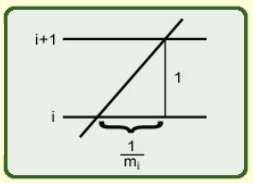
Poligon kitöltése

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

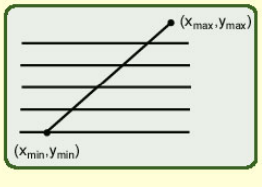
$$x_{i+1} = x_i + \frac{1}{m_i}$$



1

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

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

vagy $[x_{i+1}] = [x_i]$ és $\{x_{i+1}\} = \{x_i\} + \Delta x$

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

2

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
(az lesz belső pont)

Egész értékű aritmetika használható:
törtrész helyett a számláló és nevező tárolása

3

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;
  denominator = ymax-ymin;
  increment = denominator; // mert (x, y+1) a bal
                           // él fölött/előtt van
  for(y = ymin; y < ymax; y++) {
    WritePixel(x,y,value);
    increment += numerator;
    if(increment > denominator) {
       x++; increment -= denominator;
    }
  }
}
```

4

Poligon kitöltése

Adatstruktúrák:

ÉT: (Élek Táblázata)

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

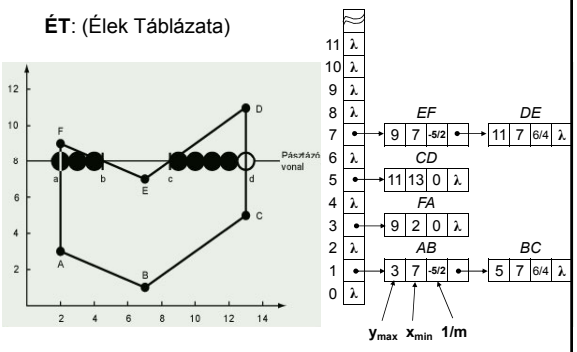
Annyi lista van, ahány pásztázó vonal. Minden listában azok az élek szerepelnek, amelyek alsó végpontja a pásztázó vonalon van. A listák az élek alsó végpontjának ***x*** koordinátája szerint rendezettek

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

5

Poligon kitöltése

ÉT: (Élek Táblázata)



6

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.

7

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:

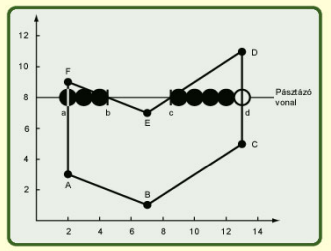
8

Algoritmus poligon kitöltésére

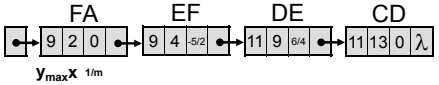
3.1 **ÉT**-ből az ***y*** -hoz tartozó listát – a rendezést megtartva – **AÉT**-hez másoljuk
3.2 **AÉT**-ből kivesszük azokat az éleket, amelyekre $y_{max} = y$ (a fölső éleket nem töltjük ki)
3.3 A kitöltési szakaszok pontjait megjelenítjük
3.4 ***y = y+1***
3.5 Minden **AÉT**-beli élben módosítjuk ***x***-et

9

Poligon kitöltése



AÉT az ***y = 8*** pásztázó vonalon:



10

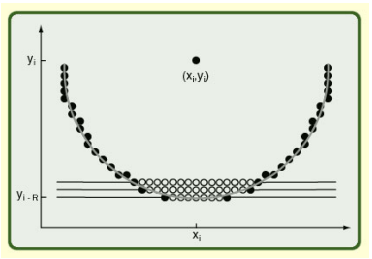
Poligon kitöltése

Megjegyzés:
Az x tengellyel párhuzamos alapú háromszögekre, trapézokra egyszerűsíthető az algoritmus, mert a pásztázó egyeneseknek a háromszögnek vagy a trapéznak legfeljebb *2* élével lehet metszéspontja (nem kell **ÉT**).

11

Kör, ellipszis kitöltése

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



12

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();
```

