Basics of 3D Graphics

I.Kolingerová

1. Geometric transformations in 3D
2. Projections
3. Hidden parts removal
4. Lighting and shading

1. Geometric transformations in 3D

- Generalization of planar transformations

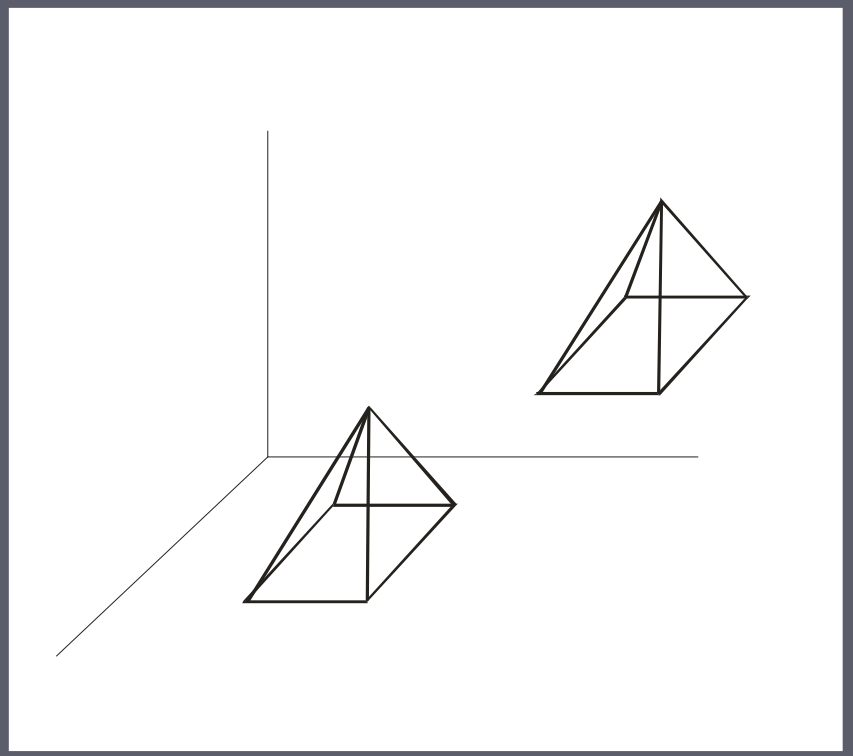
1. Translation

by a vector (x_t, y_t, z_t):

$$x' = x + x_t$$

$$y' = y + y_t$$

$$z' = z + z_t$$



2. Rotation

one of coordinate axes is the axis of rotation

- x axis (in the yz plane):

$x' = x$

$y' = y*\cos(\alpha) - z*\sin(\alpha)$

$z' = y*\sin(\alpha) + z*\cos(\alpha)$

- y axis (in the zx plane):

$x' = x*\cos(\alpha) + z*\sin(\alpha)$

$y' = y$

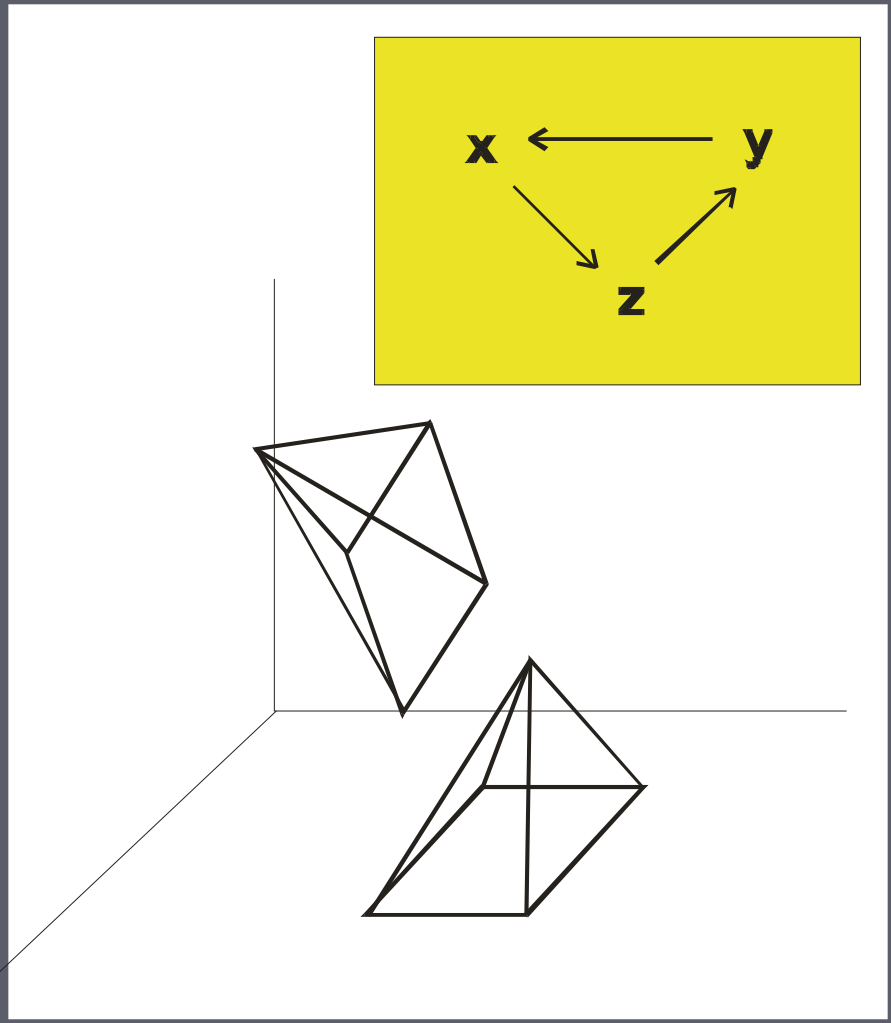
$z' = -x*\sin(\alpha) + z*\cos(\alpha)$

- z axis (in the xy plane):

$x' = x*\cos(\alpha) - y*\sin(\alpha)$

$y' = x*\sin(\alpha) + y*\cos(\alpha)$

$z' = z$



Rotation around a general axis $\mathbf{x}=\mathbf{P}+t(\mathbf{Q}-\mathbf{P})$:

