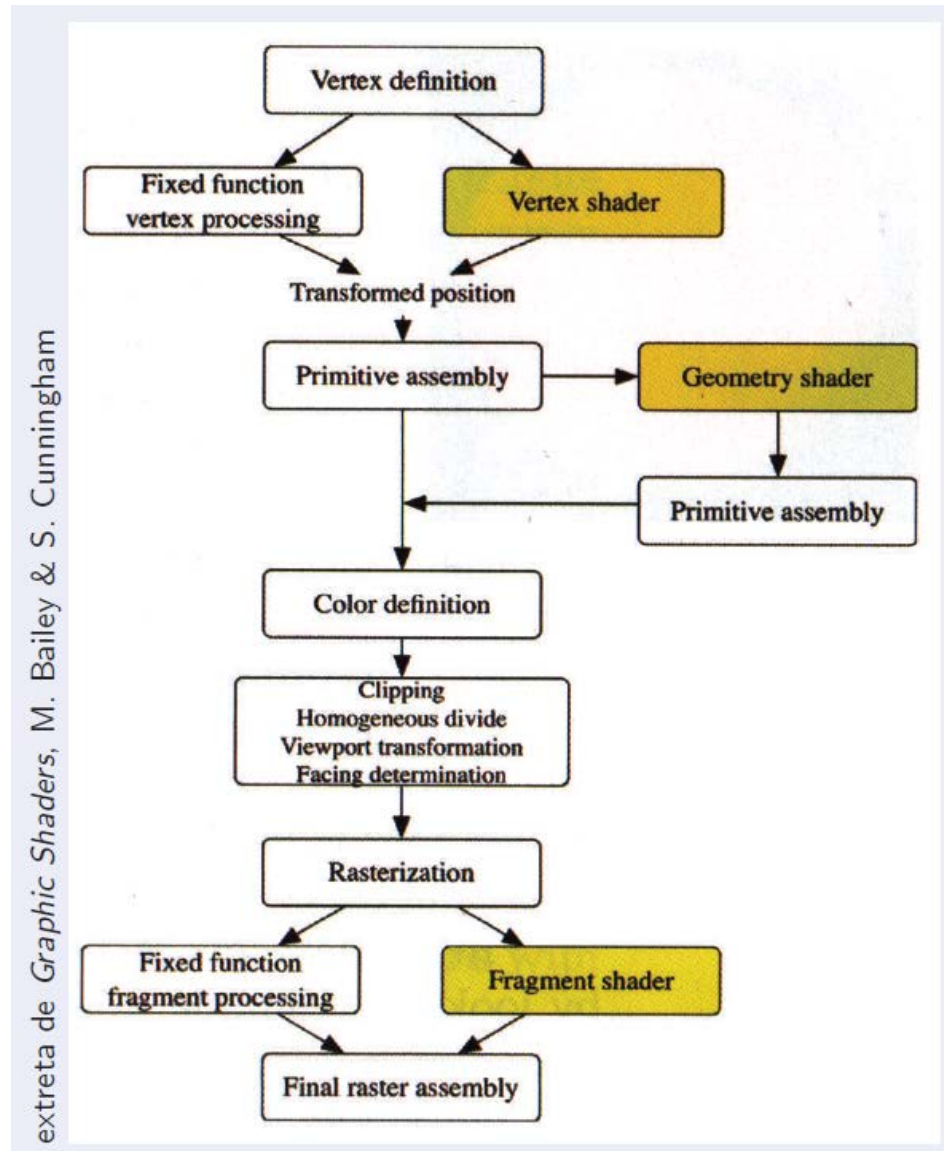


PIPELINE PROGRAMMABLE

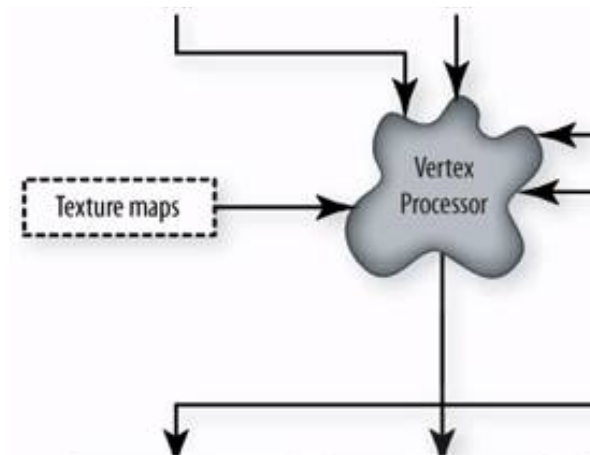
Pipeline programmable (amb GS)



Vertex shader (3.3 core)

Attributes (user-defined)

```
vec3 vertex; // object space  
vec3 normal;  
vec3 color;  
vec2 texCoord; ...
```



Uniforms (user-defined, read-only)

```
mat4 modelViewMatrix;  
mat3 normalMatrix;  
vec4 lightAmbient;
```

...