13

Háromszög kitöltése (OpenGL)

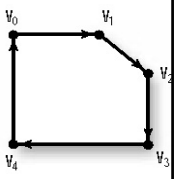
Több színnel (Gouraud-féle módon interpolálva)

```
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();
```

14

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);
```

15

Poligon (OpenGL)

3D-s poligonoknak két oldaluk van: első és hátsó 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;`

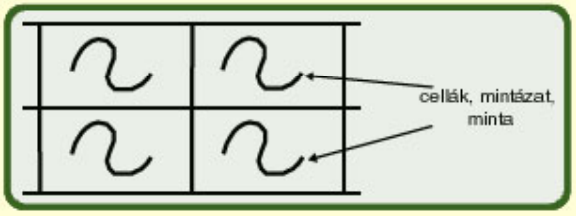
`mode:`

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

16

Kitöltés mintával

Általában: terület kitöltése szabályosan ismétlődő grafikus elemekkel

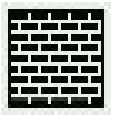


Képmátrixok (raszter) esetében a cella egy (kisméretű) mátrix

17

Kitöltés mintával

Példa:



Tégla minta

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

18

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)

19

Kitöltés mintával

Legyen:

minta *M* * *N* -es mátrix

minta [*0,0*] → képernyő [*0,0*]

ekkor

1. módszer: Pásztázás soronként (átlátszó)

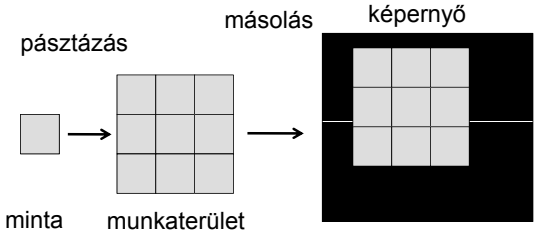
```
if(minta[x % M][y % N])
   WritePixel(x,y,érték);
```

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

20

Kitöltés mintával

2. módszer: Téglalap írás



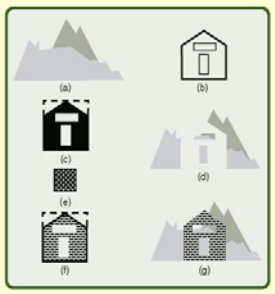
Csak akkor érdemes használni, ha a primitívet sokszor kell használni

Pl. karakterek megjelenítése

21

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

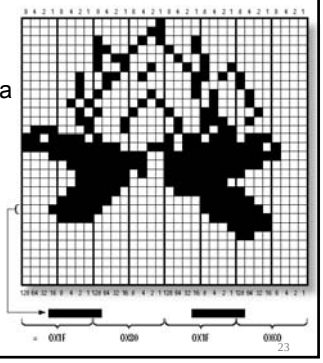
22

Kitöltés mintával (OpenGL)

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

Kitöltési minta beállítása

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



23

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
}
```

24

Kitöltés mintával (OpenGL)

Példa:

25

VASTAG PRIMITÍVEK RAJZOLÁSA

Képpontok ismétlése
Mozgó ecset
Területkitöltés
Közelítés vastag szakaszokkal

26

VASTAG PRIMITÍVEK RAJZOLÁSA

Több képpontnyi vastagságú vonalak

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

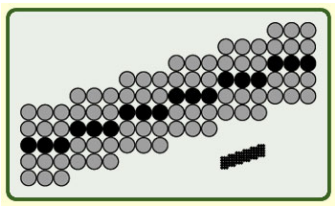
27

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



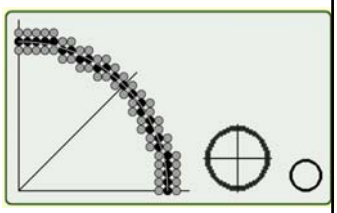
28

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



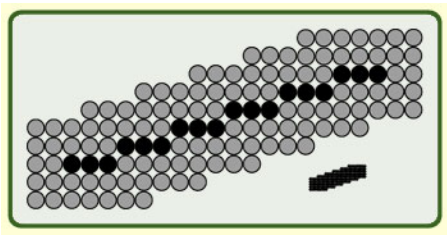
Jó módszer, ha nem túl vastag a vonal

29

VASTAG PRIMITÍVEK RAJZOLÁSA

2. Mozgó ecset

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



30

Mozgó ecset

Tulajdonságai:

