

# Procés de visualització projectiu

C. Andujar

Sep 2020

# **SISTEMES DE COORDENADES I TRANSFORMACIONS GEOMÈTRIQUES**

# Paradigma projectiu simplificat



# Informació geomètrica al pipeline

vertices.push\_back( QVector3D(**0, 1, 0**) );  Coordenades d'un vèrtex

...

normals.push\_back( QVector3D(**0, 0, 1**) );  Components de la normal

setUniformValue("lightPosition", QVector3D(**0, 0, 0**));

setUniformValue("lightDir", QVector3D(**0, 1, 0**));  Posició d'una llum

Direcció d'una llum

# Informació geomètrica al pipeline

- Punts
- Vectors
  - Normals (perpendiculars a la superfície)
  - Resta de vectors

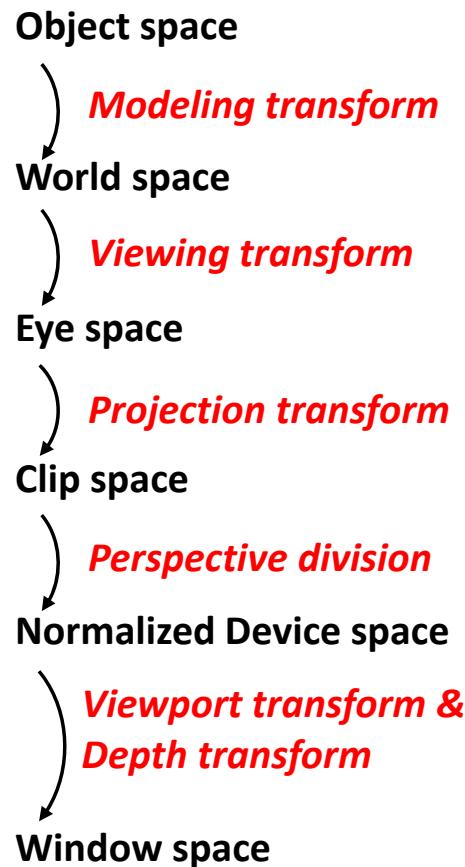
# Coordenades homogènies

Punts

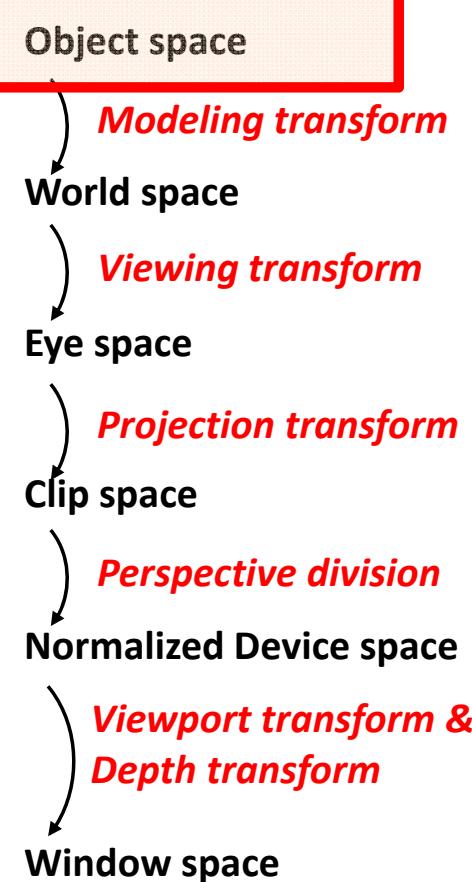
Vectors

# **TRANSFORMACIÓ DE PUNTS**

# Sistemes de coordenades



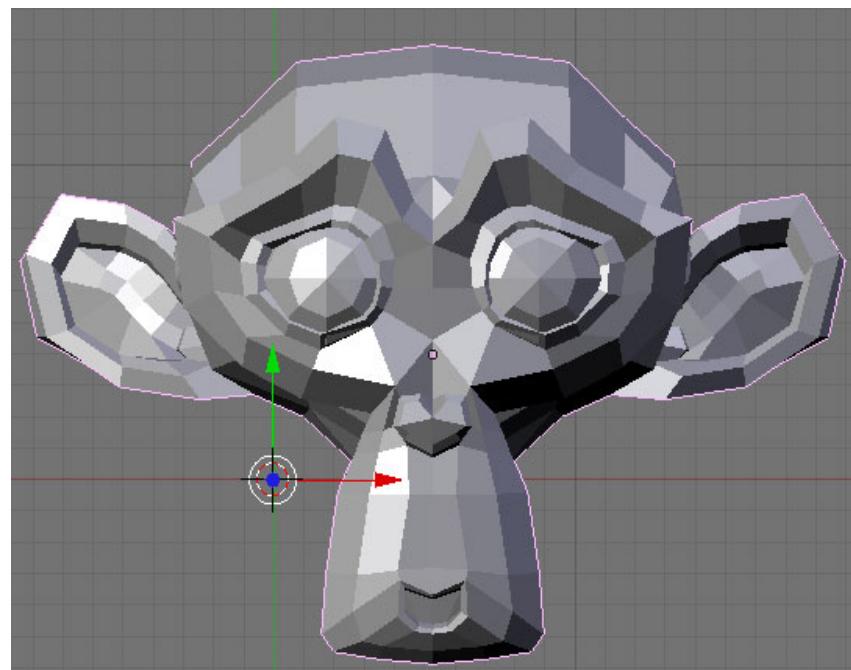
# Sistemes de coordenades



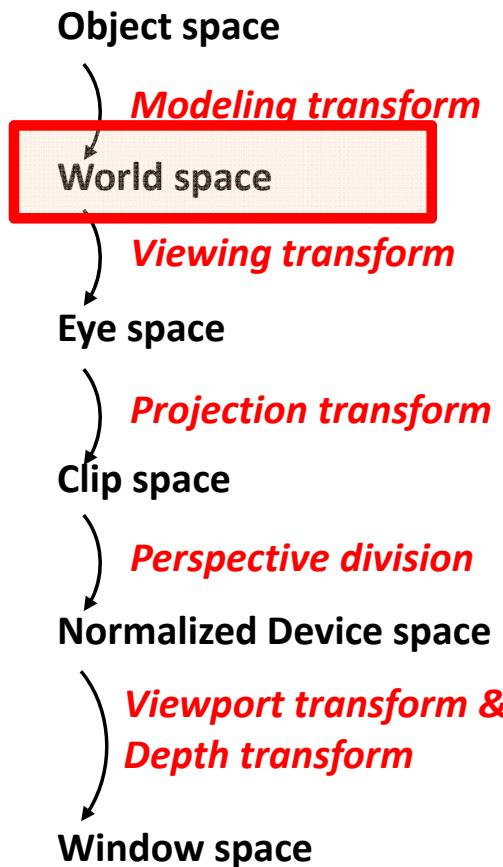
**Object space**  $(x_m, y_m, z_m, w_m)$

*Model space, SC del model, SC de l'objecte*

- *SC utilitzat per modelar l'objecte.*
- $w_m$  normalment serà 1.0 (punts)



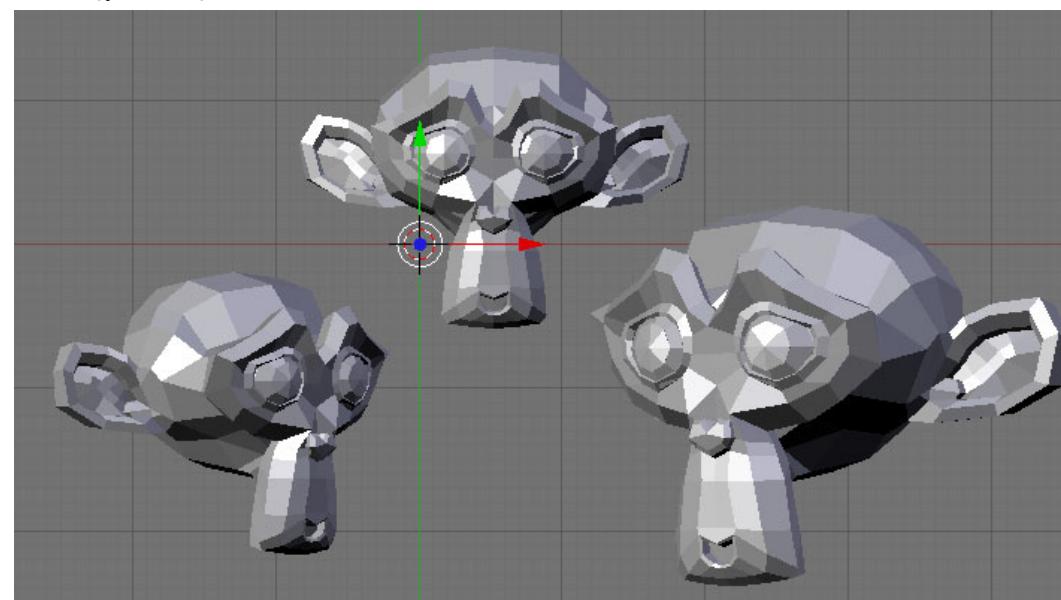
# Sistemes de coordenades



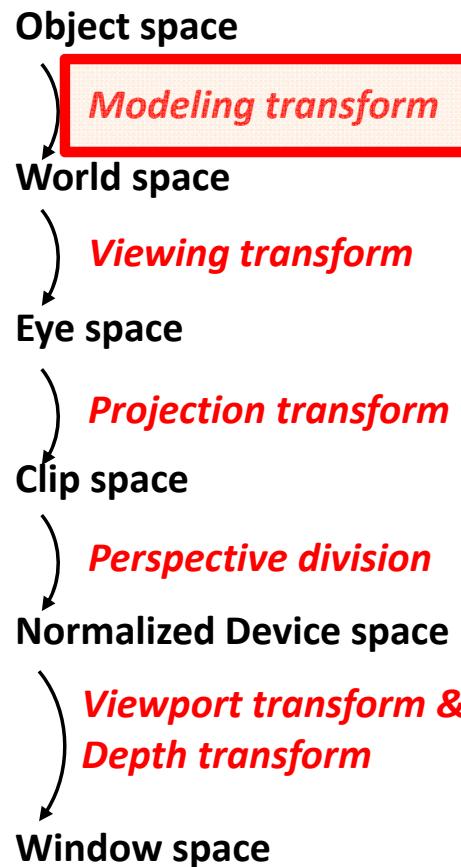
**World space ( $x_a, y_a, z_a, w_a$ )**

*SC de mon, SC de l'aplicació*

- *SC utilitzat per representar l'escena*
- *La transformació de modelat sovint preserva la component homogènia i per tant,  $w_a$  normalment serà 1.0 (punts)*



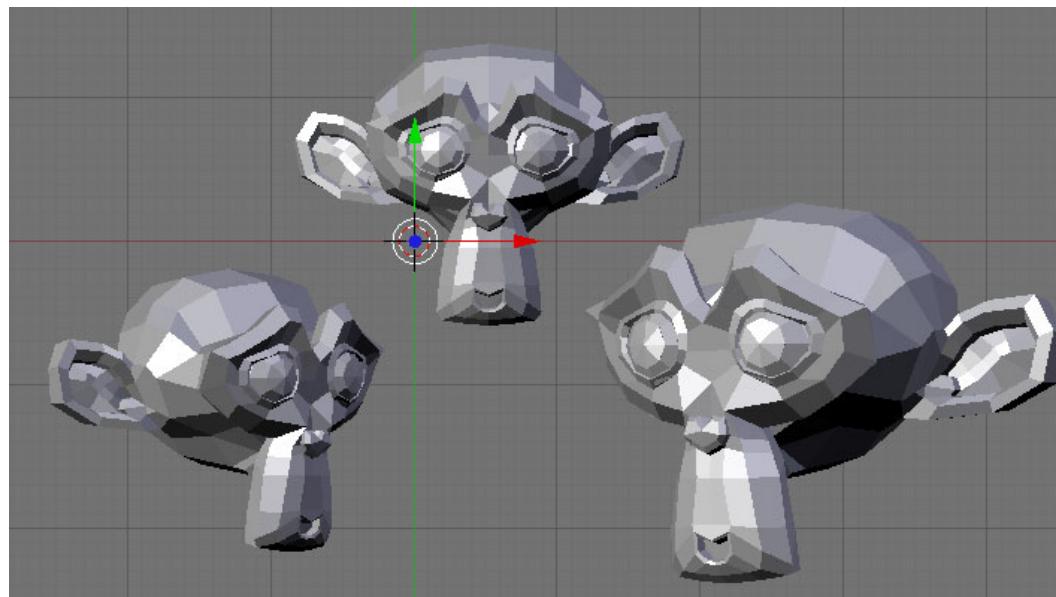
# Sistemes de coordenades



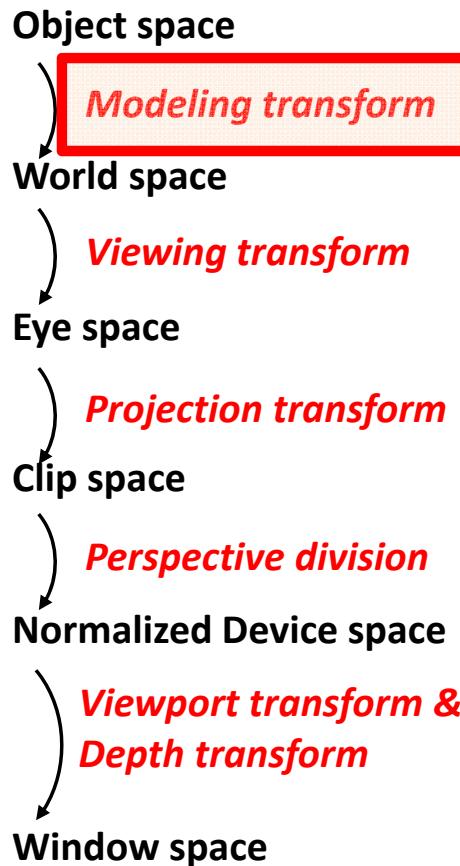
## Modeling transform

Transformació de modelat

- Aquesta transformació situa cada instància d'un objecte en relació a l'escena.
- Sovint és la identitat, o una composició de translacions, rotacions i escalats



# Sistemes de coordenades



**Modeling transform**  
*Transformació de modelat*

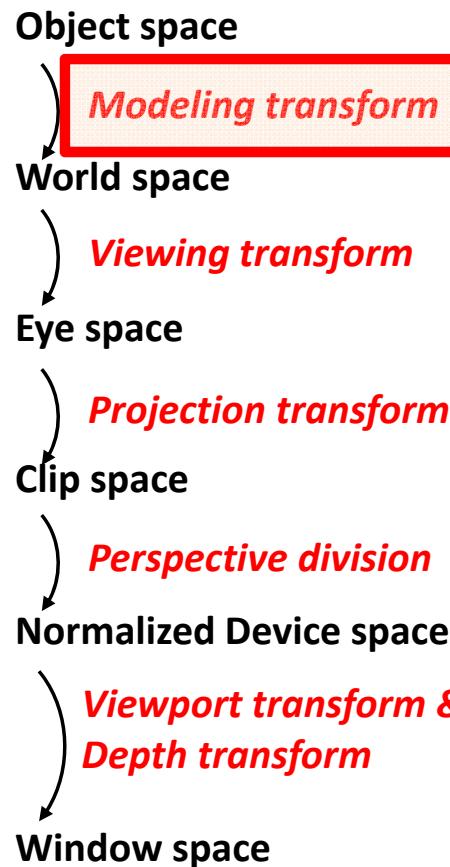
$$\text{Translate}(t_x, t_y, t_z) \quad T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Scale}(s_x, s_y, s_z) \quad T = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Rotate}(a, x, y, z) \quad T = \begin{bmatrix} x^2d + c & xyd - zs & xzd + ys & 0 \\ yxd + zs & y^2d + c & yzd - xs & 0 \\ xzd - ys & yzd + xs & z^2d + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c=cos(a), s=sin(a), d=1-cos(a)

