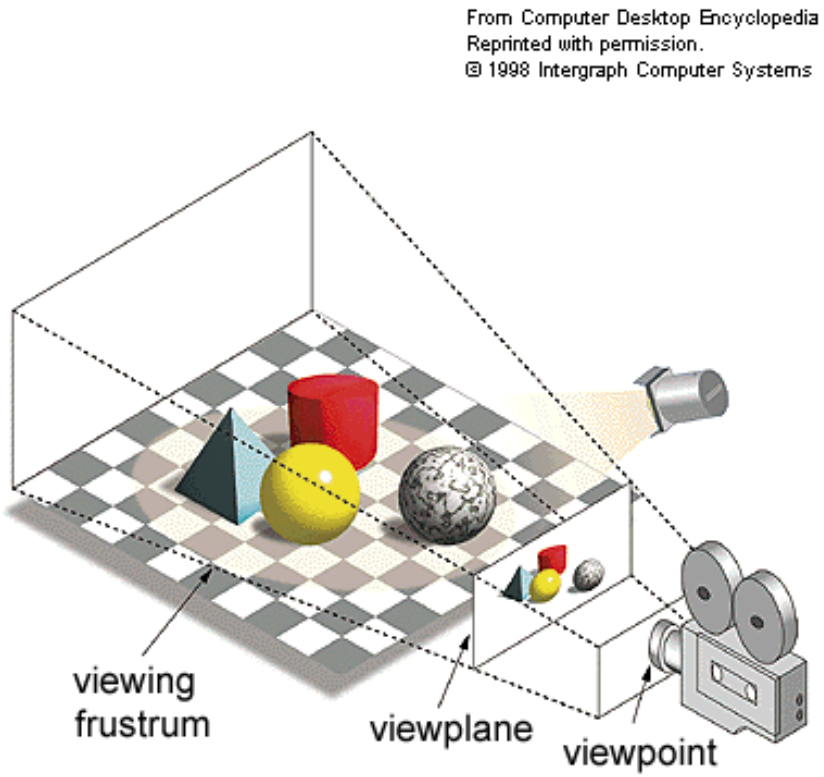
1. A SZÁMÍTÓGÉPES GRAFIKA TÁRGYA, ALKALMAZÁSAI

Számítógépes grafika vs. digitális képfeldolgozás

- Számítógépes grafika
  - Valós és képzeletbeli objektumok (pl. tárgyak képei, függvények) *szintézise* számítógépes modelljeikből (pl. pontok, élek, lapok)



Grafika példák

- Tapasztalat, hogy képek formájában az adatok gyorsabban és hatásosabban feldolgozhatók az ember számára

- Fejlődés:

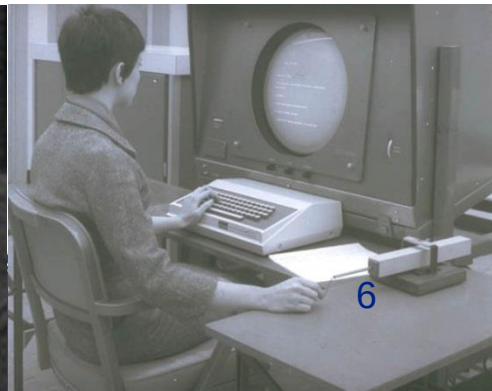
  fotózás → televízió → számítógépes grafika

Alkalmazási területek

- felhasználói programokhoz grafikus előtét
- üzlet, tudomány, technika (pl. dokumentum készítés)
- számítógéppel segített tervezés (CAD)
- szimuláció, animáció (pl. tudomány, szórakozás)
- művészet, kereskedelem
- folyamatirányítás
- térképészet

Történeti áttekintés

- Kezdetben:
  - képek megjelenítése teletype-on, nyomtatókon
- 1950:
  - MIT: számítógéppel vezérelt képernyő
  - SAGE légvédelmi rendszer (a programok képernyőről történő vezérlése fényceruzával)



Történeti áttekintés

- 1963:
  - A modern interaktív grafika megjelenése
  - I. Sutherland: Sketchpad
  - Adatstruktúrák szimbolikus struktúrák tárolására
  - Interaktív megjelenítés, választás, rajzolás

Történeti áttekintés

- 1964:
  - CAD – DAC-1 (IBM)
  - Autók tervezésére (General Motors)

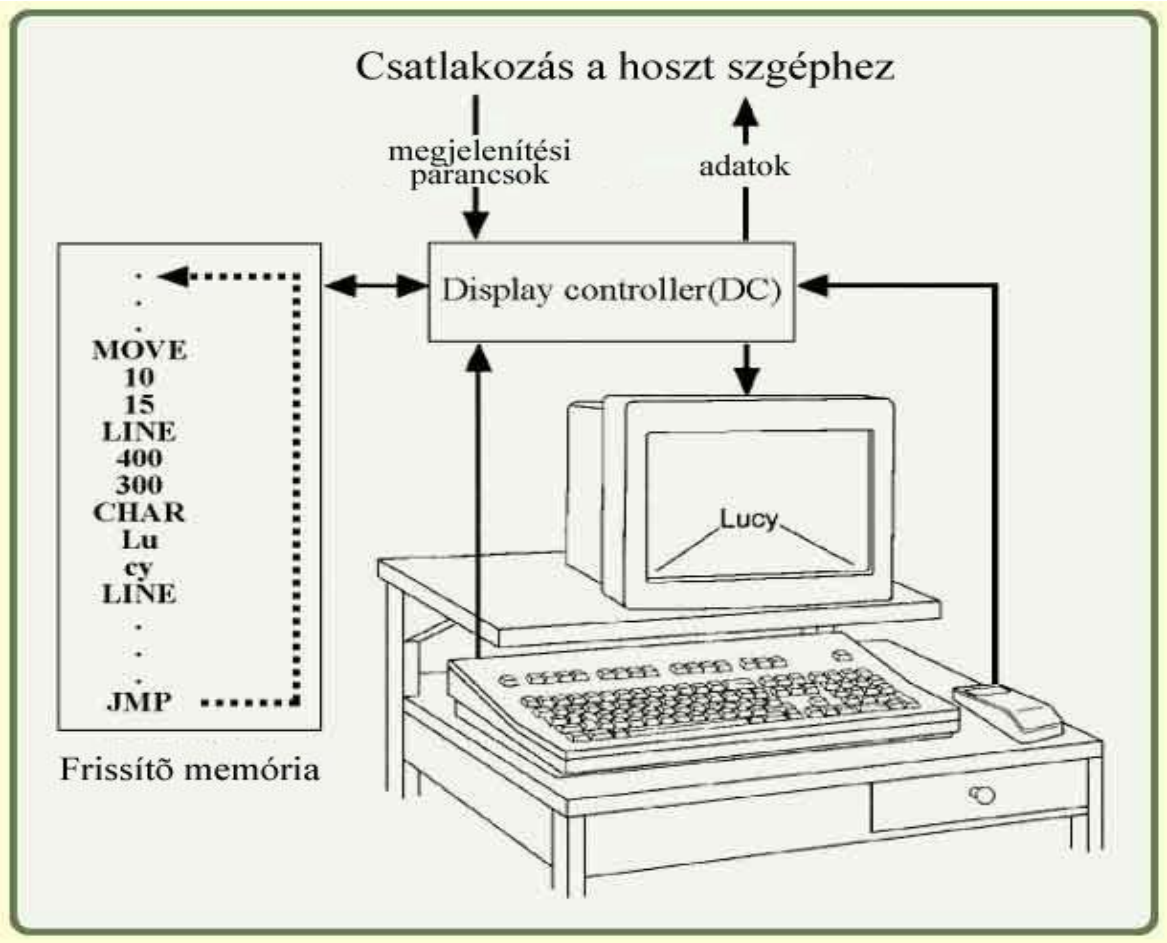
Történeti áttekintés

- Lassú fejlődés, mert
  - Drága a hardver
  - Drága számítógépes erőforrások (nagy adatbázis, interaktív manipuláció, intenzív adatfeldolgozás)
  - Nehéz volt nagy programokat írni
  - A szoftver nem volt hordozható

Történeti áttekintés

- 1960-as évek:
  - Jellemző output-eszköz az ún. vektor-képernyő (szakaszokat rajzol -tól -ig)
- Részei:
  - Képernyő processzor (DP) - mint I/O periféria kapcsolódik a központi egységhez
  - Képernyő tároló memória - a megjelenítéshez szükséges program és adat tárolására
  - Képernyő - katódsugárcső

Történeti áttekintés



Történeti áttekintés

utasítás → koordináták → elektromos jel

↑ ↑

képernyő processzor vektor generátor

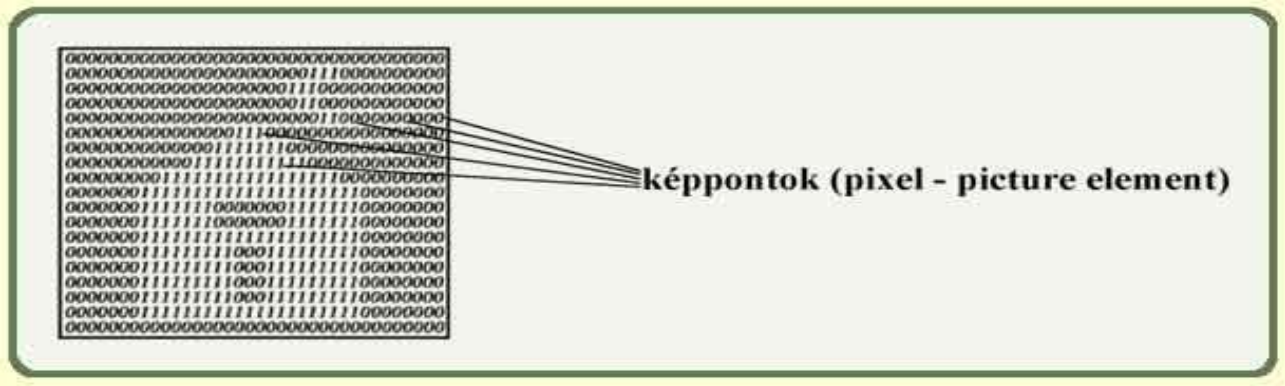
- 30 Hz-es frissítés (foszforeszkáló ernyő - nem villog annyira)
- 1960-as évek vége:
  - DVST (direct-view storage tube) - a látványt közvetlenül tároló cső olcsóbb

képernyő ↔ kisszámítógép

felszabadul a központi gép

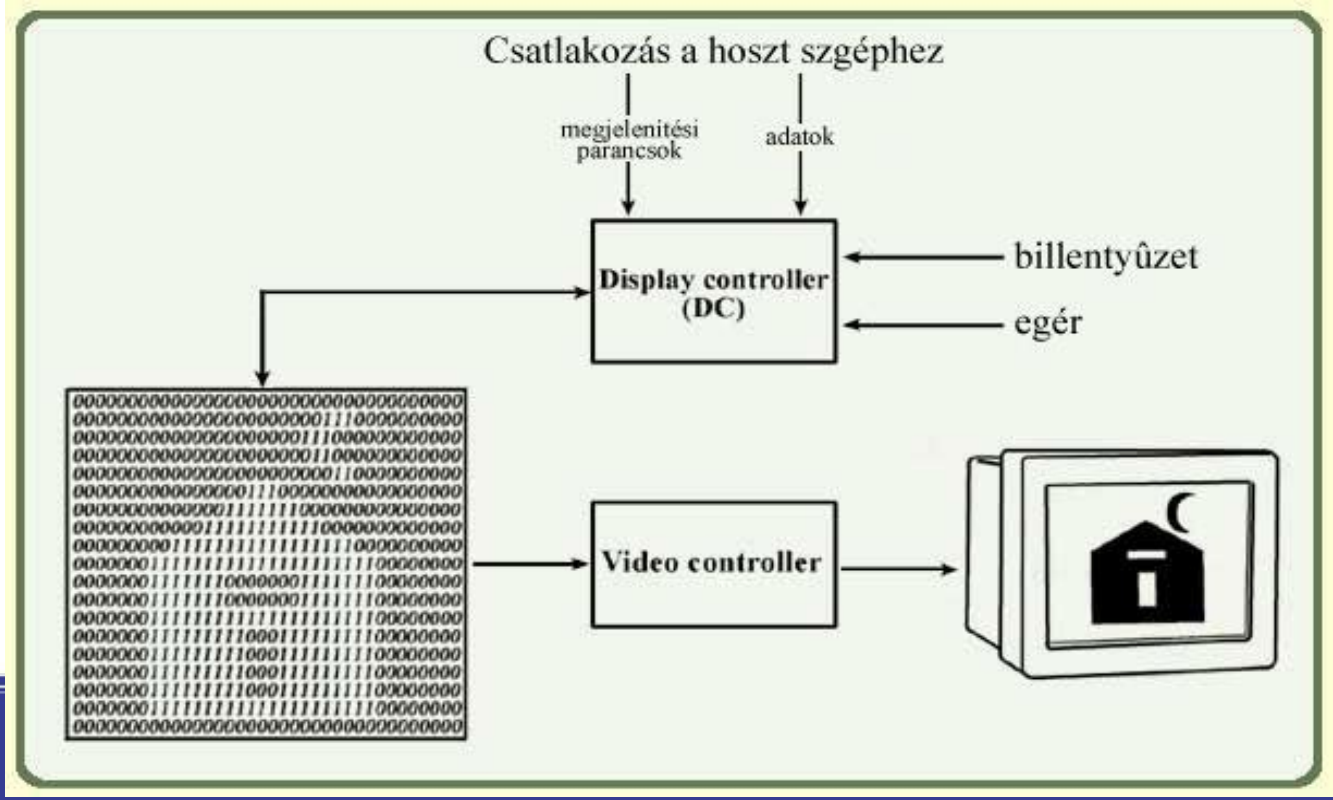
Történeti áttekintés

- 1968:
  - A hardver képes a skálát változtatni, a képet mozgatni, vetületeket előállítani valós időben
- 1970-es évek:
  - Jellemző output eszköz az un. raszter-képernyő (TV - technika), bit-térképes grafika
  - Bit-térkép (bitmap):
    képek reprezentálása bináris mátrixszal



Történeti áttekintés

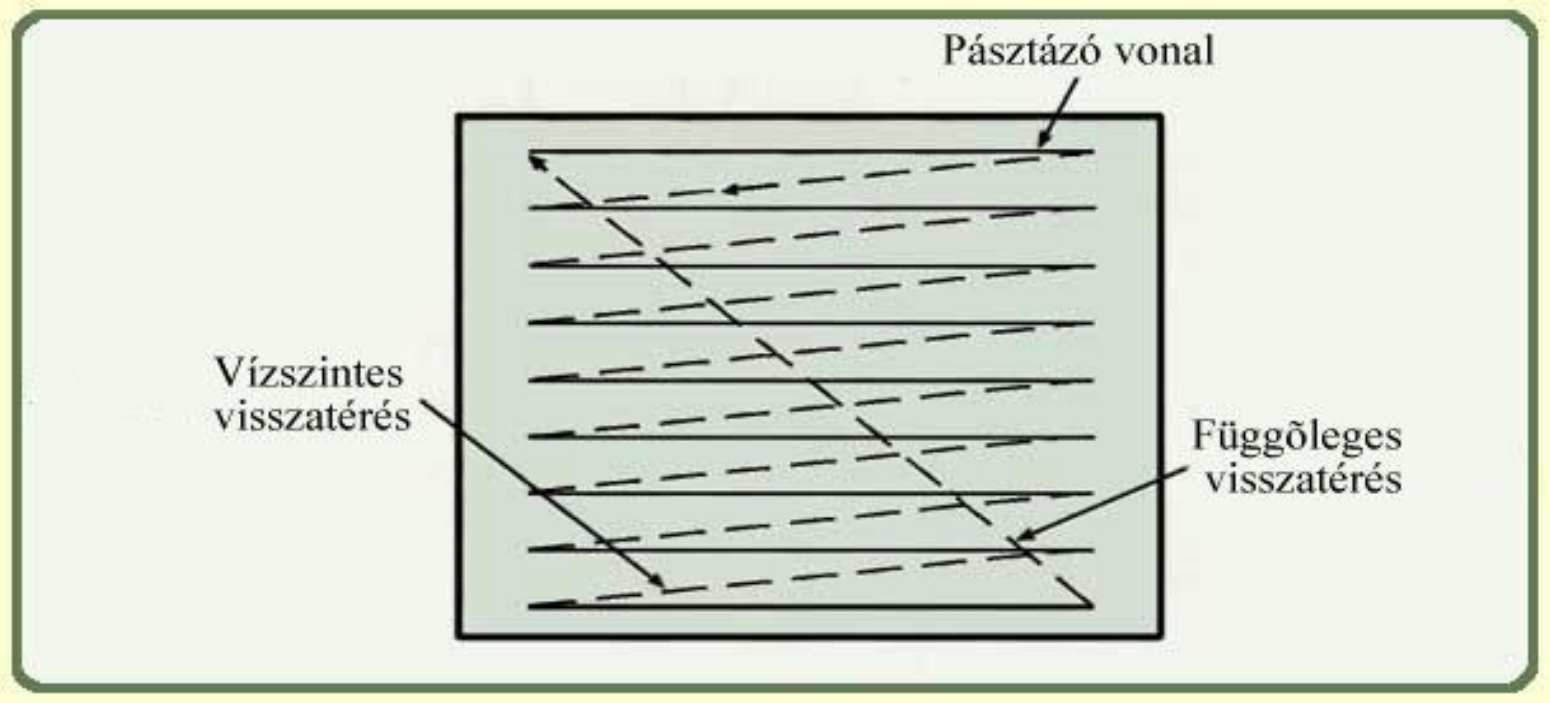
- A raszteres képernyők a grafikus primitíveket (pixel - képpont) az ún. frissítő tárolóban tartják



Történeti áttekintés

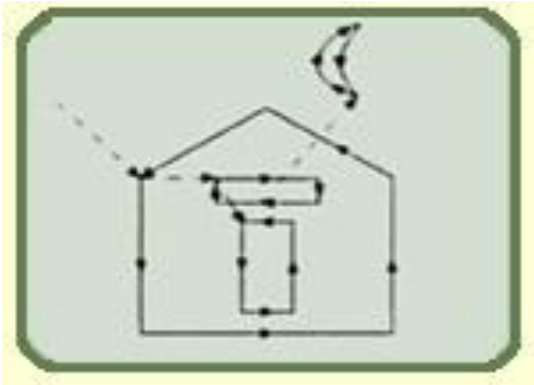
- Előnyei:
  - Olcsó logikájú processzor (soronként olvas)
  - A területek színekkel kitölthetők
  - Az ábra bonyolultsága nem befolyásolja a megjelenítés sebességét
- Hátrányai:
  - A grafikus elemeket (pl. vonal, poligon) át kell konvertálni (RIP - raster image processor)
  - A geometriai transzformációk számításigényesek

Történeti áttekintés



Megjelenítés raszteres képernyőn

Ideális vonalas rajz



Vektoros kép

Raszteres kép vonallal

Raszteres kép területkitöltéssel

Történeti áttekintés

- 1980-as évekig:
  - A számítógépes grafika szűk, speciális terület a drága hardver miatt
- Újdonságok:
  - Személyi számítógépek (Apple Macintosh, IBM PC)
  - Raszteres képernyők
  - Ablak technika (window manager)
- Eredmény:
  - Sok alkalmazás
  - Sok I/O eszköz (pl. egér, tábla, ...)
  - Kevesebbet használjuk a billentyűzetet (menük, ikonok, ...)

„Hordozható” szoftverek, szabványok

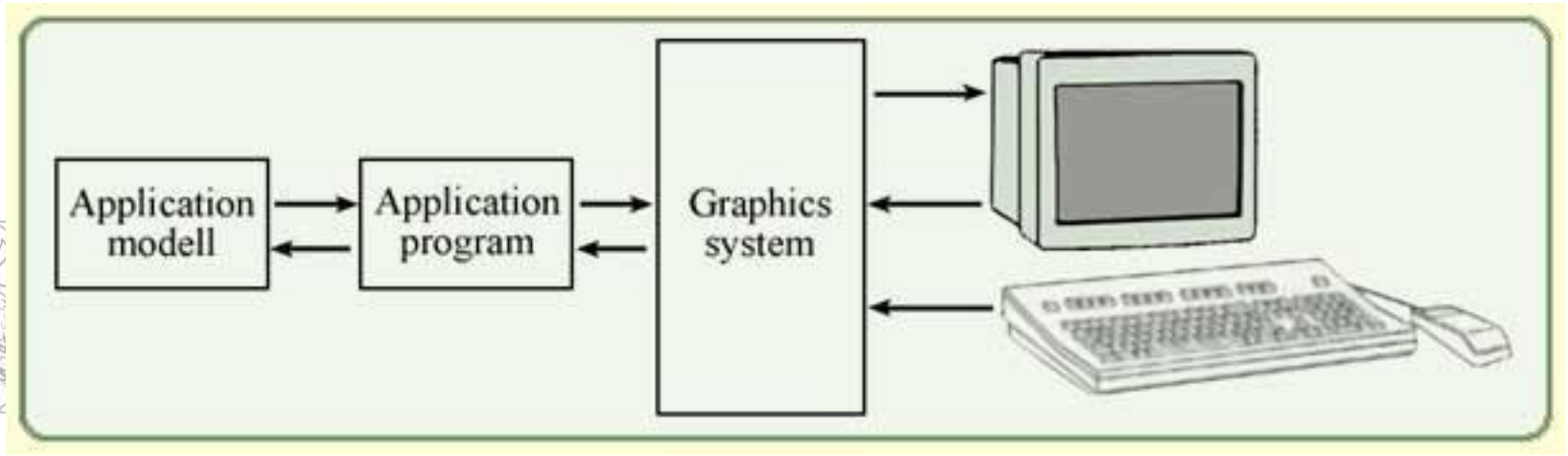
- Eszköz-függő → eszköz független
- Így lehet "hordozható" a felhasználói szoftver
- 1977:
  - 3D Core Graphics System
- 1985:
  - GKS (Graphical Kernel System) 2D

„Hordozható” szoftverek, szabványok

- 1988:
  - GKS - 3D
  - PHIGS (Programmer's Hierarchical Interactive Graphics System)
    - Logikailag kapcsolódó primitívek csoportosítása szegmensekbe
    - 3D primitívek egymásba ágyazott hierarchiája
    - Geometriai transzformációk
    - Képernyő automatikus frissítése, ha az adatbázis változik
- 1992:
  - OpenGL (SGI)
- 2011:
  - WebGL (Khronos Group)

Interaktív grafikai rendszerek

- Interaktivitás:
  - A felhasználó vezérli az objektumok kiválasztását, megjelenítését billentyűzetről, vagy egérrel...



Interaktív grafikai rendszerek

- Felhasználói modell (adatbázis):
  - Adatok, objektumok, kapcsolatok (adattömb, hálózati adatok listája, relációs adatbázis)
  - Primitívek (pontok, vonalak, felületek)
  - Attribútumok (vonal stílus, szín, textúra)

Az interaktivitás kezelése

- Tipikus az esemény-vezérelt programhurok:

```
init();
render();
while(true) {
  processInput();
  update();
  render();
}
```

A számítógépes grafika osztályozása

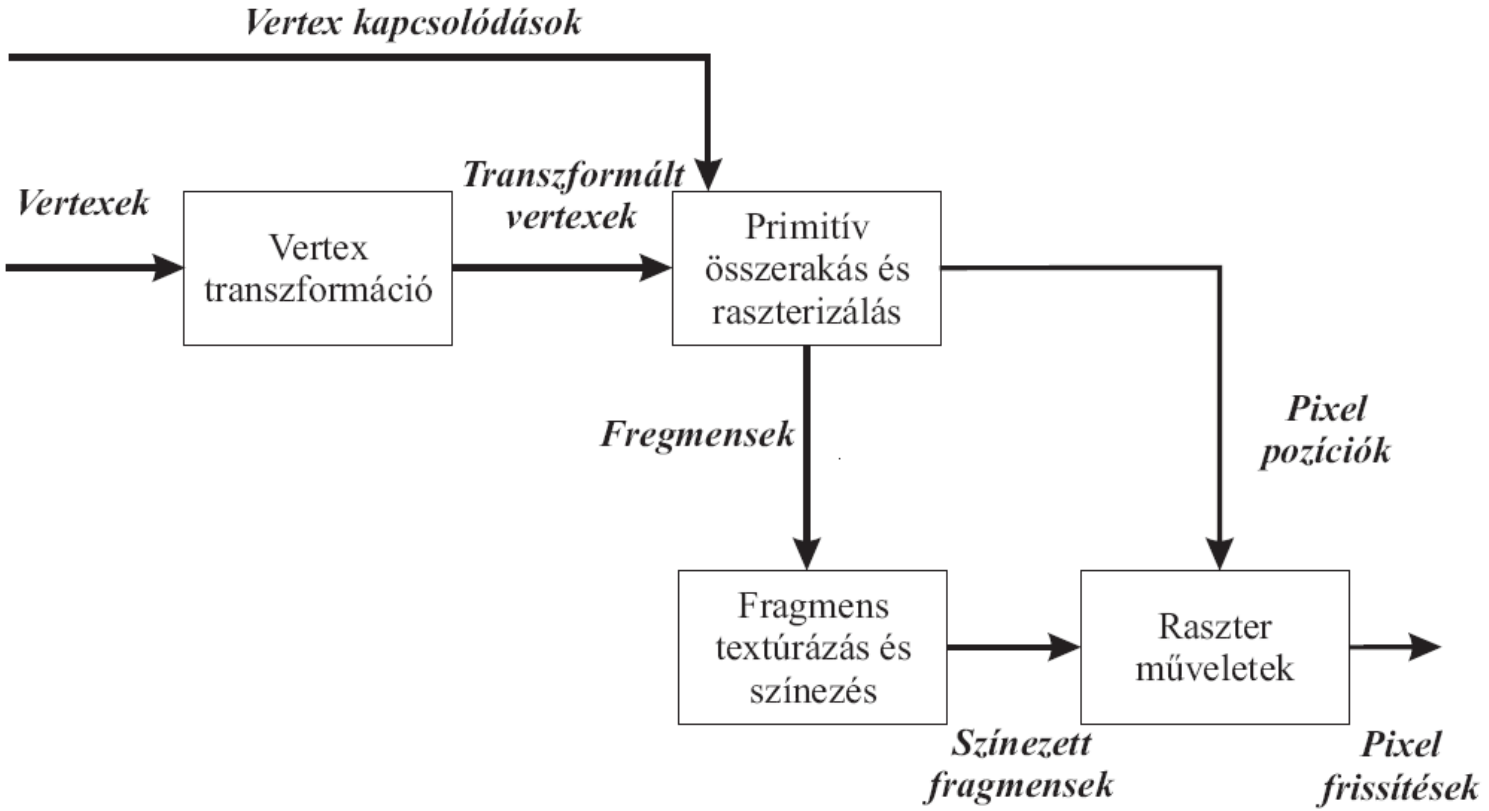
- Dimenzió szerint
  - 2-D
  - 3-D
- Képfajta szerint
  - vonalas
  - szürke
  - színes (árnyékolt)
- Kép szerepe szerint
  - Végtermék
  - Közbülső termék
- Interaktivitás szerint
  - Off-line rajzolás
  - Interaktív rajzolás (változó paraméterek)
  - Objektum előre meghatározása és körüljárása
  - Interaktív tervezés

2. A 3D MODELLEZÉS ALAPJAI, GRAFIKUS CSŐVEZETÉK

A 3D számítógépes grafikai modellezés „szereplői”

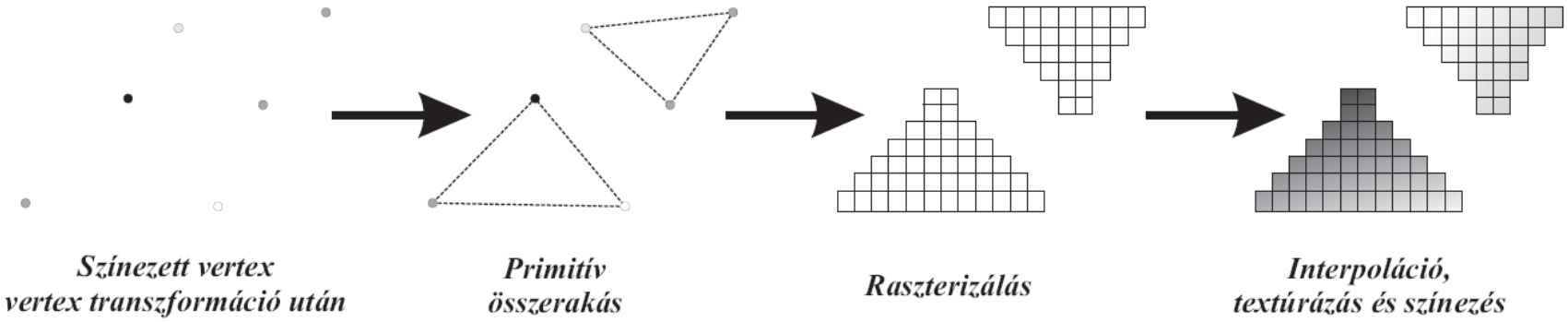
- Tárgy(ak) – (geometria, anyag)
- Fényforrás(ok) – (megvilágítás)
- Kamera – (vetítés)
- Képernyő – (raszterizálás)
- Felhasználó – (interakció)

A grafikus csővezeték



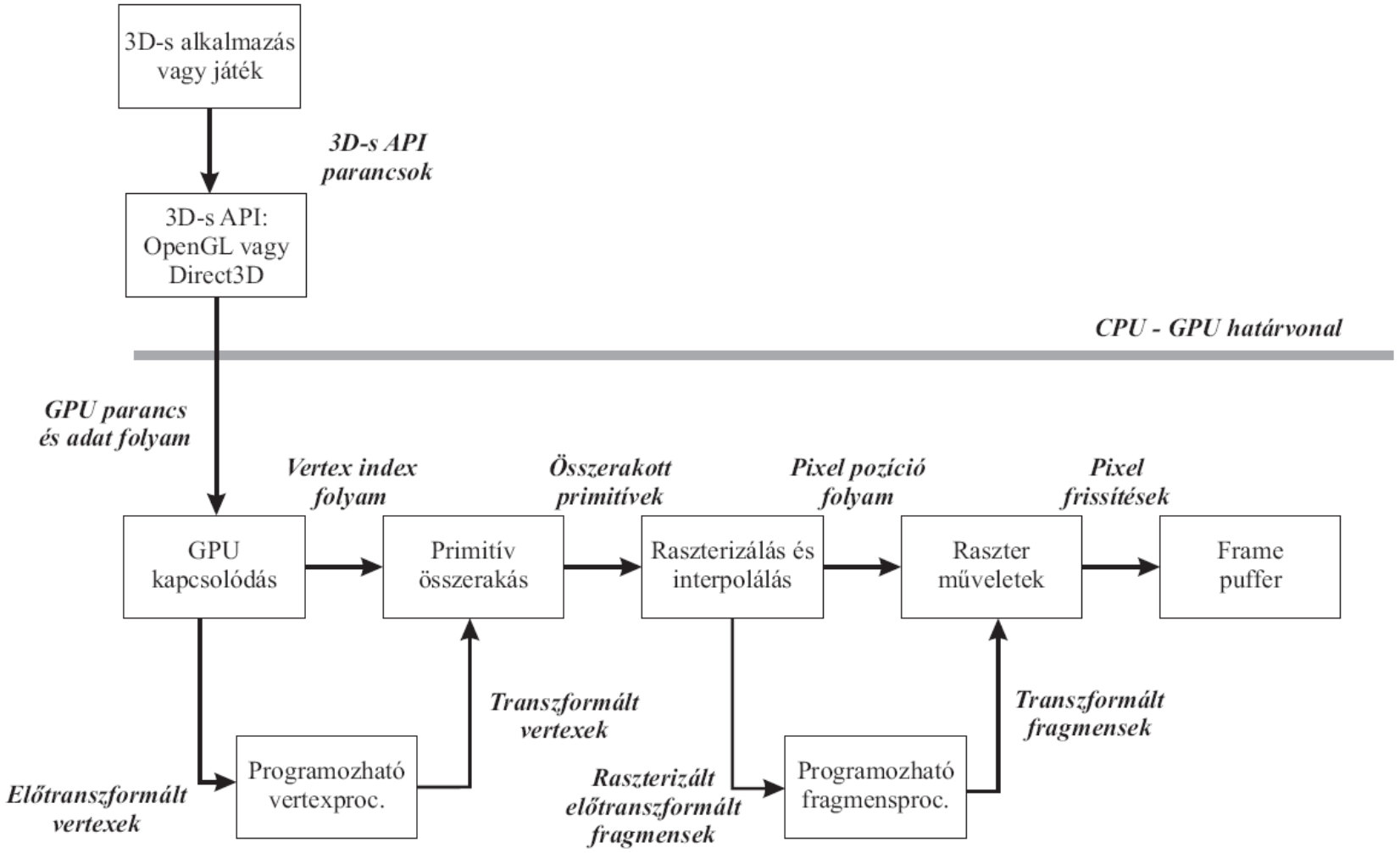
Ábra forrása: Nagy Antal „Fejlett grafikai algoritmusok” c. jegyzet

A grafikus csővezeték



Ábra forrása: Nagy Antal „Fejlett grafikai algoritmusok” c. jegyzet

A programozható grafikus csővezeték



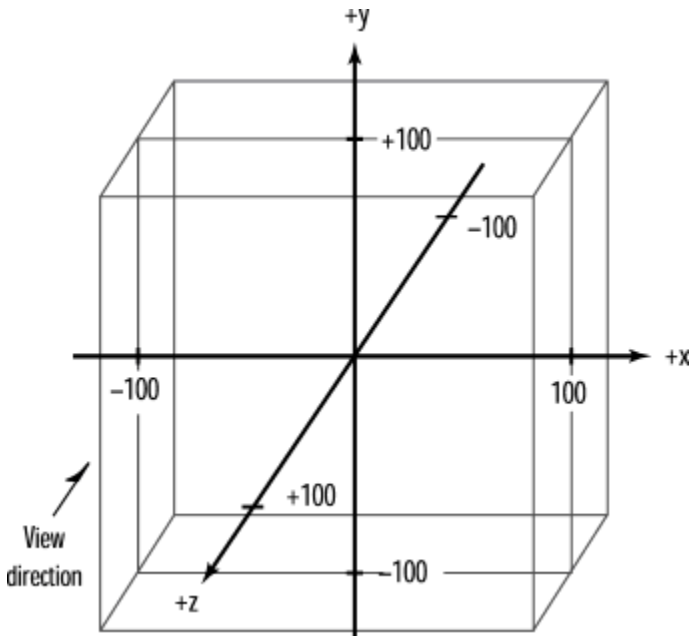
Ábra forrása: Nagy Antal „Fejlett grafikai algoritmusok” c. jegyzet

GRAFIKAI PRIMITÍVEK

(OpenGL illusztrációkkal)

Pontok, vonalak, poligonok

- Az objektumok és a látvány egyszerű, kicsi alakzatokból épülnek fel
- Primitívek
  - 1 vagy 2 dimenziós objektumok
- 3D-s festővászon

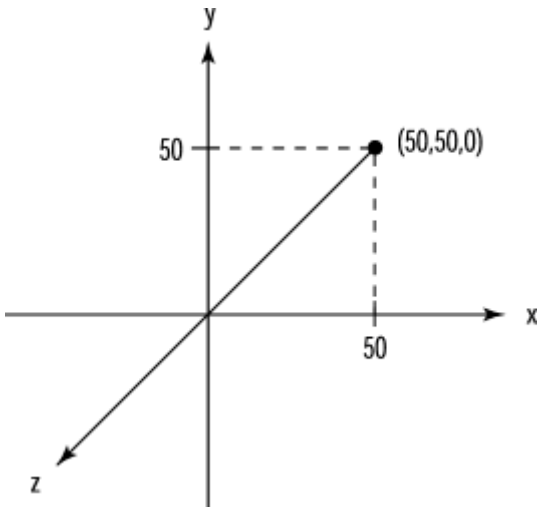


Vertex

- glVertex()
  - 2, 3 vagy 4 paraméter
  - Különböző típusok

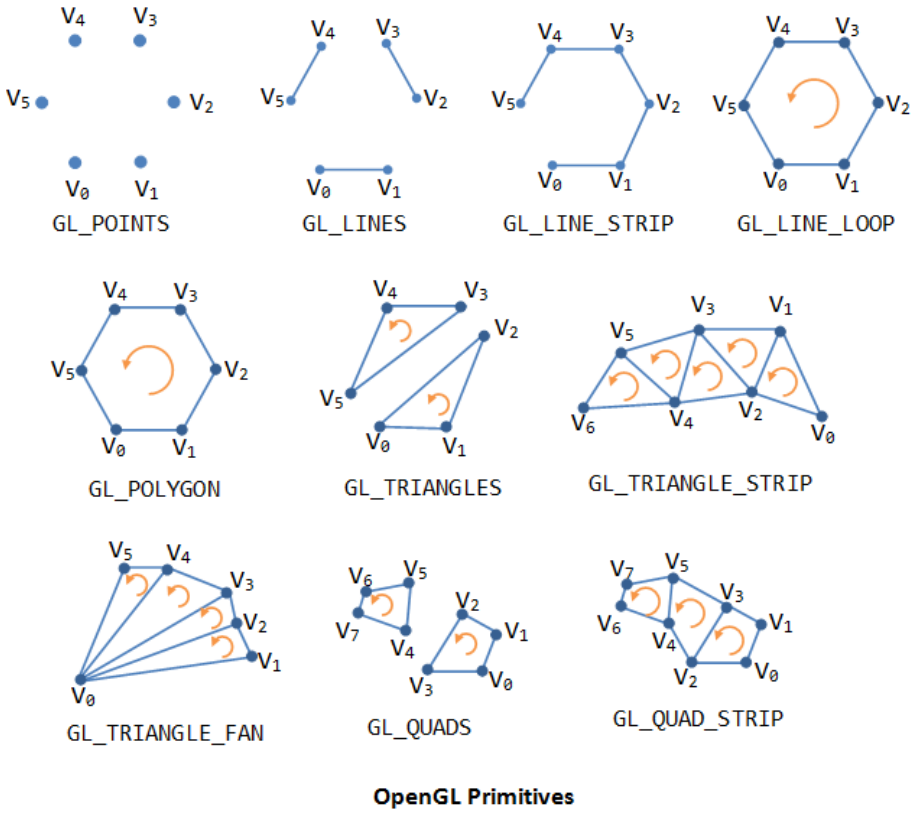
glVertex3f(50.0f, 50.0f, 0.0f)
glVertex2f(50.0f, 50.0f)
glVertex4f(50.0f, 50.0f, 0.0f, 1.0f)

nagyítás



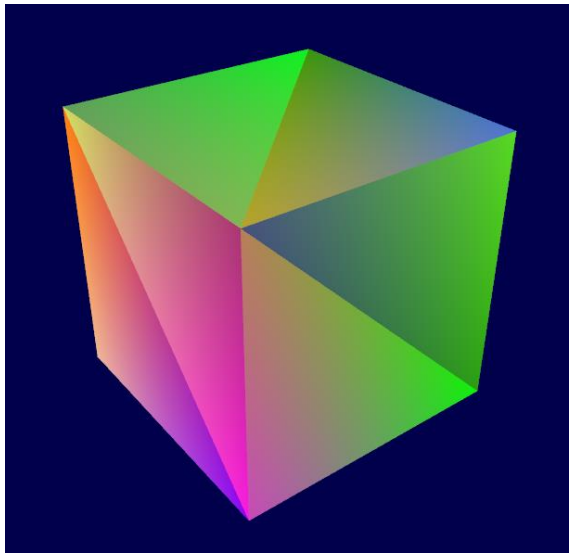
Grafikai primitívek

- Milyen primitívet kell létrehozni
  - Pl.: az OpenGL-ben:
    - GL_POINTS
    - GL_LINES
    - GL_LINE_STRIP
    - GL_LINE_LOOP
    - GL_TRIANGLES
    - GL_TRIANGLE_STRIP
    - GL_TRIANGLE_FAN
    - GL_QUADS
    - GL_QUAD_STRIP
    - GL_POLYGON



Grafikai primitívek

- A primitívek csúcspontjaihoz tulajdonságokat rendelünk.
  - glVertex
  - glColor
  - glIndex
  - glNormal
  - glTexCoord
  - glPointSize
  - Stb.
- A poligonoknak is vannak tulajdonságai:
  - glMaterial

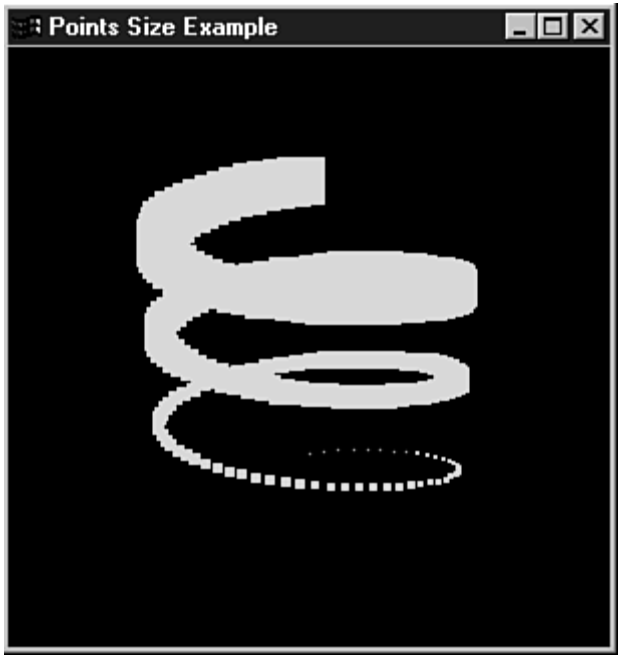


Pont méret

- void glPointSize(GLfloat size)

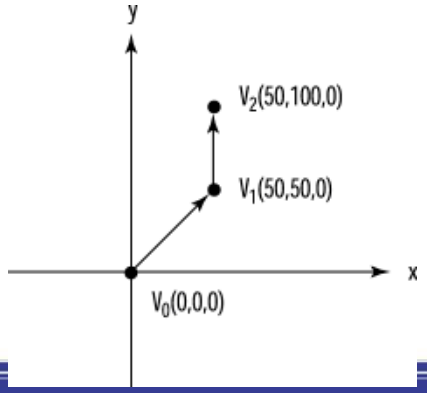
Pl.:

glPointsSize(4);

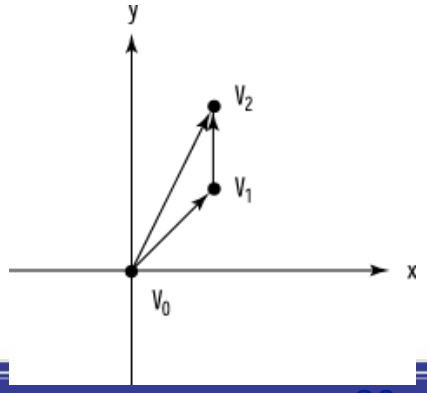


Vonal rajzolás

- glBegin(GL_LINES)
  glVertex3f(0.0f, 0.0f, 0.0f);
  glVertex3f(50.0f, 50.0f, 50.0f);
  glEnd();
  - Páratlan számú vertex esetén az utolsót nem veszi figyelembe
- glBegin(GL_LINE_STRIP) /* GL_LINE_LOOP */
  glVertex3f(0.0f, 0.0f, 0.0f); /*v1*/
  glVertex3f(50.0f, 50.0f, 50.0f); /*v2*/
  glVertex3f(50.0f, 100.0f, 0.0f); /*v3*/
  glEnd();



GL_LINE_STRIP

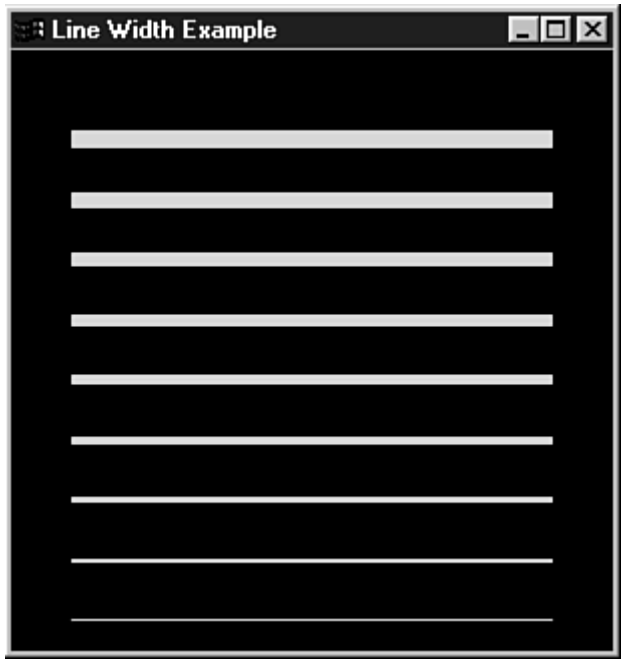


GL_LINE_LOOP

Vonal vastagság

- glLineWidth(GLfloat width)

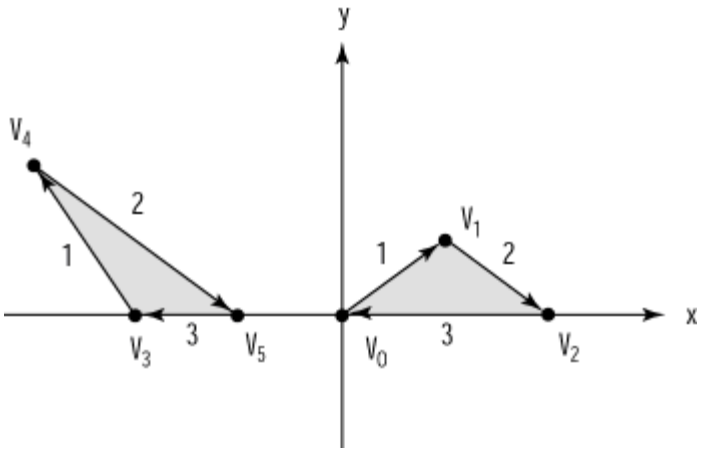
```
GLfloat sizes[2];
GLfloat step;
    glGetFloatv(GL_LINE_WIDTH
    _RANGE, sizes);
glGetFloatv(GL_LINE_WIDTH_G
    RANULARITY,
    &step);
```



Háromszög

- glBegin(GL_TRIANGLES);
  glVertex2f(0.0f, 0.0f); /*V0*/
  glVertex2f(25.0f, 25.0f); /*V1*/
  glVertex2f(50.0f, 0.0f); /*V2*/

  glVertex2f(-50.0f, 0.0f); /*V3*/
  glVertex2f(-75.0f, 50.0f); /*V4*/
  glVertex2f(-25.0f, 0.0f); /*V5*/
  glEnd():

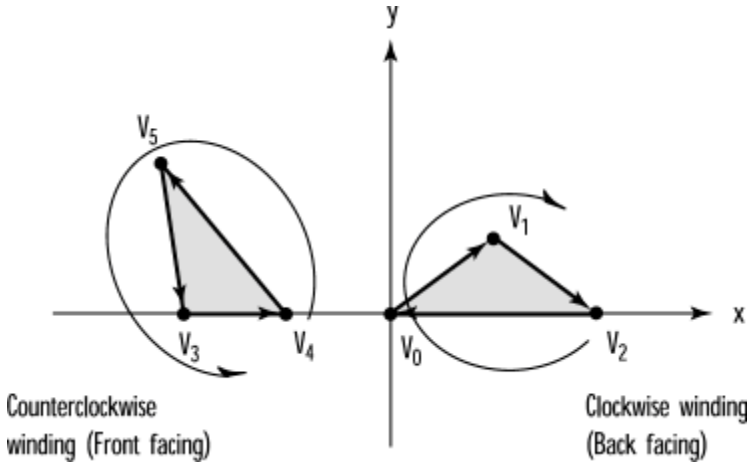


- Mindegyik poligon előállítható háromszögekből
- A videókártyák optimalizálva vannak a háromszögek kirajzolására

