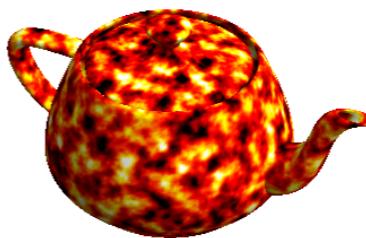


Sessió Setmana 12

GiVD 2022-23

Guió de la sessió

1. Planificació de la setmana
 2. Recap de shaders a la GPU
a la pràctica
 3. Problemes de shaders
 4. Introducció a textures a GL
-



1. Planificació de la setmana

- Veure el campus: Setmana 12



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

1. Planificació de la setmana

- Veure el campus: Setmana 12



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

1. Planificació de la setmana

Dates a escollir	Dilluns, 29/05 – a confirmar
	Dijous, 08/06 de 15:00 a 19:00 - a confirmar
	Divendres, 09/06 de 15:00 a 19:00

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 Study

[**Link a l'excel \(data limit 11.05 a les 23:59\)**](#)

1. Planificació Pràctica 2

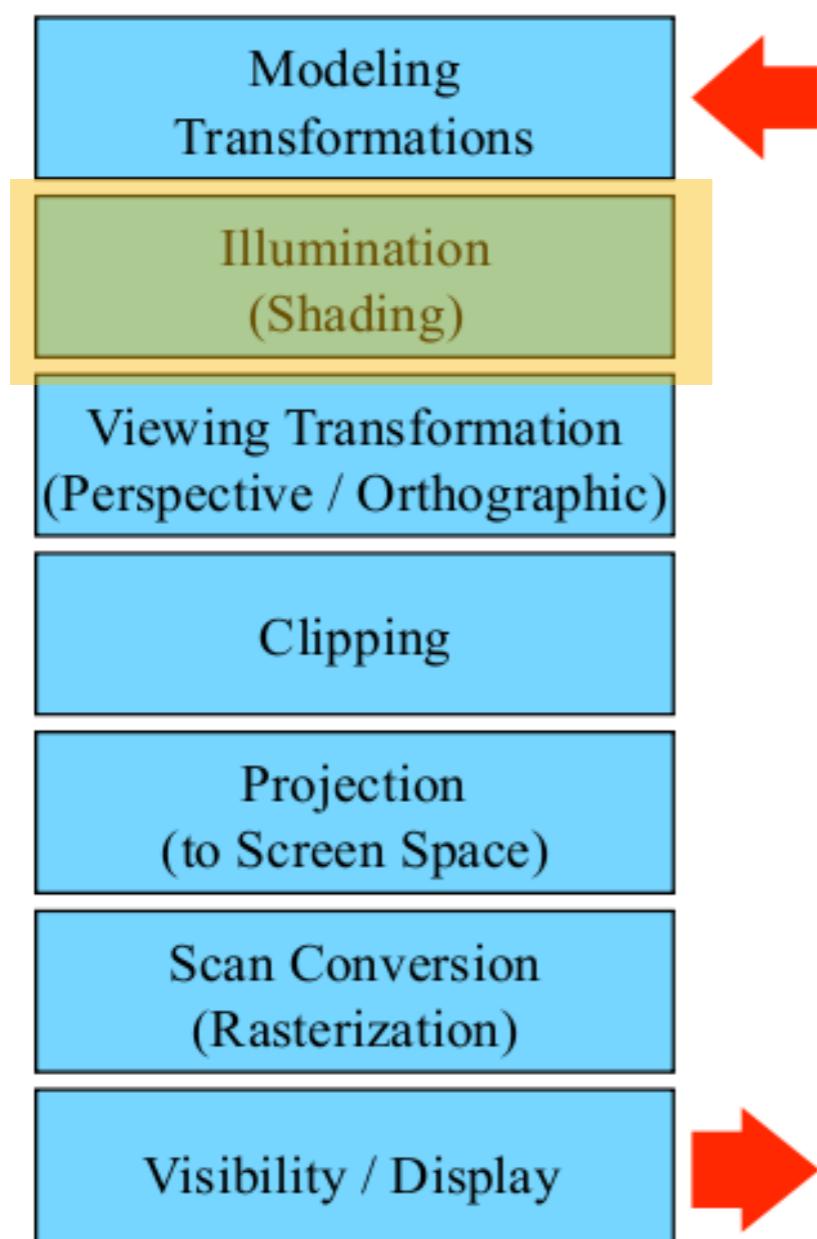
- Veure el campus: Pràctica 2

Maig 2023	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	Entrega P2. Entrevista i prova

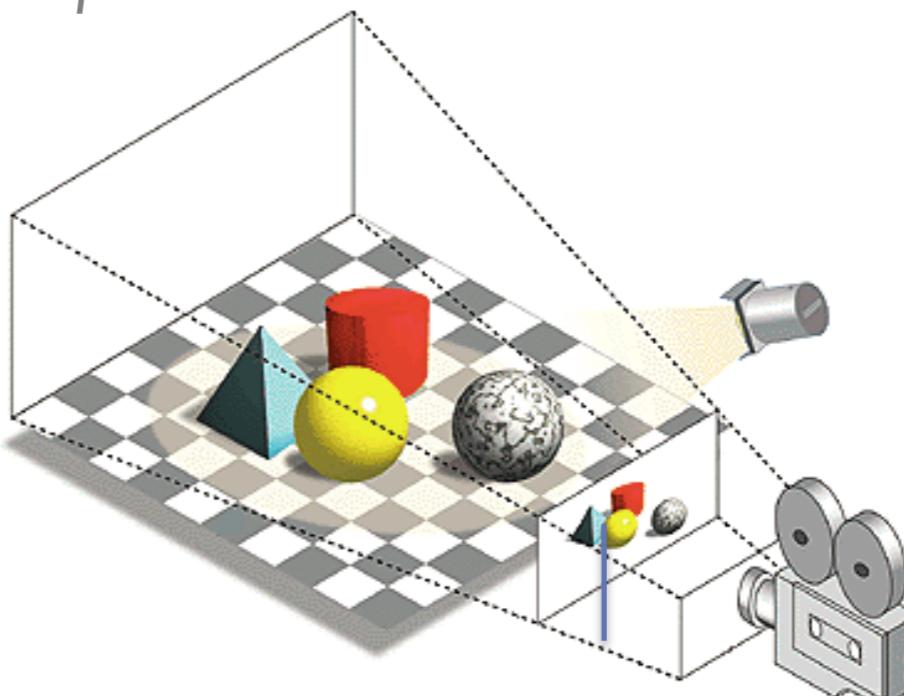
2. Recap de shaders i qüestions de la pràctica

2. Recap 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))

2. Recap Shading a la GPU

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

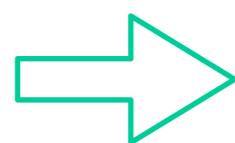
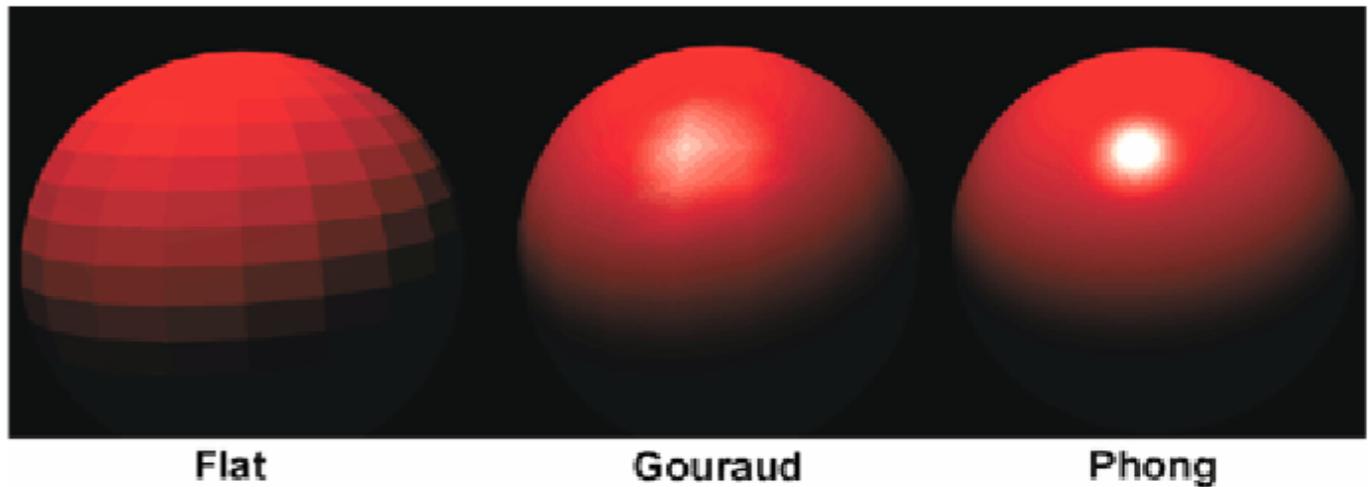
$$I_{total} = I_{a_{global}}K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d \max(L_i \cdot N, 0.0) + I_{s_i} K_s \max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$

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

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

Es necessiten:

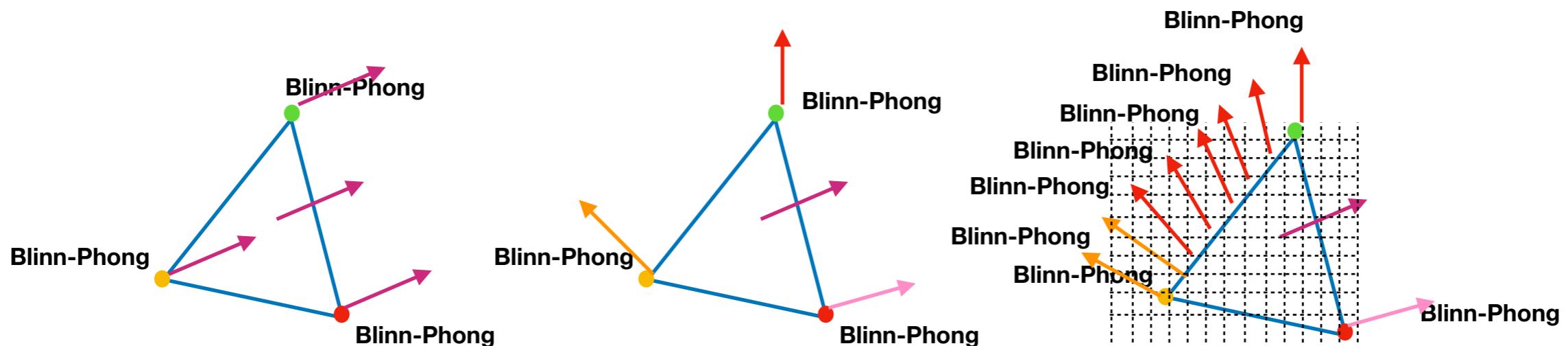
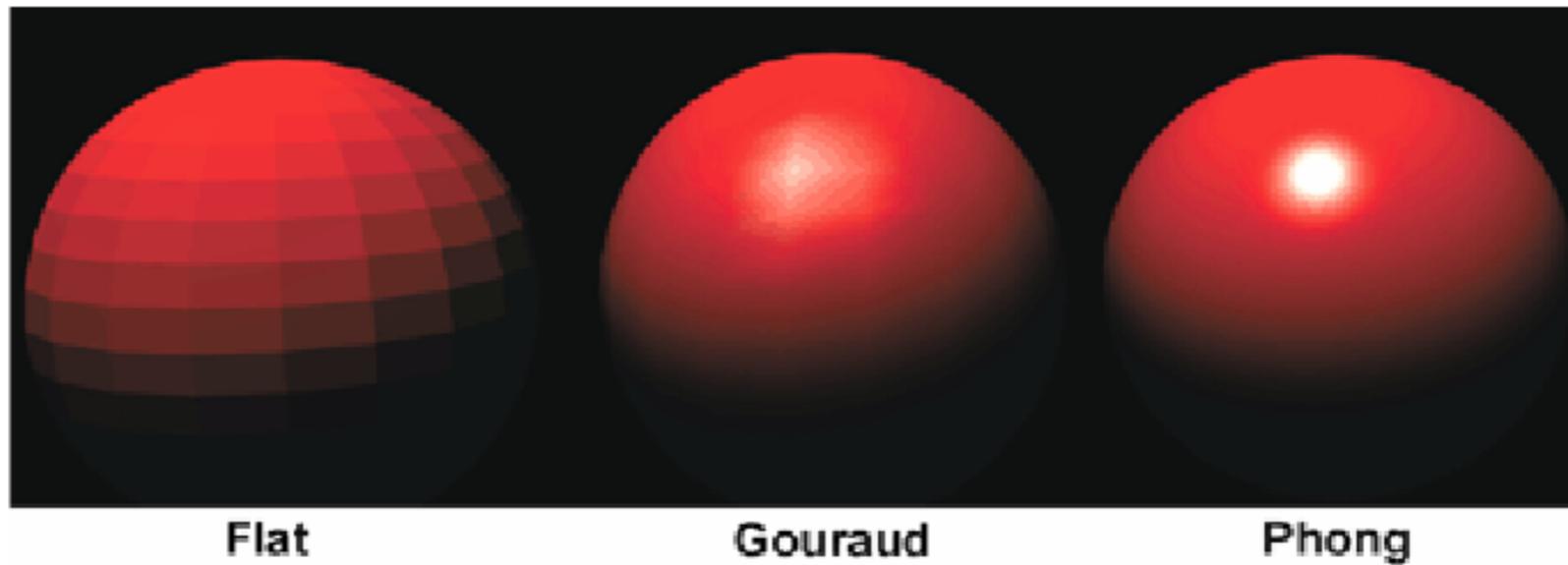
- les normals
- les llums
- la càmera
- les propietats dels materials



