

# Gràfics i Visualització de Dades

Tema 4: Dels vèrtexs als fragments: projecció,  
clipping i mètodes de rasterització (part 2)

Anna Puig

**NOTA:** Aquestes transparències es corresponen a les explicacions del tema 6 del llibre de referència bàsica

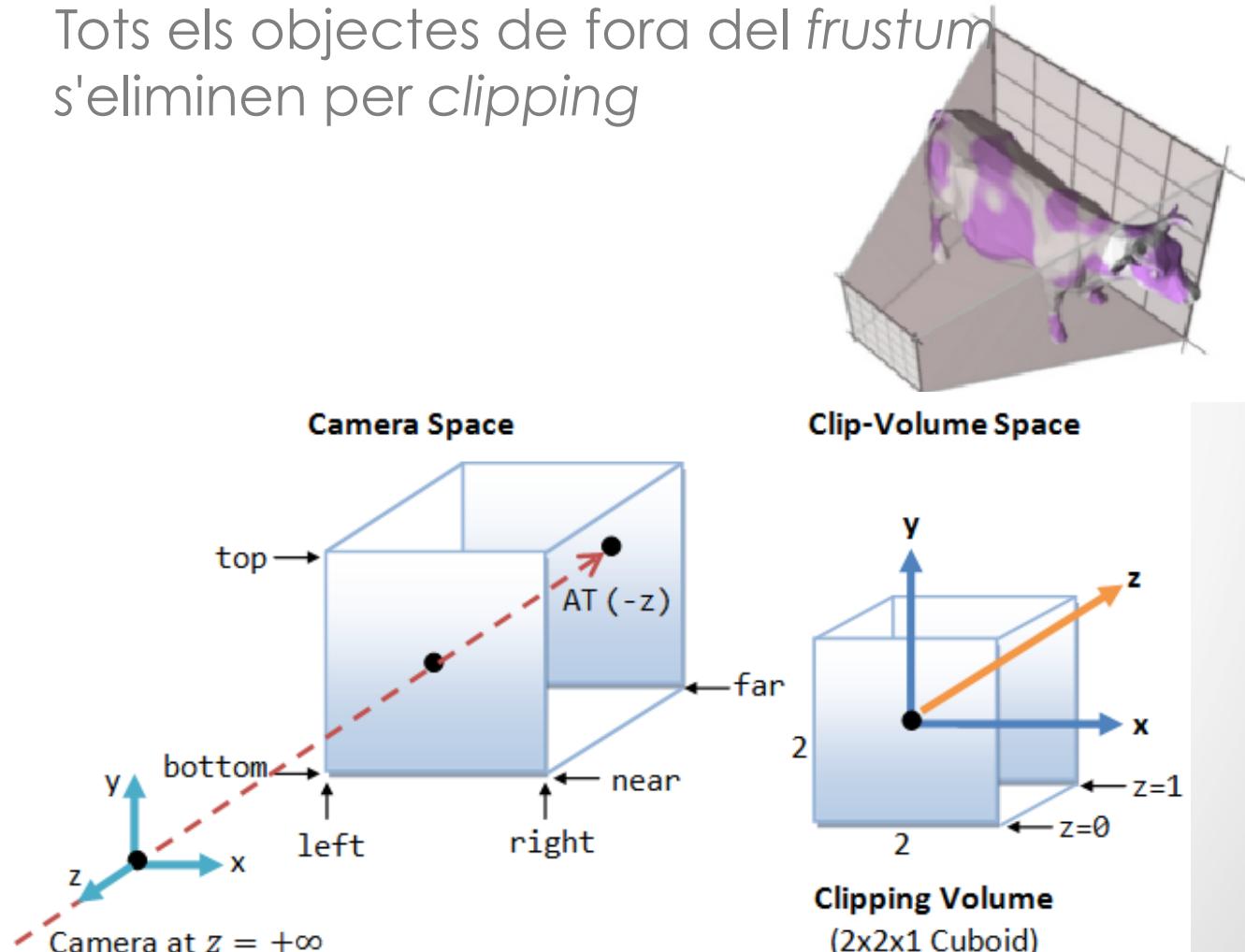
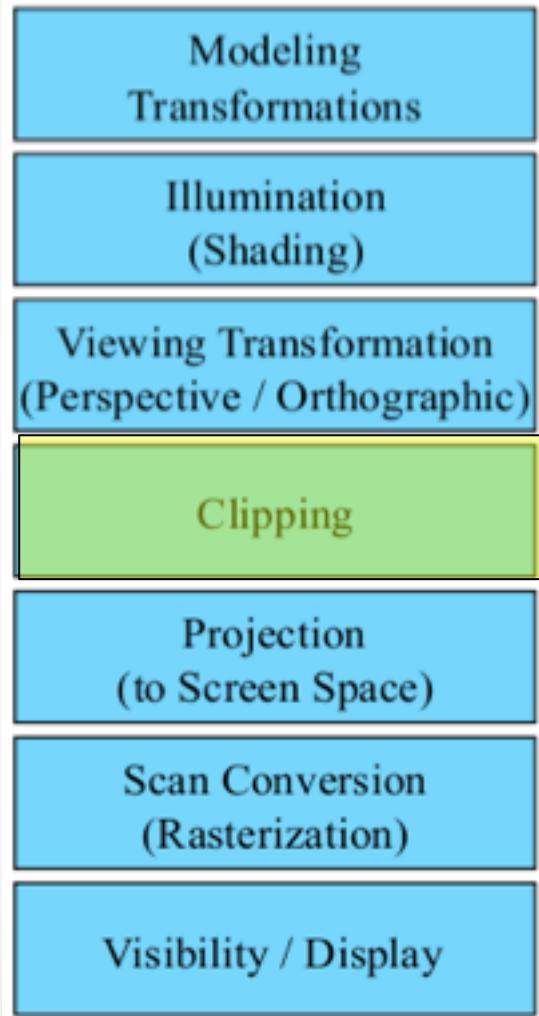
[Angel2011] Edward Angel, Dave Shreiner, **Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL, 6/E**, ISBN-10: 0132545233. ISBN-13: 9780132545235, Addison-Wesley, 2011

# Índex

- 4.1. Introducció
- 4.2. Viewing: Projeccions
- 4.3. Volum de visió: definició del clipping
- 4.4. Normalització
- 4.5. Implementació en OpenGL
- 4.6. Implementació en GPUs
- 4.7. Clipping: mètodes
- 4.8. Rasterització
- 4.9. Eliminació de parts amagades
- 4.10. Implementació en OpenGL i en GPUs

# Pipeline de visualització

- Es **normalitzen** les coordenades dels vèrtexs
- Tots els objectes de fora del *frustum* s'eliminen per clipping



# 4.7. Clipping (mètodes)

D'entrada es tenen els vèrtexs, la topologia i els materials i la sortida són els píxels amb colors associats. Per a realitzar aquest procés hi han dues estratègies:

- orientades a **objectes** (*pintor*, per exemple)

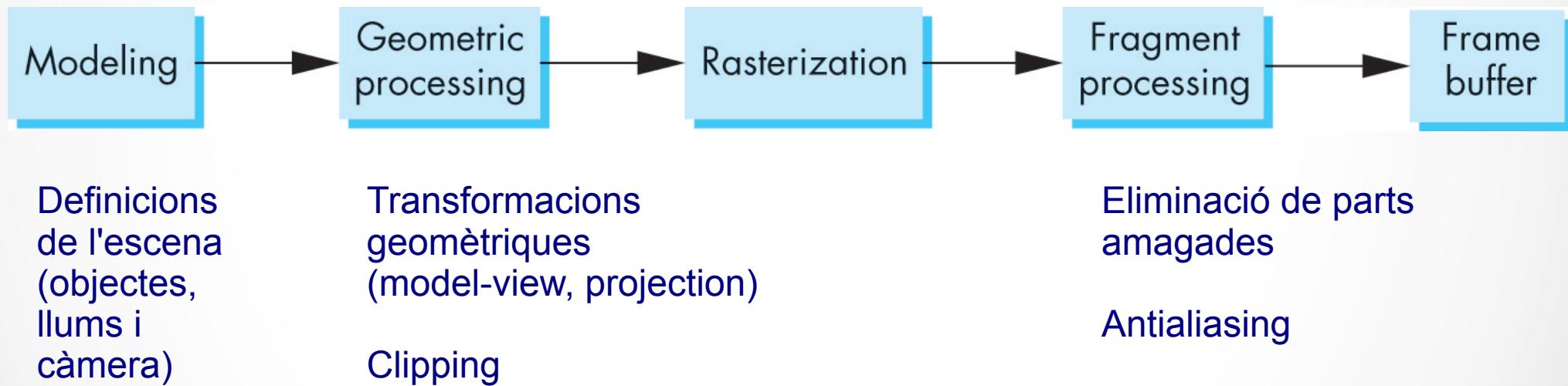
per a cada objecte( $obj$ )  
visualitza ( $obj$ )

- orientades a **píxels** (*raytracing*, per exemple)

per a cada píxel del frame buffer ( $p$ )  
assigna color a píxel ( $p$ )

# 4.7. Clipping (mètodes)

- Si ens fixem en el pipeline d'OpenGL, des dels vèrtexs fins als píxels finals, els processos s'agrupen en mòduls:

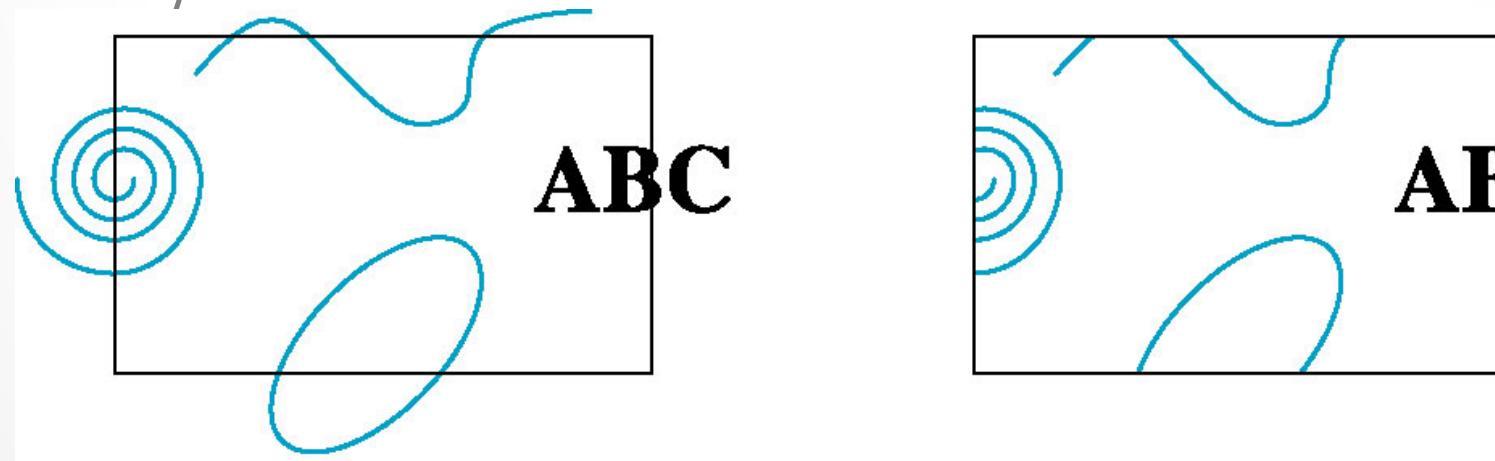


# 4.7. Clipping (mètodes)

El *clipping* permet descartar objectes o parts d'objectes que són for a del volum de visió. Es fa:

- Clipping 2D contra la window
- Clipping 3D contra el volum de visió

Es fàcil per segments rectes, però és difícil per corbes i text (una estratègia és convertir-los en polígons primer)

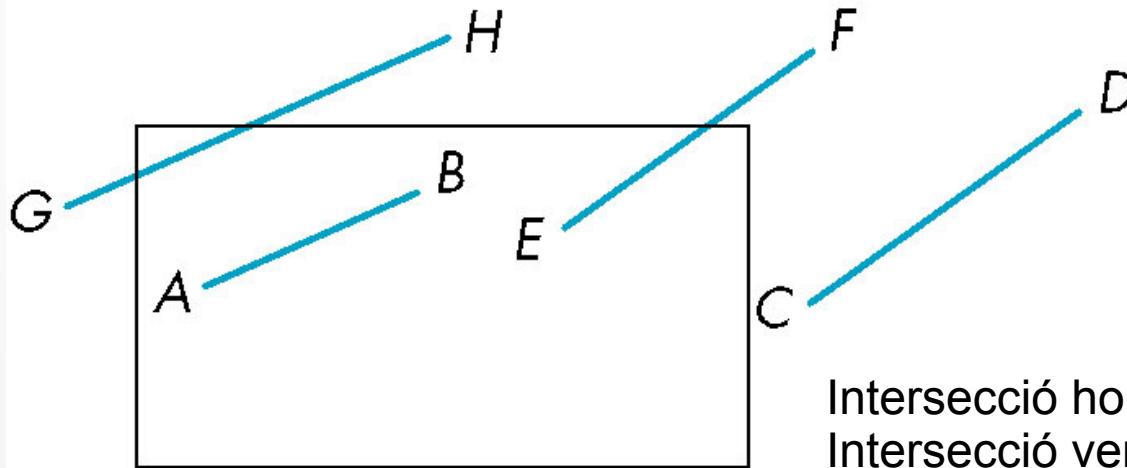


# 4.7. Clipping (mètodes)

- **Clipping 2D de segments de recta:**

Mètode de força bruta: calcular les interseccions de totes les arestes de la finestra de clipping contra les arestes dels objectes.

Ineficient: cada intersecció requereix com a mínim una divisió en coma flotant.



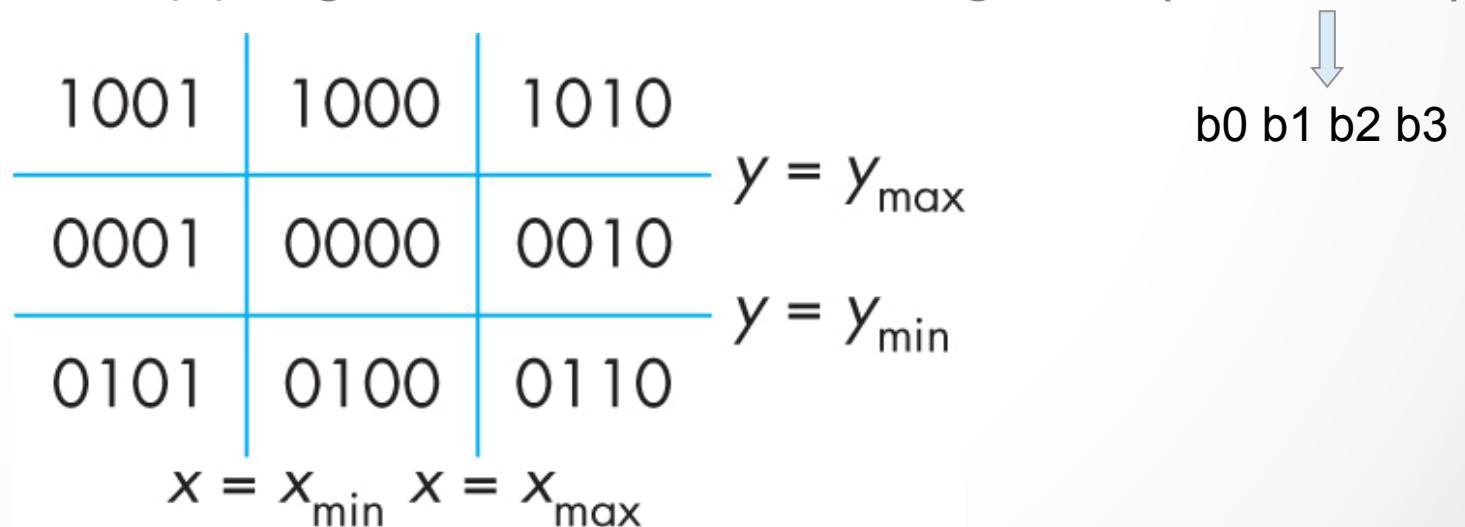
Intersecció horitzontal:  $x = x_0 + m * (y - y_0)$   
Intersecció vertical:  $y = y_0 + (1/m) * (x - x_0)$

# 4.7. Clipping (mètodes)

**Cohen-Sutherland Algorithm:** combinació de restes en coma flotant i operacions amb bits.

**Idea:** eliminació de tots els casos possibles sense calcular les interseccions

**PAS 1:** Es començà amb les 4 rectes infinites de les arestes de la finestra de clipping codificació de 9 regions (outcode)



# 4.7. Clipping (mètodes)

- **Definició de codis:** Es defineixen codis per cada punt extrem:

$b_0 b_1 b_2 b_3$

$b_0 = 1$  if  $y > y_{\max}$ , 0 otherwise  
 $b_1 = 1$  if  $y < y_{\min}$ , 0 otherwise  
 $b_2 = 1$  if  $x > x_{\max}$ , 0 otherwise  
 $b_3 = 1$  if  $x < x_{\min}$ , 0 otherwise

1001	1000	1010	$y = y_{\max}$
0001	0000	0010	$y = y_{\min}$
0101	0100	0110	
$x = x_{\min}$	$x = x_{\max}$		

- Els codis divideixen l'espai en 9 regions

# 4.7. Clipping (mètodes)

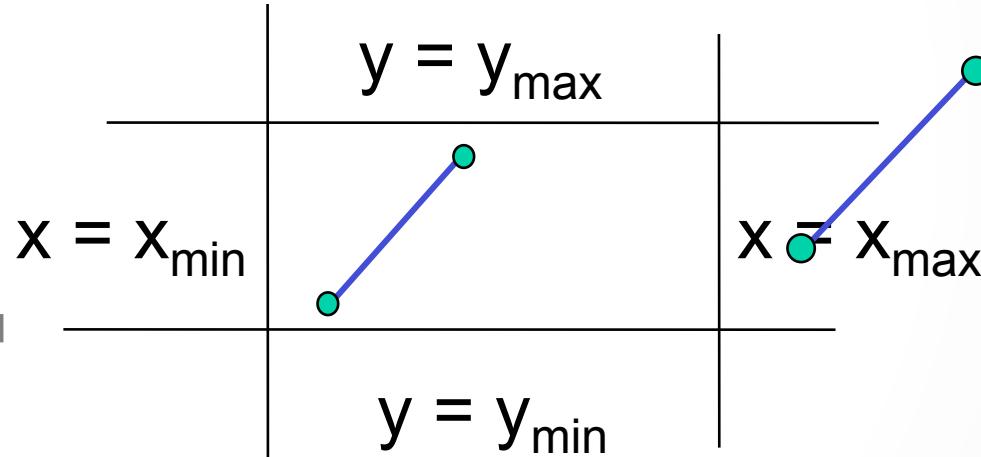
## Algorisme Cohen-Sutherland

**PAS 2:** Es defineixen els diferents casos

( $o_1 = \text{outcode}(x_1, y_1)$ ,  $o_2 = \text{outcode}(x_2, y_2)$ )

**Cas 1:** els dos extrems de la recta son dins de la finestra de clipping ( $o_1 = o_2 = 0$ )

↓  
Es dibuixa la recta



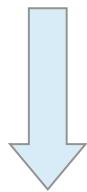
**Cas 2:** els dos extrems de la recta són fora de totes les rectes de la finestra de clipping i pel mateix cantó  
( $o_1 \& o_2 \neq 0$ )



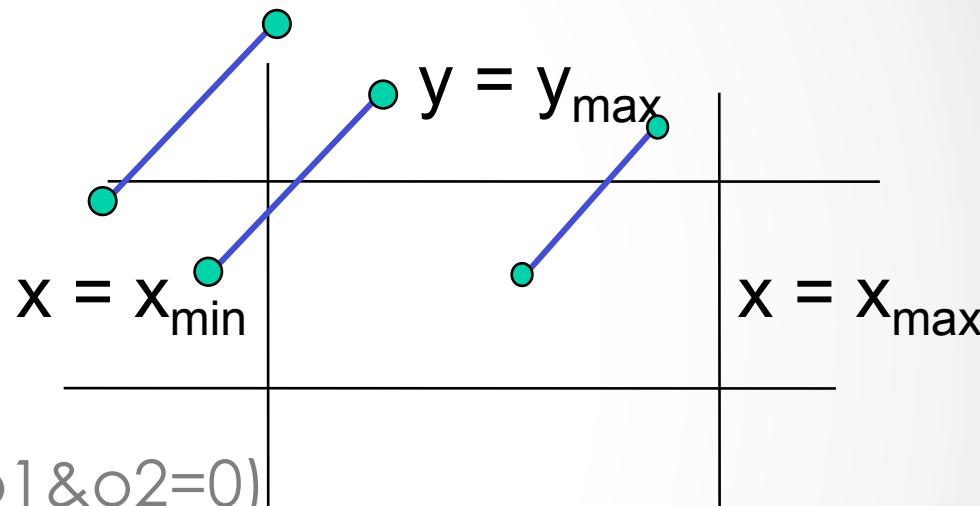
Es descarta la recta

# 4.7. Clipping (mètodes)

**Cas 3:** un dels extrems dins i l'altre fora ( $o_1 \neq 0$  i  $o_2 = 0$  o a l'inrevés)



S'ha de fer almenys una intersecció



**Cas 4:** Els dos extrems són fora ( $o_1 \& o_2 = 0$ )

Pot ser que n'hi hagi una part dins



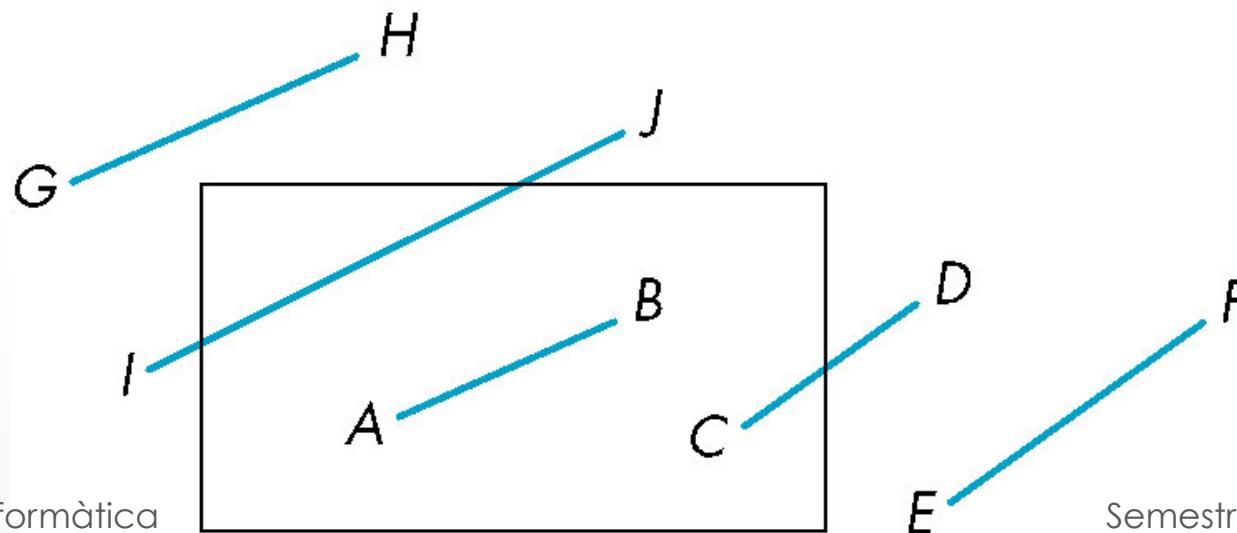
S'ha de calcular almenys una intersecció amb una de les arestes de la window i testejar el outcode del nou punt

