

## Számítógépes Grafika

Valasek Gábor

valasek@inf.elte.hu

Eötvös Loránd Tudományegyetem  
Informatikai Kar

2011/2012. œszi félév

# Tartalom

## 1 Emlékeztető

## 2 Vágás

- Motiváció
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

## 3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció

# Inkrementális képszintézis

- Inkrementális elv

## Inkrementális képszintézis

- Inkrementális elv
  - A transzformációk szemszögéből végignéztük a grafikus szerelőszalagot

# Inkrementális képszintézis

- Inkrementális elv
- A transzformációk szemszögéből végignéztük a grafikus szerelőszalagot
- Lényegében egy pont útját követtük végig, a transzformációkon át a képernyőig

# 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
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

## 3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció



# Vágás

- A képernyőn kívüli dolgok azonosítása(3D-ben a nézeti csonkagúlán kívül geometriai elemeket kiszűrése, 2D-ben az ablakkeretre)



# Vágás

- A képernyőn kívüli dolgok azonosítása(3D-ben a nézeti csonkagúlán kívül geometriai elemeket kiszűrése, 2D-ben az ablakkeretre)
- A nézeti csonkagúla határozza meg a színtérünknek azt a részét, amely majd leképeződik a képernyőre (ld. múlt óra)



# Vágás

- A képernyőn kívüli dolgok azonosítása(3D-ben a nézeti csonkagúlán kívül geometriai elemeket kiszűrése, 2D-ben az ablakkeretre)
- A nézeti csonkagúla határozza meg a színtérünknek azt a részét, amely majd leképeződik a képernyőre (ld. múlt óra)
- Miért érdemes egyáltalán vágni?



# Vágás

- A képernyőn kívüli dolgok azonosítása(3D-ben a nézeti csonkagúlán kívül geometriai elemeket kiszűrése, 2D-ben az ablakkeretre)
- A nézeti csonkagúla határozza meg a színtérünknek azt a részét, amely majd leképeződik a képernyőre (ld. múlt óra)
- Miért érdemes egyáltalán vagni?
  - Degenerált esetek kiszűrése (ld. pl. múlt óra középpontos vetítés)



# Vágás

- A képernyőn kívüli dolgok azonosítása(3D-ben a nézeti csonkagúlán kívül geometriai elemeket kiszűrése, 2D-ben az ablakkeretre)
- A nézeti csonkagúla határozza meg a színterünknek azt a részét, amely majd leképeződik a képernyőre (ld. múlt óra)
- Miért érdemes egyáltalán vagni?
  - Degenerált esetek kiszűrése (ld. pl. múlt óra középpontos vetítés)
  - Ne számoljunk feleslegesen (amit úgyse látunk, ne számoljuk sokat)

# Vágás az ablakra

- Pont vágás:  $(x, y) \in \mathbb{R}^2$ , vágjuk a tengelyekkel párhuzamos ablakra (dobjuk el, ha ablakon kívül van). Akkor tartjuk meg a pontot, ha  $(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$ .

# Vágás az ablakra

- Pont vágás:  $(x, y) \in \mathbb{R}^2$ , vágjuk a tengelyekkel párhuzamos ablakra (dobjuk el, ha ablakon kívül van). Akkor tartjuk meg a pontot, ha  $(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$ . Megoldás (nem csak tengelypárhuzamos ablakra): a képernyő oldalainak félsíkjaival vágás
- Szakasz vágás: a szakasznak csak azokat a pontjai akarjuk megtartani, amik benne vannak  $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ -ban.

# Vágás az ablakra

- Pont vágás:  $(x, y) \in \mathbb{R}^2$ , vágjuk a tengelyekkel párhuzamos ablakra (dobjuk el, ha ablakon kívül van). Akkor tartjuk meg a pontot, ha  $(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$ . Megoldás (nem csak tengelypárhuzamos ablakra): a képernyő oldalainak félsíkjaival vágás
- Szakasz vágás: a szakasznak csak azokat a pontjai 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.
- Poligon vágás: a poligonból egy új poligont akarunk csinálni, ami nem lóg ki az ablakból.

# Vágás az ablakra

- Pont vágás:  $(x, y) \in \mathbb{R}^2$ , vágjuk a tengelyekkel párhuzamos ablakra (dobjuk el, ha ablakon kívül van). Akkor tartjuk meg a pontot, ha  $(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$ . Megoldás (nem csak tengelypárhuzamos ablakra): a képernyő oldalainak félsíkjaival vágás
- Szakasz vágás: a szakasznak csak azokat a pontjai 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.
- Poligon vágás: a poligonból egy új poligont akarunk csinálni, ami nem lóg ki az ablakból.  
Megoldás: A poligon minden oldalát (mint szakaszt) vágjuk.

# Vágás félsíkra

- Egy síkbeli elemről el kell dönteneni, hogy egy adott félsíkon kívül van-e

# Vágás félsíkra

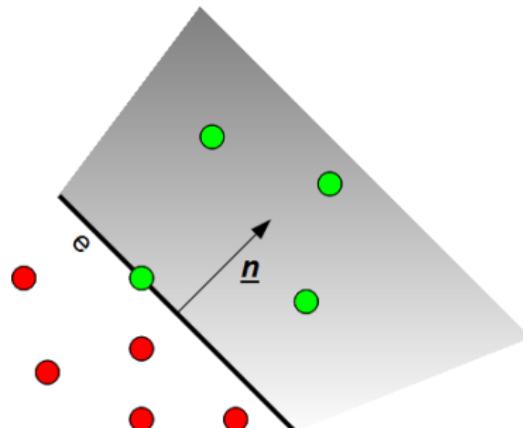
- Egy síkbeli elemről el kell döntenи, hogy egy adott félsíkon kívül van-e
  - Félsík megadása: egy határoló egyenesével és egy vektorral, ami a félsík belseje felé mutat (az egyenesre eső pontokat is megtartjuk!) - 2.EA

# Vágás félsíkra

- Egy síkbeli elemről el kell dönteneni, hogy egy adott félsíkon kívül van-e
- Félsík megadása: egy határoló egyenesével és egy vektorral, ami a félsík belseje felé mutat (az egyenesre eső pontokat is megtartjuk!) - 2.EA
- Vágni kell egy elemet, ha nincs a félsíkban

## Vágás félsíkra

- Egy síkbeli elemről el kell döntenи, hogy egy adott félsíkon kívül van-e
  - Félsík megadása: egy határoló egyenesével és egy vektorral, ami a félsík belseje felé mutat (az egyenesre eső pontokat is megtartjuk!) - 2.EA
  - Vágni kell egy elemet, ha nincs a félsíkban



# Vágás félterre

- Egy térbeli elemről el kell döntenи, hogy egy adott féltéren kívül van-e

# Vágás félterre

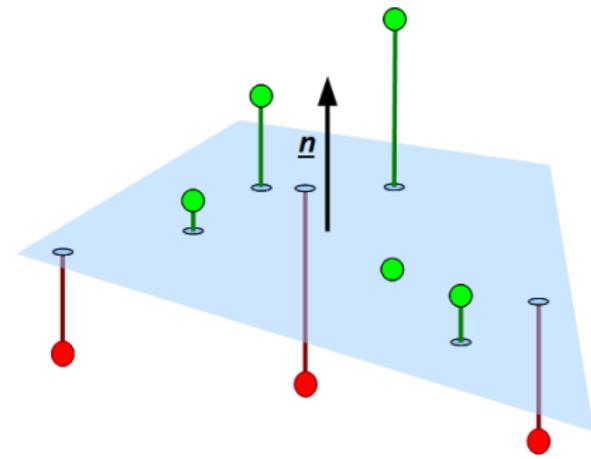
- Egy tébeli elemről el kell dönten, hogy egy adott féltéren kívül van-e
  - Féltér megadása: határoló síkjával és befelé mutató vektorral - 2.EA

# Vágás félfelülete

- Egy térbeli elemről el kell dönteneni, hogy egy adott féltéren kívül van-e
- Féltér megadása: határoló síkjával és befelé mutató vektorral - 2.EA
- Vágni kell egy elemet, ha nincs a féltérben

# Vágás félterre

- Egy térbeli elemről el kell döntenи, hogy egy adott féltéren kívül van-e
  - Féltér megadása: határoló síkjával és befelé mutató vektorral - 2.EA
  - Vágni kell egy elemet, ha nincs a féltérben





## Pontok vágása

# Tartalom

### 1 Emlékeztető

### 2 Vágás

- Motiváció
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

### 3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció



## Pontok vágása

## Pontok vágása

- Itt most pontokat vágunk félsíkra (egyenesre), féltérre (síkra)



## Pontok vágása

## Pontok vágása

- Itt most pontokat vágunk félsíkra (egyenesre), féltérre (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, az egyenes és a normális által meghatározott *fél sík része*-e).



