

Számítógépes Grafika

Hajder Levente

hajder@inf.elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2017/2018. II. félév

Inkrementális képszintézis

- Inkrementális elv
- A transzformációk szemszögéből végignéztük a grafikus szerelőszalagot
- Lényegében egy pont útját követtük végig, a transzformációkon át a képernyőig
- Most az inkrementális képszintézis szerelőszalagját vizsgáljuk tovább

Tartalom

- 1 Emlékeztető
- 2 Vágás
 - Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás
- 3 Raszterizálás
 - Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

Motiváció

Tartalom

- 1 Emlékeztető
- 2 Vágás
 - Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás
- 3 Raszterizálás
 - Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

Vágás

- A vágás során a nézeti csonkagúlán kívül geometriai elemeket szűrjük ki
- A nézeti csonkagúla határozza meg a színterünknek azt a részét, amely majd leképeződik a képernyőre (ld. múlt óra)
- Miért érdemes egyáltalán vágni?
 - Degenerált esetek kiszűrése (ld. pl. múlt óra középpontos vetítés)
 - Ne számoljunk feleslegesen (amit úgyse látunk, ne számoljuk sokat)

Tartalom

- 1 Emlékeztető
- 2 Vágás
 - Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás
- 3 Raszterizálás
 - Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

Vágás az ablakra

- Pont vágás: $(x, y) \in \mathbb{R}^2$. Akkor tartjuk meg a pontot, ha $(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$.
Megoldás: a határoló egyenesekkel vágás (ha tengelypárhuzamosak: csak összehasonlítás)
- Szakasz vágás: a szakasznak csak azokat a pontjait akarjuk megtartani, amik benne vannak $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ -ban.
Megoldás: az ablak négy élével, mint négy félsíkkal vágjuk a szakaszt.
- Polygon vágás: a poligonból egy új poligont akarunk csinálni, ami nem lóg ki az ablakból.
Megoldás: A poligon minden oldalát (mint szakaszt) vágjuk.

Pontok vágása

- Itt most pontokat vágunk egyenesre, síkra
- Azaz, azt akarjuk meghatározni, hogy a bemeneti pont az egyenes vagy sík normálisával megegyező irányban fekszik-e az egyenes/sík pontjaihoz képest ("előtte" van-e).
- Ami mögötte van, nem kell nekünk → ha rajta fekszik, azt is tartjuk még meg
- Síkra 3D-ben vágunk → a nézeti csongúla 6 sík "előtti" rész (6 féltér metszete)
- Egyenesre 2D-ben → a képernyőn a monitorra kerülő rész 4 egyenes "előtti" rész (4 félsík metszete)

Az egyenes normálvektoros egyenlete a síkban

- Az egyenes megadható egy $P(p_x, p_y)$ pontjával és egy, az egyenes irányára merőleges $\mathbf{n} = [n_x, n_y]^T \neq \mathbf{0}$ normálvektorral:
- Az egyenes pontjai azon $Q(x, y)$ pontok, amelyek kielégítik a

$$\langle X - P, \mathbf{n} \rangle = 0$$

$$(x - p_x)n_x + (y - p_y)n_y = 0$$

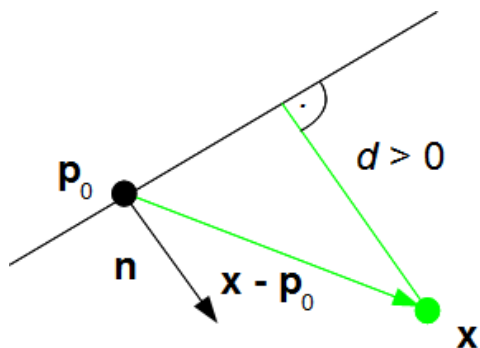
egyenletet.

- Az $\langle X' - P, \mathbf{n} \rangle < 0$ és $\langle X' - P, \mathbf{n} \rangle > 0$ az egyenesünk által meghatározott két félsík egyenlete.

