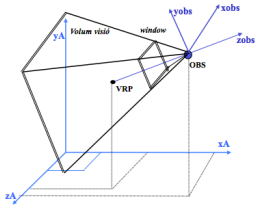
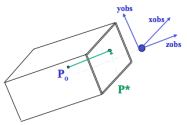
Classe 6: contingut

- Realisme: El·liminació 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 Inicial i interacció

EPA i Il·uminació

`view & project transform:`
viewMatrix, projectMatrix

viewport

ModelMatrix$_i$ (TG)

PintaModel$_i$

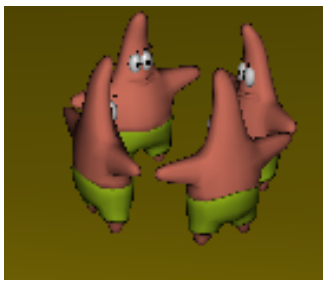
VAO$_i$

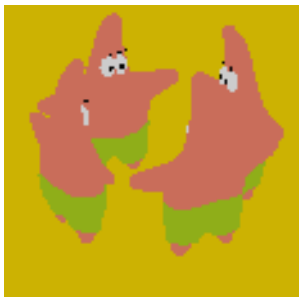
Vertex Shader

OpenGL-Visualització

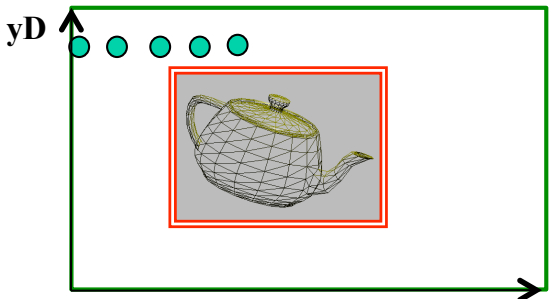
Fragment Shader

Píxel(x,y,c)





Finestra OpenGL (Pantalla)

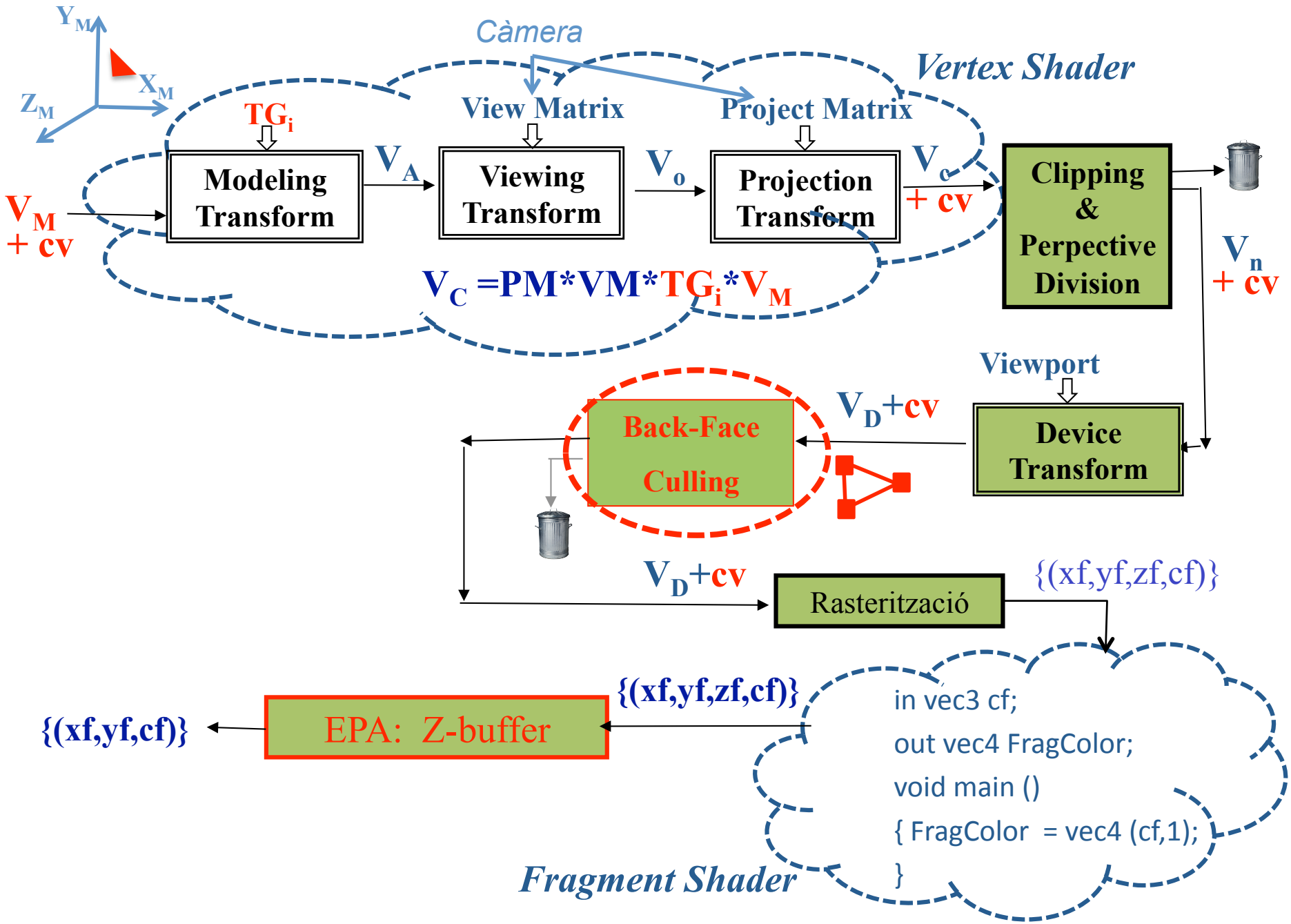


Viewport/Vista

Classe 6: contingut

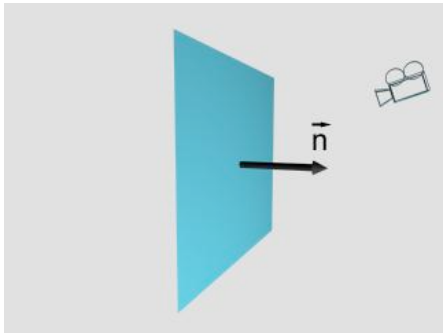
- **Realisme: El·liminació de parts ocultes**
  - Back-face culling
  - Depth-buffer
- Realisme: Il·luminació (1)
  - Càlcul del color en un punt
- Exercicis càmera

Paradigma projectiu bàsic amb OpenGL 3.3

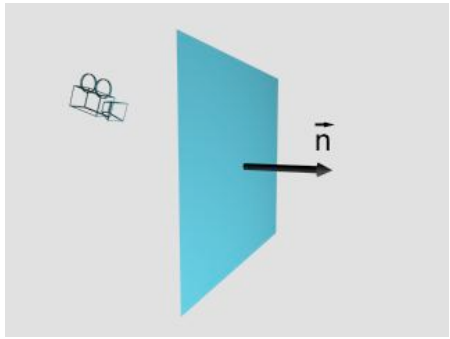


Back-face Culling

- 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 <u>conservatiu</u> (determina les cares que "segur" no són visibles)

