Számítógépes grafika

Bevezetés
Történeti áttekintés
„Hordozható” szoftverek, szabványok
Interaktív grafikai rendszerek
A számítógépes grafika osztályozása

1

Bevezetés

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

2

Bevezetés

Számítógépes képfeldolgozás:
Képek *analízise*, objektumok modelljeinek rekonstrukciója képeikből (pl. légi-, űr-, orvosi felvételek kiértékelése, torzított képek helyreállítása)

3

Bevezetés

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

Fejlődés:

Fotózás → televízió → számítógépes grafika

4

Bevezetés

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

5

Történeti áttekintés

Kezdetben: képek megjelenítése teletype-on, nyomtatókon

1950:

MIT: számítógéppel vezérelt képernyő

SAGE légvédelmi rendszer (a programok képernyőről történő vezérlése fényceruzával)

6

Történeti áttekintés II

1963:

A modern interaktív grafika megjelenése

I. Sutherland: Sketchpad

Adatstruktúrák szimbolikus struktúrák tárolására

Interaktív megjelenítés, választás, rajzolás

7

Történeti áttekintés III

1964:

CAD – DAC-1 (IBM)

Autók tervezésére (General Motors)

8

Történeti áttekintés IV

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

9

Történeti áttekintés V

1960-as évek:

Jellemző output-eszköz az ún.

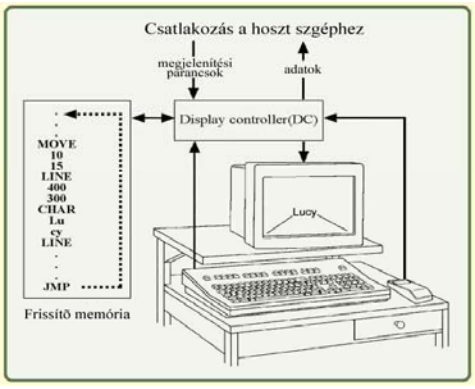
<u>vektor-képernyő</u> (szakaszokat rajzol -tól -ig)

Részei:

- Képernyő processzor (DP) - mint I/O periféria kapcsolódik a központi egységhez
- Képernyő tároló memória – a megjelenítéshez szükséges program és adat tárolására
- Képernyő - katód sugár cső

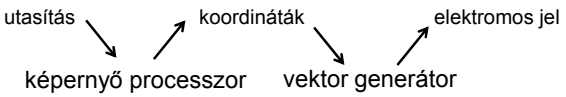
10

Történeti áttekintés VI



11

Történeti áttekintés VII



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

12

Történeti áttekintés VIII

1968:

A hardver képes a skálát változtatni, a képet mozgatni, vetületeket előállítani valós időben

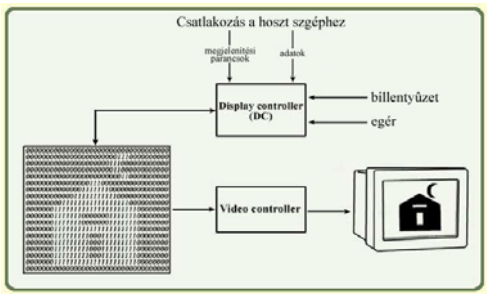
1970-es évek:

Jellemző output eszköz az un. **raszter-képernyő** (TV - technika), bit-térképes grafika

Bit-térkép (bitmap):
képek reprezentálása bináris mátrixszal

13

Történeti áttekintés IX



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

14

Történeti áttekintés X

mátrix -- raszter sorok -- képpontok

Bit-térképek, pl.:

*1024 * 1024 * 1 = 128* K - bináris kép

Pixel-képek, pl.:

*1024 * 1024 * 8 = 256* szürkeségi fokozat v. szín

*1024 * 1024 * 24 = 2^{24}* szürkeségi fokozat v. szín

Ma tipikus:

*1280 * 1024 * 24 ≈ 3.75* MB RAM

15

Történeti áttekintés XI

Előnyei:

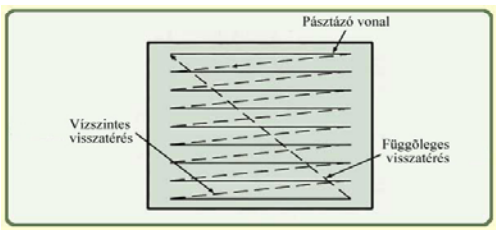
- Olcsó logikájú processzor (soronként olvas)
- A területek színekkel kitölthetők
- Az ábra bonyolultsága nem befolyásolja a megjelenítés sebességét

16

Történeti áttekintés XII

Hátrányai:

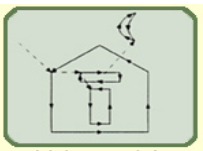
- A grafikus elemeket (pl. vonal, poligon) át kell konvertálni (RIP - raster image processor)
- A geometriai transzformációk számításigényesek



17

Megjelenítés raszteres képernyőn


Ideális vonalas rajz


Vektoros kép


Raszteres kép vonallal


Raszteres kép területkitöltéssel

18

Történeti áttekintés XIII

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, ...)

19

„Hordozható" szoftverek, szabványok

Eszköz-függő —fejlődés→ eszköz független

Így lehet "hordozható" a felhasználói szoftver

1977:

3D Core Graphics System

1985:

GKS (Graphical Kernel System) 2D

20

„Hordozható" szoftverek, szabványok

1988:

GKS - 3D

PHIGS (Programmer's Hierarchical Interactive Graphics System)

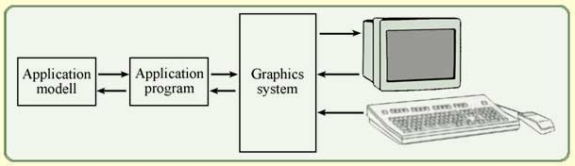
- Logikailag kapcsolódó primitívek csoportosítása szegmensekbe,
- 3D primitívek egymásba ágyazott hierarchiája,
- Geometriai transzformációk,
- Képernyő automatikus frissítése, ha az adatbázis változik

1992

OpenGL (SGI)

21

Interaktív grafikai rendszerek



Interaktivitás:

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

22

Interaktív grafikai rendszerek II

Felhasználói modell (adatbázis):

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

23

Interaktív grafikai rendszerek III

Az interaktivitás kezelése:

Tipikus az esemény-vezérelt programhurok:

```
kezdeti képernyő beállítás;
while(true) {
  parancsok vagy objektumok választhatók;
  várakozás, amíg a felhasználó választ;
  switch(válaszás){
    case 'választott': a modell és a
     képernyő frissítésére; break;
    ...
    case(quit) exit(0);
}
```

24

A számítógépes grafika osztályozása I

Dimenzió szerint:

- 2-D
- 3-D

Képfajta szerint:

- vonalas
- szürke
- színes (árnyékolt)

25

A számítógépes grafika osztályozása II

Interaktivitás szerint:

- Off-line rajzolás
- Interaktív rajzolás (változó paraméterek)
- Objektum előre meghatározása és körüljárása
- Interaktív tervezés

Kép szerepe szerint:

- Végtermék
- Közbülső termék

26

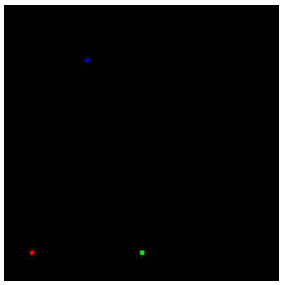
OpenGL

Pontok rajzolása

27

Pontok rajzolása

Rajzoljunk egy piros pontot a (10, 10), egy zöld pontot az (50, 10) és egy kék pontot a (30, 80) koordinátákba (az ablak 100*100-as méretű)



28

Színek és színmódok

RGBA színmód:
Minden színt négy komponens definiál: (**R**, **G**, **B**, **A**)
vörös (Red), zöld (Green), kék (Blue), alfa (Alpha)

Minél nagyobb az **RGB** komponens értéke, annál intenzívebb a színkomponens

A (átlátszóság): 1.0 - nem átlátszó,
0.0 - teljesen átlátszó

Pl.: (0.0, 0.0, 0.0, 0.0) – átlátszó fekete

29

Törlő szín

```
void glClearColor(
     GLclampf red,
     GLclampf green,
     GLclampf blue,
     GLclampf alpha);
```

Aktuális törlő szín beállítása
Alapértelmezés: `(0.0, 0.0, 0.0, 0.0)`

`GLclampf - float`

30

Mátrix mód beállítása

Transzformációk: mátrixokkal definiálva
nézeti (viewing),
modellezési (modelling),
vetítési (projection)

```
void glMatrixMode(enum mode);
```

Ha *mode* == `GL_PROJECTION`, akkor vetítési mátrix
pl.:

```
glMatrixMode(GL_PROJECTION);
```

```
void glLoadIdentity(void);
```

az érvényes mátrix az egységmátrix lesz

31

Vetítési mátrix megadása (2D)

```
void gluOrtho2D(double left, double
    right, double bottom, double top);
```

az objektumok 2D merőleges vetítése a
(*left, right, bottom, top*) téglalapra
pl.:

```
gluOrtho2D(0, 100, 0, 100);
```

32

Program (pontrajzoló) I

```
#include <GL/glut.h>
void init(void) {
     glClearColor(0.0,0.0,0.0,0.0);
   // fekete a törlőszín
     glMatrixMode(GL_PROJECTION);
       // az aktuális mátrix mód: vetítés
     glLoadIdentity();
       // legyen az egységmátrix
     gluOrtho2D(0,100,0,100);
    // párhuzamos vetítés specifikálása
}
```

33

Pufferek törlése

```
void glClear(GLbitfield mask);
```

Pufferek tartalmának a törlése
A pufferek:

`GL_COLOR_BUFFER_BIT`,
`GL_DEPTH_BUFFER_BIT`,
`GL_STENCIL_BUFFER_BIT` vagy
`GL_ACCUM_BUFFER_BIT`

Pl. a szín puffer törlése az aktuális törlőszínnel:

```
glClear(GL_COLOR_BUFFER_BIT);
```

34

Objektumok megadása

```
void glBegin(enum mode);
. . .
void glEnd(void);
```

geometriai objektumok specifikációja

mode értéke lehet pl.
`POINTS, LINES, POLYGON`

35

Színbeállítás

```
void glColor{34}{bsifd ubusui}
     (T components);
```

`b` byte, `s` single, `i` integer, `f` float, `d` double, `u` unsigned

Színbeállítás csúcspontokhoz van hozzárendelve
Pl.

```
     glColor3f(1.0,0.0,0.0);      //
piros
     glColor3f(0.0,1.0,0.0);      // zöld
     glColor3f(0.0,0.0,1.0);      // kék
```

36

Csúcspontok megadása

```
void glVertex{234}{sifd}( T coords );
```

Csúcspont(ok) (vertex) megadása

Pl.:

```
    glVertex2i(10,10);
        // a pont koordinátája (10, 10)
```

37

Program (pontrajzoló) II

```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
              // képernyő tötlés
    glBegin(GL_POINTS);
              // pontokat specifikálunk
         glColor3f(1.0,0.0,0.0);  // piros
         glVertex2i(10,10); // piros pont
         glColor3f(0.0,1.0,0.0);  // zöld
         glVertex2i(50,10);  // zöld pont
         glColor3f(0.0,0.0,1.0);  // kék
         glVertex2i(30,80);  // kék pont
    glEnd();       // több pont nem lesz
    glFlush();     // rajzolj!
}
```

38

Program (pontrajzoló) III

```
void keyboard(unsigned char key,
              int x, int y){
  switch(key) {  // billentyű kezelés
    case 27:     // ha escape
       exit(0);  // kilép a programból
       break;
  }
}
```

39

Képernyő mód

```
void glutInitDisplayMode
    (unsigned int mode);
```

A képernyő módot definiálja

Pl. ha ***mode***

```
    GLUT_SINGLE | GLUT_RGB
```

akkor az ún. *egyszeresen pufferelt*, ***RGB*** *módban* specifikál ablakot

40

Ablak

```
void glutInitWindowSize
    (int width, int height);
```

Az ablak méretét definiálja pixelekben

```
void glutInitWindowPosition(int x, int y);
```

Az ablak bal felső sarkának pozíciója

```
int glutCreateWindow(char *name);
```

Megnyit egy ablakot az előző rutinokban specifikált jellemzőkkel. Ha az ablakozó rendszer lehetővé teszi, akkor ***name*** megjelenik az ablak fejlécén. A visszatérési érték egy egész, amely az ablak azonosítója.

41

Callback függvények

```
void glutDisplayFunc(void(*func)(void));
```

Azt a ***callback*** függvényt specifikálja, amelyet akkor kell meghívni, ha az ablak tartalmát újra akarjuk rajzoltatni. Pl.:

```
    glutDisplayFunc(display);
```

```
void glutKeyboardFunc(void(*func)
    (unsigned char key, int x, int y);
```

Azt a ***callback*** függvényt specifikálja, melyet egy billentyű lenyomásakor kell meghívni. ***key*** egy **ASCII** karakter. Az ***x*** és ***y*** paraméterek az egér pozícióját jelzik a billentyű lenyomásakor (ablak relatív koordinátákban). Pl.:

```
    glutKeyboardFunc(keyboard);
```

42

Program (pontrajzoló) IV

```
int main(int argc, char* argv[]) {
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
         //az ablak egyszeresen pufferelt,és RGB módú
     glutInitWindowSize(100, 100); // 100x100-as
     glutInitWindowPosition(100, 100);
         // az ablak bal felső sarkának koordinátája
     glutCreateWindow("3point");  // neve 3point
     init();                      // inicializálás
     glutDisplayFunc(display);
        // a képernyő események kezelése
     glutKeyboardFunc(keyboard);
        // billentyűzet események kezelése
     glutMainLoop();  // belépés az esemény hurokba
     return 0;
}
```

43

ALGORITMUSOK RASZTERES GRAFIKÁHOZ

Egyenes rajzolása

Kör rajzolása

Ellipszis rajzolása

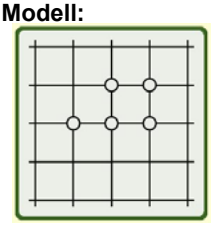
44

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:

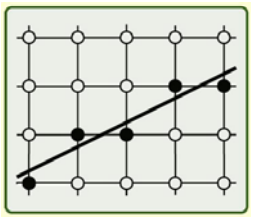


képpont (= körlap), amely a négyzetháló csúcspontjaiban helyezhető el.
A koordináták: egész számok

45

Egyenes rajzolása

Tegyük fel, hogy "vékony" egyenes: $y = mx + b$
meredeksége: $0 < m < 1$
($m = 0,1,...$ triviális speciális esetek)
más esetekben visszavezetjük $0 < m < 1$ -re



Legyen: $x_0 < x_1$, $y_0 < y_1$

46

Egyenes rajzolása

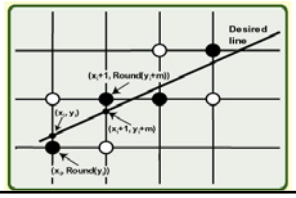
1. Alap inkrementális algoritmus

(x_0, y_0)-t kirajzoljuk. Haladjunk $\Delta x = 1$ növekménnyel balról jobbra, válasszuk a legközelebbi képpontot:

$$(x_{i+1}, [y_{i+1}+0.5]) = (x_{i+1}, [mx_{i+1}+b+0.5])$$

A szorzás kiküszöbölhető inkrementálással:

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



47

Alap inkrementális algoritmus

Algoritmus:

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

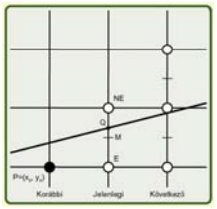
```
void Line(int x0, int y0,
          int x1, int y1, int value) {
  int x;
  double dy, dx, y, m;
  dy = y1-y0; dx = x1-x0;
  m = dy/dx; y = y0;
  for(x = x0; x < x1; x++) {
    WritePixel(x, Round(y), value);
    y += m
  }
} // Line
```

48

Egyenes rajzolása

2. Felezőpont algoritmus egyenesre

egész aritmetika elegendő (Bresenham)



Elv: Azt a képpontot válasszuk *NE* és *E* közül, amelyik a *Q* metszésponthoz közelebb van.

Másképp: a választásban az döntsön, hogy *Q* az *M* **felezőpont** melyik oldalán van.

Tegyük fel, hogy:

$x_0 < x_1$, $y_0 < y_1$

49

Felezőpont algoritmus egyenesre

Az (*x0, y0*) és (*x1, y1*) ponton átmenő egyenes egyenlete: $(x - x0) / (x1 - x0) = (y - y0) / (y1 - y0)$

innen: $(x - x0)(y1 - y0) - (y - y0)(x1 - x0) = 0$

Legyen $dx = x1 - x0 \; (> 0)$, $dy = y1 - y0 \; (> 0)$,

akkor: $(x - x0)\,dy - (y - y0)\,dx = 0$

innen: $x\,dy - x0\,dy - y\,dx + y0\,dx = 0$

Legyen: $F(x,y) = x\,dy - x0\,dy - y\,dx + y0\,dx$

Világos, hogy

$F(x,y)$ { > 0, ha az egyenes (*x, y*) fölött fut,
$= 0$, ha (*x, y*) az egyenesen van,
< 0, ha az egyenes (*x, y*) alatt fut.

50

Felezőpont algoritmus egyenesre

$F(x,y) = x\,dy - x0\,dy - y\,dx + y0\,dx$

(x_p,y_p) rajzolása után a **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***

***N**orth (észak)*
***E**ast (kelet)*

d változása a következő pontnál:

ha előzőleg ***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 előzőleg ***NE***-t választottuk, akkor

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

51

Felezőpont algoritmus egyenesre

$F(x,y) = x\,dy - x0\,dy - y\,dx + y0\,dx$

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

$2F(x,y) = 2\cdot(x\cdot dy - y\cdot dx + y0\cdot dx - x0\cdot dy)$

függvényt, ennek az előjele megegyezik F előjelével, és ekkor $d_{start} = 2dy - dx$ már egész szám.

52

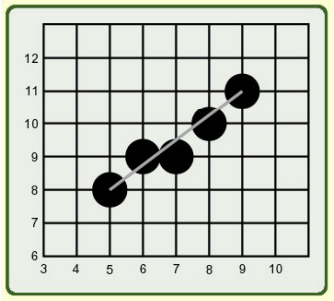
Felezőpont algoritmus egyenesre

```
void MidpointLine(int x0, int y0,
                  int x1, int y1, int value) {
  int dx, dy, incrE, incrNE, d, x, y;
  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) {
    if(d <= 0) {
       x++; d += incrE;
    } else {
       x++; y++; d += incrNE;
    }
    WritePixel(x, y, value);
  } // while
} // MidpointLine
```

53

Felezőpont algoritmus egyenesre

Eredmény: pl.



Tulajdonságok:

- csak összeadás és kivonás
- általánosítható körre, ellipszisre

54

Egyenes rajzolása

Megjegyzés:

Nem mindig lehet csak balról jobbra haladva rajzolni az egyeneseket.
Pl. szaggatott vonallal rajzolt zárt poligon

Problémák:

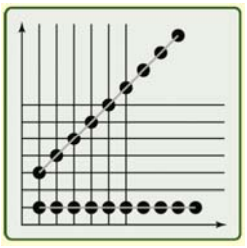
1. Különböző pontsorozat lesz az eredmény, ha balról jobbra, vagy ha jobbról balra haladunk.

Legyen a választás:
balról jobbra: $d = 0$ → *E*-t választani
jobbról balra: $d = 0$ → *SW*-t választani

55

Egyenes rajzolása

2. A vonal pontjainak a sűrűsége függ a meredekségétől



Megoldás: - intenzitás változtatása,
- kitöltött téglalapnak tekinteni az egyenes pontjait

56

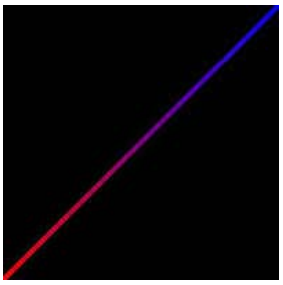
OpenGL

Egyenes szakasz rajzolása

57

Program (szakaszrajzoló) I

Rajzoljunk egy 5 pixel vastagságú egyenest, melynek egyik végpontja piros, a másik kék!