## Pontok vágása

## Pontok vágása

- Itt most pontokat vágunk félsíkra (egyenesre), féltérre (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, az egyenes és a normális által meghatározott *fél sík része-e*).
- Ami mögötte van, nem kell nekünk → de ha rajta fekszik, azt is tartsuk még meg



## Pontok vágása

## Pontok vágása

- Itt most pontokat vágunk félsíkra (egyenesre), féltérre (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, az egyenes és a normális által meghatározott **fél sík része**-e).
- Ami mögötte van, nem kell nekünk → de ha rajta fekszik, azt is tartsuk még meg
- Síkra 3D-ben vágunk → a nézeti csonkgúla 6 síkja "előtti" rész (a 6 féltér metszete) ami a tér képernyőre képeződő részét tartalmazza



## Pontok vágása

## Pontok vágása

- Itt most pontokat vágunk félsíkra (egyenesre), féltérre (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, az egyenes és a normális által meghatározott **fél sík része**-e).
- Ami mögötte van, nem kell nekünk → de ha rajta fekszik, azt is tartsuk még meg
- Síkra 3D-ben vágunk → a nézeti csonkgúla 6 síkja "előtti" rész (a 6 féltér metszete) ami a tér képernyőre képeződő részét tartalmazza
- Egyenesre 2D-ben → a képernyőn a monitorra kerülő rész 4 egyenes "előtti" része (4 félsík metszete)



## Pontok vágása

## Az egyenes normálvektoros egyenlete a síkban

- Az egyenes megadható egy  $\mathbf{p} = (p_x, p_y)$  pontjával és egy, az egyenes irányára merőleges  $\mathbf{n} = (n_x, n_y) \neq \mathbf{0}$  normálvektorral:



## Az egyenes normálvektoros egyenlete a síkban

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

$$\langle \mathbf{q} - \mathbf{p}, \mathbf{n} \rangle = 0$$



## Az egyenes normálvektoros egyenlete a síkban

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

$$\langle \mathbf{q} - \mathbf{p}, \mathbf{n} \rangle = 0$$
$$(x - p_x)n_x + (y - p_y)n_y = 0$$

egyenletet.



### Az egyenes normálvektoros egyenlete a síkban

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

$$\langle \mathbf{q} - \mathbf{p}, \mathbf{n} \rangle = 0$$

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

egyenletet.

- Az  $\langle \mathbf{q}' - \mathbf{p}, \mathbf{n} \rangle < 0$  és  $\langle \mathbf{q}' - \mathbf{p}, \mathbf{n} \rangle > 0$  az egyenesünk által adott két félsíkot határozza meg.



## Pontok vágása

## Pont vágása pont-normálvektoros félsíkra

- 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  $e$  egyenes és  $\mathbf{n}$  normális által meghatározott félsíkra!



## Pontok vágása

## Pont vágása pont-normálvektoros félsíkra

- 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  $e$  egyenes és  $\mathbf{n}$  normális által meghatározott félsíkra!
- Megtartjuk, ha:  $\langle \mathbf{p} - \mathbf{p}_0, \mathbf{n} \rangle \geq 0$



## Pontok vágása

## Pont vágása pont-normálvektoros félsíkra

- 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  $e$  egyenes és  $\mathbf{n}$  normális által meghatározott félsíkra!
- 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 merüleges vetülete a normálisra

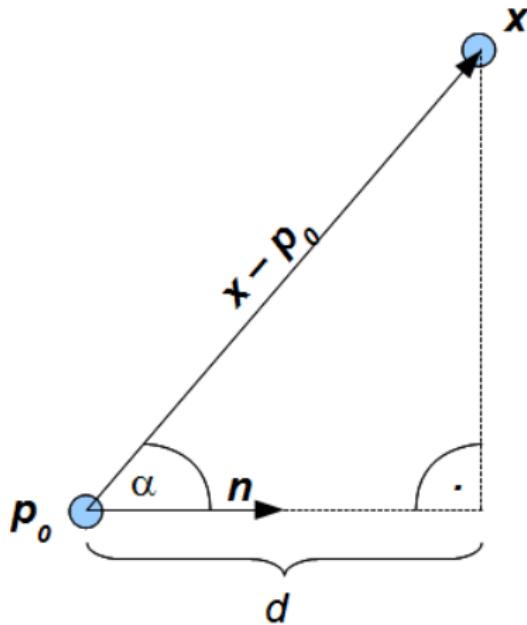


## Pontok vágása

## Pont vágása pont-normálvektoros félsíkra

- 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  $e$  egyenes és  $\mathbf{n}$  normális által meghatározott félsíkra!
- 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 merüleges vetülete a normálisra  $|\mathbf{n}| = 1 \rightarrow$  előjeles távolság az egyenestől, ha a normális irányában van a pont, akkor pozitív, ha a normálissal ellentétes irányban negatív (miért?)

# Előjeles vetület

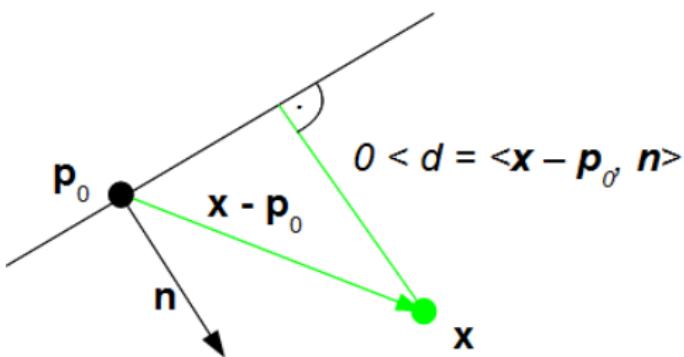


$$\cos\alpha = d / |x - p_o|$$

$$d = |\mathbf{x} - \mathbf{p}_o| \cos \alpha$$

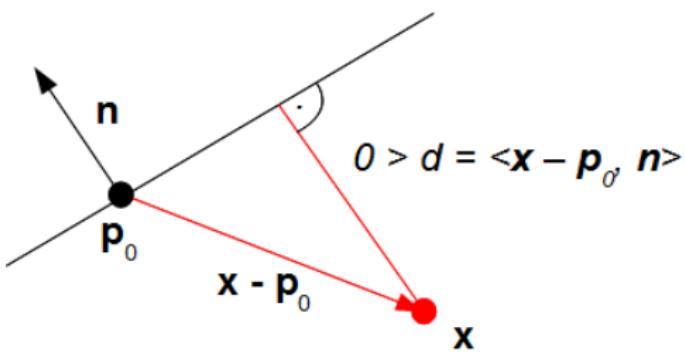
$$= \langle \mathbf{x} - \mathbf{p}_o, \mathbf{n} \rangle, |\mathbf{n}| = 1$$

## Pont-egyenes távolsága példa



## Pontok vágása

## Pont-egyenes távolsága példa





## Pontok vágása

## Pont vágása pont-normálvektoros félsíkra

- Ha csak az érdekel minket, hogy előttünk van-e egy pont és nem számít, hogy milyen messze van, akkor nem kell, hogy a normális egységnyi hosszúságú legyen!



## Pontok vágása

## Pont vágása pont-normálvektoros félsíkra

- Ha csak az érdekel minket, hogy előttünk van-e egy pont és nem számít, hogy milyen messze van, akkor nem kell, hogy a normális egységnyi hosszúságú legyen!
- Hiszen csak a skaláris szorzat előjelére vagyunk kíváncsiak, amit a vektor nemnegatív számmal való szorzása nem változtat, továbbra is csak a bezárt szögtől függ (a  $\cos$  miatt)



## Pontok vágása

## Homogén koordinátás alak

- A kibővített (projektív) sík egy egyenese megadható az  $\mathbf{e} = [e_1, e_2, e_3]$  valós számhármassal, úgynévezett *vonalkoordinátákkal*, amelyek felhasználásával az egyenes minden  $\mathbf{x} = [x_1, x_2, x_3]^T$  pontjára

$$\mathbf{e}\mathbf{x} = e_1x_1 + e_2x_2 + e_3x_3 = 0$$



## Pontok vágása

## Homogén koordinátás alak

- A kibővített (projektív) sík egy egyenese megadható az  $\mathbf{e} = [e_1, e_2, e_3]$  valós számhármaszal, úgynévezett *vonalkoordinátákkal*, amelyek felhasználásával az egyenes minden  $\mathbf{x} = [x_1, x_2, x_3]^T$  pontjára

$$\mathbf{e}\mathbf{x} = e_1x_1 + e_2x_2 + e_3x_3 = 0$$

- Az sík minden  $[x_1, x_2, 0]$  ideális pontjára illeszkedő ideális egyenes vonalkoordinátái  $[0, 0, 1]$ .



## Pontok vágása

# Pont vágása vonalkoordinátás egyenesre

- Használjuk ki, hogy a vágásnál már homogén koordinátákban dolgozik a rendszer!