# 4.7. Clipping (mètodes)

Si mirem els 5 casos de sota:

- AB: codi(A) = codi(B) = 0    S'accepta el segment de recta
- CD: codi(C) = 0 codi(D)  $\neq 0$

Càlcul de la intersecció. La posició de 1 en el codi(D) determina l'aresta que interseca. En el cas que hi hagués un segment cap a un punt amb el codi amb dos 1's, s'haurien de fer 2 interseccions (cal re-executar l'algorisme)



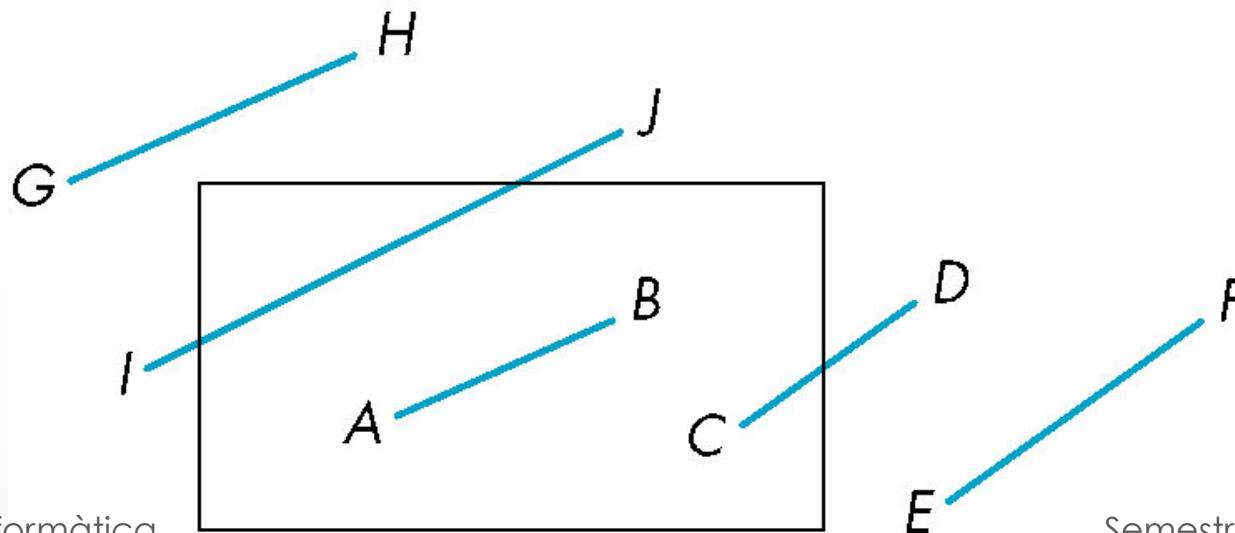
# 4.7. Clipping (mètodes)

- EF:  $\text{codi}(E) \text{ AND codi}(F) \neq 0$

Els dos codis tenen el 1 al mateix lloc i per tant estan al mateix cantó de la recta. Es descarta

- GH i IJ:  $\text{codi}(GH)=\text{codi}(IJ)$

tenen els mateixos codis, no són zeros però el AND dóna zero. Un tros de segment interseca amb una de les rectes de la finestra de clipping. Es calcula el codi de la intersecció (com a nou punt extrem del segment). Cal re-executar l'algorisme.



# 4.7. Clipping (mètodes)

## Algorisme:

- Intersecció horitzontal:  $x = x_0 + m * (y - y_0)$
- Intersecció vertical:  $y = y_0 + (1/m) * (x - x_0)$

CalculaOutCode( $x_0$ ,  $y_0$ , outcode0);  
CalculaOutCode( $x_1$ ,  $y_1$ , outcode1);

## Repeat

Trivial = ComprovaCasTrivial

Amb el punt que està fora de la window

**if TOP then**

$x = x_0 + (x_1 - x_0) * (ymax - y_0) / (y_1 - y_0);$

$y = ymax;$

**else if BOTTOM then**

$x = x_0 + (x_1 - x_0) * (ymin - y_0) / (y_1 - y_0);$

$y = ymin;$

**else if RIGHT then**

$y = y_0 + (y_1 - y_0) * (xmax - x_0) / (x_1 - x_0);$   
 $x = xmax;$

**else if LEFT then**

$y = y_0 + (y_1 - y_0) * (xmin - x_0) / (x_1 - x_0);$   
 $x = xmin;$

**if ( $x_0, y_0$  is the outer point) then**

$x_0 = x; y_0 = y;$

CalculaOutCode( $x_0$ ,  $y_0$ , outcode0)

**else**

$x_1 = x; y_1 = y;$

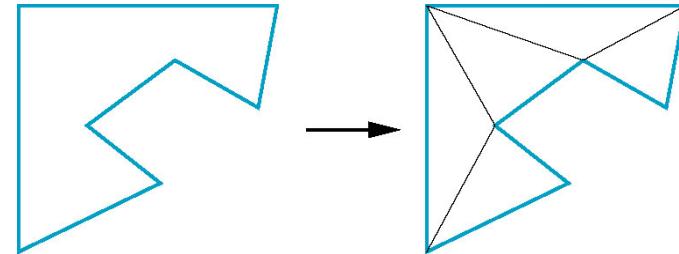
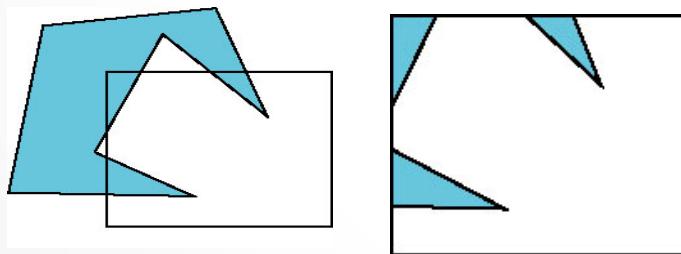
CalculaOutCode( $x_1$ ,  $y_1$ , outcode1)

**until** Trivial

# 4.7. Clipping (mètodes)

## Clipping de polígons:

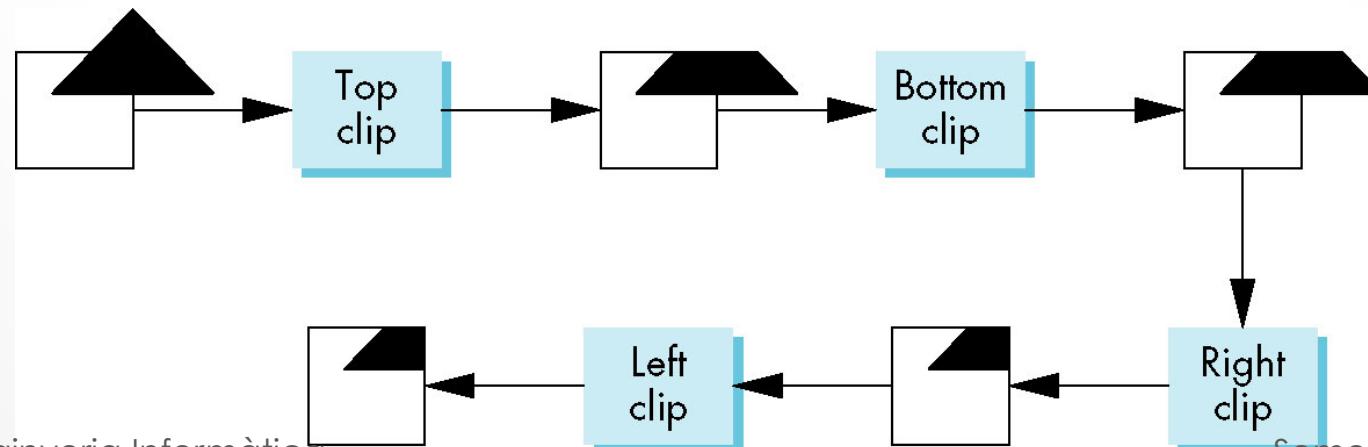
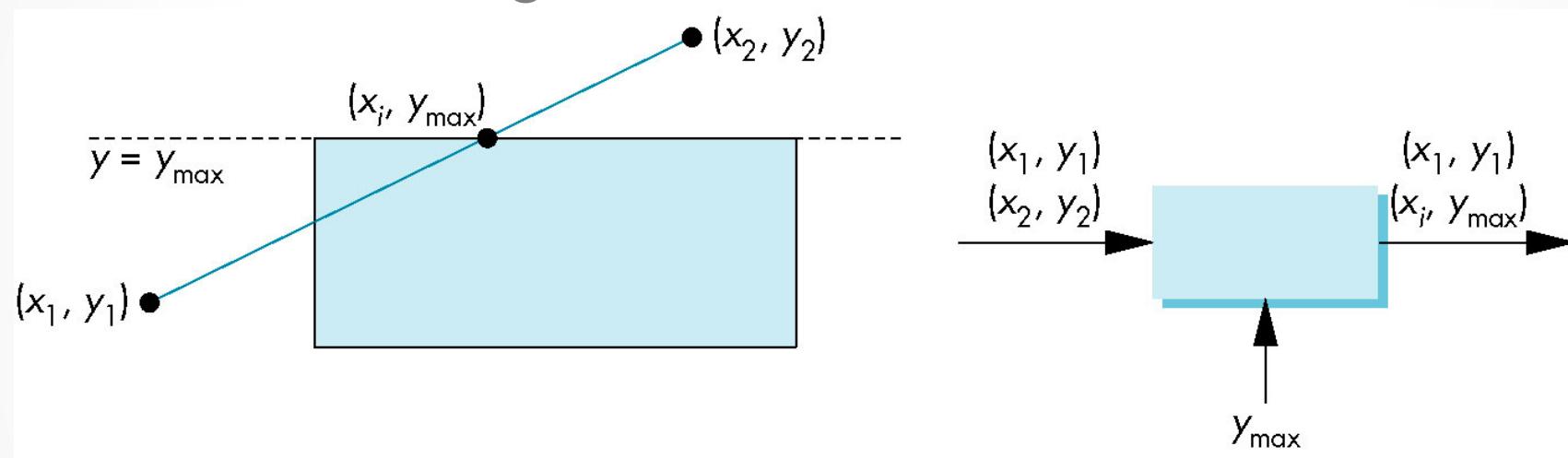
- No és tan simple com el clipping de segments de rectes
- El clipping d'un polígon pot definir diferents polígons
- No obstant, el clipping d'un polígon convex dóna com a molt un altre polígon.
- Una possibilitat és dividir el polígon còncau en un conjunt de polígons convexos (es pot fer en la GPU)



