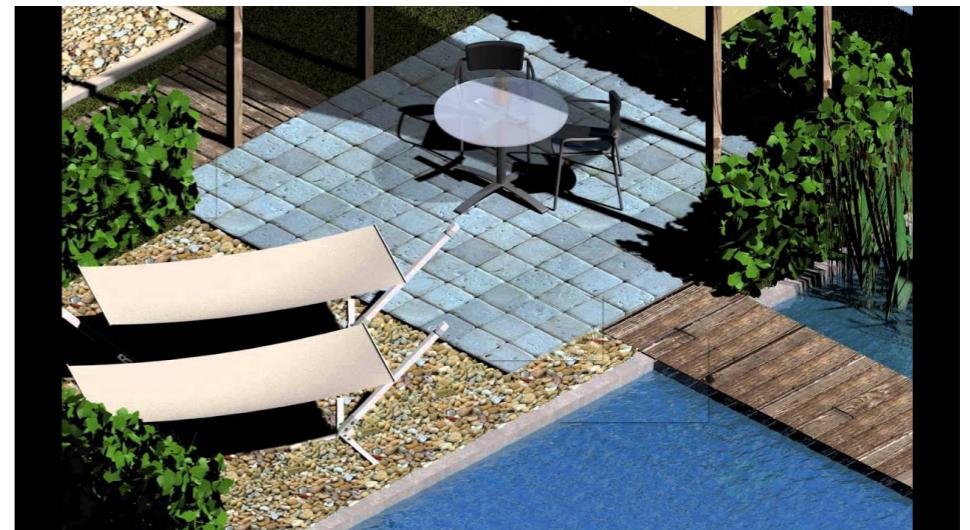
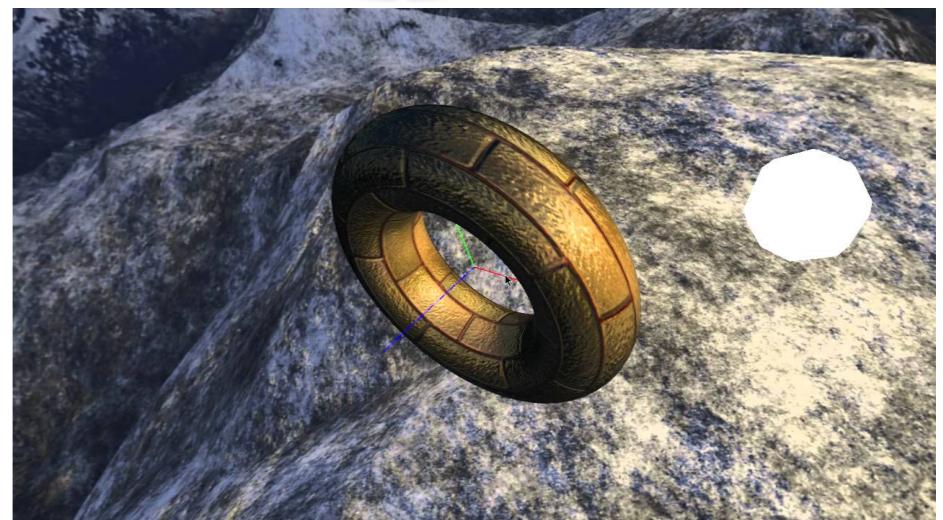
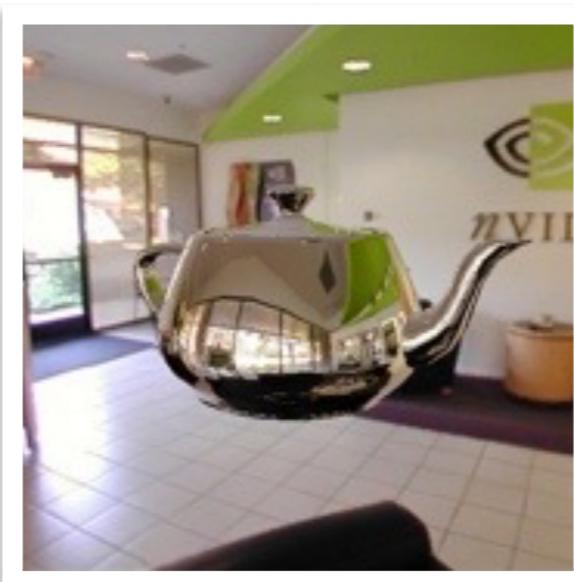


Sessió Setmana 13

GiVD 2022–23

Guió de la sessió

1. Planificació de la setmana 13
 2. Projectes: funcionament
 3. Recap shading - textures
 4. Reflexions, transparències i textures avançades
 5. Consultes de pràctiques
-



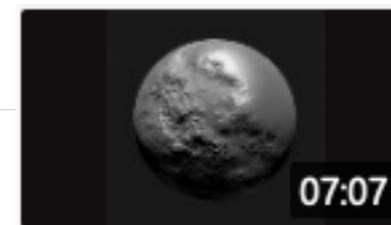
1. Planificació de la setmana

- Veure el campus: Setmana 13



Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	
15	16	17	18	19	20	21	
Tema 3	Pràctica 2						Set. 13
22	23	24	25	26	27	28	
Entrevista/Prova Practica 2	Entrevista/Prova Practica 2						Set. 14
29	30	31	01	02	03	04	
	Problemes 3						

Environmental mapping
Reflexions
Transparències
Bump-mapping
Displacement mapping



2. Projectes: enunciats

Identificador	Títol del treball
MÈTODES PER A EXPLORAR DIFERENTS VISUALITZACIONS I EFECTES	
4.1	Reflexions a la GPU
4.2	Fentombres a la GPU
4.3	Ambient occlusion in GPU
4.4	Tècnica realista de Photon Mapping
4.5	Tècniques avançades usant deferred shdaers
4.6	Illustració en la GPU (Non-Photorealistic techniques)
4.7	Transparències a la GPU
4.8	Tècniques per definir càmeres
MÈTODES PER A VISUALITZAR ALTRES TIPUS DE DADES	
5.1	Visualització d'arbres
5.2	Volume Rendering
5.3	Visualitzant Fractals 3D
5.4	Visualització de Point Clouds
5.5	Visualització i animació de sistemes de partícules
5.6	Visualització de superfícies implícites
PROJECTES PER EXPLORAR TÈCNIQUES USADES A JOCS	
6.1	Minecraft: A RyBox Intersection Alorithmm and Efficient Dynamic Voxel Rendering
6.2	Analitzant un frame: The Rendering of Jurassic World: Evolution
6.3	Analitzant un frame: Metal Gear Solid V - Graphics Study
6.4	Analitzant un frame: Mafia: Definitive Edition 2020
6.5	Analitzant un frame: Nanite
6.6	Doom - Graphics Stud

2023 June

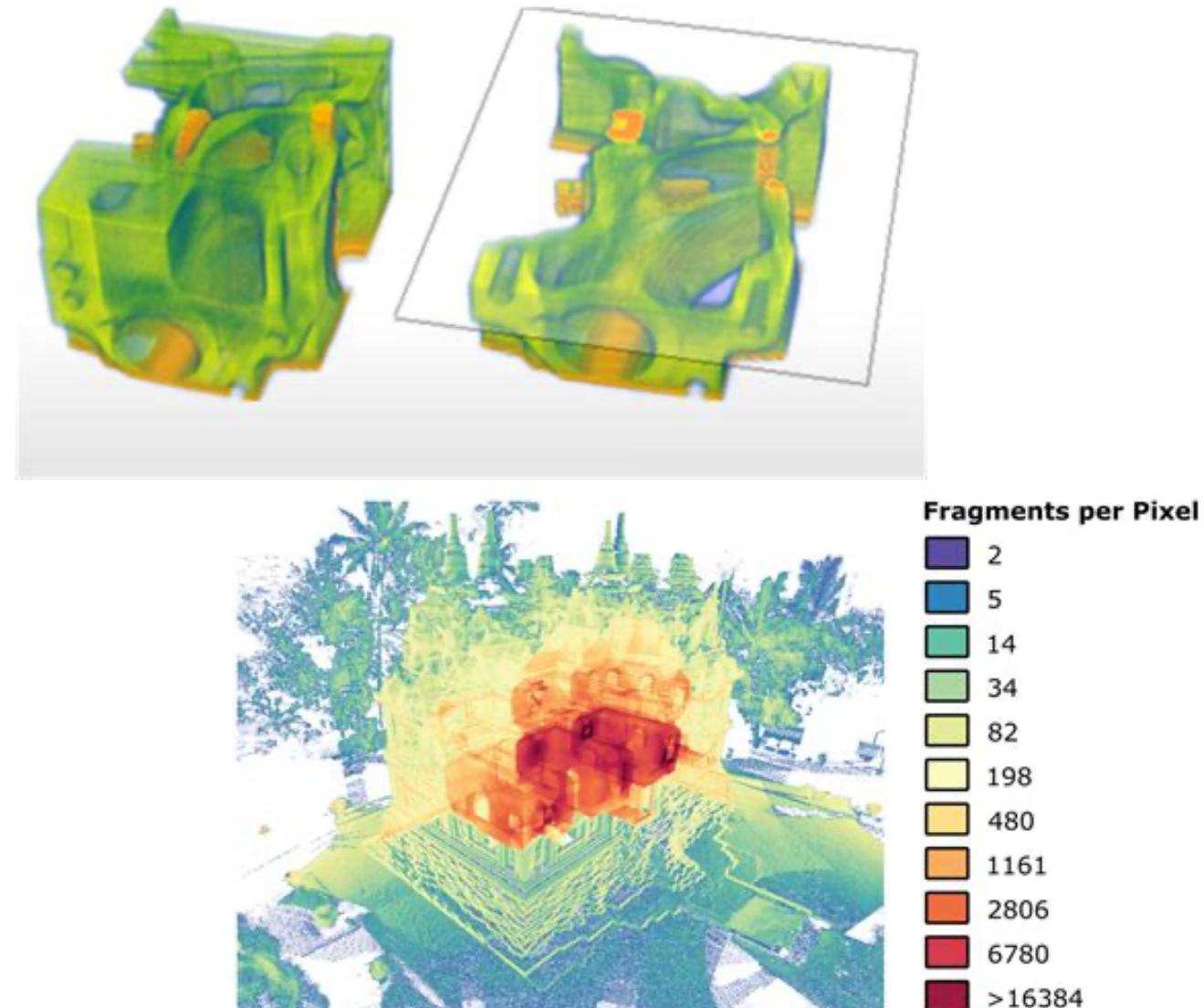


Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
29	30	31	01	02	03	04
1er Torn 15:00-20:00						
05	06	07	08	09	10	11
			2on Torn: 15:00-20:00	3er Torn: B3 15:00-20:00		