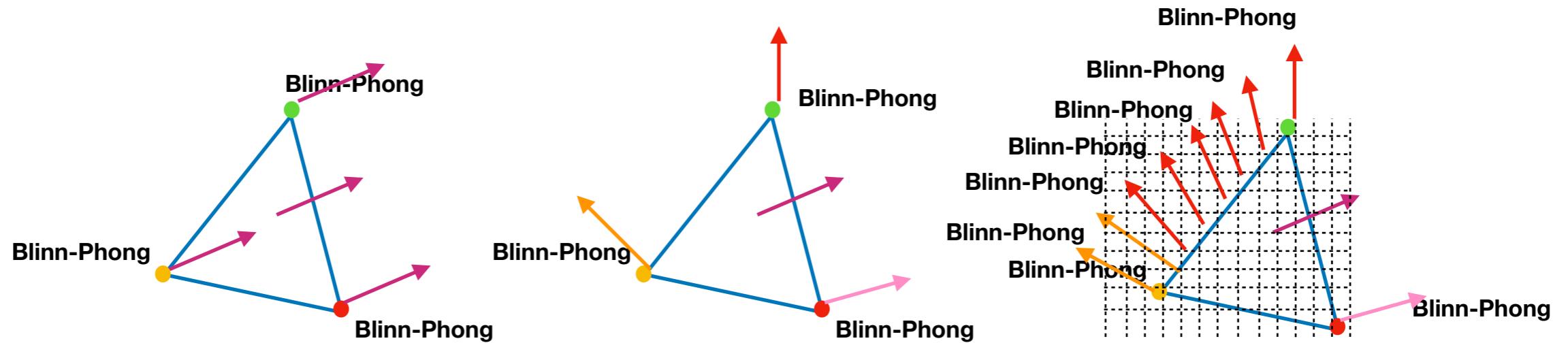
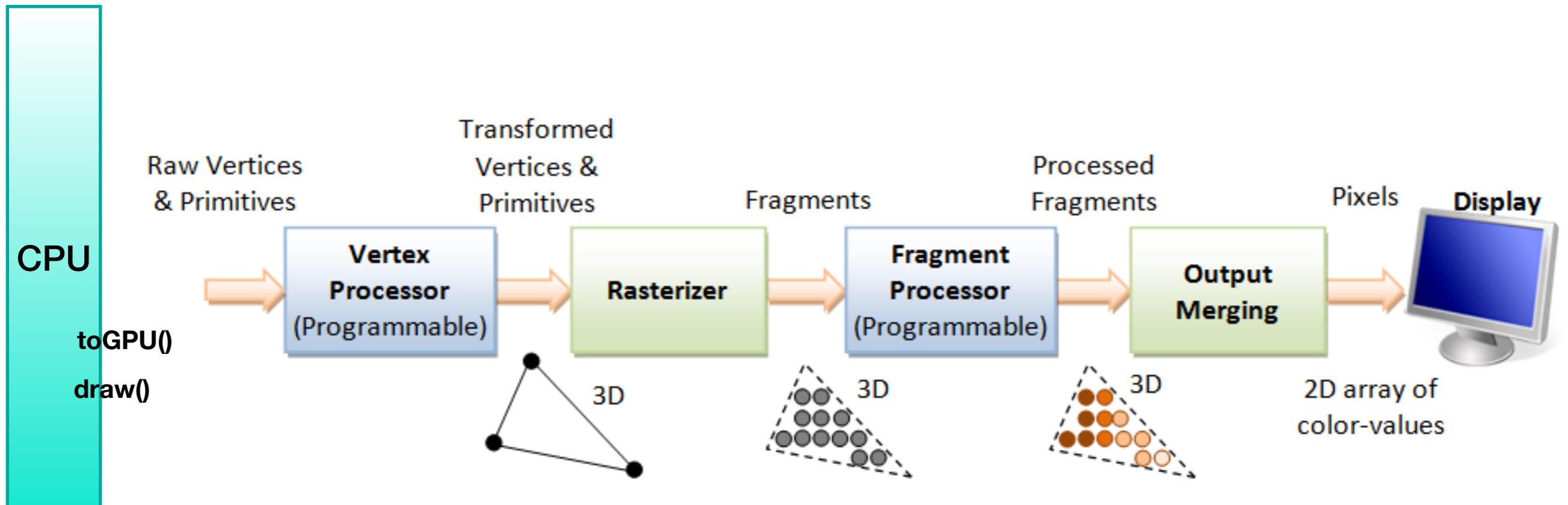
S'envia tota la informació als shaders

2. Recap Shading a la GPU

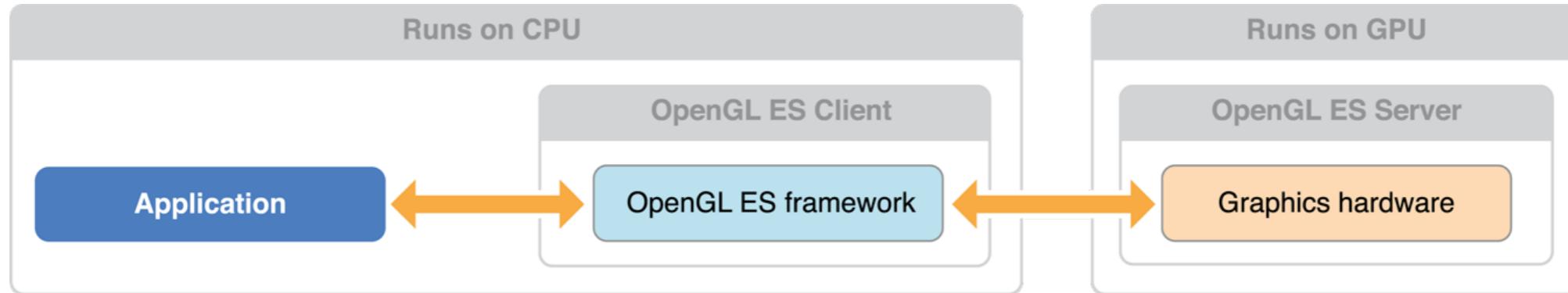
$$I_{total} = I_{a_{global}}K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d \max(L_i \cdot N, 0.0) + I_{s_i} K_s \max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$