# 4.7. Clipping (mètodes)

**Clipping de polígons:** (Sutherland and Hodgeman)

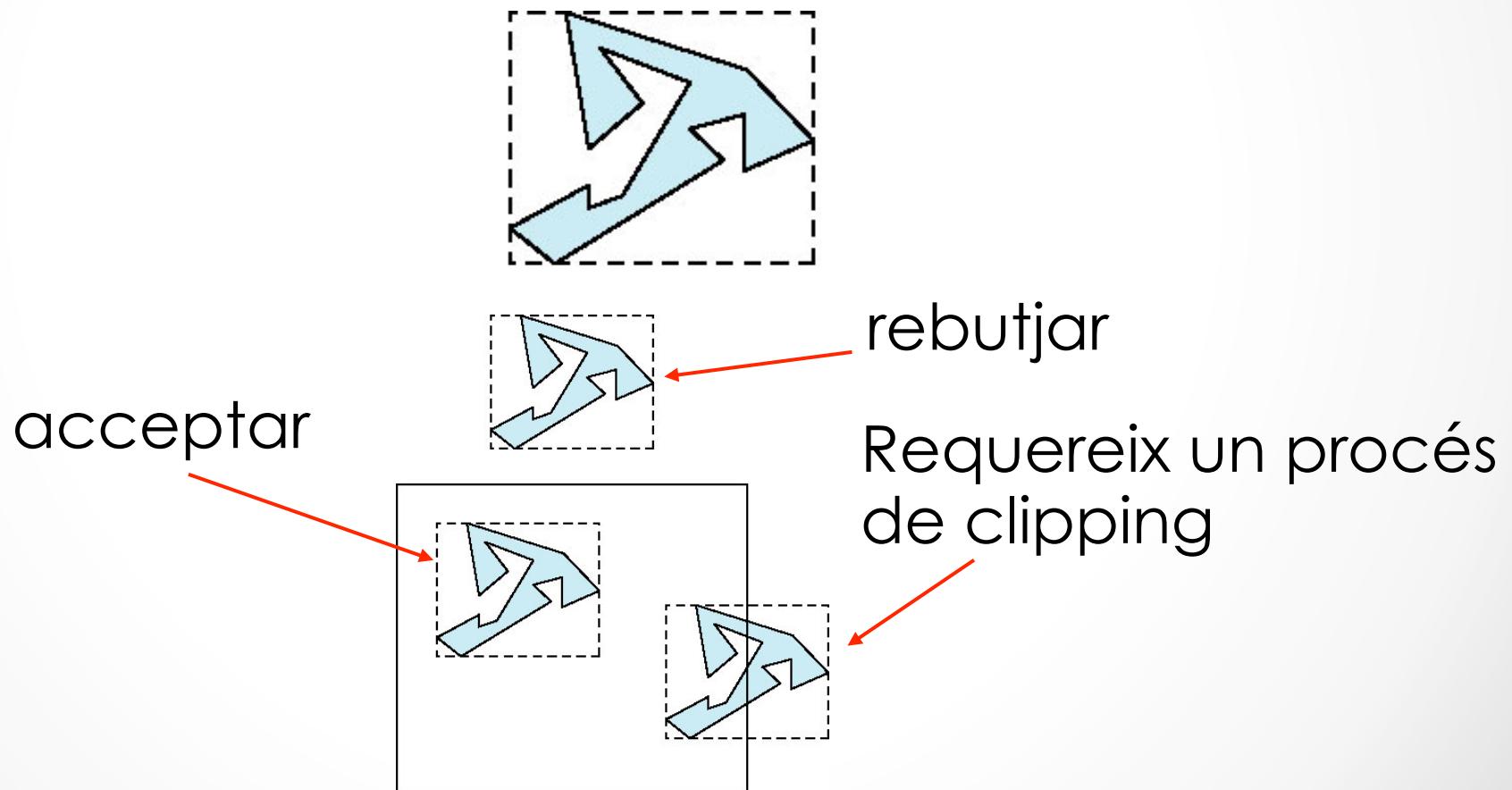
es pot tenir en compte cada aresta per separat de la window com un segment de recta



# 4.7. Clipping (mètodes)

## Capses mínimes contenidores (bounding box):

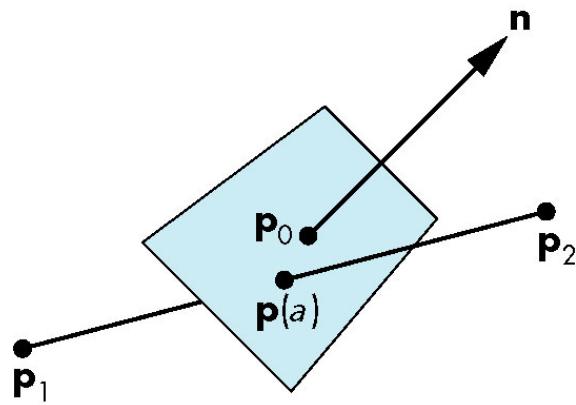
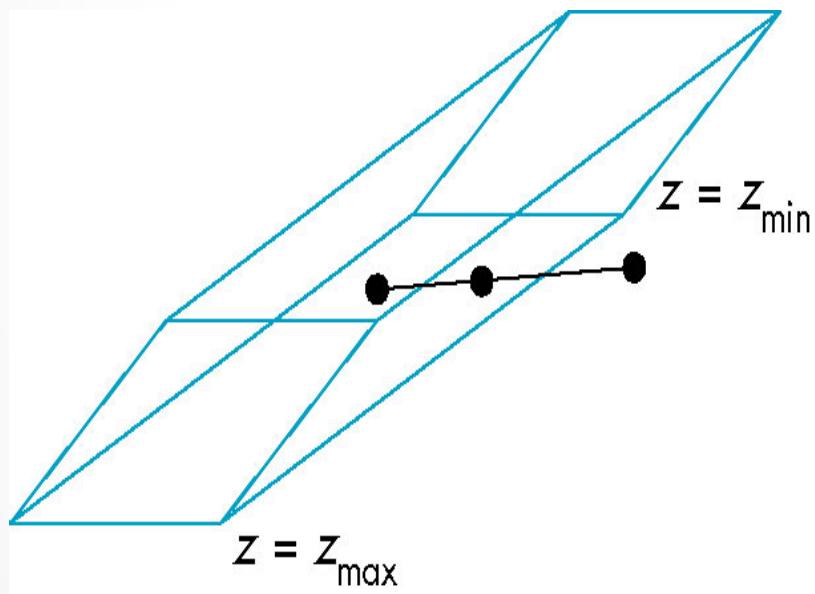
En un preprocés es poden descartar els polígons que són totalment fora o totalment dins per evitar fer-los el clipping



# 4.7. Clipping (mètodes)

## Clipping 3D:

Un clipping general en 3D requereix interseccions de segments de recta contra plans arbitraris

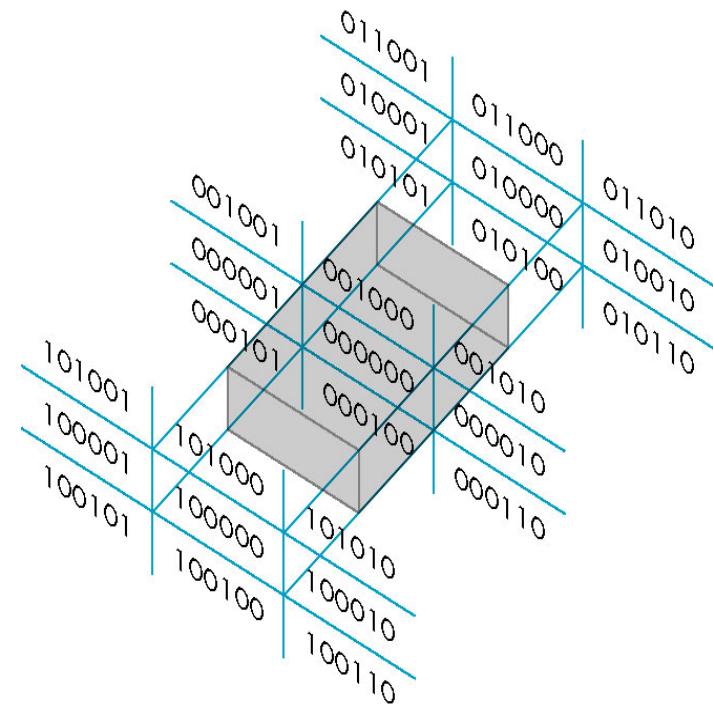
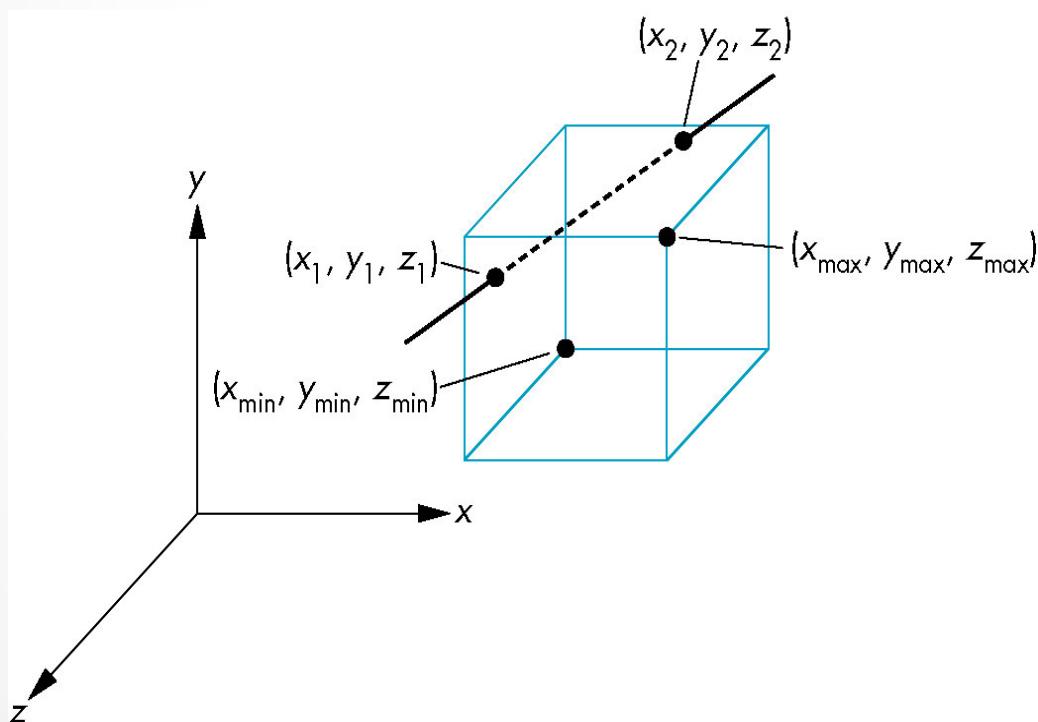


$$\alpha = \frac{n \cdot (p_o - p_1)}{n \cdot (p_2 - p_1)}$$

# 4.7. Clipping (mètodes)

## Algorisme de Cohen-Sutherland 3D:

- Es defineixen codis de 6 bits.
- Quan es calcula, cal interseccar el segment de recta amb un pla



# 4.7. Clipping (mètodes)

La **normalització** ens permet interseccar sempre amb plans d'un paral·lelepípede.