hasonló 1-hez, de

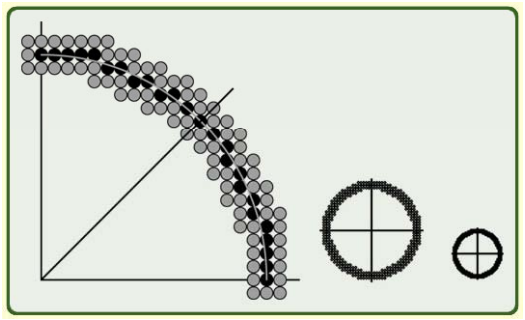
a) a végpontok „nagyobbak”

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

31

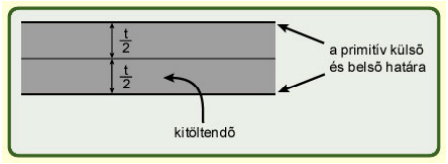
Mozgó ecset



32

VASTAG PRIMITÍVEK RAJZOLÁSA

3. Területkitöltés



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

33

Területkitöltés

Tulajdonságai:

a) ugyanolyan jó páros és páratlan vastagra

b) a vonal vastagsága nem függ a meredekségtől

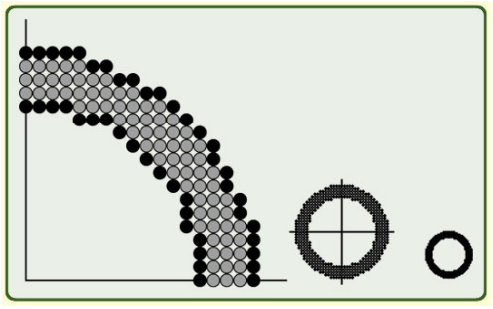
Kör esetén: külső és belső kör

Ellipszis esetén:

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

34

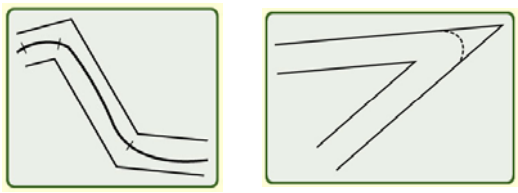
Területkitöltés



35

VASTAG PRIMITÍVEK RAJZOLÁSA

4. Közelítés vastag szakaszokkal



Szakaszonként lineáris approximáció

a) szép

b) vastag vonalakat símán kell illeszteni

36

Pont mérete (OpenGL)

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

Nem minden méretet támogatnak az implementációk:

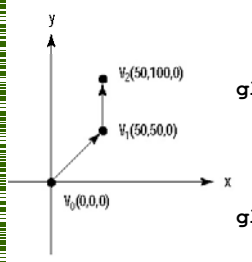
```
GLfloat sizes[2]; // méret tartomány
GLfloat step; // támogatott lépés

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

37

Szakaszok rajzolása (OpenGL)

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

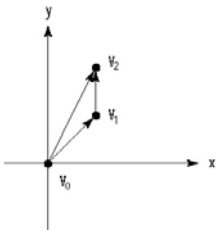



```
glBegin(GL_LINE_STRIP);
    glVertex3d(0,0,0);
    glVertex3d(50,50,0);
    glVertex3d(50,100,0);
glEnd();
```

38

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();
```

39

Szakaszok rajzolása (OpenGL)