„Irányítottság”

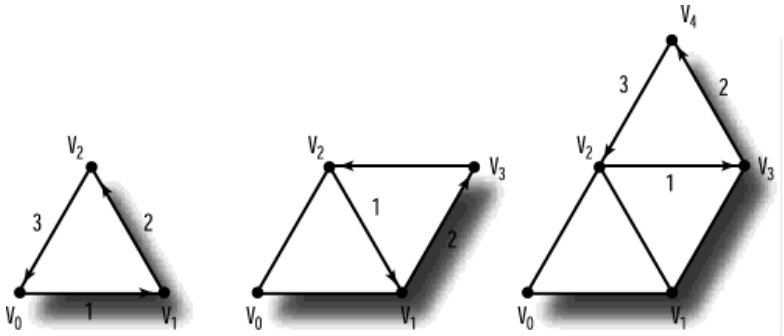
- OpenGL-ben alapértelmezésben órajárással ellentétes irányítottságú elölnézet
- Jobb-sodrású
- Fontos pl. a színezésnél, hátsó terület elrejtésénél

- glFrontFace(GL_CW)
  - GL_CW
  - GL_CCW



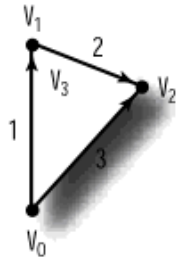
Háromszögsáv

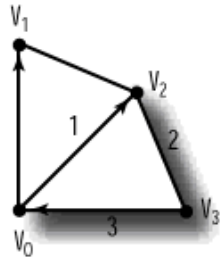
- GL_TRIANGLE_STRIP
  - Megőrzi az irányítottságot mindegyik háromszögre
- Előnyök
  - Az első három pont megadása után egy új háromszöghöz csak egy pontot kell megadni
  - Kevesebb vertex, gyorsabb végrehajtás a transzformációknál

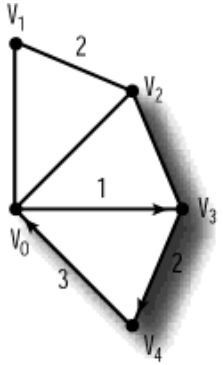


Háromszög-legyező

- TRIANGLE_FAN
  - Egy középpontnál kapcsolódó háromszögek csoportja
- Az első vertex a középpont
- Órajárással megegyező

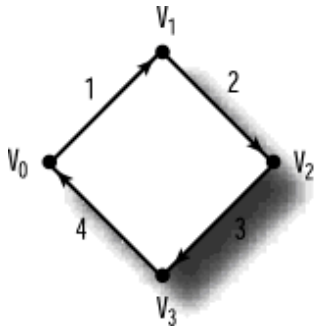




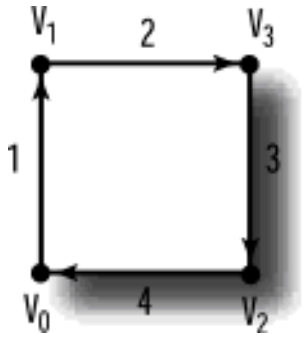


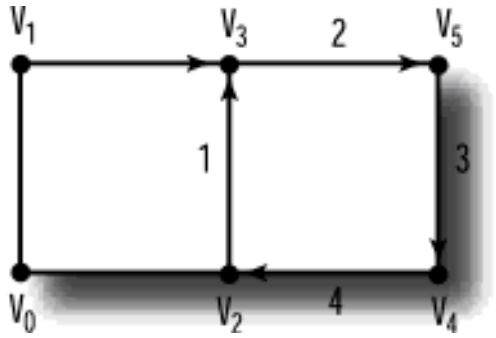
Egyéb primitívek

- GL_QUADS
  - Négy oldalú poligon

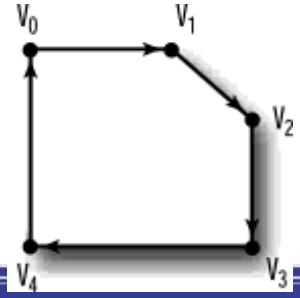


- GL_QUAD_STRIP
  - Négy oldalú csík





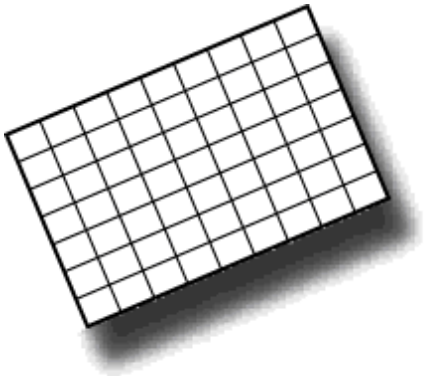
- GL_POLYGON
  - Általános poligon



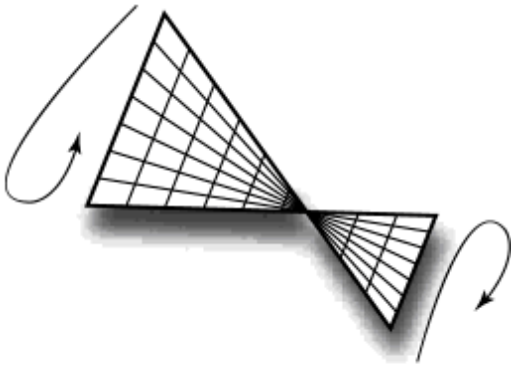
- glRectf(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2)
  - Téglalap

Poligonokra vonatkozó szabályok

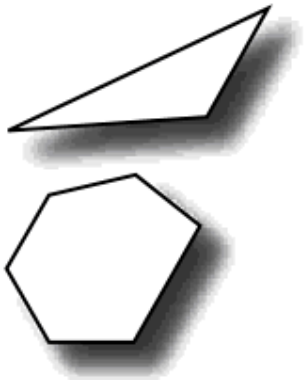
- Minden pont egy síkon van
- A poligonok élei nem metszhetik egymást
- Konvex-ek



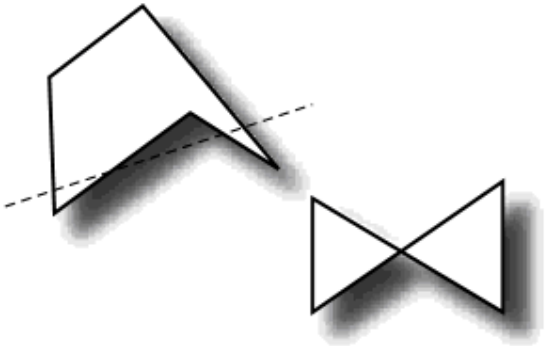
Planar polygon



Nonplanar polygon



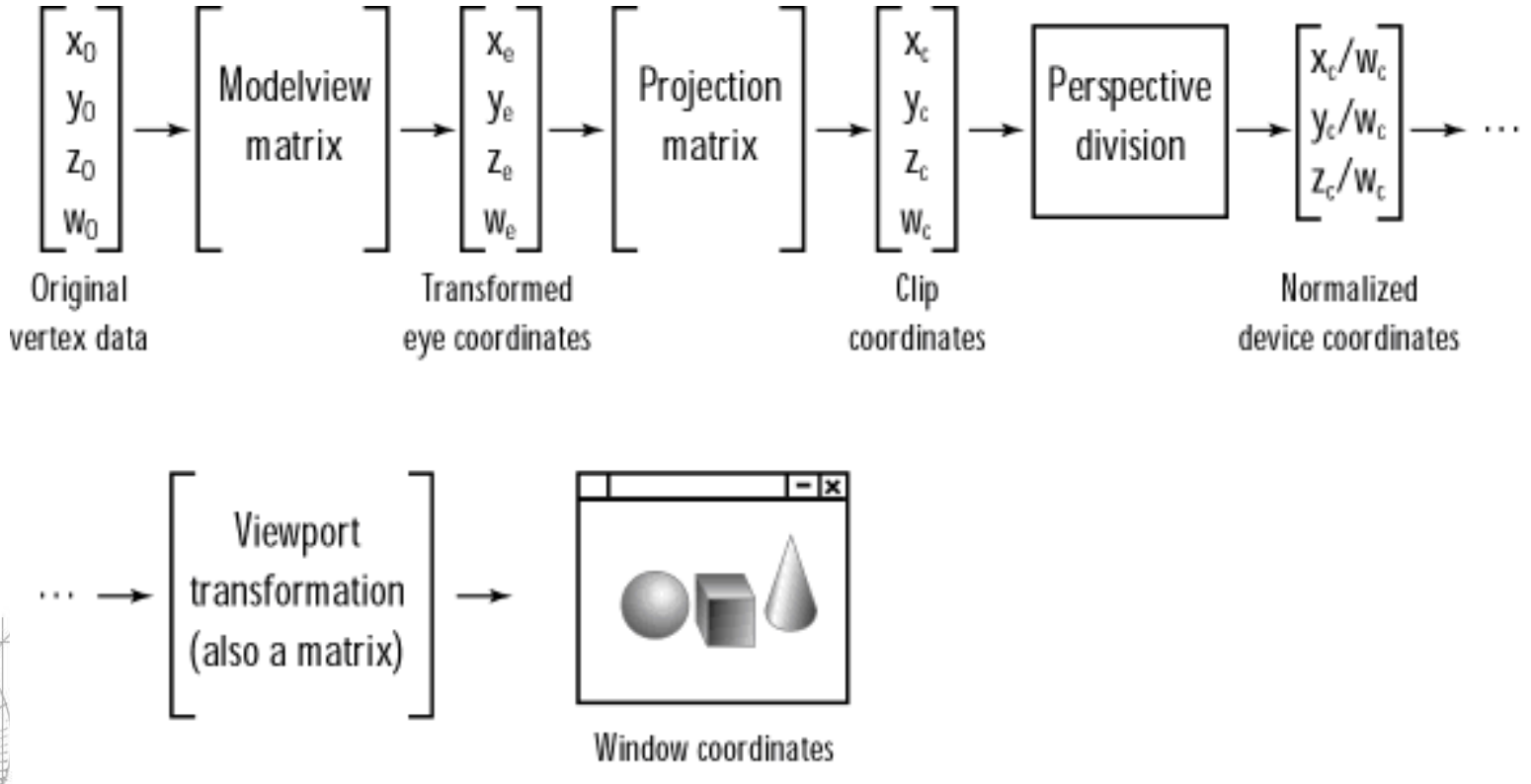
Valid polygons



Invalid polygons

3. GEOMETRIAI TRANSZFORMÁCIÓK

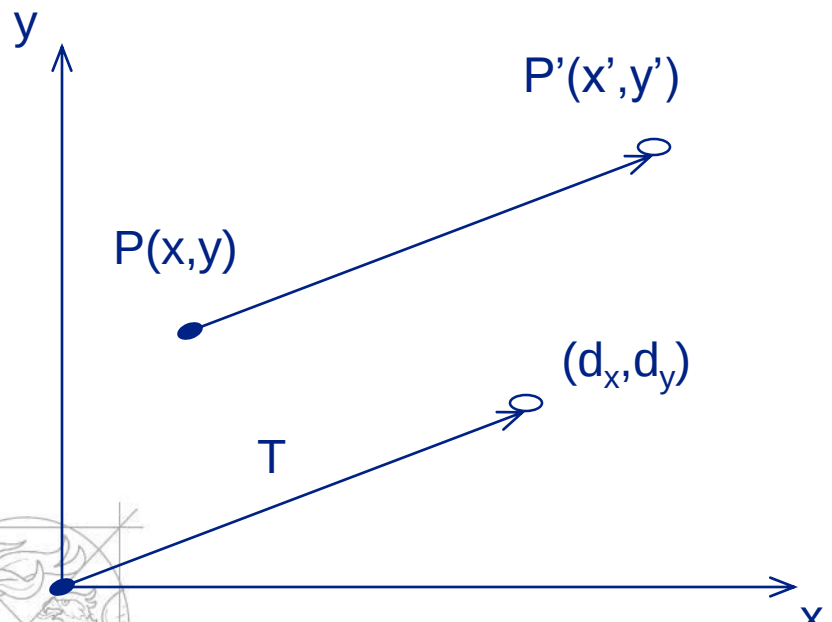
Transzformációs sor



- Számítógépes grafikában gyakran használt 2D és 3D transzformációk
  - eltolás
  - nagyítás, kicsinyítés (skálázás)
  - forgatás

Pont 2D eltolása

- A hosszak és a szögek változatlanok



$x' = x + d_x$

$y' = y + d_y$

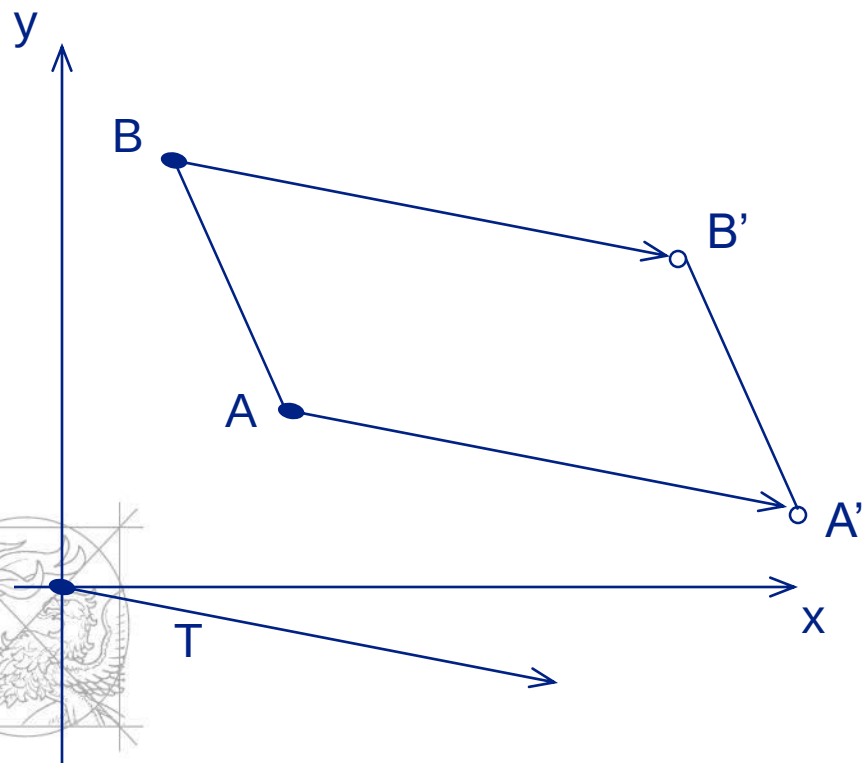
(oszlop-)vektorokkal:

$$P = \begin{pmatrix} x \\ y \end{pmatrix} \quad P' = \begin{pmatrix} x' \\ y' \end{pmatrix} \quad T = \begin{pmatrix} dx \\ dy \end{pmatrix}$$

$P' = P + T$

Szakasz 2D eltolása

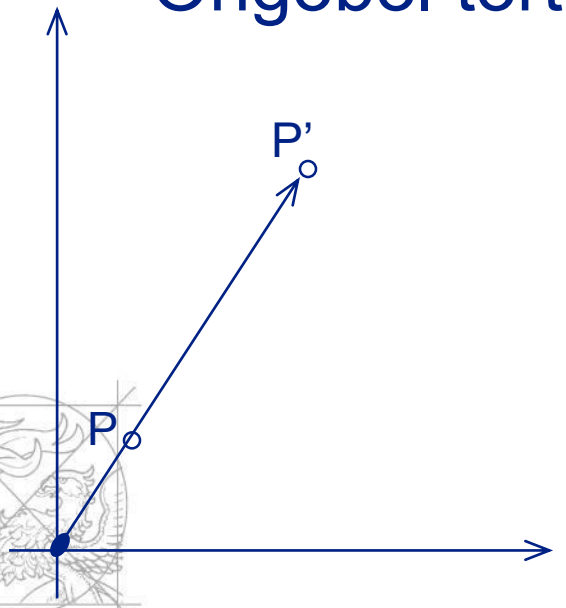
- Elegendő az új végpontokat számolni



A'=A+T
B'=B+T

2D nagyítás/kicsinyítés

- A szögek változatlanok
- Szokták a kettőt együtt skálázásnak említeni
- Origóból történő nagyítás



$x' = x \cdot s_x$

$y' = y \cdot s_y$

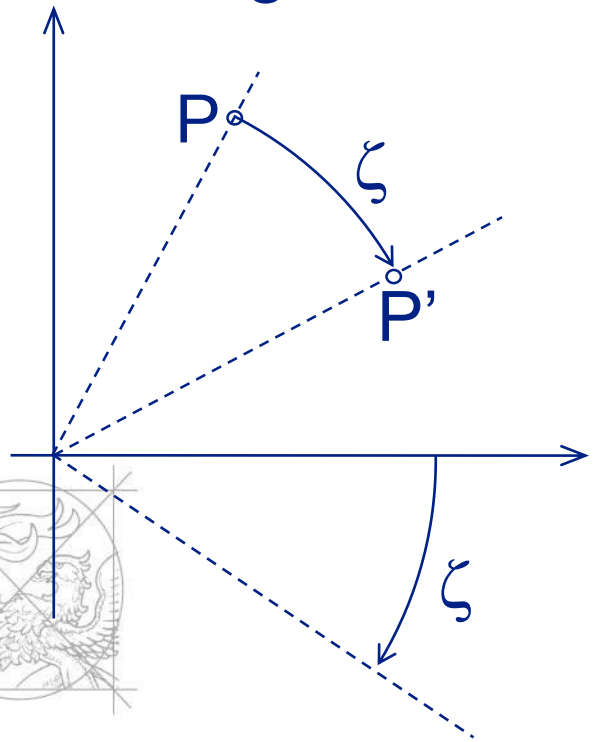
(oszlop-)vektorokkal:

$$P = \begin{pmatrix} x \\ y \end{pmatrix} \quad P' = \begin{pmatrix} x' \\ y' \end{pmatrix} \quad S = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$

$P' = S \cdot P$

2D forgatás

- A hosszak és a szögek változatlanok
- Origó körüli forgatás



$$x' = x \cdot \cos\zeta - y \cdot \sin\zeta$$
$$y' = x \cdot \sin\zeta + y \cdot \cos\zeta$$

$$P = \begin{pmatrix} x \\ y \end{pmatrix} \quad P' = \begin{pmatrix} x' \\ y' \end{pmatrix} \quad R = \begin{pmatrix} \cos\zeta & -\sin\zeta \\ \sin\zeta & \cos\zeta \end{pmatrix}$$

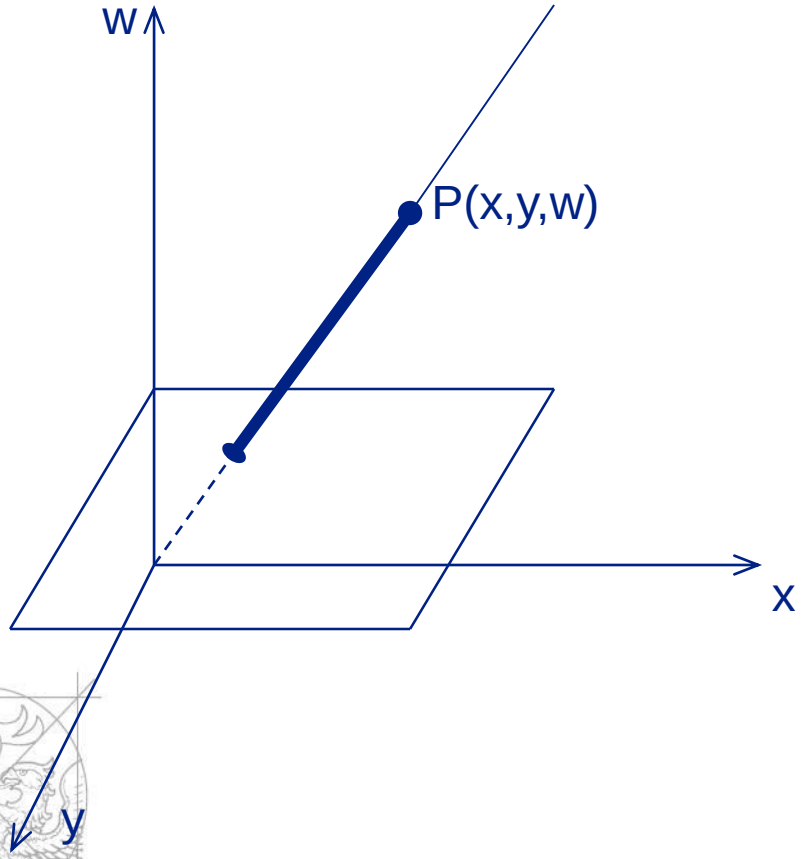
$$P' = R \cdot P$$

Homogén koordináták

- (x, y) jelölése homogén koordinátákkal: (x, y, w)
- Egyenlőség: (x,y,w) = (x',y',w') , ha van olyan α, hogy x'= α·x, y'= α·y, w'= α·w
- pl: (2, 3, 6) = (4, 6, 12)

- Egy ponthoz végtelen sok (x,y,w) tartozik
- Ha w≠0, akkor (x,y) szokásos jelölése (x/w,y/w,1)
- Ha w=0, akkor (x,y,w) végtelen távoli pont
- (0,0,0) nem megengedett!

Kapcsolat 2D és 3D közt



- (t·x, t·y, t·w)
- egyenes a 3D térben
- végtelen távoli pontok nincsenek a síkon

$$\left(\frac{x}{w}, \frac{y}{w}, 1\right)$$

P vetülete a w=1 síkon

2D eltolás

$P' = T(d_x, d_y)\, P$

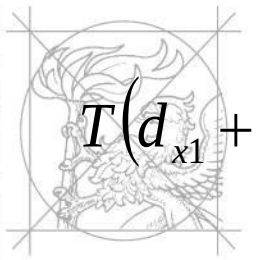
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Ismételt eltolások (kompozíció):

$P \xrightarrow{T(d_{x1}, d_{y1})} P' \xrightarrow{T(d_{x2}, d_{y2})} P''$

$P' = T(d_{x1}, d_{y1})\, P$,

$P'' = T(d_{x2}, d_{y2})\, P' = T(d_{x2}, d_{y2})\, (T(d_{x1}, d_{y1})\, P)$



$$T(d_{x1}+d_{x2}, d_{y1}+d_{y2}) = \begin{pmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_{x1}+d_{x2} \\ 0 & 1 & d_{y1}+d_{y2} \\ 0 & 0 & 1 \end{pmatrix}$$

2D skálázás

$P' = S(s_x, s_y)\, P$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Ismételt skálázások (kompozíció):

$P \xrightarrow{S(s_{x1}, s_{y1})} P' \xrightarrow{S(s_{x2}, s_{y2})} P''$

$P' = S(s_{x1}, s_{y1})\, P$,

$P'' = S(s_{x2}, s_{y2})\, P' = S(s_{x2}, s_{y2})\, (S(s_{x1}, s_{y1})\, P)$

$$S(s_{x1}s_{x2}, s_{y1}s_{y2}) = \begin{pmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_{x2}s_{x1} & 0 & 0 \\ 0 & s_{y2}s_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2D forgatás

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

P'=R(ζ) P (R(ζ) ortogonális)

Ismételt forgatások:

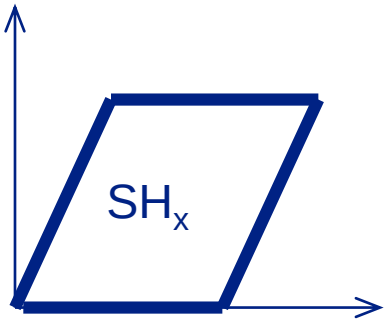
P' = R(ζ_1) P,

P''= R(ζ_2) P' = R(ζ_2) (R(ζ_1) P) = R(ζ_1+ ζ_2) P

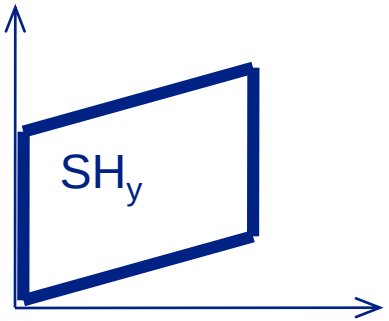
2D nyírás

- A hosszak és a szögek változhatnak
- Párhuzamos egyenesek képe párhuzamos

$$SH_x = \begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad SH_y = \begin{pmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + ay \\ y \\ 1 \end{pmatrix}$$



Affin transzformációk

- Eltolások, skálázások, forgatások és nyírások tetszőleges számú és sorrendű egymás utáni alkalmazásával kapott transzformáció

Általános kompozíció mátrix

- Skálázás/forgatás utáni eltolás
- r: skálázás/forgatás, t: eltolás

$$M = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & 0 \\ r_{21} & r_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

- Az M alakú mátrixokat kompozíció mátrixoknak nevezzük

2D transzformációk kompozíciója 1. példa

Forgatás ζ-val egy tetszőleges P(x,y) pont körül.

a) eltolás P-ből O-ba T(-x,-y)

b) forgatás az origó körül R(ζ)

c) eltolás O-ból P-be T(x,y)

$$\begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\zeta & -\sin\zeta & 0 \\ \sin\zeta & \cos\zeta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos\zeta & -\sin\zeta & x(1-\cos\zeta)+y\sin\zeta \\ \sin\zeta & \cos\zeta & y(1-\cos\zeta)-x\sin\zeta \\ 0 & 0 & 1 \end{pmatrix}$$

2D transzformációk kompozíciója 2. példa

Nagyítás egy tetszőleges P(x,y) pontból

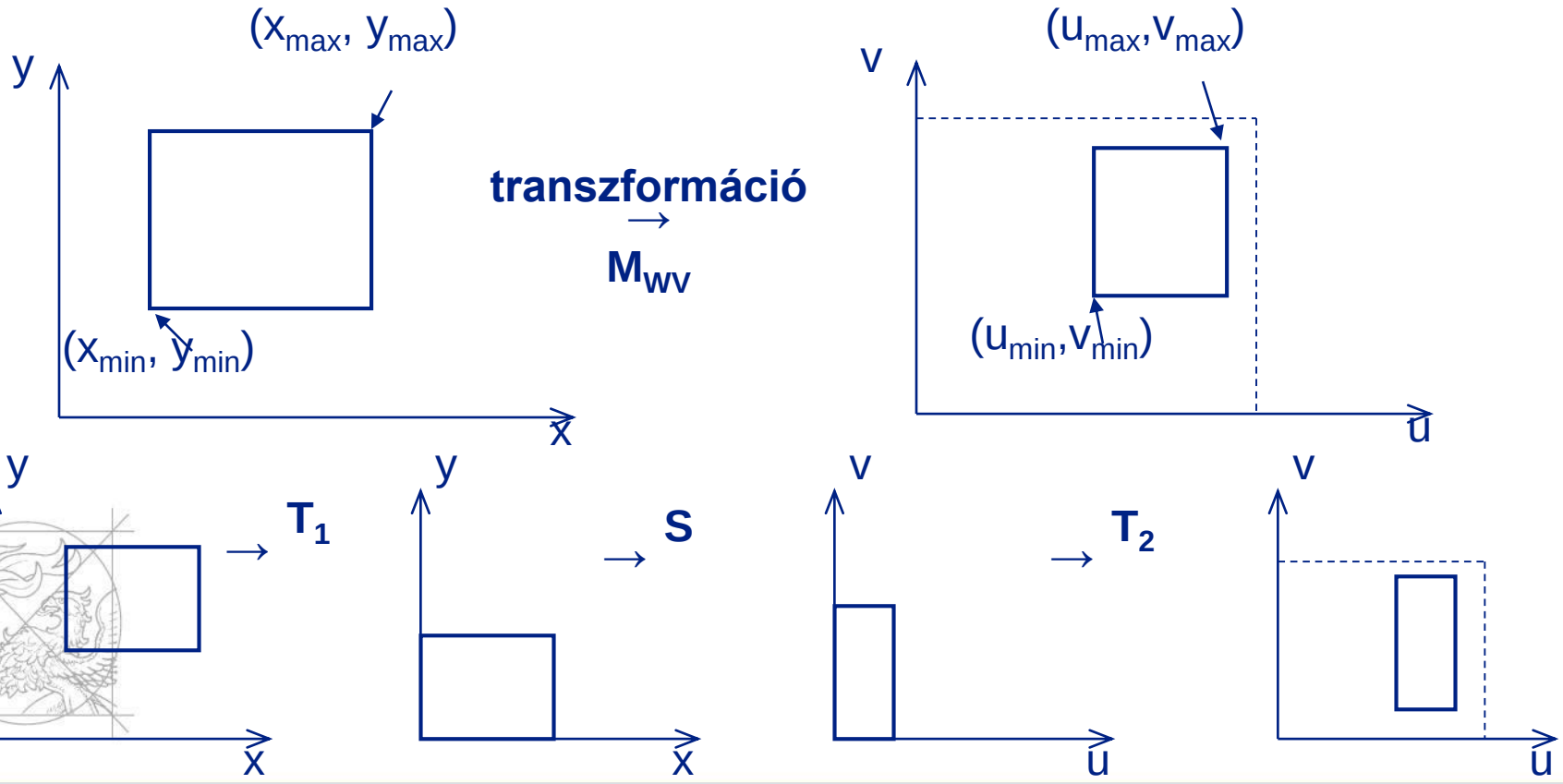
$$T(x,y) \cdot S(s_x,s_y) \cdot T(-x,-y) =$$

$$\begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & x(1-s_x) \\ 0 & s_y & y(1-s_y) \\ 0 & 0 & 1 \end{pmatrix}$$

2D transzformációk kompozíciója 3. példa / 1

Világ koordináta rendszer

Képernyő koordináta rendszer



2D transzformációk kompozíciója 3. példa /2

$$M_{wv} = T(u_{\min}, v_{\min}) S\left(\frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}}, \frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}}\right) T(-x_{\min}, -y_{\min}) =$$

$$= \begin{pmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\min} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}} & 0 & 0 \\ 0 & \frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}} & 0 & -x_{\min}\frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}} + u_{\min} \\ 0 & \frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}} & -y_{\min}\frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}} + v_{\min} \\ 0 & 0 & 1 \end{pmatrix}$$

2D transzformációk kompozíciója 3. példa /3

$$M_{wv} = \begin{pmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & -x_{min}\frac{u_{max}-u_{min}}{x_{max}-x_{min}}+u_{min} \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & -y_{min}\frac{v_{max}-v_{min}}{y_{max}-y_{min}}+v_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

tehát

$$P' = M_{wv}P(x,y) = \begin{pmatrix} (x-x_{min})\frac{u_{max}-u_{min}}{x_{max}-x_{min}}+u_{min} \\ (y-y_{min})\frac{v_{max}-v_{min}}{y_{max}-y_{min}}+v_{min} \end{pmatrix}$$

Gyorsítások

M·P számításakor:

9 szorzás és 6 összeadás helyett elegendő

$x' = x \cdot r_{11} + y \cdot r_{12} + t_x$

$y' = x \cdot r_{21} + y \cdot r_{22} + t_y$

kiszámítása, ami csak 4 szorzás és 4 összeadás

$$M = \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

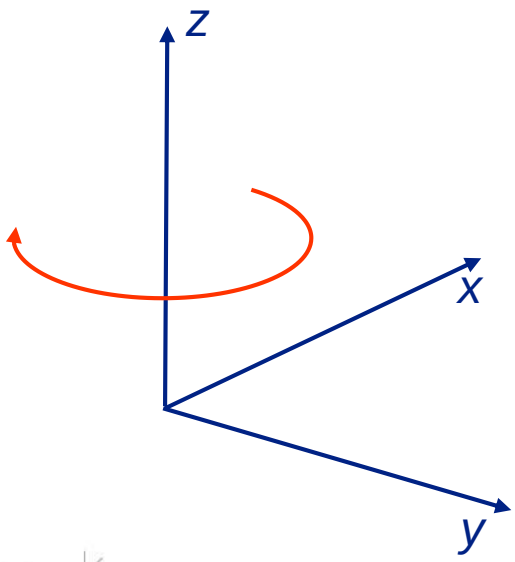
Ha kicsi ζ szöggel forgatunk, akkor $\cos \zeta \backsimeq 1$, így

$x' = x \cdot \cos \zeta - y \cdot \sin \zeta \backsimeq x - y \cdot \sin \zeta$

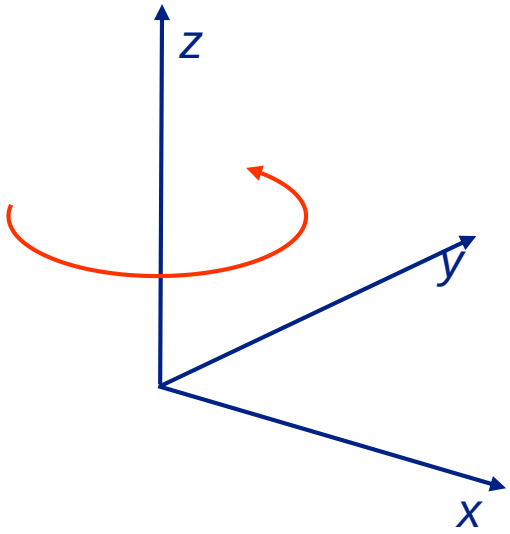
$y' = x \cdot \sin \zeta + y \cdot \cos \zeta \backsimeq x \cdot \sin \zeta + y$,

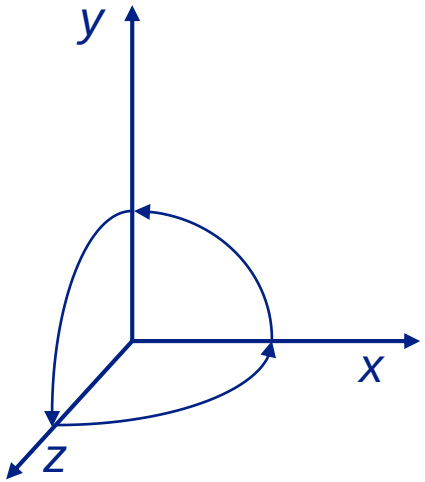
ami csak 2 szorzás és két összeadás (+ sin kiszámítása)

3D koordináta-rendszerek



balkezes
bal-sodrású





jobbkezes
jobb-sodrású

3D transzformációk - homogén koordináták

- (x,y,z) megadása homogén koordinátákkal: (x,y,z,1)
- (x,y,z,w) = (x',y',z',w'), ha van olyan α, hogy x' = α·x, y' = α·y, z' = α·z és w' = α·w
- Ha w≠0: (x/w,y/w,z/w,1) a szokásos jelölés
- Ha w=0: (x,y,z,0) végtelen távoli pont
- Kapcsolat: (x,y,z) - egyenes a 4-dimenziós térben aminek a w=1 -re való vetülete a homogén koordináta

3D eltolás

$$T(d_x, d_y, d_z) = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

mert

$$T(d_x, d_y, d_z) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{pmatrix}$$

$$T^{-1}(d_x, d_y, d_z) = T(-d_x, -d_y, -d_z)$$

3D skálázás (nagyítás/kicsinyítés)

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

mert

$$S(s_x, s_y, s_z) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \cdot s_x \\ y \cdot s_y \\ z \cdot s_z \\ 1 \end{pmatrix}$$

$$S^{-1}(s_x, s_y, s_z) = \begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D forgatások

A z-tengely körül

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

Az x-tengely körül

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

Az y-tengely körül

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

3D kompozíció-mártix

Skálázás/forgatás utáni eltolás

$$M = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} & R & & T \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{pmatrix} \qquad M\,P = R\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} + T$$

Ilyen módon hatékonyabban számítható!

3D nyírás (Z mentén)

$$SH_{xy}(sh_x, sh_y) = \begin{pmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

mert

$$SH_{xy}(sh_x, sh_y) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + sh_x z \\ y + sh_y z \\ z \\ 1 \end{pmatrix}$$

3D síkok transzformációi

- A sík egyenlete: Ax + By + Cz + D = 0
- Legyen P a sík tetszőleges pontja!
- Ha $P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$, akkor $N = \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix}$ a sík normálisa, hiszen $N^T P = 0$
- Ha a sík pontjait M-el transzformáljuk, akkor hogy transzformálódik a sík normálisa?

3D síkok transzformációi

- Legyen P tetszőleges pont a síkban! Ekkor

$$N^T P = 0.$$

- Melyik az a Q mátrix, amelyre $(QN)^T(MP) = 0$?
- Ha M^{-1} létezik, akkor

$$((M^{-1})^T N)^T (MP) = N^T((M^{-1})^T)^T MP = N^T P = 0$$

$$\Downarrow$$

$$Q=(M^{-1})^T$$

$$N' = (M^{-1})^T N$$

- (NEM BIZTOS, hogy M inverze létezik! Pl: vetítés)

3D koordináta-rendszerek váltása /1

$P^{(j)}$: a P pont az j koordináta-rendszerben

$M_{i \leftarrow j}$: transzformáció, amely
a j koordináta-rendszerbeli pontokat
az i koordináta-rendszerbe viszi át

Ekkor

$$P^{(i)} = M_{i \leftarrow j} P^{(j)}$$

Ha $P^{(j)} = M_{j \leftarrow k} P^{(k)}$, akkor

$$P^{(i)} = M_{i \leftarrow j} P^{(j)} = M_{i \leftarrow j} (M_{j \leftarrow k} P^{(k)}) = M_{i \leftarrow k} P^{(k)}$$

ahol

$$M_{i \leftarrow k} = M_{i \leftarrow j} M_{j \leftarrow k}$$

3D koordináta-rendszerek váltása /2

Továbbá

$$M_{i\leftarrow j} = M_{j\leftarrow i}^{-1}$$

pl:

a) Ha $M_{i\leftarrow j} = T(t_x,t_y)$, akkor $M_{j\leftarrow i} = T(-t_x,-t_y)$.

b) Ha R: jobb-kezes koordináta-rendszer
L: bal-kezes koordináta-rendszer, akkor

$$M_{R\leftarrow L} = M_{L\leftarrow R}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D transzformációk alakja különböző koordináta-rendszerekben

$P^{(j)}$: pont a j koordináta-rendszerben

$Q^{(j)}$: transzformáció a j koordináta-rendszerben

Melyik az a $Q^{(i)}$, amelyre

$$Q^{(i)} P^{(i)} = M_{i \leftarrow j} Q^{(j)} P^{(j)} \quad ?$$

Mivel $P^{(i)} = M_{i \leftarrow j} P^{(j)}$, ezért

$$Q^{(i)} M_{i \leftarrow j} P^{(j)} = M_{i \leftarrow j} Q^{(j)} P^{(j)}$$

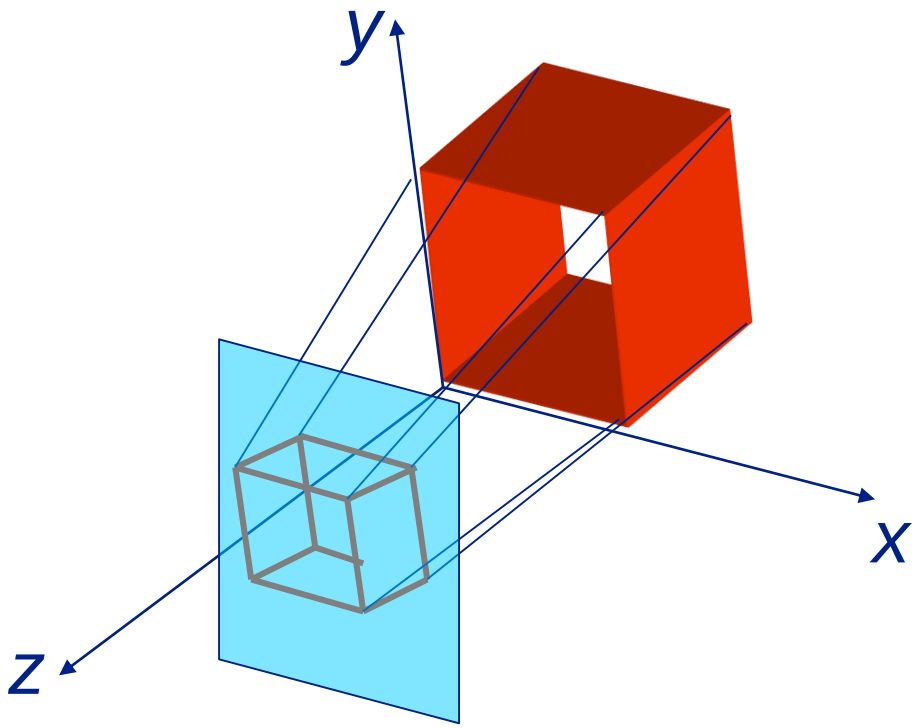
$$\Downarrow$$

$$Q^{(i)} M_{i \leftarrow j} = M_{i \leftarrow j} Q^{(j)}$$

$$Q^{(i)} = M_{i \leftarrow j} Q^{(j)} M_{i \leftarrow j}^{-1}$$

4. VETÍTÉSEK

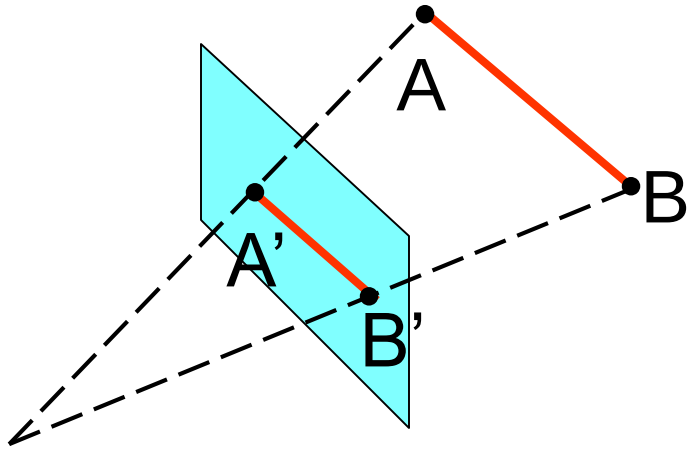
VETÍTÉSEK



- Transzformációk, amelyek *n*-dimenziós objektumokat kisebb dimenziós terekbe visznek át. Pl. 3D→2D

Vetítések fajtái /1

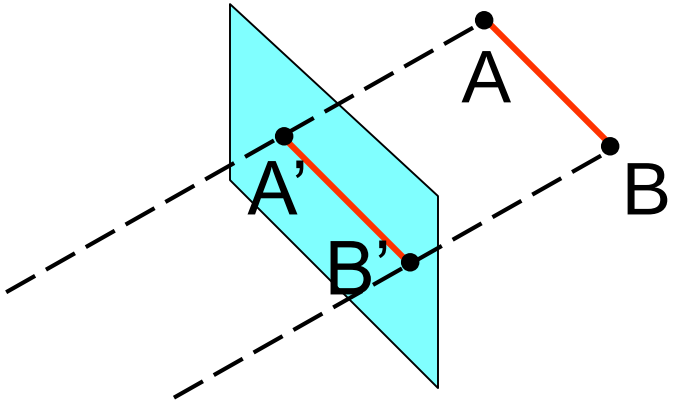
perspektívikus



Vetítés középpontja

Vetítési sík

párhuzamos



Vetítés középpontja a végtelenben

Vetítési sík

Vetítések fajtái /2

Perspektívikus

Vetítési középpont

- közel áll látásunkhoz
- általában: nem mér-hetők a távolságok (rövidülés) és a szögek

Párhuzamos

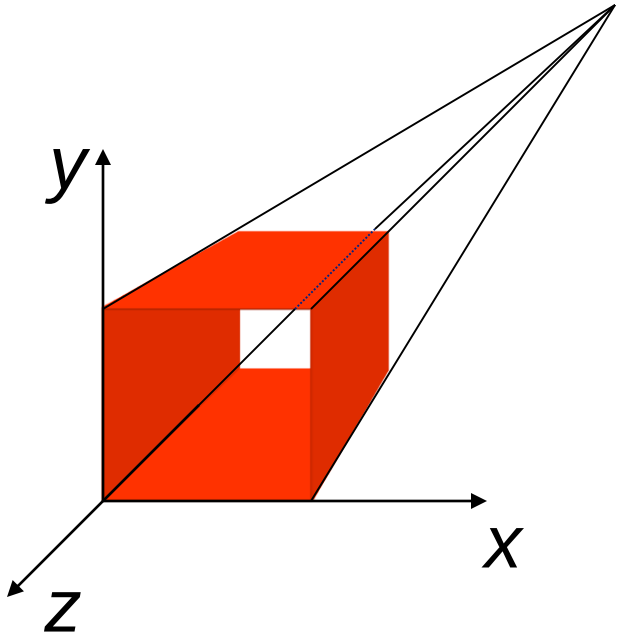
Vetítési irány

- kevésbé realisztikus
- mérhető távolságok, szögek változnak

Perspektív vetítések /1

A vetítési síkkal nem, de egymással párhuzamos egyenesek vetületei
egy pontban metszik egymást = <u>távlatpont</u>

Elsődleges távlatpont:
valamelyik fő(tengely) irányhoz tartozó távlatpont



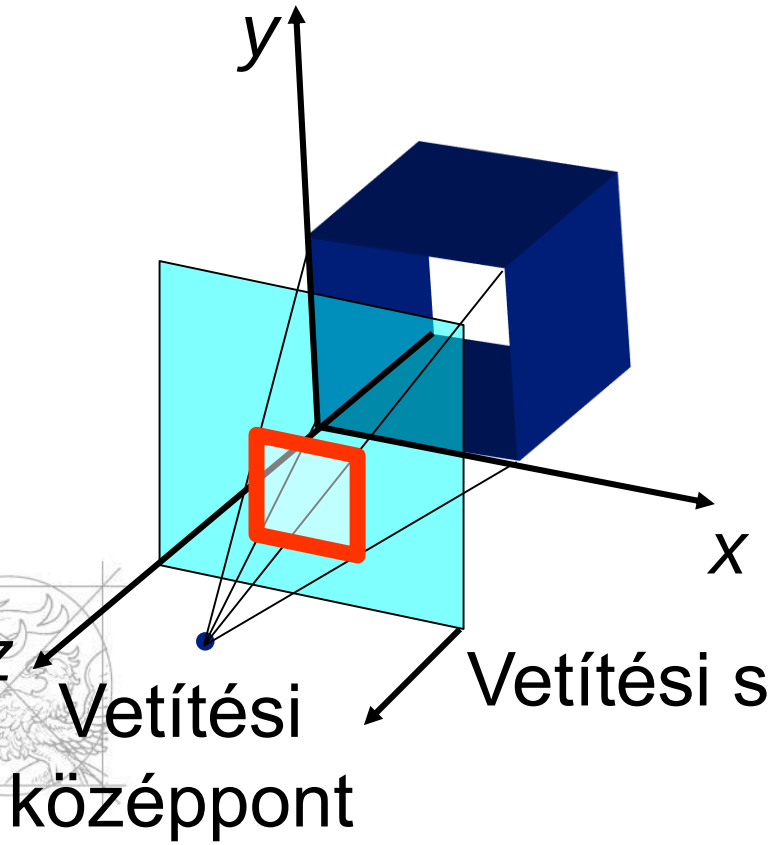
Perspektív vetítések /2

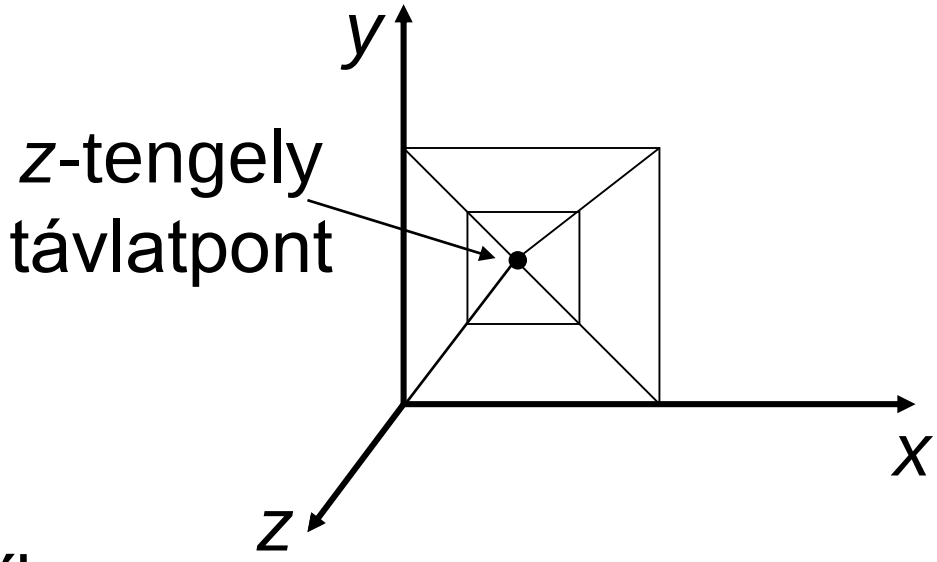
Perspektív vetítések osztályozása:

az elsődleges távlatpontok száma szerint (1, 2, 3).