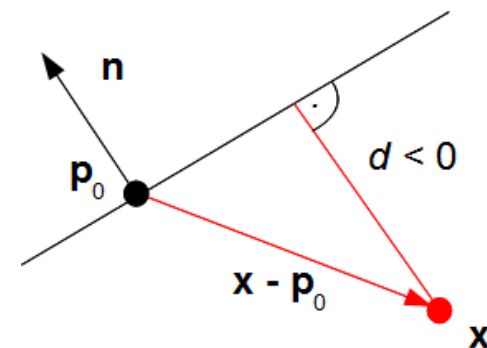
Pont vágása pont-normálvektoros egyenesre

- Feladat: adott \mathbf{p} pont és egy e egyenes (\mathbf{p}_0 pontjával és \mathbf{n} normálvektorával, $|\mathbf{n}| = 1$) a síkban. Vágjuk a pontot az e egyenesre!
- Megtartjuk, ha: $\langle \mathbf{p} - \mathbf{p}_0, \mathbf{n} \rangle \geq 0$
- Ilyenkor a skaláris szorzat eredménye az egyenes egy pontjából az adott pontba mutató vektor előjeles vetülete a normálisra $|\mathbf{n}| = 1 \rightarrow$ előjeles távolság az egyenestől

Pont-egyenes távolsága



Pont-egyenes távolsága



- Használjuk ki, hogy a vágásnál már homogén koordinátákban dolgozik a rendszer!
- Feladat: adott $\hat{\mathbf{p}}$ pont homogén koordinátás alakja és egy \mathbf{e} egyenes \mathbf{e} vonalkoordinátáival, úgy, hogy $e_1^2 + e_2^2 + e_3^2 = 1$
- Megtartjuk, ha: $\mathbf{e} \cdot \hat{\mathbf{p}} \geq 0$

- Pont-normálisal adott sík esetén megtartjuk a pontot, ha:

$$\langle \mathbf{p} - \mathbf{p}_0, \mathbf{n} \rangle \geq 0$$

- Sík-koordinátás (\mathbf{s}) megadás esetén (ha $s_1^2 + s_2^2 + s_3^2 = 1$) megtartjuk a pontot, ha:

$$\mathbf{s} \cdot \hat{\mathbf{p}} \geq 0$$

- A sík megadható egy $P(p_x, p_y, p_z)$ pontjával és a síkra merőleges $\mathbf{n} = [n_x, n_y, n_z]^T$ normálvektorával:

$$\langle X - P, \mathbf{n} \rangle = 0$$

- Félterek: $\langle X - P, \mathbf{n} \rangle < 0$, $\langle X - P, \mathbf{n} \rangle > 0$

- Emlékeztető
- Vágás
 - Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás
- Raszterizálás
 - Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

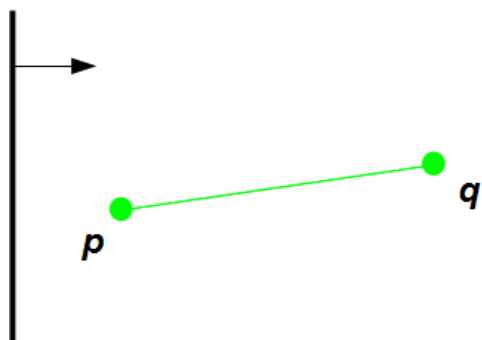
Emlékeztető	Vágás	Raszerizálás
	○○○○○○○○○○○○●○○○○○○○○○○○○○○○○	○○○○○○○○○○○○○○○○○○○○○○○○○○
Szakaszok vágása		
Szakasz vágása félsíkra		

- A szakasz p és q végpontjait vágjuk az e egyenes és normálisa által meghatározott félsíkra!
 - Az előbb látott módon végezhetjük a vágást!
 - Az eredmény viszont most bonyolultabb egy kicsit

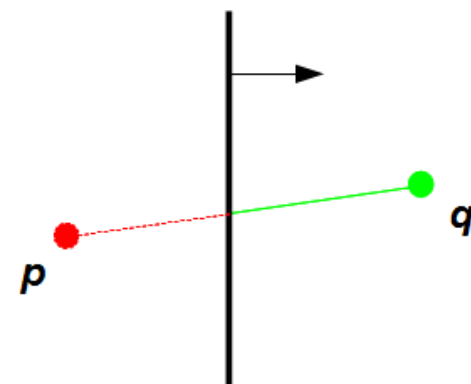
Emlékeztető	Vágás	Raszerizálás
	○○○○○○○○○○○○○○●○○○○○○○○○○○○○○	○○○○○○○○○○○○○○○○○○○○○○○○○○
Szakaszok vágása		
Szakasz vágása félsíkra		



Emlékeztető	Vágás	Raszerizálás
	○○○○○○○○○○○○○○●○○○○○○○○○○○○○○	○○○○○○○○○○○○○○○○○○○○○○○○○○
Szakaszok vágása		
Szakasz vágása félsíkra		