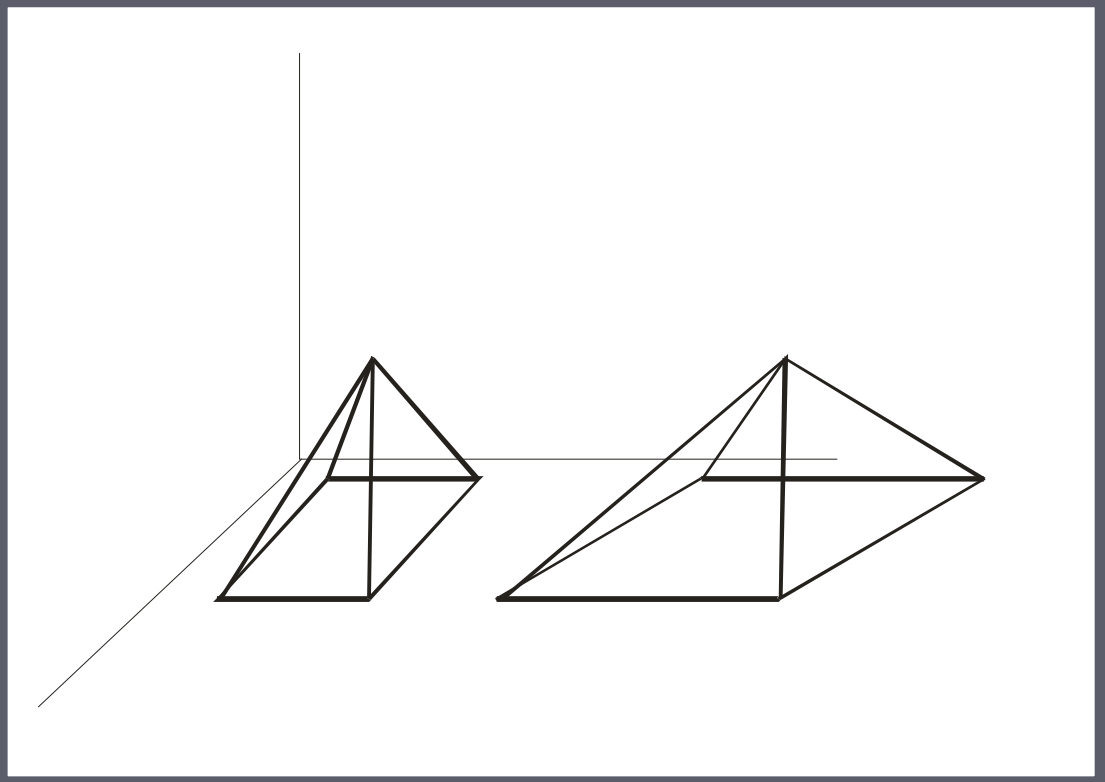
- Translate the coordinate system (CS) to P (1)
- Rotate around one of CS axes, so that the axis of rotation is one of the CS axes (2)
- Rotate by the required angle (3)
- Inverse rotation (2^{-1})
- Inverse translation (1^{-1})

3. Scale

$x' = sc_x * x$

$y' = sc_y * y$

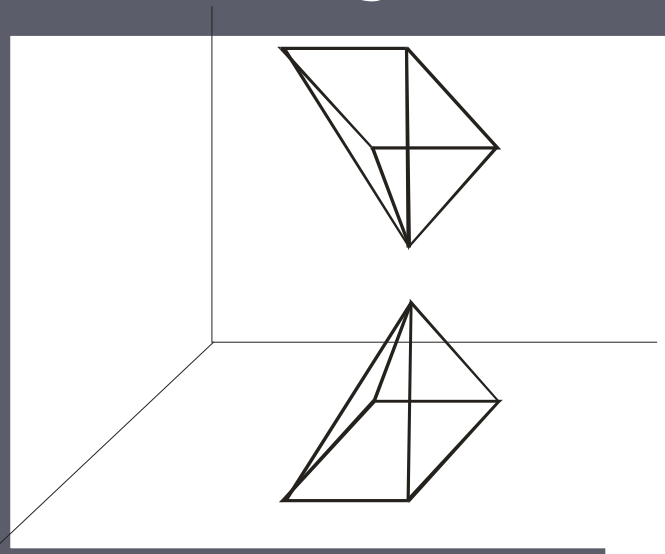
$z' = sc_z * z, \quad sc_x, sc_y, sc_z <> 0$



Other, less important transformations:

4. Mirror

- one of coordinates gets "-"



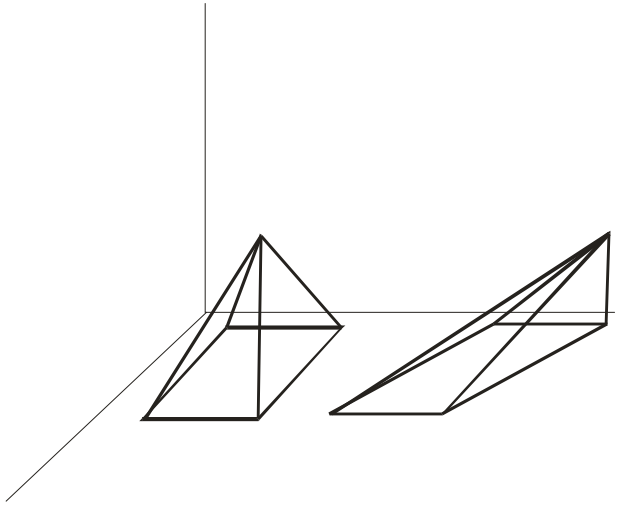
5. Shear

- e.g., in the x, y directions

$x' = x + sh_x * z$

$y' = y + sh_y * z$

$z' = z$

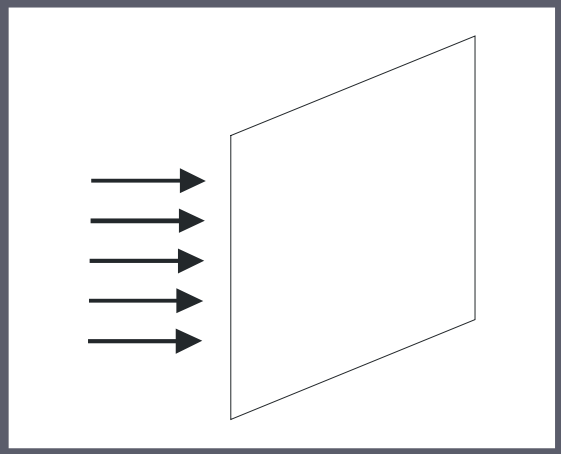


2. Projections

- A tranformation from nD to mD, n>m
- Terms:
  - projecting ray – a line through the projected point, the direction according to the selected projection method
  - projection plane – a plane in 3D onto which the projecting rays make a projection; usually // with xy
  - If the projection plane is planar, 3D lines project to planar lines => we project only vertices.

