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.

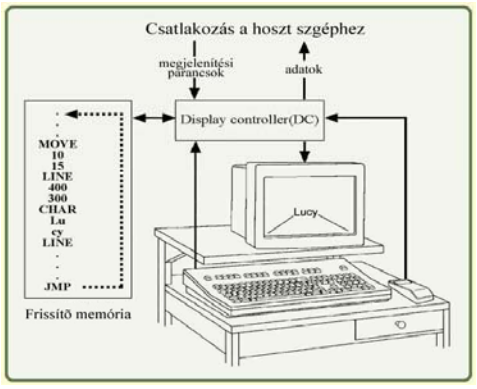
<u>vektor-képernyő</u> (szakaszokat rajzol -tól -ig)

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

utasítás → képernyő processzor → koordináták → vektor generátor → elektromos jel

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

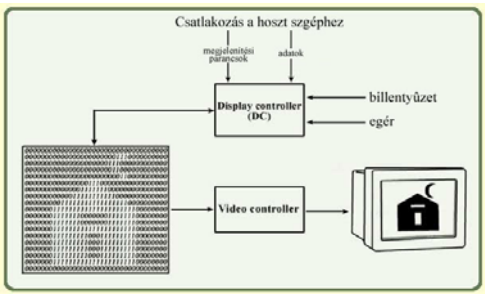
1970-es évek:

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:

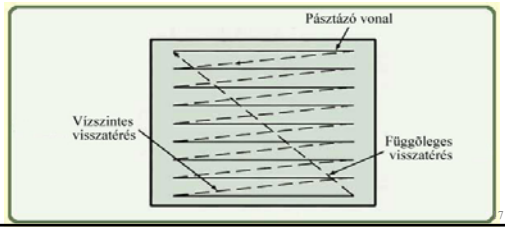
- Olcsó logikájú processzor (soronként olvas)
- A területek színekkel kitölthetők
- Az ábra bonyolultsága nem befolyásolja a megjelenítés sebességét

16

Történeti áttekintés XII

Hátrányai:

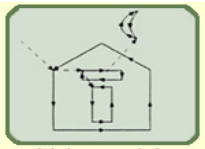
- A grafikus elemeket (pl. vonal, poligon) át kell konvertálni (RIP - raster image processor)
- A geometriai transzformációk számításigényesek



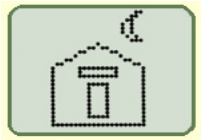
17

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)

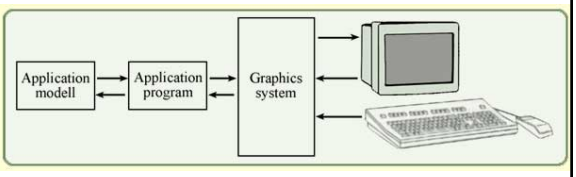
- Logikailag kapcsolódó primitívek csoportosítása szegmensekbe,
- 3D primitívek egymásba ágyazott hierarchiája,
- Geometriai transzformációk,
- Képernyő automatikus frissítése, ha az adatbázis változik

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 frissítése; break;
    ...
    case(quit) exit(0);
}
```

24

A számítógépes grafika osztályozása

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

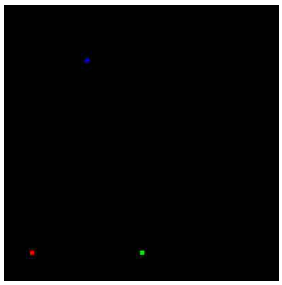
OpenGL

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 ub us ui}
    (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 törlés
     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 (a képenyő jobb felső sarka az orogó)

```
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

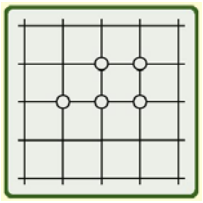
44

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:

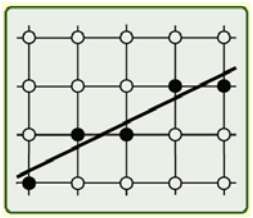


képpont (= körlap), amely a négyzetháló csúcspontjaiban helyezhető el. A koordináták:
egész számok