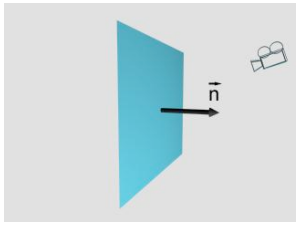


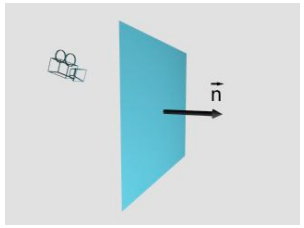
visible



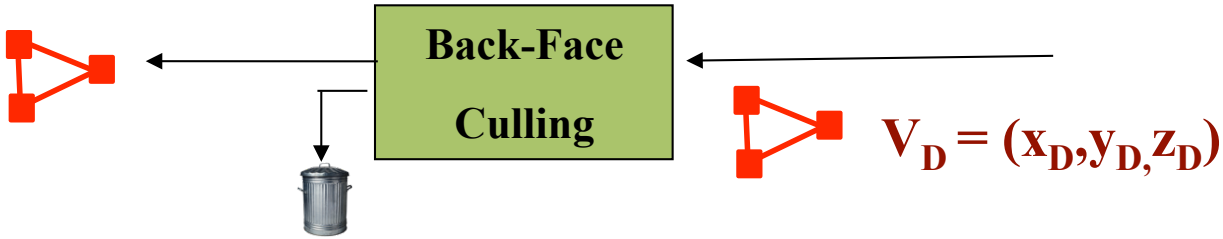
no visible

Back-face Culling



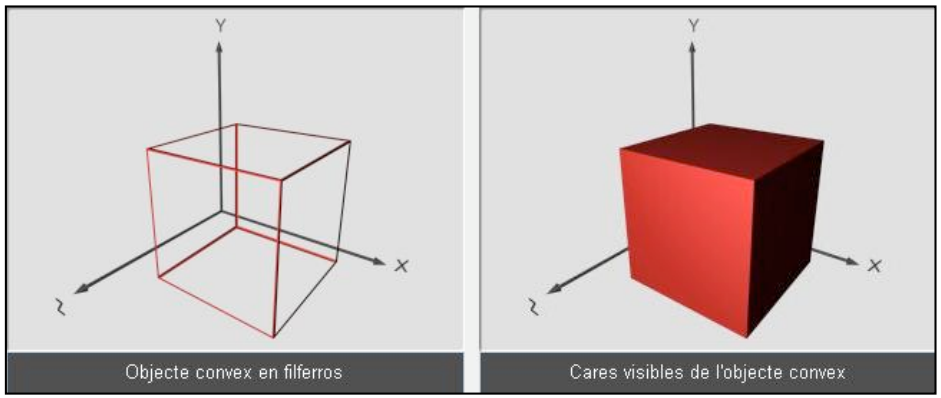


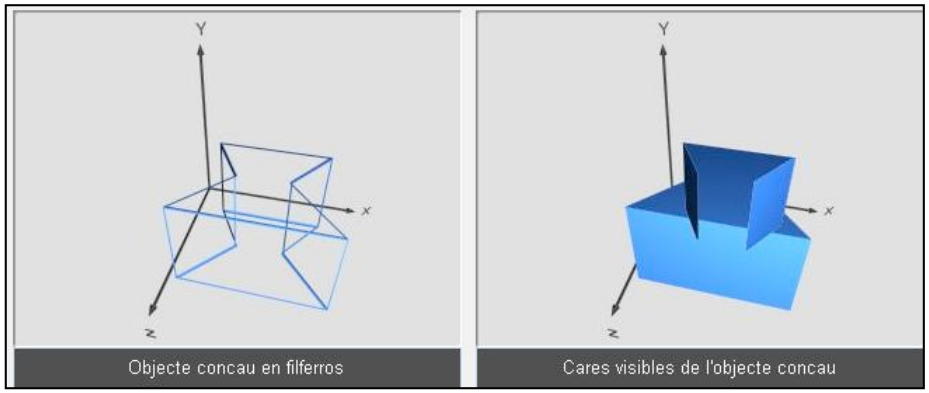
- OpenGL fa el càlcul en coord. dispositiu
  - direcció de visió (0,0,-1)
  - visibles les cares amb n_z >0 (ordenació vèrtexs antihorari)
  - el càlcul de la normal de la cara el fa OpenGL a partir dels vèrtexs en coordenades de dispositiu => **importància ordenació vèrtexs.**

