

# **Sessió Setmana 10**

**GiVD 2022-23**

# Guió de la sessió

---

1. Planificació
  2. Recap activitat ZBuffer
  3. Activitat CPU-GPU
  4. Shadings poligonals
- 



# 1. Planificació projectes

- Veure el campus: Setmana 11



2023 May						
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
01	02	03	04	05	06	07
	Pràctica 2			Publicació dels projectes de l'examen final		
08	09	10	11	12	13	14
Tema 3	Pràctica 2			Assignació definitiva de projectes		
15	16	17	18	19	20	21
Tema 3	Pràctica 2					
22	23	24	25	26	27	28
Tema 4	Entrevista/Prova Practica 2					Problemes 3

# 1. Planificació dels projectes

- Veure el campus: Setmana 11



Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
29	30	31	01	02 <b>Examen Final: 15:00-20:00</b>	03	04
05	06	07	08	09 <b>Examen Final: 15:00-20:00</b>	10	11
12	13	14	15	16	17	18
19	20	21	22	23	<b>AVALUACIÓ CONTÍNUA</b>	
26	27 <b>Reavaluació 15:00-20:00</b>	28	29	30	<b>L'avaluació es separa en dos blocs</b>	

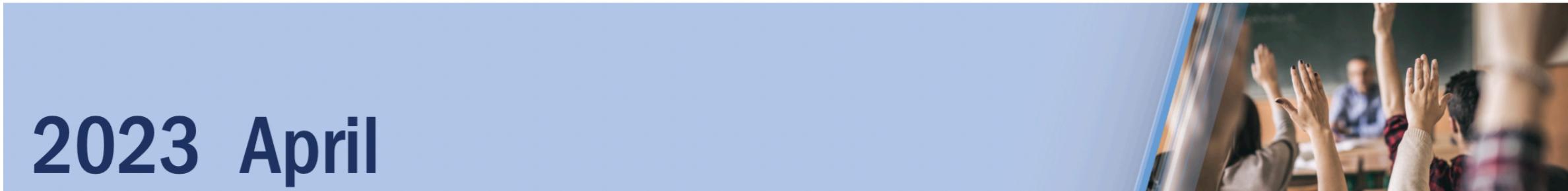
# 1. Planificació Pràctica 2

- Veure el campus: Pràctica 2

<b>Abril 2023</b>	17	18	19	20	21	Fase 0 pràctica 2: comprovació de la instal·lació i primers projectes a la GPU
	24	25	26	27	28	Fase 1 pràctica 2: Passos 1 i 2: Materials i Llums a la GPU
	1	2	3	4	5	Fase 1 pràctica 2: Pas 3: Shadings a la GPU
	8	9	10	11	12	Fase 1 pràctica 2: Pas 4: Textures. Fase 2: Exercicis
	15	16	17	18	19	Fase 2 pràctica 2 Exercicis, i opcionals: Animacions, mapeig esfèric de gizmos i/o shadings avançats.
	22	23	24	25	26	<b>Entrega P2. Entrevista i  prova</b>

# 1. Planificació de la setmana

- Veure el campus: Setmana 9



Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
24 Tema 3: Shading poligonal i exercicis de shaders	25 L10: Enunciat de la pràctica 2. Fase 1 pràctica 2: Passos 1, 2 i 3	26	27	28	29	30
						Set. 10



Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
01	02 L11: Passos 3 Shadings a la GPU	03	04	05 Publicació dels projectes de l'examen final	06	07
						Set. 11

# Índex

3.1. Introducció a ZBuffer

3.2. Pipeline de visualització

**3.3. Pipeline de visualització a GL**

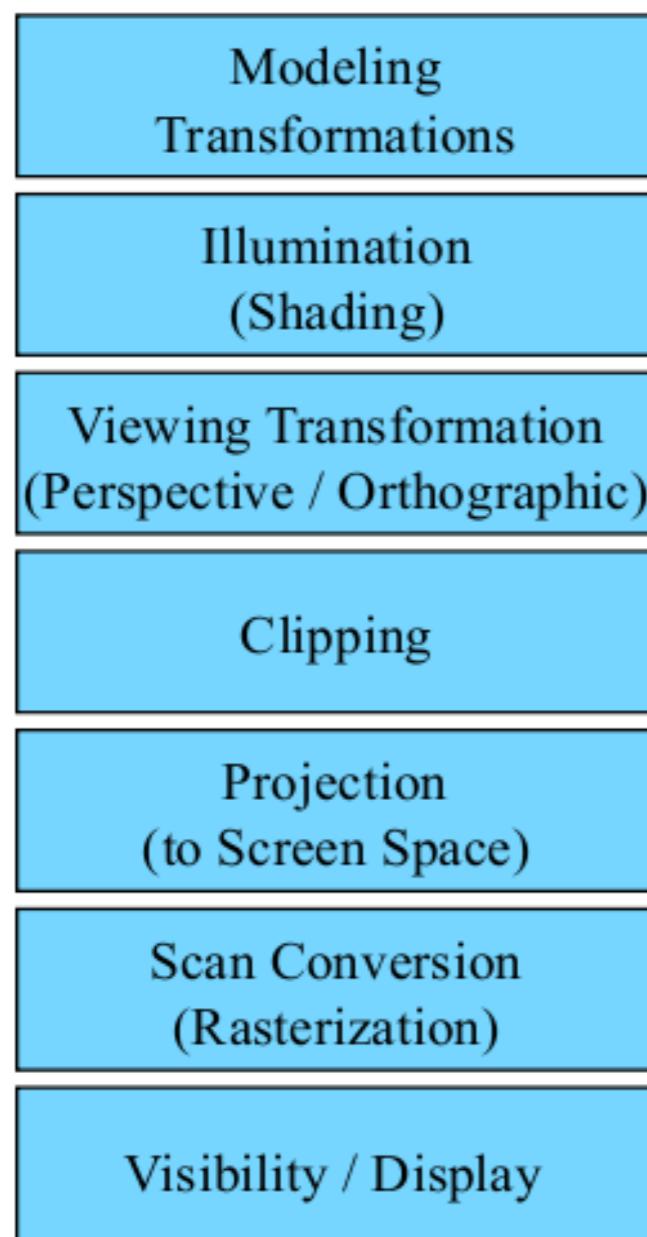
**3.4. Il·luminació usant shaders**

3.5. Textures

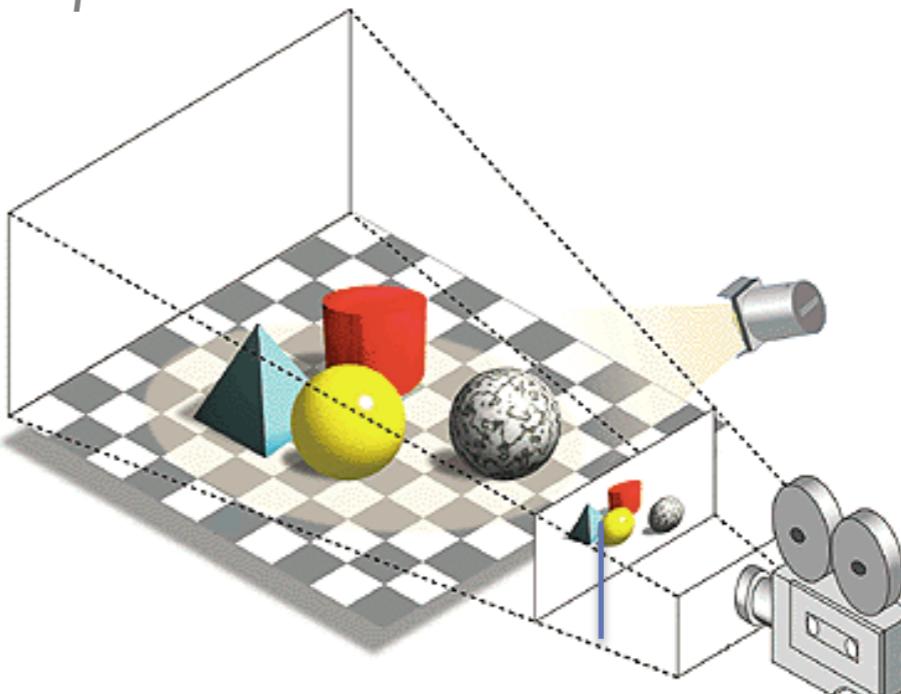
3.6. Reflexions i Transparències

# 2. Recap ZBuffer-Pipeline

**Pipeline de visualització:** conjunt d'etapes\* que a partir de l'escena 3D, llums, càmera i viewport, permet obtenir la imatge 2D amb els colors finals.



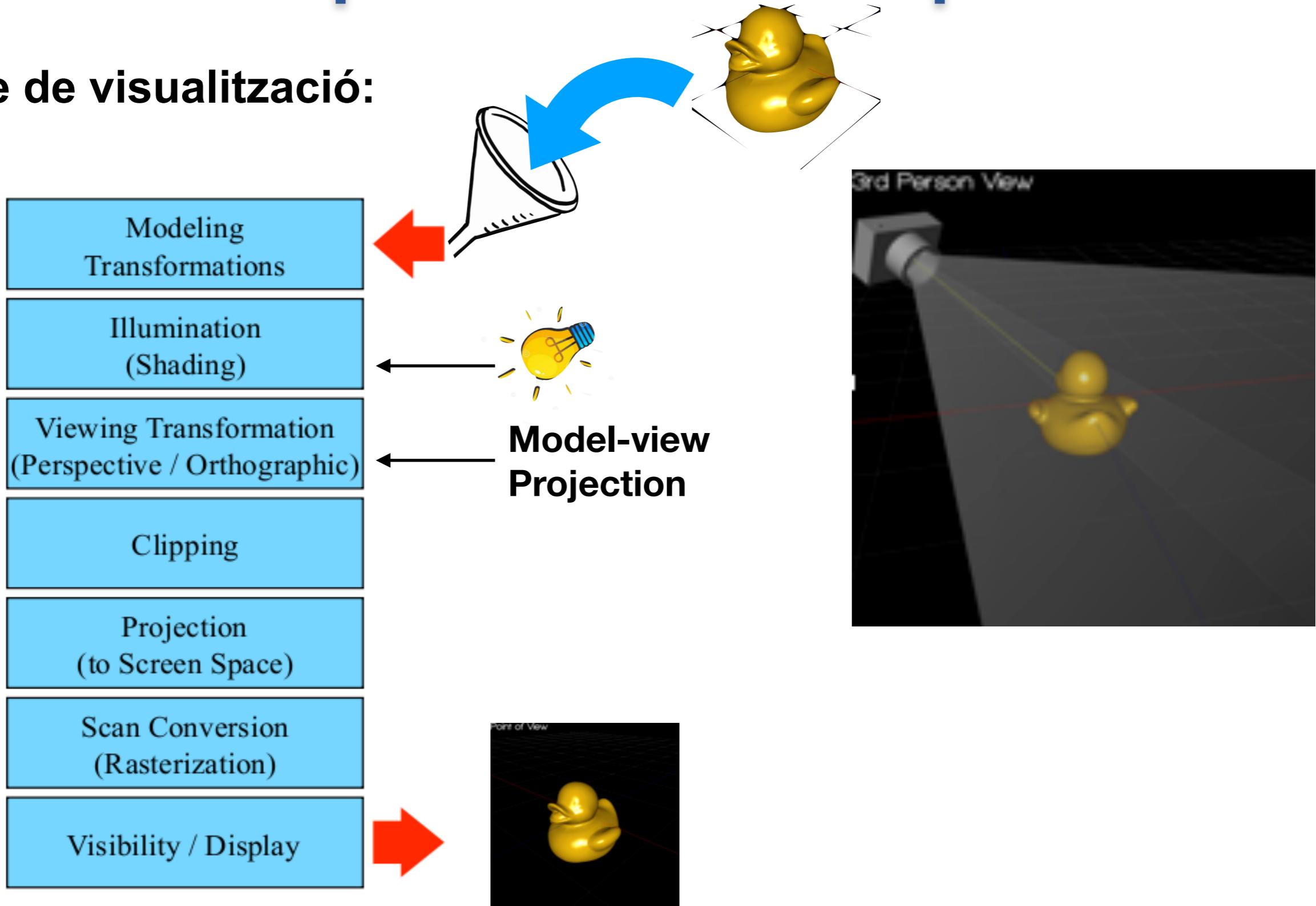
- **Input:** Objectes, Llum, Càmera i viewport



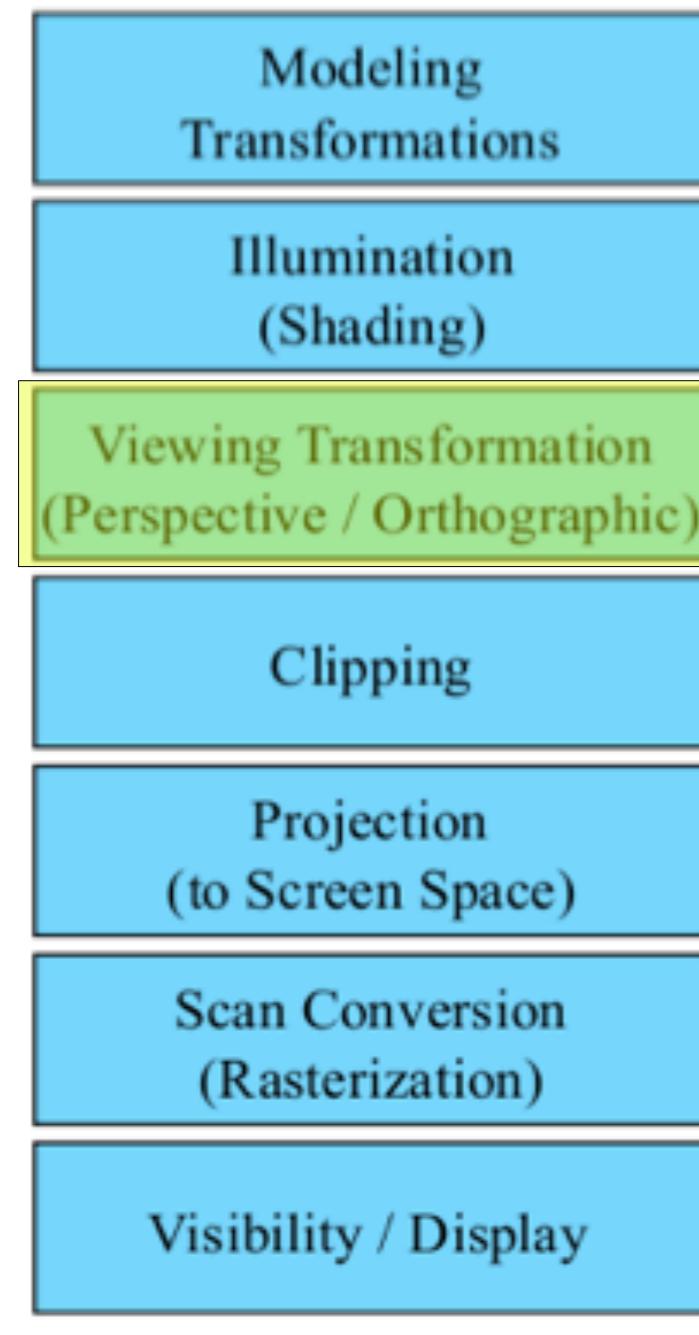
- **Output:** Frame buffer  
(Colors/Intensitats (ex. 24-bit RGBA a cada píxel))

# 2. Recap ZBuffer-Pipeline

Pipeline de visualització:

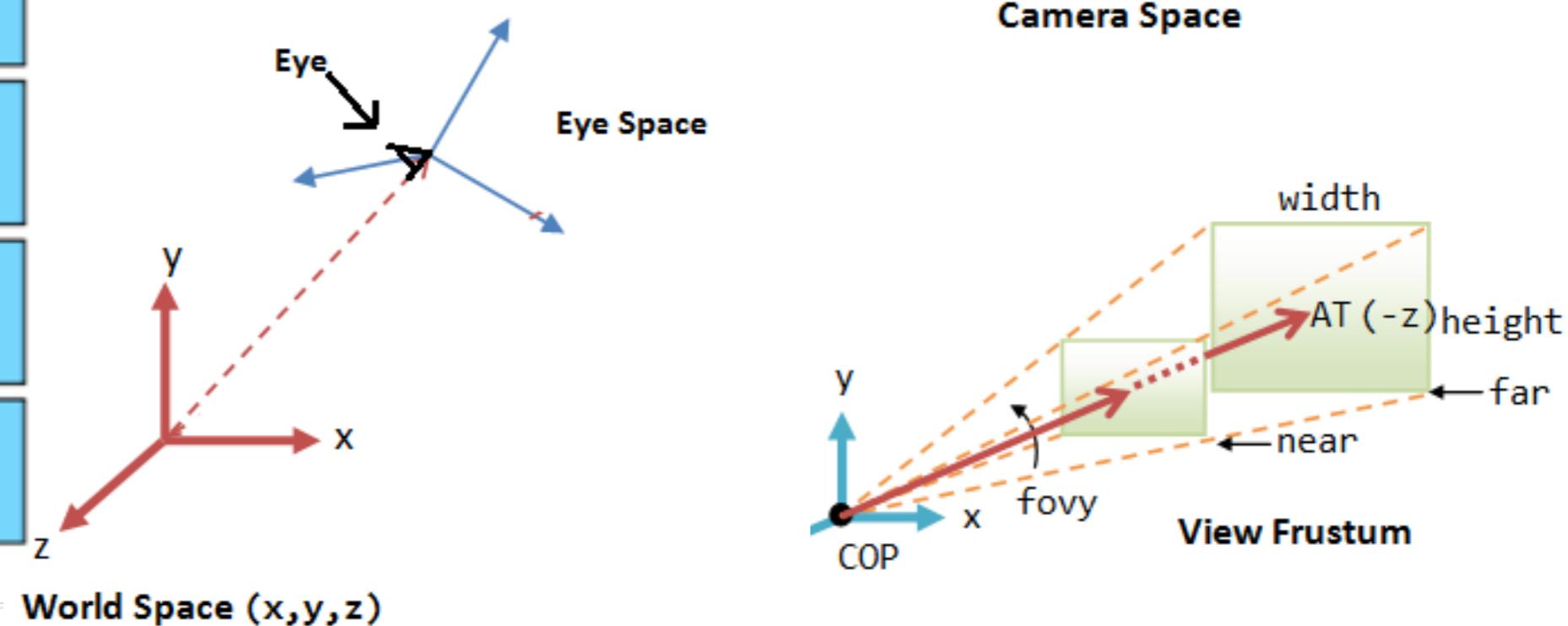


# 2. Recap ZBuffer-Pipeline



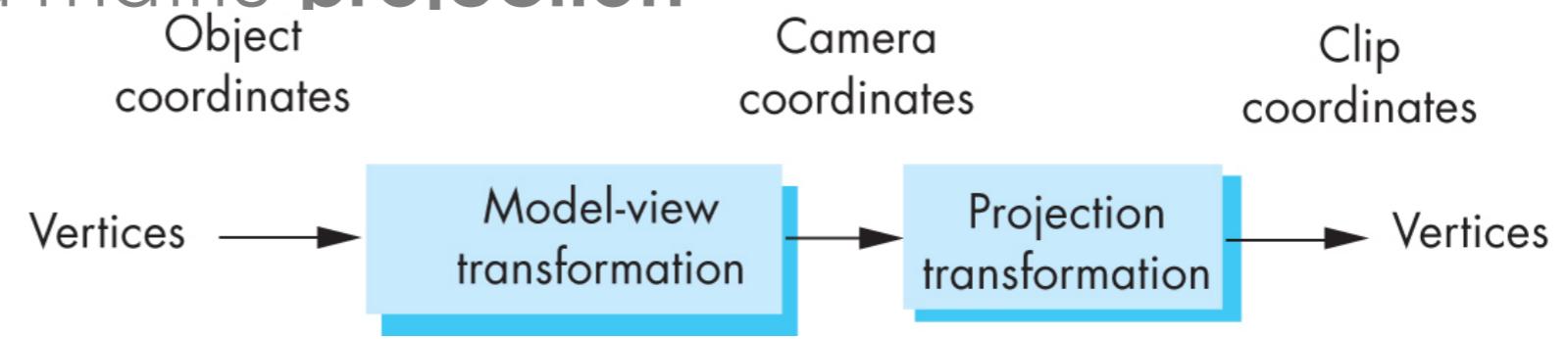
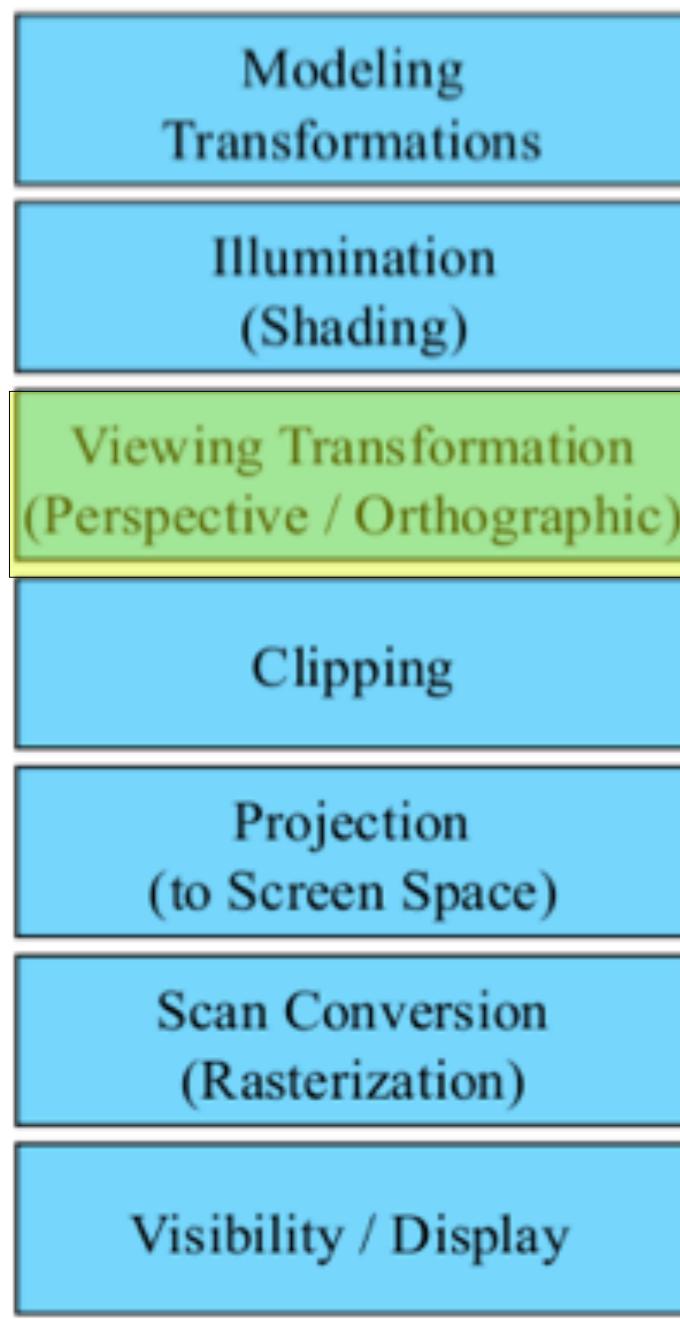
- **Transformacions de càmera:**

- Es fan les transformacions per a “girar” l’escena segons la posició de la càmera (matriu **model-view**)
- Es projecten els vèrtexs 3D segons el tipus de projecció en el pla de projecció (matriu **projection**)
- S’obtenen punts 2D en l’espai continu

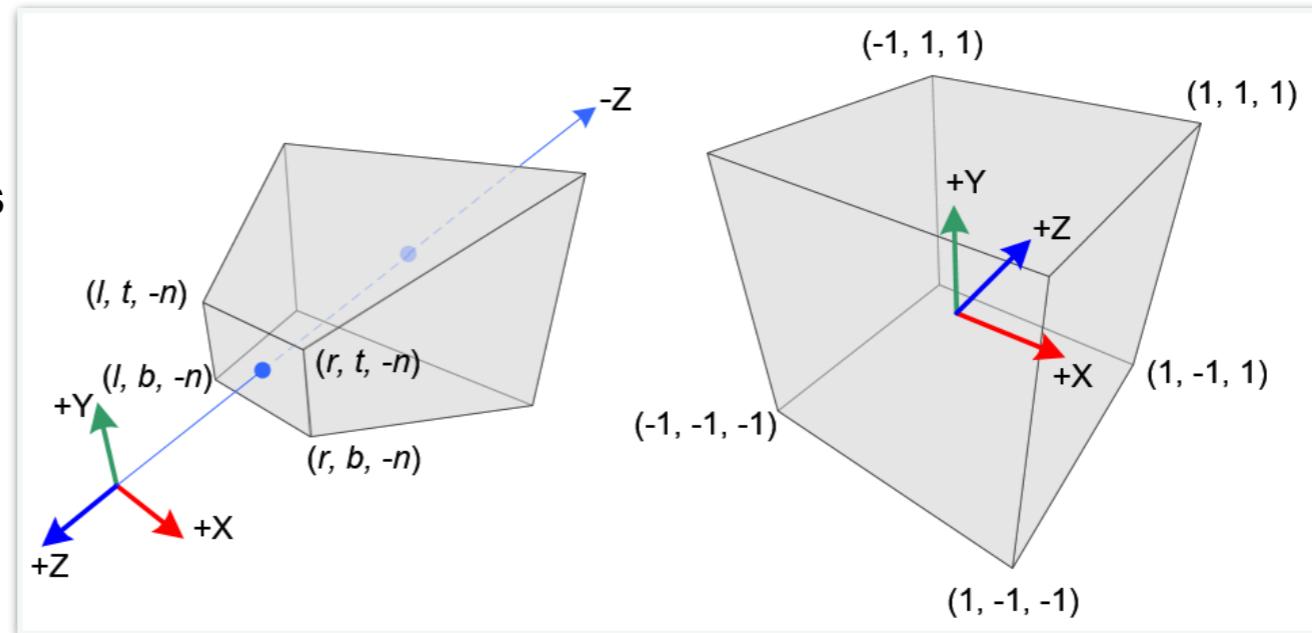


# 2. Recap ZBuffer-Pipeline

La **normalització** és una transformació que permet convertir el volum de visió en un cub centrat a l'origen i d'aresta 2. Es codifica dins de la matriu **projection**



**Convencions de GL:**



Coordenades de càmera en el sistema de la mà dreta

Coordenades normalitzades en el sistema de la mà esquerra

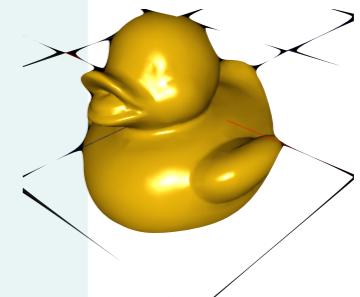
# 2. Recap ZBuffer-Pipeline



L'algorisme de zbuffer només es planteja la visualització de malles poligonals:

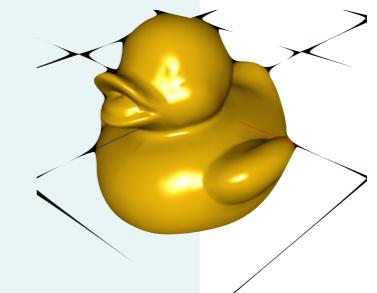
Trieu-ne una:

- a. CERT
- b. FALS



# 2. Recap ZBuffer-Pipeline

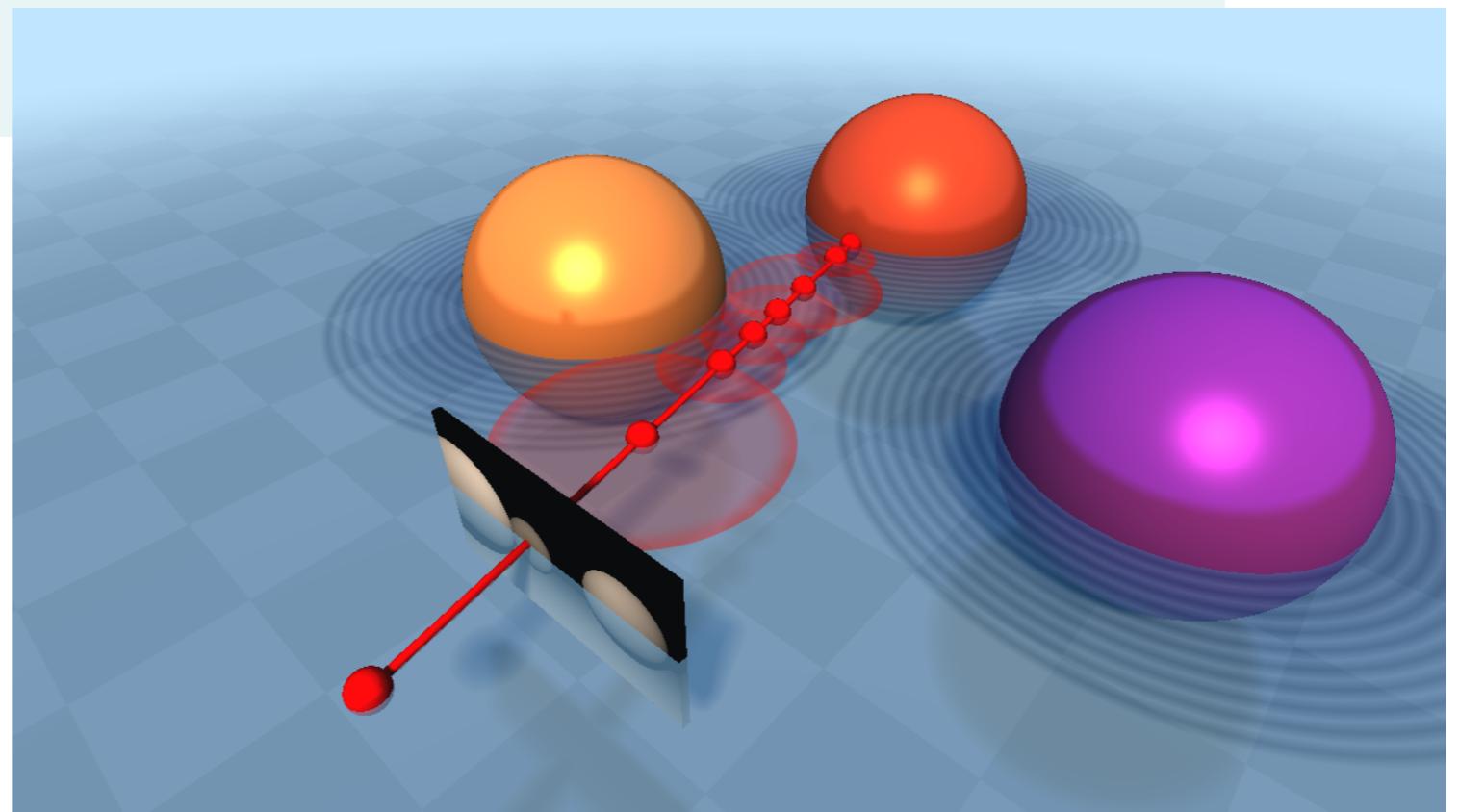
L'algorisme de zbuffer només es planteja la visualització de malles poligonals:



Trieu-ne una:

- a. CERT
- b. FALS

[Esborra la meva selecció](#)



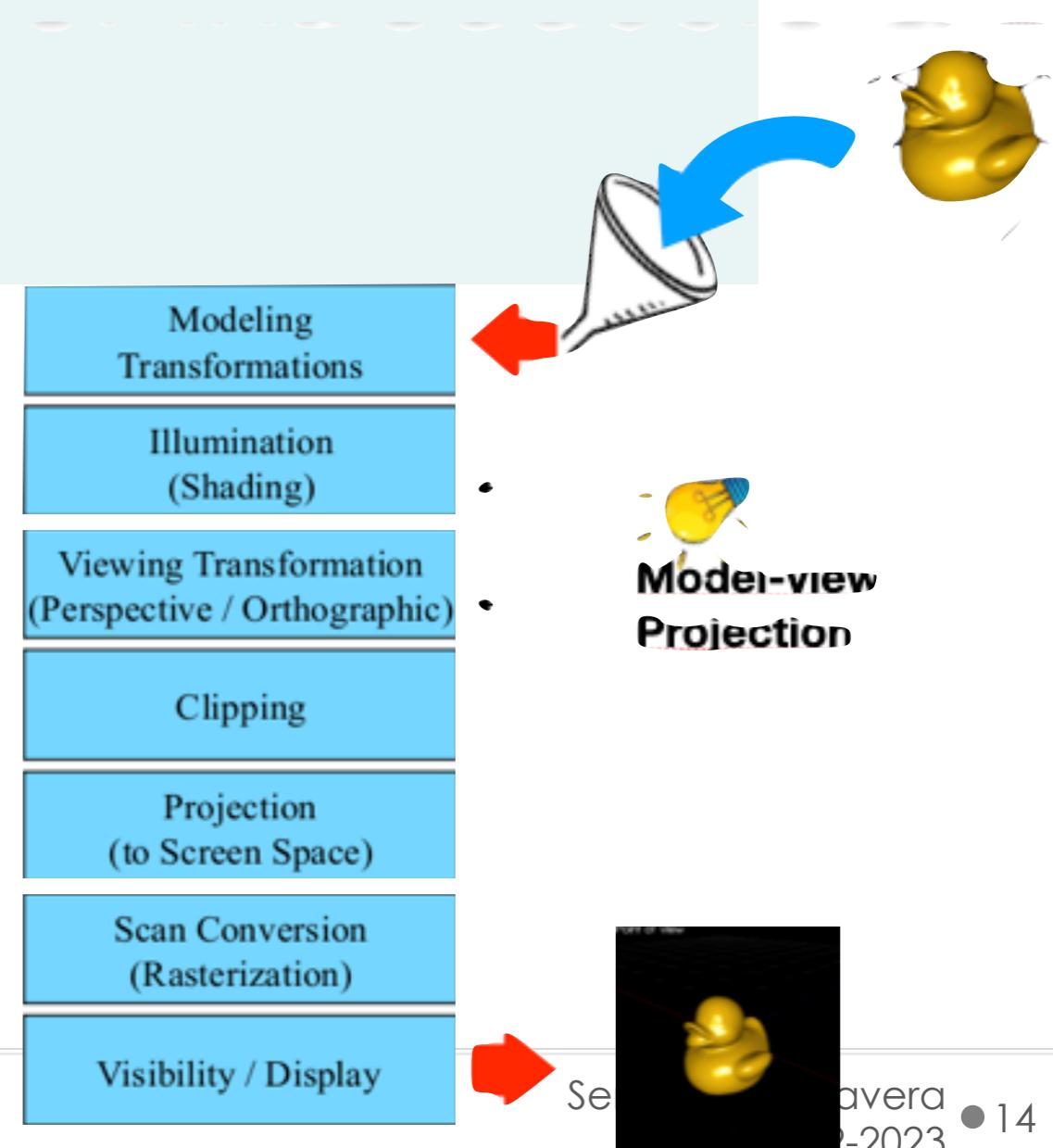
# 2. Recap ZBuffer-Pipeline

## Raytracing vs ZBuffer

El pipeline de visualització consisteix en el conjunt de passos a realitzar per visualitzar una escena, ja sigui en ZBuffer com en Raytracing.

Trieu-ne una:

- a. FALS
- b. CERT



# 2. Recap ZBuffer-Pipeline

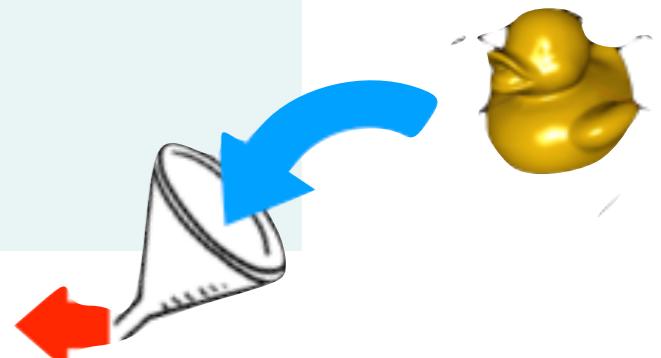
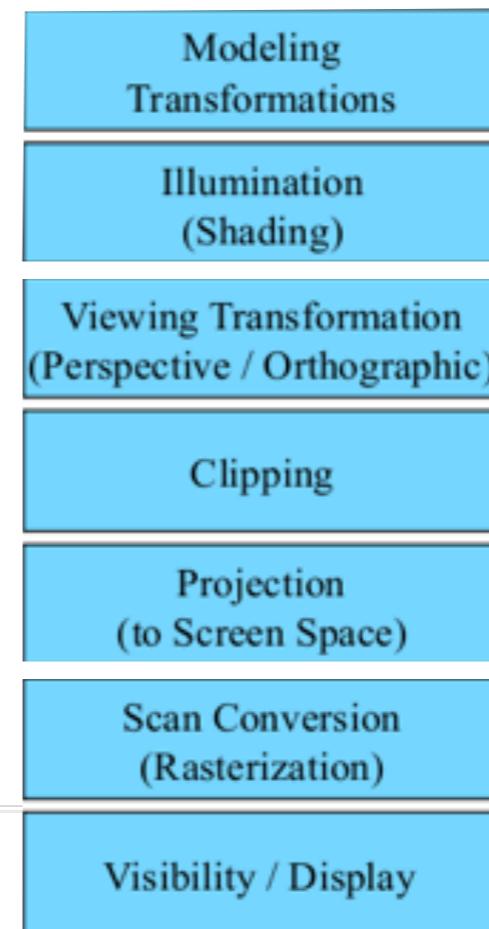


El pipeline de visualització consisteix en el conjunt de passos a realitzar per visualitzar una escena, ja sigui en ZBuffer com en Raytracing.

