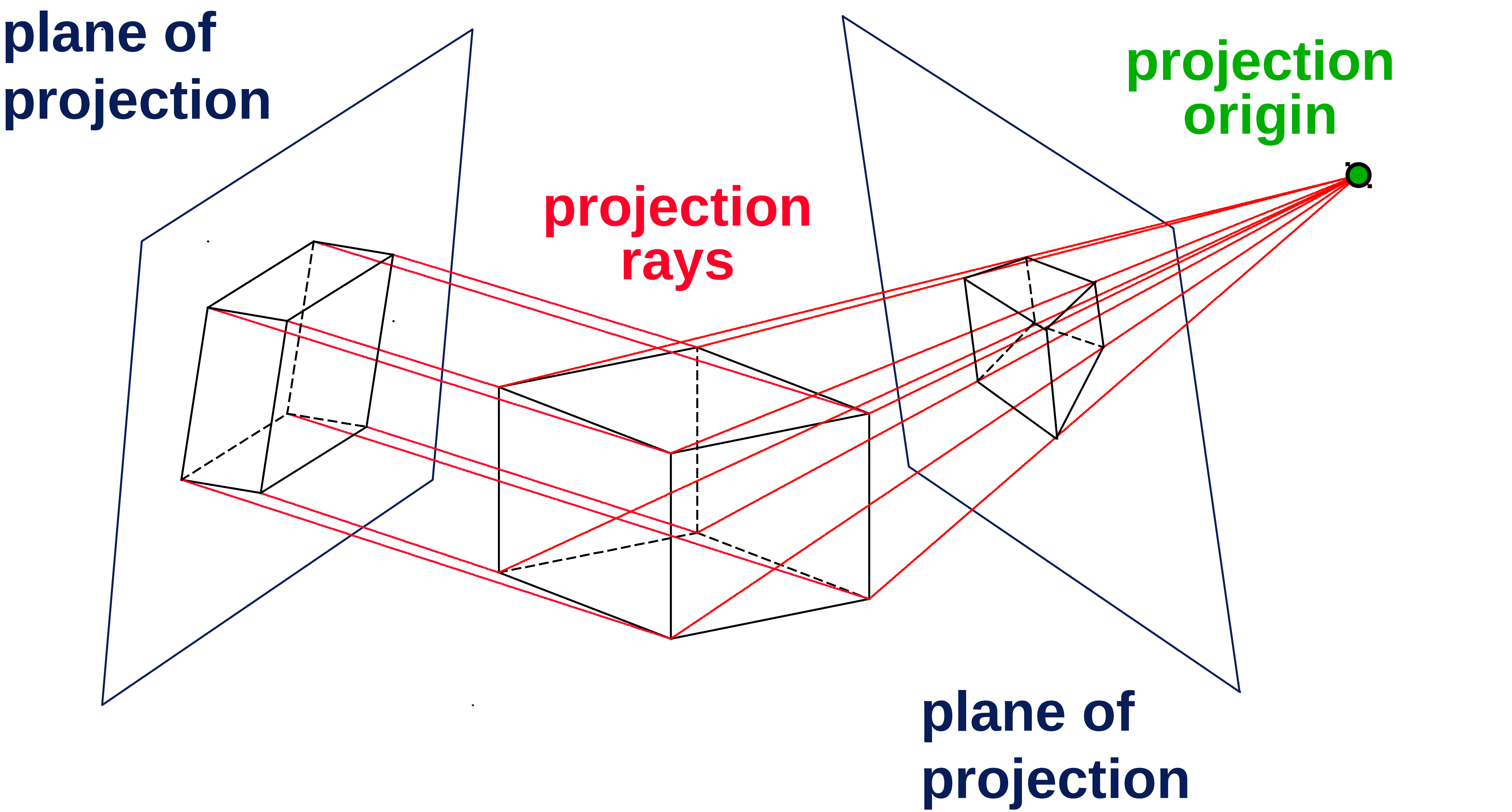
Computer Graphics Charles University

Projections

© 1995-2015 Josef Pelikán & Alexander Wilkie
CGG MFF UK Praha

pepca@cgg.mff.cuni.cz
http://cgg.mff.cuni.cz/~pepca/

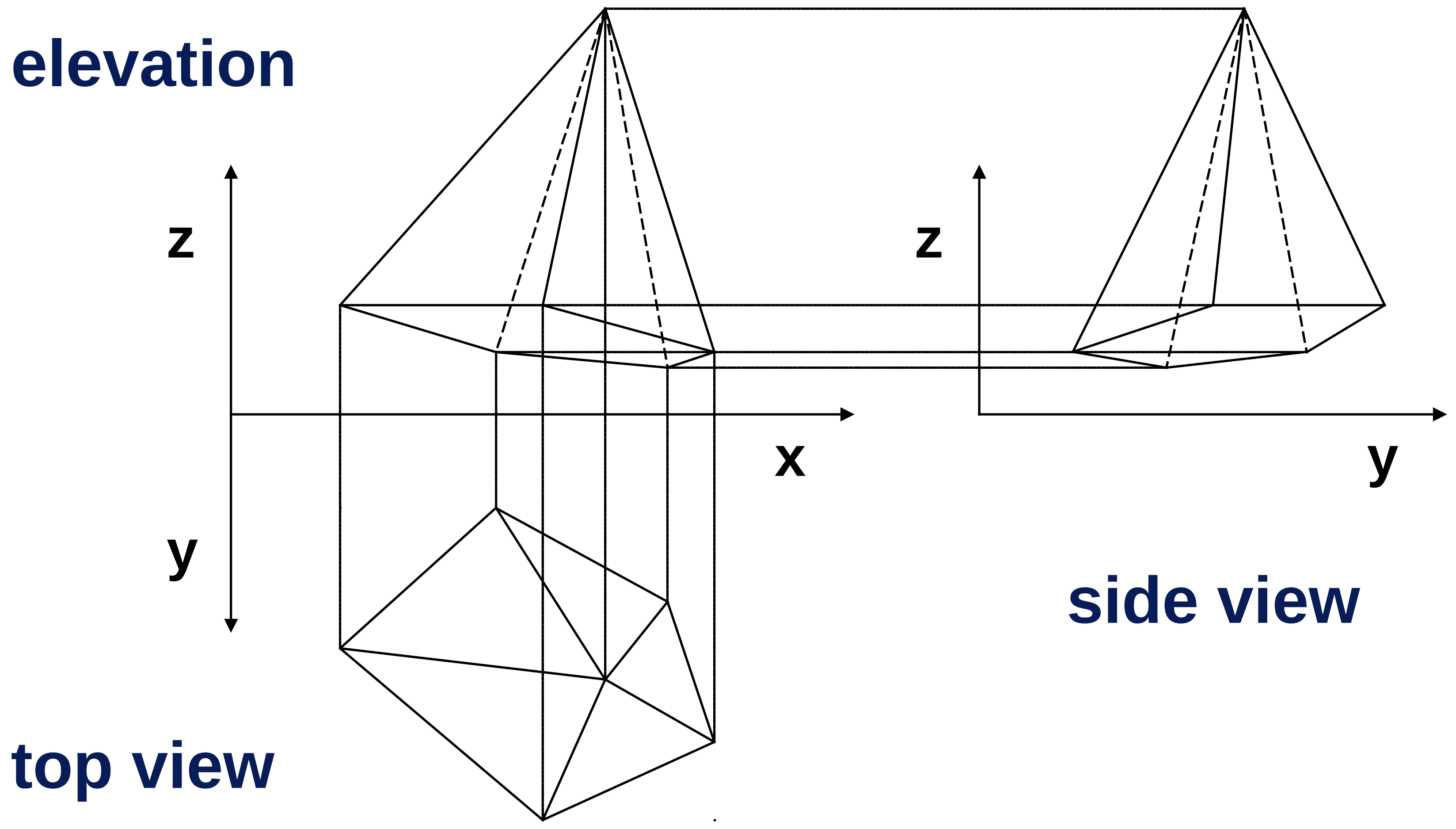
Basic Concepts



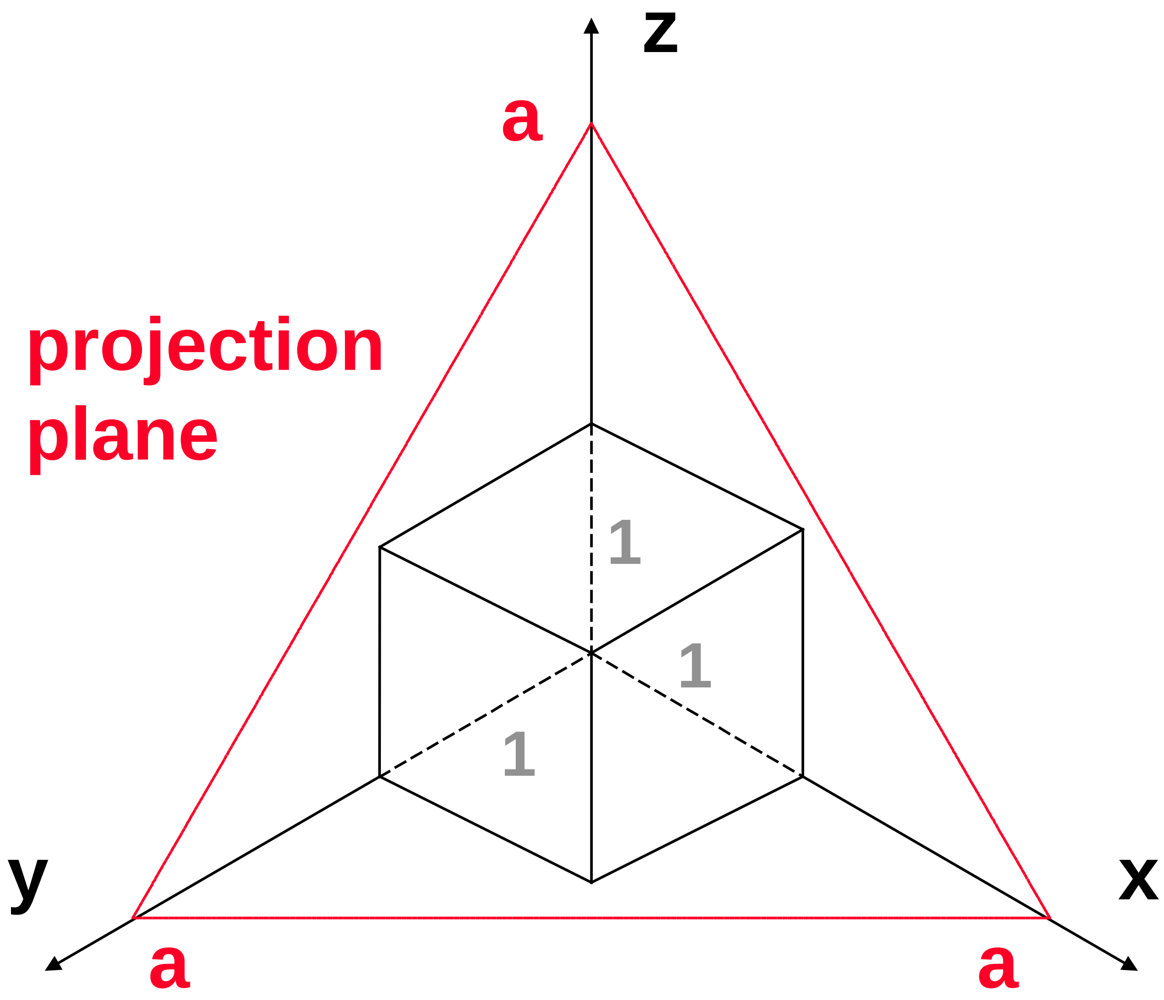
Classification of Linear Projections

- **Parallel projections**
  - Projection rays are parallel to each other
- **Orthogonal projections**
  - Projection rays are orthogonal to the projection plane
  - Monge projection, floor plan, elevation, side view
  - Axonometry (general orthogonal projection)
- **Oblique projections**
  - Cabinet projection (the **z** axis has ½ scale)
  - Cavalier projection (same scale on all axes)