## Pontok vágása

## Pont vágása vonalkoordinátás egyenesre

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



## Pontok vágása

## Pont vágása vonalkoordinátás egyenesre

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



## Pontok vágása

## A sík normálvektoros egyenlete

- A sík megadható egy  $\mathbf{p} = (p_x, p_y, p_z)$  pontjával és a síkra merőleges  $\mathbf{n} = (n_x, n_y, n_z)$  normálvektorával. Ekkor a sík minden  $\mathbf{x}$  pontjára:

$$\langle \mathbf{x} - \mathbf{p}, \mathbf{n} \rangle = 0$$



## Pontok vágása

## A sík normálvektoros egyenlete

- A sík megadható egy  $\mathbf{p} = (p_x, p_y, p_z)$  pontjával és a síkra merőleges  $\mathbf{n} = (n_x, n_y, n_z)$  normálvektorával. Ekkor a sík minden  $\mathbf{x}$  pontjára:

$$\langle \mathbf{x} - \mathbf{p}, \mathbf{n} \rangle = 0$$

- Félterek:  $\langle \mathbf{x} - \mathbf{p}, \mathbf{n} \rangle < 0$ ,  $\langle \mathbf{x} - \mathbf{p}, \mathbf{n} \rangle > 0$



## Pontok vágása

## Homogén koordinátás alak

- A kibővített tér egy síkja is megadható "síkkordinátákkal", egy olyan  $\mathbf{s} = [s_1, s_2, s_3, s_4]$  négyessel, amely a sík minden  $\mathbf{x} = [x_1, x_2, x_3, x_4]^T$  pontjára

$$\mathbf{s}\mathbf{x} = s_1x_1 + s_2x_2 + s_3x_3 + s_4x_4 = 0$$



## Pontok vágása

## Pont vágása síkra

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

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



## Pontok vágása

## Pont vágása síkra

- 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:

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

**Szakaszok vágása**

# Tartalom

## 1 Emlékeztető

## 2 Vágás

- Motiváció
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

## 3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció



## Szakaszok vágása

## Szakasz vágása félsíkra

- A szakasz **p** és **q** végpontjait vágjuk az e egyenes és normálisa által meghatározott félsíkra!



## 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!



## 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



## Szakaszok vágása

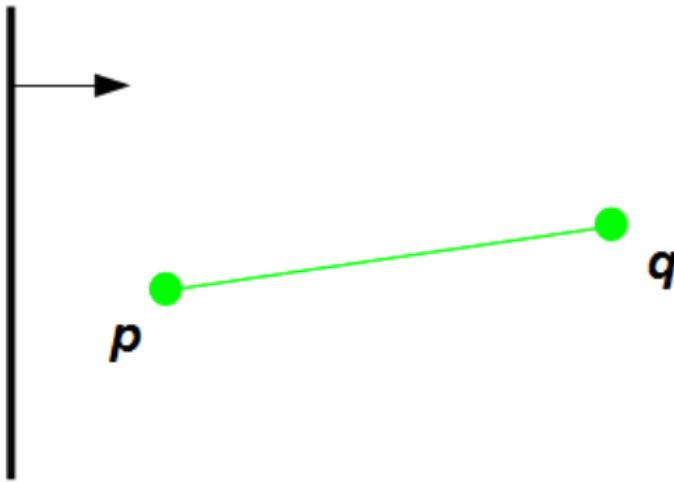
## Szakasz vágása félsíkra



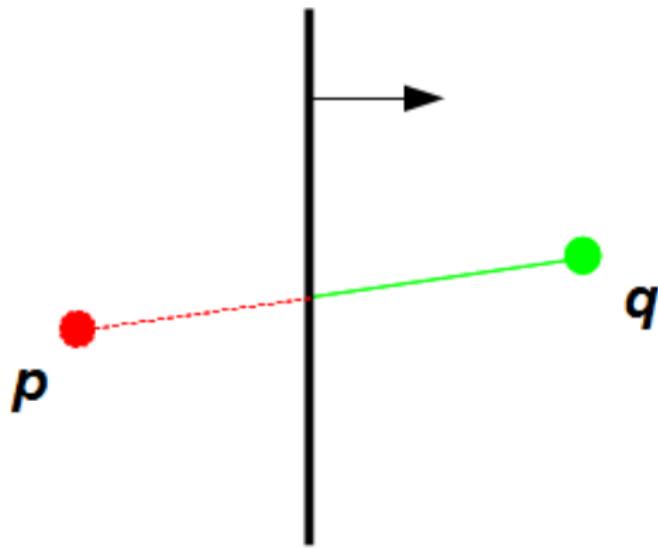


## Szakasz vágása

## Szakasz vágása félsíkra



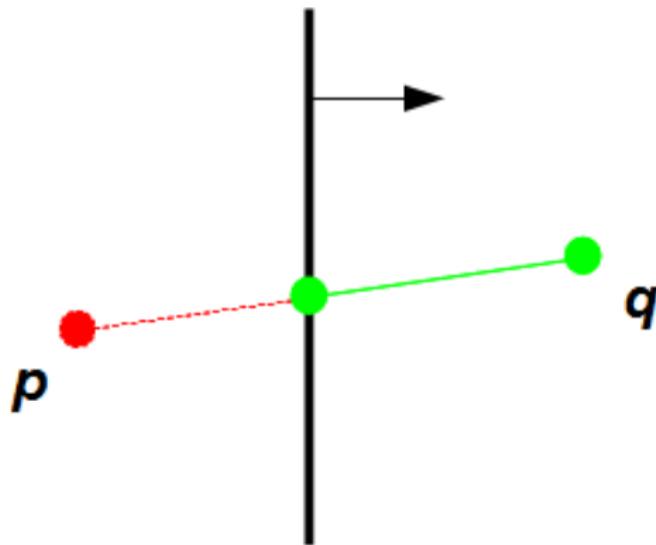
# Szakasz vágása félsíkra



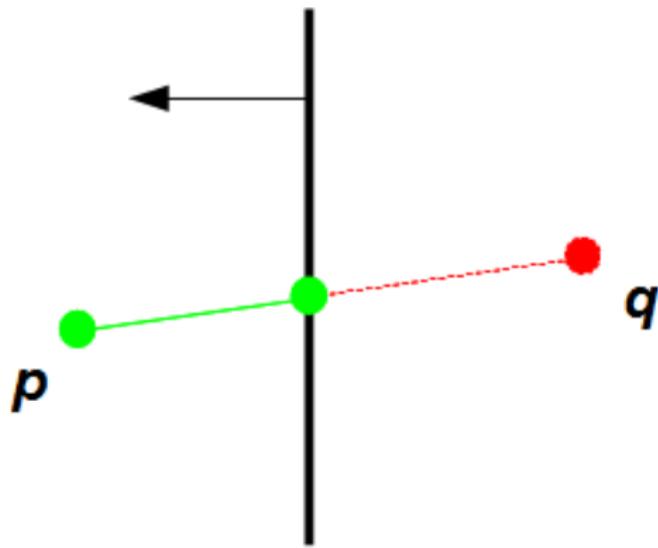


## Szakaszok vágása

## Szakasz vágása félsíkra - új "kezdőpont"!

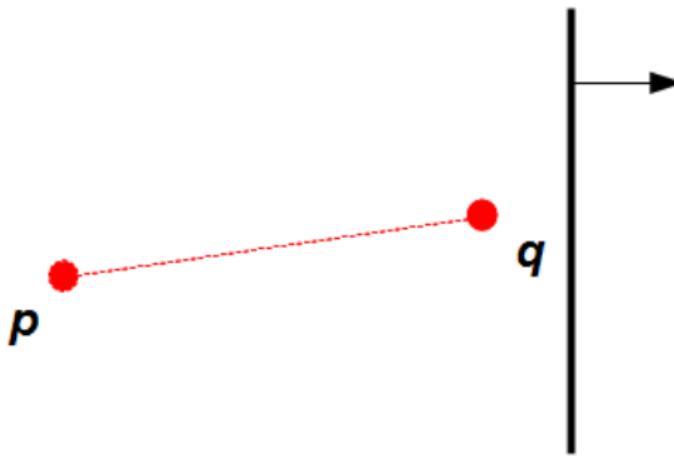


## Szakasz vágása félsíkra



## Szakasz vágása

## Szakasz vágása félsíkra





## Szakaszvágás

## Tartalom

## 1 Emlékeztető

## 2 Vágás

- Motiváció
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás**
- Poligonvágás

## 3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció



## Szakaszvágás

# Szakasz vágása félsíkra

- Csak azt a speciális esetet nézzük, amikor a vágó egyenes párhuzamos valamelyik tengellyel.



## Szakaszvágás

# Szakasz vágása félsíkra

- Csak azt a speciális esetet nézzük, amikor a vágó egyenes párhuzamos valamelyik tengellyel.
- Tegyük fel, hogy az egyik pontja a szakasznak a vágási félsíkon belül, a másik kívül van és a szakasz nem párhuzamos a tengelyekkel



## Szakaszvágás

# Szakasz vágása félsíkra

- Csak azt a speciális esetet nézzük, amikor a vágó egyenes párhuzamos valamelyik tengellyel.
- Tegyük fel, hogy az egyik pontja a szakasznak a vágási félsíkon belül, a másik kívül van és a szakasz nem párhuzamos a tengelyekkel
- $(x_1, y_1) - (x_2, y_2)$  szakasz egyenlete:

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

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



## Szakaszvágás

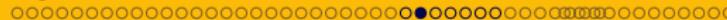
# Szakasz vágása félsíkra

- Csak azt a speciális esetet nézzük, amikor a vágó egyenes párhuzamos valamelyik tengellyel.
- Tegyük fel, hogy az egyik pontja a szakasznak a vágási félsíkon belül, a másik kívül van és a szakasz nem párhuzamos a tengelyekkel
- $(x_1, y_1) - (x_2, y_2)$  szakasz egyenlete:

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

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

- Vágó egyenes: pl.  $x = x_{min}$



## Szakaszvágás

# Szakasz vágása félsíkra

- Csak azt a speciális esetet nézzük, amikor a vágó egyenes párhuzamos valamelyik tengellyel.
- Tegyük fel, hogy az egyik pontja a szakasznak a vágási félsíkon belül, a másik kívül van és a szakasz nem párhuzamos a tengelyekkel
- $(x_1, y_1) - (x_2, y_2)$  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:  $x_{min} = x_1 + t(x_2 - x_1) \Rightarrow t = \frac{x_{min} - x_1}{x_2 - x_1}$



## Szakaszvágás

# Szakasz vágása félsíkra

- Csak azt a speciális esetet nézzük, amikor a vágó egyenes párhuzamos valamelyik tengellyel.
- Tegyük fel, hogy az egyik pontja a szakasznak a vágási félsíkon belül, a másik kívül van és a szakasz nem párhuzamos a tengelyekkel
- $(x_1, y_1) - (x_2, y_2)$  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:  $x = x_{min}, y = y_1 + \frac{x_{min} - x_1}{x_2 - x_1}(y_2 - y_1)$



# Szakasz vágása - nehézségek

- Eredmény kezelése: az előbb kapott pont lesz az új végpontja a szakasznak - ahelyett, amelyik kilógott a vágó félsíkból



## Szakaszvágás

# Szakasz vágása - nehézségek

- Eredmény kezelése: az előbb kapott pont lesz az új végpontja a szakasznak - ahelyett, amelyik kilógott a vágó félsíkból
- Kivételek: tengelyekkel párhuzamos szakaszok vágása



# Szakasz vágása - nehézségek

- Eredmény kezelése: az előbb kapott pont lesz az új végpontja a szakasznak - ahelyett, amelyik kilógott a vágó félsíkból
- Kivételek: tengelyekkel párhuzamos szakaszok vágása
- Képernyőre vágásnál is rengeteg eset: négy félsík, két végpont, mindenkor eshet bárhova



## Szakaszvágás

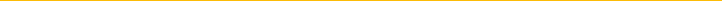
# Szakasz vágása - nehézségek

- Eredmény kezelése: az előbb kapott pont lesz az új végpontja a szakasznak - ahelyett, amelyik kilógott a vágó félsíkból
- Kivételek: tengelyekkel párhuzamos szakaszok vágása
- Képernyőre vágásnál is rengeteg eset: négy félsík, két végpont, minden kettő eshet bárhova
- Feladat: Kell-e vágni? Melyik félsík(ok)ra kell vágni? Triviálisan bent/kint van-e a szakasz?



# Szakasz vágása - nehézségek

- Eredmény kezelése: az előbb kapott pont lesz az új végpontja a szakasznak - ahelyett, amelyik kilógott a vágó félsíkból
- Kivételek: tengelyekkel párhuzamos szakaszok vágása
- Képernyőre vágásnál is rengeteg eset: négy félsík, két végpont, mindenkor eshet bárhova
- Feladat: Kell-e vágni? Melyik félsík(ok)ra kell vágni?  
Triviálisan bent/kint van-e a szakasz?
- Megoldás: *Cohen-Sutherland* vágás



## Cohen-Sutherland vágás

- A képernyő széleit reprezentáló egyenesekkel osszuk kilenc részre a síkot!



## Cohen-Sutherland vágás

- A képernyő széleit reprezentáló egyenesekkel osszuk kilenc részre a síkot!
- Mindegyik síkrészhez rendeljünk egy 4 bitből álló bit-kódot: TBRL



## 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!



## 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



## 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



## 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

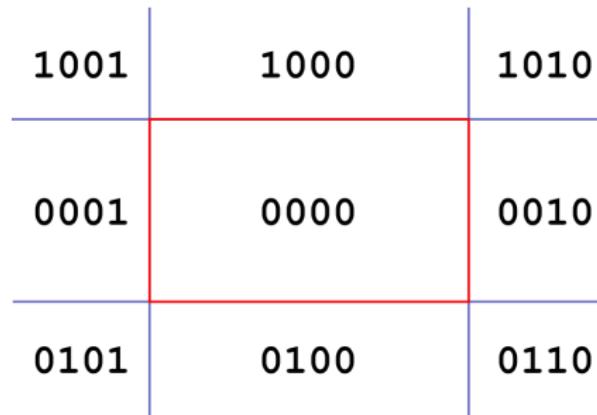


## Cohen-Sutherland vágás

- 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

# Cohen-Sutherland vágás

code = Top, Bottom, Right, Left





## Cohen-Sutherland vágás

- $\text{code} = (y > y_{max}, y < y_{min}, x > x_{max}, x < x_{min})$ , ha a képernyő oldalegyenesei párhuzamosak a tengelyekkel
- Vizsgáljuk a két végpont bit-kódjait,  $\text{code}_a$ -t és  $\text{code}_b$ -t!



## Cohen-Sutherland vágás

- $\text{code} = (y > y_{max}, y < y_{min}, X > X_{max}, X < X_{min})$ , ha a képernyő oldalegyenesei párhuzamosak a tengelyekkel
- Vizsgáljuk a két végpont bit-kódjait,  $\text{code}_a$ -t és  $\text{code}_b$ -t!
  - Ha  $\text{code}_a \text{ OR } \text{code}_b == 0$ : a szakasz egésze (mindkét végpontja) az ablakban van  $\Rightarrow$  megtartjuk.



## Szakaszvágás

# Cohen-Sutherland vágás

- $\text{code} = (y > y_{max}, y < y_{min}, x > x_{max}, x < x_{min})$ , ha a képernyő oldalegyenesei párhuzamosak a tengelyekkel
- Vizsgáljuk a két végpont bit-kódjait,  $\text{code}_a$ -t és  $\text{code}_b$ -t!
  - Ha  $\text{code}_a \text{ OR } \text{code}_b == 0$ : a szakasz egésze (mindkét végpontja) az ablakban van  $\Rightarrow$  megtartjuk.
  - Ha  $\text{code}_a \text{ AND } \text{code}_b != 0$ : a szakasz az ablak valamelyik oldalán (fölül, alul, jobbra, balra), kívül van  $\Rightarrow$  eldobjuk.



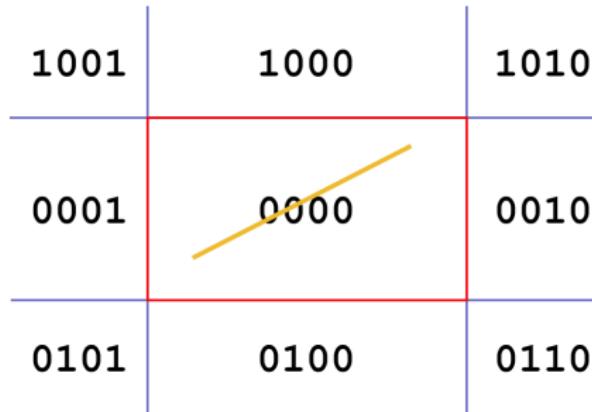
## Cohen-Sutherland vágás

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



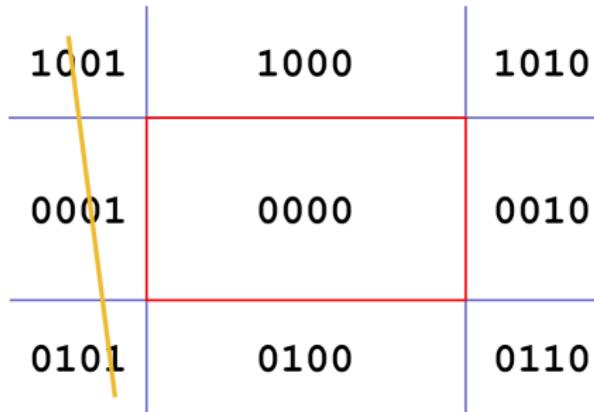
## Cohen-Sutherland vágás

$\text{code}_a \text{ OR } \text{code}_b == 0:$



# Cohen-Sutherland vágás

$\text{code}_a \text{ AND } \text{code}_b \neq 0:$



# Cohen-Sutherland vágás

Különben:





## Poligonvágás

# Tartalom

### 1 Emlékeztető

### 2 Vágás

- Motiváció
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás
- **Poligonvágás**

### 3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció



## Polygonvágás

## 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.



## Polygonvágás

## 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.



## Poligonvágás

# *Sutherland-Hodgman poligonvágás*

- Legyen  $p$  tömb a bemeneti-,  $q$  a kimeneti-poligon csúcsainak tömbje!



## Poligonvágás

# *Sutherland-Hodgman poligonvágás*

- Legyen  $p$  tömb a bemeneti-,  $q$  a kimeneti-poligon csúcsainak tömbje!
- Legyen  $n$  a csúcsok száma, és  $p[0] = p[n]!$



# *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 alablaiban, és  $p[i + 1]$  is:  
 $\Rightarrow$  adjuk hozzá  $p[i]$ -t a  $q$ -hoz.