Perspektív vetítések /3

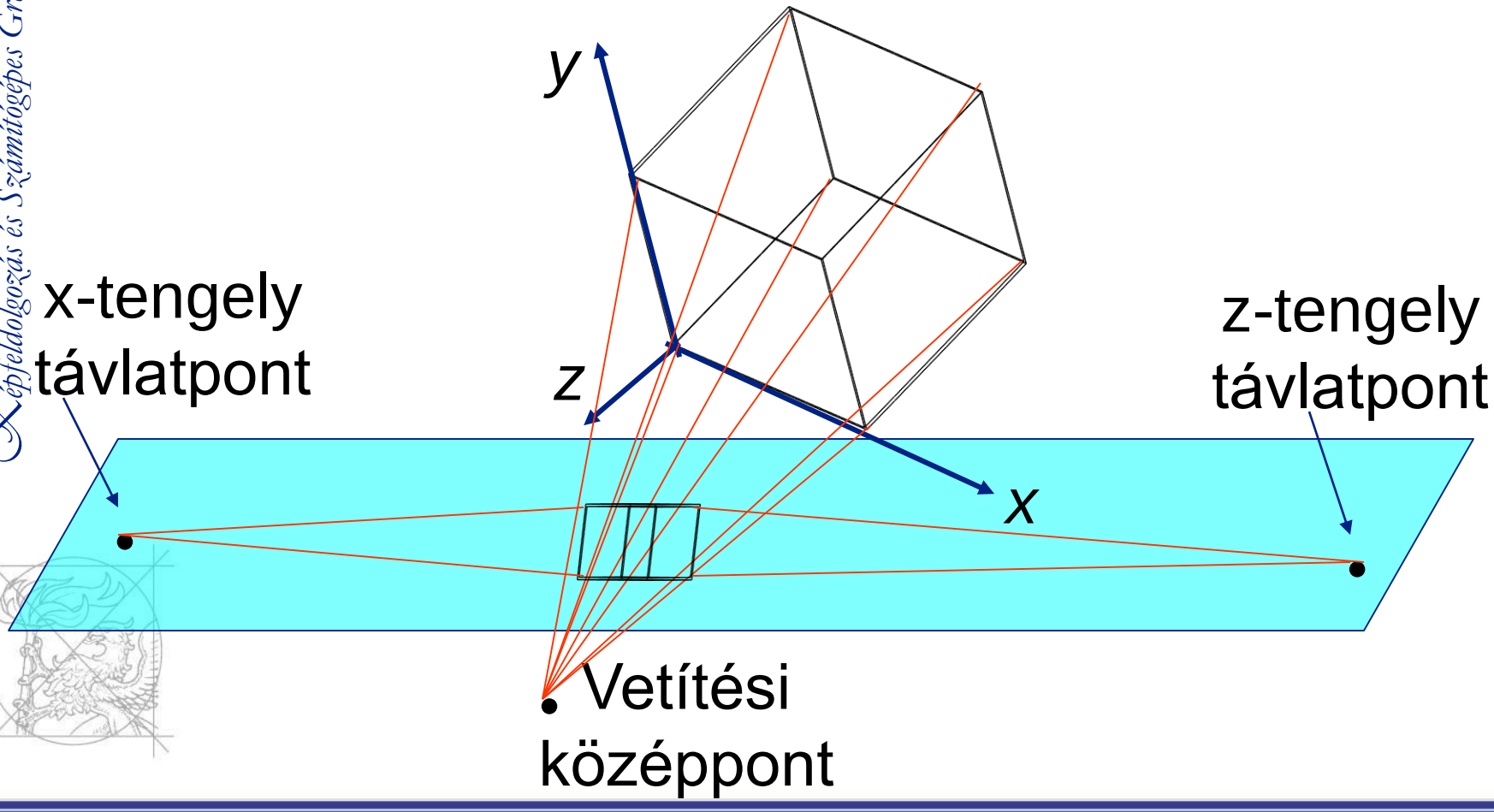
1. Egypontos perspektív vetítés





Perspektív vetítések /4

2. Kétpontos perspektív vetítés



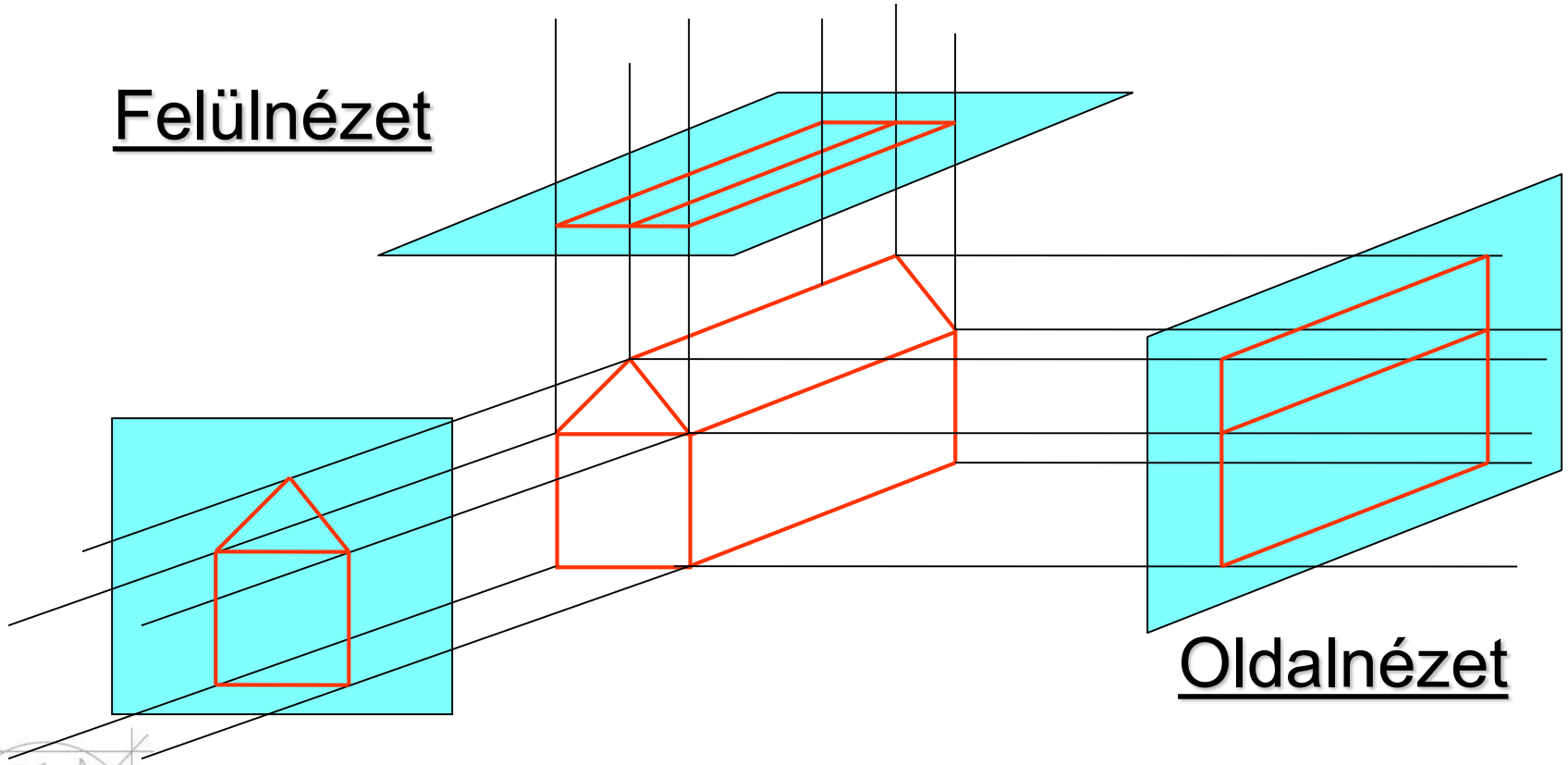
Párhuzamos vetítések

A párhuzamosság megmarad

Osztályozásuk a vetítési irány és a vetítési sík egymáshoz viszonyított helyzete szerint:

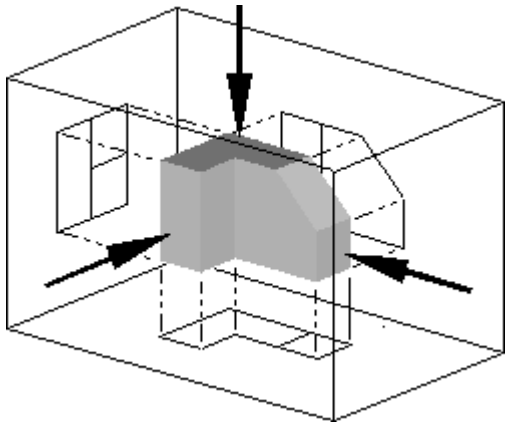
1. Merőleges (ortografikus)
2. Tetszőleges irányú

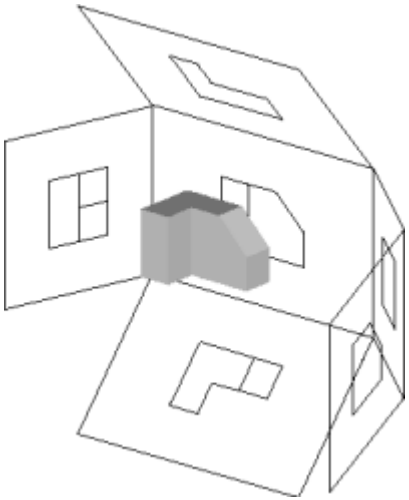
Merőleges (ortografikus) /1

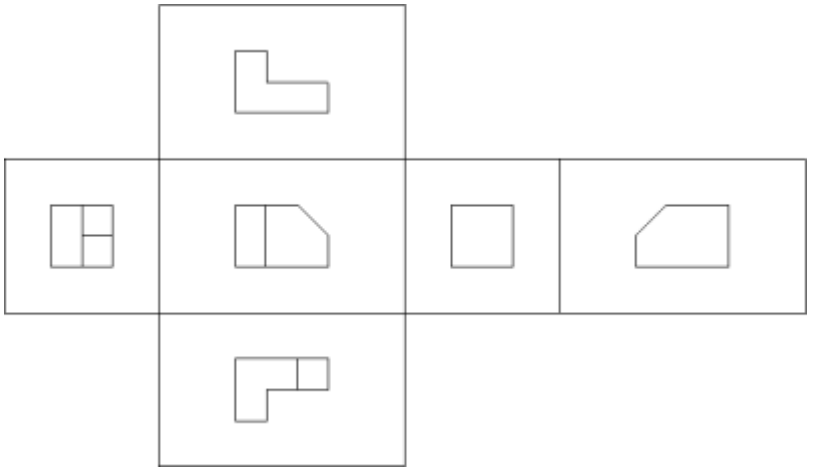


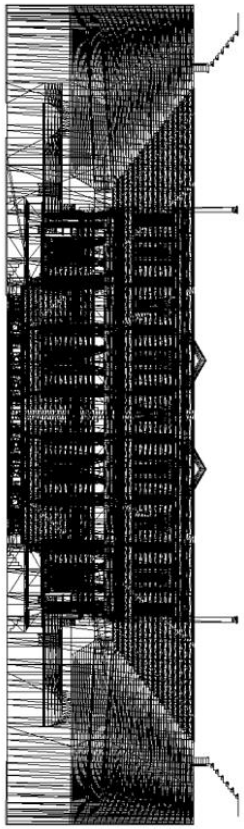
A párhuzamosság megmarad,
a távolságok megmaradnak vagy számíthatók

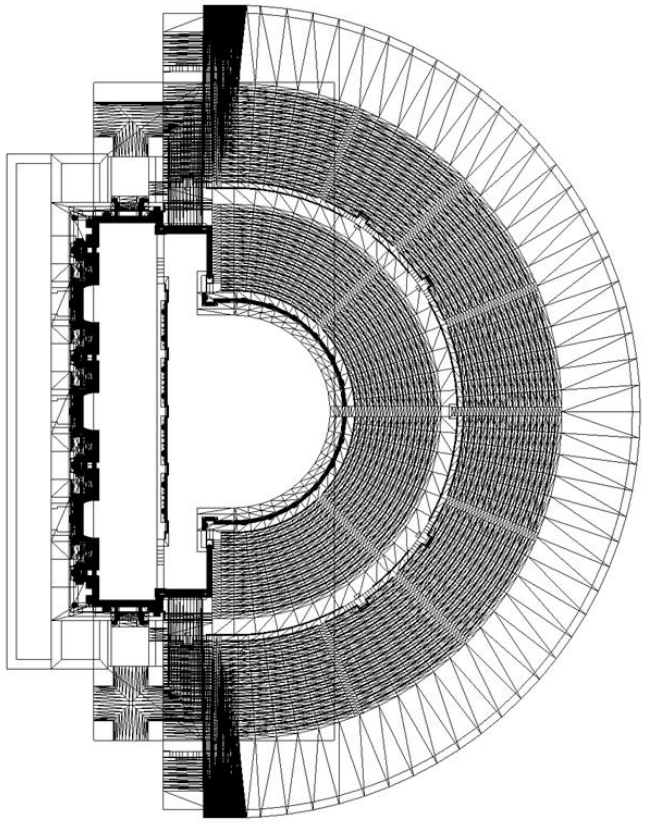
Példa

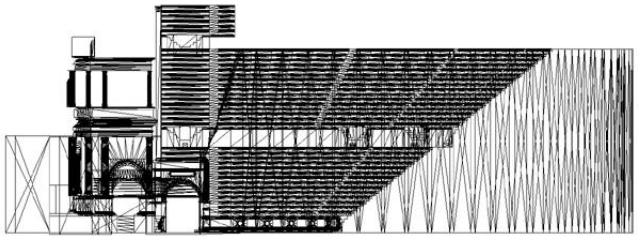












Amfiteátrium Jerash-ban

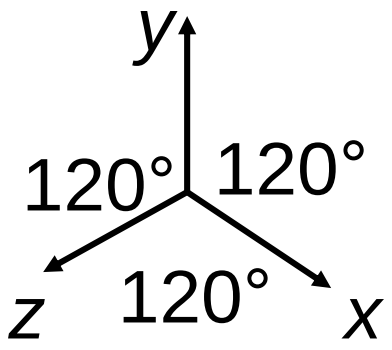
Merőleges (ortografikus) /2

Axonometrikus (nem merőleges egyik tengelyre sem); szög nem marad meg, távolság számítható

Izometrikus (a vetítési irány (d_x, d_y, d_z) minden tengellyel ugyanakkora szöget zár be), azaz

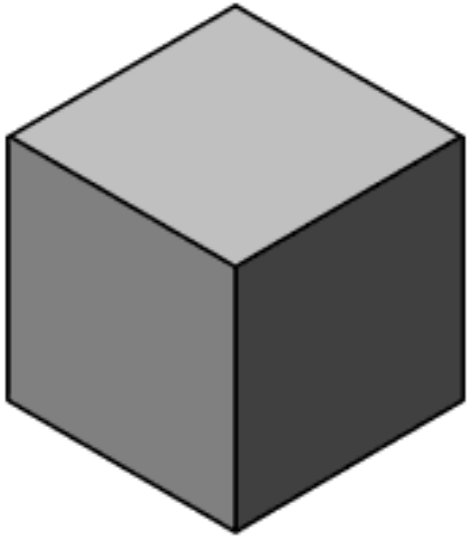
$$|d_x| = |d_y| = |d_z|,$$

$$\pm d_x = \pm d_y = \pm d_z$$

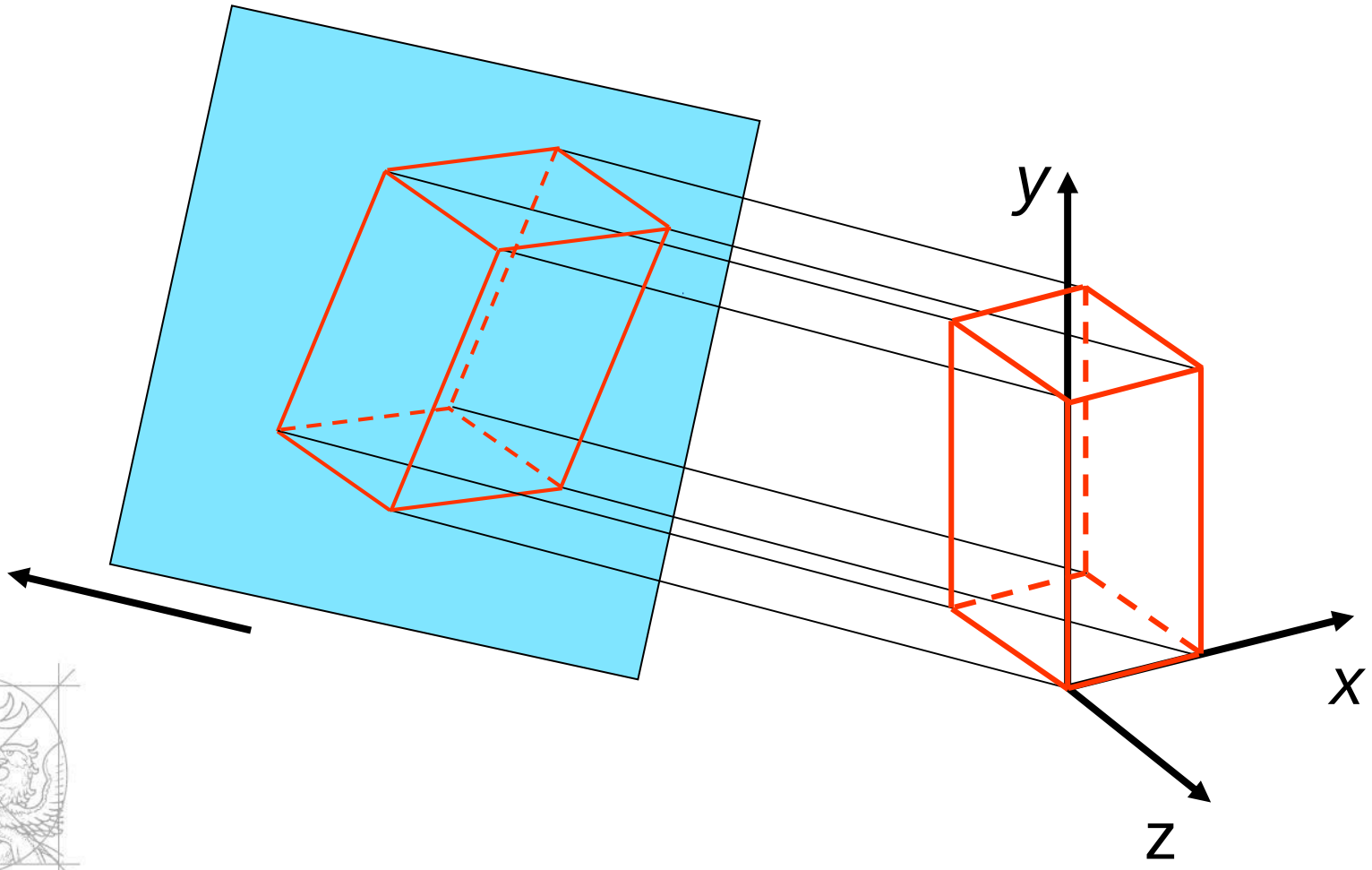


8 ilyen irány létezik

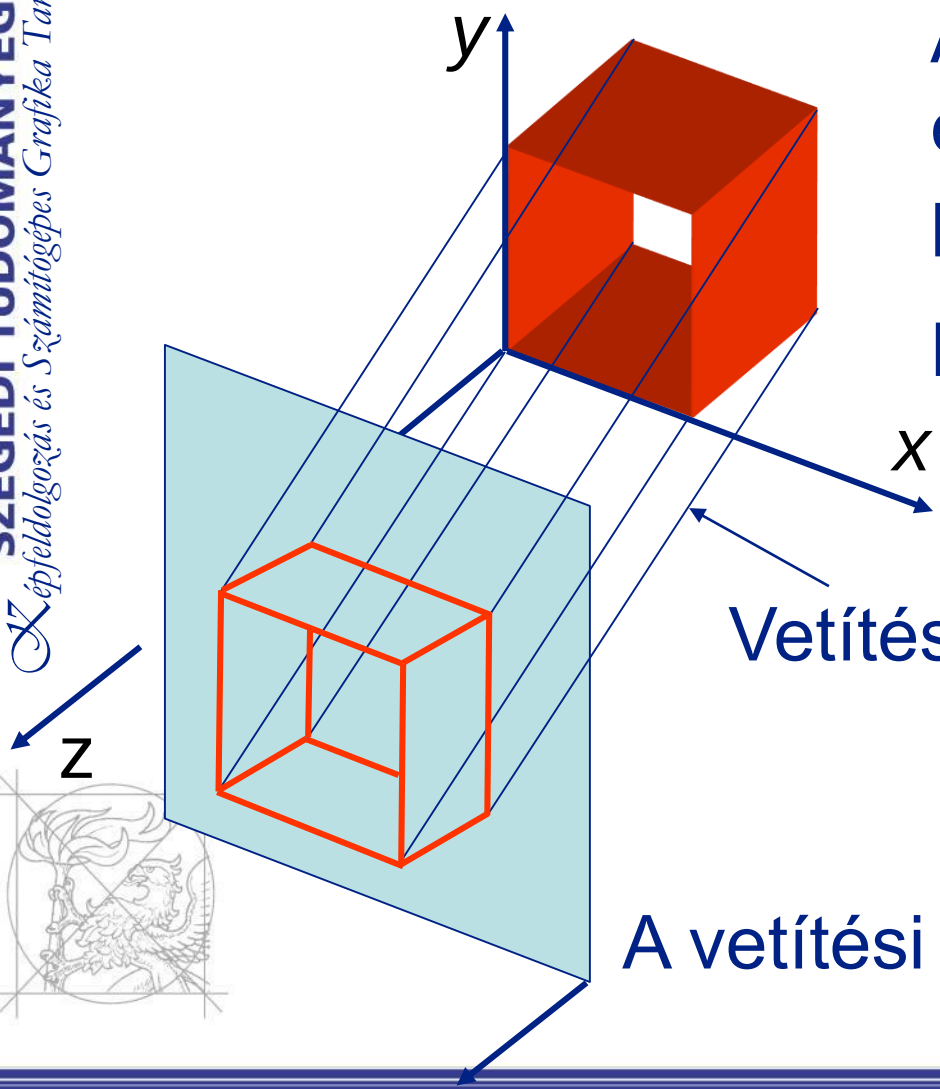
Izometrikus vetítés



Merőleges (ortografikus) /3



Tetszőleges irányú



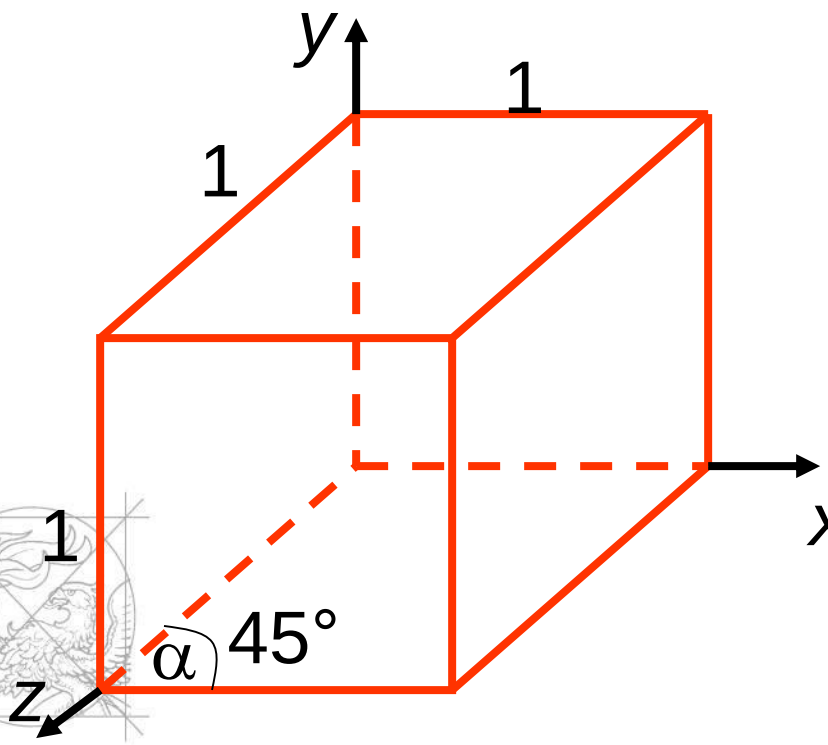
A vetítési sík normálisa és a vetítési irány nem párhuzamos

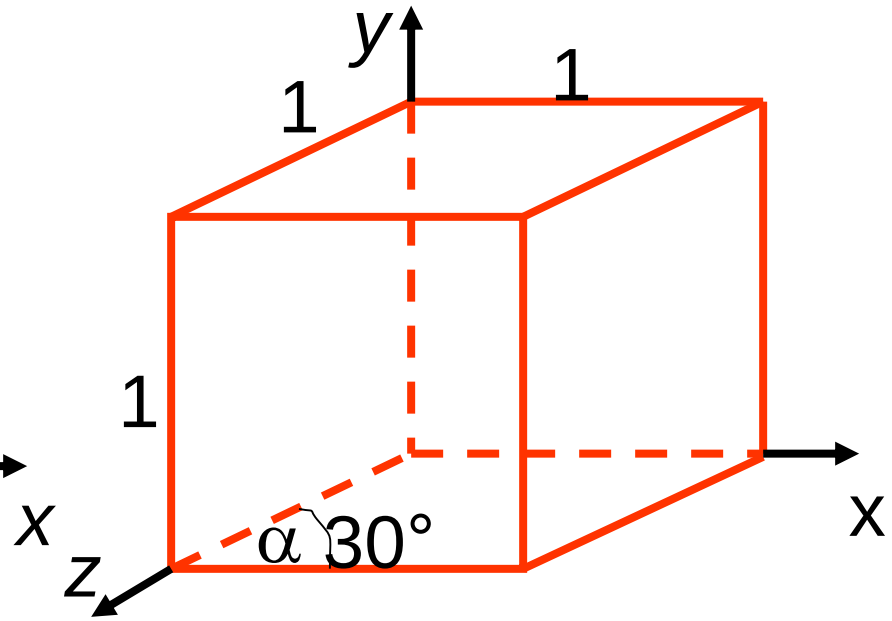
Leggyakoribb fajtái:

- kavalier
- kabinet

Ferde irányú

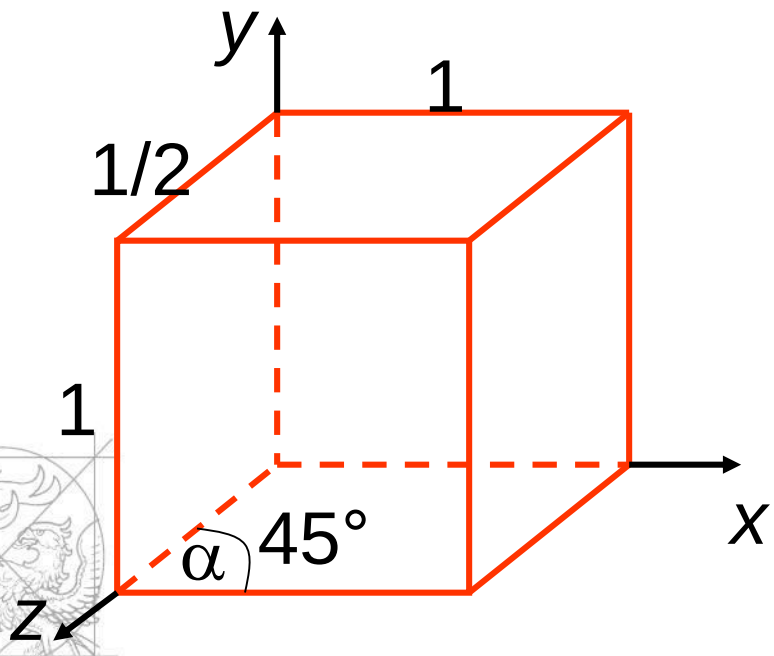
Kavalier:

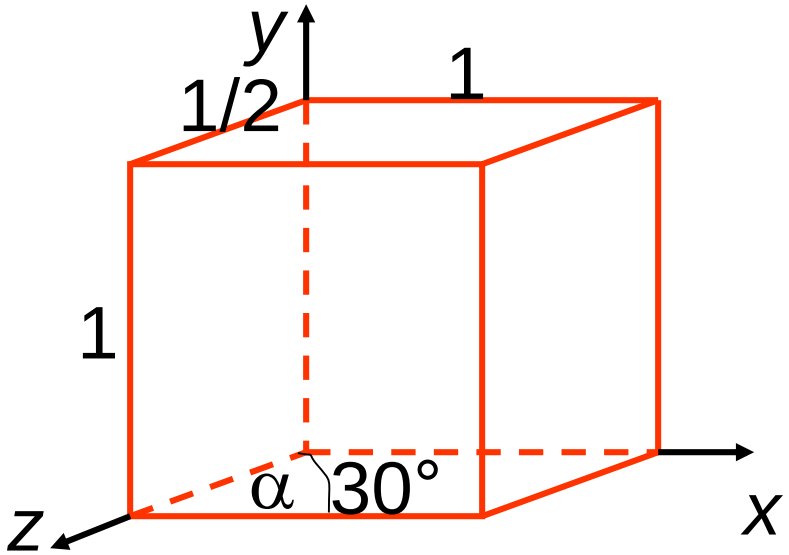




Ferde irányú

Kabinet:





3-D megjelenítés specifikálása /1

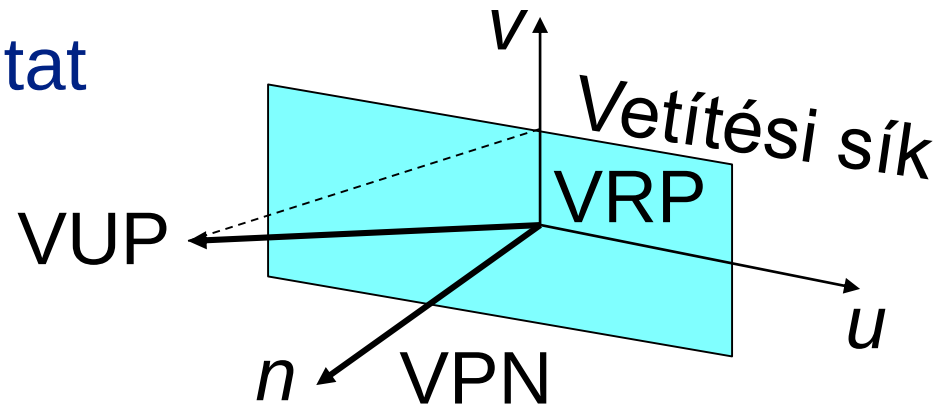
Szükséges: - látótér, - vetítés } megadása

Vetítési sík megadása:

egy pontjával - referencia pont (VRP)

és a normálisával (VPN)

v a fölfelé mutató vektor (VUP) vetülete irányába mutat



3-D megjelenítés specifikálása /2

3D referencia koordináta rendszer (VRC) megadása:

Origó = VRP

A tengelyek:

n = VPN,

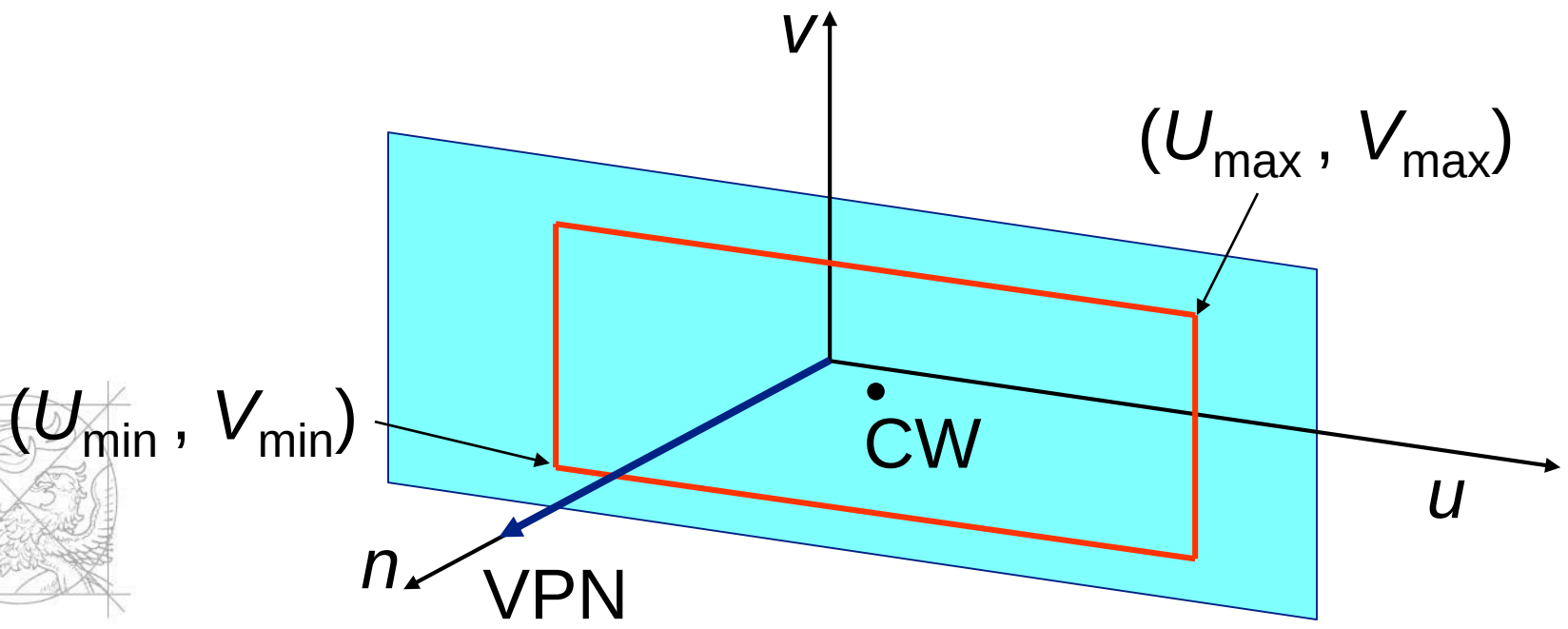
v = VUP-nek a vetítési síkra eső vetülete

u = olyan, hogy *u, v, n* jobb-kezes derékszögű koordináta rendszert határozzon meg

3-D megjelenítés specifikálása /3

Ablak: Téglalap a vetítési síkon. Ami azon belül van, az megjelenik, a többi nem

CW a közepe

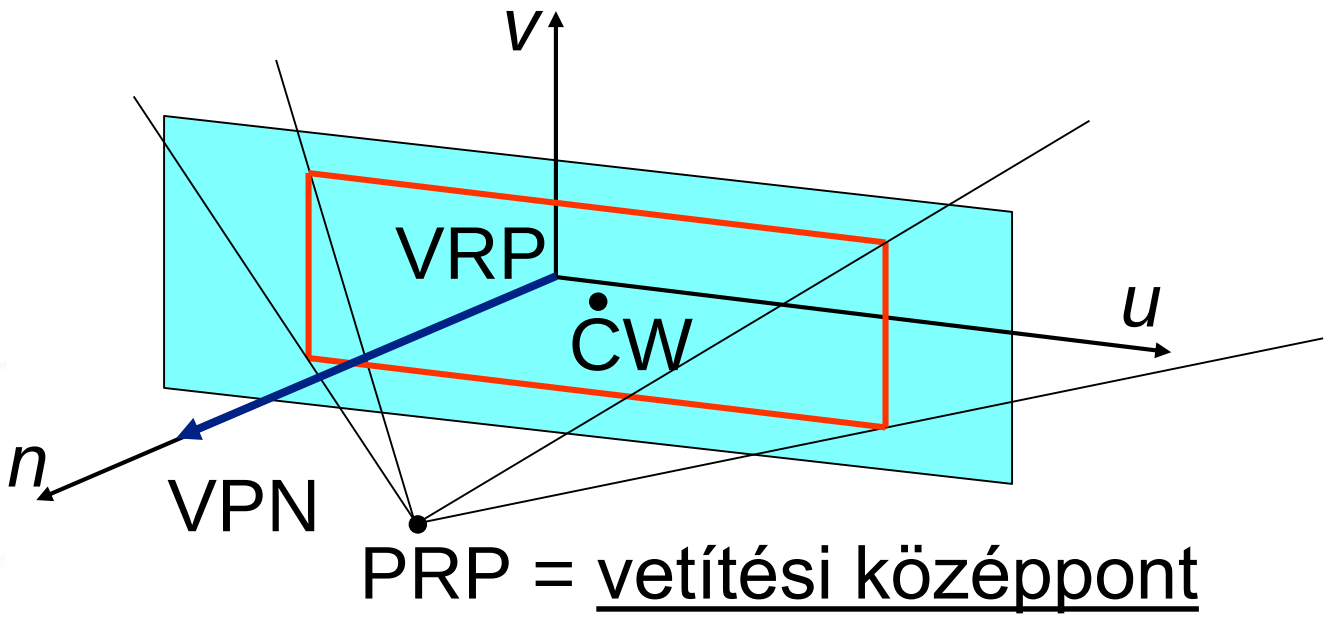


3-D megjelenítés specifikálása /4

PRP: vetítési referencia pont:

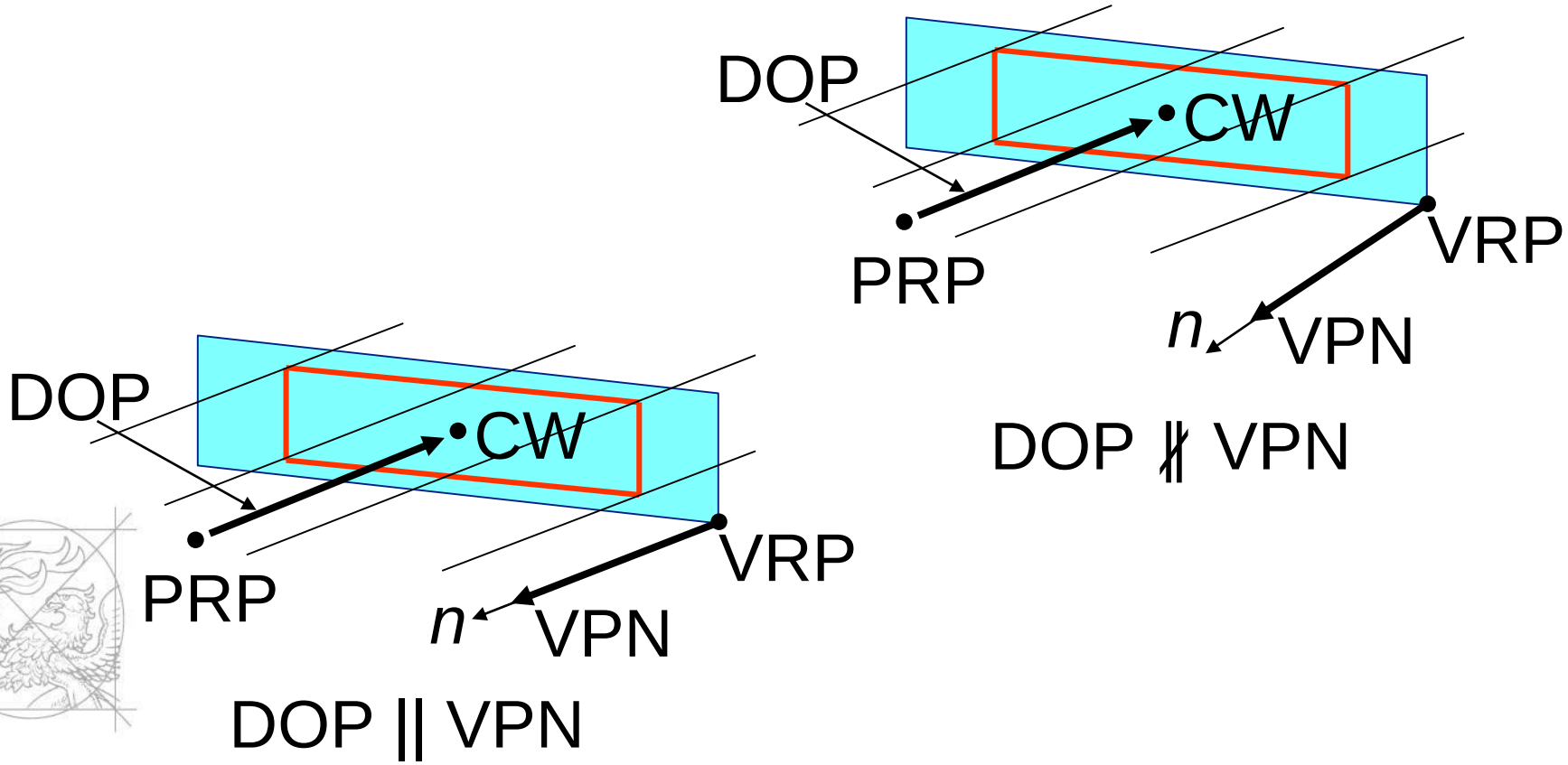
(párhuzamos és perspektív vetítésre is)

Perspektívikus vetítésnél



3-D megjelenítés specifikálása /5

DOP: vetítési irány

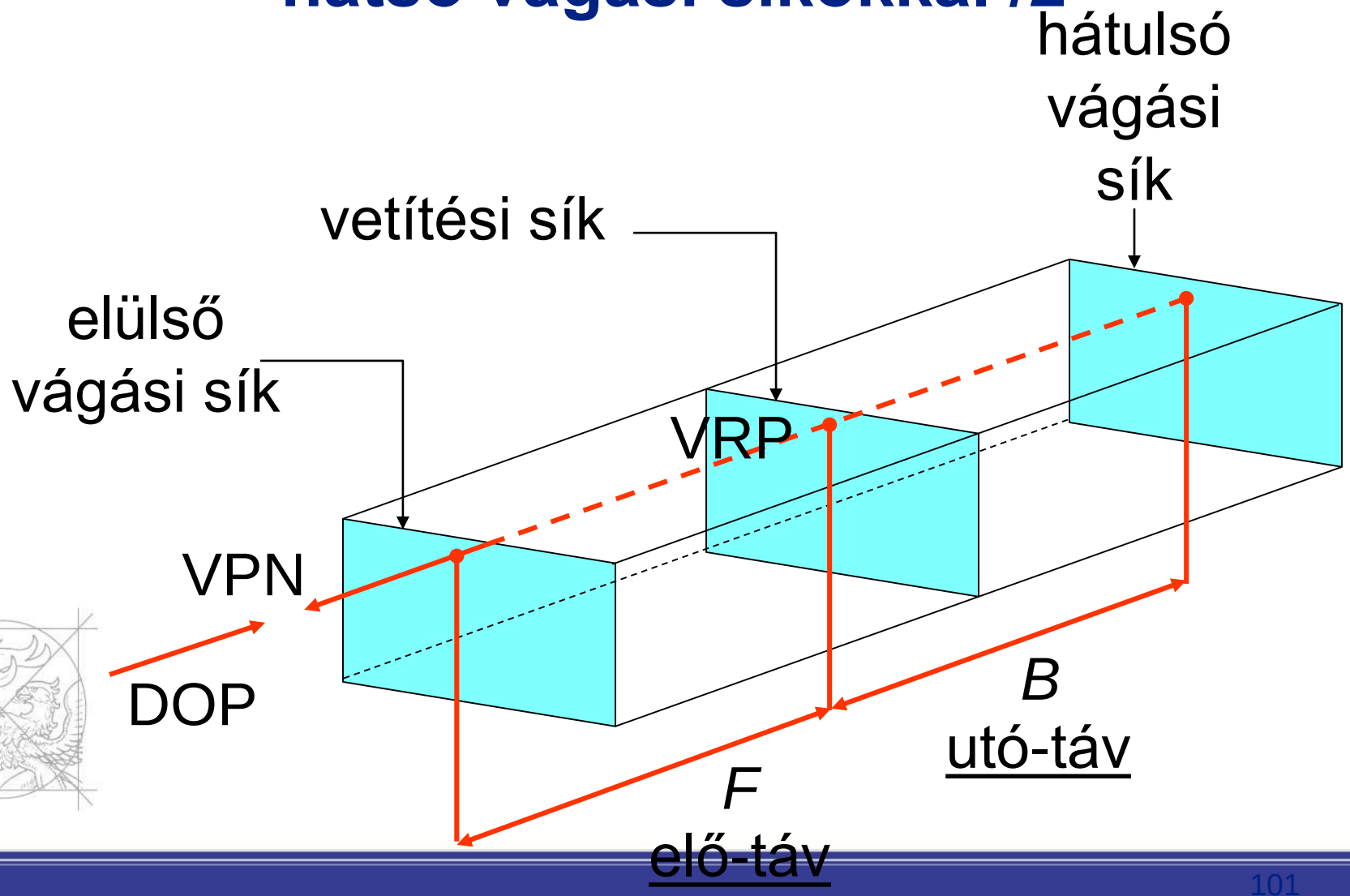


Látótér maghatározása elülső és hátsó vágási síkokkal /1

Fajtái:

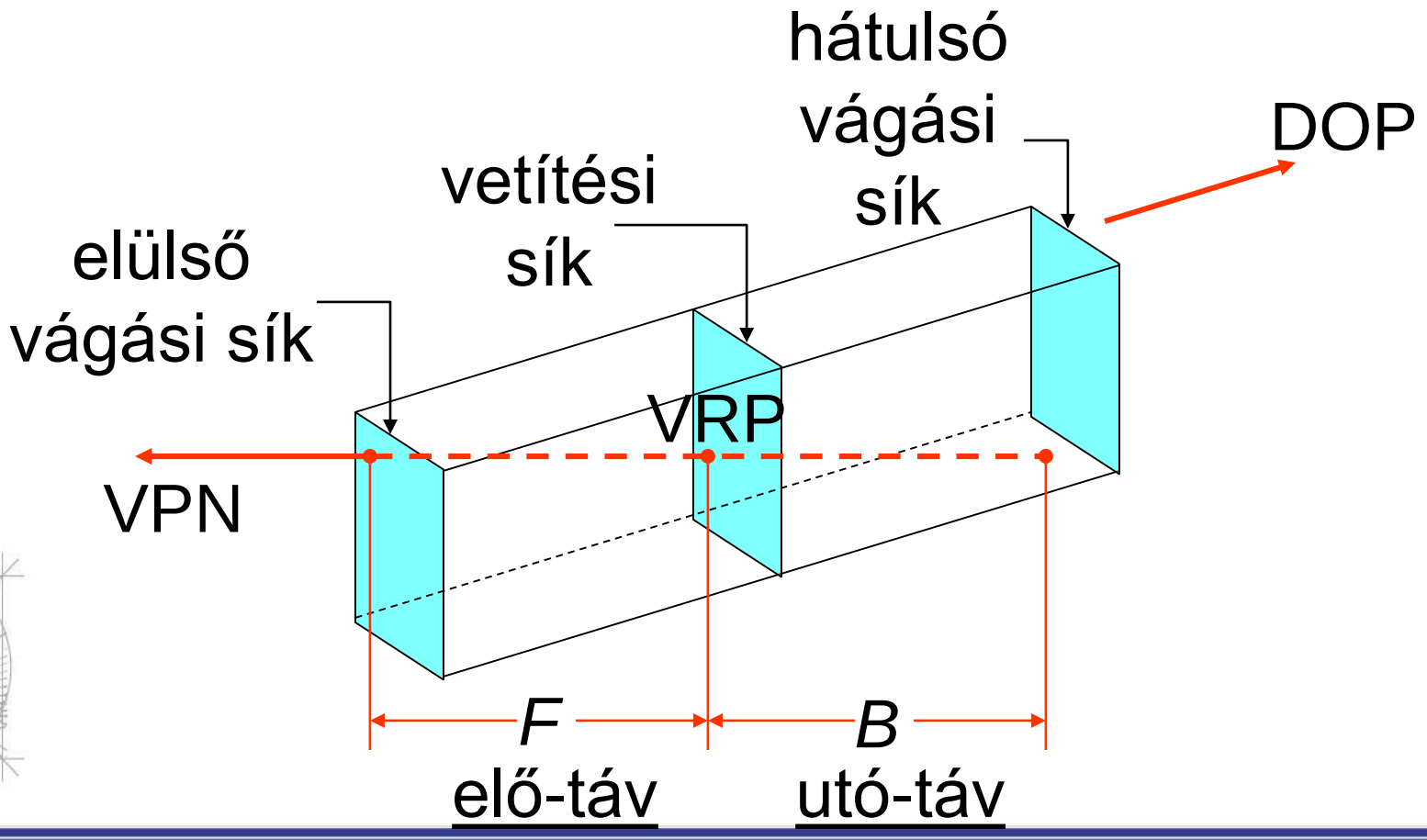
- párhuzamos (ortografikus)
- párhuzamos (tetszőleges irányú)
- perspektivikus

Látótér maghatározása elülső és hátsó vágási síkokkal /2



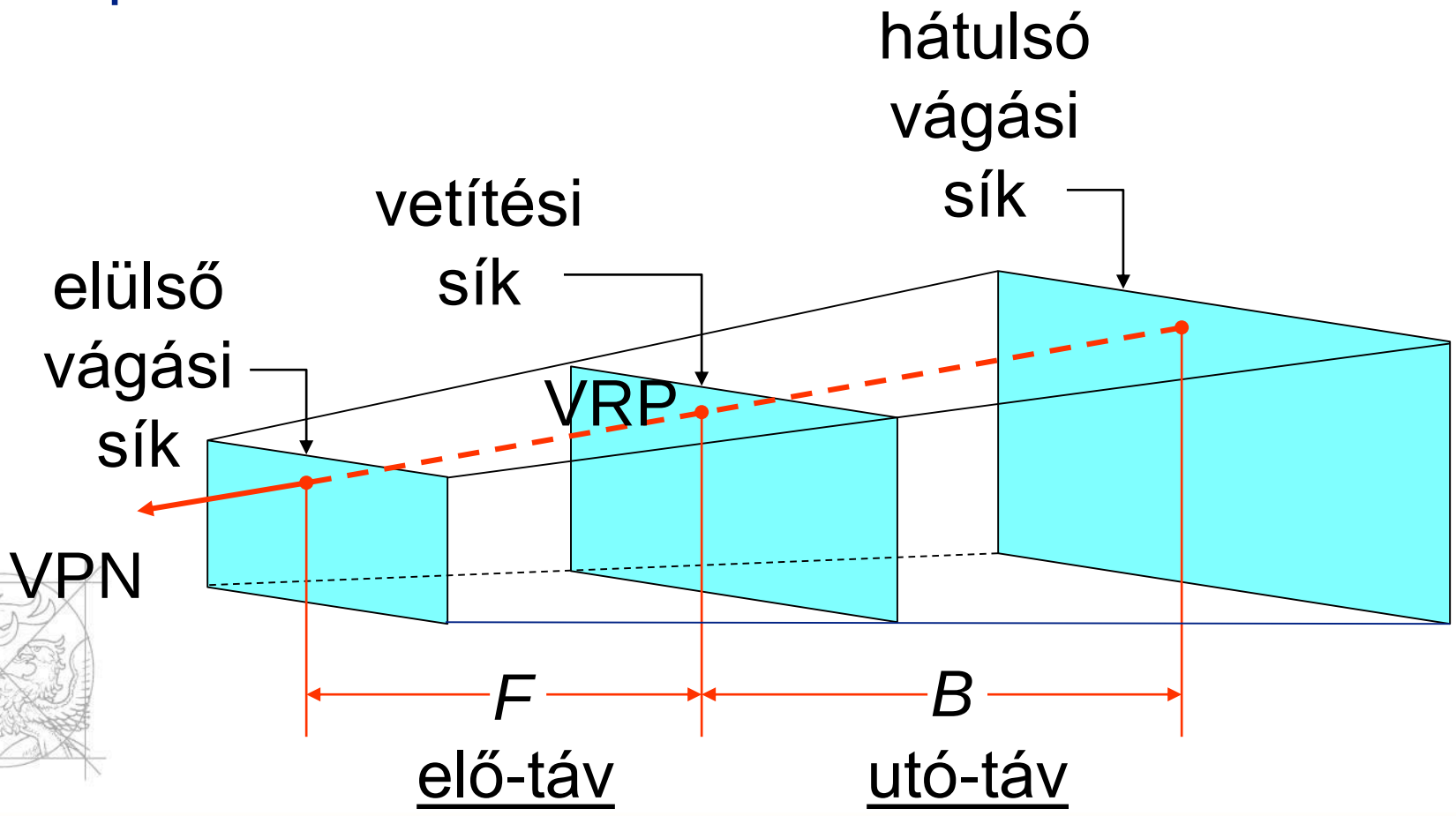
Látótér maghatározása elülső és hátsó vágási síkokkal /3

Párhuzamos (tetszőleges irányú):



Látótér maghatározása elülső és hátsó vágási síkokkal /4

Perspektivikus:



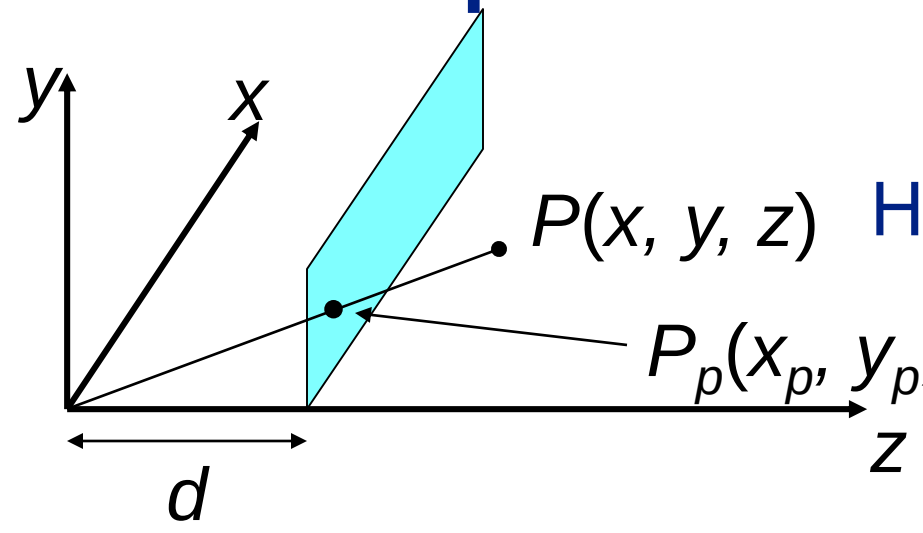
Vetítések matematikai leírása

Perspektívikus vetítések /1

Az egyszerűség kedvéért tegyük fel hogy:

a) A vetítési sík merőleges a *z*-tengelyre $z = d$-nél, PRP = 0

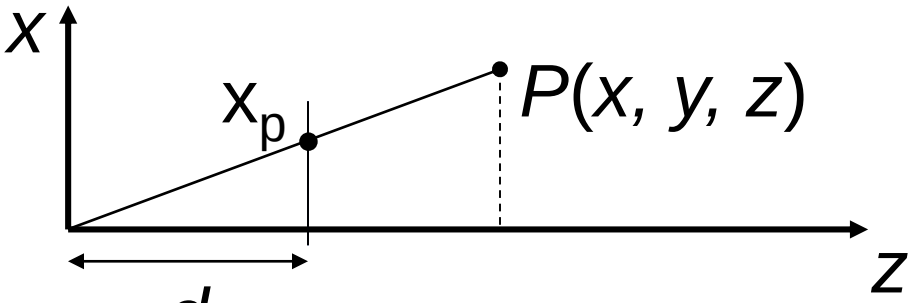
Perspektívikus vetítések /2

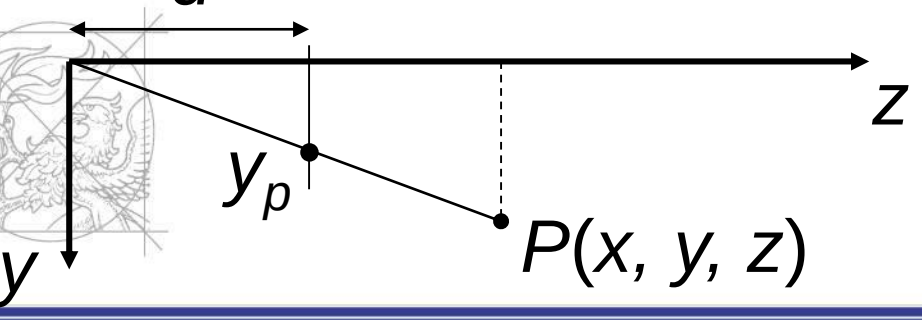


Hasonló háromszögekből:

$$\frac{x_p}{d} = \frac{x}{z}, \qquad \frac{y_p}{d} = \frac{y}{z}, \qquad z \neq 0$$

$$x_p = \frac{x}{z/d}, \quad y_p = \frac{y}{z/d},$$





vetületi síkok

Perspektívikus vetítések /3

Hasonló háromszögekből:

$$\frac{x_p}{d} = \frac{x}{z}, \qquad \frac{y_p}{d} = \frac{y}{z}, \qquad z \neq 0,$$

$$x_p = \frac{x}{z/d}, \quad y_p = \frac{y}{z/d},$$

$$\begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} = \left.\begin{pmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{pmatrix}\right\} P_p$$

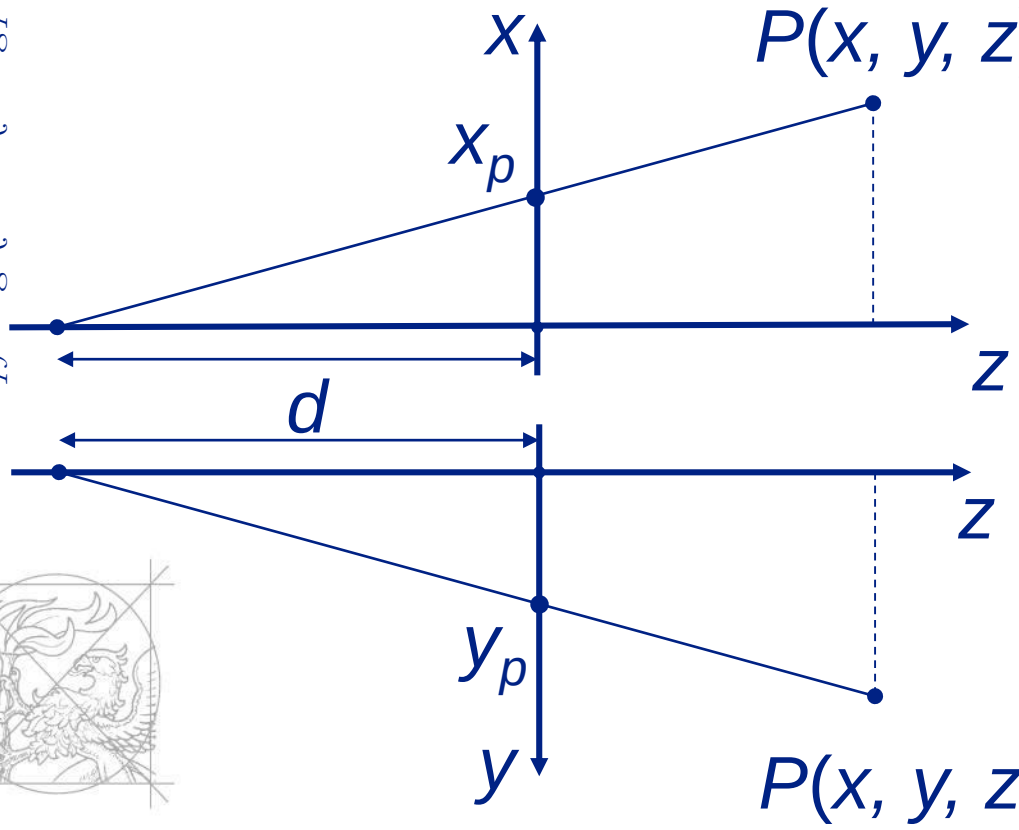
, mivel ez homogén koordináta,

tehát $M_{per} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$, mert $M_{per} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix}$

Perspektívikus vetítések /4

Más lehetőség

A vetítési sík merőleges a *z*-tengelyre *z* = *0*-ban



$$\frac{x_p}{d} = \frac{x}{z+d}, \qquad \frac{y_p}{d} = \frac{y}{z+d},$$

$$x_p = \frac{x}{z/d+1}, \qquad y_p = \frac{y}{z/d+1},$$

Perspektívikus vetítések /5

$$\frac{x_p}{d} = \frac{x}{z+d}, \quad \frac{y_p}{d} = \frac{y}{z+d},$$

$$x_p = \frac{x}{z/d+1}, \quad y_p = \frac{y}{z/d+1},$$

tehát $$P'_{per} = \begin{pmatrix} \frac{x}{z/d+1} \\ \frac{y}{z/d+1} \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ z/d+1 \end{pmatrix}$$

ezért $$M'_{per} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}$$

Párhuzamos ortografikus vetítés

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \xrightarrow{M_{ort}} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = P_p,$$

ahol

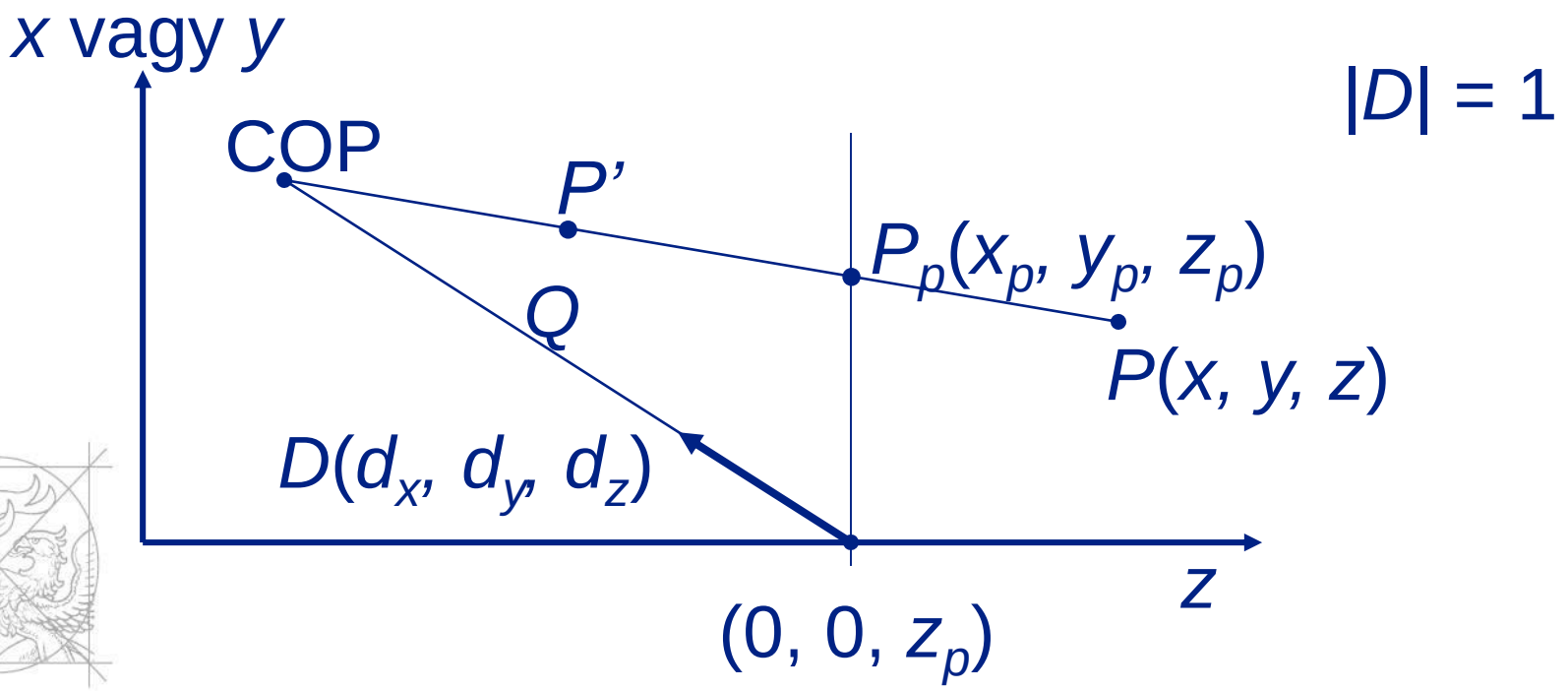
$$M_{ort} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(határértéke M'_{per}-nek, $d \to \infty$).

Vetítések általános alakja /1

Vetítési sík ⊥ z-tengely, $z = z_p$-ben,

COP *Q* távolságra van $(0, 0, z_p)$-től



Vetítések általános alakja /2

Parametrikus alak:

$$P' = \mathrm{COP} + t \cdot (P - \mathrm{COP}), \qquad t \in [0,1]$$

másrészt

$$\mathrm{COP} = (0, 0, z_p) + Q \cdot (d_x, d_y, d_z),$$

Így

$$x' = Q \cdot d_x + (x - Q \cdot d_x) \cdot t,$$

$$y' = Q \cdot d_y + (y - Q \cdot d_y) \cdot t,$$

$$z' = (z_p + Q \cdot d_z) + (z - (z_p + Q \cdot d_z)) \cdot t.$$

Innen $z' = z_p$ esetén

$$t = \frac{z_p - (z_p + Q \cdot d_z)}{z - (z_p + Q \cdot d_z)} = \frac{z - z_p}{z_p + Q \cdot d_z - z}$$

Vetítések általános alakja /3

Behelyettesítve és átalakítva:

$$x' = x_p = \frac{x - z \cdot \frac{d_x}{d_z} + z_p \cdot \frac{d_x}{d_z}}{\frac{z_p - z}{Q \cdot d_z} + 1}, \qquad y' = y_p = \frac{y - z \cdot \frac{d_y}{d_z} + z_p \cdot \frac{d_y}{d_z}}{\frac{z_p - z}{Q \cdot d_z} + 1}$$

$$z' = z_p = \frac{-z \cdot \frac{z_p}{Q \cdot d_z} + \frac{z_p^2 + z_p \cdot Q \cdot d_z}{Q \cdot d_z}}{\frac{z_p - z}{Q \cdot d_z} + 1}$$

Vetítések általános alakja /4

Így

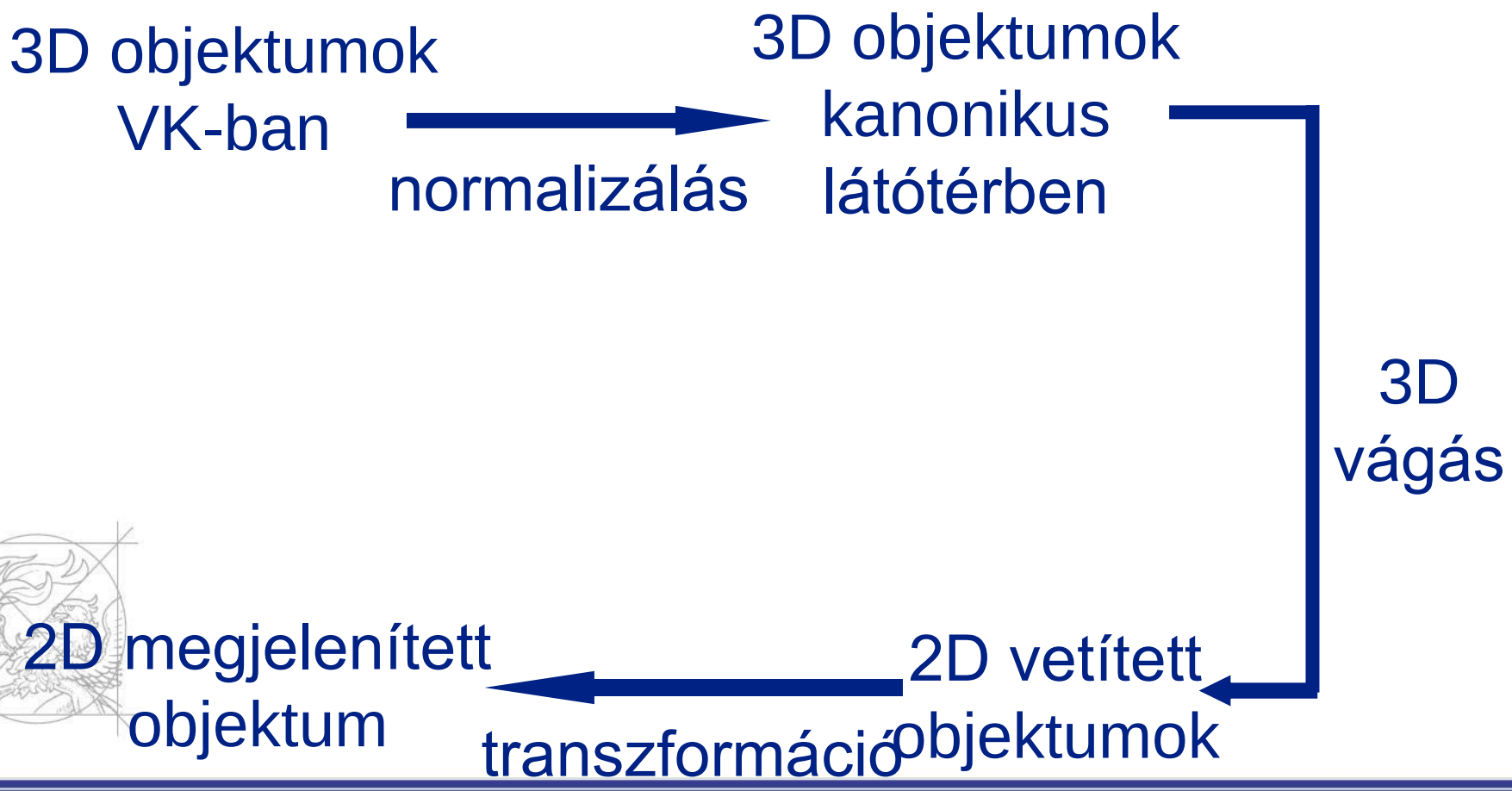
$$M_{ált} = \begin{pmatrix} 1 & 0 & \frac{-d_x}{d_z} & z_p \cdot \frac{d_x}{d_z} \\ 0 & 1 & \frac{-d_y}{d_z} & z_p \cdot \frac{d_y}{d_z} \\ 0 & 0 & \frac{-z_p}{Q \cdot d_z} & \frac{z_p^2}{Q \cdot d_z} + z_p \\ 0 & 0 & \frac{-1}{Q \cdot d_z} & \frac{z_p}{Q \cdot d_z} + 1 \end{pmatrix}$$

Tartalmazza M_{per}, M'_{per} és M_{ort}-ot (még többet is).

Pl.:

M_{ort}: $z_p=0$, $Q=\infty$, $(d_x, d_y, d_z)=(0, 0, -1)$

3D megjelenítés implementálása /1



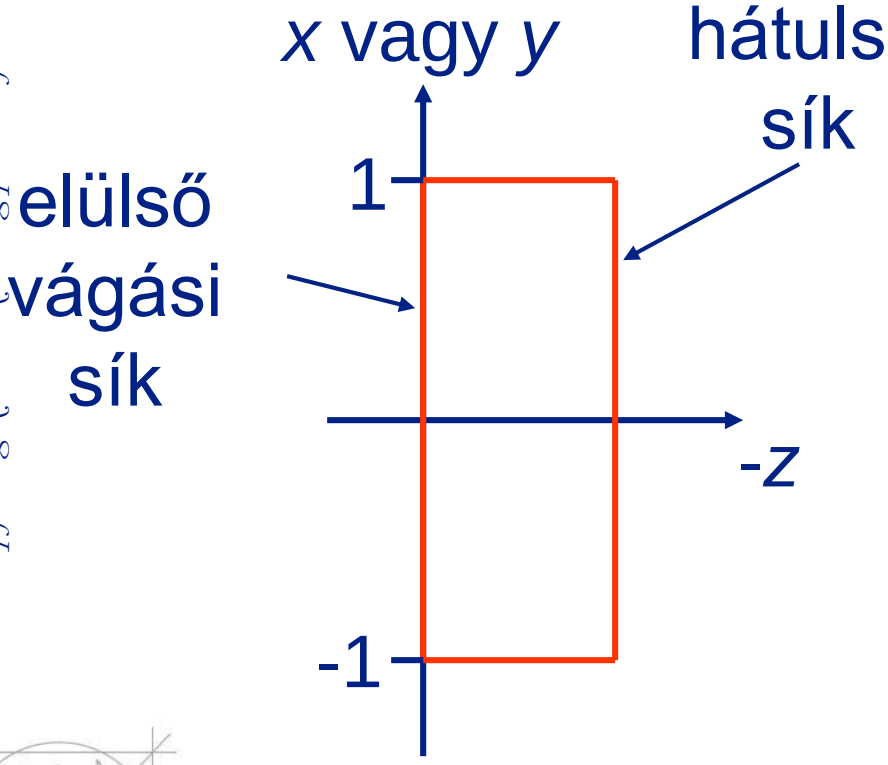
3D megjelenítés implementálása /2

A 3D vágás drága művelet, ezért érdemes előtte a 3D objektumokat egy u.n. kanonikus látótérbe transzformálni (normalizálás), ahol a vágás egyszerűbb és gyorsabb.

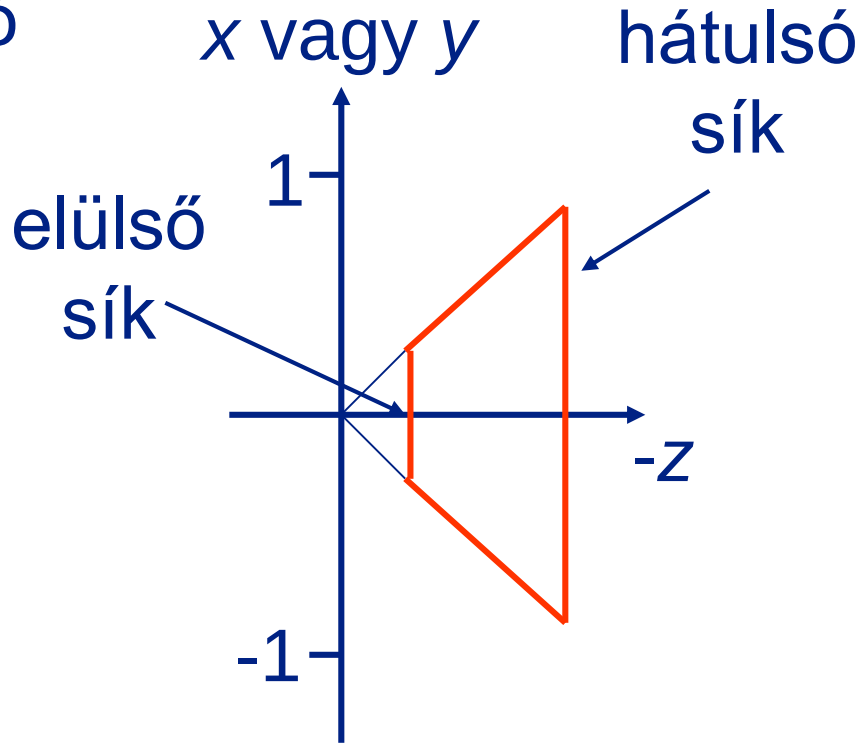
Kanonikus látóterek síkjai:

$x = -1,$	$x = 1$	$x = z,$	$x = -z$
$y = -1,$	$y = 1$	$y = z,$	$y = -z$
$z = 0,$	$z = -1$	$z = -z_{min},$	$z = -1$

3D megjelenítés implementálása /3



párhuzamos vetítésnél

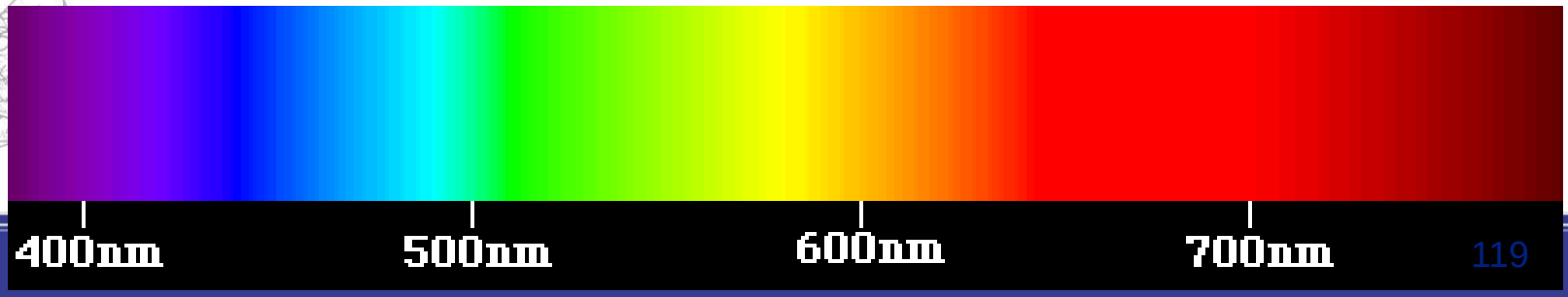


perspektív vetítésnél

5. SZÍNMODELLEK

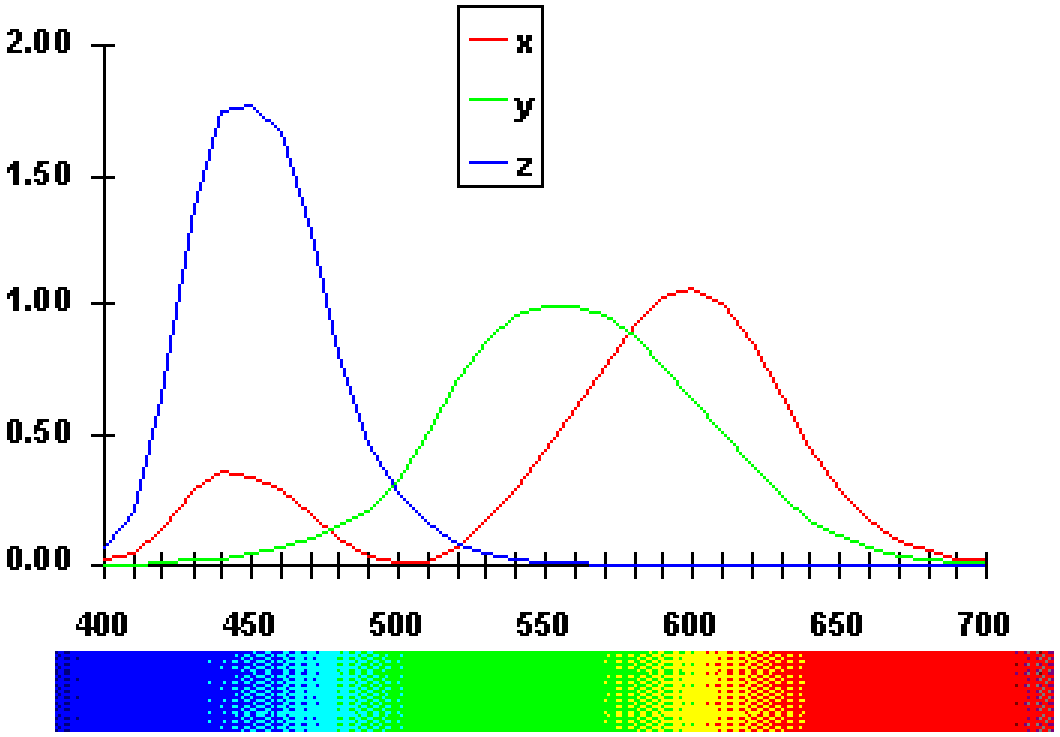
A látható színek összehasonlítása

Név	Alias	Jel	Hullámhossz	Szín	Arány
rho	vörös	r	590 nm	sárga-narancs	30%
gamma	zöld	g	550 nm	sárgás zöld	60%
beta	kék	b	440 nm	kék ibolya	10%

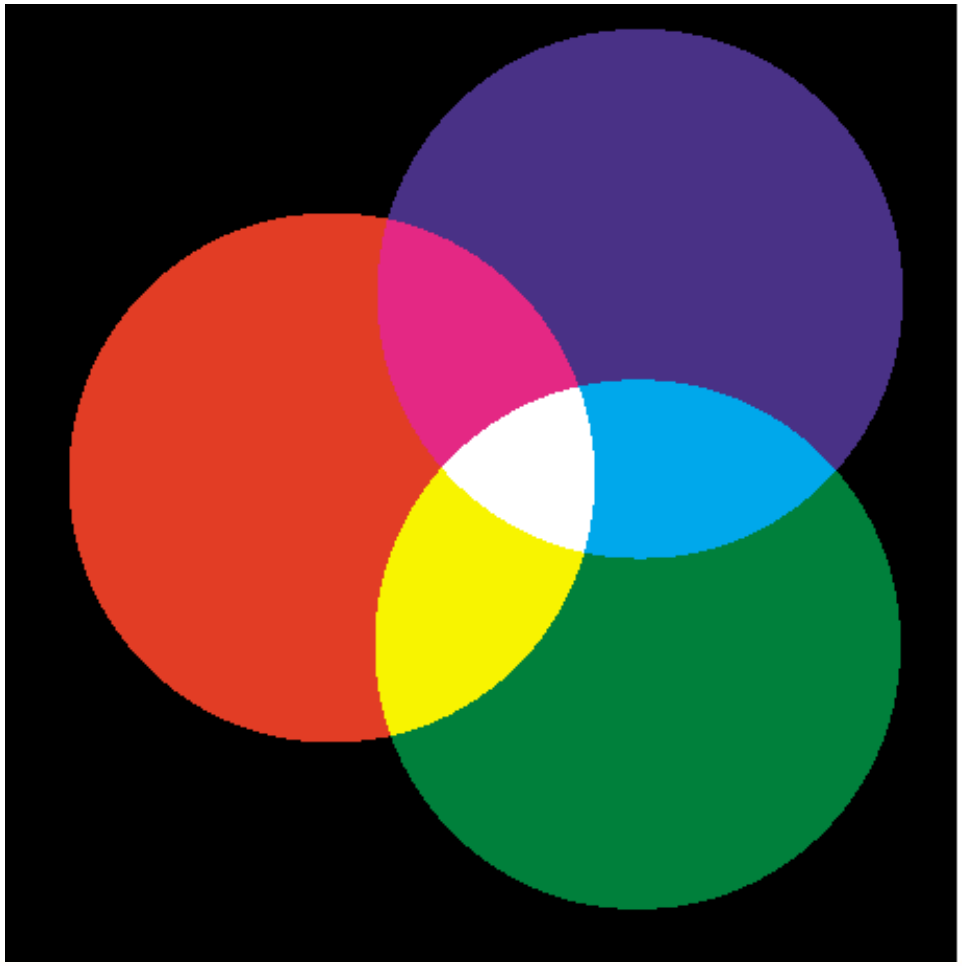


CIE 1931 - 2° standard observer

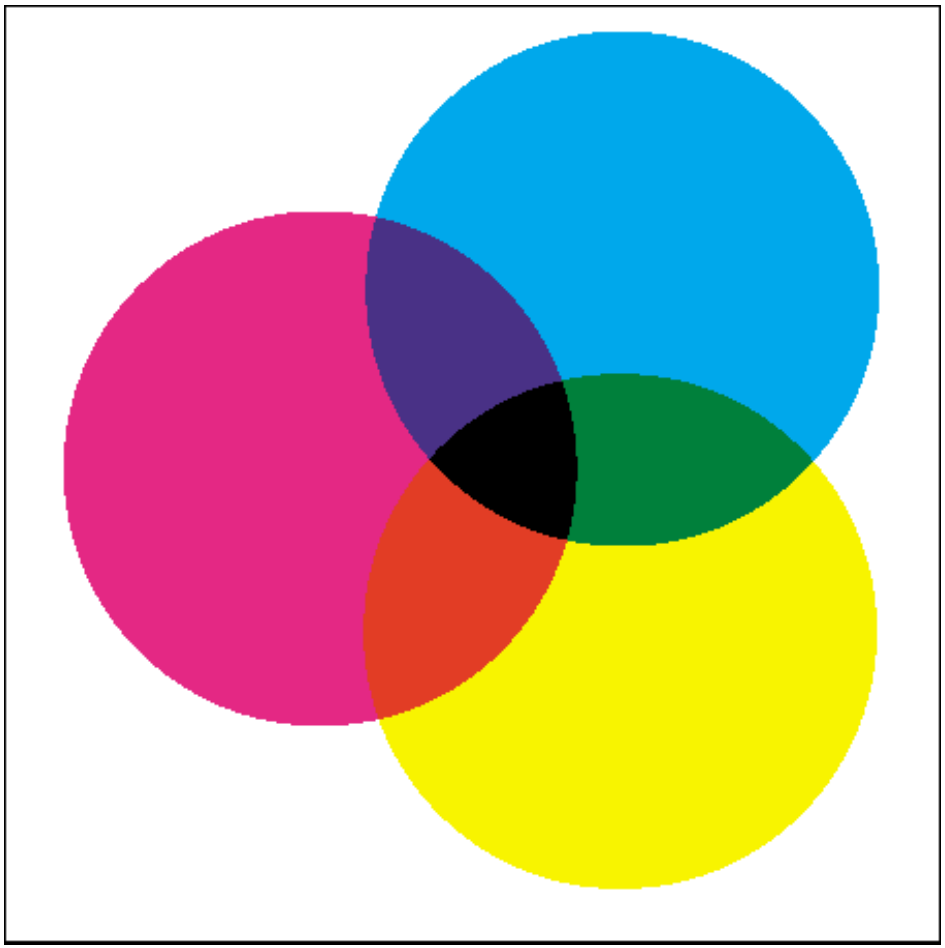
Tristimulus values of the spectral colours



Összeadó színkeverés

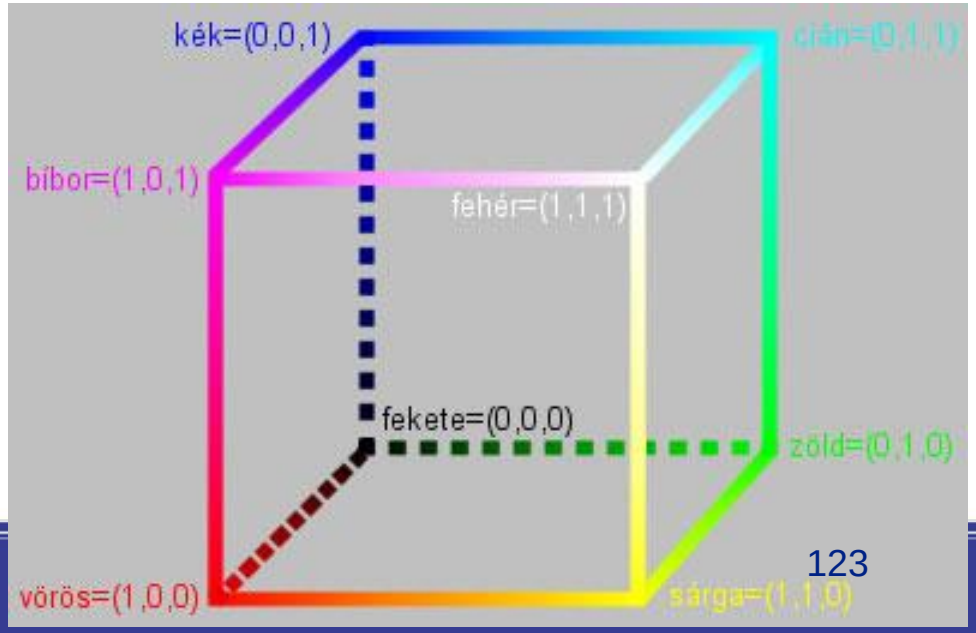


Kivonó színkeverés



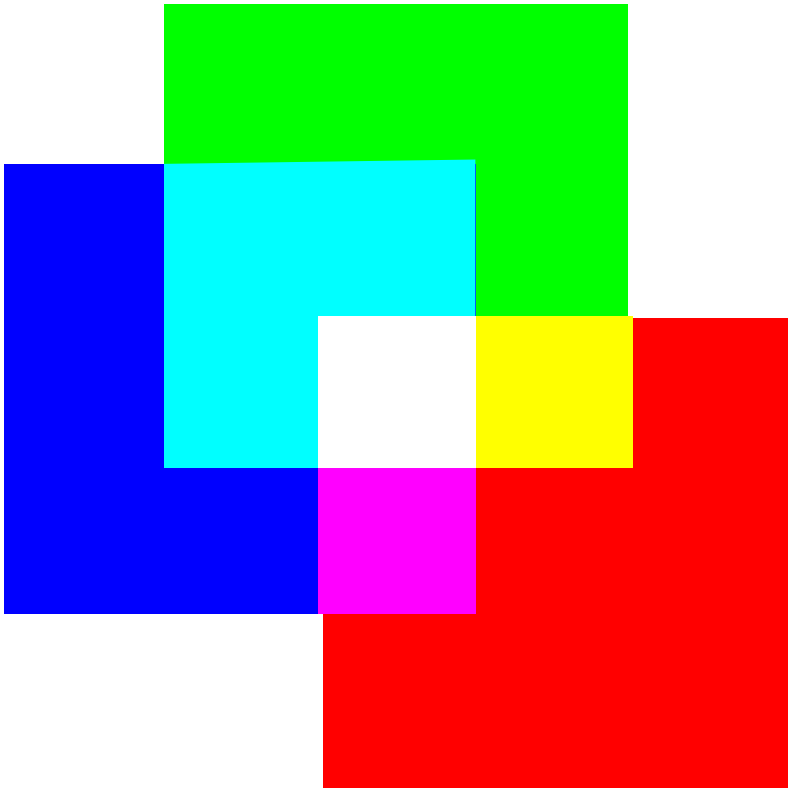
RGB (Red, Green, Blue – vörös, zöld, kék)

- **Additív** komponensek: Alkalmas súlyokkal vett összegük ad egy összetett színt
- Pl. színes képernyőnél



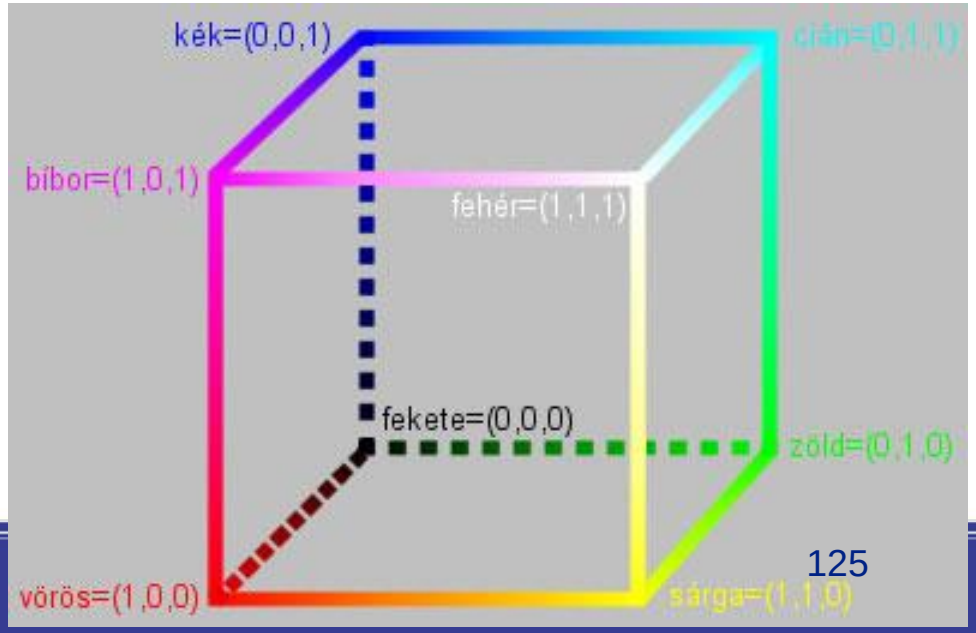
RGB (Red, Green, Blue – vörös, zöld, kék)