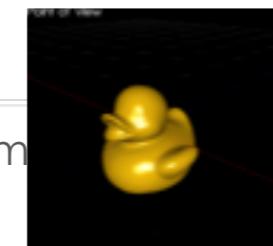
Trieu-ne una:

- a. FALS
- b. CERT

[Esborra la meva selecció](#)



Model-view  
Projection

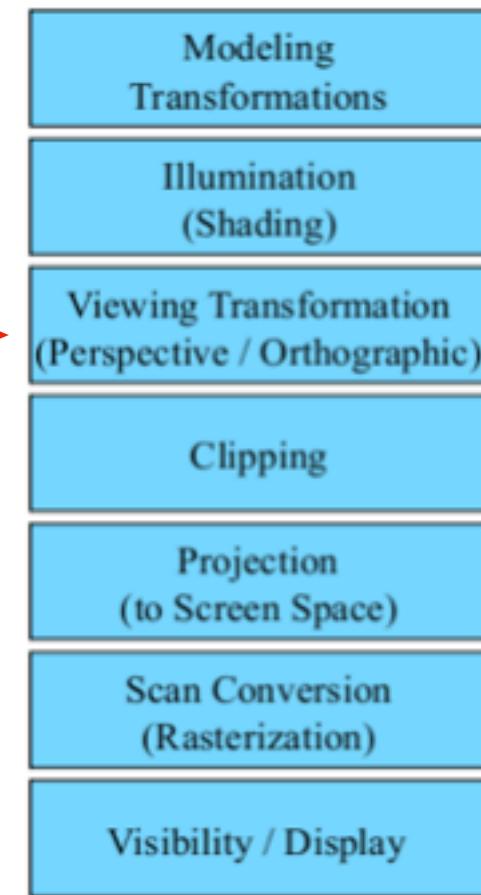


# 2. Recap ZBuffer-Pipeline

Les transformacions que s'han de fer sobre els objectes de l'escena virtual per a aconseguir la imatge vista des d'una camera són les mateixes en els algorismes de Raytracing i de ZBuffer

Trieu-ne una:

- a. CERT
- b. FALS



L'algorisme de ZBuffer implica la transformacions dels objectes de l'escena virtual de coordenades de món a coordenades de càmera:

Trieu-ne una:

- a. CERT
- b. FALS

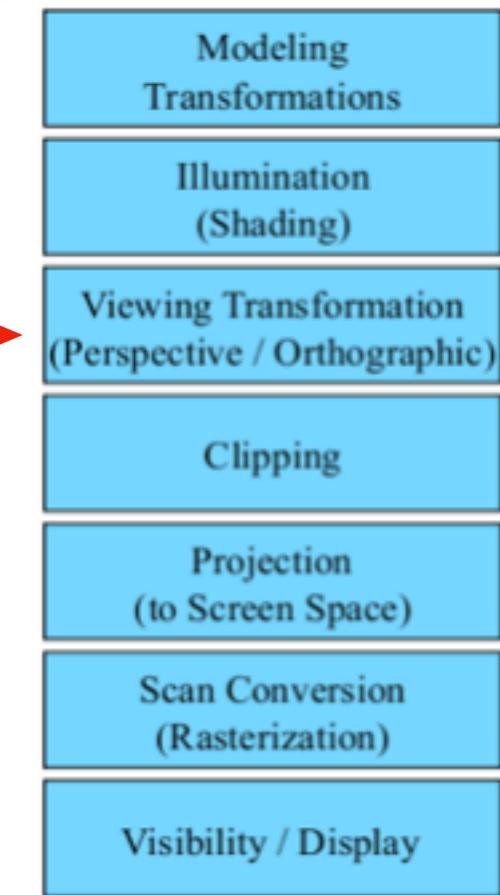
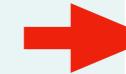
# 2. Recap ZBuffer-Pipeline

Les transformacions que s'han de fer sobre els objectes de l'escena virtual per a aconseguir la imatge vista des d'una camera són les mateixes en els algorismes de Raytracing i de ZBuffer

Trieu-ne una:

- a. CERT
- b. FALS

Esborra la meva selecció

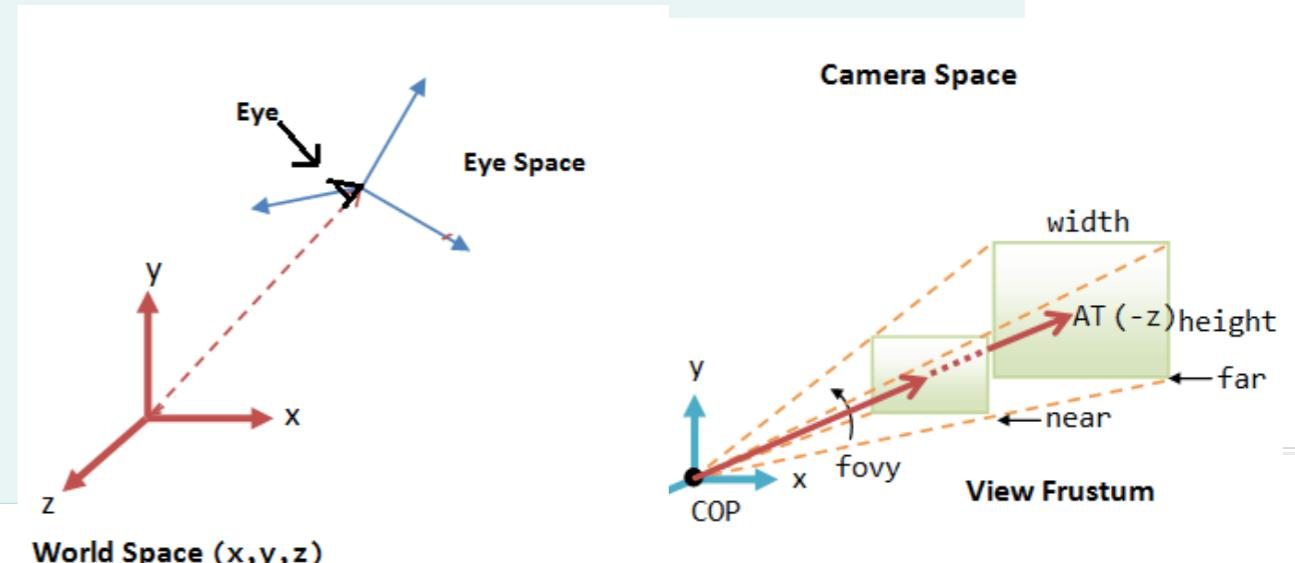


L'algorisme de ZBuffer implica la transformacions dels objectes de l'escena virtual de coordenades de món a coordenades de càmera:

Trieu-ne una:

- a. CERT
- b. FALS

Esborra la meva selecció



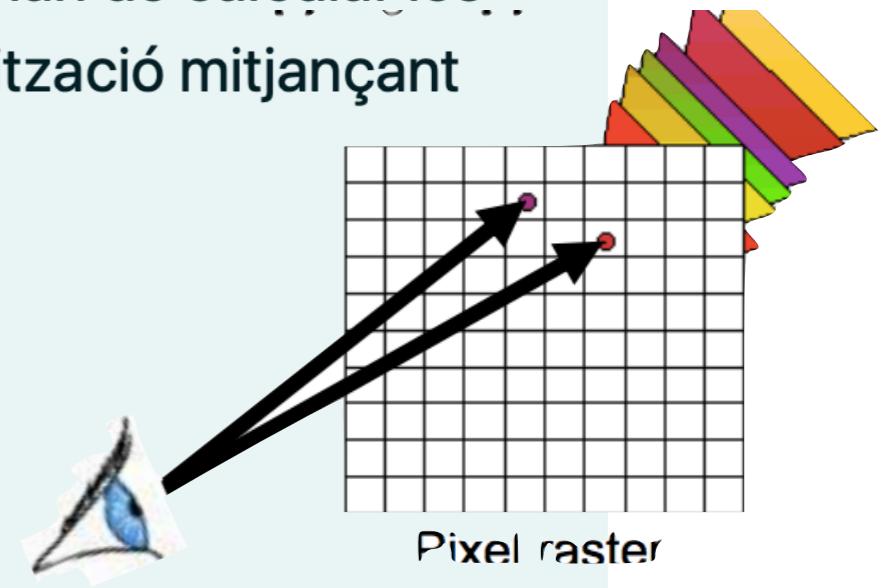
# 2. Recap ZBuffer-Pipeline



El procés de rasterització en el RayTracing implica saber la profunditat de cada píxel i per tant, si s'està visualitzant un triangle, s'han de calcular les profunditats a les que estan els píxels de la seva rasterització mitjançant l'ús de coordenades baricèntriques:

Trieu-ne una:

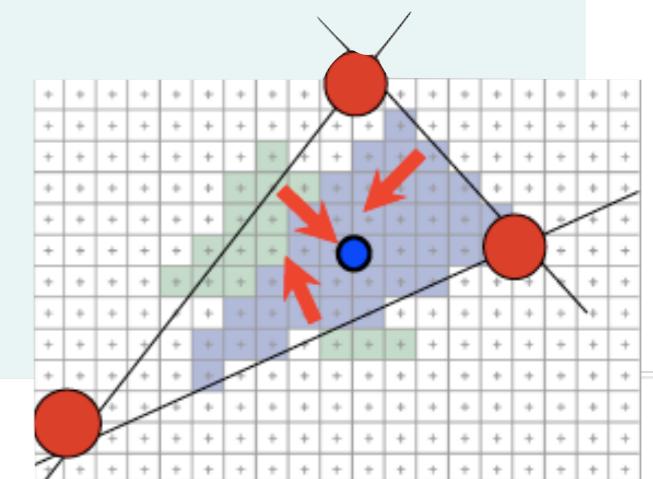
- a. CERT
- b. FALS



En rasteritzar un triangle, cal calcular la profunditat associada als píxels interiors del triangle a partir de la profunditat dels vèrtexs del triangle i les coordenades baricèntriques associades a cadascun dels píxels interiors.

Trieu-ne una:

- a. CERT
- b. FALS



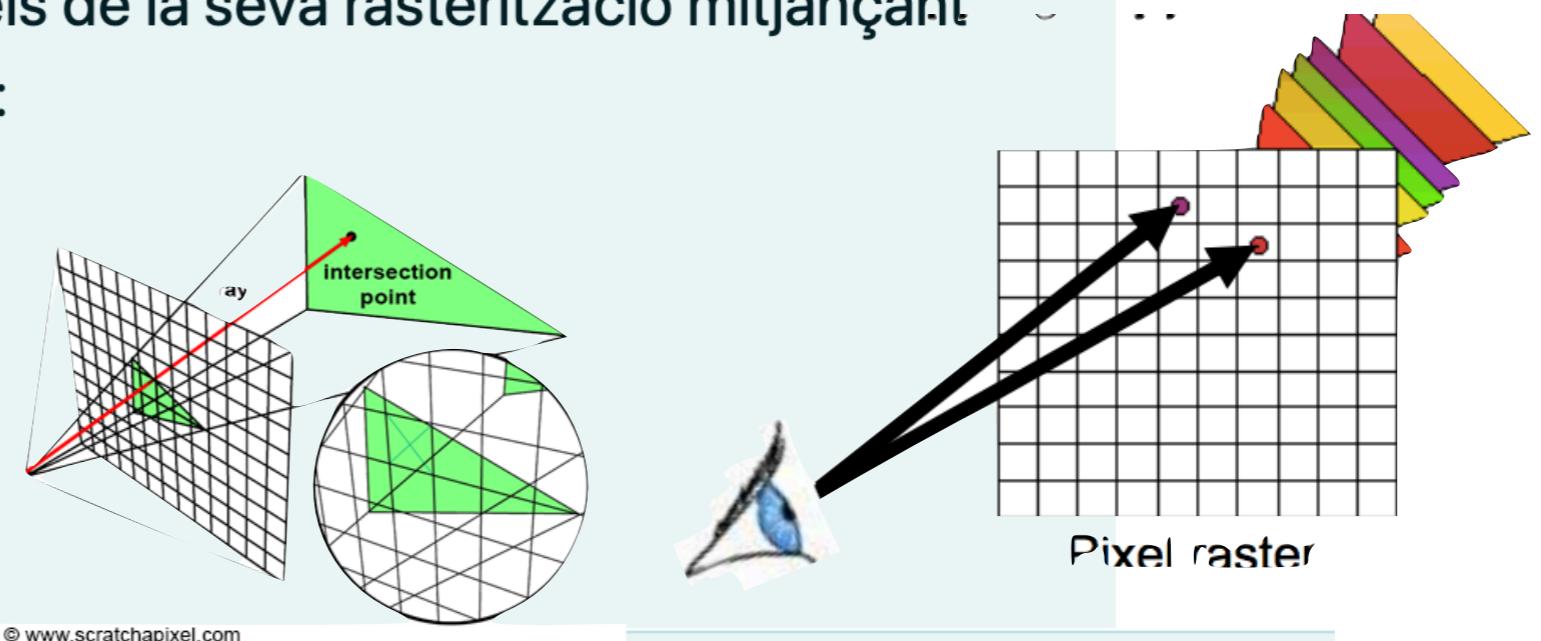
# 2. Recap ZBuffer-Pipeline

El procés de rasterització en el RayTracing implica saber la profunditat de cada píxel i per tant, si s'està visualitzant un triangle, s'han de calcular les profunditats a les que estan els píxels de la seva rasterització mitjançant l'ús de coordenades baricèntriques:

Trieu-ne una:

- a. CERT
- b. FALS

[Esborra la meva selecció](#)

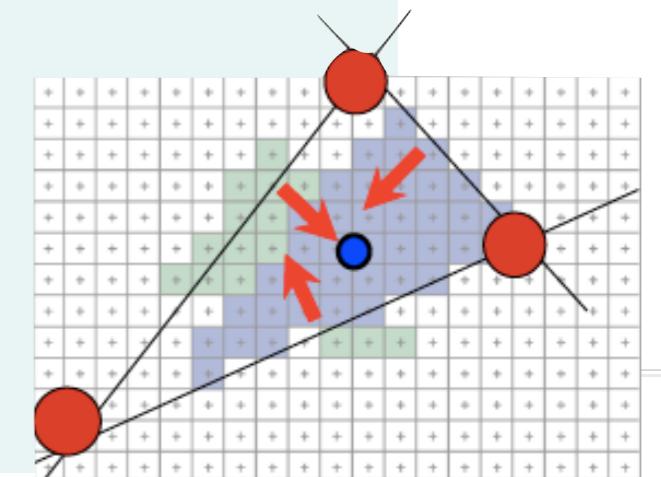


En rasteritzar un triangle, cal calcular la profunditat associada als píxels interiors del triangle a partir de la profunditat dels vèrtexs del triangle i les coordenades baricèntriques associades a cadascun dels píxels interiors.

Trieu-ne una:

- a. CERT
- b. FALS

[Esborra la meva selecció](#)



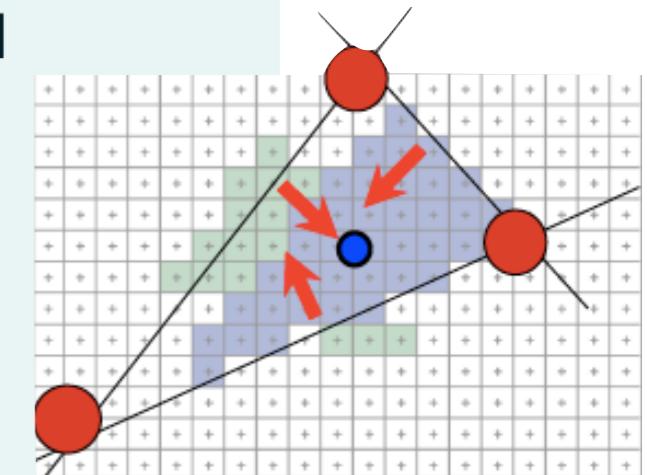
# 2. Recap ZBuffer-Pipeline



En rasteritzar un triangle, la profunditat associada als píxels on es situen els seus vèrtexs està directament calculada gràcies a la transformació de coordenades de món a coordenades d'observador i, per tant, no cal interpolar-la

Trieu-ne una:

- a. CERT
- b. FALS



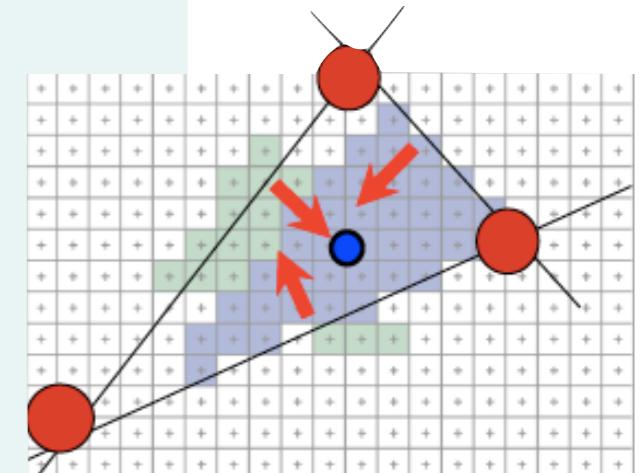
# 2. Recap ZBuffer-Pipeline

En rasteritzar un triangle, la profunditat associada als píxels on es situen els seus vèrtexs està directament calculada gràcies a la transformació de coordenades de món a coordenades d'observador i, per tant, no cal interpolar-la

Trieu-ne una:

- a. CERT
- b. FALS

[Esborra la meva selecció](#)

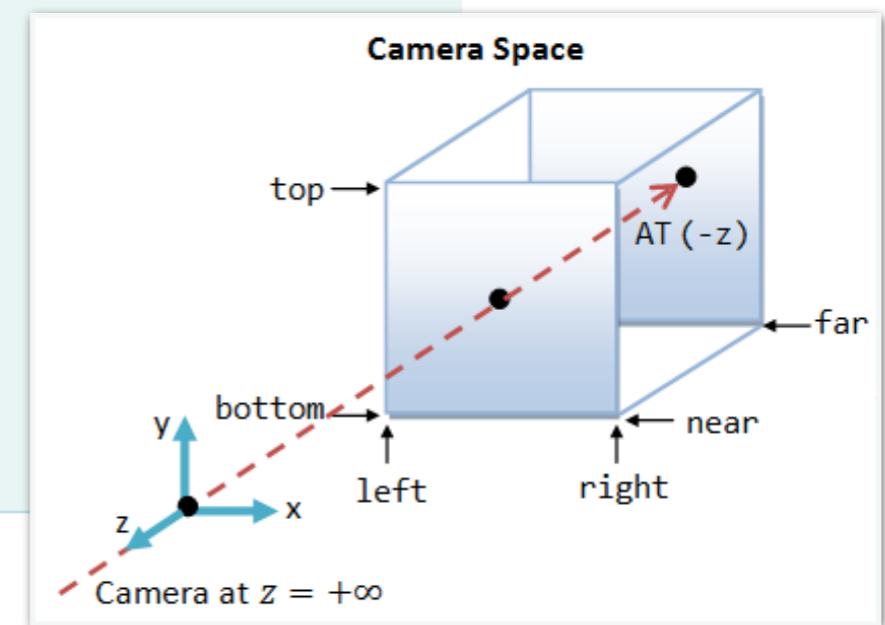


# 2. Recap ZBuffer-Pipeline

Quan s'aplica l'algorisme de ZBuffer, les z's que es comparen estan en coordenades de món

Trieu-ne una:

- a. FALS
- b. CERT



Quan s'aplica l'algorisme de ZBuffer, les z's que es comparen estan en coordenades d'observador o de càmera

Trieu-ne una:

- a. CERT
- b. FALS

# 2. Recap ZBuffer-Pipeline

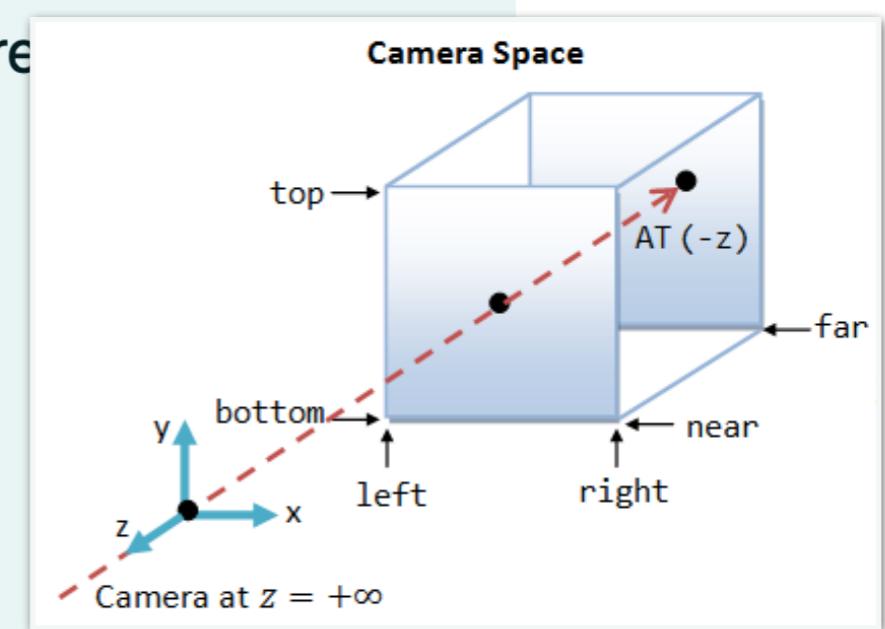
## Coordenades de Z's

Quan s'aplica l'algorisme de ZBuffer, les z's que es comparen estan en coordenades de món

