

Sessió Setmana 9

GiVD 2022-2023

Guió de la sessió

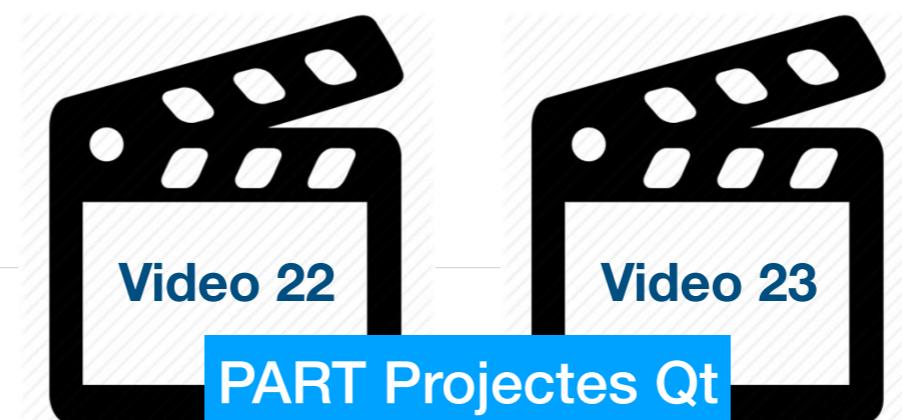
1. Planificació de la setmana 9
 2. Introducció ZBuffer i GL
 3. Introducció als projectes CubGL i CubGPU
-



1. Planificació de la setmana

2023 April

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
10	11 L8: Dubtes sobre instal.lacions o cubTexturesGL	12	13	14	15	16
17 Tema 3: Intro de nou Zbuffer. Arquitectures i zbuffer	18 L9: CubGL, cubGPU i CubGPUMTextures	19	 Video 19	 Video 20	 Video 21	23
				PART TEÓRICA		



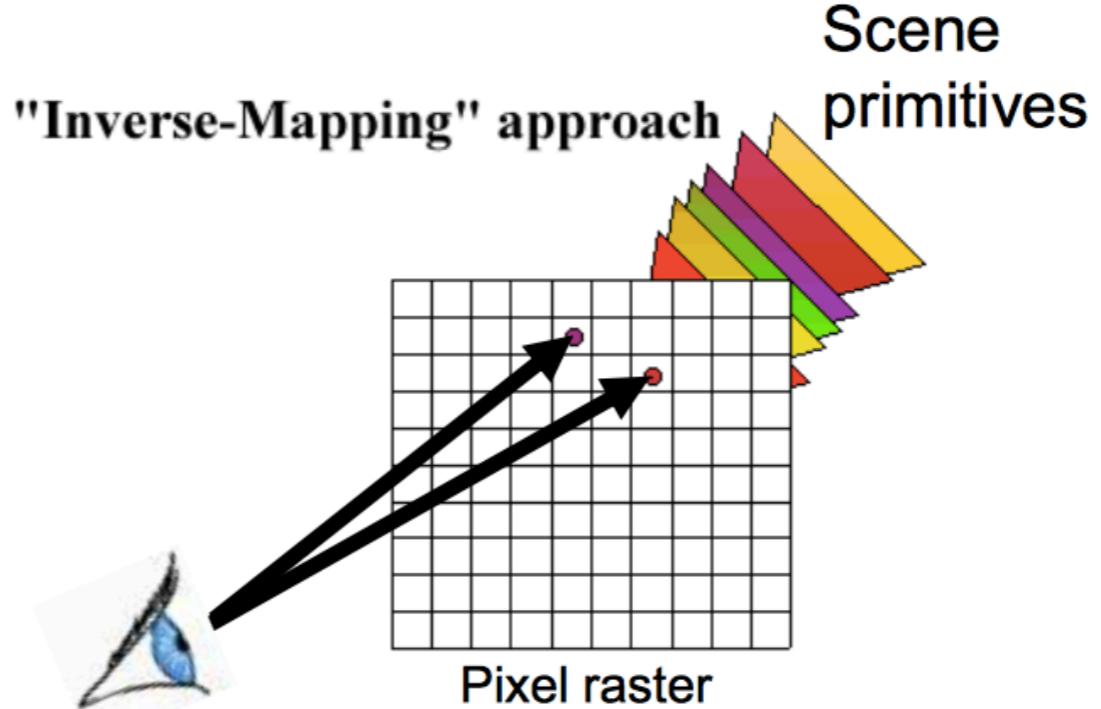
2. Introducció a ZBuffer

Suposem que anem a visualitzar
Models superficials: malles
poligonals



RayTracing

Per a cada **pixel** (raig)
Per a cada **triangle**
Intersecció raig-triangle?
Trobar la intersecció mes
propera



2. Introducció a ZBuffer

- RayTracing

Per a cada pixel (raig)

Per a cada triangle

Intersecta **raig-triangle**?

Trobar la intersecció més
propera

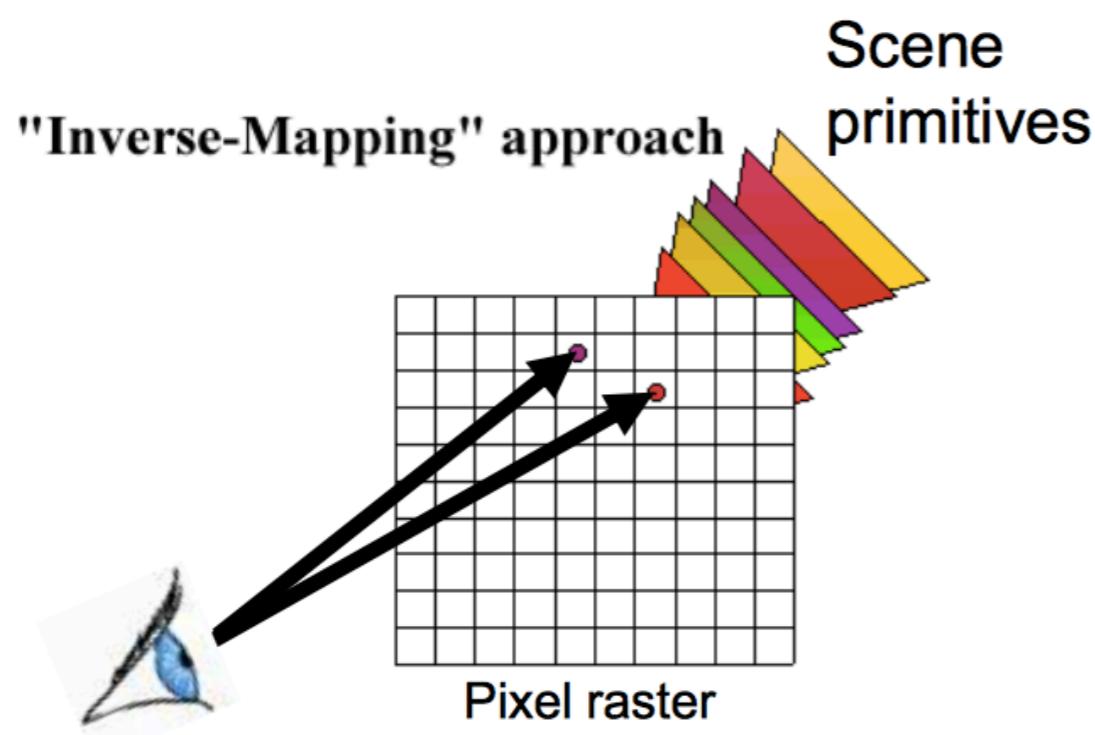
- Projectius

Per a cada triangle

Per a cada pixel

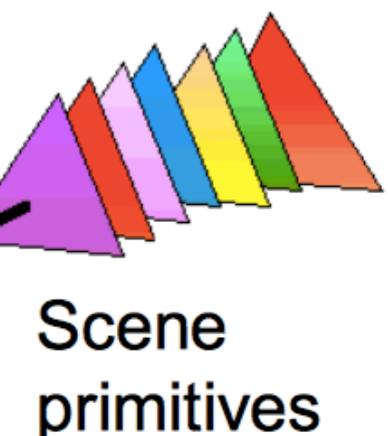
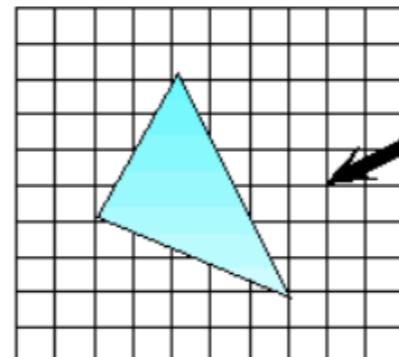
El **triangle** cubreix el píxel?

Trobar la projecció més
propera



"Forward-Mapping" approach

Pixel raster



2. Introducció a ZBuffer

- RayTracing
- Projectius

Per a cada pixel (raig)

Per a cada triangle

Intersecta **raig-triangle**?

Trobar la intersecció més
propera

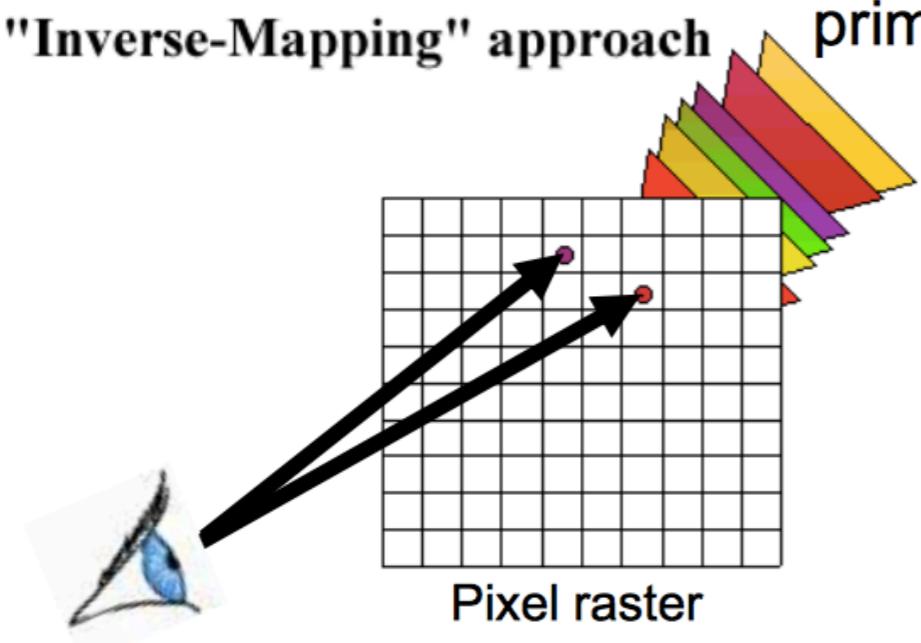
Per a cada triangle

Per a cada pixel

El **triangle** cubreix el píxel?

Trobar la projecció més
propera

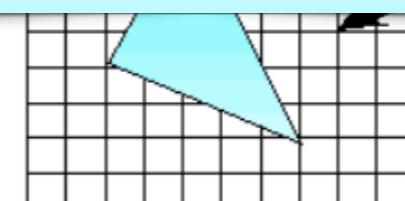
"Inverse-Mapping" approach



1. Situar els objectes segons la càmera

2. Trobar els píxels on es projecten

3. Càlcul de la visibilitat



Scene
primitives

Situar els objectes segons la càmera

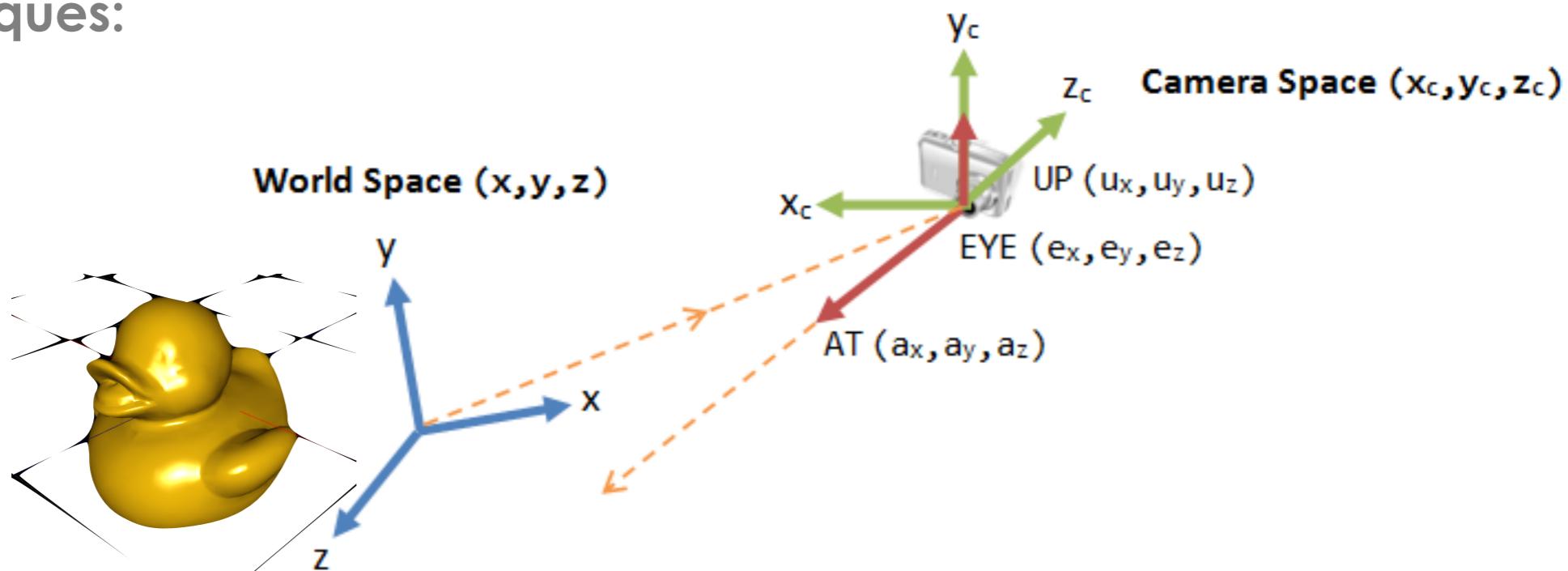
- RayTracing
- GPU (Zbuffer)

Per a cada **pixel** (raig)
Per a cada **triangle**
Intersecta **raig-triangle**?
Trobar la intersecció més
propera

Per a cada **triangle**
Per a cada **pixel**
El **triangle** cubreix el píxel?
Trobar la projecció més
propera

Característiques:

- Càmera



Situar els objectes segons la càmera

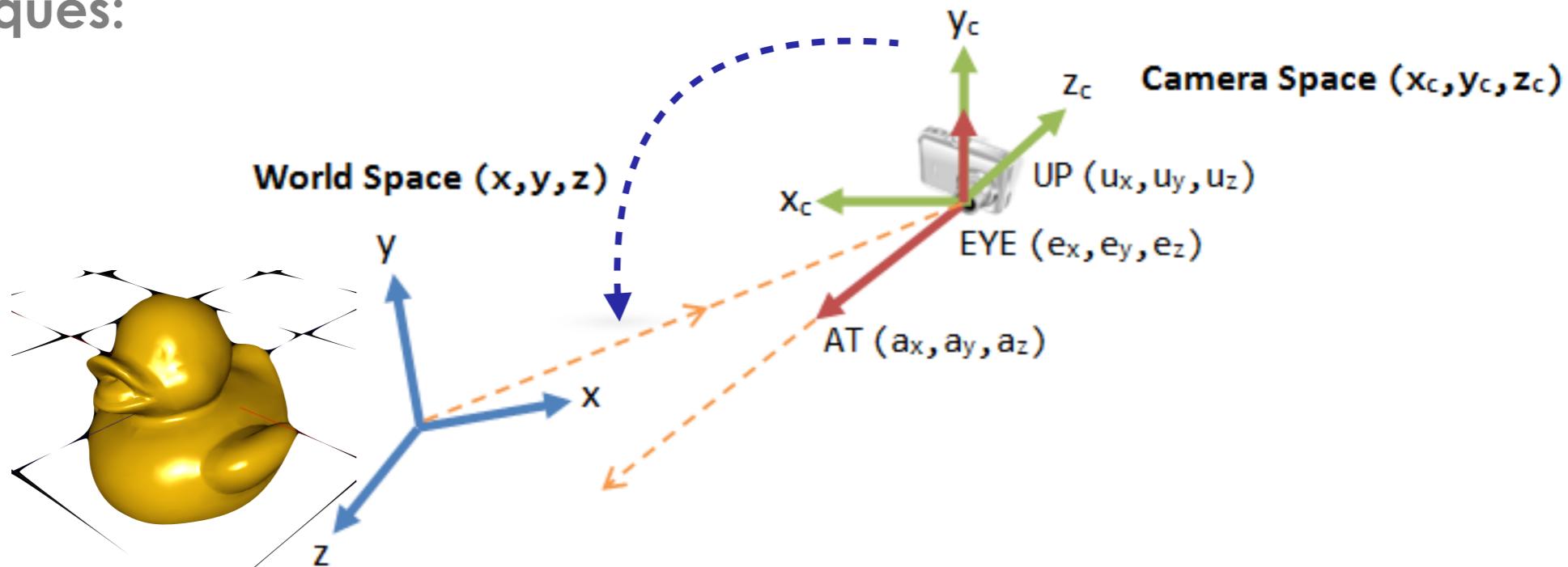
- RayTracing
- GPU (Zbuffer)

Per a cada **pixel** (raig)
Per a cada **triangle**
Intersecta **raig-triangle**?
Trobar la intersecció més
propera

Per a cada **triangle**
Per a cada **pixel**
El **triangle** cubreix el píxel?
Trobar la projecció més
propera

Característiques:

- Càmera



Situar els objectes segons la càmera

- RayTracing
- GPU (Zbuffer)

Per a cada **pixel** (raig)

Per a cada **triangle**

Intersecta **raig-triangle**?