- Vörös
- Zöld
- Kék
- Cián
- Sárga
- Bíbor



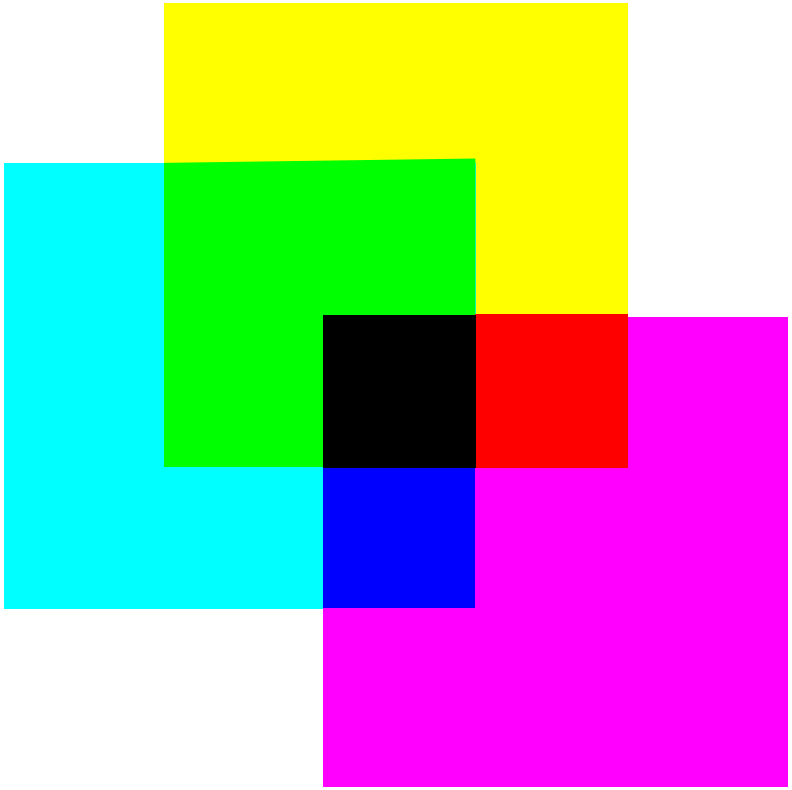
CMY (Cyan, Magenta, Yellow – türkiz, bíbor, sárga)

- **Szubtraktív** komponensek: Sokszor szűrőként használjuk, hogy kiszűrjék a fehérből a megfelelő színt
- Pl. színes nyomtatóknál



CMY (Cyan, Magenta, Yellow – türkiz, bíbor, sárga)

- Türkiz
- Bíbor
- Sárga
- Vörös
- Zöld
- Kék
- Fekete



CMY (Cyan, Magenta, Yellow – türkiz, bíbor, sárga)

Konverzió:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

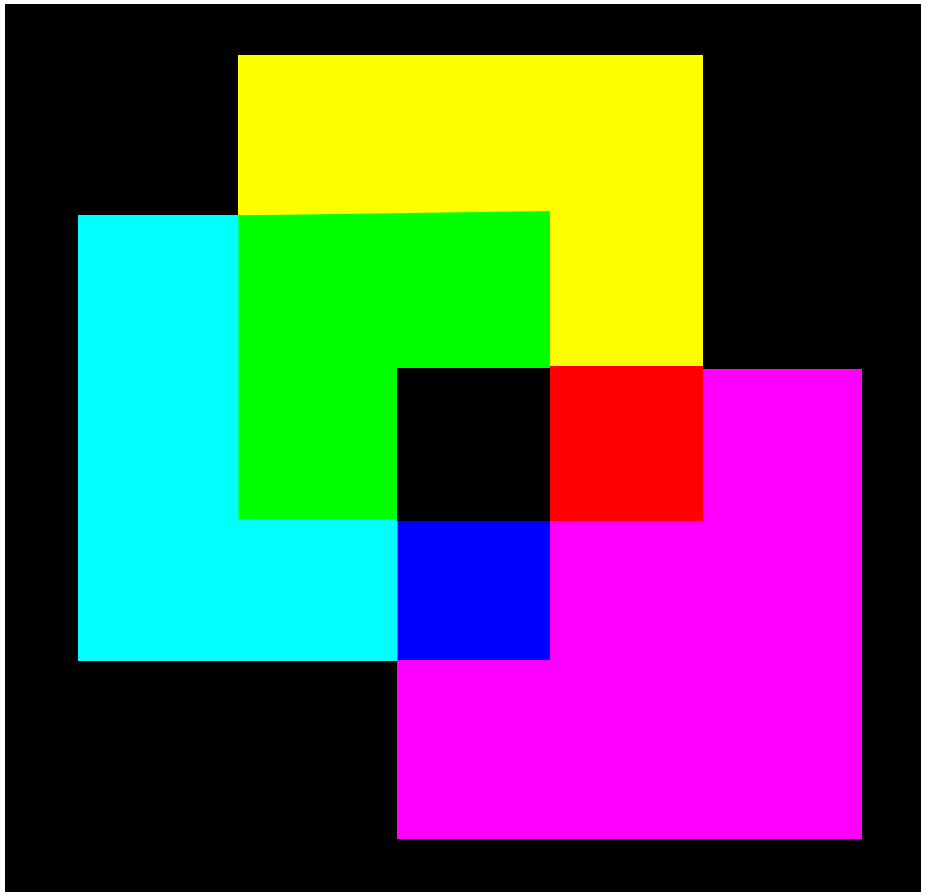
RGB → CMY

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

RGB ← CMY

CMYK (Cyan,Magenta,Yellow, BlacK - Key : fekete)

- Türkiz
- Bíbor
- Sárga
- Vörös
- Zöld
- Kék
- Fekete



CMYK (Cyan,Magenta,Yellow, BlacK - Key : fekete)

CMY → CMYK konverzió:

$K = \min \{C, M, Y\}$

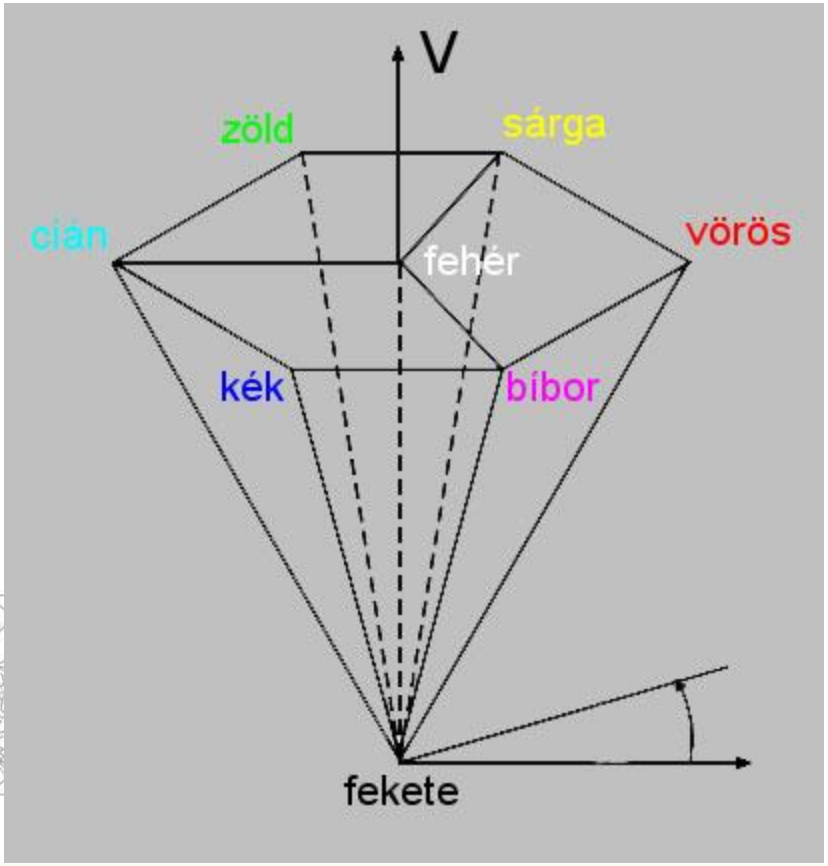
$C = C - K$

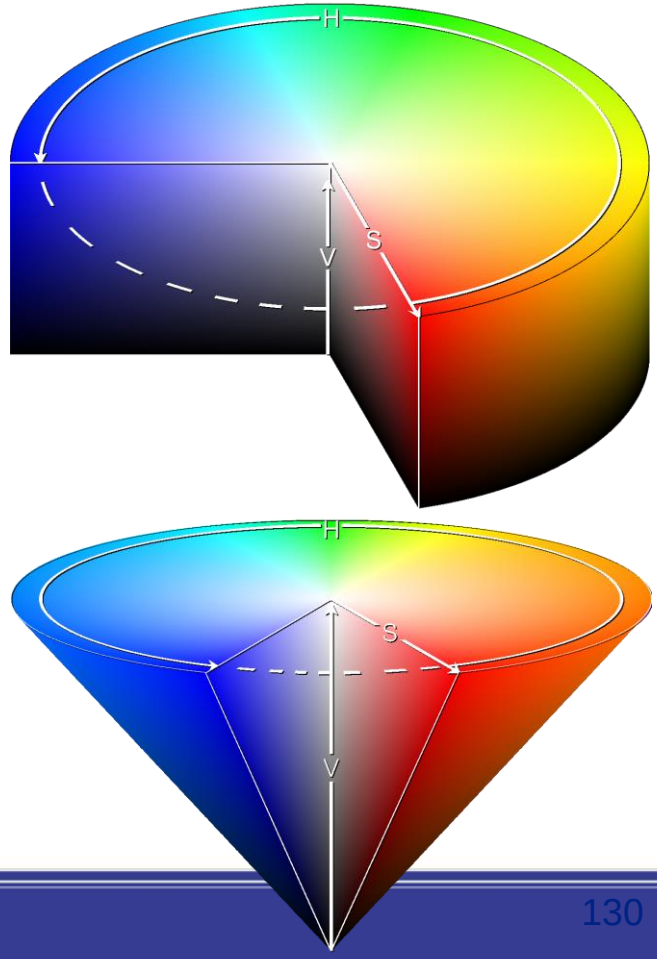
$M = M - K$

$Y = Y - K$

HSV (Hue,Saturation,Value – árnyalat,telítettség,fényesség)

Esztétikai alapú (ahogy a festők keverik a színeket)





6. ÁRNYALÁS, A MEGVILÁGÍTÁS MODELLJEI

Világító tárgyak

- Minden tárgynak saját intenzitású fénye van
- A megvilágítás egyenlete:

$$I = k_i$$

- k_i - a tárgy saját fényének az intenzitása
- független a pont helyzetétől

Környezeti (szórt - ambient) fény

- Minden irányból egyenletes
- A megvilágítás egyenlete:

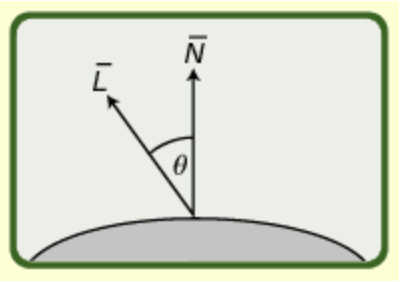
$$I = I_a k_a$$

- I_a : környezeti fény intenzitása
- k_a: a környezeti fény visszaverődési együtthatója (anyagtól függ), $0 \leq k_a \leq 1$

Diffúz (diffuse) visszaverődés

- Lambert-féle visszaverődés
- Minden irányban ugyanannyi fényt ver vissza
- A felület fényessége (I) függ a fényforrás iránya (L) és a felület normálisa (N) közötti szögtől:
- N, L egységvektorok

$$I = I_p \, k_d \cos \Theta = I_p \, k_d \, (N \, L)$$



- I_p: a pontforrás intenzitása
- k_d: a szórt visszaverődés együtthatója (anyagtól függ), $0 \leq k_d \leq 1$

Környezeti fény és diffúz visszaverődés együtt

$$I = I_a k_a + I_p k_d (N L)$$

- Ha a fényforrás és a tárgy közötti távolságot (d_L) is figyelembe vesszük, akkor:

$$I = I_a k_a + I_p / d_L^2 k_d (N L)$$

$$\smile = f_{att}$$

- (fényelnyelődési faktor)

Színes fény és felületek

- Színes fény és felületek esetén a komponensekre:

$$I_R = I_{aR} k_a O_{dR} + f_{att} I_{pR} k_d O_{dR} (N L)$$

$$I_G = ...$$

$$I_B = ...$$

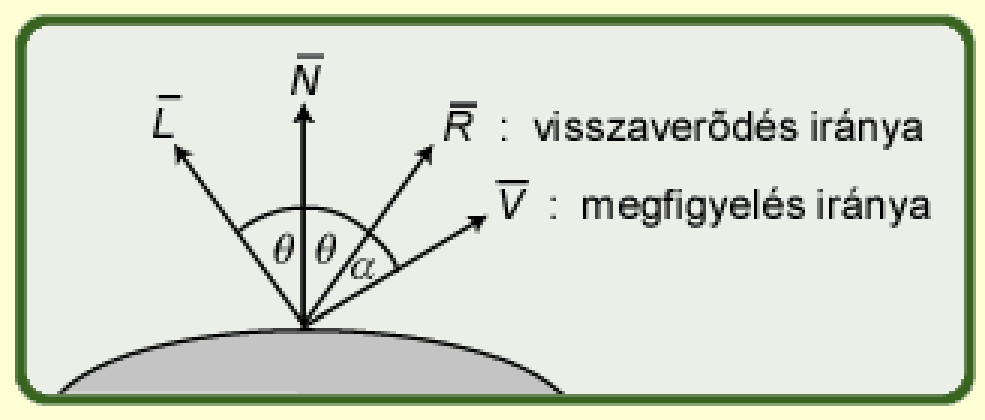
- O_{dR} a tárgy szórt vörös komponense
- I_{pR} a megvilágítás vörös komponens
- $k_d O_{dR}$ visszaverődési hányad komponense
- ...
- Általában:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} k_d O_{d\lambda} (N L)$$

- ahol λ a hullámhossz

Tükröző (specular) visszaverődés

- Fényes felületekről (tükörről)



Tükröző (specular) visszaverődés

- Phong-féle modell:

$$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + f_{att}\, I_{p\lambda} \left[k_d\, O_{d\lambda} \cos\theta + W(\theta) \cos^n\alpha \right]$$

- n: a tükrözési visszaverődés kitevője (csillogás)
- (tompa) $1 \le n \le 1000$ (éles fény)
- W(θ): a tükrözötten visszaverődő fény hányada, lehet konstans, k_s ($0 \le k_s \le 1$)

Tükröző (specular) visszaverődés

- A tárgy anyagát is figyelembe véve:

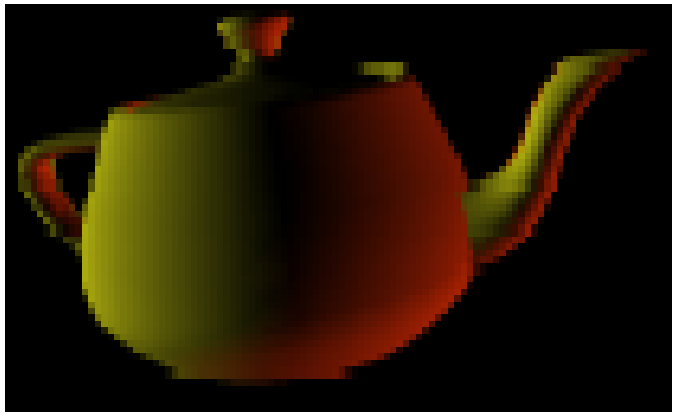
$$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + f_{att}\, I_{p\lambda} [\, k_d\, O_{d\lambda}\, (N\ L) + k_s\, O_{s\lambda}\, (R\ V)^n]$$

- Több fényforrásra:

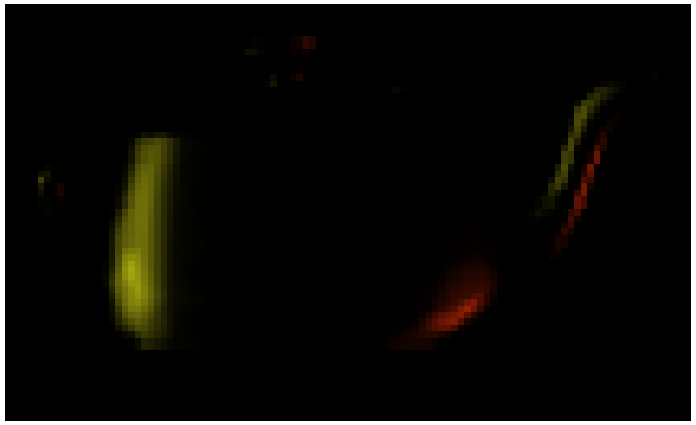
$$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + \sum f_{att}\, I_{p\lambda} [\, k_d\, O_{d\lambda}\, (N\ L) + k_s\, O_{s\lambda}\, (R\ V)^n]$$

Megvilágítási modellek (példa)

szórt



diffúz



tükröző, csillogás = 20

szórt + diffúz + tükröző

Poligonokból álló felületek fényességének meghatározása

- Minden pontban kiszámítjuk a megvilágítási egyenlet szerinti intenzitást (nagyon drága módszer)

Konstans fényesség

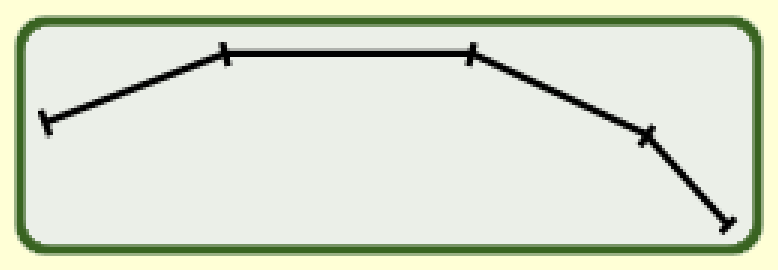
- Az egész poligon ugyanolyan intenzitású
- Jó, ha:
  - végtelen távoli fényforrás (N L konstans)
  - végtelen távoli megfigyelő (R V konstans)
  - poligon oldalú felület

Interpolált fényesség

- Az intenzitást a csúcsokban számított intenzitásból kapjuk interpolációval

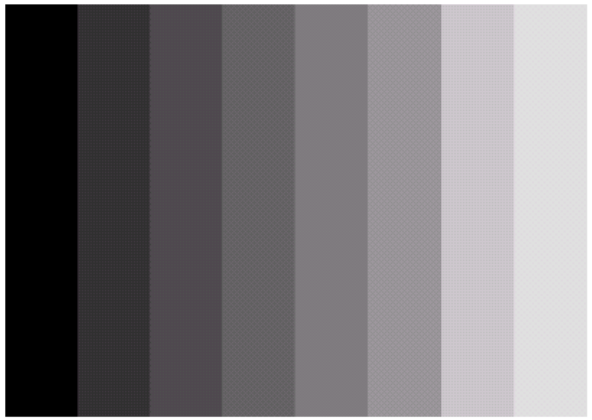
Poligon-hálózat fényessége

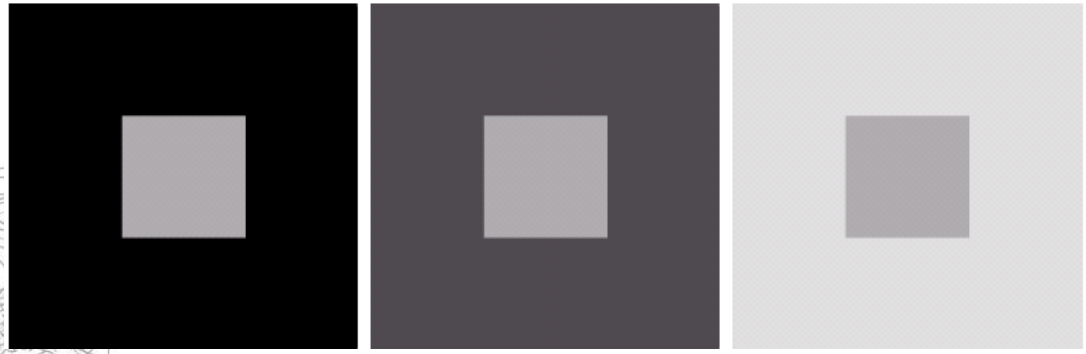
- Az egyes poligonok konstans fényessége csak kiemelné a poligonok közötti éleket

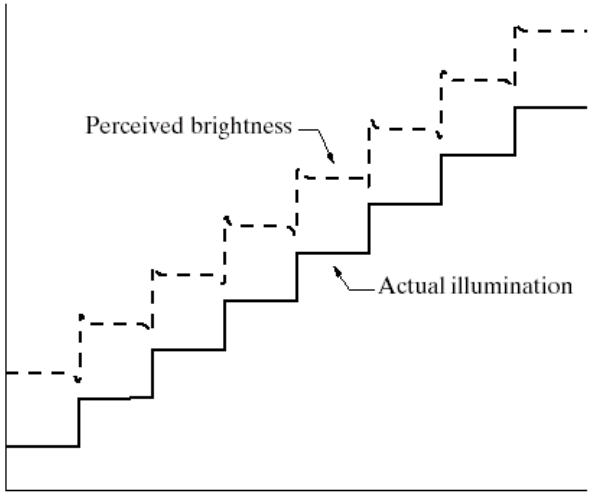


Mach-féle hatás

- Az intenzitás változását eltúlozva érezzük ott, ahol az intenzitás folytonossága megszűnik

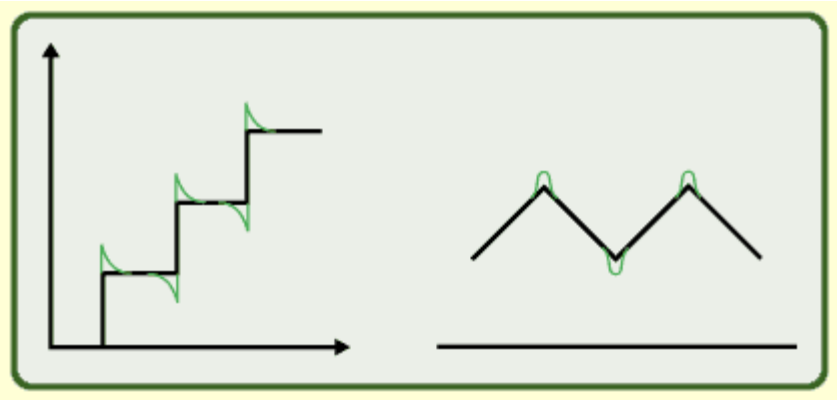






Poligon-hálózat fényessége

- Mach-hatás

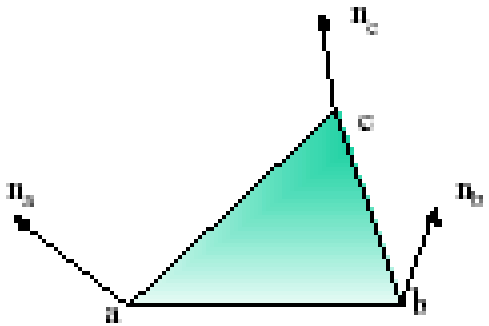


- Megoldás:
  - minden poligon fényességét változó intenzitásúnak generáljuk

Gouraud-féle fényesség

1. A poligonok normálisait ismerve határozzuk meg a csúcspontok normálisait (pl. az ott érintkező poligonok normálisainak átlagaként)
2. Számoljuk ki az intenzitásokat a csúcspontokban (itt a normális már ismert)
3. Az élek mentén lineáris interpolációval számoljuk az intenzitást
4. Az élek között (a pásztázó vonalak mentén) lineáris interpolációval számoljuk az intenzitást

Gouraud-féle fényesség

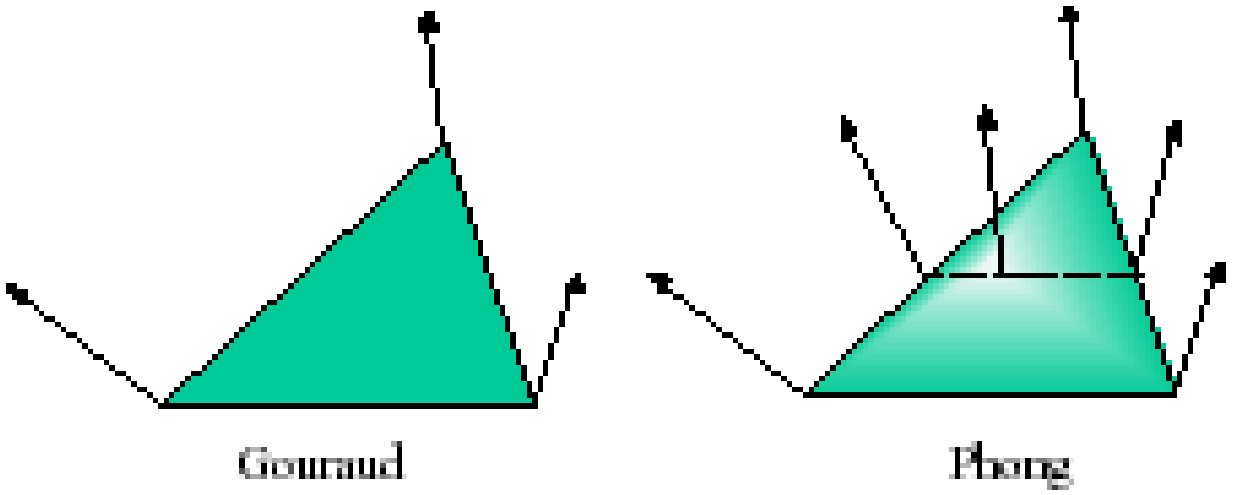


Phong-féle fényesség

1. A normálvektorokat számoljuk ki a csúcspontokban
2. Interpolációval a csúcspontok között az élek mentén a normálvektorokat
3. Interpolációval az élek között
4. Intenzitás számítása

Sokszor jobb, mint a Gouraud-féle módszer

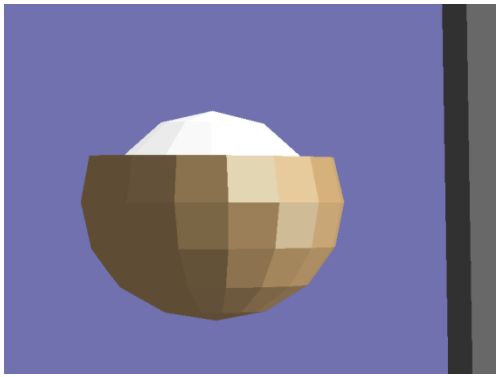
Gouraud v. Phong-féle fényesség



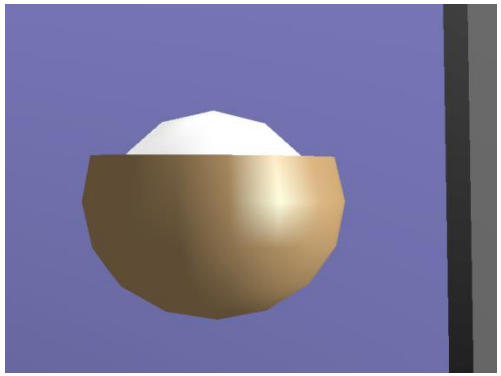
Phong- és Gouraud-féle fényesség

Konstans Gouraud Phong

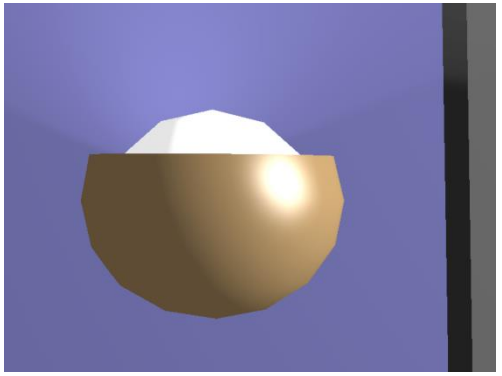
Példák



Konstans



Gouraud



Phong

Példák

szórt

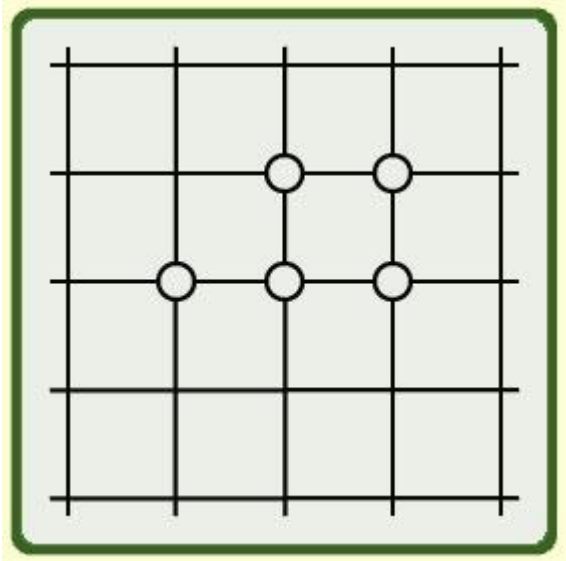
Phong (szórt + diffúz)

Phong (szórt + diffúz + tükröző)

7. VONALAS PRIMITÍVEK RASZTERIZÁLÁSA

Algoritmusok raszteres grafikához

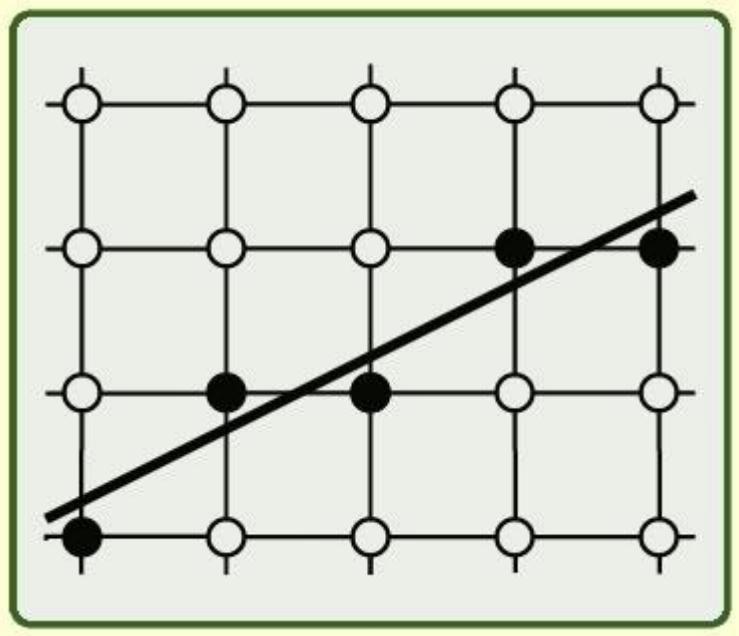
- Feladat
  - grafikai primitíveket (pl. vonalat, síkidomot) ábrázolni kép-mátrixszal, meghatározni azokat a képpontokat, amelyek a primitív pontjai, vagy közel vannak a primitívhez
- Modell
  - képpont (= körlap), amely a négyzetháló csúcspontjaiban helyezhető el
  - a koordináták egész számok



Egyenes rajzolása

- Tegyük fel, hogy "vékony" egyenes:

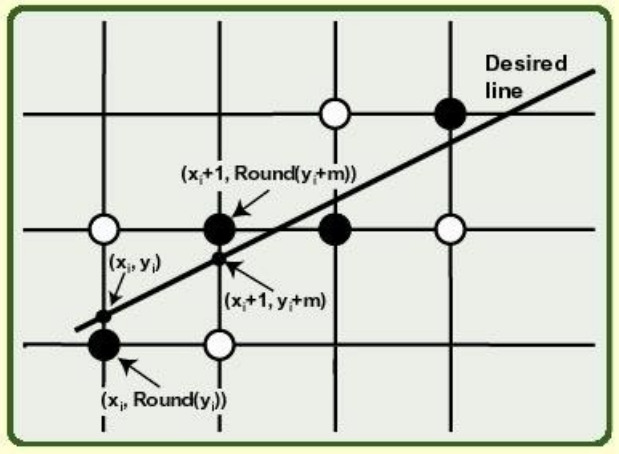
  $y = mx+b$
- A meredeksége: $0 < m < 1$
- ($m = 0,1,...$ triviális speciális esetek)
- más esetekben visszavezetjük az $0 < m < 1$ esetre
- Legyen: $x_0 < x_1$, $y_0 < y_1$



Alap inkrementális algoritmus

- Haladjunk $\Delta x = 1$ növekménnyel balról jobbra, válasszuk a legközelebbi képpontot:

$$(x_i, [y_i+0.5]) = (x_i, [mx_i+b+0.5])$$



- A szorzás kiküszöbölhető inkrementálással:

$$y_{i+1}=mx_{i+1}+b=m(x_i+\Delta x)+b=y_i+m\cdot\Delta x=y_i+m$$

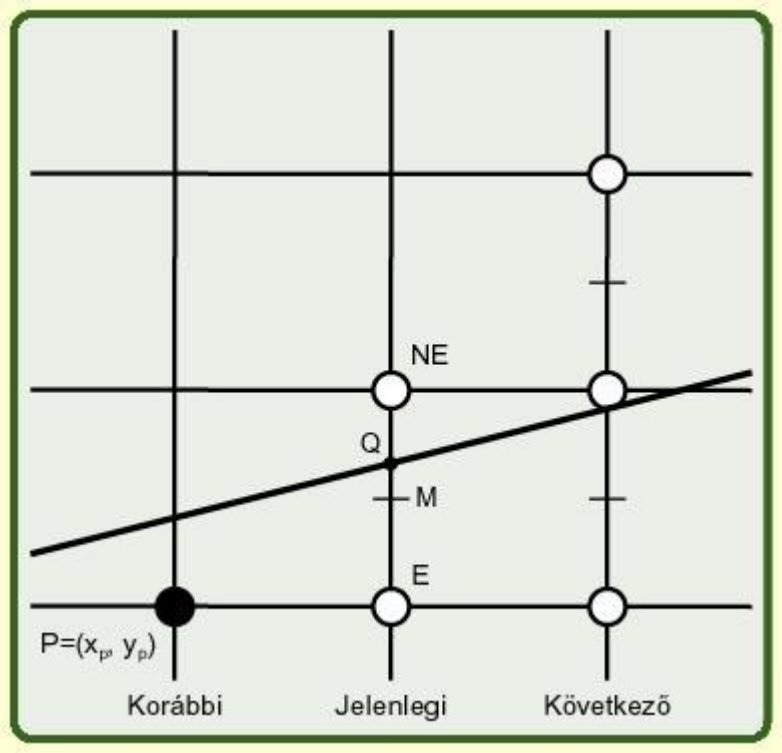
Program

```
(ha |m|>1, akkor x-et cseréljük y-nal)

procedure Line(x0,y0, x1,y1, value: integer);
var x : integer;
   dy,dx,y,m : real;
begin
   dy := y1-y0; dx := x1-x0;
   m := dy/dx; y := y0;
   for x := x0 to x1 do
       begin
         WritePixel(x, Round(y), value);
         y := y+m
       end
end; {Line}
```

Felezőpont algoritmus egyenesre

- Egész aritmetika elegendő (Bresenham)
- Elv: Azt a képpontot válasszuk NE és E közül, amelyik közelebb van a Q metszésponthoz
- Másképp: az döntsön a választásban, hogy Q az M felezőpont melyik oldalán van
- Tegyük fel, hogy: $x_0 < x_1$, $y_0 < y_1$



Felezőpont algoritmus egyenesre

- Az (x_0, y_0) és (x_1, y_1) ponton átmenő egyenes egyenlete:

$$(x - x_0) / (y - y_0) = (x_1 - x_0) / (y_1 - y_0)$$

- Legyen

$$dx = x_1 - x_0 \; (> 0),$$
$$dy = y_1 - y_0 \; (> 0),$$

- akkor:

$$x\,dy - y\,dx + y_0\,dx - x_0\,dy = 0$$

- Legyen:

$$F(x,y) = x\,dy - y\,dx + y_0\,dx - x_0\,dy$$

$$F(x,y) \begin{cases} > 0, \text{ ha az egyenes } (x, y) \text{ fölött fut,} \\ = 0, \text{ ha } (x, y) \text{ az egyenesen van,} \\ < 0, \text{ ha az egyenes } (x, y) \text{ alatt fut.} \end{cases}$$

Felezőpont kritérium

$d = F(M) = F(x_p+1, y_p+½)$ (d : döntési változó)

az egyenes	választás:
> 0, M fölött,	***NE***
$= 0$, M-en át,	***NE*** vagy ***E***
< 0, M alatt fut	***E***

A következő pontnál:

ha ***E***-t választottuk, akkor

$$\Delta_E = d_{új} - d_{régi} = F(x_p+2, y_p+½) - F(x_p+1, y_p+½) = dy,$$

ha ***NE***-t választottuk, akkor

$$\Delta_{NE} = F(x_p+2, y_p+3/2) - F(x_p+1, y_p+½) = dy - dx$$

Felezőpont algoritmus egyenesre

- Kezdés:

$$d_{start} = F(x_0+1, y_0+½) = F(x_0, y_0)+dy - dx/2$$
$$= dy - dx/2$$

- Azért, hogy egész aritmetikával számolhassunk, használjuk inkább az alábbi függvényt

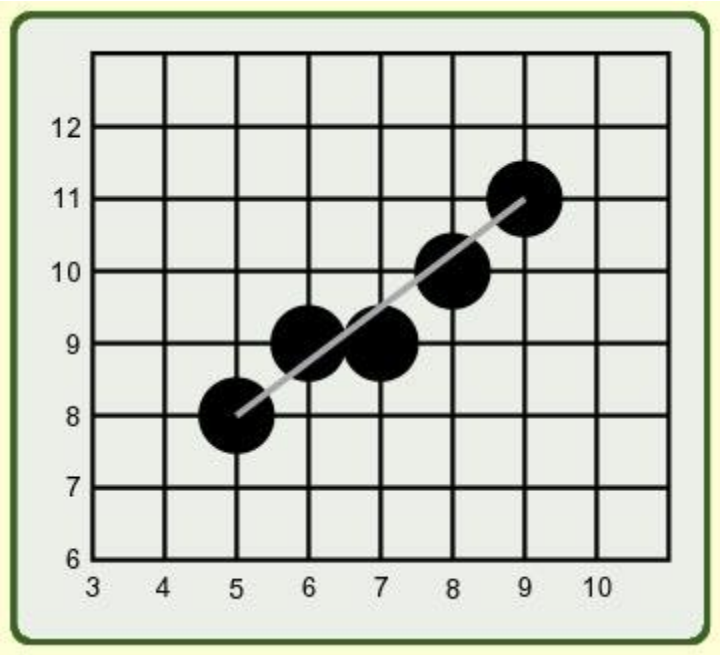
$$F(x,y) = 2\cdot(x\cdot dy - y\cdot dx + y_0\cdot dx - x_0\cdot dy)$$

Program

```
procedure MidpointLine (x0,y0,x1,y1,value: integer);
var dx,dy,incrE,incrNE,d,x,y : integer;
begin
  dx := x1-x0; dy := y1-y0; d := 2*dy-dx;
  incrE := 2*dy; incrNE := 2*(dy-dx);
  x := x0; y := y0;
  WritePixel(x,y,value);
  while x < x1 do begin
    if d <= 0 then begin
      x := x + 1; d := d + incrE;
    end
    else begin
      x := x + 1; y := y + 1; d := d + incrNE;
    end;
    WritePixel(x,y,value)
  end {while}
end; {MidpointLine}
```

Tulajdonságok

- csak összeadás és kivonás
- általánosítható körre, ellipszisre

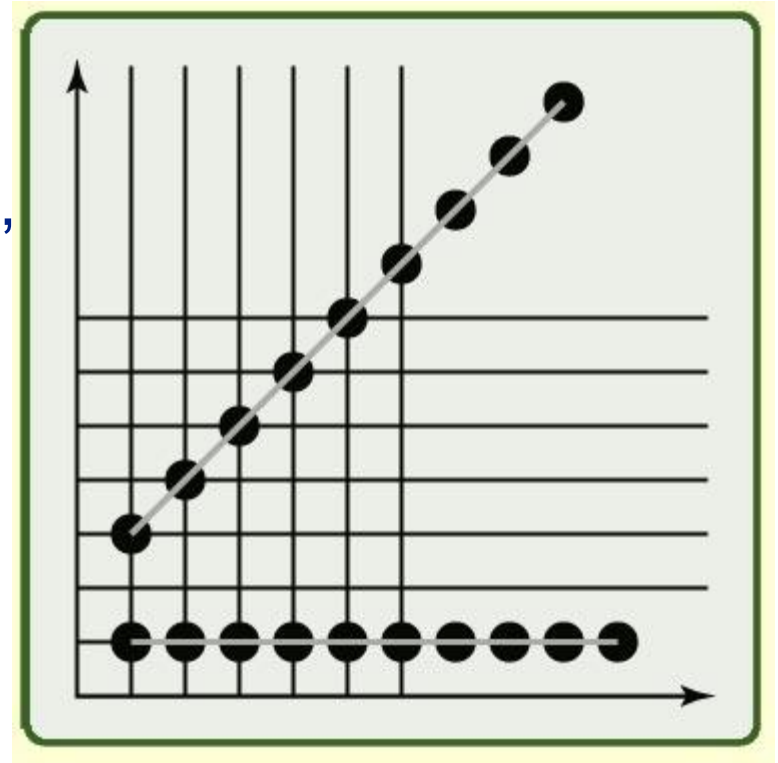


1. probléma

- Különböző pontsorozat lesz az eredmény ha balról jobbra, vagy ha jobbról balra haladunk.
  - Legyen a választás:
    - balról jobbra: d = 0 → E-t választani
    - jobbról balra: d = 0 → SW-t választani
  - Megjegyzés:
    - Nem mindig lehet csak balról jobbra haladva rajzolni az egyeneseket.
      Pl. szaggatott vonallal rajzolt zárt poligon

2. probléma

- A vonal pontjainak a sűrűsége függ a meredekségétől
  - Megoldás:
    - intenzitás változtatása,
    - kitöltött téglalapnak tekinteni az egyenes pontjait



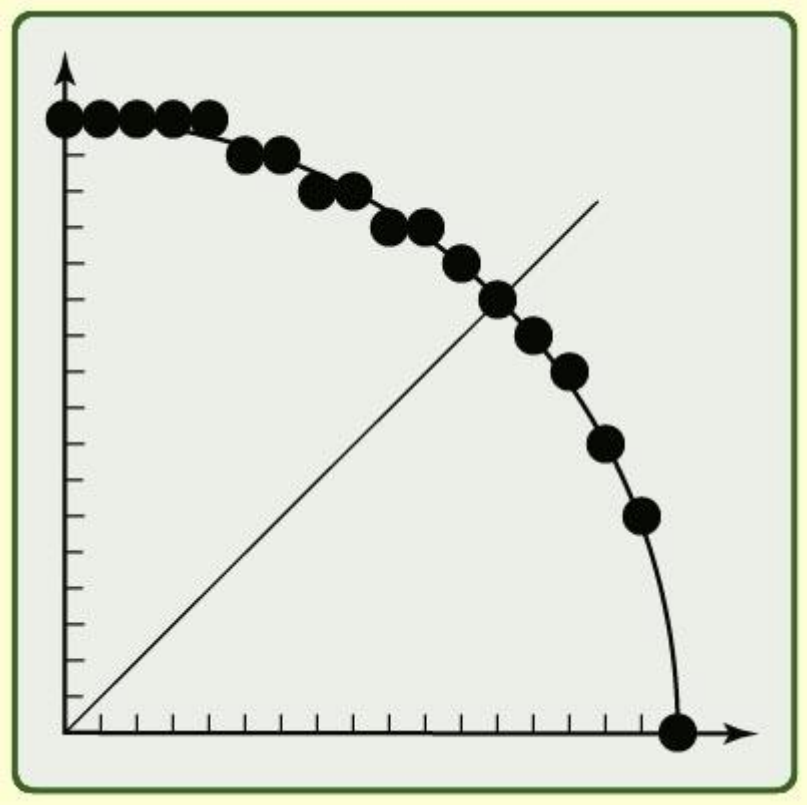
Kör rajzolása

$x^2+y^2 = R^2$ R : egész

- Elég egy kör-negyedet megrajzolni (a többi rész a szimmetria alapján transzformációkkal - pl. tükrözés - előáll)

x 0-tól R-ig növekszik,

$y = \sqrt{R^2 - x^2}$



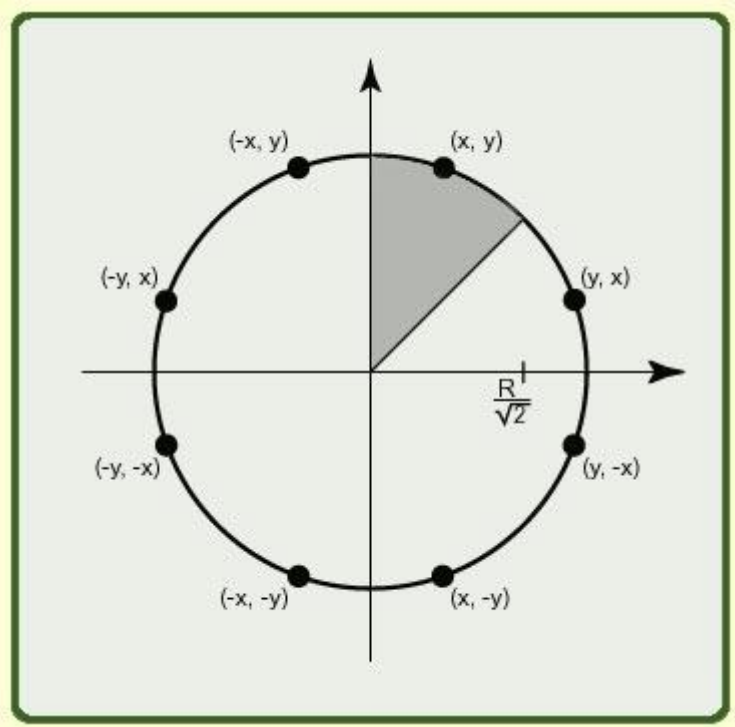
- Drága eljárás (szorzás, gyökvonás)
- Nem egyenletes

Polárkoordinátás alak

- Elég egy nyolcad kört kiszámítani:

$$x = R \cdot \cos \Theta$$
$$y = R \cdot \sin \Theta$$

Θ 0°-tól 90°-ig növekszik



- Drága eljárás (sin, cos)

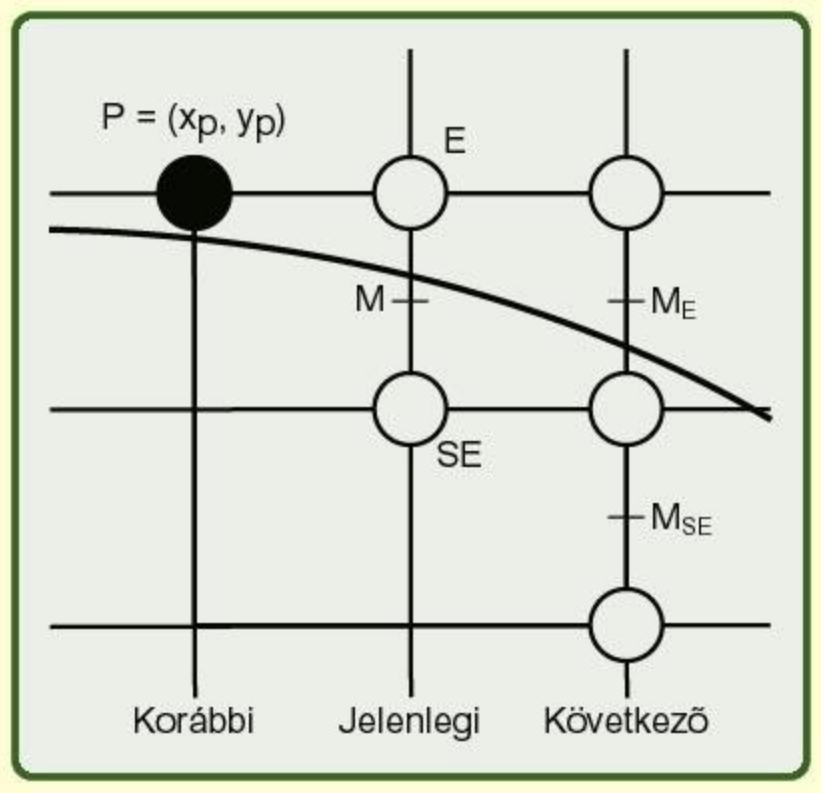
Program

```
procedure Circlepoints(x, y, value:
  integer);
begin
  WritePixel ( x,  y, value);
  WritePixel ( y,  x, value);
  WritePixel ( y, -x, value);
  WritePixel ( x, -y, value);
  WritePixel (-x, -y, value);
  WritePixel (-y, -x, value);
  WritePixel (-y,  x, value);
  WritePixel (-x,  y, value);
end; {CirclePoints}
```

Felezőpont algoritmus körre

x 0-tól $R/\sqrt{2}$ -ig
(amíg x ≤ y)

- **Elv:**
  - ***E*** és ***SE*** közül azt a pontot választjuk, amelyikhez a körív metszéspontja közelebb van



Felezőpont algoritmus körre

$$F(x,y) = x^2+y^2 - R^2 \begin{cases} > 0, \text{ ha } (x,y) \text{ kívül van} \\ = 0, \text{ ha } (x,y) \text{ rajta van} \\ < 0, \text{ ha } (x,y) \text{ belül van} \end{cases}$$

$$d = F(M) =$$

$$F(x_p+1,\ y_p - ½) \begin{cases} >0 \rightarrow \textbf{\textit{SE}}\text{-t választani} \\ =0 \rightarrow \underline{\textbf{\textit{SE}}} \text{ vagy } \textbf{\textit{E}} \\ <0 \rightarrow \textbf{\textit{E}}\text{-t választani} \end{cases}$$

Felezőpont algoritmus körre

- A következő pontnál:
  - ha E-t választottuk, akkor

$$\Delta_E = d_{új} - d_{régi} =$$
$$F(x_p+2, y_p-½) - F(x_p+1, y_p-½) =$$
$$= 2x_p+3$$

  - ha SE-t választottuk, akkor

$$\Delta_{SE} = F(x_p+2, y_p-3/2) - F(x_p+1, y_p-½) =$$
$$= 2x_p-2y_p+5$$

Felezőpont algoritmus körre

- Az iterációs lépések:
  - a döntési változó előjele alapján kiválasztjuk a következő képpontot

$$d = d + \Delta_{SE} \text{ vagy } d + \Delta_{E}$$

(a választástól függően)

- Figyeljük meg: d értéke egész számmal változik!
- Kezdés:

kezdőpont: (0, R)

felezőpont: (1, R – 1/2)

d = F(1, R – 1/2) = 5/4 – R

Program

```
procedure MidpointCircle (radius, value : integer);
var x, y : integer; d: real;
begin
  x := 0; y := radius; d := 5/4-radius;
  CirclePoints (x,y,value);
  while y>x do begin
    if d<0 then begin
      x := x+1;
      d := d+2*x+3;
    end
    else begin
      x := x+1; y := y-1;
      d := d+2*(x-y)+5;
    end;
    CirclePoints (x,y,value)
  end {while}
end; {MidpointCircle}
```

Felezőpont algoritmus körre

- Nem egész aritmetika, ezért legyen h új döntési változó:

$$h = d - \frac{1}{4}$$

$$h + \frac{1}{4} = d < 0$$

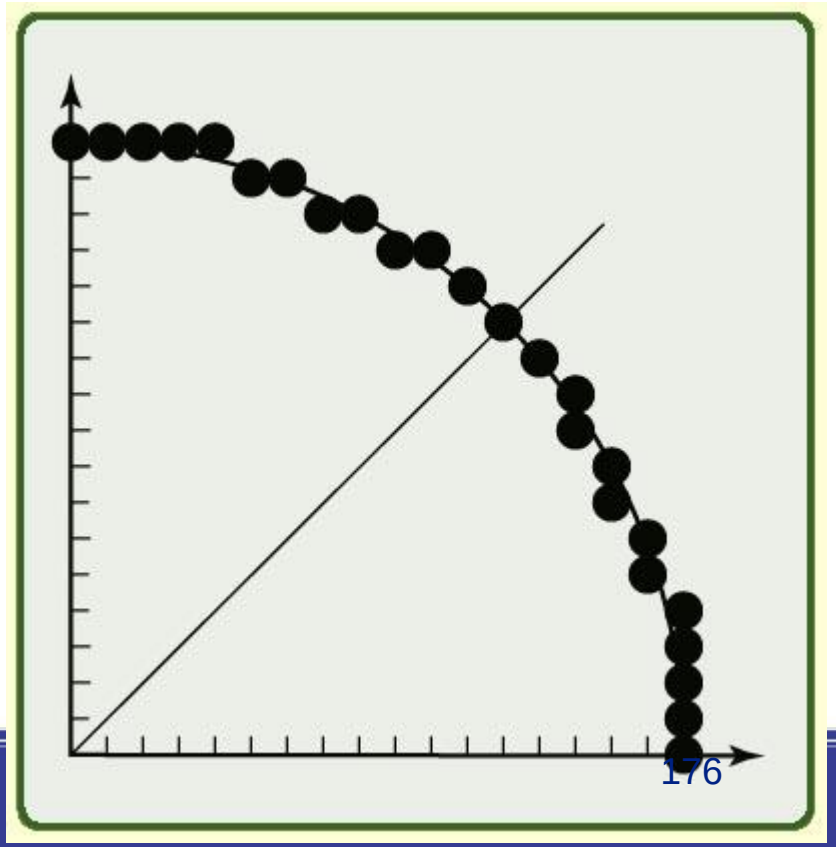
- Ekkor kezdéskor

$$h = 1 - R$$

- Kezdetben és a későbbiek során is h egész szám!
- Igaz, hogy $d < 0$ helyett $h < -\frac{1}{4}$ -et kellene vizsgálni, de ez h egész volta miatt ekvivalens $h < 0$ -val, tehát egész aritmetika használható.