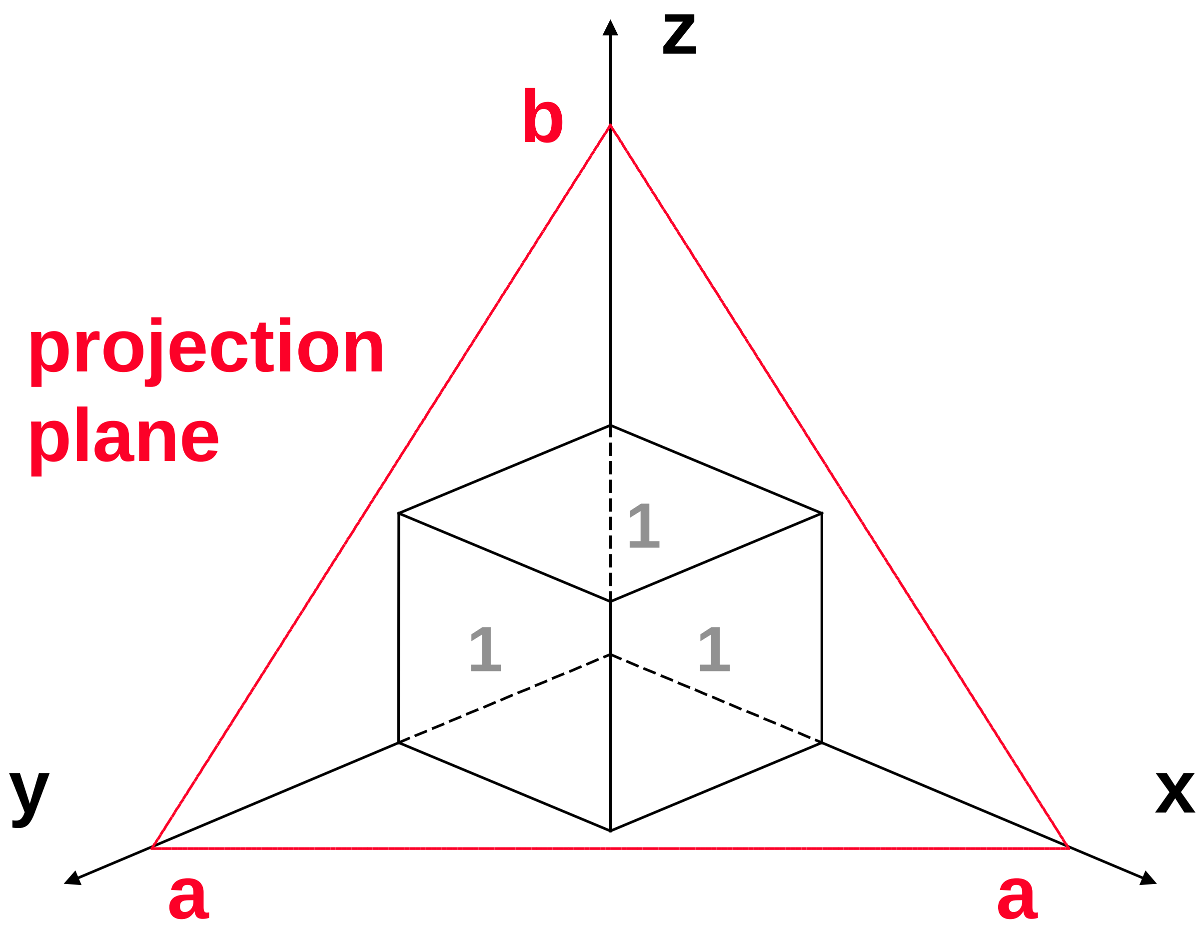
Monge projection



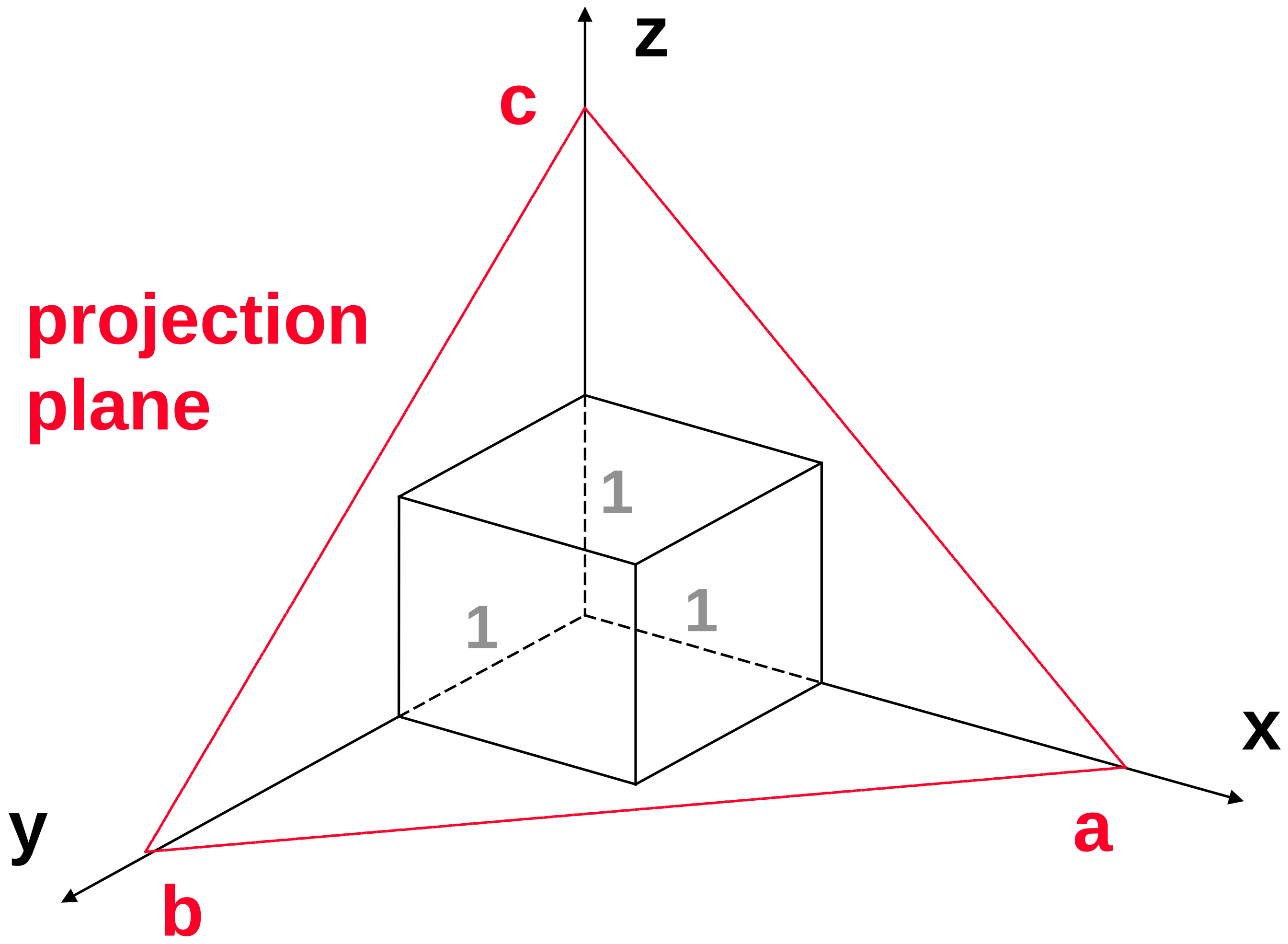
Axonometry – Isometric Projection



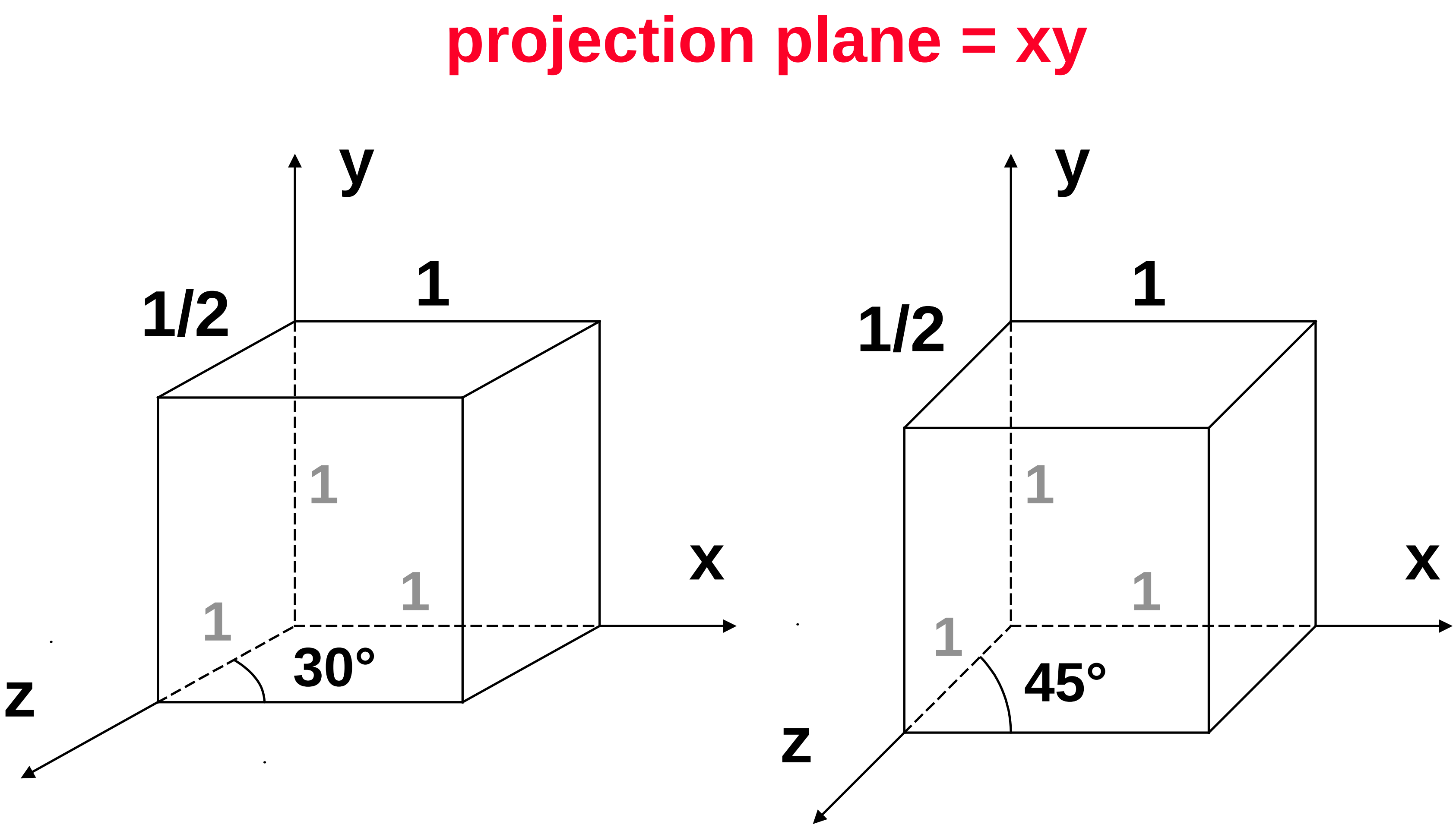
Axonometry – Dimetric Projection



Axonometry – Trimetric Projection

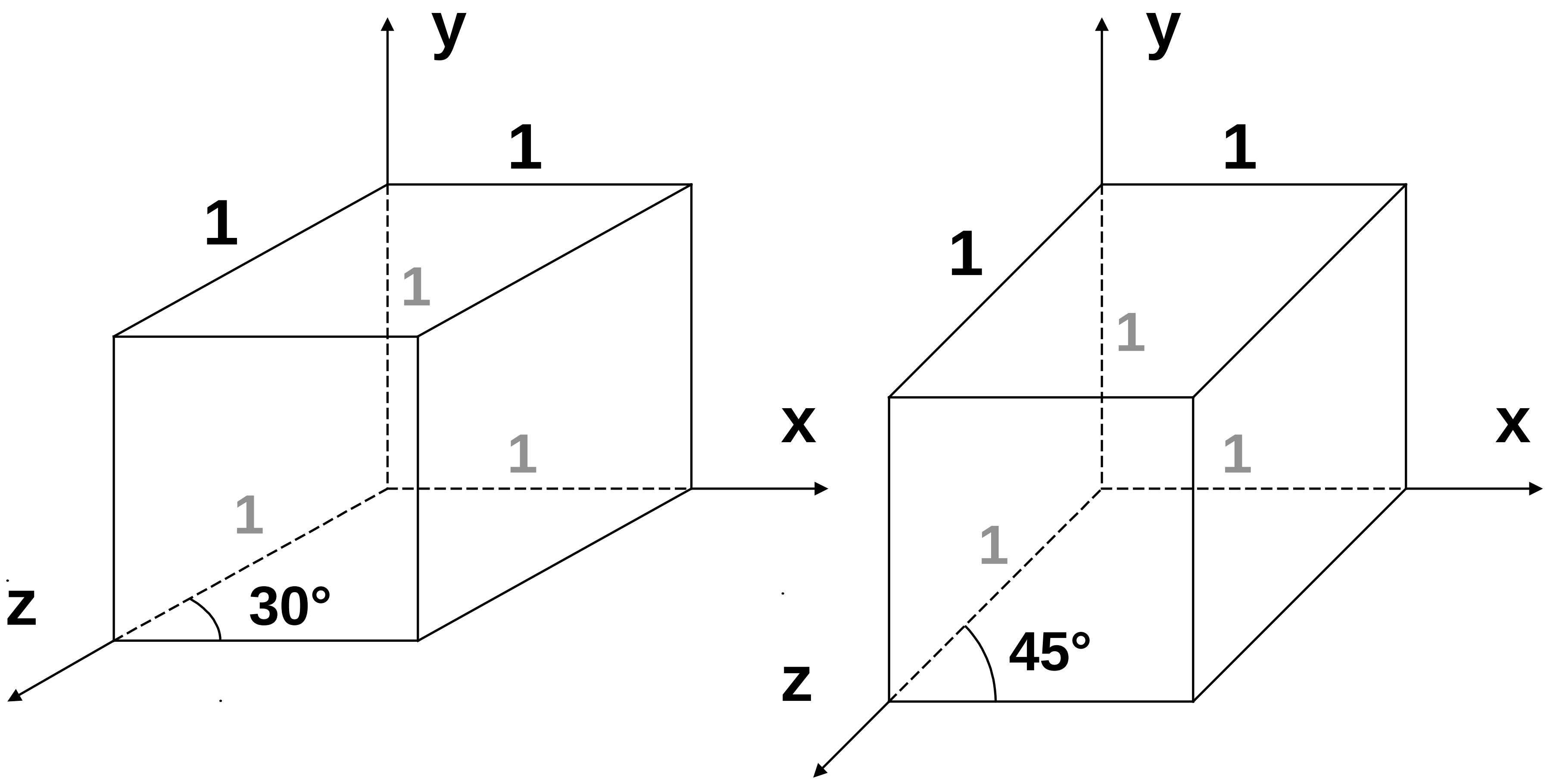


Cabinet Projection