Parallel projection

- All rays are parallel, 2 projection types according to their angle with the projection plane:
  - Rectangular (90°)
  - Oblique(-angled) projections (other angles)

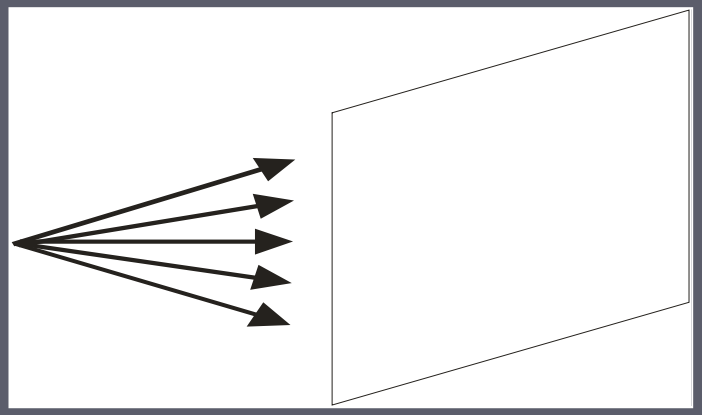


- In CG usually the rectangular projection
- Properties: keeps parallelism, the size of the projections is not influenced by the distance from the projection plane
- Use – technical applications

- Parallel projection into the xy-plane by rectangular rays described by s=(0,0,-1) – leaving out the z-coordinates
- Other directions: leave out other coordinate or first translate, rotate

Central (perspective) projection

- All projecting rays start in one point – a centre of projection



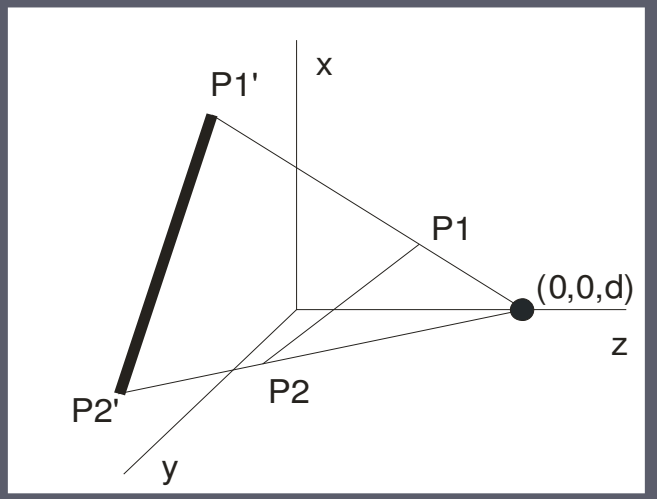
- 3D lines segments again project into planar
- Properties: generally it does not keep parallelism (kept only for line segments in the plane // the projection plane), the size of projections is influenced by the distance from the proj. plane (more distant objects – smaller)
- Use – architecture, virtual reality ...

a) Projection from (0,0,d) to z=0:

P=(xp,yp,zp), **P**´=(x´,y´,0), z <> d

Ray from the centre via **P** to **P**´:

x´= 0+x*t
y´= 0+y*t
0 = d+(z-d)*t



=> t= d/(d-z)

x´=x*d/(d-z), y´=y*d/(d-z), z´= 0

or

x´=x/(1-z/d), y´=y/(1-z/d), z´= 0

b) Projection from (0,0,0) to z=d:

P=(xp,yp,zp), **P**´=(x´,y´,0), z <> 0

Ray from the centre via **P** to **P**´:

$$x' = 0 + x*t$$
$$y' = 0 + y*t$$
$$d = 0 + z*t$$

=> t= d/z

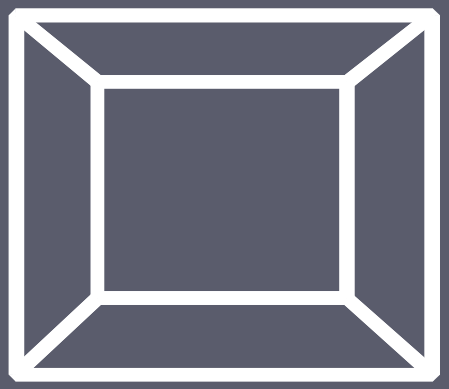
$$x' = x*d/z, \quad y' = y*d/z, \quad z' = d$$