Program

```
procedure MidpointCircle (radius, value : integer);
var x,y,d : integer;
begin
  x := 0; y := radius; h := 1-radius;
  CirclePoints (x,y,value);
  while y>x do begin
    if h<0 then begin
      x := x+1;
      h := h+2*x+3;
    end
    else begin
      x := x+1; y := y-1;
      h := h+2*(x-y)+5;
    end;
    CirclePoints (x,y,value)
  end {while}
end; {MidpointCircle}
```



Másodrendű differenciák

- Stratégia:
  - egy függvény értéke becsülhető két pontban felvett értékei különbségét használva (ami polinomok esetében alacsonyabb fokszámú lesz)
  - használjunk első- és másodrendű parciális differenciákat

Kör rajzolása

- tudjuk, hogy

$$\Delta_E(x_p,y_p) = 2\cdot x_p+3$$

$$\Delta_{SE}(x_p,y_p) = 2\cdot x_p-2\cdot y_p+5,$$

- ha E-t választottuk $(x_p,y_p) \rightarrow (x_p+1,y_p)$,

$$\Delta_{E\ új} - \Delta_{E\ régi} = 2, \quad \Delta_{SE\ új} - \Delta_{SE\ régi} = 2$$

- ha SE-t választottuk $(x_p,y_p) \rightarrow (x_p+1,y_p-1)$

$$\Delta_{E\ új} - \Delta_{E\ régi} = 2, \quad \Delta_{SE\ új} - \Delta_{SE\ régi} = 4$$

Kör rajzolása

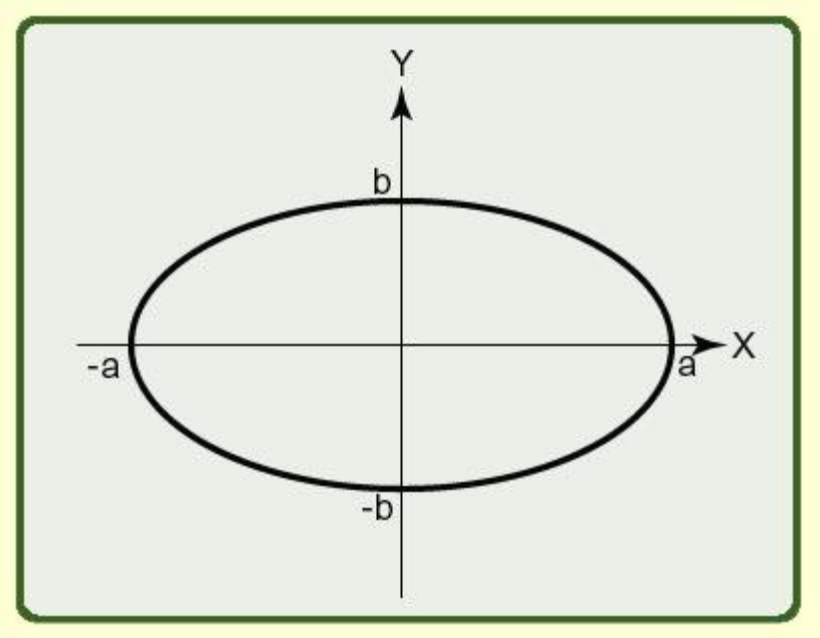
- tehát az iterációs lépések:
  - d előjele alapján a következő képpont kiválasztása
  - d = d+Δ (a választástól függően Δ_E vagy Δ_{SE})
  - Δ_E és Δ_{SE} módosítása (a választástól függően)

Program

```
procedure MidpointCircle (radius,value : integer);
var  x,y,d,deltaE,deltaSE : integer;
begin
  x := 0; y := radius; d := 1-radius;
  deltaE := 3; deltaSE := -2*radius+5;
  CirclePoints(x,y,value);
  while y>x do begin
    if d<0 then begin
      d := d+deltaE;
      deltaE:=deltaE+2; deltaSE:=deltaSE+2;
      x := x+1
    end
    else begin
      d := d+deltaSE;
      deltaE:=deltaE+2; deltaSE:=deltaSE+4;
      x := x+1; y := y-1
    end;
    CirclePoints(x,y,value)
  end {while}
end; {MidpointCircle}
```

Ellipszis rajzolása

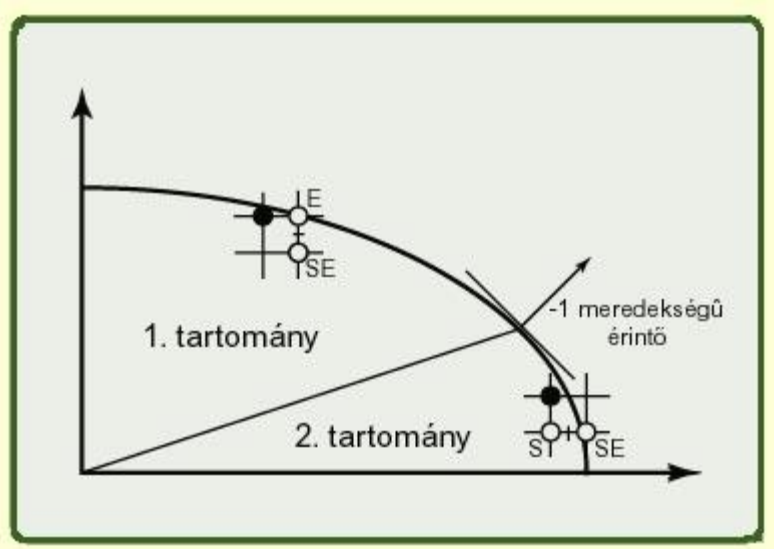
- F(x,y) = $b^2x^2 + a^2y^2 – a^2b^2$
- a,b egész
- Szimmetria miatt: elég az első síknegyedben megrajzolni



Ellipszis rajzolása

- Da Silva algoritmusa (felezőpont módszer)
- Bontsuk a negyedet két tartományra
- Az 1. tartományban:

$a^2 (y_p - ½) > b^2 (x_p+1)$



Da Silva algoritmusa

Az 1. tartományban:

$$d_1 = F(x_p+1, y_p - ½)$$

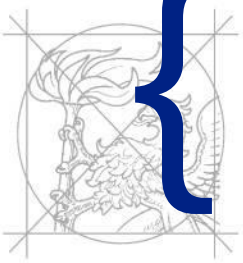
- ≥ 0 — ***SE***-t választjuk
- < 0 — ***E***-t választjuk

$$d_{régi} - d_{új} = F(x_p+1, y_p - ½) - F(x_p+2, y_p - 3/2)$$

$$d_{régi} - d_{új} = F(x_p+1, y_p - ½) - F(x_p+2, y_p - ½)$$

$$\Delta_{SE} = b^2 (2x_p+3) + a^2 (-2y_p+2)$$

$$\Delta_{E} = b^2 (2x_p+3)$$



Da Silva algoritmusa

- Kezdés:

  kezdőpont: (0, b)

  felezőpont: (1, b – ½)

  $d = F(1, b - ½) = b^2 + a^2 (– b + ¼)$

- A 2. tartományban hasonlóan

Program

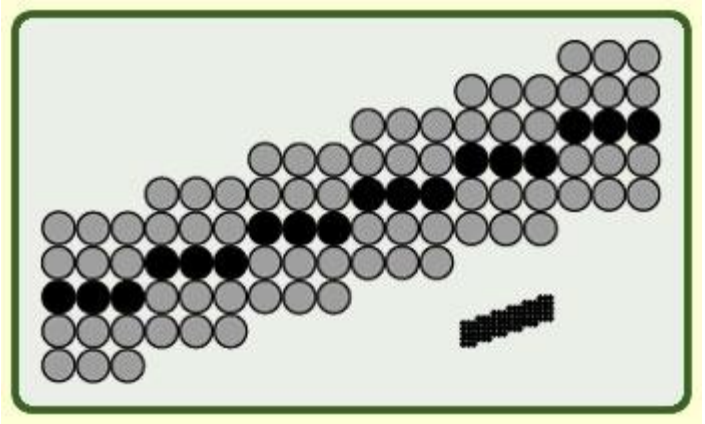
```
procedure MidpointEllipse
    (a,b,value : integer);
var  x,y: integer; d1,d2: real;
begin
  x := 0; y := b;
  a2:=a*a; b2:= b*b;
  d1 := b2-a2b+a2/4;
  EllipsePoints (x,y,value);
  while (a2(y-1/2) > b2(x+1)) do
  begin
    if d1<0 then begin
      d1 := d1+b2(2x+3); x := x+1
    end
    else begin
      d1 := d1+b2(2x+3)+a2(-2y+2);
      x := x+1; y := y-1
    end
    EllipsePoints (x,y,value)
  end; {Region1}
  d2 := b2(x+1/2)2+a2(y-1)2-a2b2;
  while (y>0) do begin
    if d2<0 then begin
      d2 := d2+b2(2x+2)+a2(-2y+3);
      x := x+1; y := y-1
    end
    else begin
      d2 := d2+a2(-2y+3); y := y-1
    end;
    EllipsePoints (x,y,value)
  end {Region2}
end; {MidpointEllipse}
```

Vastag primitívek rajzolása

- Több képpontnyi vastagságú vonalak
- Milyen alakú legyen az ecset?
  - Kör?
  - Téglalap?
  - Forduljon a vonallal?

Képpontok ismétlése

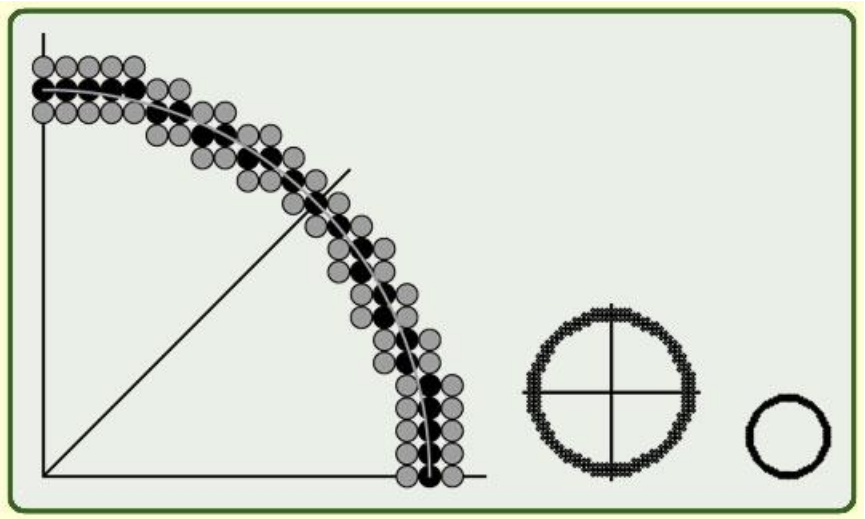
- A pásztázó vonalas algoritmus kiterjesztése:
  - ha -1<*m*<1, akkor a képpontokat többszörözzük meg az oszlopokban; különben a sorokban



Képpontok ismétlésének tulajdonságai

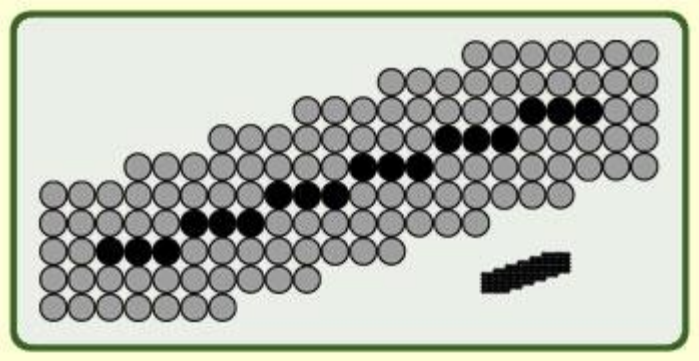
- gyors
- a vonal végek mindig vízszintesek vagy függőlegesek
- a vonal vastagsága függ a meredekségtől
- a duplázás nem megy: a vonal valamelyik oldala felé vastagabb

Jó módszer, ha nem túl vastag a vonal



Mozgó ecset

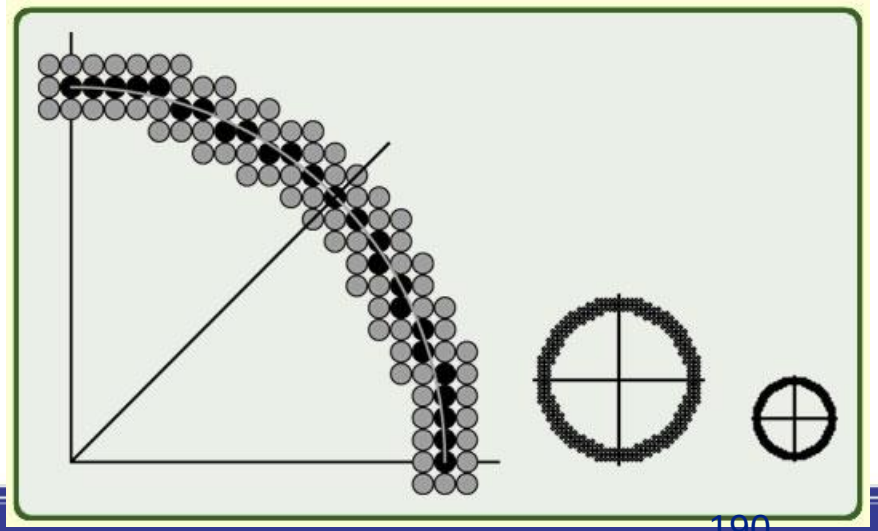
- Téglalap alakú „ecset”, aminek a középpontja (vagy csúcspontja) az 1 pixel vastag vonalon mozog (az ecset nem "forog")



Mozgó ecset tulajdonságai

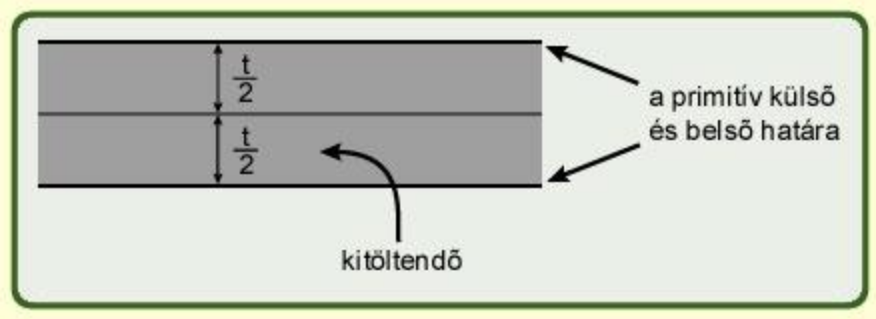
- hasonló a képpont ismétléshez, de
- a végpontokban vastagabb
- a vonal vastagsága függ a meredekségtől és az ecset alakjától
  - jobb a kör alakú ecset

- Implementáció: ecset (= minta) másolása az 1 pixel vastag vonal minden pontjába



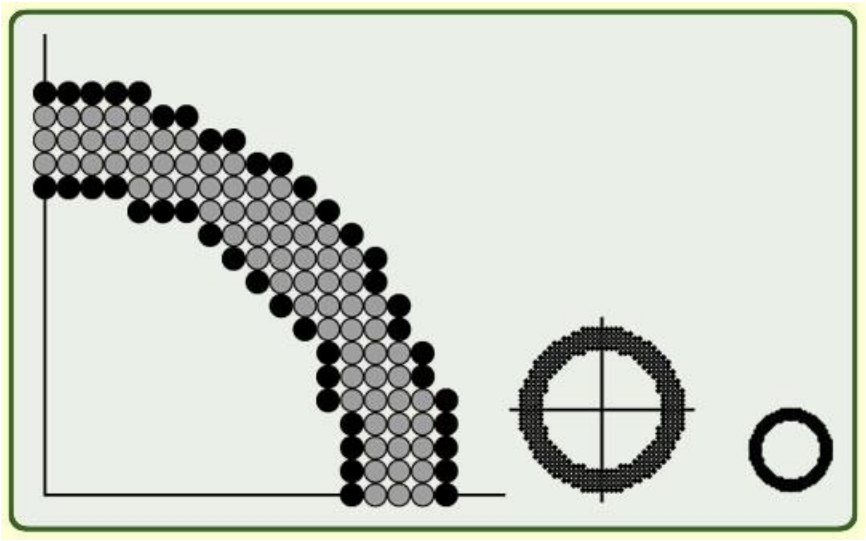
Területkitöltés

- Terület primitíveknél elég a külső határvonalhoz a külsőt, belsőhöz a belsőt meghatározni



Területkitöltés tulajdonságai

- ugyanolyan jó páros és páratlan vastagra
- a vonal vastagsága nem függ a meredekségtől



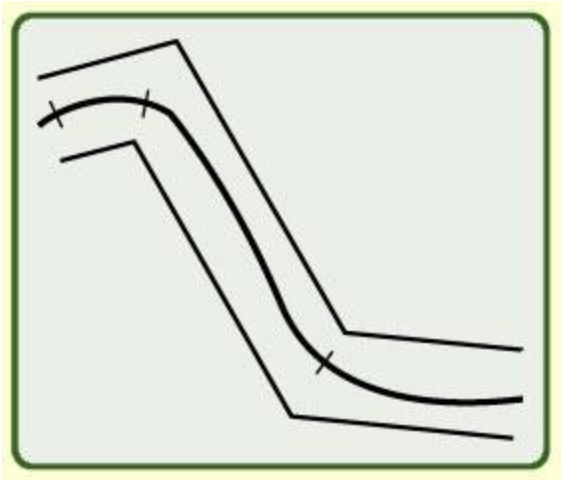
Kör esetén:
külső és belső kör

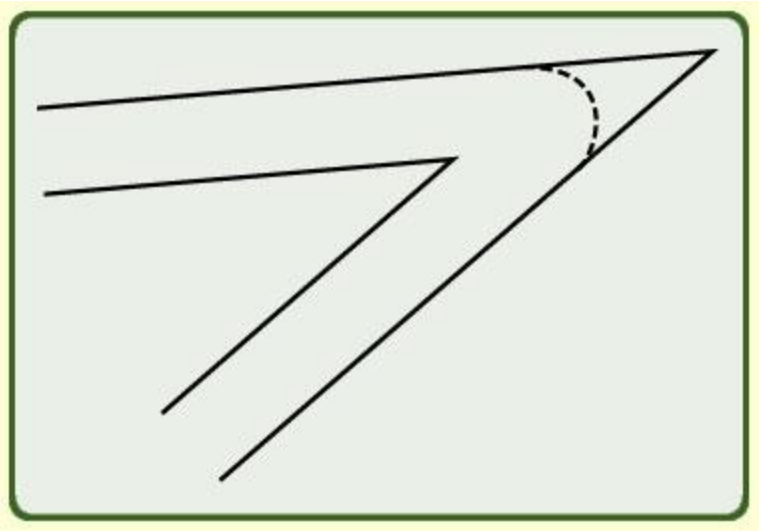
Ellipszis esetén:

a – t/2, b – t/2	belső	} ellipszisek
a + t/2, b + t/2	külső	

Közelítés vastag szakaszokkal

- Szakaszonként lineáris approximáció
  - szép
  - vastag vonalakat simán kell illeszteni

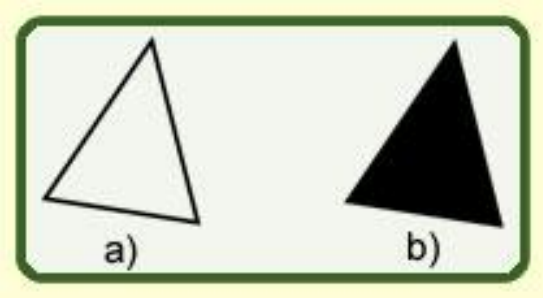




8. TERÜLETI PRIMITÍVEK RASZTERIZÁLÁSA

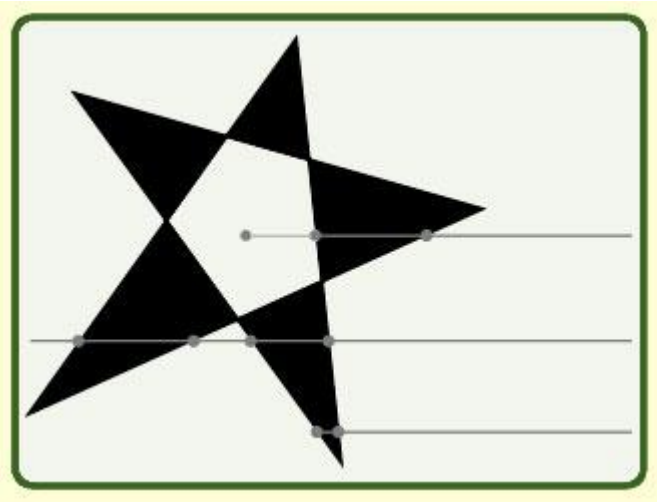
Grafikai primitívek kitöltése

- Területi primitívek
  - Zárt görbék által határolt területek (pl. kör, ellipszis, poligon)
- Megjeleníthetők
  - Csak a határvonalat reprezentáló pontok kirajzolásával (kitöltetlen)
  - Minden belső pont kirajzolásával (kitöltött)



Grafikai primitívek kitöltése

- Alapkérdés:
  - Mely képpontok tartoznak a grafikai primitívekhez?
- Páratlan paritás szabálya:
  - Páros számú metszéspont: külső pont
  - Páratlan számú metszéspont: belső pont

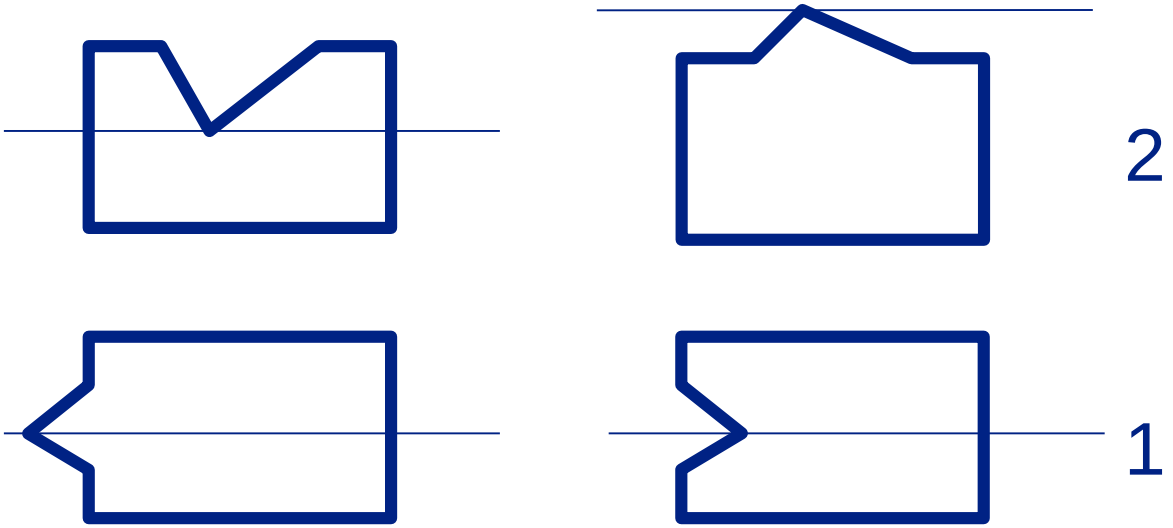


Grafikai primitívek kitöltése

- Elv:
  - Balról jobbra haladva minden egyes pásztázó (scan) vonalon kirajzoljuk a primitív belső pontjait (egyszerre egy szakaszt kitöltve)

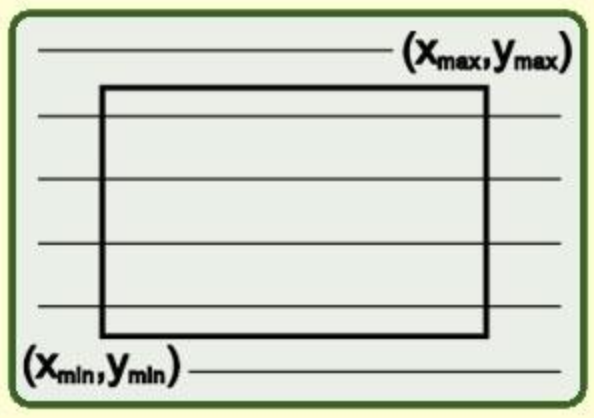
Grafikai primitívek kitöltése

- Csúcspontok metszésekor:
  - Ha a metszett csúcspont lokálisminimum vagy maximum, akkor kétszer számítjuk, különben csak egyszer



Téglalap kitöltése

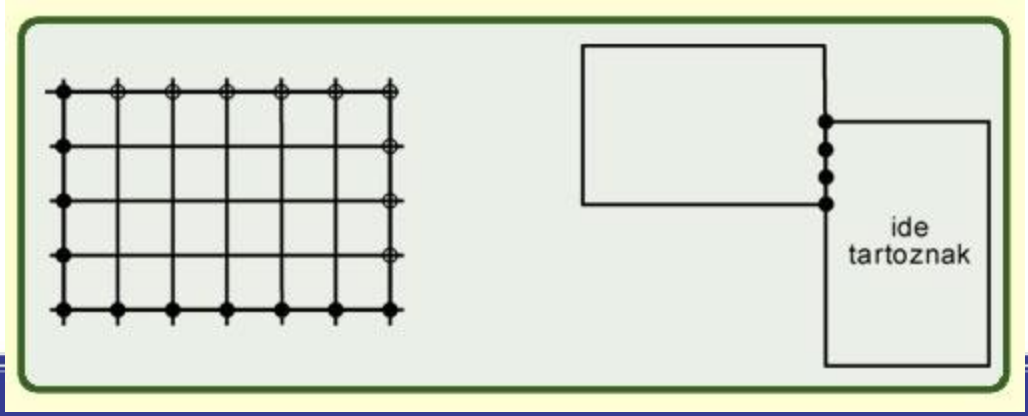
- Probléma:
  - Az egész koordinátájú határpontok hova tartozzanak?




```
for y from ymin to ymax
    for x from xmin to xmax
        WritePixel (x,y,value)
```

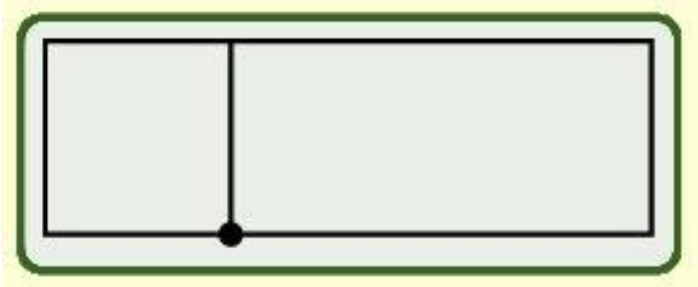
Téglalap kitöltése

- Legyen a szabály pl.:
  - Egy képpont akkor nem tartozik a primitívhez, ha a rajta áthaladó él, és a primitív által meghatározott félsík a képpont alatt, vagy attól balra van. Pl.:
  - Vagyis a pásztázó vonalon a kitöltési szakasz balról zárt, jobbról nyitott



Téglalap kitöltése

- Megjegyzések:
  - Általánosítható poligonokra
  - A felső sor és jobb szélső oszlop hiányozhat
  - A bal alsó sarok kétszeresen tartozhat téglalaphoz

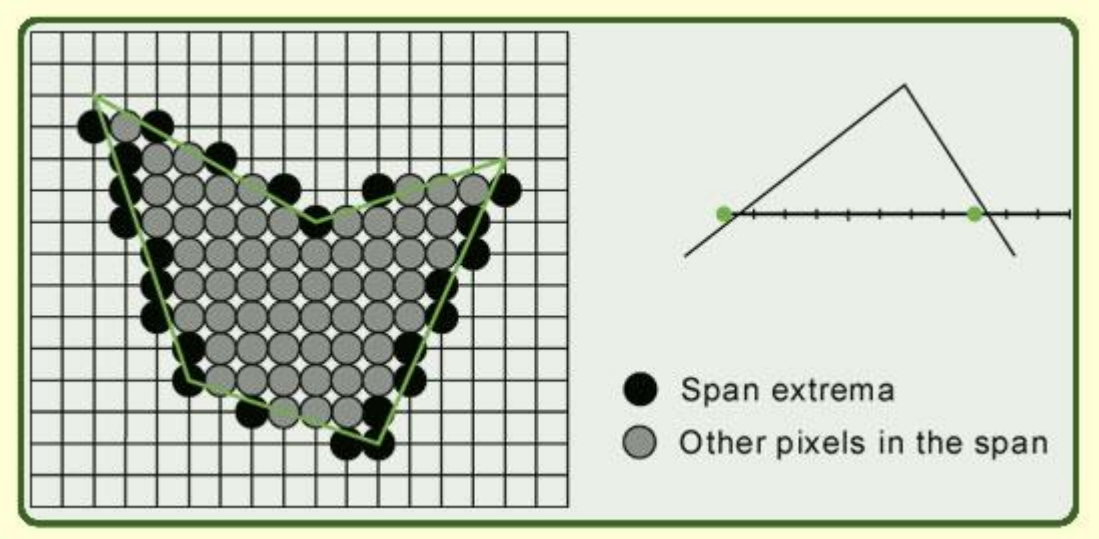


Poligon kitöltése

- A poligon lehet:
  - konvex
  - konkáv
  - önmagát metsző
  - lyukas
- Haladjunk a pásztázó egyeneseken és keressük a kitöltési szakaszok végpontjait:

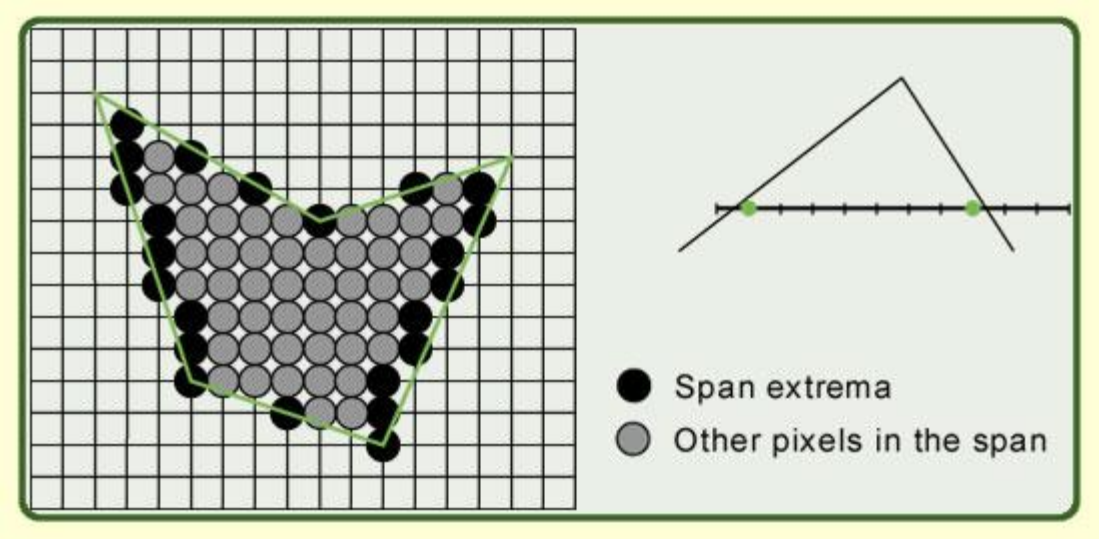
Poligon kitöltése

- a) A felezőpont algoritmus szerint választjuk a végpontokat (azaz, nem számít, hogy azok a poligonon kívül, vagy belül vannak)



Poligon kitöltése

- b) A végpontokat a poligonhoz tartozó képpontok közül választjuk



Algoritmus poligonok kitöltésére

Minden pásztázó egyenesre:

1. A pásztázó egyenes és a poligon élei metszéspontjainak a meghatározása
2. A metszéspontok rendezése növekvő x-koordinátáik szerint

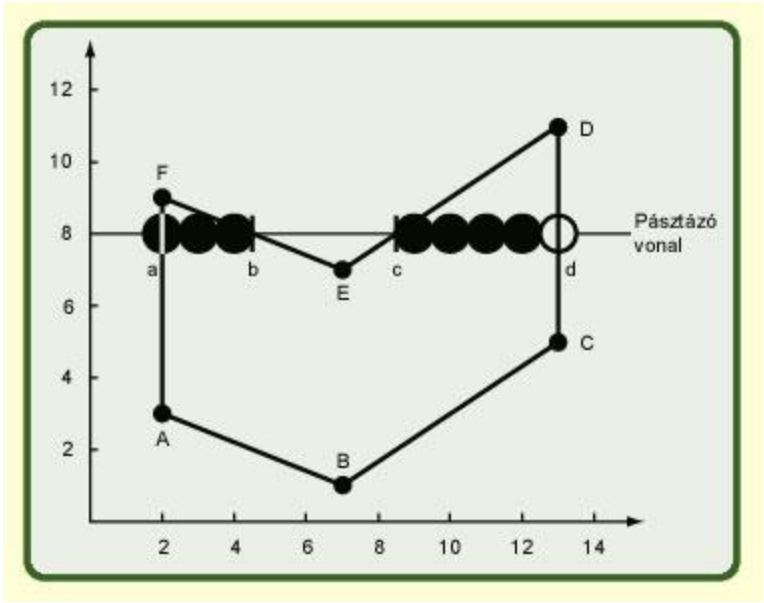
Algoritmus poligonok kitöltésére

3. A poligon belsejébe tartozó szakasz(ok) végpontjai közötti képpontok kirajzolása
   - Használjuk a páratlan paritás szabályát: Tegyük fel, hogy a bal szélen kívül vagyunk, utána minden egyes metszéspont megváltoztatja a paritást

Algoritmus poligonok kitöltésére

3.1. Adott x nem egész értékű metszéspont
- Ha kívül vagyunk, akkor legyen a végpont a fölfelé kerekített x
- Ha belül vagyunk, akkor legyen a végpont a lefelé kerekített x

3.2. Adott x egész értékű metszéspont
- Ha ez bal végpont, akkor ez belső pont
- Ha ez jobb végpont, akkor ez külső pont



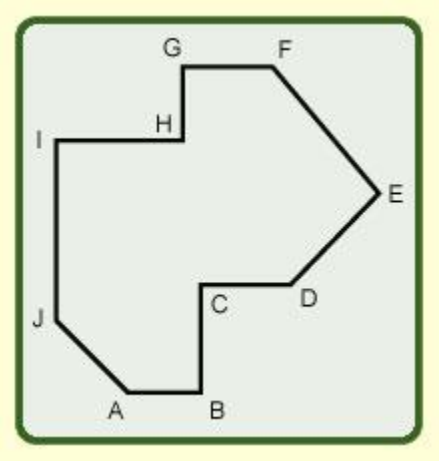
Algoritmus poligonok kitöltésére

3.2.1. A poligon csúcspontjaiban

- ymin csúcspont beszámít a paritásba
- ymax csúcspont nem számít a paritásba, tehát ymax csúcspont csak akkor lesz kirajzolva, ha az egyben a szomszédos él ymin pontja is

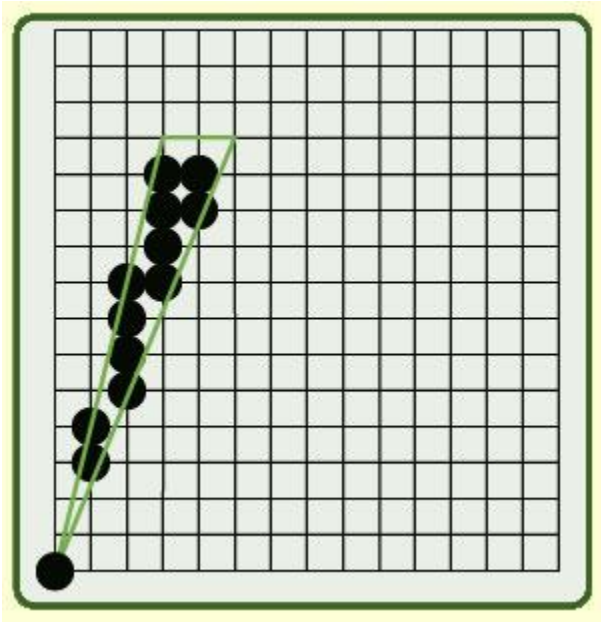
3.2.2. Vízszintes él esetén

- Az alsó élet rajzolunk, felsőt nem. Az ilyen élek csúcspontjai nem számítanak a paritásba



Megjegyzések

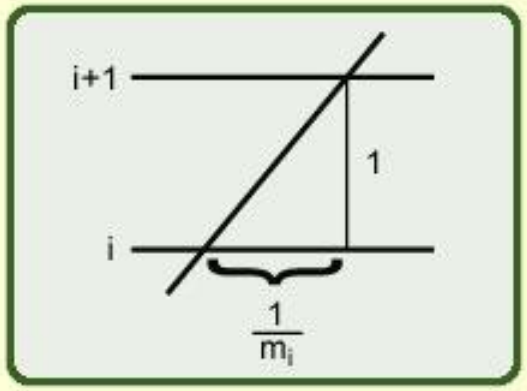
- Diszjunkt poligonoknak lehet közös képpontjuk
- Szilánkok keletkezhetnek
  - olyan poligon-területek, amelyek belsejében nincs kitöltendő szakasz = hiányzó képpontok



Megjegyzések

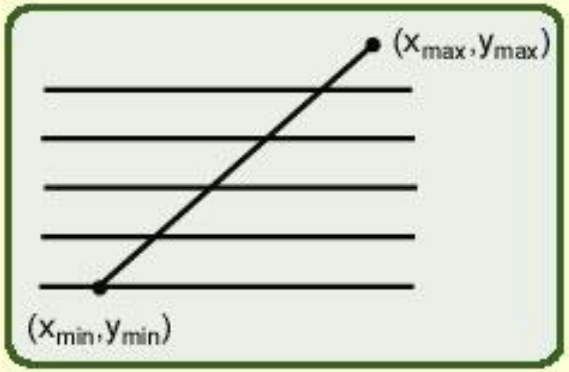
- Implementáció:
  - Nem kell minden egyes pásztázó vonalra újra kiszámolni minden metszéspontot, mert általában csak néhány metszéspont érdekes az i-edik pásztázó vonalról az i+1-edikre átlépve

$$x_{i+1} = x_i + \frac{1}{m_i}$$



Poligon kitöltése

- Tegyük fel hogy: m>1



$$\Delta x = \frac{1}{m} = \frac{x_{max} - x_{min}}{y_{max} - y_{min}} \quad (<1)$$

x = egész rész + tört rész

$[x_i]$ $\{x_i\}$

$[x_i+1] = [x_i]$ vagy $[x_i] + 1$

$\{x_i+1\} = \{x_i\}$ vagy $\{x_i\} - 1$

Poligon kitöltése

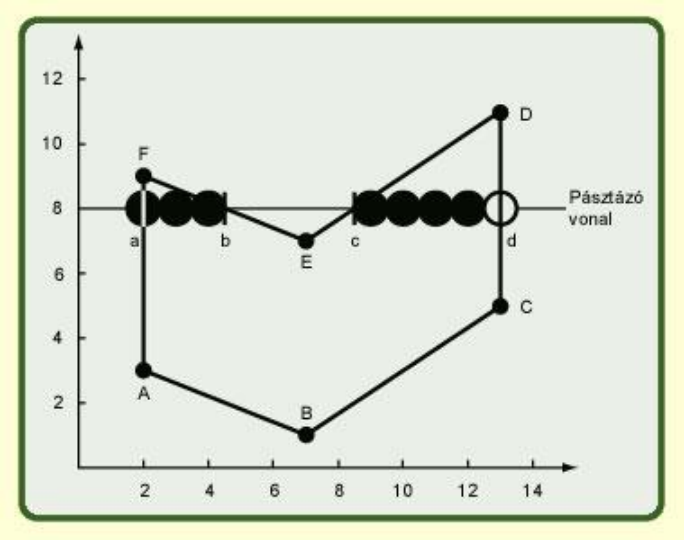
- Tegyük fel, hogy a bal határon vagyunk!
  - Ha $\{x_i\} = 0$, akkor (x, y)-t rajzolni kell (a vonalon van)
  - Ha $\{x_i\} \neq 0$, akkor fölfelé kell kerekíteni x-et (belső pont)
- Egész értékű aritmetika használható: törtrész helyett a számláló és nevező tárolása

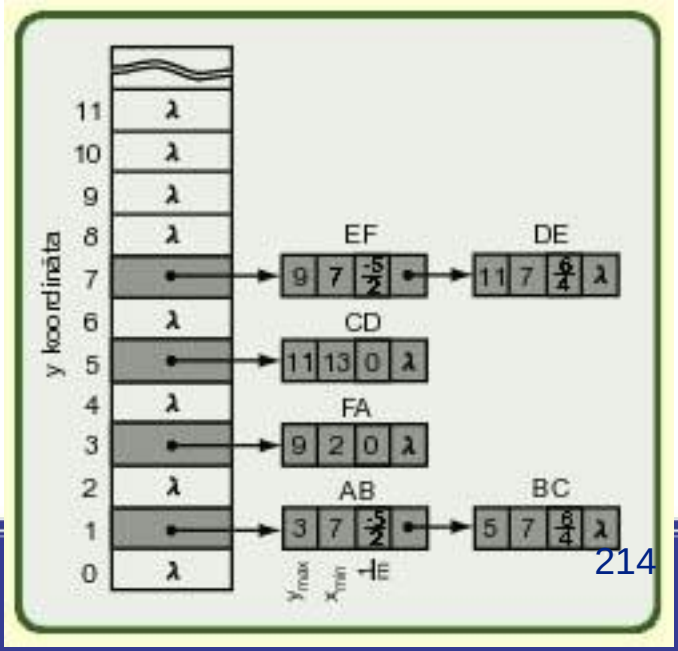
Program

```
procedure LeftEdgeScan (xmin, ymin, xmax, ymax, value :integer);
var x, y, numerator, denominator, increment : integer;
begin
  x := xmin; numerator := xmax-xmin;
  denimonator := ymax-ymin;
  increment := denominator;
  for y := ymin to ymax do begin
    WritePixel (x,y,value);
    increment := increment+numerator;
    if increment > denominator then begin
      x := x+1;
      increment := increment-denominator
    end;
  end
end;
```

