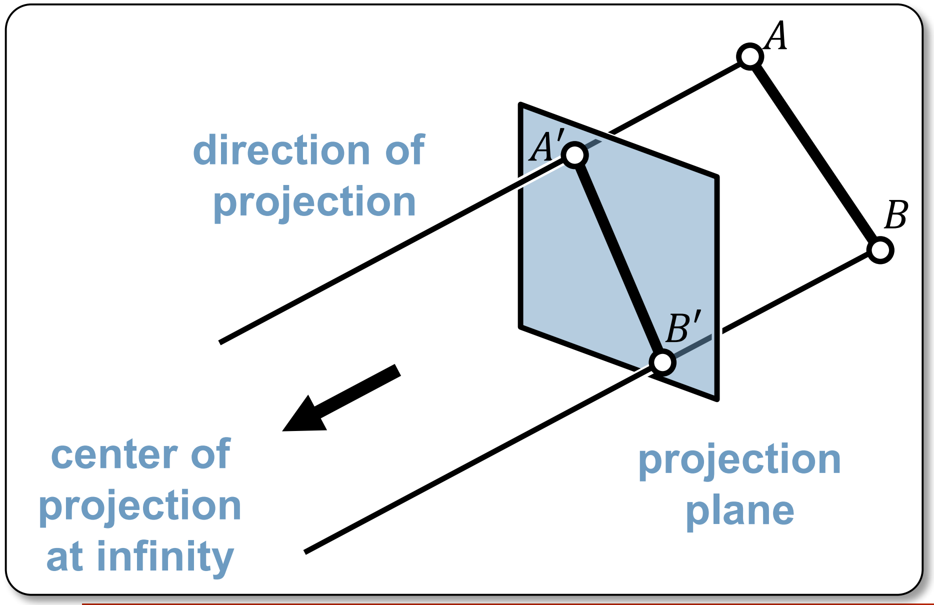
- Because displays are 2d-ouput devices, 3d-objecte must be plotted in 2d-views.
  - A space point A is mapped along a **projection ray** (projector, direction of projection) to a given **projection plane**.

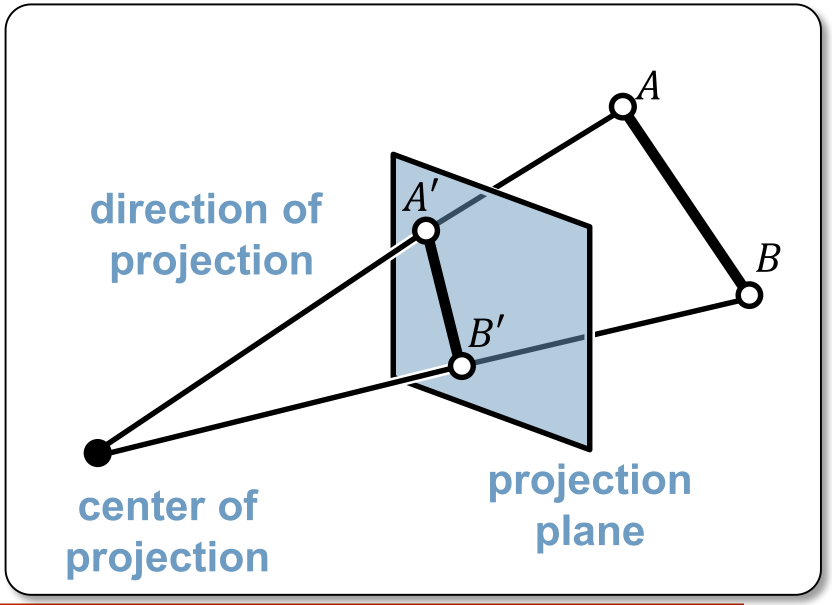




3.6 Projections

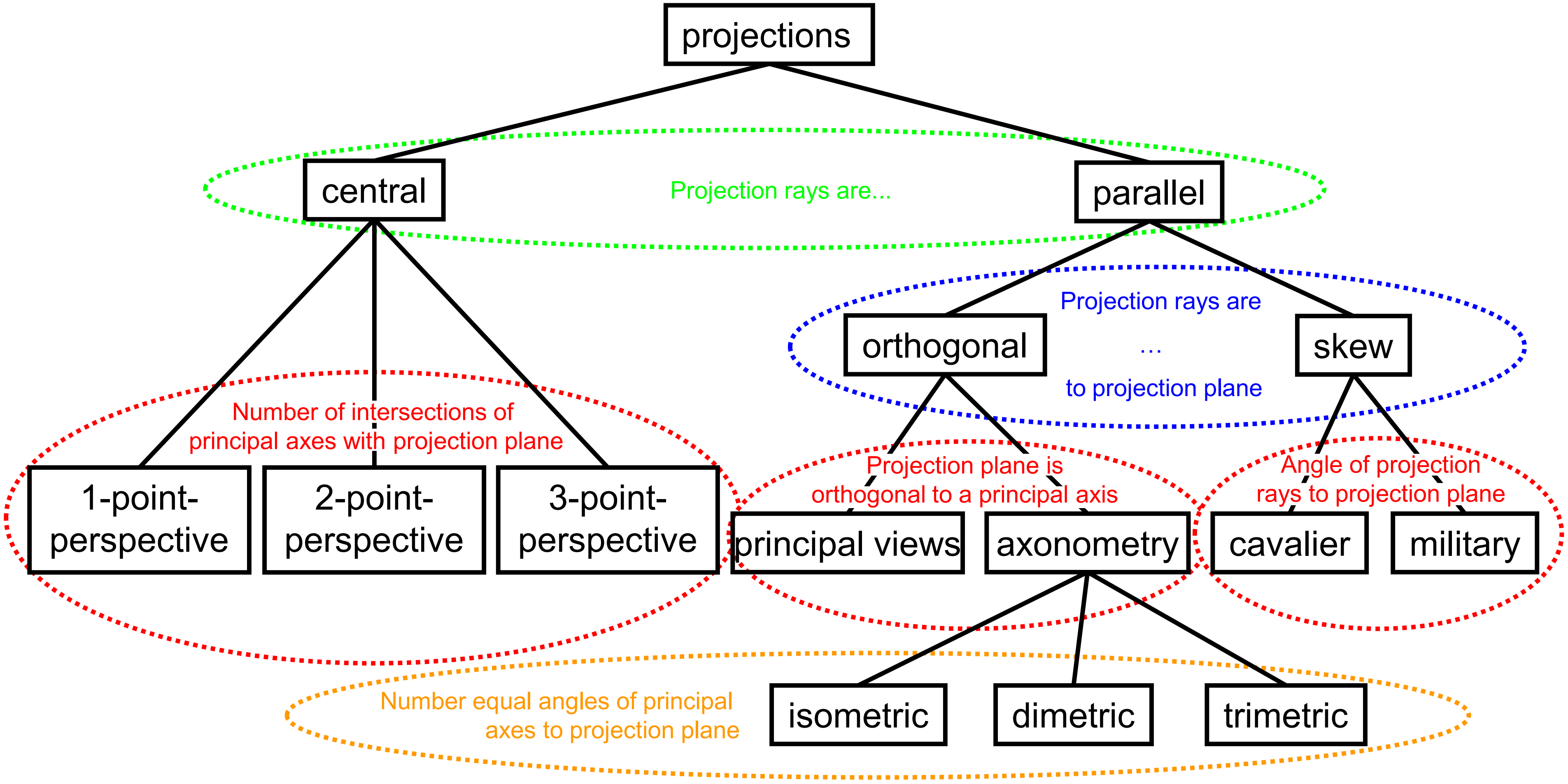
- The direction of a projection is determined by the **center of projection** and a space point A yielding projection rays.
- The intersection of the projection rays with the projection plane determines the **projected space point** A'.





3.6 Projections

Classification of projections



3.6 Projections

Classification of projections

Abbreviations: projection ray = PR, projection plane = PP, principal axes = PA

- Direction of PR:

(1) central

➨ Relative position of PA to PP

- Number of intersections of PA with PP.

(2) parallel

➨ Angle of PR to PA

a) orthogonal
b) skew

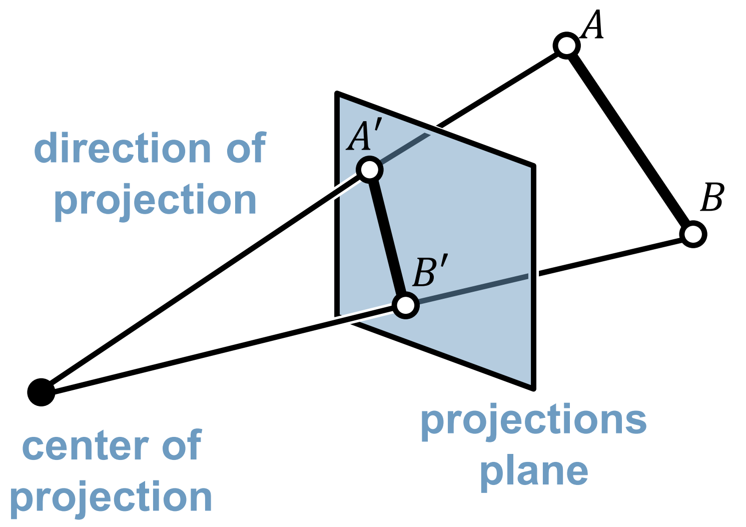
$\Big\}$ ➨ Relative position of PA/PR to PP

- Angle of PA/PR to PP.