2. Recap Shading a la GPU



2. Recap Shading a la 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;

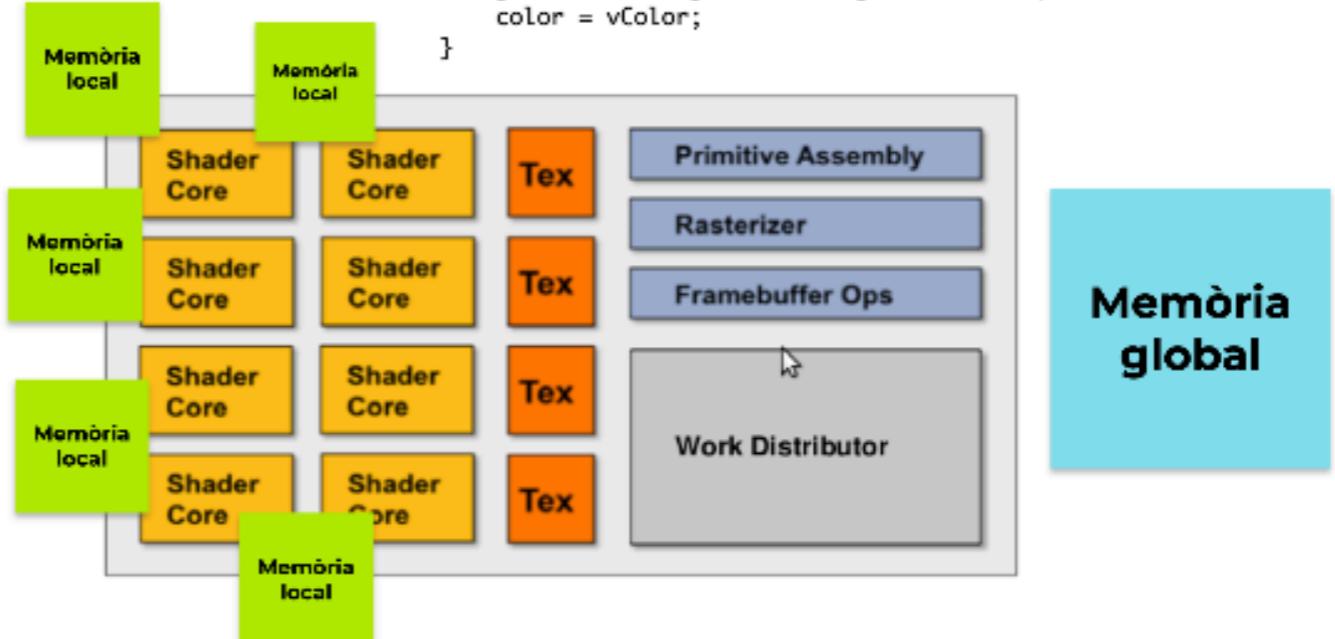
void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;
    color = vColor;
}
```

GLWidget

initializeGL()

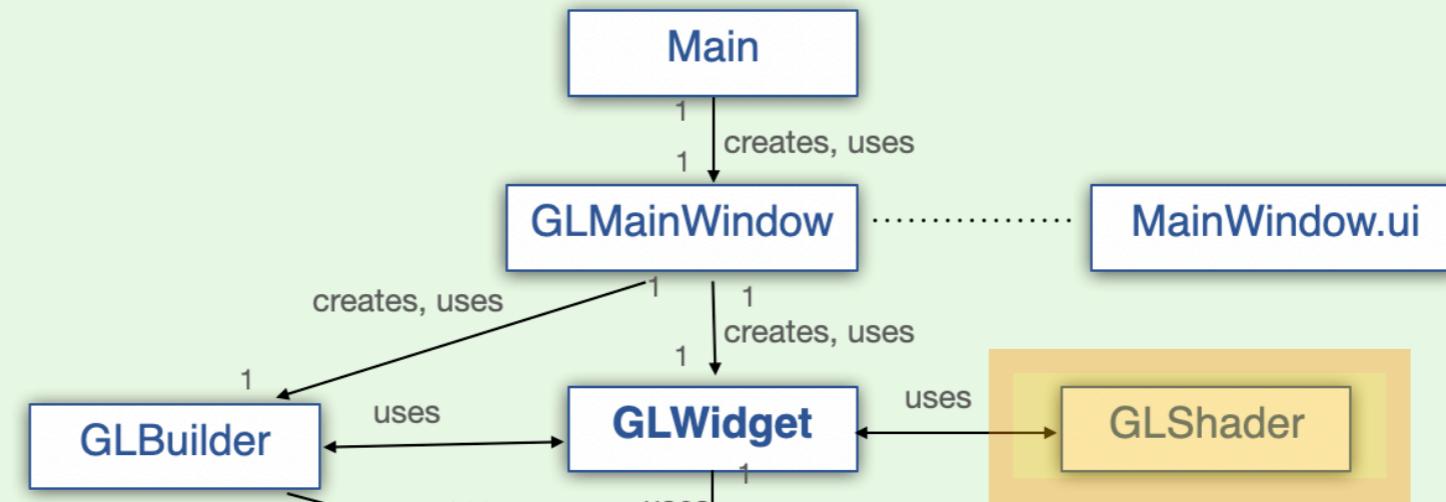
paintGL()

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



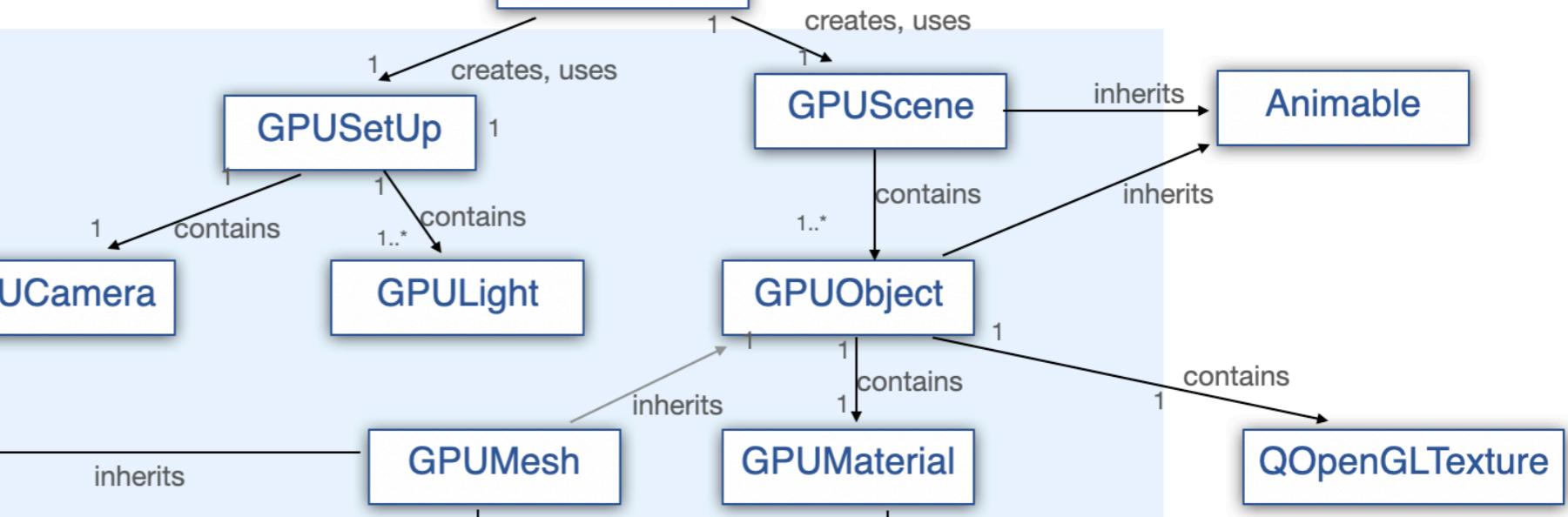
2. Recap Shading a la GPU

ViewGL



GPUConnections

GPUFactories*



PointLight

Light

Face

Mesh

Material

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

3. ACTIVITAT CPU-GPU

Activitat 1

CPU-to-GPU	Granularitat	Creació		Comunicació CPU-GPU			A nivell de vertex shader						
	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)	Memòria local o global						
Program (parell vertex i fragment shader) Programes a executar en el pipeline de la GPU													
Vèrtexs d'un objecte Conjunt de vèrtexs d'una malla triangular													
Normals d'un objecte Conjunt de normals d'una malla triangular													
Càmera Implica les matrius modelview i projection, i la posició de l'observador													
Llum global ambient	Scene	hardcoded? Fitxer?	via fitxer	hardcoded: GLWidget?	metode??	GPUSETUP	mètode i classe?						
Llums													
Material Propietats òptiques associades a un objecte	vertex	objecte				InitializeGL()	paintGL()						
	Free Sticky Notes	vèrtex	objecte	Scene	via fitxer	nomClasse	nomMetode	Altres	InitializeGL()	paintGL()	Altres	IN/OUT	uniform
												IN/OUT	uniform



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!

3. ACTIVITAT CPU-GPU



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!

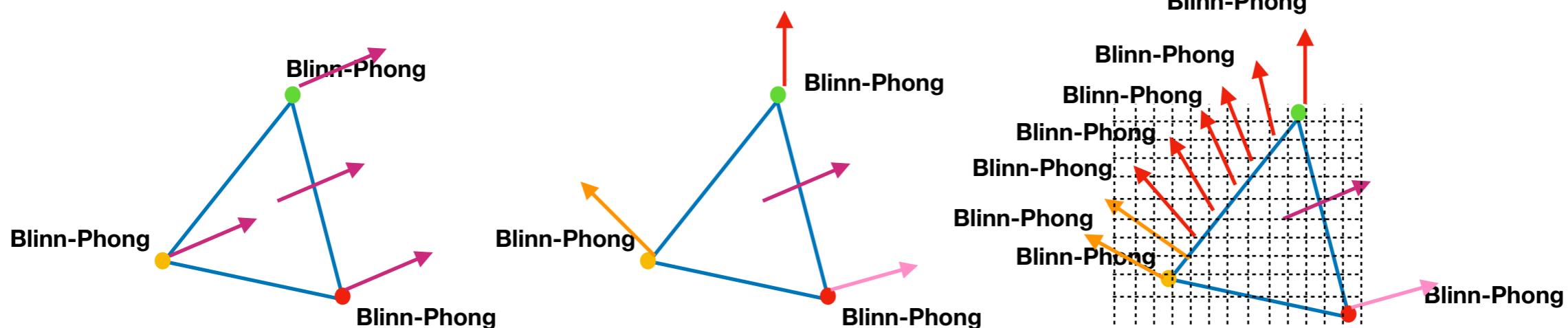
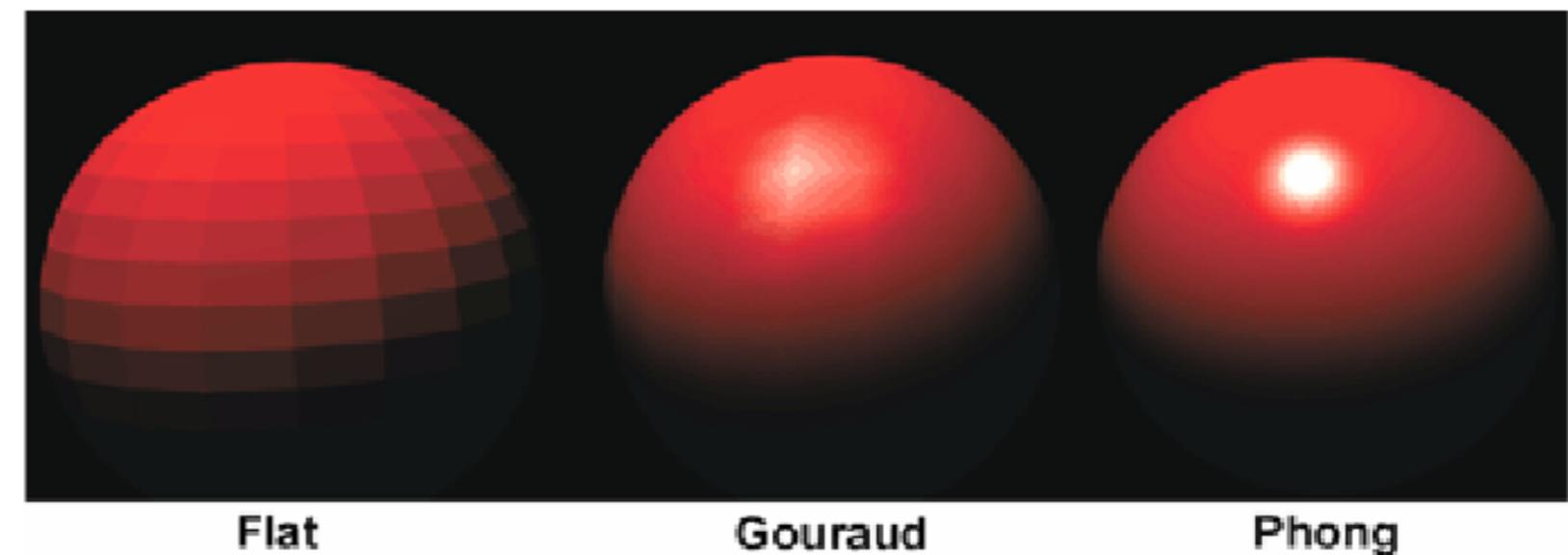
4. Shading a la GPU

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)	

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



2. Recap Shading a la GPU

- Com canviar de shaders?
- Repensar què s'ha d'enviar a la GPU i quan/on s'ha d'enviar (granularitat fina)
 - On s'inicialitzen les llums? On es passen a la GPU?
 - On s'inicialitzen els materials? Quan es passen a la GPU?
 - Com es passen les normals a la GPU i com s'usen?
 - el mètode **calculNormal** no s'ha d'implementar sinó es vol fer Flat shading

3. Problèmes de Shaders

Inputs /outputs de vertexs/fragment shaders

- Quina de les següents afirmacions és **CERTA**?
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, el *Vertex Shader* pot donar com a sortida el promig de les posicions dels tres vèrtexs del triangle.
 - El *Fragment shader* pot usar la normal interpolada
 - El *Vertex shader* pot donar un color de sortida per a ser usat pel Fragment shader
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, a qualsevol *Fragment shader* pot usar la posició del tercer vèrtex del triangle.

Inputs /outputs de vertexs/fragment shaders

- Quina de les següents afirmacions és **CERTA**?
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, el *Vertex Shader* pot donar com a sortida el promig de les posicions dels tres vèrtexs del triangle.
 - El *Fragment shader* pot usar la normal interpolada
 - El *Vertex shader* pot donar un color de sortida per a ser usat pel Fragment shader
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, a qualsevol *Fragment shader* pot usar la posició del tercer vèrtex del triangle.

Inputs /outputs de vertexs/fragment shaders

- Quina de les següents afirmacions és **CERTA**?
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, el *Vertex Shader* pot donar com a sortida el promig de les posicions dels tres vèrtexs del triangle.
 - El *Fragment shader* pot usar la normal interpolada
 - El *Vertex shader* pot donar un color de sortida per a ser usat pel Fragment shader
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, a qualsevol *Fragment shader* pot usar la posició del tercer vèrtex del triangle.

Inputs /outputs de vertexs/fragment shaders

- Quina de les següents afirmacions és **CERTA**?
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, el *Vertex Shader* pot donar com a sortida el promig de les posicions dels tres vèrtexs del triangle.
 - El *Fragment shader* pot usar la normal interpolada
 - El *Vertex shader* pot donar un color de sortida per a ser usat pel Fragment shader
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, a qualsevol *Fragment shader* pot usar la posició del tercer vèrtex del triangle.

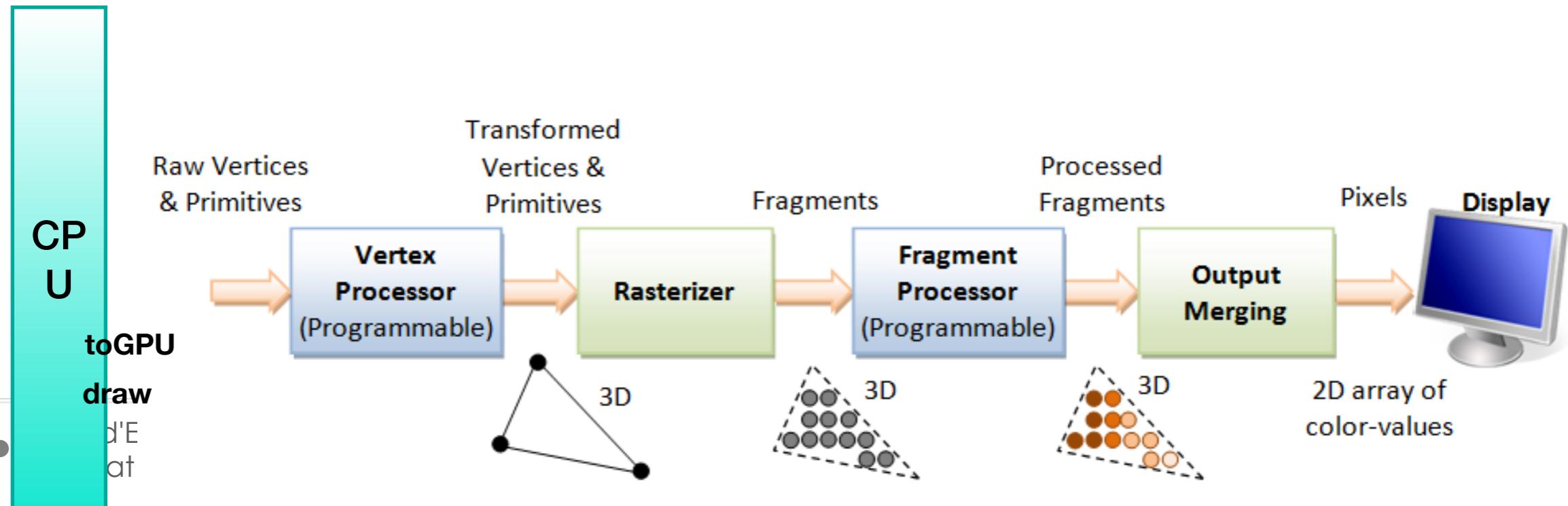
Inputs /outputs de vertexs/fragment shaders

- Quina de les següents afirmacions és **CERTA**?
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, el *Vertex Shader* pot donar com a sortida el promig de les posicions dels tres vèrtexs del triangle.
 - El *Fragment shader* pot usar la normal interpolada
 - El *Vertex shader* pot donar un color de sortida per a ser usat pel Fragment shader
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, a qualsevol *Fragment shader* pot usar la posició del tercer vèrtex del triangle.

Inputs /outputs de vertexs/fragment shaders

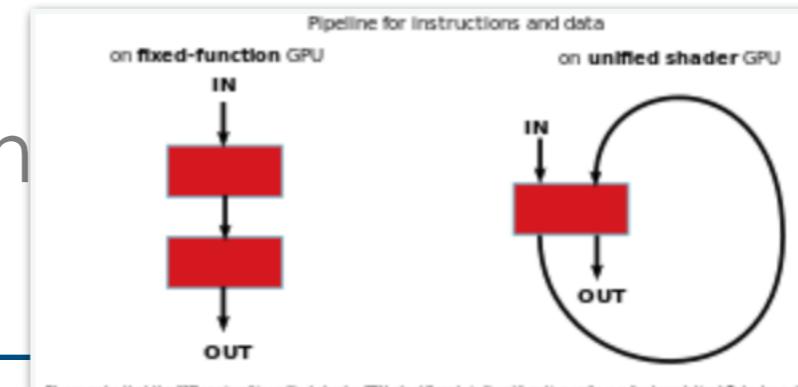
- Quina de les següents afirmacions és **CERTA**?
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, el *Vertex Shader* pot donar com a sortida el promig de les posicions dels tres vèrtexs del triangle.
 - **El Fragment shader pot usar la normal interpolada**
 - **El Vertex shader pot donar un color de sortida per a ser usat pel Fragment shader**
 - En el cas de visualitzar un triangle, segons el mètode `toGPU()` de la pràctica, a qualsevol *Fragment shader* pot usar la posició del tercer vèrtex del triangle.

- **Qüestions obertes:** Què podria suposar una millora en el pipeline d'OpenGL en quant a **eficiència**? (no totes són certes)
 - Invertir l'ordre del *fragment shader* i el *vertex shader*
 - Fer el test de Z-Buffer abans de executar el *fragment shader*, fent avortar l'execució d'aquell thread si falla el test de z
 - Fer que els processadors puguin ser *vertexs* i *fragment shaders*
 - Dividir cada triangle en 4 triangles més petits i visualitzar aquests 4



Pipeline integrat a OpenGL

- **Qüestions obertes:** Què podria suposar una millora en el pipeline d'OpenGL? (no totes són certes)
 - Invertir l'ordre del *fragment shader* i el *vertex shader*
 - **Fer el test de Z-Buffer abans de executar el *fragment shader*, fent abortar l'execució d'aquell thread si falla el test de z (early z)**
 - **Fer que els processadors puguin ser *vertexs* i *fragment shaders* (unified shaders)**
 - Dividir cada triangle en 4 triangles més petits per visualitzar aquests 4



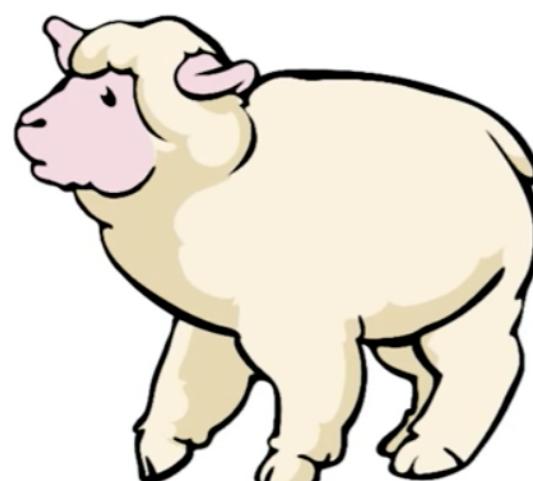
Totes dues opimitzacions estan integrades a OpenGL:

Early z: https://www.khronos.org/opengl/wiki/Early_Fragment_Test

Unified shaders: https://wikimili.com/en/Unified_shader_model

Problema: Toon Shading

- Tècnica no-fotorealista
- Usa pocs colors (tons) en funció de l'angle entre la llum i la normal de l'objecte
- Els tons s'escullen en funció del *cosinus* de l'angle entre la *llum* i la *normal* a la superfície.
- Si la **normal** és propera a la **direcció de la llum**, s'escull un to més clar. Si és llunyana, s'escull un to més fosc.
- On s'implementa aquest shader? On té menys cost?

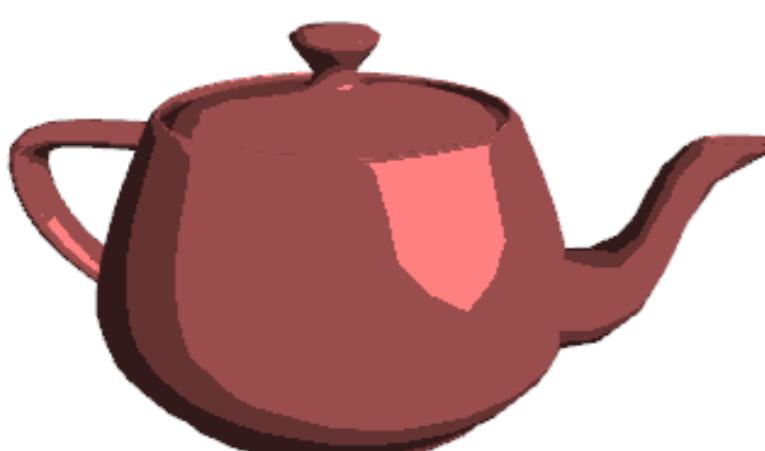


Problema: Toon Shading

Toon shading: Implementable en el fragment shader directament: interpolant el **color** o les normals.