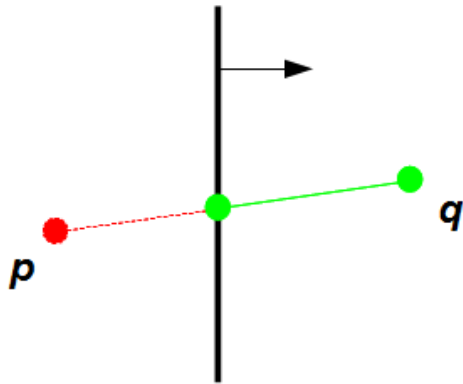
Emlékeztető	Vágás	Raszerizálás
	○○○○○○○○○○○○○○●○○○○○○○○○○○○○○	○○○○○○○○○○○○○○○○○○○○○○○○○○
Szakaszok vágása		
Szakasz vágása félsíkra		



Emlékeztető
Vágás
Raszterizálás

Szakaszok vágása

Szakasz vágása félsíkra - új "kezdőpont"!

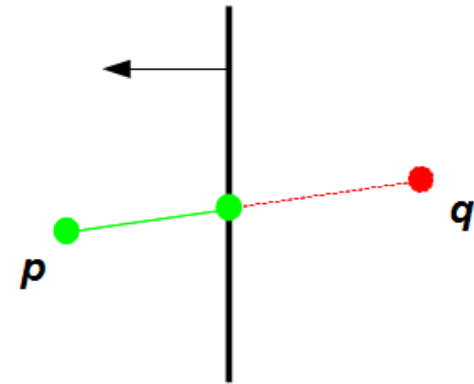


Navigation icons

Emlékeztető
Vágás
Raszterizálás

Szakaszok vágása

Szakasz vágása félsíkra

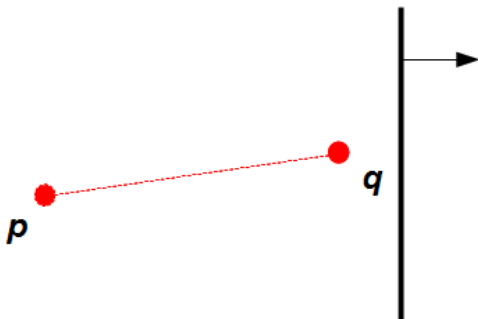


Navigation icons

Emlékeztető
Vágás
Raszterizálás

Szakaszok vágása

Szakasz vágása félsíkra



Navigation icons

Emlékeztető
Vágás
Raszterizálás

Szakaszvágás

Tartalom

- Emlékeztető
- Vágás
 - Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás**
 - Poligonvágás
- Raszterizálás
 - Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

Navigation icons

Emlékeztető
Vágás
Raszerizálás

Szakaszvágás

Szakasz vágása félsíkra

- Csak azt a speciális esetet nézzük, amikor a vágó egyenes párhuzamos valamelyik tengellyel.
- $(x_1, y_1) - (x_2, y_2)$ szakasz egyenlete:

$$x(t) = x_1 + t(x_2 - x_1)$$

$$y(t) = y_1 + t(y_2 - y_1)$$

- Vágó egyenes: pl. $x = x_{min}$
- Megoldás: $x_{min} = x_1 + t(x_2 - x_1) \Rightarrow t = \frac{x_{min} - x_1}{x_2 - x_1}$
- Metszéspont: $x = x_{min}, y = y_1 + \frac{x_{min} - x_1}{x_2 - x_1}(y_2 - y_1)$

Navigation icons

Emlékeztető
Vágás
Raszerizálás

Szakaszvágás

Szakasz vágása - nehézségek

- Eredmény kezelése: $t \leq 0$ vagy $t \geq 1 \rightarrow$ nincs is igazi vágás
- Kivételek: tengelyekkel párhuzamos szakaszok vágása
- Képernyőre vágásnál is rengeteg eset: négy félsík, két végpont, mindkettő eshet bárhova
- Feladat: Kell-e vágni? Melyik félsík(ok)ra kell vágni? Triviálisan bent/kint van-e a szakasz?
- Megoldás: *Cohen-Sutherland* vágás

Navigation icons

Emlékeztető
Vágás
Raszerizálás

Szakaszvágás

Cohen-Sutherland vágás

- A képernyő széleit reprezentáló egyenesekkel osszuk kilenc részre a síkot!
- Mindegyik síkrészhez rendeljünk egy 4 bitből álló bit-kódot: TBRL
- Számítsuk ki ezt a kódot a szakasz végpontjaira!
 - $T = 1$, ha a pont az képernyő fölött van, különben 0
 - $B = 1$, ha a pont az képernyő alatt van, különben 0
 - $R = 1$, ha a pont az képernyőtől jobbra van, különben 0
 - $L = 1$, ha a pont az képernyőtől balra van, különben 0