3.6 Projections

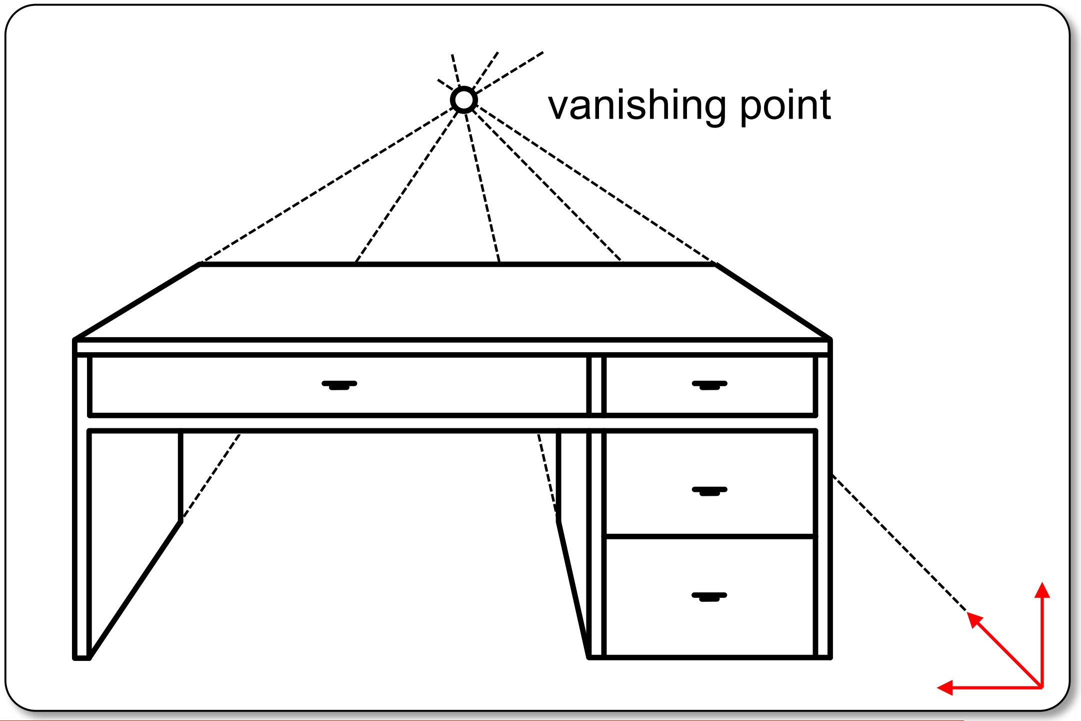
3.6.1 Perspective projection / central projection

- All **projection rays** pass through the **center of projection**, which coincides with the eye point of the viewer (camera).
- This methods generates an optical depth impression and dates back to the beginnings of ancient paintings.



3.6 Projections

- **Properties**
  - The image lines of two parallel lines in space, which are not parallel to the the projection plane, intersect in a unique point (**vanishing point**).
    - The lines also intersect the projection plane!
  - There are infinitely many vanishing points.
    - There is one vanishing point per direction, that is not parallel to the projection plane.

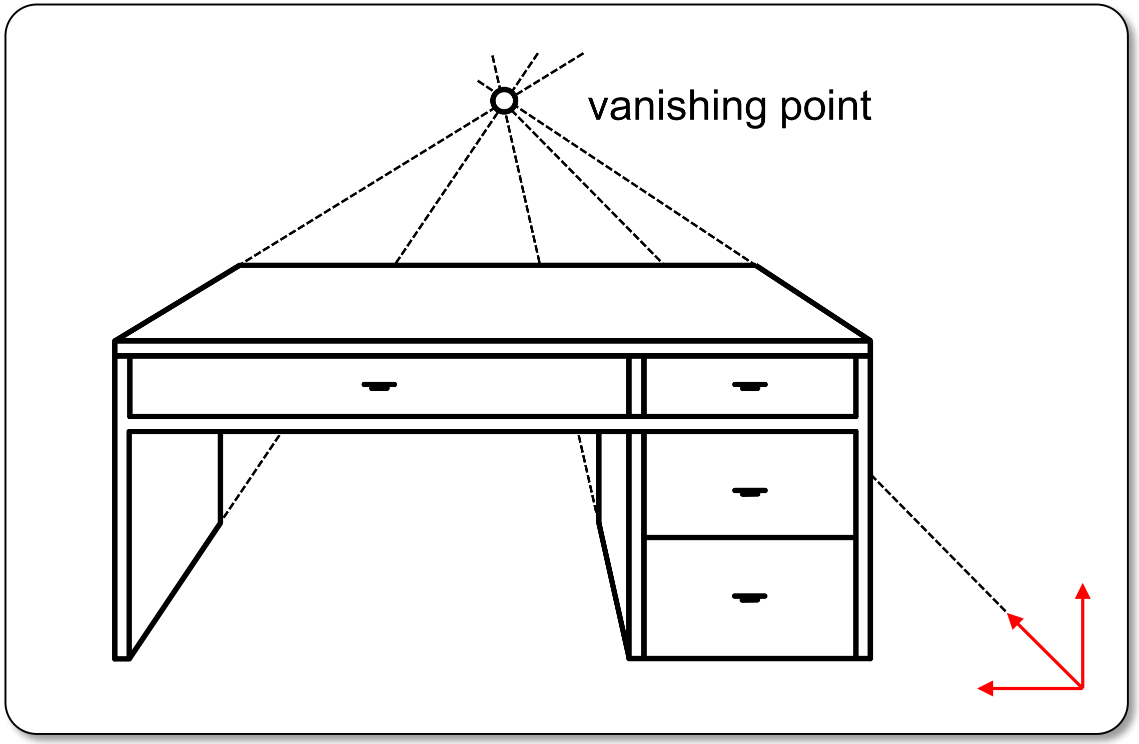


3.6 Projections

- **Properties**
  - The image lines of two parallel lines in space, which are not parallel to the the projection plane, intersect in a unique point (**vanishing point**).
    - The lines also intersect the projection plane!
  - There are infinitely many vanishing points.
    - There is one vanishing point per direction, that is not parallel to the projection plane.
  - The vanishing points of the principal axes are highlighted.
    - E.g. lines, parallel to the x-axis (of the world system), intersect in the x-vanishing point, etc.
- Perspective projections are classified by the number of principal axes, that intersect the projection plane.
  - 1-point-, 2-point- and 3-point-perspectives.