# Sistemes de coordenades



## Modeling transform

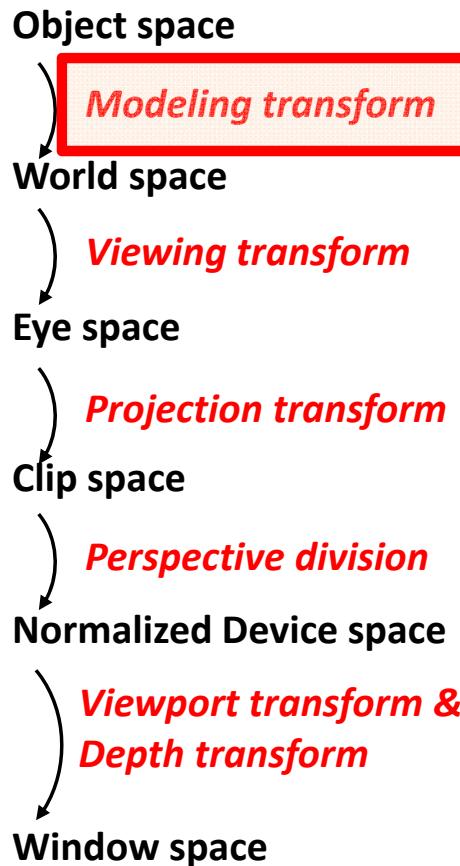
Transformació de modelat

$$\text{glRotate}^*(\alpha, 1, 0, 0): \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate}^*(\alpha, 0, 1, 0): \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate}^*(\alpha, 0, 0, 1): \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

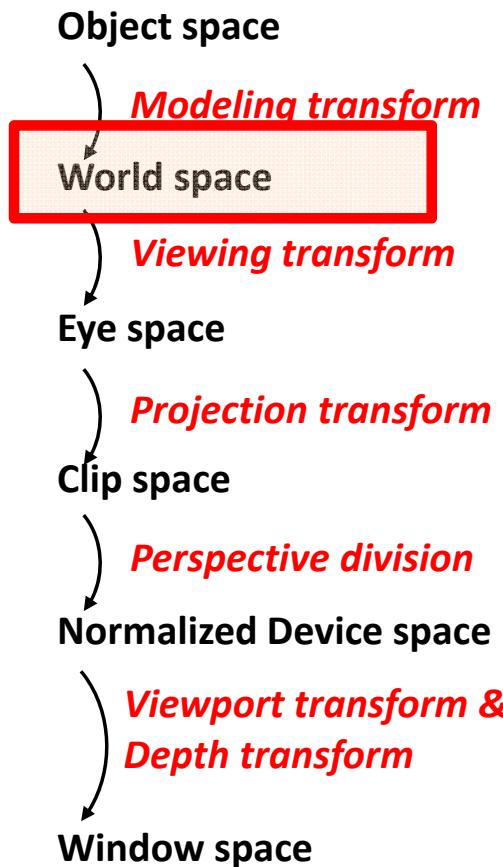
# Sistemes de coordenades



- Punts: matriu  $4 \times 4$  multiplicada pel punt (l'ordre és important!)

$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ w_a \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

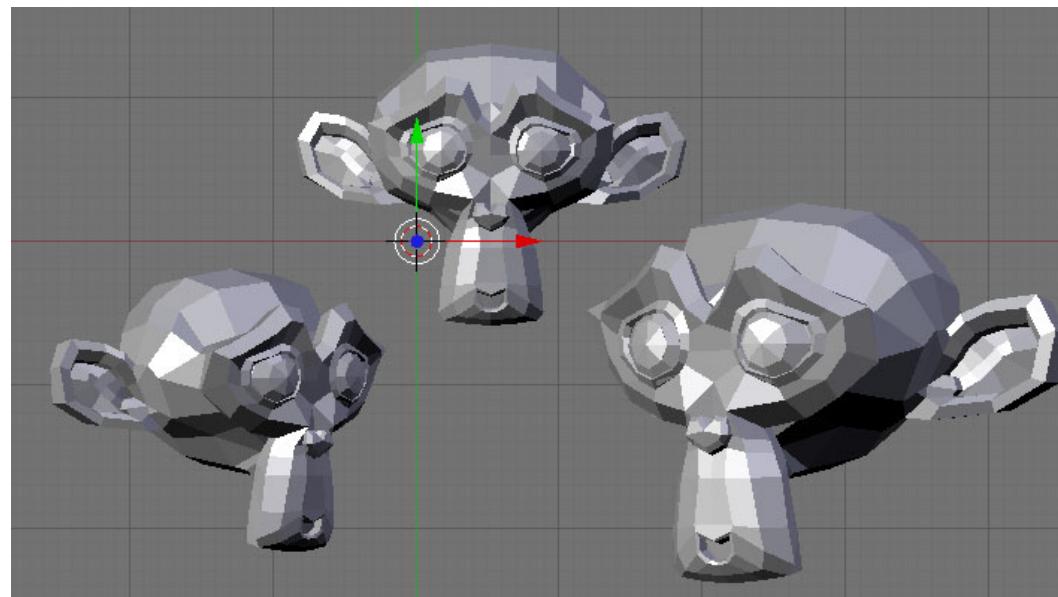
# Sistemes de coordenades



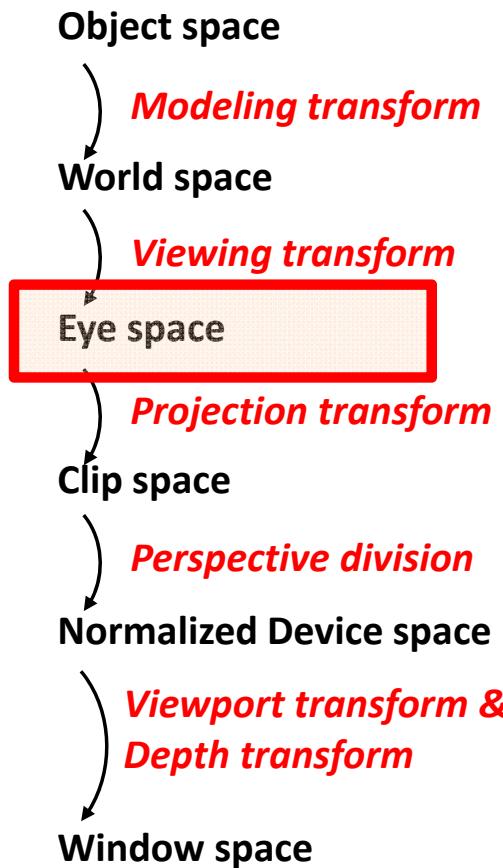
**World space ( $x_a, y_a, z_a, w_a$ )**

SC de mon, SC de l'aplicació

- SC utilitzat per representar l'escena
- La transformació de modelat sovint preserva la component homogènia i per tant,  $w_a$  normalment serà 1.0 (punts)



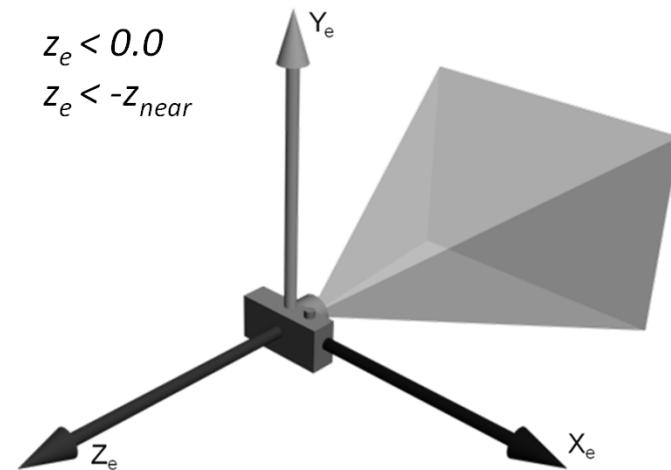
# Sistemes de coordenades



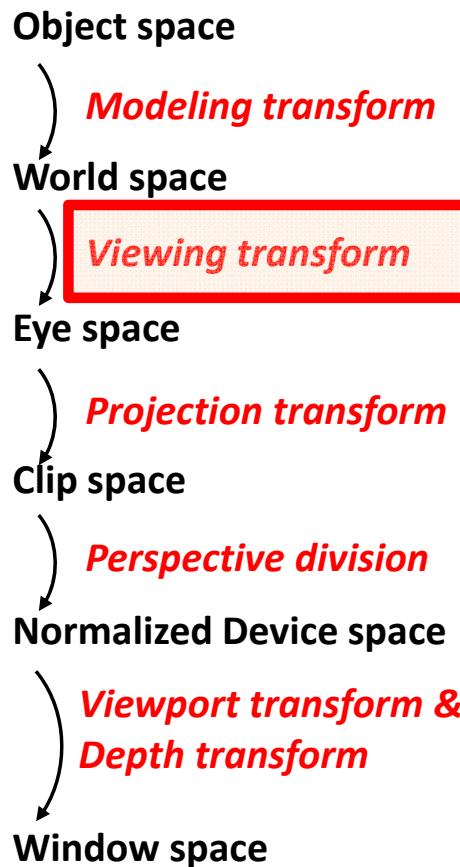
**Eye space** ( $x_e, y_e, z_e, w_e$ )

*SC de l'observador, SC de la càmera*

- *SC associat a la càmera*
- *La transformació de visualització sovint preserva la component homogènia i per tant,  $w_e$  normalment serà 1.0 (punts)*
- *Si la càmera és perspectiva, per tal que el punt sigui visible...*



# Sistemes de coordenades

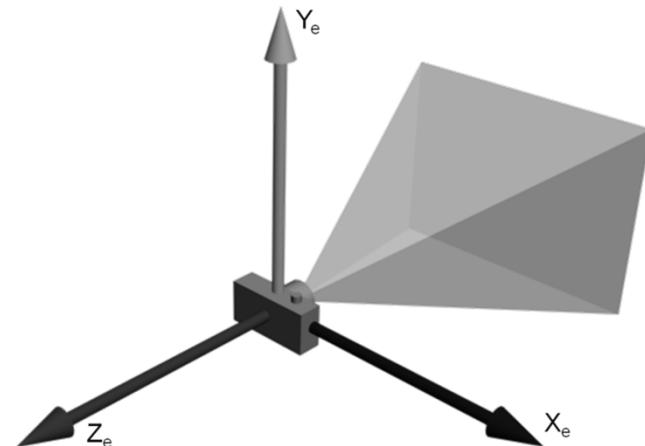


## Viewing transform

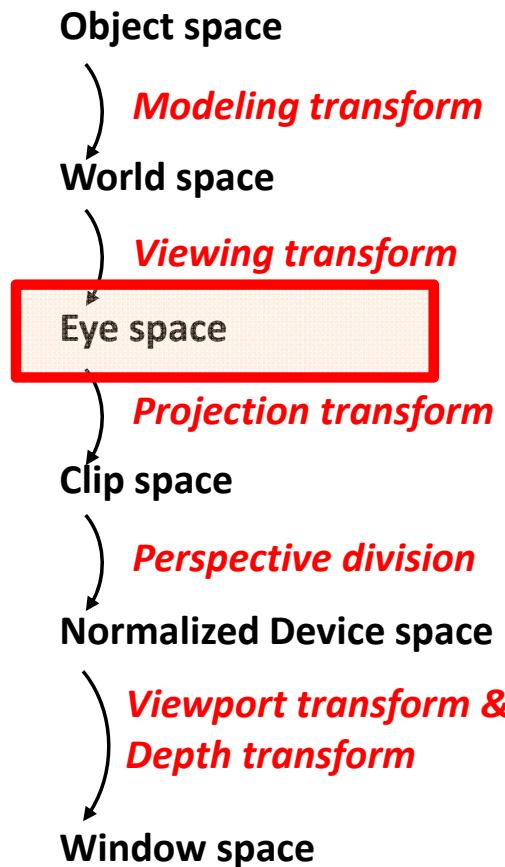
Transformació de visualització, transformació de càmera

- Fa un canvi al sistema de referència de la càmera.
- Depèn de la posició i orientació de la càmera
- Sovint es defineix amb crides tipus **lookAt**, o amb una composició de translacions i rotacions:

- *lookAt(eye, target, up)*
- $T(0,0,-d)*R_z(-\varphi)*R_x(\Theta)*R_y(-\Psi)*T(-VRP)$



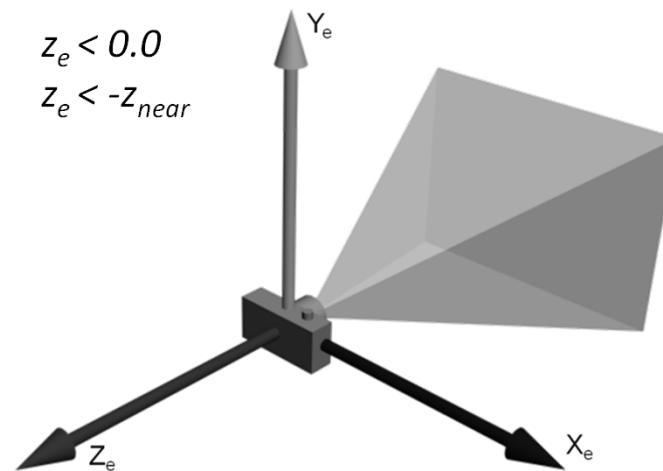
# Sistemes de coordenades



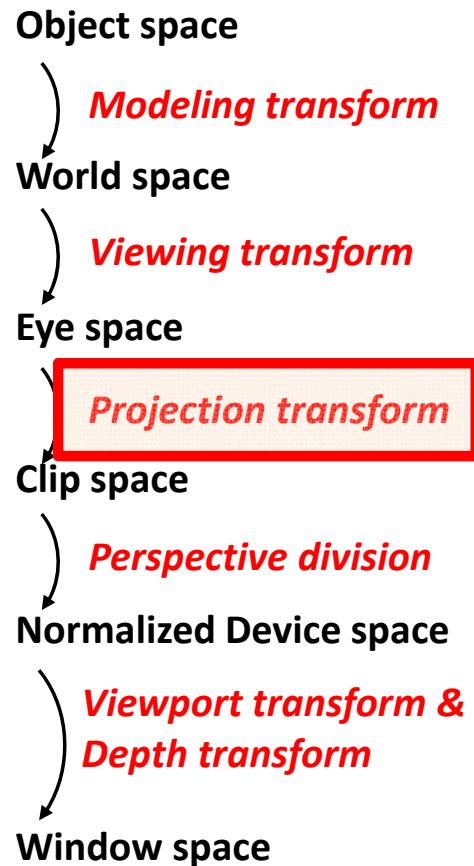
**Eye space** ( $x_e, y_e, z_e, w_e$ )

*SC de l'observador, SC de la càmera*

- *SC associat a la càmera*
- *La transformació de visualització sovint preserva la component homogènia i per tant,  $w_e$  normalment serà 1.0 (punts) o 0.0 (vectors)*
- *Si la càmera és perspectiva, per tal que el punt sigui visible...*