58

Program (szakaszrajzoló) II

```
void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLineWidth(5.0); // 5 pixel vastag vonal
  glShadeModel(GL_SMOOTH);
  glBegin(GL_LINES);
     glColor3d(1.0,0.0,0.0); //A piros végpont
     glVertex2d(0.0,0.0);
     glColor3d(0.0,0.0,1.0); // A kék végpont
     glVertex2d(200.0,200.0);
  glEnd();
  glFlush();
}
```

59

Program (szakaszrajzoló) III

Megjegyzés:

`glShadeModel(GL_SMOOTH)`

`GL_SMOOTH`: ekkor a két végpont között a hozzájuk megadott színekkel interpolál

`GL_FLAT`: utolsó végpont színével rajzol (`GL_POLYGON` esetében az elsőével)

60

Program (szakaszrajzoló) IV

```
int main(int argc, char* argv[]) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(200,200);
  glutInitWindowPosition(100,100);
  glutCreateWindow(„szakasz");
  init();
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

61

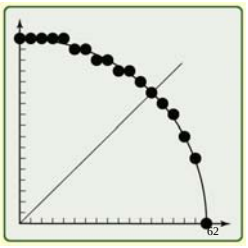
Kör rajzolása

$x^2+y^2 = R^2$ R: egész

1. Elég egy kör-negyedet/nyolcadot 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



62

Kör rajzolása

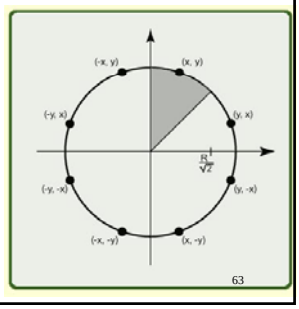
2. Polárkoordinátás alak

Most is 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)



63

Program (nyolcad kör)

Egyszerre 8 pontot helyezünk el:

```
void Circlepoints(int x, int y, value) {
   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);
} // CirclePoints
```

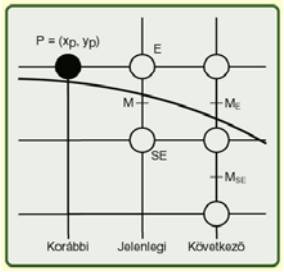
64

Kör rajzolása

3. Felezőpont algoritmus körre

x 0-tól $R/\sqrt{2}$ -ig (amíg $x \le y$)

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



65

Felezőpont algoritmus körre

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

$d = F(M) =$

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

66

Felezőpont algoritmus körre

$F(x,y) = x^2+y^2 - R^2$

d változása a következő pontnál:

ha előzőleg ***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 előzőleg ***SE***-t választottuk, akkor

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

$= 2x_p - 2y_p+5$

67

Felezőpont algoritmus körre

Az iterációs lépések:

1. a döntési változó előjele alapján kiválasztjuk a következő képpontot
2. $d = d + \Delta_{SE}$ vagy $d + \Delta_E$ (a választástól függően).

Figyeljük meg: *d* értéke egész számmal változik!

Kezdés:

kezdőpont: (*0, R*)

felezőpont: (*1, R – 1/2*)

$d = F(1, R - 1/2) = 5/4 - R$ **nem egész szám!**

68

Felezőpont algoritmus körre

Nem tudunk egész aritmetikát használni, ezért legyen *h* új döntési változó:

$h = d - ¼$ $h+¼ = d < 0$

Ekkor kezdéskor

$h = 5/4 - R - ¼ = 1 - R$

Kezdetben, és a későbbiek során is *h* egész szám!

Igaz, hogy $d < 0$ helyett $h < -¼$ -et kellene vizsgálni, de ez *h* egész volta miatt ez ekvivalens $h < 0$ -val, tehát egész aritmetika használható.

Megjegyzés: *F* helyett *4F*-fel is dolgozhatnánk.

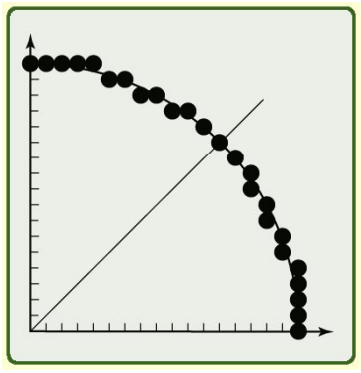
69

Felezőpont algoritmus körre

```
void MidpointCircle(int R, int value) {
   int h;
   x = 0; y = R; h = 1-R;
   CirclePoints(x,y,value);
   while(y >= x) {
      if(h < 0) {
         x++; h += 2*x+3;
      } else {
         x++; y--;
         h += 2*(x-y)+5;
      }
      CirclePoints(x,y,value);
   } // while
} // MidpointCircle
```

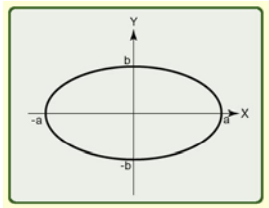
70

Felezőpont algoritmus körre



71

Ellipszis rajzolása



$x^2/a^2 + y^2/b^2 = 1$

$b^2x^2 + a^2y^2 - a^2b^2 = 0$

a, b egész

$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2$

Szimmetria miatt:

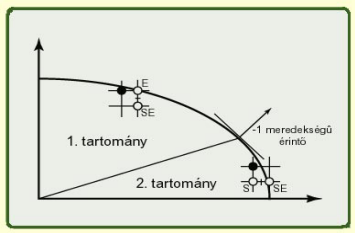
elég az első síknegyedben megrajzolni

72

Ellipszis rajzolása

Da Silva algoritmusa (felezőpont algoritmus)

Bontsuk a negyedet két tartományra:



Az 1. tartományban $a^2 (y_p - ½) > b^2 (x_p+1)$

73

Da Silva algoritmusa

$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2$

Az 1. tartományban:

$$d_1 = F(x_p+1, y_p - ½) \begin{cases} \geq 0 & \textbf{E}\text{-t választjuk} \\ < 0 & \textbf{SE}\text{-t választjuk} \end{cases}$$

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

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

74

Da Silva algoritmusa

$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2$

Az 1. tartományban *d* változása,

ha előzőleg ***E***-t választottuk:

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

$\Delta_E = b^2 (2x_p+3)$

ha előzőleg ***SE***-t választjuk

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

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

Δ_E és Δ_{SE} egész szám.

75

Da Silva algoritmusa

$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2$

Kezdés:

kezdőpont: $(0, b)$

felezőpont: $(1, b - ½)$

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

Ha *F* helyett *4F*-el dolgozunk, akkor egész aritmetikát használhatunk.

Házi feladat

Az algoritmus a 2. tartományban

76

Da Silva algoritmusa

```
void MidpointEllipse(int a, int b, int value) {
int   x, y, a2, b2; double d1,d2;
   x = 0; y = b; a2 = a*a; b2 = b*b;
   d1 = b2 - a2*b + a2/4;
   EllipsePoints(x,y,value);
   while(a2*(y-1/2) > b2*(x+1)) {
     if(d1 <0) {
      d1 += b2*(2*x+3); x++;
     } else {
      d1 += b2*(2*x+3)+ a2*(-2*y+2); x++; y--;
     }
     EllipsePoints(x,y,value);
   } // Region1
   d2 = b2*(x+1/2)*(x+1/2)+a2*(y-1)*(y-1)- a2*b2;
   while(y > 0) {
     if(d2 < 0) {
       d2 += b2*(2*x+2)+ a2*(-2*y+3); x++; y--;
     } else {
       d2 += a2*(-2*y+3); y--;
     }
     EllipsePoints(x,y,value);
   } // Region2
} // MidpointEllipse
```

77

OpenGL

Feladat:

Kör rajzolása felezőpont algoritmussal

78

GRAFIKUS PRIMITÍVEK KITÖLTÉSE

Téglalap kitöltése
Poligon kitöltése
Kör, ellipszis kitöltése
Kitöltés mintával

79

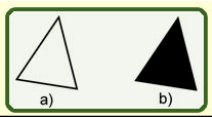
GRAFIKUS PRIMITÍVEK KITÖLTÉSE

Területi primitívek:

Zárt görbék által határolt területek (pl. kör, ellipszis, poligon)

Megjeleníthetők

a) Csak a határvonalat reprezentáló pontok kirajzolásával (kitöltetlen)
b) Minden belső pont kirajzolásával (kitöltött)



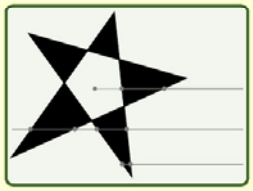
80

GRAFIKUS PRIMITÍVEK KITÖLTÉSE

Alapkérdés:

Mely képpontok tartoznak a grafikus primitívekhez?

Páratlan paritás szabálya:



Páros számú metszéspont: külső pont

Páratlan számú metszéspont: belső pont

81

GRAFIKUS PRIMITÍVEK KITÖLTÉSE

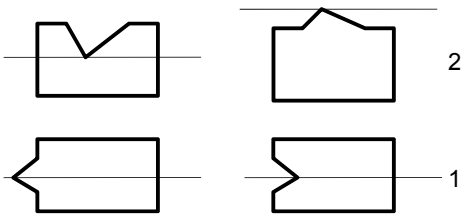
Primitívek kitöltésének az elve:

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

82

GRAFIKUS PRIMITÍVEK KITÖLTÉSE

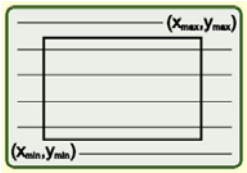
Csúcspontok metszésekor:



Ha a metszett csúcspont lokális minimum vagy maximum, akkor kétszer számítjuk, különben csak egyszer.

83

Téglalap kitöltése




```
for(y = ymin; y < ymax; y++)
   for(x = xmin ; x < xmax; x++)
      WritePixel(x,y,value);
```

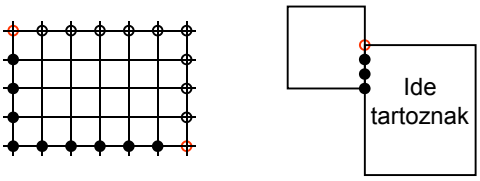
Probléma:

Egész koordinátájú határpontok hova tartozzanak?

84

Téglalap kitöltése

Legyen a szabály pl.: Egy képpont akkor nem tartozik a primitívhez, ha 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

85

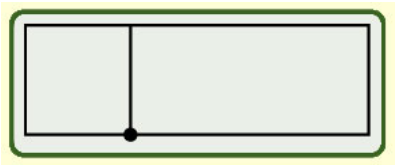
Téglalap kitöltése

Megjegyzések:

a) Általánosítható poligonokra

b) A felső sor és jobb szélső oszlop hiányozhat

c) A bal alsó sarok kétszeresen tartozhat a téglalaphoz



86

Poligon kitöltése

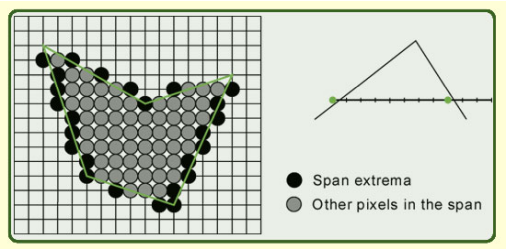
A poligon lehet:
- konvex,
- konkáv,
- önmagát metsző,
- lyukas

Haladjunk a pásztázó egyeneseken és keressük a kitöltési szakaszok végpontjait:

87

Poligon kitöltése

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

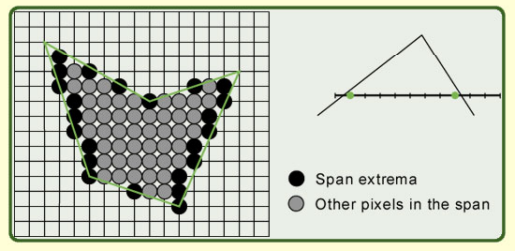


Diszjunkt poligonoknak lehet közös képpontjuk

88

Poligon kitöltése

b) A végpontokat a poligonhoz tartozó képpontok közül választjuk



89

Algoritmus poligonok kitöltésére

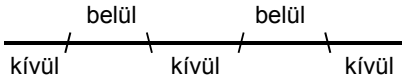
Minden pásztázó egyenesre:

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

90

Algoritmus poligonok kitöltésére

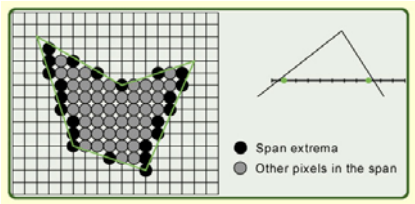
3. A poligon belsejébe tartozó szakasz(ok) végpontjai közötti képpontok kirajzolása Használjuk a páratlan paritás szabályát: Tegyük fel, hogy a bal szélen kívül vagyunk, utána minden egyes metszéspont megváltoztatja a paritást



91

Algoritmus poligonok kitöltésére

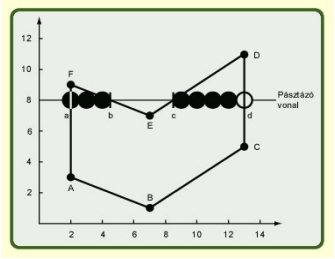
3.1 Adott ***x*** nem egész értékű metszéspont.
Ha kívül vagyunk, akkor legyen a végpont a fölfelé kerekített ***x***
Ha belül vagyunk, akkor legyen a végpont a lefelé kerekített ***x***



92

Algoritmus poligonok kitöltésére

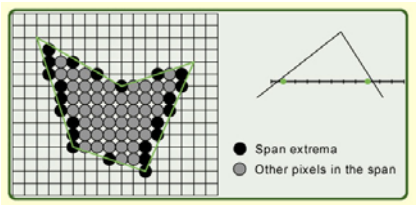
3.2 Adott ***x*** egész értékű metszéspont
Ha ez bal végpont, akkor ez belső pont
Ha ez jobb végpont, akkor ez külső pont



93

Algoritmus poligonok kitöltésére

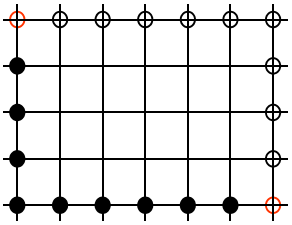
3.2.1 A poligon csúcspontjaiban:
y_{min} csúcspont beszámít a paritásba
y_{max} csúcspont nem számít a paritásba, tehát y_{max} csúcspont csak akkor lesz kirajzolva, ha az a szomszédos él y_{min} pontja is



94

Algoritmus poligonok kitöltésére

3.2.2 Vízszintes él esetén:
Az ilyen élek csúcspontjai nem számítanak a paritásba
Egész y koordináta esetén az alsó élet rajzoljunk, a felsőt nem



95

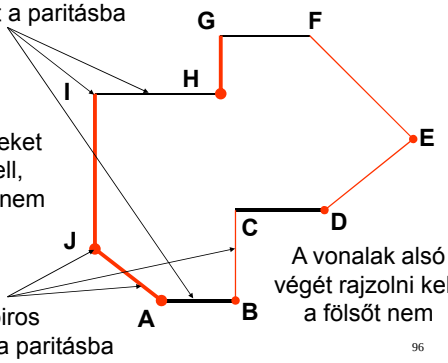
Példa poligon kitöltésére

A fekete nem számít a paritásba

A vastag éleket rajzolni kell, a vékonyat nem

A piros beszámít a paritásba

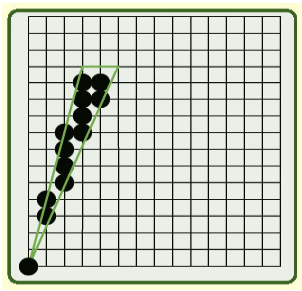
A vonalak alsó végét rajzolni kell, a fölsőt nem



96

Poligon kitöltése

Szilánkok: olyan poligon-területek, amelyek belsejében nincs kitöltendő szakasz = hiányzó képpontok



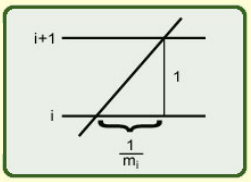
97

Poligon kitöltése

Implementáció:

Nem kell minden egyes pásztázó vonalra újra kiszámolni minden metszéspontot, mert általában csak néhány metszéspont érdekes az *i*-dik pásztázó vonalról az *i+1*-dikre átlépve

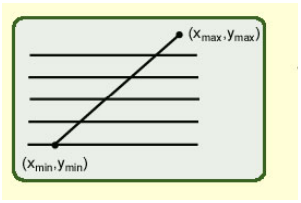
$$x_{i+1} = x_i + \frac{1}{m_i}$$



98

Poligon kitöltése

Tegyük fel hogy: *m>1*
(*m = 1* triviális, *m < 1* kicsit bonyolultabb)



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

x = egész rész + tört rész

$[x]$ $\{x\}$

$[x_{i+1}] = [x_i]$ vagy $[x_i] + 1$

$\{x_{i+1}\} = \{x_i\} + \Delta x$ vagy $\{x_i\} + \Delta x - 1$

99

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 számlálót és nevezőt kell tárolni

100

Poligon kitöltése

```
void LeftEdgeScan(int xmin, int ymin,
            int xmax, int ymax, int value) {
  int x, y, numerator, denominator, increment;
  x = xmin; numerator = xmax - xmin;
  denimonator = ymax-ymin;
  increment = denominator;
  for(y = ymin; y < ymax; y++) {
    WritePixel(x,y,value);
    increment += numerator;
    if(increment > denominator) {
     x++; increment -= denominator;
    }
  }
}
```

101

Poligon kitöltése

Adatstruktúrák:

ÉT: (Élek Táblázata)

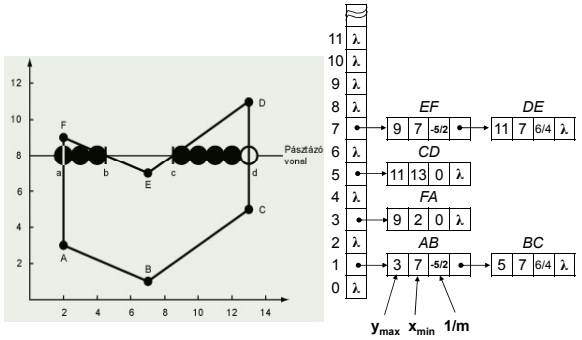
A kisebbik ***y*** értékük szerint rendezve az összes élet tartalmazza. **A vízszintes élek kimaradnak**!

Annyi lista van, ahány pásztázó vonal. Minden listában azok az élek szerepelnek, amelyek alsó végpontja a pásztázó vonalon van. A listák az élek alsó végpontjának ***x*** koordinátája, ezen belül a meredekség reciproka szerint rendezettek

Minden lista elem tartalmazza az él y_{max}, x_{min} koordinátáját és a meredekség reciprokát.

102

Poligon kitöltése



103

Poligon kitöltése

AÉT: (Aktív Élek Táblázata)

A pásztázó vonalat metsző éleket tartalmazza a metszéspontok ***x*** koordinátája szerint rendezve. Ezek a metszéspontok kitöltési szakaszokat határoznak meg az aktuális pásztázó vonalon.

Ez is lista.

104

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:

105

Algoritmus poligon kitöltésére

3.1 **ÉT**-ből az ***y*** -hoz tartozó listát – a rendezést megtartva – **AÉT**-hez másoljuk

3.2 **AÉT**-ből kivesszük azokat az éleket, amelyekre $y_{max} = y$ (a fölső éleket nem töltjük ki)

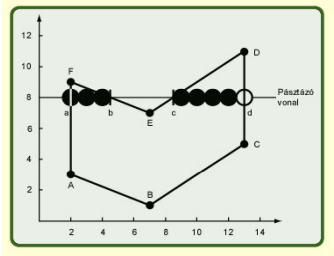
3.3 A kitöltési szakaszok pontjait megjelenítjük

3.4 ***y = y+1***

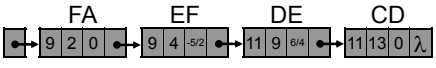
3.5 Minden **AÉT**-beli élben módosítjuk ***x***-et

106

Poligon kitöltése



AÉT az ***y = 8*** pásztázó vonalon:



107

Poligon kitöltése

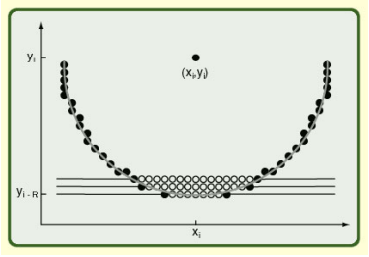
Megjegyzés:

Háromszögekre, trapézokra egyszerűsíthető az algoritmus, mert a pásztázó egyeneseknek legfeljebb *2* metszéspontja lehet egy háromszöggel vagy egy trapézzal (nem kell **ÉT**).

108

Kör, ellipszis kitöltése

P belül van, ha ***F(P) < 0***, de most is használható a felezőpont módszer. Hasonló algoritmussal számíthatók a kitöltési szakaszok.



109

Háromszög kitöltése (OpenGL)

Egyetlen színnel

```
glBegin(GL_TRIANGLES);
     glColor3f(0.1, 0.2, 0.3);
     glVertex3f(0, 0, 0);
     glVertex3f(1, 0, 0);
     glVertex3f(0, 1, 0);
 glEnd();
```

110

Háromszög kitöltése (OpenGL)

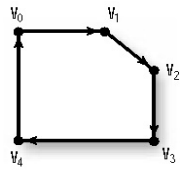
Több színnel (Gouraud-féle módon interpolálva)

```
glShadeModel(GL_SMOOTH); //G-árnyalás
        glBegin(GL_TRIANGLES);
           glColor3d(1.0,0.0,0.0);
           glVertex3d(5.0,5.0,0.0);
           glColor3d(0.0,0.0,1.0);
           glVertex3d(195.0,5.0,0.0);
           glColor3d(0.0,1.0,0.0);
           glVertex3d(100.0,195.0,0.0);
        glEnd();
```

111

Poligon létrehozása (OpenGL)

```
glBegin(GL_POLYGON);
        glVertex3d(0,100,0);
        glVertex3d(50,100,0);
        glVertex3d(100,50,0);
       glVertex3d(100,0,0);
       glVertex3d(0,0,0)
glEnd();
```



Az **OpenGL** csak síkbeli konvex sokszögek helyes kirajzolását garantálja

Az elsőként specifikált csúcspont színe lesz a primitív színe, ha

```
glShadeModel(GL_FLAT);
```

112

Poligon (OpenGL)

3D-s poligonoknak két oldaluk van: elülső és hátulsó oldal. Alapértelmezésben mindkét oldal ugyanúgy rajzolódik ki, de ezen lehet változtatni:

```
void glPolygonMode(enum face, enum mode);
```

`face:`

- `GL_FRONT_AND_BACK`
- `GL_FRONT`
- `GL_BACK;`

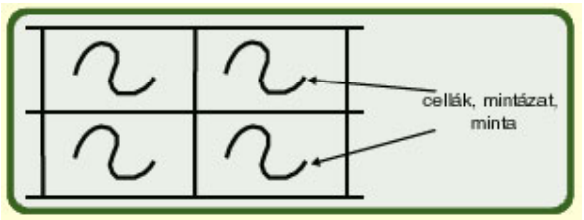
`mode:`

- `GL_POINT` csak a csúcspontokat rajzolja ki
- `GL_LINE` a határvonalat rajzolja ki
- `GL_FILL` kitölti a poligont

113

Kitöltés mintával

Általában: terület kitöltése szabályosan ismétlődő grafikus elemekkel

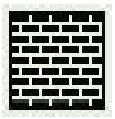


Képmátrixok (raszter) esetében a cella egy (kisméretű) mátrix

114

Kitöltés mintával

Példa:



Tégla minta

Lehet a kitöltés "átlátszó" is: nem minden képpontot írunk felül, csak azokat, ahol a minta nem *0*

115

Kitöltés mintával

Fajtái:

1. Válasszunk egy pontot a **primitív**ben (pl. bal felsőt), egy pontot a mintában (pl. bal felsőt), illesszük azokat egymásra, a többi pont illeszkedése már kiszámítható
2. Válasszunk egy pontot a **képernyő**n (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)

116

Kitöltés mintával

Legyen:

minta *M* * *N* -es mátrix

minta [*0,0*] → képernyő [*0,0*]

ekkor

1. módszer: Pásztázás soronként (átlátszó)

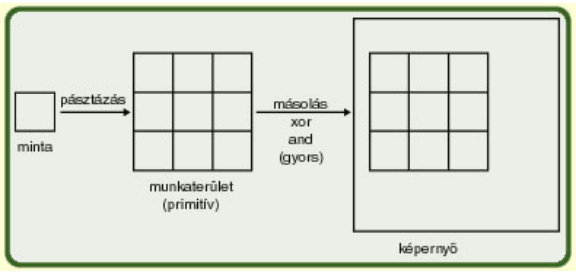
```
if(minta[x % M][y % N])
   WritePixel(x,y,érték);
```

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

117

Kitöltés mintával

2. módszer: Téglalap írás

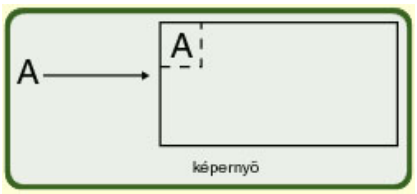


118

Kitöltés mintával

Csak akkor érdemes használni, ha a primitívet sokszor kell használni

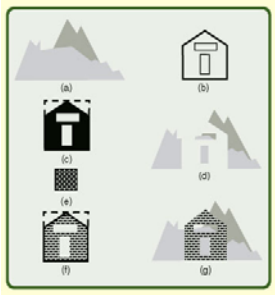
Pl. karakterek megjelenítése



119

Kitöltés mintával

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

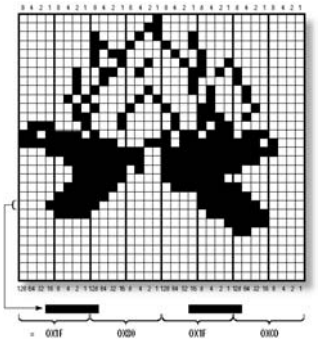


(a) hegyek, (b) ház vonalai, (c) a ház kitöltött bitmap képe, (d) (a)-ból kitöröltük (c)-t, (e) tégla minta, (f) (b) tégla mintával kitöltve, (g) (e) (d)-re másolva

120

Kitöltés mintával (OpenGL)