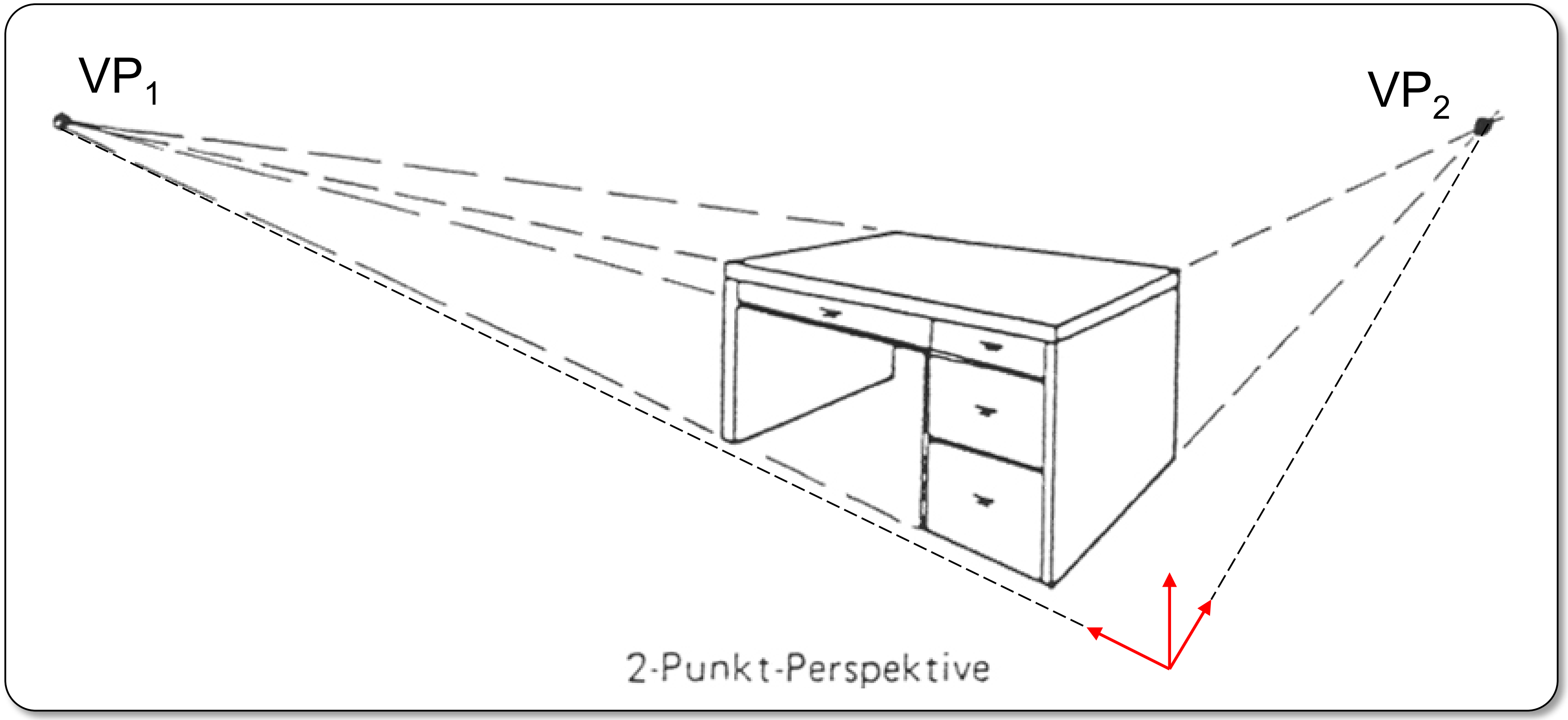
3.6 Projections

Example: 1-point-perspektive



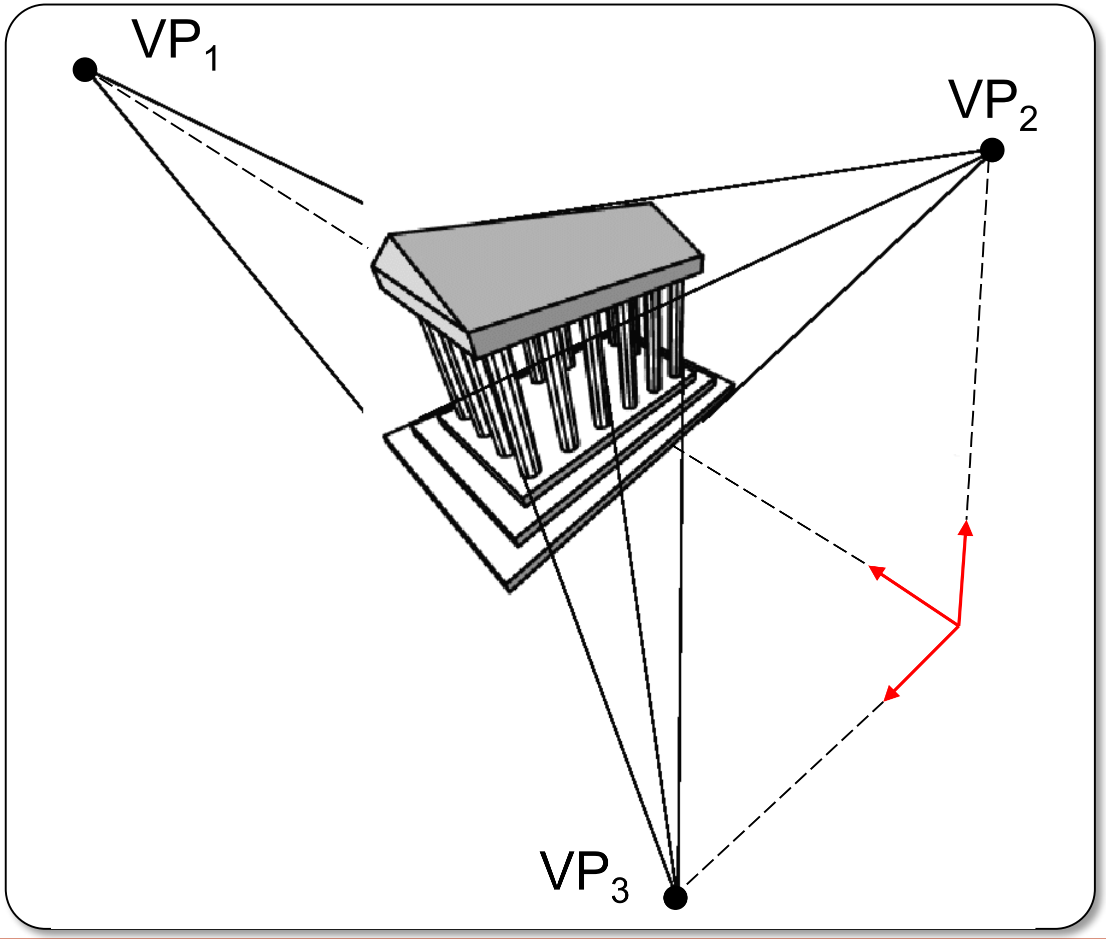
3.6 Projections

Example: 2-point-perspektive



3.6 Projections

Example: 3-point-perspektive



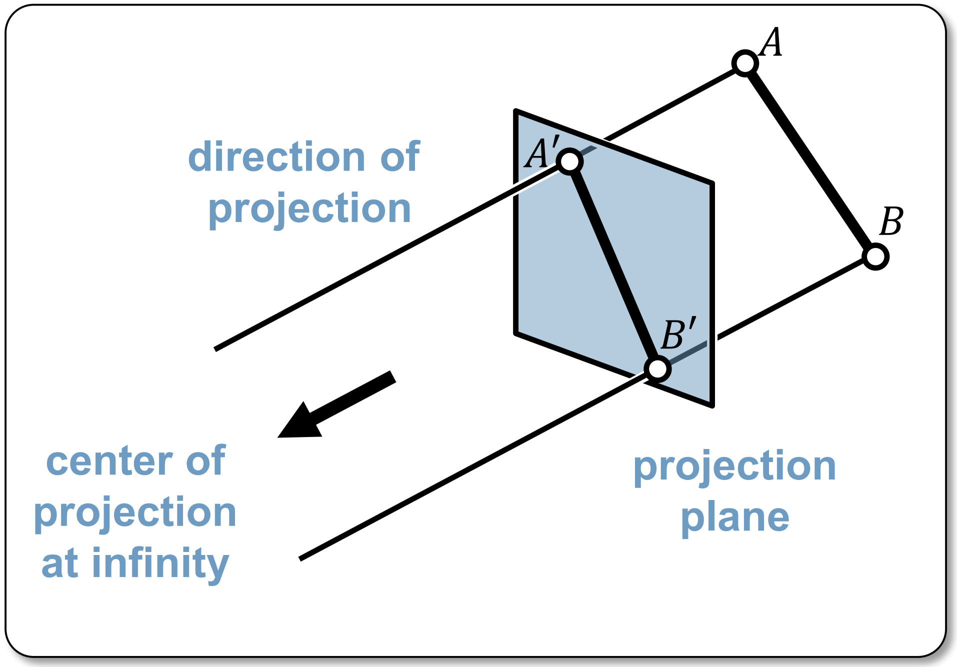
3.6 Projections

3.6.2 Parallel projections

- All **projection rays** are parallel to the same direction.
- The **center of projection** is at a point at infinity (far point).

▶ In projective geometry the parallel projection is just a special case a central projection.

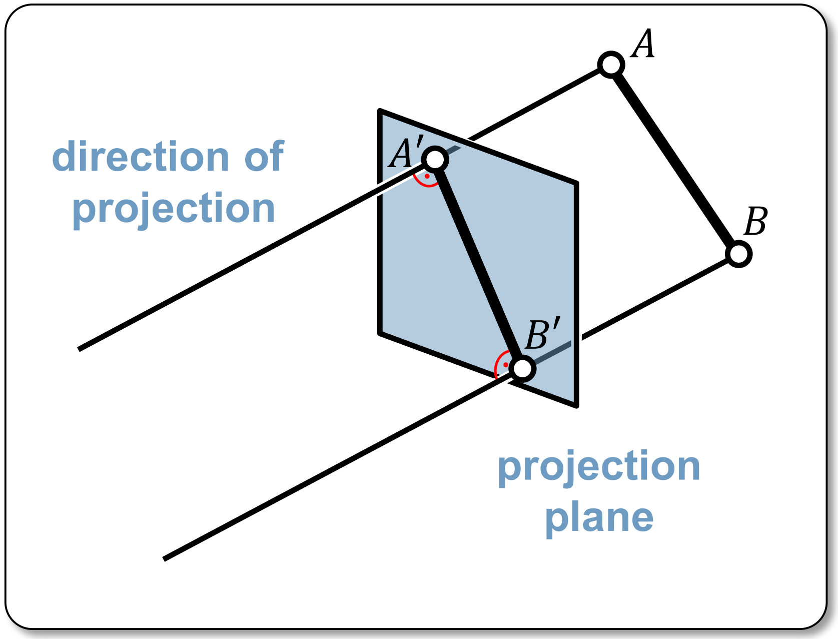
- **Disadvantage:** Less realistic.
- **Advantage:** Allows for exact measurements of distances in the image.

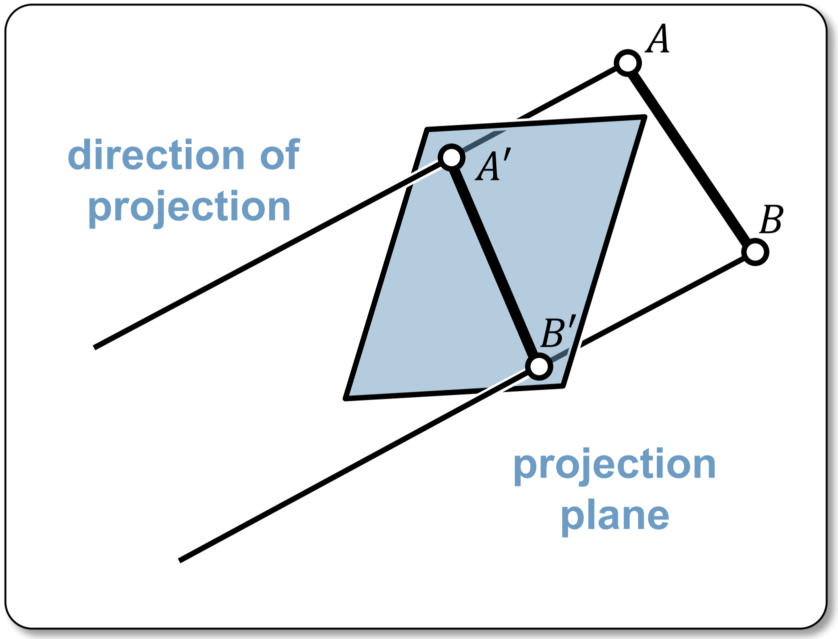


3.6 Projections

- The projection rays of a parallel projection can be
  - perpendicular (**orthographic projection**) or
  - skew (**oblique projection**)

  to the projection plane.