# Sistemes de coordenades

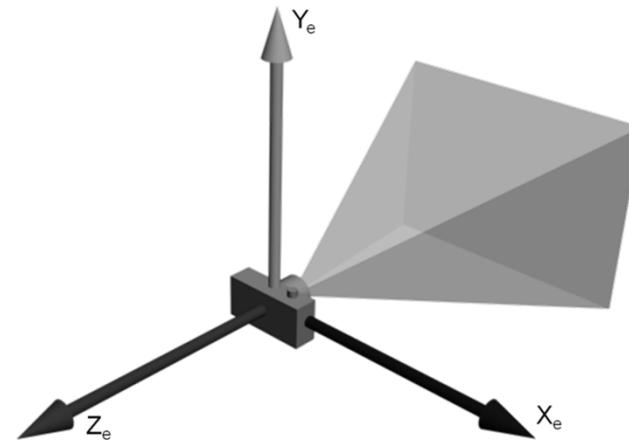


## Projection transform

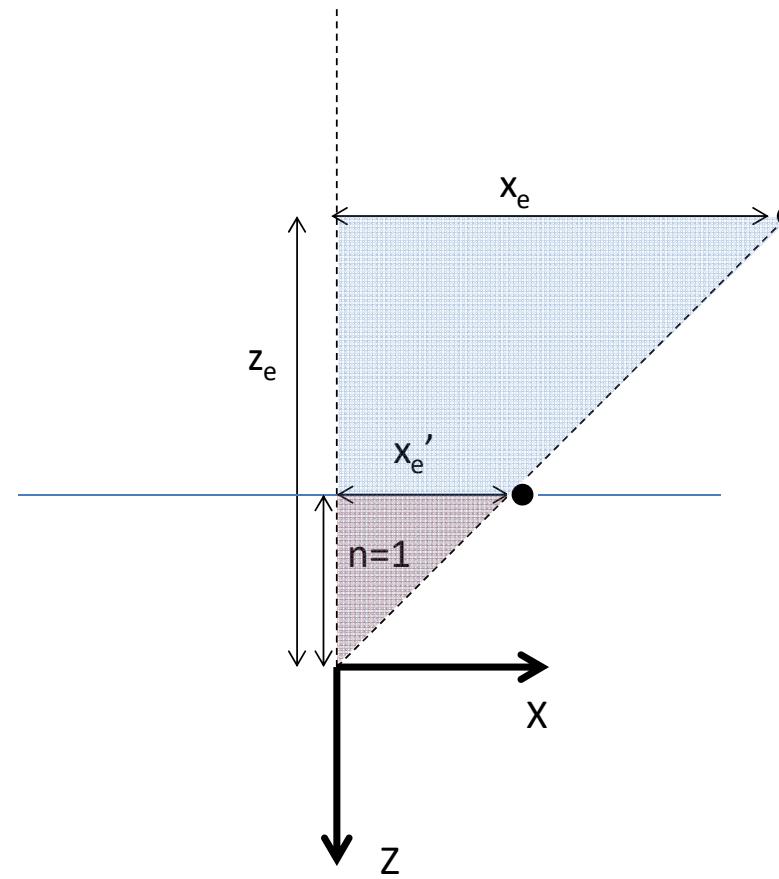
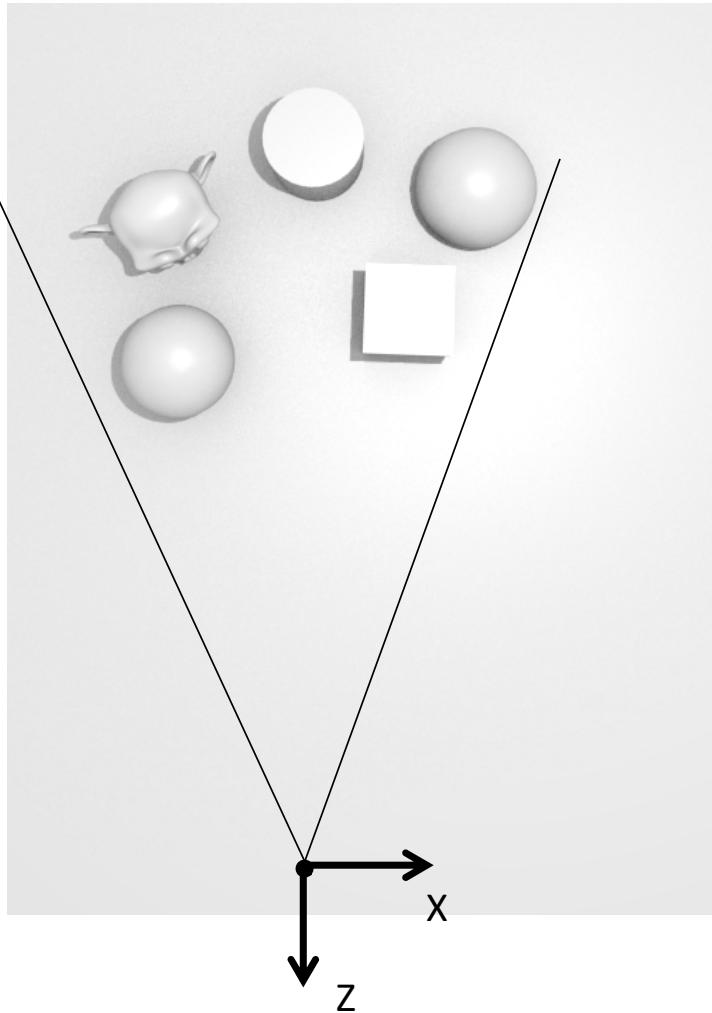
## *Transformació de projecció*

- Depèn de la forma de la piràmide de visió i per tant del tipus de càmera (perspectiva, axonomètrica)
  - Sovint es defineix amb crides del tipus ***perspective, frustum, ortho***

*perspective(fovy, aspect, near, far)*



# Projecció i divisió per $z_e$



`perspective(90, 1, 1, 11);`

$$\begin{bmatrix} \frac{\cot \frac{fovy}{2}}{aspect} & 0 & 0 & 0 \\ 0 & \frac{\cot \frac{fovy}{2}}{2} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2*n*f}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix} \xrightarrow{\substack{fovy=90, \text{ aspect}=1 \\ n=1, f=11}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Pas eye space  $\rightarrow$  clip space

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \begin{bmatrix} x_e \\ y_e \\ -1.2z_e - 2.2 \\ -z_e \end{bmatrix}$$

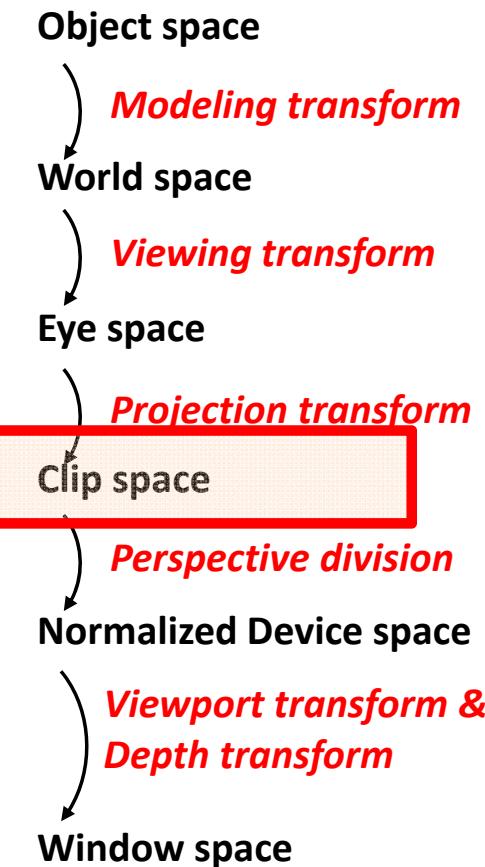
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_e \\ y_e \\ -1 \\ 1 \end{bmatrix}$$

Punt sobre el pla znear

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ -11 \\ 1 \end{bmatrix} = \begin{bmatrix} x_e \\ y_e \\ 11 \\ 11 \end{bmatrix}$$

Punt sobre el pla zfar

# Sistemes de coordenades



**Clip space**  $(x_c, y_c, z_c, w_c)$

Coordenades de clipping, coordenades de retallat

- Si un punt és interior al frustum,

$$-w_c \leq x_c \leq w_c$$

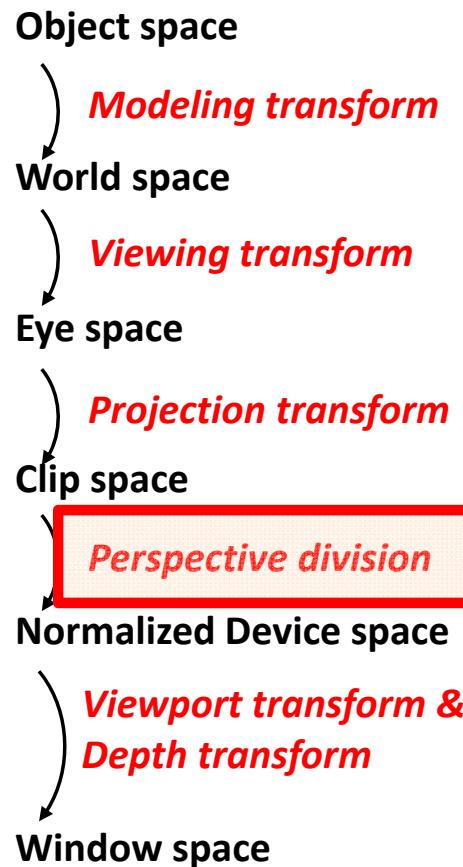
$$-w_c \leq y_c \leq w_c$$

$$-w_c \leq z_c \leq w_c$$

- Si la càmera és perspectiva, llavors

$$w_c = -z_e$$

# Sistemes de coordenades



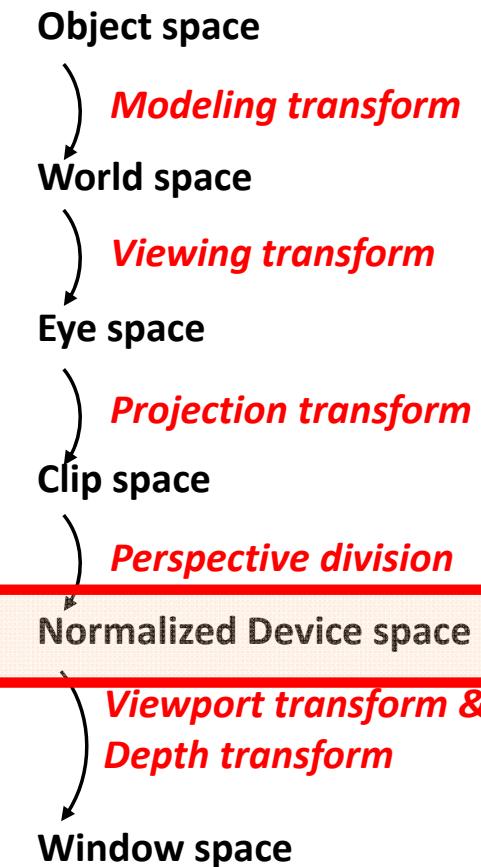
## Perspective division

Divisió de perspectiva

- Simplement és el pas de coordenades homogènies a 3D
- Es divideix cada coordenada per la coord homogènia

$$(x_c, y_c, z_c, w_c) \rightarrow (x_c/w_c, y_c/w_c, z_c/w_c)$$

# Sistemes de coordenades



**Normalized device space** ( $x_n, y_n, z_n$ )

*NDC, cordenades normalitzades*

- Si un punt és interior al frustum,

$$-1 \leq x_n \leq 1$$

$$-1 \leq y_n \leq 1$$

$$-1 \leq z_n \leq 1$$

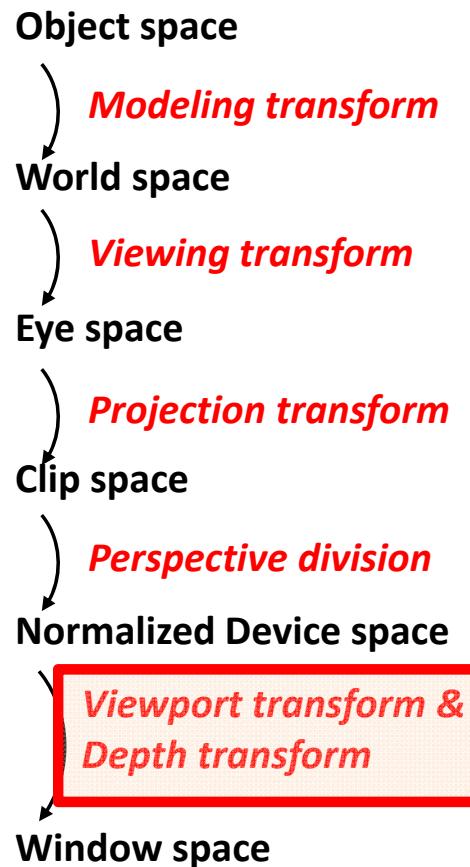
- Els punts situats sobre  $z_{\text{near}}$  tenen

$$z_n = -1$$

- Els punts situats sobre  $z_{\text{far}}$  tenen

$$z_f = +1$$

# Sistemes de coordenades



## Viewport transformation

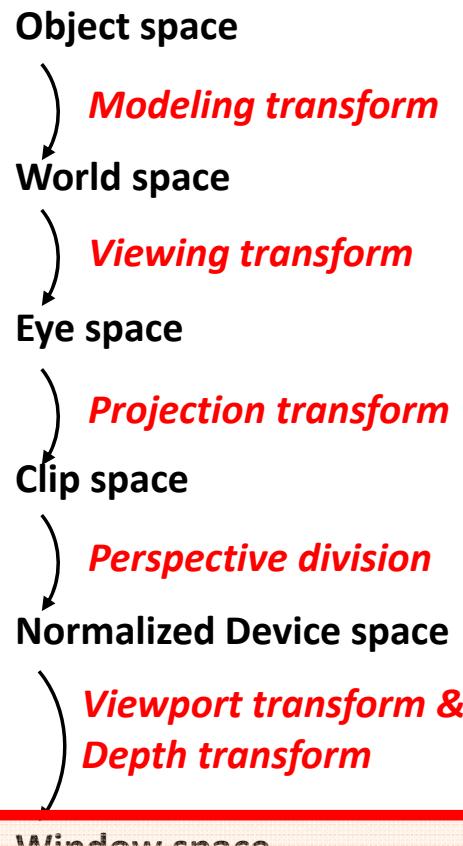
Transformació mon-dispositiu

- Depèn del viewport definit amb `glViewport`

## Depth range transformation

- Depèn de l'interval definit amb `glDepthRange` (per defecte  $[0,1]$ )

# Sistemes de coordenades



## Window space ( $x_d, y_d, z_d$ )

Cordenades de finestra, coordenades de dispositiu

- Si un punt és interior al frustum,

$$0 \leq x_d \leq w$$

$$0 \leq y_d \leq h$$

$$0 \leq z_d \leq 1$$

- Els punts situats sobre znear tenen

$$z_d = 0$$

- Els punts situats sobre zfar tenen

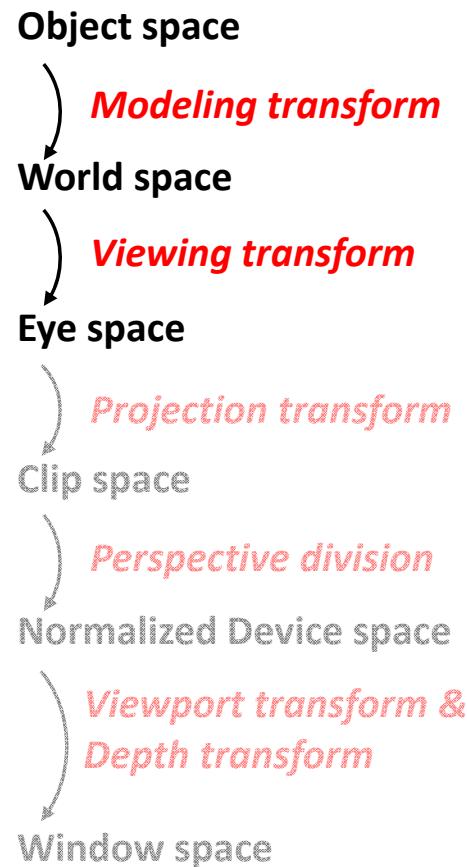
$$z_d = 1$$

- Quan més endavant es generin fragments,

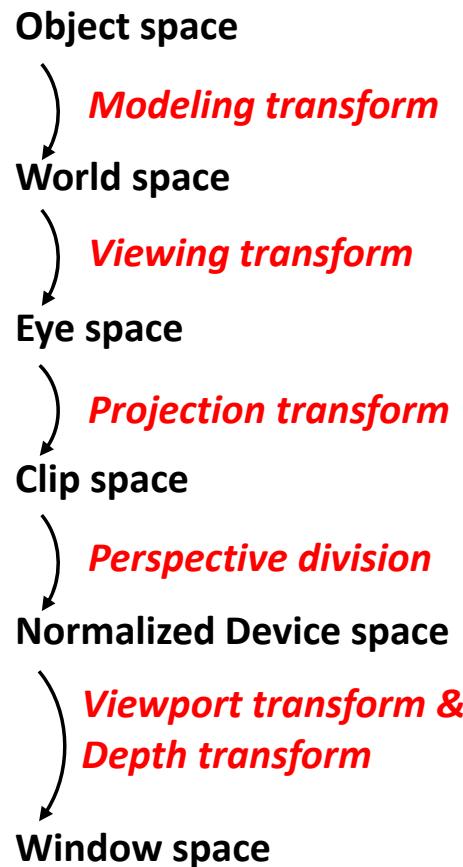
$$\text{gl_FragCoord.w} = 1/w_c = -1/z_e$$

