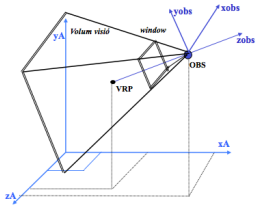
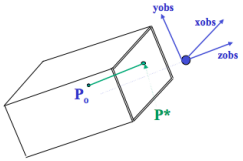
# Classe 6: contingut

- Realisme: Eliminació de parts ocultes
  - Back-face culling
  - Depth-buffer
- Realisme: Il·luminació (1)
  - Càlcul del color en un punt
- Exercicis càmera

# Càmera i procés de visualització




Càmera (posició i òptica) i interacció

EPA i Il·luminació

*viewMatrix, projectMatrix*

viewport

ModelMatrix$_i$ (TG)

**PintaModel$_i$**

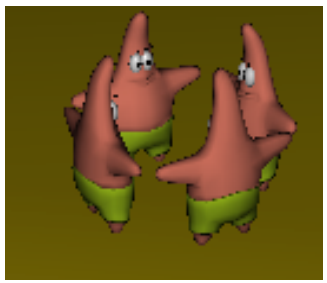
VAO$_i$

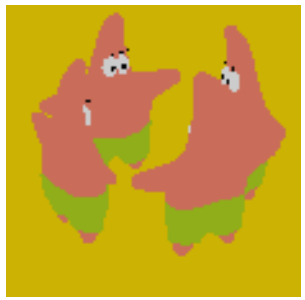
**Vertex Shader**

OpenGL-Visualització

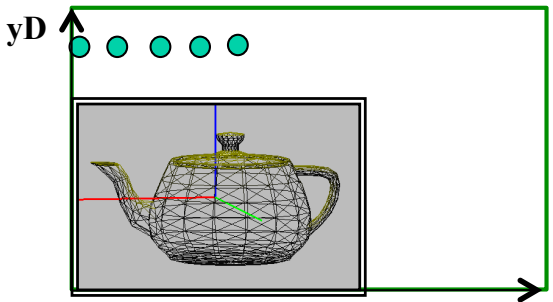
**Fragment Shader**

Píxel(x,y,c)





Finestra OpenGL (Pantalla)

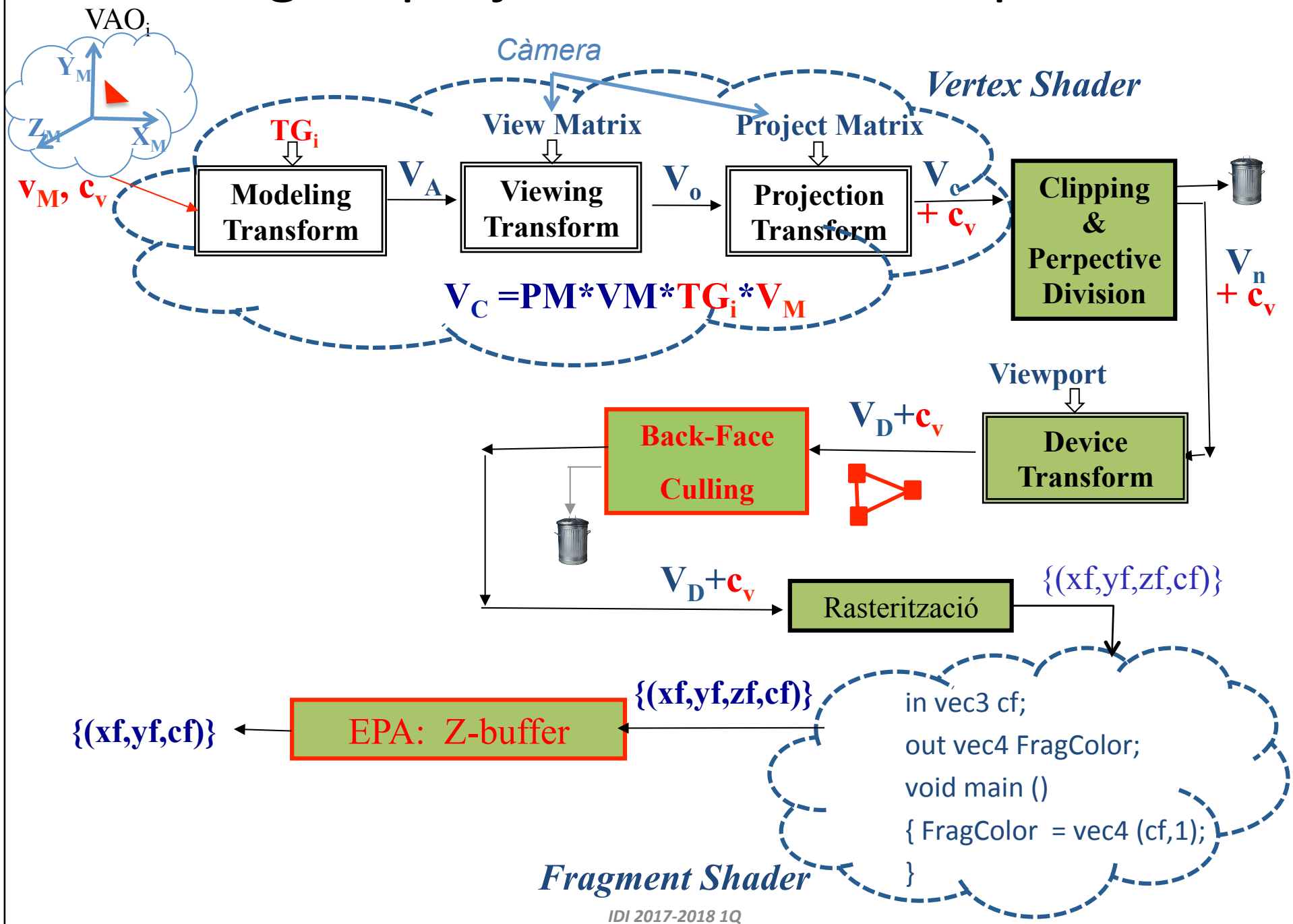



Viewport/Vista

# Classe 6: contingut

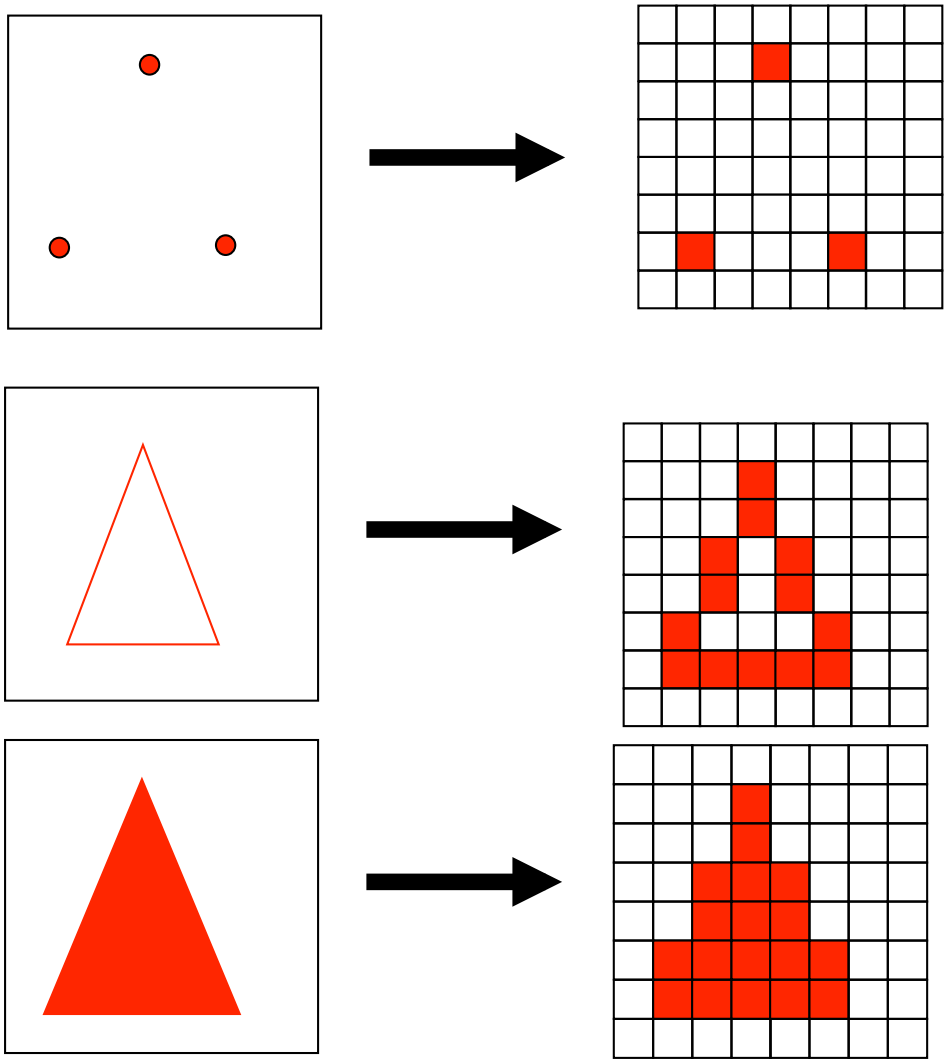
- **Realisme: Eliminació de parts ocultes**
  - Depth-buffer
  - Back-face culling
- Realisme: Il·luminació (1)
  - Càlcul del color en un punt
- Exercicis càmera