Cavalier Projection

projection plane = xy

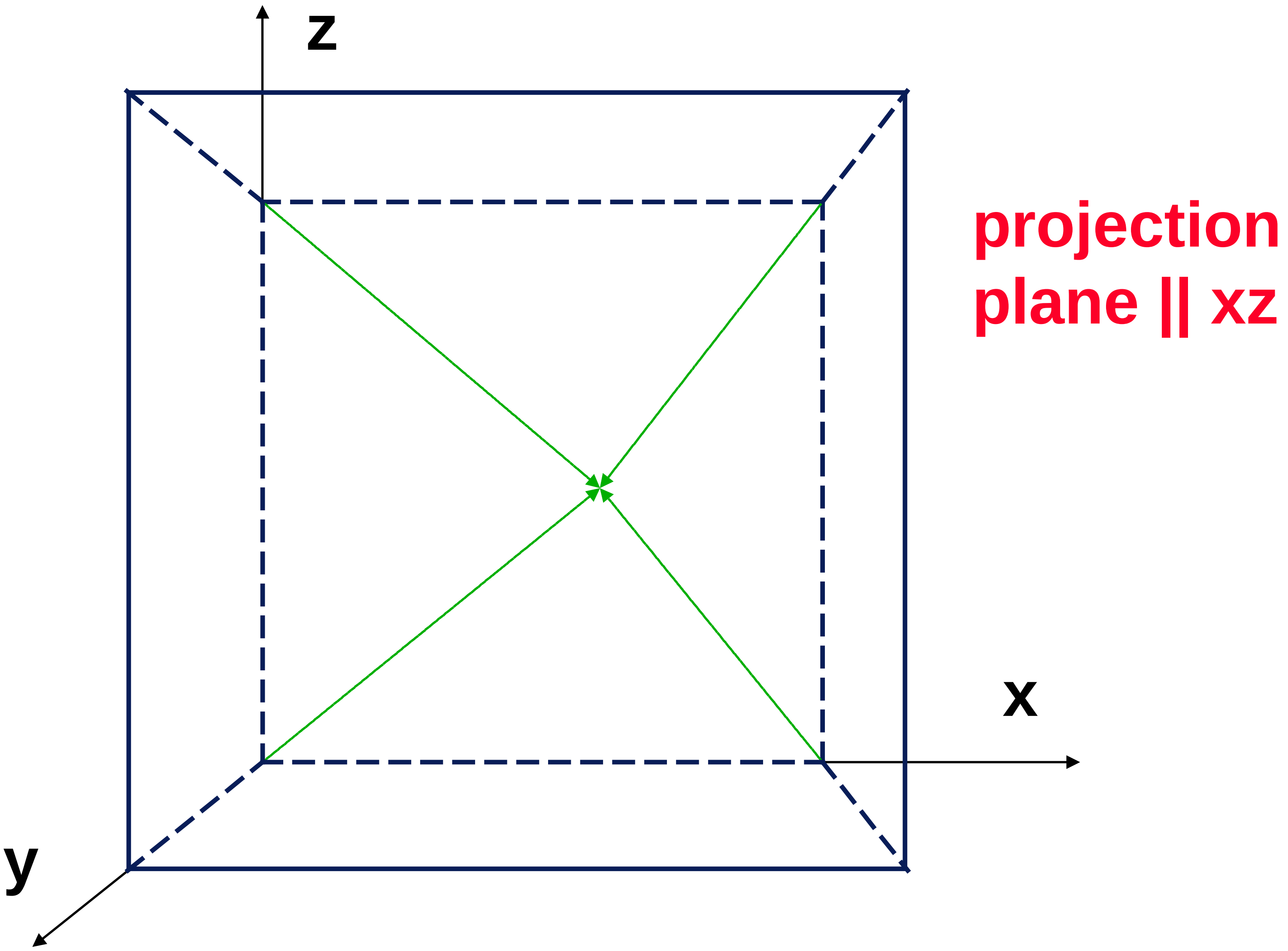


Classification of Linear Projections

- **(Central) perspective projections**
  - Projection rays form a beam that pass through a single point, the **center of the projection**
  - Do not preserve parallelism (vanishing points!)