```
// vertex shader
uniform vec3 lightDir;
layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vNormal;
out float intensity;
void main()
{
    intensity = dot(lightDir, vnormal);
    ...
}

// fragment shader
in float intensity;
void main()
{
    vec4 color;
    if (intensity > 0.95)
        color = vec4(1.0,0.5,0.5,1.0);
    else if (intensity > 0.5)
        color = vec4(0.6,0.3,0.3,1.0);
    else if (intensity > 0.25)
        color = vec4(0.4,0.2,0.2,1.0);
    else
        color = vec4(0.2,0.1,0.1,1.0);
    gl_FragColor = color;
}
```



Problema: Toon Shading

Toon shading: Implementable en el fragment shader directament: interpolant el color o les **normals**.

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

void main()
{
    normal = vNormal;
}
```

```
// fragment shader
uniform vec3 lightDir;
in vec3 normal;

void main()
{
    float intensity;
    vec4 color;
    intensity = dot(lightDir, normalize(normal));

    if (intensity > 0.95)
        color = vec4(1.0, 0.5, 0.5, 1.0);
    else if (intensity > 0.5)
        color = vec4(0.6, 0.3, 0.3, 1.0);
    else if (intensity > 0.25)
        color = vec4(0.4, 0.2, 0.2, 1.0);
    else
        color = vec4(0.2, 0.1, 0.1, 1.0);
    gl_FragColor = color;
}
```



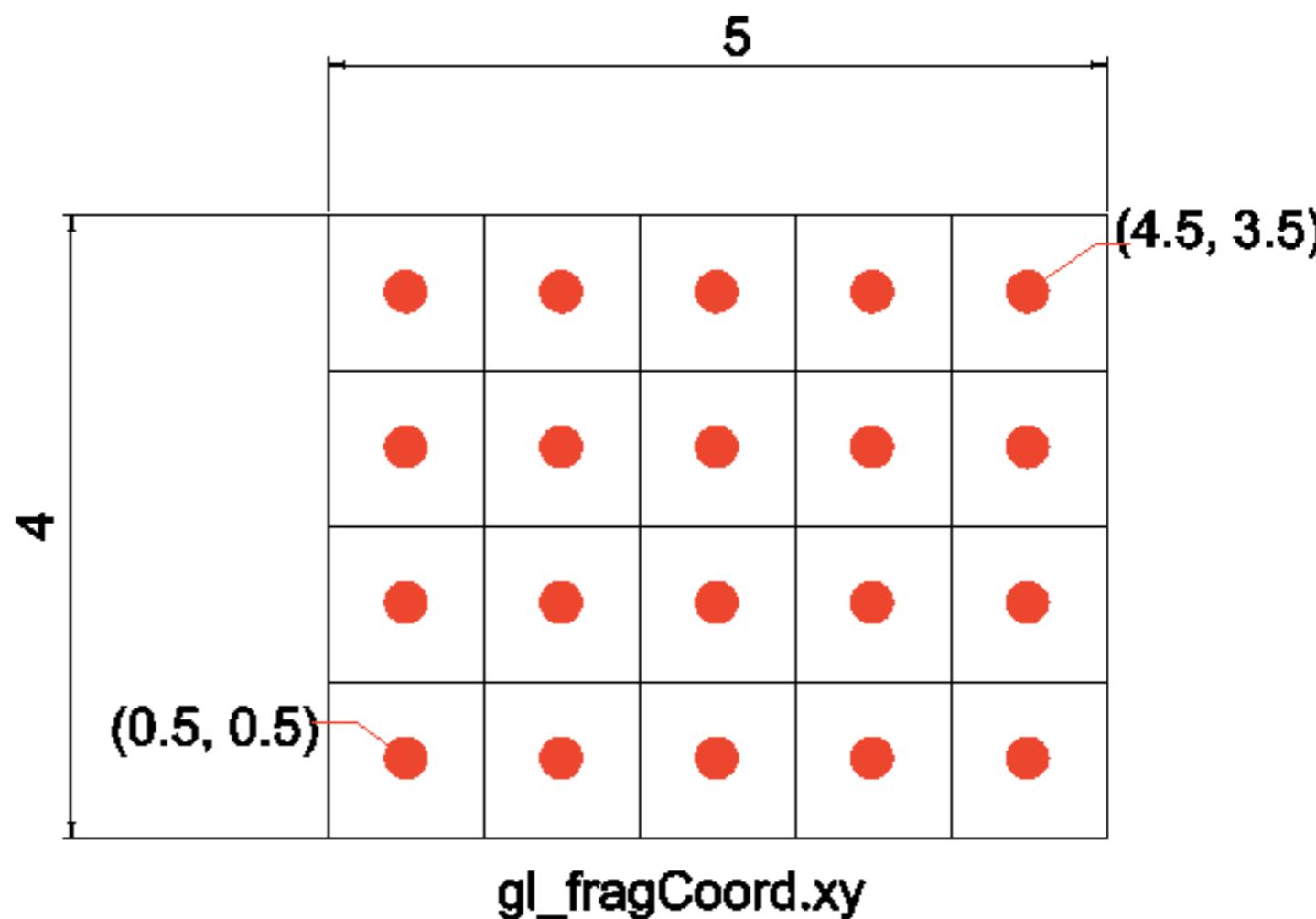
Problema: Fent un target en un cercle verd