Navigation icons

Emlékeztető
Vágás
Raszerizálás

Szakaszvágás

Cohen-Sutherland vágás

code = Top, Bottom, Right, Left

1001	1000	1010
0001	0000	0010
0101	0100	0110

Navigation icons

Emlékeztető
Vágás
Raszerizálás

Szakaszvágás

Cohen-Sutherland vágás

- $code = (y > y_{max}, y < y_{min}, x > x_{max}, x < x_{min})$
- Vizsgáljuk a két végpont bit-kódjait, $code_a$ -t és $code_b$ -t!
 - Ha $code_a \text{ OR } code_b == 0$: a szakasz egésze (mindkét végpontja) az ablakban van \Rightarrow megtartjuk.
 - Ha $code_a \text{ AND } code_b != 0$: a szakasz az ablak valamelyik oldalán (fölül, alul, jobbra, balra), kívül van \Rightarrow eldobjuk.
 - Különben vágni kell: az 1-es bitek adják meg, hogy melyik félsíkra, utána újra számoljuk a bit-kódokat, és az új szakasszal újrakezdjük az algoritmust.

Emlékeztető
Vágás
Raszerizálás

Szakaszvágás

Cohen-Sutherland vágás

$code_a \text{ OR } code_b == 0$:

1001	1000	1010
0001	0000	0010
0101	0100	0110

Emlékeztető
Vágás
Raszerizálás

Szakaszvágás

Cohen-Sutherland vágás

$code_a \text{ AND } code_b != 0$:

1001	1000	1010
0001	0000	0010
0101	0100	0110

Emlékeztető
Vágás
Raszerizálás

Szakaszvágás

Cohen-Sutherland vágás

Különben:

1001	1000	1010
0001	0000	0010
0101	0100	0110

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
- Pont és szakaszvágás
- Szakaszok vágása
- Szakaszvágás
- **Poligonvágás**

3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció

Poligonvágás

Feladat

Adott egy vágandó és egy vágó poligon. Keressük azt a poligont, amit a vágandóból kapunk, ha csak a vágó poligonba tartozó részeit vesszük. Röviden: keressük a kettő közös részét.

Sutherland-Hodgman poligonvágás

- Legyen p tömb a bemeneti-, q a kimeneti-poligon csúcsainak tömbje!
- Legyen n a csúcsok száma, és $p[0] = p[n]$!
- Vágás egyenesre:
 - Ha $p[i]$ bent van az alablakban, és $p[i + 1]$ is:
⇒ adjuk hozzá $p[i]$ -t a q -hoz.
 - Ha $p[i]$ bent van az alablakban, de $p[i + 1]$ nincs:
⇒ adjuk hozzá $p[i]$ -t a q -hoz, számítsuk ki $p[i], p[i + 1]$ szakasz metszéspontját a vágó egyenessel, majd ezt is adjuk hozzá q -hoz.
 - Ha $p[i]$ nincs bent az alablakban, de $p[i + 1]$ benne van:
⇒ számítsuk ki $p[i], p[i + 1]$ szakasz metszéspontját a vágó egyenessel, és ezt adjuk hozzá q -hoz.
 - Ha $p[i]$ nincs bent az alablakban, és $p[i + 1]$ sincs:
⇒ *SKIP*

Sutherland-Hodgman poligonvágás - Pszeudó-kód

```
PolygonClip(in p[n], out q[m], in line) {
    m = 0;
    for( i=0; i < n; i++) {
        if (IsInside(p[i])) {
            q[m++] = p[i];
            if (!IsInside(p[i+1]))
                q[m++] = Intersect(p[i], p[i+1], line);
        } else {
            if (IsInside(p[i+1]))
                q[m++] = Intersect(p[i], p[i+1], line);
        }
    }
}
```

Emlékeztető

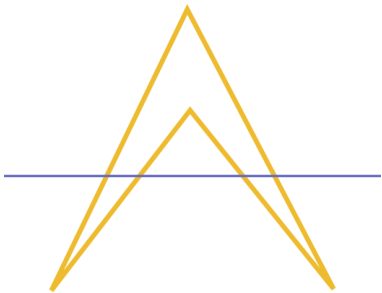
Vágás

Rasztelizálás

Poligonvágás

Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.