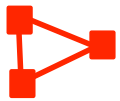
# Paradigma projectiu bàsic amb OpenGL 3.3



# Algorismes de rasterització

La discretització és diferent per a cada primitiva: punt, segment, polígon





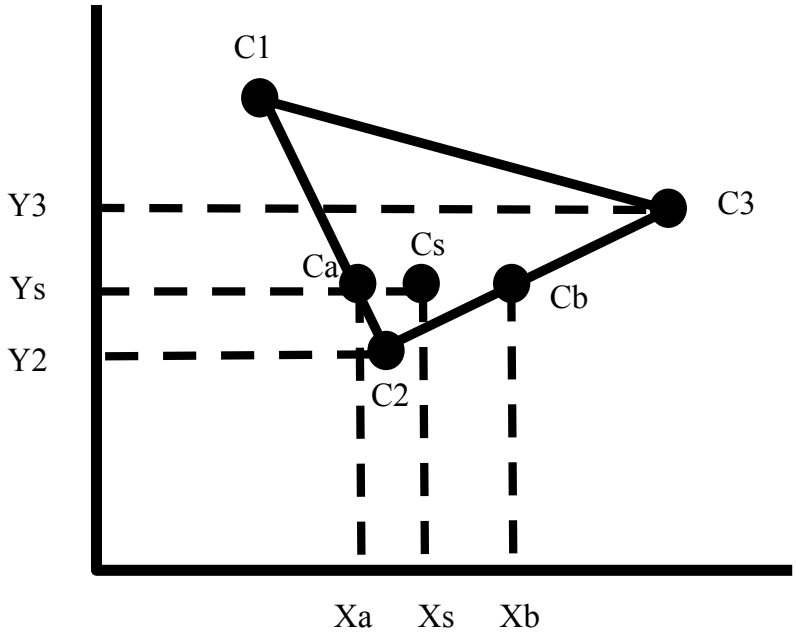
$V_D = (x_D, y_D, z_D, cv)$

Rasterització

{(xf,yf,zf,cf)}

# Shading (colorat) de polígons

- Colorat Constant ≡ Flat *shading* ➔ $C_f = C1$
  *color uniforme per tot el polígon (funció del color calculat en un vèrtex); cada cara pot tenir diferent color.*
- Colorat de Gouraud ≡ *Gouraud shading* ≡ *Smooth shading*