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 càlcul? Amb quines coordenades?



Problema: Fent un target en un cercle verd

Target amb cercle verd:



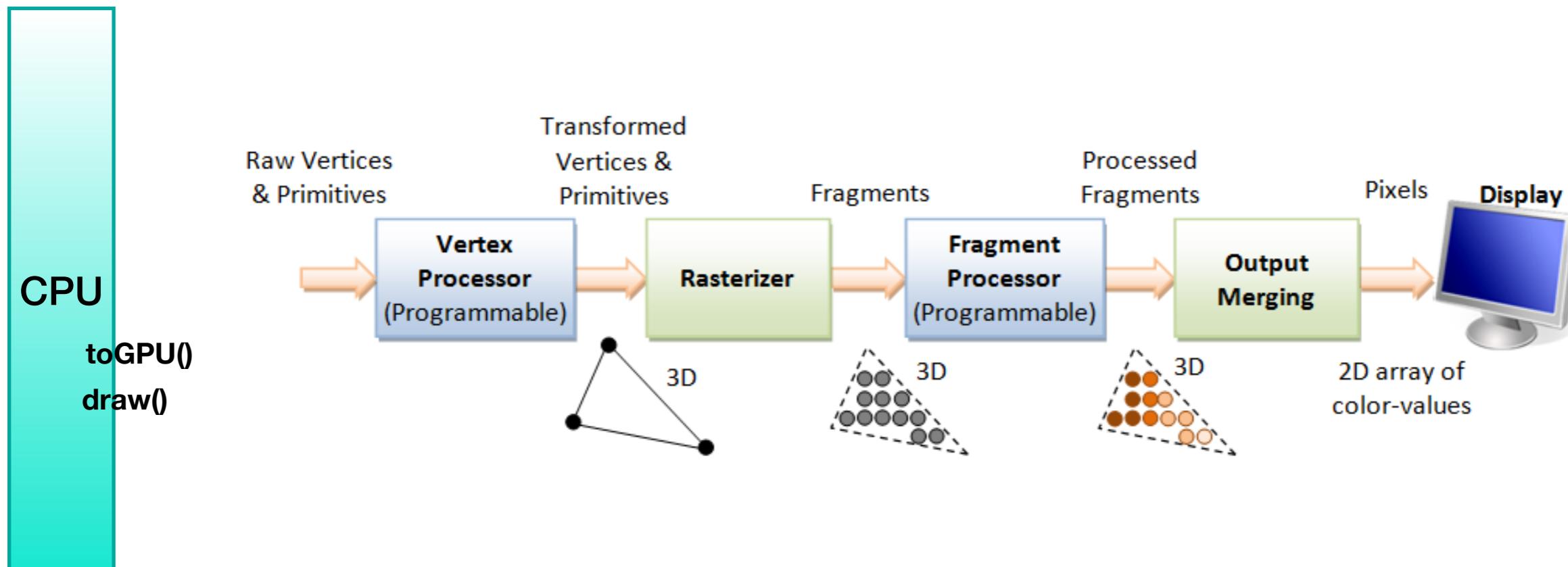
Problema: La tempesta de Fornite

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

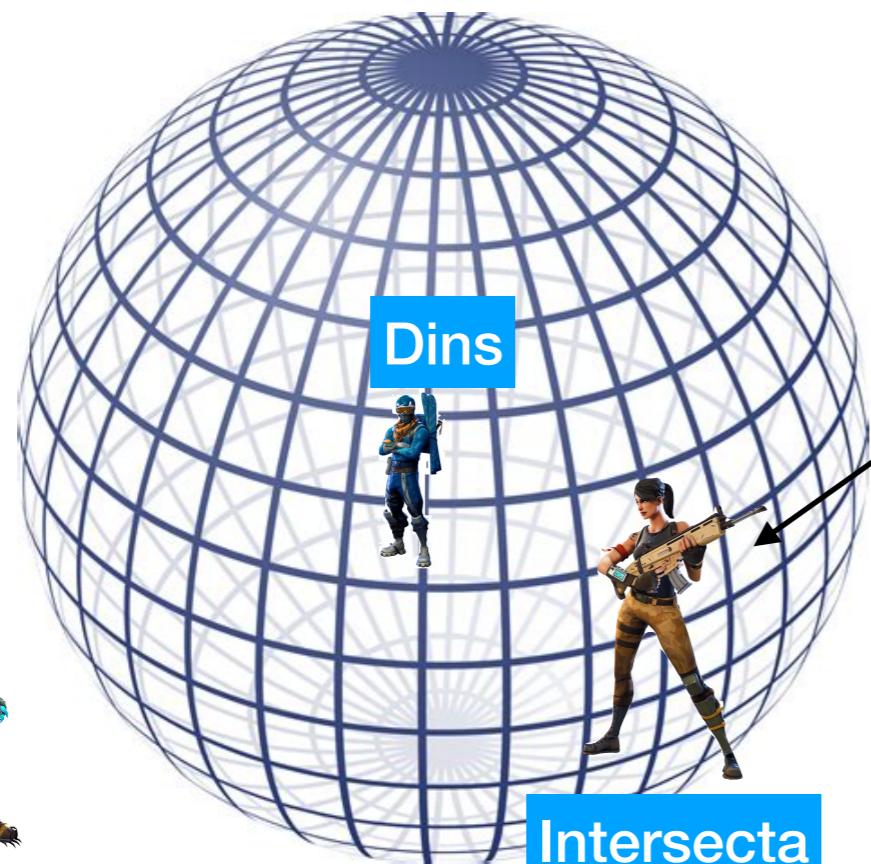
La Tempesta de Fornite: Durant el joc de Fornite, existeix una tempesta que afecta a una regió exterior d'una esfera centrada en el $(0,0,0)$ de món de radi 9000. Les regions afectades es visualitzen de colors blaus amb *Gouraud Shading* i les que no estan afectades es veuen en *Phong Shading*.



Problema: La tempesta de Fornite

La Tempesta de Fornite:

Quantes parelles de vertex-fragment shader els associem?



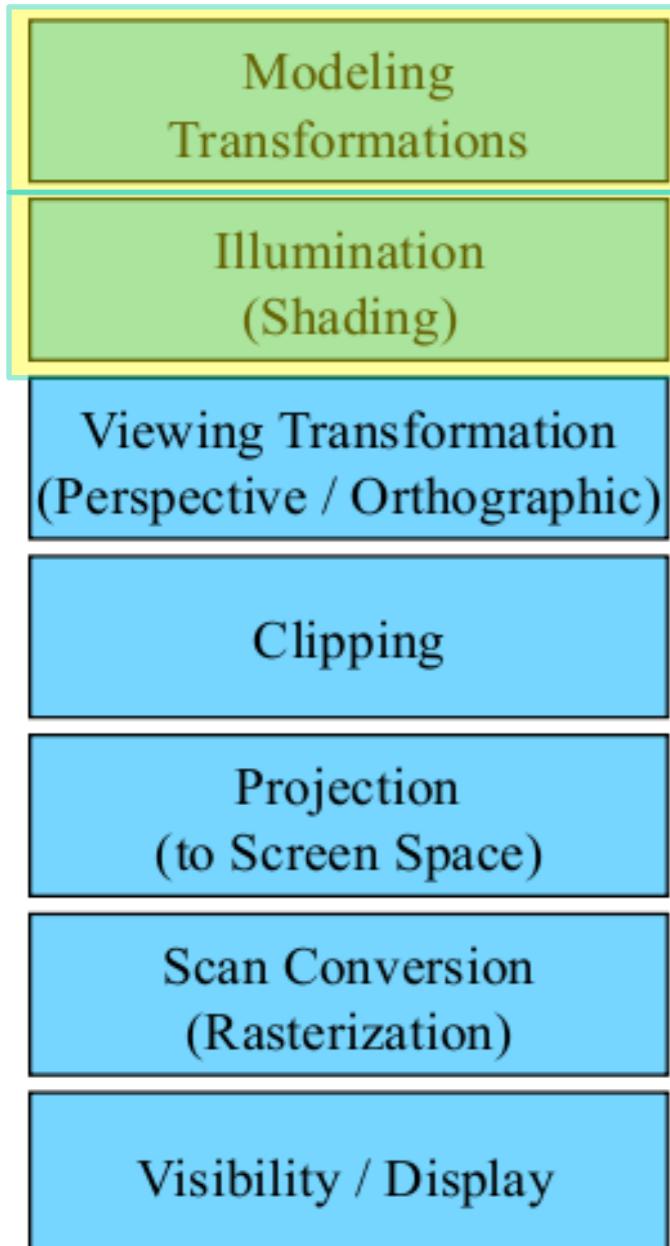
Problema:

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

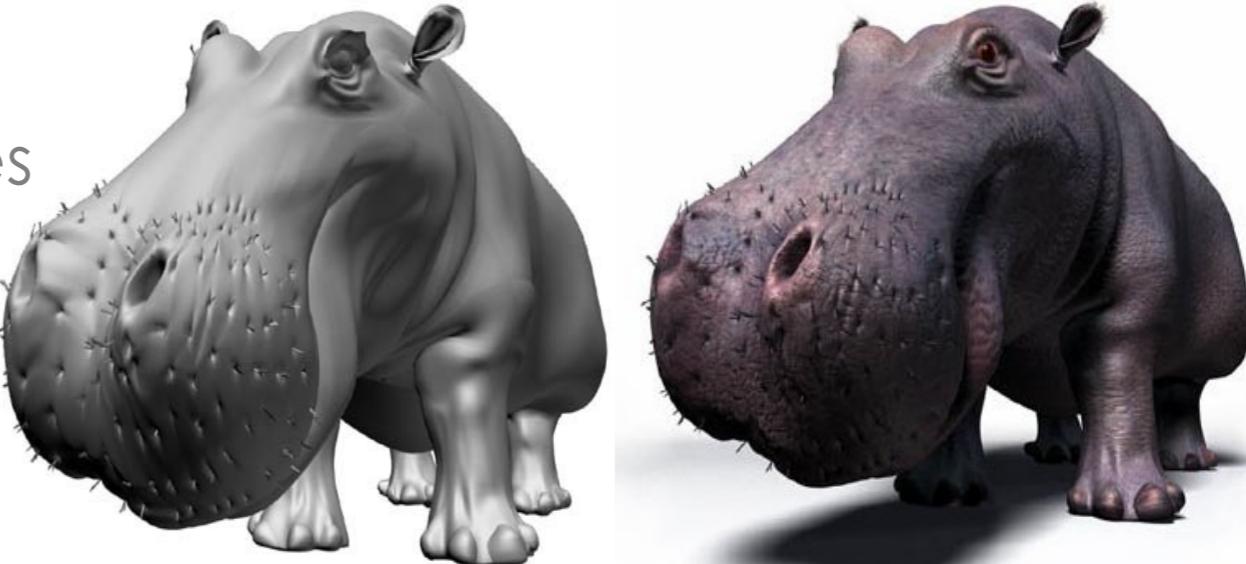
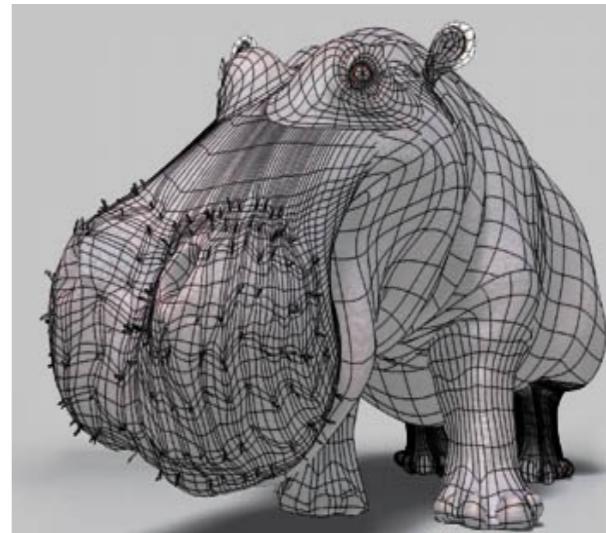
4. Introducció a textures

4. Textures

Objectiu: incrementar el grau de realisme amb petits detalls a la superfície. Com es pot aconseguir?



- en la fase de **modelatge**: incrementant punts i triangles a la superfície
- en la fase de **shading**: ús de materials i textures

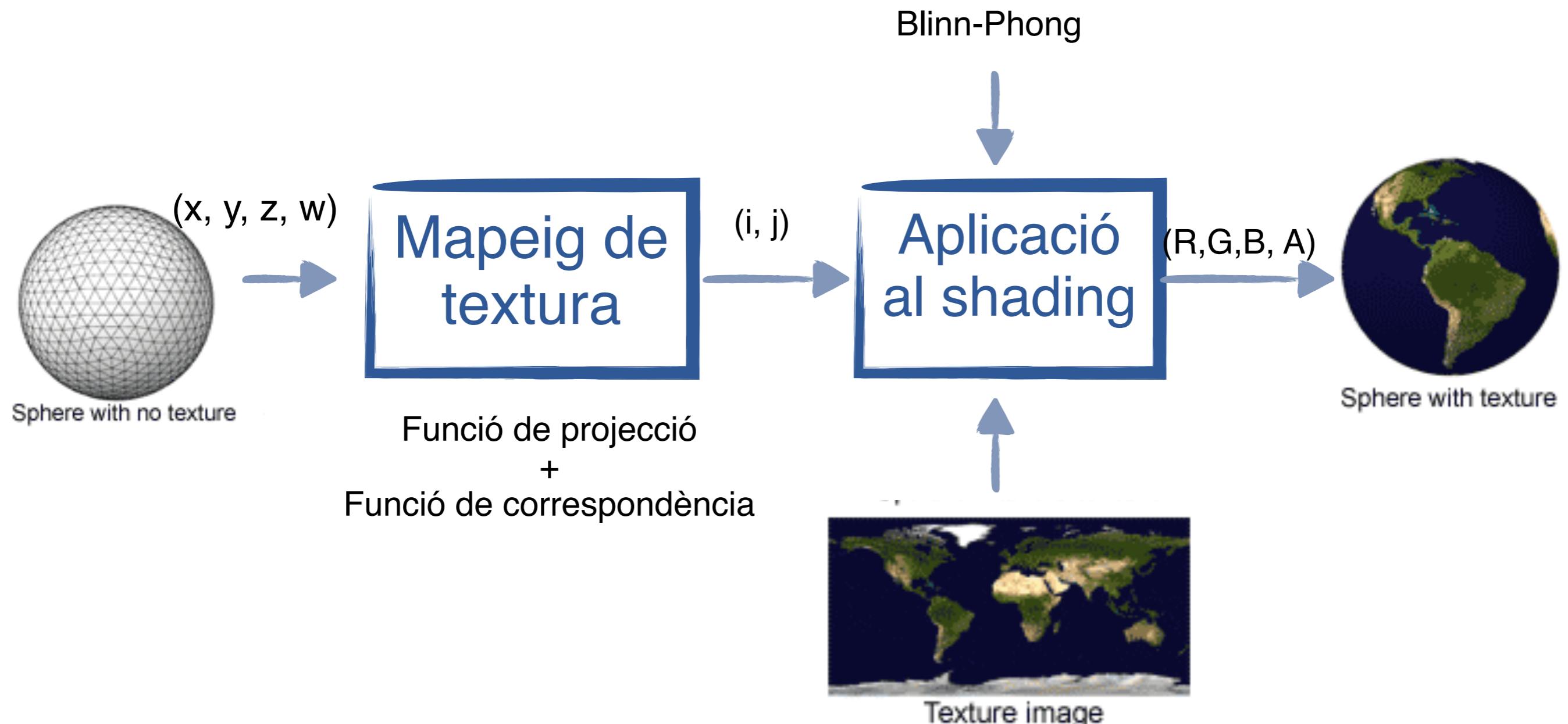


Els objectes reals no són uniformes en termes de color i normals: existeixen detalls amb altres freqüències que no són directament modelables.

4. Textures

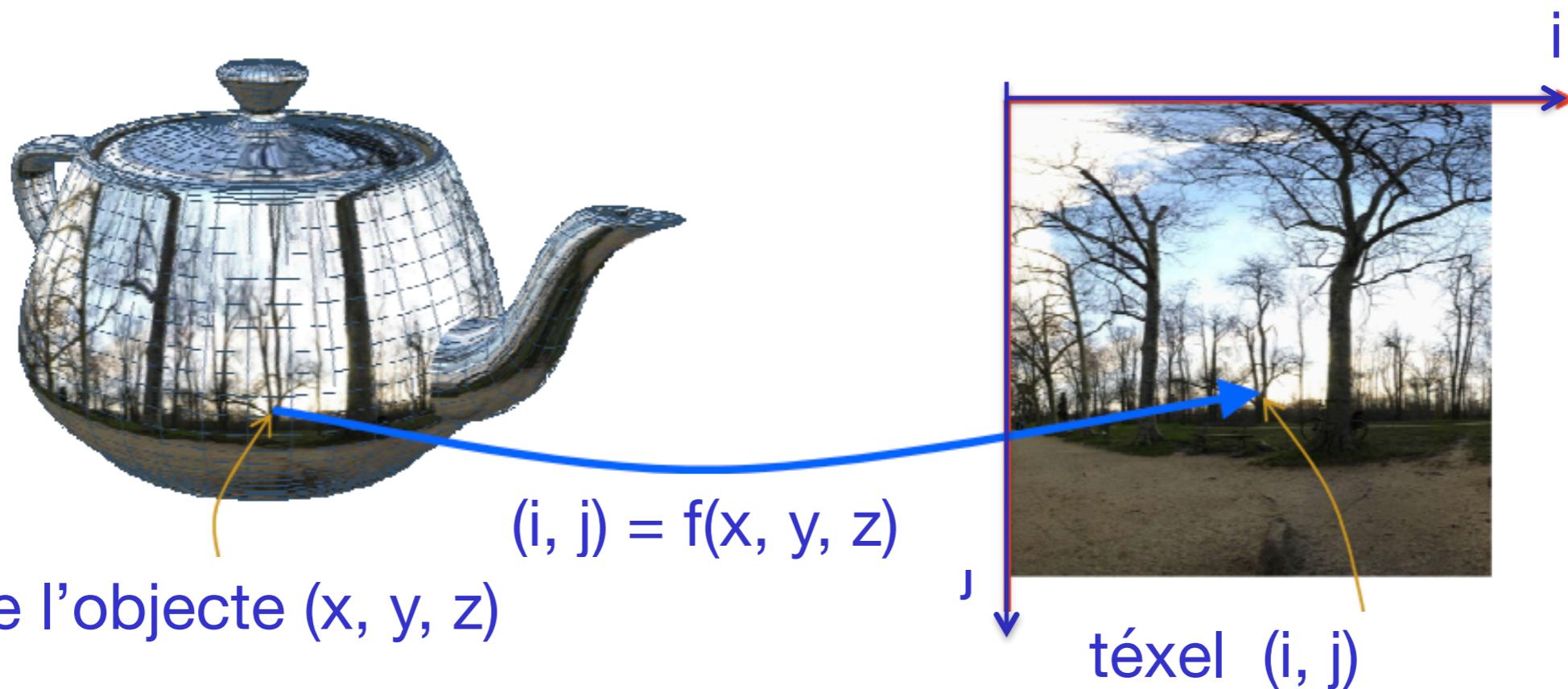
Com s'apliquen les textures? (tema 2h)

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



4. Textures: Mapeig

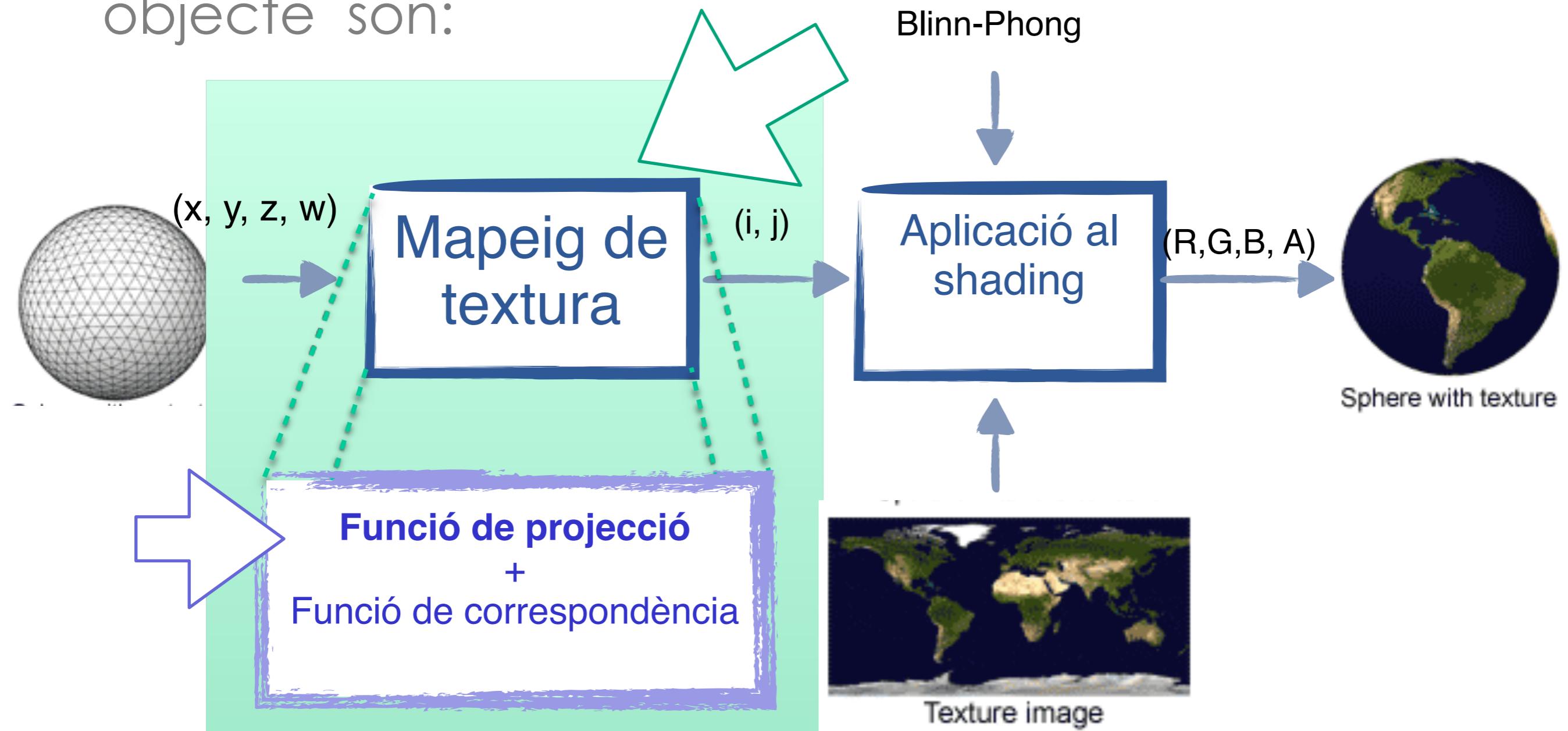
- 'Enganxar' o fer el **mapping** d'una textura sobre un objecte consisteix en definir una funció unívoca (*funció de mapping*) que a cada punt d'un objecte (x, y, z) li fa corresponder un punt d'una imatge (i, j).
- El mapping pot ser **directa** o en **dues fases**