2. Projectes: enunciats

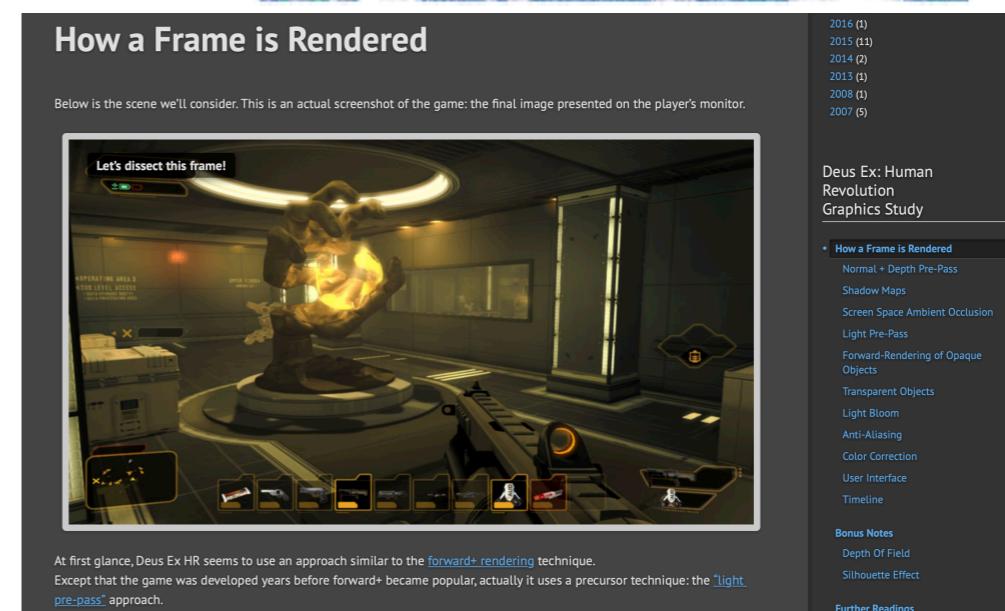
Per al tema escollit cal que:

- repartició dels **subtemes o estratègies**
- Memòria (segons les macros de Latex)
MÀXIM de 12 pàgines
- **NO** cal fer implementacions
- Presentació en PDF:
 - MÍNIM 15 min i MÀXIM de 20 min.



Lliuraments per tothom:

- El dia del TORN assignat: abans de 10h
- MEMÒRIA + Presentació
- Després de cada presentació hi haurà torn de preguntes



2. Projectes: enunciats

1. Breu resum o abstract de la contribució global de l'equip (feina grupal)
2. Introducció del problema a resoldre: Explicació i motivació del problema (feina grupal)
3. Per a cadascuna de les 4 possibles estratègies o 4 possibles parts: (feina individual):
 - (a) Explicació de la solució inicial des del punt de vista teòric (si escau): Quina és la part teòrica? La seva fórmula o el seu marc teòric? Quin model d'objectes gràfics es fan servir? En quina estratègia de visualització es fa servir? com s'integra en el pipeline gràfic?
 - (b) Estratègia de la solució: descripció
 - (c) Aplicacions, simulacions de l'estratègia
 - (d) Avantatges i inconvenients
4. Anàlisi conjunta de les estratègies i mètodes estudiats (feina grupal)
5. Feina realitzada per a cada membre de l'equip
6. Bibliografia

2. Recap Shading - Textures

Pregunta 4

4. Es tenen **tres** objectes que tenen **comportaments amb la llum iguals** excepte el seu color base: un és de color vermell, l'altre blau i l'altre verd. Quan es vol fer un ZBuffer de forma que es calculi la il·luminació per shading per Gouraud, per fer eficients els càlculs i l'enviament de memòria de la CPU a la GPU,....
- a. En el **toGPU()** de l'escena es passa el material d'un dels objectes com a UNIFORM al vertex shader, i a **cada draw() de l'objecte** cal passar la component difusa al vertex shader com a UNIFORM.
 - b. En el **draw()** de l'escena cal enviar per a cada objecte TOT el seu material com a UNIFORM al vertex shader, ja que sinó no es visualitzarien correctament.
 - c. En el mètode **initializeGL()** cal passar només un cop els tres materials de tots els objectes i en el **vertex shader** comprovar quin objecte és i pintar-lo segons el seu material, que es passa com un identificador enter i UNIFORM al vertex shader en el **draw()** de l'objecte
 - d. En el **toGPU()** de cada objecte es passa el seu material com a UNIFORM al vertex shader.

Pregunta 4

4. Es tenen **tres** objectes que tenen **comportaments amb la llum iguals** excepte el seu color base: un és de color vermell, l'altre blau i l'altre verd. Quan es vol fer un ZBuffer de forma que es calculi la il·luminació per shading per Gouraud, per fer eficients els càlculs i l'enviament de memòria de la CPU a la GPU,....

- a. En el `toGPU()` de l'escena es passa el material d'un dels objectes com a UNIFORM al vertex shader, i a cada draw de l'objecte cal passar la component difusa al vertex shader com a UNIFORM.
- b. En el `draw()` de l'escena cal enviar per a cada objecte TOT el seu material com a UNIFORM al vertex shader, ja que sinó no es visualitzarien correctament.
- c. En el mètode `initializeGL()` cal passar només un cop els tres materials de tots els objectes i en el vertex shader comprovar quin objecte és i pintar-lo segons el seu material, que es passa com un identificador enter i UNIFORM al vertex shader en el draw de l'objecte
- d. En el `toGPU()` de cada objecte es passa el seu material com a UNIFORM al vertex shader.

Problema: Fent un target en un cercle verd

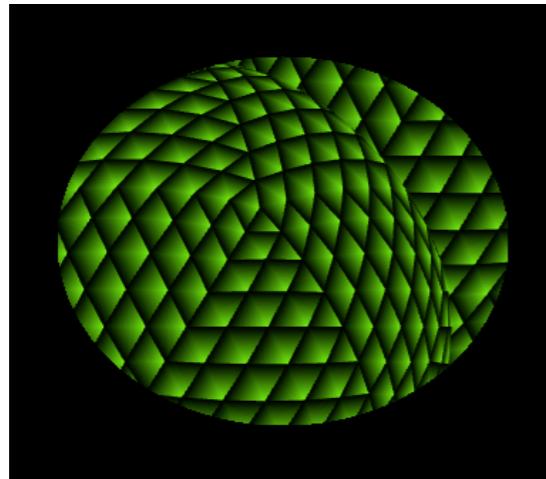
Target amb cercle verd: Es volen visualitzar tots els objectes que estan en un cercle de mida la mitat del viewport (suposant que es té un viewport de 512x512, el radi seria 128) mitjançant Phong Shading només tenint en compte el canal verd. La resta es pintarien de color negre. **On es faria el càcul?**



Problema: Fent un target en un cercle verd

Target amb cercle verd: El càlcul es faria en el *fragment shader*. Es posaria el color de fons a negre. Hi han dues possibilitats:

1. Comparant les coordenades dels vèrtexs en coordenades de càmera que arriben al *fragment shader* (estan entre -1 i 1)



```
#version 330

in vec4 color;
in vec4 pos;
out vec4 colorOut;

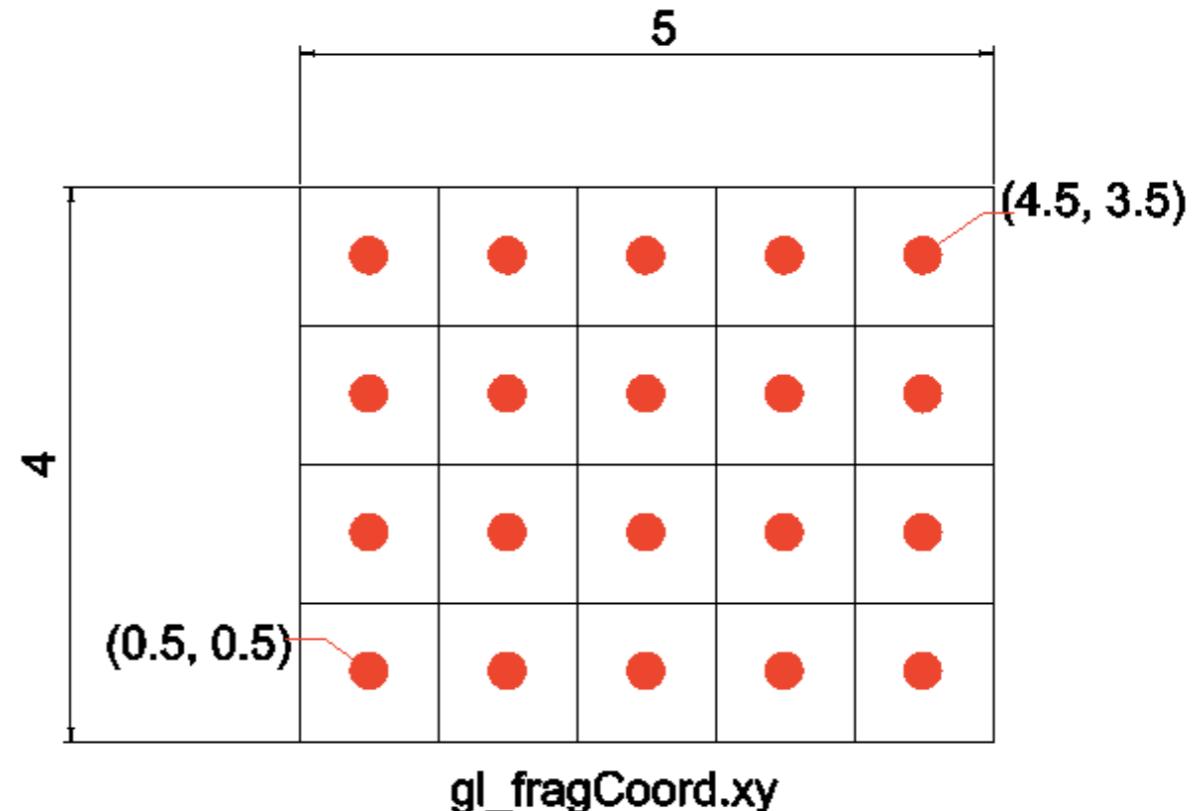
void main()
{
    if (pos.x*pos.x + pos.y*pos.y < 1)
        colorOut = tons de verd
    else
        colorOut = vec4(0.0, 0.0, 0.0, 1.0);
}
```

Problema: Fent un target en un cercle verd

Target amb cercle verd: El càlcul es faria en el *fragment shader*. Es posaria el color de fons a negre.
Hi han dues possibilitats:

2. Passant com a *uniform* les mides del viewport i comparant *glFragCoord*

$$(x - x_c)^2 + (y - y_c)^2 \leq r^2$$



Problema: Fent un target en un cercle verd

Target amb cercle verd: Es volen visualitzar tots els objectes que estan en un cercle de mida la mitat del viewport (suposant que es té un viewport de 512x512, el radi seria 128) mitjançant Phong Shading només tenint en compte el canal verd. La resta es pintarien de color negre. **Com es fa per que es pinta tot de verd, tot i que no hi hagi objecte?**



fitted plane?

Problema: Fent un target en un cercle verd

Target amb cercle verd: Es volen visualitzar tots els objectes que estan en un cercle de mida la mitat del viewport (suposant que es té un viewport de 512x512, el radi seria 128) mitjançant Phong Shading només tenint en compte el canal verd. La resta es pintarien de color negre. **Com aconseguir les tonalitats dels verds?**

Pots fer tonalitats de colors usant el gris de la següent fórmula com a base i multiplicant pel color pur:



$$\text{factor} = \sqrt{\text{color.r} * \text{color.r} + \text{color.b} * \text{color.b} + \text{color.g} * \text{color.g}} / 3.0$$

$$\text{color gris} = (\text{factor}, \text{factor}, \text{factor})$$

Problema: La tempesta de Fornite

La Tempesta de Fornite: Durant el joc de Fornite, existeix una tempesta que afecta a **una regió exterior** d'una esfera centrada en el (0,0,0) de món de radi **9000**. Les regions afectades es visualitzen de colors blaus amb **Gouraud Shading** i les que no estan afectades es veuen en **Phong Shading**.

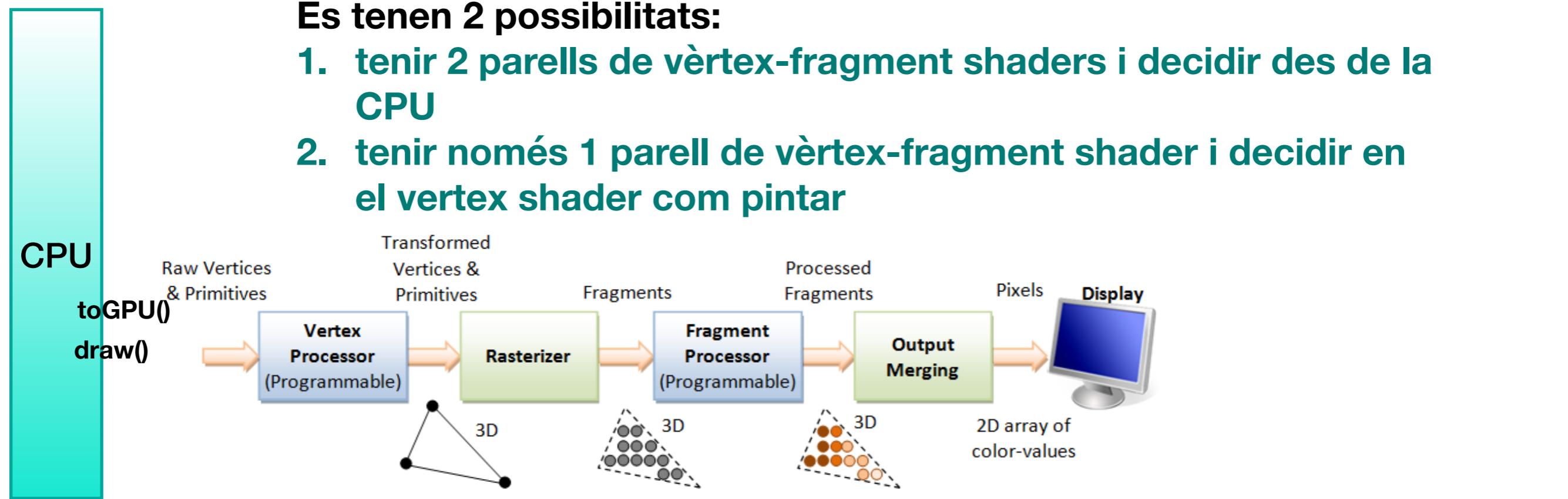


Problema: La tempesta de Fornite

La Tempesta de Fornite: Durant el joc de Fornite, existeix una tempesta que afecta a una regió exterior d'una esfera centrada en el $(0,0,0)$ de món de radi 9000. Les regions afectades es visualitzen de colors blaus amb *Gouraud Shading* i les que no estan afectades es veuen en *Phong Shading*.

Es tenen 2 possibilitats:

- 1. tenir 2 parells de vèrtex-fragment shaders i decidir des de la CPU**
- 2. tenir només 1 parell de vèrtex-fragment shader i decidir en el vertex shader com pintar**

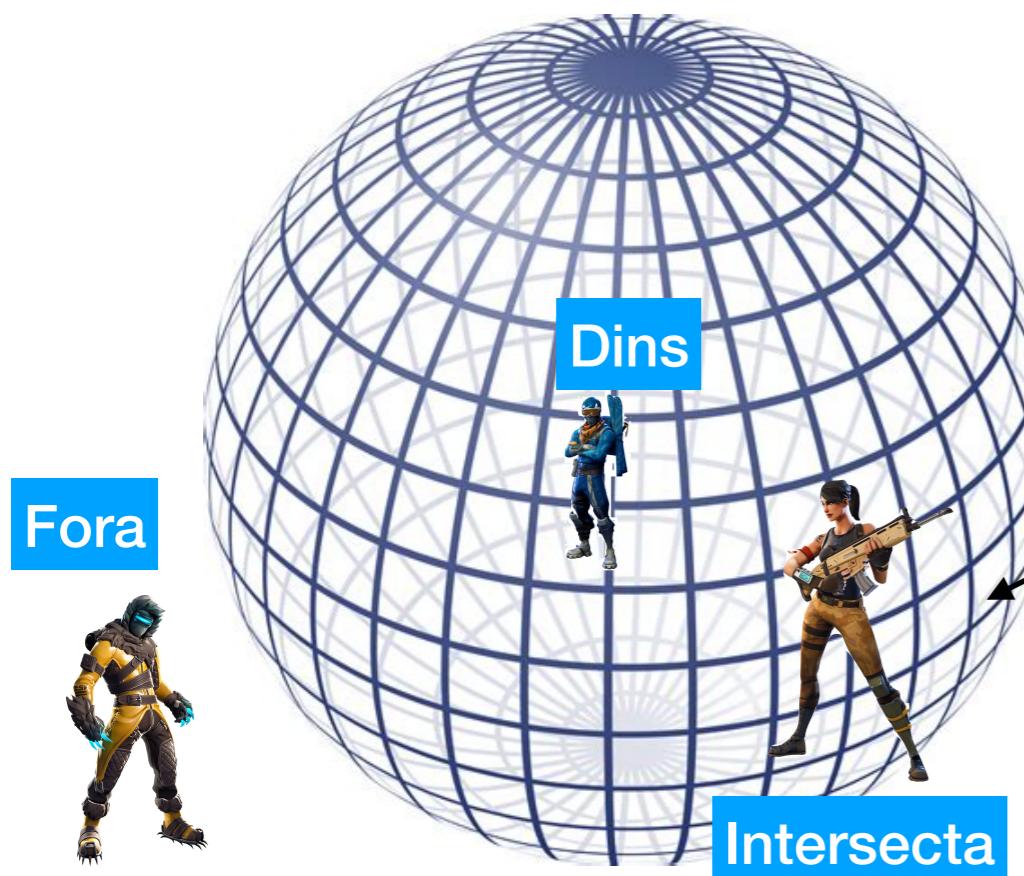


Problema: La tempesta de Fornite

La Tempesta de Fornite:

Es tenen 2 possibilitats:

1. tenir 2 parells de vèrtex-fragment shaders (un per Gouraud i un per a Phong shading) i decidir des de la CPU quin shader activar dependent de si els vèrtexs de l'objecte estan dins o fora de l'esfera de radi 9000



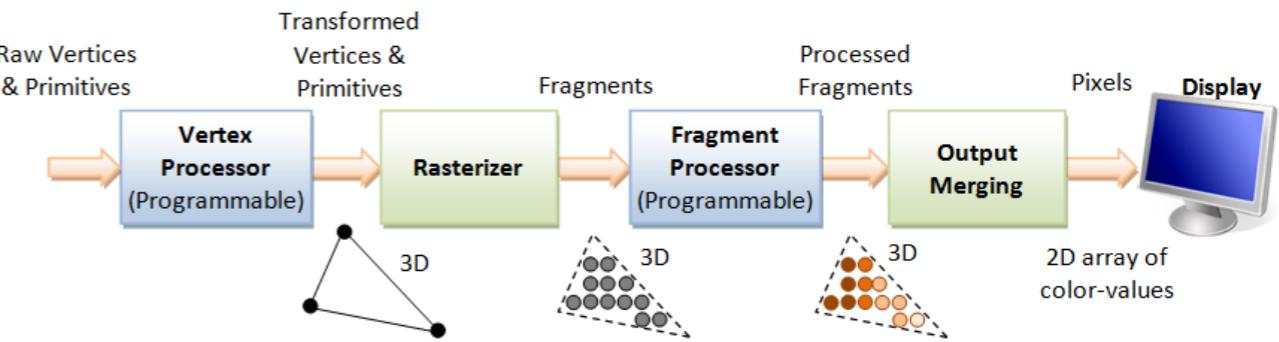
Problema:

Què passa amb els objectes que estan just a la frontera de l'esfera (es a dir, els objectes que tene algun triangle que interseca amb l'esfera)?

Quina parella de vertex-fragment shader els associem?

Problema: La tempesta de Fornite

La Tempesta de Fornite:



2. tenir només 1 parell de vèrtex-fragment shader i decidir en el vertex o en el fragment shader com pintar:

```
// vertex shader
layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vNormal;

out vec4 colorOut;
out vec3 normal;
out bool dins;
void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;

    if (vPosition.x*vPosition.x+
        vPosition.y*vPosition.y+
        vPosition.z*vPosition.z < 9000*9000) {
        colorOut = // Calcul Blinn-Phong
        dins = true;
    }
    else
        dins = false;
    normal = vnormal;
}
```

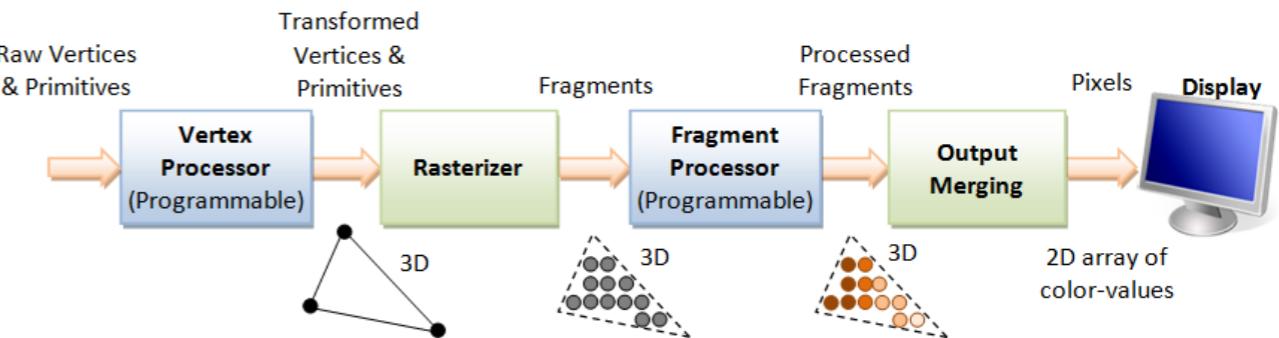
1a versió: Control del càlcul de Blinn-Phong en el fragment shader usant un booleà.

```
// fragment shader
in bool dins;
in vec3 colorOut;
in vec3 normal;

void main()
{
    if (!dins)
        gl_FragColor = // Calcul Blinn-Phong
    else
        gl_FragColor = color;
}
```

Problema: La tempesta de Fornite

La Tempesta de Fornite:



- tenir només 1 parell de vèrtex-fragment shader i decidir en el vertex o en el fragment shader com pintar

```
// vertex shader
layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vNormal;

out vec4 colorOut;
out vec3 normal;
out bool dins;
void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;

    if (vPosition.x*vPosition.x+
        vPosition.y*vPosition.y+
        vPosition.z*vPosition.z < 9000*9000) {
        colorOut = // Calcul Blinn-Phong
        dins = true;
    } else
        dins = false;
    normal = vnormal;
}
```

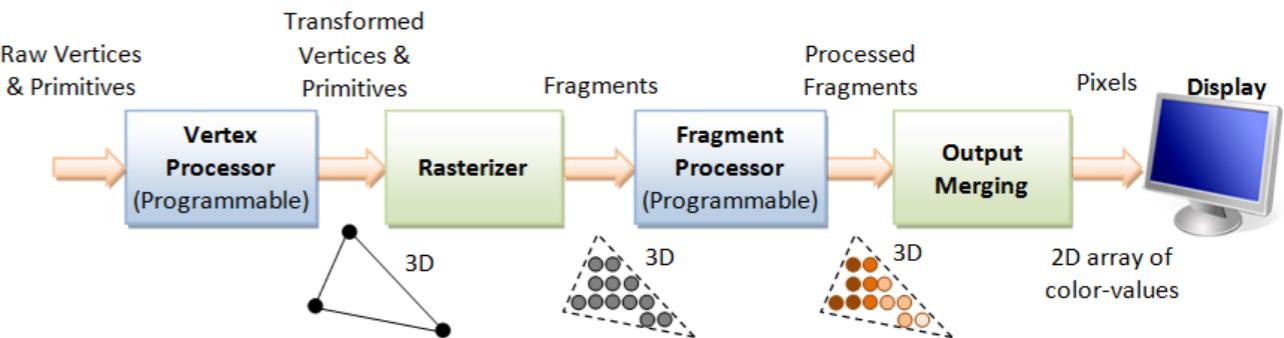
1a versió: Control del càlcul de Blinn-Phong en el fragment shader usant un booleà.
INCORRECTA, ja que llençar un booleà o un enter a rasteritzar (interpolar) des del vertex al fragment shader no té gaire sentit

```
// fragment shader
in bool dins;
in vec3 colorOut;
in vec3 normal;

void main()
{
    if (!dins)
        gl_FragColor = // Calcul Blinn-Phong
    else
        gl_FragColor = color;
}
```

Problema: La tempesta de Fornite

La Tempesta de Fornite:



2. tenir només 1 parell de vèrtex-fragment shader i decidir en el vertex o en el fragment shader com pintar

```
// vertex shader
layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vNormal;
out vec4 colorOut;
out vec3 normal;
out flat bool dins;
void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;

    if (vPosition.x*vPosition.x+
        vPosition.y*vPosition.y+
        vPosition.z*vPosition.z < 9000*9000) {
        colorOut = // Calcul Blinn-Phong
        dins = true;
    }
    else
        dins = false;

    normal = vNormal;
}
```

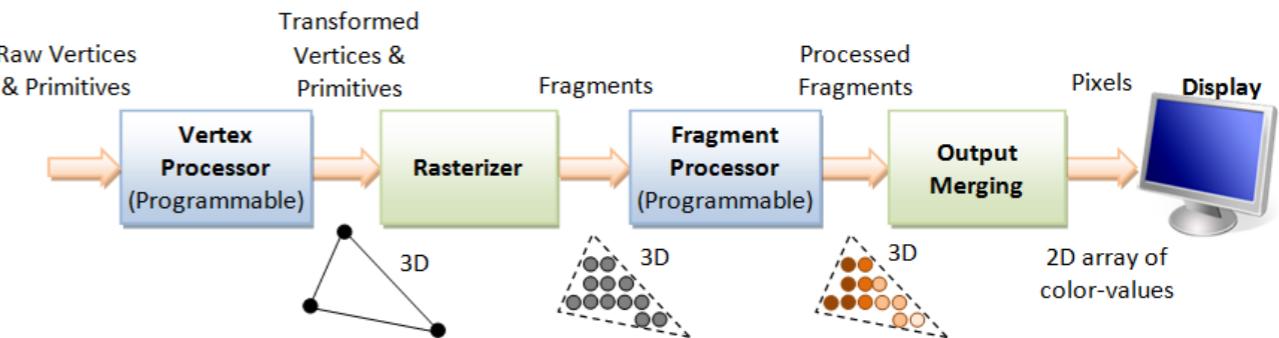
2a versió:
Ús del modificador **flat**. El modificador flat fa que el valor del booleà sigui el mateix valor a tota la primitiva. Ha d'haver-hi un vertex de la primitiva que sigui el líder (o provoking vertex) d'aquell valor.

```
// fragment shader
in bool dins;
in vec3 colorOut;
in vec3 normal;

void main()
{
    if (!dins)
        gl_FragColor = // Calcul Blinn-Phong
    else
        gl_FragColor = color;
}
```

Problema: La tempesta de Fornite

La Tempesta de Fornite:



2. tenir només 1 parell de vèrtex-fragment shader i decidir en el vertex o en el fragment shader com pintar

```
// vertex shader
layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vNormal;
out vec4 colorOut;
out vec3 normal;
out flat bool dins;
void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;

    if (vPosition.x*vPosition.x+
        vPosition.y*vPosition.y+
        vPosition.z*vPosition.z < 9000*9000) {
        colorOut = // Calcul Blinn-Phong
        dins = true;
    }
    else
        dins = false;

    normal = vNormal;
}
```