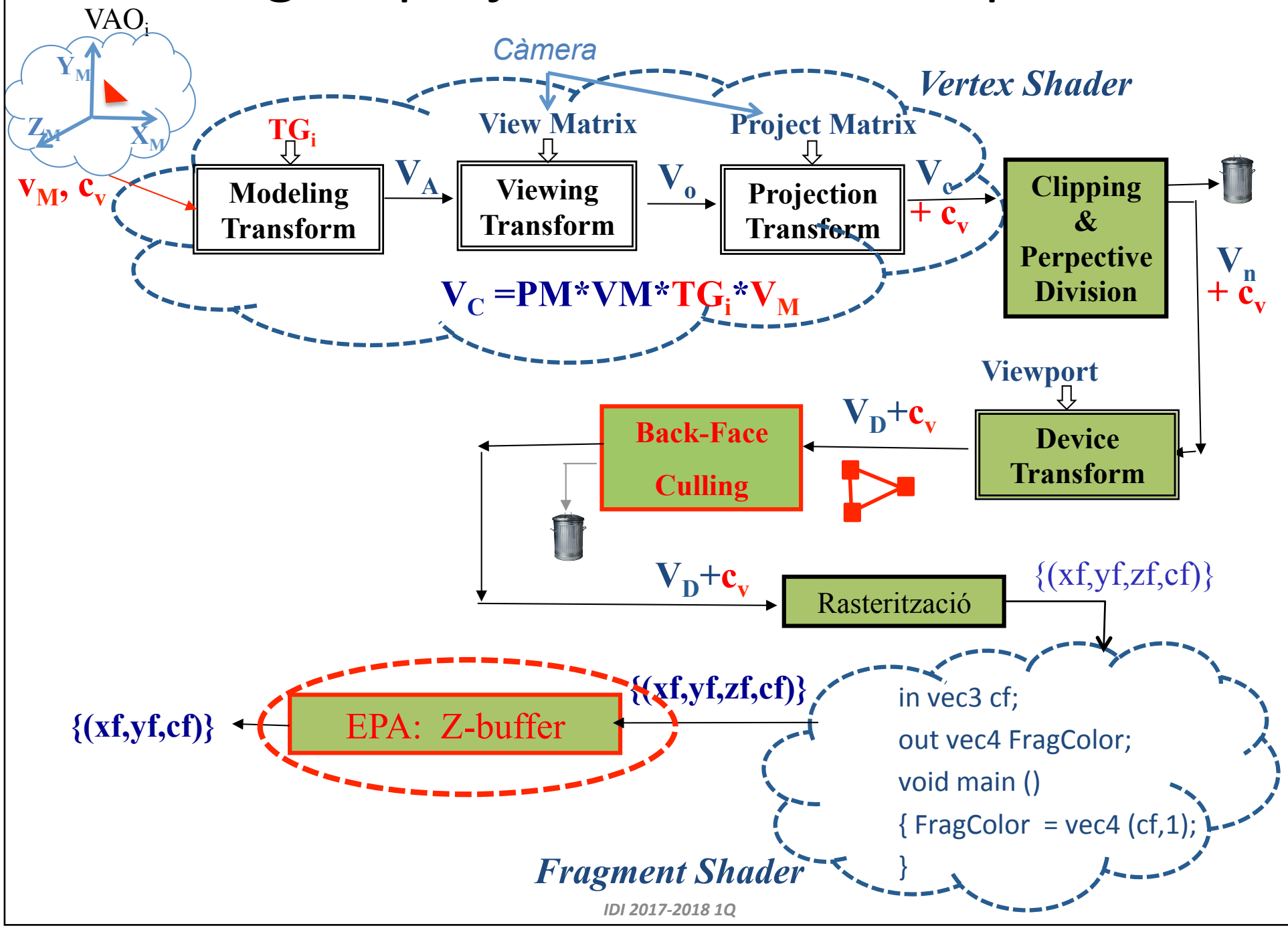



$$Ca = \frac{1}{Y1 - Y2}(C1(Ys - Y2) + C2(Y1 - Ys))$$

$$Cb = \frac{1}{Y3 - Y2}(C2(Y3 - Ys) + C3(Ys - Y2))$$

$$Cs = \frac{1}{Xb - Xa}(Ca(Xb - Xs) + Cb(Xs - Xa))$$

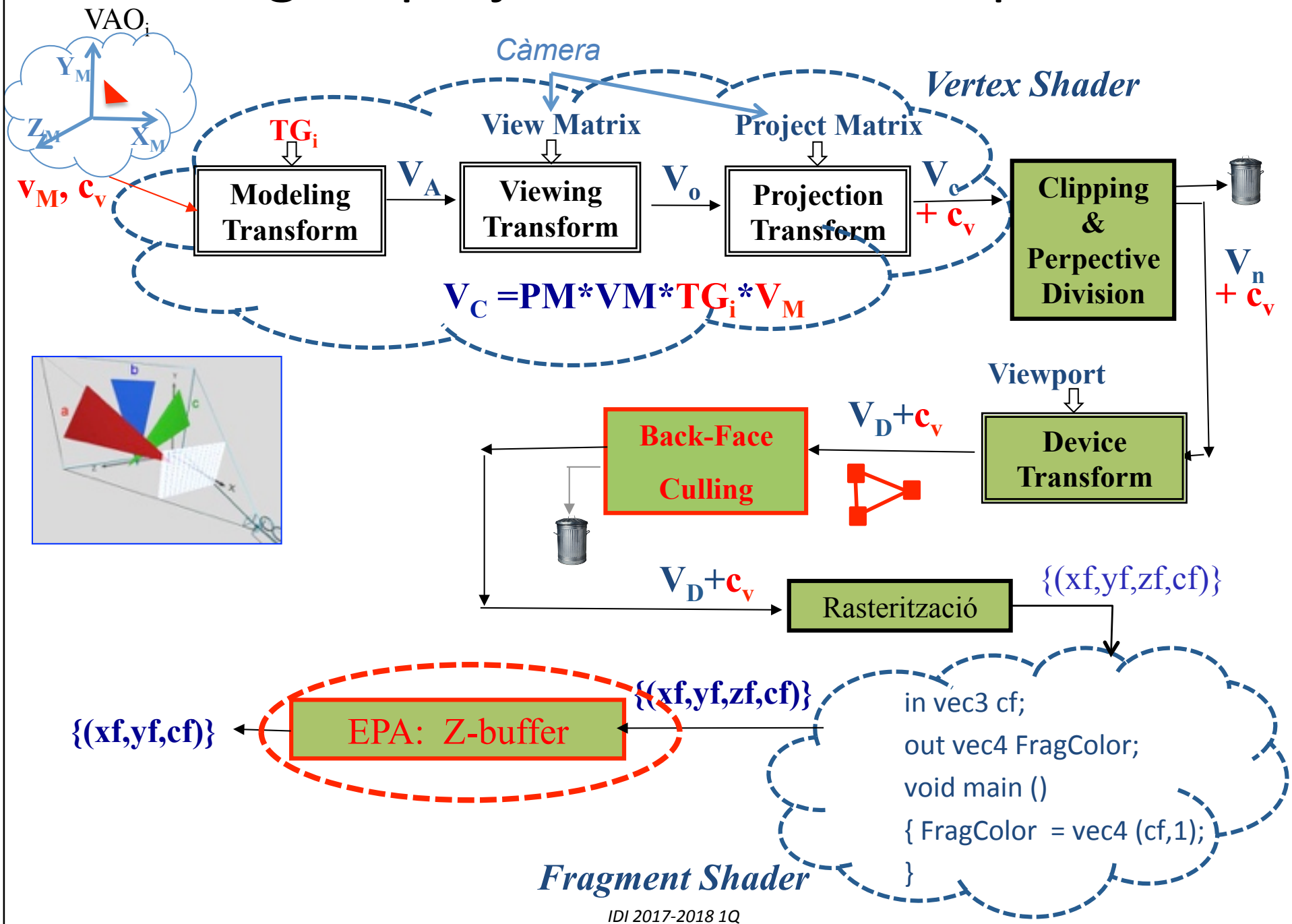
# Paradigma projectiu bàsic amb OpenGL 3.3



# Depth Buffer

- Mètode EPA en espai imatge *(a nivell de píxel/fragment)*
- Després de la **rasterització i del Fragment Shader**
- Requereix conèixer per a cada píxel, un valor (depth) que sigui proporcional a la distància a l'observador a la que es troba el polígon que es projecta en el píxel.
- No importa ordre en que s'enviïn a pintar els triangles
- No requereix tenir el Back-face culling activat

# Paradigma projectiu bàsic amb OpenGL 3.3



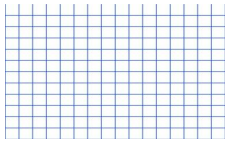
# Depth Buffer (z-buffer)

- Dos buffers de la mateixa resolució que la pantalla

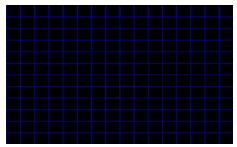
  Buffer color (frame_buffer) — (r, g, b)$\in$ [0, $2^n$-1]

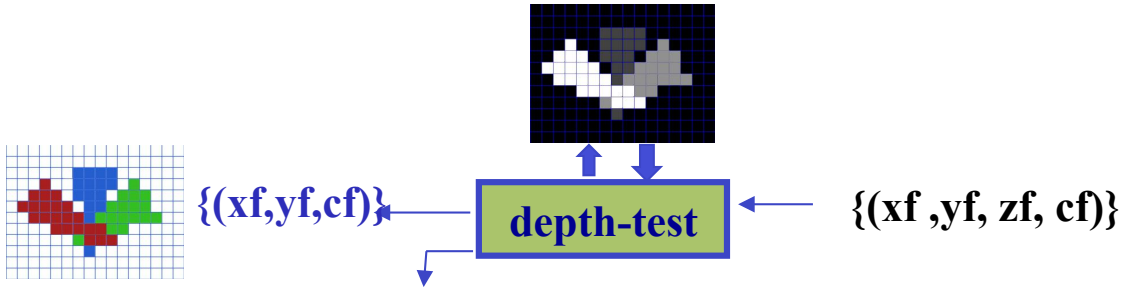
  Buffer profunditats (depth_buffer) — z $\in$ [0, $2^{nz}$-1]

1. **Inicialitzar al color de fons**



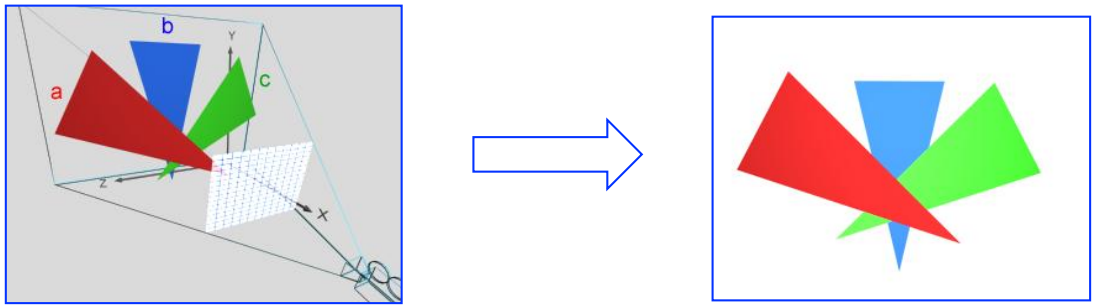
1. **Inicialitzar al més lluny possible**