- Llavors, la intersecció només es fa fent una divisió en punt flotant, per exemple

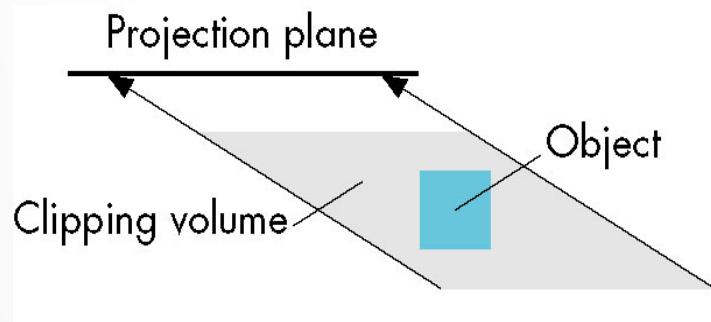
if TOP then

$$x = x_0 + (x_1 - x_0) * (y_{max} - y_0) / (y_1 - y_0);$$

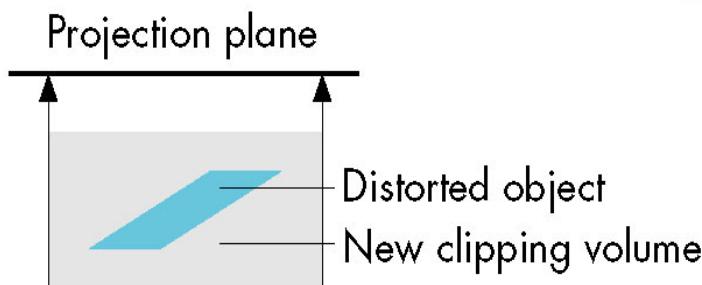
$$z = z_0 + (z_1 - z_0) * (y_{max} - y_0) / (y_1 - y_0);$$

$$y = y_{max};$$

top view



Abans de normalitzar



Després de normalitzar

# Índex

4.1. Introducció

4.2. Viewing: Projeccions

4.3. Volum de visió: definició del clipping

4.4. Normalització

4.5. Implementació en OpenGL

4.6. Implementació en GPUs

4.7. Clipping: mètodes

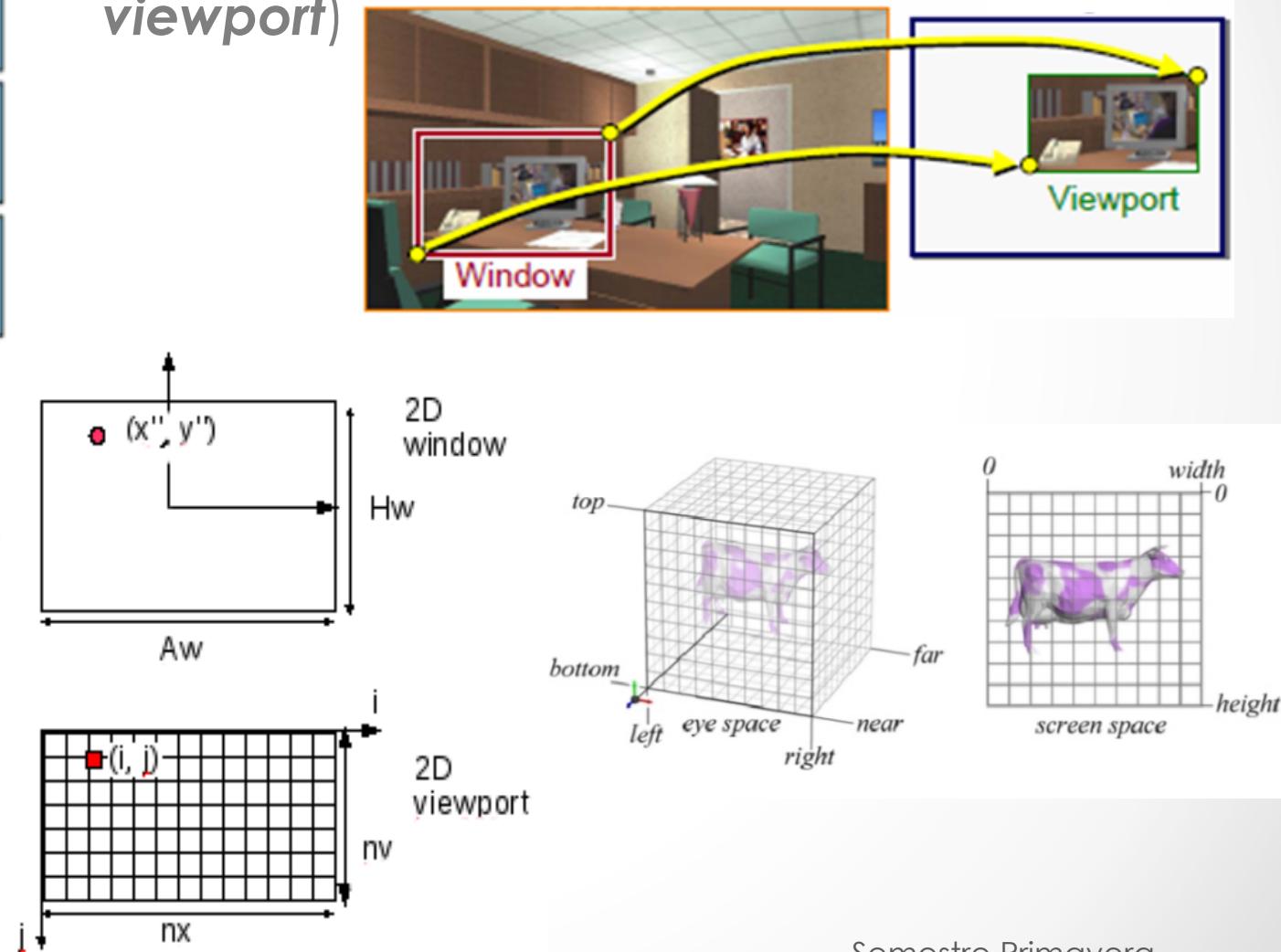
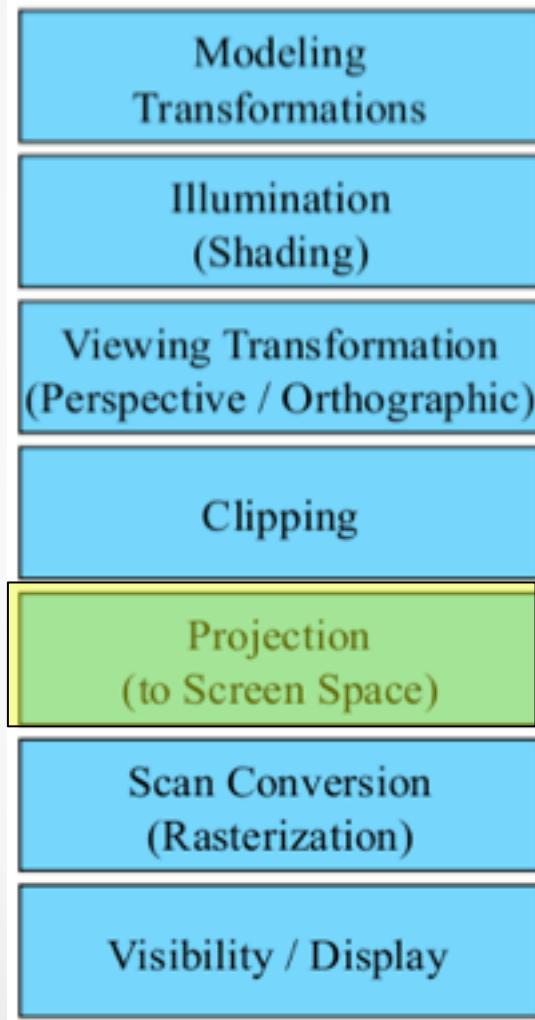
4.8. Rasterització

4.9. Eliminació de parts amagades

4.10. Implementació en OpenGL i en GPUs

# Pipeline de visualització

- Es projecta cada vèrtex de l'espai 2D continu (**window**) al frame buffer (o **viewport**)



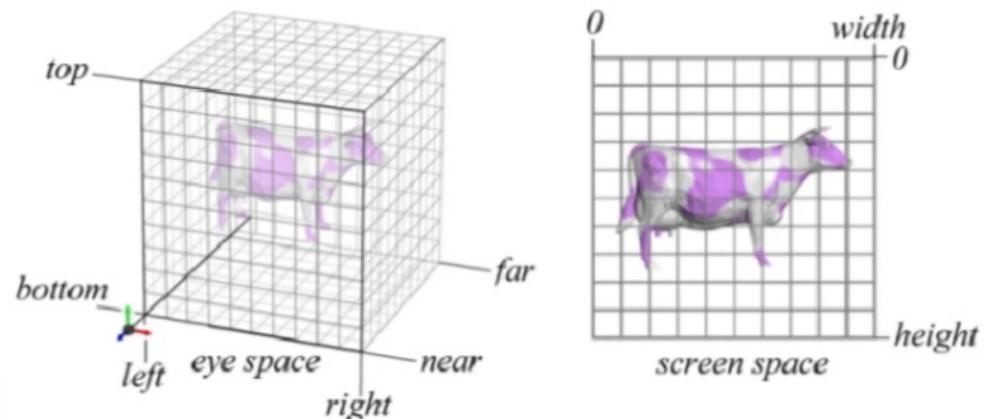
# 4.8. Rasterització

Les projeccions obtenen coordenades normalitzades pertanyents a  $\mathbb{R}^4$ :

- la capsà mínima contenidora 2D d'aquests punts transformats en el pla de projecció s'anomena **window**.

Aquestes coordenades es corresponen a coordenades de pantalla discretes:

- el **viewport** és l'àrea de la pantalla que ocuparà la visualització final.