# **TRANFORMACIÓ DE VECTORS (EXCEPTE NORMALS)**

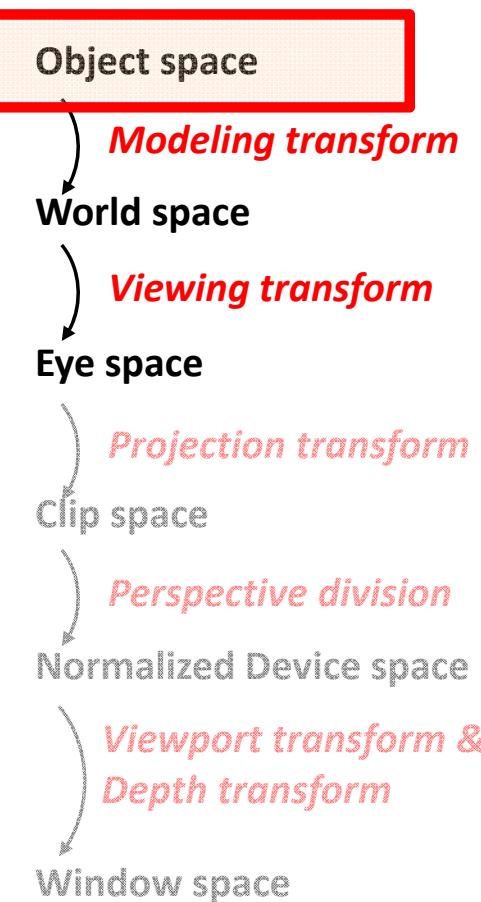
# Sistemes de coordenades



# Sistemes de coordenades



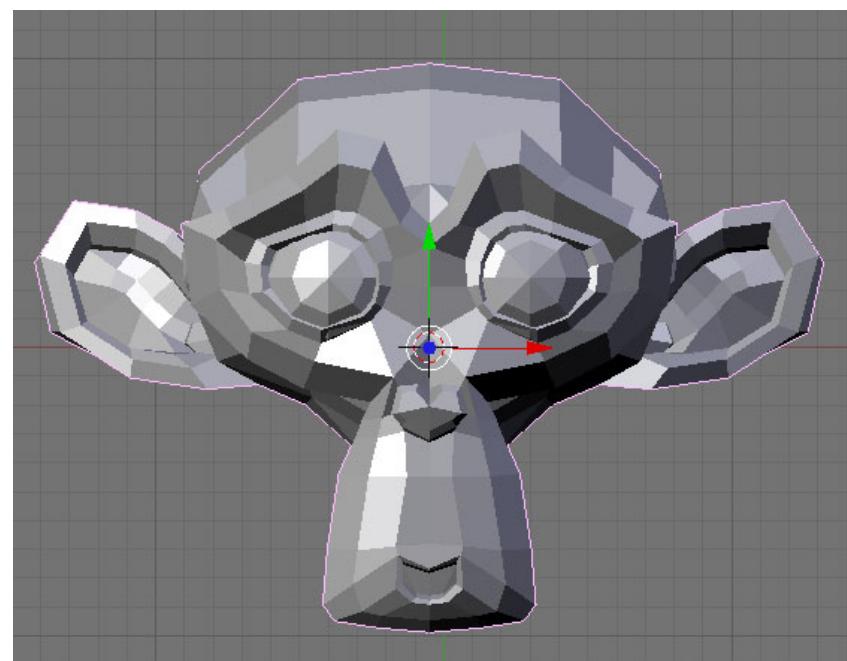
# Sistemes de coordenades



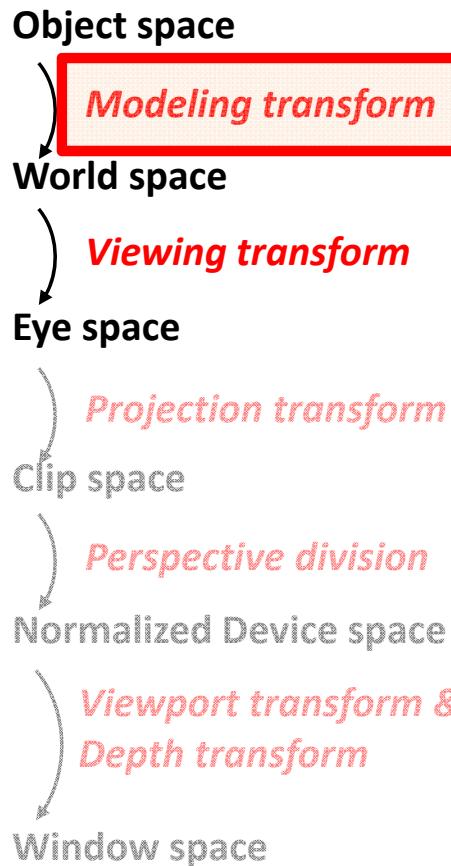
**Object space**  $(x_m, y_m, z_m, w_m)$

*Model space, SC del model, SC de l'objecte*

- *SC utilitzat per modelar l'objecte.*
- $w_m$  normalment serà 1.0 (punts) o 0.0 (vectors)



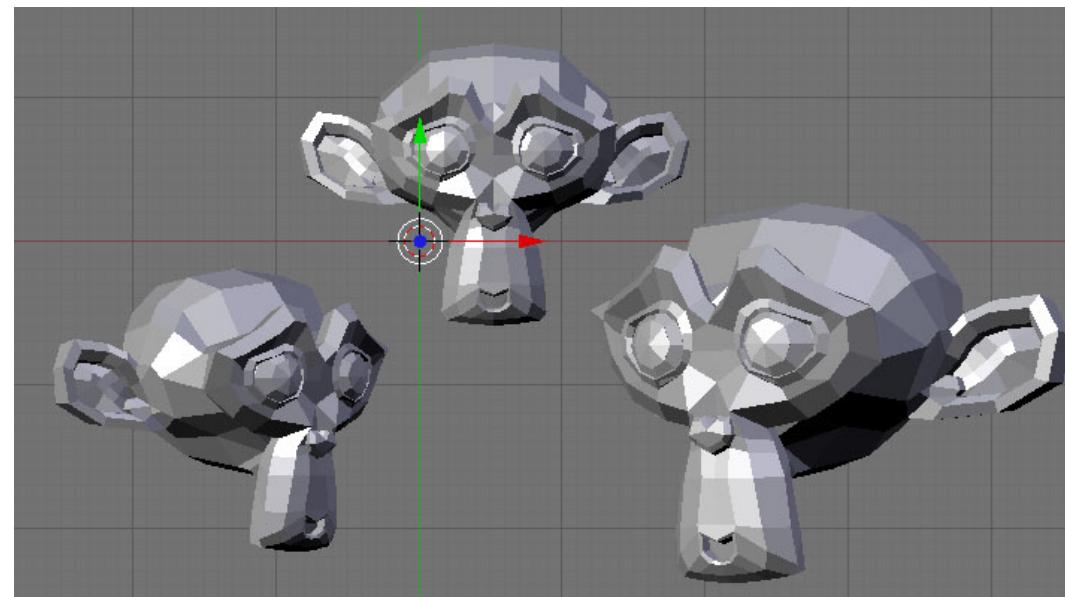
# Sistemes de coordenades



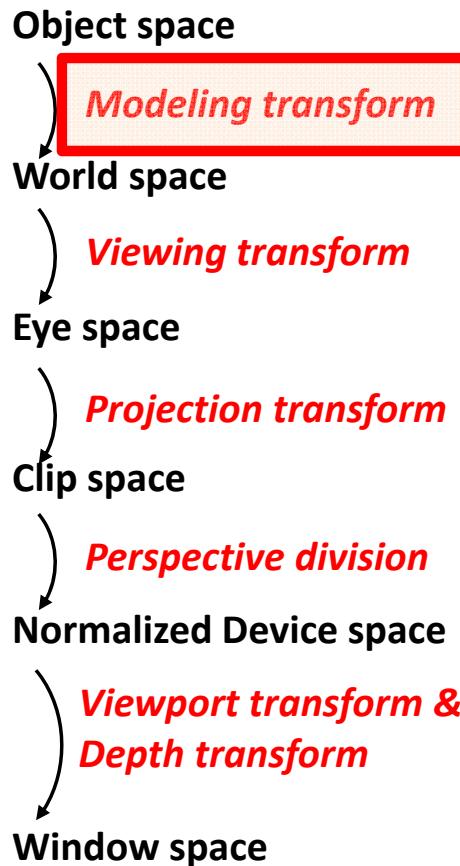
## Modeling transform

Transformació de modelat

- Aquesta transformació situa cada instància d'un objecte en relació a l'escena.
- Sovint és la identitat, o una composició de translacions, rotacions i escalats



# Sistemes de coordenades



**Modeling transform**  
Transformació de modelat

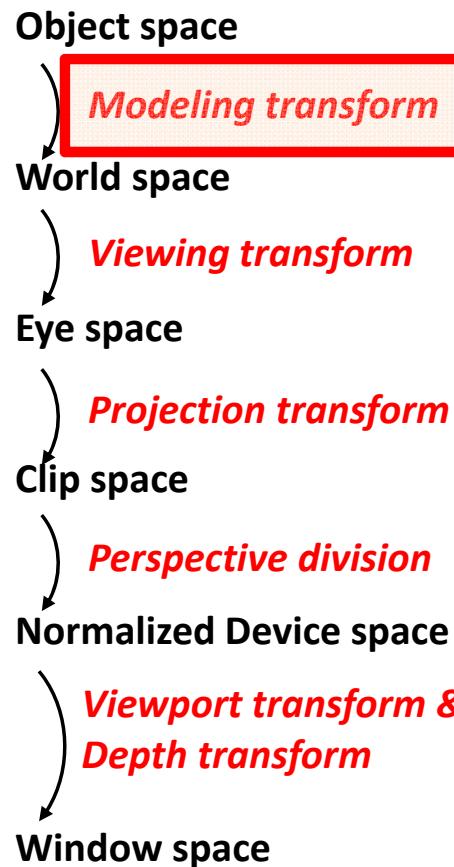
$$\text{translate}(t_x, t_y, t_z) \quad T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{scale}(s_x, s_y, s_z) \quad T = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{rotate}(a, x, y, z) \quad T = \begin{bmatrix} x^2d + c & xyd - zs & xzd + ys & 0 \\ yxd + zs & y^2d + c & yzd - xs & 0 \\ xzd - ys & yzd + xs & z^2d + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c=cos(a), s=sin(a), d=1-cos(a)

# Sistemes de coordenades



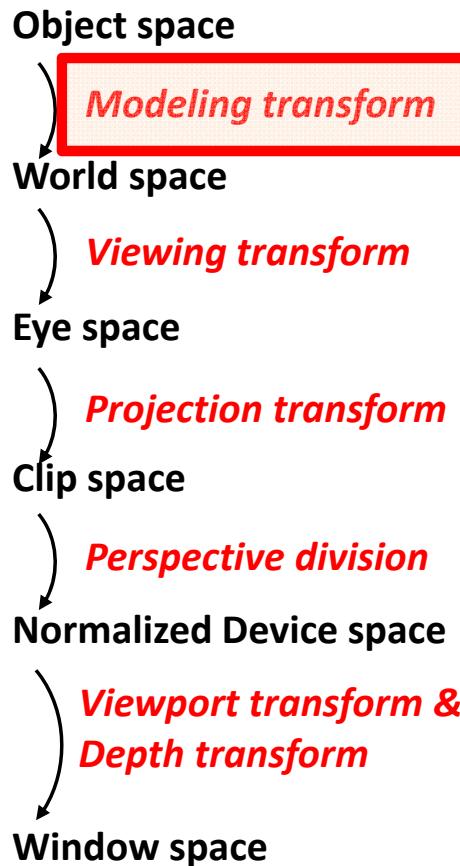
**Modeling transform**  
*Transformació de modelat*

$$\text{glRotate}^*(\alpha, 1, 0, 0): \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate}^*(\alpha, 0, 1, 0): \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate}^*(\alpha, 0, 0, 1): \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Sistemes de coordenades



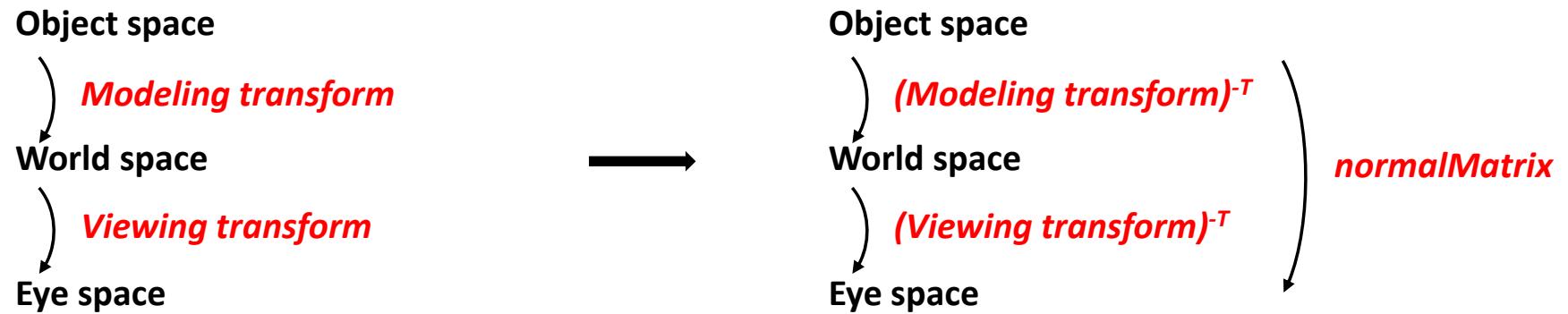
- Punts, vectors: matriu  $4 \times 4$  multiplicada pel punt/vector

$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ w_a \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