- **One point perspective**
  - The plane of projection is parallel to two coordinate axes
  - Lines parallel to the third coordinate axis meet in one vanishing point

One Point Perspective

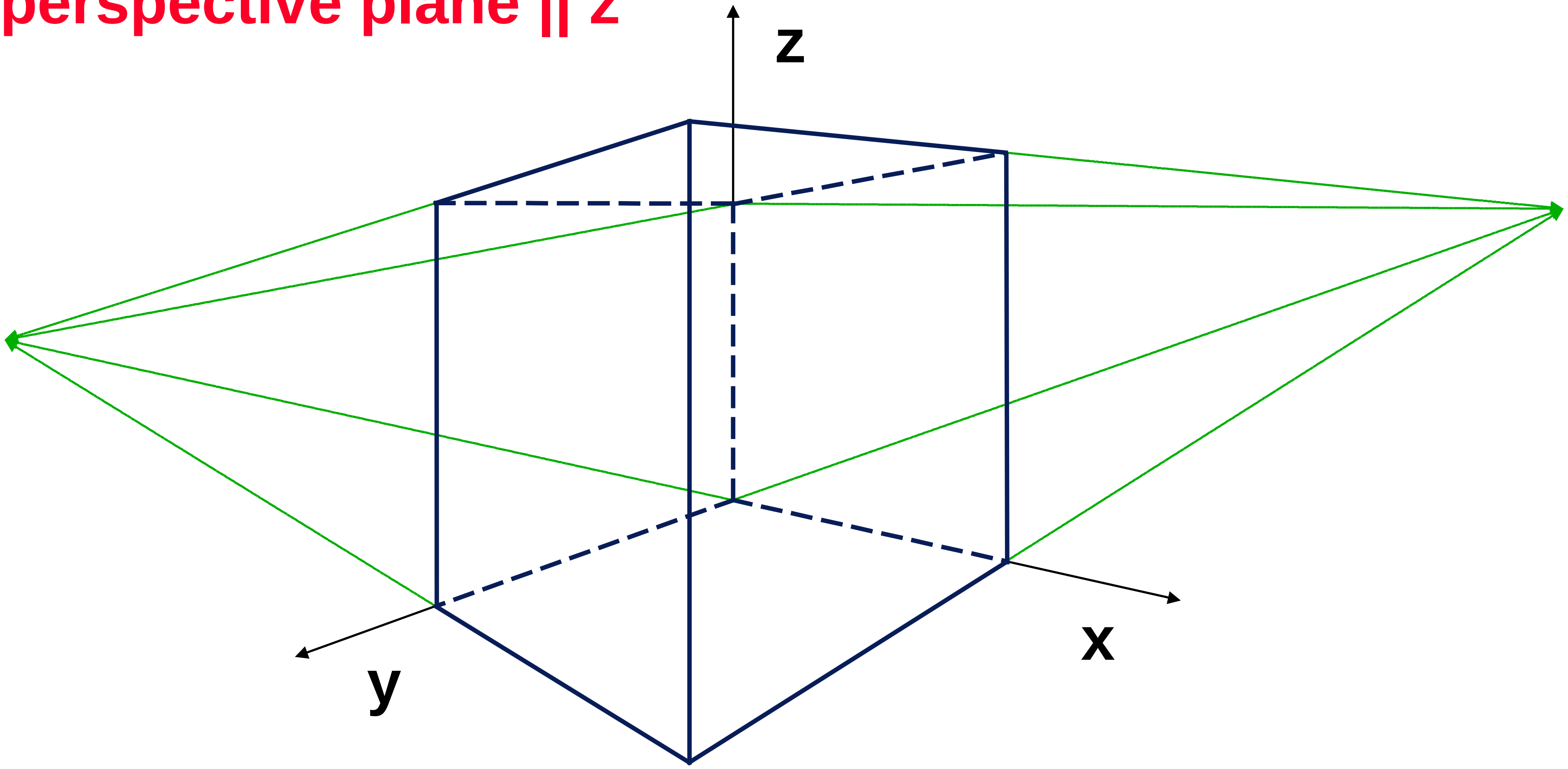


Classification of Linear Projections

- **Two point perspective**
  - The plane of projection is parallel to one coordinate axis
  - Lines parallel to the other axes meet in two vanishing points
- **Three point perspective**
  - The plane of projection is in an arbitrary orientation
  - Lines parallel to the coordinate axes meet in three vanishing points