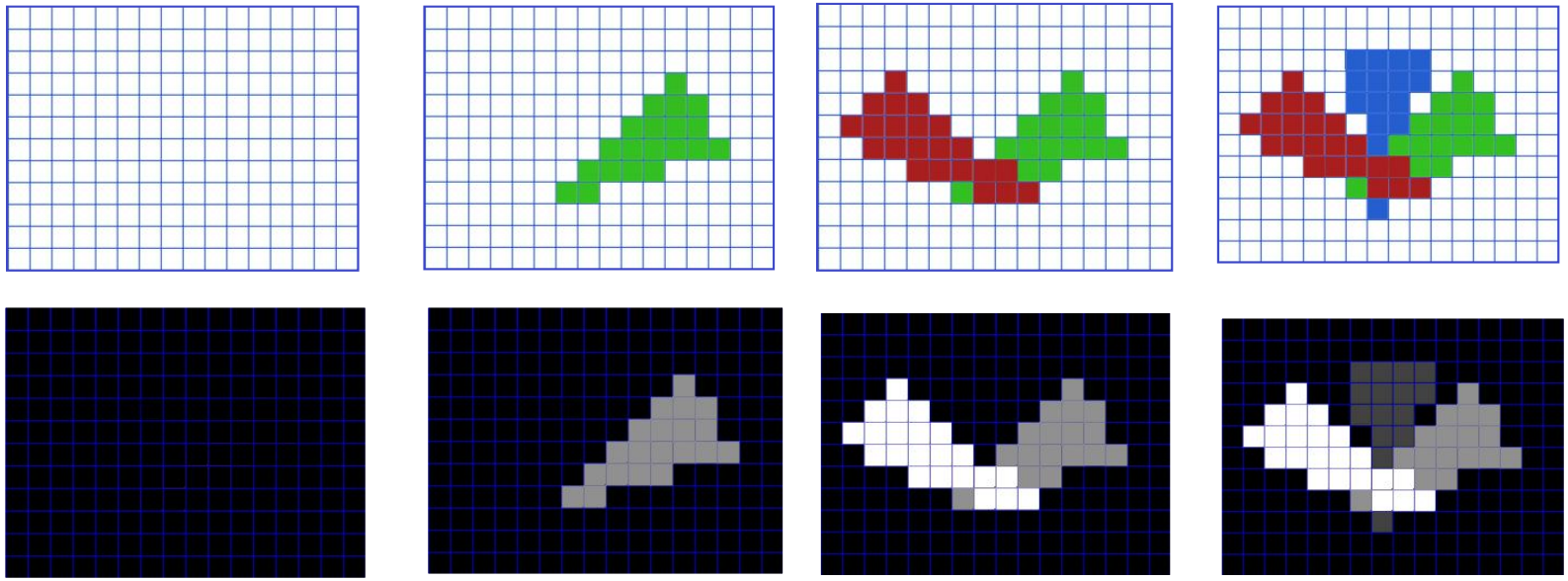




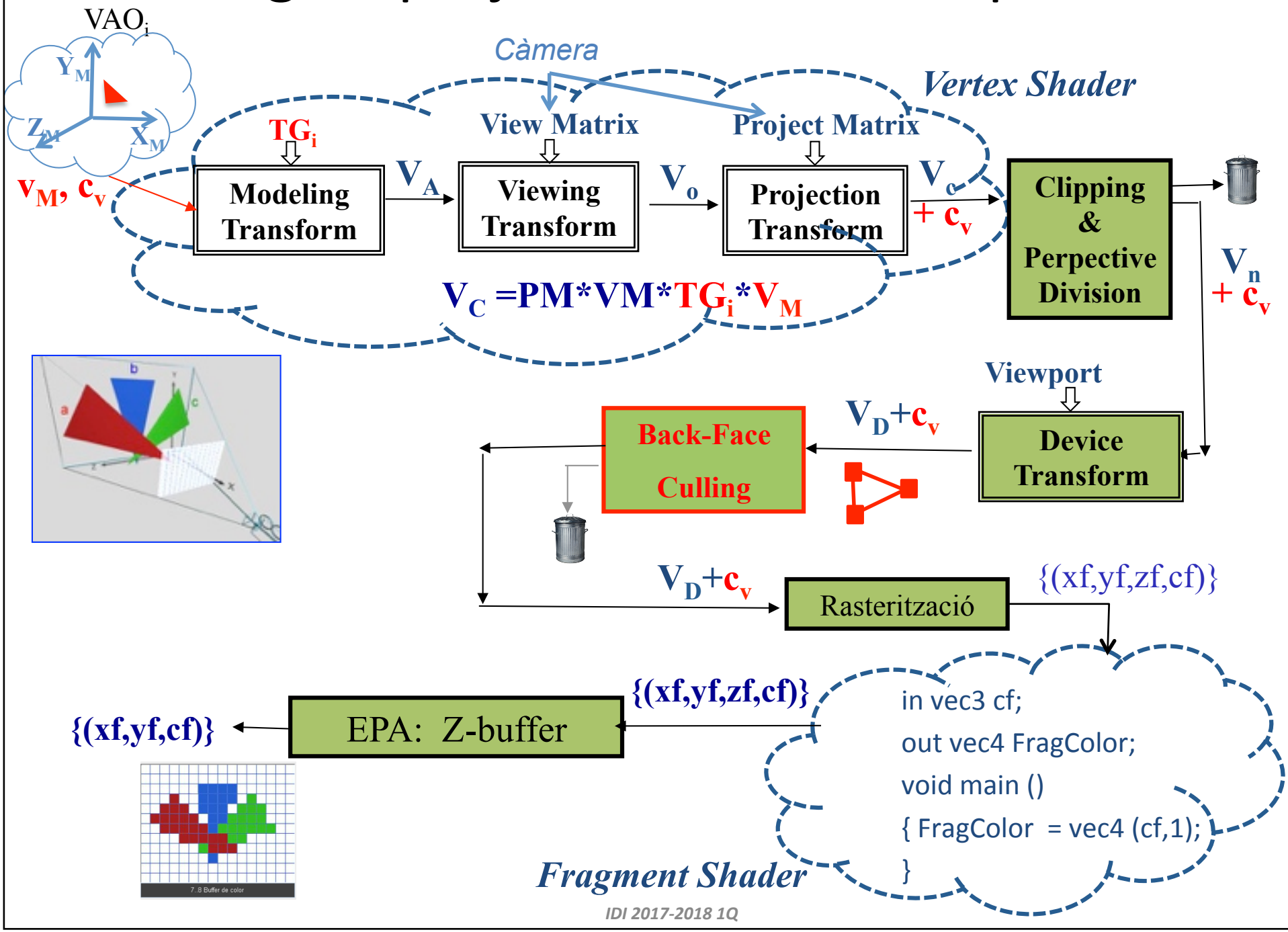

```
if (zf < depth_buffer[xf,yf]) {
        depth_buffer [xf,yf] = zf;
        color_buffer [xf,yf] = cf;
}
```

# Depth Buffer (z-buffer)



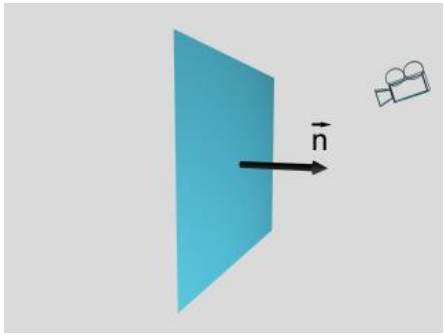



# Paradigma projectiu bàsic amb OpenGL 3.3

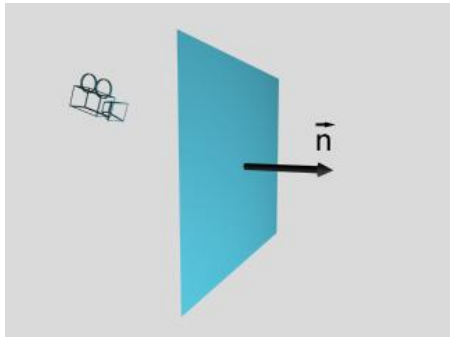


# Back-face Culling

- Mètode EPA en espai *objecte (a nivell de triangle)*
- Requereix cares orientades, opaques, objectes tancats
- Considera escena formada només per la *cara* i l'*observador*
- És conservatiu (determina les cares que "segur" no són visibles)