Trobar la intersecció més
propera

Per a cada **triangle**

Per a cada **pixel**

El **triangle** cubreix el píxel?

Trobar la projecció més
propera

Característiques:

- Càmera:
A RayTracing:

Coordenades
Viewport

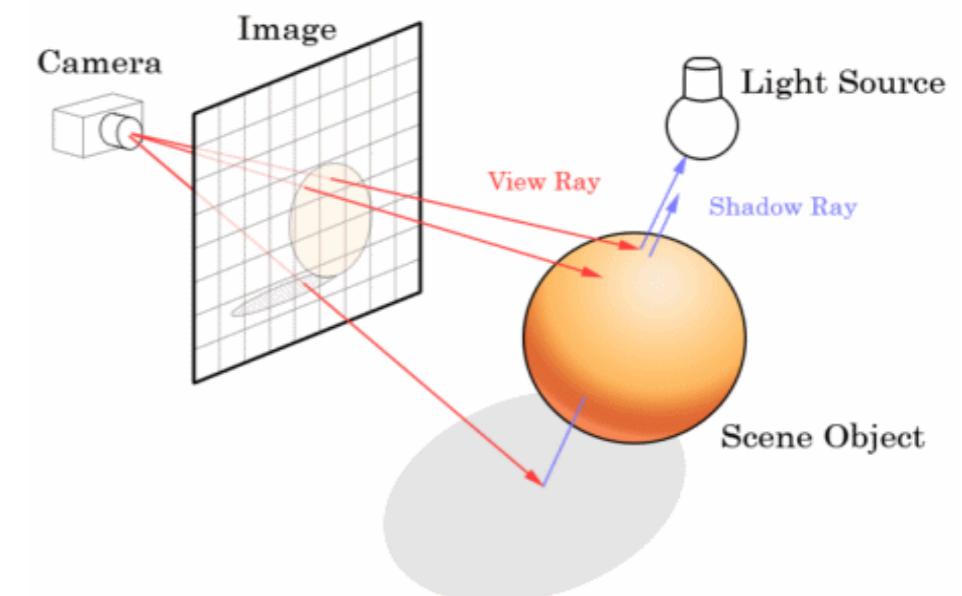
Píxel

Coordenades
Càmera

(u, v, w)

Coordenades
Món

Rraig: $p_0(x, y, z), v$



Situar els objectes segons la càmera

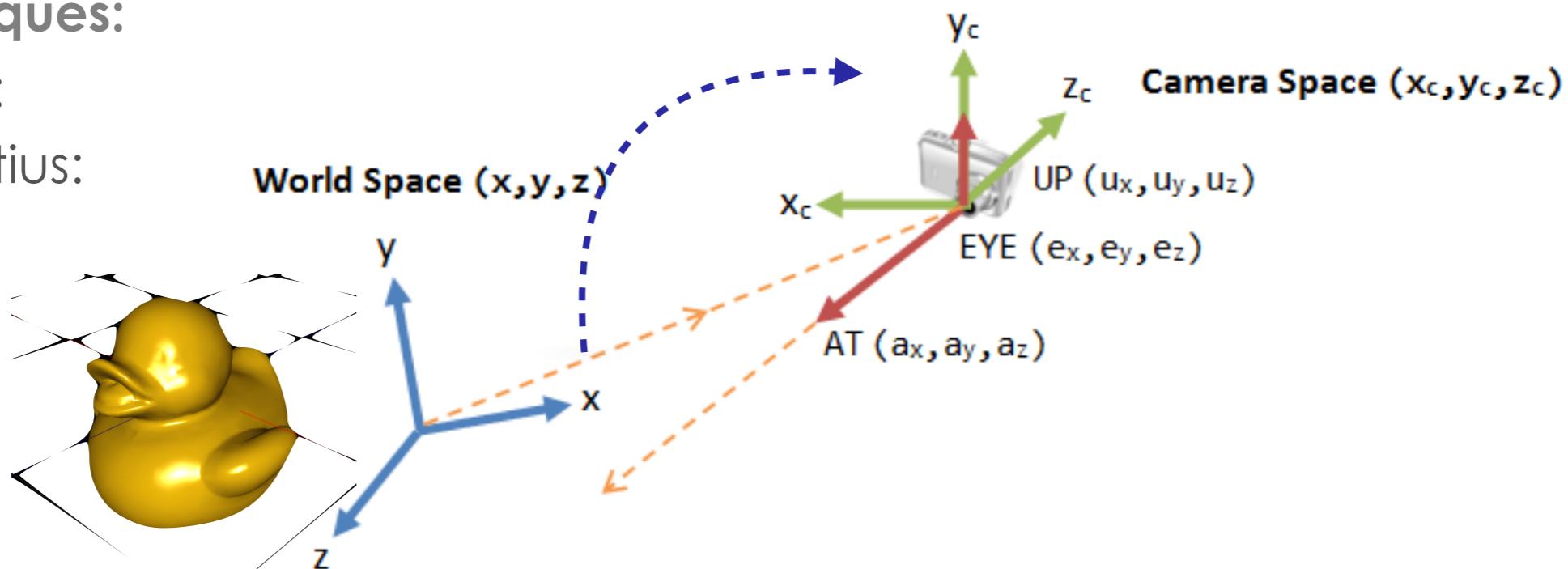
- RayTracing
- GPU (Zbuffer)

Per a cada **pixel** (raig)
Per a cada **triangle**
Intersecta **raig-triangle**?
Trobar la intersecció més
propera

Per a cada **triangle**
Per a cada **pixel**
El **triangle** cubreix el píxel?
Trobar la projecció més
propera

Característiques:

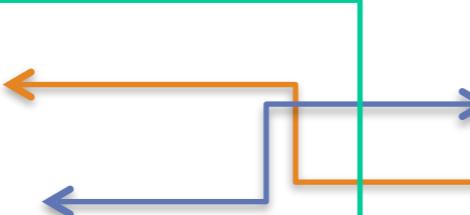
- Càmera:
A Projectius:



Situar els objectes segons la càmera

- RayTracing
- GPU (Zbuffer)

Per a cada **pixel** (raig)
Per a cada **triangle**
Intersecta **raig-triangle**?
Trobar la intersecció més
propera



Per a cada **triangle**
Per a cada **pixel**
El **triangle** cubreix el píxel?
Trobar la projecció més
propera

Característiques:

- Càmera:
A Projectius:



Triangle (p_1, p_2, p_3)
formats per (x, y, z)

(u, v, w)

Píxel

Situar els objectes segons la càmera

- RayTracing

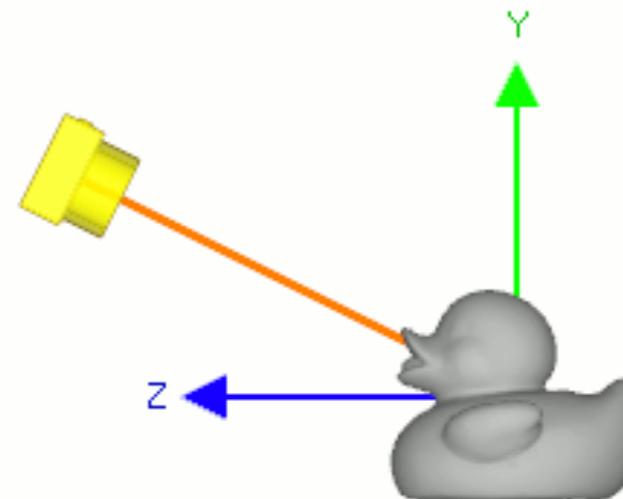
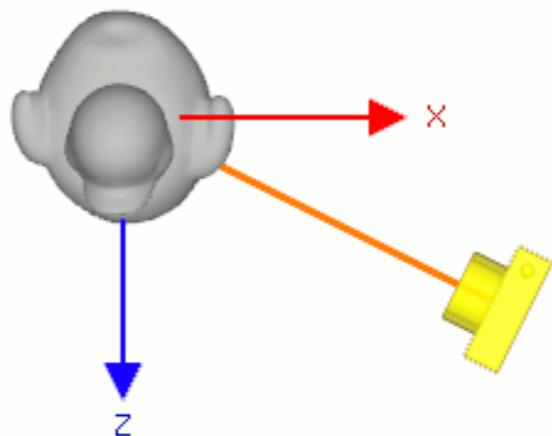
Per a cada **pixel** (raig)
Per a cada **triangle**
Intersecta **raig-triangle**?
Trobar la intersecció més
propera

- GPU (Zbuffer)

Per a cada **triangle**
Per a cada **pixel**
El **triangle** cubreix el píxel?
Trobar la projecció més
propera

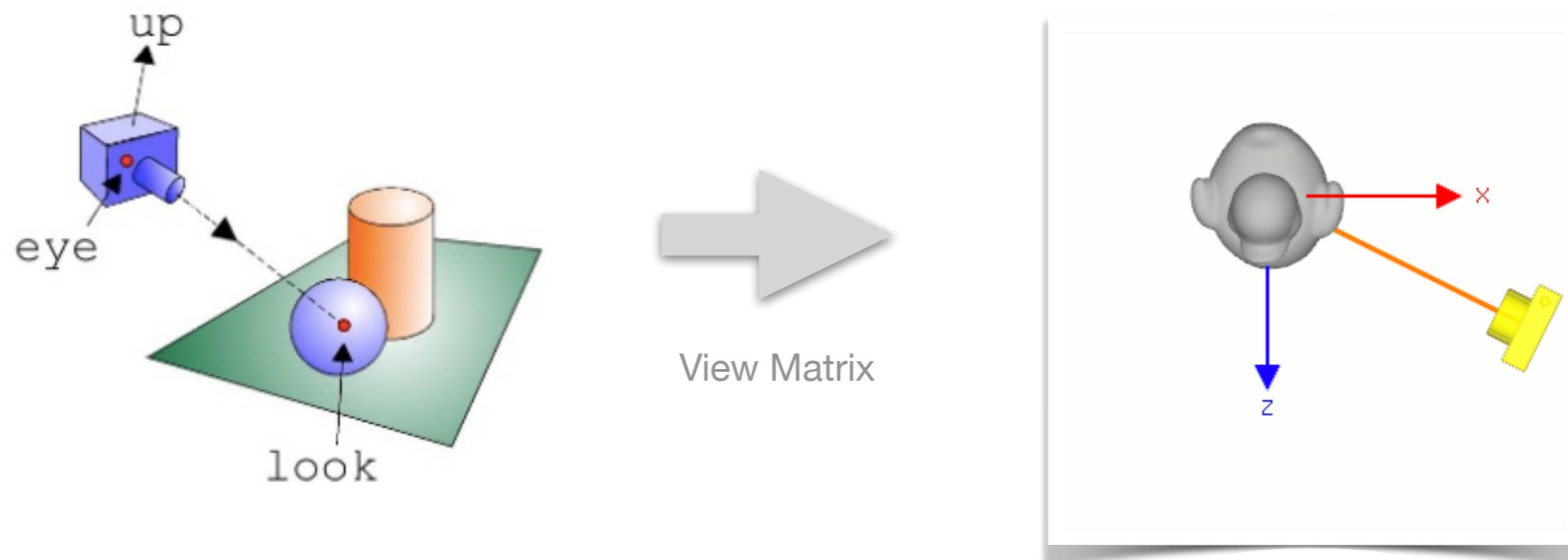
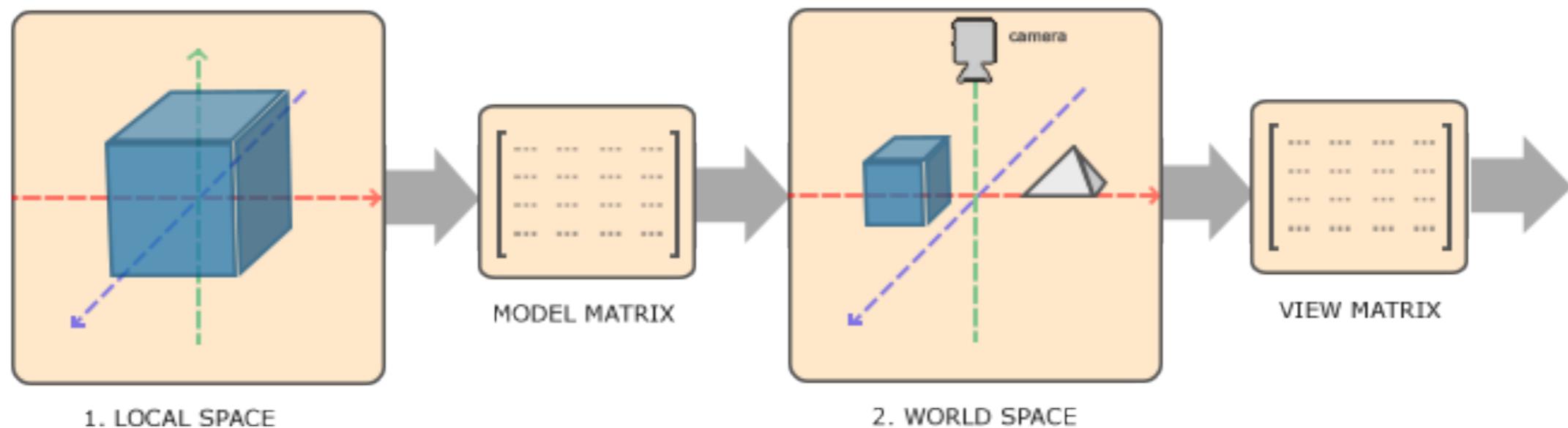
Característiques:

- Càmera
A Projectius

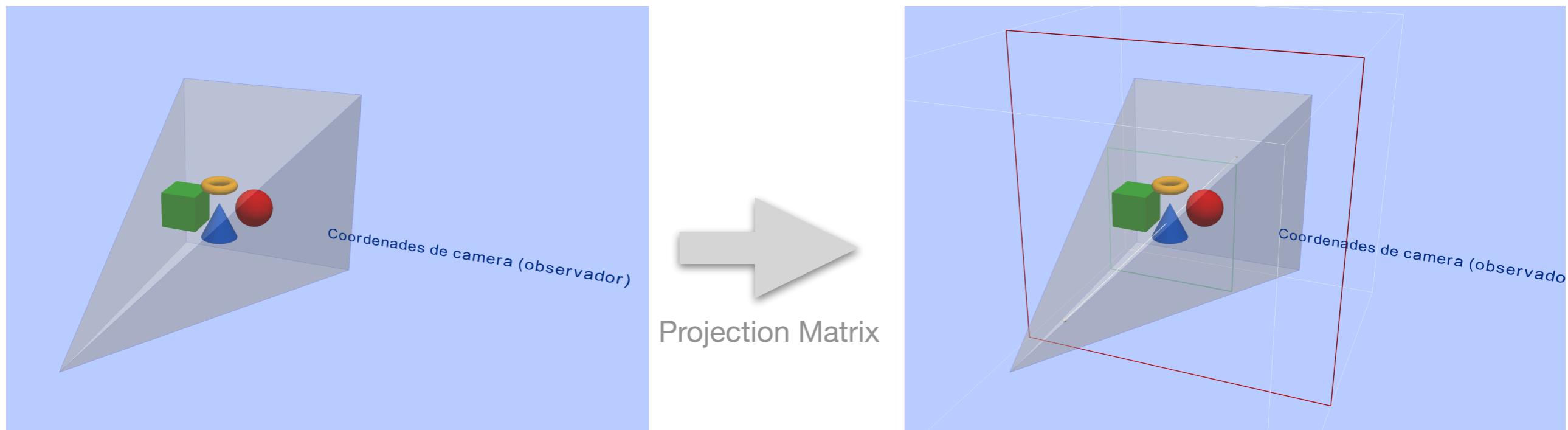
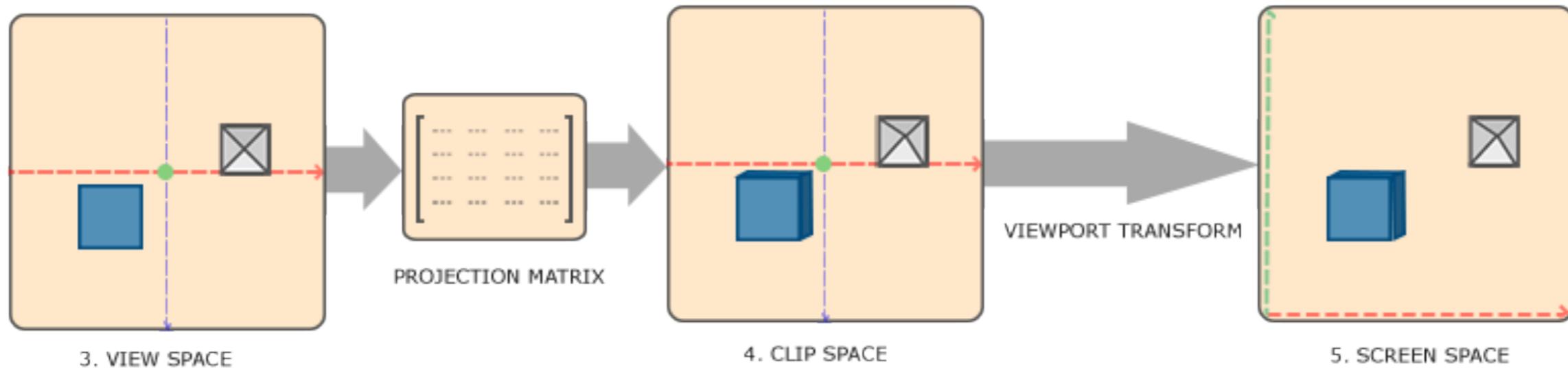


matriu modelView

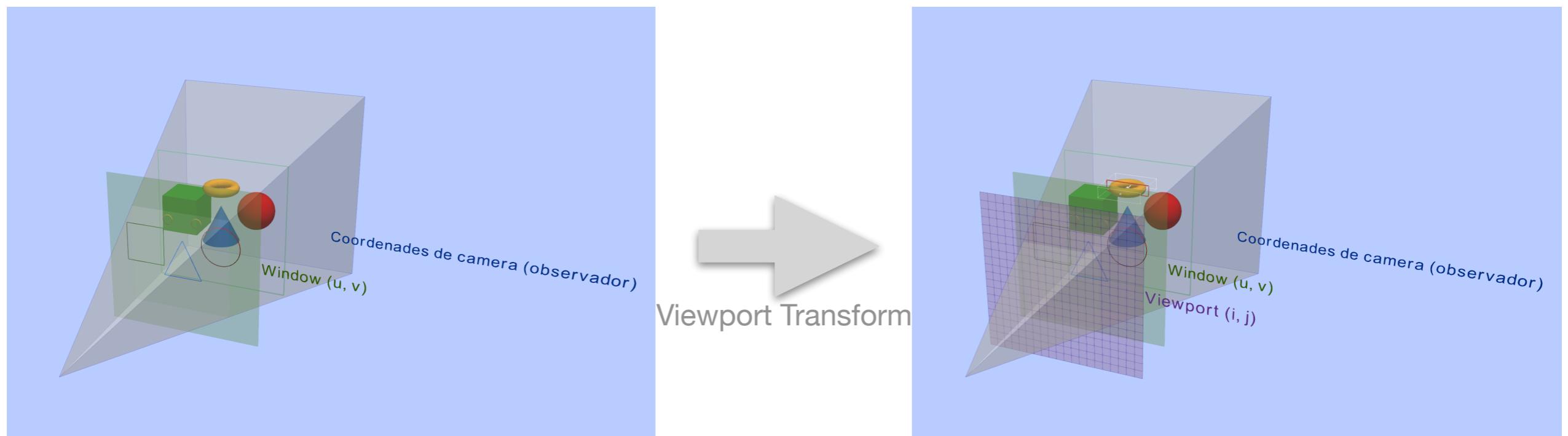
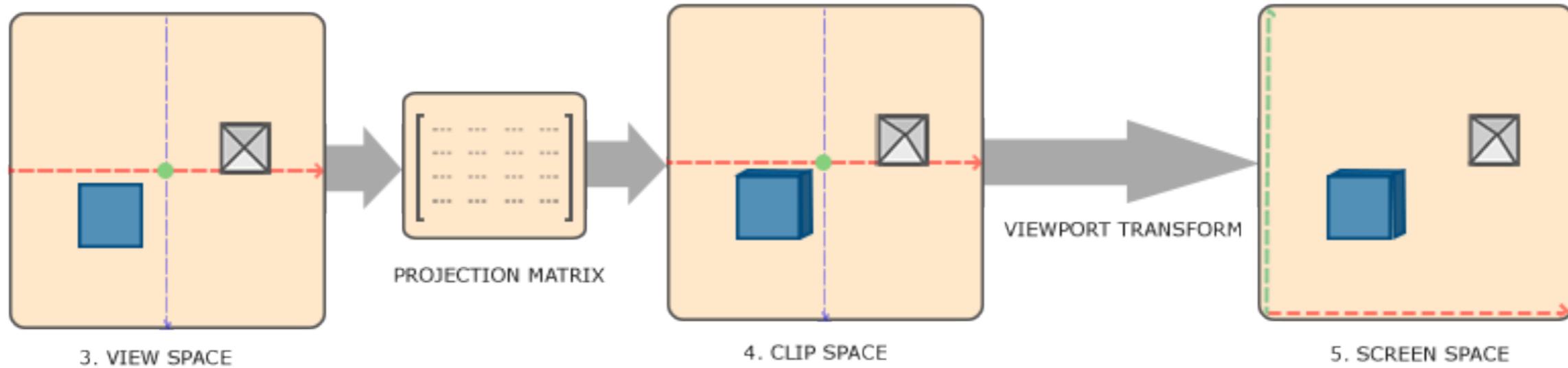
Situar els objectes segons la càmera



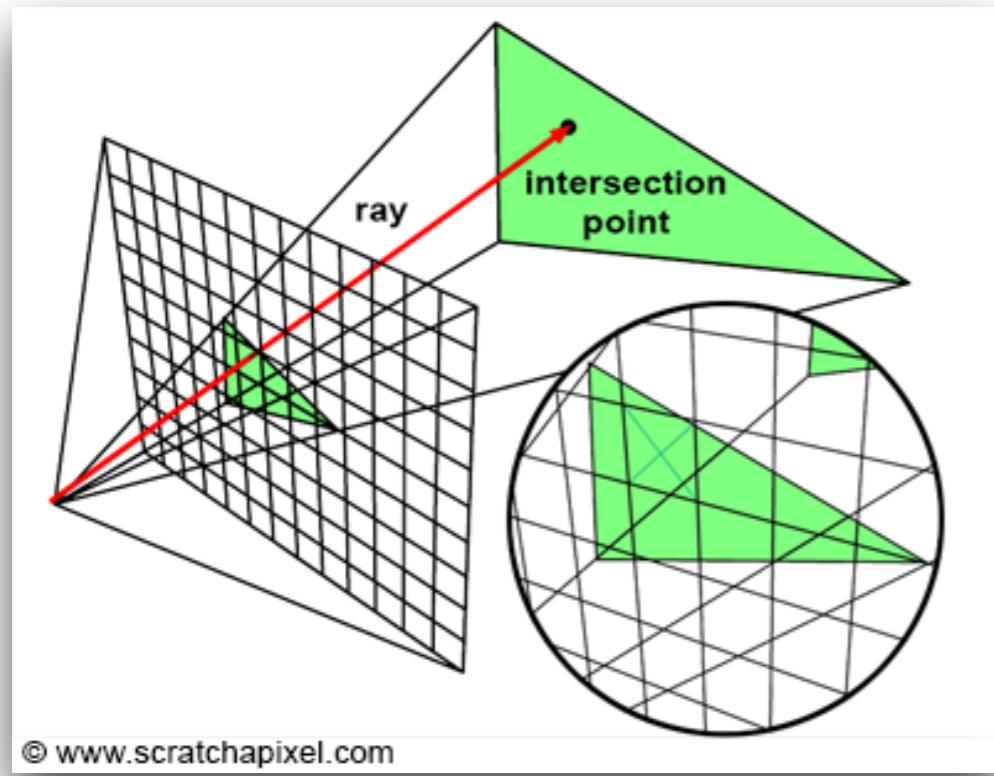
Càcul dels píxels de la projecció



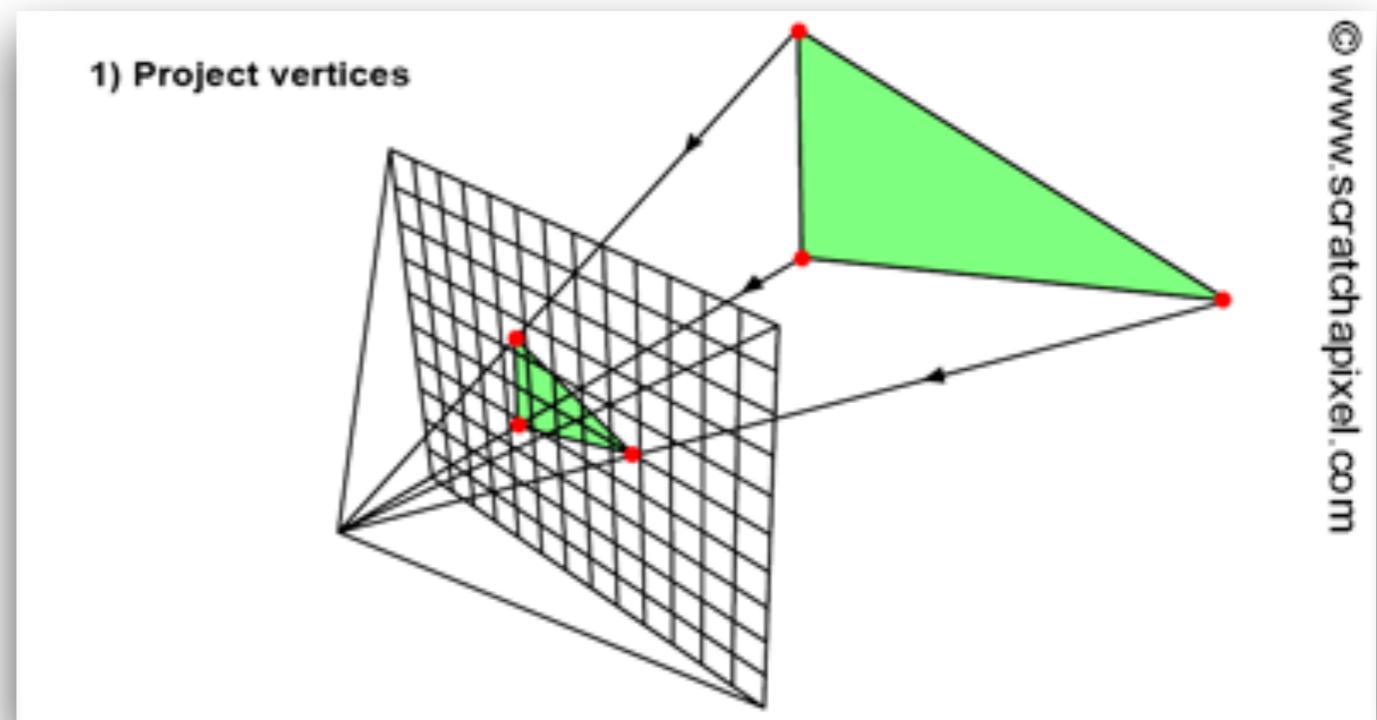
Càcul dels píxels de la projecció



Càlcul dels píxels de la projecció



Raytracing



ZBuffer

Càlcul de la visibilitat

- **RayTracing**

Per a cada **pixel** (raig)

Per a cada **triangle**

Intersecció raig-triangle?

Trobar la intersecció més
propera

- Projectius

Per a cada **triangle**

Per a cada **pixel**

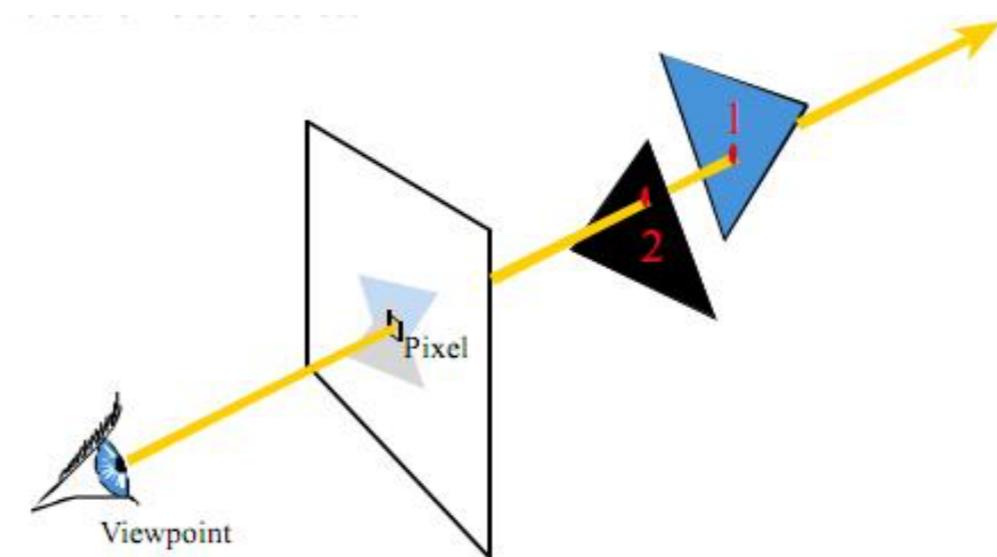
El triangle cubreix el píxel?

Trobar la projecció més
propera

Característiques:

- Visibilitat:

Raytracing: intersecció raig primari i tots els objectes de l'escena, agafant la t minima



Càlcul de la visibilitat

- RayTracing

Per a cada **pixel** (raig)

Per a cada **triangle**

Intersecta raig-triangle?

Trobar la intersecció més
propera

- Projectius

Per a cada **triangle**

Per a cada **pixel**

El triangle cubreix el píxel?

Trobar la projecció més
propera

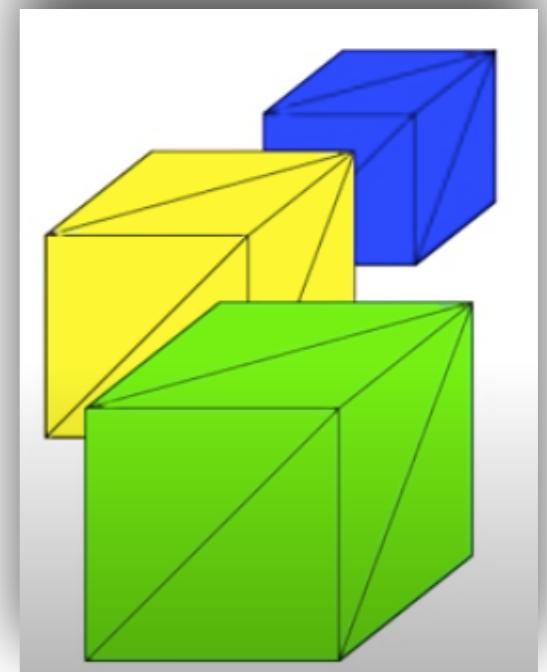
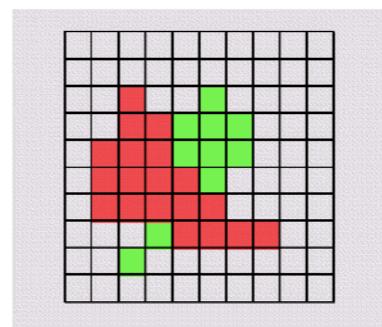
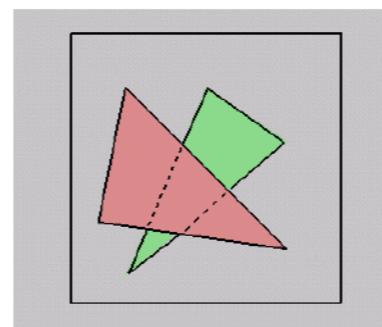
Característiques:

- Visibilitat:

Projectius: l'objectiu és acabar pintant el triangle
més proper a l'observador.