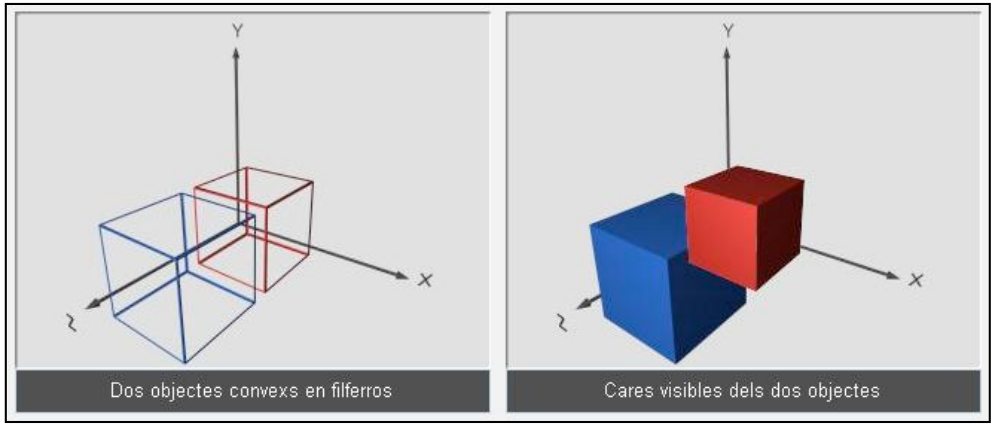


Back-face Culling

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

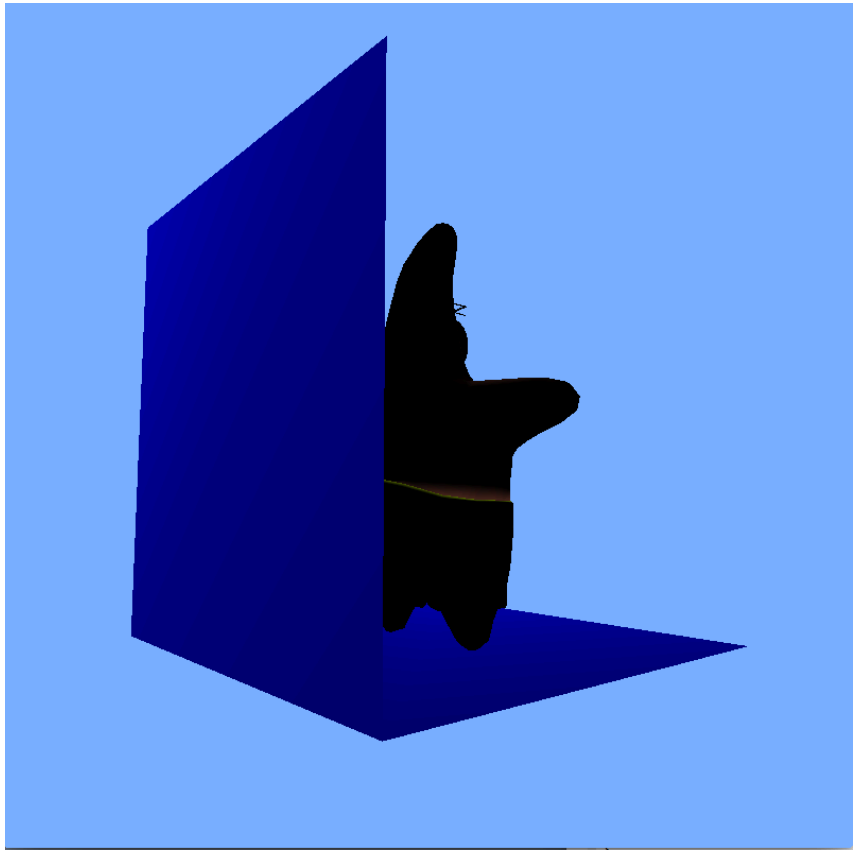




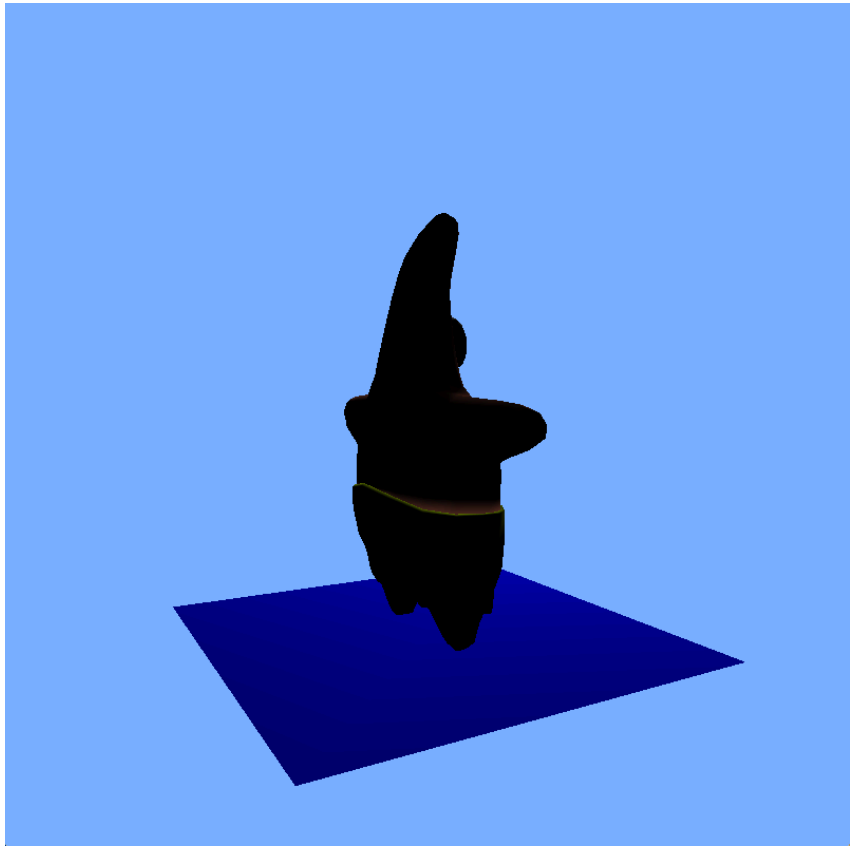