2a versió:
Ús del modificador flat. El modificador flat fa que el valor del booleà sigui el mateix valor a tota la primitiva. Ha d'haver-hi un vertex de la primitiva que sigui el líder (o provoking vertex) d'aquell valor. En aquest cas, no tenim cap vertex que mani!! Solució INCORRECTA

```
// fragment shader
in bool dins;
in vec3 colorOut;
in vec3 normal;

void main()
{
    if (!dins)
        gl_FragColor = // Calcul Blinn-Phong
    else
        gl_FragColor = color;
}
```

Problema: La tempesta de Fornite

La Tempesta de Fornite:

2. tenir només 1 parell de vèrtex-fragment shader i decidir en el vertex o en el fragment shader com pinta

```
// vertex shader
layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vNormal;
out vec4 colorOut;
out vec3 normal;

void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;

    if (vPosition.x*vPosition.x+
        vPosition.y*vPosition.y+
        vPosition.z*vPosition.z < 9000*9000)
        colorOut = // Calcul Blinn-Phong
    else
        colorOut = vec4(-1.0, -1.0, -1.0, 1.0);

    normal = vnormal;
}
```

3a versió:

Posar un valor no vàlid en el color per a controlar els càlculs en el fragment shader.

```
// fragment shader
in vec3 colorOut;
in vec3 normal;

void main()
{
    if (colorOut.r < 0.0)
        gl_FragColor = // Calcul Blinn-Phong
    else
        gl_FragColor = color;
}
```

Problema: La tempesta de Fornite

La Tempesta de Fornite:

- tenir només 1 parell de vèrtex-fragment shader i decidir en el vertex o en el fragment shader com pinta

```
// vertex shader
layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vNormal;
out vec4 colorOut;
out vec3 normal;

void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;

    if (vPosition.x*vPosition.x+
        vPosition.y*vPosition.y+
        vPosition.z*vPosition.z < 9000*9000)
        colorOut = // Calcul Blinn-Phong
    else
        colorOut = vec4(-1.0, -1.0, -1.0, 1.0);

    normal = vnormal;
}
```

3a versió:

Posar un valor no vàlid en el color per a controlar els càlculs en el fragment shader.

INCORRECTA, ja que el color arriba al fshader interpolat. No hi ha control d'on canvia el pixel de dins i fora de l'esfera

```
// fragment shader
in vec3 colorOut;
in vec3 normal;

void main()
{
    if (colorOut.r < 0.0)
        gl_FragColor = // Calcul Blinn-Phong
    else
        gl_FragColor = color;
}
```

Problema: La tempesta de Fornite

La Tempesta de Fornite:

2. tenir només 1 parell de vèrtex-fragment shader i decidir en el vertex o en el fragment shader com pintar:

4a versió:

Càcul de Blinn-Phong tant en el vèrtex com en el fragment shader.

Només controlar si el punt és dins o fora de l'esfera en el fragment shader (a nivell de píxel).

VERSIÓ CORRECTA

Problema: La tempesta de Fornite

La Tempesta de Fornite:

Versió optimitzada

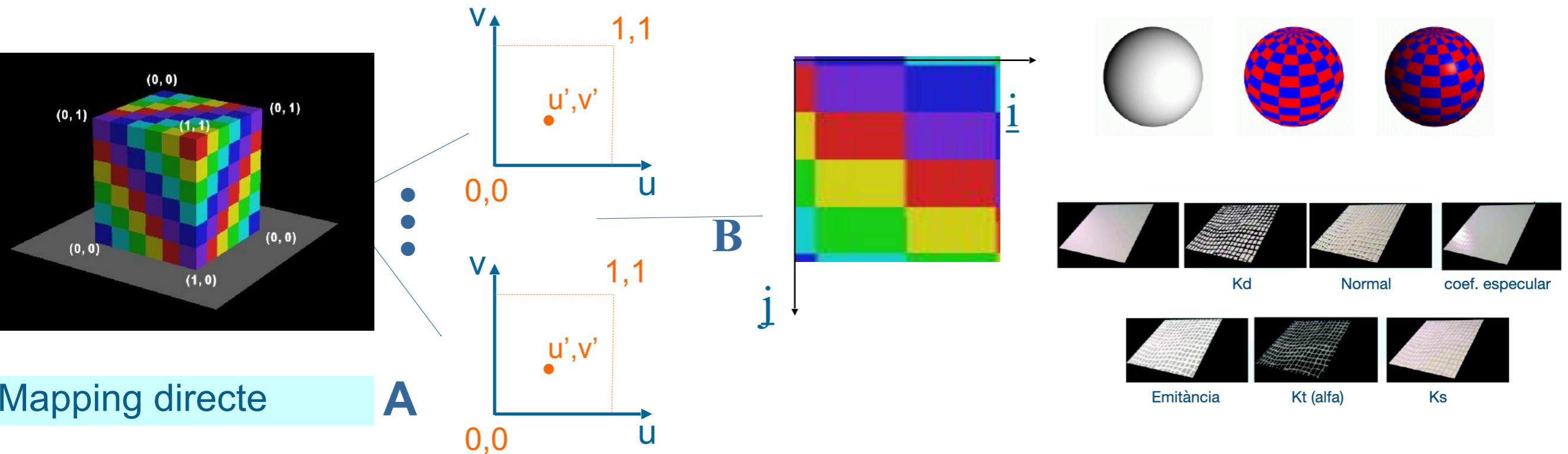
VERSIÓ CORRECTA

3. Combinació de solucions des de la CPU i la GPU:

- Es tenen tres parells de vertex-fragment shaders
 1. una parella que implementa Gouraud
 2. una parella que implementa Phong shading
 3. una parella com la de la transparència anterior

CPU		GPU
Si la capsà contenidora (o esfera contenidora) de l'objecte és dins de l'esfera de radi 9000	s'activa Phong shading	Tots els triangles del l'objecte es pinten segons Phong shading
Si la capsà contenidora de l'objecte (o esfera contenidora) és fora de l'esfera de radi 9000	s'activa Gouraud shading	Tots els triangles del l'objecte es pinten segons Gouraud shading
Si la capsà (o esfera) contenidora intersecta l'esfera de radi 9000	s'activa la parella de shaders mixta (transp.24)	Segons el píxel rasteritzat del triangle, es pinta segons Phong o segon Gouraud

3. Recap. textures



A : Funció de (x, y, z) a (u, v) .

Funció de projecció a (u,v)

B : Funció de (s, t) al texel (i, j)

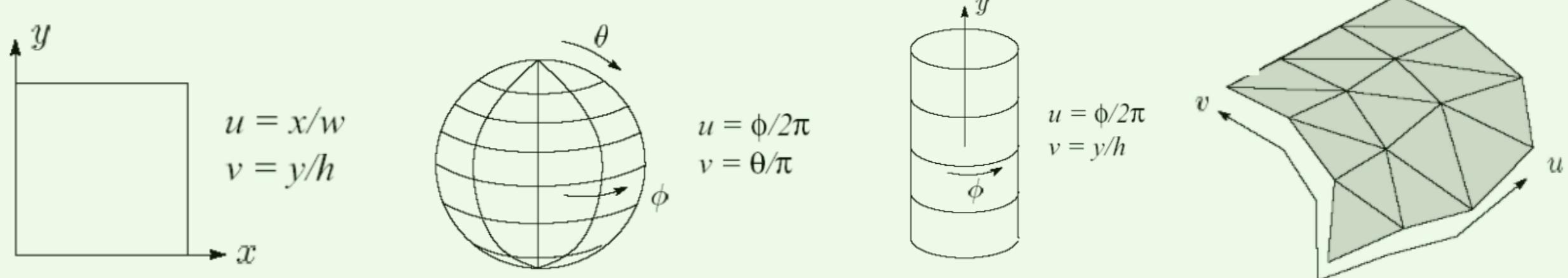
Funció de correspondència (texel (i,j))

3. Recap. textures

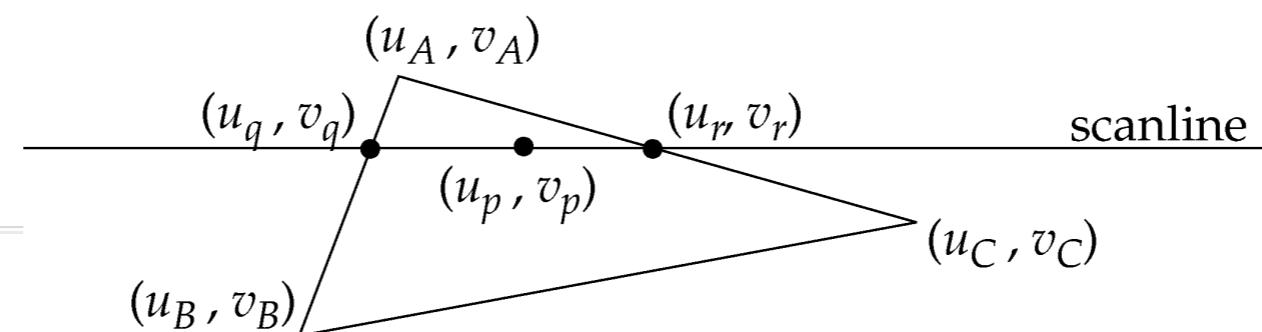
Funció de projecció: $\mathbf{A} : (x, y, z) \rightarrow (u, v)$

Les coordenades (x, y, z) d'un objecte es poden escriure en coordenades paramètriques (u, v) mitjançant la funció de projecció A:

1. Per a cada vèrtex de cada polígon de l'objecte, es calculen les coordenades paramètriques (u, v)

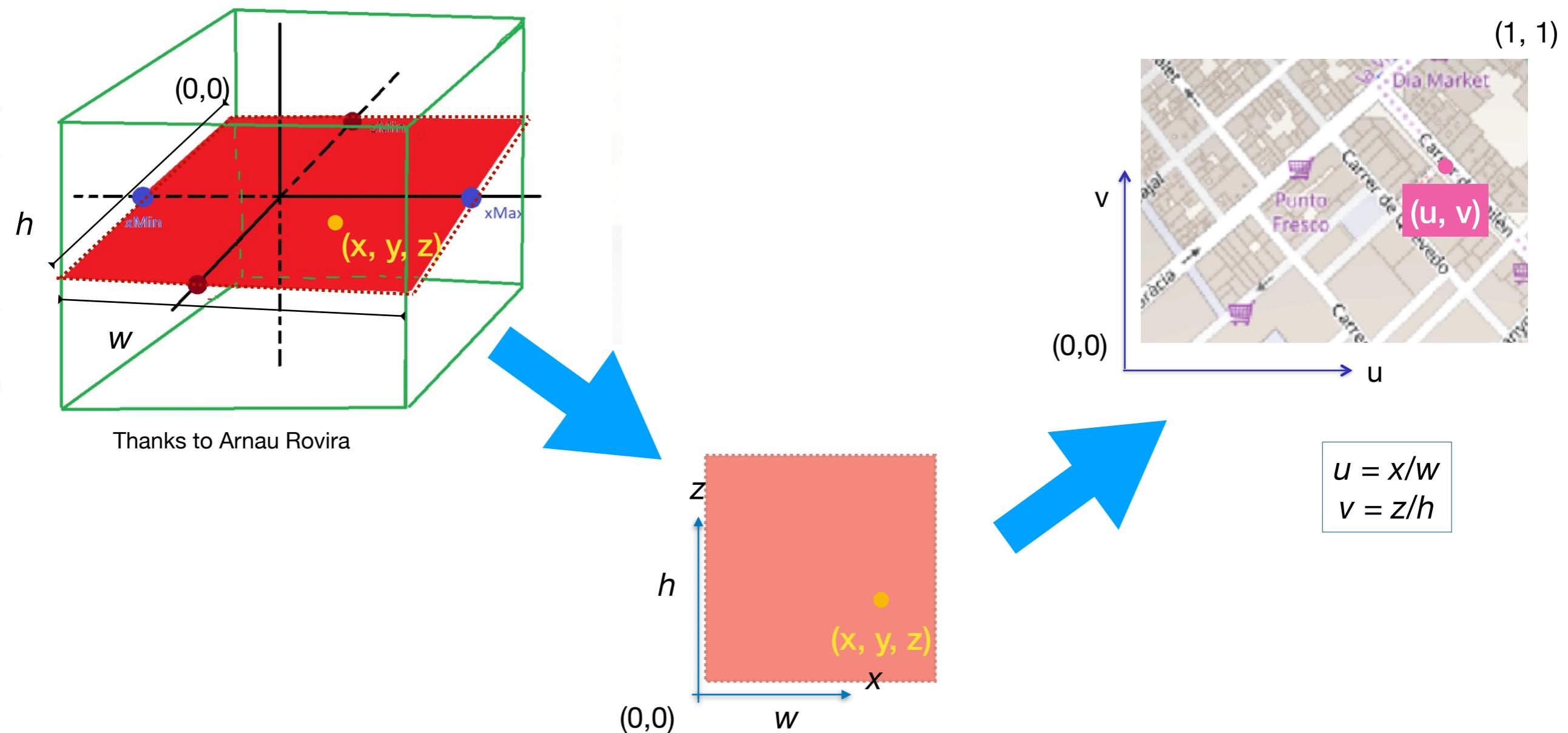


2. Per a cada punt interior del polígon, es calculen les coordenades paramètriques a partir de les coordenades paramètriques dels vèrtexs extrems del polígon



3. Recap. textures

- **Pla:** $(x, y, z) \rightarrow (u, v)$



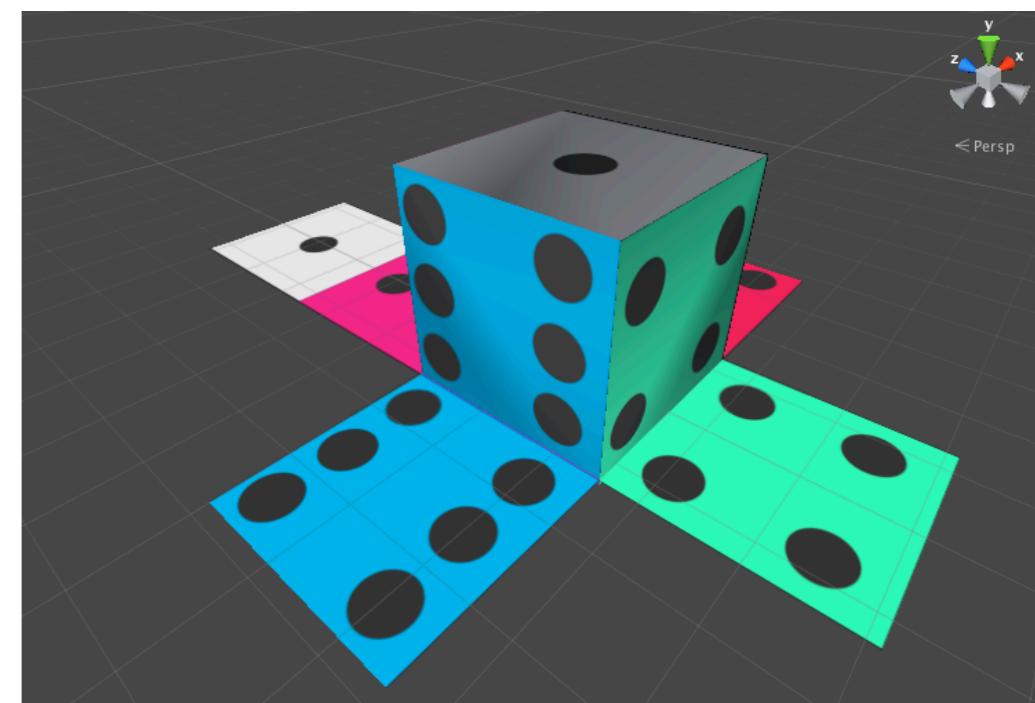
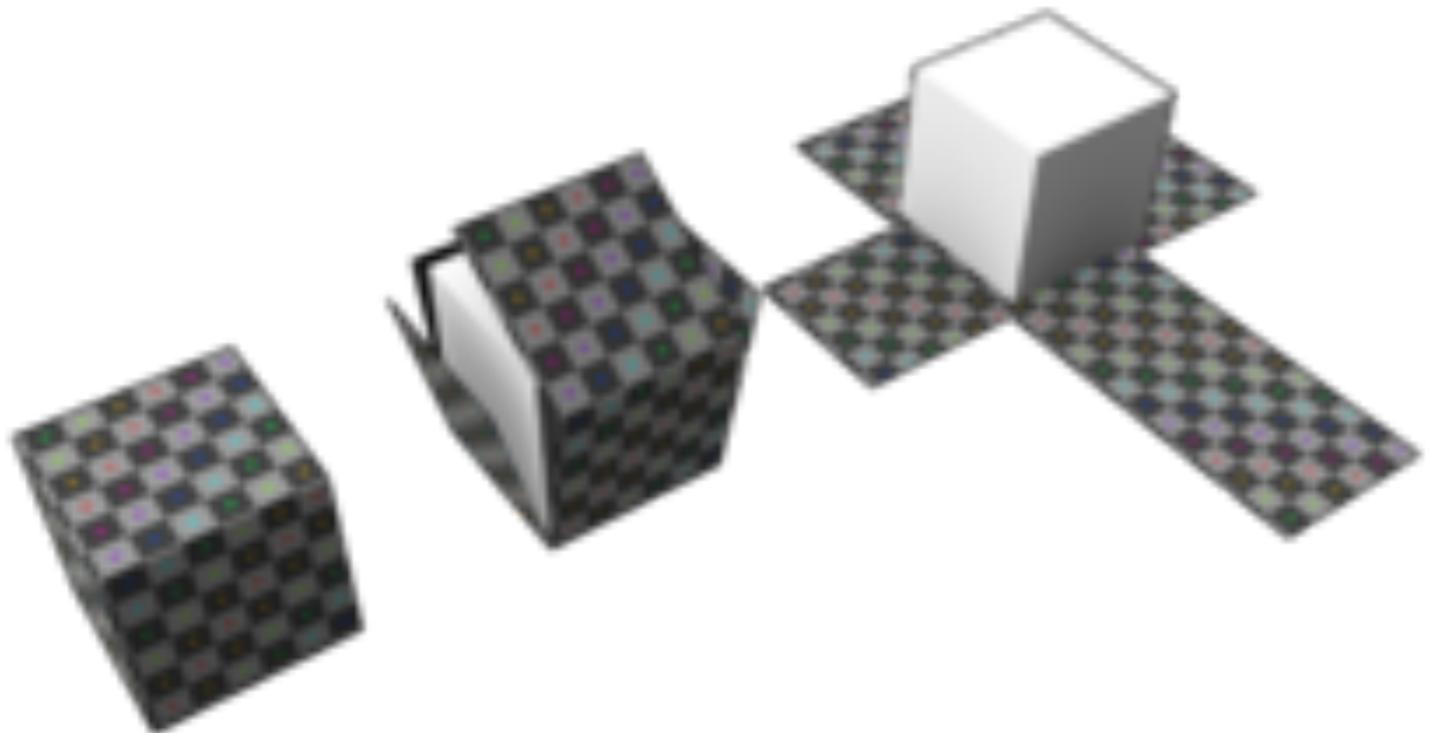
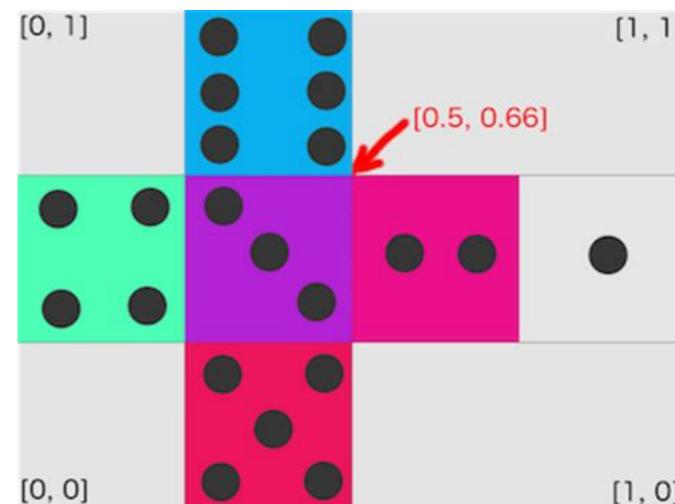
3. Recap. textures