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

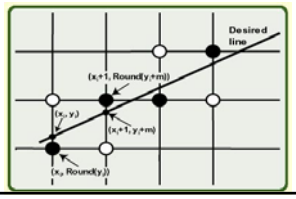
1. Alap inkrementális algoritmus

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:

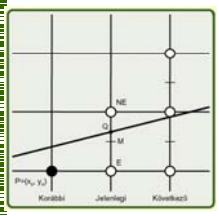
(ha $|m|>1$, akkor x-et cseréljük y-nal)

```
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 választjuk *NE* és *E* közül, amelyik közelebb van a *Q* metszésponthoz.

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

Tegyük fel, hogy:

$$x_0 < x_1 , y_0 < y_1$$

49

Felezőpont algoritmus egyenesre

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

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

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

50

Felezőpont algoritmus egyenesre

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

Felezőpont kritérium:

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

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

A következő pontnál:

ha ***E***-t választottuk, akkor

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

ha ***NE***-t választottuk, akkor

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

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

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

függvényt.

52

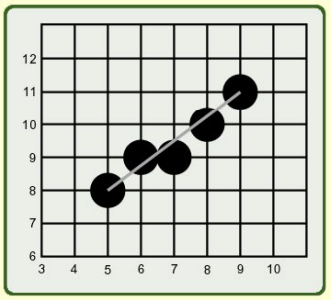
Felezőpont algoritmus egyenesre

```
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.

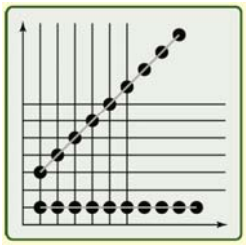
Legyen a választás:

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

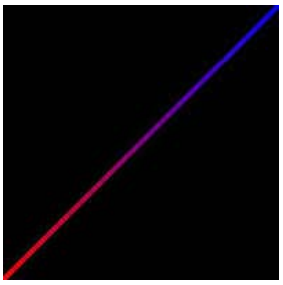
OpenGL

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() {
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(200,200);
  glutInitWindowPosition(100,100);
  glutCreateWindow(„szakasz");
  init();
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

61

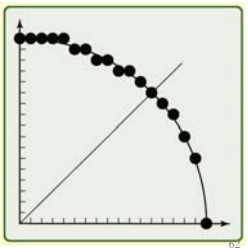
Kör rajzolása

$x^2+y^2 = R^2$ 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

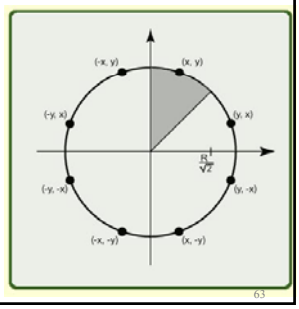
2. Polárkoordinátás alak

Elég egy nyolcad kört kiszámítani:

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

Θ 0°-tól 45°-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
```

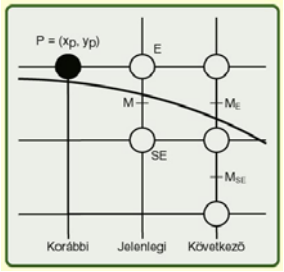
64

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

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

$d = F(M) =$

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

66

Felezőpont algoritmus körre

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

A következő pontnál:

ha ***E***-t választottuk, akkor

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

$$= 2x_p+3$$

ha ***SE***-t választottuk, akkor

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

$$= 2x_p - 2y_p+5$$

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*)

$d = F(1, R - 1/2) = 5/4 - R$

68

Felezőpont algoritmus körre

```
void MidpointCircle(int R, int value) {
   float d;
   x = 0; y = R; d = 5/4-R;
   CirclePoints(x,y,value);
   while(y > x) {
      if(d < 0) {
         x++; d += 2*x+3;
      } else {
         x++; y--;
         d += 2*(x-y)+5;
      }
      CirclePoints(x,y,value);
   } // while
} // MidpointCircle
```

69

Felezőpont algoritmus körre

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

$h = d - ¼$ $\qquad$ $h+¼ = d < 0$

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 ekvivalens $h < 0$ -val, tehát egész aritmetika használható.

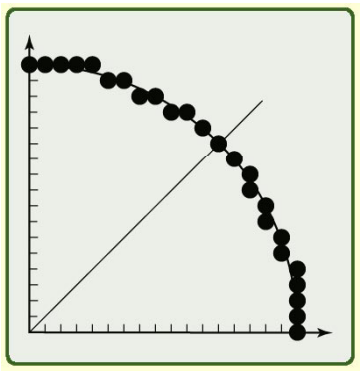
70

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
```