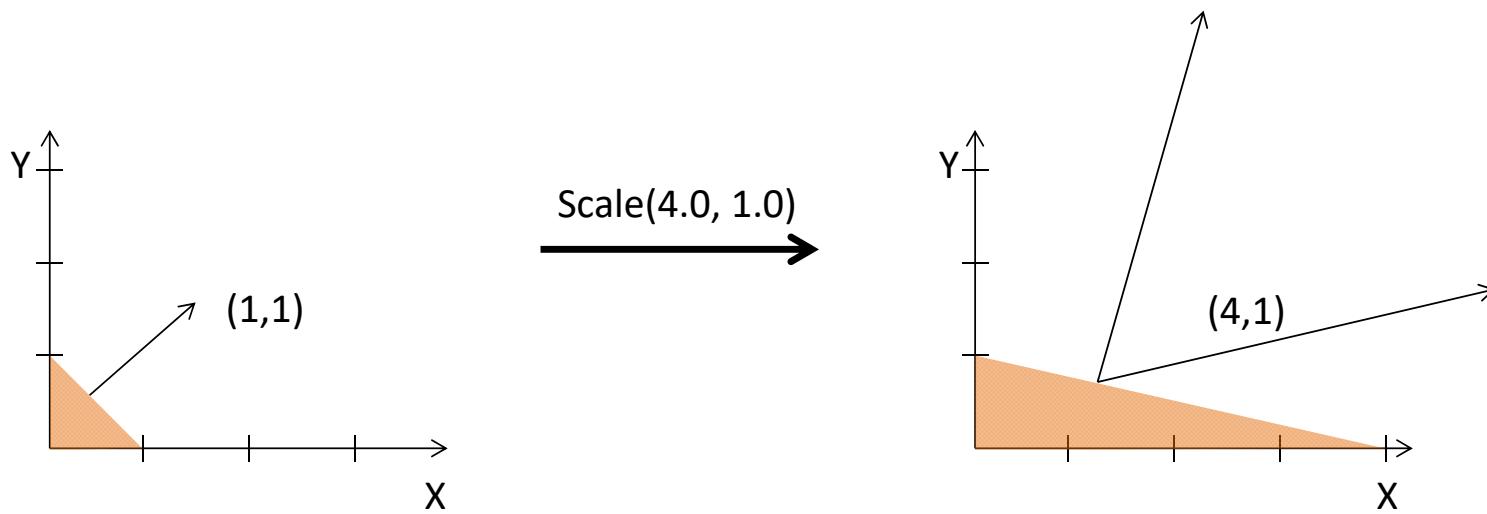
$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ w_a \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 0 \end{bmatrix}$$

# **TRANFORMACIÓ NORMALS**

# Sistemes de coordenades



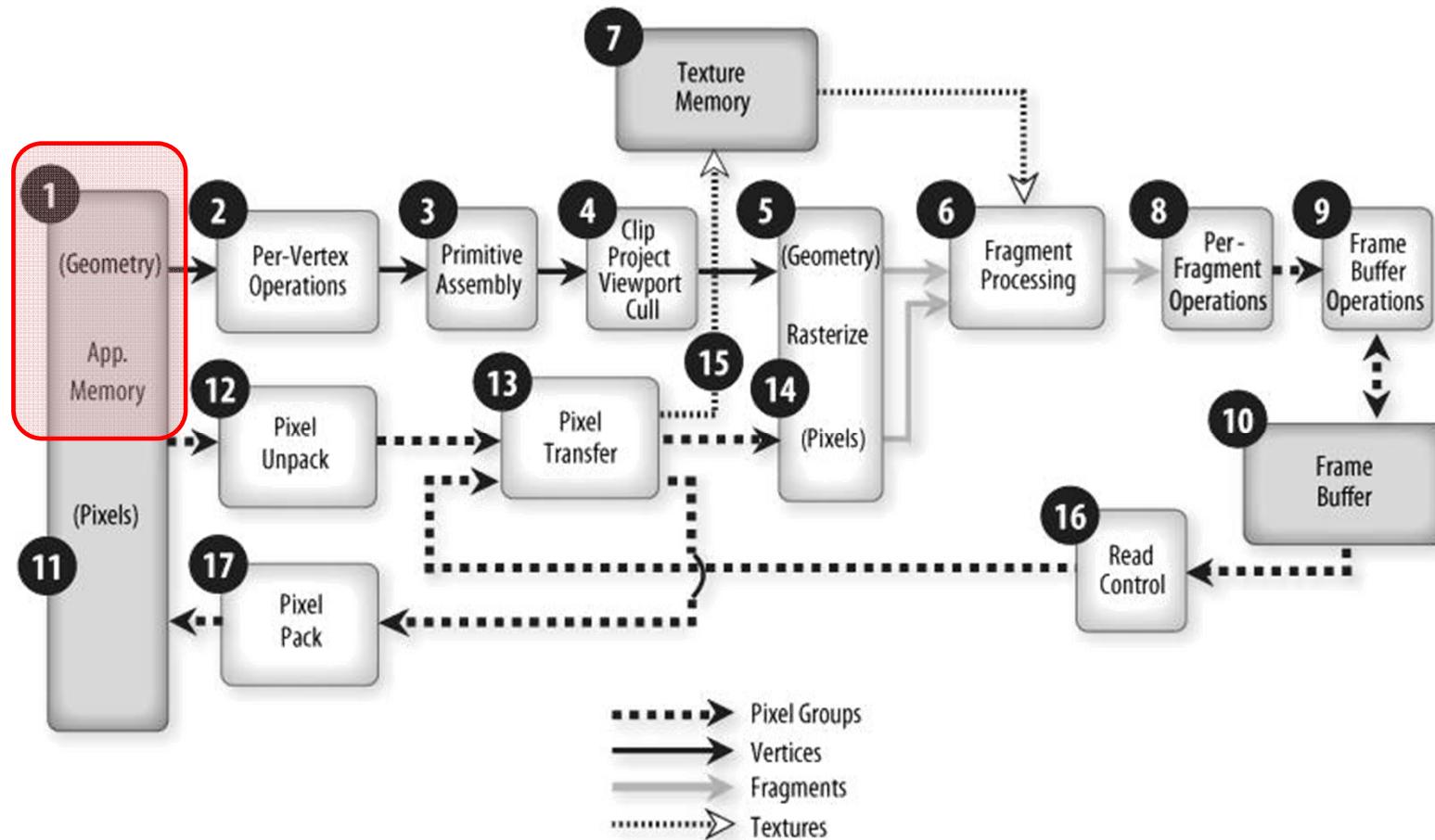
# Sistemes de coordenades



# Transformació de normals

# **PIPELINE OPENGL**

# Pipeline OpenGL



# Pipeline OpenGL

## 1. Dibuix de primitives

Les primitives (punts, línies, polígons...) es poden pintar:

- Vèrtex a vèrtex: *glBegin*, *glVertex*, *glEnd* (compatibility profile)
- Vertex array object: *glDrawArrays*, *glDrawElements*...

# Exemple (*compatibility*)

```
glBegin(GL_POLYGON);
    glNormal3f(nx0, ny0, nz0);
    glVertex3f(x0, y0, z0);           Coords i normals en
    glNormal3f(nx1, ny1, nz1);       object space
    glVertex3f(x1, y1, z1);
...
glEnd();
```

# Exemple (*compatibility/core*)

```
// createBuffers
glm::vec3 Vertices[...]; // Tres vèrtexs amb X, Y i Z
Vertices[0] = glm::vec3(-1.0, -1.0, 0.0);
...
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);

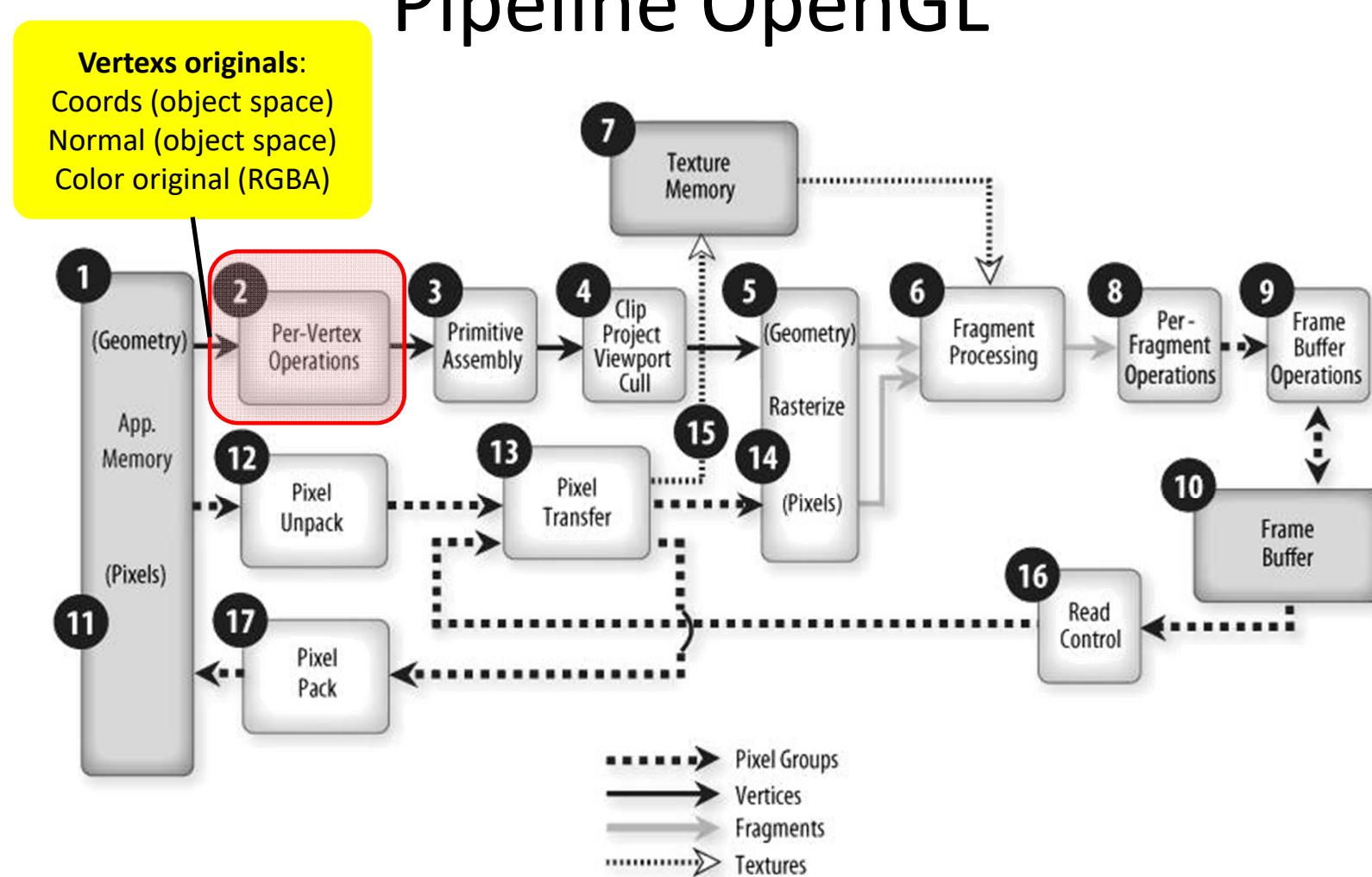
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
 glEnableVertexAttribArray(0);

glBindVertexArray(0);

// paintGL
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, numVerts);
glBindVertexArray(0);
```

Coords i normals en  
*object space*

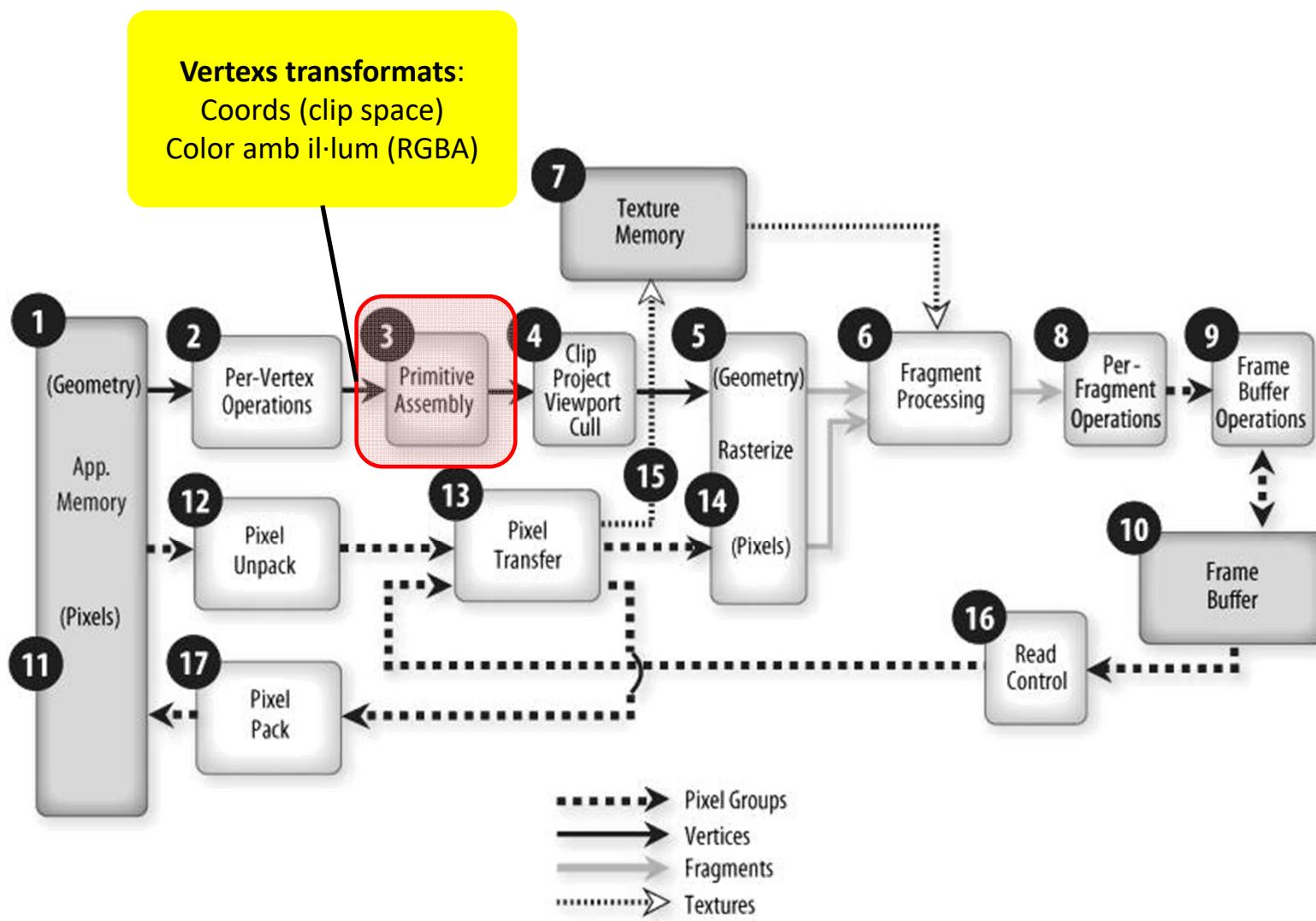
# Pipeline OpenGL



# Pipeline OpenGL

## 2. Per-vertex operations

- Es transformen els vèrtexs (modelview i projection).
- Es transformen les normals (trasposta de l'inversa de la submatriu 3x3 de la modelview) i es pot calcular la il·luminació del vèrtex.



# Pipeline OpenGL

## 3. Primitive assembly

- Els vèrtexs s'agrupen formant primitives.
- Cada primitiva requereix un clipping diferent.

# Pipeline OpenGL - glDrawArrays

```
glDrawArrays(mode, first, count);
```

*vertex*

	x	y	z
v0	1.0	0.0	0.0
v1			
v2			
v3			
v4			
v5			
v6			
v7			
...			