- **Malles poligonals:**

Definir (u, v) a cada vèrtex

UV unwrapping:

mètode automàtic per a desplegar una superfície en l'espai paramètric UV



3. Recap. textures

- Procés de UV-unwrapping



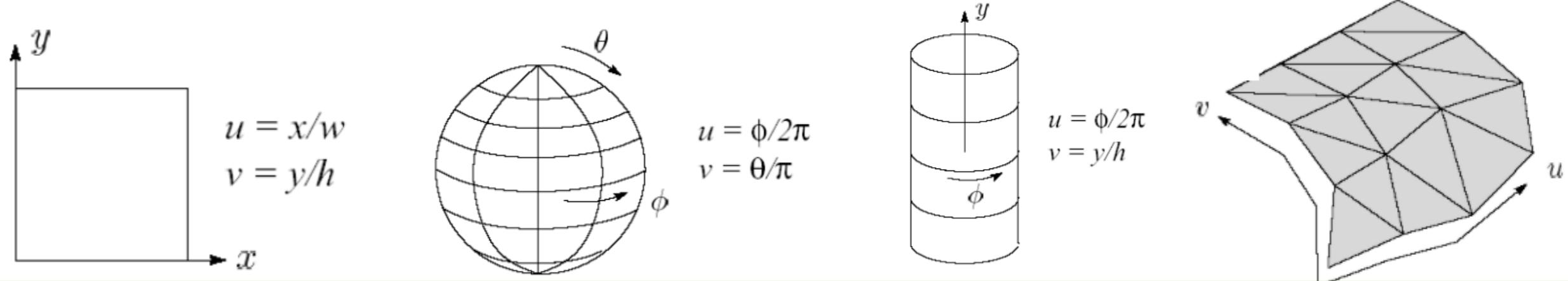
1. Retallat de la superfície en trossos
 2. Desplegament de la superfície en un pla
 3. Mapeig en una imatge
 4. Obtenció dels colors
- Utilitzat en els modeladors com Blender, Maya, etc.
 - Estan codificats en els .obj que es donen (veure el READ.ME)

3. Recap. textures

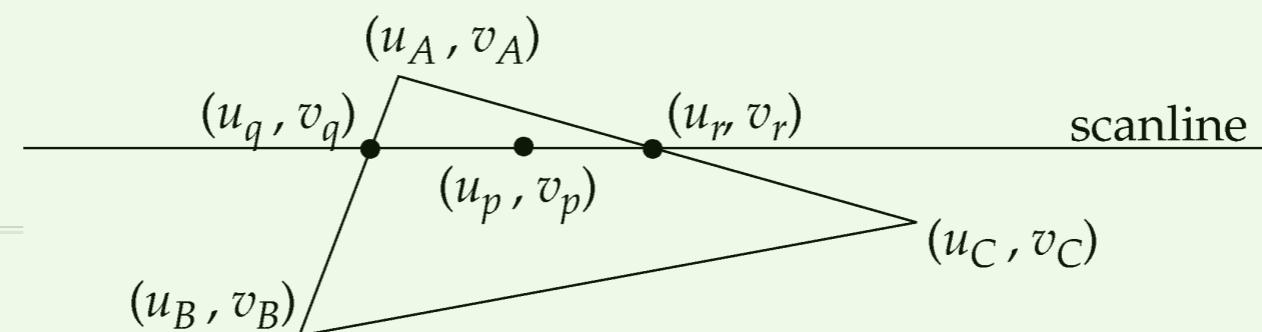
Funció de projecció: $\mathbf{A} : (x, y, z) \rightarrow (u, v)$

Les coordenades (x, y, z) d'un objecte es poden escriure en coordenades paramètriques (u, v) mitjançant la funció de projecció A:

1. **Per a cada vèrtex de cada polígon de l'objecte, es calculen les coordenades paramètriques (u, v)**

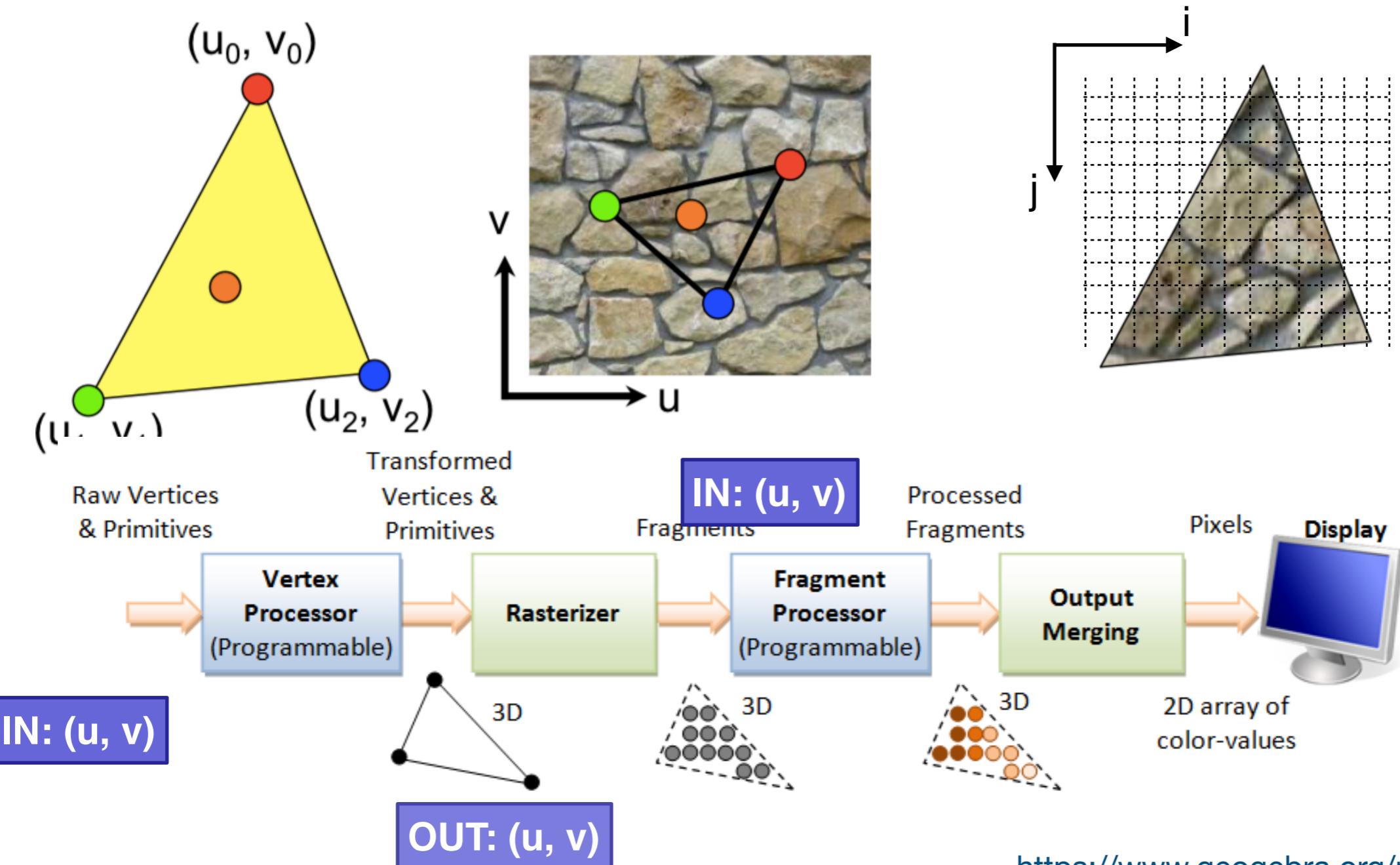


2. **Per a cada punt interior del polígon, es calculen les coordenades paramètriques a partir de les coordenades paramètriques dels vèrtexs extrems del polígon**