Outputs

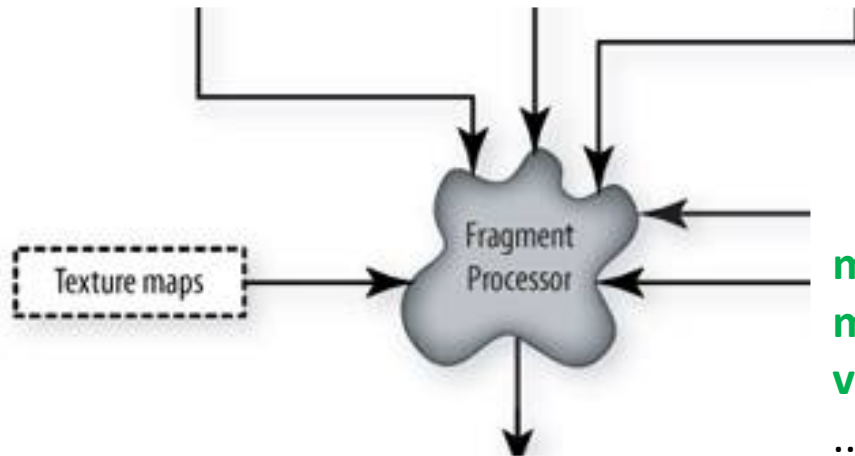
```
vec4 gl_Position; // predefinit; usualment en clip space  
vec4 frontColor;
```

Fragment shader (3.3 core)

Inputs

```
vec4 gl_FragCoord; // window space  
bool gl_FrontFacing;
```

```
vec4 frontColor; ...
```



Uniforms (user-defined, read-only)

```
mat4 modelViewMatrix;  
mat3 normalMatrix;  
vec4 lightAmbient;
```

...

Outputs

```
float gl_FragDepth; // z in window space
```

```
vec4 fragColor;
```

Viewer

- Funciona en linux32 i linux64
- És recomanable crear una carpeta amb els shaders que anireu creant:
 - **mkdir shaders** (on vulgueu)
 - **cd shaders/**
 - **/assig/grau-g/viewer**

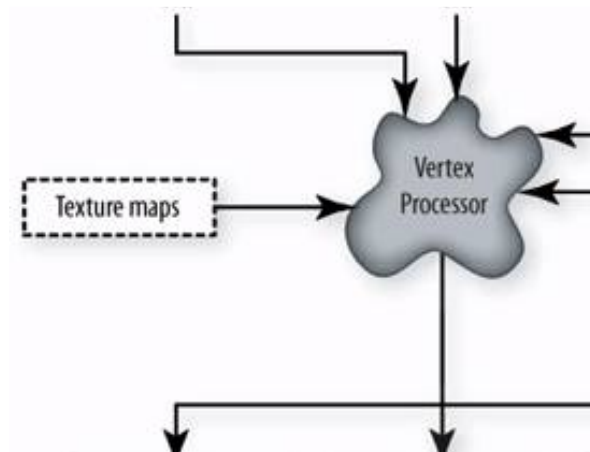
Premeu [SPACE] per accedir al memu

VERTEX SHADERS

Vertex shader (3.3 core)

Attributes (user-defined)

```
vec3 vertex; // object space  
vec3 normal;  
vec3 color;  
vec2 texCoord; ...
```



Uniforms (user-defined, read-only)

```
mat4 modelViewMatrix;  
mat3 normalMatrix;  
vec4 lightAmbient;
```

...