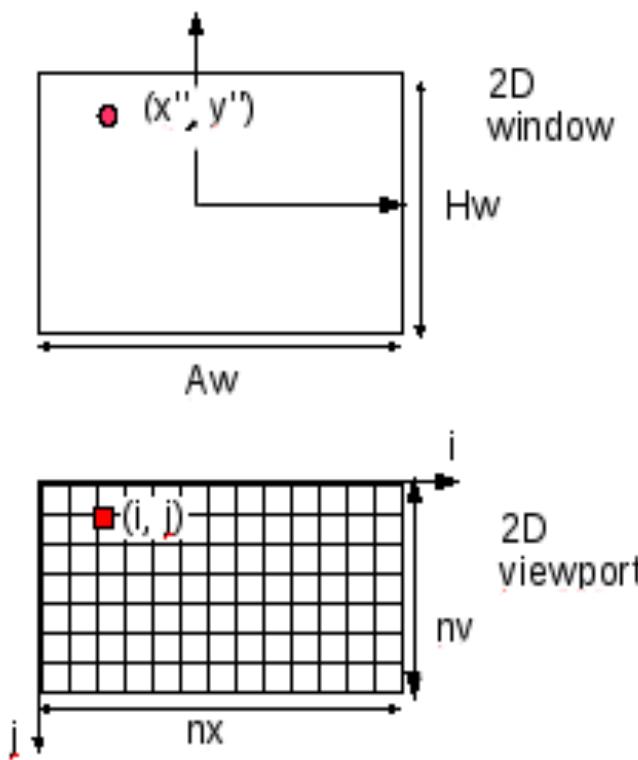


# 4.8. Rasterització

Per a realizar la conversió entre coordenades de window ( $x'', y'', 1$ ) i coordenades de viewport ( $i, j$ ), podem trobar la relació següent:

$$\frac{i}{nx} = \frac{0.5 Aw + x''}{Aw}$$

$$\frac{j}{ny} = \frac{0.5 Hw - y''}{Hw}$$



# 4.8. Rasterització

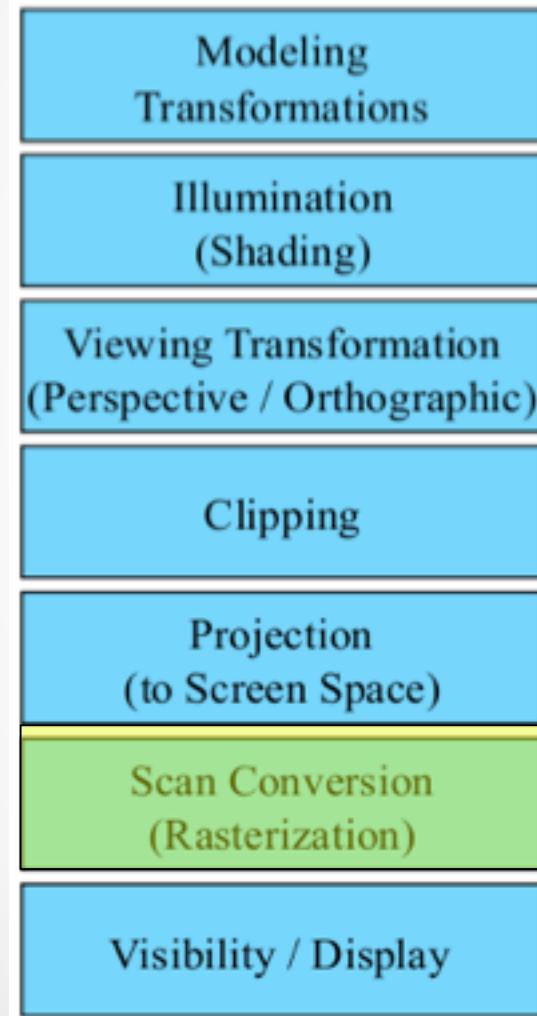
Aïllant les variables  $i, j$ , obtenim les següents expressions que es poden posar en forma matricial, amb un escalat i una translació (transformació window-viewport):

$$i = \frac{nx}{Aw} * x'' + nx * 0.5$$

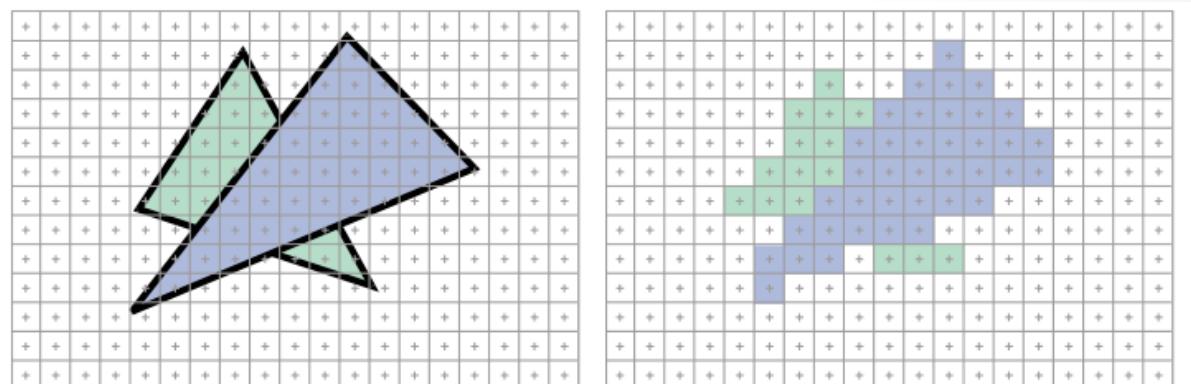
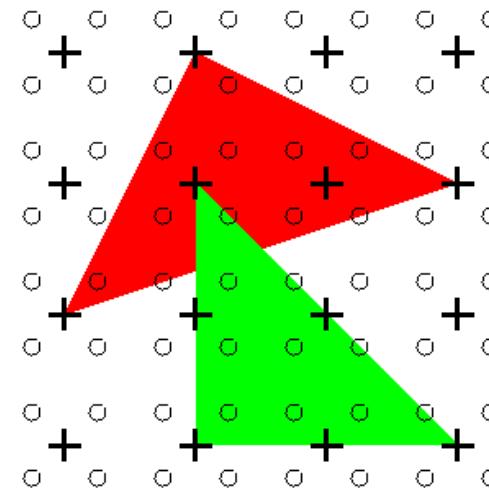
$$j = \frac{-ny}{Hw} * y'' + ny * 0.5$$

$$\mathbf{T}_{\text{window-viewport}} = \mathbf{T}_{xy} \cdot \mathbf{S}_{xy} = \begin{bmatrix} \frac{nx}{Aw} & 0 & 0 \\ 0 & -\frac{ny}{Hw} & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & nx * 0.5 \\ 0 & 1 & ny * 0.5 \\ 0 & 0 & 0 \end{bmatrix}$$

# Pipeline de visualització



- Es rasteritzen les àrees definides pels punts projectats (**scan conversion**)



# 4.8. Rasterització

Els algorismes implicats en la discretització en píxels o de **rasterització**:

- Determinen quins píxels són dins de la primitiva definida pel conjunt de vèrtexs
- Produeixen un conjunt de fragments
- Els fragments tenen una posició (*pixel location*) i altres atributs com el color o les coordenades de textures, que es determinen interpolant els valors dels vèrtexs
- Els colors dels píxels es determinen després usant color, textures i altres propietats dels vèrtexs (ho veurem en el tema 5)

# 4.8. Rasterització

El procés de rasterització implica:

- 1. un procés de discretització (**scan conversion**):  
Existeixen diferents mètodes:
  - DDA
  - Algorisme de Bresenham
- 2. un **omplert de polígons**:
  - Punt interior a un polígon
  - Germen
- 3. un procés d'**antialiasing** posterior

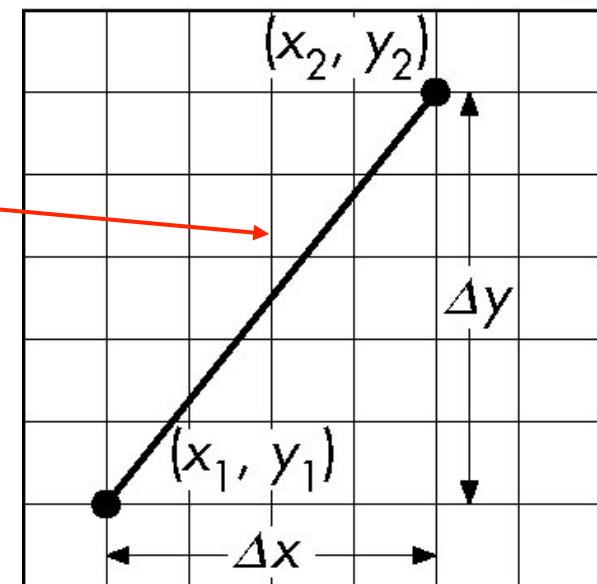
# 4.8. Rasterització

## Scan conversion d'un segment de recta:

- Es comença amb el segment de recta en coordenades de window amb valors enters en els seus extrems
- Suposem que es té la funció write\_pixel ja implementada.

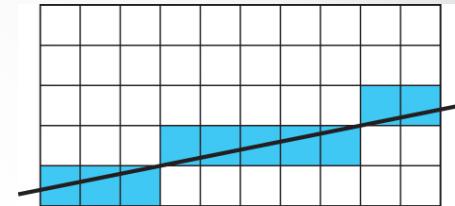
$$m = \frac{\Delta y}{\Delta x}$$

$$y = mx + h$$



# 4.8. Rasterització

## Digital Differential Algorithm (DDA):

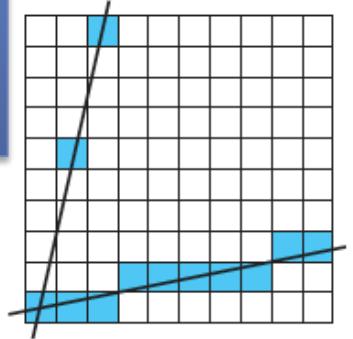


- La recta  $y = mx + b$  satisfà l'equació diferencial:

$$\frac{dy}{dx} = m = \Delta y / \Delta x = (y_2 - y_1) / (x_2 - x_1)$$

- Si explorem la línia de scan  $\Delta x = 1$

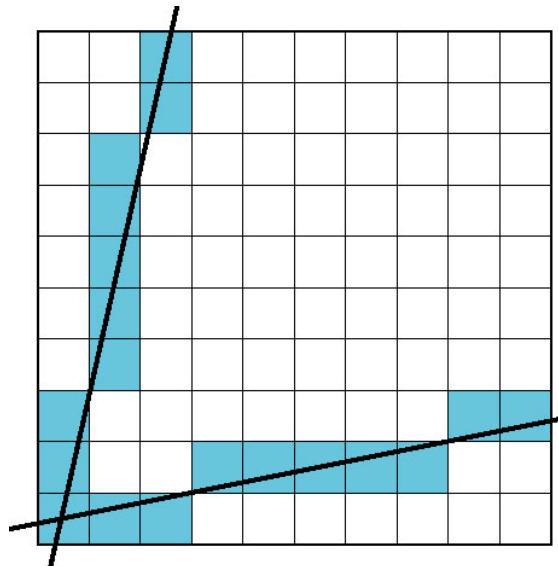
```
for(x=x1; x<=x2, ix++) {  
    y+=m;  
    write_pixel(x, round(y), line_color)  
}
```



- Problema amb rectes molt pENDENTS (amb y's molt separades per x molt properes)

# 4.8. Rasterització

**Solució:** S'utilitza per  $1 \geq m \geq 0$  i per  $m > 1$ , es canvia el paper de x per y:



Algorisme que utilitza operacions a coma flotant.