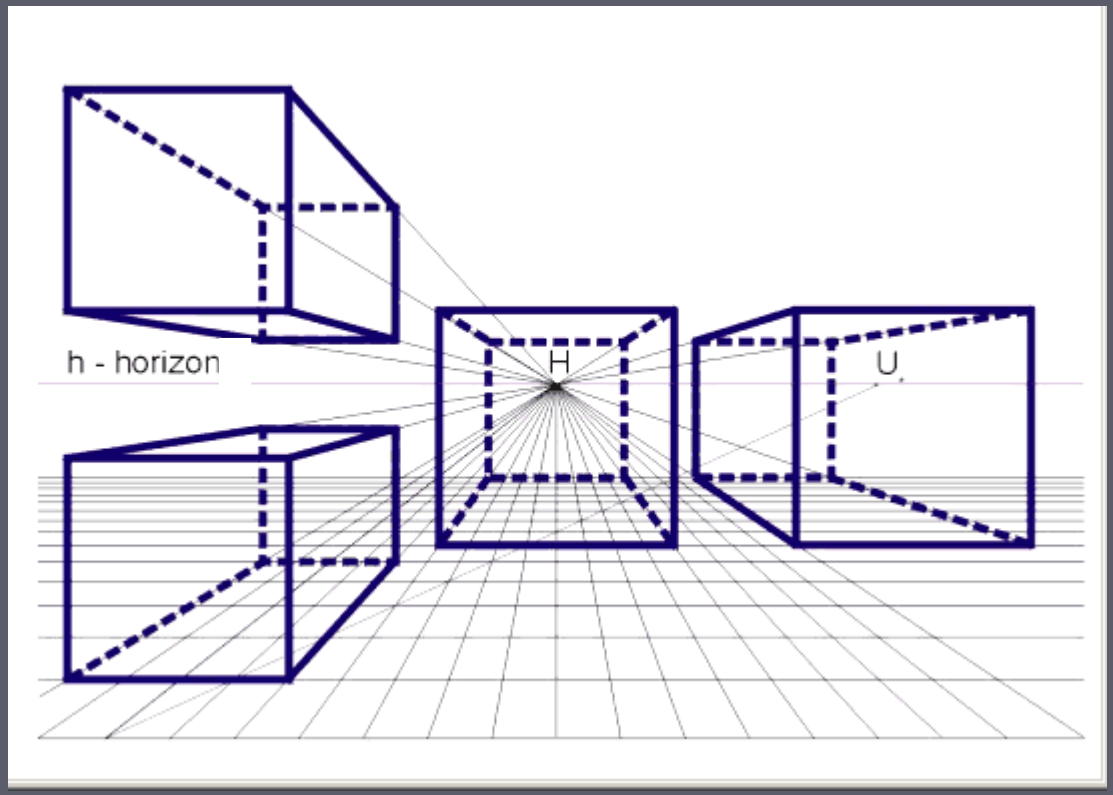
3 orientations of the projection plane to the CS axes:

1. One-point perspective – the proj. plane cuts one CS axis, all line (segments) perpendicular to the plane go to one point – main vanish-point
2. Two-points perspective – the proj. plane cuts two CS axes, the edges of axis-oriented cuboids go to two main vanish-points
3. Three-points perspective – the proj. plane cuts all three CS axes, the edges of axis-oriented cuboids go to three main vanish-points

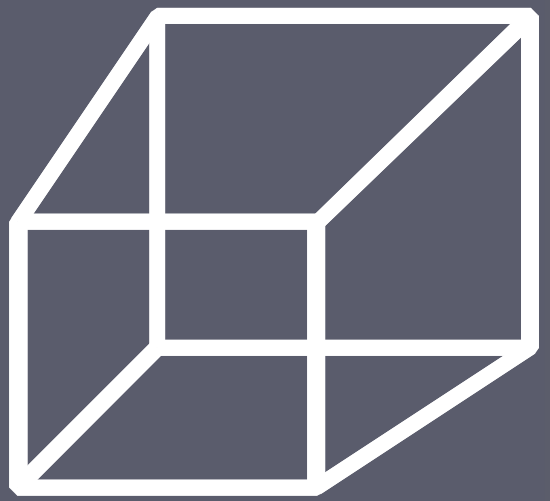
Vanish-point – "infinity, where the parallel lines meet".

There are many other vanish-points in the perspective projection, but only the 3 described are main.