Emlékeztető

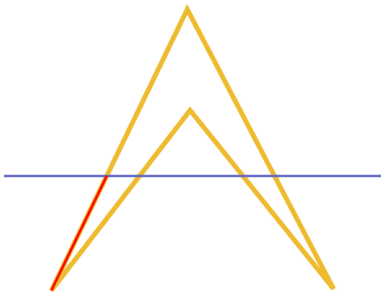
Vágás

Rasztelizálás

Poligonvágás

Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.



Emlékeztető

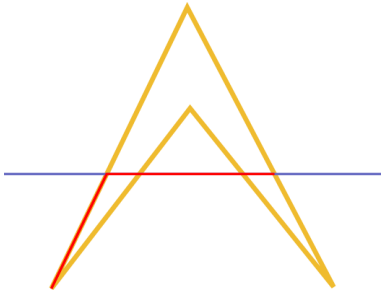
Vágás

Rasztelizálás

Poligonvágás

Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.



Emlékeztető

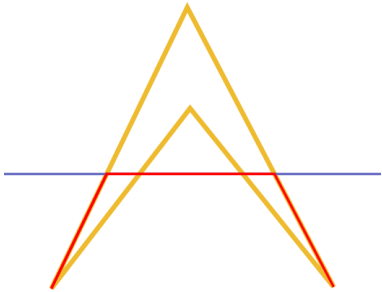
Vágás

Rasztelizálás

Poligonvágás

Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.

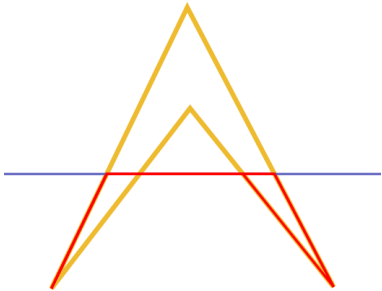


EmlékeztetőVágásRasztelizálás

Poligonvágás

Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.

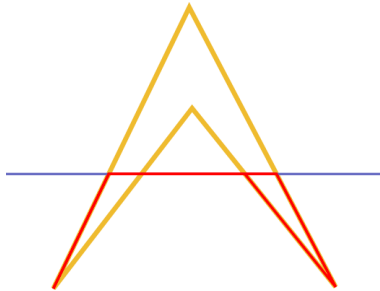


EmlékeztetőVágásRasztelizálás

Poligonvágás

Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.

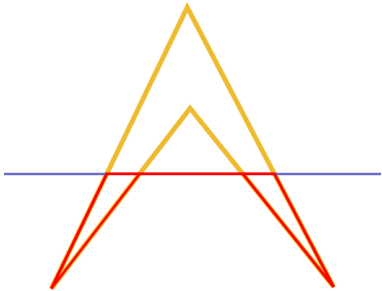


EmlékeztetőVágásRasztelizálás

Poligonvágás

Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.

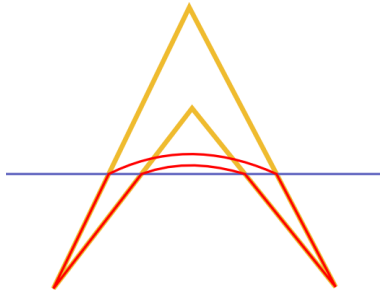


EmlékeztetőVágásRasztelizálás

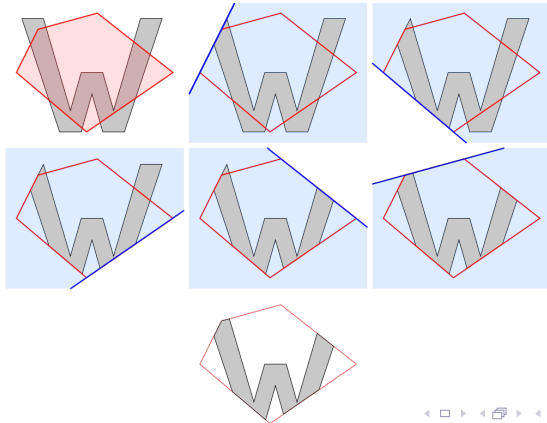
Poligonvágás

Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.