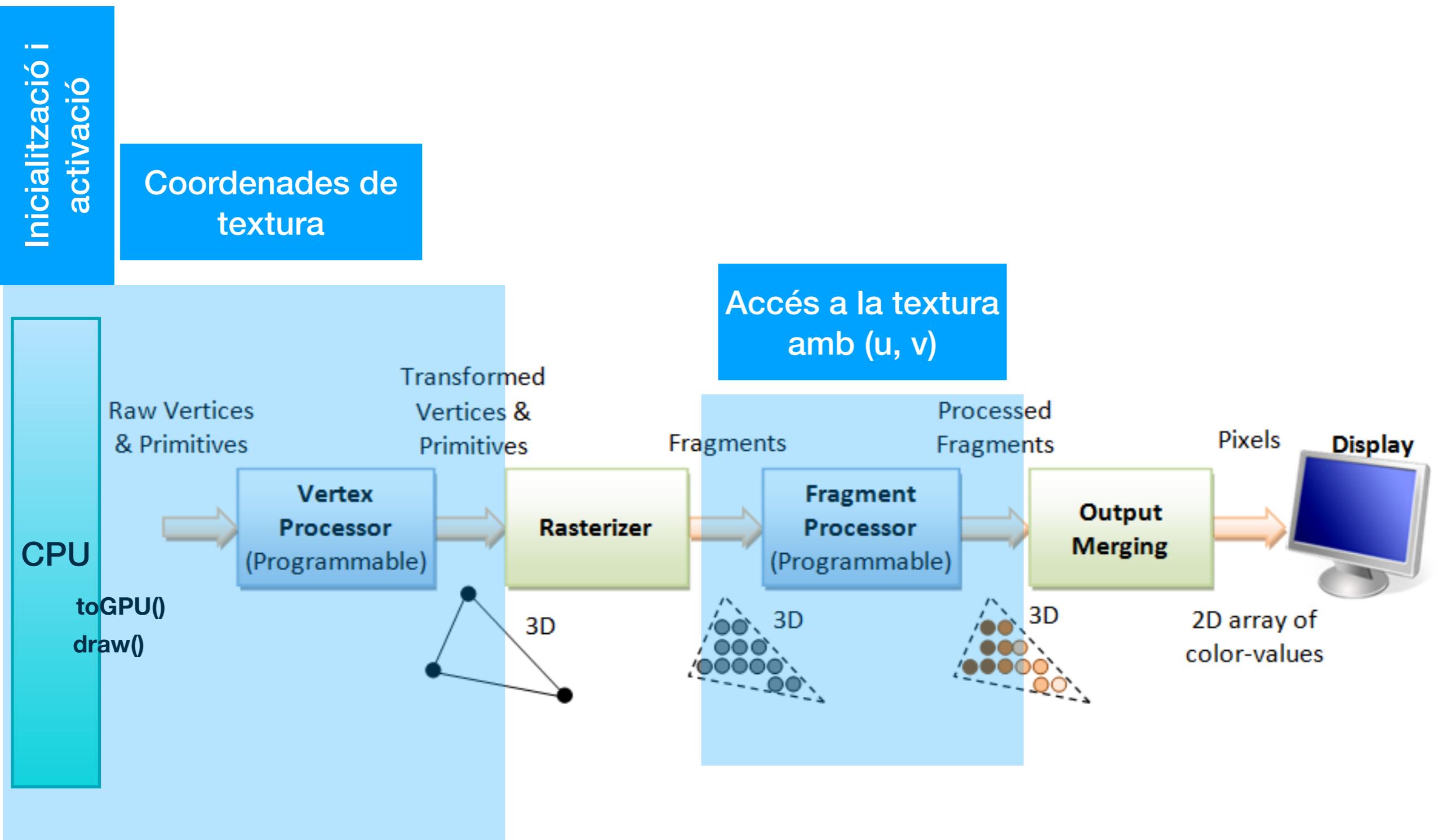


3. Textures: funció de projecció

- **Triangle:** $(x, y, z) \rightarrow (u, v)$



3. Recap. textures



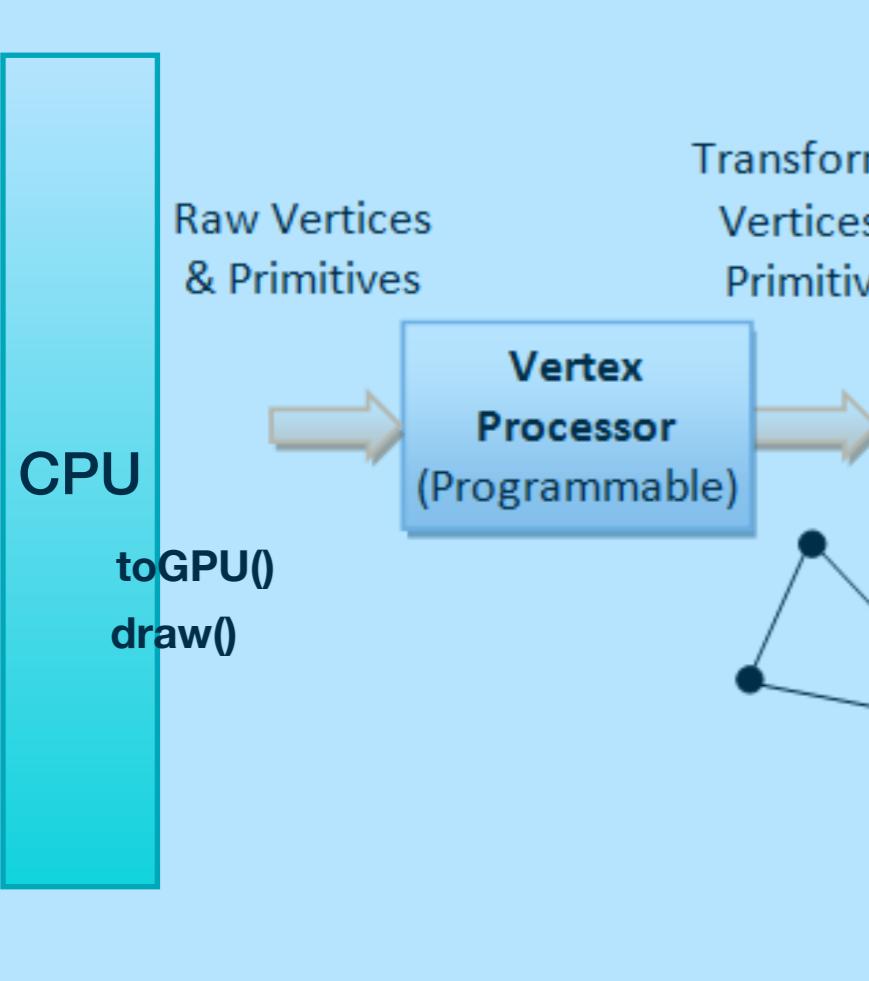
3. Recap. textures

Des d'on es crea la textura?

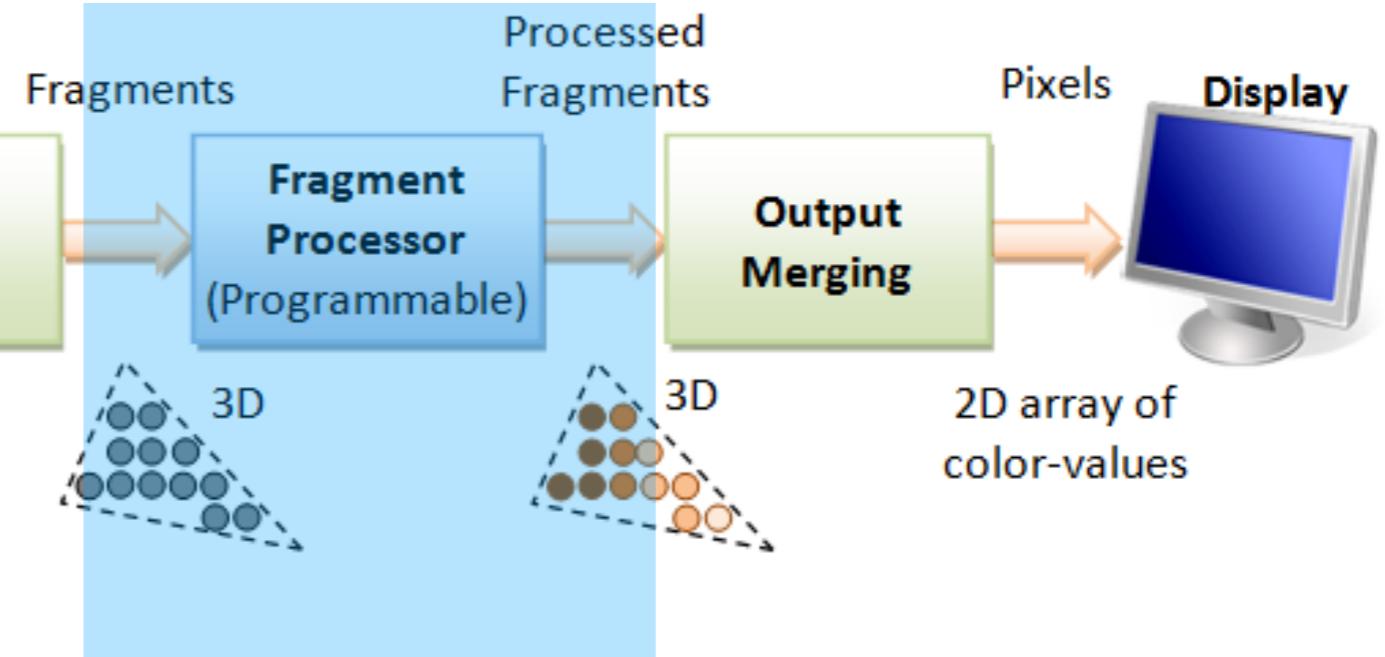
```
glActiveTexture(GL_TEXTURE0);  
texture[0] = new QOpenGLTexture(QImage(":/resources/textures/MonkeyTex.png"));
```

Inicialització i activació

Coordenades de textura



Accés a la textura amb (u, v)



Des d'on es passa la textura a la GPU?

```
texture[0]->bind(0);  
program->setUniformValue("texMap", 0);
```