```
void glPolygonStipple(const ubyte *mask);
```



Kitöltési minta beállítása
`mask:` egy 32×32-es bittérkép (minta)

121

Kitöltés mintával (OpenGL)

A kitöltési minta
`glEnable(GL_POLYGON_STIPPLE)` engedélyezése
`glDisable(GL_POLYGON_STIPPLE)` tiltása

Pl.:

```
void display() {
  glClear(GL_COLOR_BUFFER_BIT);
        //Képernyő törlés
  glShadeModel(GL_FLAT);
        //Árnyalási mód: FLAT
  glPolygonStipple(mask); // A minta
  glEnable(GL_POLYGON_STIPPLE);//engedélyezés
  rajz();             //Az alakzat kirajzolása
  glDisable(GL_POLYGON_STIPPLE); //tiltás
}
```

122

Kitöltés mintával (OpenGL)

Példa:

123

VASTAG PRIMITÍVEK RAJZOLÁSA

Képpontok ismétlése
Mozgó ecset
Területkitöltés
Közelítés vastag szakaszokkal

124

VASTAG PRIMITÍVEK RAJZOLÁSA

Több képpontnyi vastagságú vonalak

Milyen alakú legyen az ecset?
- Kör?
- Téglalap?
- Forduljon a vonallal?

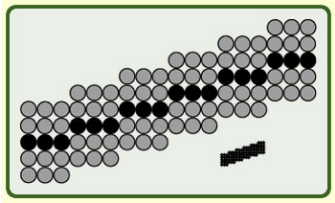
125

VASTAG PRIMITÍVEK RAJZOLÁSA

1. Képpontok ismétlése

A pásztázó vonalas algoritmus kiterjesztése:

ha $-1 < m < 1$, akkor a képpontokat többszörözzük meg az oszlopokban;
különben a sorokban



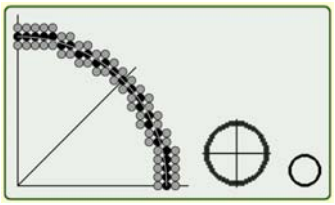
126

Képpontok ismétlése

Tulajdonságai:

a) gyors,
b) a vonal végek mindig vízszintesek vagy függőlegesek,
c) a vonal vastagsága függ a meredekségtől

d) a duplázás nem megy: a vonal valamelyik oldala felé vastagabb



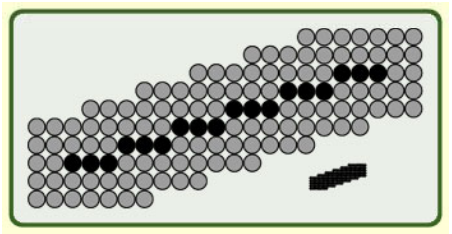
Jó módszer, ha nem túl vastag a vonal

127

VASTAG PRIMITÍVEK RAJZOLÁSA

2. Mozgó ecset

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



128

Mozgó ecset

Tulajdonságai:

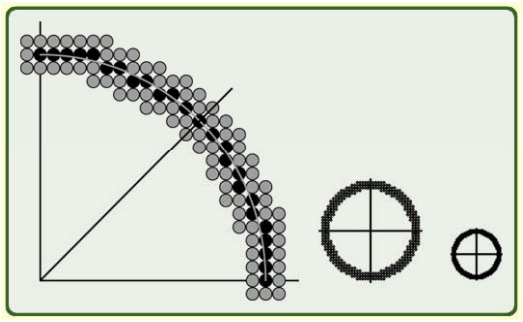
hasonló 1-hez, de

a) a végpontok „nagyobbak”
b) 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

129

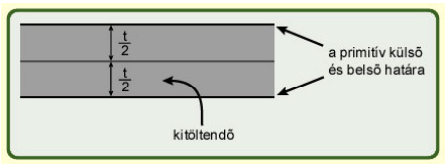
Mozgó ecset



130

VASTAG PRIMITÍVEK RAJZOLÁSA

3. Területkitöltés



Terület primitíveknél a külső határvonalhoz használhatjuk az eredeti határvonalat, elegendő tehát a belsőt meghatározni

131

Területkitöltés

Tulajdonságai:

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

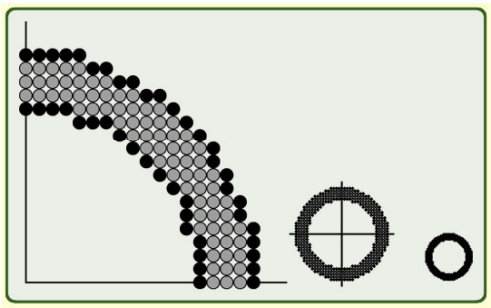
Kör esetén: külső és belső kör

Ellipszis esetén:

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

132

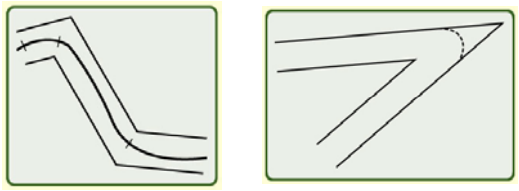
Területkitöltés



133

VASTAG PRIMITÍVEK RAJZOLÁSA

4. Közelítés vastag szakaszokkal



Szakaszonként lineáris approximáció

a) szép

b) vastag vonalakat símán kell illeszteni

134

Pont mérete (OpenGL)

```
Void glPointSize(GLfloat size);
```

Nem minden méretet támogatnak az implementációk:

```
GLfloat sizes[2]; // méret tartomány
GLfloat step; // támogatott lépés

glGetFloatv(GL_POINT_SIZE_RANGE, sizes);
glGetFloatv(GL_POINT_SIZE_GRANULARITY,
  &step);
```

135

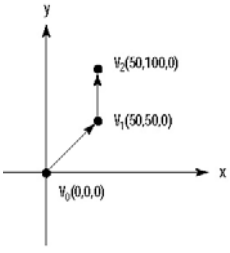
Szakaszok rajzolása (OpenGL)

Független szakasz (`GL_LINES`):
Az elsőként specifikált két csúcspont határozza meg az első szakaszt, a második két csúcspont a második szakaszt, ... (nincsenek összekötve)

136

Szakaszok rajzolása (OpenGL)

Szakasz sorozat (`GL_LINE_STRIP`):
Egy vagy több összekötött szakasz specifikálása a végpontok sorozatának megadásával. Pl.:

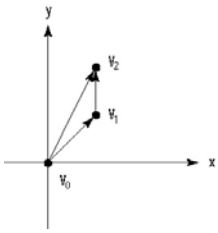



```
glBegin(GL_LINE_STRIP);
    glVertex3d(0,0,0);
    glVertex3d(50,50,0);
    glVertex3d(50,100,0);
glEnd();
```

137

Szakaszok rajzolása (OPenGL)

Szakasz hurok (`GL_LINE_LOOP`):
Ugyanaz, mint a `LINE_STRIP`, de az utolsóként specifikált csúcspontot összekötjük az elsőként specifikált csúcsponttal. Pl.:

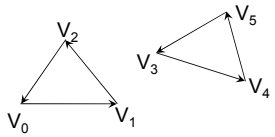



```
glBegin(GL_LINE_LOOP);
    glVertex3d(0,0,0);
    glVertex3d(50,50,0);
    glVertex3d(50,100,0);
glEnd();
```

138

Háromszögek rajzolása (OpenGL)

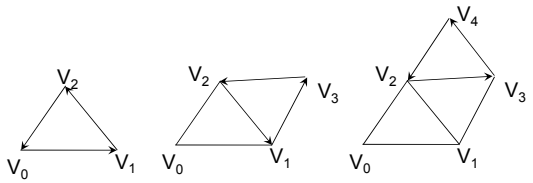
Független háromszögek(`GL_TRIANGLES`):
Az elsőként specifikált három csúcspont határozza meg az első háromszöget, a második három csúcspont a második háromszöget, ...



139

Háromszögek rajzolása (OpenGL)

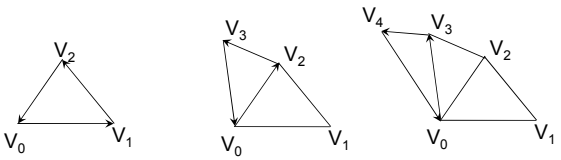
Háromszög sorozat (`GL_TRIANGLE_STRIP`):
Egy vagy több szomszédos háromszög specifikálása a csúcspontok sorozatának megadásával. Pl.:



140

Háromszögek rajzolása (OpenGL)

Háromszög legyező (`GL_TRIANGLE_FAN`):
Egy vagy több szomszédos háromszög specifikálása a csúcspontok sorozatának megadásával. Pl.:



141

Vonal vastagsága (OpenGL)

```
Void glLineWidth(GLfloat width);
```

Nem minden vastagságot támogatnak az implementációk:

```
GLfloat sizes[2]; // vastagság tartomány
GLfloat step; // támogatott lépés

glGetFloatv(GL_LINE_WIDTH_RANGE, sizes);
glGetFloatv(GL_LINE_WIDTH_GRANULARITY,&step);
```

142

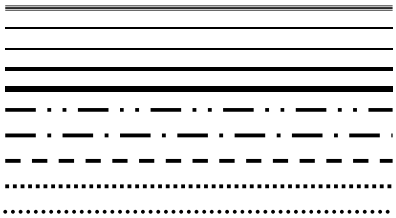
Szakaszok élsimítása (OpenGL)

A szakaszok élsimítását engedélyezni a **`GL_LINE_SMOOTH`** argumentummal meghívott **`glEnable`**, tiltani a **`glDisable`** függvénnyel lehet.

Ha az élsímítás engedélyezett, akkor nem egész szélességek is megadhatók, és ekkor a szakasz szélén kirajzolt képpontok intenzitása kisebb, mint a szakasz közepén lévő képpontoké.

143

VONAL STÍLUS



144

VONAL STÍLUS

Primitívek attribútumai:

- vonal vastagság
- szín
- vonal stílus
- stb...

Vonal stílus:

- folytonos
- szaggatott
- pontozott
- felhasználó által definiált

145

VONAL STÍLUS

Stílus:

8 vagy *16* bites maszk írja le, hogy mely biteknek megfelelő pontok legyenek kirajzolva, mint a vonal pontjai

Pl:11111111 = folytonos

11101110 = szaggatott

Rajzolás: (maszkolással)

146

VONAL STÍLUS

Hátránya:

A szaggatások távolsága függ a meredekségtől

Megoldás:

A távolságot számolva rajzolni a szakaszokat

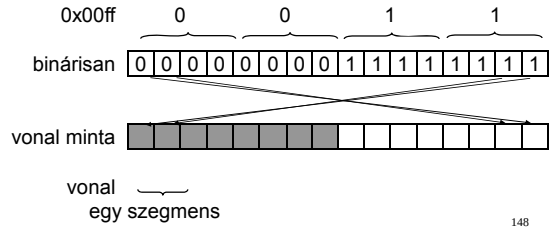
147

Szakasz stílus (OpenGl)

```
void glLineStipple(int factor,
                   ushort pattern);
```

`Pattern` (maszk): 16 bites bináris jelsorozat

`factor`: A pattern-ben levő minden bit `factor`-szor kerül alkalmazásra.



148

Szakasz stílus (OpenGl)

Pl.:

```
glLineStipple(1, 0x3F07);
glEnable(GL_LINE_STIPPLE);
```

A minta: 0011111100000111
(az alacsony helyértékű bittel kezdünk).

```
glLineStipple(2, 0x3F07);
glEnable(GL_LINE_STIPPLE);
```

A minta: 00001111111111110000000000111111

Szakasz stílus

`glEnable(GL_LINE_STIPPLE)` engedélyezése

`glDisable(GL_LINE_STIPPLE)` tiltása

149

Szakasz stílus (OpenGl)

Megoldandó feladat:

Rajzoljunk ötszöget olyan egyenes szakaszokból, amelyek a következő mintákból épülnek fel:

150

VÁGÁS

A vágásról általában

Pontok vágása

Vonalak, szakaszok vágása egyenletrendszer megoldásával

A COHEN - SUTHERLAND -féle vonal vágás

Parametrikus vonal vágó algoritmus

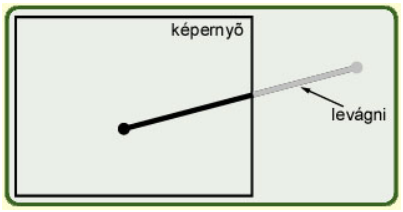
Körök és ellipszisek vágása

Poligonok vágása

151

VÁGÁS

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



152

VÁGÁS

Módszerek:

1. Vágjuk le a megjelenítés előtt, azaz számítsuk ki a metszéspontokat és az új végpontokkal rajzoljunk
2. 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

153

VÁGÁS

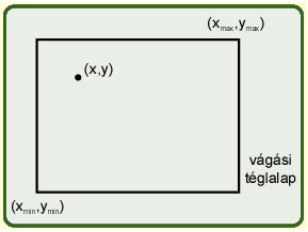
Pontok vágása:

(x,y) belül van, ha

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

és

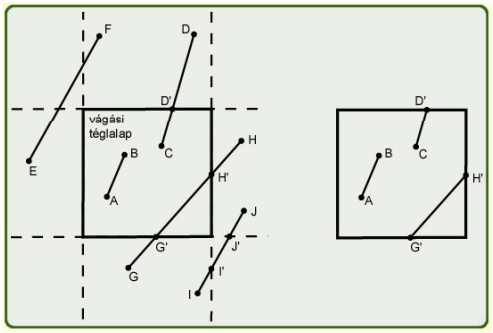
$y_{min} \leq y \leq y_{max}$



154

VÁGÁS

Szakaszok vágása egyenletrendszer megoldásával



155

Szakaszok vágása egyenletrendszer megoldásával

Elég a végpontokat vizsgálni:

a) Ha mindkét végpont belül van, akkor a teljes vonal belül van, nincs vágás;

b) Ha pontosan egy végpont van belül, akkor metszéspontot kell számolni és vágni;

c) Ha mindkét végpont kívül van, akkor további vizsgálat szükséges: lehet, hogy nincs közös része a vágási téglalappal, de lehet, hogy van.

156

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)

157

Szakaszok vágása egyenletrendszer megoldásával

Javítás: parametrikus alak

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

$t \in [0,1]$ (szakaszt ír le)

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

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

158

COHEN - SUTHERLAND - féle szakasz vágás

A végpontok kódolása:

$y>y_{max}$	$y<y_{min}$	$x>x_{max}$	$x<x_{min}$
y_{max}-y előjele	$y-y_{min}$ előjele	x_{max}-x előjele	$x-x_{min}$ előjele

	x_{min}		x_{max}
	1001	1000	1010
y_{max}	0001	0000	0010
y_{min}	0101	0100	0110

159

COHEN - SUTHERLAND - féle szakasz vágás

A végpontok kódolása: Minden végpont annak megfelelő kódot ($code_1$, $code_2$) kap, hogy melyik tartományban van.

(x_1, y_1) és (x_2, y_2) a szakasz két végpontja.

	x_{min}		x_{max}
	1001	1000	1010
y_{max}	0001	0000	0010
y_{min}	0101	0100	0110

160

COHEN - SUTHERLAND - féle szakasz vágás

Előzetes vizsgálatok:

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

$code_1 = code_2 = 0000$

	x_{min}		x_{max}
	1001	1000	1010
y_{max}	0001	0000	0010
y_{min}	0101	0100	0110

161

COHEN - SUTHERLAND - féle szakasz vágás

különben, ha (bitenként) $code_1$ AND $code_2$ = TRUE

2. ha $x_1,x_2 < x_{min}$ (...1)
   vagy $x_1,x_2 > x_{max}$ (..1.)
   vagy $y_1,y_2 < y_{min}$ (.1..)
   vagy $y_1,y_2 > y_{max}$ (1...)

   } minden kívül van (triviális elvetés)

	x_{min}		x_{max}
	1001	1000	1010
y_{max}	0001	0000	0010
y_{min}	0101	0100	0110

162

COHEN - SUTHERLAND - féle szakasz vágás

különben:

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), számítsuk ki a metszéspontot.
   A két részre vágott szakasz egyik fele a 2. pont alapján triviálisan elvethető.

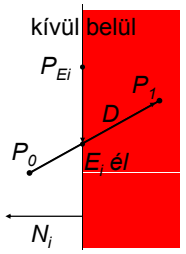
COHEN - SUTHERLAND - féle szakasz vágás

Interaktív módon is használható

Hatékony, mert gyakori, hogy 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):

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

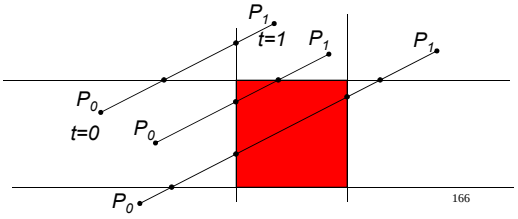
$$N_i\,(P_0 + D\,t - P_{E\,i}) = 0$$

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

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 élével való 4 metszéspontja (4 db t érték).
Melyik két t a megfelelő?



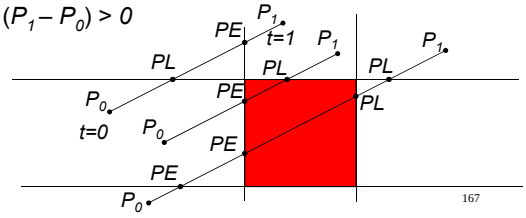
Parametrikus szakasz vágó algoritmus

PE olyan pont, ahol P_0-ból P_1-felé haladva belépünk egy belső félsíkba (potential entering), ekkor

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

PL olyan, ahol kilépünk egy belső félsíkból (potential leaving), ekkor

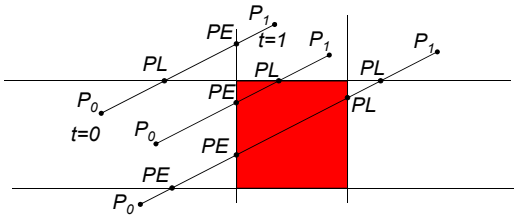
$$N_i\,(P_1 - P_0) > 0$$



Parametrikus szakasz vágó algoritmus

Legyen $t_E = \max\{0, \max\{t_{PE}\}\}$,

$t_L = \min\{1, \min\{t_{PL}\}\}$



- Ha $t_E > t_L$, akkor nincs belső metszés
- különben t_E , $t_L \in [0,1]$, és ez a belső szakasz

Parametrikus szakasz vágó algoritmus számítása

A metszéspontok számítása:

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})	$(x_0 - x, y_0 - y_{max})$	$\frac{(y_0 - y_{max})}{-(y_1 - y_0)}$

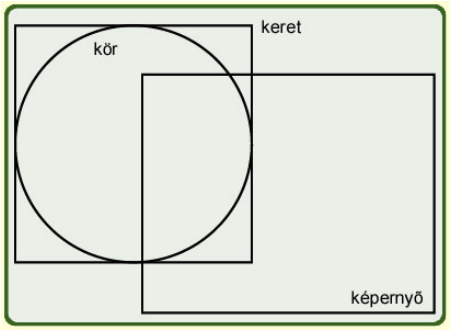
169

Parametrikus szakasz vágó algoritmus