- S'utilitza l'alternativa de **Bresenham** que només usa enters

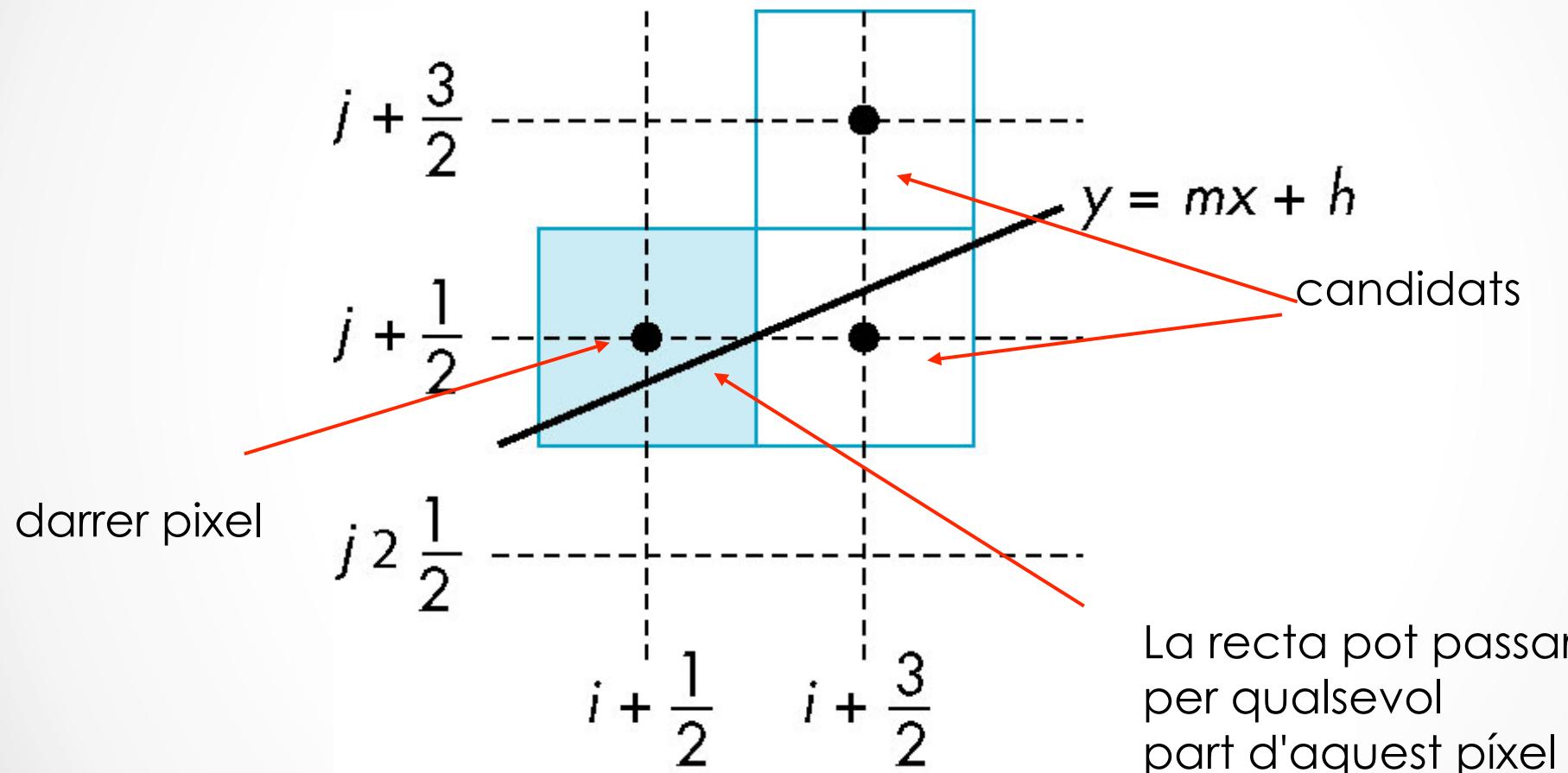
# 4.8. Rasterització

## Algorisme de Bresenham:

- considerem només  $1 \geq m \geq 0$   
La resta de casos es calculen per simetria
- es suposa que els centres dels píxels estan situats en les posicions 0.5
- Si es comença per un píxel, només hi han dos possibles candidats del següent píxel.

# 4.8. Rasterització

$$1 \geq m \geq 0$$



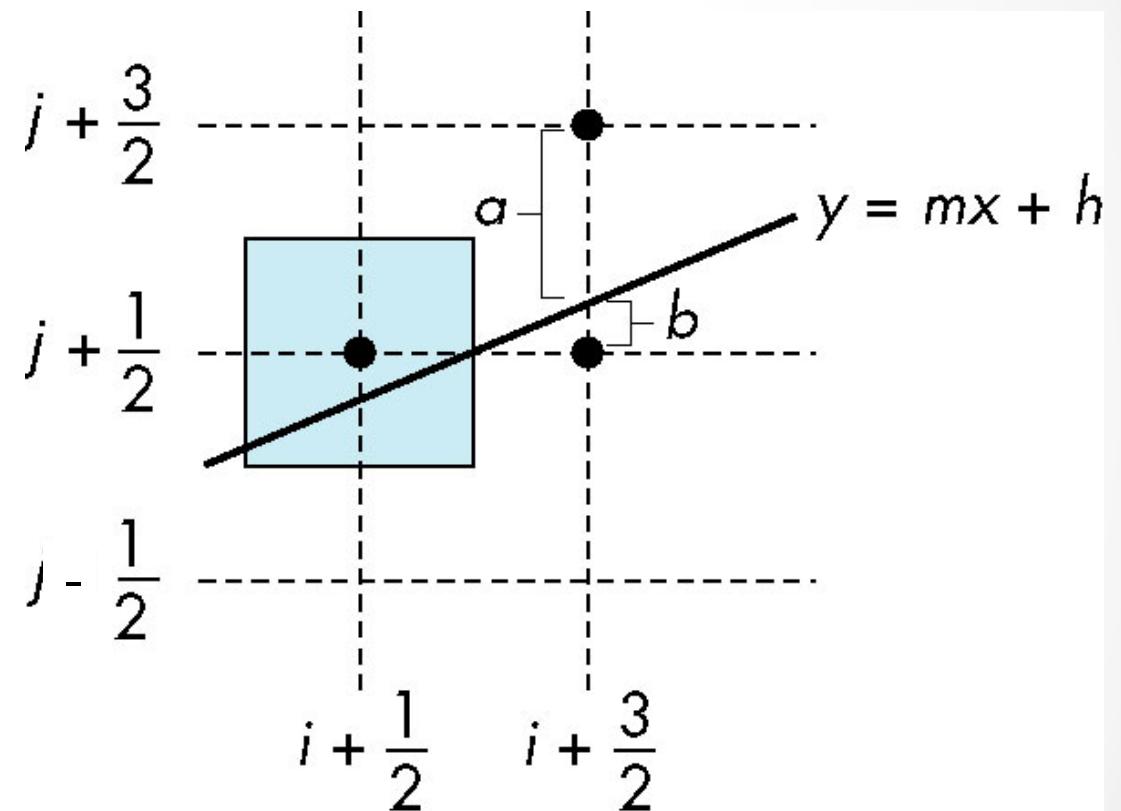
# 4.8. Rasterització

$$d = \Delta x(b-a)$$

$d$  és un enter

$d > 0$  s'agafa el píxel de dalt

$d < 0$  s'agafa el píxel d'abaix



# 4.8. Rasterització

## Forma incremental:

És més eficient si fixem en  $d_k$ , el valor de la variable de decisió a  $x = k$

$$d_{k+1} = d_k - 2\Delta y, \text{ if } d_k < 0$$

$$d_{k+1} = d_k - 2(\Delta y - \Delta x), \text{ en un altre cas}$$

- Per a cada  $x$ , només es necessita una suma entera i un test
- Es una instrucció senzilla en el xip de la targeta gràfica

[http://www.cs.bham.ac.uk/~vjk201/Teach/Graphics/  
Bresenham\\_derivation.pdf](http://www.cs.bham.ac.uk/~vjk201/Teach/Graphics/Bresenham_derivation.pdf)

# 4.8. Rasterització

## Algorisme de Bresenham:

Punts extrems de la recta a discretitzar:  $(x_0, y_0)$  i  $(x_n, y_n)$

$$\Delta x = x_n - x_0; \Delta y = y_n - y_0;$$

$$p_1 = 2 \Delta y - \Delta x;$$

`write_pixel(x0, y0)`

**repetir** fins a arribar a  $(x_n, y_n)$  :

if  $p_i < 0$

`write_pixel (xi +1, yi )`

$p_{i+1} = p_i + 2 \Delta y$

if  $p_i \geq 0$

`write_pixel (xi +1, yi + 1 )`

$p_{i+1} = p_i + 2(\Delta y - \Delta x)$

# 4.8. Rasterització

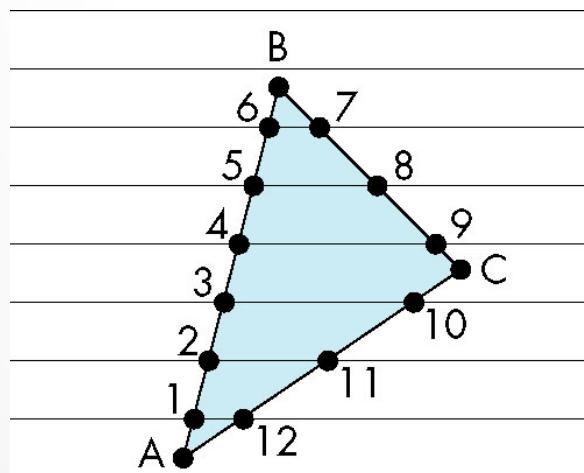
## Rasterització de polígons: Omplert de polígons

- El test més fàcil és preguntar a cada píxel, si és dins o fora del polígon.
- Com es fa per saber si un píxel és dins o fora?
  - En els casos de polígons **convexes** és més fàcil
  - Es fa el test de nombre d'interseccions senars
  - Es contén el nombre d'interseccions de lesarestes
  - En casos de polígons **còncaus**, una possible solució és subdividir-los en un conjunt de polígons convexes.

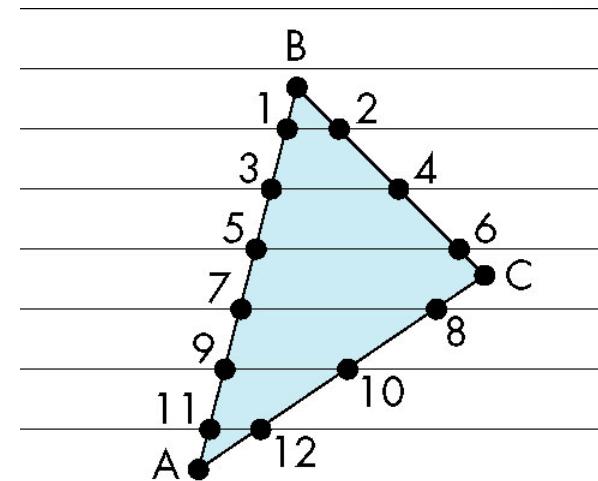
# Omplert de polígons

Hi ha diferents mètodes: flood-fill, scan-line.