4. Textures

Com s'apliquen les textures? (tema 2b)

Les etapes a realitzar per a aplicar textures a un objecte són:

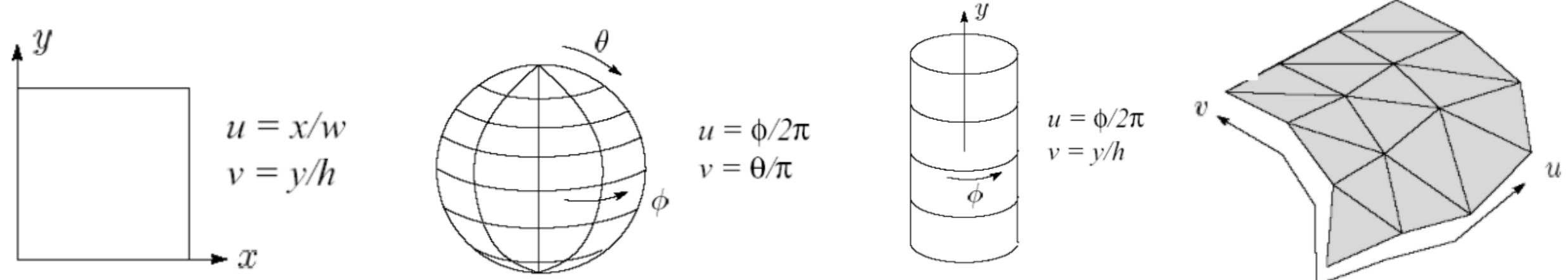


4. Textures: funció de projecció

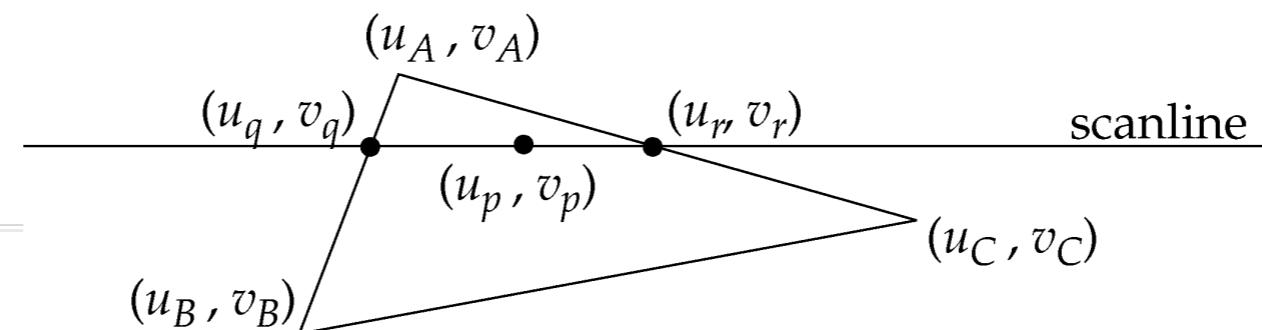
Funció de projecció: $\mathbf{A} : (x, y, z) \rightarrow (u, v)$

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)**

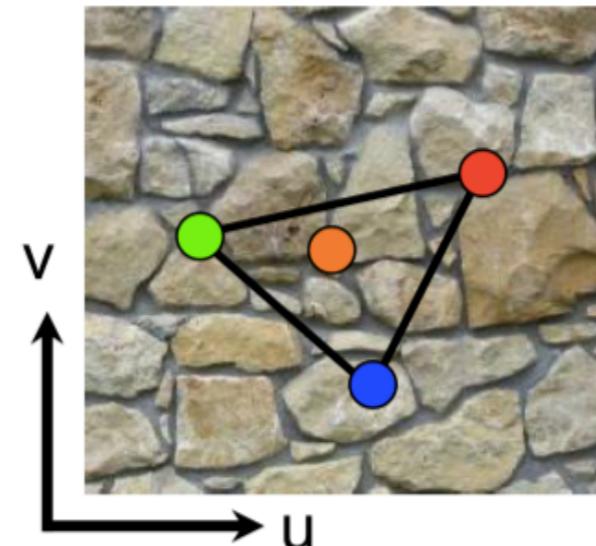
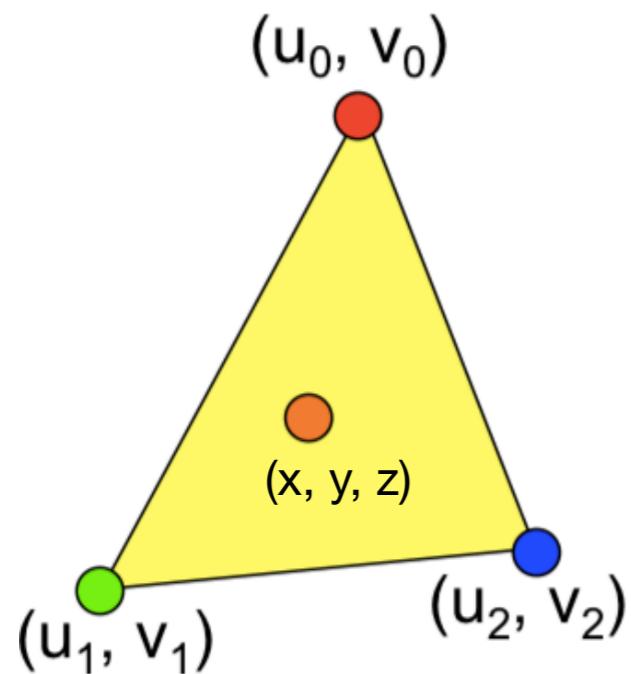


2. **Per a cada punt interior del polígon, es calculen les coordenades paramètriques a partir de les coordenades paramètriques dels vèrtexs extrems del polígon**



4. Textures: funció de projecció

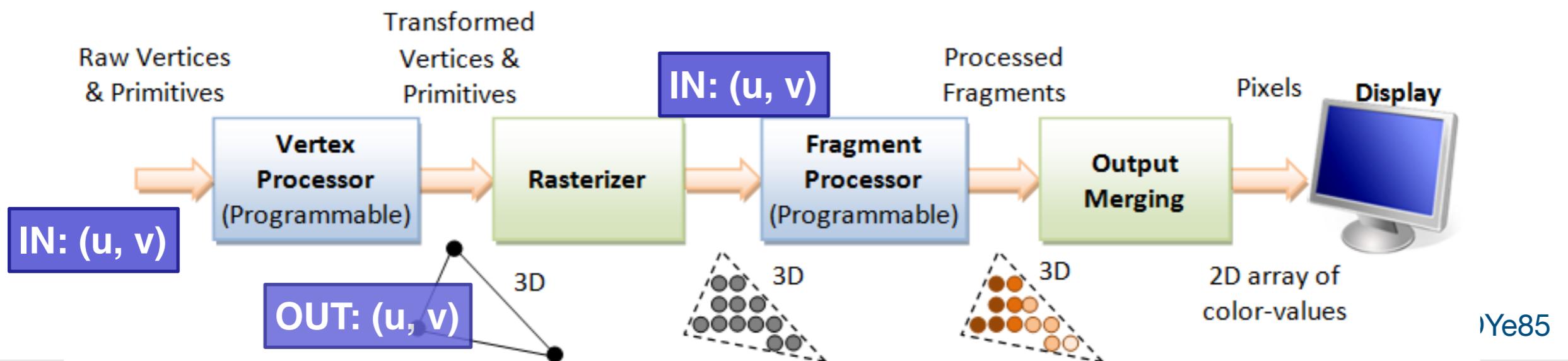
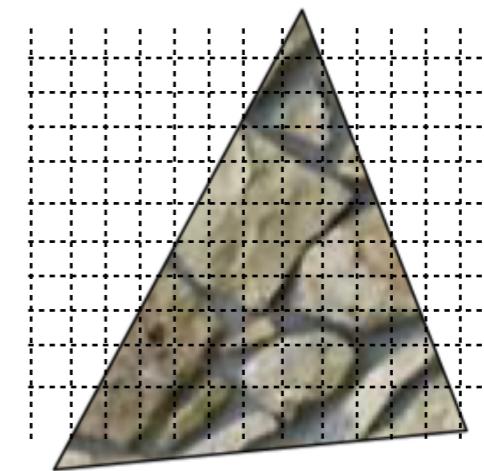
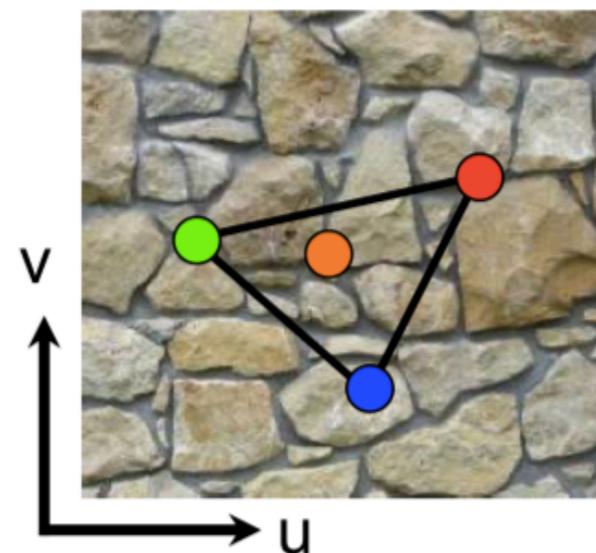
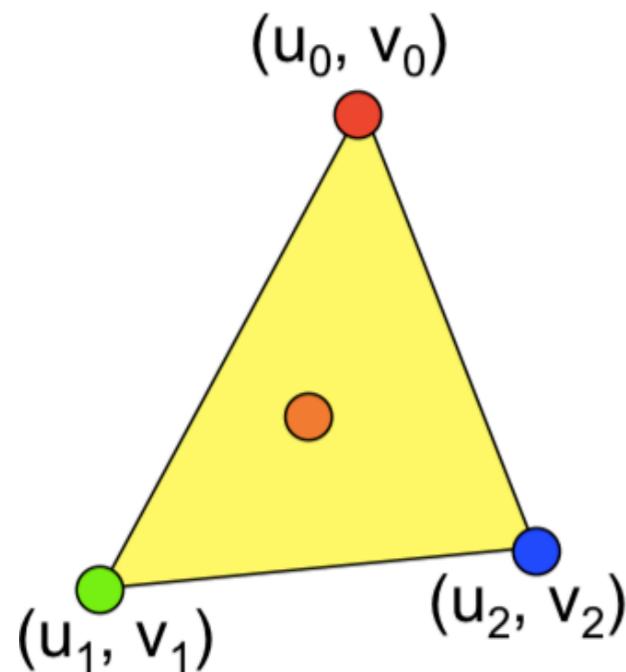
- **Triangle:** $(x, y, z) \rightarrow (u, v)$
- Per a cada vèrtex del triangle es tenen definides les coordenades (u, v)
- A cada punt interior del triangle s'interpolen les seves coordenades (u, v) segons les seves coordenades baricèntriques