GL\_POINTS

GL\_LINES

GL\_TRIANGLES

GL\_TRIANGLE\_STRIP

# Pipeline OpenGL - glDrawArrays

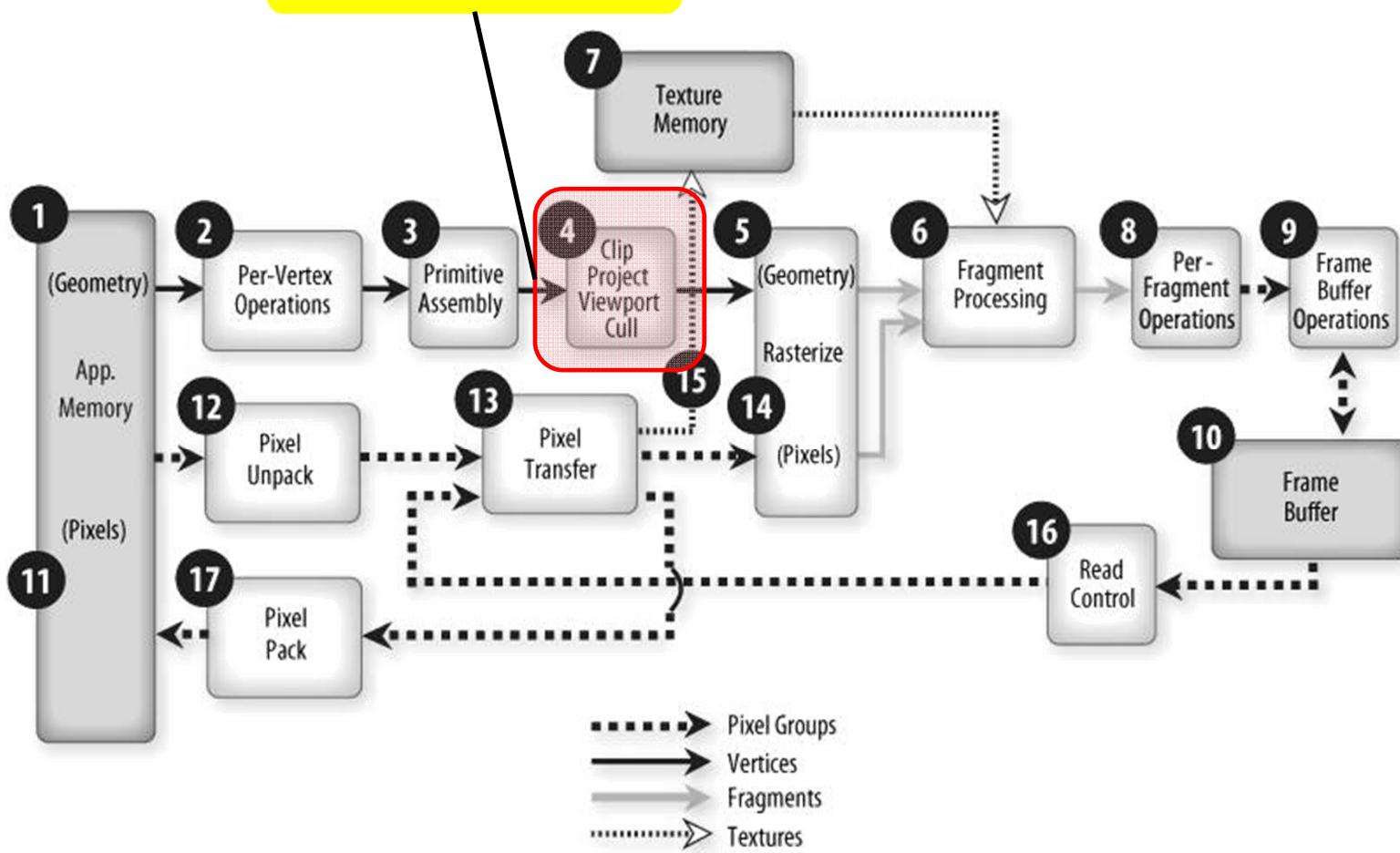
```
glDrawElements(mode, count, type, indices);
```

	<i>vertex</i>		<i>indices</i>
	x	y	z
v0	1.0	0.0	0.0
v1			
v2			
v3			
v4			
v5			
v6			
v7			
	...	...	...

GL\_TRIANGLES

GL\_TRIANGLE\_STRIP

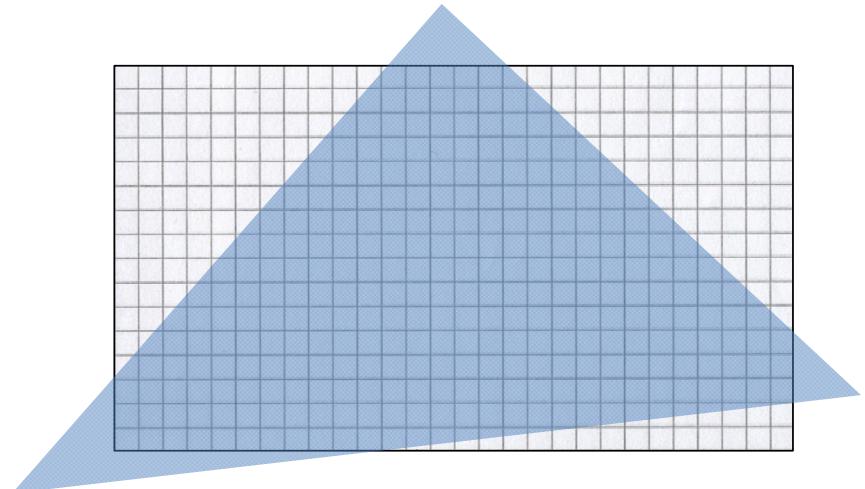
## Primitives transformades: (clip space) i il·luminades

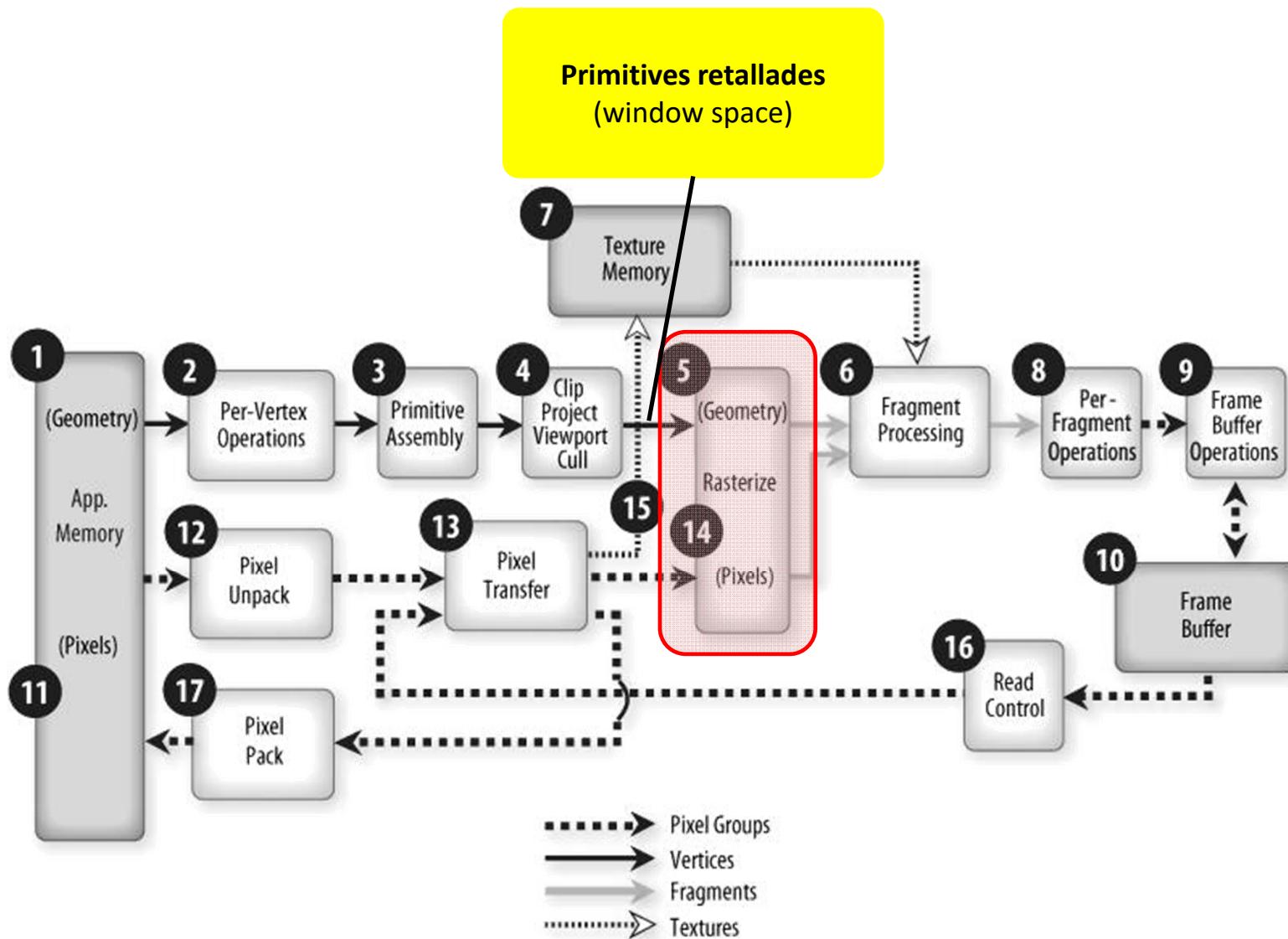


# Pipeline OpenGL

## 4. Primitive processing

- Clipping (retallat) a la piràmide de visió.
- Divisió de perspectiva: es divideix  $(x,y,z)$  per  $w$
- Viewport & depth transform → window space
- Backface culling – `glEnable(GL_CULL_FACE)`





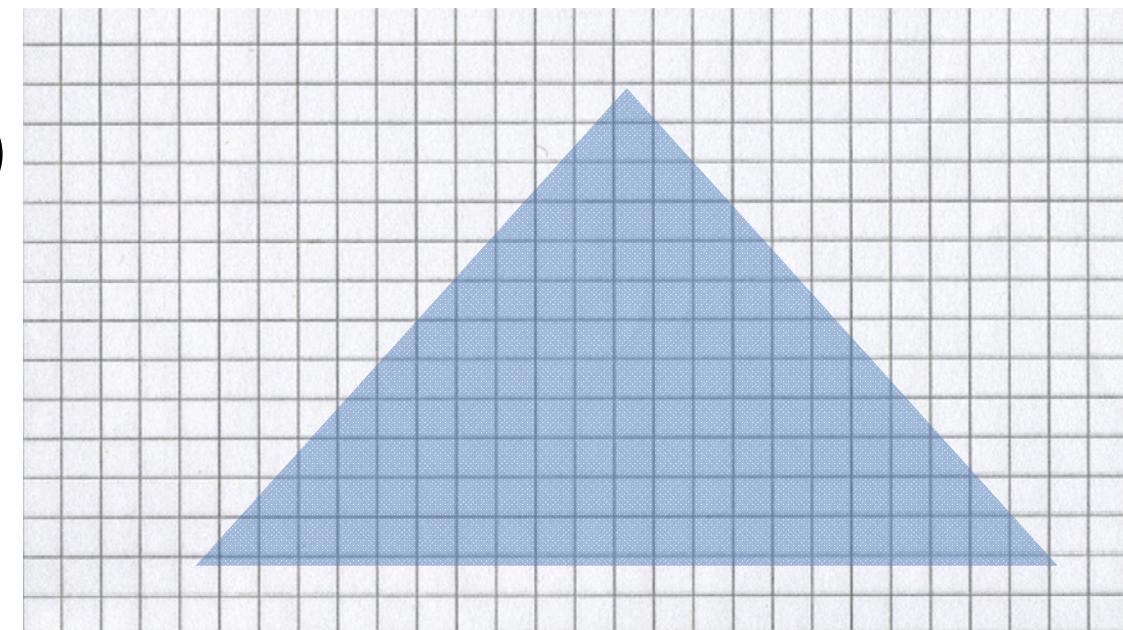
# Pipeline OpenGL

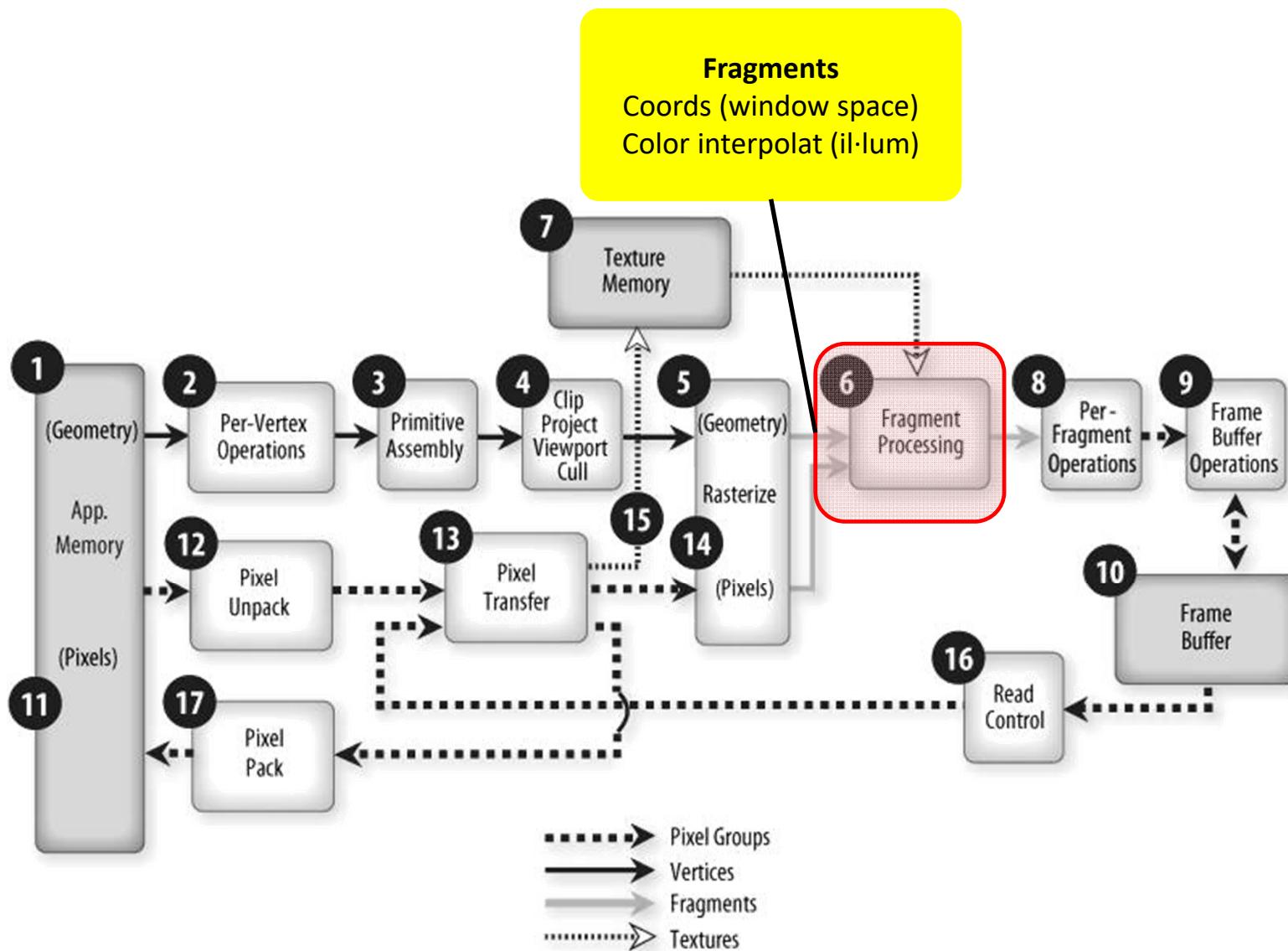
## 5. Rasterització

Generació dels fragments corresponents a la primitiva retallada.