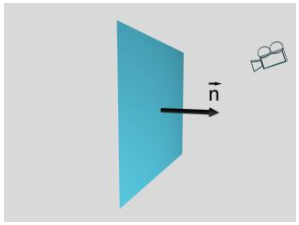


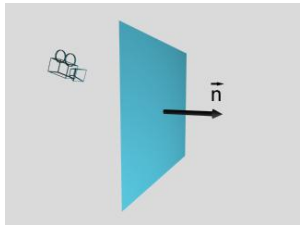

**visible**



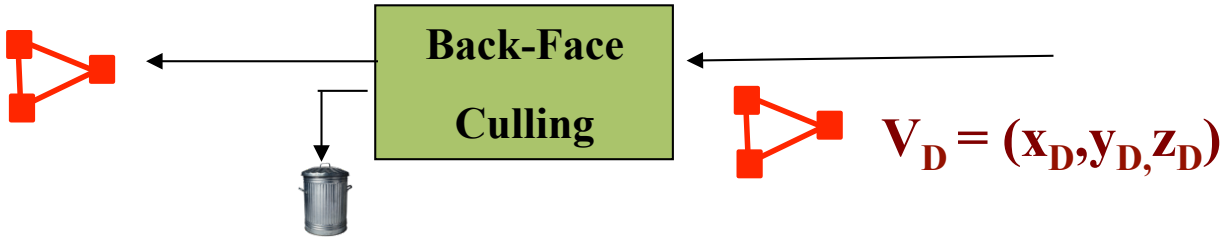

**no visible**

# Back-face Culling



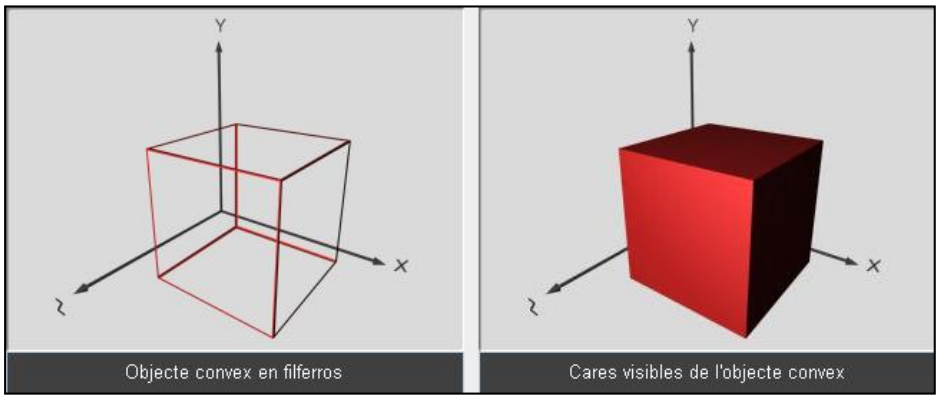



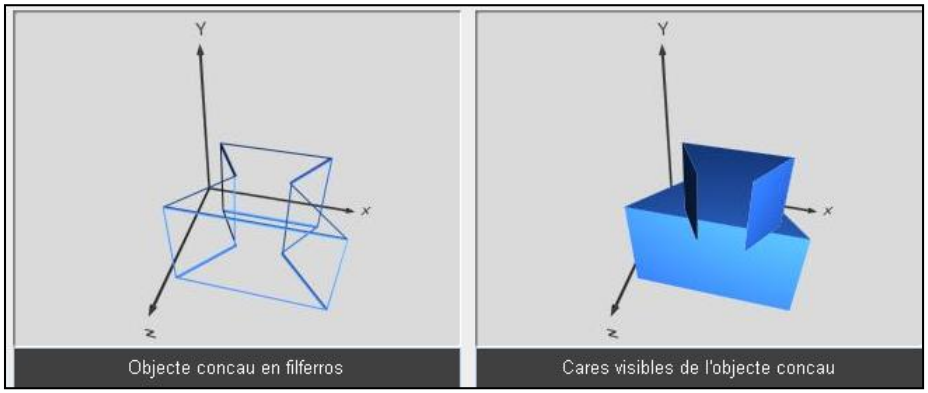

- OpenGL fa el càlcul en coord. dispositiu
  - direcció de visió (0,0,-1)
  - visibles les cares amb $n_z$ >0 (ordenació vèrtexs antihorari)
  - el càlcul de la normal de la cara el fa OpenGL a partir dels vèrtexs en coordenades de dispositiu => **importància ordenació vèrtexs.**

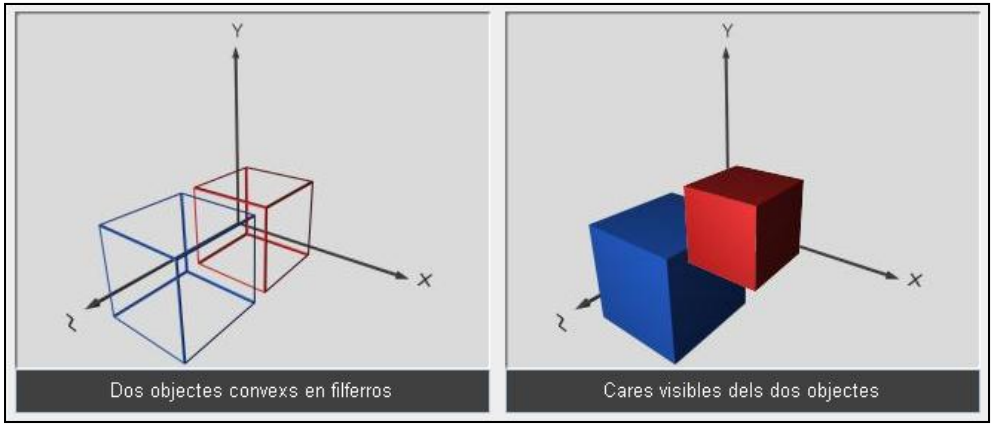


# Back-face Culling

- Culling com EPA només si l'escena conté un únic objecte convex.