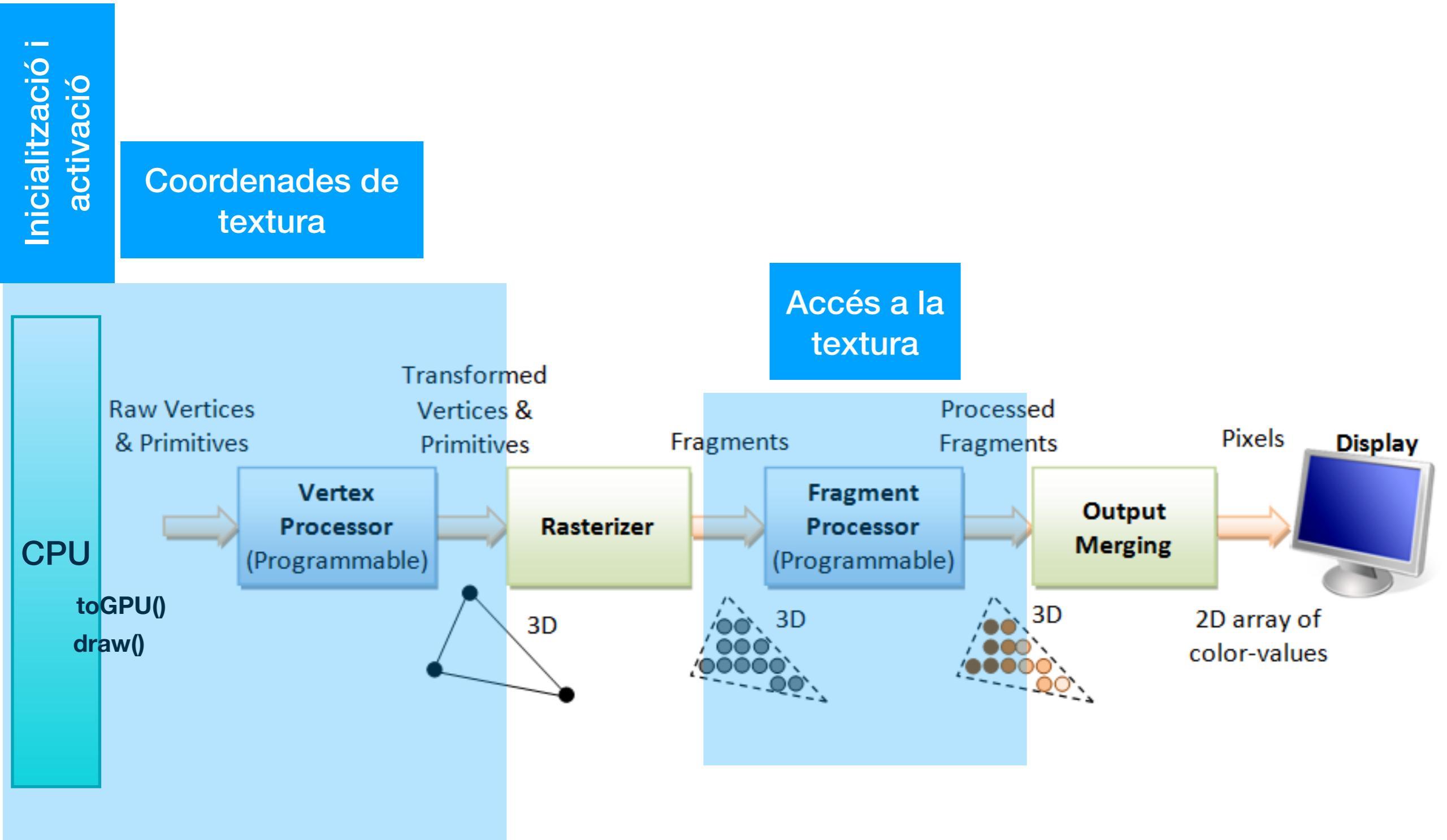
<https://www.geogebra.org/m/sUuDYe85>

4. Textures: funció de projecció

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



4. Textures



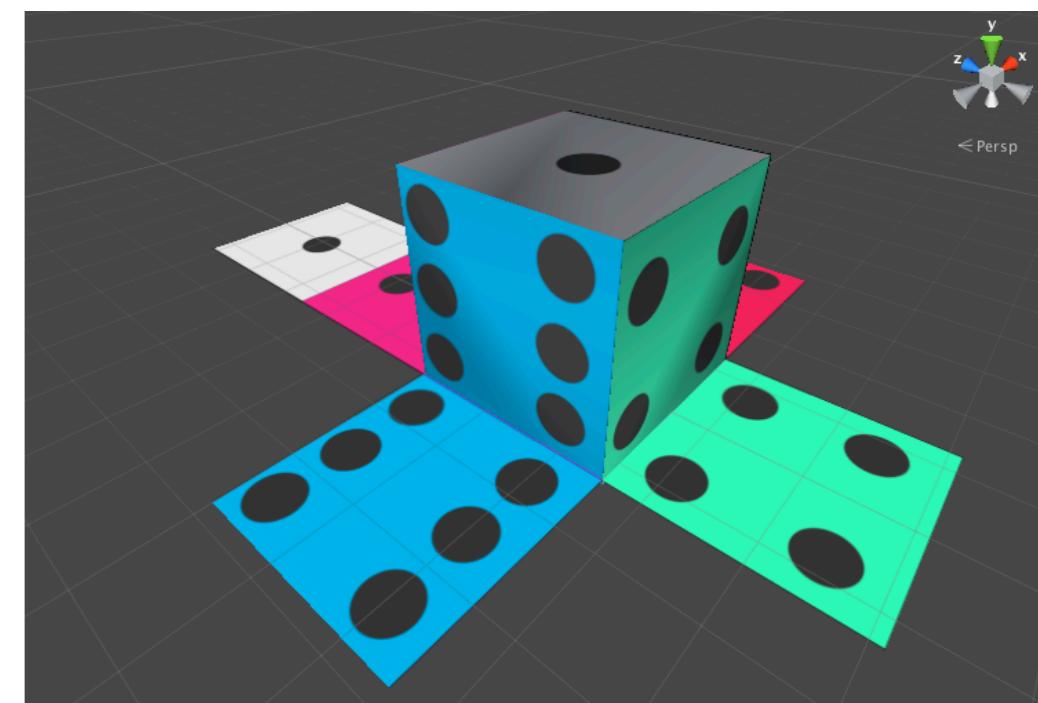
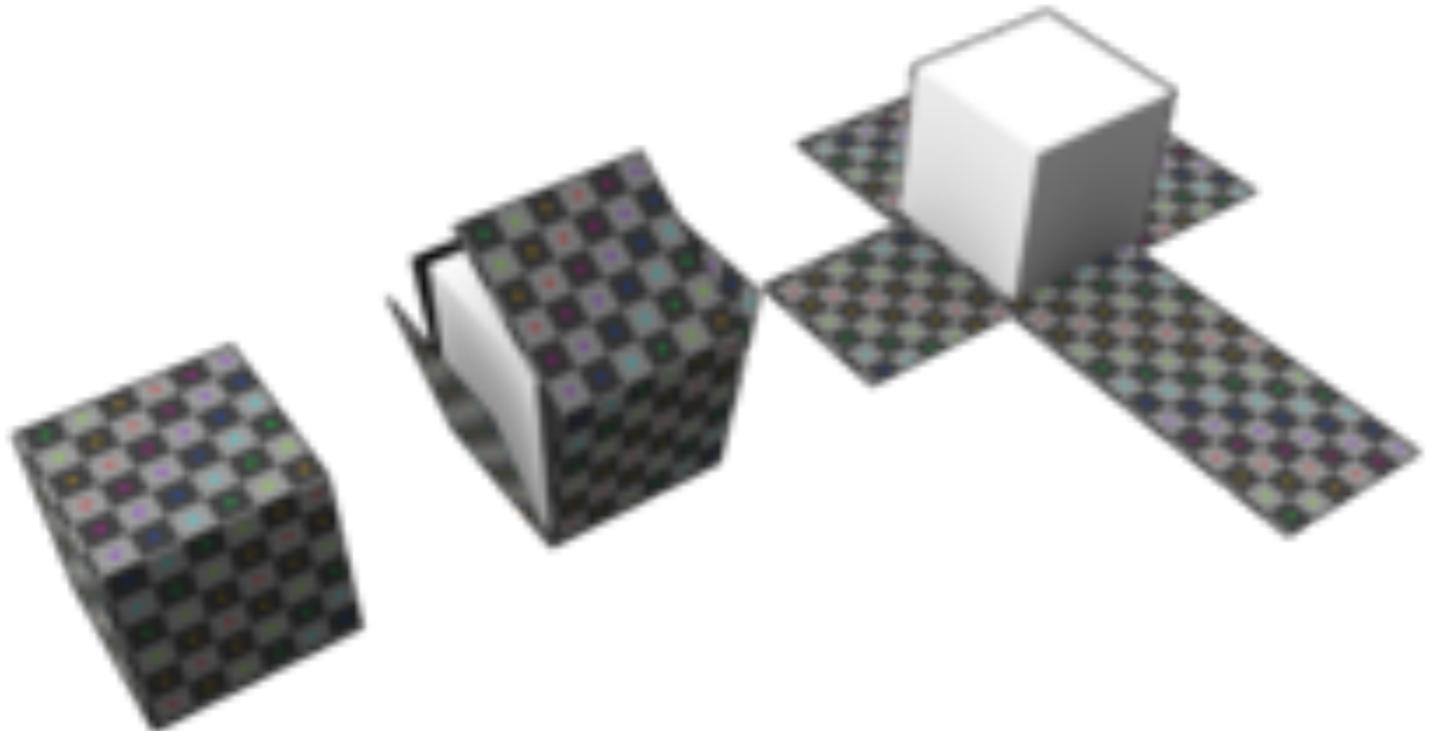
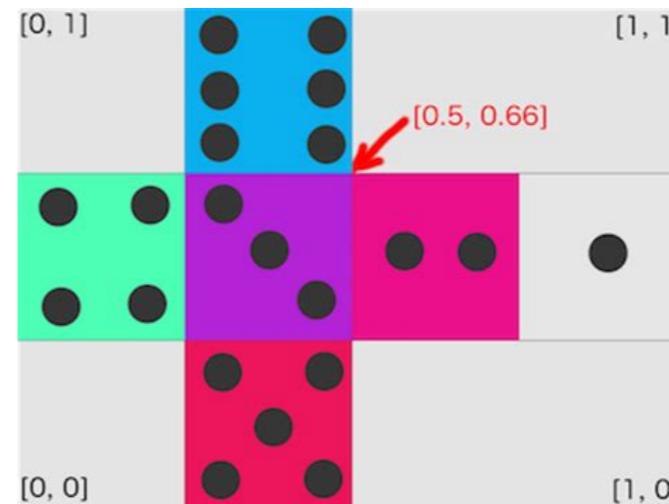
4. Textures: funció de projecció

- **Malles poligonals:**

Definir (u, v) a cada vèrtex

UV unwrapping:

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



4. Textures: funció de projecció

- 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)