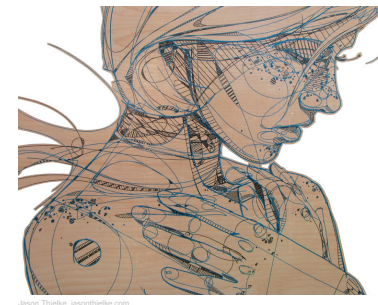
A vágópoligon minden élére vágunk, az előző vágás eredmény éllistáját felhasználva kiindulásként



- Projektív transzformáció előtt → a vágósíkok világtérbeli megadásával (mik az egyenletei?)
- NPKR-ben (projektív trafó után, homogén osztás előtt) → a homogén koordináták "ellenére" ez a legegyszerűbb!
- Transzformált 3D térben (homogén osztás után) → vetítési síkon átmenő objektumok...

- 1 Emlékeztető
- 2 Vágás
 - Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás
- 3 Raszterizálás
 - Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

- Egyik leggyakrabban primitív
- Fontos, hogy szépen tudjuk rajzolni
- Mégjobb, ha gyorsan is

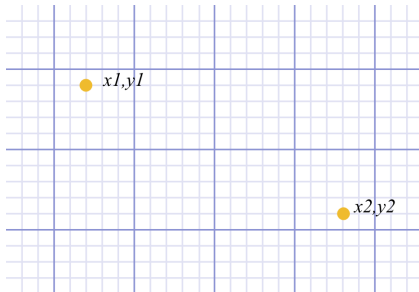


EmlékeztetőVágásRaszterizálás

Szakasz raszterizálása

Hogyan rajzolunk szakaszt?

- Adott a két végpont.

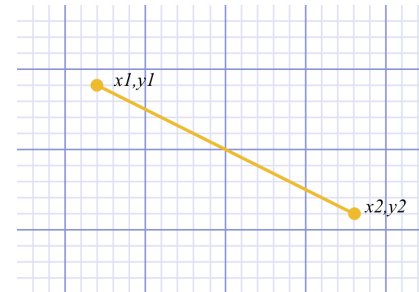


EmlékeztetőVágásRaszterizálás

Szakasz raszterizálása

Hogyan rajzolunk szakaszt?

- Adott a két végpont.
- Hogyan tudjuk összekötni őket?

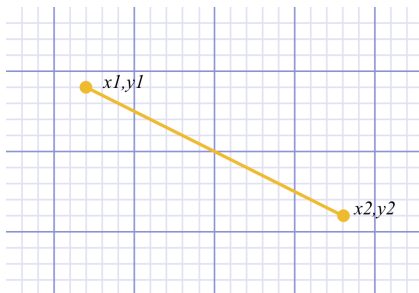


EmlékeztetőVágásRaszterizálás

Szakasz raszterizálása

Hogyan rajzolunk szakaszt?

- Adott a két végpont.
- Hogyan tudjuk összekötni őket?
- Csak miniatűr téglalapjaink vannak (amiket pixelnek nevezünk).

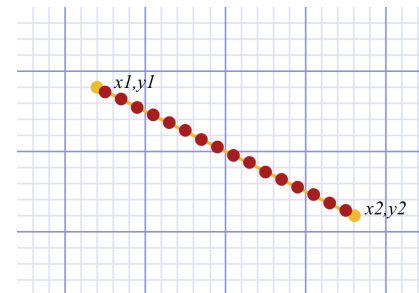


EmlékeztetőVágásRaszterizálás

Szakasz raszterizálása

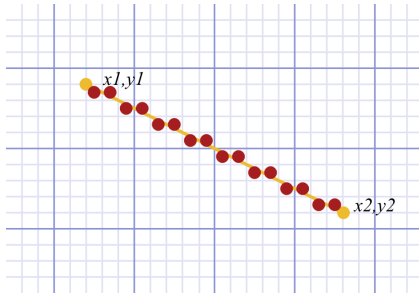
Hogyan rajzolunk szakaszt?

- Adott a két végpont.
- Hogyan tudjuk összekötni őket?
- Csak miniatűr téglalapjaink vannak (amiket pixelnek nevezünk).



Hogyan rajzolunk szakaszt?

- Adott a két végpont.
- Hogyan tudjuk összekötni őket?
- Csak miniatűr téglalapjaink vannak (amiket pixelnek nevezünk).