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

ezek a szakaszok nincsenek összekötve

40

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);
```

41

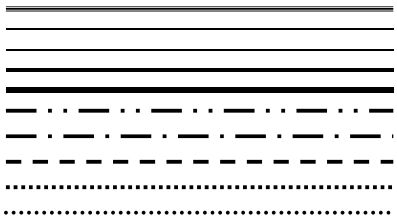
Szakaszok élsimítása (OpenGL)

A `GL_LINE_SMOOTH` argumentummal meghívott szakaszok élsimítását engedélyezni a `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.

42

VONAL STÍLUS



43

VONAL STÍLUS

Primitívek attribútumai:

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

Vonal stílus:

- folytonos
- szaggatott
- pontozott
- felhasználó által definiált

44

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)

45

VONAL STÍLUS

Hátránya:

A szaggatások távolsága függ a meredekségtől

Megoldás:

A távolságot számolva rajzolni a szakaszokat

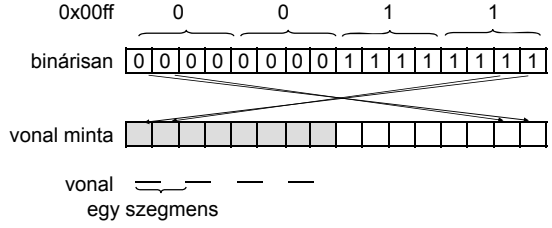
46

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.



47

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

48

Szakasz stílus (OpenGl)

Megoldandó feladat:

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

49

VÁGÁS

A vágásról általában

Pontok vágása

Vonalak, szakaszok vágása egyenletrendszer megoldásával

A COHEN - SUTHERLAND -féle vonal vágás

Parametrikus vonal vágó algoritmus

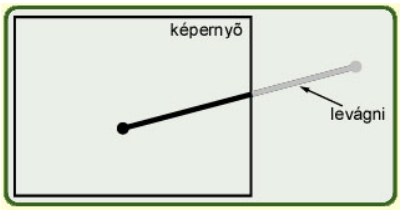
Körök és ellipszisek vágása

Poligonok vágása

50

VÁGÁS

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



51

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

52

VÁGÁS

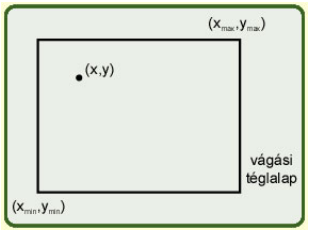
Pontok vágása:

(x,y) belül van, ha

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

és

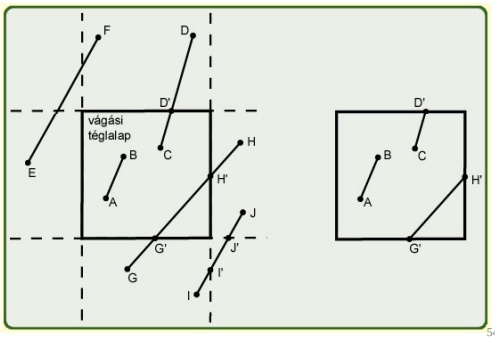
$y_{min} \leq y \leq y_{max}$



53

VÁGÁS

Szakaszok vágása egyenletrendszer megoldásával



54

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.

55

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)

56

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

57

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

58

COHEN - SUTHERLAND - féle szakasz vágás

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

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

59

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

60

COHEN - SUTHERLAND - féle szakasz vágás

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

1001	**1000**	**1010**
0001	**0000**	**0010**
0101	**0100**	**0110**

különben: kód

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)

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

61

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.

62

COHEN - SUTHERLAND - féle szakasz vágás

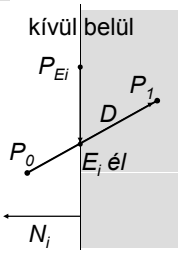
Interaktív módon is használható

Hatékony, mert gyakori, hogy sok vagy kevés szakasz van belül

A legáltalánosabban használt eljárás

63

Parametrikus szakasz vágó algoritmus



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

A metszéspontra (skalárszorzat):

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

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

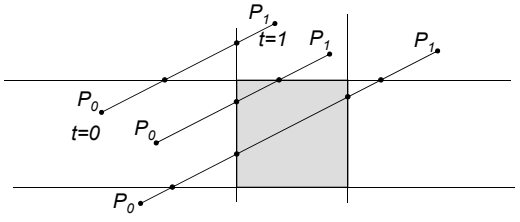
$$t = \frac{N_i\,(P_0 - P_{E\,i})}{-N_i\,D}$$

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

64

Parametrikus szakasz vágó algoritmus

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



65

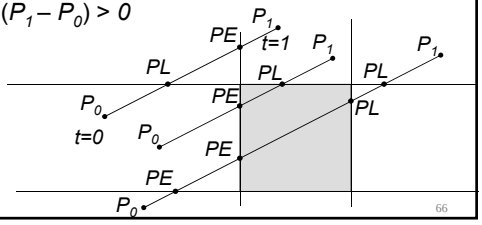
Parametrikus szakasz vágó algoritmus

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

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

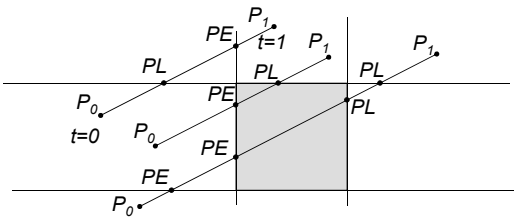


66

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

67

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

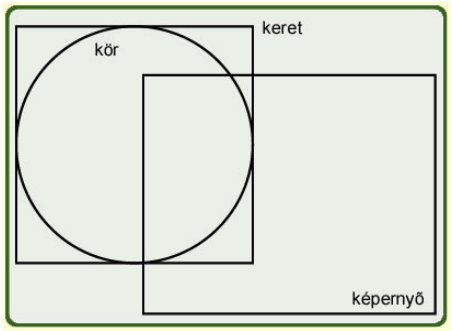
68

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 élekre
      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
```