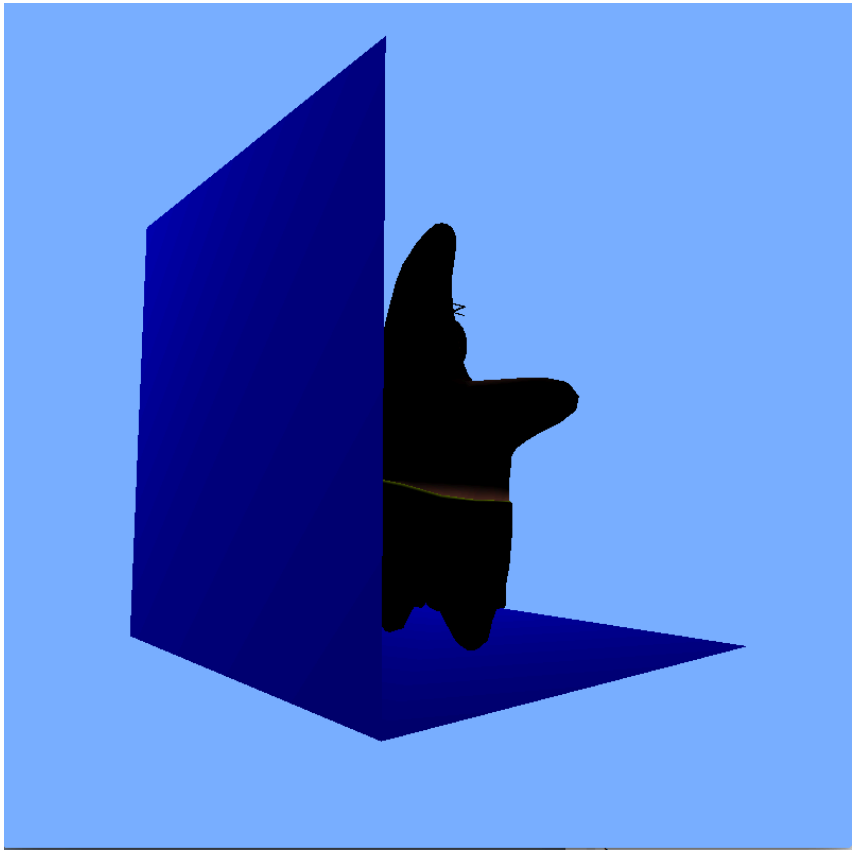


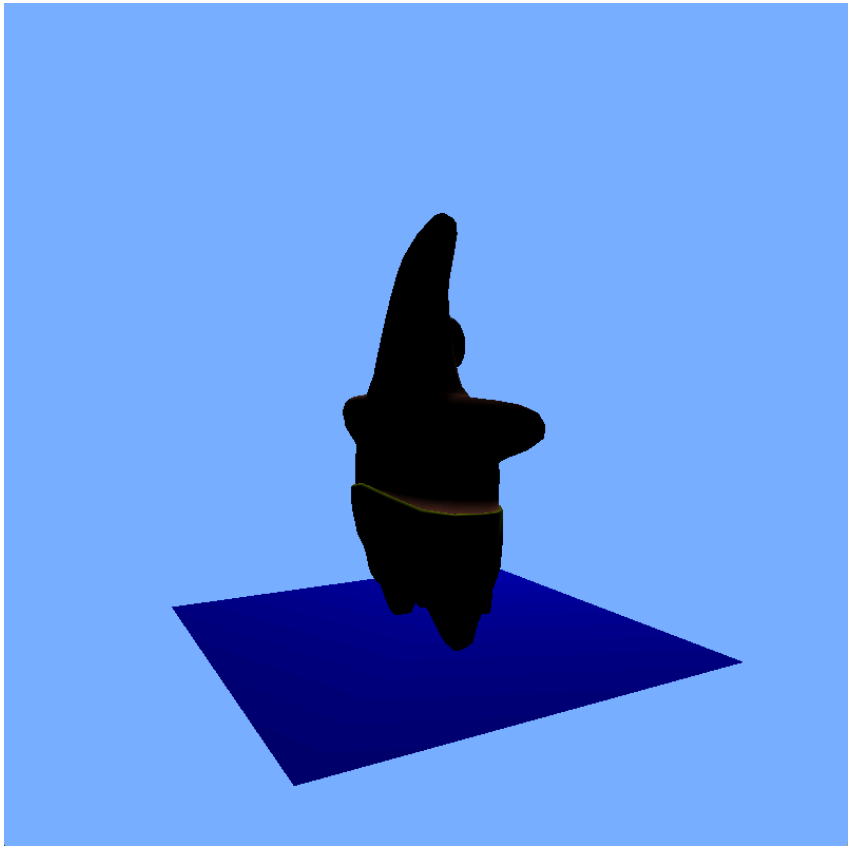



# Imatges que podreu comprovar al laboratori

Sense culling

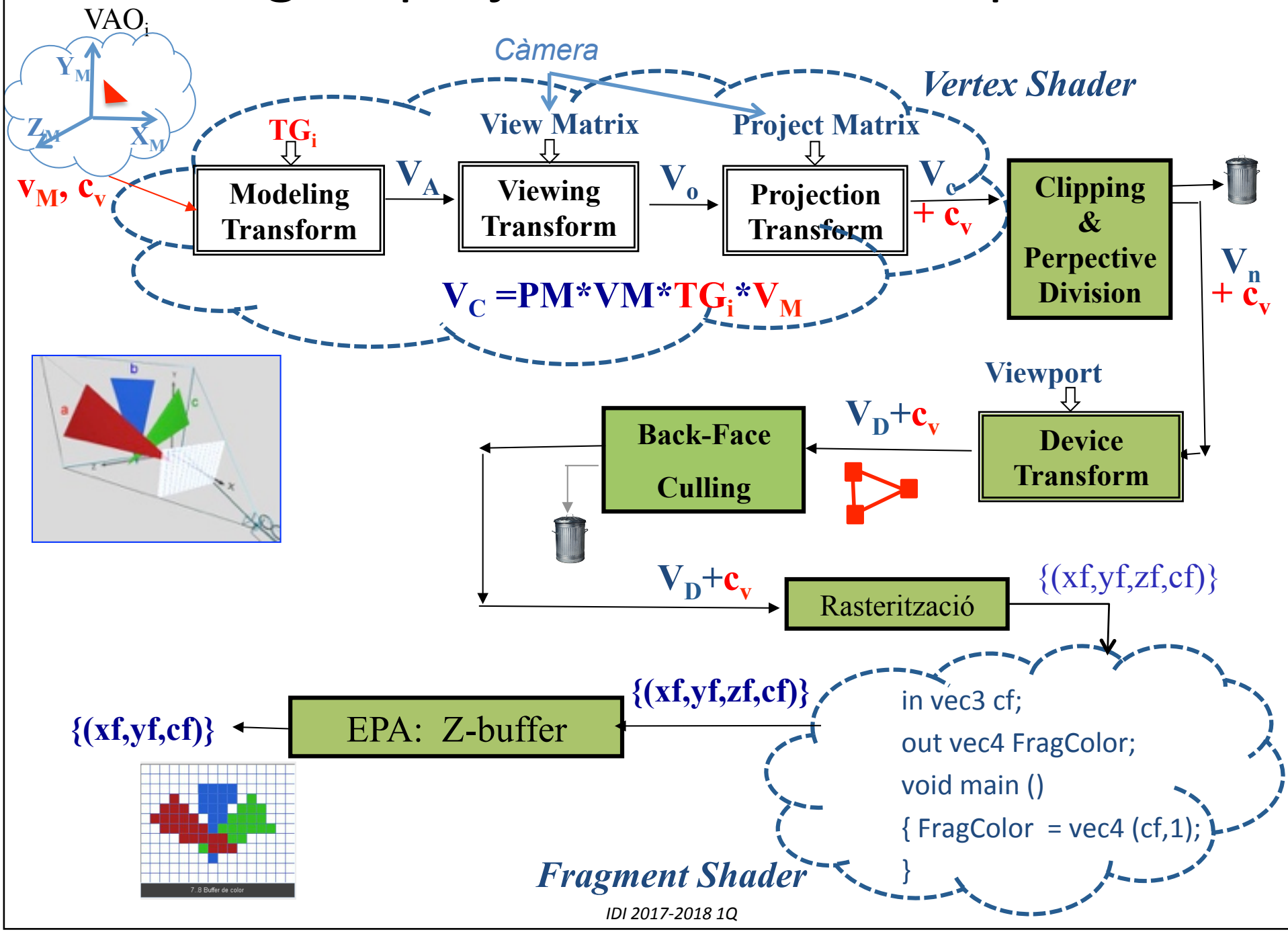


Amb culling



glEnable (GL_CULL_FACE);

# Paradigma projectiu bàsic amb OpenGL 3.3

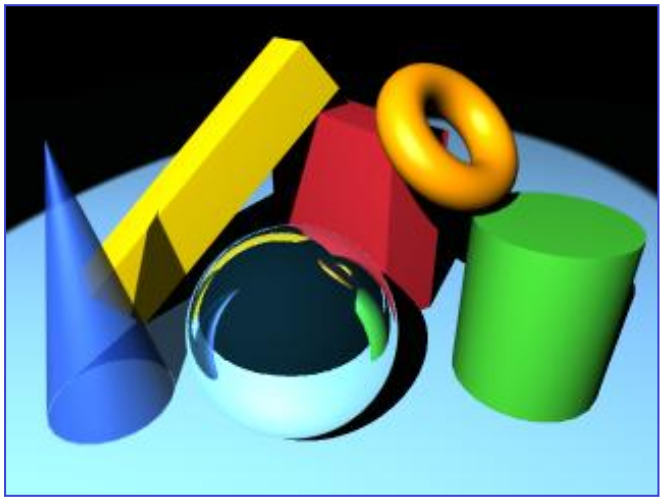


# Classe 6: contingut

- Realisme: Eliminació de parts ocultes
  - Back-face culling
  - Depth-buffer
- **Realisme: Il·luminació (1)**
  - Càlcul del color en un punt
- Exercicis càmera

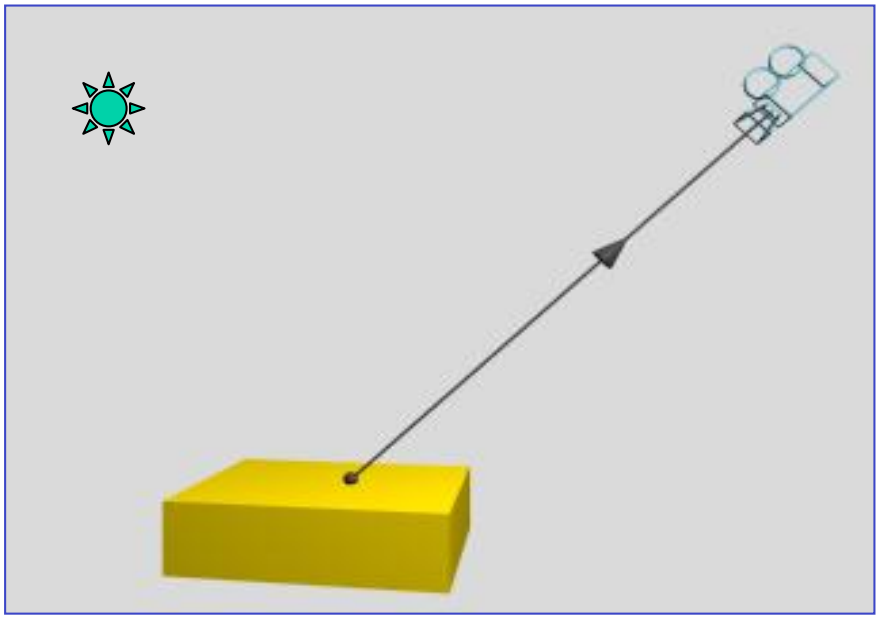
# Introducció

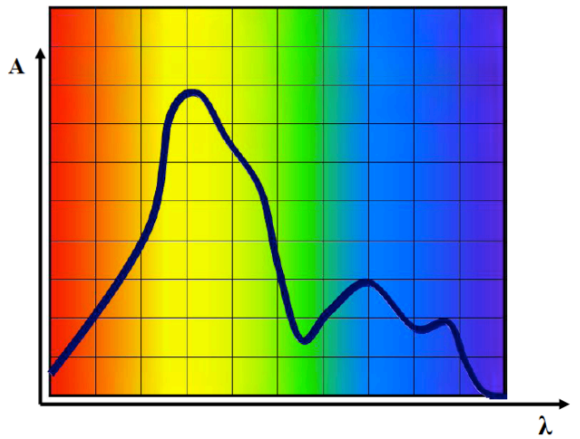
- Els models d'il·luminació simulen el comportament de la llum per determinar el color d'un punt de l'escena.
- Permeten obtenir imatges molt més realistes que pintant cada objecte d'un color uniforme:



# Color d'un punt

El color amb el que un Observador veu un punt P de l'escena és el color de la llum que arriba a l'Obs procedent de P: $I_\lambda(P \rightarrow Obs)$



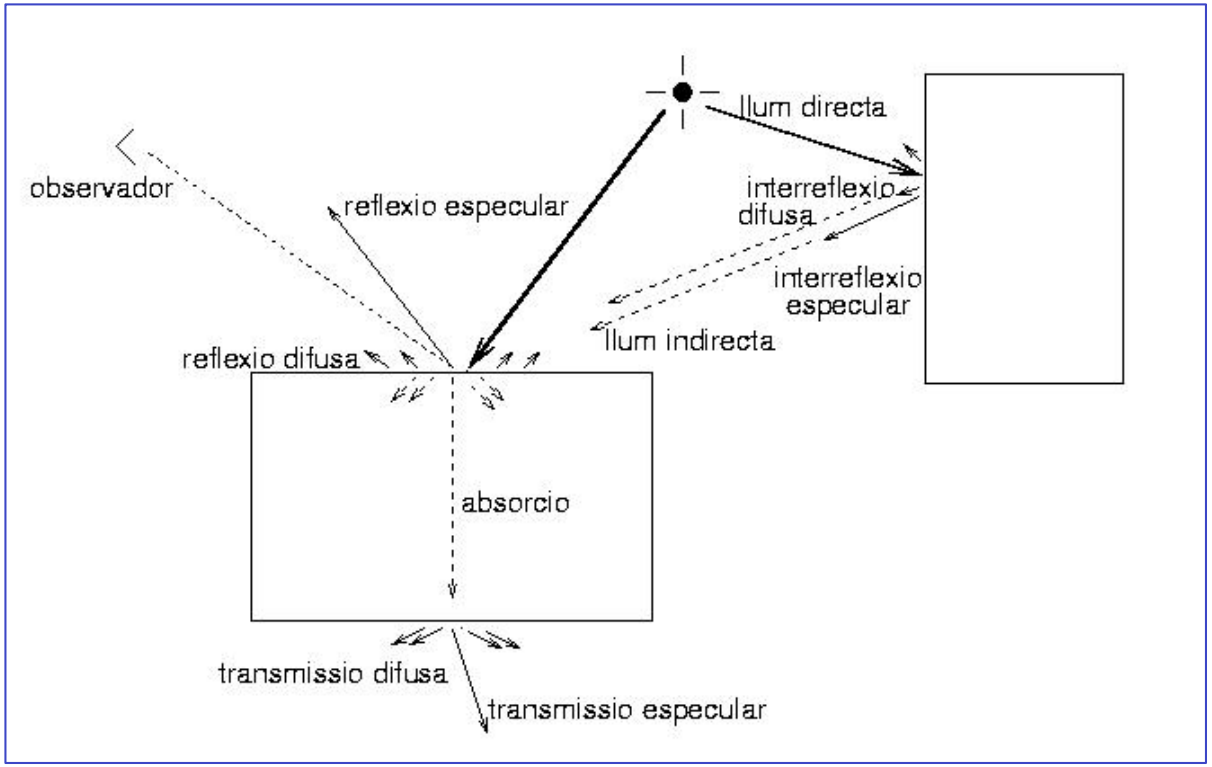



$I_\lambda(P \rightarrow Obs) \quad \lambda \in \{r,g,b\}$

# Elements que intervenen

El color que arriba a l'Obs procedent de P, $I_\lambda(P \rightarrow Obs)$, depèn de:

- Fonts de llum
- Materials
- Altres objectes
- Posició de l'observador
- Medi pel que es propaga



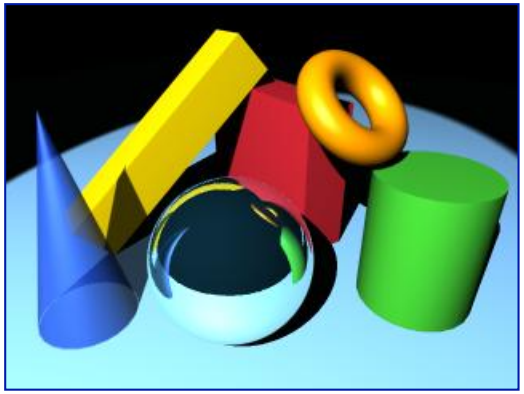
# Models d'il·luminació

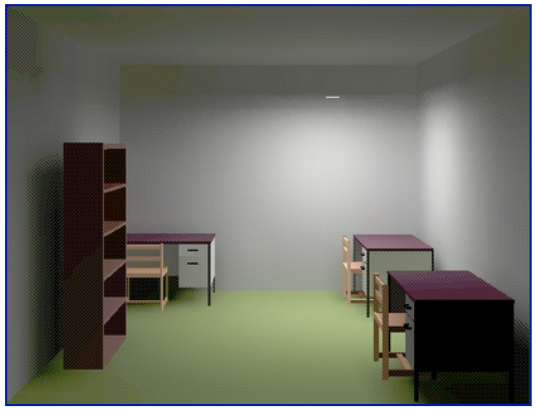
- Els models d'il·luminació simulen les lleis físiques que determinen el color d'un punt.
- El càlcul exacte és computacionalment inviable.
- Una primera simplificació és usar només les energies corresponents a les llums vermella, verda i blava.

$$I_\lambda(P \rightarrow Obs) \quad \lambda \in \{r,g,b\}$$

# Models d'il·luminació: Classificació

- Models Locals o empírics
- Models Globals: traçat de raig, radiositat





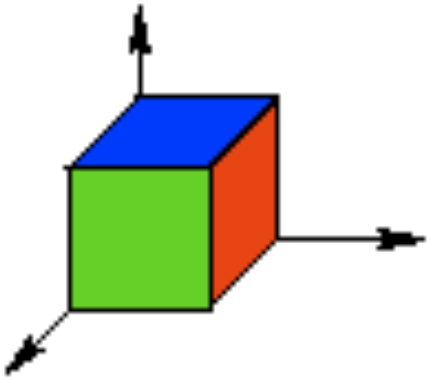
# Classe 6: contingut

- Realisme: Eliminació de parts ocultes
  - Back-face culling
  - Depth-buffer
- Realisme: Il·luminació (1)
  - Càlcul del color en un punt
- **Exercicis càmera**

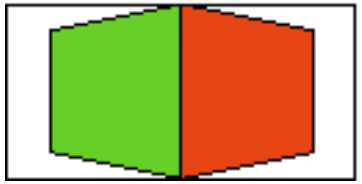
77. (2015-2016P Q1) Tenim una escena amb un cub de costat 2 orientat amb els eixos i de manera que el seu vèrtex mínim està situat a l'origen de coordenades. La cara del cub que queda sobre el pla x=2 és de color vermell, la cara que queda sobre el pla z=2 és de color verd i la resta de cares són blaves.

a) Indica TOTS els paràmetres d'una càmera perspectiva que permeti veure completes a la vista només les cares vermella i verda. La relació d'aspecte del viewport (vista) és 2. Fes un dibuix indicant la imatge final que s'obtindria.

b) Quin efecte tindria en la imatge final modificar l'òptica a axonomètrica? Definir òptica axonomètrica



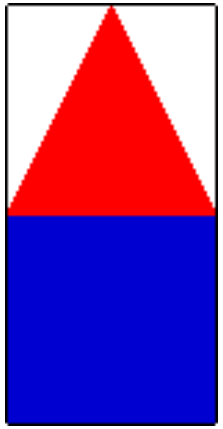
VRP= (2,1,2)
OBS= (3,1,3)
Up = (0,1,0)



100. (2016-2017P Q1) Dos estudiants discuteixen respecte a la implementació del zoom amb òptica axonomètrica (ortogonal) i perspectiva. Quina de les seves afirmacions és certa?

a) En òptica ortogonal només es pot obtenir un efecte de zoom modificant OBS i VRP en la direcció de visió.
b) En òptica perspectiva cal modificar FOV, Znear i Zfar.
c) En les dues òptiques es pot fer zoom modificant el window de la càmera.
d) En òptica perspectica si avancem OBS i VRP en la direcció de visió cal anar amb compte amb la raw.