Cada fragment té usualment diversos atributs:

- Coordenades (window space)
- Color (interpolat)
- Coordenades de textura (interpolades)

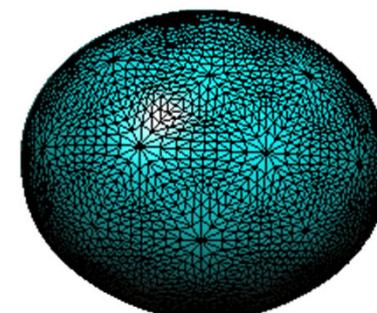
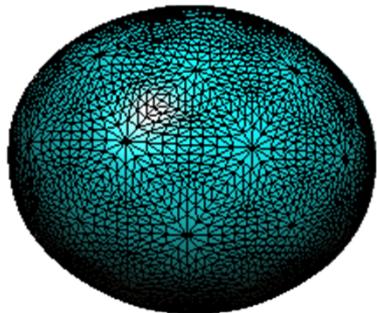


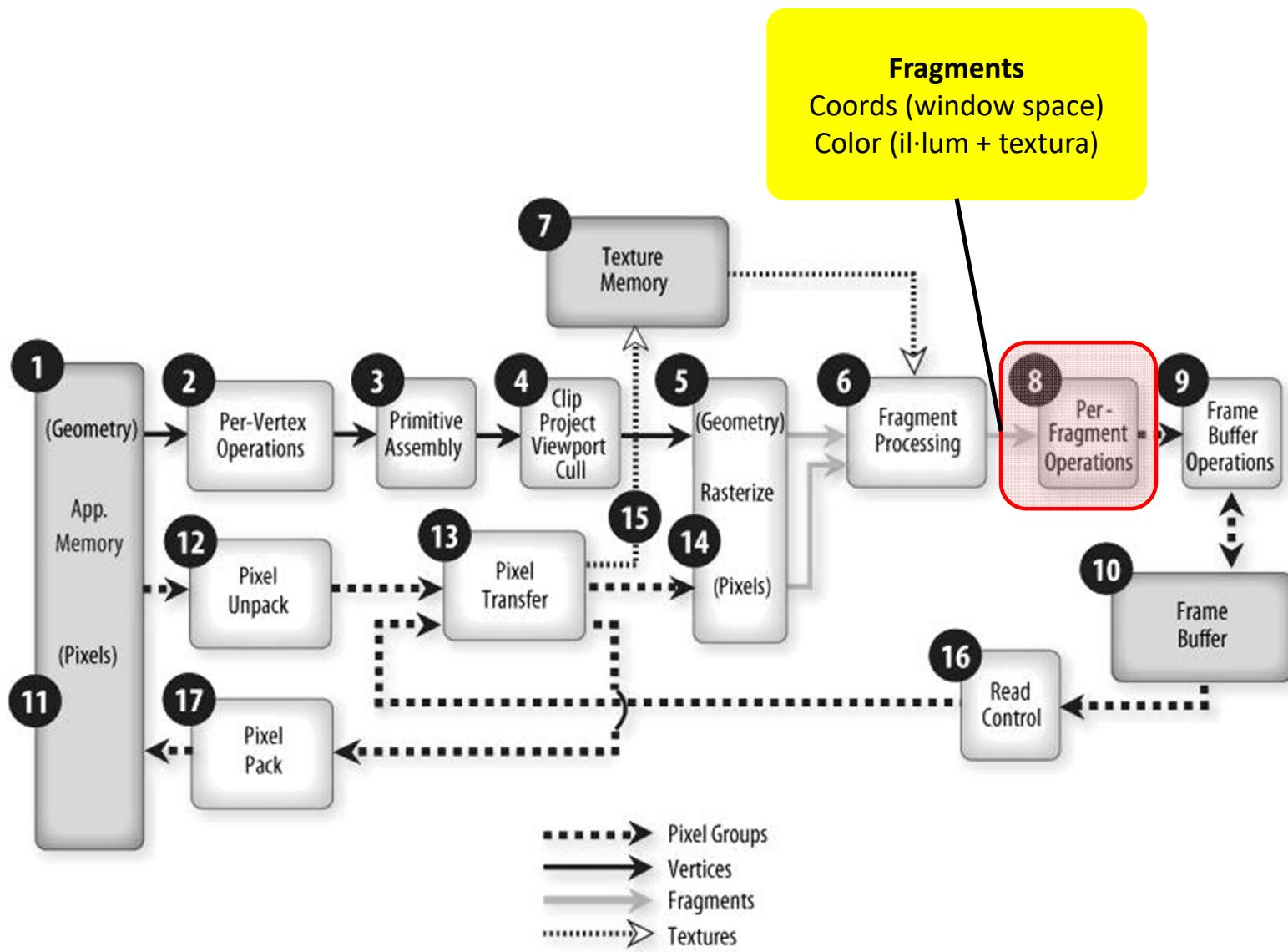


# Pipeline OpenGL

## 6. Fragment processing (“shading”)

- Càlcul del color del fragment (texture mapping, il·lum per fragment, etc).





# Pipeline OpenGL

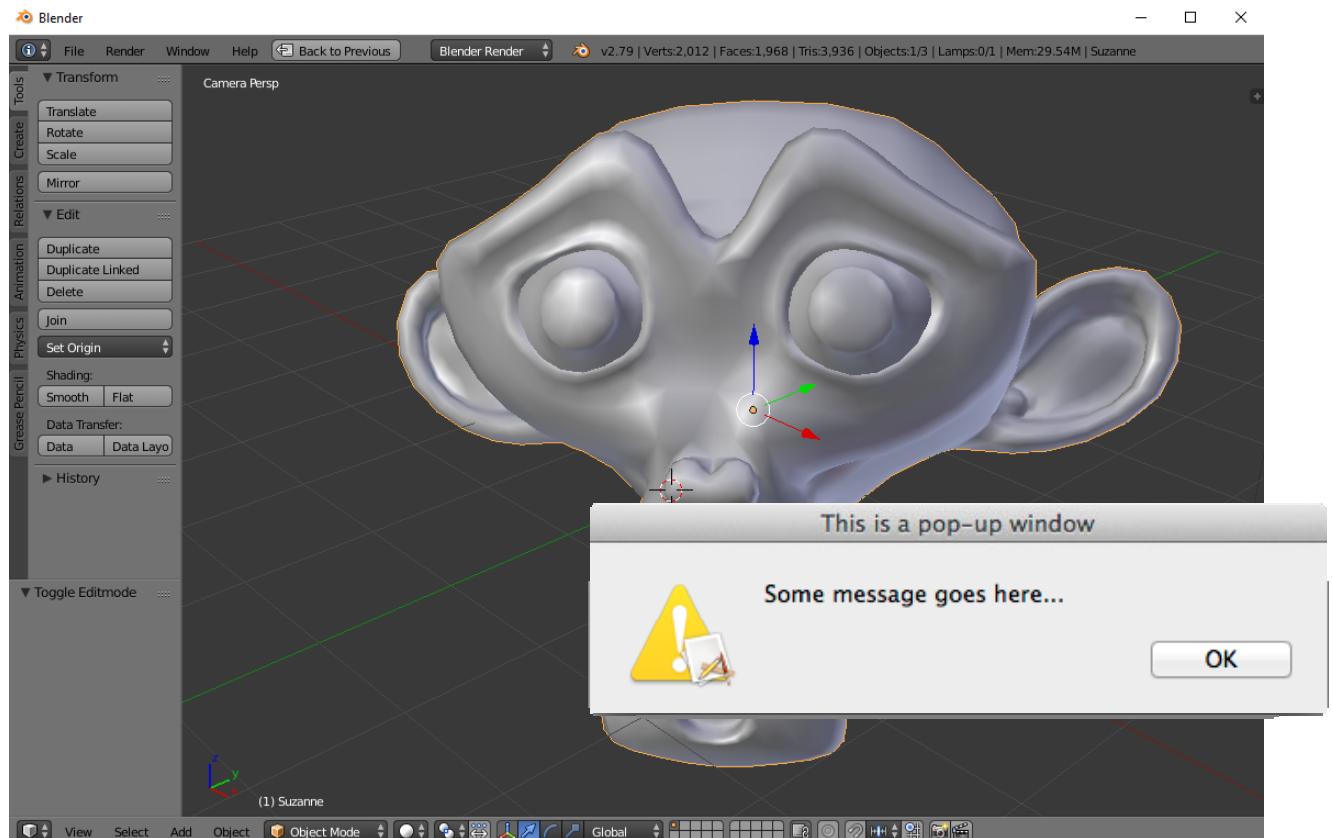
## 8. Per-fragment operations (“raster operations”)

- Pixel ownership test
- *Scissor test*
- *Alpha test*
- Stencil test
- Depth test (test Z-buffer)
- Blending
- *Dithering*
- *Logical Ops (glLogicOp)*

# Pipeline OpenGL

## 8. Per-fragment operations (“raster operations”)

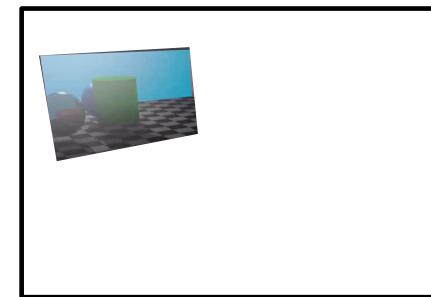
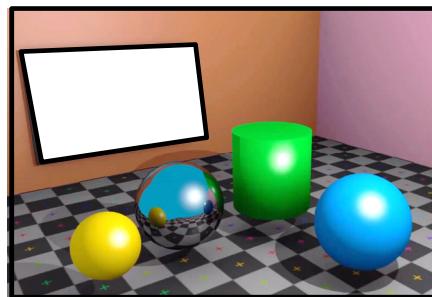
- **Pixel ownership test**
- *Scissor test*
- *Alpha test*
- Stencil test
- Depth test (test Z-buffer)
- Blending
- *Dithering*
- *Logical Ops (glLogicOp)*



# Pipeline OpenGL

## 8. Per-fragment operations (“raster operations”)

- Pixel ownership test
- *Scissor test*
- *Alpha test*
- **Stencil test**
- Depth test (test Z-buffer)
- Blending
- *Dithering*
- *Logical Ops (glLogicOp)*

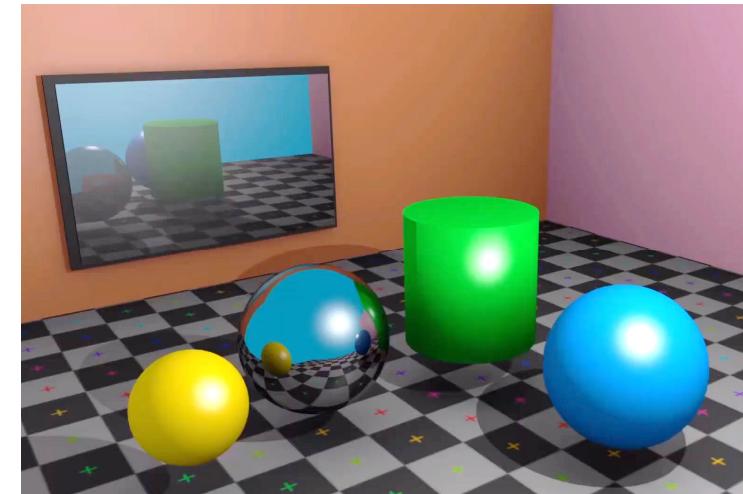


```
glEnable(GL_STENCIL_TEST);
```

```
glStencilFunc(comparació, valorRef, mask)  
    GL_ALWAYS, GL_EQUAL, ...
```

```
glStencilOp(fail, zfail, zpass)
```

```
GL_KEEP, GL_ZERO, GL_INCR, GL_REPLACE
```



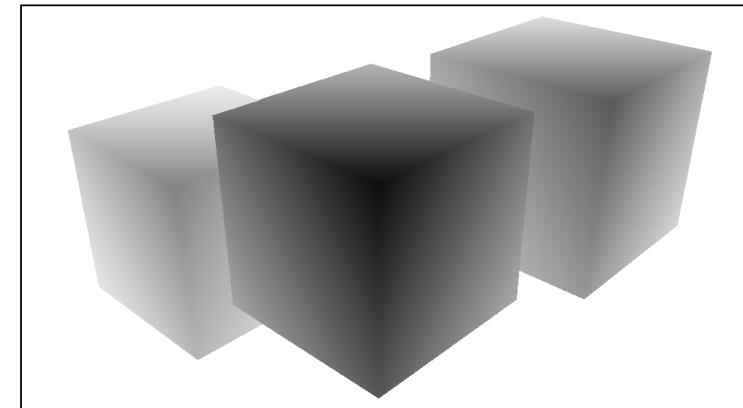
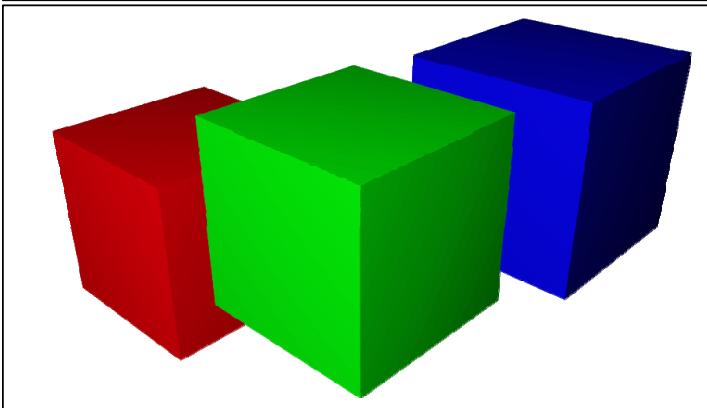
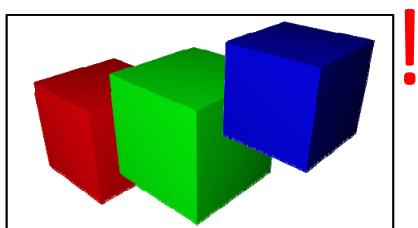
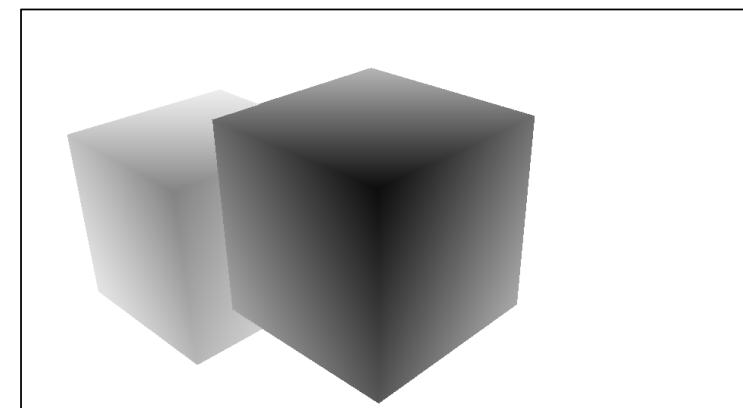
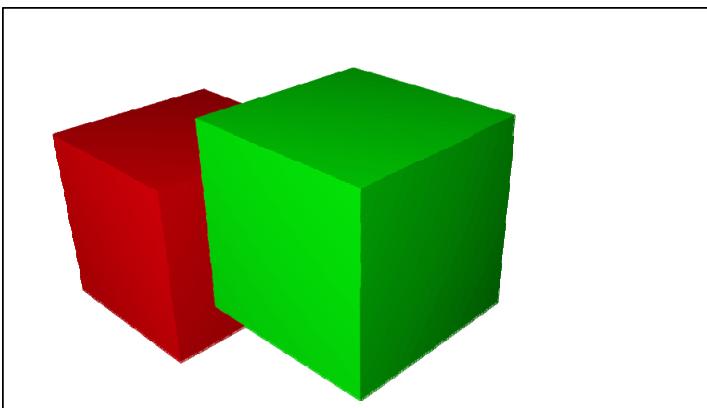
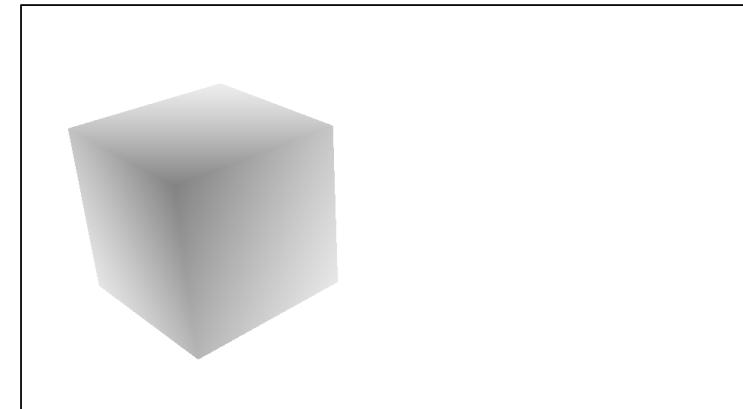
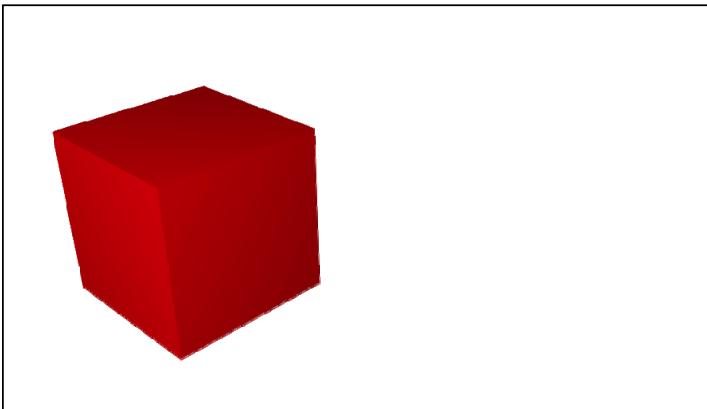
# Pipeline OpenGL