69

Körök és ellipszisek vágása



70

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:

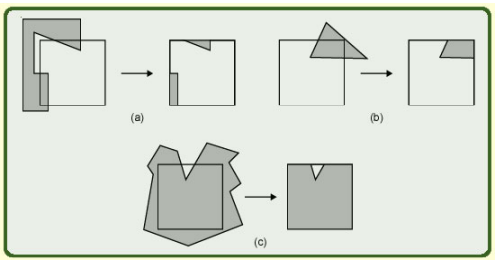
Környegyedekre (nyolcadokra) kiszámítjuk a kör és a téglalap élének metszéspontját, utána pásztázás
Ha a kör nem nagy, akkor pixelenként dönthetünk.

Ellipszis: hasonlóan.

71

Poligonok vágása

Sok eset lehet:



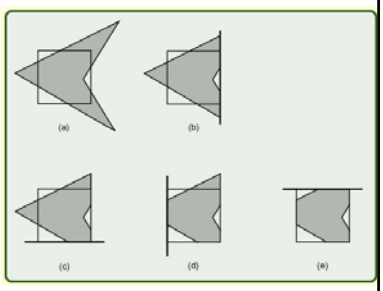
Általában minden éllel vágni kell

72

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

73

KARAKTEREK GENERÁLÁSA

ATTRIBÚTUMOK

KARAKTEREK DEFINIÁLÁSA

74

Attribútumok

Attribútumok:

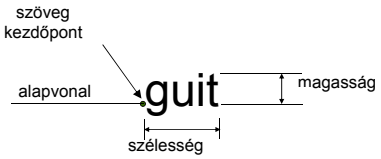
stílus (font): Times, Roman, Helvetica, Clarinda, ...