## Poligonvágás

# 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.



## 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.



## 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



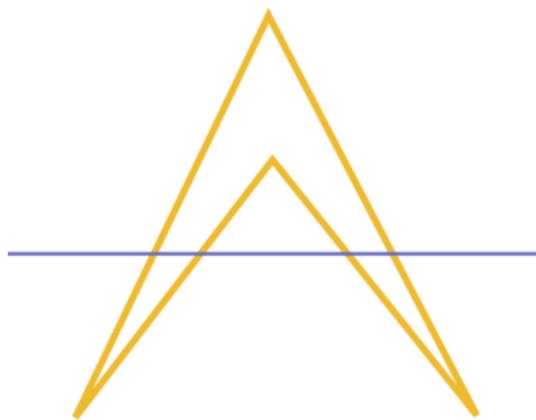
## Poligonvágás

# *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);  
        }  
    }  
}
```

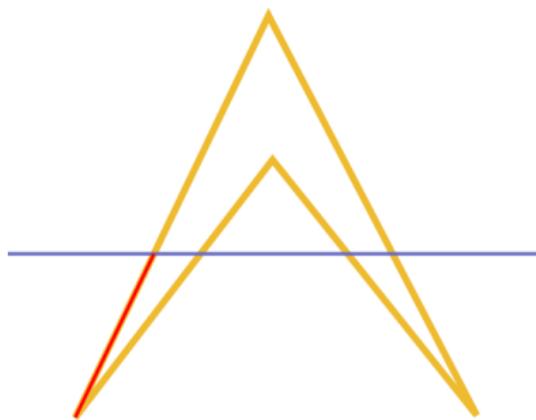
## *Sutherland-Hodgeman: konkáv poligonok*

Konkáv poligonokra átfedő éleket hoz létre.



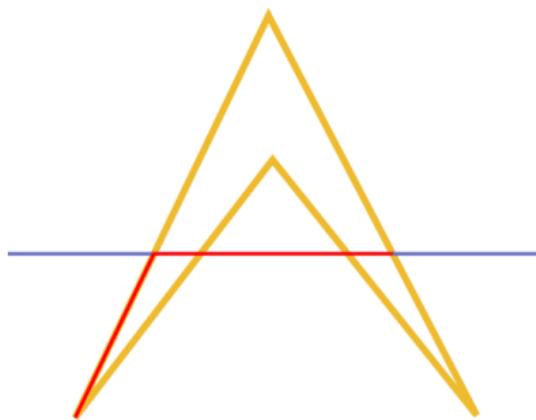
## *Sutherland-Hodgeman: konkáv poligonok*

Konkáv poligonokra átfedő éleket hoz létre.



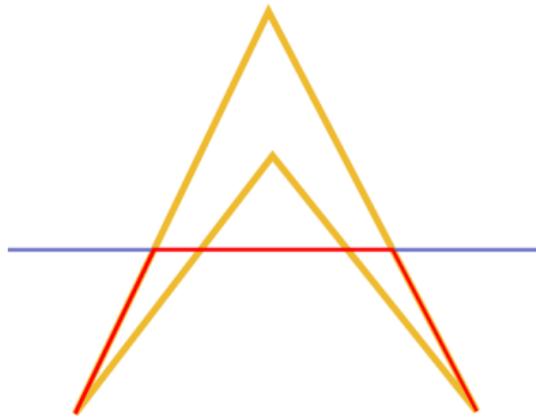
## *Sutherland-Hodgeman: konkáv poligonok*

Konkáv poligonokra átfedő éleket hoz létre.



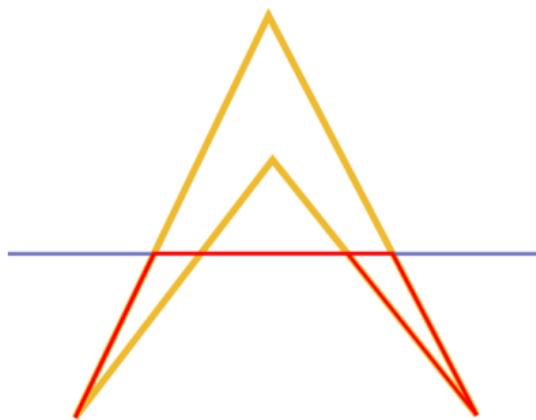
## *Sutherland-Hodgeman: konkáv poligonok*

Konkáv poligonokra átfedő éleket hoz létre.



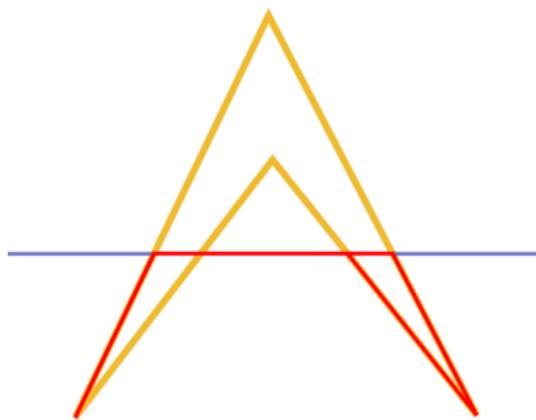
## *Sutherland-Hodgeman: konkáv poligonok*

Konkáv poligonokra átfedő éleket hoz létre.



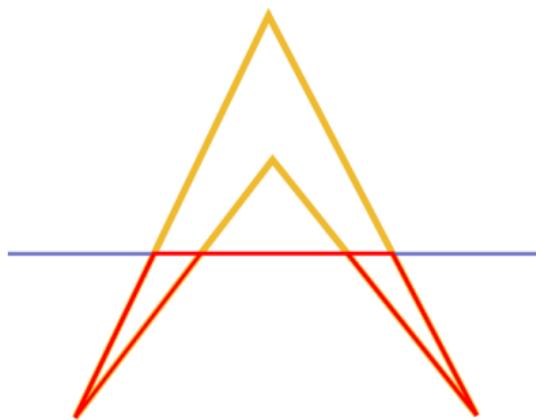
## *Sutherland-Hodgeman: konkáv poligonok*

Konkáv poligonokra átfedő éleket hoz létre.



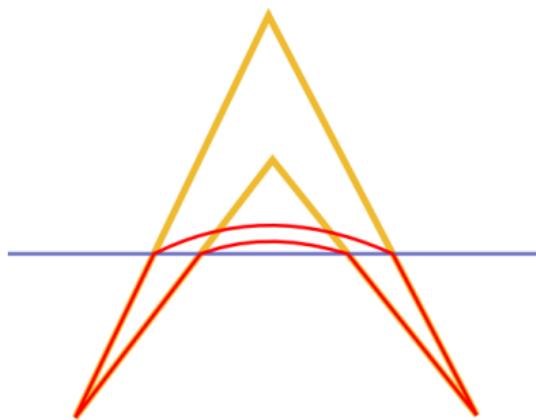
### *Sutherland-Hodgeman: konkáv poligonok*

Konkáv poligonokra átfedő éleket hoz létre.



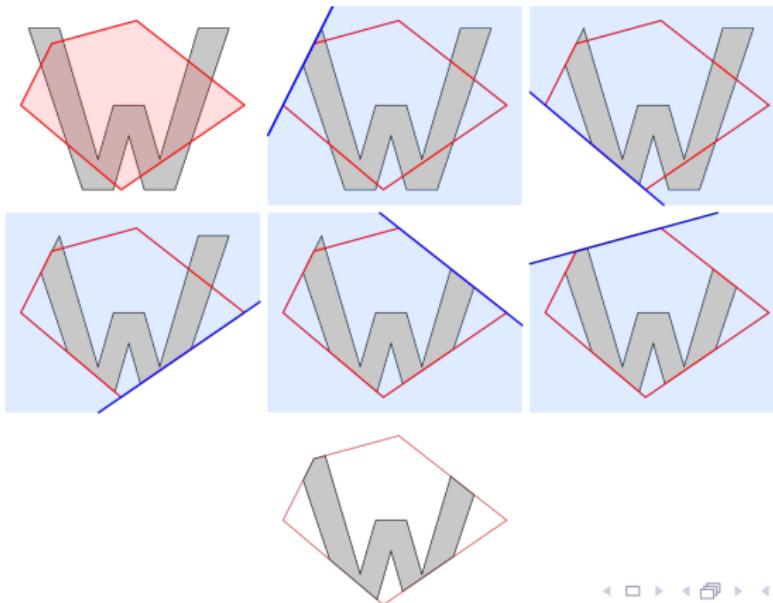
### *Sutherland-Hodgeman: konkáv poligonok*

Konkáv poligonokra átfedő éleket hoz létre.



# Sutherland-Hodgeman: poligonra vágása poligonnak

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





## Poligonvágás

# Mikor vágunk?

- Projektív transzformáció előtt



## Poligonvágás

# Mikor vágunk?

- Projektív transzformáció előtt → a vágósíkok világtérbeli megadásával (mik az egyenletei?)



## Poligonvágás

# Mikor vágunk?

- Projektív transzformáció előtt → a vágósíkok világtérbeli megadásával (mik az egyenletei?)
- NPKR-ben (projektív trafó után, homogén osztás előtt)



## Poligonvágás

# Mikor vágunk?

- 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!



## Poligonvágás

# Mikor vágunk?

- 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)



## Poligonvágás

# Mikor vágunk?

- 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...



## Szakasz raszterizálása

## Tartalom

## 1 Emlékeztető

## 2 Vágás

- Motiváció
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

## 3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció



## Szakasz raszterizálása

## Szakasz rajzolás

- Egyik leggyakrabbr primitív



## Szakasz raszterizálása

## Szakasz rajzolás

- Egyik leggyakrabbról primitív
- Fontos, hogy szépen tudjuk rajzolni



## Szakasz raszterizálása

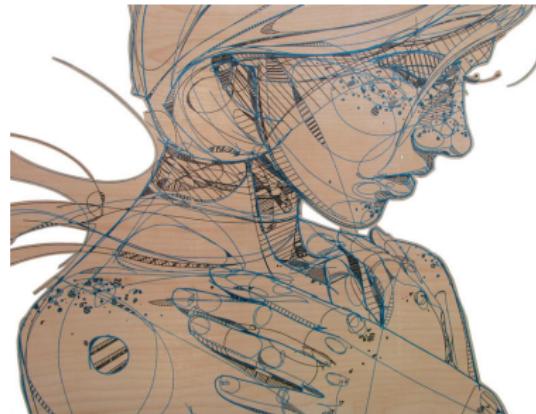
# Szakasz rajzolás

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

## Szakasz rászterizálása

## Szakasz rajzolás

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



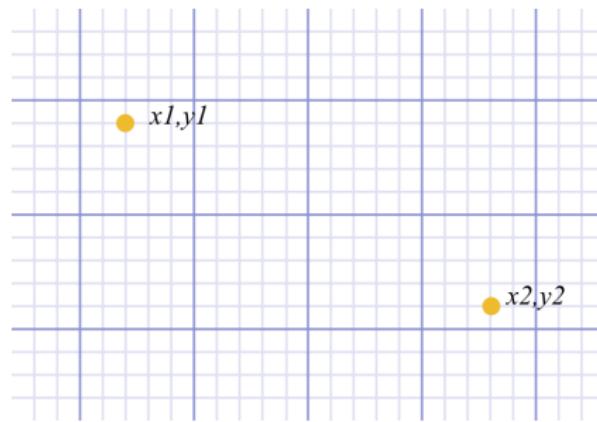
Jason Thielke, jasonthielke.com



## Szakasz raszterizálása

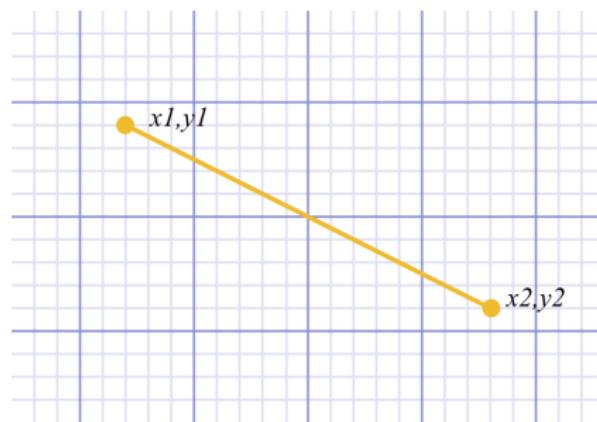
## Hogyan rajzolunk szakaszt?

- Adott a két végpont.



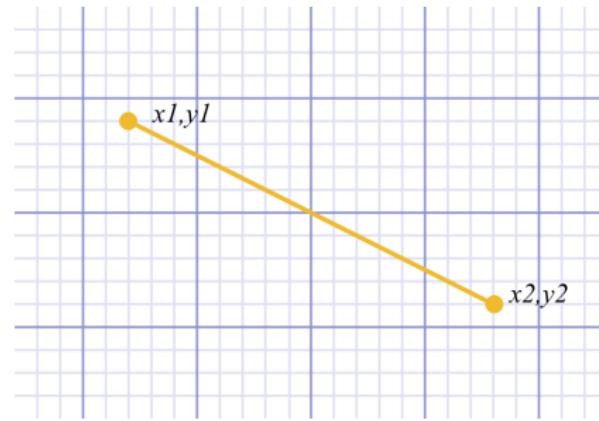
## Hogyan rajzolunk szakaszt?

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



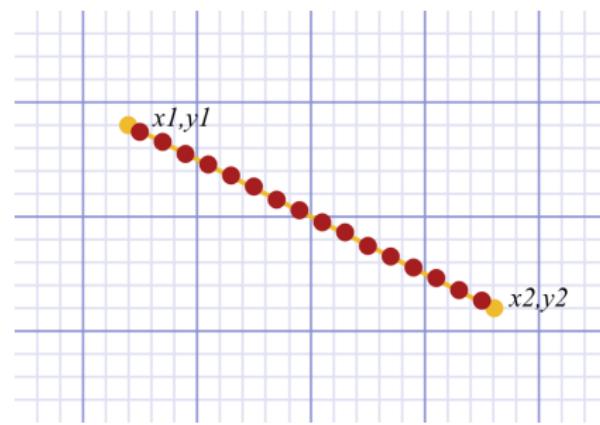
## Hogyan rajzolunk szakaszt?

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



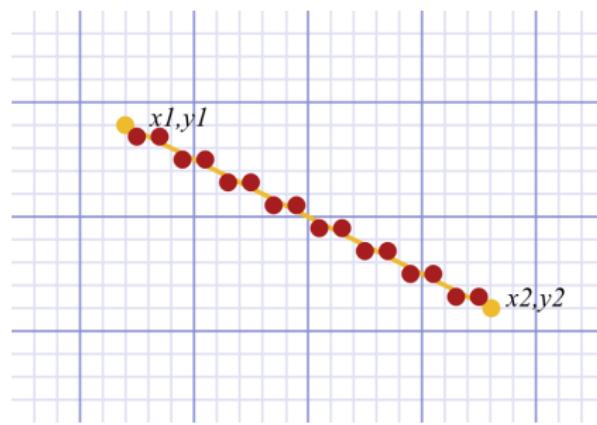
## Hogyan rajzolunk szakaszt?

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



## Hogyan rajzolunk szakaszt?

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





## Szakasz raszterizálása

## Szakasz megadása (sokadszor)

- Végpontok:  $(x_1, y_1), (x_2, y_2)$



## Szakasz raszterizálása

## Szakasz megadása (sokadszor)

- Végpontok:  $(x_1, y_1), (x_2, y_2)$
- Tfh. nem függőleges,  $x_1 \neq x_2$ , és  $x_1 < x_2$  (ha nem: csere)



## Szakasz raszterizálása

## Szakasz megadása (sokadszor)

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

## Szakasz raszterizálása

# Szakasz megadása (sokadszor)

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

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

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

$$b = y_1 - mx_1$$

## Szakasz raszterizálása

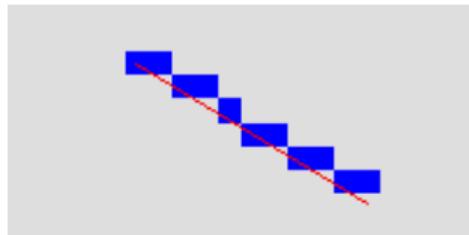
## Naív algoritmus