## 8. Per-fragment operations (“raster operations”)

- Pixel ownership test
- *Scissor test*
- *Alpha test*
- Stencil test
- **Depth test (test Z-buffer)**
- Blending
- *Dithering*
- *Logical Ops (glLogicOp)*

```
glEnable(GL_DEPTH_TEST);  
glDepthFunc(GL_LESS);  
    GL_LESS, GL_EQUAL, GL_GREATER, ...
```



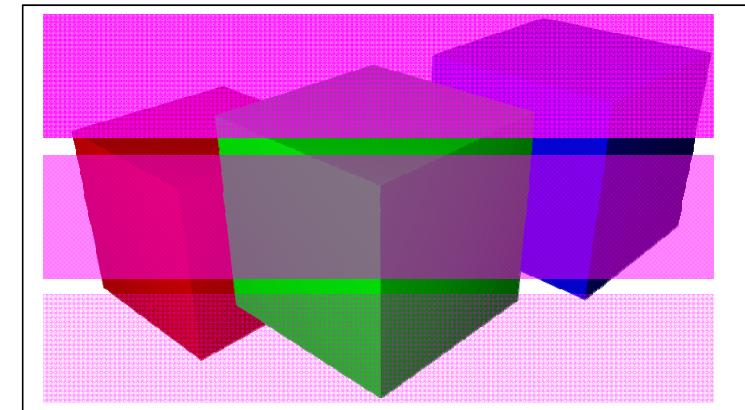
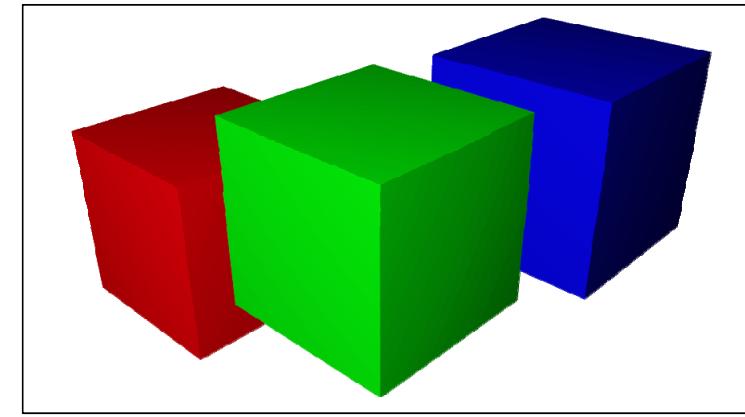


# Pipeline OpenGL

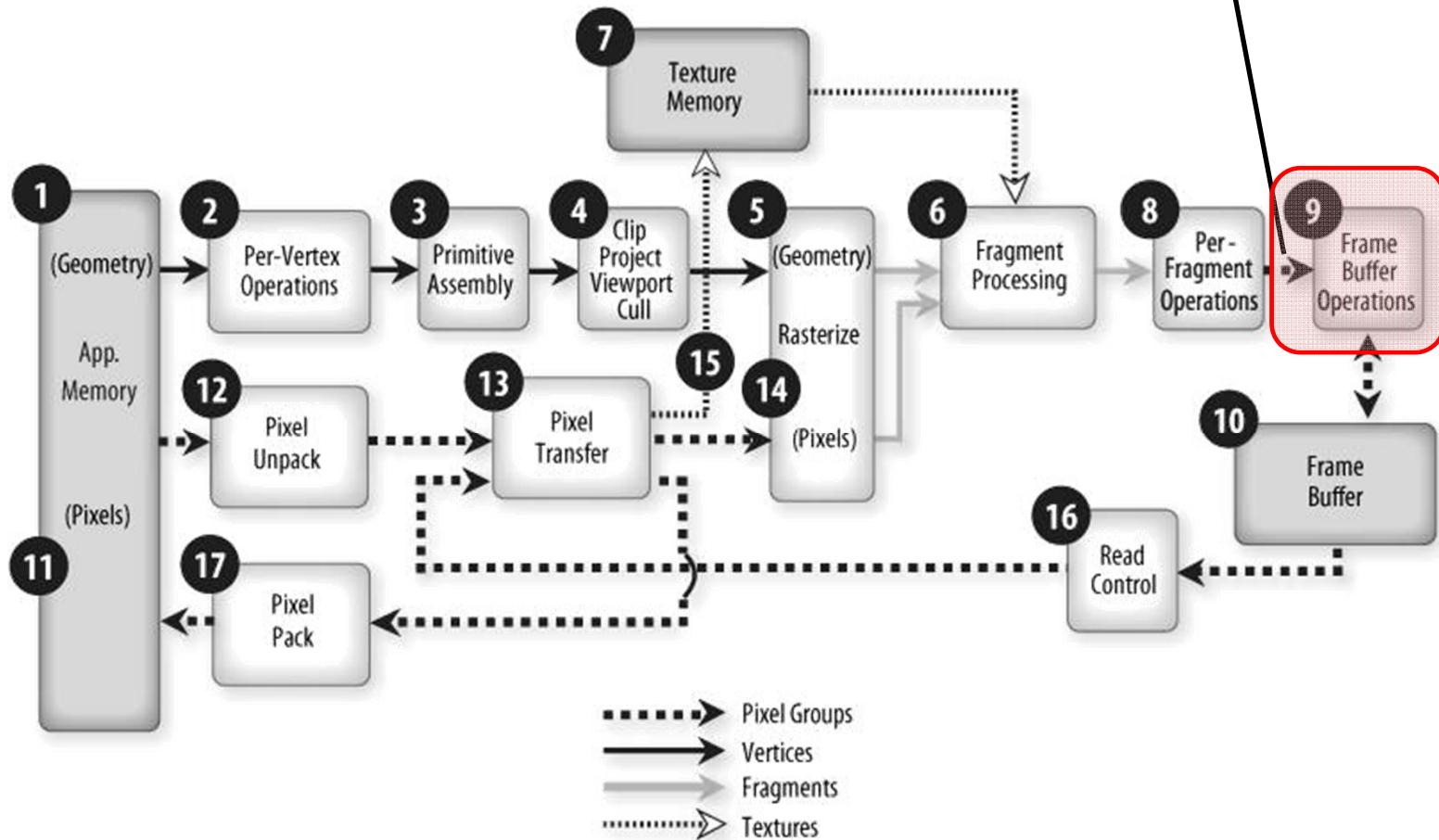
## 8. Per-fragment operations (“raster operations”)

- Pixel ownership test
- *Scissor test*
- *Alpha test*
- Stencil test
- Depth test (test Z-buffer)
- **Blending**
- *Dithering*
- *Logical Ops (glLogicOp)*

```
glEnable(GL_BLEND);  
glBlendFunc(...);
```



**Fragments visibles**  
 Coords (window space)  
 Color (il·lum + texture +  
 alpha blending)



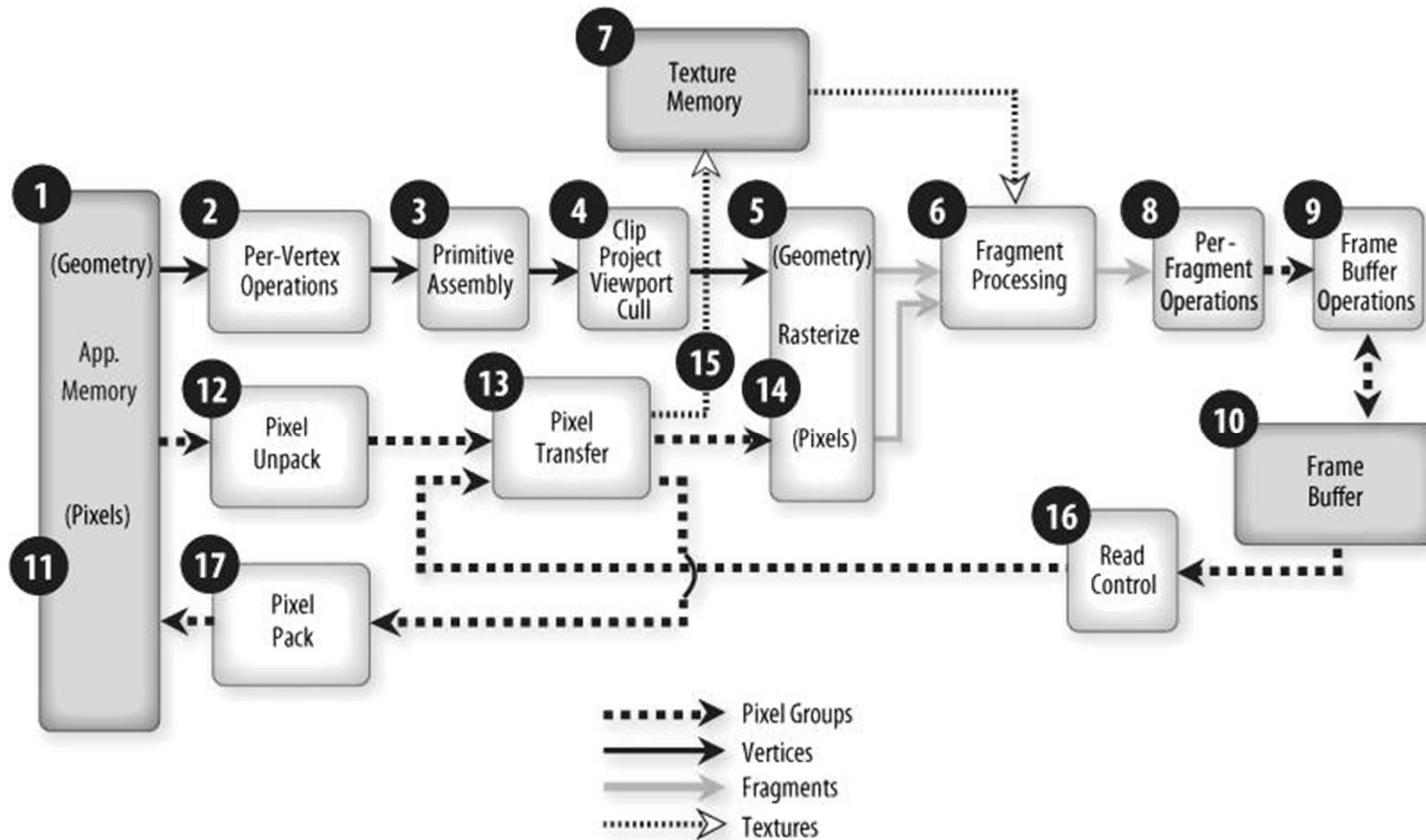
# Pipeline OpenGL

## 9. Frame buffer operations

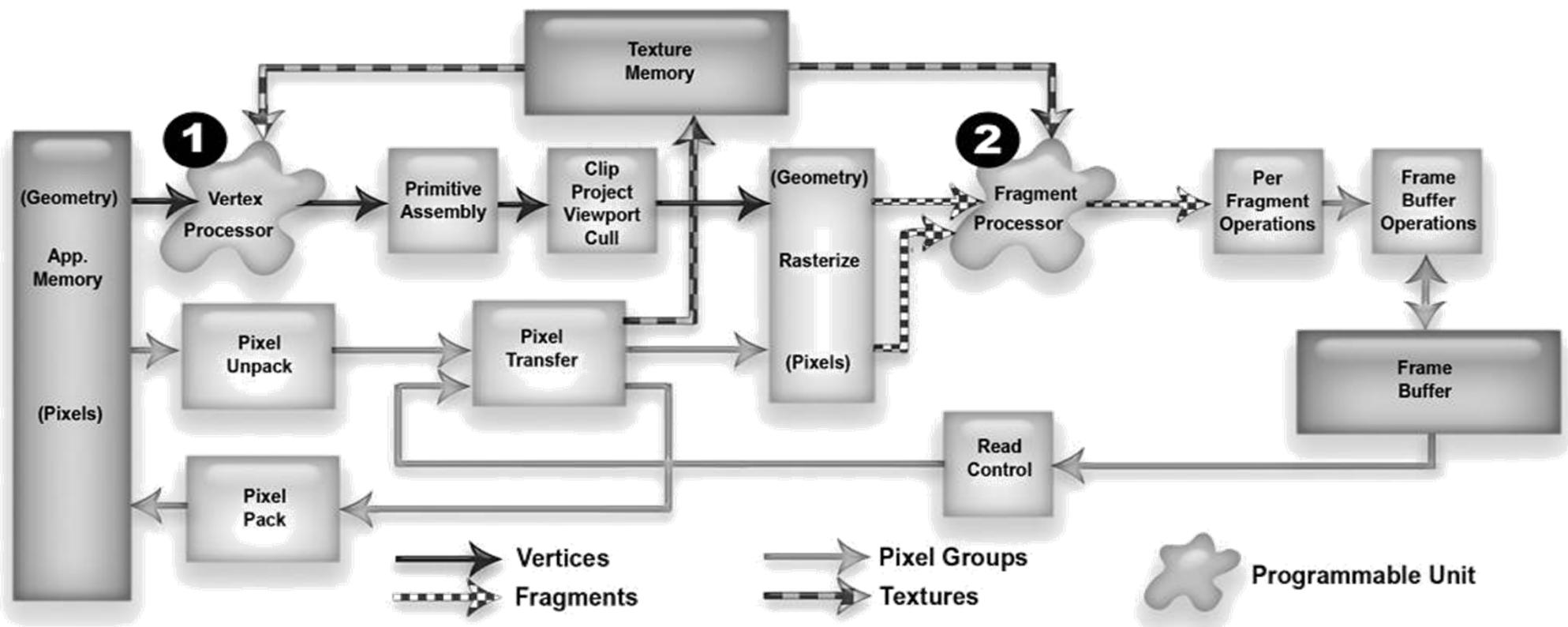
- Es modifiquen els buffers que s'hagin escollit amb glDrawBuffers
- Es veu afectada per glColorMask, glDepthMask...

**PIPELINE PROGRAMABLE**

# Pipeline fix...



# i pipeline programmable



# Pipeline programable

- **Vertex processor:** part de la GPU capaç d'executar un programa per cada vèrtex.
- **Fragment processor:** part de la GPU capaç d'executar un programa per cada fragment.
- **Shader:** codi font d'un programa (o part) per la GPU
  - Vertex shader, fragment shader, geometry shader
- **Program:** executable d'un programa per la GPU
  - Vertex program, fragment program, geometry program

# Pipeline programable

- Quan s'activa un **vertex program**, en lloc d'executar-se les operacions **per-vèrtex** prefixades d'OpenGL, s'executa el vertex program.
- Quan s'activa un **fragment program**, en lloc d'executar-se les operacions **per-fragment** prefixades d'OpenGL, s'executa el fragment program.
- Normalment el vertex/fragment program més senzill haurà de reproduir part de la funcionalitat fixa d'OpenGL.