Imatges que podreu comprovar al laboratori

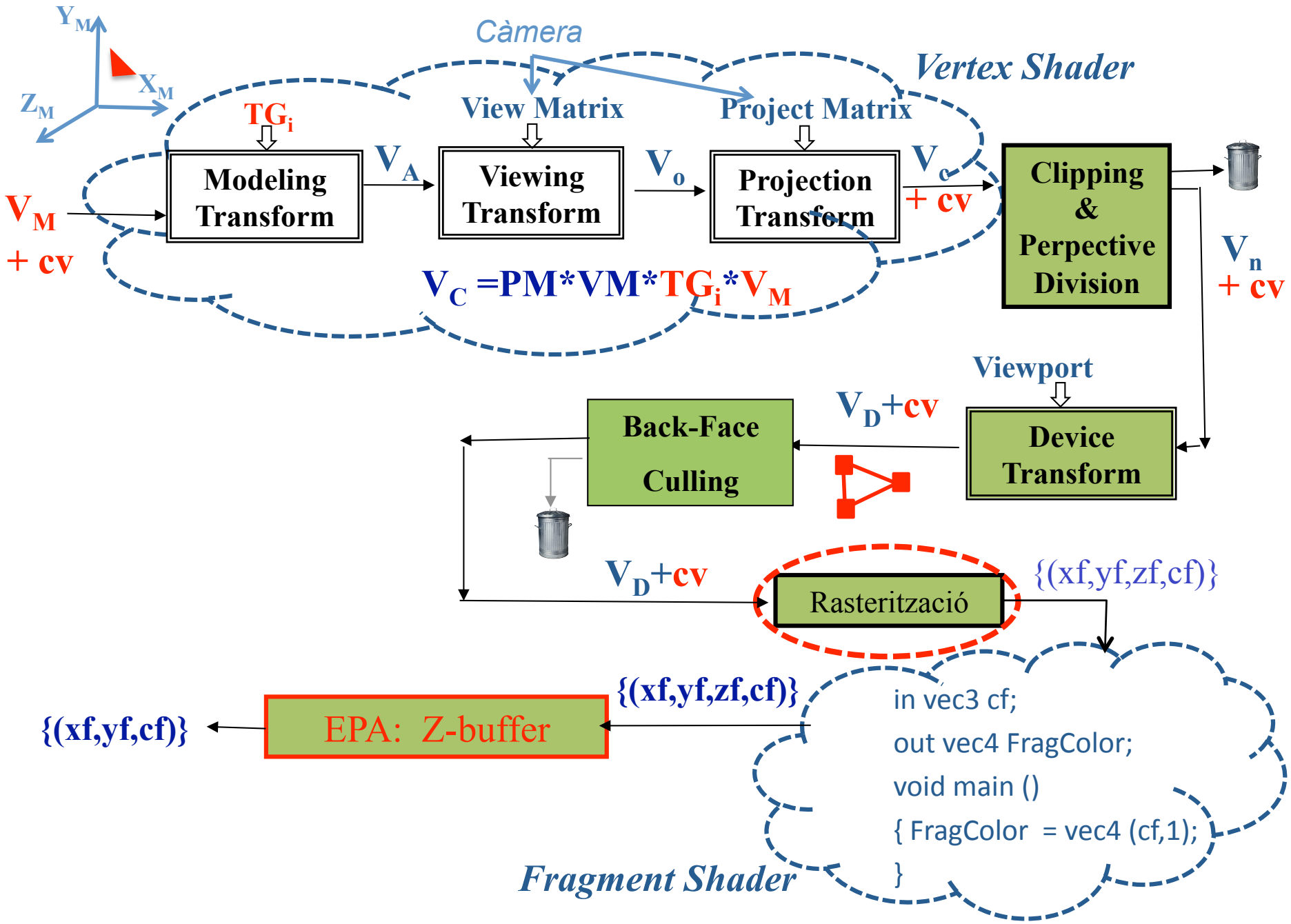
Sense culling



Amb culling

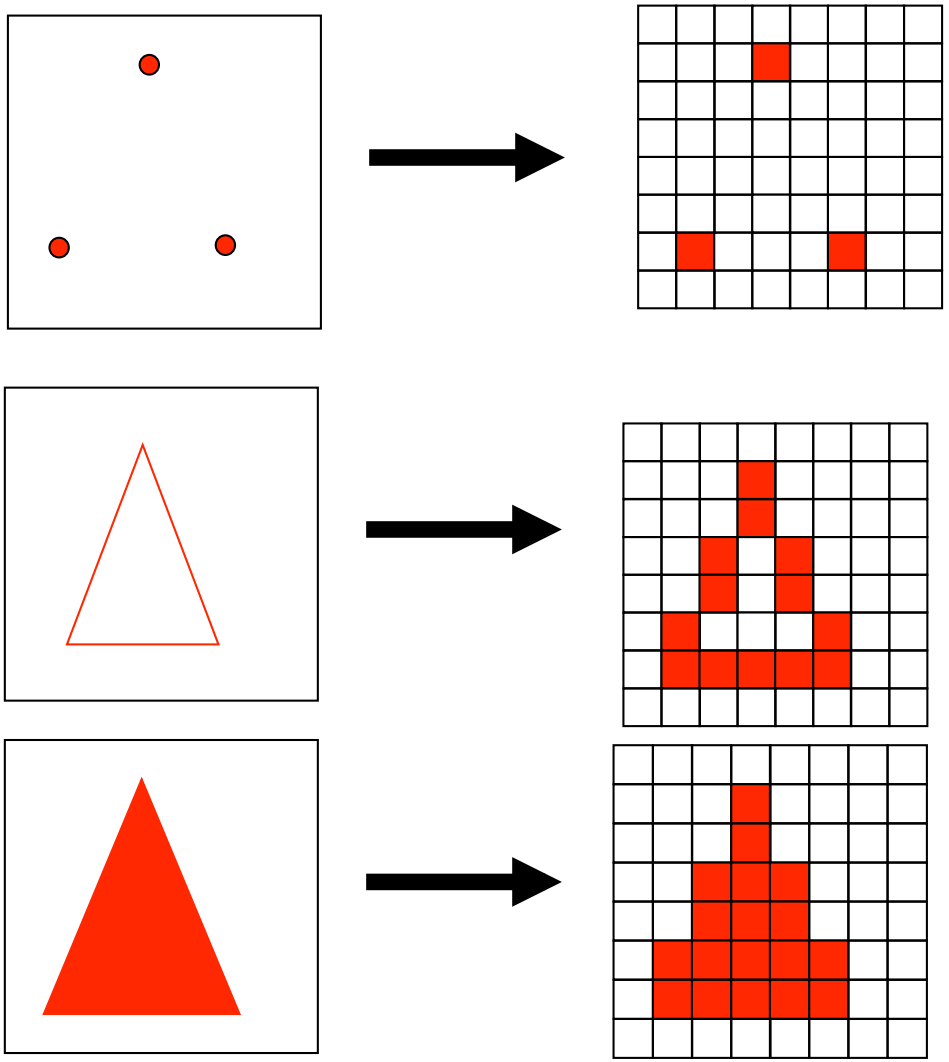


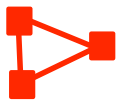
Paradigma projectiu bàsic amb OpenGL 3.3