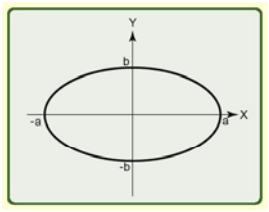
71

Felezőpont algoritmus körre



72

Ellipszis rajzolása



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

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

a, b egész

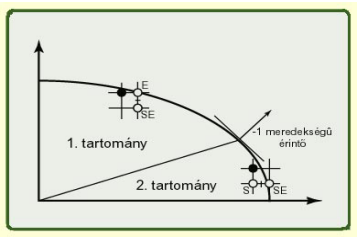
$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2$

Szimmetria miatt:
elég az első síknegyedben megrajzolni

73

Ellipszis rajzolása

Da Silva algoritmusa (felezőpont algoritmus)
Bontsuk a negyedet két tartományra:



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

74

Da Silva algoritmusa

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

Az 1. tartományban:

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

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

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

75

Da Silva algoritmusa

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

Kezdés:
kezdőpont: $(0, b)$
felezőpont: $(1, b - ½)$
$d = F(1, b - ½) = b^2 + a^2(-b + ¼)$

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

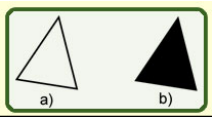
GRAFIKUS PRIMITÍVEK KITÖLTÉSE

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)



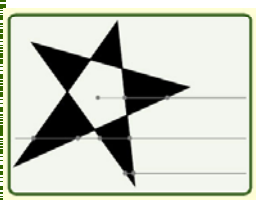
80

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

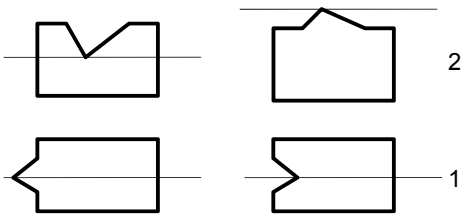
Primitívek kitöltésének az elve:

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

82

GRAFIKUS PRIMITÍVEK KITÖLTÉSE

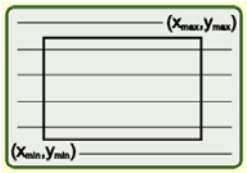
Csúcspontok metszésekor:



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

83

Téglalap kitöltése



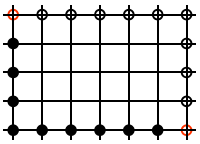

```
for(y = ymin; y < ymax; y++)
   for(x = xmin ; x < xmax; x++)
     WritePixel(x,y,value);
```

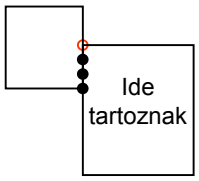
Probléma:
Egész koordinátájú határpontok hova tartozzanak?

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

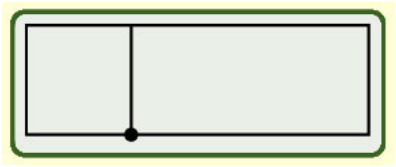
Téglalap kitöltése

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

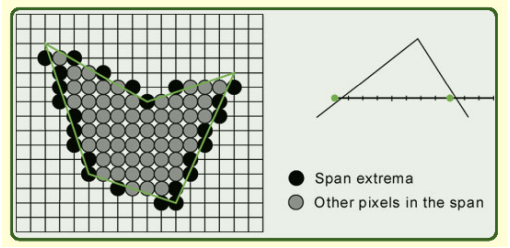
A poligon lehet:
- konvex,
- konkáv,
- önmagát metsző,
- lyukas

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);