```
begin
  Ni kiszámítása, PEi kiválasztása minden élre;
  for szakaszokra
  if P1 = P0 then pont vágása;
  else begin
    tE = 0; tL = 1; D = P1-P0;
    for él és szakasz párokra
      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 belső metszés
    else P(tE)-től P(tL)-ig belső metszés
  end
end
```

170

Körök és ellipszisek vágása



171

Körök és ellipszisek vágása

Triviális vizsgálat:

Ha a keret belül van, akkor a kör is belül van, nincs mit vágni;
Ha a keret kívül van, akkor a kör is kívül van, nincs mit vágni.

Különben:

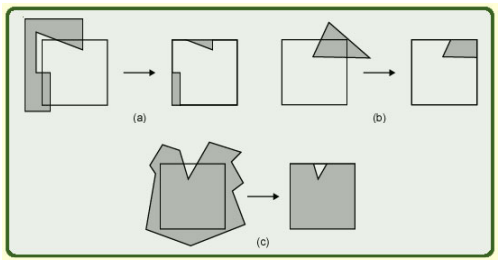
Környegyedekre (nyolcadokra) kiszámítjuk a kör és a téglalap élének metszéspontját, utána pásztázás
Ha a kör nem nagy, akkor pixelenként dönthetünk.

Ellipszis: hasonlóan.

172

Poligonok vágása

Sok eset lehet:



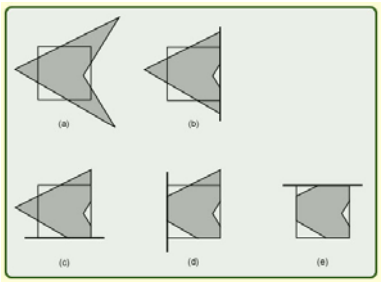
Általában minden éllel vágni kell

173

Poligonok vágása

SUTHERLAND, HODGMAN:

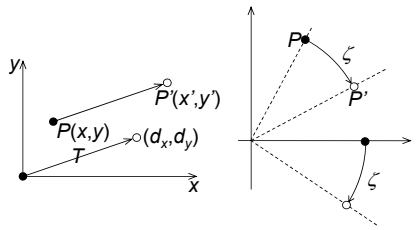
vágjunk egyenként az összes éllel



$(v_1, v_2, ..., v_n)$ csúcspontok → algoritmus → $(v'_1, v'_2, ..., v'_m)$ új csúcspontok

174

Geometriai transzformációk



175

Bevezetés - Transzformációk

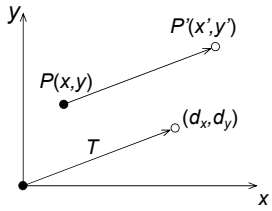
A Számítógépes Grafikában használatos 2- és 3-dimenziós transzformációk:

- eltolás
- nagyítás, kicsinyítés (skálázás)
- forgatás

176

Pont 2D eltolása

Hosszak és a szögek változatlanok



$x'=x+d_x$
$y'=y+d_y$
(oszlop-)vektorokkal:

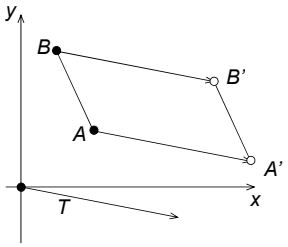
$$P=\begin{pmatrix}x\\y\end{pmatrix} \quad P'=\begin{pmatrix}x'\\y'\end{pmatrix} \quad T=\begin{pmatrix}dx\\dy\end{pmatrix}$$

$P'=P+T$

177

Szakasz 2D eltolása

Elegendő az új végpontokat számolni



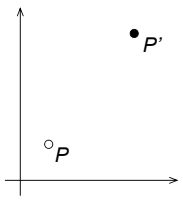
$A'=A+T$
$B'=B+T$

178

2D nagyítás/kicsinyítés

A szögek változatlanok
Szokták a kettőt együtt **SKÁLÁZÁSKÉNT** említeni

Origóból történő nagyítás



Általános skálázás

$x'=x\cdot s_x$
$y'=y\cdot s_y$
(oszlop-)vektorokkal:

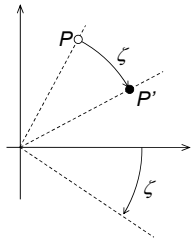
$$P=\begin{pmatrix}x\\y\end{pmatrix} \quad P'=\begin{pmatrix}x'\\y'\end{pmatrix} \quad S=\begin{pmatrix}S_x & 0\\0 & S_y\end{pmatrix}$$

$P'=S\cdot P$

179

2D forgatás

A hosszak és a szögek változatlanok
Origó körüli forgatás



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

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

$P' = R \cdot P$

180

Homogén koordináták

(x, y) jelölése homogén koordinátákkal: (x, y, w)

Egyenlőség:

(x, y, w) = (x', y', w') , ha van olyan α hogy:

$x' = \alpha \cdot x$, $y' = \alpha \cdot y$, $w' = \alpha \cdot w$

pl: (2, 3, 6) = (4, 6, 12)

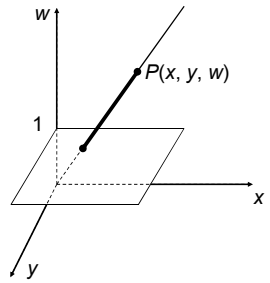
Egy ponthoz végtelen sok (x, y, w) tartozik.

Ha $w = 0$, akkor (x, y, w) végtelen távoli pont

(0, 0, 0) nem megengedett!

181

Kapcsolat 2D és 3D közt



($t \cdot x$, $t \cdot y$, $t \cdot w$)
egyenes a 3D térben

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

P vetülete a $w = 1$ síkon

A végtelen távoli pontok nincsenek a síkon

182

2D eltolás - matematikailag

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

ahol $T(d_x, d_y) = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}$ $\quad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

Ismételt eltolások (kompozíció):

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

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

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

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

183

2D skálázás - matematikailag

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

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

Ismételt skálázások (kompozíció):

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

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

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

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

184

2D forgatás - matematikailag

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

$P' = R(\zeta)\, P$ ($R(\zeta)$ ortogonális)

Ismételt forgatások:

$P' = R(\zeta_1)\, P$,

$P'' = R(\zeta_2)\, P' = R(\zeta_2)\, (R(\zeta_1)\, P) = R(\zeta_1 + \zeta_2)\, P$

Bizonyítás: Házi feladat

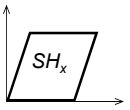
185

2D nyírás

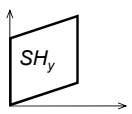
A hosszak és a szögek változhatnak

Párhuzamos egyenesek képe párhuzamos

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



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



186

Affin transzformációk

Affin transzformáció: eltolások, skálázások, forgatások és nyírások tetszőleges számú és sorrendű egymás utáni alkalmazásával kapott transzformáció

187

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 $R(\zeta)$

c) eltolás O-ból P-be $T(x,y)$

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

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

188

2D transzformációk kompozíciója 2. példa

Nagyítás egy tetszőleges $P(x, y)$ pontból:

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

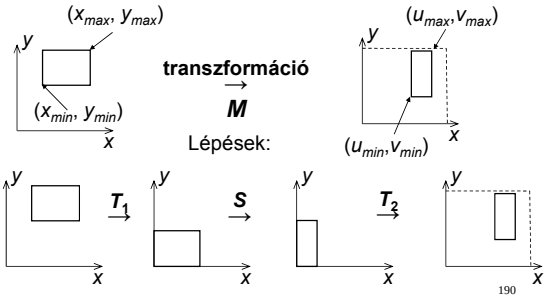
$$\begin{pmatrix}1&0&x\\0&1&y\\0&0&1\end{pmatrix}\begin{pmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&-x\\0&1&-y\\0&0&1\end{pmatrix}=$$

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

189

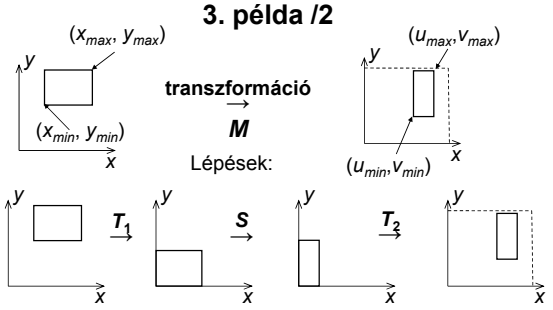
2D transzformációk kompozíciója 3. példa /1

„Világ koordináta rendszer” „Képernyő koordináta rendszer ”



190

2D transzformációk kompozíciója 3. példa /2



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

191

2D transzformációk kompozíciója 3. példa /3

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

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

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

192

2D transzformációk kompozíciója 3. példa /4

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

tehát

$$P'=MP(x,y,1)=\begin{pmatrix} (x-x_{min})\frac{u_{max}-u_{min}}{x_{max}-x_{min}}+u_{min} \\ (y-y_{min})\frac{v_{max}-v_{min}}{y_{max}-y_{min}}+v_{min} \\ 1 \end{pmatrix}$$

193

Általános kompozíció mátrix

Skálázások, forgatások, nyírások és eltolások kompozíciója a legáltalánosabb esetben is

$$M=\begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

alakú mátrixot eredményez.

194

Gyorsítások

$$M=\begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

***M*·*P* számításakor:**

9 szorzás és 6 összeadás helyett elegendő

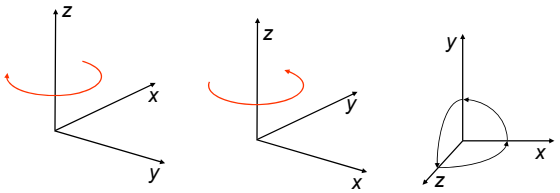
$x' = x\cdot t_{11} + y\cdot t_{12} + t_{13}$

$y' = x\cdot t_{21} + y\cdot t_{22} + t_{23}$

kiszámítása, ami csak 4 szorzás és 4 összeadás

195

3D koordináta-rendszerek



balkezes
bal-sodrású

jobbkezes
jobb-sodrású

196

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** 3D térrel való metszete a homogén koordináta

197

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

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

198

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

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

199

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

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

200

3D nyírás

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

mert

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

201

3D kompozíció-mártix

A legáltalánosabb kompozíció alakja:

$$M=\begin{pmatrix} t_{11} & t_{12} & t_{13} & t_{14}\\ t_{21} & t_{22} & t_{23} & t_{24}\\ t_{31} & t_{32} & t_{33} & t_{34}\\ 0 & 0 & 0 & 1\end{pmatrix}$$

A mátrix szorzáshoz képest most is meg lehet takarítani műveleteket.

202

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?

203

3D - síkok transzformációi

Legyen ***P*** tetszőleges pont a síkban! Ekkor $N^T P = 0$.

Melyik az a ***Q*** mátrix, amelyre $(Q\,N)^T (M\,P) = 0$?

Ha M^{-1} létezik, akkor

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

$$\Downarrow$$

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

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

(NEM BIZTOS, hogy M^{-1} létezik! Pl.: vetítés esetén)

204

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

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

205

3D koordináta-rendszerek váltása /2

Továbbá

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

pl:

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

b) Ha R: jobb-, L: bal-kezes koordináta-rendszer azonos origóval és párhuzamos tengelyekkel, akkor

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

206

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

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

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

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

$$\Downarrow$$

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

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

207

Mátrix műveletek (OpenGL)

OpenGL-ben oszlopfolytonosan tároljuk a mátrixokat.

Az egység mátrix:

```
GLfloat M[] = {
   1.0, 0.0, 0.0, 0.0,
   0.0, 1.0, 0.0, 0.0,
   0.0, 0.0, 1.0, 0.0,
   0.0, 0.0, 0.0, 1.0}
```

$$\begin{pmatrix} a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \\ a_4 & a_8 & a_{12} & a_{16} \end{pmatrix}$$

Új aktuális mátrix betöltése:

```
void glLoadMatrix{fd}(T M[16]);

glMatrixMode(GL_MODELVIEW);  // típus
glLoadMatrix(M);             // betöltés
```

208

Mátrix műveletek (OpenGL)

Az aktuális mátrix legyen az egységmátrix:

```
void glLoadIdentity(void);
```

Az aktuális mátrix szorzása:

```
void glMultMatrix{fd}(T M[16]);
```

Pl.:

```
GLfloat M[] = {
          1.0, 0.0, 0.0, 10.0,
          0.0, 1.0, 0.0, 0.0,
          0.0, 0.0, 1.0, 0.0,
          0.0, 0.0, 0.0, 1.0}
glMatrixMode(GL_MODELVIEW);
glMultMatrix(M);
```

A szorzat lesz az új aktuális mátrix

209

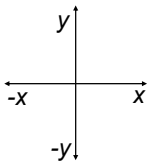
Koordináta transzformációk (OpenGL)

- **Nézeti (Viewing)**
  a néző (kamera) helyének a megadása
- **Modell (Modeling)**
  az objektumok (modell) mozgatása
- **Modell-nézet (ModelView)**
  a nézeti és a modell transzformációk együtt
- **Vetítési (Projection)**
  a nézet vágása és látótérbe méretezése
- **Ablak**
  az eredmény ablakra való leképezése

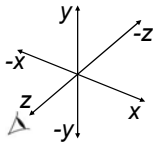
210

Nézeti koordináták (OpenGL)

- A megfigyelő nézőpontja kezdetben (0, 0, 0)
- A megfigyelő a z tengely negatív irányába néz. Virtuálisan rögzített koordináta rendszer



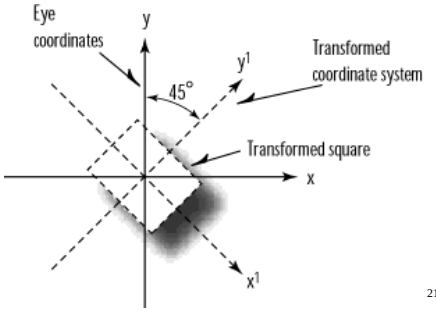
Ahogy a megfigyelő látja a modellt



Így látnánk oldalról a megfigyelőt a pozíciójának a *z* tengely irányába történő elmozdítása után

Nézeti koordináták (OpenGL)

A nézeti koordináta rendszer elforgatása 45^0-kal az ótamutató járásával megegyező irányban



Nézeti (Viewing) transzformáció (OpenGL)

Ez hajtódik végre először, ezt kell legelőször definiálni

Nézőpont meghatározása

- Kezdeti nézőpont (0, 0, 0)
- **`gluLookAt`** paranccsal módosítható

Nézeti (Viewing) transzformáció (OpenGL)

```
void gluLookAt(
   GLdouble eyex, GLdouble eyey,
   GLdouble eyez,
   GLdouble centerx,  GLdouble centery,
   GLdouble centerz,
   GLdouble upx, GLdouble upy,
   GLdouble upz)
```

`(eyex, eyey, eyez)` a szem pozíciója

`(centerx, centery, centerz)`
referenciapont, ahová a szem néz

`(upx, upy, upz)`
felfelé mutató vektor (up-vektor,VUP)

Pl.:

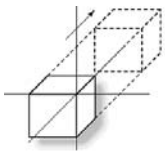
```
gluLookAt(0.0,0.0,2.0, 0.0,0.0,0.0,
  0.0,1.0,0.0);
```


Nézeti (Viewing) transzformáció (OpenGL)

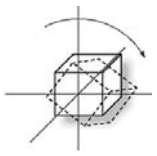
A **`gluLookAt`** eljárás kiszámítja a megadott nézeti transzformáció inverzét, majd az aktuális mátrixot megszorozza a kapott inverz transzformációs mátrixszal

Az altuális mátrix mód a **`GL_MODELVIEW`** legyen!

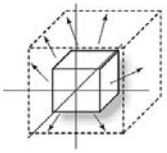
Modell (Modeling) transzformáció (OpenGL)



eltolás (transzláció)



forgatás (rotáció)



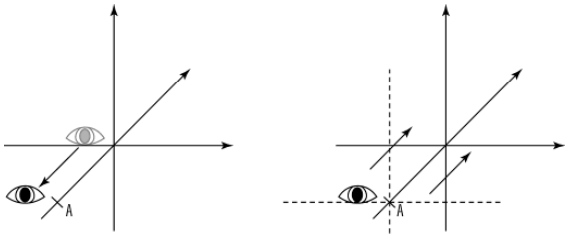
skálázás

A modell vagy egy részének a transzformálására használjuk

A csúcspont (vertex) koordinátákat transzformálja

Modell-nézeti dualitás (OpenGL)

A nézeti és a modell transzformációk duálisak, ezért elegendő csak a modell koordináta rendszert transzformálni



nézeti koordináta rendszer mozgatás

modell koordináta rendszer mozgatás

217

Modell transzformációk (OpenGL)

```
glMatrixMode(GL_MODELVIEW);
```

```
void glRotate{fd}(T a, T x, T y, T z);
```

`a`: forgatás fokban; `(x, y, z)`: forgási tengely

pl. 45 fokos forgatás az *x*-tengely körül:

```
glRotated(45, 1.0, 0.0, 0.0)
```

```
void glTranslate{fd}(T x, T y, T z);
```

`(x, y, z)`: az eltolás vektora

pl.: *x*-tengely mentén 50 egységgel való eltolás

```
glTranslated(50, 0, 0)
```

```
void glScale{fd}(T x, T y, T z );
```

`(x, y, z)` skálázás mértéke a tengelyek mentén

pl.: `glScaled(0.5, 0.5, 0.5)`

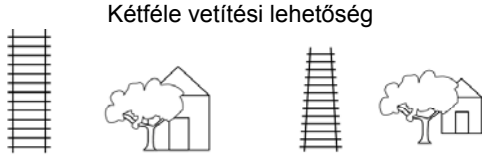
0.5-szörös uniform nagyítás

218

Vetítési (projection) transzformáció (OpenGL)

```
glMatrixMode(GL_PROJECTION);
```

Kétféle vetítési lehetőség



orthografikus és perspektivikus

Megadjuk a látóteret is

Végrehajtás:

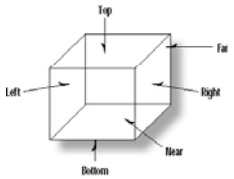
új projekciómátrix =
projekciómátrix · specifikált mátrix

219

Ortografikus vetítés (OpenGL)

```
void glOrtho(double left, double right,
             double bottom, double top,
             double near, double far);
```

Orthogonális (ortografikus) vetítés vágási terének megadása



2D eset:

```
void gluOrtho2D(
        double left, double right,
        double bottom, double top);
```

220

Perspektív vetítés (OpenGL)

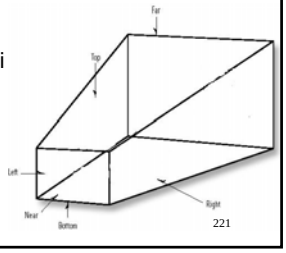
```
void glFrustum (double left, double right,
                double bottom, double top,
                double znear, double zfar);
```

`left, right`: a bal és jobb oldali vágósík *x* koordinátája

`bottom, top`: az alsó és felső vágósík *y* koordinátája

`znear, zfar`: a közeli és távoli vágósík *z* koordinátája.

Nézőpont:
az origó: (0, 0, 0)



221

Perspektív vetítés (OpenGL)

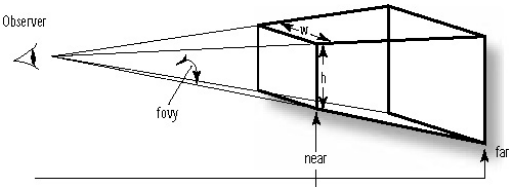
Szimmetrikus látótér megadása:

```
void gluPerspective (double fovy,
 double aspect, double near, double far);
```

`fovy`: a látótér szöge y irányban

`aspect`: w/h

`near, far`: a vágósíkok távolsága a megfigyelőtől



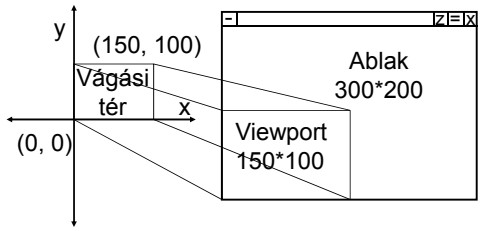
222

Ablak (OpenGL)

2D-s leképzés az ablak egy téglalap alakú (viewport) részébe:

```
void glViewport(GLint x, GLint y,
     GLsizei width, GLsizei height);
```

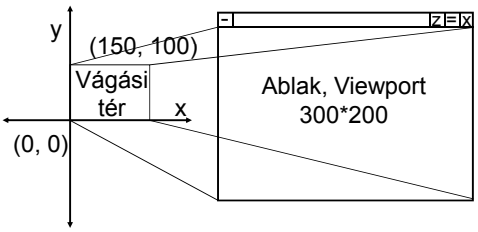
`x, y`: a viewport bal alsó sarka az ablakban,
`width, height`: a viewport mérete pixelben,



223

Ablak (OpenGL)

Alapértelmezés: `(0, 0, winWidth, winHeight)`, ahol `winWidth` és `winHeight` az ablak méretei



224

Perspektív vetítés (OpenGL)

Pl.: Módosítsuk viewport-ot és a vágási teret perspektív vetítésnél

```
void ChangeSize(GLsizei w, GLsizei h){
    GLfloat fAspect;
    if (h == 0)  h = 1; // ne ossszunk 0-val
  // az ablakon beállítjuk a viewport-ot
     glViewport(0, 0, w, h);
    fAspect = (GLfloat)w/(GLfloat)h;
  // vetítési mátrix
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
  // vágási tér megadás, perspektív vetítés
    gluPerspective(60.0f, fAspect,
         1.0, 400.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

225

Transzformációs mátrixok (OpenGL)

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{pmatrix} \rightarrow \begin{pmatrix} \text{Modelview} \\ \text{mátrix} \end{pmatrix} \rightarrow \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} \rightarrow \begin{pmatrix} \text{Projekció} \\ \text{mátrix} \end{pmatrix} \rightarrow$$

Eredeti koordináták — Szem koordináták

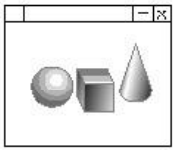
$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} \rightarrow \begin{pmatrix} \text{Perspektív} \\ \text{osztás} \end{pmatrix} \rightarrow \begin{pmatrix} x_c / w_c \\ y_c / w_c \\ z_c / w_c \end{pmatrix} \rightarrow \begin{pmatrix} \text{Viewport} \\ \text{transzf.} \\ \text{mátrix} \end{pmatrix} \rightarrow$$

Vágási koordináták — Normalizált koordináták

226

Transzformációs mátrixok (OpenGL)

$$\rightarrow \begin{pmatrix} x_c / w_c \\ y_c / w_c \\ z_c / w_c \end{pmatrix} \rightarrow \begin{pmatrix} \text{Viewport} \\ \text{transzf.} \\ \text{mátrix} \end{pmatrix} \rightarrow$$



Normalizát (inhomogén) koordináták

227

Mátrix vermek (OpenGL)

Mátrix módok: `GL_TEXTURE, GL_MODELVIEW, GL_COLOR, GL_PROJECTION`

Minden mátrix mód számára van egy mátrix verem

Az aktuális mátrix a verem tetején lévő mátrix.
A műveletek:

```
void glPushMatrix( void );
```

```
void glPopMatrix( void );
```

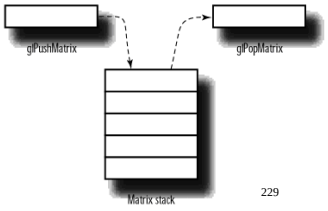
228

Mátrix vermek (OpenGL)

glGet(GL_MAX_MODELVIEW_STACK_DEPTH)
(Microsoft: maximális mélység 32)

glGet(GL_MAX_PROJECTION_STACK_DEPTH)
(Microsoft: maximális mélység 2)

GL_STACK_OVERFLOW, GL_STACK_UNDERFLOW



Alapállapot:
egységmátrix,
GL_MODELVIEW

229

Feladat (OpenGL)

Rajzoljuk meg egy dobókocka perspektivikus képét!