Algorismes de rasterització

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





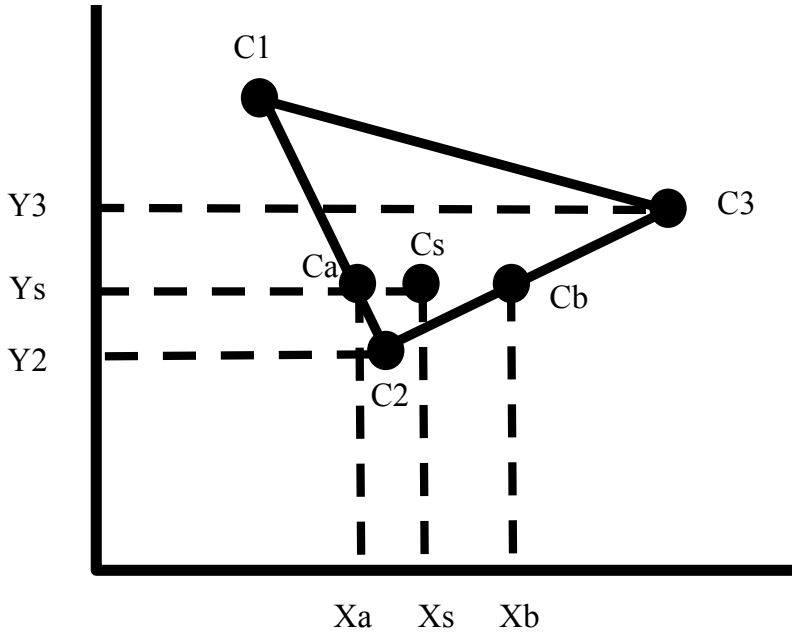
$V_D = (x_D, y_D, z_D, cv)$

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*

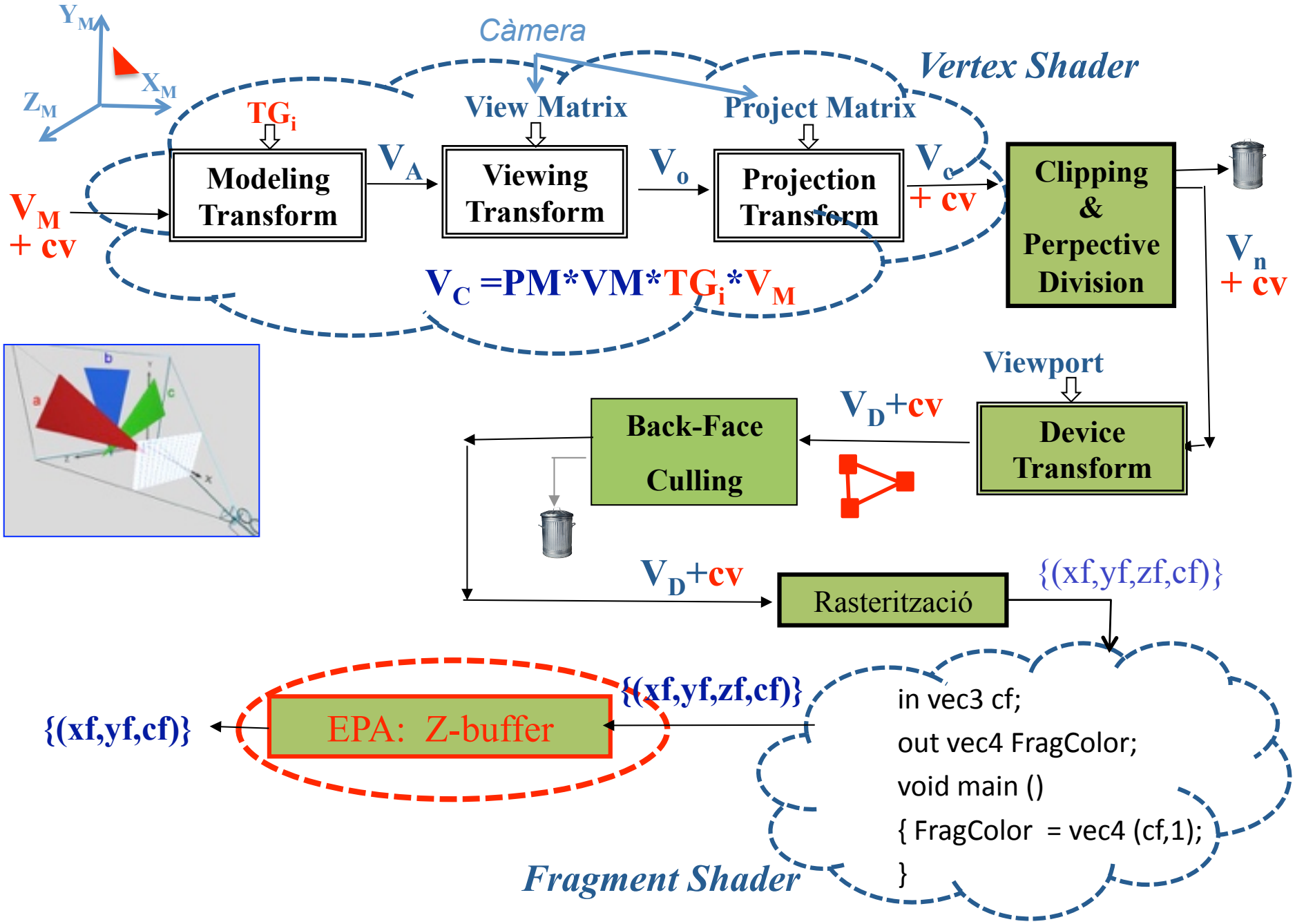


$$Ca = \frac{1}{Y1 - Y2}(C1(Ys - Y2) + C2(Y1 - Ys))$$

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

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

Paradigma projectiu bàsic amb OpenGL 3.3



Depth Buffer

- Mètode EPA en espai imatge *(a nivell de píxel/fragment)*
- Després de la **rasterització**
- No requereix tenir el Back-face culling activat
- 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

Depth Buffer (z-buffer)

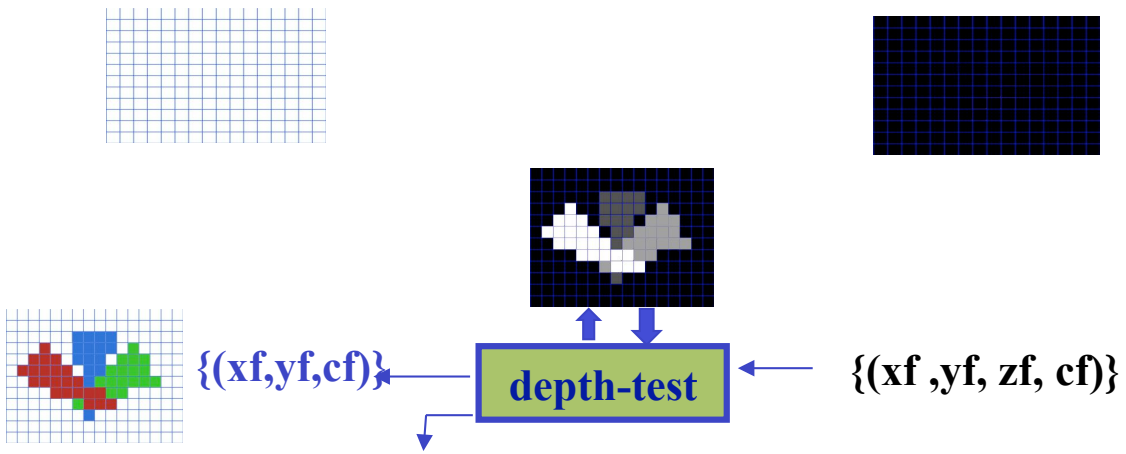
- Dos buffers de la mateixa resolució que la pantalla

  Buffer color (frame_buffer) Buffer profunditats (depth_buffer)

  (r, g, b)$\in$ [0, 2^n-1] z $\in$ [0, 2^{nz}-1]

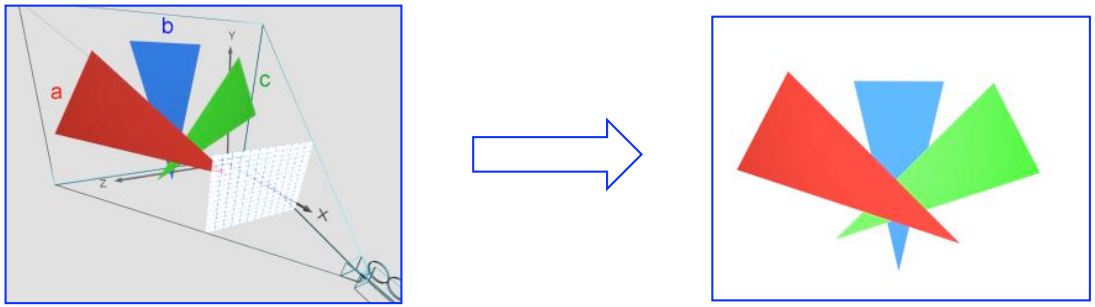
1. **Inicialitzar al color de fons**

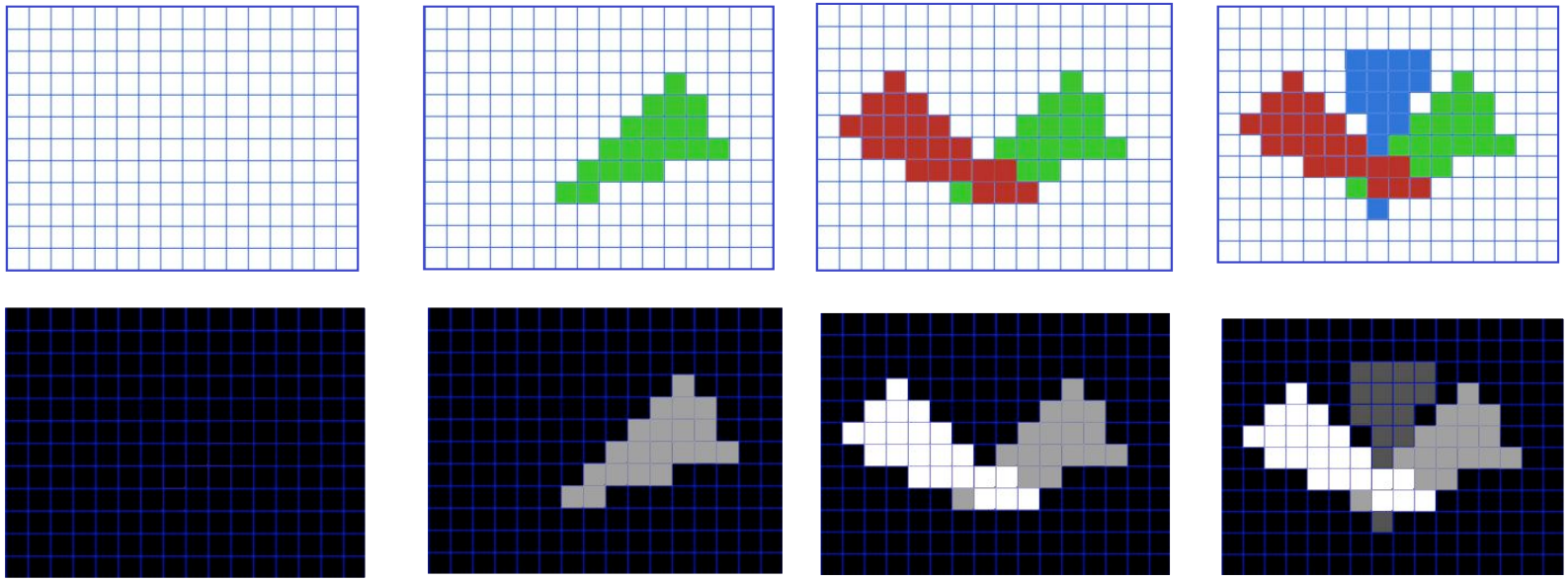
1. **Inicialitzar al més lluny possible**