3.6 Projections

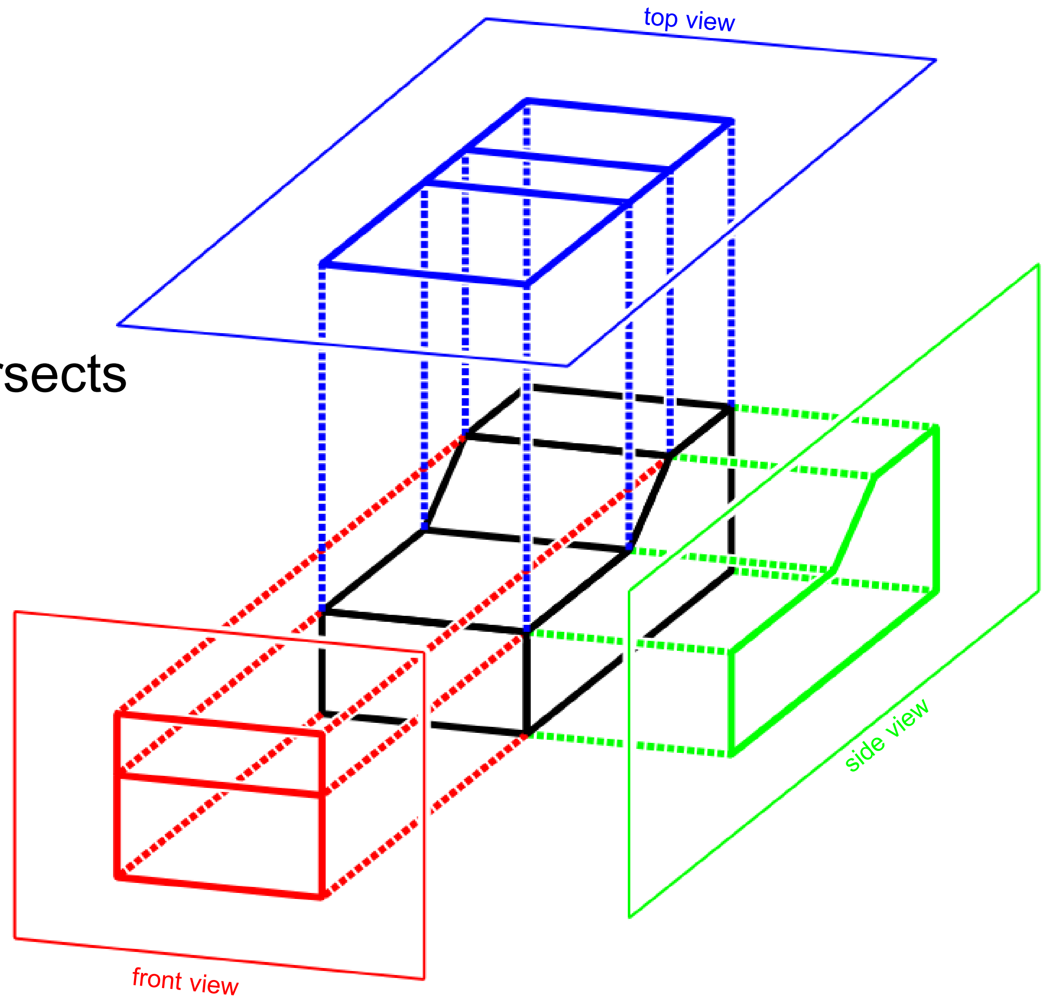
3.6.3 Orthographic projection

- The direction of the projection coincides with the normal of the projection plane.
- There are two kinds of orthographic projections: **principal views** and **axonometries**.

3.6 Projections

- For the **principal views**
  - top view,
  - front view,
  - side view

  the projection plane intersects only one **principal axis** (coordinates axis of the world system).
- The normal of the projection plane is also parallel to one of the principal axes.

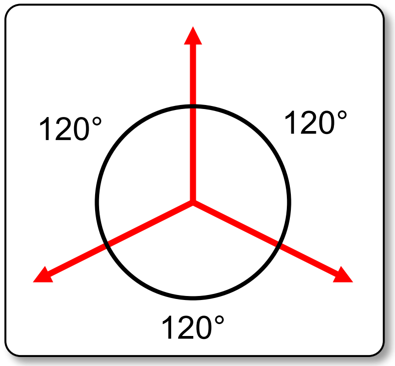


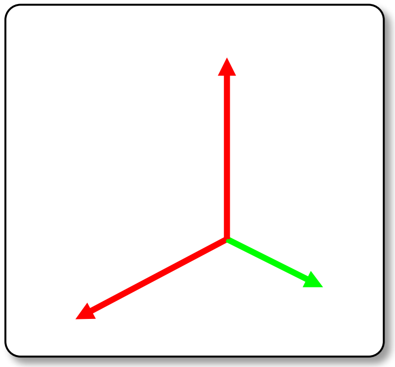
3.6 Projections

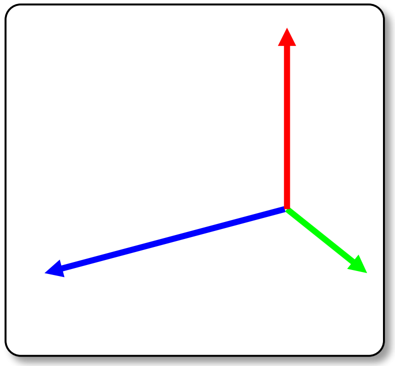
- For an **axonometry** the projection plane is not orthogonal to any of the coordinate axes (of the world system).
  - Parallel lines are mapped to parallel lines.
  - Angels are not preserved.
  - Distances can be measured along principal axes, although using different scales for each axis.

3.6 Projections

- For an **isometric axonometry** the angle between the projection plane and all three principal axes is equal.
  - Uniform shrinkage for all three coordinate axes.
- For a **dimetric axonometry** the angle between the projection plane and two of the principal axes is equal.
  - Uniform shrinkage for two of the coordinate axes.
- For a **trimetric axonometry** the angle between the projection plane and all three principal axes is different.
  - The shrinkage/scale for each coordinate axis is different.







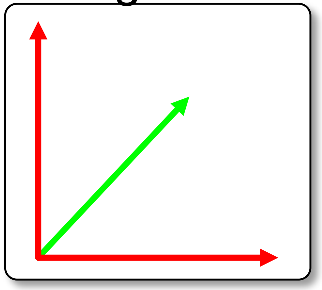
3.6 Projections

3.6.4 Oblique parallel projection

- The direction of projection is not orthogonal to the projection plane.
- There are two most common skew parallel projection: **Cavalier** and **military projection**.

Cavalier projection

- Angle between projection direction and image plane is 45°.
  - The length of the projection of lines perpendicular to the image plane remains unchanged.
- There are infinitely many Cavalier projections, one per direction in the image plane.

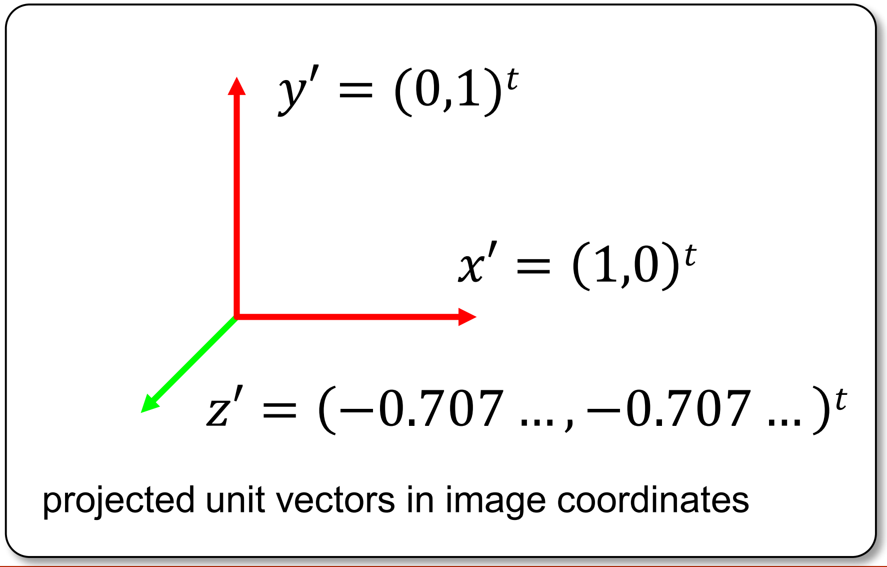


3.6 Projections

Military projection / cabinet projection

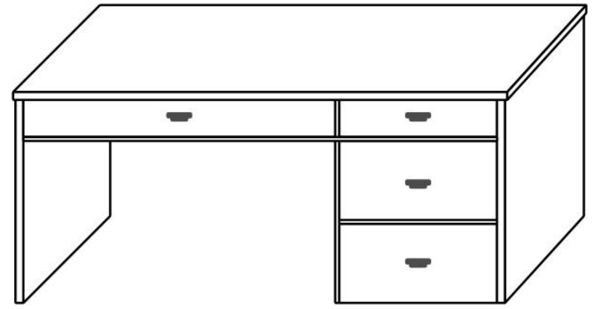
- Here the length of the projection of a line perpendicular to the projection plane should be halved.
- ▶ The angle between projection direction and image plane is $\operatorname{atan}(2) = 63.4°$.