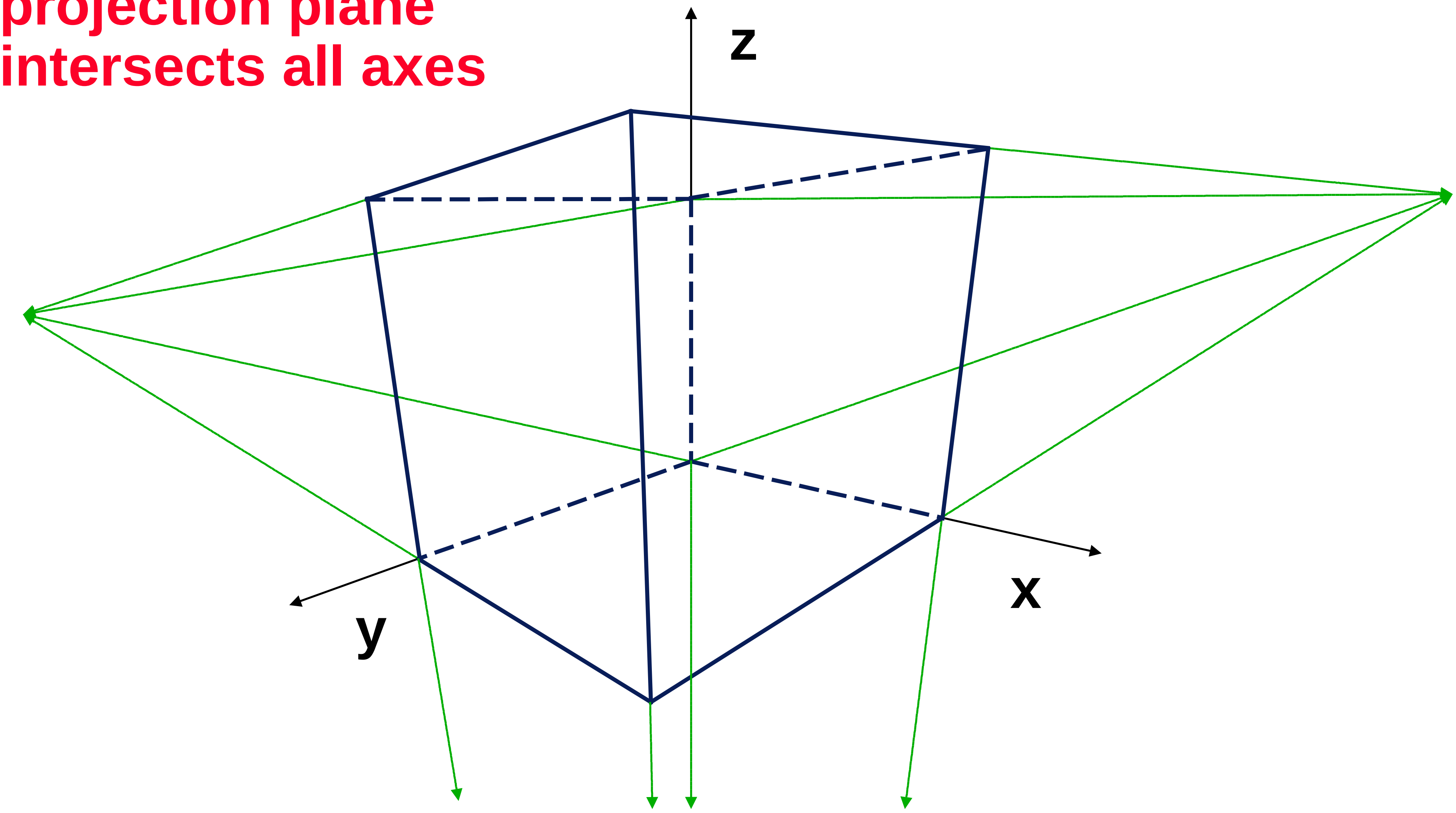
Two Point Perspective

perspective plane || z



Three Point Perspective

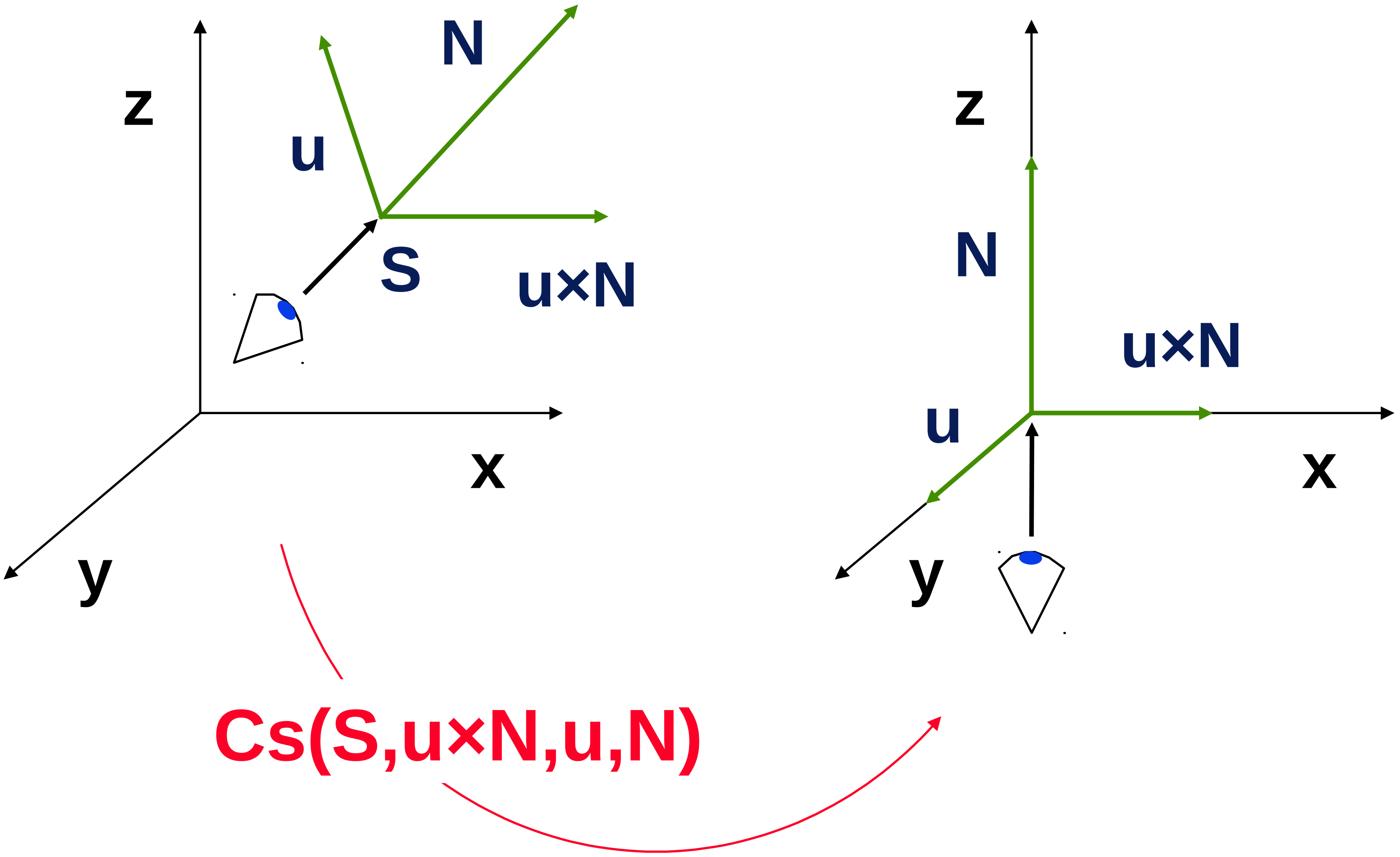
projection plane intersects all axes



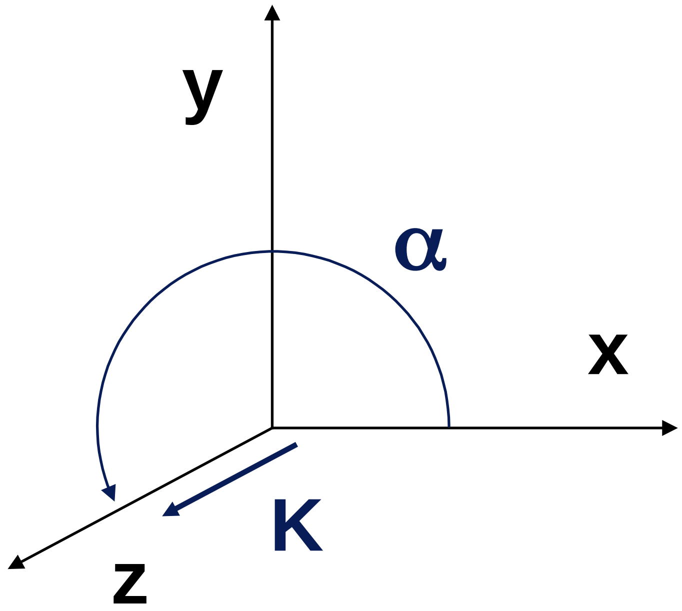
Orthogonal Projection Implementation

- **[x,y]** are usually coordinates in the viewing plane, and **z** depth (distance from the viewer)
- **Fundamental views** (top, front, side)
  - These are just permutations of the **x**, **y** and **z** axes (with possible sign change)
- **General orthogonal projection** (isometric)
  - **View direction** (normal of the projection plane): **N**
  - **Orientation vector (up)**: **u**
  - Transformation: **Cs(S,u×N,u,N)**