Adatstruktúrák

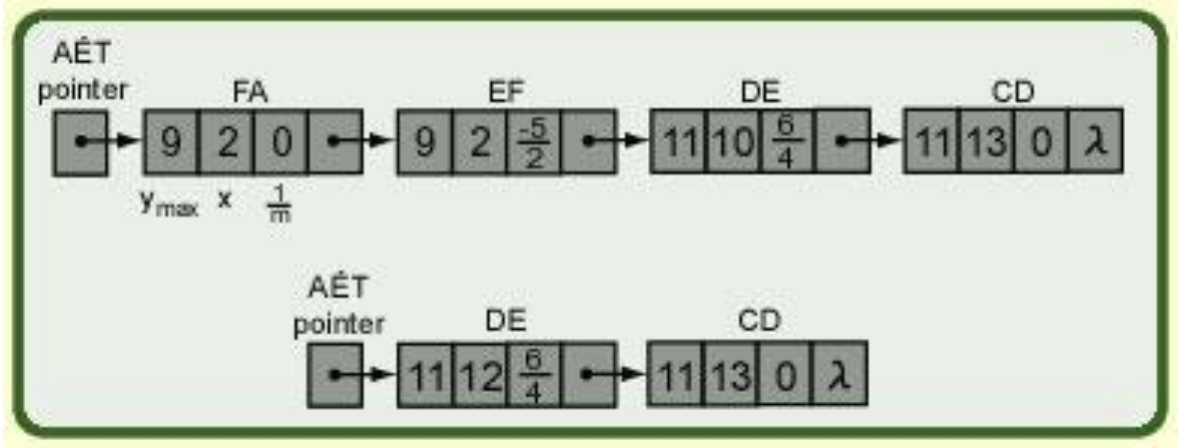
- ÉT: (Élek Táblázata)
  - A kisebbik y értékük szerint rendezve az összes élet tartalmazza. A vízszintes élek kimaradnak!
  - Annyi lista, ahány pásztázó vonal, minden listában az élek az alsó végpont x koordinátája szerint rendezettek





Adatstruktúrák

- AÉT: (Aktív Élek Táblázata)
  - Azokat az éleket tartalmazza rendezve, amelyeknek a metszéspontjai kitöltési szakaszokat határoznak meg az aktuális pásztázó vonalon
  - Ez is lista



Algoritmus poligon kitöltésére

0. ÉT kialakítása
1. y legyen az ÉT-ben levő nem üres listák közül a legkisebb y
2. AÉT inicializálása (üres)
3. A továbbiakat addig ismételjük, amíg ÉT végére érünk és AÉT üres lesz:

Algoritmus poligon kitöltésére

3.1 ÉT-ből az y-hoz tartozó listát AÉT-ba másoljuk és AÉT-t x szerint rendezzük

3.2 A kitöltési szakaszok pontjainak a megjelenítése

3.3 AÉT-ből kivesszük azokat az éleket, amelyekre ymax = y (a következő pásztázó egyenessel nincs közös részük)

3.4 y = y+1

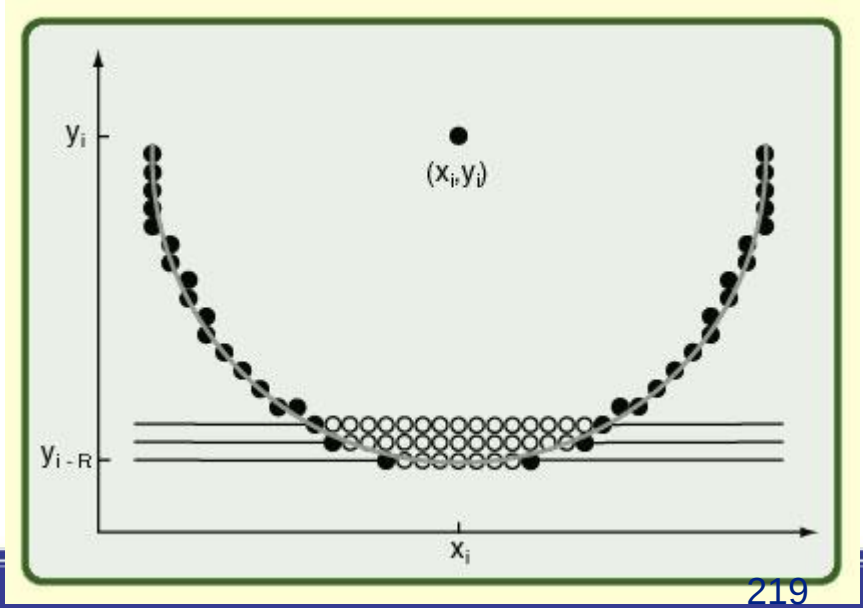
3.5 Minden AÉT-beli élben módosítjuk x-et

Megjegyzés

- Háromszögekre, trapézokra egyszerűsíthető, mert a pásztázó egyeneseknek max 2 metszéspontja lehet a primitívvel

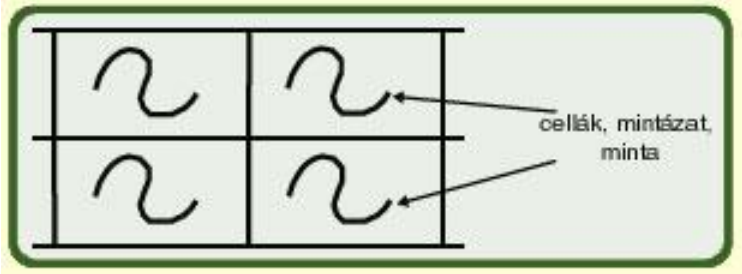
Kör, ellipszis kitöltése

- P belül van, ha F(P) < 0, de most is használható a felezőpont módszer (nem kell ÉT – csak 2 metszéspont lehet)
- Hasonló algoritmussal számíthatók a kitöltési szakaszok



Kitöltés mintával

- Általában: terület kitöltése szabályosan ismétlődő grafikus elemekkel



- Képmátrixok (raszter) esetében a cella egy (kisméretű) mátrix
- Lehet a kitöltés "átlátszó" is: nem minden képpontot írunk felül, csak azokat, ahol a minta nem 0

Mintával kitöltés fajtái

- Válasszunk egy pontot a primitívben (pl. bal felsőt), egy pontot a mintában (pl. bal felsőt), illesszük azokat egymásra, a többi pont illeszkedése már kiszámítható (a mintázat a primitívhez van rögzítve)

- Válasszunk egy pontot a képernyőn (pl. bal felsőt), egy pontot a mintában (pl. bal felsőt), illesszük azokat egymásra, a többi pont illeszkedése már kiszámítható (most a mintázat a képernyőhöz van rögzítve)

Kitöltés mintával

Legyen:

minta M * N -es mátrix

minta [0,0] → képernyő [0,0]

ekkor

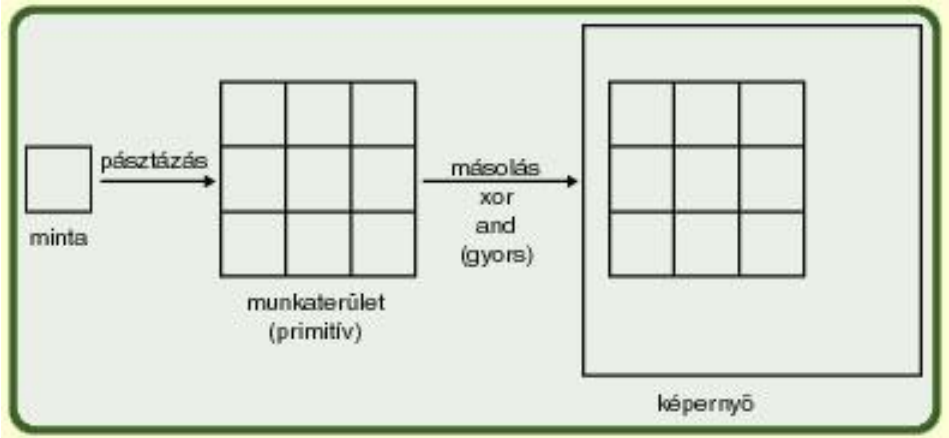
1. módszer: Pásztázás soronként (átlátszó)

```
if minta [x mod M, Y mod N] then
  WritePixel(x,y,érték)
```

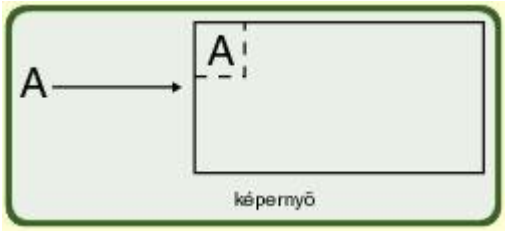
- Gyorsabb: több képpont (sor) egyszerre történő másolásával (esetleg maszkolás is szükséges a sor elején vagy végén)

Kitöltés mintával

- 2. módszer: Téglalap írás



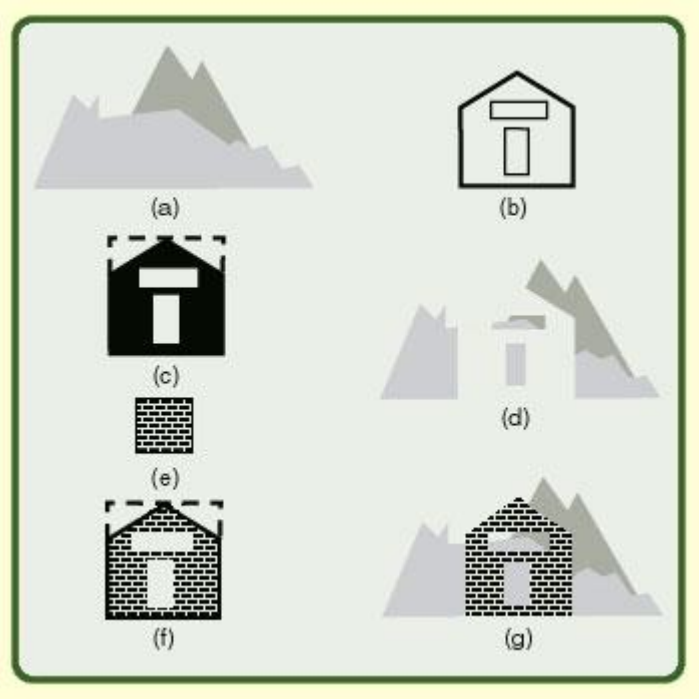
- Csak akkor érdemes használni, ha a primitívet sokszor kell használni
  Pl. karakterek megjelenítése



Kitöltés mintával

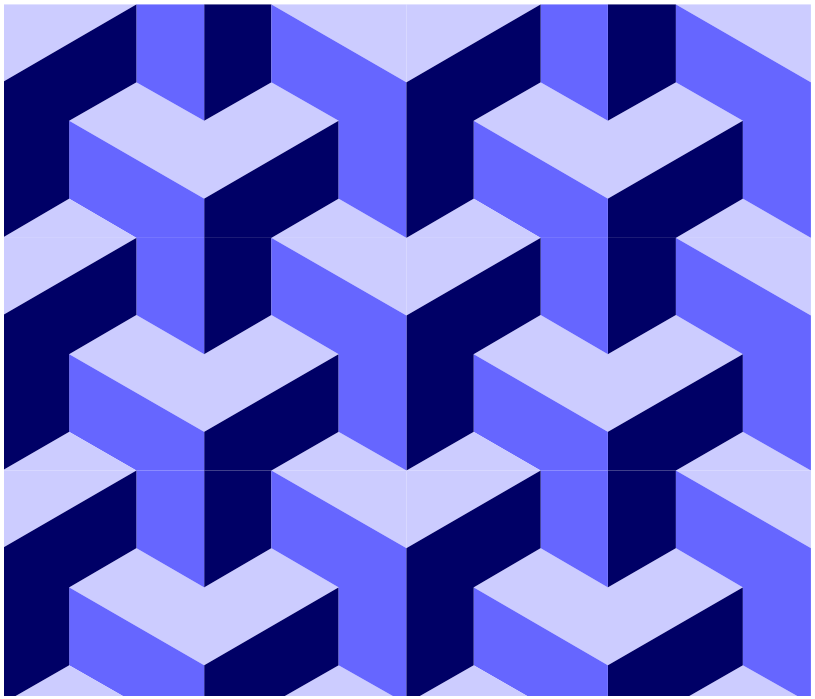
- A téglalap írás kitöltés kombinálható képek közötti műveletekkel, így bonyolult ábrák készíthetők:

(a) hegyek
(b) ház vonalai
(c) a ház kitöltött bitmap képe
(d) (a)-ból kitöröltük (c)-t
(e) tégla minta
(f) (b) tégla mintával kitöltve
(g) (e) (d)-re másolva



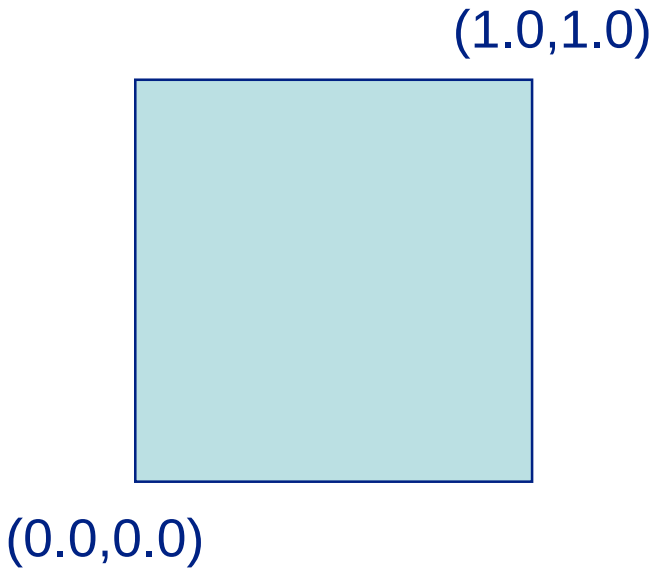
Textúrák

- Képek feszítése a poligonokra
- Nagy számításigényű
- Textúra ~ kép
  - 1D-s textúra
  - 2D Windows .bmp
  - 3D (volume) MRI
    - 256x256x256
    - OpenGL 1.2



Textúra koordináták

```
glTexCoord1{dfi}( s )
glTexCoord2{dfi}( s, t )
```

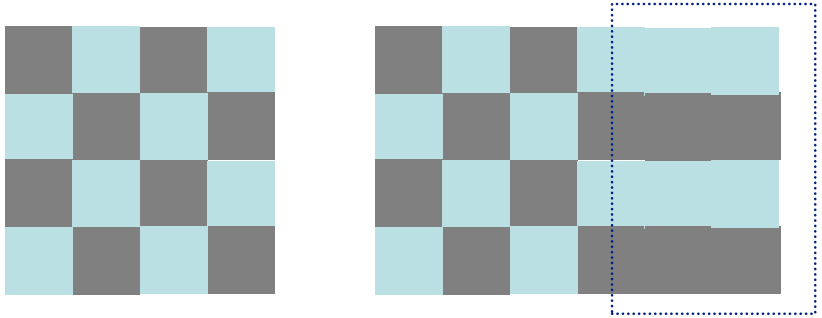


Textúra csomagolás

- A textúra koordináták 0.0-1.0 között vannak
- Koordináták kívül kerülnek
  - összekapcsolódnak
  - ismétlődnek
- További keret (border) pixelek:
  - a keret pixeleket használja majd

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_S,
   wrap)
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_T,
   wrap)
```

- wrap
  GL_CLAMP
  GL_REPEAT



Mipmapped textúrák

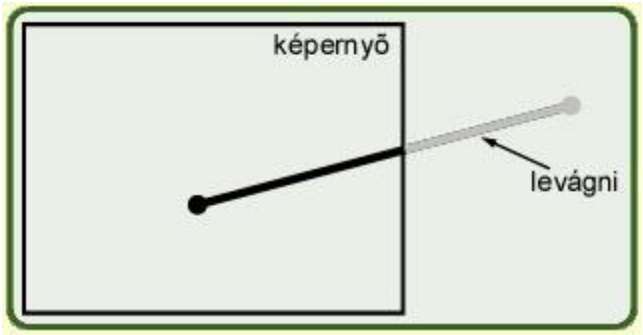
- Mip = many = sok
- Több szintű részletek
- Távolság függő
- Hosszabb betöltési idő
- Hatásosabb vizuális eredmény
- Gyorsabb megjelenítés
- Definiálás a szint megadásával a glTexImage1D, ... fv.-eknél
- 0 legnagyobb felbontású
- GL_TEXTURE_MIN_FILTER

```
static unsigned char im0[16][3];
static unsigned char im1[8][3];
static unsigned char im2[4][3];
glTexImage1D(GL_TEXTURE_1D, 0, 3, 16, 0, GL_RGB,
    GL_UNSIGNED_BYTE, im0);
glTexImage1D(GL_TEXTURE_1D, 1, 3, 16, 0, GL_RGB,
    GL_UNSIGNED_BYTE, im1);
glTexImage1D(GL_TEXTURE_1D, 2, 3, 16, 0, GL_RGB,
    GL_UNSIGNED_BYTE, im2);
```

11. GRAFIKUS PRIMITÍVEK VÁGÁSA

Vágás

- A primitívekből csak annyit szabad mutatni, amennyi látszik belőlük (takarás, kilóg a képből)



Módszerek

1. Vágjuk le a megjelenítés előtt, azaz számítsuk ki a metszéspontokat és az új végpontokkal rajzoljunk
2. Ollózás: pásztázzuk a teljes primitívet, de csak a látható képpontokat jelenítjük meg: minden (x, y)-ra ellenőrzés
3. A teljes primitívet munkaterületre rajzoljuk, majd innen átmásoljuk a megfelelő darabot

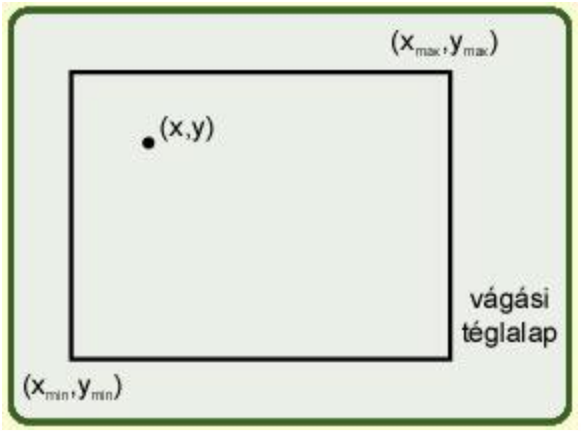
Pontok vágása

(x,y) belül van, ha

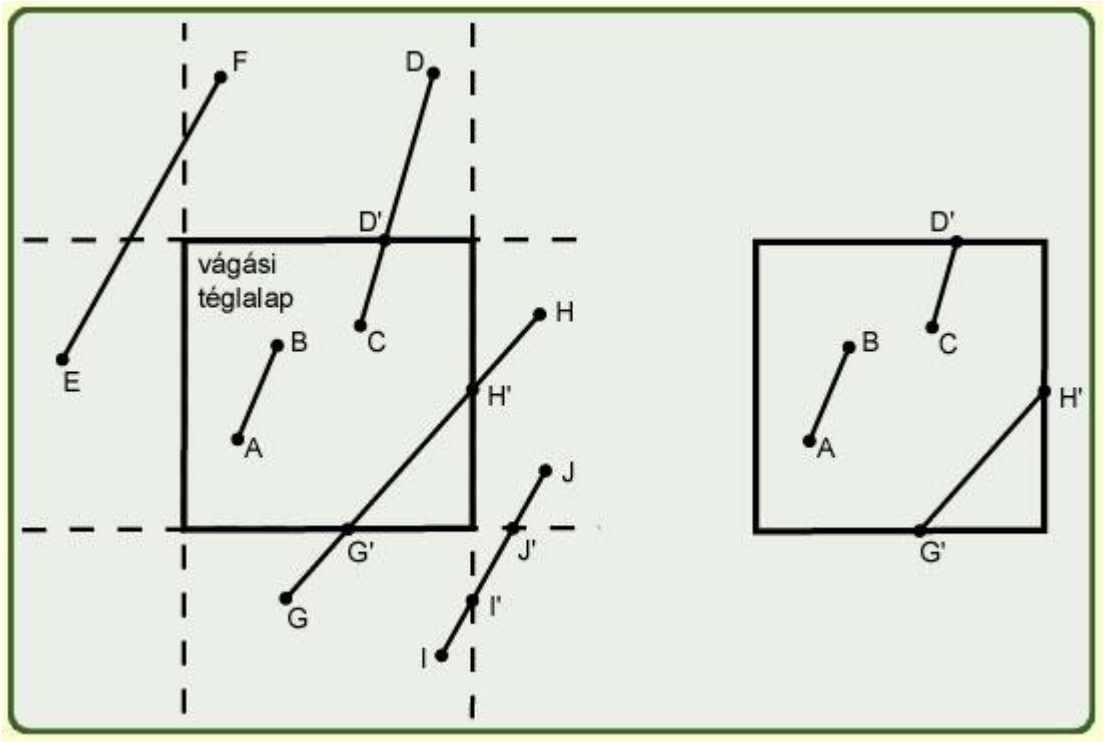
$$x_{min} \leq x \leq x_{max}$$

és

$$y_{min} \leq y \leq y_{max}$$



Szakaszok vágása egyenletrendszer megoldásával



Szakaszok vágása egyenletrendszer megoldásával

- Elég a végpontokat vizsgálni:
  - Ha mindkét végpont belül van, akkor a teljes szakasz belül van, nincs vágás
  - Ha pontosan egy végpont van belül, akkor metszéspontot kell számolni és vágni
  - Ha mindkét végpont kívül van, akkor további vizsgálat szükséges: lehet, hogy nincs közös része a vágási téglalappal, de lehet, hogy van

Szakaszok vágása egyenletrendszer megoldásával

- A vágási téglalap minden élére megvizsgáljuk, van-e az élnek közös része a szakasszal
- Egyenesek metszéspontjának meghatározása, és az élen belül van-e a metszéspont

- Problémák:
  - Egyenesek (nem szakaszok!) metszéspontjai
  - Speciális esetek (vízszintes, függőleges egyenesek)

Szakaszok vágása egyenletrendszer megoldásával

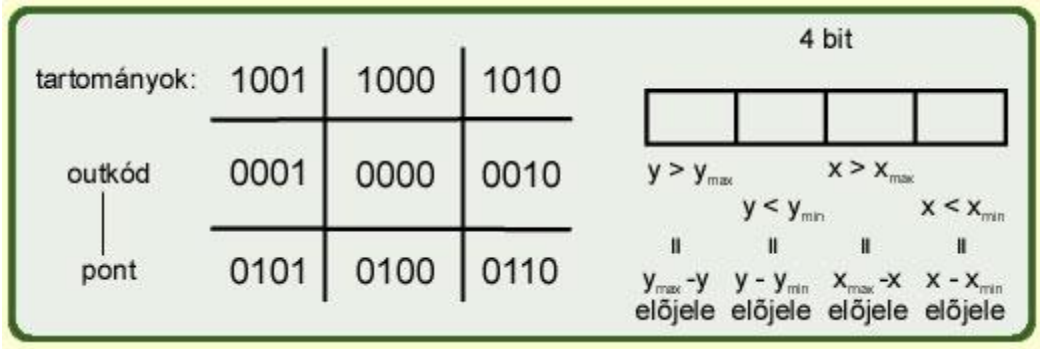
- Javítás: parametrikus alak

$$x = x_0 + t \cdot (x_1 - x_0)$$
$$y = y_0 + t \cdot (y_1 - y_0)$$

- $t \in [0,1]$ (szakaszt ír le)
- Metszéspont:
  $t_{él}$: a metszéspont paramétere az élen
  t_{vonal} : a metszéspont paramétere a vonalon
- Ha $t_{él}$, $t_{vonal} \in [0,1]$, akkor belül van
- Még így sem hatékony a módszer, mert sokat kell ellenőrizni és számolni

Cohen-Sutherland-féle szakasz vágás

- (x_1, y_1) és (x_2, y_2) a szakasz két végpontja
- A végpontok kódolása: Minden végpont annak megfelelő kódot $(code_1, code_2)$ kap, hogy melyik tartományban van



Cohen-Sutherland-féle szakasz vágás

1. Ha a végpontok belül vannak, akkor nincs mit vágni (triviális elfogadás)

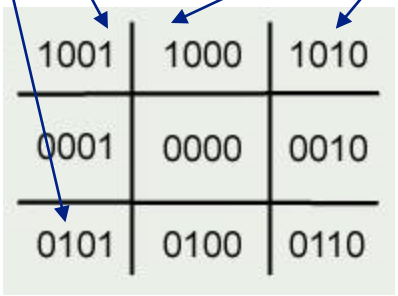
1001	1000	1010
0001	0000	0010
0101	0100	0110

$code_1 = code_2 = 0000$

$code_1$ OR $code_2$ = FALSE (bitenként)

Cohen-Sutherland-féle szakasz vágás

2. Ha $x_1,x_2 < x_{min}$ vagy $x_1,x_2 > x_{max}$
vagy $y_1,y_2 < y_{min}$ vagy $y_1,y_2 > y_{max}$
akkor minden kívül van (triviális elvetés)



$code_1$ AND $code_2$ = TRUE (bitenként)

Cohen-Sutherland-féle szakasz vágás

3. az $(x_1,y_1) – (x_2,y_2)$ szakasz metszi valamelyik élet
- Vegyünk egy külső végpontot (legalább egyik az; ha több van, akkor válasszuk közülük felülről lefelé és jobbról balra haladva az elsőt), és számítsuk ki a metszéspontot
- A két részre vágott szakasz egyik fele a 2. pont alapján triviálisan elvethető

Tulajdonságok

- Interaktív módon is használható
- Hatékony, mert gyakran sok vagy kevés szakasz van belül
- A legáltalánosabban használt eljárás

Parametrikus szakasz vágó algoritmus

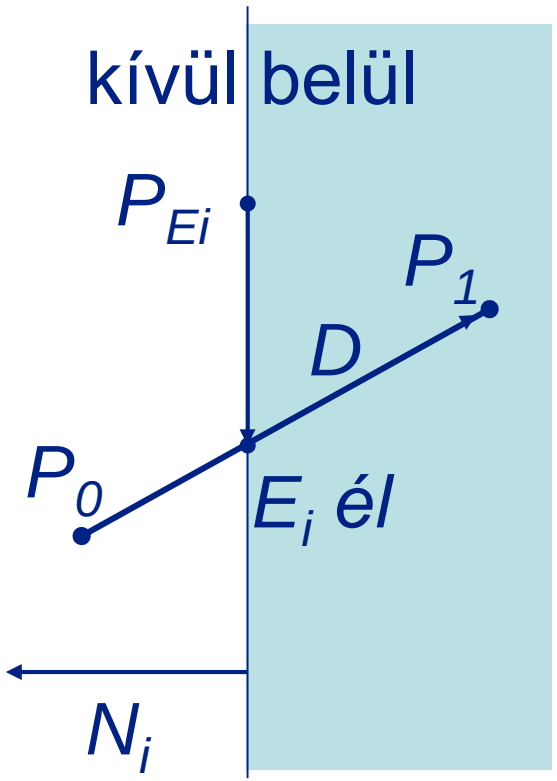
$$P(t) = P_0 + \underbrace{(P_1 - P_0)}_{D}\, t = P_0 + D\, t$$

A metszéspontra (skalárszorzat):

$$N_i\,(P(t) - P_{E\,i}) = 0$$

$$N_i\,(P_0 + D\,t - P_{E\,i}) = 0$$

$$t = \frac{N_i\,(P_0 - P_{E\,i})}{-N_i\,D}$$

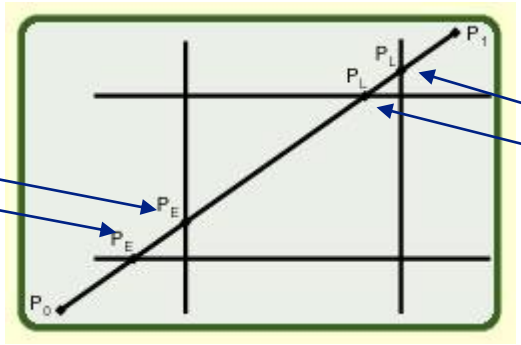


Ha $N_i\,D$ {
- < 0, akkor belép a félsíkba,
- = 0, akkor párhuzamos a félsík élével,
- > 0, akkor kilép a félsíkból.

Parametrikus szakasz vágó algoritmus

- Meghatározható az egyenesnek a téglalap 4 egyenesével való 4 metszéspontja (4 db t érték)
- Melyik két t a megfelelő?

Belépő pontok



Kilépő pontok

- Legyen

$$t_E = \max\{0, \max\{t_{PE}\}\}$$
$$t_L = \min\{1, \min\{t_{PL}\}\}$$

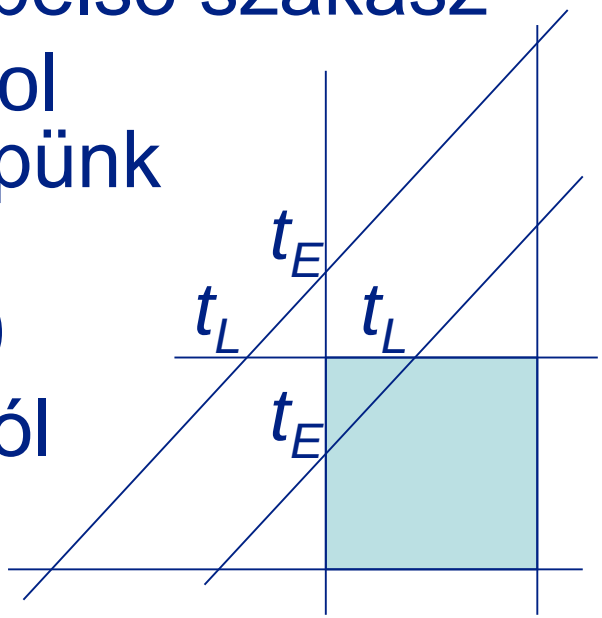
Parametrikus szakasz vágó algoritmus

- Ha $t_E > t_L$, akkor nincs belső metszés
- Ha $t_E , t_L \in [0,1]$, akkor ez belső szakasz
- különben P_E olyan pont, ahol P_0-ból P_1-felé haladva belépünk egy belső félsíkba, ekkor

$$N_i (P_1 - P_0) < 0$$

- és P_L olyan pont, ahol P_0-ból P_1-felé haladva kilépünk egy belső félsíkból, ekkor

$$N_i (P_1 - P_0) > 0$$



Parametrikus számítás

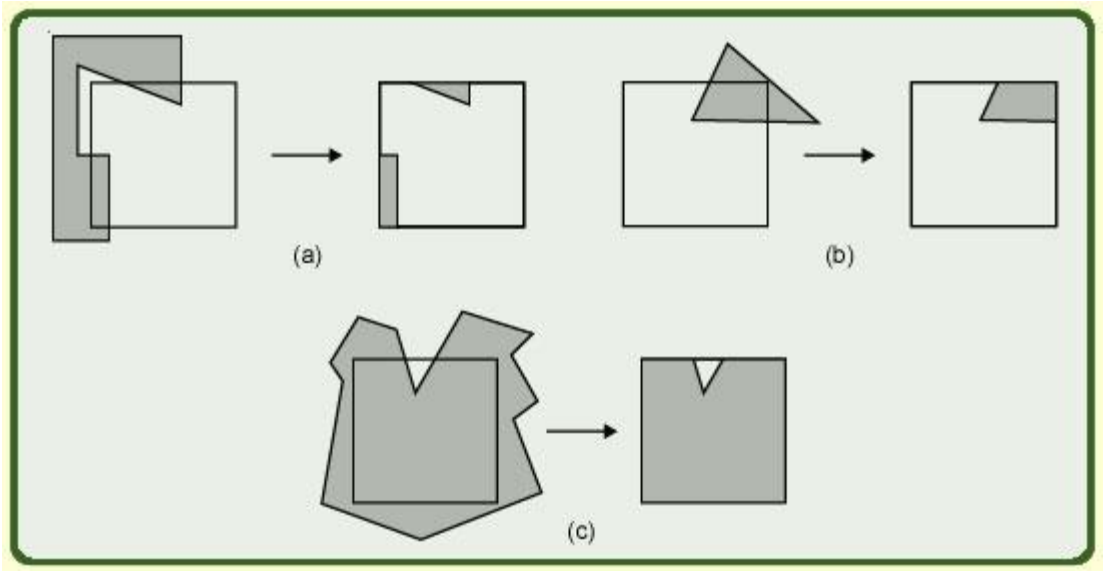
él$_i$ vágás	N_i	P_{Ei}	$P_0 - P_{Ei}$	$t = \frac{N_i \cdot (P_0 - P_{Ei})}{-N_i \cdot D}$
bal $x=x_{min}$	$(-1, 0)$	(x_{min}, y)	$(x_0 - x_{min}, y_0 - y)$	$\frac{-(x_0 - x_{min})}{(x_1 - x_0)}$
jobb $x=x_{max}$	$(1, 0)$	(x_{max}, y)	$(x_0 - x_{max}, y_0 - y)$	$\frac{(x_0 - x_{max})}{-(x_1 - x_0)}$
lent $y=y_{min}$	$(0, -1)$	(x, y_{min})	$(x_0 - x, y_0 - y_{min})$	$\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$
fent $y=y_{max}$	$(0, 1)$	(x, y_{max})	$(x_0 - x, y_0 - y_{max})$	$\frac{(y_0 - y_{max})}{-(y_1 - y_0)}$

Parametrikus szakasz vágó algoritmus

```
begin
  Ni kiszámítása, PEi kiválasztása;
  for szakaszokra
    if P1 = P0 then pont vágása;
    else begin
      tE = 0; tL = 1; D = P1-P0;
      for élekkel való metszéspontokra
        if Ni*D <> 0 then begin
          t kiszámítása. Ni*D <0: PE, >0: PL;
          if PE then tE = max(tE,t);
          if PL then tL = min(tL,t)
        end;
        if tE > tL then nincs metszéspont
        else P(tE) és P(tL) metszéspont
    end
end
```

Poligonok vágása

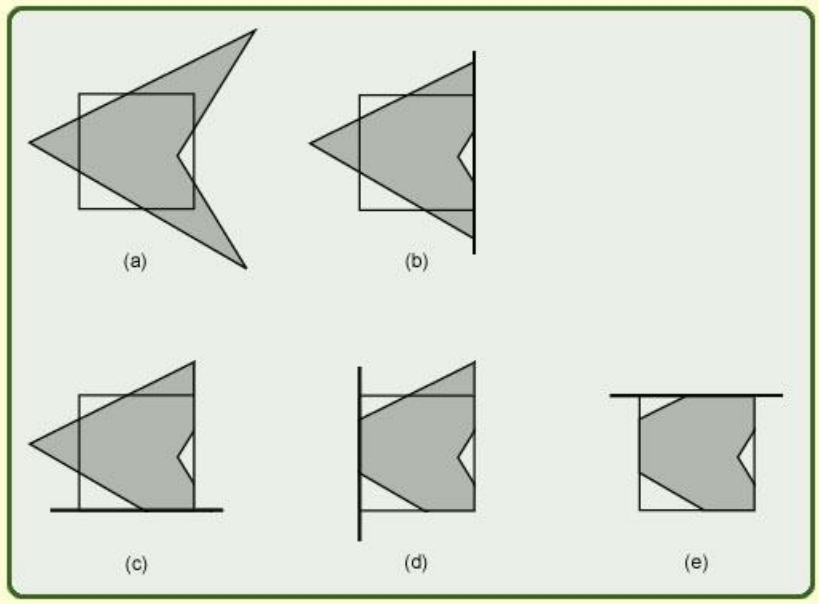
- Sok eset lehet:



- Általában minden éllel vágni kell

Sutherland-Hodgman-féle poligon-vágás

Vágjunk egyenként az összes éllel



$(v_1, v_2, \ldots, v_n)$ csúcspontok

algoritmus

$(v'_1, v'_2, \ldots, v'_m)$ új csúcspontok

12. LÁTHATÓSÁG, TAKARÁS

Látható felszín meghatározása

- Adott 3D tárgyak egy halmaza és egy projekció specifikációja

- Mely vonalak és felületek lesznek láthatók?
- Melyek lesznek takarva?

- Nehéz feladat (időigényes)

1. megközelítés

```
for minden képpontra do begin
   határozzuk meg azt a tárgyat,
   amelyet a nézőpontból a képponton
   keresztül húzott egyenes leghamarabb
   metsz
   rajzoljuk ki a képpontot a megfelelő
   színben
end
```

- A szükséges idő: O(np)
- n: a tárgyak száma
- p: a képpontok száma

2. megközelítés