230

Animáció (OpenGL)

Annak érdekében, hogy a megjelenített képek változása „sima” legyen, dupla puffert kell használnunk:

```
glutInitDisplayMode
  (GLUT_DOUBLE | GLUT_RGB);
```

Dupla puffer használata esetén a megjelenítés
glFlush(); helyett
glutSwapBuffers();

231

Animáció (OpenGL)

Fontos, hogy a megjelenítő függvény mindig azonos módon fusson, és mellékhatás mentes legyen!

Ha pl. transzformációt is tartalmaz, akkor a megjelenítő függvény elején vermeljük a módosítandó mátrixot:

```
glPushMatrix();
```

Majd a megjelenítés végén mentsük vissza a veremből:

```
glPopMatrix();
```

A változás globális változók értékének változásán alapuljon!

232

További call back függvények (OpenGL)

```
void glutReshapeFunc(
  (*func)(Glsizei w, Glsizei h));
```

Ha módosítjuk az ablak alakját, méretét, akkor a **glutMainLoop** végrehajtja a
func(w, h) függvényhívást,
ahol **w** az ablak szélessége, **h** a magassága. Így szerezhetünk tudomást az ablak módosított méretéről.

A **func** függvényen belül globális változóba tehetjük **w** és **h** értékét.

A **glutPostRedisplay();** függvényhívással jelezhetjük, hogy föl kell hívni a megjelenítő függvényt.

233

További call back függvények (OpenGL)

```
void glutTimerFunc(unside int ms,
  (*func)(int value), int value);
```

ms millisec lejárta után a **glutMaiLoop** végrehajtja a
func(value) függvény hívást.

Használata:

A **func** függvény módosítja a globális változók némelyikének az értékét, ezáltal módosít a modellen vagy a megjelenítésen, majd meghívja a **glutPostRedisplay**, függvényt a módosítás megjelenítésére, és általában a **glutTimerFunc** függvényt, ez utóbbit azért, hogy a **func** függvény később újra fölhívásra kerüljön.

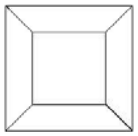
234

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
#include<GL/glut.h>

void glutWireCube(GLdouble size);
```

`size` : a kocka élhossza (a kocka középpontja az origóban lesz)



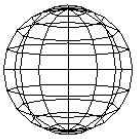
235

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
void glutWireSphere(GLdouble radius,
     GLint slices, Glint stacks);
```

`radius`: a gömb sugara,
`slices`: a *z*-tengely körüli beosztások (szélességi körök) száma,
`stacks`: a *z*-tengely menti beosztások (hosszúsági körök) száma

A középpont az origóban lesz

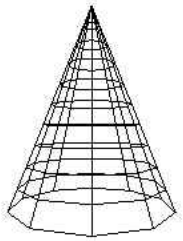


236

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
void glutWireCone(GLdouble base,
     GLdouble height,GLint slices,
     GLint stacks);
```

`base`: a kúp alapjának sugara,
`height` : a kúp magassága,
`slices`: a *z*-tengelyre merőleges beosztások száma,
`stacks` : a *z*-tengelyen átmenő beosztások száma

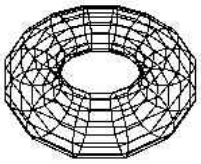


237

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
void glutWireTorus(GLdouble
     innerRadius,GLdouble outerRadius,
     GLint nsides, GLint rings);
```

`innerRadius`: a tórusz belső sugara,
`outerRadius`: a tórusz külső sugara,
`nsides`: a radiális részek oldalainak száma,
`rings`: a tórusz radiális beosztásainak száma.



238

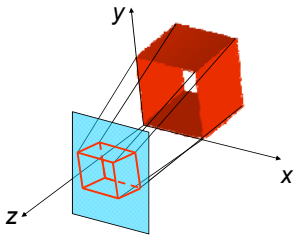
3D geometriai formák megjelenítése (OpenGL)

Tömör formák megjelenítéséhez hasonló függvények állnak rendelkezésre:

```
void glutSolid...
```

239

VETÍTÉSEK

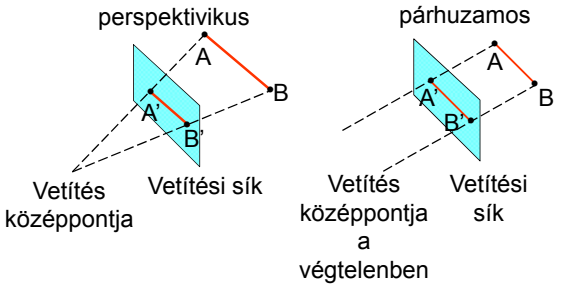


Transzformációk, amelyek *n*-dimenziós objektumokat kisebb dimenziós terekbe visznek át.

Pl. 3D→2D

240

Vetítések fajtái /1



241

Vetítések fajtái /2

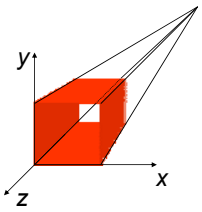
Perspektivikus	**Párhuzamos**
Vetítési középpont	**Vetítési irány**
• közel áll látásunkhoz • általában: nem mérhetők a távolságok (rövidülés) és a szögek	• kevésbé realisztikus • mérhető távolságok, szögek változnak

242

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



243

Perspektív vetítések /2

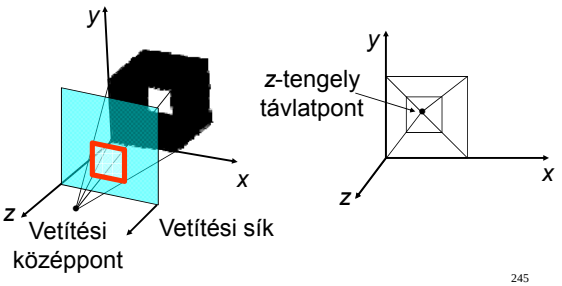
Perspektív vetítések osztályozása:

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

244

Perspektív vetítések /3

1. Egypontos perspektív vetítés



245

Párhuzamos vetítések

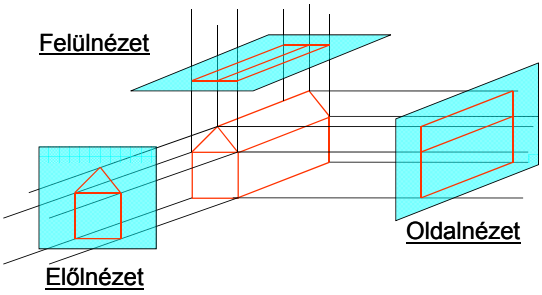
A párhuzamosság megmarad

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

1. Merőleges (ortografikus)
2. Tetszőleges irányú

246

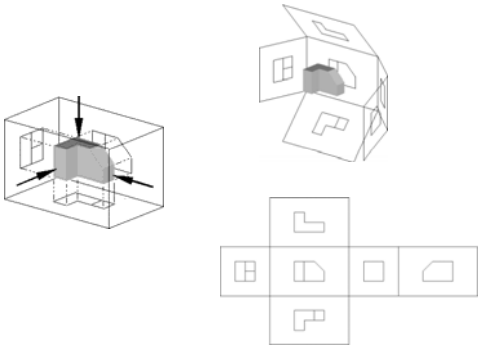
Merőleges (ortografikus) /1



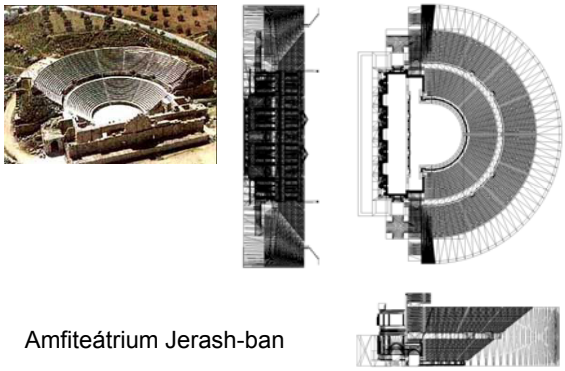
A párhuzamosság megmarad, a távolságok megmaradnak vagy számíthatók

247

Példa



248



Amfiteátrium Jerash-ban

249

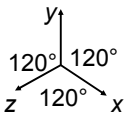
Merőleges (ortografikus) /2

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

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

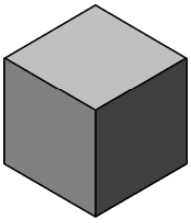
$\pm d_x = \pm d_y = \pm d_z$



8 ilyen irány létezik

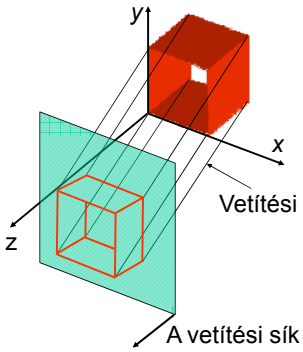
250

Izometrikus vetítés



251

Tetszőleges irányú /1



A vetítési sík normálisa és a vetítési irány nem párhuzamos

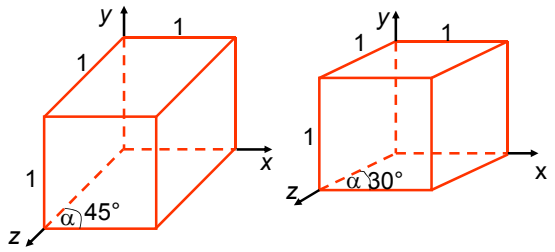
Leggyakoribb fajtái:
- kavalier
- kabinet

252

Tetszőleges irányú /2

Kavalier:

a vetítési irány és a vetítési sík szöge = 45°

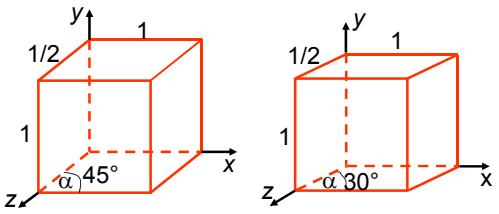


253

Tetszőleges irányú /3

Kabinet:

a vetítési irány és a vetítési sík szöge = arctg(2) = 63,4°



254

3-D megjelenítés specifikálása /1

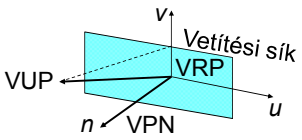
Szükséges: - látótér
- vetítés
} megadása

Vetítési sík megadása:

egy pontjával - referencia pont (VRP)

és a normálisával (VPN)

v a fölfelé mutató vektor (VUP) vetülete irányába mutat



255

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

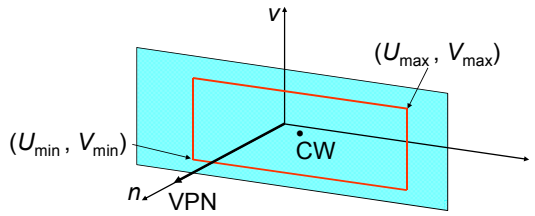
u = olyan, hogy *u, v, n* jobb-kezes derékszögű koordináta rendszert határozzon meg

256

3-D megjelenítés specifikálása /3

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

CW a közepe



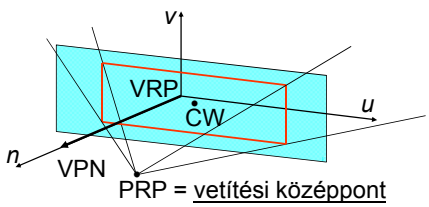
257

3-D megjelenítés specifikálása /4

PRP: vetítési referencia pont:

(párhuzamos és perspektív vetítésre is)

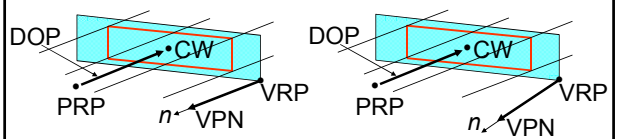
Perspektívikus vetítésnél



258

3-D megjelenítés specifikálása /5

DOP: vetítési irány



DOP || VPN DOP ∦ VPN

259

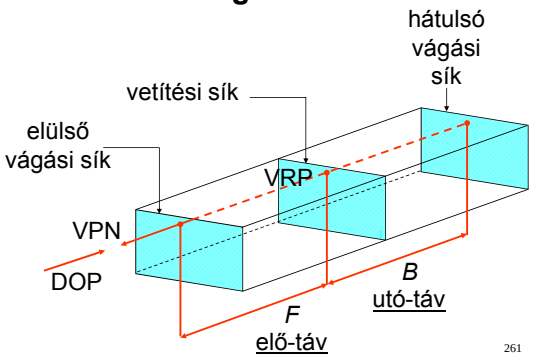
Látótér maghatározása elülső és hátulsó vágási síkokkal /1

Fajtái:

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

260

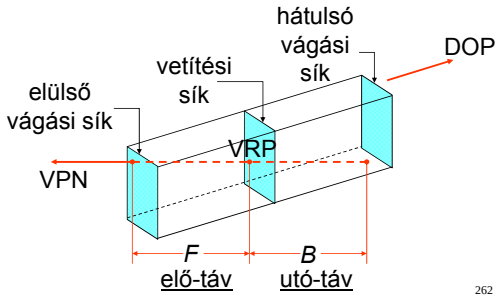
Látótér maghatározása elülső és hátulsó vágási síkokkal /2



261

Látótér maghatározása elülső és hátulsó vágási síkokkal /3

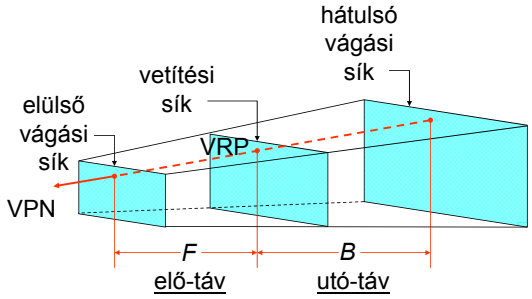
Párhuzamos (tetszőleges irányú):



262

Látótér maghatározása elülső és hátulsó vágási síkokkal /4

Perspektivikus:



263

Vetítések matematikai leírása

264

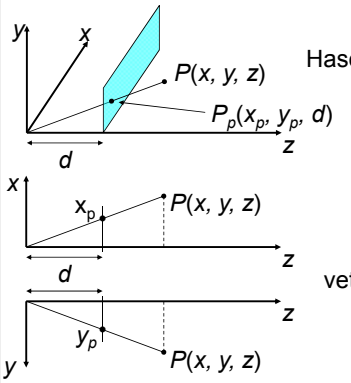
Perspektivikus vetítések /1

Az egyszerűség kedvéért tegyük fel hogy:

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

265

Perspektivikus vetítések /2



Hasonló háromszögekből:

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

vetületi síkok

266

Perspektívikus vetítések /3

Hasonló háromszögekből: $\frac{x_p}{d}=\frac{x}{z},\quad \frac{y_p}{d}=\frac{y}{z},\quad z\neq 0,$
$x_p=\frac{x}{z/d},\quad y_p=\frac{y}{z/d},$

$$\begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} = \left.\begin{pmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{pmatrix}\right\} P_p,$$ mivel ez homogén koordináta,

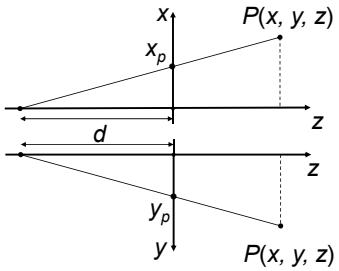
tehát $M_{per}=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$, mert $M_{per}\cdot\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}=\begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix}$

267

Perspektívikus vetítések /4

Más lehetőség

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



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

268

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

269

Párhuzamos ortografikus vetítés

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

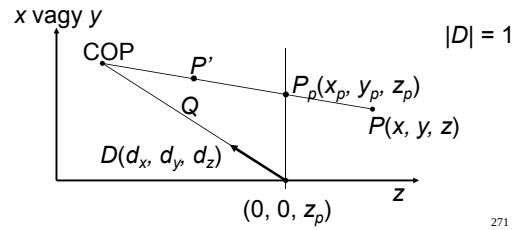
ahol

$M_{ort}=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (határértéke M'_{per}-nek, $d\to\infty$).

270

Vetítések általános alakja /1

Vetítési sík ⊥ z-tengely, $z = z_p$-ben,
COP *Q* távolságra van $(0, 0, z_p)$-től



271

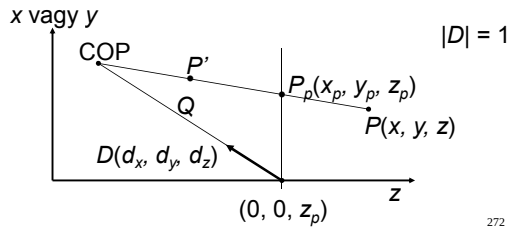
Vetítések általános alakja /2

Parametrikus alak:

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

másrészt

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



272

Vetítések általános alakja /3

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

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

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

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

273

Vetítések általános alakja /4

Behelyettesítve és átalakítva:

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

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

274

Vetítések általános alakja /5

Így

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

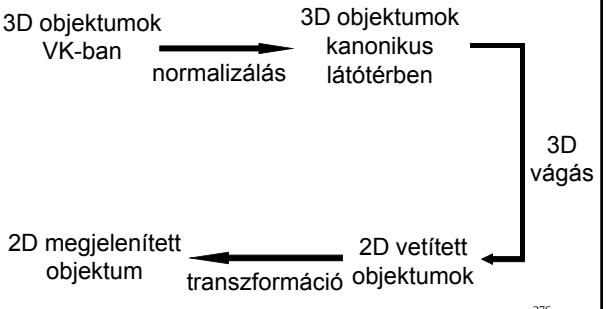
Tartalmazza M_{per}, M'_{per} és M_{ort}-ot (még többet is).

Pl.:

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

275

3D megjelenítés implementálása /1



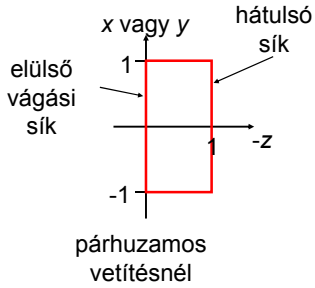
276

3D megjelenítés implementálása /2

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

277

3D megjelenítés implementálása /3



Kanonikus látótér

síkjai:

$x = -1$, $x = 1$

$y = -1$, $y = 1$

$z = 0$, $z = -1$

278

3D megjelenítés implementálása /4

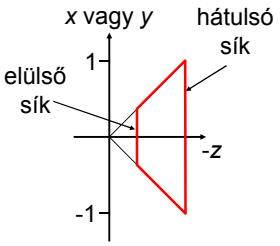
Kanonikus látótér

síkjai:

$x = z$, $x = -z$

$y = z$, $y = -z$

$z = -z_{min}$, $z = -1$



279

LÁTHATÓ VONALAK MEGHATÁROZÁSA

Robert-féle algoritmus

Apple-féle algoritmus

Megszakított vonalak

280

Látható vonalak meghatározása

Tárgy-alapú módszerek

Output: látható élek listája

281

Látható vonalak meghatározása

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)

A fennmaradó esetek:

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

282

Látható vonalak meghatározása

Apple-féle algoritmus

Az élek pontjaihoz hozzárendel egy egész számot a pontot takaró előre néző lapok számát (kvantitatív láthatatlanság)

A kvantitatív láthatatlanság csak akkor változik, ha az él egy un. kontúr vonal mögött halad.

Kontúr vonal: előre és hátra néző lap közötti él.

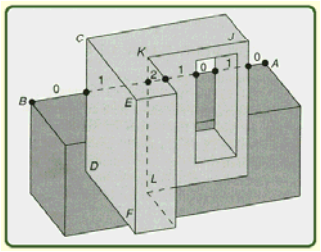
283

Apple-féle algoritmus

A kvantitatív láthatatlanság számítása:

++, ha az él előre néző poligon mögé megy,

- -, ha az él előre néző poligon mögül jön ki.



Az él akkor látszik, ha a kvantitatív láthatatlansága = 0

284

Apple-féle algoritmus

Egymáson átható poligonok nem megengedettek!

Az algoritmus kétféle megvalósítása:

1. Válasszunk ki egy csúcspontot, határozzuk meg a kvantitatív láthatatlanságát (direkt módszer)
2. Haladjunk az éleken, és közben módosítsuk a kvantitatív láthatatlansági értéket, *0* érték esetén rajzolunk

285

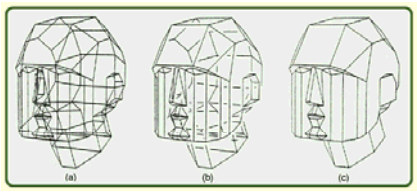
Látható vonalak meghatározása

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

286

Látható vonalak meghatározása

Megszakított vonalak



(*a*): minden vonal látszik
(*b*): mintha minden vonalnak lenne egy takaró sávja, ami eltakarja a mögötte lévő részeket
(*c*): csak a látható vonalak látszanak

287

Megszakított vonalak

Az algoritmus az élek vetületének metszéspontja körül csak a közelebbit rajzolja, a távolabbit megszakítja

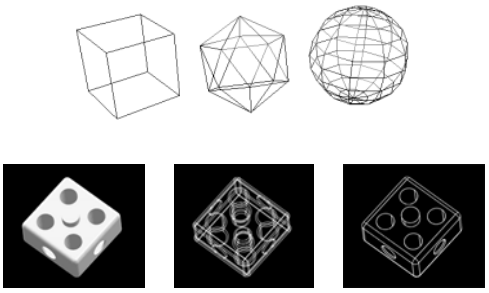
Algoritmus:

Minden vonalhoz megkeressük az előtte levőket
Csak a látható szakaszokat őrizzük meg

Ha minden metszésponttal végeztük, akkor rajzolunk.