Orthogonal Projection



Oblique Projection Implementation



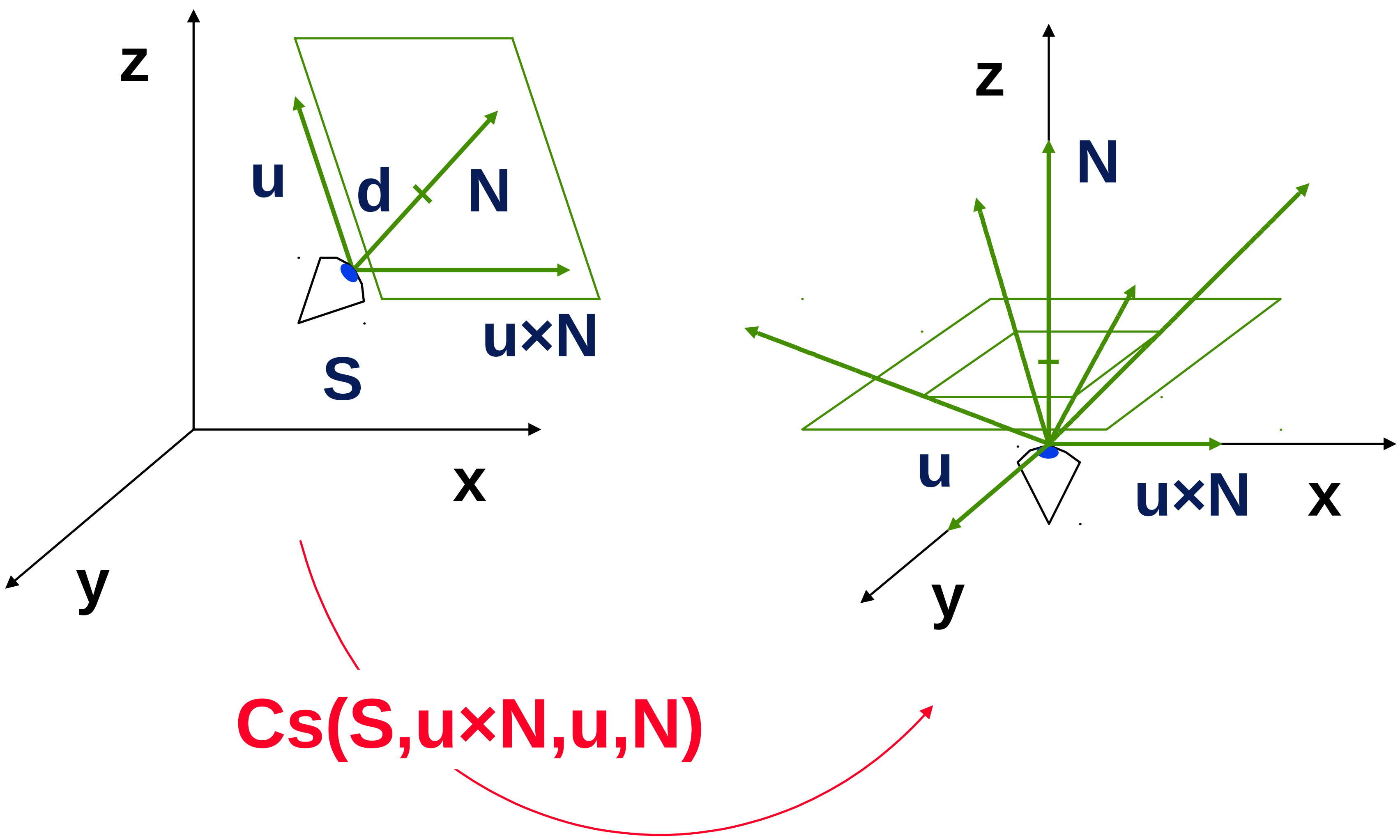
perspective plane: xy
foreshortening coefficient: K
angle of the projection axis z: α

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ K \cdot \cos\alpha & K \cdot \sin\alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

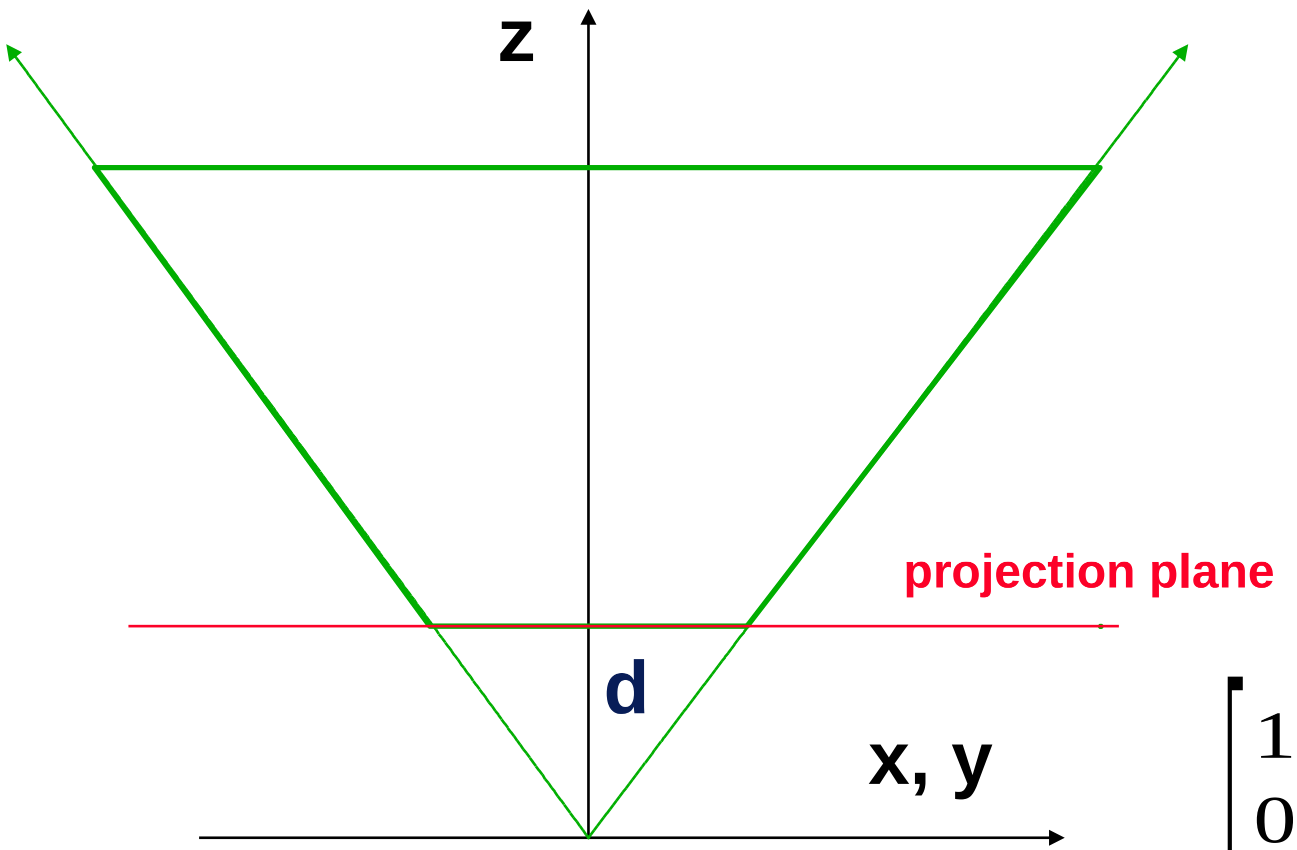
Central Projection Implementation

- **General perspective projection**:
  - **Center of the projection**: **S**
  - **View direction** (normal of the perspective plane): **N**
  - **Distance of the plane** from the center of the projection: **d**
  - **Orientation vector (up)**: **u**
- **Projection transformation**:
  - Use **standard orientation** (center at the origin, view direction along **z**): **Cs(S,u×N,u,N)**
  - **Perspective projection**: e.g. **[x· d/z, y· d/z, z]**

Using the Standard Orientation



Perspective Transform



Does NOT conserve linearity!