Trieu-ne una:

- a. FALS
- b. CERT

[Esborra la meva selecció](#)



Quan s'aplica l'algorisme de ZBuffer, les z's que es comparen estan en coordenades d'observador o de càmera

Trieu-ne una:

- a. CERT
- b. FALS

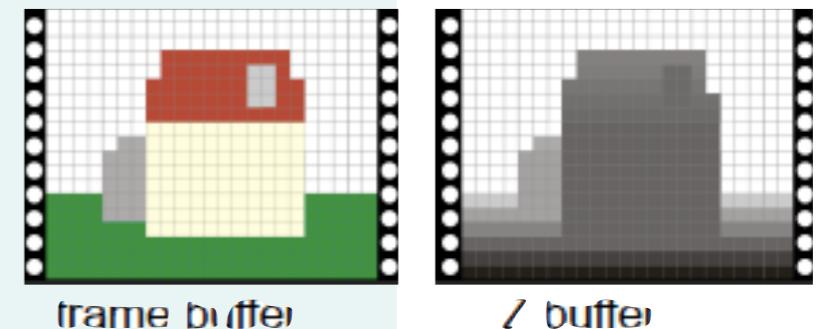
[Esborra la meva selecció](#)

# 2. Recap ZBuffer-Pipeline

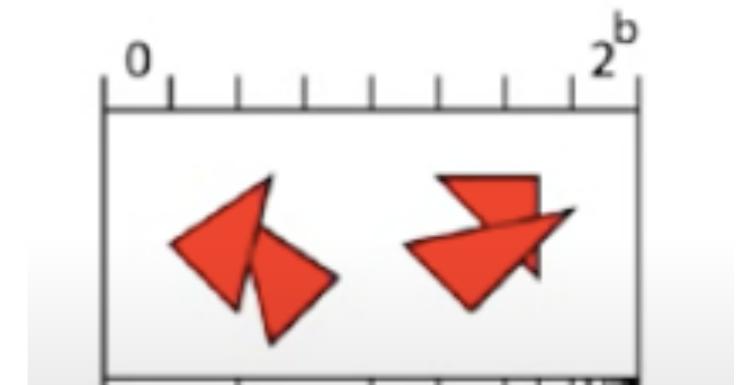
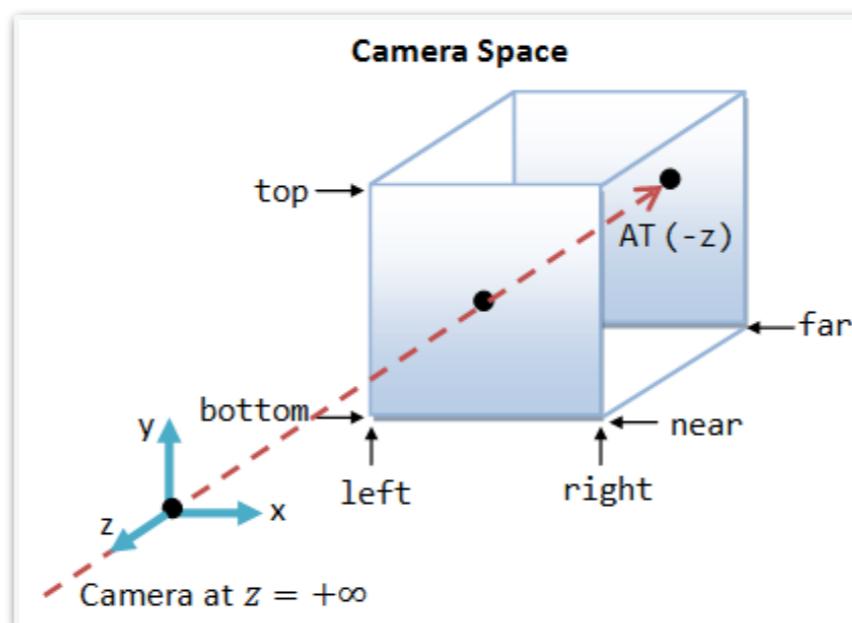
El buffer de profunditats en el ZBuffer guarda z's com a enters positius

Trieu-ne una:

- a. CERT
- b. FALS



Coordenades de Z's



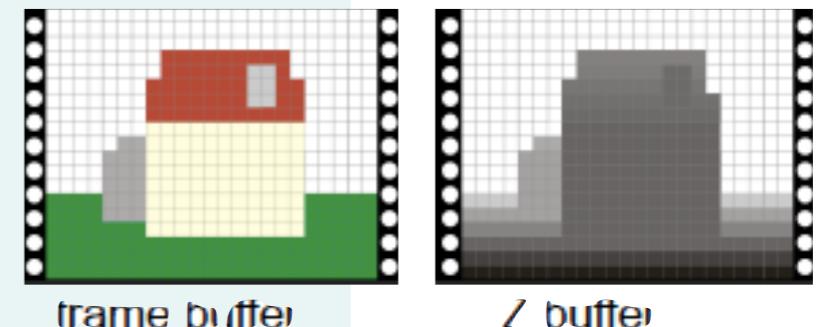
# 2. Recap ZBuffer-Pipeline

El buffer de profunditats en el ZBuffer guarda z's com a enters positius

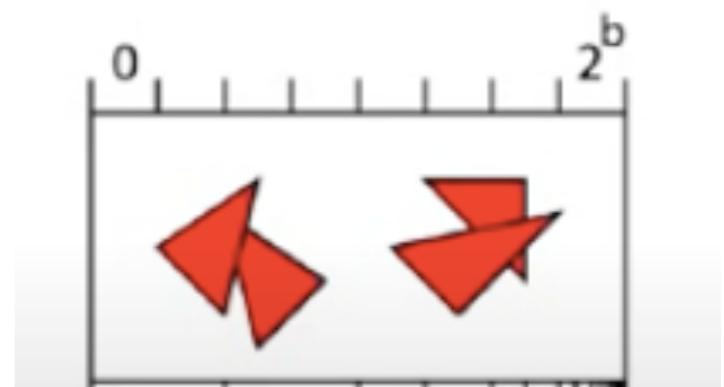
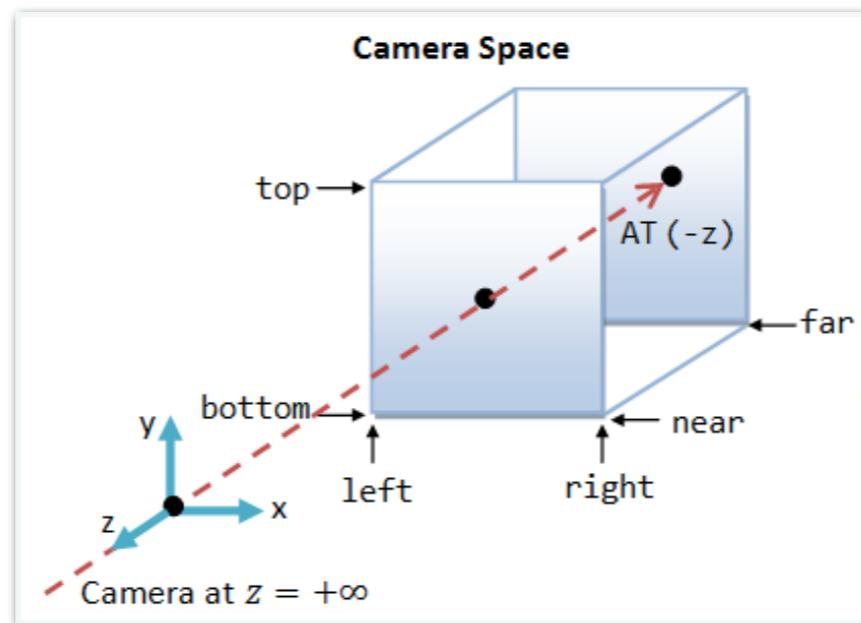
Trieu-ne una:

- a. CERT
- b. FALS

[Esborra la meva selecció](#)



Coordenades de Z's

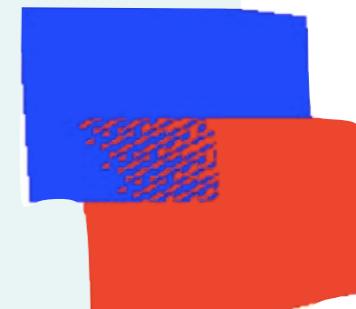


# 2. Recap ZBuffer-Pipeline

L'efecte de Z-fighting ve donat per què el buffer de profunditats (o depth buffer) enmagatzema enters i no floats. **sobretot**

Trieu-ne una:

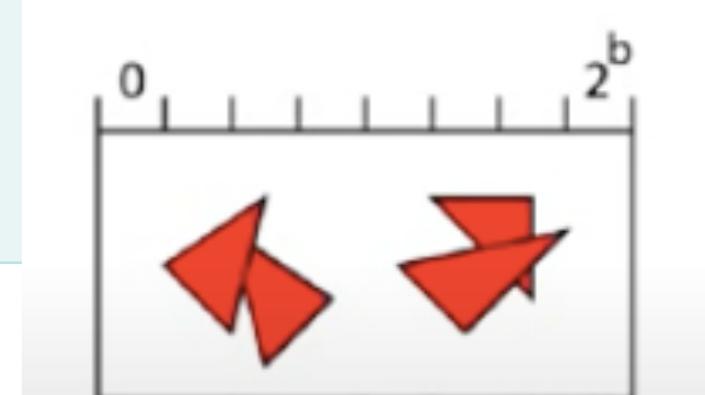
- a. FALS
- b. CERT



La precisió de buffer de profunditats (o depth buffer) no es pot graduar pels plans de retallat anterior posteriors (`z_near` i `zfar`), ja que ve determinada pel tipus de targeta gràfica

Trieu-ne una:

- a. FALS
- b. CERT



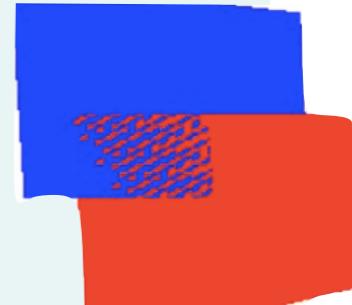
# 2. Recap ZBuffer-Pipeline

L'efecte de Z-fighting ve donat per què el buffer de profunditats (o depth buffer) enmagatzema enters i no floats.

Trieu-ne una:

- a. FALS
- b. CERT

[Esborra la meva selecció](#)

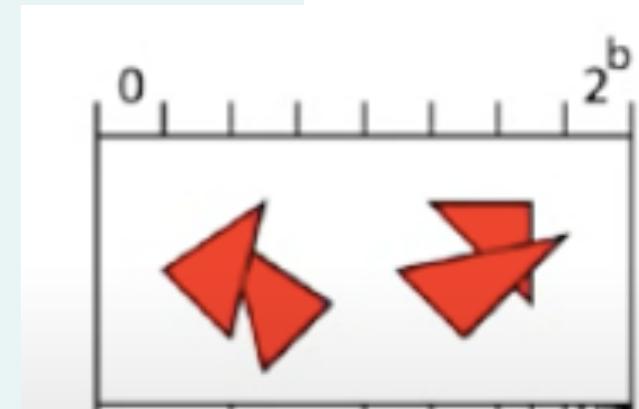


La precisió de buffer de profunditats (o depth buffer) no es pot graduar pels plans de retallat anterior posteriors (`z_near` i `zfar`), ja que ve determinada pel tipus de targeta gràfica

Trieu-ne una:

- a. CERT
- b. FALS

[Esborra la meva selecció](#)



# Càcul de la visibilitat

## Z-Buffer

Els valors de Z sempre són positius i van des del pla de clipping anterior ( $z_{near}$ ) al posterior ( $Z_{far}$ ).

El z-buffer enmagatzema **enters** positius  
 $b = \{0, 1, \dots, B-1\}$

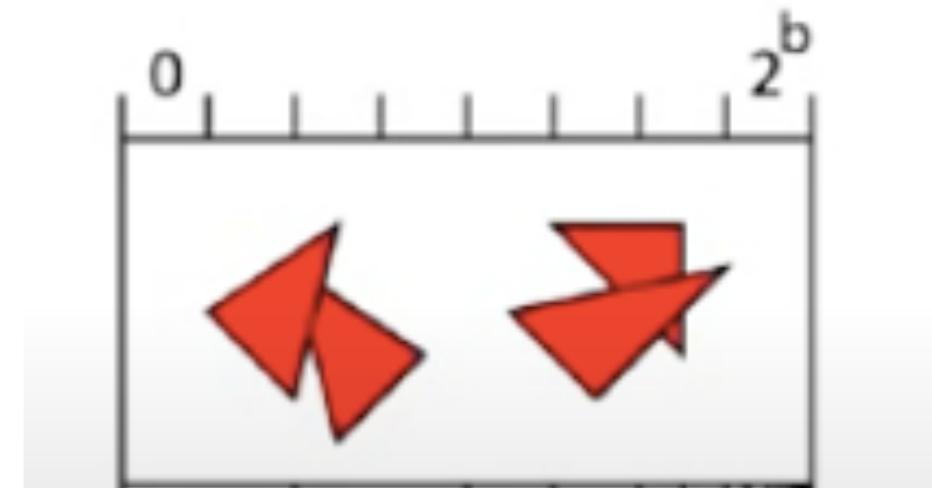
Els valors de z's en coordenades de càmera es mapegen a aquests valors, en intervals regulars de mida

$$\Delta z = \frac{(Z_{far} - Z_{near})}{B}$$

I un z d'un punt es calcula com:

$$\frac{z - z_{near}}{Z_{far} - z_{near}} * (B - 1)$$

Si el zbuffer és un canal de b bits, es tenen  $2^b$  buckets

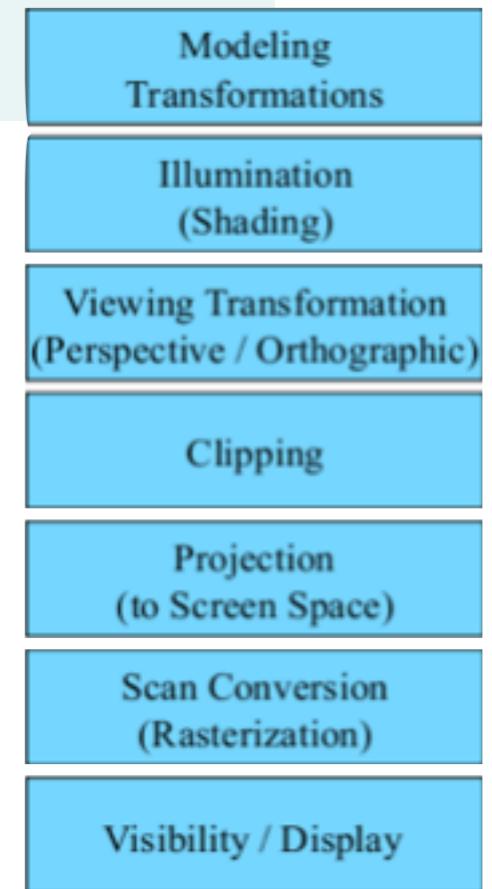
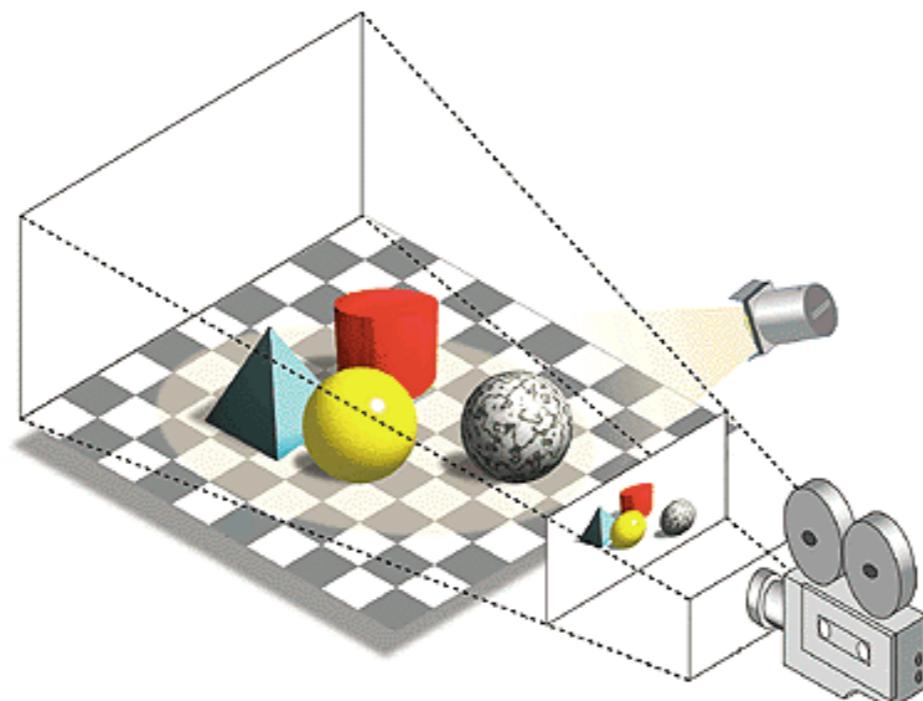


# 2. Recap ZBuffer-Pipeline

La visibilitat en l'algorisme de Raytracing ve donada pel càlcul de la intersecció més propera al raig primari en cas que els objectes siguin opacs

Trieu-ne una:

- a. CERT
- b. FALS



# 2. Recap ZBuffer-Pipeline

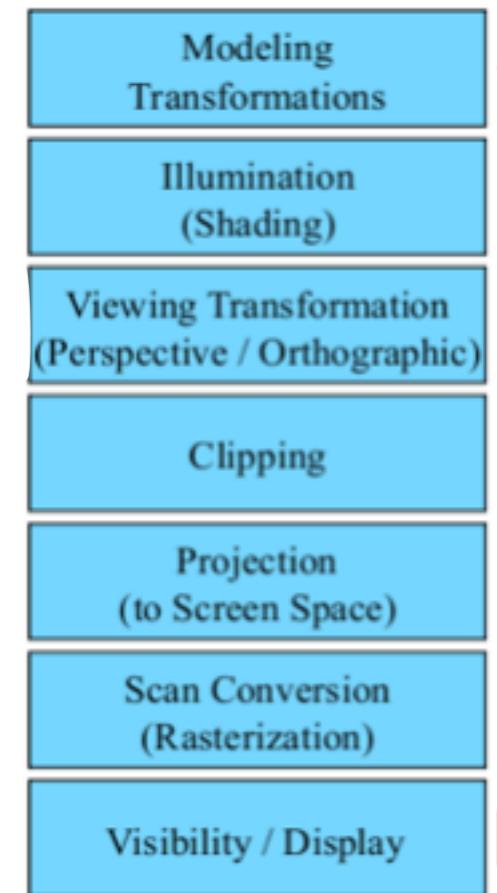
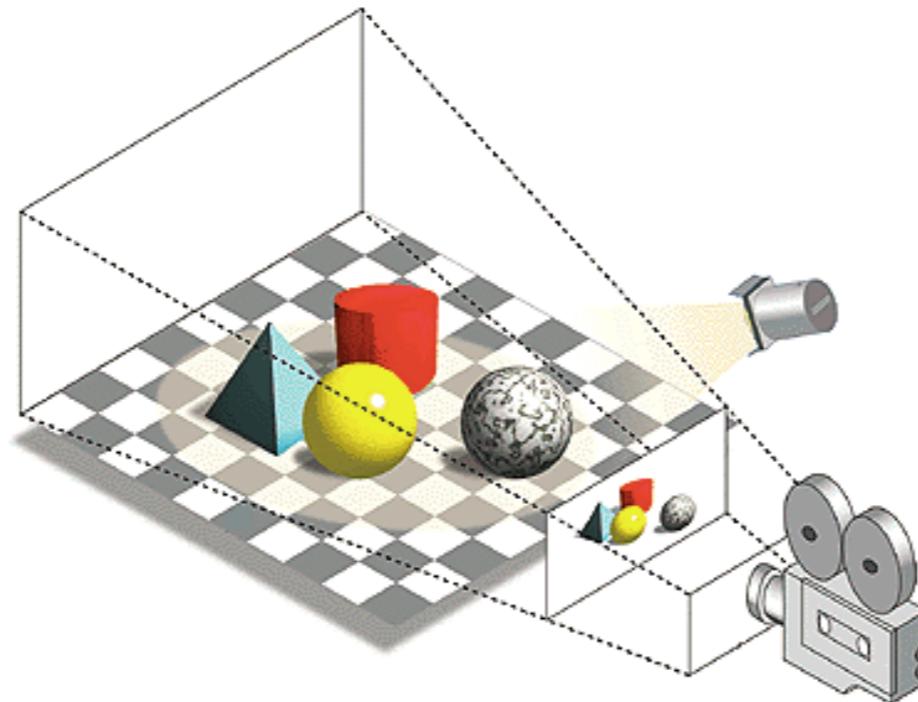
La visibilitat en l'algorisme de Raytracing ve donada pel càlcul de la intersecció més propera al raig primari en cas que els objectes siguin opacs

Trieu-ne una:

- a. CERT
- b. FALS

[Esborra la meva selecció](#)

Visibilitat a RayTracing?