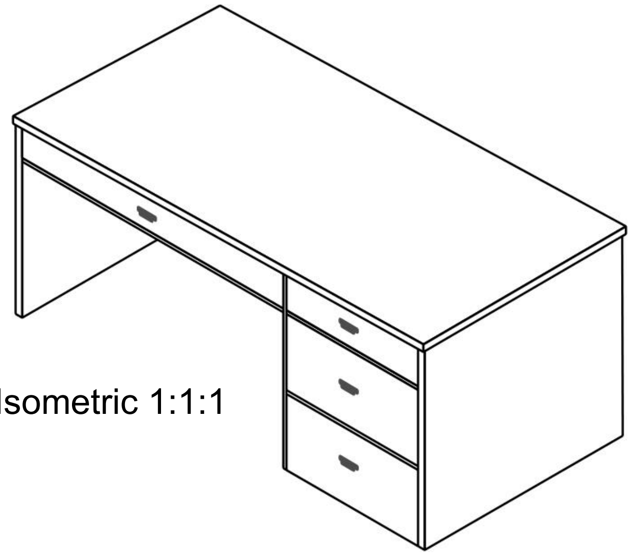
Example:

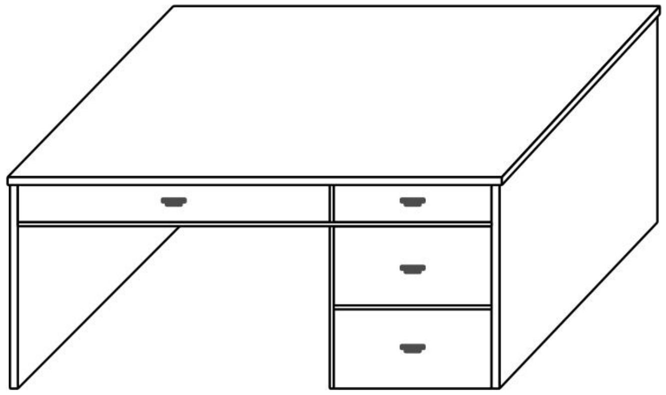

$y' = (0,1)^t$
$x' = (1,0)^t$
$z' = (-0.707\ldots, -0.707\ldots)^t$
projected unit vectors in image coordinates

3.6 Projections

Examples:


Military projection


Isometric 1:1:1

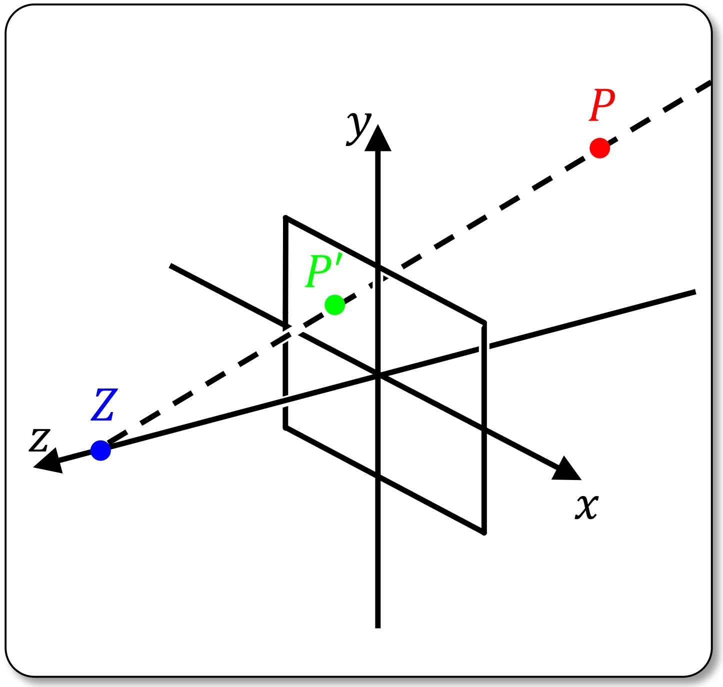

Cavalier projection

3.6 Projections

3.6.5 Realization of a central projection

- The implementation of a perspective projection depends on the setup of the applications, that can be achieved using suitable transformations.

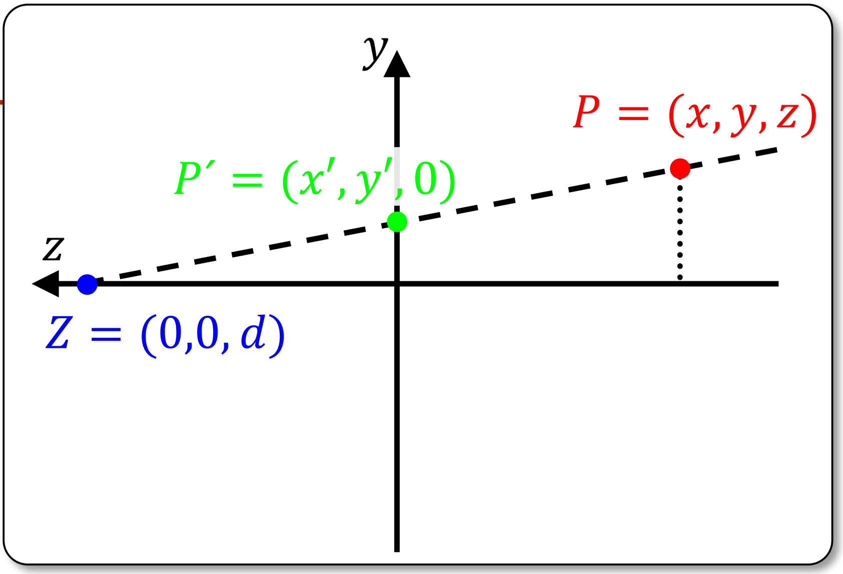
- **Simple example setup:**
  - Center of projection Z and eye point coincide and lie on the positive z-axis with distance $d > 0$ to the origin, i.e. $Z = (0,0,d)$.
  - View direction is the negative z-axis.
  - Image plane lies in the (x, y)-plane.



3.6 Projections

From the interception theorem follows

$$\frac{y'}{d} = \frac{y}{d-z} \quad \text{and} \quad \frac{x'}{d} = \frac{x}{d-z}$$



- Thus,

$$y' = y \cdot \left(\frac{d}{d-z}\right) = y \cdot \left(1 - \frac{z}{d}\right)^{-1},$$

$$x' = x \cdot \left(\frac{d}{d-z}\right) = x \cdot \left(1 - \frac{z}{d}\right)^{-1}.$$

- The central projection in this setup can be realized *in homogeneous coordinates* by the matrix

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -d^{-1} & 1 \end{pmatrix}.$$