Outputs

```
vec4 gl_Position; // predefinit; usualment en clip space  
vec4 frontColor;
```

Vertex shaders

- **Attribute** variables: són variables que representen els *atributs d'un vèrtex*. Poden canviar de valor per cada vèrtex d'una mateixa primitiva. Pel VS són d'entrada.
- **Attributes definits pel viewer** (cal declarar-los):

layout (location = 0) in vec3 **vertex**; // similar a gl_Vertex (però 3D)

layout (location = 1) in vec3 **normal**; // idèntic a gl_Normal

layout (location = 2) in vec3 **color**; // similar a gl_Color (però RGB)

layout (location = 3) in vec2 **texCoord**; // similar a gl_MultiTexCoord0

Vertex shaders

- **Uniform** variables: són variables que canvien amb poca freqüència. Com a molt poden canviar un cop *per cada primitiva* (però no pas per cada vèrtex de la primitiva).

Vertex shaders

Variables uniform que envia el viewer (cal declarar-les)

```
uniform mat4 modelMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 projectionMatrix;  
uniform mat4 modelViewMatrix;  
uniform mat4 modelViewProjectionMatrix;
```

```
uniform mat4 modelMatrixInverse;  
uniform mat4 viewMatrixInverse;  
uniform mat4 projectionMatrixInverse;  
uniform mat4 modelViewMatrixInverse;  
uniform mat4 modelViewProjectionMatrixInverse;
```

```
uniform mat3 normalMatrix;
```

Vertex shaders

Variables uniform que envia el viewer:

uniform vec4 lightAmbient;	// similar a gl_LightSource[0].ambient
uniform vec4 lightDiffuse;	// similar a gl_LightSource[0].diffuse
uniform vec4 lightSpecular;	// similar a gl_LightSource[0].specular
uniform vec4 lightPosition;	// similar a gl_LightSource[0].position
	// (sempre estarà en <i>eye space</i>)

uniform vec4 matAmbient;	// similar a gl_FrontMaterial.ambient
uniform vec4 matDiffuse;	// similar a gl_FrontMaterial.diffuse
uniform vec4 matSpecular;	// similar a gl_FrontMaterial.specular
uniform float matShininess;	// similar a gl_FrontMaterial.shininess

Vertex shaders

Variables uniform que envia el viewer:

uniform vec3 boundingBoxMin; // cantonada de la capsa englobant

uniform vec3 boundingBoxMax; // cantonada de la capsa englobant

uniform vec2 mousePosition; // coordenades del cursor (window space)
 // origen a la cantonada inferior esquerra

Vertex shaders

Output variables:

- **out vec4** gl_Position (predeclarada)
- Variables “varying”: el VS les passa al FS
 - Pel VS són de sortida.
 - Pel FS són d’entrada, i es calculen per interpolació.
 - Exemples típics (depenen de l’aplicació): color, normal, coordenades del vèrtex, coordenades de textura...

Vertex shaders

Un vertex shader sempre ha d'escriure a

vec4 gl_Position

(usualment les coordenades del vèrtex en clip space).

Normalment ho farà multiplicant el vèrtex per la matriu
modelViewProjectionMatrix.

Vertex shaders

- El VS s'executa per cada vèrtex que s'envia a OpenGL.
- Les tasques *habituals* d'un VS són:
 - Transformar el vèrtex (object space → clip space)
 - Transformar i normalitzar la normal (eye space)
 - Calcular la il·luminació del vèrtex
 - Generar o passar les coords de textura pel vèrtex

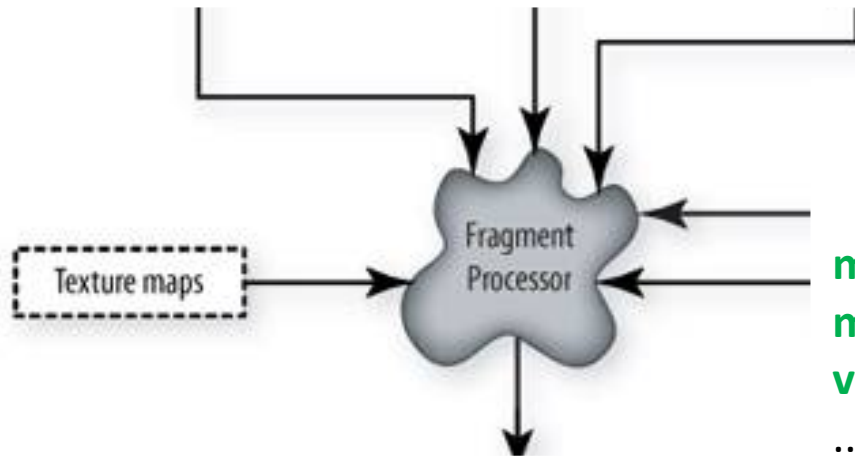
FRAGMENT SHADERS

Fragment shader (3.3 core)

Inputs

```
vec4 gl_FragCoord; // window space  
bool gl_FrontFacing;
```

```
vec4 frontColor; ...
```



Uniforms (user-defined, read-only)

```
mat4 modelViewMatrix;  
mat3 normalMatrix;  
vec4 lightAmbient;
```

...