```
for minden tárgyra do begin
  határozzuk meg a tárgynak azokat a
  részeit, amelyek nincsenek
  takarásban saját maga vagy más
  tárgyak miatt
  a kiválasztott részeket rajzoljuk ki
  a megfelelő színben
end
```

- A szükséges idő: $O(n^2)$
- n: a tárgyak száma

A látható felszín meghatározására szolgáló általános algoritmusok

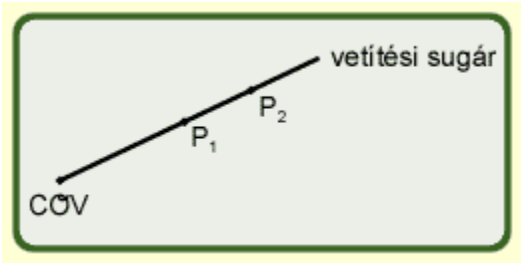
- A tárgyak takarják-e egymást?
- Mely tárgy látható?

- Általánosan: nehéz probléma

Pontok láthatósága

$P_1 = (x_1, y_1, z_1)$ és $P_2 = (x_2, y_2, z_2)$

- Takarja-e egyik a másikat?
- Ha ugyanazon a vetítési sugáron vannak, akkor a közelebbi takarja a másikat, különben nem



Mélységbeli összehasonlítás

- Helye: a normalizálási transzformáció után, ekkor
  - párhuzamos vetítésnél:
    a vetítési sugarak párhuzamosak a z-tengellyel, ekkor P_1 és P_2 ugyanazon a vetítési sugáron van, ha

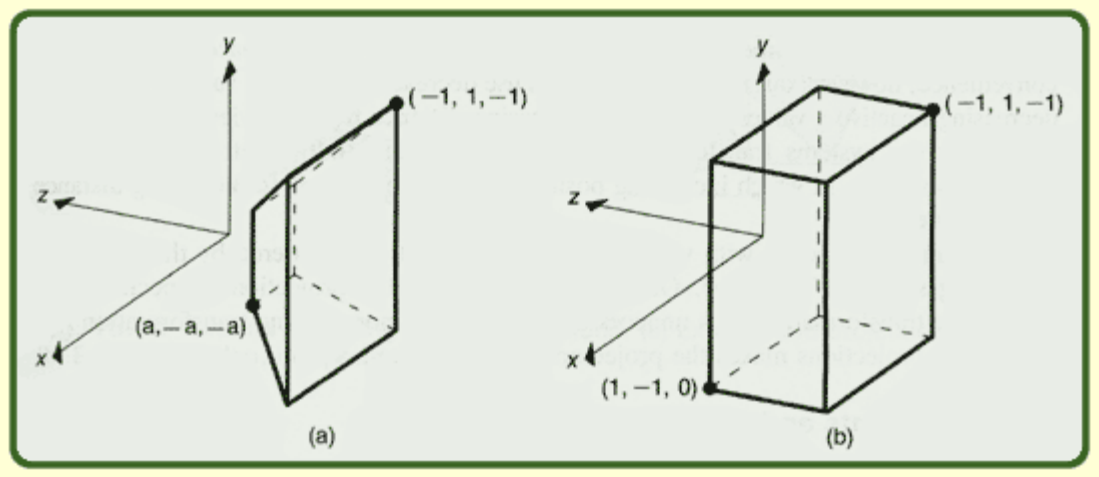
$$x_1 = x_2 \text{ és } y_1 = y_2$$

  - perspektív vetítésnél:
    a vetítési sugarak COV-ből indulnak ki, ekkor P_1 és P_2 ugyanazon a vetítési sugáron van, ha

$$x_1 / z_1 = x_2 / z_2 \text{ és } y_1 / z_1 = y_2 / z_2$$

Kanonikus térfogat

- Perspektív vetítésnél azt a transzformációt használjuk, amely a perspektív kanonikus térfogatot átviszi párhuzamos kanonikus térfogatba



Kanonikus térfogatba transzformáló mátrix

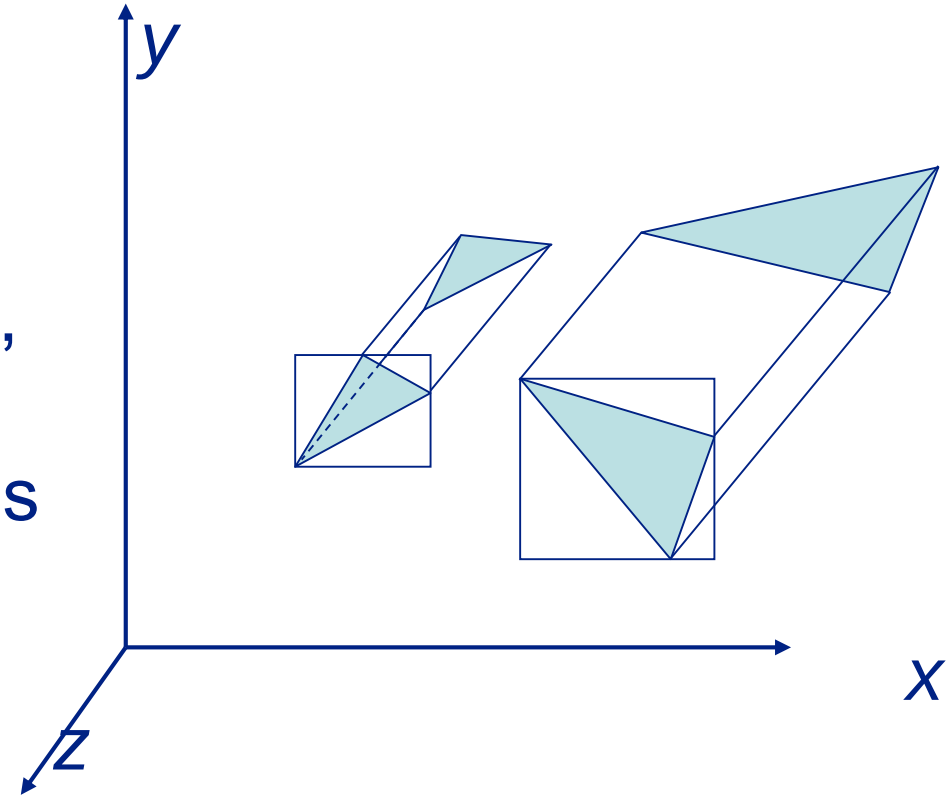
- Perspektív kanonikus térfogatot párhuzamos kanonikus térfogatba transzformáló mátrix

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & \frac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

- Ekkor a vetítési sugarak már párhuzamosak a z-tengellyel, így egyszerűbben végezhető a vágás

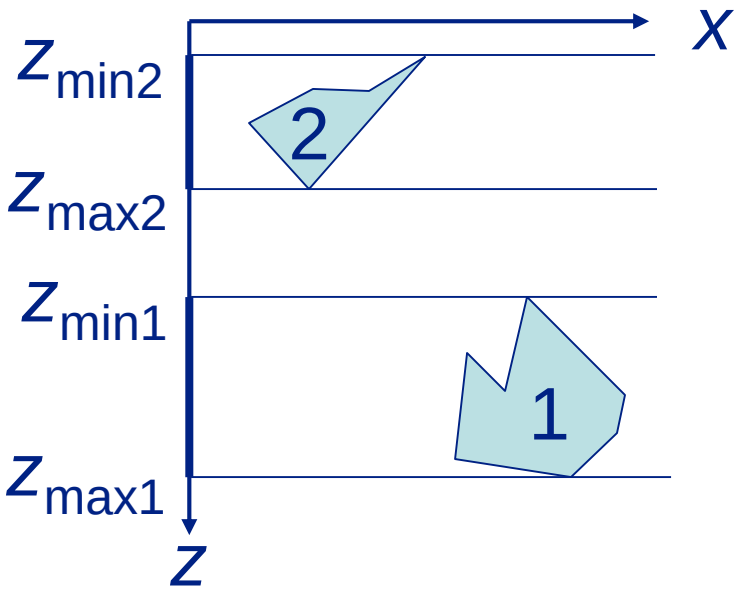
Határoló-téglalap teszt

- Ha a határoló téglalapok nem fedik egymást, akkor a vetületek sem fedik egymást, különben további vizsgálat szükséges



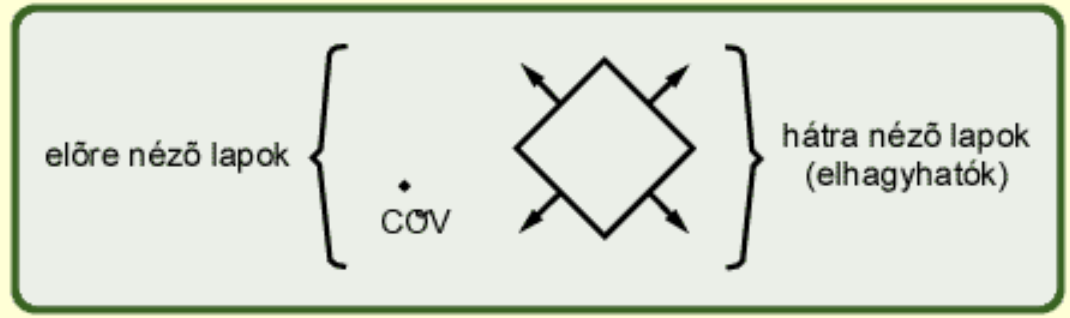
Min-max teszt

- 1-dimenziós kiterjedés (határoló intervallum)
- A kiterjedés minimális és maximális értékeinek összehasonlításával döntjük el a takarást
- A kiterjedés meghatározása: a tárgy (csúcs)pontjaihoz tartozó koordináták min. és max. értékeiből



Hátra néző lapok kiválogatása

- Tegyük fel, hogy a tárgy poligon határú síklapokkal határolt és adottak a síklapoknak a tárgyból kifelé mutató normálisai. Ekkor azok a lapok nem láthatók, amelyek normálisai a "megfigyelőtől ellentétes” irányba mutatnak



Hátra néző lapok kiválogatása

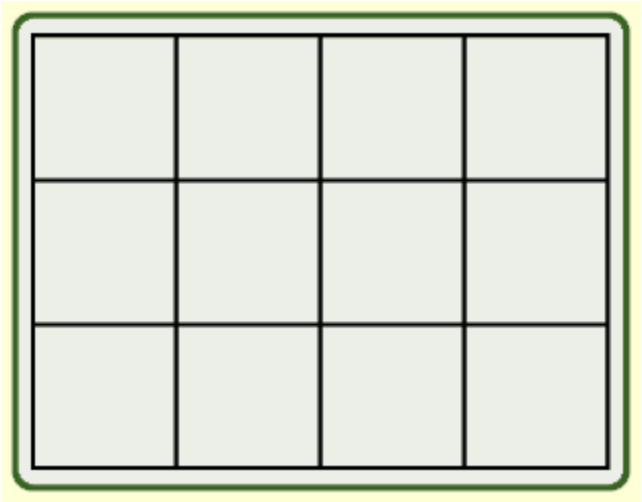
n : normális (n_x ,n_y ,n_z)
v : COV-ből a poligon tetszőleges pontjába mutat

Ha n ·v < 0 előre néz
> 0 hátra néz
= 0 csak az éle látszik

Speciálisan: Az (x,y) síkra történő ortografikus vetítés esetén
Ha n_z < 0 hátra néz
> 0 előre néz
= 0 csak az éle látszik

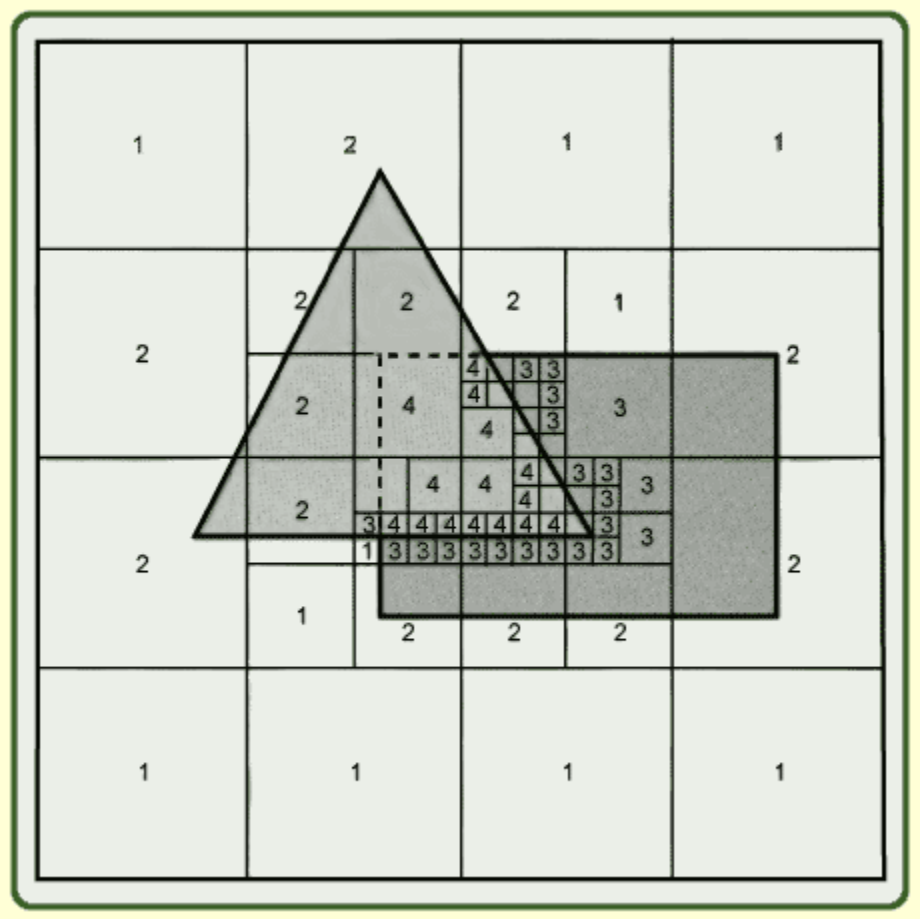
Térbeli partícionálás

- Észrevétel: nem minden tárgynak van minden vetítési sugárral metszéspontja (pl. távol vannak, más irány) → osszuk fel (partícionáljuk) a képernyőt
- Meghatározzuk, hogy mely tárgyak vetülete van benne a megfelelő részben (partícióban) és csak azokkal keresünk metszéspontokat



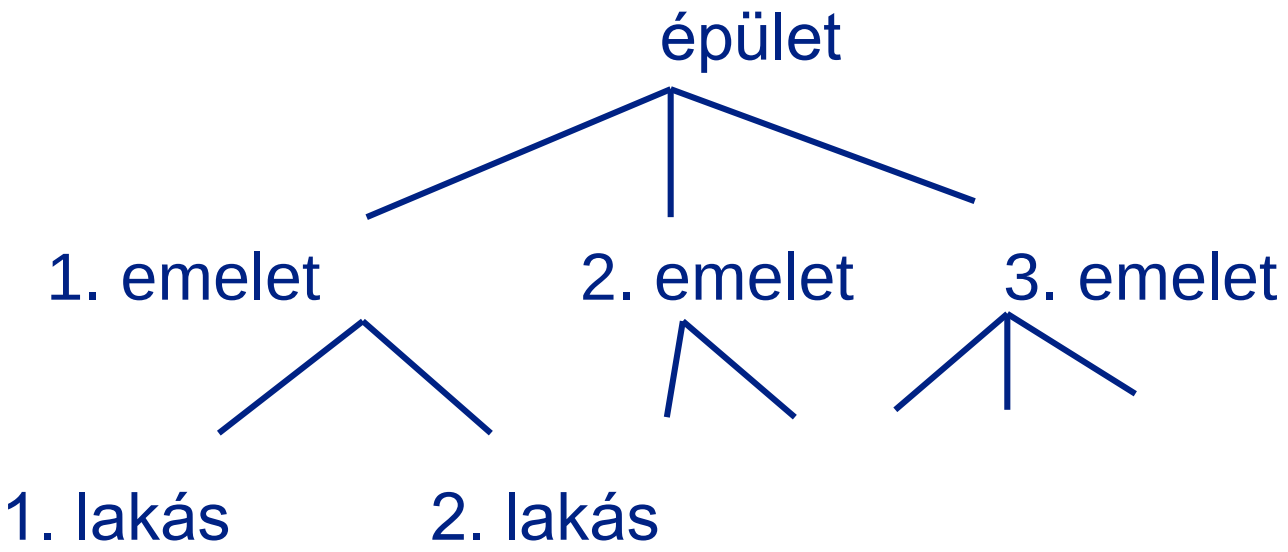
Térbeli partícionálás

- Ez jó módszer, ha a tárgyak vetületei egyenletesen oszlanak el a teljes képernyőn, különben különböző méretű partíciókat érdemes készíteni: kisebb partíciót ott, ahol több tárgy vetülete van



Hierarchikus struktúrák alkalmazása

pl.



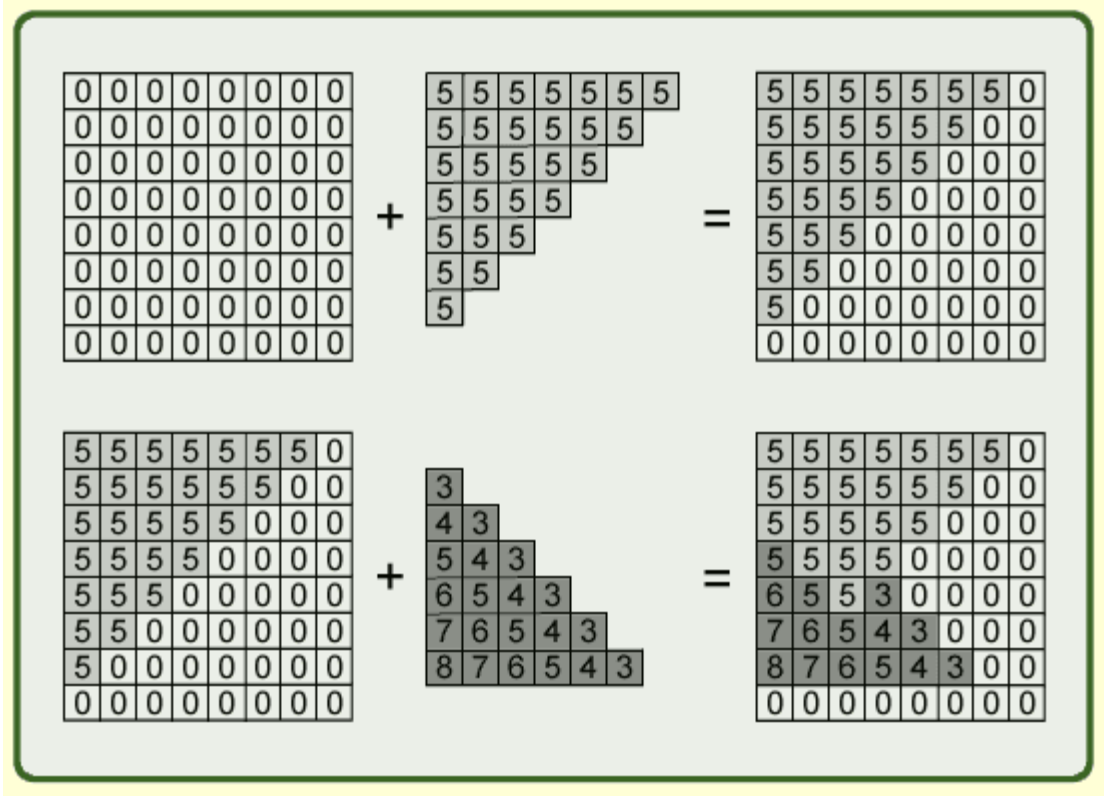
- Ha a vetítési sugár nem metszi az épületet, akkor az emeleteit és az emeletek lakásait sem (tehát nem kell vizsgálni azokat)

Z-buffer vagy mélység-puffer algoritmus

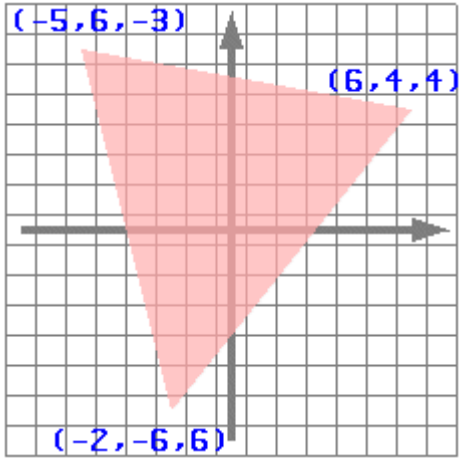
- Kép alapú
- F : kép-puffer (képpontok tárolására)
  - kezdeti értéke: háttérszín
- Z : mélység-puffer (minden pontban a megfelelő z érték)
  - kezdeti értéke: z-max (hátsó vágási sík)
- Pásztázás közben F-be és Z-be bekerül az új pont, ha nincs messzebb, mint az eddigi z érték

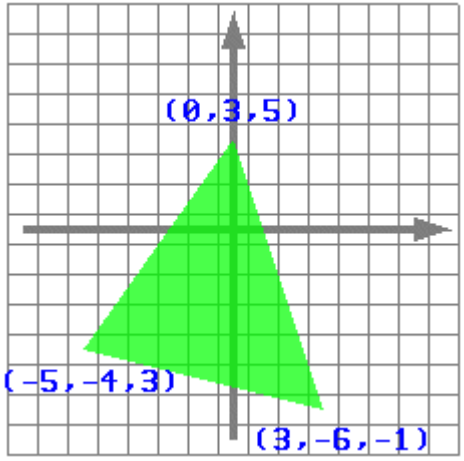
Mélység-puffer

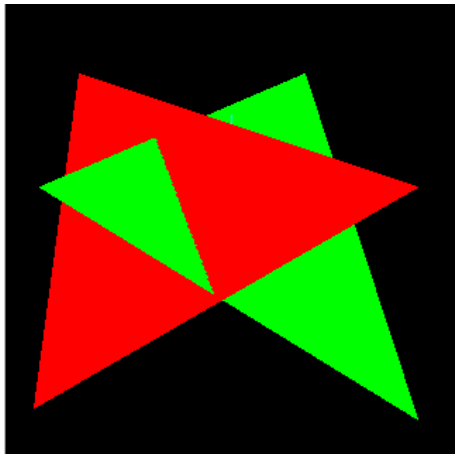
- A hátsó vágási sík a z = 0



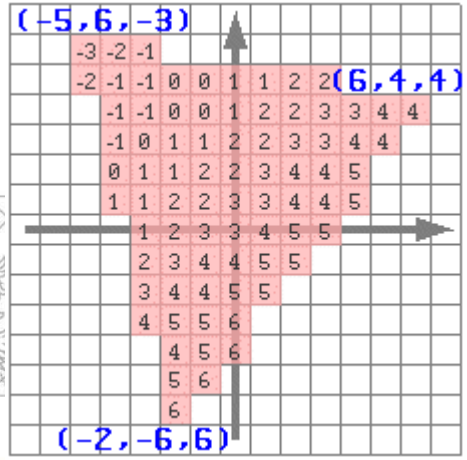
Z-buffer algoritmus

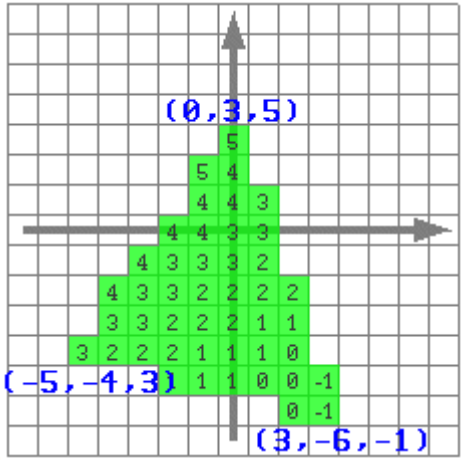


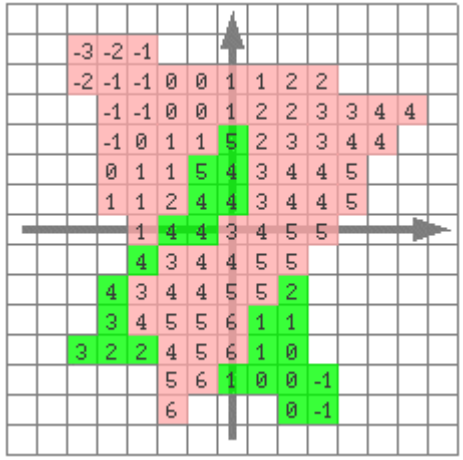




?





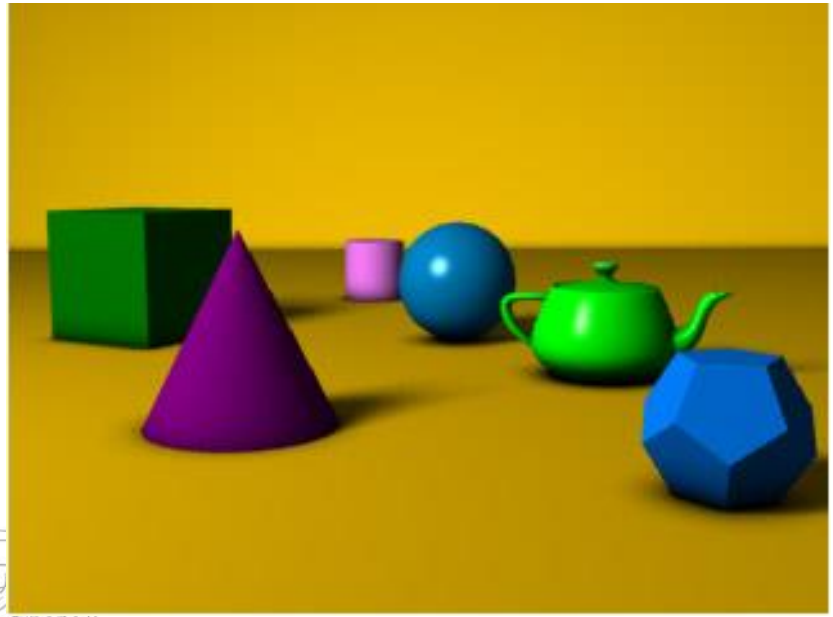


Tulajdonságok

- Nincs tárgyak rendezése, összehasonlítása, metszéspontok számítása
- Poligononként végezhető el, "poligonok összeadása"
- Nem csak poligonokra jó
- Nagy helyigény, de lehet sávonként haladni
- Könnyű implementálni
- Könnyű egy újabb tárgy képét hozzávenni és utána elvenni (maszkolás)
- Az értékek felhasználhatók terület és térfogat számításra

Z-buffer algoritmus

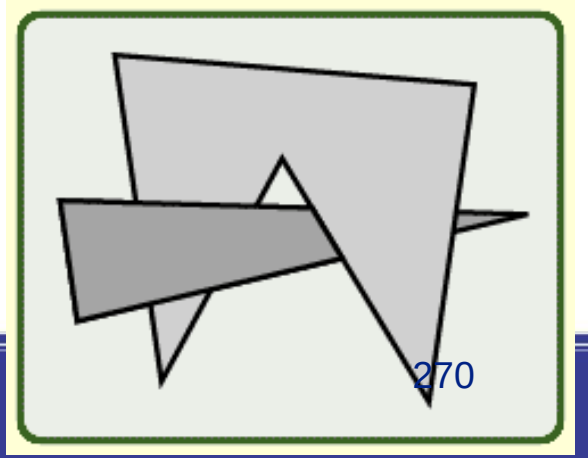
Szín puffer



Mélység puffer

Lista-prioritás algoritmusok

- Meghatározzák a tárgyaknak azt a sorrendjét, ami a kép kirajzolásához kell
- Pl.: Ha a z értékekben nincs átfedés, akkor a tárgyakat növekvő z értékük szerint kell rendezni és utána távolról közelre haladva megjeleníteni
- Néha még akkor is lehet ilyen sorrendet megadni, ha a z értékekben van átfedés, de nem mindig
- Ilyenkor szétvágjuk a tárgyakat és a darabokat rendezzük sorba

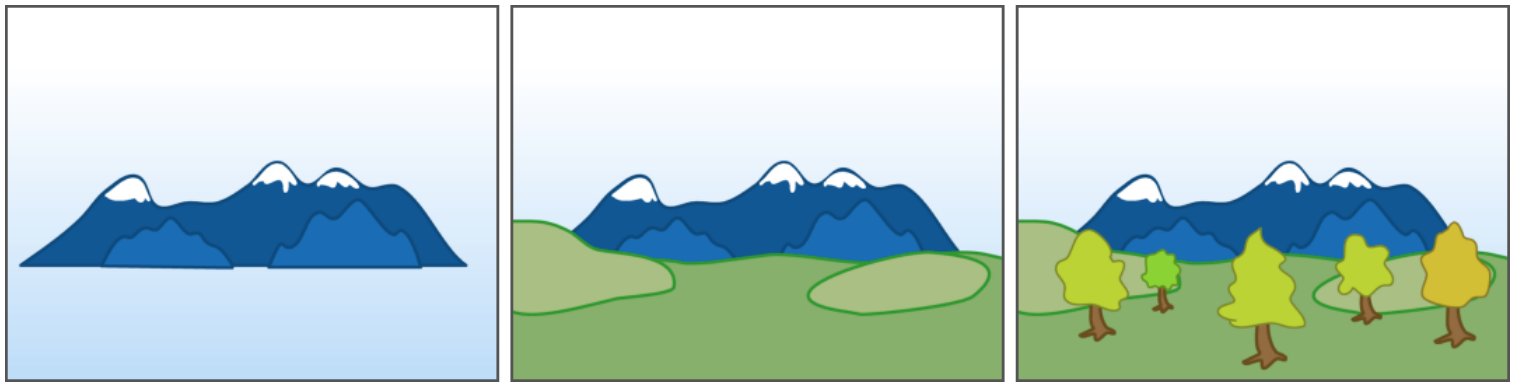


Mélység szerint rendező algoritmus

1. Rendezzük a poligonokat legtávolabbi z koordinátájuk szerint
2. Vágjuk szét az átfedő poligonokat (ha szükséges)
3. Pásztázzunk minden poligont hátulról előre haladva

- Ha a poligonok párhuzamos síkokban fekszenek, akkor a 2. lépés kimaradhat

Festő algoritmus



Mélység szerint rendező algoritmus

- Tegyük fel hogy a P poligon legtávolabbi z koordinátája szerint a lista végén van.
- Pásztázás előtt össze kell hasonlítani a lista azon Q elemeivel, amelyeknek z irányú kiterjedése átfedi P z irányú kiterjedését, és meg kell vizsgálni, hogy P átfedi-e Q-t?

P átfedi-e Q-t?

1. ha P és Q x kiterjedése nem átfedő, akkor nem;
2. ha P és Q y kiterjedése nem átfedő, akkor nem;
3. ha COV-ból nézve P teljes terjedelmében Q síkjának a túlsó oldalán van, akkor nem;
4. ha COV-ból nézve Q teljes terjedelmében P síkjának az innenső oldalán van, akkor nem;
5. ha P és Q xy síkra való vetülete nem átfedő, akkor nem;

különben (hátha Q-t kell előbb rajzolni):

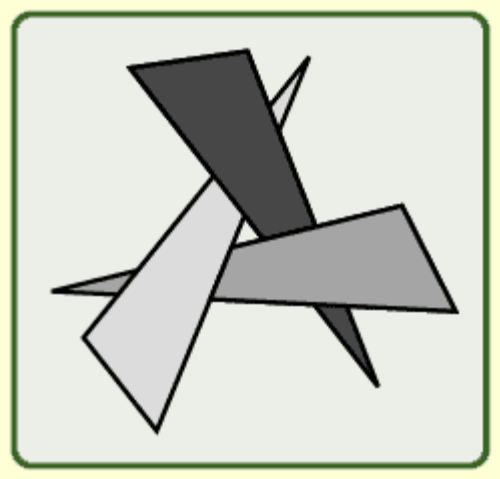
3‘ ha COV-ból nézve Q teljes terjedelmében a P síkjának a túlsó oldalán van, akkor P ↔ Q csere;

4‘ ha COV-ból nézve P teljes terjedelmében a Q síkjának az innenső oldalán van, akkor P ↔ Q csere;

különben P-t vagy Q-t fel kell darabolni a másik síkjával majd feldarabolt helyett a darabokat kell beilleszteni a listába

Fedési hurkok

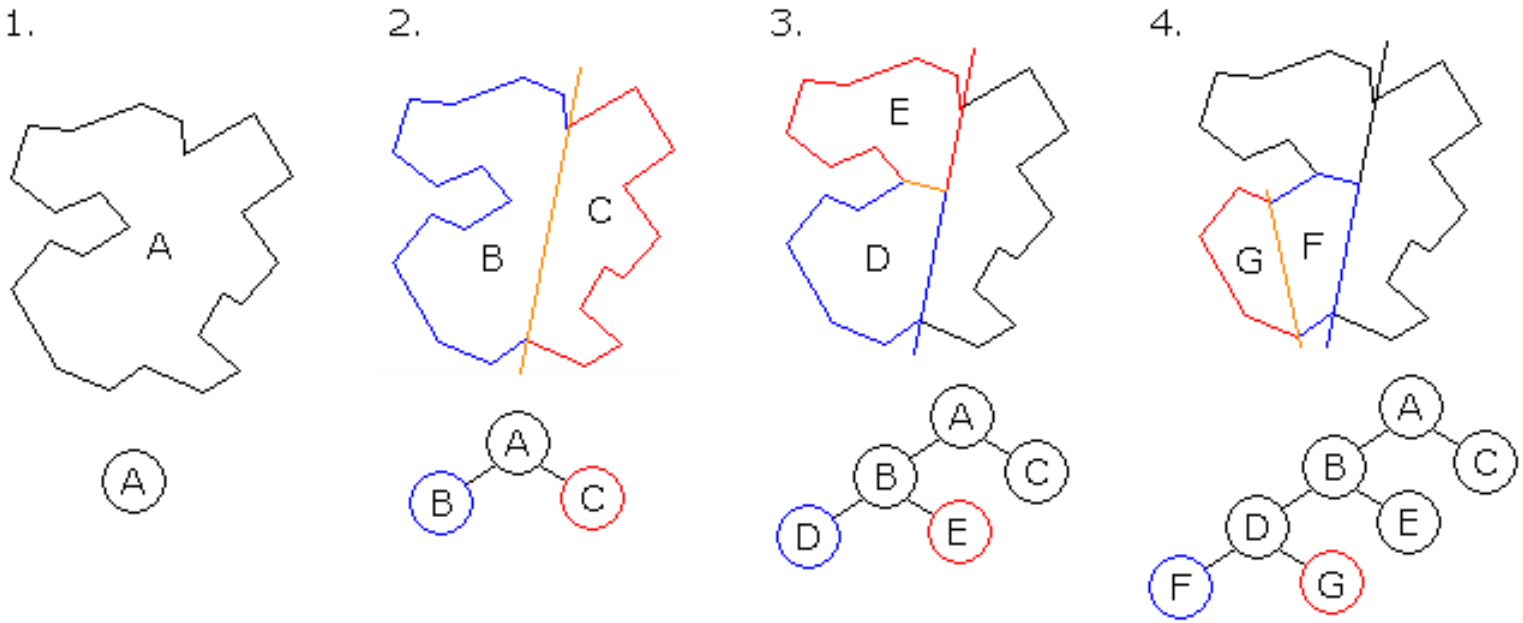
- Végtelen ciklust eredményezne, ezért megjelöljük azokat a poligonokat, amelyeket egyszer már a lista végére tettünk és ha újra előjönnek, akkor darabolunk



Bináris tér-partícionáló (BSP) fa algoritmus

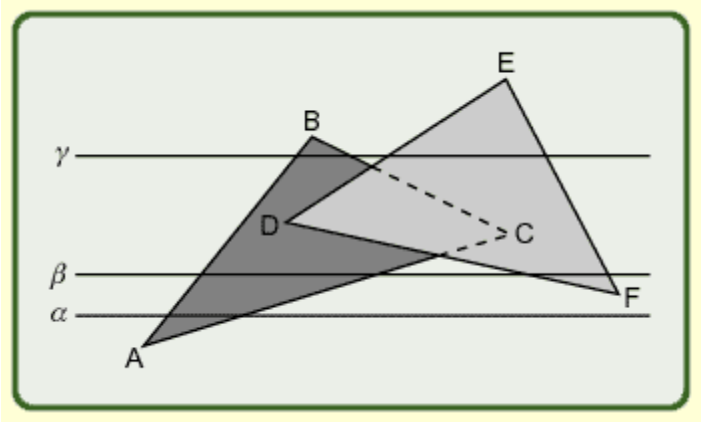
- Ötlet: Ha van olyan sík, amely a tárgyakat (teljes egészükben) két féltérbe osztja, akkor a COV-t tartalmazó féltér tárgyait nem takarhatják el a másik féltér tárgyai
- BSP fa: Csomópontok - poligonok (darabjai)
  - bal oldalra: azok a poligonok, amelyek elöl vannak (később kell rajzolni)
  - jobb oldalra: azok a poligonok, amelyek hátul vannak (korábban kell rajzolni)
- A csomóponthoz tartozó poligon síkjával darabolhatjuk a többi poligont és azok darabjaival folytathatjuk a fát

Bináris tér-partícionálás



Pásztázó vonal algoritmus

- Hasonló a poligonok kitöltését végző algoritmushoz
- Vízszintesen pásztázunk



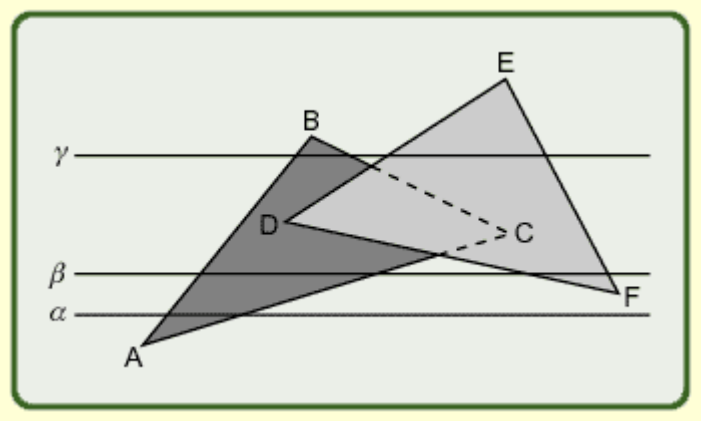
- Most több poligon lehet

ÉT (élek táblázata)

- A poligonok nem vízszintes éleit tartalmazza az élek kisebbik y-koordinátája szerint növekvő, ezen belül az élek meredeksége szerint csökkenő sorrendbe rendezve
- Egy elem részei:
  - a kisebb y-koordinátájú csúcspont x-koordinátája
  - a másik csúcspont y-koordinátája
  - x növekménye: $\Delta x = 1/m$
  - poligon azonosító

PT (poligonok táblázata)

- Egy elem részei:
  - a sík egyenletének együtthatói: Ax + By + Cz + D = 0
  - színezési információ
  - ki-be jelző (kezdő érték: ki)
  - azonosító



- ÉT: AB, AC, FD, FE, CB, DE
- PT: ABC, DEF

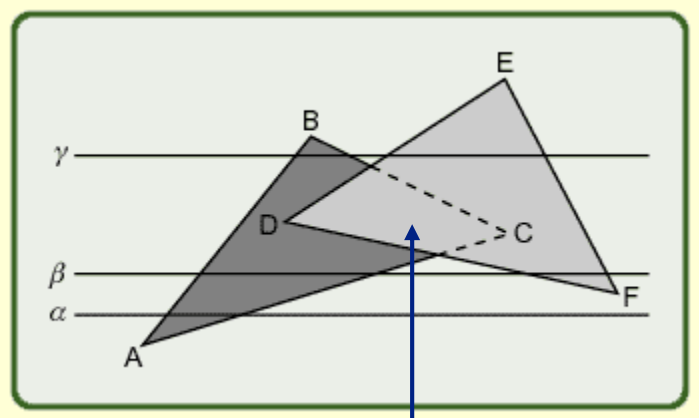
AÉT (aktív élek táblázata)

α: AB, AC
ABC
be ki

β: AB, AC, FD, FE
ABC DEF
be ki be ki

γ: AB, DE, CB, FE
ABC
be ki
DEF
be ki

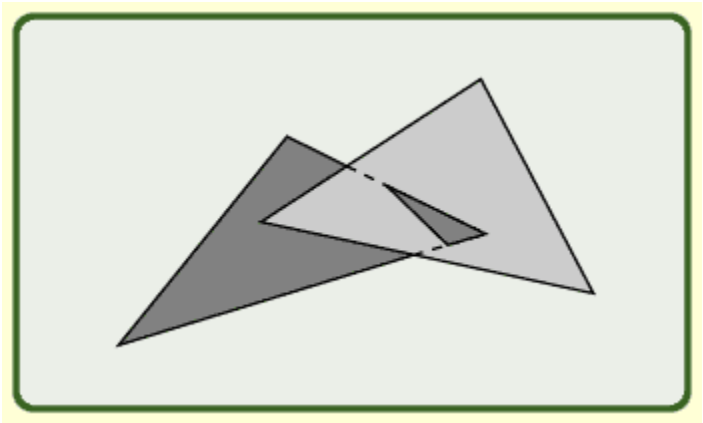
- Balról jobbra, alulról felfelé haladva



- A síkok egyenletéből dönthető el, hogy melyik van közelebb

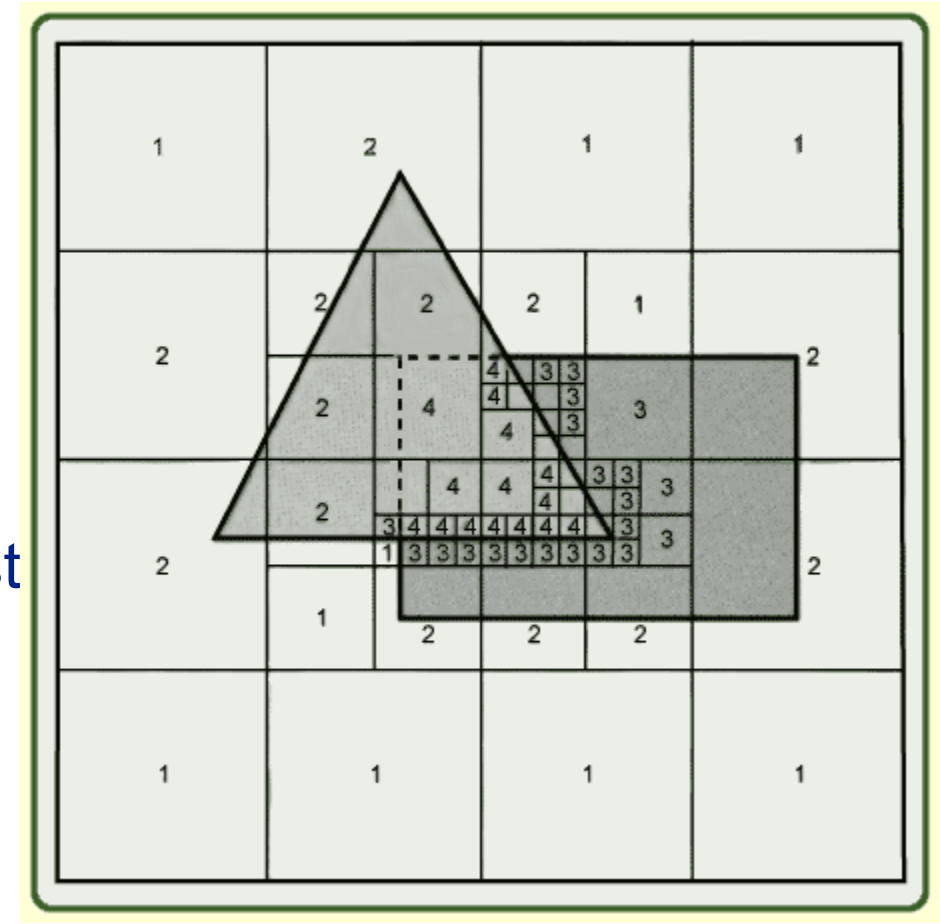
Gyorsítás

- Ha a poligonokat felvágjuk a metszeteik mentén, akkor nem kell minden pontban megvizsgálni a poligonok sorrendjét, elegendő csak akkor, ha egy "takaró" poligon véget ér

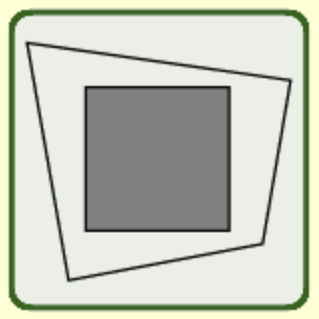


Terület-osztó algoritmus

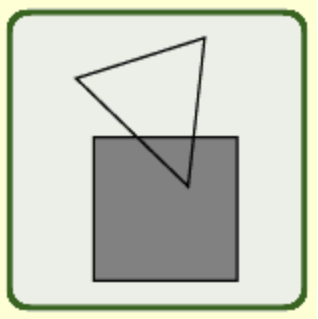
- Ha egy területen könnyen eldönthető, hogy melyik poligon jeleníthető meg, akkor azt rajzoljuk ki, különben osszuk fel a területet, és alkalmazzuk az eljárást a rész területekre



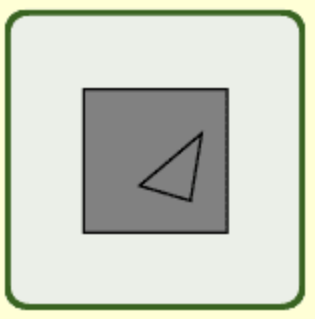
Poligon és téglalap viszonya



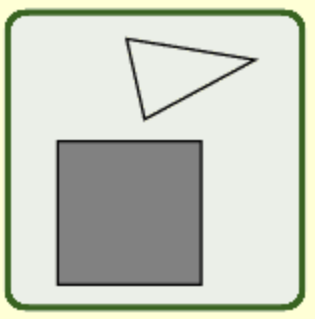
Tartalmazó poligon



Metsző poligon



Tartalmazott poligon



Idegen poligon

Mikor dönthető el könnyen, hogy mi rajzolható?

- Minden poligon idegen a területtől (háttér)
- Egyetlen metsző vagy tartalmazott poligon (háttér + pásztázással poligon)
- Egyetlen tartalmazó poligon (rajz a poligon színével)
- Van olyan tartalmazó poligon, amelyik a többi előtt van

Látható élek meghatározása

- Tárgy-alapú módszerek
- Output: látható élek listája

Robert-féle algoritmus

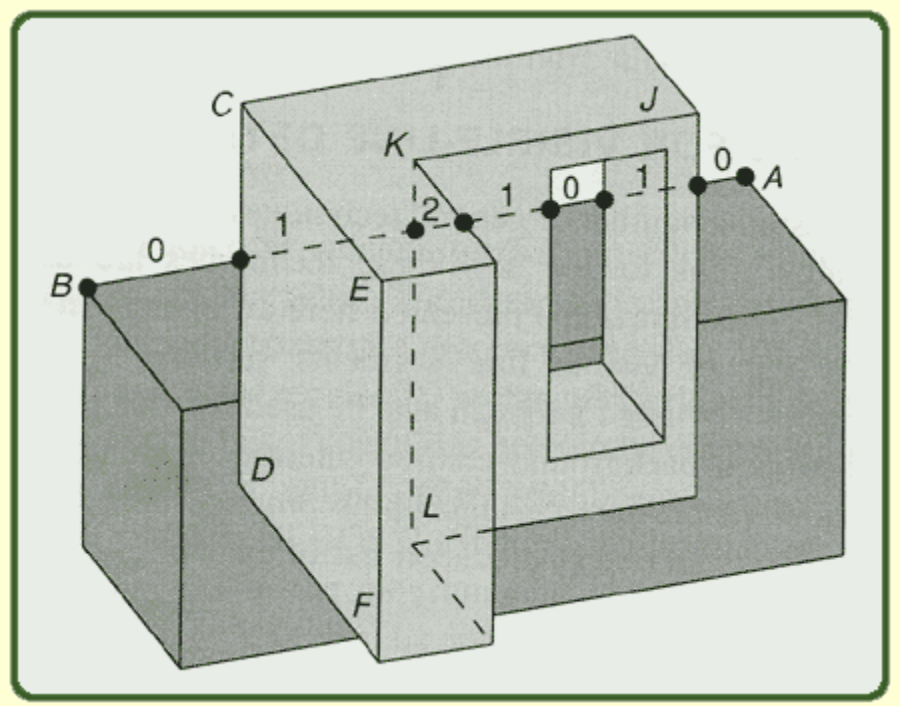
- Síklapokkal határolt konvex testek éleire

1. A hátrafelé néző lapok meghatározása
2. A hátrafelé néző lapok közös élei elhagyhatók (azok nem láthatók)
3. Minden megmaradt élt minden testtel összehasonlítunk (kiterjedés vizsgálattal sok test triviálisan kizárható)

   Esetek:
   - Az élet egy test teljesen eltakarja
   - Az élnek egy szakasza látszik a test mögül
   - Az élnek két szakasza látszik a (konvex) test mögül

Appel-féle algoritmus

- Az élek pontjaihoz hozzárendel egy egész számot (kvantitatív láthatatlanság): a pontot takaró előre néző lapok száma
- A pont látható ↔ kvantitatív láthatatlanság = 0



Appel-féle algoritmus

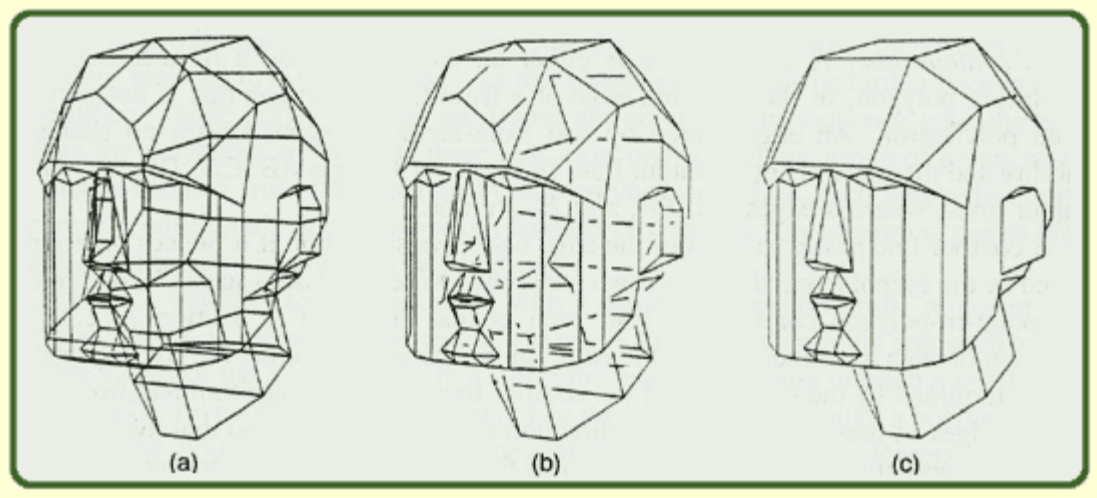
- A kvantitatív láthatatlanság számítása:
  - ++, ha az él előre néző poligon mögé megy
  - - -, ha az él előre néző poligon mögül jön ki
- A kvantitatív láthatatlanság csak akkor változik, ha az él egy un. kontúr vonal mögött halad
- Kontúr vonal: előre és hátra néző lap közötti él

Appel-féle algoritmus

- Az algoritmus kétféle megvalósítása:
  1. Kiválasztunk egy csúcspontot, meghatározzuk a kvantitatív láthatatlanságát (direkt módszer)
  2. Haladjunk az éleken és közben módosítsuk a kvantitatív láthatatlansági értéket, 0 érték esetén rajzolunk

Megszakított vonalak

- A látható vonal algoritmusok arra is használhatók, hogy a nem látható vonalak szaggatottak, pontozottak, halványabbak vagy megszakítottak legyenek



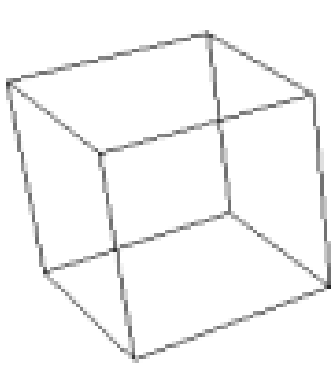
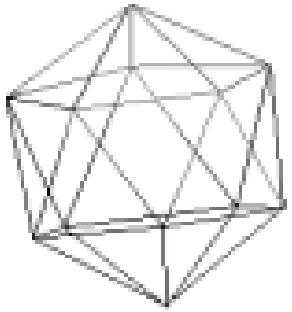
- (b): mintha minden vonalnak lenne egy takaró sávja, ami eltakarja a mögötte lévő részeket

Megszakított vonalak

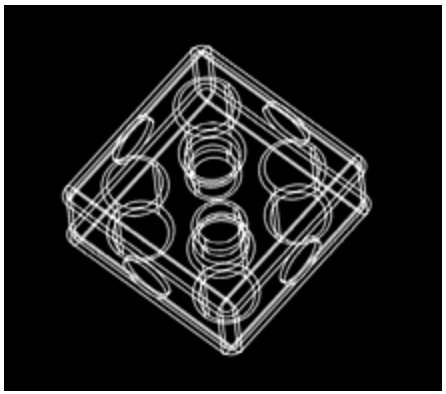
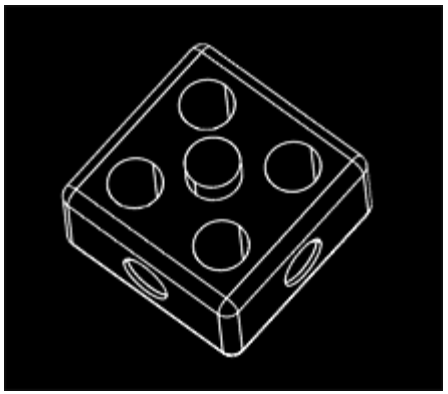
- A megszakított vonalak rajzolásához csak a metszéspontok környezetében kell módosítani az algoritmust

- Algoritmus:
  - Minden vonalhoz megkeressük az előtte levőket
  - Csak a látható szakaszokat őrizzük meg
  - Ha minden metszésponttal végeztük, akkor rajzolunk

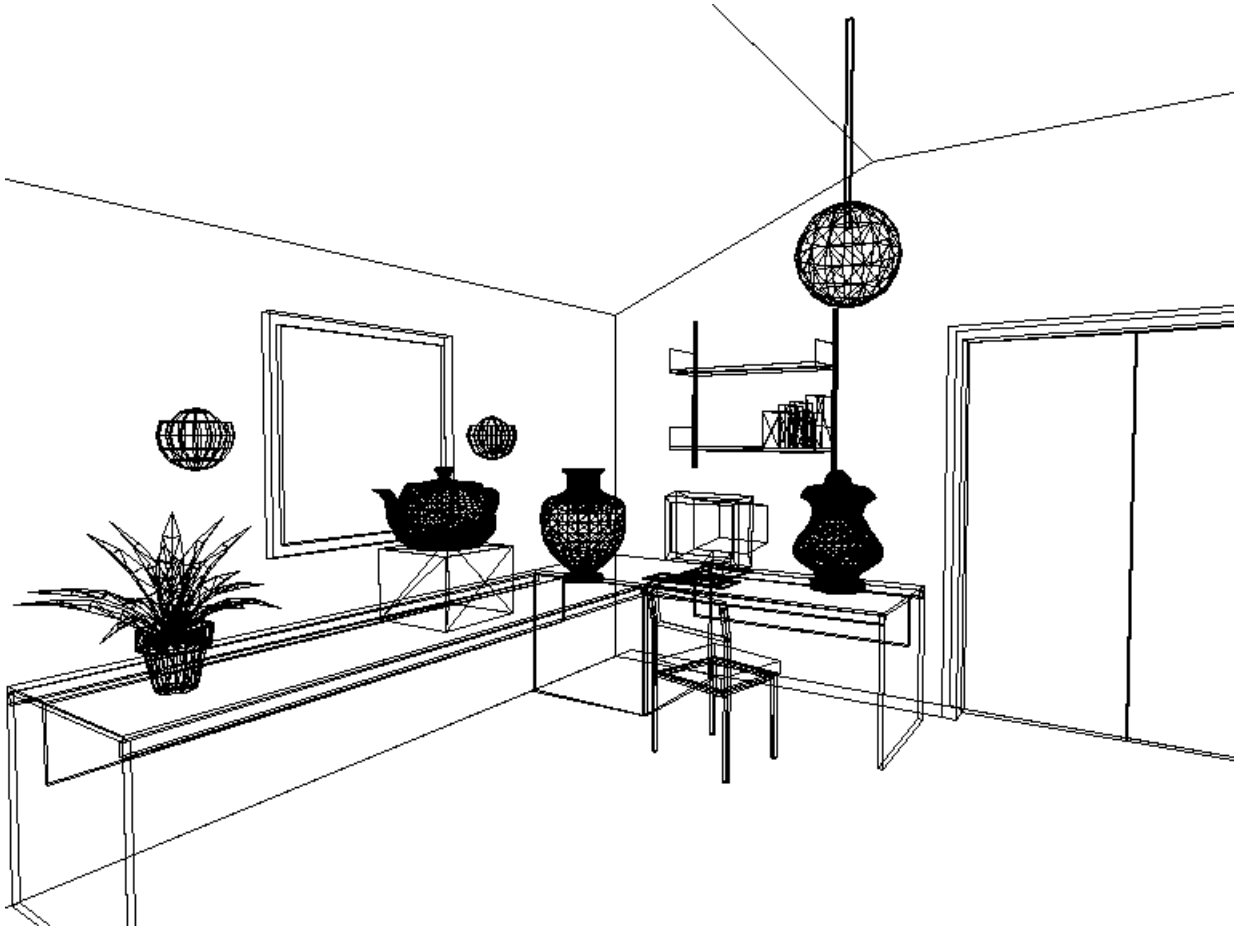
Példák



Példák



Példák