Dos estratègies:

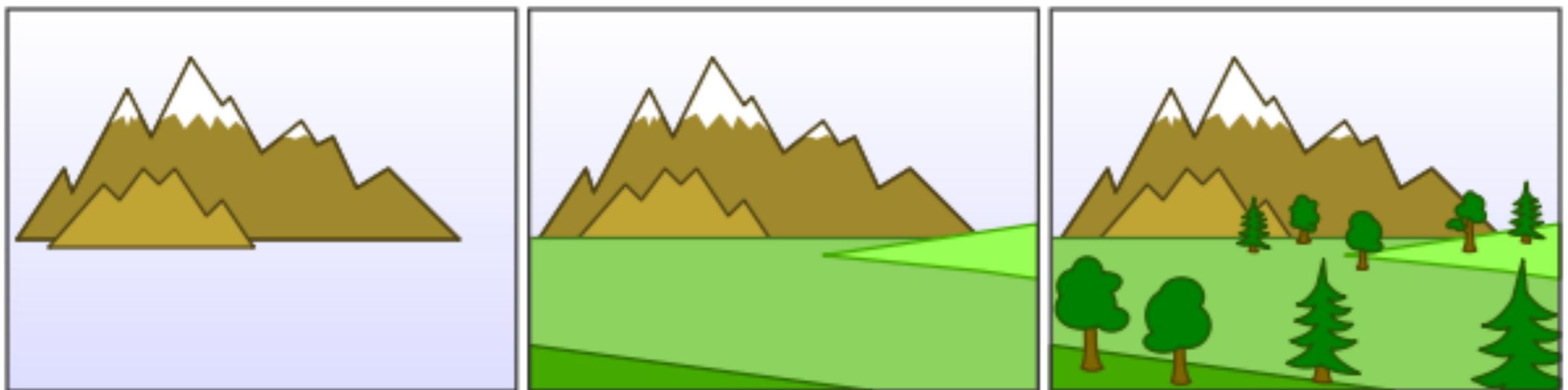
- algorisme del **Pintor**
- algorisme de **ZBuffer**



Càlcul de la visibilitat

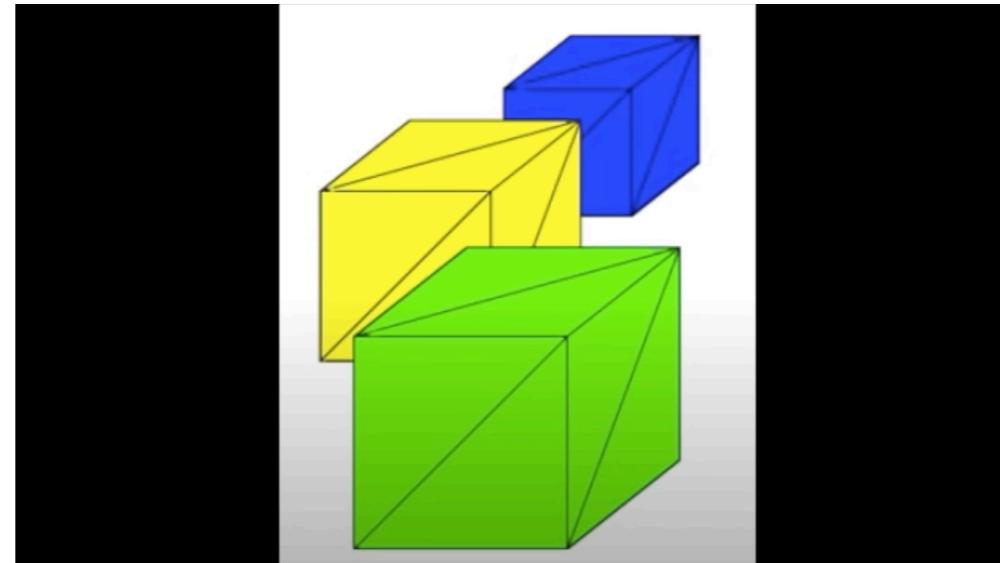
Algorisme del Pintor (Painter's Algorithm)

- Inspirat de com pinten els pintors
- S'ordenen els objectes o triangles de z's més llunyanes a z's més properes
- Es projecten els triangles en aquest ordre

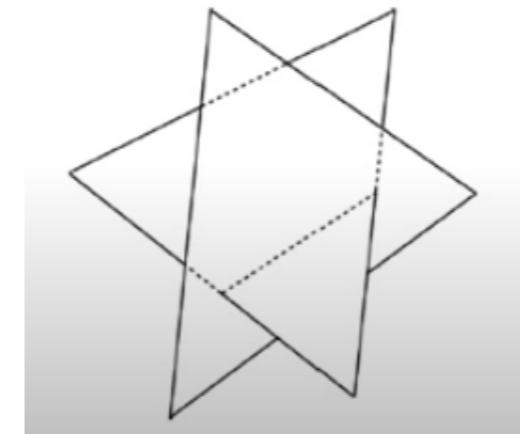
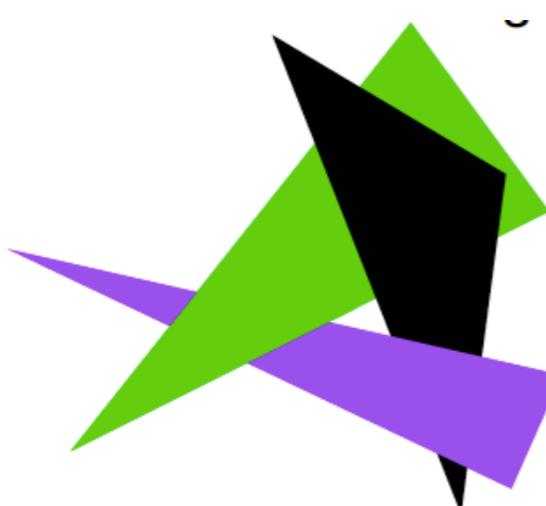


Càlcul de la visibilitat

Algorisme del Pintor (Painter's Algorithm)



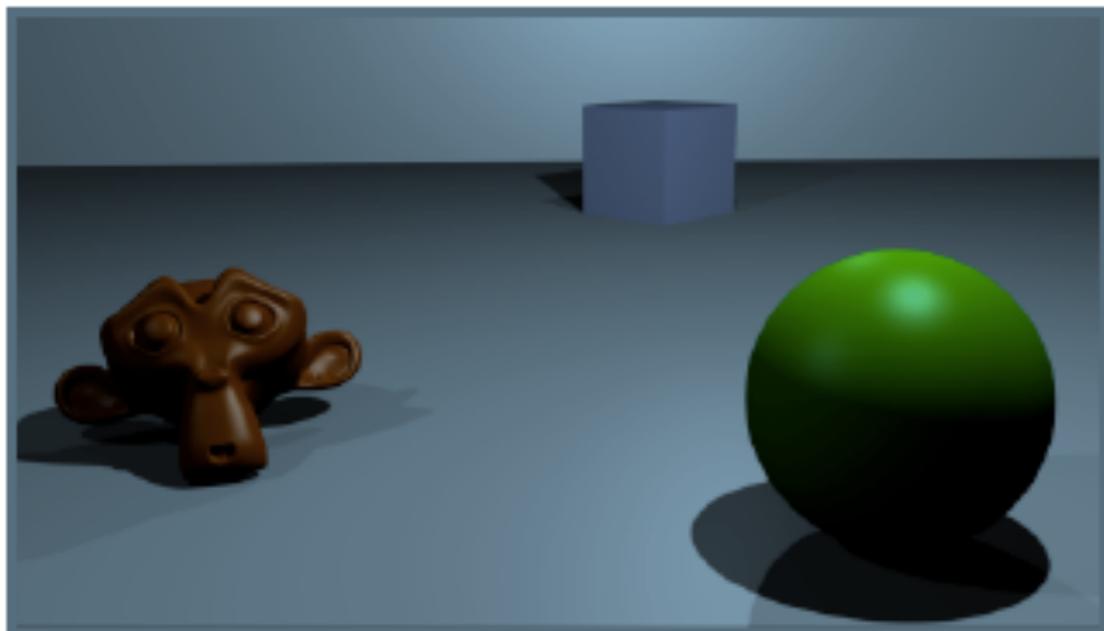
Problemes: Depèn de com estiguin situats els triangles, aquest algorisme no funciona bé



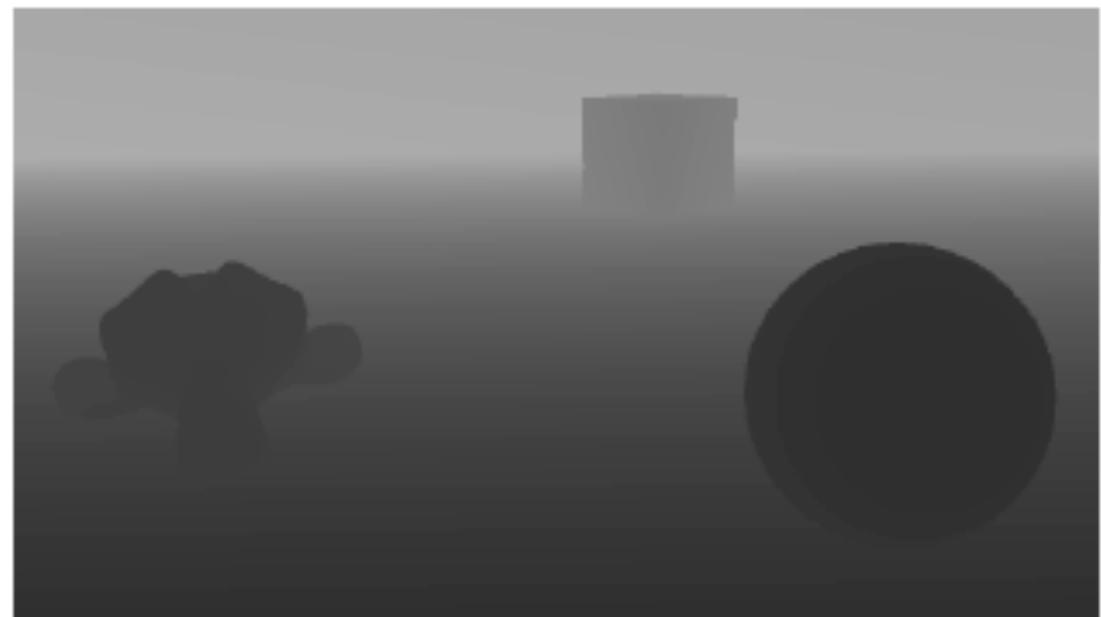
Càlcul de la visibilitat

Z-Buffer

- Aplicat en coordenades de càmera
- Les z's augmenten a mesura que s'allunyen de l'observador



Escena



Depth-Buffer o z-buffer
(fosc: proper, clar: lluny)

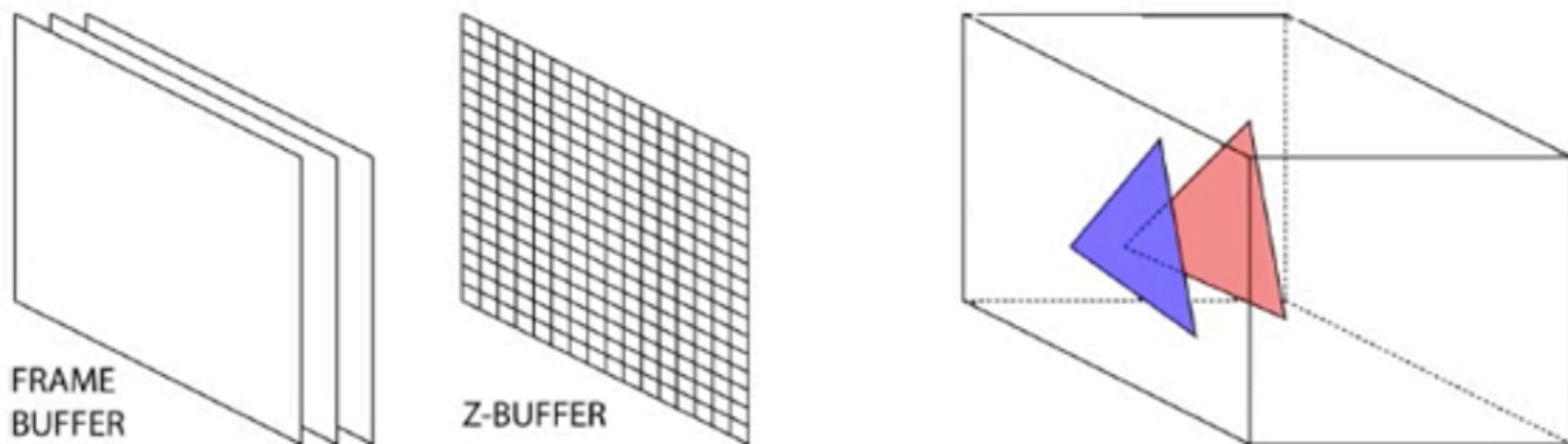
Càlcul de la visibilitat

Z-Buffer

- Aplicat en coordenades de càmera
- Les z's augmenten a mesura que s'allunyen de l'observador

Estructures:

- Frame Buffer (on es guarden els colors finals)
- Z-Buffer (on es guarden les profunditats)



Càlcul de la visibilitat

Z-Buffer

Inicialitzar el **frame buffer** (colors) al color de background

Inicialitzar el **depth buffer** (depth) a profunditats més allunyades

Per a cada triangle t

Per a cada píxel (i, j) de la rasterització del triangle t

$z = \text{calculProfunditat}(i, j)$

$c = \text{calculColorTriangle}(i, j)$

si $z < \text{depth}(i, j)$ llavors

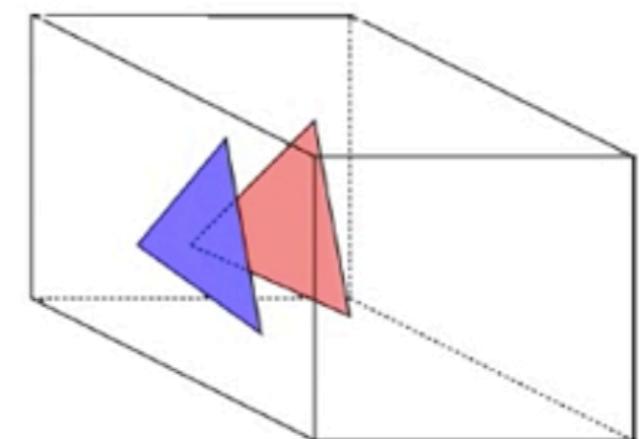
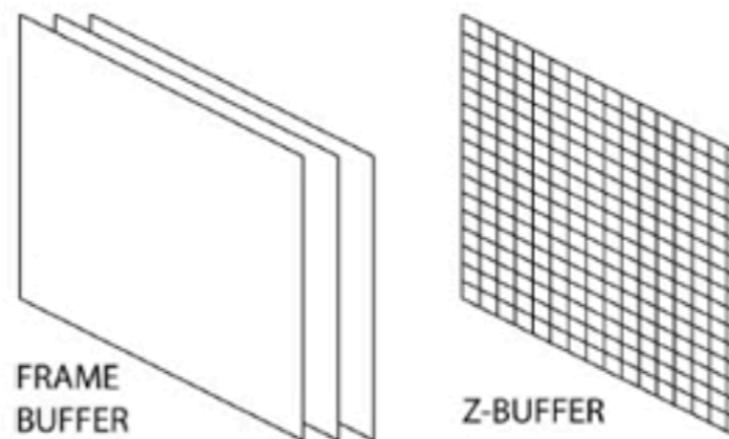
$\text{depth}(i, j) = z$