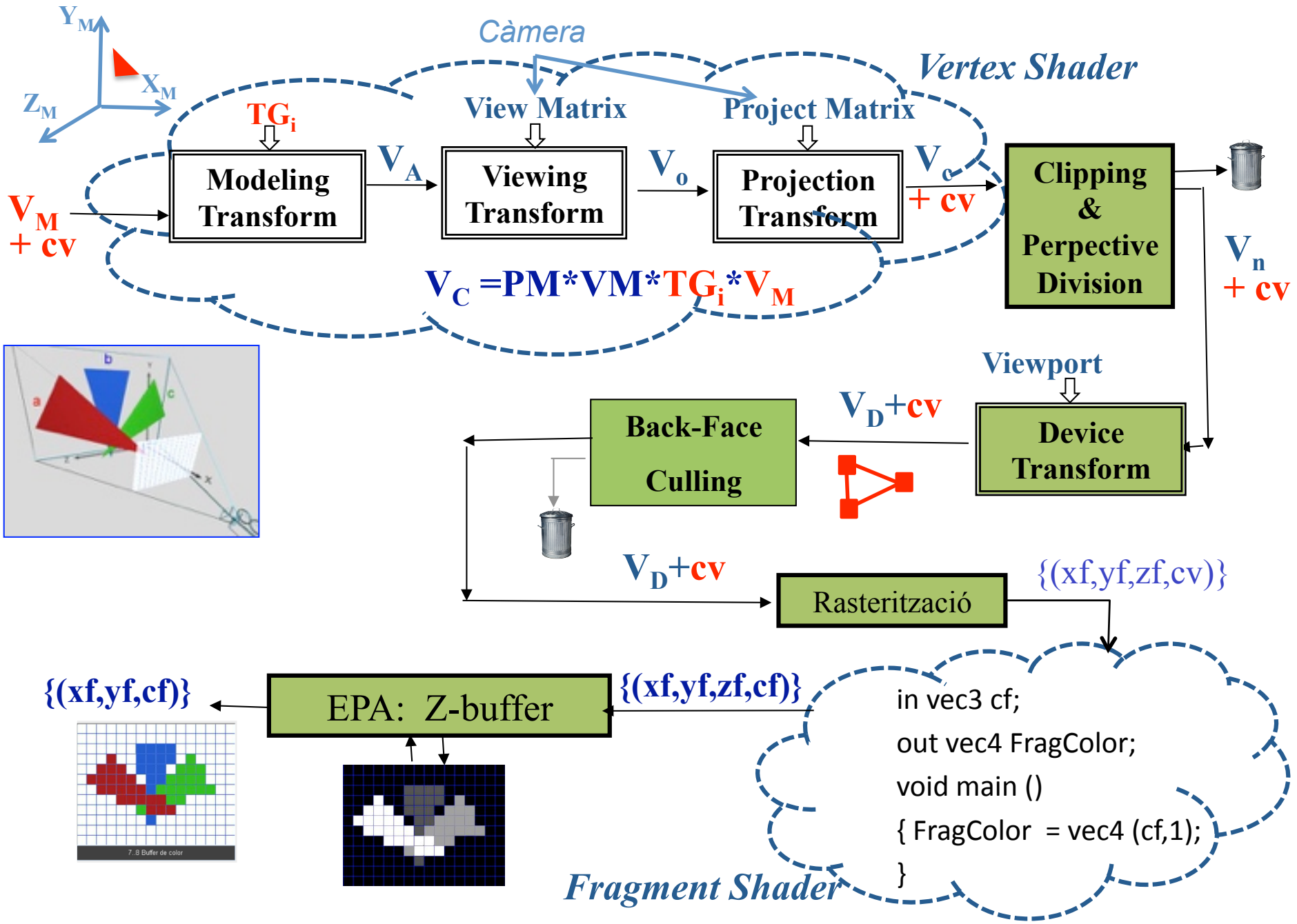

```
if (zf < depth_buffer[xf,yf]) {
        depth_buffer [xf,yf] = zf;
        color_buffer [xf,yf] = cf;
}
```

Depth Buffer (z-buffer)





Paradigma projectiu bàsic amb OpenGL 3.3



```
/* un cop: InicializeGL():
   CreateBuffers(); CarregaShaders()
   glEnable (GL_CULL_FACE);
   glEnable (GL_DEPTH_TEST);
   IniCamera() calcular paràmetres càmera i
   matrius inicials VM i PM*/
//viewTransform()
VM = lookAt(OBS,VRP,UP);
viewMatrix(VM);
//projectTransform()
PM=perspective (FOV,ra,zN,ZF);
projectMatrix(PM);
//resize(...)
glViewport (0,0,w.h);
/*PaintGL();  cada cop que es requerix
   refresc*/
  glClear(GL_COLOR...|GL_DEPTH...);
  /*per cada model: modelTransform()
     Calcula TGi i passar a OpenGL*/
      modelTransform_i(TG);
      modelMatrix(TG);
      Pinta_model(VAO);
```

```
in vec3 vertex, color;
out vec3 cv;
uniform mat4 TG, VM, TP;
void main ()
{ gl_Position= TP*VM*TG*vec4(vertex,1.0);
  cv=color;
}
```

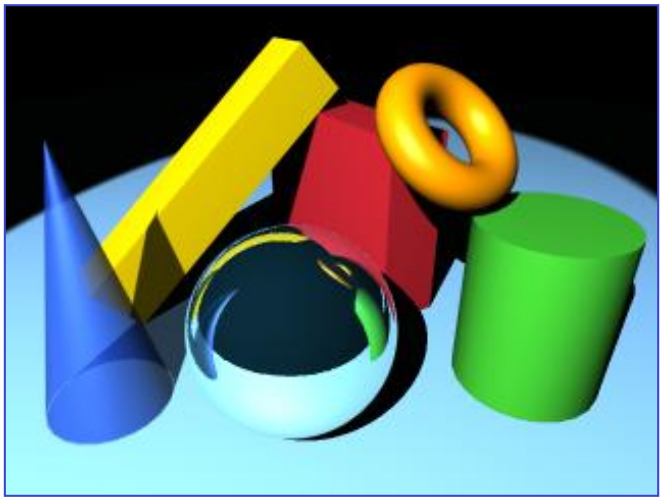
```
in vec3 cv;
out vec4 FragColor;
void main ()
{ FragColor  = vec4 (cv,1);
}
```