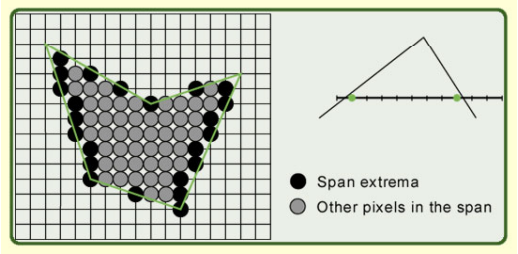


Diszjunkt poligonoknak lehet közös képpontjuk

88

Poligon kitöltése

b) A végpontokat a poligonhoz tartozó képpontok közül választjuk



89

Algoritmus poligonok kitöltésére

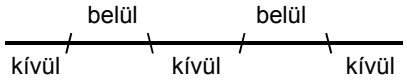
Minden pásztázó egyenesre:

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

90

Algoritmus poligonok kitöltésére

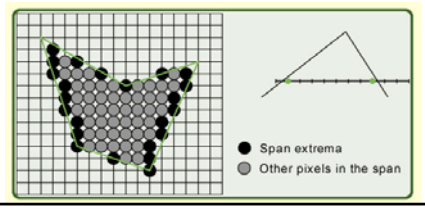
3. A poligon belsejébe tartozó szakasz(ok) végpontjai közötti képpontok kirajzolása
Használjuk a páratlan paritás szabályát:
Tegyük fel, hogy a bal szélen kívül vagyunk, utána minden egyes metszéspont megváltoztatja a paritást



91

Algoritmus poligonok kitöltésére

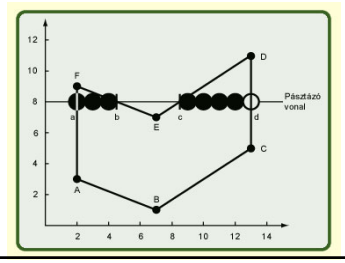
3.1 Adott ***x*** nem egész értékű metszéspont.
Ha kívül vagyunk, akkor legyen a végpont a fölfelé kerekített ***x***
Ha belül vagyunk, akkor legyen a végpont a lefelé kerekített ***x***



92

Algoritmus poligonok kitöltésére

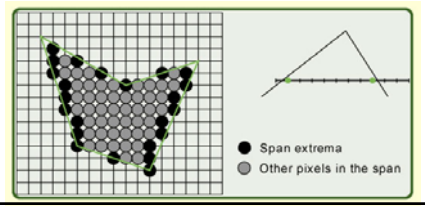
3.2 Adott ***x*** egész értékű metszéspont
Ha ez bal végpont, akkor ez belső pont
Ha jobb végpont, akkor ez külső pont



93

Algoritmus poligonok kitöltésére

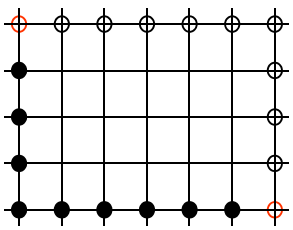
3.2.1 A poligon csúcspontjaiban:
y_{min} csúcspont beszámít a paritásba
y_{max} csúcspont nem számít a paritásba, tehát y_{max} csúcspont csak akkor lesz kirajzolva, ha az a szomszédos él y_{min} pontja is



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

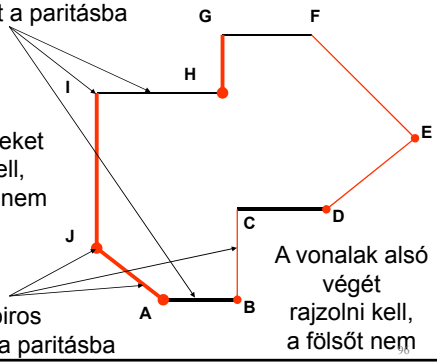
Példa poligon kitöltésére

A fekete nem számít a paritásba

A vastag éleket rajzolni kell, a vékonyat nem

A piros beszámít a paritásba

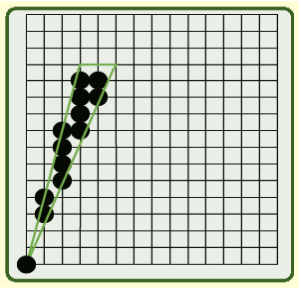
A vonalak alsó végét rajzolni kell, a fölsőt nem



96

Poligon kitöltése

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



97