```
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
```

## Szakasz raszterizálása

## Naív algoritmus

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

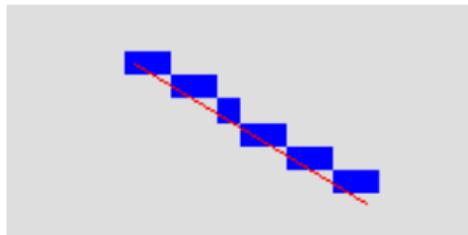




## Szakasz raszterizálása

## Naív algoritmus

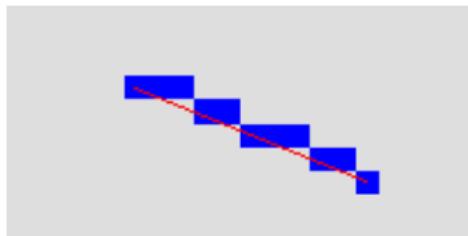
- $m = \text{float}(y_2 - y_1) / (x_2 - x_1)$   
nem pontos
- $y += m \rightarrow$  a hiba gyűlik  $y$ -ban



## Szakasz raszterizálása

## Naív algoritmus

- $m = \text{float}(y_2 - y_1) / (x_2 - x_1)$   
nem pontos
- $y += m \rightarrow$  a hiba gyűlik  $y$ -ban

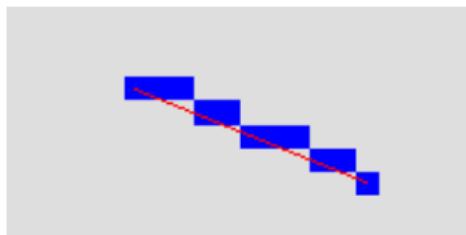




## Szakasz raszterizálása

## Naív algoritmus

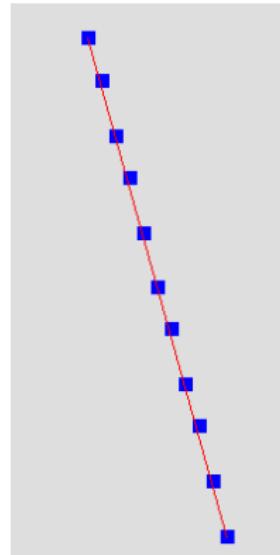
- $m = \text{float}(y_2 - y_1) / (x_2 - x_1)$   
nem pontos
- $y += m \rightarrow$  a hiba gyűlik  $y$ -ban
- `draw.point((x, y))` →  
int-eket vár, lassú a konverzió



## Szakasz raszterizálása

## Naív algoritmus

- $m = \text{float}(y_2 - y_1) / (x_2 - x_1)$   
nem pontos
- $y += m \rightarrow$  a hiba gyűlik  $y$ -ban
- `draw.point((x, y))` →  
int-eket vár, lassú a konverzió
- csak  $|m| < 1$ -re működik  
helyesen



## Szakasz raszterizálása

## Javítsuk az algoritmust 1.