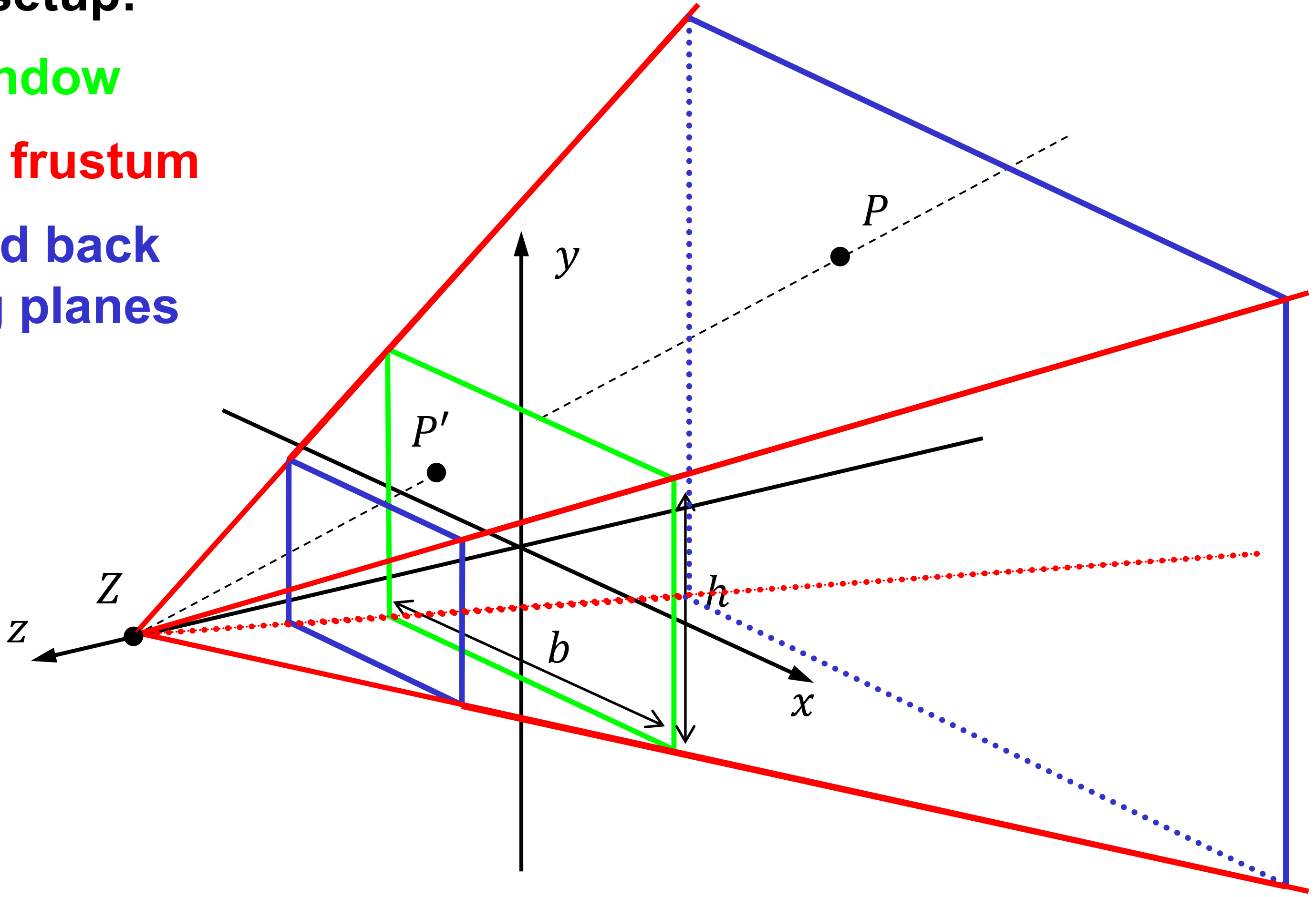
3.6 Projections

- **Extended setup:**
  - In the image plane a **view window** is defined by
    - width b, height h, ration width-height (aspect-ratio) and
    - it is symmetrically arranged to the origin.
  - The projectors through the corners of the view window define the so-called **viewing frustum**.
  - Additionally two planes parallel to the image plane (**front and back clipping plane**) bound the viewing frustum in z-direction.
  - The viewing frustum bounds the volume of the spatial scene that is visible (see §3.6 Clipping).

3.6 Projections

Extended setup:

- 🟩 **view window**
- 🟥 **viewing frustum**
- 🟦 **front and back clipping planes**



3.7 Clipping

To visualize only objects (in the image plane) that lie inside the view window, we need to cut away objects that lie outside the view window:

clipping at the window boundary.

3.7 Clipping

3.7.1 Clipping of lines

Clipping of a line at a rectangular, axis-parallel window.

There are three cases, two of which are easy:

1. Both end points of the line are inside the window

   plot the line.

2. Both end points of the line are above, below, left, or right of the window

   do not plot the line.

3. Otherwise,

   Compute the intersections of the line with the window boundaries and determine the visible part of the line.

3.7 Clipping

3.7.2 Cohen-Sutherland line-clipping algorithm

Idea: Fast method to classify a line as inside, outside or intersecting with the window.

Input: Window $(x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})$

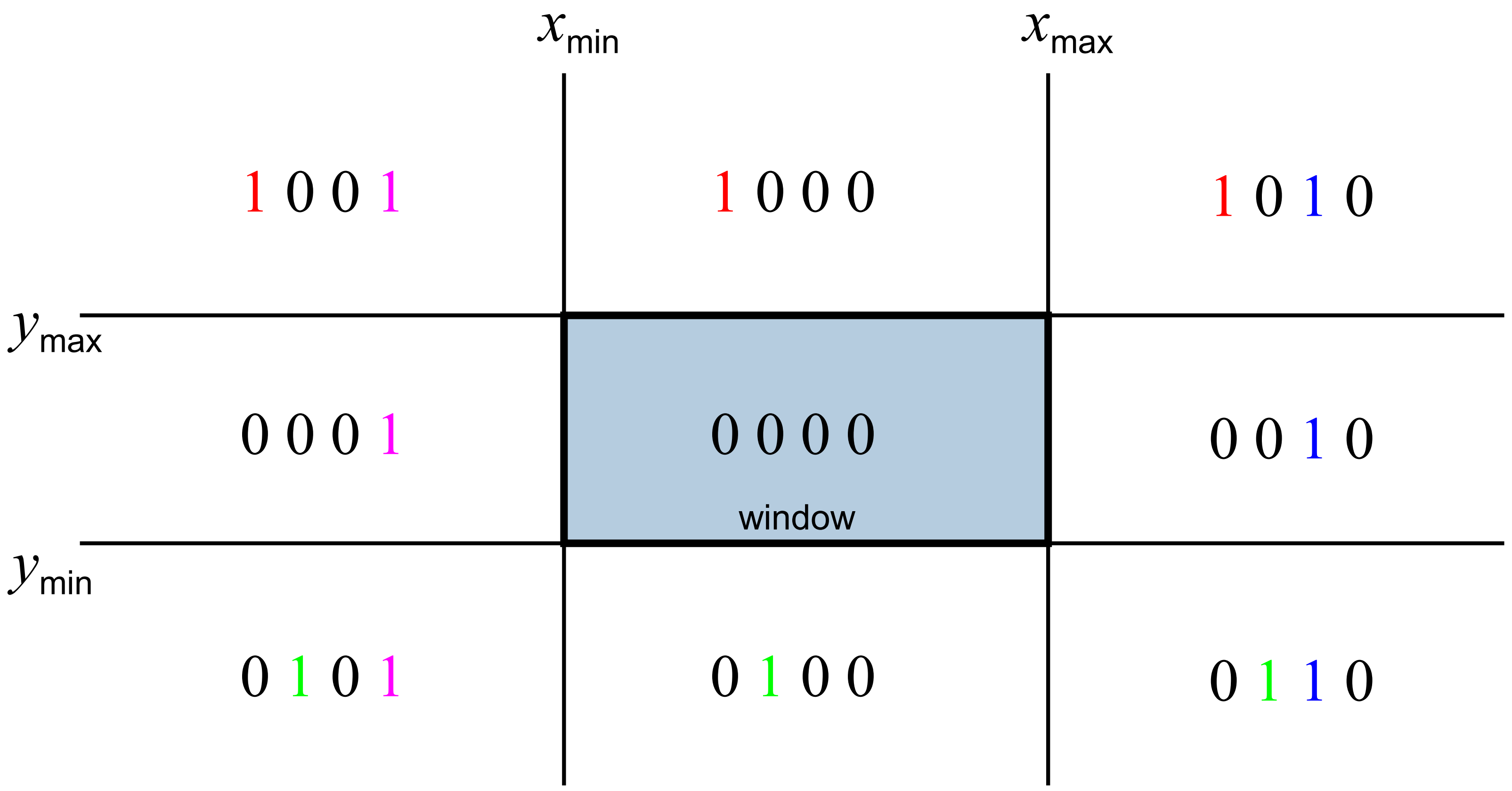
- The bounding lines partition the image plane into nine regions.

► A 4-bit-code encodes the position of the endpoint with respect to the window:

Bit	... set, for region ...	
0	... left of the window	$x < x_{\text{min}}$
1	... right of the window	$x > x_{\text{max}}$
2	... below of the window	$y < y_{\text{min}}$
3	... above of the window	$y > y_{\text{max}}$

3.7 Clipping

Codes for window and surrounding regions.



3.7 Clipping

- Determine for the end points of a line the 4-bit-codes:
  - The line is **completely inside the window**, if both end points have the 4-bit-code 0000 (i.e. bit-wise-OR is zero).
  - The line is **completely outside the window**, if the end points have at least one common bit at the same position (bit-wise-AND).
  - Otherwise:
    - Intersect all lines at the lines bounding the window.
    - Subdivide every line into two parts and determine the 4-bit-codes for the new end points.
    - The part outside the window is eliminated; the other part might be subdivided further.

3.7 Clipping

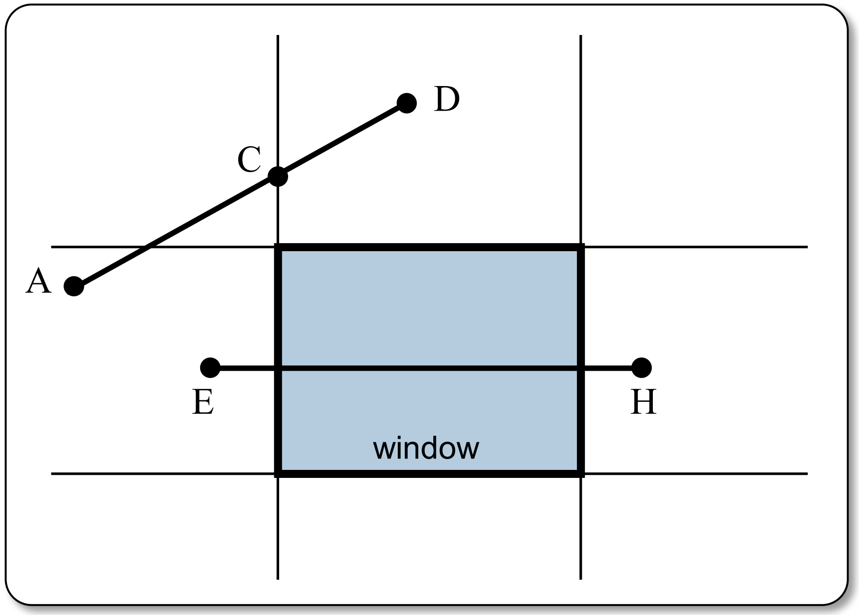
Example:

- **line AD:**

  Codes 0001 and 1000 $\rightarrow$ intersection

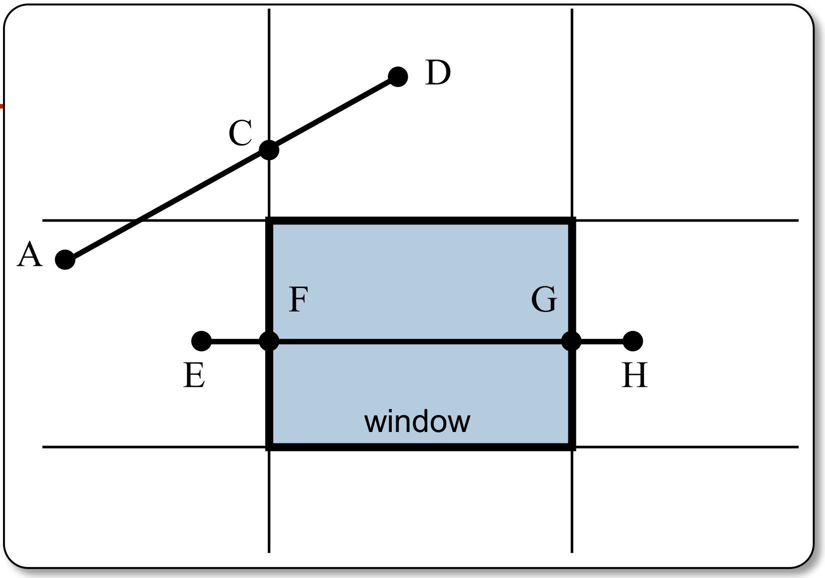
  Intersection with left window boundary yields C $\rightarrow$ eliminate AC

  C and D are above the window $\rightarrow$ eliminate CD



3.7 Clipping

Example:



- **line EH:**

  Codes 0001 and 0010 → intersection

  Intersection with left window boundary yields F → eliminate EF

- **line FH:**

  Codes 0000 and 0010 → intersection

  Intersection with right window boundary yields G → eliminate GH

- **line FG:**

  Codes 0000 and 0000 → draw FG

3.7 Clipping

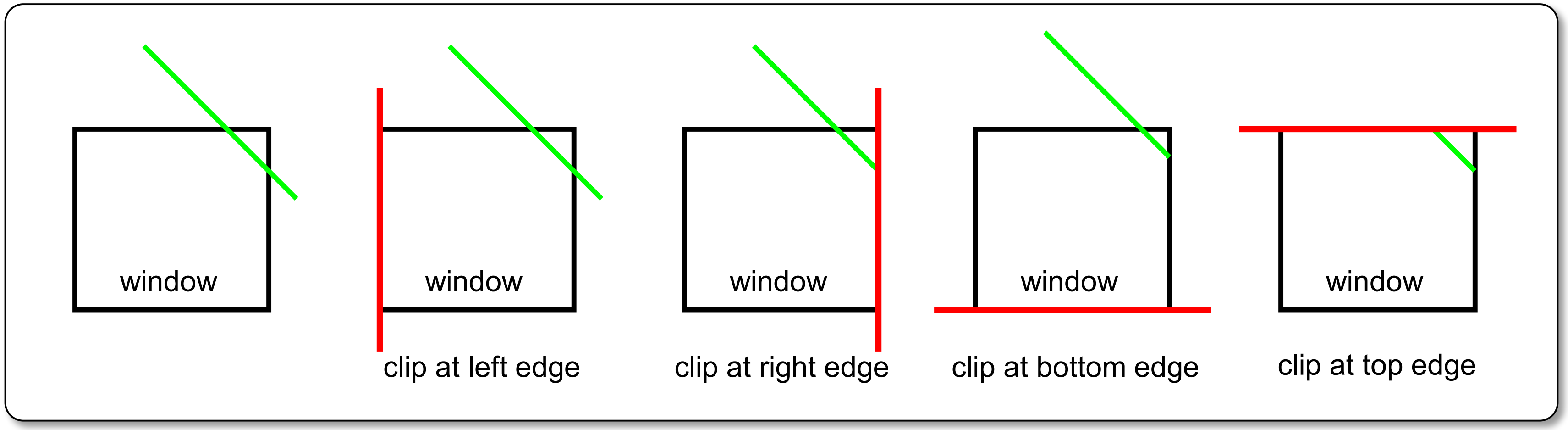
Special cases and speed-up

- For Vertical/horizontal lines, test and intersect only y-/x-bounds.
- If exactly one end point is inside the window:
  - ► There is only one intersection with the window-boundary.
- Determine unnecessary intersection-tests from the bit-codes:
  - Every bit corresponds to exactly one window-boundary.
  - ► Test only with those window-boundaries, whose corresponding bits differ in the end point bit-codes.
- Avoid bisection to compute the intersections:
  - Subdivide lines, that are neither completely inside nor outside the window, until the length of each sub-line is less than a pixel.
  - 2^{10} pixel in one row/column, yields at most 10 subdivisions.
  - In hardware, this variant is much faster due to a fast division by 2 and its parallelizability.

3.7 Clipping

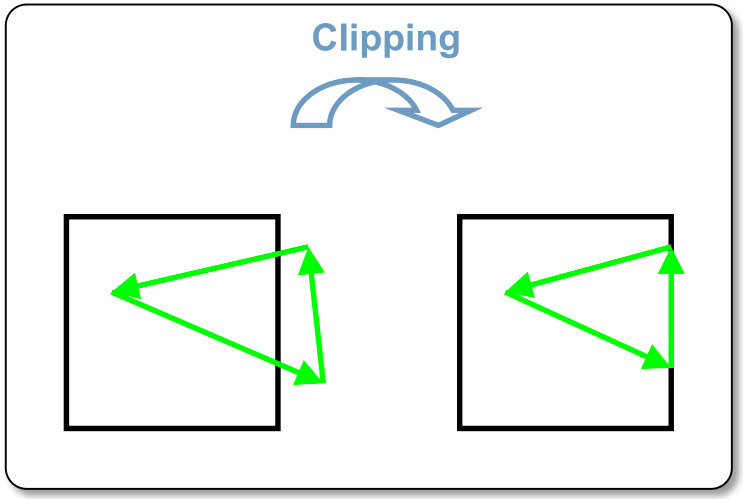
3.7.3 Clipping of polygons

- Polygons are important as boundaries of surfaces or surface areas.
- **Naïve approach:** Each edge is clipped at the window.

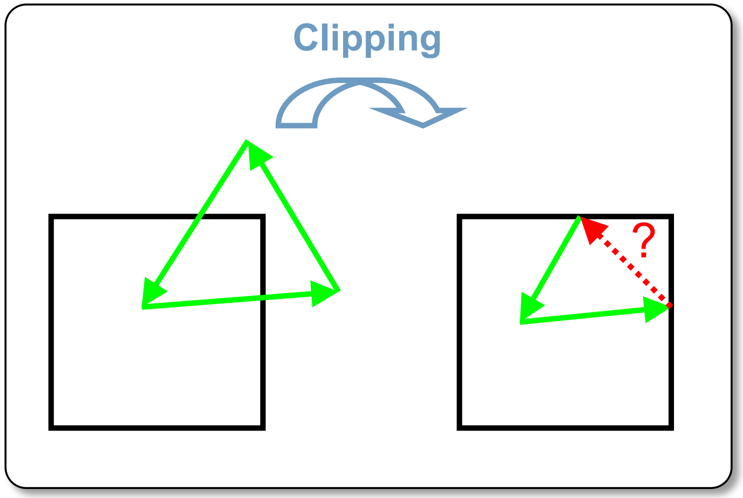


3.7 Clipping

- **Problem:** The output of polygon-clipping must be a closed polygon, i.e. it might be necessary that parts of the window boundary become edges of the clipped polygon.