# 2. Recap ZBuffer-Pipeline

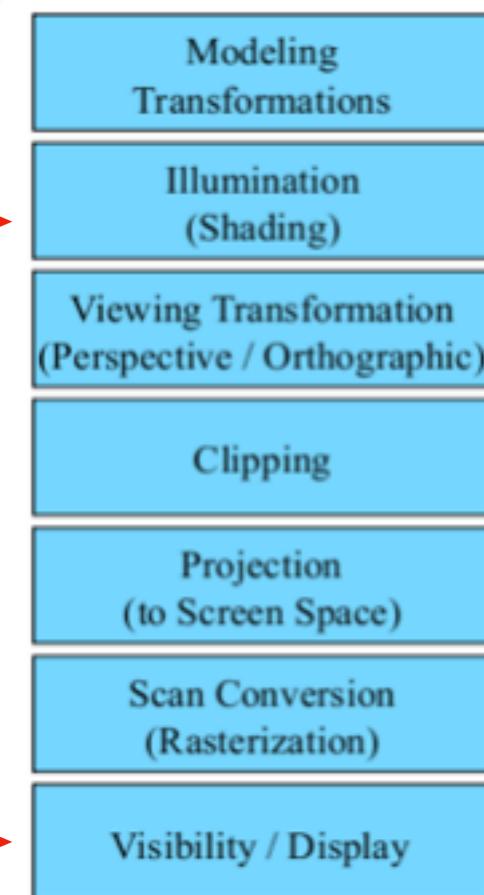
Coordinades de món o de càmera?



L'etapa d'il·luminació en el pipeline de visualització es pot fer tant abans → com després de la transformació de coordenades de món a coordenades de càmera (o d'observador)

Trieu-ne una:

- a. CERT
- b. FALS



L'etapa de zbuffer en el pipeline de visualització es pot fer tant abans com després de la transformació de coordenades de món a coordenades de càmera (o d'observador)

Trieu-ne una:

- a. CERT
- b. FALS

# Model de Blinn-Phong

En els mètodes projectius el model més utilitzat és el model del Blinn-Phong (veure tema 2b)

$$I_{total} = I_{a_{global}}K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d \max(L_i \cdot N, 0.0) + I_{s_i} K_s \max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$

Es necesiten:

- les normals
- les llums (posicions i característiques)
- la càmera
- les propietats dels materials

# 2. Recap ZBuffer-Pipeline

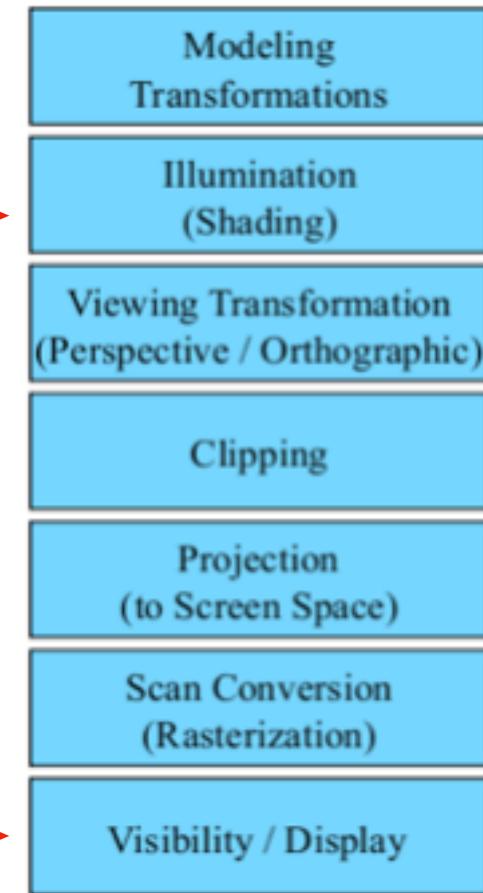
Coordenades de món o de càmera?

L'etapa d'il·luminació en el pipeline de visualització es pot fer tant abans com després de la transformació de coordenades de món a coordenades de càmera (o d'observador)

Trieu-ne una:

- a. CERT
- b. FALS

Esborra la meva selecció



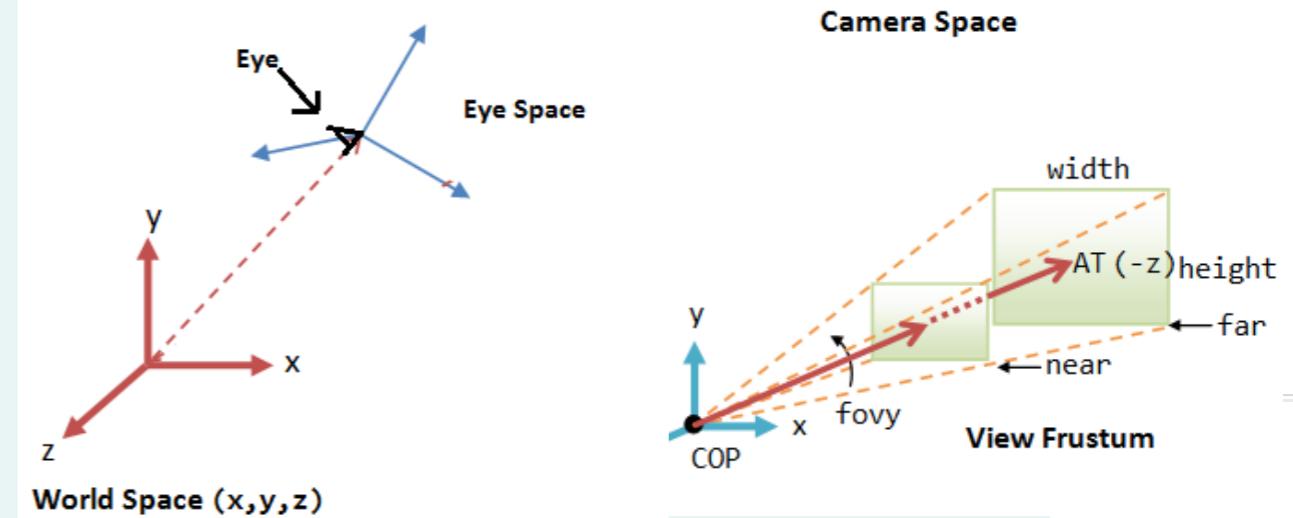
Ordre de les etapes

L'etapa de zbuffer en el pipeline de visualització es pot fer tant abans com després de la transformació de coordenades de món a coordenades de càmera (o d'observador)

Trieu-ne una:

- a. CERT
- b. FALS

Esborra la meva selecció

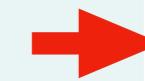


# 2. Recap ZBuffer-Pipeline

Coordenades de món o de càmera?  
Ordre de les etapes

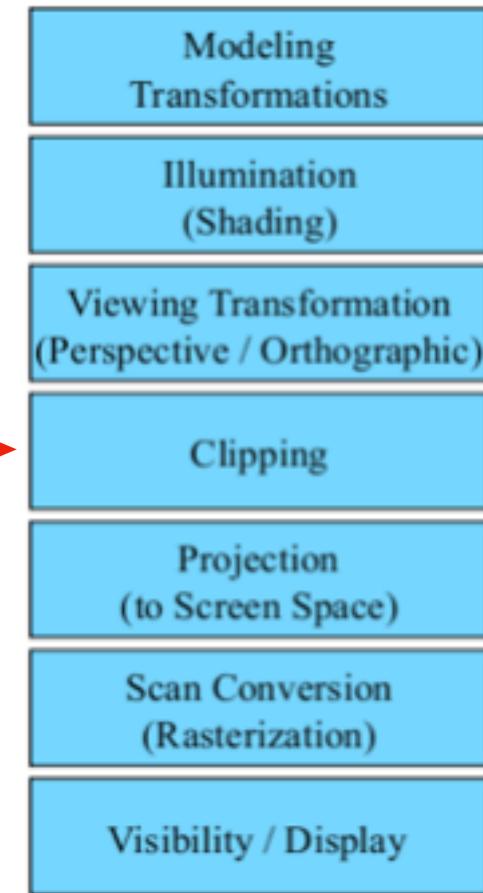
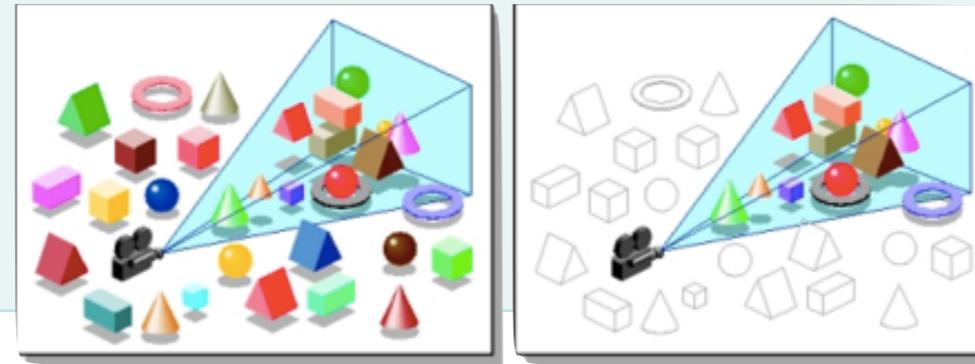


L'etapa de clipping en el pipeline de visualització es pot fer tant abans com després de la transformació de coordenades de món a coordenades de càmera (o d'observador)



Trieu-ne una:

- a. FALS
- b. CERT



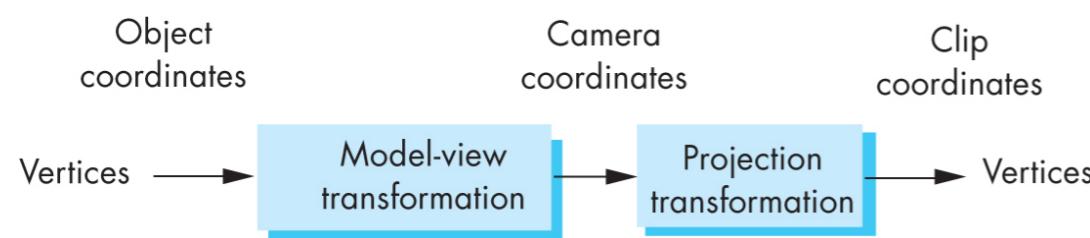
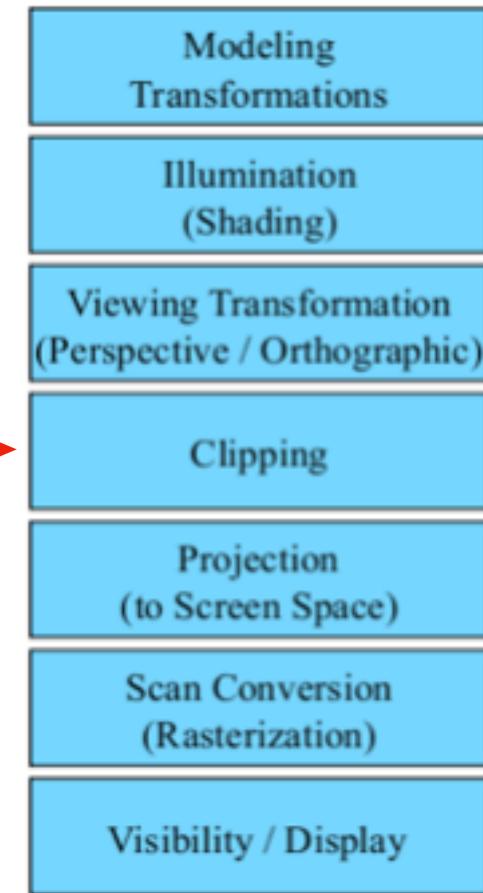
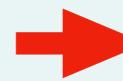
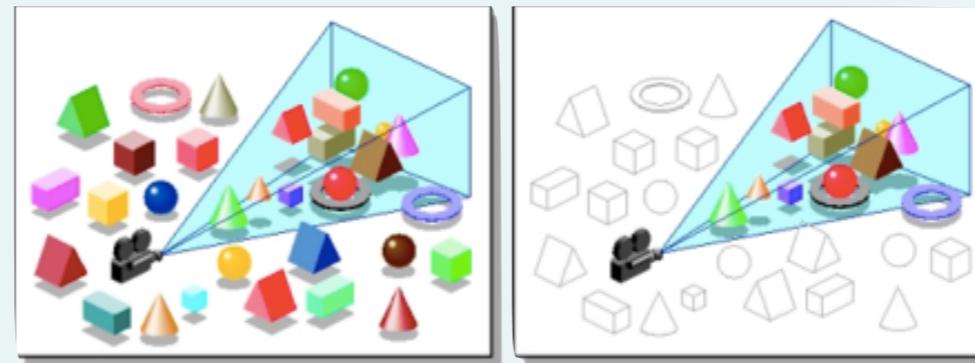
# 2. Recap ZBuffer-Pipeline

Coordenades de món o de càmera?

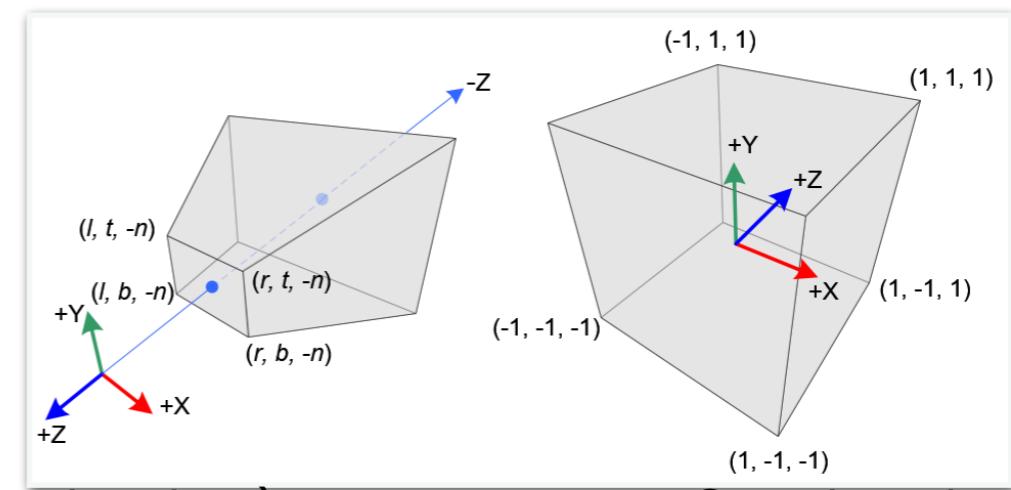
L'etapa de clipping en el pipeline de visualització es pot fer tant abans com després de la transformació de coordenades de món a coordenades de càmera (o d'observador)

- Trieu-ne una:
- a. FALS
  - b. CERT

Esborra la meva selecció



Convencions de GL:

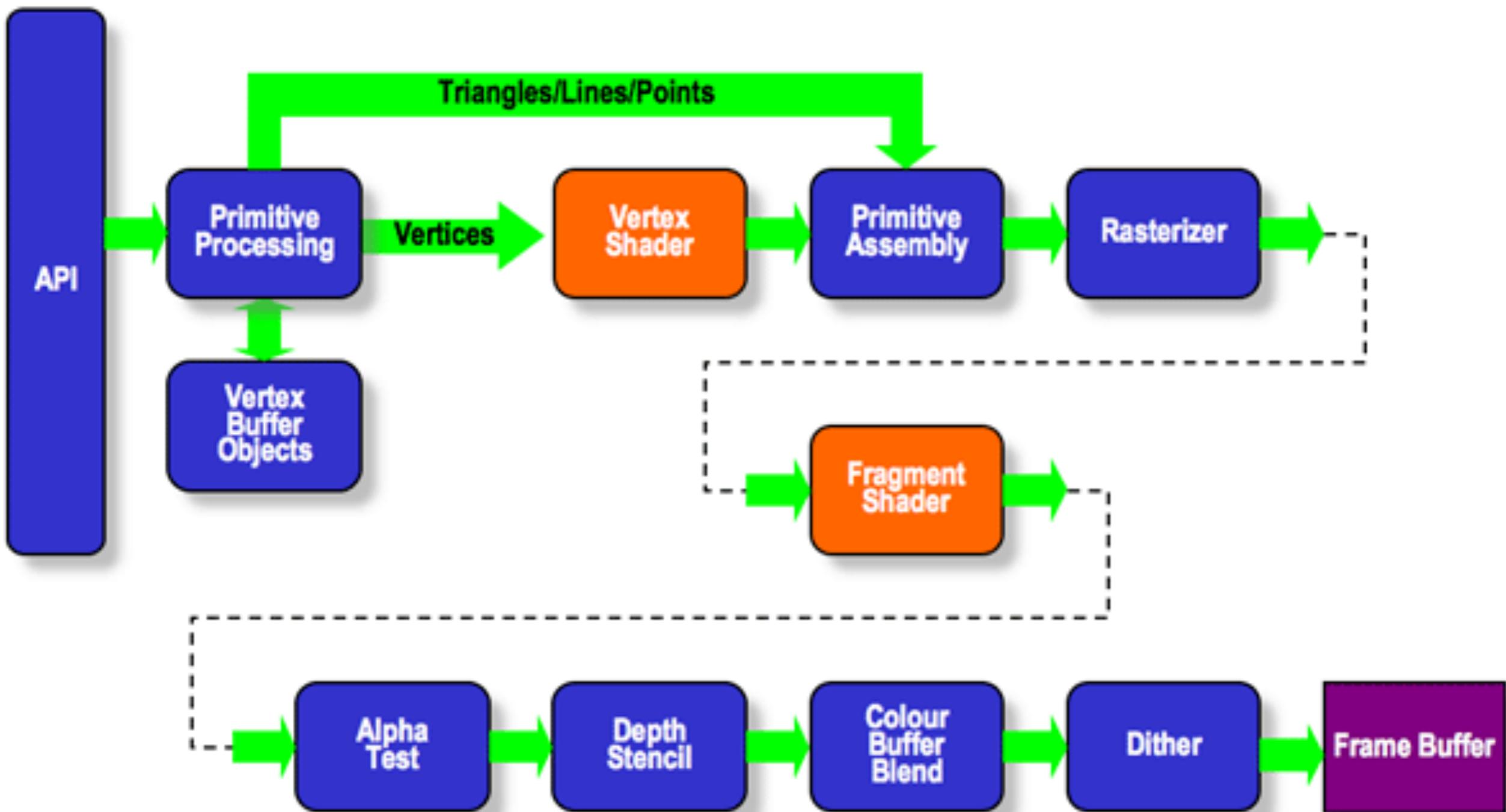


Coordenades de càmera  
en el sistema de la mà  
dreta

Coordenades  
normalitzades en el  
sistema de la mà esquerra

# 2. Recap ZBuffer-Pipeline

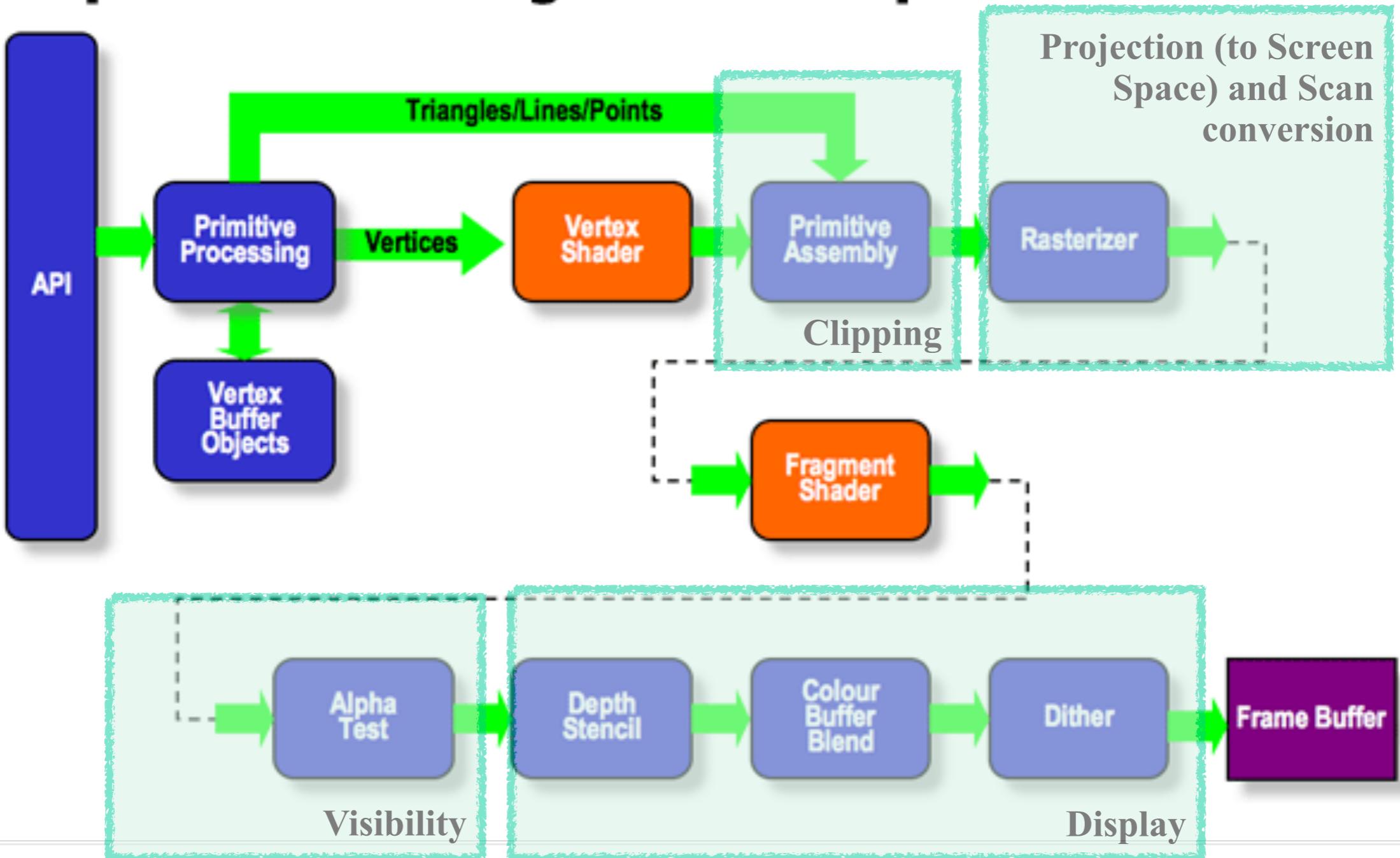
## OpenGL ES 2.0 Programmable Pipeline



# 2. Recap ZBuffer-Pipeline

- Com es mapeja el pipeline amb OpenGL?

## OpenGL ES 2.0 Programmable Pipeline



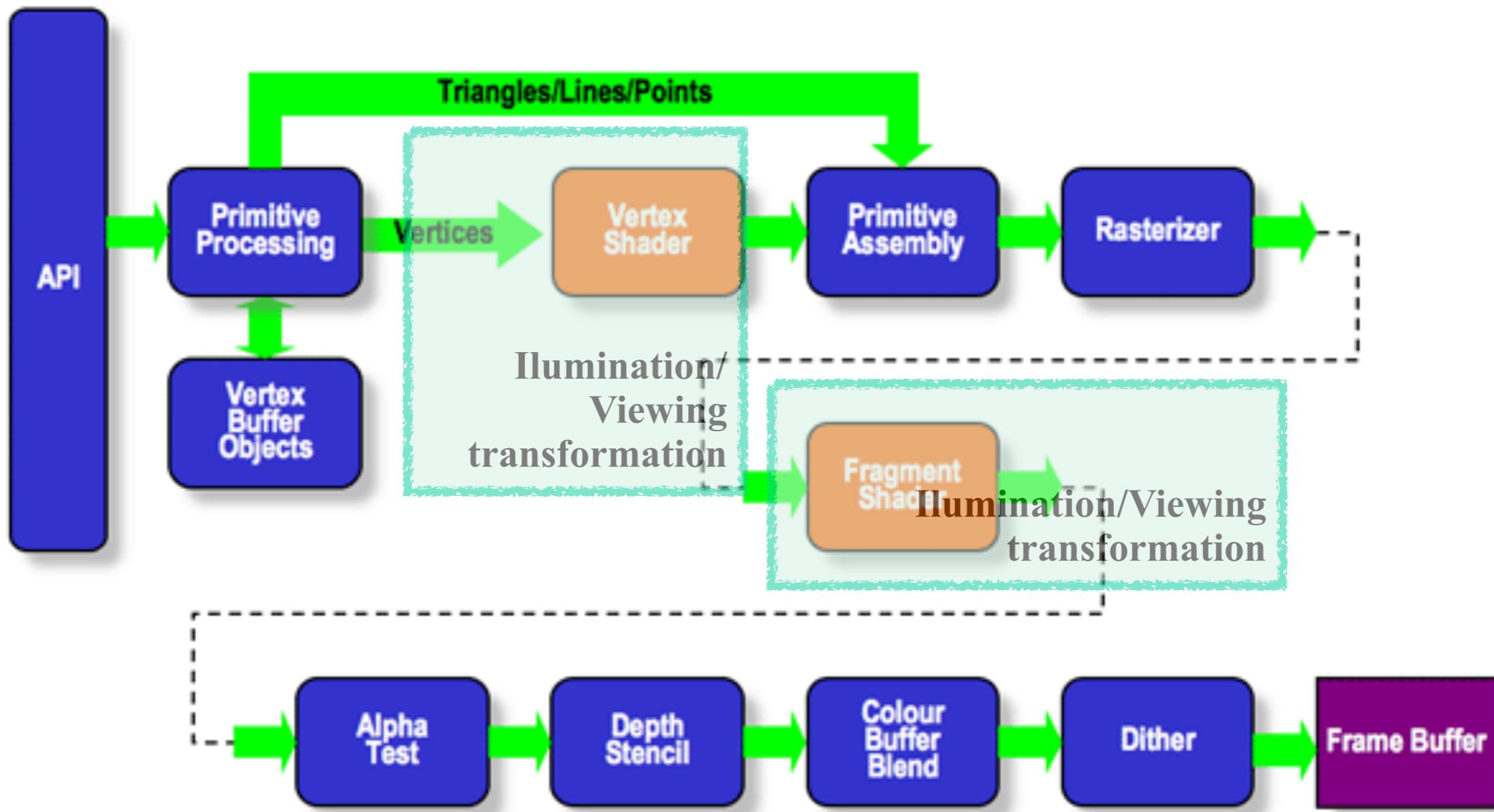
Modeling Transformations
Illumination (Shading)
Viewing Transformation (Perspective / Orthographic)
Clipping
Projection (to Screen Space)
Scan Conversion (Rasterization)
Visibility / Display

# 2. Recap ZBuffer-Pipeline

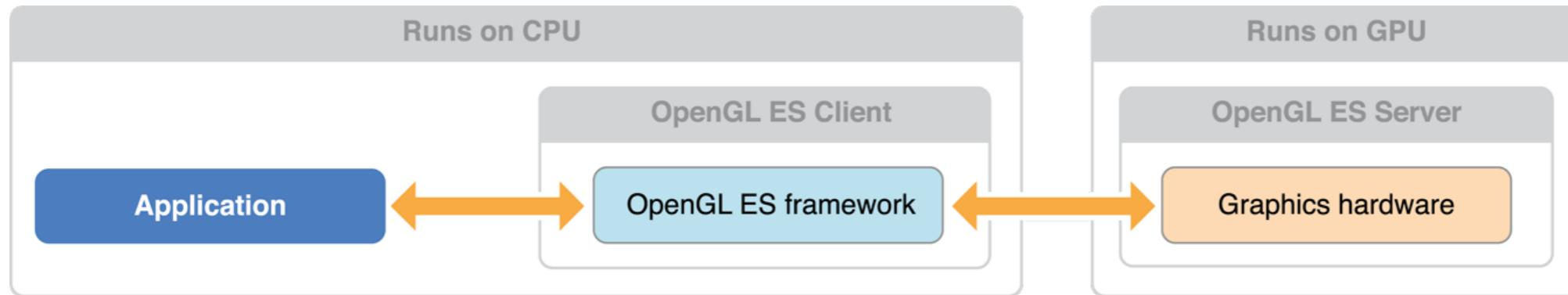


- Com es mapeja el pipeline amb OpenGL?

## OpenGL ES 2.0 Programmable Pipeline



# 3. ACTIVITAT CPU-GPU



Client:

1. S'inicialitzen els programes o **shaders** a ser executats a la GPU (*vertex shader* i *fragment shader*)
2. S'envien les dades a la GPU (vèrtexs, normals, llums, materials): uniform/ in
3. Per a cada objecte, s'executa el pintat (glDraw)

Servidor:

1. Carrega els **shaders** a cada core de la GPU
2. Es guarden a memòria les dades
3. S'executa el *pipeline* de GL

```
layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vColor;

uniform mat4 model_view;
uniform mat4 projection;

out vec4 color;

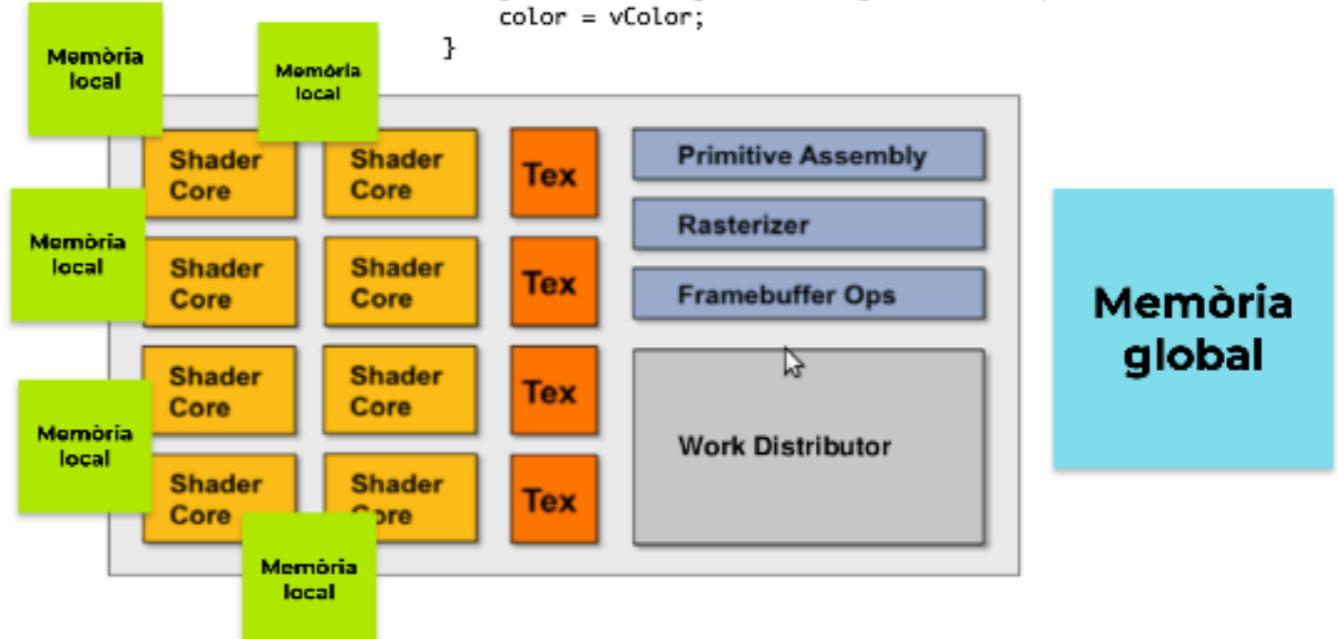
void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;
    color = vColor;
}
```

## GLWidget

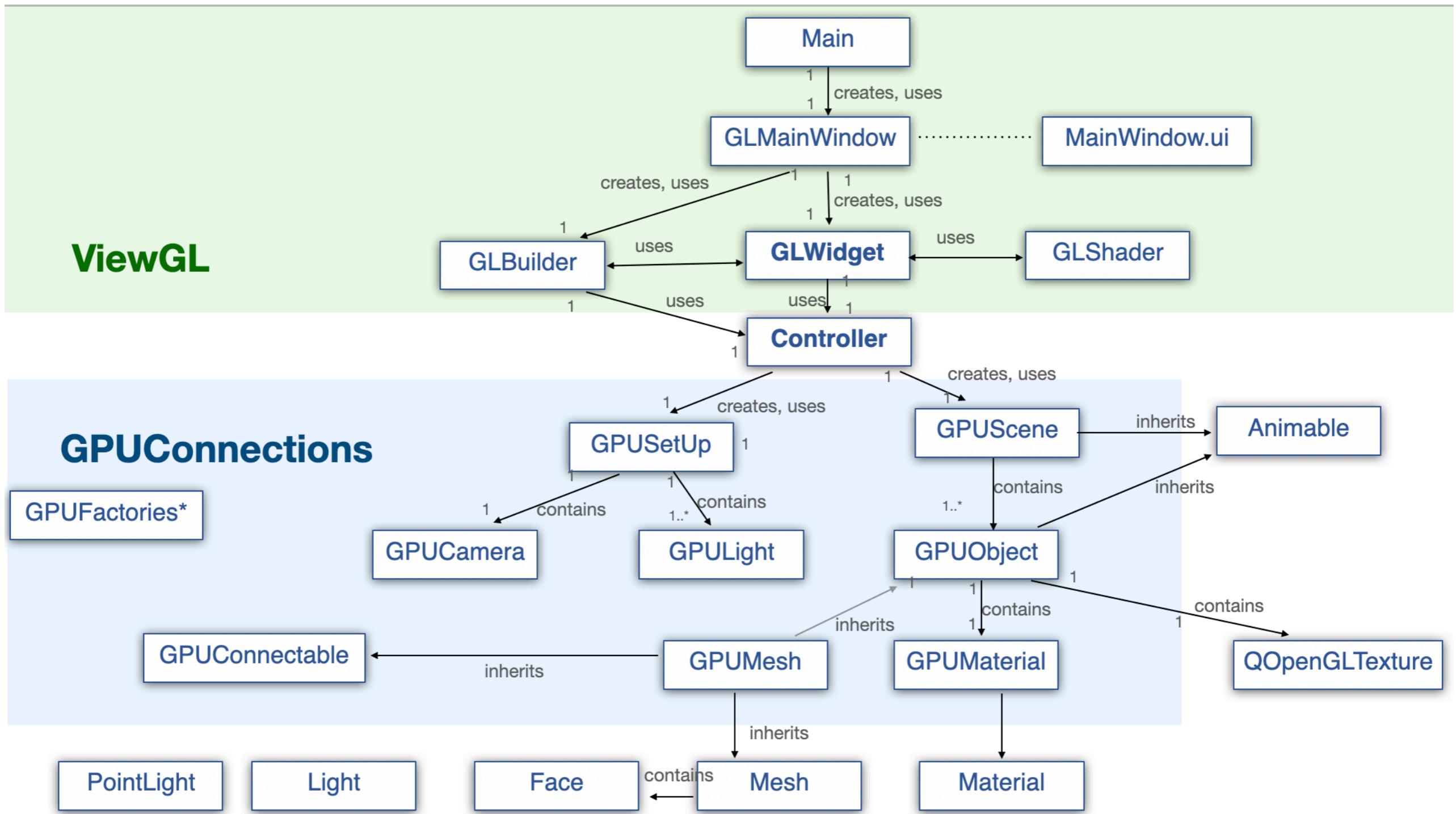
initializeGL()

paintGL()

altres mètodes repartits (GPUScene, GPMesh, GPU Camera, GPUMaterial, GPU Lum, ...)



# 3. ACTIVITAT CPU-GPU



$$I_{total} = I_{global} K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d max(L_i \cdot N, 0.0) + I_{s_i} K_s max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$

# 3. ACTIVITAT CPU-GPU

## INSTRUCCIONS

A cada fila es té un objecte necessari pel càlcul de la il·luminació. Suposa que el color final es calcularà usant la fórmula de Blinn-Phong en el vèrtex shader

1. Activitat Individual: Tria un color que et representi. Usa les Sticky Notes d'aquell color per a omplir les caselles buides
2. Usa una Sticky Note per casella.
3. Hi han algunes Sticky Notes amb text predefinit, però si necessites algun altre text, escriu-lo a la nota.
4. Usa els emojis per indicar el grau de seguretat que tens a la resposta.