```
def line2(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
```



## Szakasz raszterizálása

## Javítsuk az algoritmust 1.

- Jó: Mindig "eltalálja" a végpontokat



## Szakasz raszterizálása

## Javítsuk az algoritmust 1.

- Jó: Mindig "eltalálja" a végpontokat
- Jó: Egyenletesebben lép az  $y$  irányban.



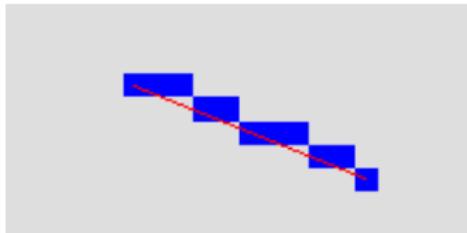
## Szakasz raszterizálása

## Javítsuk az algoritmust 1.

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

## Javítsuk az algoritmust 1.

Naív:



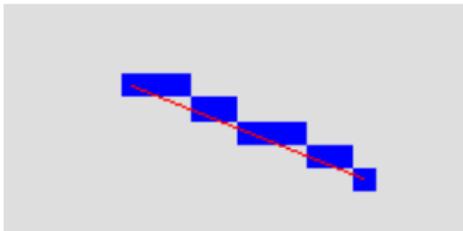
- Jó: Mindig "eltalálja" a végpontokat
  - Jó: Egyenletesebben lép az  $y$  irányban.
  - Rossz: Még mindig használunk float-okat
  - Rossz: Csak  $0 \leq m \leq 1$ -re működik csak

## Szakasz raszterizálása

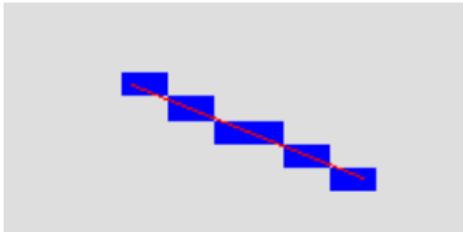
## Javítsuk az algoritmust 1.

- Jó: Mindig "eltalálja" a végpontokat
- Jó: Egyenletesebben lép az  $y$  irányban.
- Rossz: Még mindig használunk float-okat
- Rossz: Csak  $0 \leq m \leq 1$ -re működik csak

Naív:



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) ←
```



## Szakasz raszterizálása

## 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) ←
```



## Szakasz raszterizálása

## Javítsuk az algoritmust 4.

- Ez a *Bresenham* algoritmus (egyik speciális esete)



## Javítsuk az algoritmust 4.

- Ez a *Bresenham* algoritmus (egyik speciális esete)
- Külön gyűjtjük a hibát e-ben



## Szakasz raszterizálása

## Javítsuk az algoritmust 4.

- Ez a *Bresenham* algoritmus (egyik speciális esete)
- Külön gyűjtjük a hibát  $e$ -ben
- Nem használunk float-okat



## 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.



## Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, minden egyik külön eset.



## Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, mindenek külön eset.
- (Előző javítások végig:  $0 \leq m \leq 1$ )



## Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, mindenek külön eset.
- (Előző javítások végig:  $0 \leq m \leq 1$ )
- El kell döntenünk, hogy  $|x_2 - x_1|$  vagy  $|y_2 - y_1|$  a nagyobb (merre meredekebb a szakasz).



## Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, mindenek külön eset.
- (Előző javítások végig:  $0 \leq m \leq 1$ )
- 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!



## Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, mindenek külön eset.
- (Előző javítások végig:  $0 \leq m \leq 1$ )
- 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:  $x_1 \leftrightarrow x_2$ ,  $y_1 \leftrightarrow y_2$ .



## Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, minden egyik külön eset.
- (Előző javítások végig:  $0 \leq m \leq 1$ )
- 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  $|y_2 - y_1|$ -nál növeljük minden lépésben

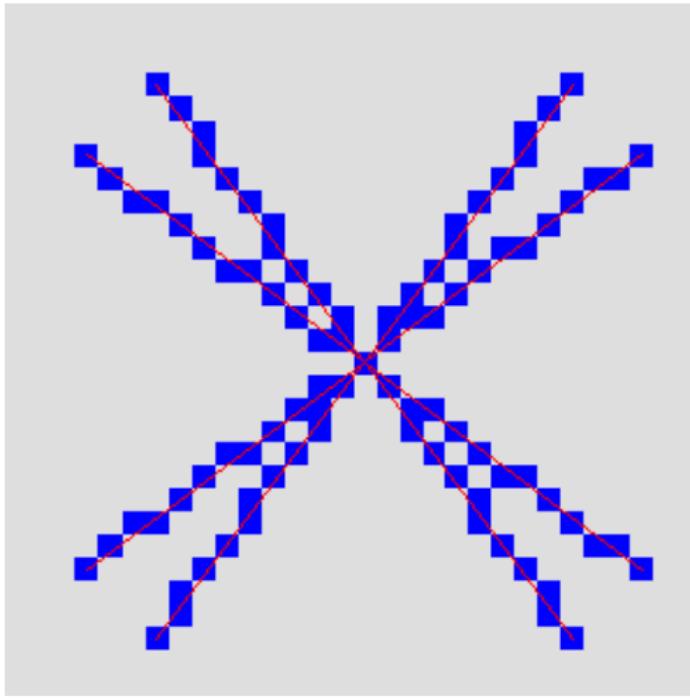


## Szakasz raszterizálása

## Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, mindenek külön eset.
- (Előző javítások végig:  $0 \leq m \leq 1$ )
- 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
- $y$ -nál  $y_2 - y_1$  előjele szerint haladunk.

# *Bresenham algoritmus*





## Szakasz raszterizálása

Teljes *Bresenham* algoritmust 1.

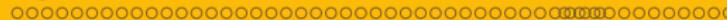
```
def Bresenham(x1,y1,x2,y2, draw):
    steep = abs(y2-y1)>abs(x2-x1)
    if steep:
        x1, y1 = y1, x1
        x2, y2 = y2, x2
    if x1>x2:
        x1, x2 = x2, x1
        y1, y2 = y2, y1
    Dy = abs(y2-y1)
    if y1<y2:
        Sy = 1
    else:
        Sy = -1
```



## 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)
```



## Háromszög raszterizálása

## Tartalom

## 1 Emlékeztető

## 2 Vágás

- Motiváció
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

## 3 Raszterizálás

- Szakasz raszterizálása
- **Háromszög raszterizálása**
- Poligon raszterizáció



## Háromszög raszterizálása

## Háromszög raszterizáció

- A háromszög oldalait tudjuk vágni - most töltük ki a belsejét!

## Háromszög raszterizálása

# Háromszög raszterizáció

- A háromszög oldalait tudjuk vágni - most töltük ki a belsejét!
- Ha egy meghatározott bejárási irányban adtuk meg az összes háromszög csúcsát, tudunk félsíkokat adni (tudjuk irányítani az éleket)



## Háromszög raszterizálása

## Háromszög raszterizáció

- A háromszög oldalait tudjuk vágni - most töltük ki a belsejét!
- Ha egy meghatározott bejárási irányban adtuk meg az összes háromszög csúcsát, tudunk félsíkokat adni (tudjuk irányítani az éleket) → u.i. ha  $(t_x, t_y)$  az irányvektora az oldalnak, akkor  $(-t_y, t_x)$  egy normális lesz



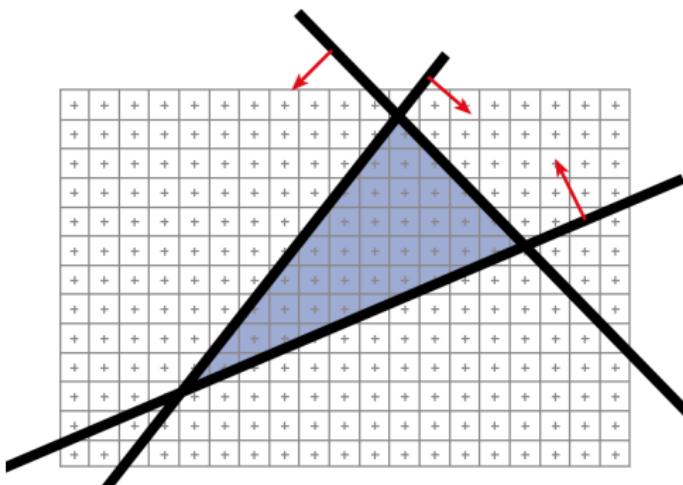
## Háromszög raszterizálása

## Háromszög raszterizáció

- A háromszög oldalait tudjuk vágni - most töltük ki a belsejét!
- Ha egy meghatározott bejárási irányban adtuk meg az összes háromszög csúcsát, tudunk félsíkokat adni (tudjuk irányítani az éleket) → 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!