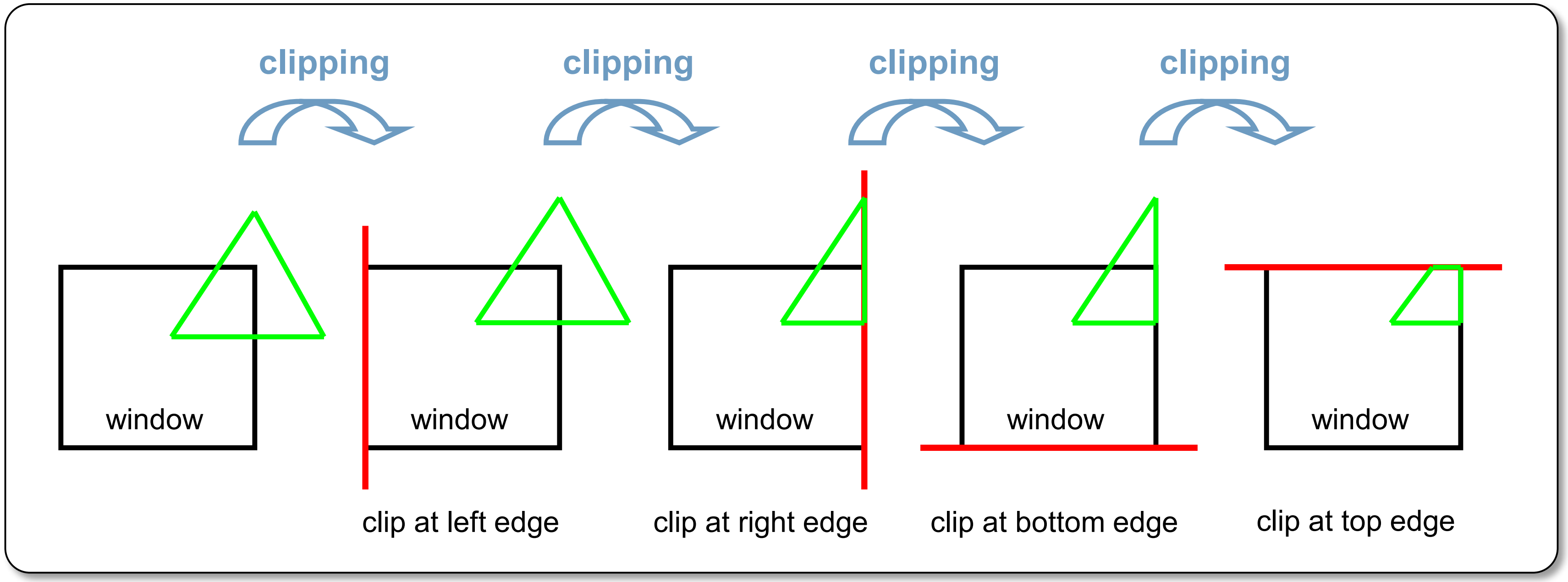
but



3.7 Clipping

3.7.4 Sutherland-Hodgman polygon-clipping algorithm

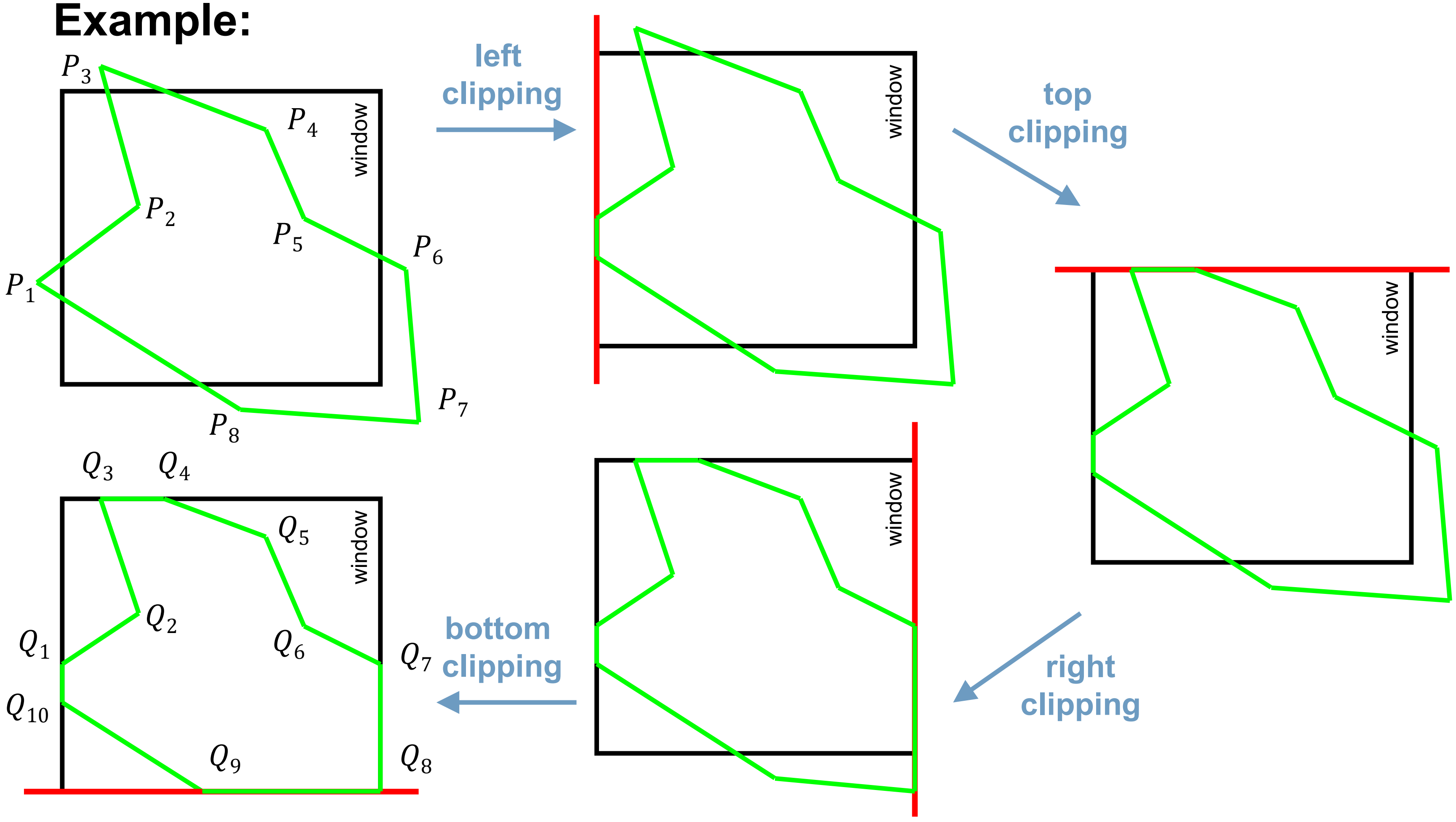
- Complete clipping of the polygon at each window boundary.



- All intermediate polygons need to be stored.

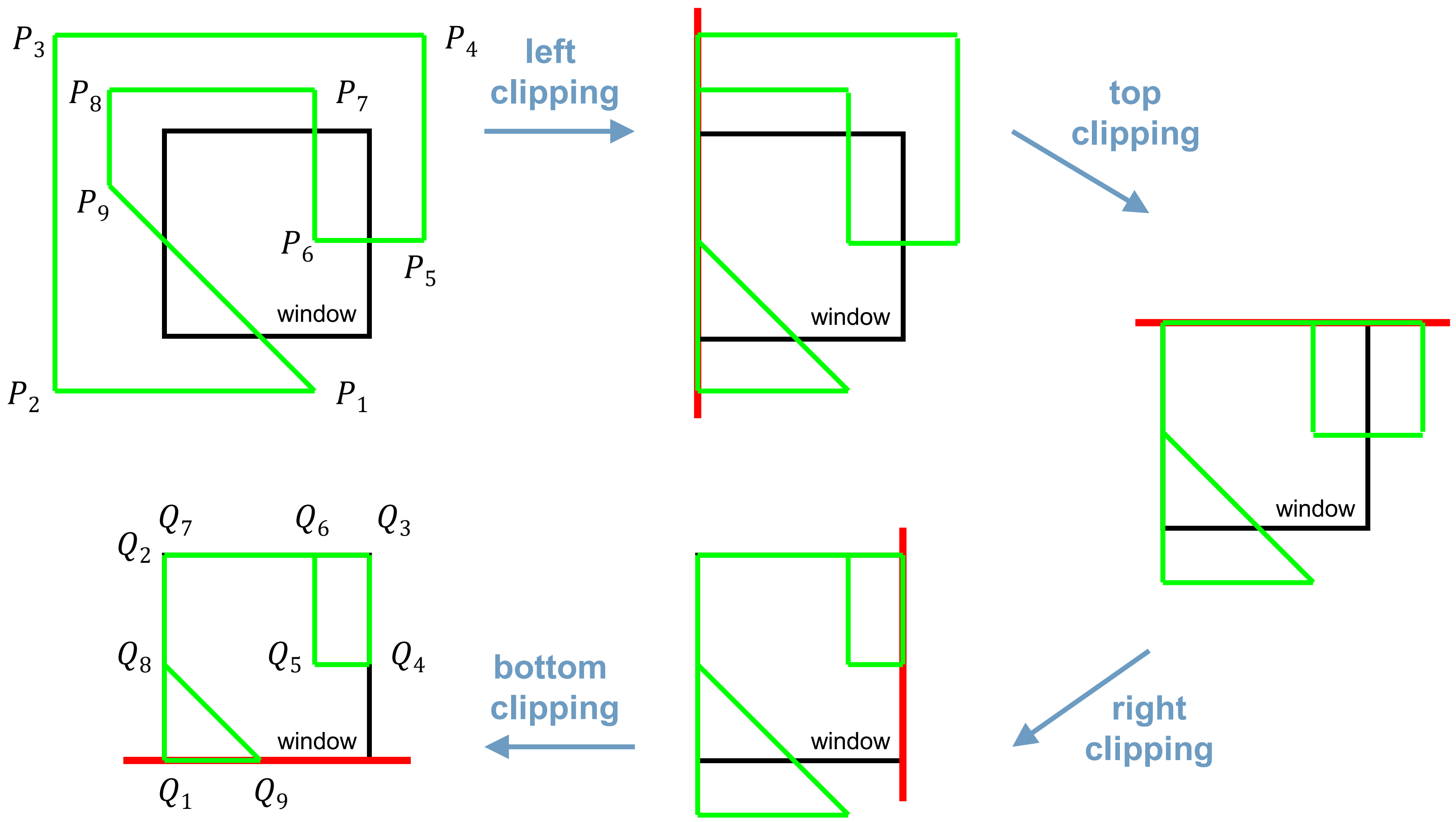
3.7 Clipping

Example:



3.7 Clipping

Example:



3.7 Clipping

3.7.5 Clipping in 3-space

- Instead of 2d-clipping after the projection in the image plane, objects can also be clipped in 3d-world space.
  - **Advantage:** Only visible objects need to be transformed and handed down to subsequent stages of the rendering pipeline.
- Using the plane equations of the boundaries of the frustum, the relative positions of given points (lines, objects, etc.) can be classified as inside or outside.
  - **Disadvantage:** More complex intersection computations.
- The methods for 2d-clipping can be generalized analogously for 3d-clipping in space.

Goals

- What is the difference between object-, world-, camera-, image-, and display-coordinates?
- How are translations, rotations, and scalings represented in 2d and 3d?
- What is an affine transformation?
- What are homogenous coordinates?
- What is a perspective projection?
- What is a parallel projection ?
- What is the frustum?
- What is the difference between window and viewport?
- What is windowing?
- What is the principle of clipping-algorithms for lines and polygons?