Szakasz megadása (sokadszor)

- Végpontok: $(x_1, y_1), (x_2, y_2)$
- Tfh. nem függőleges: $x_1 \neq x_2$.
- Szakasz egyenlete:

$$y = mx + b, x \in [x_1, x_2]$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

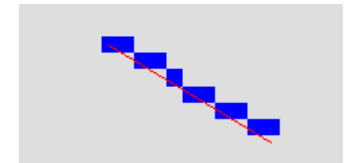
$$b = y_1 - mx_1$$

Naív algoritmus

```
def line1(x1, y1, x2, y2, draw):
    m = float(y2 - y1) / (x2 - x1)
    x = x1
    y = float(y1)
    while x <= x2:
        draw.point((x, y))
        x += 1
        y += m
```

Naív algoritmus

- $m = \text{float}(y_2 - y_1) / (x_2 - x_1)$
nem pontos

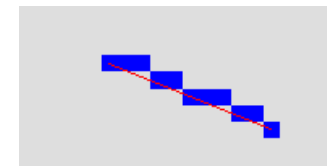


Javítsuk az algoritmust 1.

```
def line(x1,y1,x2,y2, draw):
    m = float(y2-y1)/(x2-x1)
    x = x1
    y = y1
    e = 0.0
    while x<=x2:
        draw.point((x,y))
        x += 1
        e += m
        if e >= 0.5:
            y += 1
            e -= 1.0
```

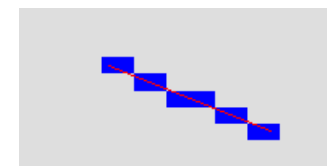
Javítsuk az algoritmust 1.

Naív:



- Jó: Mindig "eltalálja" a végpontokat
- Jó: Egyenletesebben lép az y irányban.
- Rossz: Még mindig használunk `float`-okat

Javítás:



Javítsuk az algoritmust 2.

```
def line3(x1,y1,x2,y2, draw):
    x = x1
    y = y1
    e = -0.5
    while x<=x2:
        draw.point((x,y))
        x += 1
        e += float(y2-y1)/(x2-x1)
        if e >= 0.0:
            y += 1
            e -= 1.0
```

Javítsuk az algoritmust 3.

```
def line4(x1,y1,x2,y2, draw):
    x = x1
    y = y1
    e = -0.5*(x2-x1) ←
    while x<=x2:
        draw.point((x,y))
        x += 1
        e += y2-y1 ←
        if e >= 0.0:
            y += 1
            e -= (x2-x1) ←
```


Emlékeztető
Vágás
Raszerizálás

Szakasz raszerizálása

Javítsuk az algoritmust 4.

```
def line5(x1,y1,x2,y2, draw):
    x = x1
    y = y1
    e = -(x2-x1) ←
    while x<=x2:
        draw.point((x,y))
        x += 1
        e += 2*(y2-y1) ←
        if e >= 0:
            y += 1
            e -= 2*(x2-x1) ←
```

Emlékeztető
Vágás
Raszerizálás

Szakasz raszerizálása

Javítsuk az algoritmust 4.

- Ez a *Bresenham* algoritmus (egyik speciális esete)
- Külön gyűjtjük a hibát e -ben
- Nem használunk `float`-okat
- Tetszőleges meredekségű szakaszokra általánosítható.

Emlékeztető
Vágás
Raszerizálás

Szakasz raszerizálása

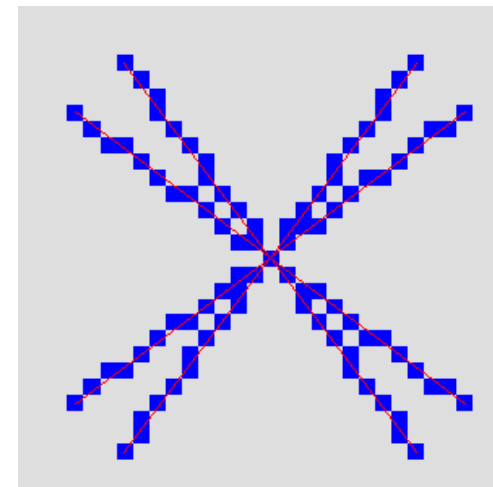
Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, mindegyik külön eset.
- (Előzők végig: jobbra-le)
- El kell döntenünk, hogy $|x_2 - x_1|$ vagy $|y_2 - y_1|$ a nagyobb (merre meredekebb a szakasz).
- Ha $|y_2 - y_1|$ a nagyobb, cseréljük fel $x_i \leftrightarrow y_i$, és rajzolásnál is fordítva használjuk!
- Ha $x_1 > x_2$, akkor csere: $x_1 \leftrightarrow x_2$, $y_1 \leftrightarrow y_2$.
- Az e hibatagot $|y_2 - y_1|$ -nal növeljük minden lépésben
- y -nal $y_2 - y_1$ előjele szerint haladunk.