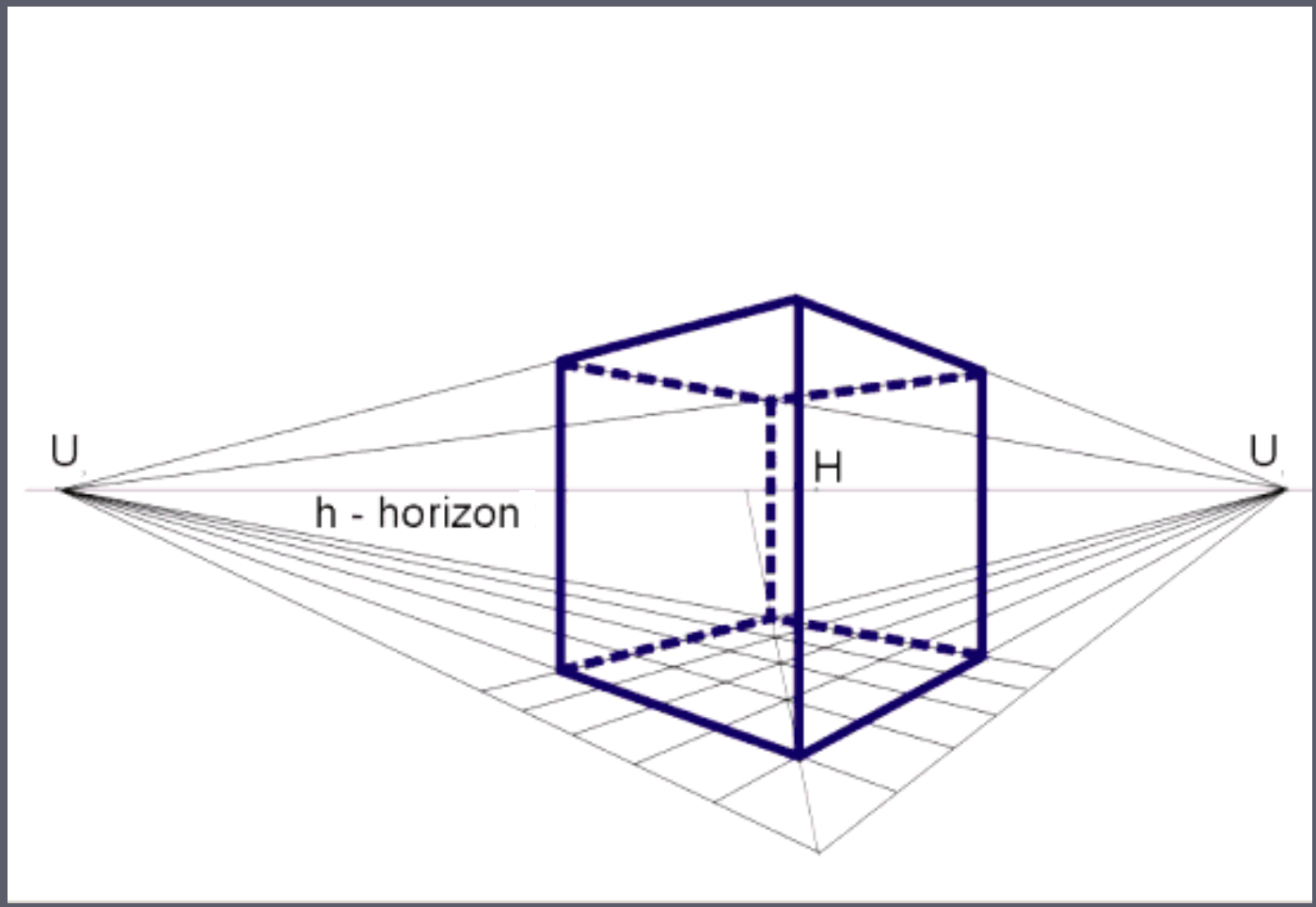




One-point perspective

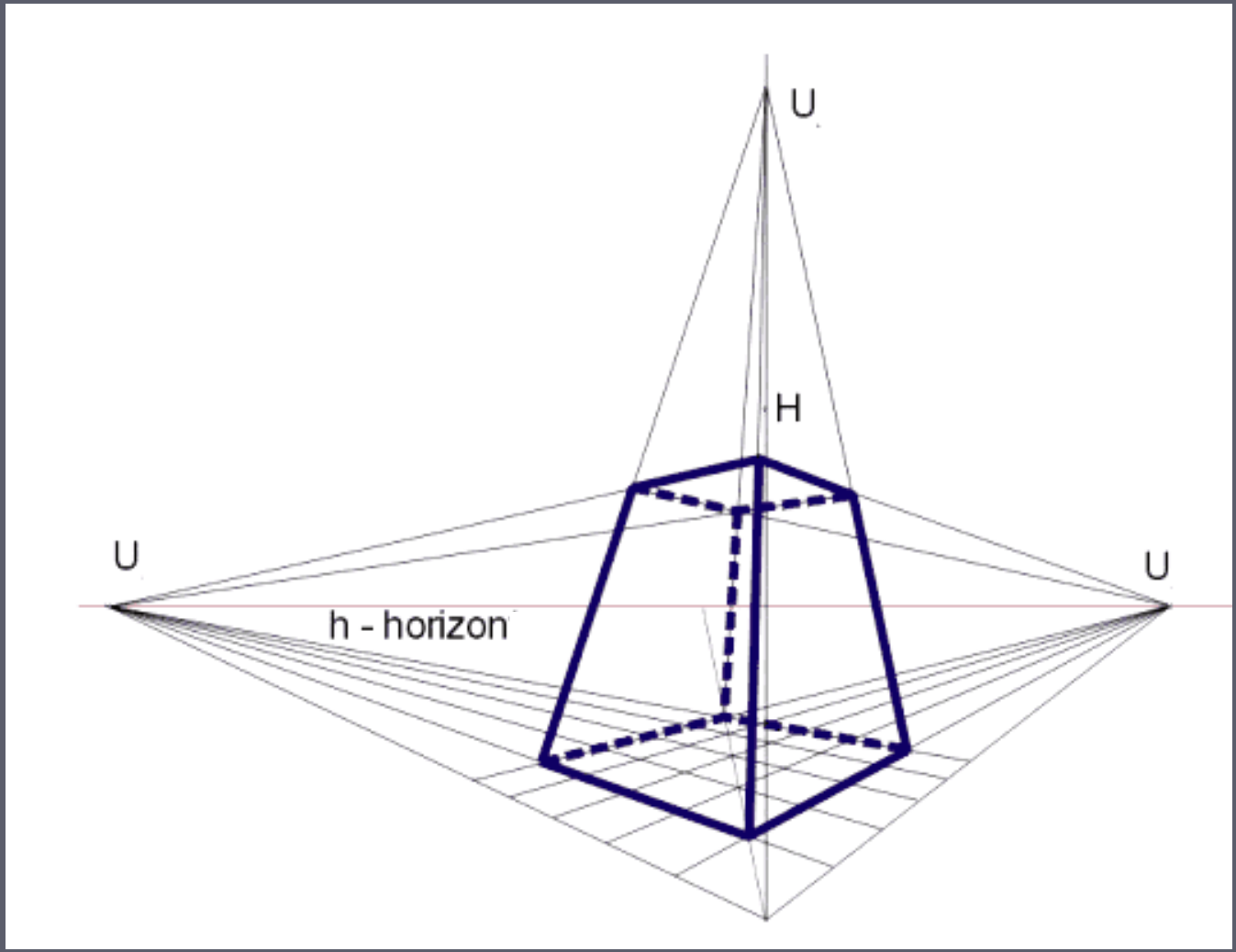


KPG-E 14/30



Two-points perspective

KPG-E 15/30

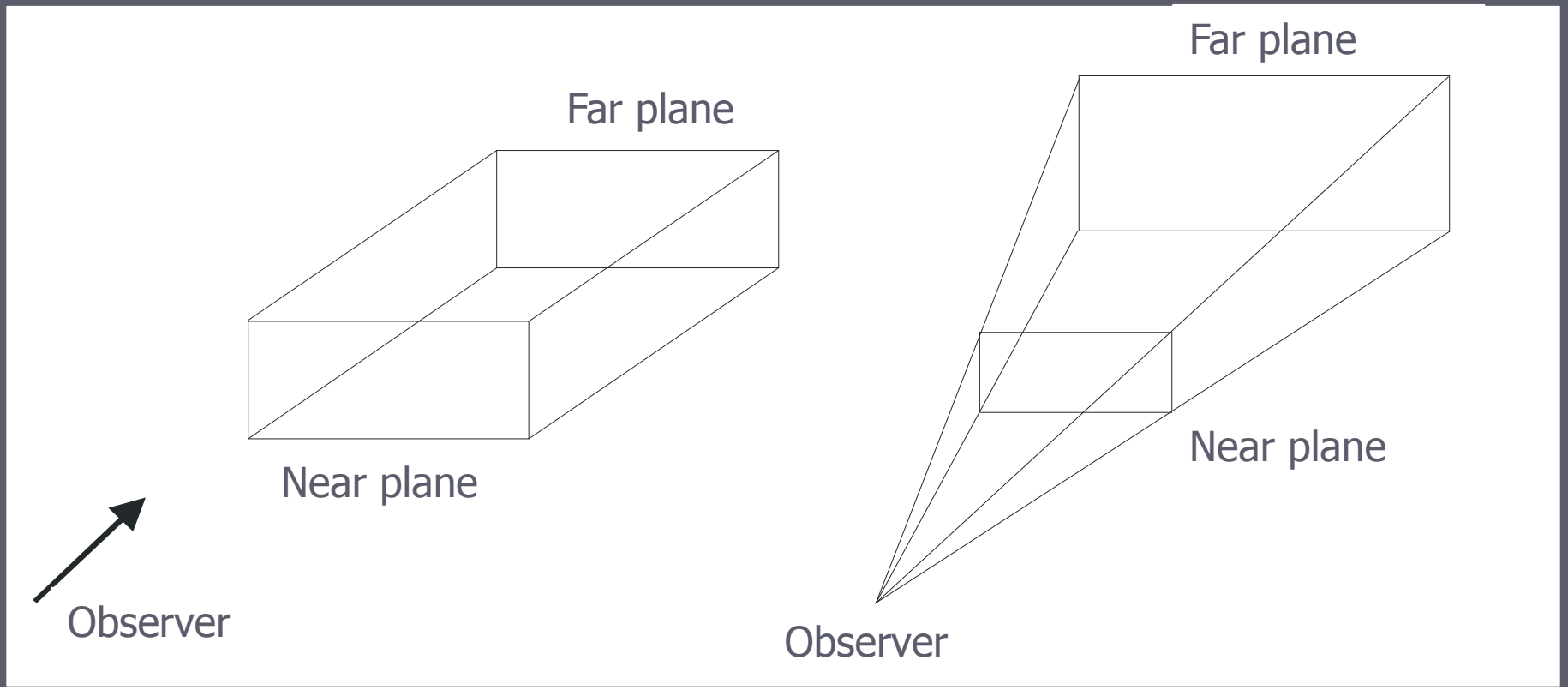


Three-points perspective

View pyramid:

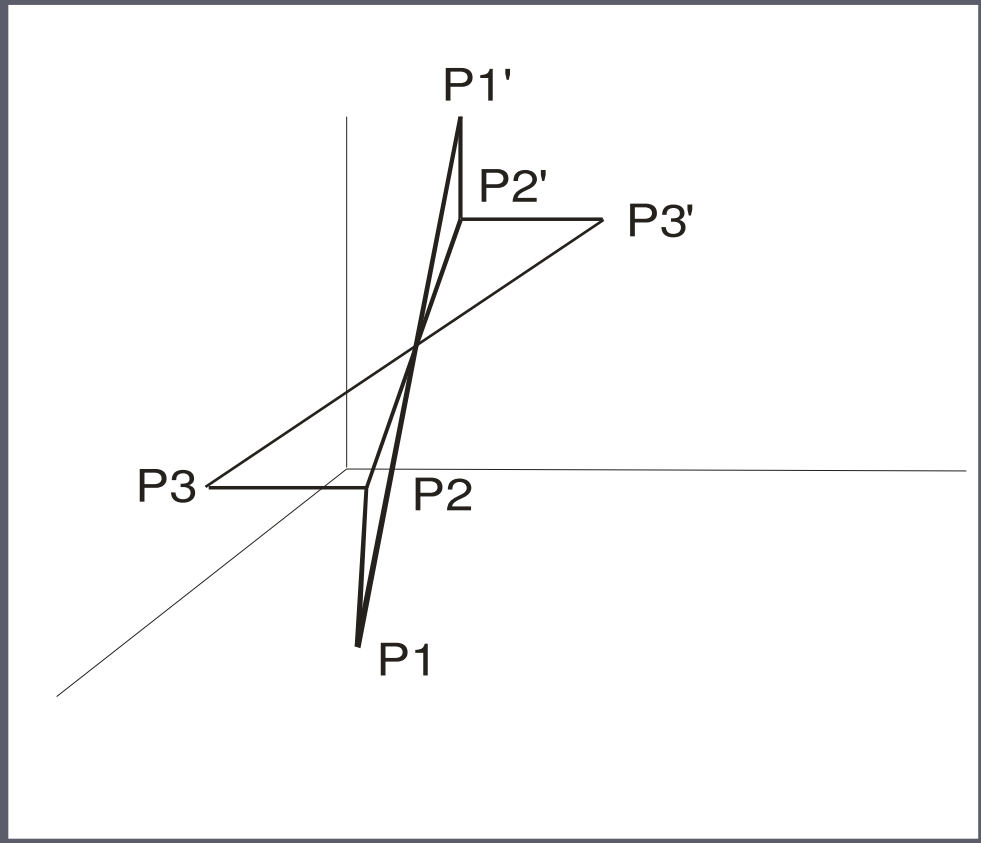
- When projecting, ignore objects out of the area of interests and prevent from perspective anomalies

=> view(ing) pyramid (viewing volume, viewing frustum) – outer objects are cut out



Perspective anomaly 1:

- View confusion– objects behind the projection centre are projected upside down and side-reversed



Perspective anomaly 2:

- Spoiled topology – the points on the plane going through the projection centre are projected to ∞, the line segment connecting the points behind and beyond the observer -> the line segment breaks and goes through “infinity”

The whole visualization pipeline:

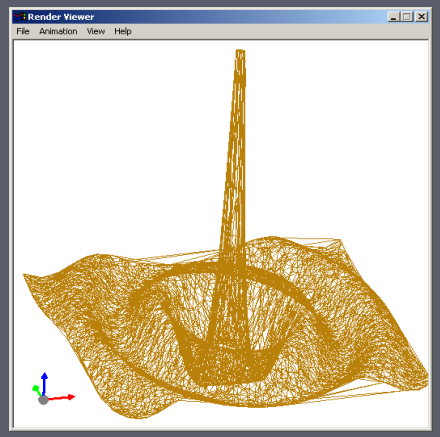
1. Transformation of an object from the given position into the position-to-be-projected
2. Cutting by the viewing volume
3. Planar projection, a scale according to the required size, respectively

1-3: viewing transformations

For step 1: choose the projection method and the projection plane placement – set the observer position (viewpoint) and the viewing direction (usually rectangular to the projection plane)