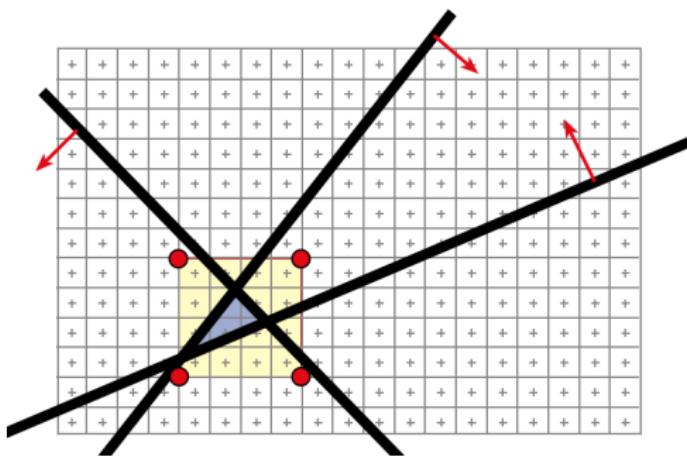
## Háromszög raszterizálása

## Háromszög raszterizáció



## Háromszög raszterizálása

## Háromszög raszterizáció - okosabban





## Háromszög raszterizálása

## Háromszög raszterizáció

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



## Háromszög raszterizálása

## Háromszög raszterizáció

- Nem egy rögzített értékkel akarunk kitölteni, a hanem a csúcsokban adott értékeket akarjuk interpolálni.



# Háromszög raszterizáció

- Nem egy rögzített értékkel akarunk kitölteni, a hanem a csúcsokban adott értékeket akarjuk interpolálni.
- Felhasználásai: szín (Gouraud-árnyalás), textúra koordináták, normálvektorok



## Háromszög raszterizálása

## Háromszög raszterizáció

- Nem egy rögzített értékkel akarunk kitölteni, a hanem a csúcsokban adott értékeket akarjuk interpolálni.
- Felhasználásai: szín (Gouraud-árnyalás), textúra koordinák, normálvektorok
- Legyen a felület egy pontja  $p = \alpha p_1 + \beta p_2 + \gamma p_3$ , az  $\alpha, \beta, \gamma$  baricentrikus koordinátákkal adott.



## Háromszög raszterizálása

## Háromszög raszterizáció

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

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



## Háromszög raszterizáció

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

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

- Ez az úgy nevezett *Gouraud interpoláció*



## Háromszög raszterizálása

## Háromszög kitöltés 1.

```
for all x:  
  for all y:  
     $\alpha, \beta, \gamma = \text{barycentric}(x, y)$   
    if  $\alpha \in [0, 1]$  and  $\beta \in [0, 1]$  and  $\gamma \in [0, 1]$ :  
       $c = \alpha c_1 + \beta c_2 + \gamma c_3$   
      draw . point((x, y), c)
```



# Baricentrikus koordináták

- A baricentrikus koordináták számíthatók a következő képletekkel:

$$f_{01}(x, y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0$$

$$f_{12}(x, y) = (y_1 - y_2)x + (x_2 - x_1)y + x_1y_2 - x_2y_1$$

$$f_{20}(x, y) = (y_2 - y_0)x + (x_0 - x_2)y + x_2y_0 - x_0y_2$$

- Ekkor az  $x, y$  ponthoz tartozó baricentrikus koordináták:

$$\alpha = f_{12}(x, y)/f_{12}(x_0, y_0)$$

$$\beta = f_{20}(x, y)/f_{20}(x_1, y_1)$$

$$\gamma = f_{01}(x, y)/f_{01}(x_2, y_2)$$

## Háromszög raszterizálása

## Háromszög kitöltés 2.

```
x_min = min(floor(x_i))
x_max = max(ceiling(x_i))
y_min = min(floor(y_i))
y_max = max(ceiling(y_i))
for y in [y_min..y_max]:
    for x in [x_min..x_max]:
        α = f12(x,y)/f12(x0,y0)
        β = f20(x,y)/f20(x1,y1)
        γ = f01(x,y)/f01(x2,y2)
        if α>0 and β>0 and γ>0:
            c = αc_1 + βc_2 + γc_3
            draw.point((x,y),c)
```



## Háromszög raszterizálása

## Háromszög kitöltés 2.

- Gyorsítás: felesleges minden  $x, y$ -t vizsgálni, elég a háromszöget tartalmazó téglalapon végig menni.



## Háromszög raszterizálása

## Háromszög kitöltés 2.

- Gyorsítás: felesleges minden  $x, y$ -t vizsgálni, elég a háromszöget tartalmazó téglalapon végig menni.
- Inkrementálissá tétele:
  - Most még lassú, nem használjuk, ki hogy sorban megyünk  $x$ -en,  $y$ -on.



## Háromszög raszterizálása

## Háromszög kitöltés 2.

- Gyorsítás: felesleges minden  $x, y$ -t vizsgálni, elég a háromszöget tartalmazó téglalapon végig menni.
- Inkrementálissá tétele:
  - Most még lassú, nem használjuk, ki hogy sorban megyünk  $x$ -en,  $y$ -on.
  - Milyenek is ezek az  $f$ -ek?



## Háromszög raszterizálása

## Háromszög kitöltés 2.

- Gyorsítás: felesleges minden  $x, y$ -t vizsgálni, elég a háromszöget tartalmazó téglalapon végig menni.
- Inkrementálissá tétele:
  - Most még lassú, nem használjuk, ki hogy sorban megyünk  $x$ -en,  $y$ -on.
  - Milyenek is ezek az  $f$ -ek?
  - Mind  $f(x, y) = Ax + By + C$  alakú.

## Háromszög raszterizálása

## Háromszög kitöltés 2.

- Gyorsítás: felesleges minden  $x, y$ -t vizsgálni, elég a háromszöget tartalmazó téglalapon végig menni.
- Inkrementálissá tétele:
  - Most még lassú, nem használjuk, ki hogy sorban megyünk  $x$ -en,  $y$ -on.
  - Milyenek is ezek az  $f$ -ek?
  - Mind  $f(x, y) = Ax + By + C$  alakú.
  - Ekkor  $f(x + 1, y) = f(x, y) + A$ , ill.
  - $f(x, y + 1) = f(x, y) + B$



## Háromszög raszterizálása

## Háromszög kitöltés 2.

- Gyorsítás: felesleges minden  $x, y$ -t vizsgálni, elég a háromszöget tartalmazó téglalapon végig menni.
- Inkrementálissá tétele:
  - Most még lassú, nem használjuk, ki hogy sorban megyünk  $x$ -en,  $y$ -on.
  - Milyenek is ezek az  $f$ -ek?
  - Mind  $f(x, y) = Ax + By + C$  alakú.
  - Ekkor  $f(x + 1, y) = f(x, y) + A$ , ill.
  - $f(x, y + 1) = f(x, y) + B$
- Megvalósítás: *házi feladat*



## Poligon raszterizáció

# Tartalom

### 1 Emlékeztető

### 2 Vágás

- Motiváció
- Pontok vágása
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

### 3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- **Poligon raszterizáció**



## Poligon raszterizáció

# Flood-fill – elárasztásos kitöltés

- Tetszőleges, már raszterizált poligon kitöltésére alkalmas.

**Poligon raszterizáció**

# Flood-fill – elárasztásos kitöltés

- Tetszőleges, már raszterizált poligon kitöltésére alkalmas.
- Bemenet: raszter kép + annak egy pontja



## Flood-fill – elárasztásos kitöltés

- Tetszőleges, már raszterizált poligon kitöltésére alkalmas.
- Bemenet: raszter kép + annak egy pontja
- Brute-force: a megadott pontból kiindulva rekurzívan haladunk:
  - Az aktuális pont színe megegyezik a kiindulási pont színével?



## Flood-fill – elárasztásos kitöltés

- Tetszőleges, már raszterizált poligon kitöltésére alkalmas.
- Bemenet: raszter kép + annak egy pontja
- Brute-force: a megadott pontból kiindulva rekurzívan haladunk:
  - Az aktuális pont színe megegyezik a kiindulási pont színével?
    - nem megállunk
    - igen átszínezzük, és



## Flood-fill – elárasztásos kitöltés

- Tetszőleges, már raszterizált poligon kitöltésére alkalmas.
- Bemenet: raszter kép + annak egy pontja
- Brute-force: a megadott pontból kiindulva rekurzívan haladunk:
  - Az aktuális pont színe megegyezik a kiindulási pont színével?
    - nem megállunk
    - igen átszínezzük, és
  - minden szomszédra újrakezdjük az algoritmust.



## Poligon raszterizáció

# Flood-fill – szomszédásgok

- Négy szomszéd: fent, lent, jobbra, balra



# Flood-fill – szomszédásgok

- Négy szomszéd: fent, lent, jobbra, balra
- Nyolc szomszéd: az előző négy + a sarkok



## Flood-fill – szomszédásgok

- Négy szomszéd: fent, lent, jobbra, balra
- Nyolc szomszéd: az előző négy + a sarkok
- Rekurzió nagyon durva: gyakorlatban ennél okosabb algoritmusok is vannak



## Flood-fill – szomszédásgok

- Négy szomszéd: fent, lent, jobbra, balra
- Nyolc szomszéd: az előző négy + a sarkok
- Rekurzió nagyon durva: gyakorlatban ennél okosabb algoritmusok is vannak → aktív éllista stb. (Haladó Grafika)