Activitat 1						
CPU-to-GPU	Granularitat	Creació		Comunicació CPU-GPU		
	és comú a nivell de ....	On es creen les dades? Classe	On es crea? (mètode)	Des d'on s'initialitzen les estructures? On va el codi de toGPU?	Des d'on es pasen a la GPU (on es crida a toGPU)	Reenviament a la GPU (en cas de canvis)
<b>Program (parell vertex i fragment shader)</b> Programes a executar en el pipeline de la GPU						
<b>Vèrtexs d'un objecte</b> Conjunt de vèrtexs d'una malla triangular						
<b>Normals d'un objecte</b> Conjunt de normals d'una malla triangular						
<b>Càmera</b> Implica les matrius modelview i projection, i la posició de l'observador						
<b>Llum global ambient</b>	Scene	hardcoded? Fixer?	via fitxer	hardcoded: QWidget?	metode??	GPUSETUP
<b>Llums</b>						mètode i classe?
<b>Material</b> Propietats òptiques associades a un objecte						
	vèrtex objecte					
	Free Sticky Notes					
	vèrtex objecte Scene		via fitxer	nomClasse	nomMetode	
						InitializeGL()
						paintGL()
						Altres
						IN/OUT uniform
						IN/OUT uniform

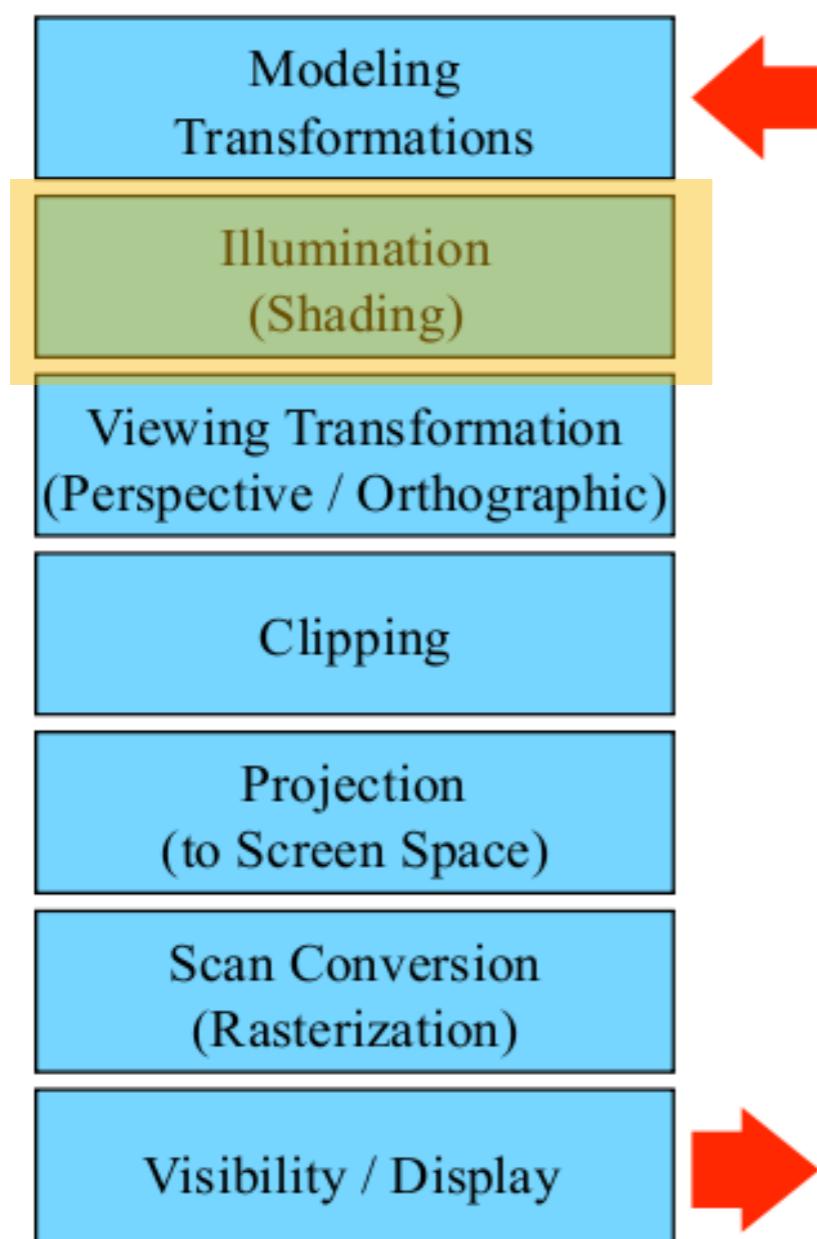


[https://miro.com/app/board/o9J\\_IHAwt34=/?share\\_link\\_id=213087106125](https://miro.com/app/board/o9J_IHAwt34=/?share_link_id=213087106125)

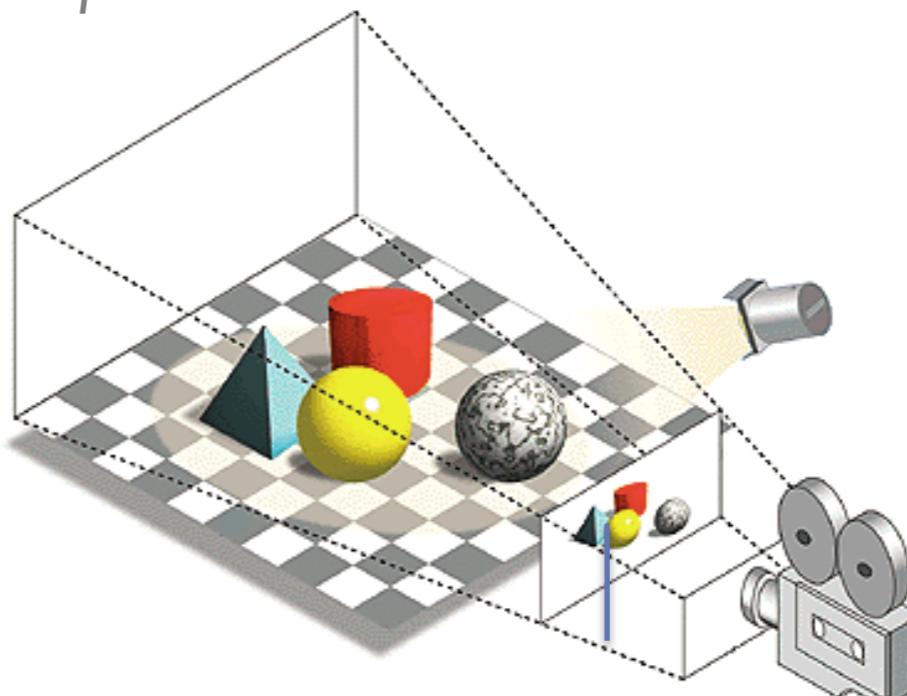
Fes una còpia en el mateix projecte de miro en el teu usuari de Miro i treballa allà!

# 4. Shading a la GPU

**Pipeline de visualització:** conjunt d'etapes\* que a partir de l'escena 3D, llums, càmera i viewport, permet obtenir la imatge 2D amb els colors finals.

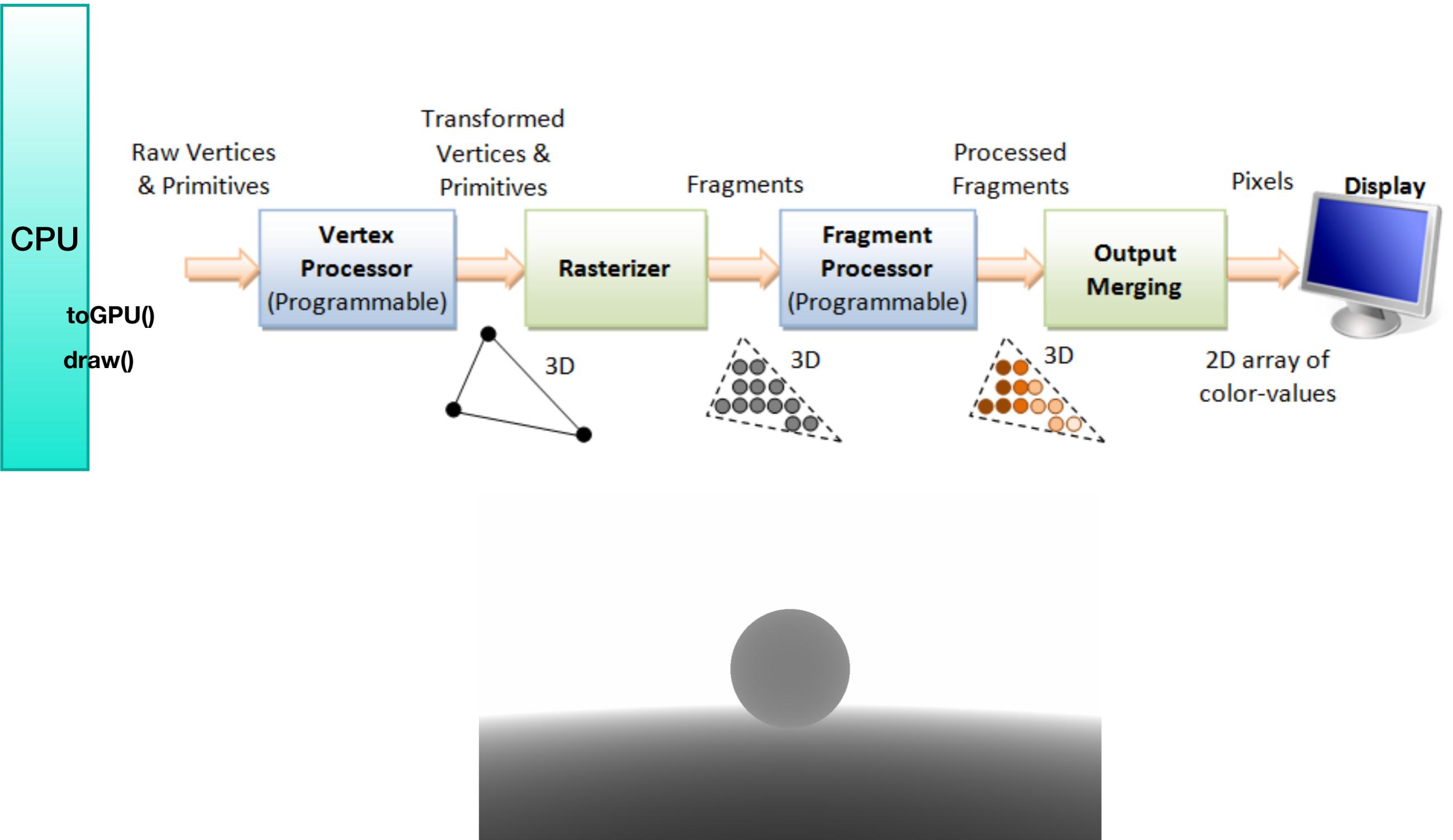


- **Input:** Objectes, Llum, Càmera i viewport



- **Output:** Frame buffer (Colors/Intensitats (ex. 24-bit RGBA a cada píxel))

# 4. Shading a la GPU



# 4. Shading a la GPU

En els mètodes projectius el model més utilitzat és el model del Blinn-Phong (veure tema 2b)

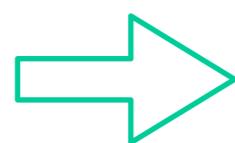
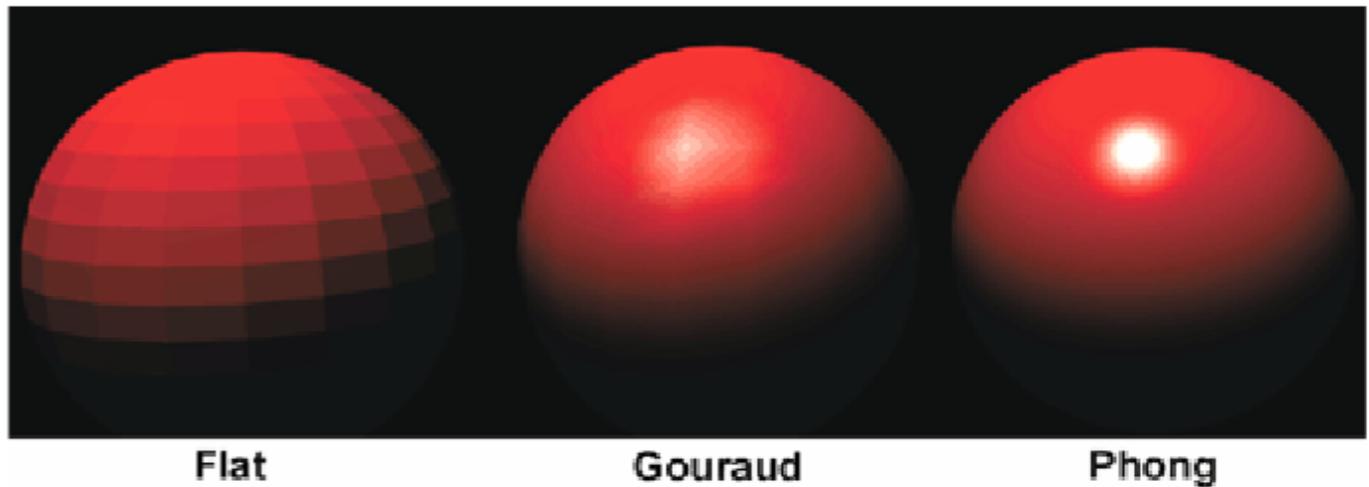
$$I_{total} = I_{a_{global}}K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d \max(L_i \cdot N, 0.0) + I_{s_i} K_s \max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$

El shading en malles poligonals pot ser calculat de diferents maneres:

- flat shading
- suau ( smooth i Gouraud )
- Phong shading

Es necessiten:

- les normals
- les llums
- la càmera
- les propietats dels materials

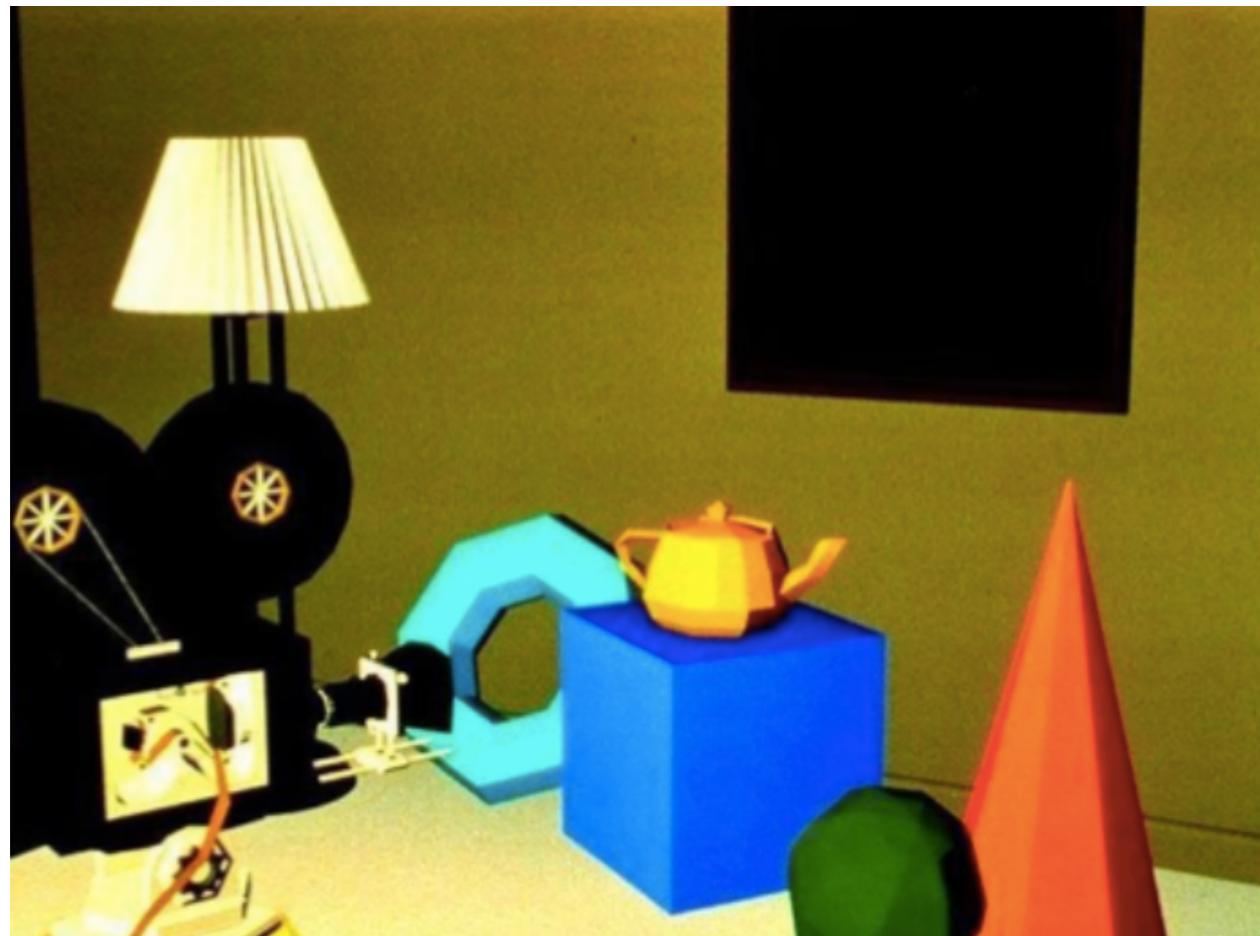


S'envia tota la informació als shaders

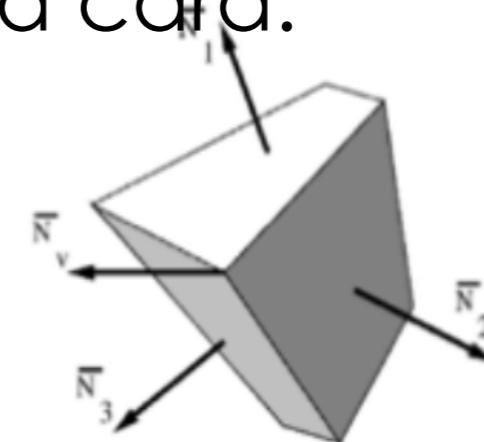
## 3.4. Il·luminació usant *shaders*

### Flat shading:

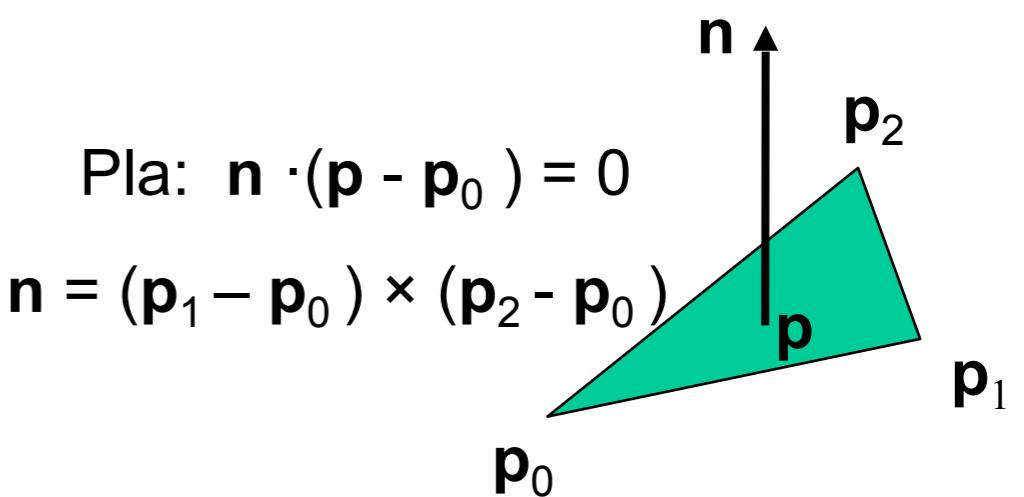
- tots els punts d'una cara es pinten d'un color molt similar, s'usa la normal al pla de la cara.