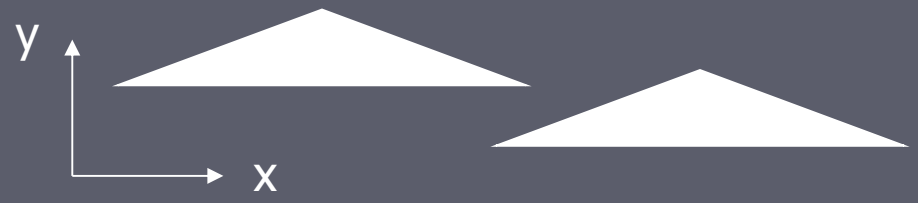
3. Hidden parts removal

- Elimination of invisible edges and faces – complex problem => many algorithms, also in hardware (z-buffer)
- The simplest: painter's algorithm (priority list): the order of drawing of faces according to their distance from the observer, the most distant first, then nearer

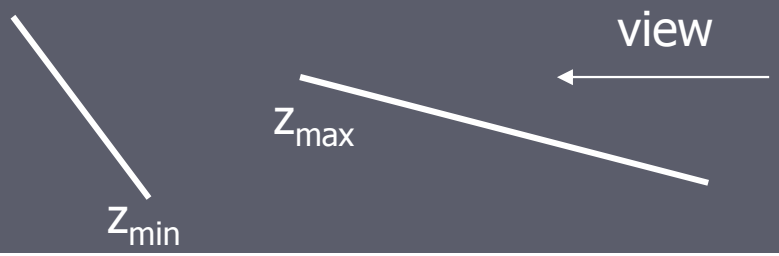


Painter's algorithm (simplified):

1. Sort the faces according to the maximum depth
2. Mark the farthest face as active, test its cover by other faces – if no cover, draw the face



3. If the face is (partly) covered, test it against the covering faces
   a) test z_{min} of one and z_{max} of the other face



b) if overlap in depth, other tests necessary

- For this simplified version, time is nearly O(n) to the number of faces, small errors

- Simple implementation, memory needed for all faces

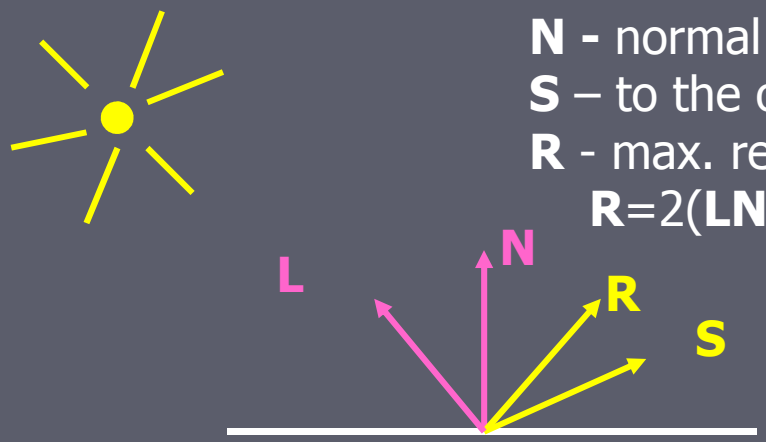
4. Lighting and shading

a) Lighting

- Various lighting models – various fidelity to the real-life light on a real-life surface, various computational complexity
- Light – 3 components - R, G, B
- Vectors – normalized

- The simplest model
  - Light ray hitting the surface partially disperse into all directions, partially is reflected as from a mirror (penetration into the object is not considered), only point light source
  - Reflection depends on the size and direction of the incoming light, observer position, surface properties

Omnidirectional reflection



N - normal
S – to the observer
R - max. reflection,
R=2(**LN**)**N**-L

Directional reflection

$I = I_a + I_d + I_s$ (3x for 3 components R,G,B)

where I_a – ambient intensity – background

I_d – diffusional intensity – omnidirectional reflection

I_s - specular intensity – mirror recflection

$$I_a = k_a * I_p$$

$$I_d = k_d * I_p * (\mathbf{L} * \mathbf{N})$$

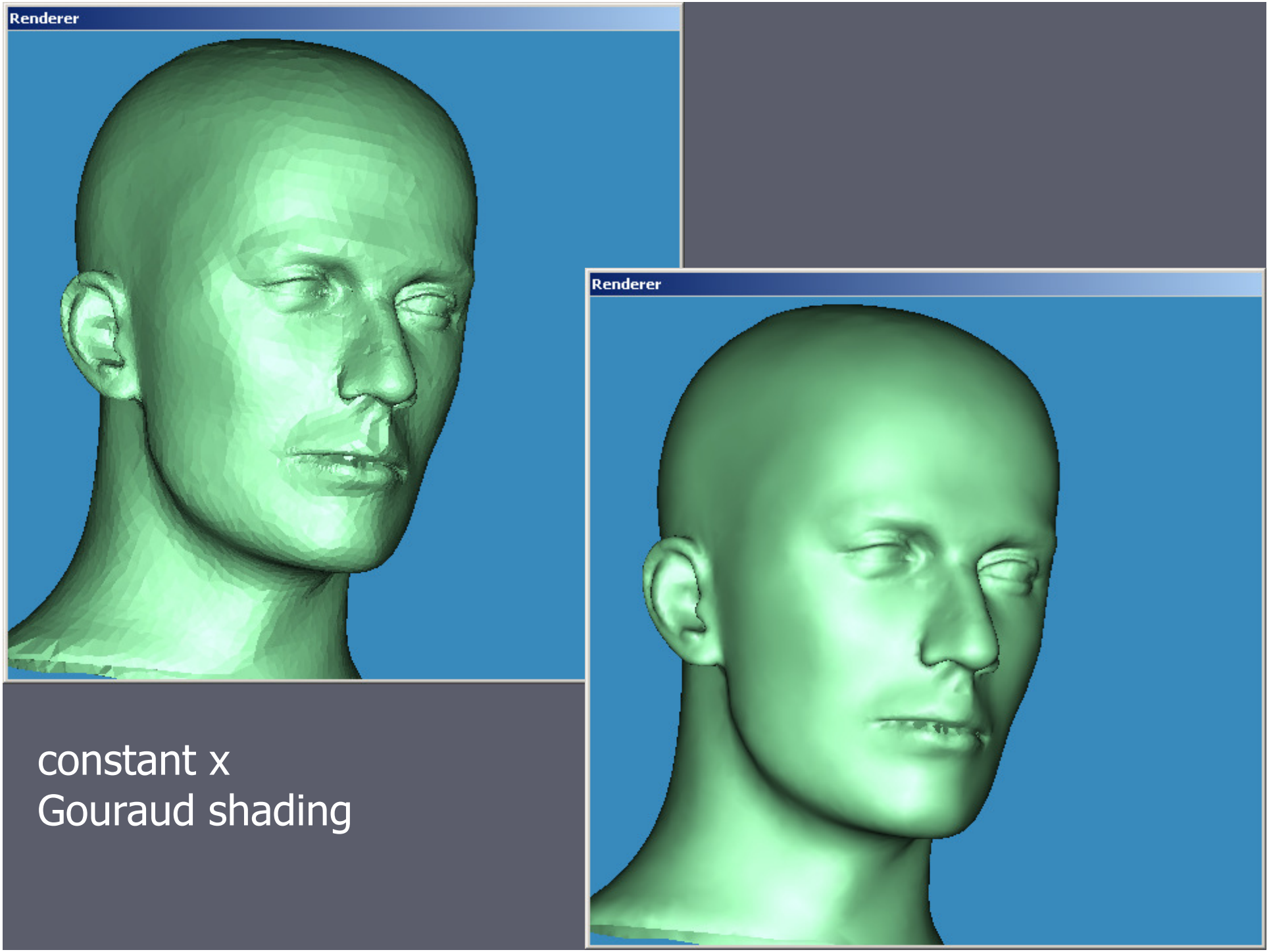
$$I_s = k_s * I_p * (\mathbf{R} * \mathbf{S})^n$$