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

75

Attribútumok

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

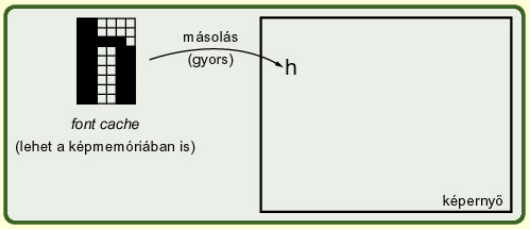


Betűk közötti távolság
Sorok közötti távolság

76

Karakterek definiálása

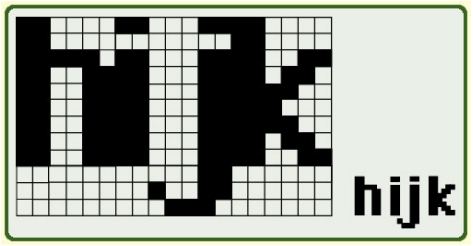
1. Bitmátrixokkal (bittérképekkel)
   az adott font minden egyes karakteréhez tartozik egy bitmátrix



77

Karakterek definiálása

A bitmátrixok előállíthatók pl. rajzoló programmal a betűk felnagyított képeiből



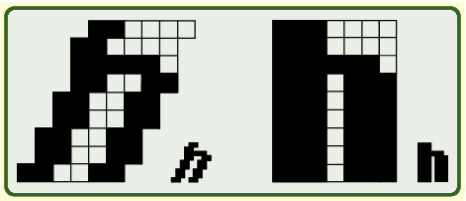
Különböző méretekhez különböző bitmátrixok

78

Karakterek definiálása

Tárolás: név
magasság
szélesség (lehet változó)
betűköz

Lehet trükközni:



79

Karakterek definiálása

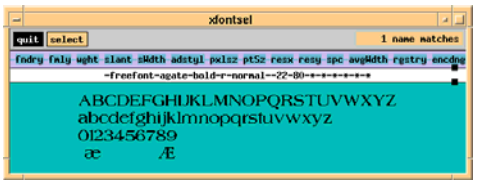
2. A betűket leíró határvonalakkal

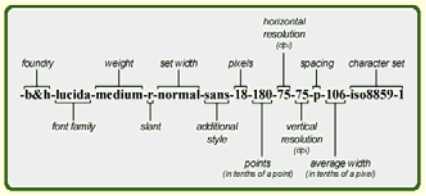
- poligonok
- ívek

- 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

80

Karakterek definiálása





81

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)

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

83

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)

84

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
```

85

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]);
  }
}
```

86

Szöveg - határvonal (OpenGL)