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Transformation of Linear Objects

- **Perspective transform of lines Per**:
  - The following equation obviously does **not hold:**

$$\mathbf{Per(A + t \cdot [B - A]) = Per(A) + t \cdot [Per(B) - Per(A)]}$$

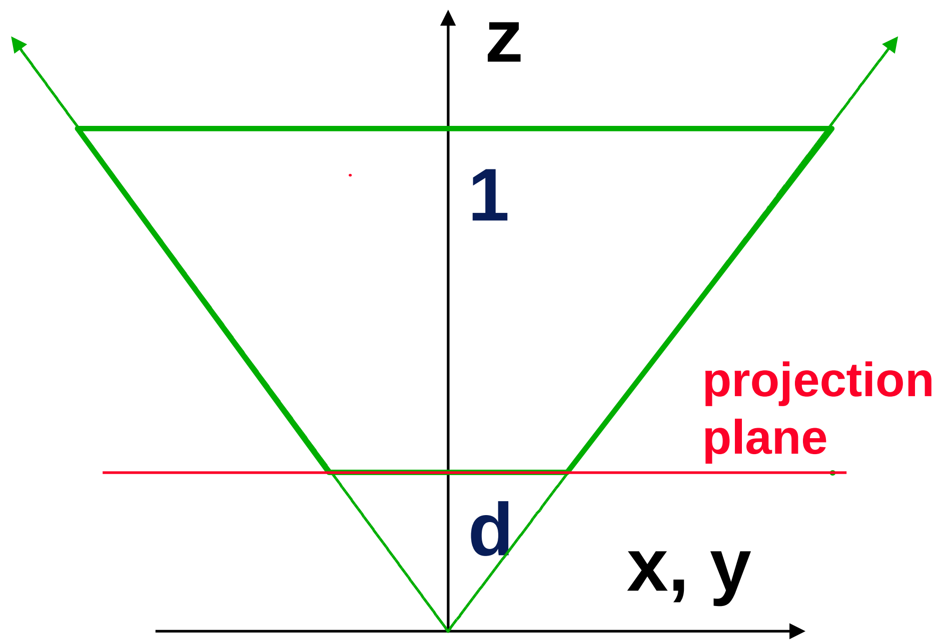
- Using a **difference algorithm (DDA)** for visibility calculations:
  - Given point **C(u)** on the segment **Per(A)Per(B)**:

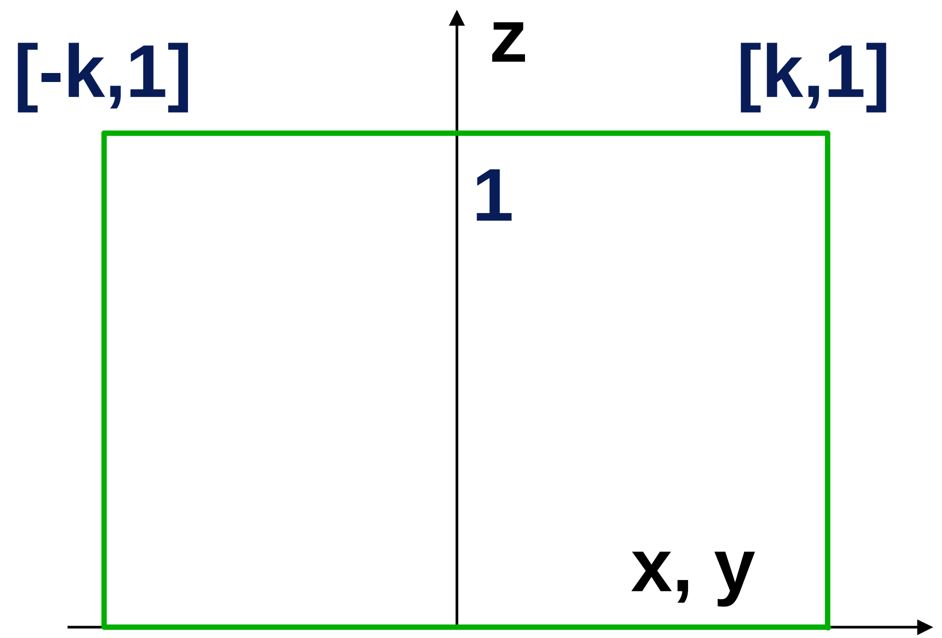
$$\mathbf{C(u)_{x,y} = Per(A)_{x,y} + u \cdot [Per(B)_{x,y} - Per(A)_{x,y}]}$$

  - This also has to hold for depth **z**:

$$\mathbf{C(u)_{z} = Per(A)_{z} + u \cdot [Per(B)_{z} - Per(A)_{z}]}$$

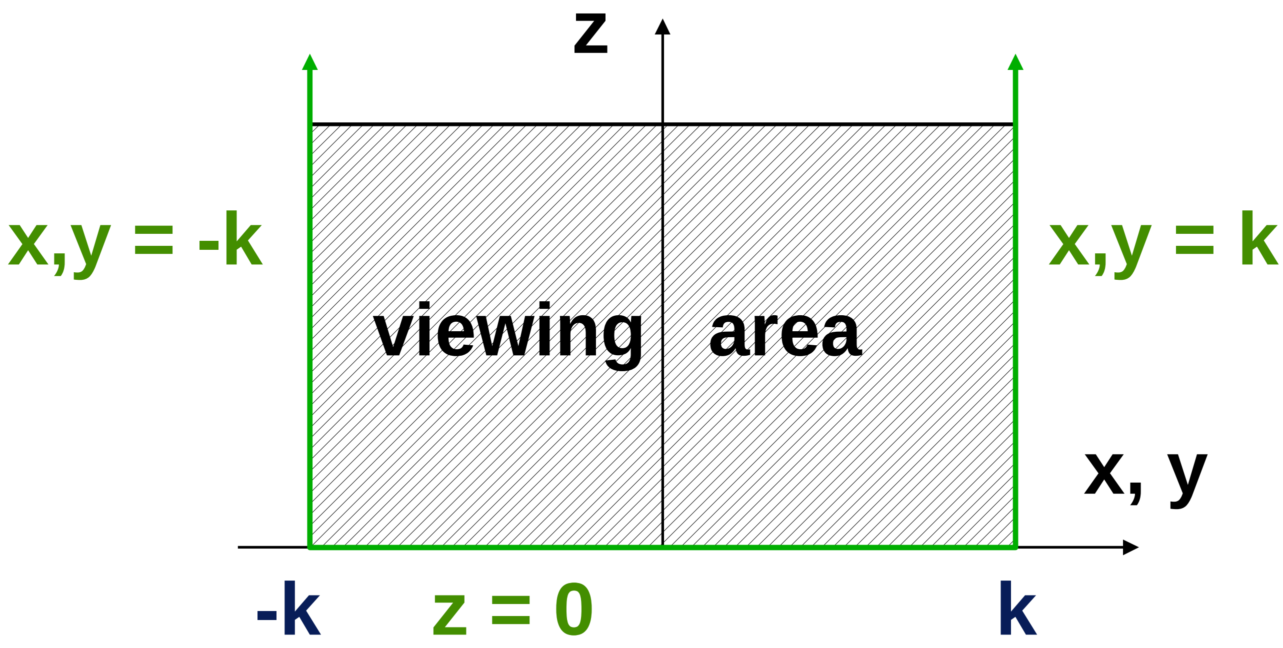
Conservation of Linearity





$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1-d} & 1 \\ 0 & 0 & \frac{-d}{1-d} & 0 \end{bmatrix}$$

4D Clipping



limit hyperplane:

x = -kw, x = kw, y = -kw, y = kw, z = 0

for w > 0: -kw < x < kw, -kw < y < kw, 0 < z

End

Further Information

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 229-283
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 277-291