$\text{color}(i, j) = c$

fsi

fper

fper



Càlcul de la visibilitat

Z-Buffer

Iniciar el frame buffer (**colors**) al color de background

Iniciar el depth buffer (**depth**) a profunditats més allunyades

Per a cada triangle t

Per a cada píxel (i, j) de la rasterització del triangle t

$z = \text{calculProfunditat}(i, j)$

$c = \text{calculColorTriangle}(i, j)$

si $z < \text{depth}(i, j)$ llavors

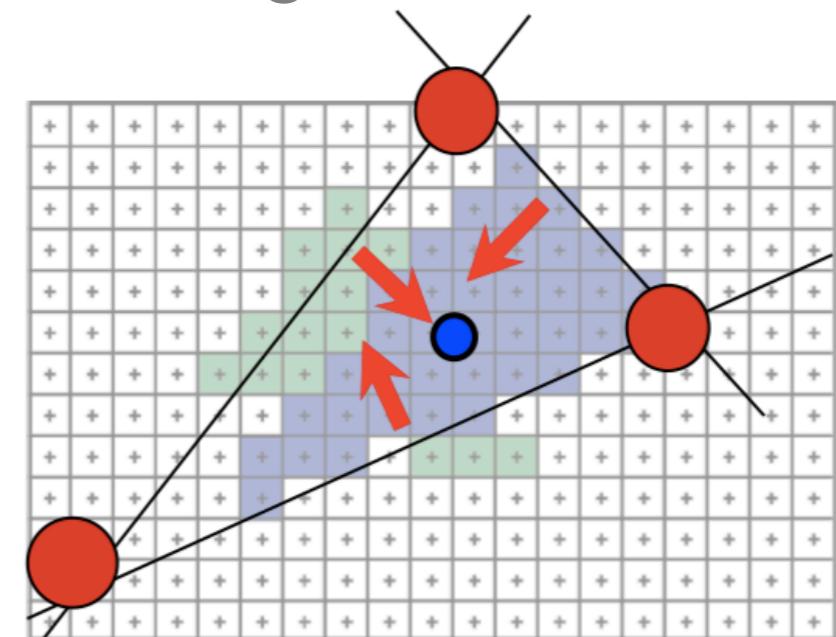
$\text{depth}(i, j) = z$

$\text{color}(i, j) = c$

fsi

fper

fper



depth dels píxels dels vèrtexs del triangle **coneguda**
depth dels píxels interns, **interpolada**

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
 $b = \{0, 1, \dots, B-1\}$

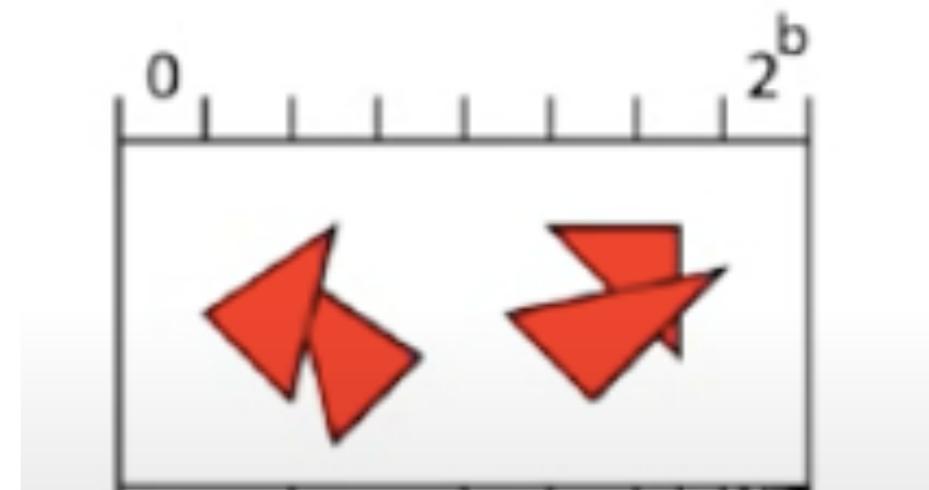
Els valors de z's en coordenades de càmera es mapegen a aquests valors, en intervals regulars de mida

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

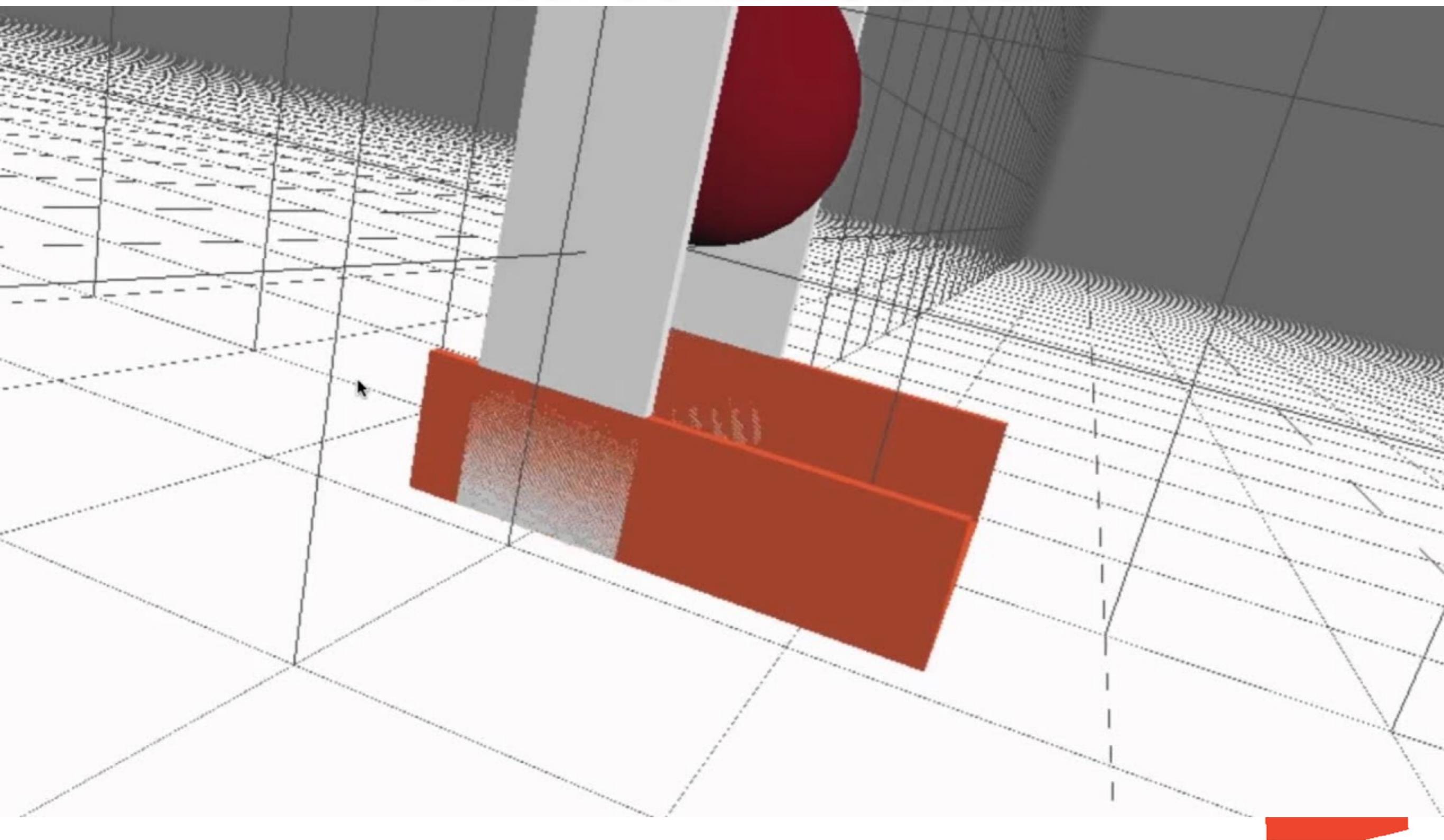
I un z d'un punt es calcula com:

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

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



Càlcul de la visibilitat

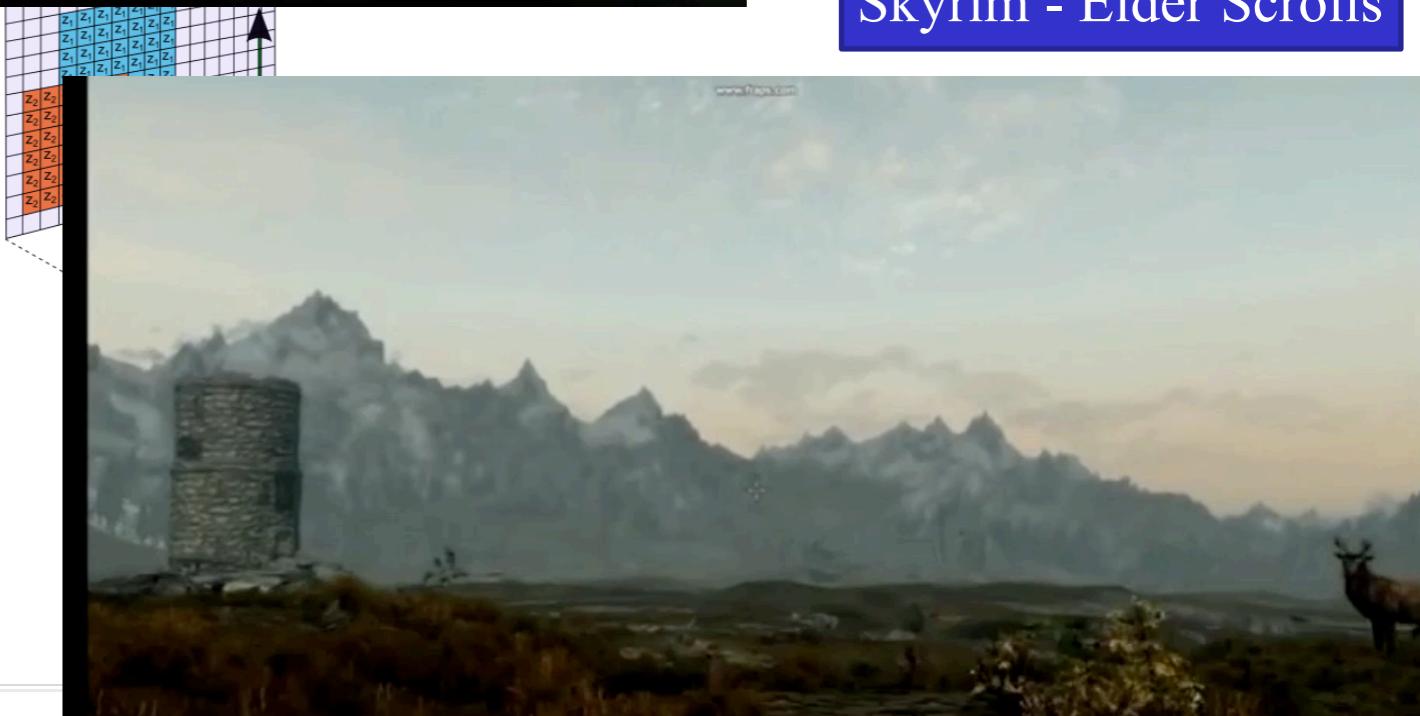


Càlcul de la visibilitat



Fallout - New Vegas

- Posant més bits de canal al ZBuffer
- Allunyant el z_{near}
- Apropant el z_{far}



Skyrim - Elder Scrolls

2. Introducció a ZBuffer

- RayTracing
- GPU (Zbuffer)

Per a cada pixel (raig)

Per a cada triangle

Intersecta raig-triangle?

Trobar la intersecció més
propera

Per a cada triangle

Per a cada pixel

El triangle cubreix el píxel?

Trobar la projecció més
propera

Ray-centric

Triangle-centric

Quina **memòria** es necessita a cada cas per a tenir la informació d'un píxel?

- **RayTracing**: Necessita tota l'escena en memòria en un cert instant

- **Zbuffer**: Necessita només un triangle a la vegada i una imatge amb les profunditats.

Una escena es sovint més complicada que una imatge.

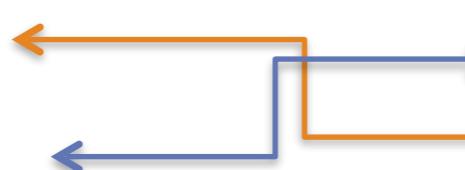
Una imatge de 1920x1080 píxels de colors codificats amb 16 bits per canal de color (RGBA) i un depth buffer 32-bits suposaria uns 24MB.

Si hi hagués més d'una mostra per píxel (4) serien aprox 100 MB

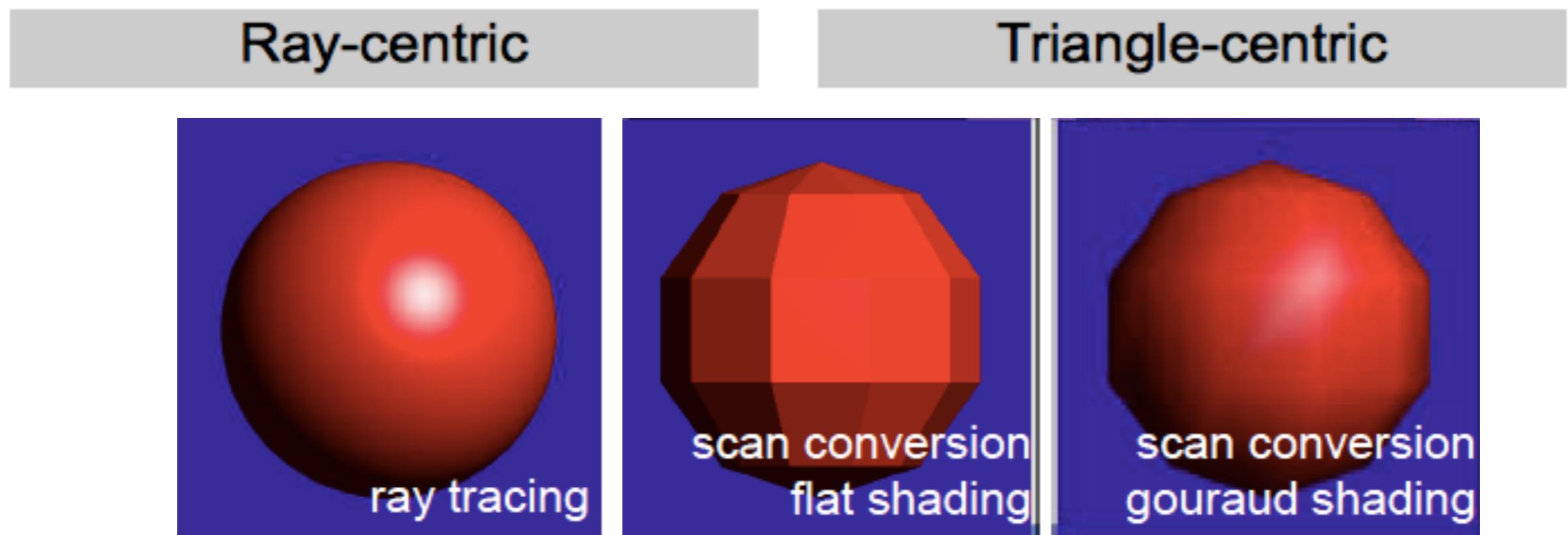
2. Introducció a ZBuffer

- RayTracing
- GPU (Zbuffer)

Per a cada pixel (raig)
Per a cada triangle
Intersecta raig-triangle?
Trobar la intersecció més propera



Per a cada triangle
Per a cada pixel
El triangle cubreix el píxel?
Trobar la projecció més propera



2. Introducció a ZBuffer

RayTracing

- Avantatges:

- **General:** es pot visualitzar tot objecte que es pot calcular la intersecció amb un raig
- Recursiu de forma que es poden calcular **ombres, reflexions, transparències**

- Desavantatges:

- Difícil d'implementar amb hardware. L'escena ha de cabre en memòria, dependències entre dades, etc.
- Lent per fer visualitzacions interactives sino es disposa d'una **gràfica molt especialitzada** (Nvidia Titan V amb arquitectura Turing GPU)

<https://devblogs.nvidia.com/introduction-nvidia-rtx-directx-raytracing/>

ZBuffer

- Avantatges:

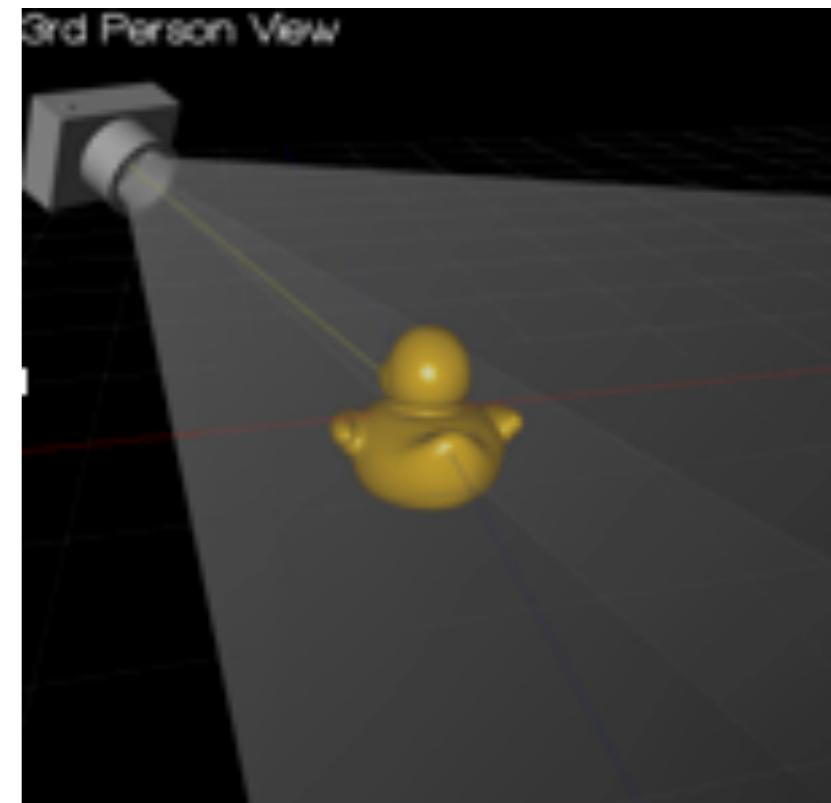
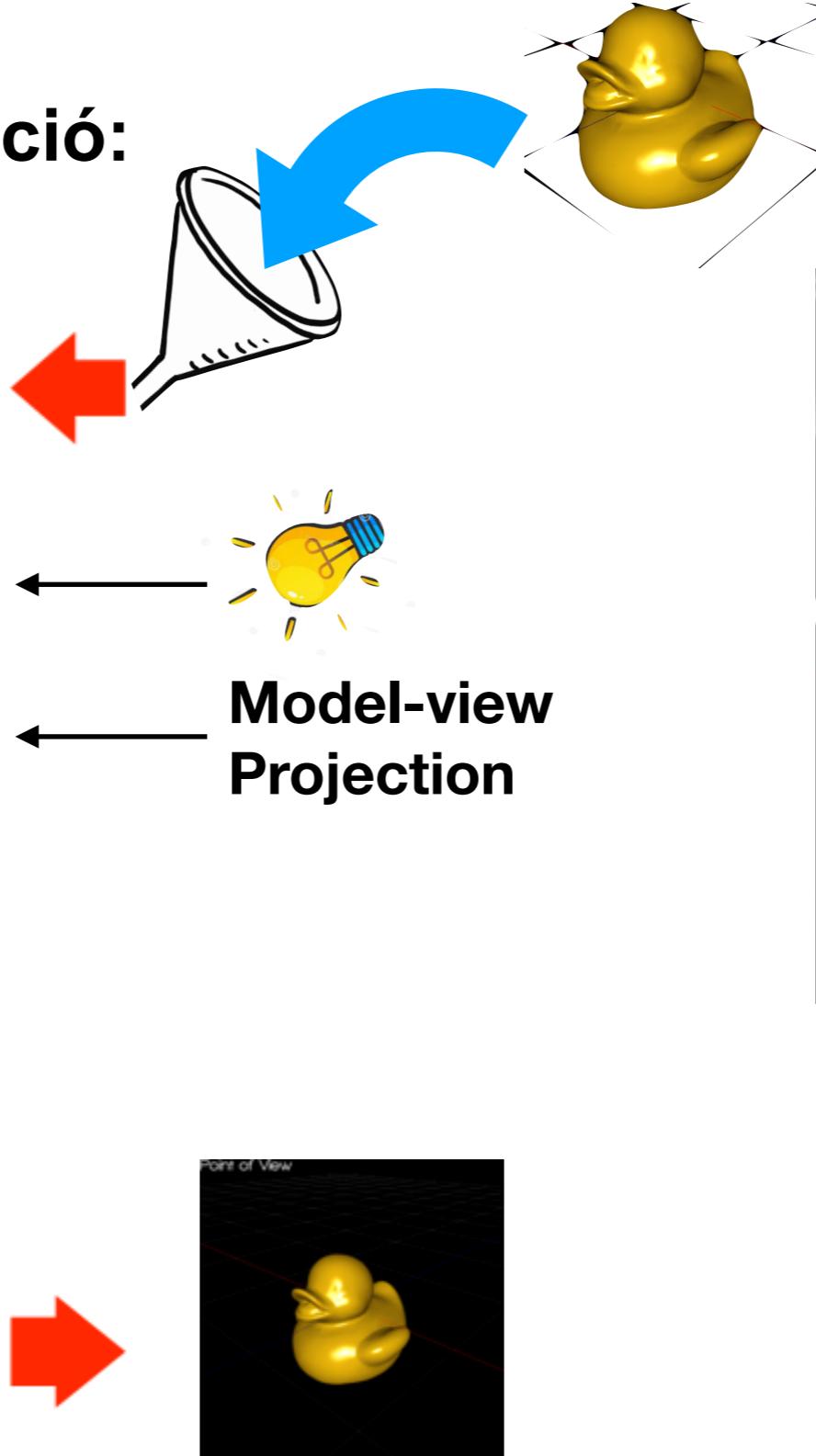
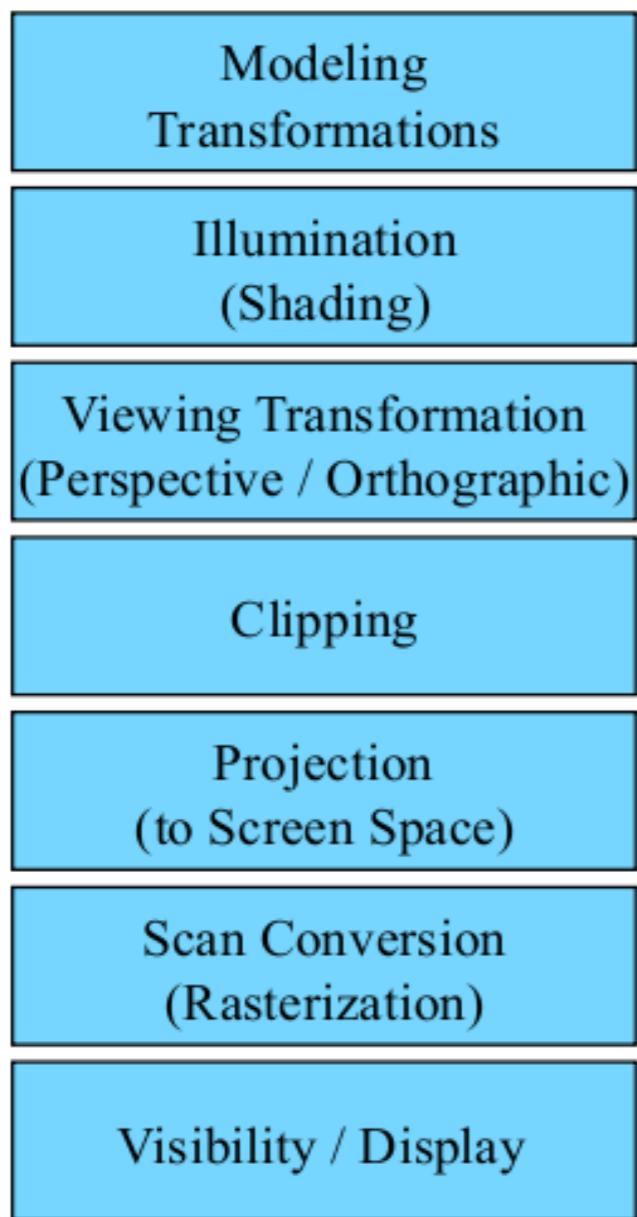
- Permet **parallelitzar** i fer streaming dels triangles en **targetes gràfiques asequibles**
- Els objectes **no** tenen perquè estar **ordenats** en profunditat
- No hi ha dependències entre dades.

- Desavantatges:

- **Restringit** a objectes formats per primitives que es puguin rasteritzar
- Artefactes de shading
- **Ombres/reflexions/transparències** no es tracten de forma directa
- Problema de redibuixat d'un mateix píxel

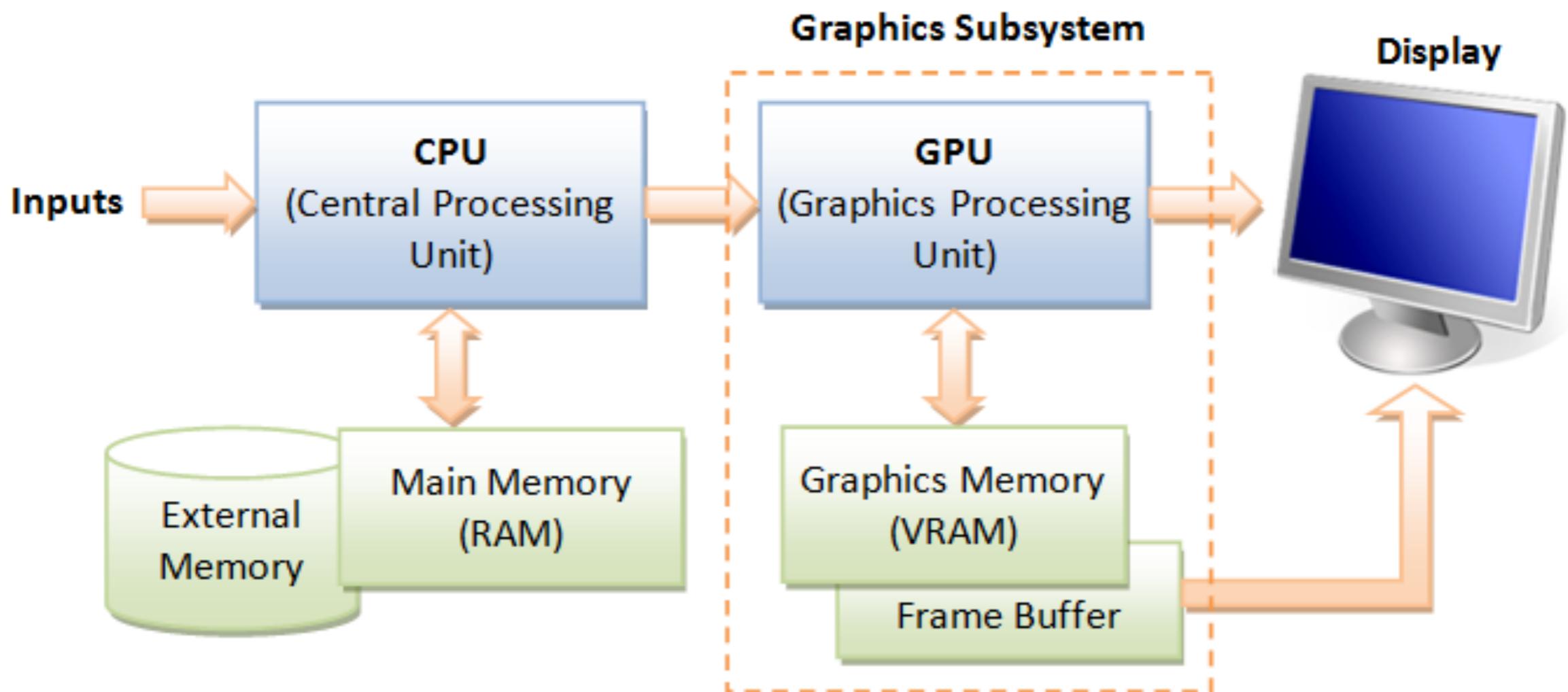
2. Introducció a ZBuffer

Pipeline de visualització:



2. Introducció a ZBuffer

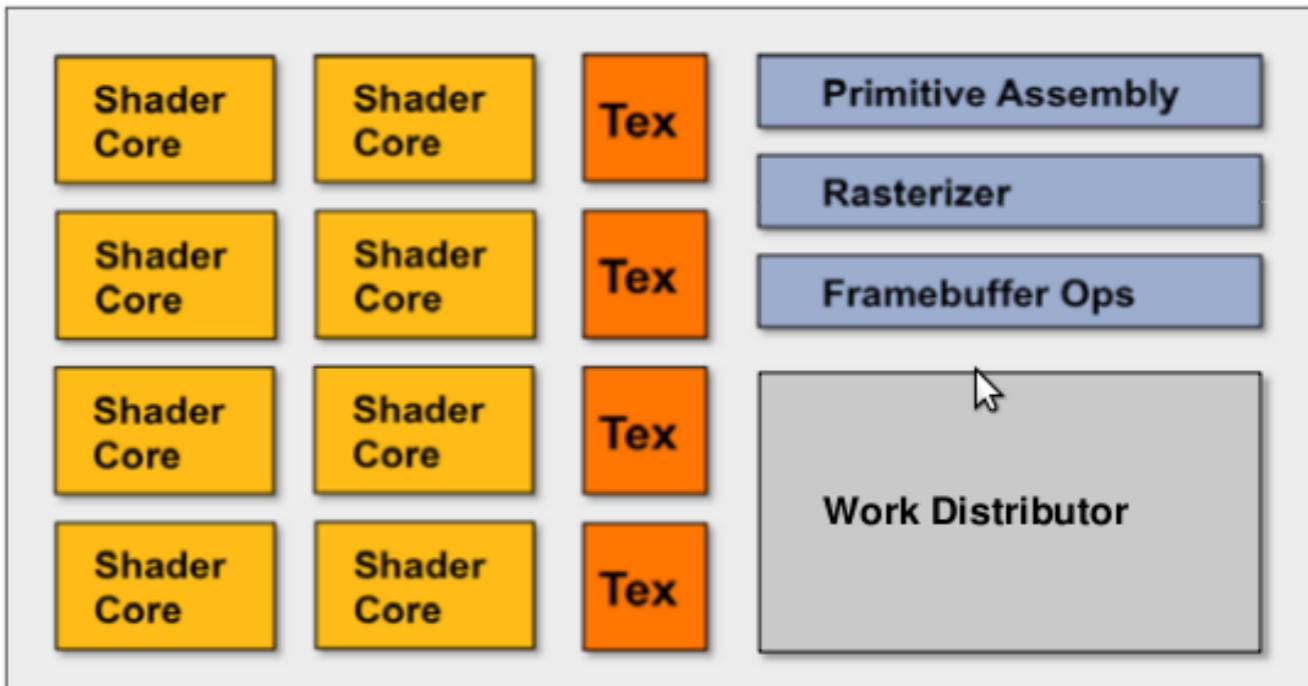
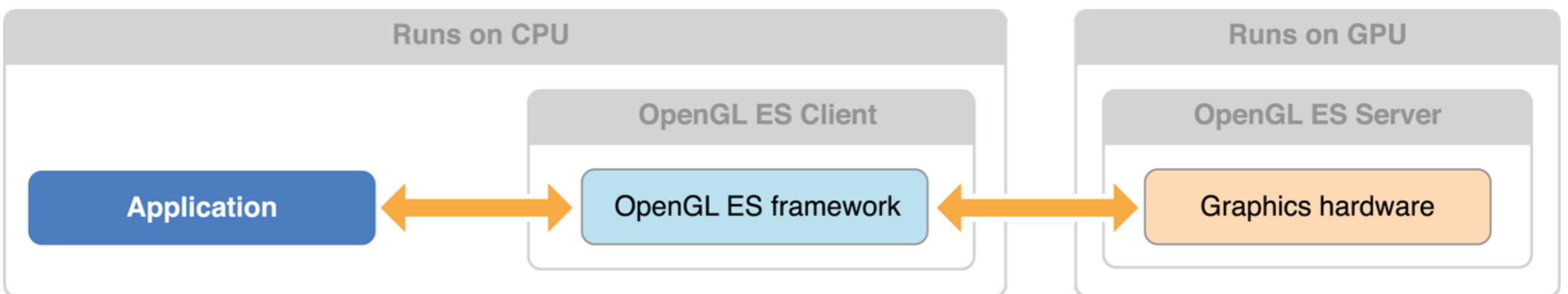
- **Pipeline Integrat a GL:** Arquitectura OpenGL (2.0 programable)