Mach banding problem



$$\text{Pla: } \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$$
$$\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$$



$$\text{normalització } \mathbf{n} = \mathbf{n} / |\mathbf{n}|$$

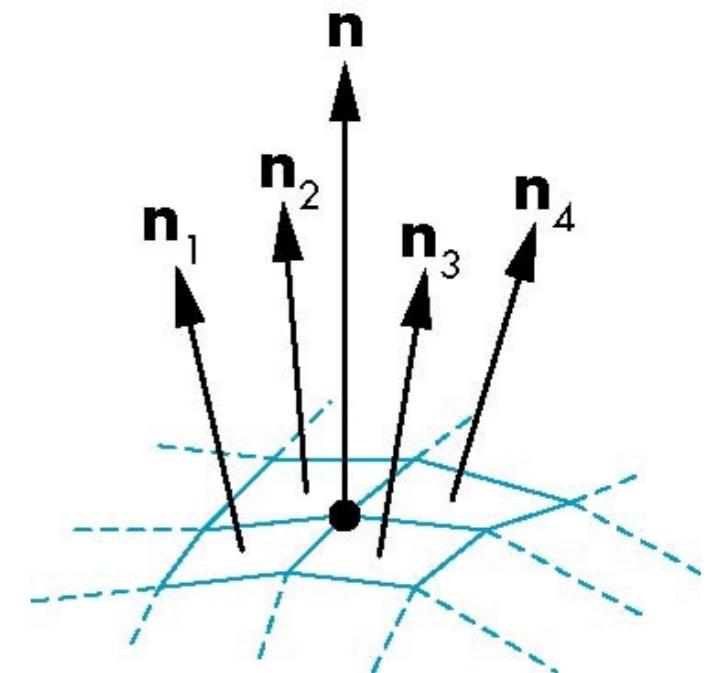
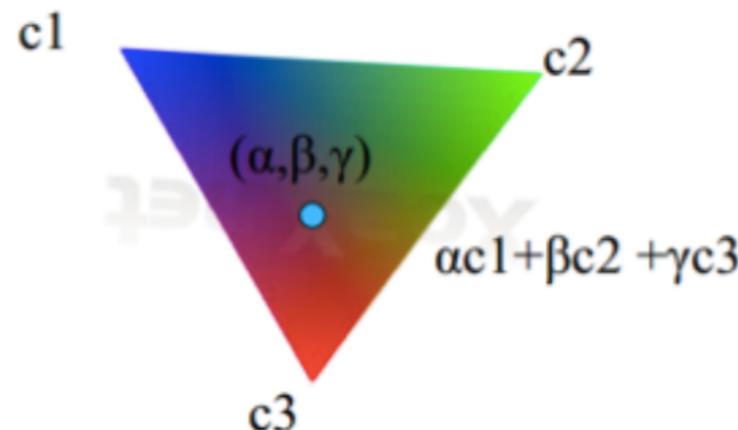
## 3.4. Il·luminació usant *shaders*

### Gouraud Shading

Usat en models **poligonals** per calcular les normals a cada vèrtex:

1. Es calcula el promig de les normals a cada vèrtex
2. S'aplica el model de Blinn-Phong a cada vèrtex
3. S'interpolen les intensitats de cada vèrtex a cada polígon

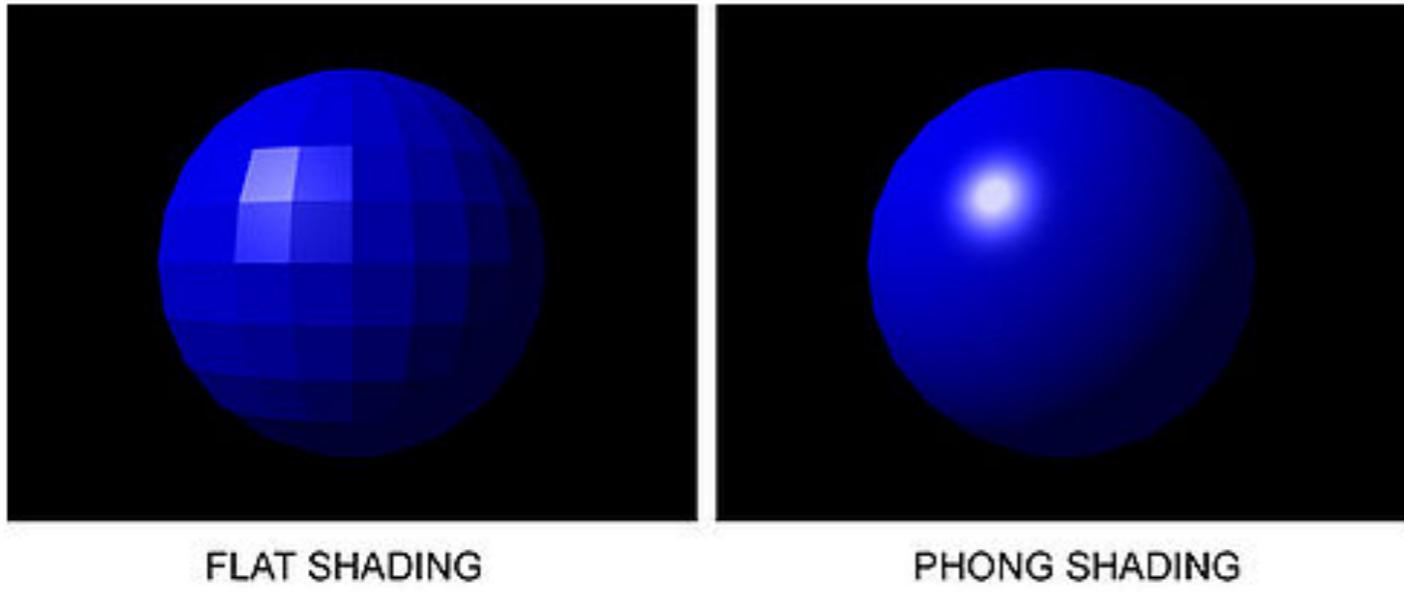
$$\mathbf{n} = (\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4) / |\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|$$



## 3.4. Il·luminació usant *shaders*

### Phong shading

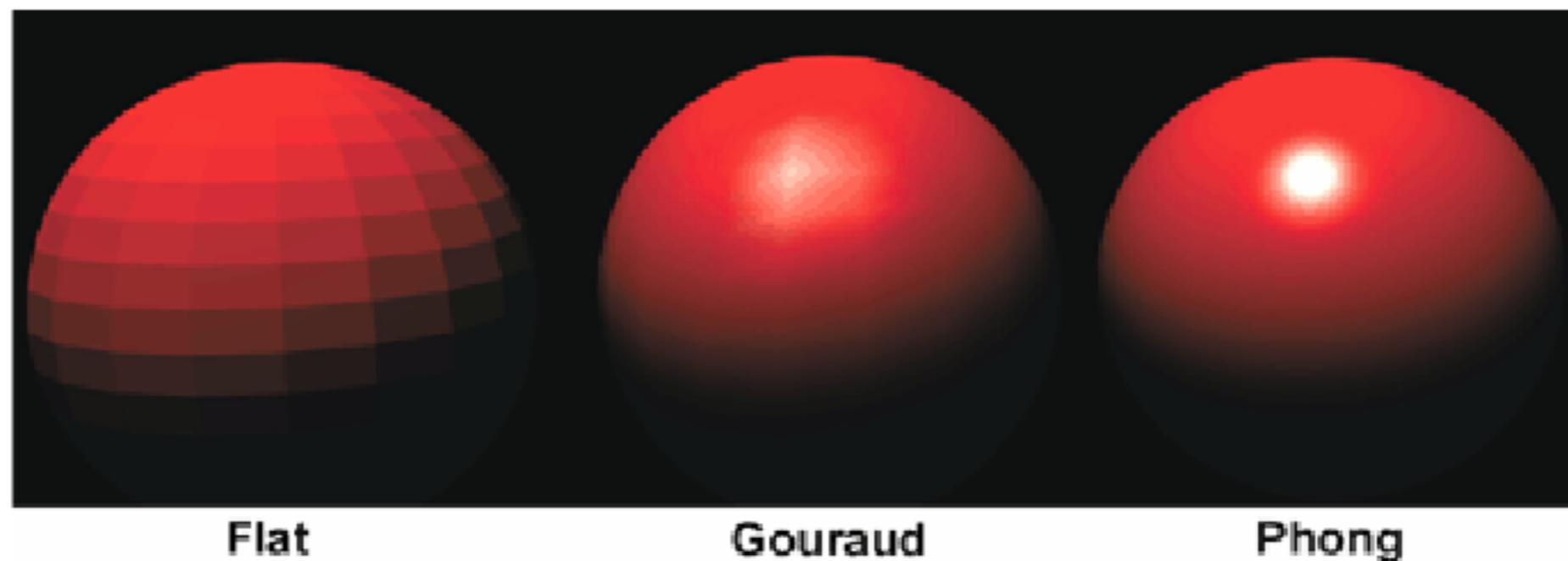
1. Es troben les normals a cada vèrtex
2. S'interpolen les normals en els punts interiors (fragments) del polígon



## 3.4. Il·luminació usant *shaders*

Si els polígons aproximen una superfície amb **curvatures molt altes**, el Phong shading es veu més suau que el Gouraud

El Phong shading implica més càlculs que el Gouraud shading



## 3.4. Il·luminació usant *shaders*

Vertex shader

```
#version 330

layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vColor;

uniform mat4 model_view;
uniform mat4 projection;

out vec4 color;

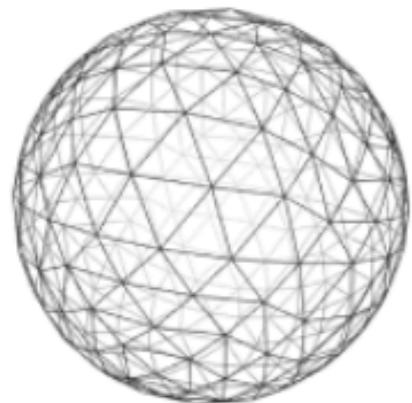
void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;
    color = vColor;
}
```

Fragment shader

```
#version 330

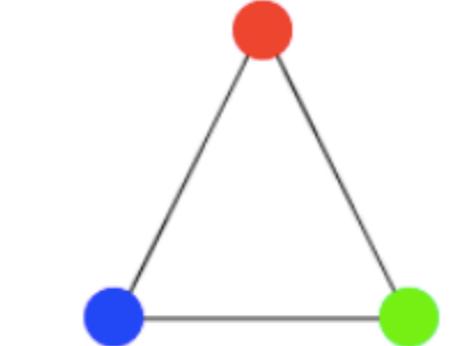
in vec4 color;
out vec4 colorOut;

void main()
{
    colorOut = color;
}
```



Mesh

Attributes →



Vertex Shader



gl\_Position

Uniforms



Varyings

out/in



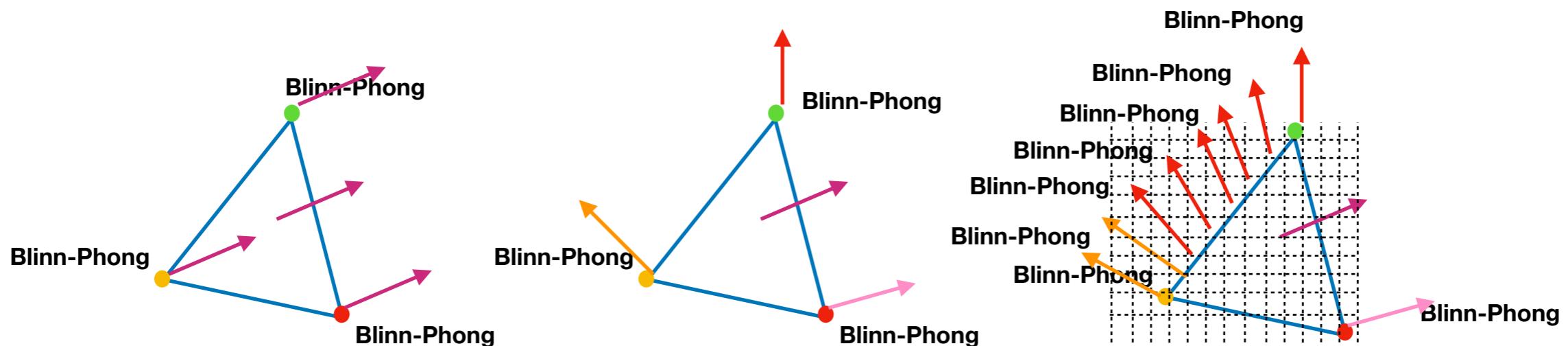
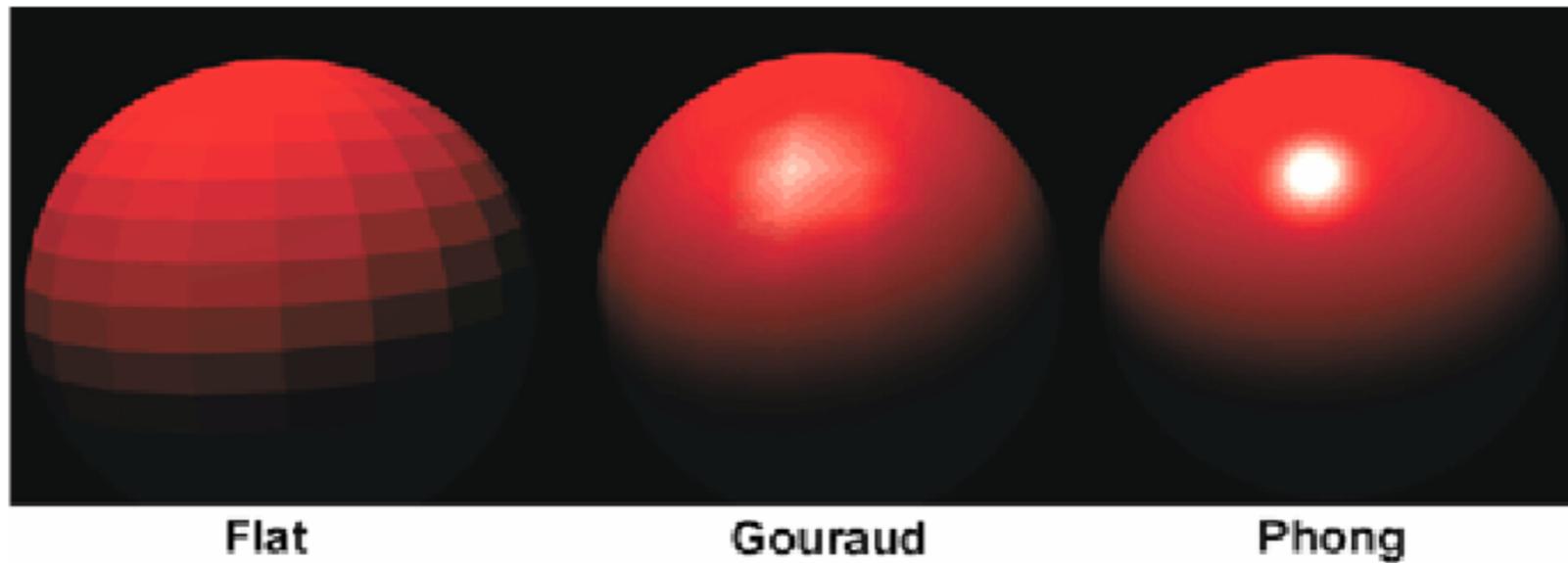
Fragment Shader



colorOut

# 4. Shading a la GPU

$$I_{total} = I_{a_{global}}K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d \max(L_i \cdot N, 0.0) + I_{s_i} K_s \max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$



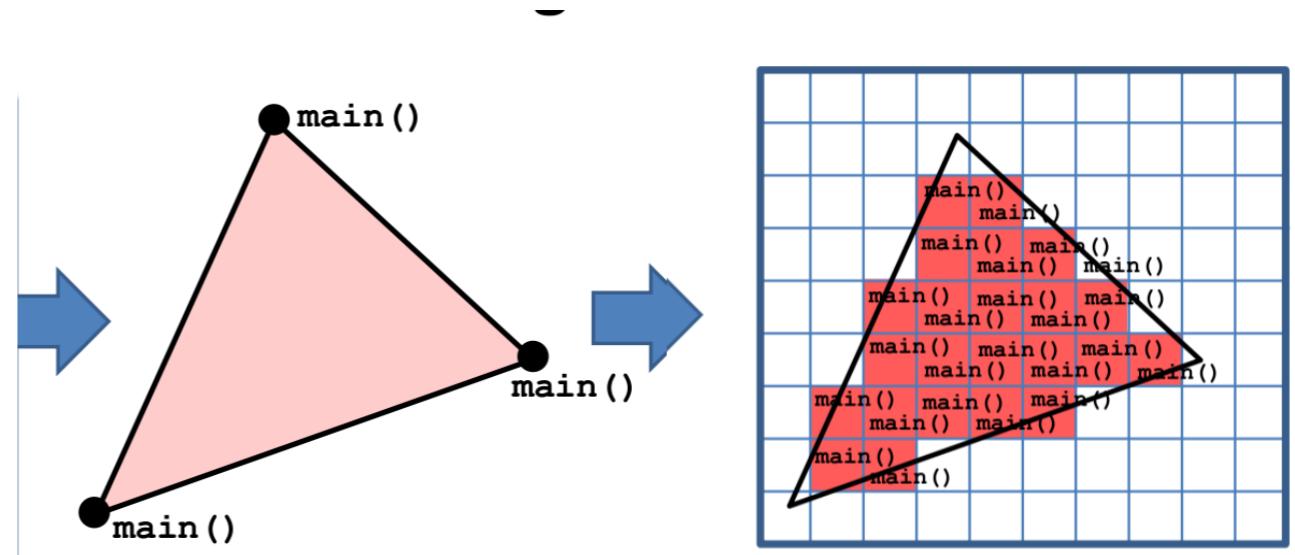
## 3.4. Il·luminació usant *shaders*

Per a calcular Blinn-Phong i el shading corresponent:

- Quines dades són uniform (comuns a tots els vèrtexs)?
- Quines dades són IN/OUT?
- Càlculs en el Vertex shader o en el Fragment shader?
- Quan es passen a la GPU?

$$I_{total} = I_{a_{global}}K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d \max(L_i \cdot N, 0.0) + I_{s_i} K_s \max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$

`glDrawArrays( GL_TRIANGLES, 0, Index );`



# 4. ACTIVITAT CPU-GPU

## INSTRUCCIONS

A cada fila es té un objecte que cal calcular. Indica on es realitzarà el càlcul (a la CPU, a la GPU, i en el cas que sigui a la GPU, indica si és el el vertex o en el fragment shader)

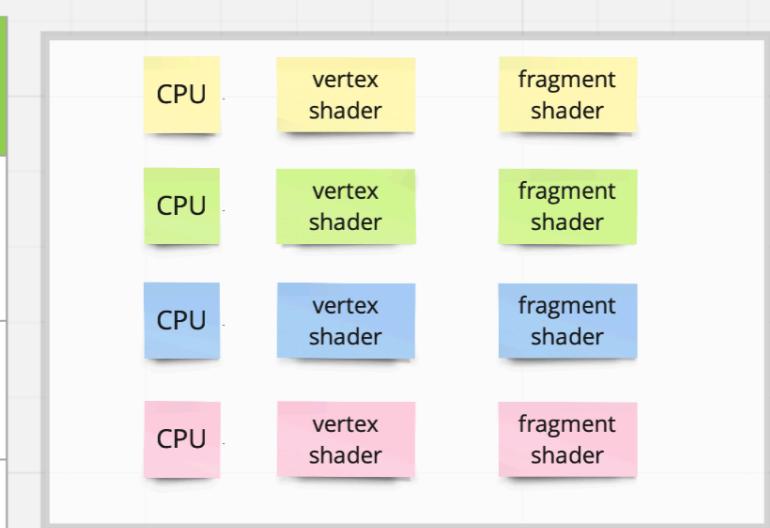
1. Activitat Individual: Tria un color que et representi. Usa les Sticky Notes d'aquell color per a omplir les caselles buides
2. Usa una Sticky Note per casella.
3. Usa els emojis per indicar el grau de seguretat que tens a la resposta.

## Activitat 2

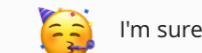
Activitat 2: Pensa on realitzar les següents operacions

Càlculs	On es calculen??
TGs	
Blinn-Phong: Flat shading	
Blinn-Phong- Gouraud shading	
Blinn-Phong - Phong Shading	
Coordenades de textura (u, v)	

Sticky Notes



Sticky Notes



I have some doubts



[https://miro.com/app/board/o9J\\_IH Aw t34=?share\\_link\\_id=289816909222](https://miro.com/app/board/o9J_IH Aw t34=?share_link_id=289816909222)

Activitat 2: Per quan hagis revisat el  
Shading Poligonal (video 24)