Outputs

```
float gl_FragDepth; // z in window space
```

```
vec4 fragColor;
```

Fragment shaders

Special input variables: calculats per OpenGL de forma automàtica; es poden llegir al fragment shader:

vec4 gl_FragCoord; // coordenades del fragment (window space)

bool gl_FrontFacing; // true si el fragment és d'un polígon frontface

Fragment shaders

- **Varying** variables: són variables que es calculen al vertex shader i arriben interpolades al fragment shader.
- **Exemple (core profile):**
in vec4 frontColor;

Fragment shaders

Output variables:

- Predefinides:

float **gl_FragDepth** // depth final del fragment (pel z-buffer)

- Definides per l'usuari:

out vec4 **fragColor** // color del fragment

Fragment shaders

- Un fragment shader s'executa per cada fragment que produeix cada primitiva.
- Les tasques habituals d'un fragment shader són:
 - Accedir a textura
 - Incorporar el color de la textura
 - Incorporar efectes a nivell de fragment (ex. boira).
- I el que no pot fer un fragment shader:
 - Canviar les coordenades del fragment (sí pot canviar **gl_FragDepth**)
 - Accedir a informació d'altres fragments (tret de dFdx, dFdy)

LLENGUATGE GLSL

Elements del llenguatge GLSL

Tipus bàsics

Escalars

`int, float, bool`

Vectorials

`vec2, vec3, vec4, mat2, mat3, mat4, ivec3, bvec4,...`

Constructors

Hi ha *arrays*: `mat2 mats[3];`

i també *structs*:

```
1      struct light{  
2          vec3 color;  
3          vec3 pos;  
4      };
```

que defineixen implícitament constructors: `light l1(col,p);`

Elements del llenguatge GLSL

Funcions

N'hi ha moltes, especialment en les àrees que poden interessar quan tractem geometria o volem dibuixar. Per exemple, `radians()`, `degrees()`, `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()` (amb un o amb dos paràmetres), `pow()`, `log()`, `exp()`, `abs()`, `sign()`, `floor()`, `min()`, `max()`, `length()`, `distance()`, `dot()`, `cross()`, `normalize()`, `noise1()`, `noise2()`, ...

EXAMPLE: PHONG SHADING (VS)

VS (1/3)

```
#version 330 core
```

```
layout (location = 0) in vec3 vertex;  
layout (location = 1) in vec3 normal;  
layout (location = 2) in vec3 color;  
layout (location = 3) in vec2 texCoord;
```

```
out vec4 frontColor;
```

```
uniform mat4 modelViewProjectionMatrix;  
uniform mat4 modelViewMatrix;  
uniform mat3 normalMatrix;
```

```
uniform vec4 matAmbient, matDiffuse, matSpecular;  
uniform float matShininess;  
uniform vec4 lightAmbient, lightDiffuse, lightSpecular, lightPosition;
```

VS (2/3)

```
vec4 light(vec3 N, vec3 V, vec3 L)
{
    N=normalize(N);
    V=normalize(V);
    L=normalize(L);
    vec3 R = normalize( 2.0*dot(N,L)*N-L );
    float NdotL = max( 0.0, dot( N,L ) );
    float RdotV = max( 0.0, dot( R,V ) );
    float Idiff = NdotL;
    float Ispec = 0;
    if (NdotL>0) Ispec=pow( RdotV, matShininess );
    return      matAmbient * lightAmbient +
               matDiffuse * lightDiffuse * Idiff +
               matSpecular * lightSpecular * Ispec;
}
```

VS (3/3)

```
void main()
{
    vec3 P = (modelViewMatrix * vec4(vertex.xyz, 1.0)).xyz;
    vec3 N = normalize(normalMatrix * normal);
    vec3 V = -P;
    vec3 L = (lightPosition.xyz - P);
    frontColor = light(N, V, L);
    gl_Position = modelViewProjectionMatrix * vec4(vertex.xyz, 1.0);
}
```