Classe 6: contingut

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

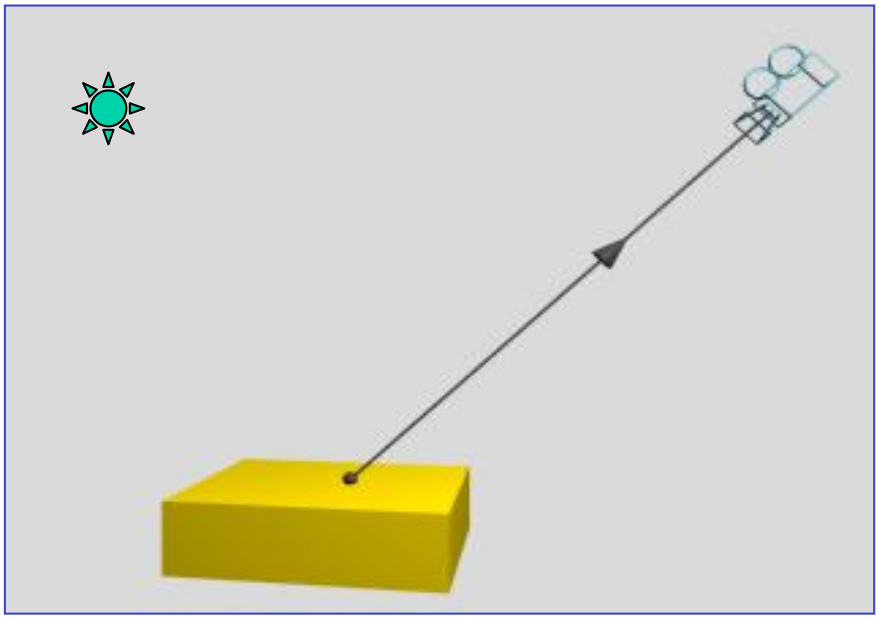
Introducció

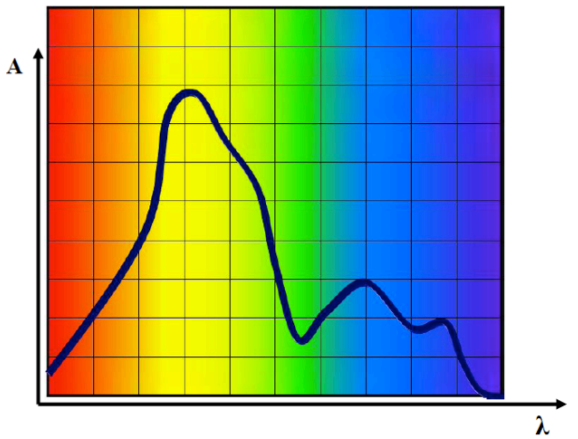
- Els models d'il·luminació simulen el comportament de la llum per determinar el color d'un punt de l'escena.
- Permeten obtenir imatges molt més realistes que pintant cada objecte d'un color uniforme:



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: $I_\lambda(P \rightarrow Obs)$



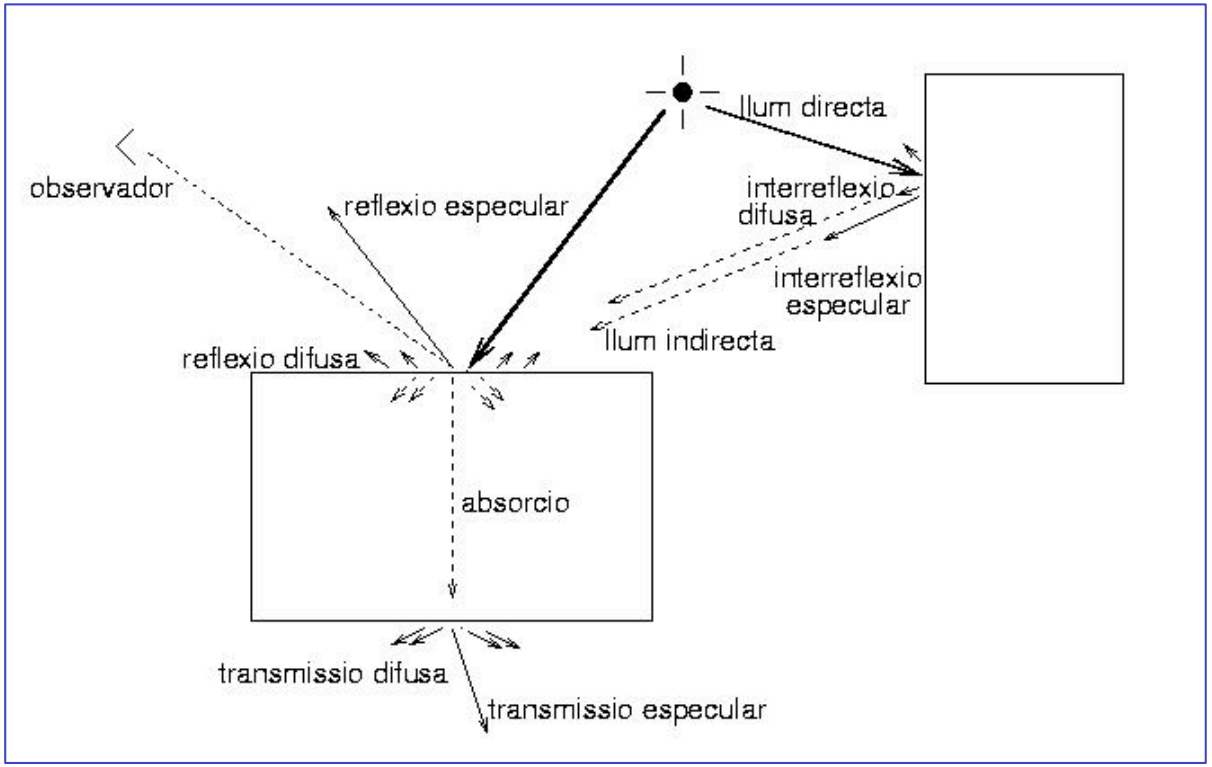


$I_\lambda(P \rightarrow Obs) \quad \lambda \in \{r,g,b\}$

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



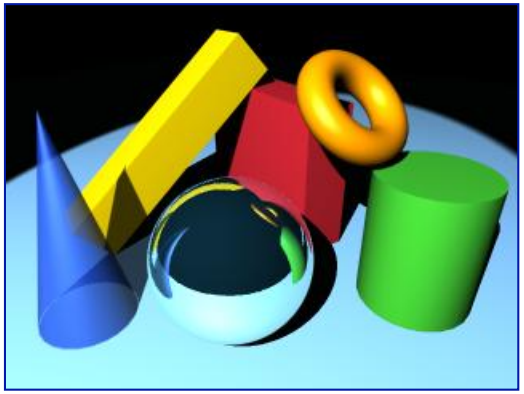
Models d'il·luminació

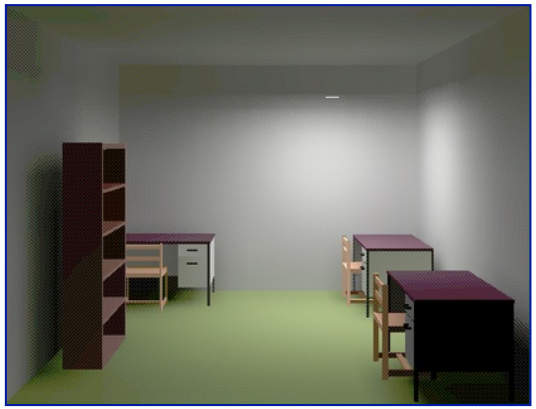
- Els models d'il·luminació simulen les lleis físiques que determinen el color d'un punt.
- El càlcul exacte és computacionalment inviable.
- Una primera simplificació és usar només les energies corresponents a les llums vermella, verda i blava.