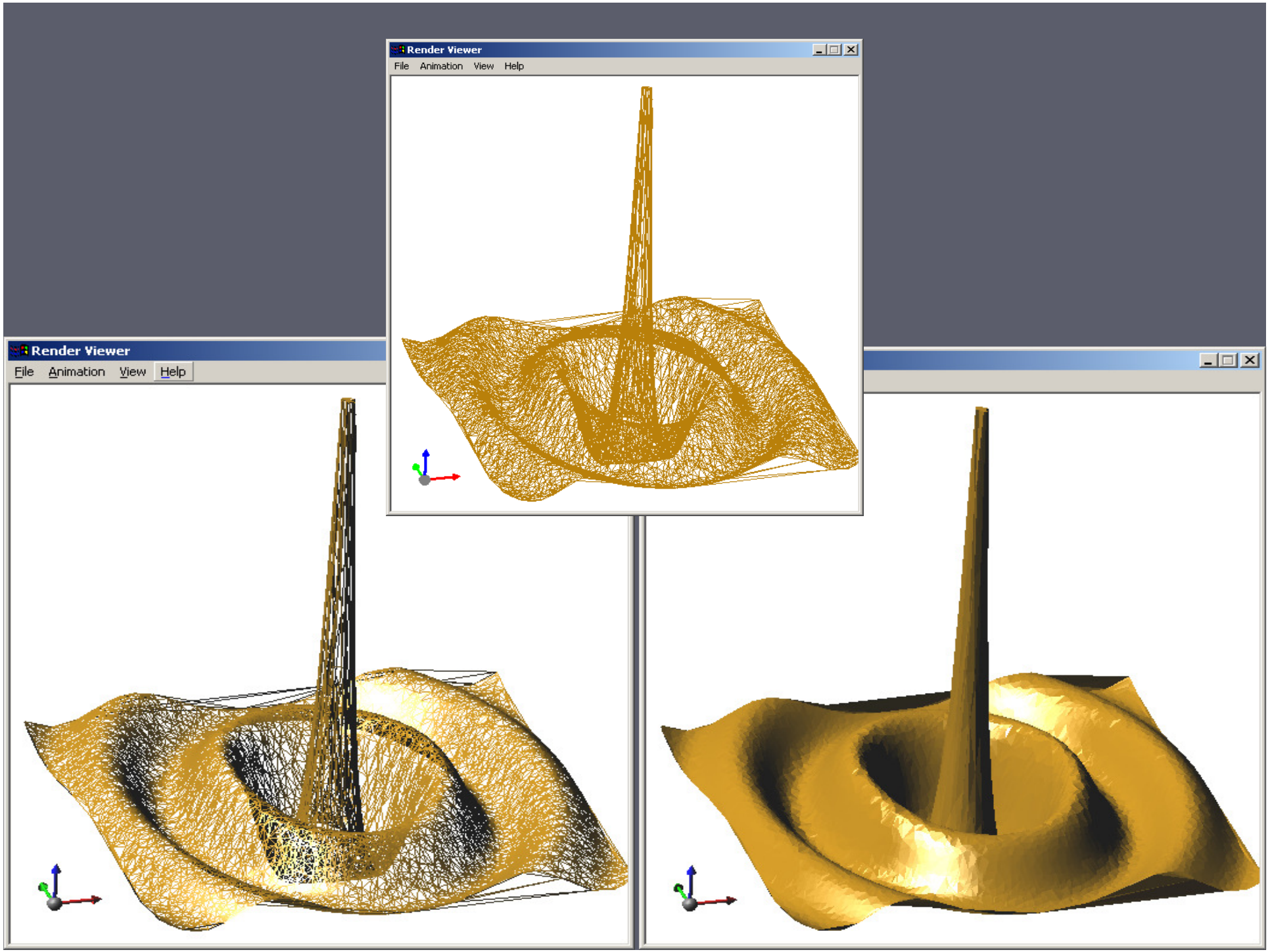
k_a, k_d, k_s – material coefficients (ambient, diffusional, specular), from <0;1>, n – surface characteristics, ≥0 – for highly shiny about 200, opaque ap to 10

- Sometimes Id and Is are divided by the distance from the observer - (d+k), where d - distance, k – const > 0
- Physically correct: d^2, here not used – too fast decrease
- If more point sources, sum over all Is, Id
- Empirical model, far from a correct physical model
- Simple, often used

b) Shading

- Drawing of colour objects by various colour shades
- The simplest: constant shading– one face, one colour
- Consumption: the light source in ∞, thus LN const. for the whole face, the observer in ∞, thus RS const. for the whole face
- For polyhedra OK, for objects which are by linear faces only approximated not suitable


Renderer
Renderer
constant x
Gouraud shading


Render Viewer
File Animation View Help
Render Viewer
File Animation View Help