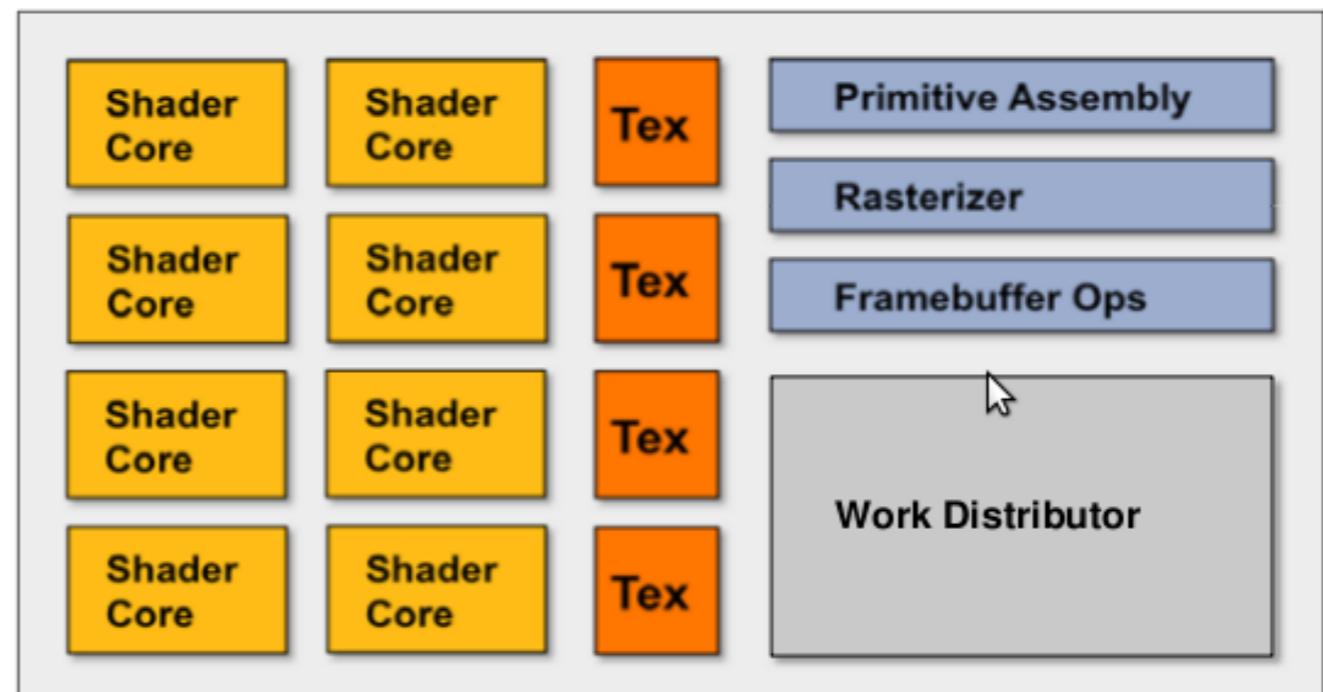
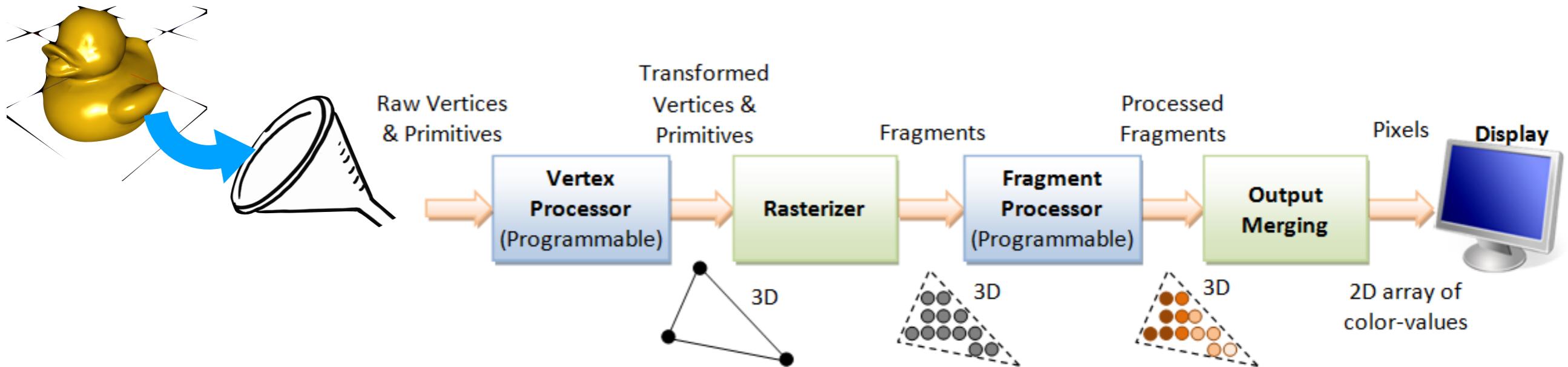
2. Introducció a ZBuffer

- Arquitectura OpenGL

Arquitectura client-servidor

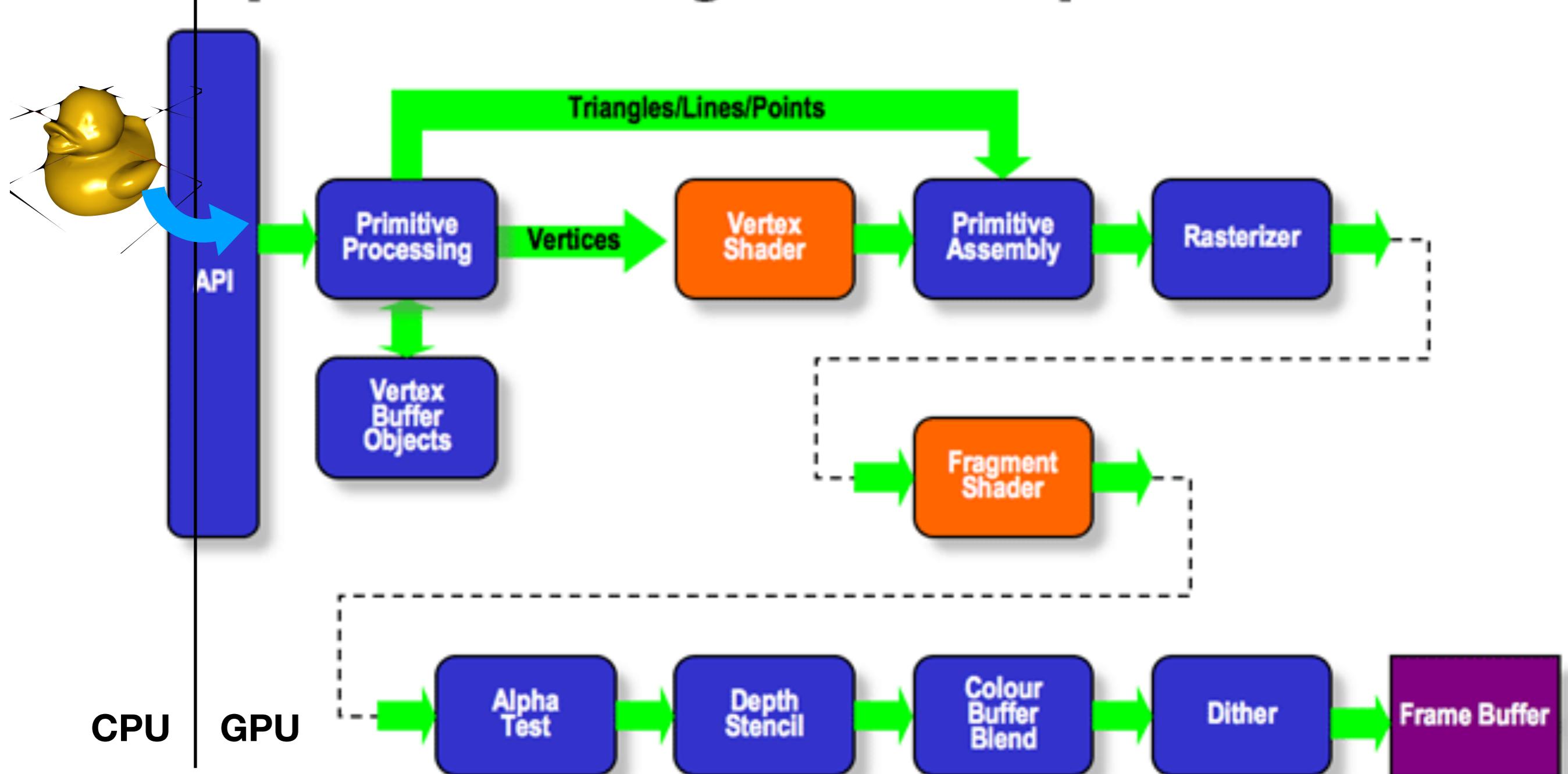


2. Introducció a ZBuffer

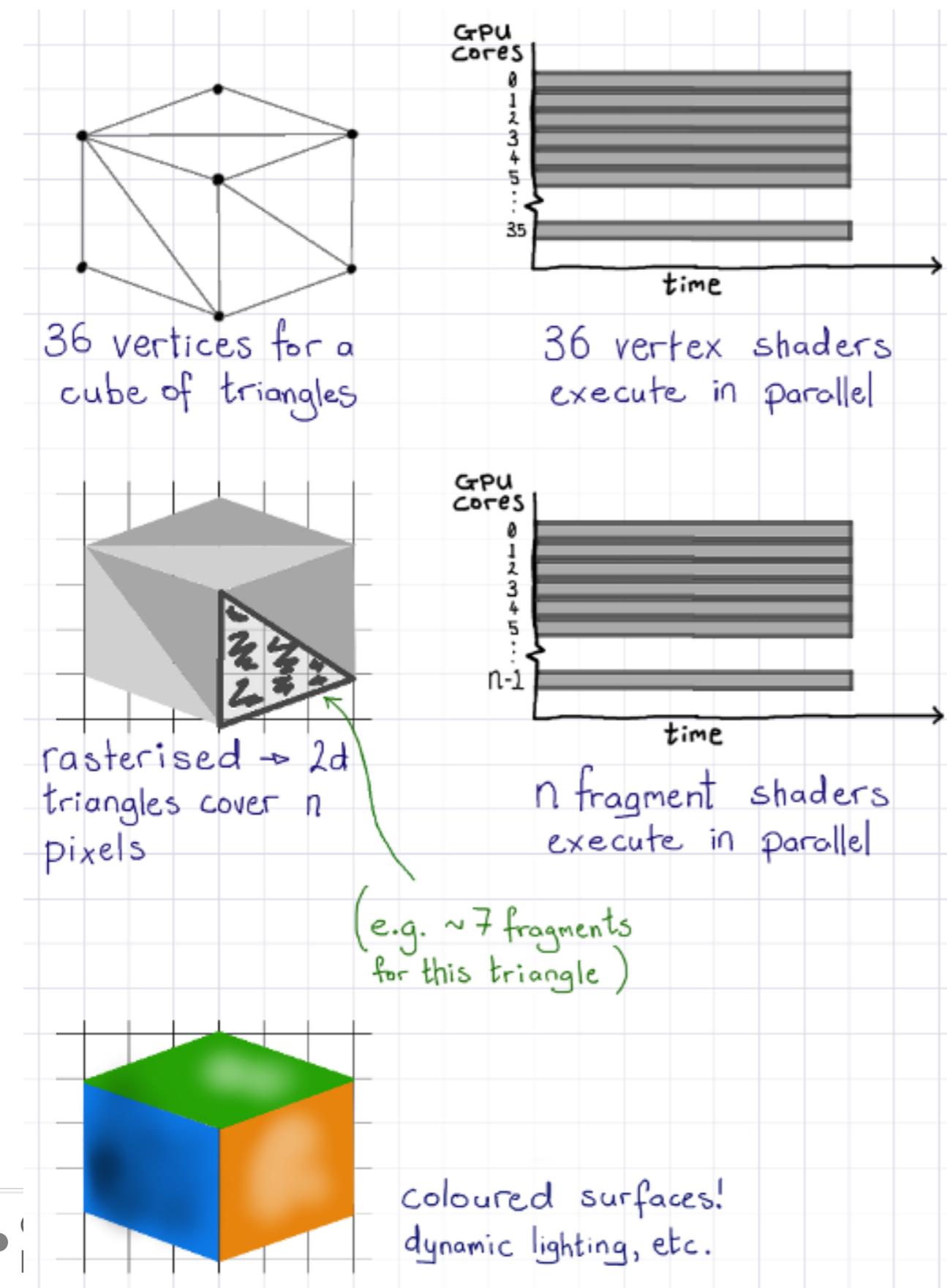


2. Introducció a ZBuffer

OpenGL ES 2.0 Programmable Pipeline



Pipeline programmable: OpenGL i glsl



- Diferència entre **fragment** (àrea equivalent a un píxel d'una superfície) i **píxel** (àrea del frame buffer)
- GPU's: <https://www.notebookcheck.net/Comparison-of-Laptop-Graphics-Cards.130.0.html>

Pipeline fixe: OpenGL



Exemple amb càmera i pintat:

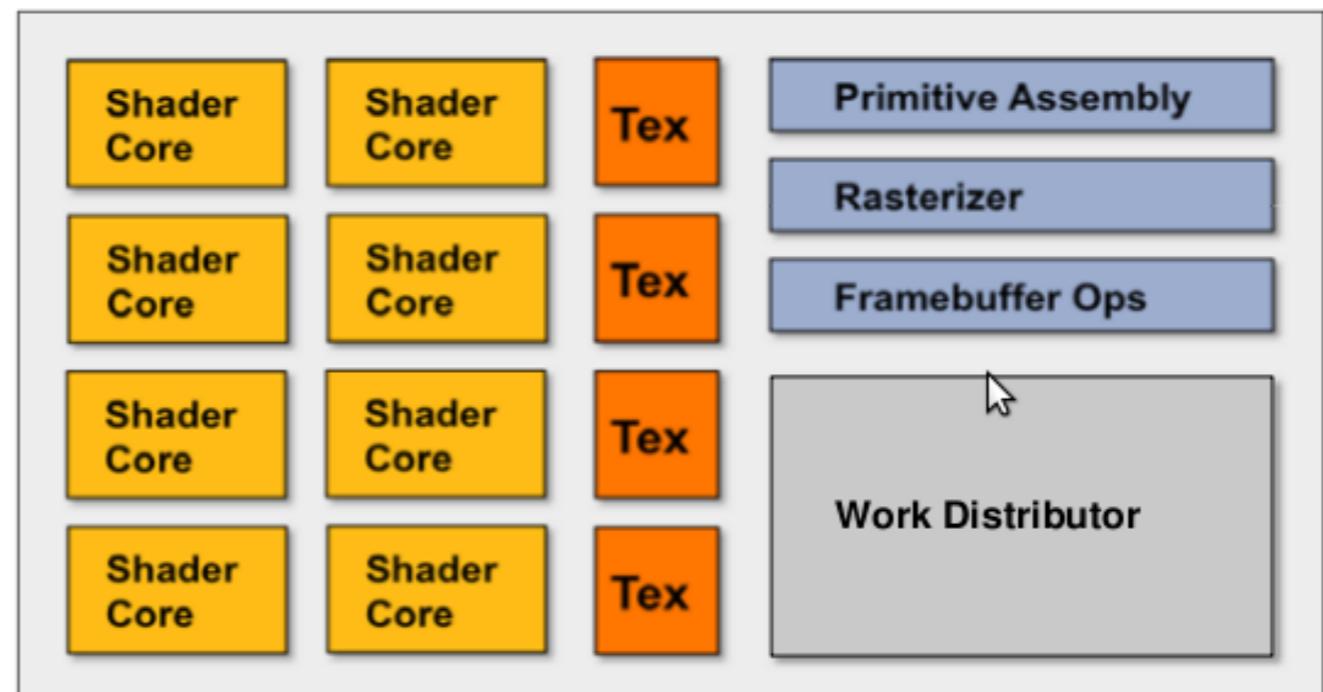
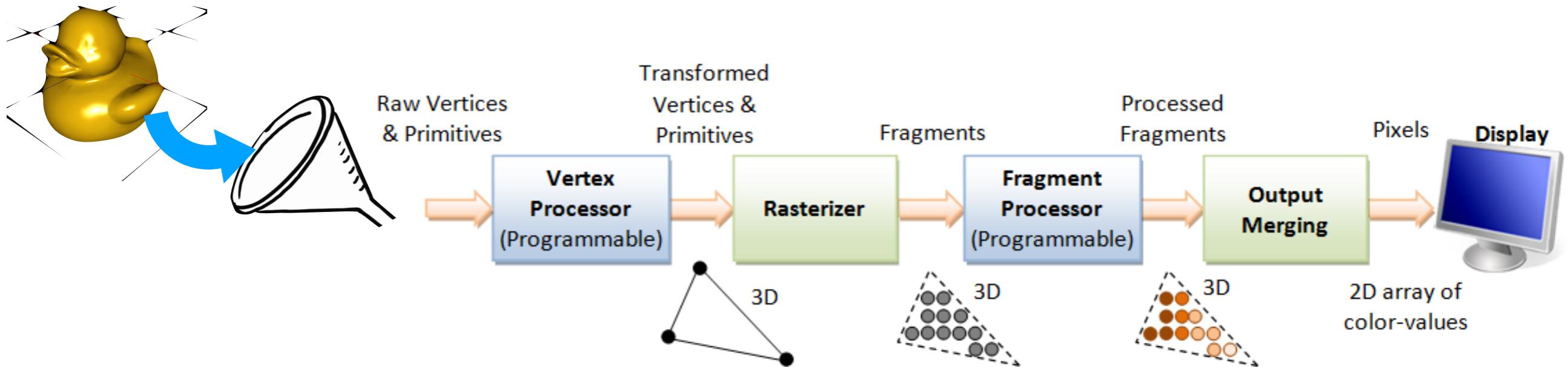
```
glClear( GL_COLOR_BUFFER_BIT );      /* color de background */
glEnable(GL_DEPTH_TEST);           /* activació del test d'eliminació de parts amagades */
glShadeModel(GL_FLAT);            /* model d'il.luminacio */
```