Határvonalával megadott (ún. stroke) karakter megjelenítése

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

`font`: pl.

```
    GLUT_STROKE_ROMAN,
    GLUT_STROKE_MONO_ROMAN
```

87

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]);
  }
}
```

88

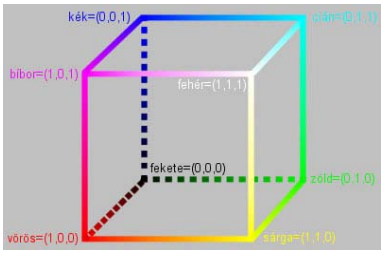
SZÍNMODELLEK

RGB, CMY, CMYK, HSV
OpenGL

89

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

Pl. színes képernyőnél

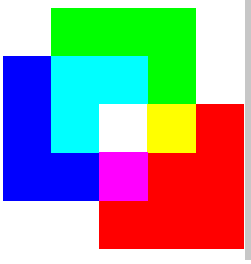


Additív komponensek:

Alkalmas súlyokkal vett összegük ad egy összetett színt

90

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

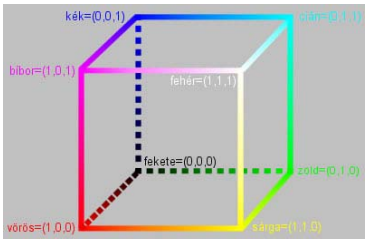


Vörös
Zöld
Kék
Cián
Sárga
Bíbor
Fehér

91

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

Pl. színes nyomtatóknál

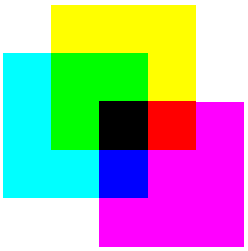


Szubtraktív komponensek:

Sokszor szűrőként használjuk, hogy kiszűrjék a fehérből a megfelelő színt

92

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



Cián
Bíbor
Sárga
Vörös
Zöld
Kék
Fekete

93

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

Konverzió:

RGB → CMY

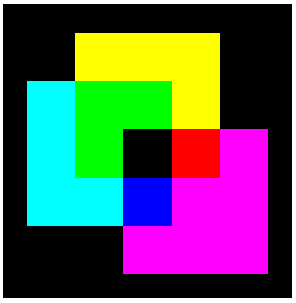
$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

CMY → RGB

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

94

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



Cián
Bíbor
Sárga
Vörös
Zöld
Kék
Fekete

95

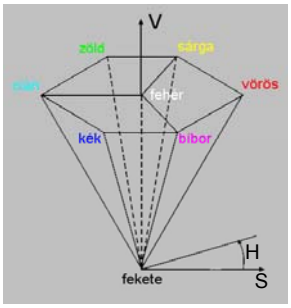
CMYK (Cyan, Magenta, Yellow, BlacK - K : fekete)

CMY → CMYK konverzió:

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

96

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



$0^0 \le H < 360^0$
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

Esztétikai alapú (ahogy a festők keverik a színeket)

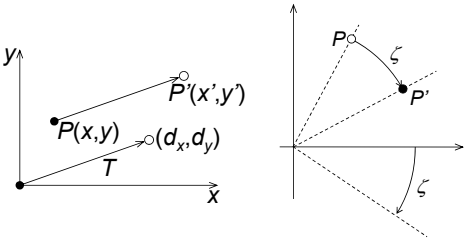
97

Feladat (OpenGL)

Szivárvány rajzolása

98

Geometriai transzformációk



99

Bevezetés - Transzformációk

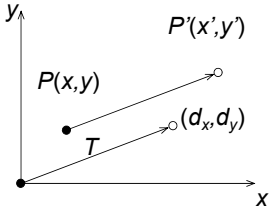
A Számítógépes Grafikában használatos 2- és 3- dimenziós transzformációk:

- eltolás
- nagyítás, kicsinyítés (skálázás)
- forgatás

100

Pont 2D eltolása

Hosszak és a szögek változatlanok



$x'=x+d_x$
$y'=y+d_y$

$P'=P+T$

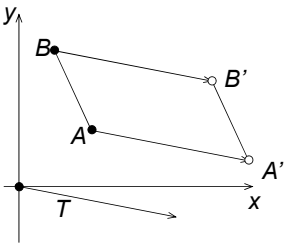
(oszlop-)vektorokkal:

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

101

Szakasz 2D eltolása

Elegendő az új végpontokat számolni

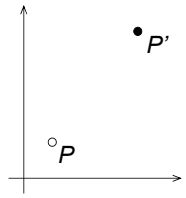


$A'=A+T$
$B'=B+T$

102

2D nagyítás/kicsinyítés

A szögek változatlanok
Szokták a kettőt együtt **SKÁLÁZÁSKÉNT** említeni
Origóból történő nagyítás



$x' = x \cdot s_x$
$y' = y \cdot s_y$
$P' = S \cdot P$

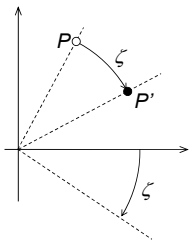
(oszlop-)vektorokkal:

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

103

2D forgatás

A hosszak és a szögek változatlanok
Origó körüli forgatás



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

$P' = R \cdot P$

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

104

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)

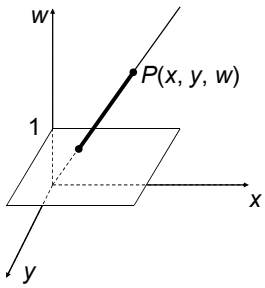
Egy ponthoz végtelen sok (x, y, w) tartozik.

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

(0, 0, 0) nem megengedett!

105

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

106