Emlékeztető
Vágás
Raszerizálás

Szakasz raszerizálása

Bresenham algoritmus



Emlékeztető
Vágás
Raszterizálás

Szakasz raszterizálása

Teljes *Bresenham* algoritmust 1.

```
def Bresenham(x1,y1,x2,y2, draw):
    steep = abs(y2-y1)>abs(x2-x1)
    if steep:
        x1, y1 = y1, x1
        x2, y2 = y2, x2
    if x1>x2:
        x1, x2 = x2, x1
        y1, y2 = y2, y1
    Dy = abs(y2-y1)
    if y1<y2:
        Sy = 1
    else:
        Sy = -1
```

Navigation icons

Emlékeztető
Vágás
Raszterizálás

Szakasz raszterizálása

Teljes *Bresenham* algoritmust 2.

```
x = x1
y = y1
e = -(x2-x1)
while x<=x2:
    if steep:
        draw.point((y,x))
    else:
        draw.point((x,y))
    x += 1
    e += 2*Dy
    if e >= 0:
        y += Sy
        e -= 2*(x2-x1)
```

Navigation icons

Emlékeztető
Vágás
Raszterizálás

Háromszög raszterizálása

Tartalom

- 1 Emlékeztető
- 2 Vágás
 - Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás
- 3 Raszterizálás
 - Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

Navigation icons

Emlékeztető
Vágás
Raszterizálás

Háromszög raszterizálása

Háromszög raszterizáció

- A háromszög oldalait tudjuk vágni - most töltsük ki a belsejét!
- Ha egy meghatározott bejárési irányban adtuk meg az összes háromszög csúcsát, tudunk félsíkokat adni (tudjuk irányítani az éleket) \rightarrow u.i. ha (t_x, t_y) az irányvektora az oldalnak, akkor $(-t_y, t_x)$ egy normális lesz
- Minden pixelre menjünk végig a képernyőn és nézzük meg, hogy a háromszög oldalai által meghatározott síkok jó oldalán van-e!

Navigation icons

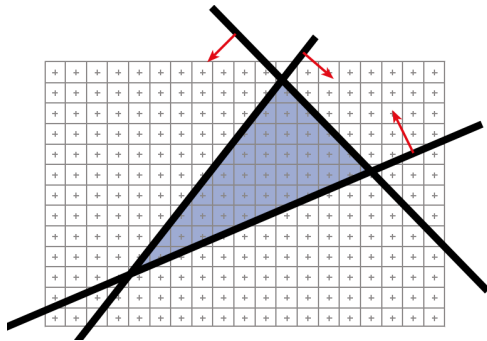
Emlékeztető

Vágás

Raszterizálás

Háromszög raszterizálása

Háromszög raszterizáció



Navigation icons: back, forward, search, etc.

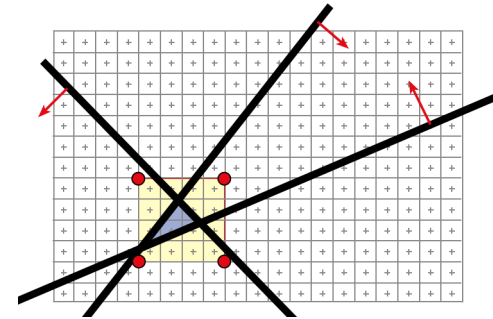
Emlékeztető

Vágás

Raszterizálás

Háromszög raszterizálása

Háromszög raszterizáció - okosabban



Navigation icons: back, forward, search, etc.

Emlékeztető

Vágás

Raszterizálás

Háromszög raszterizálása

Háromszög raszterizáció

- Lehetne még okosabban is csinálni, de: gyakorlatban ez a brute-force megközelítés nagyon jól alkalmazható!

Navigation icons: back, forward, search, etc.

Emlékeztető

Vágás

Raszterizálás

Háromszög raszterizálása

Háromszög raszterizáció

- Nem egy rögzített értékkel akarunk kitölteni, a hanem a csúcspokban adott értékeket akarjuk interpolálni.
- Felhasználásai: szín (Gouraud-árnyalás), textúra koordináták, normálvektorok
- Legyen a felület egy pontja $p = \alpha p_1 + \beta p_2 + \gamma p_3$, az α, β, γ baricentrikus koordinátákkal adott.
- Ekkor bármilyen más értéket is végig tudunk interpolálni ugyan így:

$$c = \alpha c_1 + \beta c_2 + \gamma c_3$$

- Ez az úgy nevezett *Gouraud interpoláció* (nem véletlenül)

Navigation icons: back, forward, search, etc.

Flood-fill – szomszédságok