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();                  /* Matriu de projeccio */
                                    /* Inicialitzacio de la matriu de projeccio */
                                    /* identity */
glFrustum( -1, 1, -1, 1, 1, 1000 ); /* Projeccio perspectiva */
```

```
glMatrixMode( GL_MODELVIEW );      /* Model view */
glLoadIdentity();                  /* Inicialitzacio de la matriu de visio */
glTranslatef( 0, 0, -3 );          /* Traslacio dels vertexs, encara que també es
                                    poden rotar segons la posicio de la camera */
```

```
glBegin( GL_POLYGON );
glColor3f( 0, 1, 0 );
glVertex3f( -1, -1, 0 );
glVertex3f( -1, 1, 0 );
glVertex3f( 1, 1, 0 );
glVertex3f( 1, -1, 0 );
glEnd();                           /* Pintat de vertexs */
                                    /* Set the current color to green */
                                    /* Issue a vertex */
                                    /* Enviament de tot a la targeta gràfica */
```

2. Introducció a ZBuffer



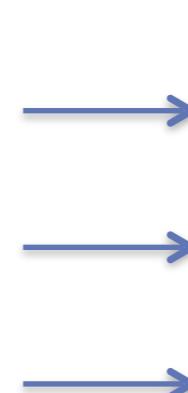
Pipeline programable: OpenGL i glsl

- Arquitectura OpenGL programant els shaders
 - Arquitectura client-servidor



Client:

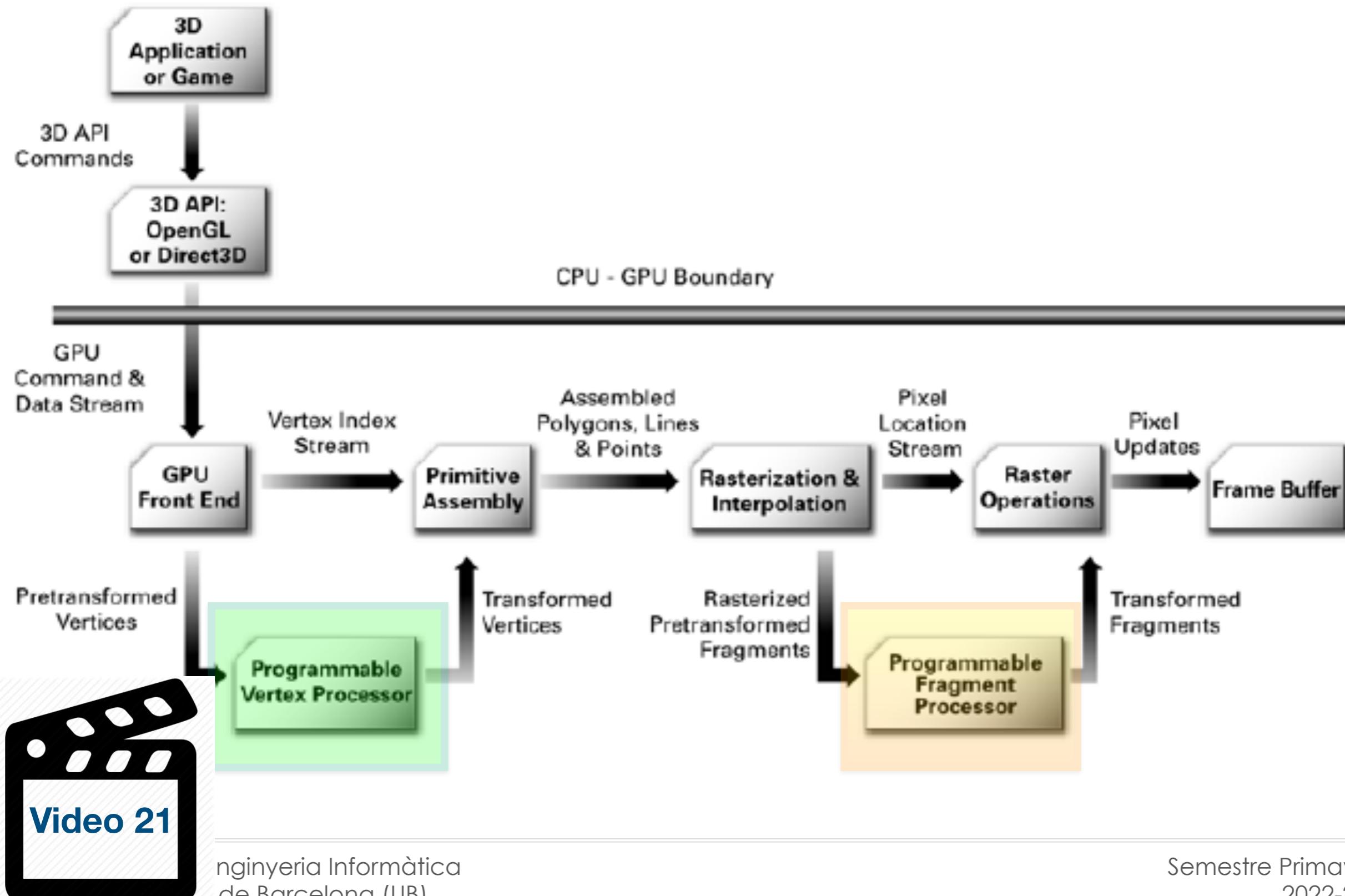
1. S'inicialitzen els programes o **shaders** a ser executats a la GPU (*vertex shader* i *fragment shader*)
2. S'envien les dades a la GPU (vèrtexs, normals, llums, materials): uniform/ in
3. S'executa el pintat (glDraw)



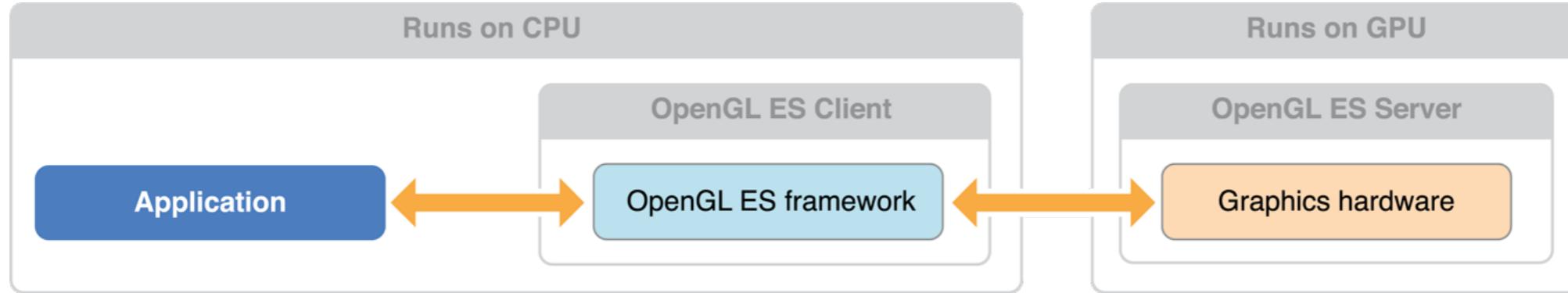
Servidor:

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

Pipeline programmable: OpenGL i glsl



Pipeline programmable: OpenGL i glsl



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

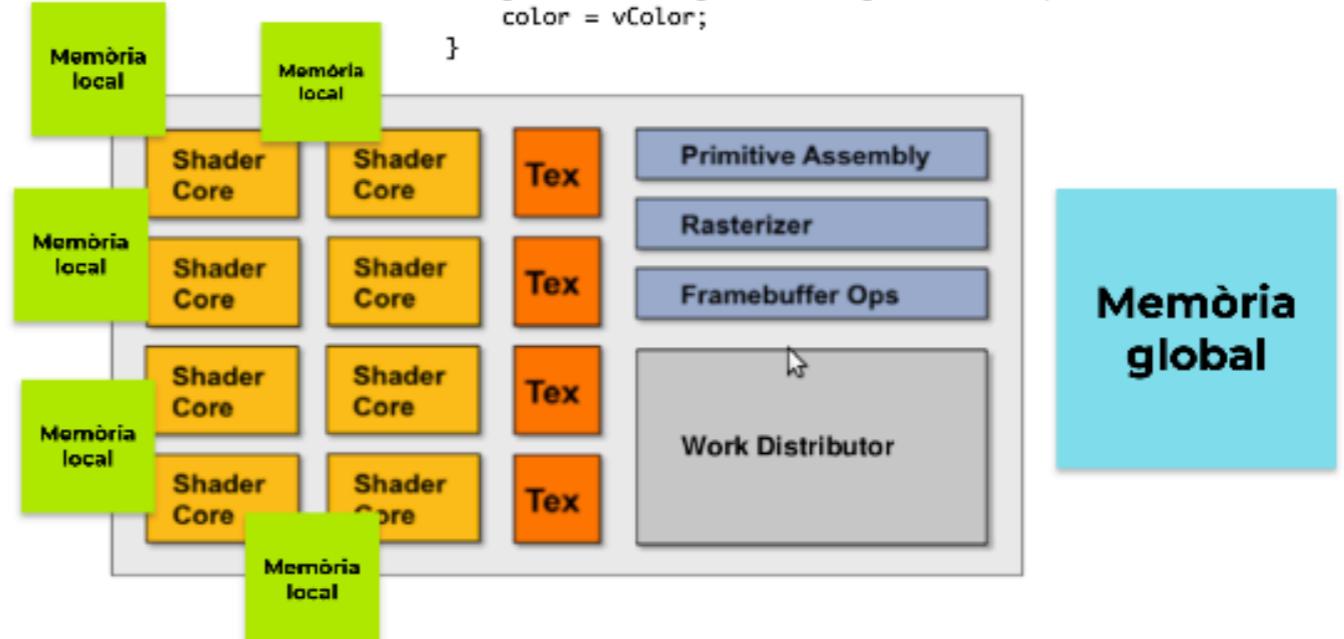
```
layout (location = 0) in vec4 vPosition;  
layout (location = 1) in vec4 vColor;  
  
uniform mat4 model_view;  
uniform mat4 projection;  
  
out vec4 color;  
  
void main()  
{  
    gl_Position = projection*model_view*vPosition;  
    gl_Position = gl_Position/gl_Position.w;  
    color = vColor;  
}
```

GLWidget

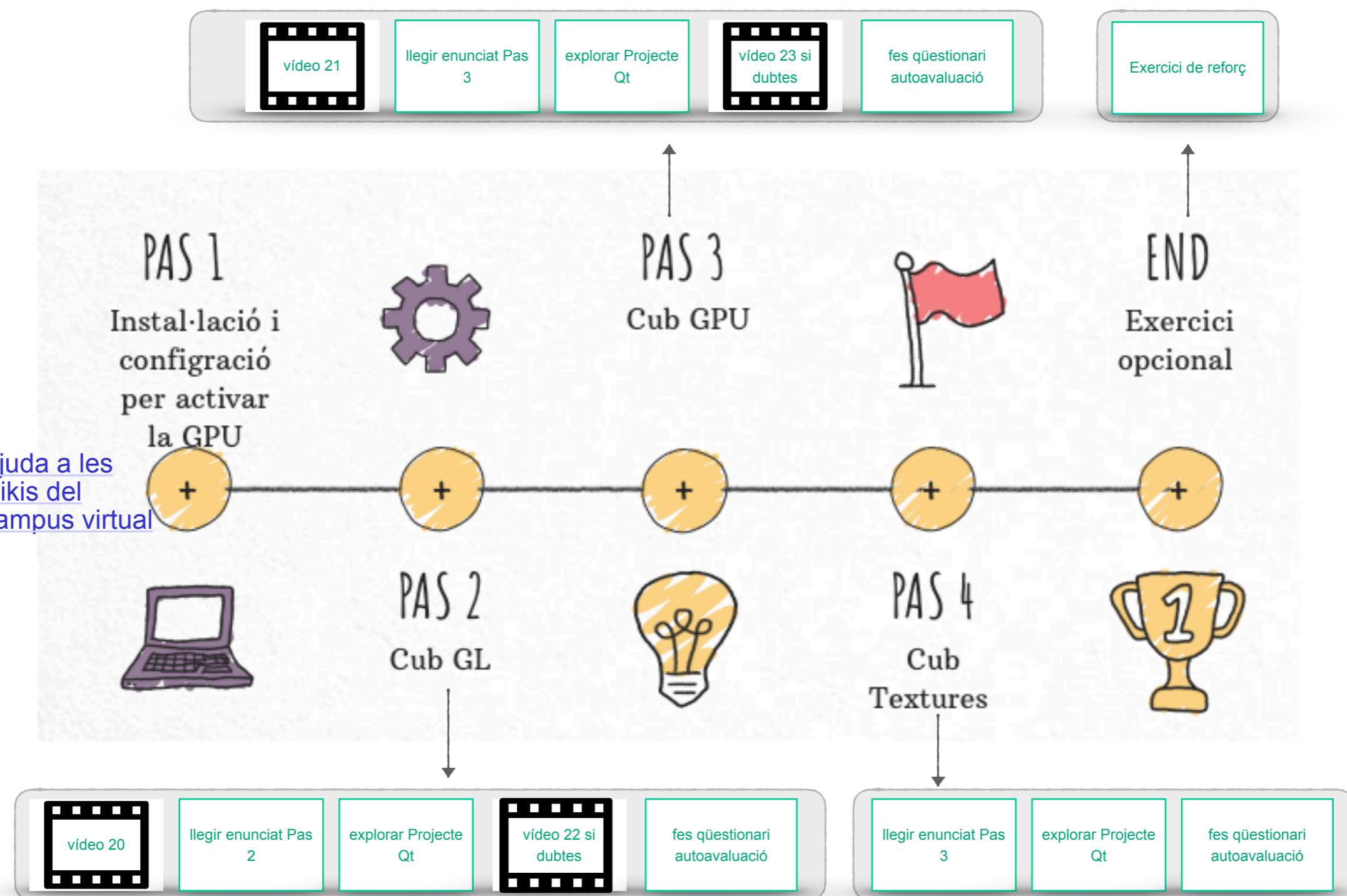
initializeGL()

altres mètodes repartits (Scene, Objects, Camera, Material, Llum, ...)

paintGL()



Fase 0 Pràctica 2



[1] Veure secció 1.7 del llibre [Angel2011]
[2] Transparències del Tema 3a