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

Models d'il·luminació: Classificació

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



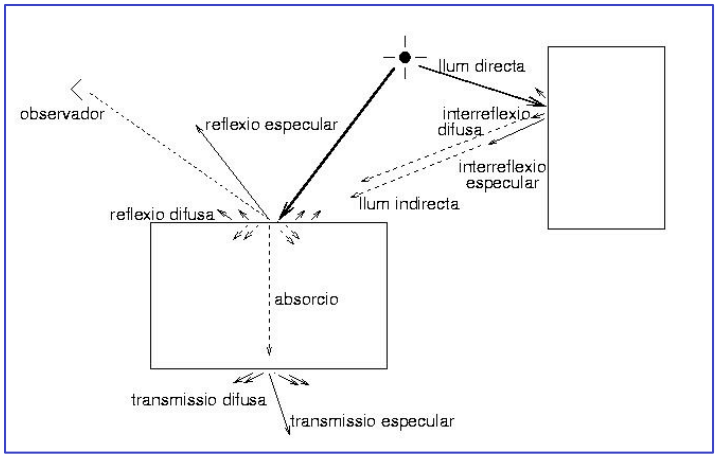


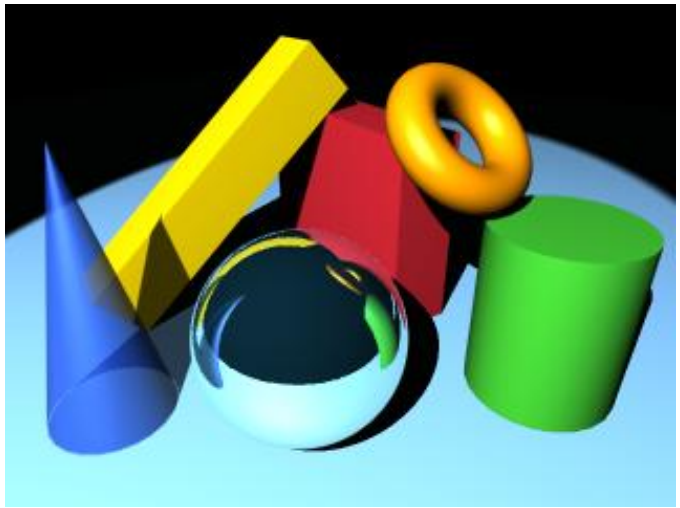
Models locals o empírics

- Només consideren per al calcul del color: el punt **P** en què es calcula, els focus de llum (sempre puntuals) i la posició de l'observador.
- No consideren altres objectes de l'escena (no ombres, no miralls, no transparències).
- Aproximen la transmissió de la llum per fórmules empíriques i les propietats de reflexió dels materials per constants.

Models de traçat de raig

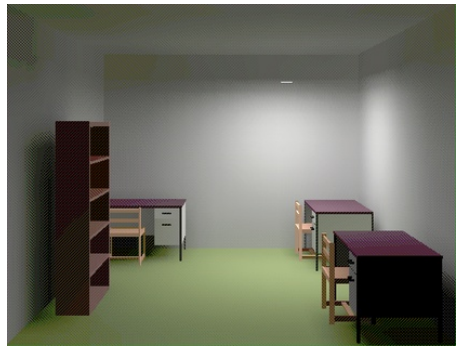
- Els models d'il·luminació de traçat de raig consideren:
  - Focus de llum puntuals
  - Altres objectes existents en l'escena però només transmissions especulars
- Permeten simular ombres, transparències i miralls.
- Són més costosos en càlcul .





Models de radiositat

- Consideren els focus de llum com un objecte qualsevol de l'escena.
- Els objectes només poden produir reflexions difuses pures.
- Com que totes les reflexions són difuses, la radiositat no considera la posició de l'observador.
- Poden modelar ombres i penombres, però no miralls ni transparències.
- Són els més costosos i es basen en l'anàlisi de l'intercanvi d'energia entre tots els objectes de l'escena.



Classe 6: contingut

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

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

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); VM = VM * Rotate (90 , 1, 0, 0); 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	Codi 2
VM= LookAt ((0,80,0),(0,50,0),(0,0,-1));	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); 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?

99. (2016-2017P Q1) Per a què el procés de visualització funcioni correctament pel que respecta a la projecció de la geometria de l'escena, hem de programar obligatòriament els VS (Vertex Shader) i FS (Fragment Shader) de manera que:

a) En el VS es calculin les coordenades de clipping del vèrtex i en el FS les de dipositiu.
b) En el VS es calculin les coordenades d'observador del vèrtex i en el FS les de clipping.
c) En el VS es calculin les coordenades de clipping del vèrtex.
d) En el VS es calculin les coordenades de clipping i en el FS les d'observador.

(2016-2017P Q1) Un estudiant implementa el Vertex Shader oblidant-se de multiplicar els vèrtexs per la viewMatrix, és a dir, el càlcul del gl Position el fa fent:

gl_Position = proj * TG * vec4(vertex, 1.0);

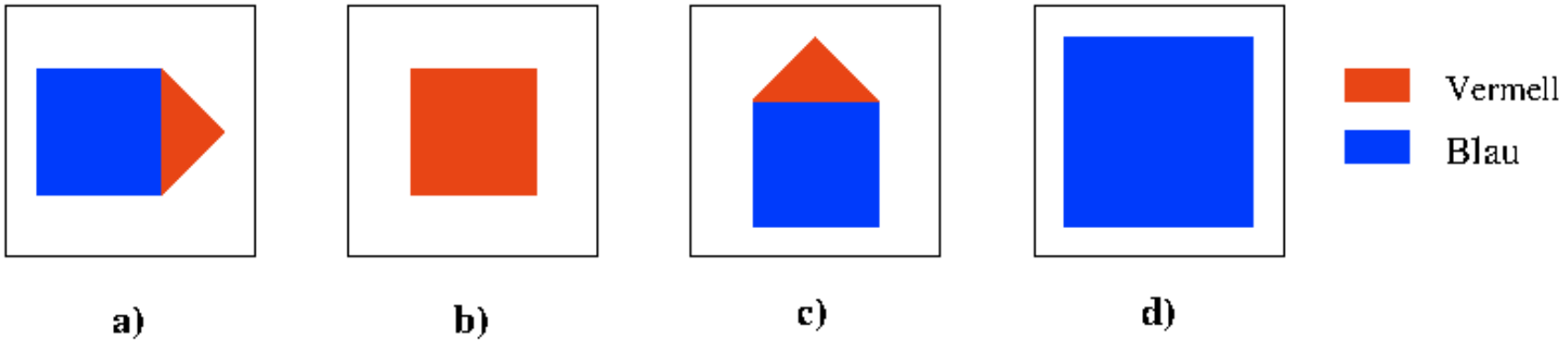
on TG és la modelMatrix i proj és la projectMatrix d'una òptica perspectiva. Quina afirmació és la correcta?

a) El volum de visió queda definit pels punts (-1,-1,-1) i (1,1,1) en coordenades de l'aplicació (SCA).
b) Té el mateix efecte que tenir OBS = (0,0,0), VRP = (0,0,-10) i up = (0,1,0).
c) No podem conèixer la posició de la càmera.
d) Cap de les altres respostes és correcta.

(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?



Hi ha una llista de molts exercicis, alguns a fer:
33, 58, 63, 53, 67, 73, 85,…