Tenim una piràmide de base quadrada de costat 5, amb la base centrada al punt (0,0,2.5) i alçada de la piràmide 5 amb l'eix en direcció Z+. A l'escena tenim també un cub de costat 5 centrat a l'origen. El viewport esta definit amb glViewport (0,0,400,800). Si a la vista es veu la imatge que teniu al dibuix (caseta), quines inicialitzacions d'una càmera axonomètrica (posició+orientació i òptica) permetrien veure aquesta imatge? Tots els angles estan en graus.



```
PM=perspective (90, 1, 5, 10);
projectionMatrix (PM)
VM=translate (0,0,-10);
VM=VM*rotate (90,1,0,0);
VM=VM*translate (0,0,-2.5);
viewMatrix (VM);
pinta_escena ();
```

```
PM=ortho (-2.5, 2.5, -5, 5,
5, 10);
projectionMatrix (PM)
VM=translate (0,0,-7.5);
VM=VM*rotate (-90,0,0,1);
VM=VM*rotate (90,0,1,0);
VM=VM*translate (0,0,-2.5);
viewMatrix (VM);
pinta_escena ();
```

```
PM=ortho (-2.5, 2.5, -5, 5, 5,
10);
projectionMatrix (PM)
VM=translate (0,0,-7.5);
VM=VM*rotate (90,0,0,1);
VM=VM*rotate (90,0,1,0);
VM=VM*translate (0,0,-2.5);
viewMatrix (VM);
pinta_escena ();
```

```
PM=ortho (-5, 5, -5, 5, 5,
10);
projectionMatrix (PM)
VM=translate (0,0,-7.5);
VM=VM*rotate (90,0,0,1);
VM=VM*rotate (90,0,1,0);
VM=VM*translate (0,0,-2.5);
viewMatrix (VM);
pinta_escena ();
```

**Exercici 43:** Indica quina de les inicialitzacions de l'òptica perspectiva és més apropiada per a una càmera que porta un observador que camina per una escena fent fotos amb una òptica constant. Esfera englobant d'escena té radi R, d és la distància entre OBS i VRP. Observació: ra_v és la relació d'aspecte del *viewport*

a) FOV = 60º, ra = ra_v, zNear = 0.1, zFar = 20

b) FOV = 60º, ra = ra_v, zNear = R, zFar = 3R;
essent R el radi de l'esfera contenidora de l'escena.

c) FOV = 2*(arcsin(R/d)*180/PI), ra = ra_v, zNear = R, zFar = 3R;
essent R el radi de l'esfera contenidora de l'escena i d la distància d'OBS a VRP.

d) FOV = 2*(arcsin(R/d)*180/PI), ra = ra_v, zNear = 0, zFar = 20;
essent R el radi de l'esfera contenidora de l'escena i d la distància d'OBS a VRP

73. (2014-2015P 2Q) Es vol realitzar una vista en planta (visió des de dalt) d'una escena/objecte que està centrat a l'origen amb una capsa contenidora de mides 10x10x10. Quina de les següents definicions et sembla correcta per definir la posició + orientació de la càmera (per a calcular la viewMatrix)? Sabem que la càmera és perspectiva i els angles de les rotacions estan en graus.

a) OBS = (0,10,0); VRP = (0,0,0); up = (0,1,0);
VM = lookAt (OBS, VRP, up);
viewMatrix(VM);
b) OBS = (0,0,0); VRP = (0,10,0); up = (0,0,-1);
VM = lookAt (OBS, VRP, up);
viewMatrix(VM);
c) VM = translate (0,0,-10);
VM = VM * rotate (90, 1,0,0);
viewMatrix(VM);
d) VM = translate (0,0,-10);
VM = VM * rotate (-90, 0,1,0);
viewMatrix(VM);

Tenim una escena amb un triangle vermell amb vèrtexs V1=(-2,0,0), V2 = (2, 0, 0) i V3= (0, 1, 0). Suposant que tenim un viewport quadrat de 600x600 píxels, i que hem inicialitzat les matrius de càmera (view) i projecció (proj) a la matriu identitat, indica quina de les següents imatges és la que sortirà en un viewport de 600x600 (sabem que el Vertex Shader i el Fragment Shader estan correctament implementats):

a)

b)

c)

d)

**Exercici 48:** Disposem d'una càmera ortogonal amb els següents paràmetres:

OBS=(0.,0.,0.), VRP=(-1.,0.,0.), up=(0.,1.,0.), window de (-5,-5) a (5,5), ra=1, zn=5, zf=10.

Indiqueu quin conjunt de paràmetres d'una càmera perspectiva defineix un volum de visió que conté l'anterior (és a dir, garanteix que es veurà, coma mínim, el mateix que amb la càmera axonomètrica):

a) FOV= 90, ra=1, zn= 5, zf=10
b) FOV= 60, ra=1, zn=5, zf=10
c) FOV= 60, ra= 2, zn=6, zf=11
d) FOV= 90, ra= 0.5, zn=5, zf=10

## **Defineixen la mateixa VM?**:

| Codi 1 |
| --- |
| VM= LookAt ((0,80,0),(0,50,0),(0,0,-1)); |

| Codi 2 |
| --- |
| VM = Translate(0, 0, -30);<br>VM = VM * Rotate (90 , 1, 0, 0);<br>VM = VM*Translate (0,-50,0) |

***Preguntes:***

*a) Defineixen la mateixa càmera?*

*b) Quina vista de l'escena?*

## Exercici 35: Defineixen la mateixa VM?:

Codi 1

```
VM= LookAt ((0,80,0),(0,50,0),(0,0,-1));
```

Codi 2

```
VM =  Translate(0, 0, -80);
VM = VM * Rotate (90 , 1, 0, 0);
```

***Preguntes:***

*a) Defineixen la mateixa càmera?*

*b) Quina vista de l'escena?*

## **Defineixen la mateixa VM?**:

Codi 1

```
VM= LookAt ((0,80,0),(0,50,0),(0,0,-1);
```

Codi 2

```
VM =  Translate(0, 0, -80);
VM = VM * Rotate (90, 0, 0, 1);
VM = VM * Rotate (90 , 1, 0, 0);
VM = VM * Rotate(-90, 0, 1, 0);
```

***Preguntes:***

*a) Defineixen la mateixa càmera?*

*b) Quina vista de l'escena?*

*c) Es poden optimitzar les TGs del codi 2?*

## **Defineixen la mateixa VM?**:

| Codi 1 |
| --- |
| VM= LookAt ((0,80,0),(0,50,0),(1,0,0)); |

| Codi 2 |
| --- |
| VM = Translate(0, 0, -80);<br>VM = VM * Rotate (90, 0, 0, 1);<br>VM = VM * Rotate (90 , 1, 0, 0);<br>VM = VM * Rotate(-90, 0, 1, 0); |

***Preguntes:***

*a) Defineixen la mateixa càmera?*

*b) Quina vista de l'escena?*

(2016-2017P Q1) Tenim una escena amb una caseta formada per:

- un cub de color blau de costat 20 amb les cares paral·leles als plans coordenats i amb el centre de la seva cara inferior situat en el punt (10,0,0).
- una piràmide de color vermell de base quadrada d'aresta 20 i alçada 10, ubicada just a sobre del cub, amb la base de la piràmide coincidint amb la cara superior del cub.

Al pintar la caseta en un viewport quadrat amb els paràmetres de càmera:
OBS = (10,40,0); VRP=(10,-30,0); up=(1,0,0); FOV = 90º; ra = 1.0;Znear =10; Zfar = 45;
Quina de les següents figures representa la imatge resultant?