- **Scan-line:** Es processa línia a línia del frame buffer
  - S'omple cada scan-line
  - Problema de l'ordre d'obtenció dels scan-lines ( $O(n \log n)$ , amb  $n$  el nombre de interseccions).

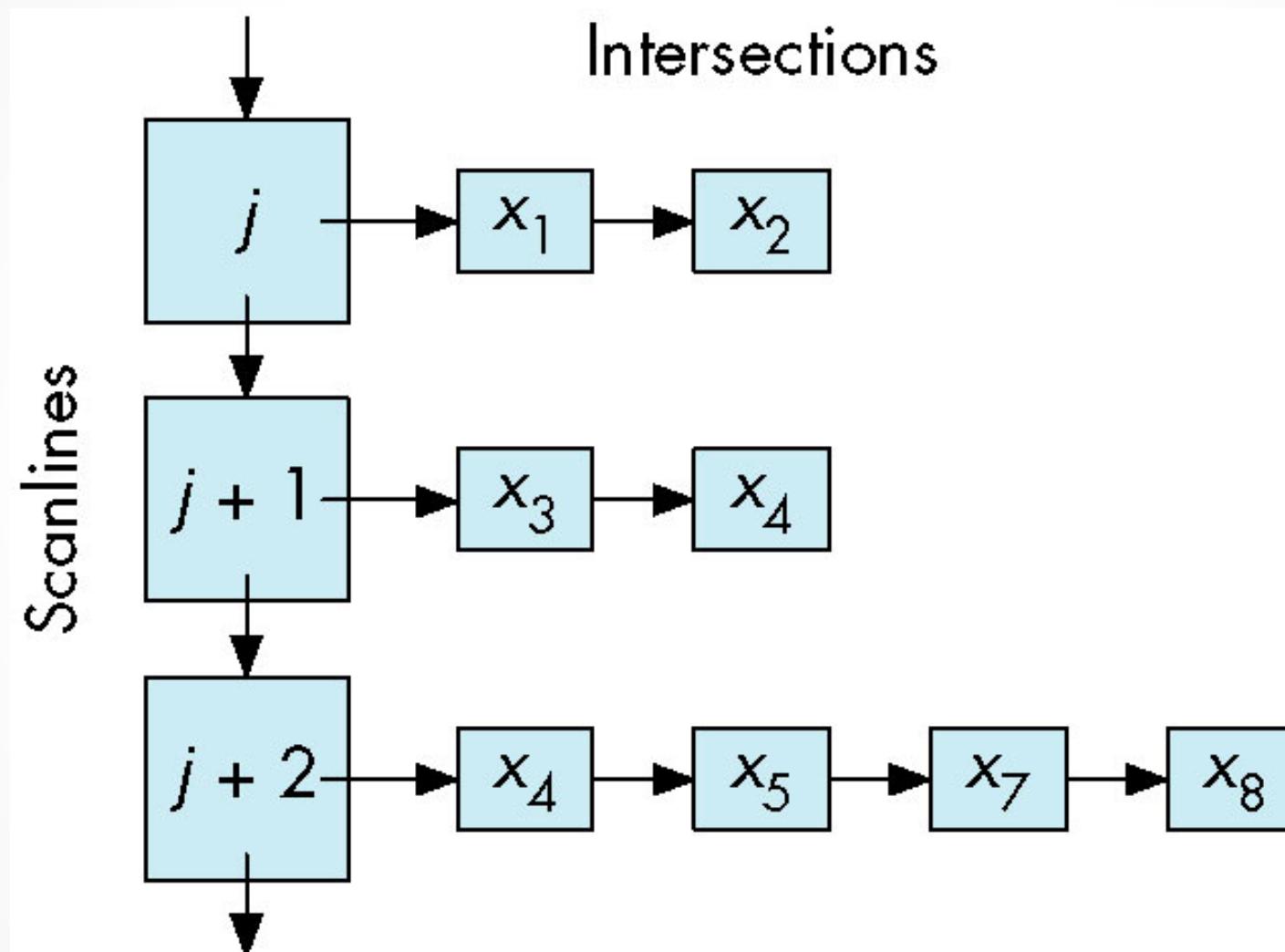


Ordre inicial



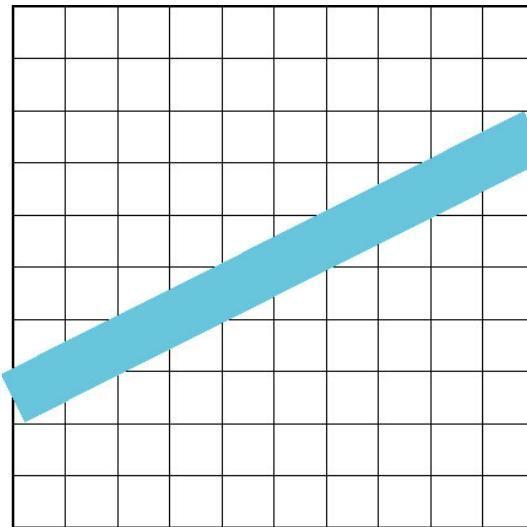
Ordre desitjat

# Omplert de polígons



# Aliasing

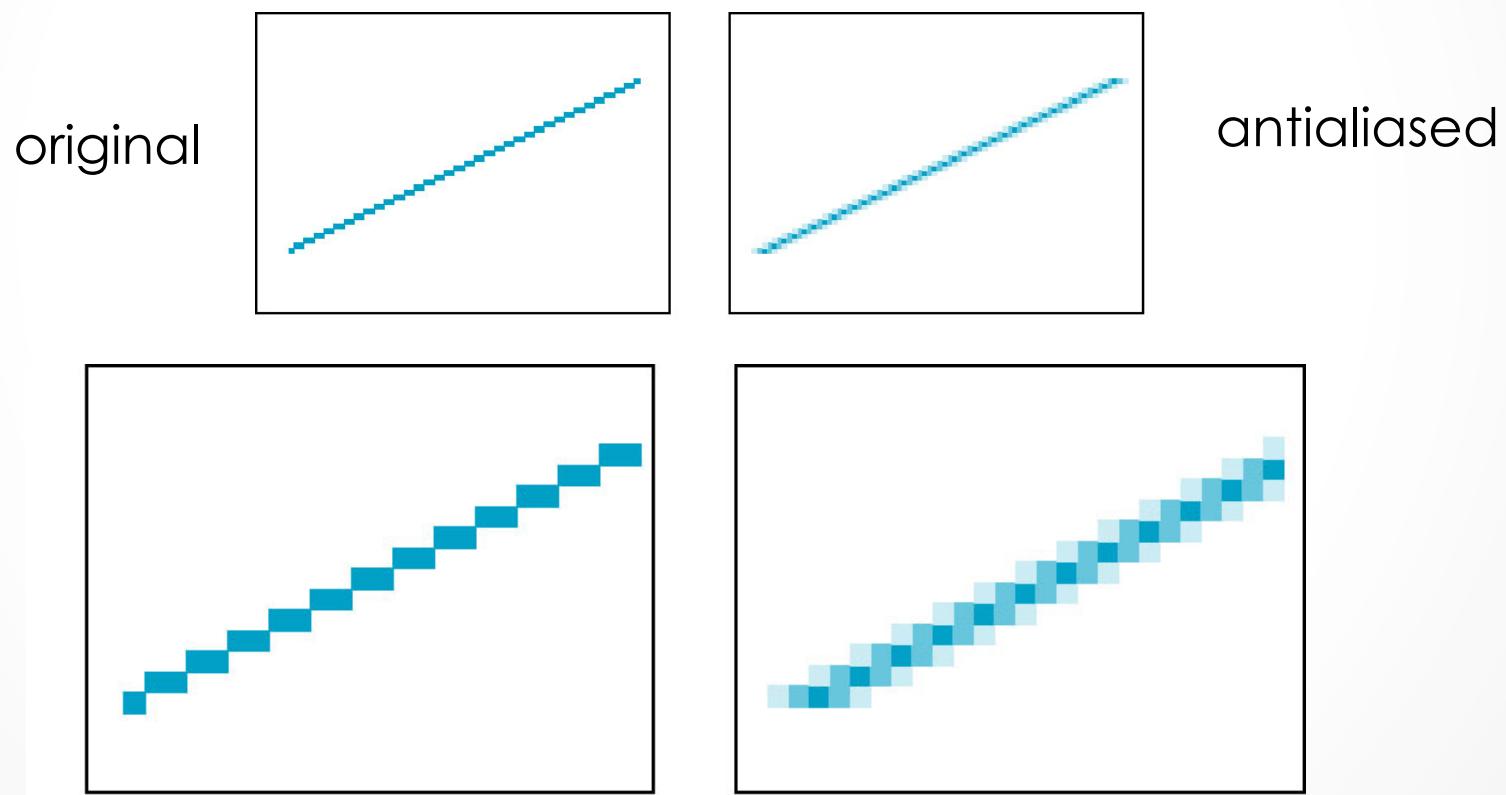
- La discretització d'una recta idealment seria una franja d'amplada d'un píxel



- El fet d'escollir x o y, segons el millor candidat produceix efectes d'aliasing

# Aliasing

S'escullen múltiples pixels en la mateix x i es fa un difuminat del color per a produir l'efecte de suavitzat o d'antialiasing

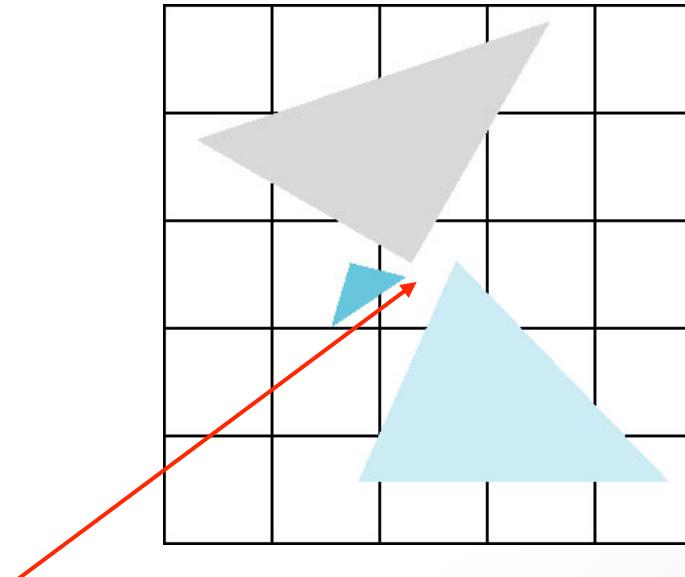


Zoom de les dues imatges anteriors

# Aliasing

El problema d'aliasing en polígons s'agreuja:

- Es poden negligir petits polígons
- Es junten arestes
- Es necessita compondre el color de diferents polígons en un únic píxel.



Tots tres polígons han de contribuir en el color del píxel final

# Conclusions

- En aquestes transparències hem après les principals tècniques de clipping i de rasterització utilitzades en aplicacions gràfiques.
- Hem après els avantatges i els inconvenients dels principals algorismes
- Hem après tècniques d'acceleració
- S'ha vist en quina part del procés en la GPU s'aplica