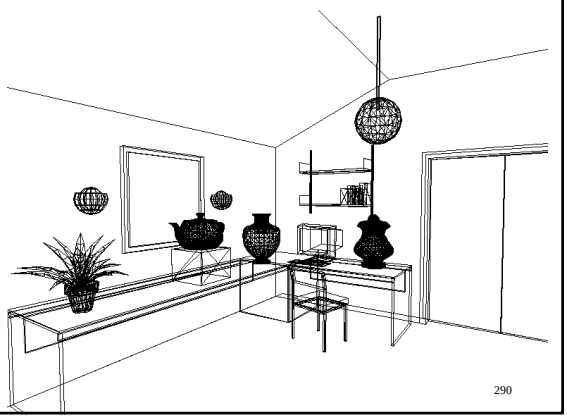
288

Példák



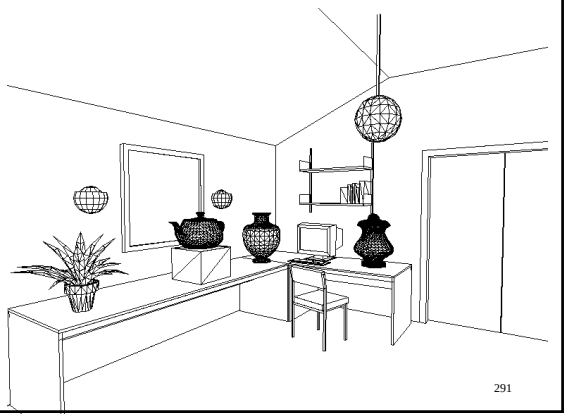
289

Példák



290

Példák



291

LÁTHATÓ FELÜLETEK MEGHATÁROZÁSA

Kétváltozós függvények ábrázolása

A látható felszín meghatározására szolgáló általános algoritmusok

Látható felszín algoritmusok

292

Látható felületek meghatározása

Adott 3D tárgyak egy halmaza, és egy projekció specifikációja.

Mely vonalak és felületek lesznek láthatók?
Melyek lesznek takarva?
Nehéz feladat (időigényes)

Kétféle megközelítés:

293

Látható felületek meghatározása

```
1. 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

294

Látható felületek meghatározása

```
2. for minden tárgyra do begin
     határozzuk meg a tárgynak
     azokat a részeit, amelyek
     nincsenek takarásban saját
     maga vagy más tárgyak által;
     ezeket a részeket rajzoljuk
     ki a megfelelő színnel
   end
```

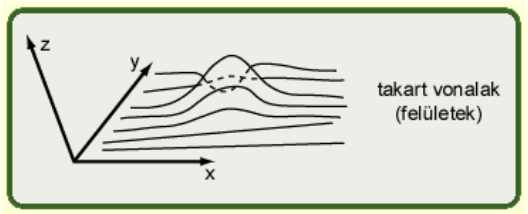
A szükséges idő: $O(n^2)$

295

Kétváltozós függvények ábrázolása

plotterrel

$y = f(x,z)$



296

Kétváltozós függvények ábrázolása

Tegyük fel, hogy *f*-et egy *m* x *n*-es *Y* mátrixszal közelíthetjük.

Drótvázas rajzot készíthetünk szakaszonként lineáris görbéket előállítva *x* és *z* irányban is.

Keressünk olyan algoritmust, amely a takart vonalakat nem rajzolja ki.

297

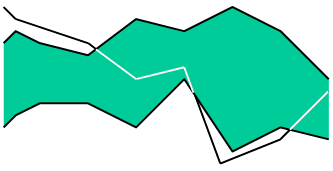
Kétváltozós függvények ábrázolása

1. Ha csak az *x*-tengellyel párhuzamos egyenesek menti értékeket kötjük össze:
   - Haladjunk elölről hátra (a távolabbi vonalak irányába, csak arra kell vigyázni, hogy a már megrajzolt látható felületeket ne "keresztezzük".
   - Elegendő az eddig rajzolt vonalak "sziluettjét" őrizni: csak az látható az új vonalból, ami ez alatt vagy fölött van.
   - Tároljuk minden törésponthoz az eddig rajzolt vonalak maximális és minimális *y* értékét (sziluett), és az új vonal *y* értékeinek megfelelően módosítsuk

   Horizont-vonal algoritmus

298

Kétváltozós függvények ábrázolása

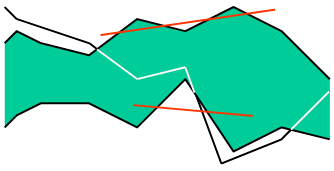


Ha az új vonal valamely szakaszának mindkét végpontja láthatatlan, akkor a szakasz sem látszik.

A részlegesen takart szakaszoknál metszéspontot kell számolni.

299

Kétváltozós függvények ábrázolása

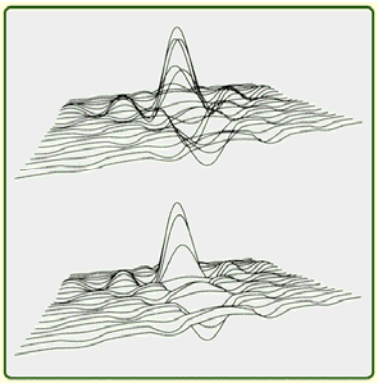


A szakasz nem töréspontól töréspontig, hanem a sziluett töréspontjától töréspontjáig terjed!

A sziluett töréspontjai sűrűsödnek!

300

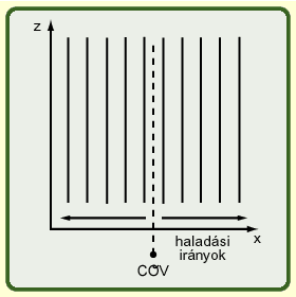
Kétváltozós függvények ábrázolása



301

Kétváltozós függvények ábrázolása

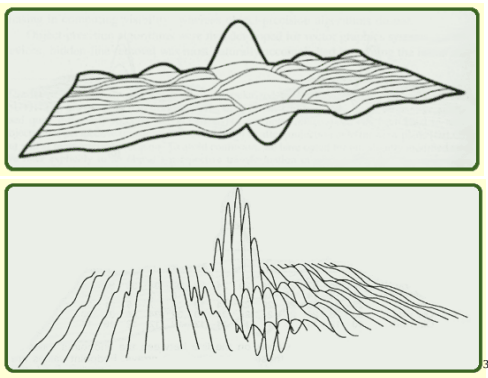
2. Ha csak a *z*-tengellyel párhuzamos egyenesek menti értékeket kötjük össze:



hasonló algoritmus

302

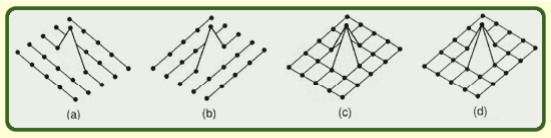
Kétváltozós függvények ábrázolása



303

Kétváltozós függvények ábrázolása

Drótvázas rajz konstans *x*- és *z*-menti görbékből

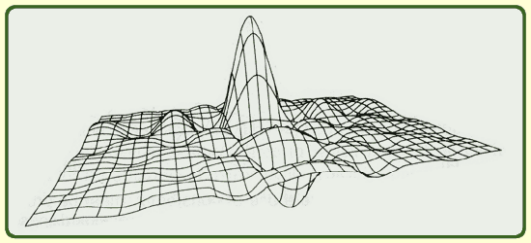


x-menti kép | z-menti kép | A két kép egymásra másolva | A korrekt kép

Nem lehet egyszerűen egymásra rakni a két képet!

304

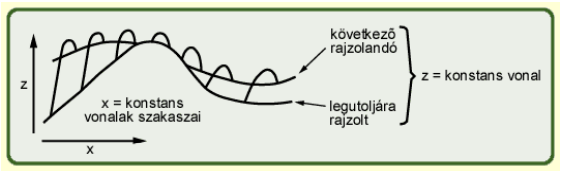
Kétváltozós függvények ábrázolása



Először az *x* irányú vonalakat rajzoljuk meg közelről távolra haladva (mint korábban), de minden vonal megrajzolása után megrajzoljuk a *z* irányú vonalaknak a két utolsó x irányú vonal közti szakaszait (mint korábban), ha látszanak.

305

Kétváltozós függvények ábrázolása



Ezeket az eljárásokat általában csak akkor használjuk, ha a rajzolandó vonalak *x* = *konstans* vagy *z* = *konstans* menti értékekből állnak)

306

A látható felszín meghatározására szolgáló általános algoritmusok

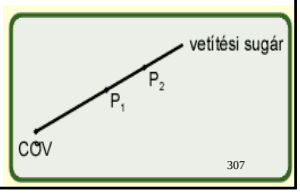
A tárgyak takarják-e egymást?

Mely tárgy látható?

Pontokra: $P_1 = (x_1, y_1, z_1)$ 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.



Nehéz probléma

307

A látható felszín meghatározására szolgáló általános algoritmusok

Mélységbeli összehasonlítás

Helye: a normalizálási transzformáció után, ekkor

a. **párhuzamos vetítésnél**: a vetítési sugarak párhuzamosak a *z-tengellyel*, ekkor P_1 és P_2 ugyanazon a vetítési sugáron van, ha

$$x_1 = x_2 \text{ és } y_1 = y_2$$

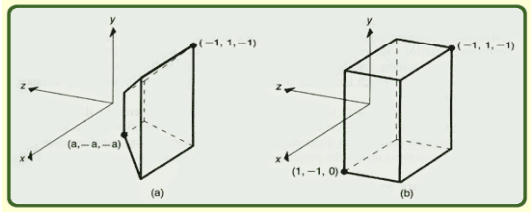
b. **perspektív vetítésnél**: a vetítési sugarak *COV*-ből indulnak ki, ekkor P_1 és P_2 ugyanazon a vetítési sugáron van, ha

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

308

A látható felszín meghatározására szolgáló általános algoritmusok

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



perspektív párhuzamos kanonikus térfogat

309

A látható felszín meghatározására szolgáló általános algoritmusok

Perspektív kanonikus térfogatot párhuzamos kanonikus térfogatba transzformáló mátrix:

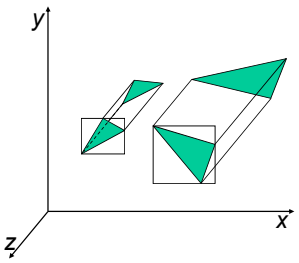
$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & \frac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Ekkor a vetítési sugarak már párhuzamosak a *z*-tengellyel. Egyszerűbben végezhető a vágás

310

A látható felszín meghatározására szolgáló általános algoritmusok

Tárgyak kiterjedése, határoló téglalapok, testek



Határoló-téglalap teszt

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

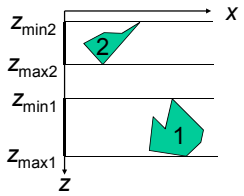
311

A látható felszín meghatározására szolgáló általános algoritmusok

1-dimenziós kiterjedés (határoló intervallum)

minmax-teszt:

A kiterjedés minimális és maximális értékeinek összehasonlításával döntjük el a takarást



A kiterjedés meghatározása:

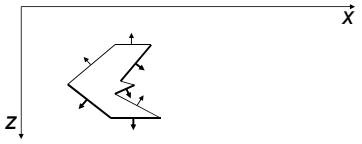
a tárgy (csúcs)pontjaihoz tartozó koordináták *min.* és *max.* értékeiből

312

A látható felszín meghatározására szolgáló általános algoritmusok

Hátra néző lapok kiválogatása

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



313

A látható felszín meghatározására szolgáló általános algoritmusok

Azonosításuk:

$\underline{n}$: normális (n_x, n_y, n_z)

$\underline{v}$: *COV*-ből a poligon tetszőleges pontjába mutat

Ha $\underline{n} \cdot \underline{v} < 0$ előre néz

> 0 hátra néz

$= 0$ csak az éle látszik

Speciálisan: Az (x,y) síkra történő ortografikus vetítés esetén

$n_z < 0$ hátra néz

> 0 előre néz

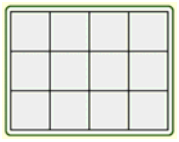
$= 0$ csak az éle látszik

314

A látható felszín meghatározására szolgáló általános algoritmusok

Térbeli partícionálás

Észrevétel: nem minden tárgynak van minden vetítési sugárral metszéspontja (pl. távol vannak, más irány) → osszuk fel (particionáljuk) a képernyőt

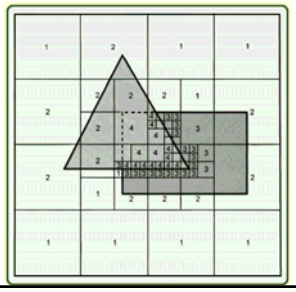


Meghatározzuk, hogy mely tárgyak vetülete van benne a megfelelő részben (partícióban) és csak azokkal keresünk metszéspontokat

315

A látható felszín meghatározására szolgáló általános algoritmusok

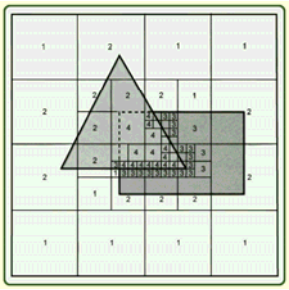
Ez jó módszer, ha a tárgyak vetületei egyenletesen oszlanak el a teljes képernyőn, különben különböző méretű partíciókat érdemes készíteni: kisebb partíciót ott, ahol több tárgy vetülete van



316

Látható felszín algoritmusok

Terület-osztó algoritmus látható felszín meghatározására: "oszd meg és uralkodj"

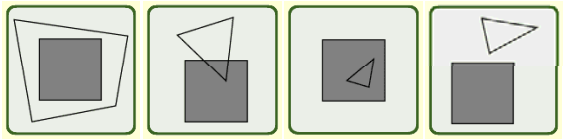


Ha egy területen könnyen eldönthető, hogy melyik poligon jeleníthető meg, akkor azt rajzoljuk ki, különben osszuk fel a területet, és alkalmazzuk az eljárást a rész területekre

317

Látható felszín algoritmusok

4 lehetőség egy poligon és egy téglalap alakú terület között:



Tartalmazó poligon | Metsző poligon | Tartalmazott poligon | Idegen poligon

318

Látható felszín algoritmusok

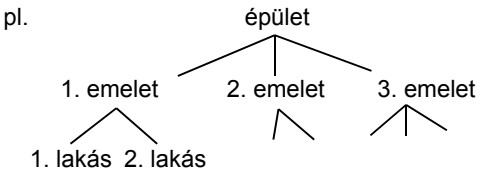
Mikor dönthető el könnyen, hogy mi rajzolható?

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

319

A látható felszín meghatározására szolgáló általános algoritmusok

Hierarchikus struktúrák alkalmazása



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

320

Látható felszín algoritmusok

***Z - buffer* vagy *mélység – puffer* algoritmus**

(kép alapú)

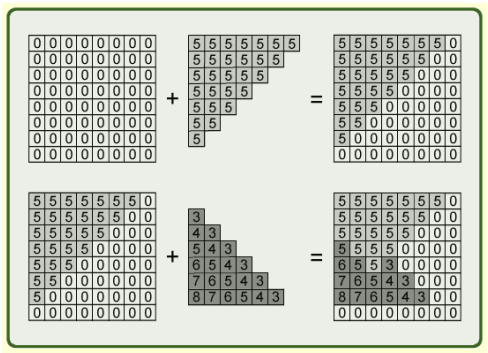
F : kép-puffer (képpontok tárolására) kezdeti értéke: háttérszín

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.

321

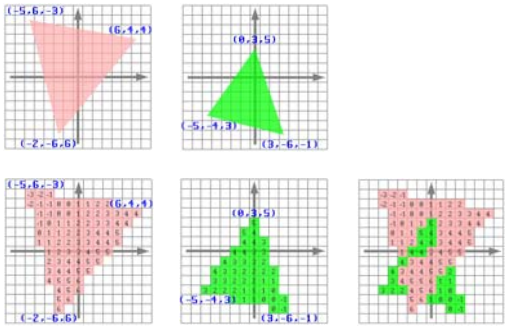
Z-buffer algoritmus



A hátsó vágási sík a $z = 0$

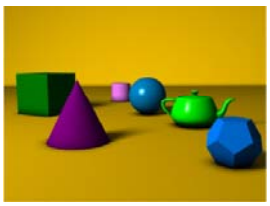
322

Z-buffer algoritmus

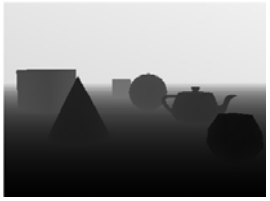


323

Z-buffer algoritmus



Kép



z-buffer

324

Z-buffer algoritmus

Tulajdonságai:

- Nincs tárgyak rendezése, összehasonlítása, metszéspontok számítása
- Polygononként végezhető el, ha nincsenek átható poligonok,
- Nem csak poligonokra jó,
- Nagy helyigény, de lehet sávonként haladni,
- Könnyű implementálni,
- Könnyű egy újabb tárgy képét hozzávenni

325

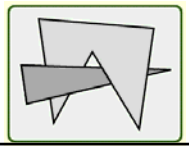
Látható felszín algoritmusok

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

326

Látható felszín algoritmusok

1. Mélység szerint rendező algoritmus

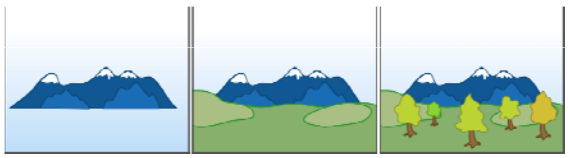
Lépések:

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

327

Festő algoritmus



328

Látható felszín algoritmusok

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 eltakarja-e *Q*-t?

329

Látható felszín algoritmusok

P eltakarja-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 a *Q* síkjának 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* (*x,y*) síkra való vetülete nem átfedő, akkor **nem** különben (hátha *Q*-t kell előbb rajzolni):

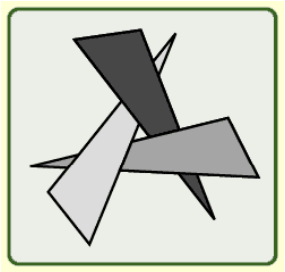
3‘ ha *COV*-ból nézve *Q* teljes terjedelmében a *P* sík túlsó oldalán van, akkor *P* *Q* csere

4‘ ha *COV*-ból nézve *P* teljes terjedelmében *Q*-nak az innenső oldalán van, akkor *P* *Q* csere

különben *P*-t vagy *Q*-t fel kell darabolni a másik síkkal, és a darabokat kell beilleszteni a listába

330

Látható felszín algoritmusok



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

331

Látható felszín algoritmusok

2. <u>Bináris tér-partícionáló fa algoritmus</u>

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

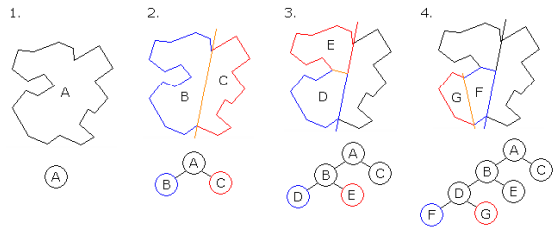
A csomóponthoz tartozó poligon síkjával darabolhatjuk a többi poligont, és azok darabjaival folytathatjuk a fát

bal oldalra: azok a poligonok, amelyek elöl vannak (később kell rajzolni)

jobb oldalra: azok a poligonok, amelyek hátul vannak (korábban kell rajzolni)

332

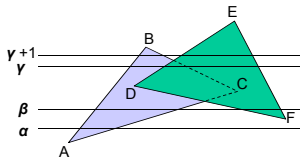
Bináris tér-partícionálás



333

Látható felszín algoritmusok

<u>Pásztázó vonal algoritmus</u> látható felszín meghatározására (hasonló a poligonok kitöltését végző algoritmushoz)



Vízszintesen pászázunk

Most több poligon lehet

334

Látható felszín algoritmusok

ÉT (élek táblázata, l. poligon kitöltése):

A kisebbik ***y*** értékük szerint rendezve az összes élet tartalmazza. **A vízszintes élek kimaradnak**!

Annyi lista van, ahány pásztázó vonal. Minden listában azok az élek szerepelnek, amelyek alsó végpontja a pásztázó vonalon van. A listák az élek alsó végpontjának ***x*** koordinátája, ezen belül a meredekség reciproka szerint rendezettek.

Minden lista elem tartalmazza az él y_{max}, x_{min} koordinátáját és a meredekség reciprokát és a **poligon azonosítóját** (több poligon lehet).

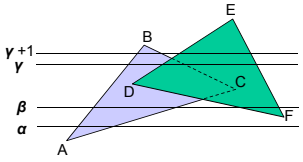
335

Látható felszín algoritmusok

PT (poligonok táblázata):

egy elem részei:

- azonosító
- a sík egyenletének együtthatói a sík egyenlete: $Ax + By + Cz + D = O$
- árnyalati/színezési információ
- ki-be jelző (kezdő érték: ki)



ÉT: *AB AC FD FE CB DE*

PT: *ABC DEF*

336

Látható felszín algoritmusok

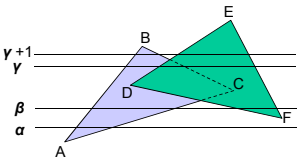
AÉT (aktív élek táblázata):

Alulról felfelé, balról jobbra haladva

α: *AB, AC* (*ABC*)
be ki

β: *AB, AC* (*ABC*), *FD, FE* (*DEF*)
be ki be ki

AB-nél **be**-, AC-nél **ki**kapcsolva
AB-től *AC*-ig *ABC* színe



337

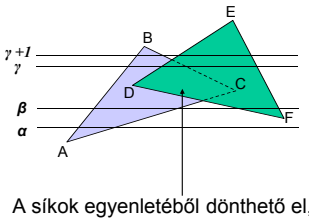
Látható felszín algoritmusok

AÉT (aktív élek táblázata):

Alulról felfelé, balról jobbra haladva

γ: *AB, DE, CB* (*ABC*), *FE* (*DEF*)
be ki be ki

$\gamma + 1$: *AB, CB* (*ABC*), *DE, FE* (*DEF*)
be ki be ki

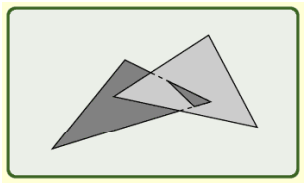


A síkok egyenletéből dönthető el, hogy melyik van közelebb

Újabb sorra térve **AÉT**-t rendezni kell!

338

Látható felszín algoritmusok



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.

339

Látható felszín (OpenGL)

OpenGL-ben: a ***mélységi-*** vagy ***z-buffer*** algoritmus

A mélységbeli összehasonlítást

```
glEnable (GL_DEPTH_TEST)      // engedélyezi
glDisable(GL_DEPTH_TEST)      // tiltja
```

340

Sokszögek oldalai (OpenGL)

Sokszögek (elülső és hátulsó) oldalai OpenGL-ben:
Elülső oldal az, amelyen a csúcspontok az óramutató járásával ellentétes irányban vannak megadva

```
void glPolygonMode(enum face, enum mode);
face: GL_FRONT_AND_BACK, GL_FRONT, GL_BACK
```

Megmondja, hogy a poligon mindkét, vagy csak az elülső vagy a hátulsó oldalát kell rajzolni

`mode:` Megmondja, hogy

- `GL_POINT` csak a poligon csúcsait,
- `GL_LINE` határvonalát kell kirajzolni, vagy
- `GL_FILL` ki is kell tölteni (alapértelmezés)

341

Sokszögek oldalai (OpenGL)

Elülső és hátulsó oldalak explicit megadása:

```
glFrontFace(GLenum mode);
```

`mode:`

- `GL_CW` az az oldal lesz elülső oldal, amelyen a csúcspontokat az óramutató járásával megegyező irányban adtuk meg,
- `GL_CCW` az ellenkezője.

342

Sokszögek oldalai (OpenGL)

A sokszög meghatározott oldalán letiltja a világítási, árnyalási és szín-számítási műveleteket
(láthatatlan oldal)

```
glCullFace(GLenum mode);
mode: GL_FRONT, GL_BACK
```

A culling-ot
`glEnable (GL_CULL_FACE)` engedélyezhetjük
illetve
`glDisable(GL_CULL_FACE)` tilthatjuk

343

MEGVILÁGÍTÁS

Világító tárgyak
Környezeti fény
Szórt visszaverődés
Környezeti fény és diffúz visszaverődés együtt
Tükröző visszaverődés
Poligonokból álló felületek fényességének meghatározása
Gouraud-féle fényesség
Phong-féle fényesség

344

Megvilágítás

a. Világító tárgyak:

A tárgynak saját intenzitású fénye van

A megvilágítás egyenlete:

$$I = k_i$$

k_i - a tárgy saját fényének az intenzitása

független a pont helyzetétől

345

Megvilágítás

b. Környezeti (szórt - ambient) fény:

Minden irányból egyenletes

$$I = I_a k_a$$

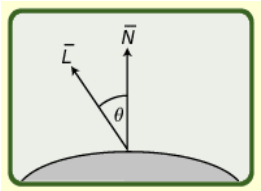
I_a : környezeti fény intenzitása
k_a: a környezeti fény visszaverődési együtthatója (anyagtól függ),
$0 \le k_a \le 1$

346

Megvilágítás

c. Diffúz (diffuse) visszaverődés (Lambert-féle)

Minden irányban ugyanannyi fényt ver vissza.
A felület fényessége (I) függ a fényforrás iránya ($\underline{L}$) és a felület normálisa ($\underline{N}$) közötti szögtől:
$\underline{N}$ *a normális,* $\underline{L}$ a fényforrás irányába mutató egységvektor.



$I = I_p k_d \cos\Theta = I_p k_d (\underline{N}\,\underline{L})$
I_p : a pontforrás intenzitása
k_d: a szórt visszaverődés együtthatója (anyagtól függ), $0 \le k_d \le 1$.

347

Megvilágítás

Környezeti fény (b) és diffúz visszaverődés (c) együtt:

$$I = I_a k_a + I_p k_d (\underline{N}\,\underline{L})$$

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

$$I = I_a k_a + I_p / d_L^2 \, k_d (\underline{N}\,\underline{L})$$

f_{att} (gyengülési faktor)

348

Megvilágítás

Színes fény és felületek esetén a komponensekre:

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

$I_G = ...$

$I_B = ...$

ahol O_{dR} a tárgy szórt vörös komponense

I_{pR} a megvilágítás vörös komponens

$k_d O_{dR}$ visszaverődési hányad komponense

...

Általában:

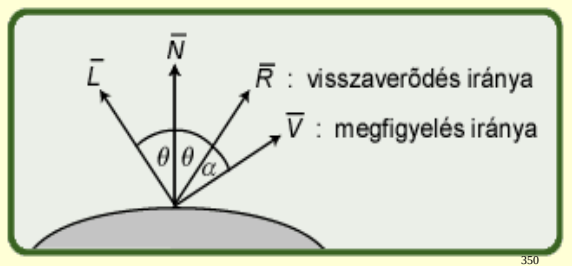
$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + f_{att}\, I_{p\lambda}\, k_d\, O_{d\lambda}\, (\underline{N}\, \underline{L})$

ahol λ a hullámhossz

349

Tükröző (specular) visszaverődés

Fényes (tükröző) felületekről



350

Tükröző (specular) visszaverődés

Phong-féle modell:

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

n: a tükrözési visszaverődés kitevője (csillogás)
(tompa) $1 \le n \le 1000$ (éles fény)

$W(\theta)$: a tükrözötten visszaverődő fény hányada,
lehet konstans, k_s ($0 \le k_s \le 1$),

351

Tükröző (specular) visszaverődés

Phong-féle modell:

A tárgy anyagát is figyelembe véve:

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

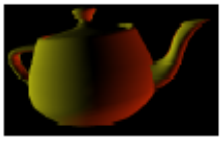
Több fényforrásra:

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

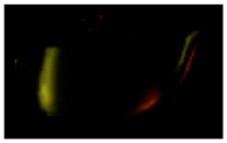
352

Megvilágítási modellek (példa)

szórt



diffúz



tükröző, csillogás = 20

szórt + diffúz + tükröző

353

Poligonokból álló felületek fényességének meghatározása

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

354

Poligonokból álló felületek fényességének meghatározása

1. Konstans fényesség

Az egész poligon ugyanolyan intenzitású. Jó, ha:

- végtelen távoli fényforrás (*N*, *L* konstans),
- végtelen távoli megfigyelő (*N*, *V* konstans) és
- poligon oldalú felület

355

Poligonokból álló felületek fényességének meghatározása

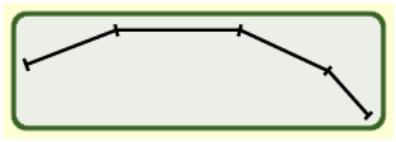
2. Interpolált fényesség

Az intenzitást a csúcsokban számított intenzitásból kapjuk interpolációval

356

Poligonokból álló felületek fényességének meghatározása

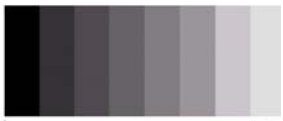
3. Poligon-hálózat fényessége



Az egyes poligonok konstans fényessége csak kiemelné a poligonok közötti éleket

357

Mach-féle hatás

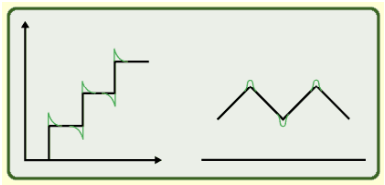


Az intenzitás változását eltúlozva érezzük ott, ahol az intenzitás folytonossága megszűnik.

358

Poligonokból álló felületek fényességének meghatározása

3. Poligon-hálózat fényessége



Mach-hatás

Megoldás:

minden poligon fényességét változó intenzitásúnak generáljuk

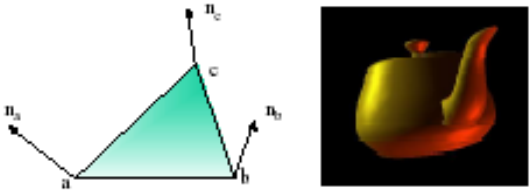
359

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
3. Az élek mentén lineáris interpolációval számoljuk az intenzitást
4. Az élek között (a pásztázó vonalak mentén) lineáris interpolációval számoljuk az intenzitást

360

Gouraud-féle fényesség



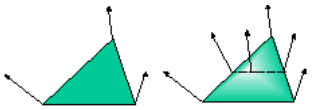
361

Phong-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 Phong

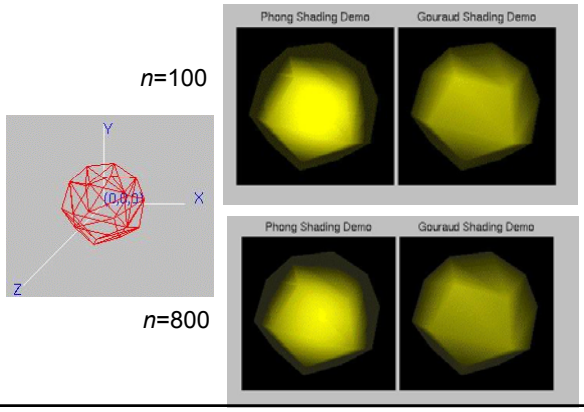
362

Phong- és Gouraud-féle fényesség

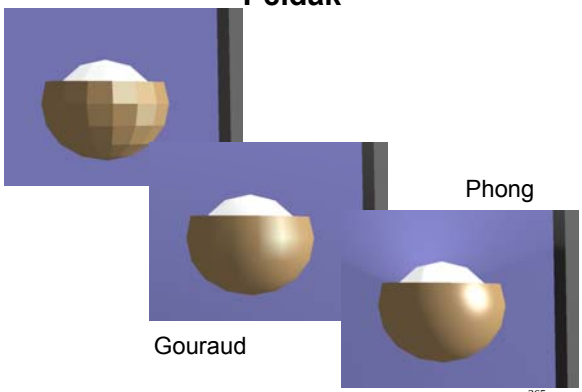
Konstans Gouraud Phong

363

Példák



Példák



365

Példák

szórt

Phong (szórt + diffúz)

Phong (szórt + diffúz + tükröző)

366

Fényforrás (OpenGL)

Világítási komponensek
RGBA értékeivel definiálható:

- **szórt** (ambient)
- **diffúz:** (diffuse) a megvilágított felületről minden irányban azonos a visszaverődés
- **tükröző** (specular)**:** fényes felületről - csillogás

367

Fényforrás (OpenGL)

A specifikálható fényforrások száma: max. 8

- **pozícionált**: az objektum közelében van
- **irányított**: végtelen távoli a pozióvektor negyedik koordinátája 0.0

A fényforrás fénysugara:

- szűk
- fókuszált
- széles

368

Fényforrás (OpenGL)

```
void glLight{if}(enum light, enum pname,
        T param);
void glLight{if}v(enum light, enum pname,
        T *params);
```

`light`: kijelöli a fényforrást (`GL_LIGHT0, ..., GL_LIGHT7`),

`pname`: a beállítandó tulajdonság,

`param`: a beállítandó tulajdonságnak az értéke.

369

Fényforrás (OpenGL)

pname	params, param (alapértelmezés)	Jelentés
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	A szórt fény RGBA intenzitása
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	A diffúz fény RGBA intenzitása
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	A tükröző fény RGBA intenzitása
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	A fényforrás (x, y, z, w) pozíciója
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	A fény (x, y, z) iránya
GL_SPOT_EXPONENT	0.0	Reflektorfény exponens
GL_SPOT_CUTOFF	180.0	Reflektorfény kúpszöge
GL_CONSTANT_ATTENUATION	1.0	Konstans elnyelő faktor
GL_LINEAR_ATTENUATION	0.0	Lineáris elnyelő faktor
GL_QUADRATIC_ATTENUATION	0,0	Négyzetes elnyelő faktor

370

Fényforrás (OpenGL)

A fényforrás távolságával a fény intenzitása gyengül

OpenGL-ben a gyengítő faktor:

$f_{att} = 1/(e_k + e_l ||VP|| + e_n ||VP||^2)$,

e_k: konstans gyengítő faktor
`GL_CONSTANT_ATTENUATION`,

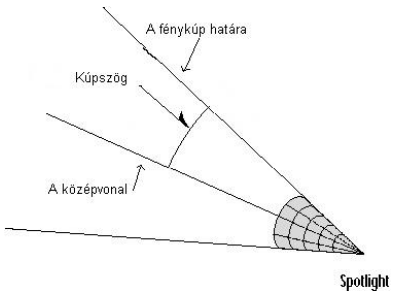
e_l: lineáris gyengítő faktor
`GL_LINEAR_ATTENUATION`,

e_n: négyzetes gyengítő faktor
`GL_QUADRATIC_ATTENUATION`,

$||VP||$: a tárgy és a fényforrás távolsága

371

Reflektorszerű fényforrás (OpenGL)



Kúpszög: `GL_SPOT_CUTOFF`
Középvonal: `GL_SPOT_DIRECTION`
Intenzitás eloszlás: `GL_SPOT_EXPONENT`

372

Világítási modell (OpenGL)

```
void glLightModel{if}(enum pname, T param);
void glLightModel{if}v(enum pname,T *params);
```

pname: a világítási modell tulajdonság kijelölése:

GL_LIGHT_MODEL_AMBIENT a szórt megvilágítás értékeinek megadása

Alapértelmezés: RGBA = (0.2, 0.2, 0.2, 1.0)

373

Világítási modell (OpenGL)

GL_LIGHT_MODEL_TWO_SIDE: egy- vagy kétoldalas világítási számításokat kell alkalmazni a poligonoknál.
Ha **param**=0.0, akkor csak az elülső oldal világít, különben mindkettő

GL_LIGHT_MODEL_LOCAL_VIEWER:
hogyan kell kiszámítani a spekuláris (tükröző) fényvisszaverődés szögét

Alapértelmezés: 0.0: a z tengely irányából, más érték esetén a nézőpontból

374

Objektumok fényvisszaverő tulajdonságai (OpenGL)

Az objektumok tulajdonságai:

- szín komponensek (meghatározzák a fénykomponensek visszavert hányadát),
- fényvisszaverődés: szórt, diffúz és tükröző fény számára,
- az objektumok saját fénye, emissziós érték.

375

Objektumok fényvisszaverő tulajdonságai (OpenGL)

Szín:

```
void glColorMaterial(enum face, enum mode);
```

face: **GL_FRONT, GL_BACK, GL_FRONT_AND_BACK**
Alapértelmezés: **GL_FRONT_AND_BACK**

mode: **GL_EMISSION, GL_AMBIENT, GL_SPECULAR, GL_DIFFUSE, GL_AMBIENT_AND_DIFFUSE**

Alapértelmezés: **GL_AMBIENT_AND_DIFFUSE**

376

Objektumok fényvisszaverő tulajdonságai (OpenGL)

```
void glMaterial{if}(enum face, enum pname,
     T param);
void glMaterial{if}v(enum face,
     enum pname,T params);
```

face: **GL_FRONT, GL_BACK, GL_FRONT_AND_BACK**
pname: a specifikálandó paraméter neve,
param(s): az érték, vagy értékek, amelyre a **pname** által jelzett paramétert be kell állítani.

377

Objektumok fényvisszaverő tulajdonságai (OpenGL)

Paraméter	Alapértelmezés	Jelentés
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	szórt RGBA fényvisszaverés
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	diffúz RGBA fényvisszaverés
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	tükröző RGBA fényvisszaverés
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	emissziós fény intenzitás
GL_SHININESS	0	tükröző exponens
GL_AMBIENT_AND_DIFFUSE		szórt és diffúz szín együtt
GL_COLOR_INDEXES	(0, 1, 1)	szórt, diffúz és tüktöző szín indexek

378

Feladat (OpenGL)

Rajzoljuk meg egy dobókocka perspektivikus képét megvilágítással és a kocka saját színeivel!

379

ÁRNYÉKOLÁS

Az árnyékolás általánosan
Egyszerű árnyék
Árnyék generálása pásztázó-vonal algoritmussal
Kétmenetes árnyékolási algoritmus
Árnyék tér
Kétmenetes *z*-pufferes árnyékolási algoritmus

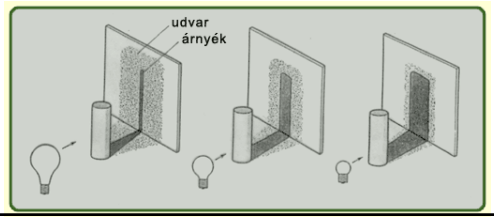
380

Az árnyékolás általánosan

Az árnyékolás általánosan **Árnyék (Shedow)**

Algoritmus, amely meghatározza, hogy melyik felszín látható a fényforrásból nézve

Általában bonyolult (sok mindentől függ, pl. a fényforrás méretétől)



381

Az árnyékolás általánosan

Egyszerűsítés:

Pontszerű fényforrásokra:

$$I_\lambda = I_{a\lambda}\, k_a\, O_{d\lambda} + \sum S_i\, f_{att_i}\, I_{p\lambda_i} [\, k_d\, O_{d\lambda}\, (\underline{N}\, \underline{L_i}) + k_s\, O_{s\lambda}\, (\underline{R_i}\, \underline{V})^n\,]$$

Ahol

$$S_i = \begin{cases} 0, & \text{ha az } i \text{ forrásból ez a pont nem látszik} \\ 1 & \text{különben.} \end{cases}$$

Csak poligon határú testekkel foglalkozunk egyetlen fényforrás esetén: egyszerű árnyék

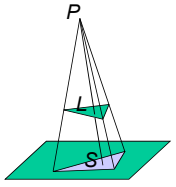
382

Egyszerű árnyék

1. (Blinn 1988) Egyetlen tárgy árnyéka sík felszínen

Pl. pontszerű fényforrás esetén az árnyék a vízszintes síklapon ($z_S = 0$) perspektív vetítéssel

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

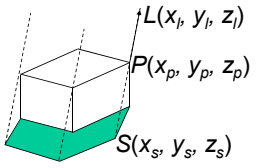


383

Egyszerű árnyék

Tetszőleges helyzetű síklap: transzformáció, vetítés

Pl.: párhuzamos megvilágításnál: síklap → $z = 0$



$$M_{par} = \begin{pmatrix} 1 & 0 & -\frac{x_l}{z_l} & 0 \\ 0 & 1 & -\frac{y_l}{z_l} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

parallel

Árnyékolás:

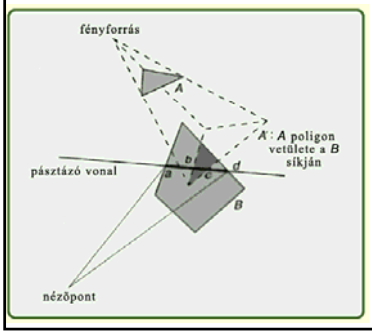
poligon (tárgy) —*projekció*→ vetített poligon —*pásztázás*→ *z*-buffer

384

Egyszerű árnyék

2. Árnyék generálás pásztázó-vonal algoritmussal

(APPEL 1968, BOUKNIGHT, KELLEY 1970)



A látható felszín meghatározására szolgáló pásztázó-vonal algoritmus kibővítése árnyék generálással

385

Árnyék generálása pásztázó-vonal algoritmussal

A ***B*** poligonon az ***A*** poligon vetülete (***A'*** poligon) adja meg az árnyékot

Tehát a pásztázó vonal és ***A'*** metszéspontjai határozzák meg az árnyék határait

Az árnyék „ki-be kapcsolása” ugyanúgy történik, mint a látható felszín meghatározásánál

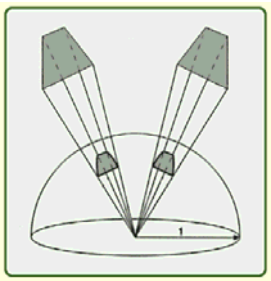
Sok számolás: ***n*** poligon esetén ***n(n-1)*** vetület számítása

386

Árnyék generálása pásztázó-vonal algoritmussal

Gyorsítás: Előfeldolgozás

A poligonok vetítése egy, a fényforrás köré írt gömb felszínére.
Ha a vetületeknek (kiterjedésüknek) nincs közös része, akkor nem árnyékolhatják egymást: ezeket a párokat kiszűrjük



387

Árnyék generálása pásztázó-vonal algoritmussal

Adatstruktúra: látható felszín, árnyék

Algoritmus:

Pásztázás - a látható szakaszokra megvizsgáljuk, hogy árnyékban vannak-e

388

Árnyék generálása pásztázó-vonal algoritmussal

Esetek:

Ha az aktuális pásztázandó szakaszhoz nincs árnyékoló poligon, akkor normális pásztázás, különben

- ha az árnyékoló poligon nem takarja a pásztázó szakaszt, akkor normális pásztázás,
- ha teljesen takarja, akkor árnyékolás,
- ha részben takarja, akkor a pásztázó szakaszt részekre osztjuk, és a részekre megismételjük az eljárást.

389

Egyszerű árnyék

3. Kétmenetes árnyékolási algoritmus

(ATHERON, WEILER, GREENBERG 1978)

Észrevétel: csak azokat a poligonokat kell árnyékolni, amelyek

- láthatók a nézőpontból és
- nem láthatók a fényforrásból

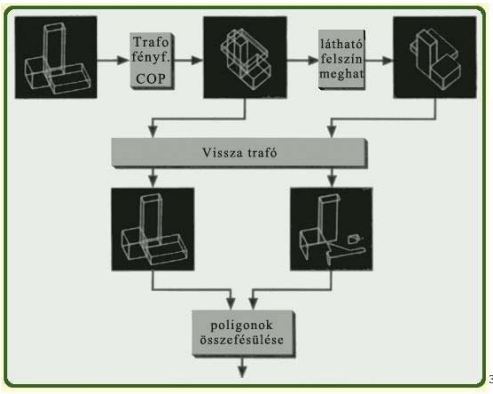
390

Kétmenetes árnyékolási algoritmus

1. lépés: A fényforrásból látható felszín meghatározása (a megvilágított poligonok listája)
   a) transzformáció: vetítés a fényforrásból
   b) látható felszín meghatározása
   c) vissza transzformáció
2. lépés: a poligonok adatbázisának összefésülése (mi van árnyékban és mi nincs)
3. lépés: a nem látható felszínek eltávolítása (a látható poligonok listája)
4. lépés: megjelenítés pl. pásztázó vonal algoritmussal

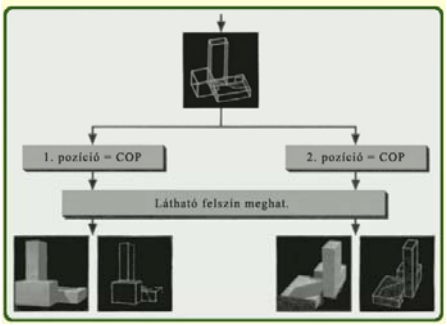
391

Kétmenetes árnyékolási algoritmus



392

Kétmenetes árnyékolási algoritmus



Előnye: jól használható animációra, ha csak a nézőpont változik

393

Egyszerű árnyék

4. Kétmenetes *z*-pufferes árnyékolási algoritmus

1. lépés: számítsuk ki a *z*-puffert a fényforrásból nézve
2. lépés: ha egy pont látható (a megfigyelő helyéről), akkor transzformáljuk a fényforrás-középpontú vetítési rendszerbe, ahol a *z*-puffer tartalmából eldönthető, hogy árnyékban van-e

   ≡

   a transzformált pont távolabb van-e a fényforrástól, mint a *z*-pufferhez tartozó pont?

394

Egyszerű árnyék

5. Árnyék tér (Shadow Volumes) (CROW 1977)

A tárgy árnyék tere: a térnek az a része, amit a tárgy eltakar a fényforrás elől. „Árnyék poligonok" határolják. A fényforrás helyéből és a tárgy kontúrvonalaiból meghatározható az árnyék tér.

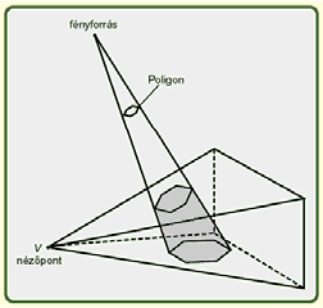
Az árnyék poligonokat nem kell megjeleníteni, de felhasználhatók az árnyékoláshoz.

Az árnyék poligonok síkjának normálisa mutasson az árnyék térből kifelé

395

Árnyék tér

Árnyékszám a *P* pontban: $s(P)$



A nézőpont felé néző árnyék poligon: $s(P)=s(P)+1$

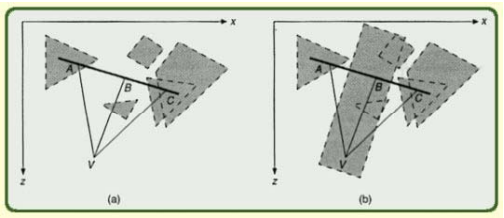
Hátra néző árnyék poligon: $s(P)=s(P)-1$

P árnyékban van ↔ $s(P) > 0$

396

Árnyék tér

Példák: Árnyékban van-e ***A***, ***B*** és ***C***?
V a megfigyelő



VA : + 1 → *s*(*A*) = 1 *VA* : 1 - 1 + 1 → *s*(*A*) = 1
VB : + 1 – 1 → *s*(*B*) = 0 *VB* : 1 + 1 - 1 → *s*(*B*) = 1
VC : + 1 + 1 → *s*(*C*) = 2 *VC* : 1 - 1 + 1 + 1 → *s*(*C*) = 2

397

SUGÁRKÖVETÉS – RAY TRACING

Általános sugárkövetés

Sugárkövetés látható felszín meghatározására

Metszéspontok kiszámítása

398

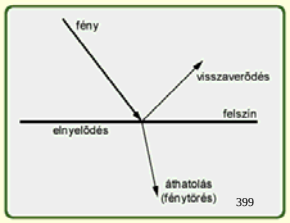
Sugárkövetés – Ray Tracing

Általános sugárkövetés

Módszer realisztikus képek előállítására.

Alapelv: A képen látható felszíni pontok színe (fényessége) más felszíni pontokból kiinduló fénysugarak hatásának az eredménye

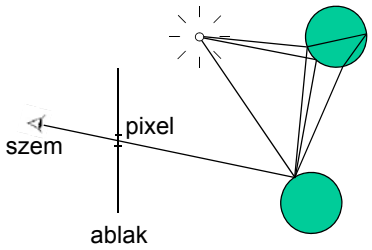
Lehetséges hatások:



399

Sugárkövetés – Ray Tracing

Kövessük a fénysugár útját a nézőponttól kiindulva visszafelé:



400

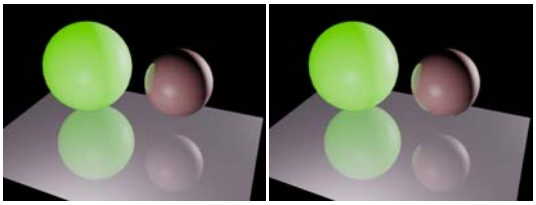
Sugárkövetés – Ray Tracing

Ha egy „fénysugár” találkozik egy tárggyal, akkor a tárgy felszínének az a pontja olyan színű lesz, amit
a pont által kisugárzott,
a pontból az adott irányban visszavert (diffúz és tükröző), valamint
a ponton áthatolt fénysugarak
együttesen határoznak meg.

Ezeknek a fénysugaraknak a színét úgy határozhatjuk meg, hogy az útjukat ugyanígy követjük visszafelé (rekurzió)
Rekurzió *3-4* mélységig

401

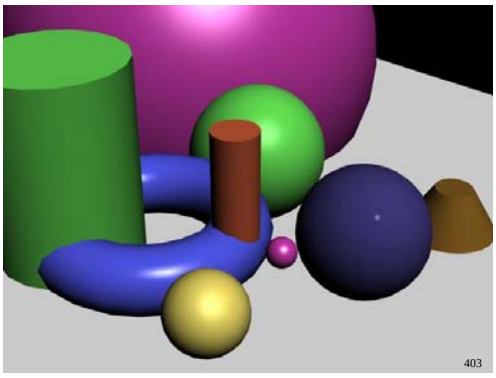
Sugárkövetés – Ray Tracing



Rekurzió *2 illetve több* mélységig

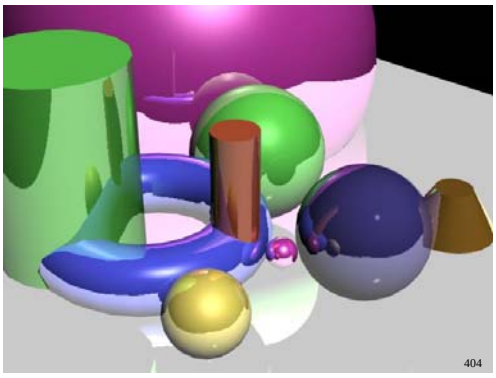
402

Sugárkövetés – Ray Tracing



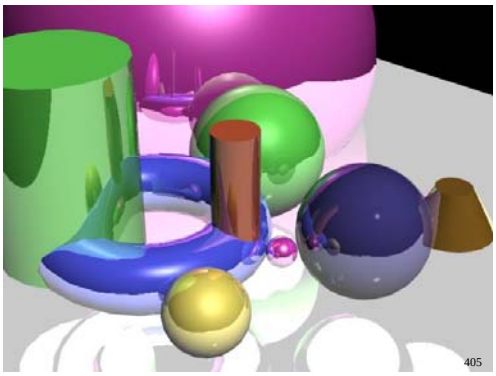
403

Sugárkövetés – Ray Tracing



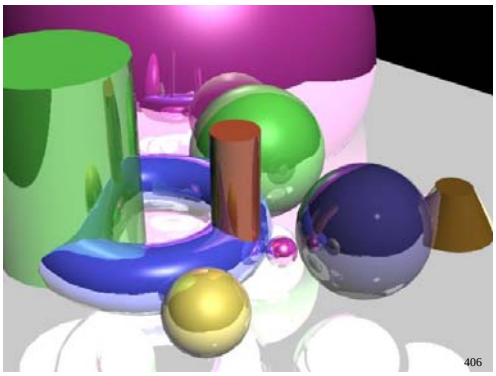
404

Sugárkövetés – Ray Tracing



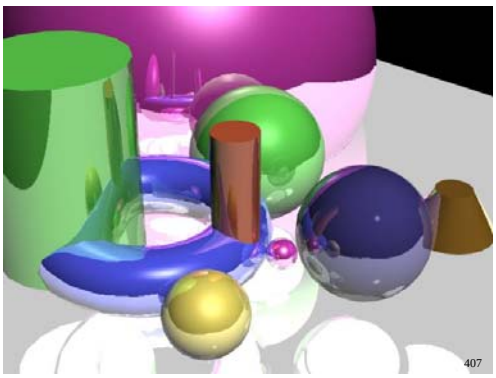
405

Sugárkövetés – Ray Tracing



406

Sugárkövetés – Ray Tracing



407

Sugárkövetés – Ray Tracing

Sok geometriai számítás
(metszéspontok, tartalmazás, …)

Részei:

- Látható felszín meghatározása
- Direkt megvilágítás számolása
- Globális megvilágítás számolása
- Árnyék meghatározása

...

408

Sugárkövetés – Ray Tracing

A sugarak követése független egymástól

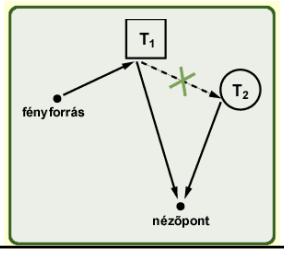
↓

Parallel feldolgozás (transzputer)

409

Sugárkövetés – Ray Tracing

Egyszerűsítési lehetőség: pl. Csak az egyszeres fényvisszaverődést vesszük figyelembe. Akkor csak a direkt megvilágítással kell számolnunk (a más tárgyakról visszavert fényt nem vesszük figyelembe)



410

Sugárkövetés – Ray Tracing

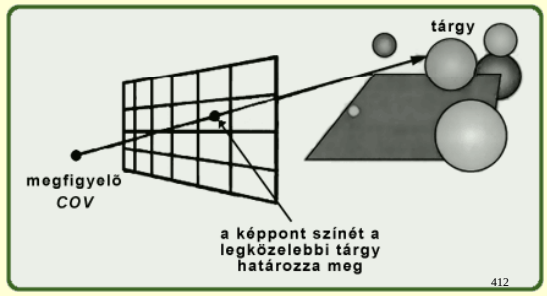
Program:

```
COV és az ablak kiválasztása;
for minden pásztázó vonalra do
 for minden képpontra do
  begin
   fénysugár meghatározása;
   for minden tárgyra do
    if a tárgyat metszi a fénysugár és
       eddig ez az első metszéspont
    then jegyezzük meg a metszéspontot
         és a tárgyat;
   Az első metszésponthoz tartozó tárgy
   színének megfelelő lesz a képpont
  end;
```

411

Sugárkövetés látható felszín meghatározására

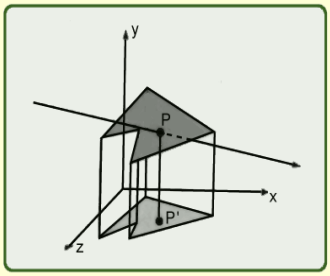
1. Visszafelé követjük a képzeletbeli fénysugár útját a megfigyelőtől a tárgyig



412

Sugárkövetés látható felszín meghatározására

2. Megvizsgáljuk, hogy a metszéspont a poligon belsejében van-e



Párhuzamosan pl. az *xz* síkra vetítünk – az *y* koordinátát elhagyjuk – majd megnézzük, hogy *P'* a vetületben van-e

413

Sugárkövetés látható felszín meghatározására

Tapasztalat:

Sugárkövetésnél az idő 75-95%-a a metszéspontok kiszámításával telik el.

Gyorsítási lehetőségek:

- konstans kifejezések kiszámítása előre,
- poligonok vetületének kiszámítása előre,
- határoló testek használata,
- tárgyak hierarchikus struktúrákba való rendezése, hogy minél kevesebb metszéspontot számoljunk.

414

Metszéspontok kiszámítása

A nézőpont (*COV*): (x_0, y_0, z_0)

A képpont közepe: (x_1, y_1, z_1)

A sugár parametrikus egyenese:

$$x = x_0 + t(x_1 - x_0) = x_0 + t \Delta x$$

$$y = y_0 + t(y_1 - y_0) = y_0 + t \Delta y$$

$$z = z_0 + t(z_1 - z_0) = z_0 + t \Delta z$$

415

Metszéspontok kiszámítása

Metszéspont poligonnal:

Először meghatározzuk az egyenes és a poligon síkjának a metszéspontját:

$$Ax + By + Cz + D = 0$$

$$A(x_0 + t \Delta x) + B(y_0 + t \Delta y) + C(z_0 + t \Delta z) + D = 0$$

$$t = -\frac{Ax_0 + By_0 + Cz_0 + D}{A \Delta x + B \Delta y + C \Delta z}$$

Ha $A \Delta x + B \Delta y + C \Delta z = 0$,
akkor az egyenes és a sík párhuzamos

416

Metszéspontok kiszámítása

Metszéspont gömbbel: középpont: (a, b, c), sugár: r

$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2$$

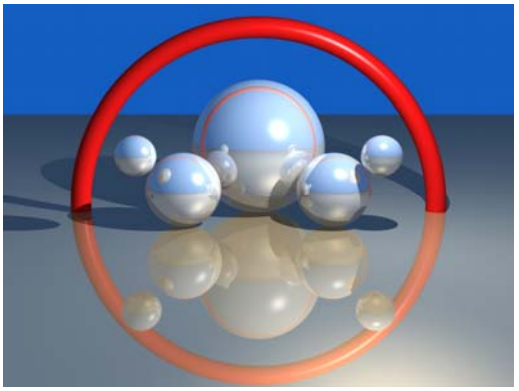
$$(x_0 + t \Delta x - a)^2 + (y_0 + t \Delta y - b)^2 + (z_0 + t \Delta z - c)^2 = r^2$$

t-re nézve másodfokú egyenlet, *2,1,0* db megoldás

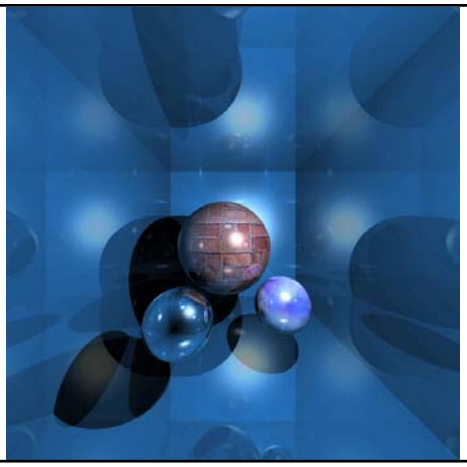
Normális vektor az (x, y, z) metszéspontban:

$$\left(\frac{x - a}{r}, \frac{y - b}{r}, \frac{z - c}{r} \right)$$

417



418

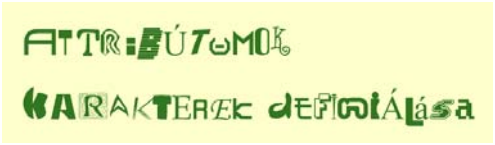


419

420

421

KARAKTEREK GENERÁLÁSA



423

Attribútumok

Attribútumok:

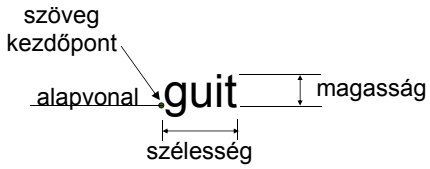
stílus (font): Times
Roman
Helvetica
Clarinda
...

megjelenés: normál
vastag (bold)
döntött (italic)
aláhúzott
...

424

Attribútumok

Méret: *x* pont (*1* pont = *1/72* inch)

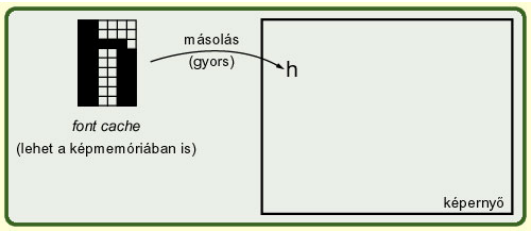


Betűk közötti távolság
Sorok közötti távolság

425

Karakterek definiálása

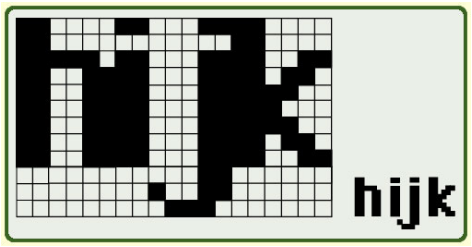
1. Bitmátrixokkal (bittérképekkel)
   az adott font minden egyes karakteréhez tartozik egy bitmátrix



426

Karakterek definiálása

A bitmátrixok előállíthatók pl. rajzoló programmal a betűk felnagyított képeiből



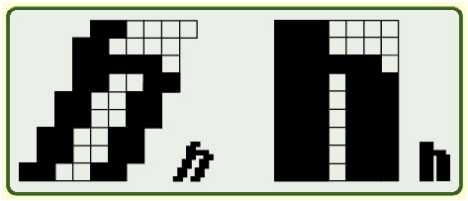
Különböző méretekhez különböző bitmátrixok

427

Karakterek definiálása

Tárolás: név
magasság
szélesség (lehet változó)
betűköz

Lehet trükközni:



428

Karakterek definiálása

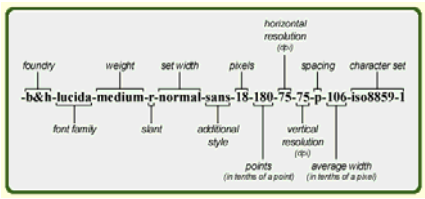
2. A betűket leíró határvonalakkal

- poligonok
- ívek

- Eszköz-független
  bármilyen megjelenítőhöz adaptálható
- Egy font megadásához általában több hely kell,
  mint a bitmátrix esetében
- Mérettől független, nagyítható (mértékkel), dönthető
- Bonyolultabb a megjelenítés, mint a bitmátrix esetében

429

Karakterek definiálása

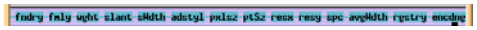


430

Karakterek definiálása

xfontsel Grafikus felület X11 font-név kiválasztására

Megmutatja, hogy mely fontok állnak az X-szerver rendelkezésére



fndry (Foundry) A fontot szolgáltató (regisztrált) cég neve (pl. Adobe)

fmly Font-család (family) a font tipográfiai stílusának családja (pl. Courier)

wght (Weight) A font tipográfiai súlyát, azaz feketeségét jelöli (pl. bold)

slant A betűkép állása (pl. italic - döntött)

431

Karakterek definiálása

sWdth (Set Width) A font vízszintes vastagsága (pl. normal, keskeny)

adstyl (Additional Style) További információ (pl. sans - talp nélküli)

pxlsz (Pixel Size) Magasság pixelben kifejezve

ptsz (Point Size) A font mérete pontokban kifejezve

resx (Horizontal Resolution) Az eredeti fontok mérete dpi-ben

432

Karakterek definiálása

resy (Vertical Resolution) Az eredeti fontok mérete dpi-ben

spc (Spacing) Betűköz (*p*: proporcionális, *m*: mono-space)

avgWdth (Average Width) Átlagos szélesség pixel/*10* -ben

rgstry (Character Set) ISO-szabvány kódja (pl. ISO8859)

encdng (Encoding) ISO-szabvány kódja (pl. ISO8859)

433

Szöveg - bittérkép (OpenGL)

Bittérképes karakter megjelenítése

```
glutBitmapCharacter(void *font,
    int character)
```

`font`: pl.:

```
    GLUT_BITMAP_8_BY_13,
    GLUT_BITMAP_TIMES_ROMAN_10,
    GLUT_BITMAP_HELVETICA_18
```

434

Szöveg - bittérkép (OpenGL)

Pl.:

```
void output(int x, int y, char *string){
  int len, i;
  glRasterPos2f(x, y);
  len = (int) strlen(string);
  for(i = 0; i < len; i++){
    glutBitmapCharacter(
      GLUT_BITMAP_HELVETICA_18,
      string[i]);
  }
}
```

435

Szöveg - határvonal (OpenGL)

Határvonalával megadott (ún. stroke) karakter megjelenítése

```
glutStrokeCharacter(void *font,
    int character)
```

`font`: pl.

```
    GLUT_STROKE_ROMAN,
    GLUT_STROKE_MONO_ROMAN
```

436

Szöveg - határvonal (OpenGL)

Pl.:

```
void output(int x, int y,
       char *string){
  int len, i;
  glRasterPos2f(x, y);
  len = (int) strlen(string);
  for(i = 0; i < len; i++){
     glutStrokeCharacter
       (GLUT_STROKE_ROMAN, string[i]);
  }
}
```

437

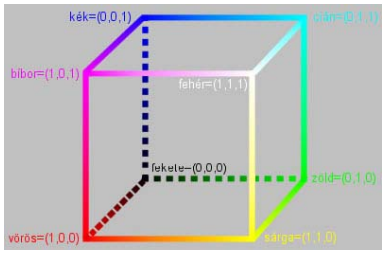
SZÍNMODELLEK

RGB, CMY, CMYK, HSV
OpenGL

438

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

Pl. színes képernyőnél

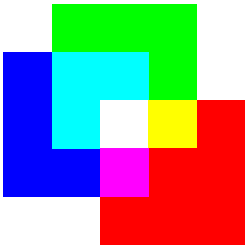


Additív komponensek:

Alkalmas súlyokkal vett összegük ad egy összetett színt

439

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

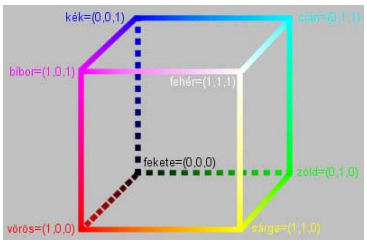


Vörös
Zöld
Kék
Cián
Sárga
Bíbor
Fehér

440

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

Pl. színes nyomtatóknál

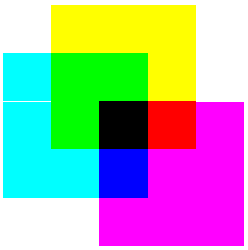


Szubtraktív komponensek:

Sokszor szűrőként használjuk, hogy kiszűrjék a fehérből a megfelelő színt

441

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



Cián
Bíbor
Sárga
Vörös
Zöld
Kék
Fekete

442

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

Konverzió:

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

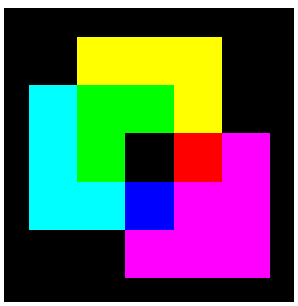
RGB → CMY

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

RGB ← CMY

443

CMYK (Cyan, Magenta, Yellow, blacK - K : fekete)



Cián
Bíbor
Sárga
Vörös
Zöld
Kék
Fekete

444

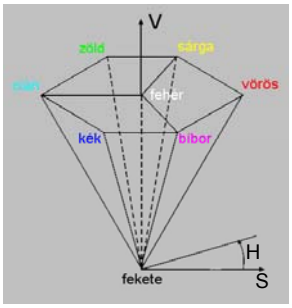
CMYK (Cyan, Magenta, Yellow, BlacK - K : fekete)

CMK → CMKY konverzió:

- **K = min {C, M, Y}**
- **C = C – K**
- **M = M – K**
- **Y = Y – K**

445

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



Esztétikai alapú (ahogy a festők keverik a színeket)

$0^0 \le H < 360^0$
0^0: R, 120^0: G, 240^0: B

$0 \le S < 1$

Ha S =0, akkor szürke
Ha S = 1, akkor nincs fehér és fekete belekeverve

$0 \le V < 1$

Ha V = 0, akkor fekete
Ha V = 1, akkor nincs fekete belekeverve

446

Feladat (OpenGL)

Szivárvány rajzolása

447