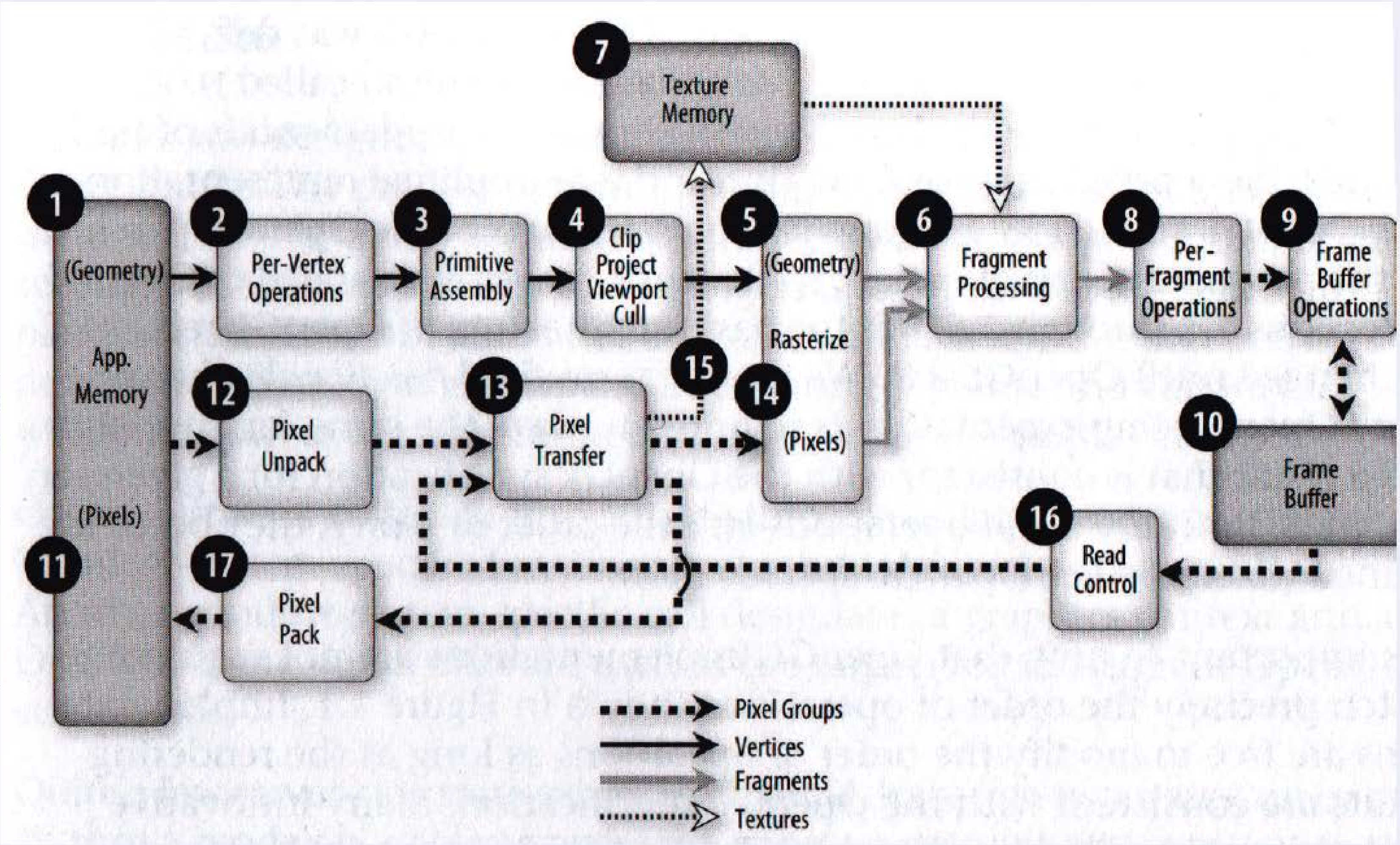
Vertex shaders i Fragment shaders
Entorn per desenvolupar shaders (viewer)

Professors de Gràfics

Febrer 2016

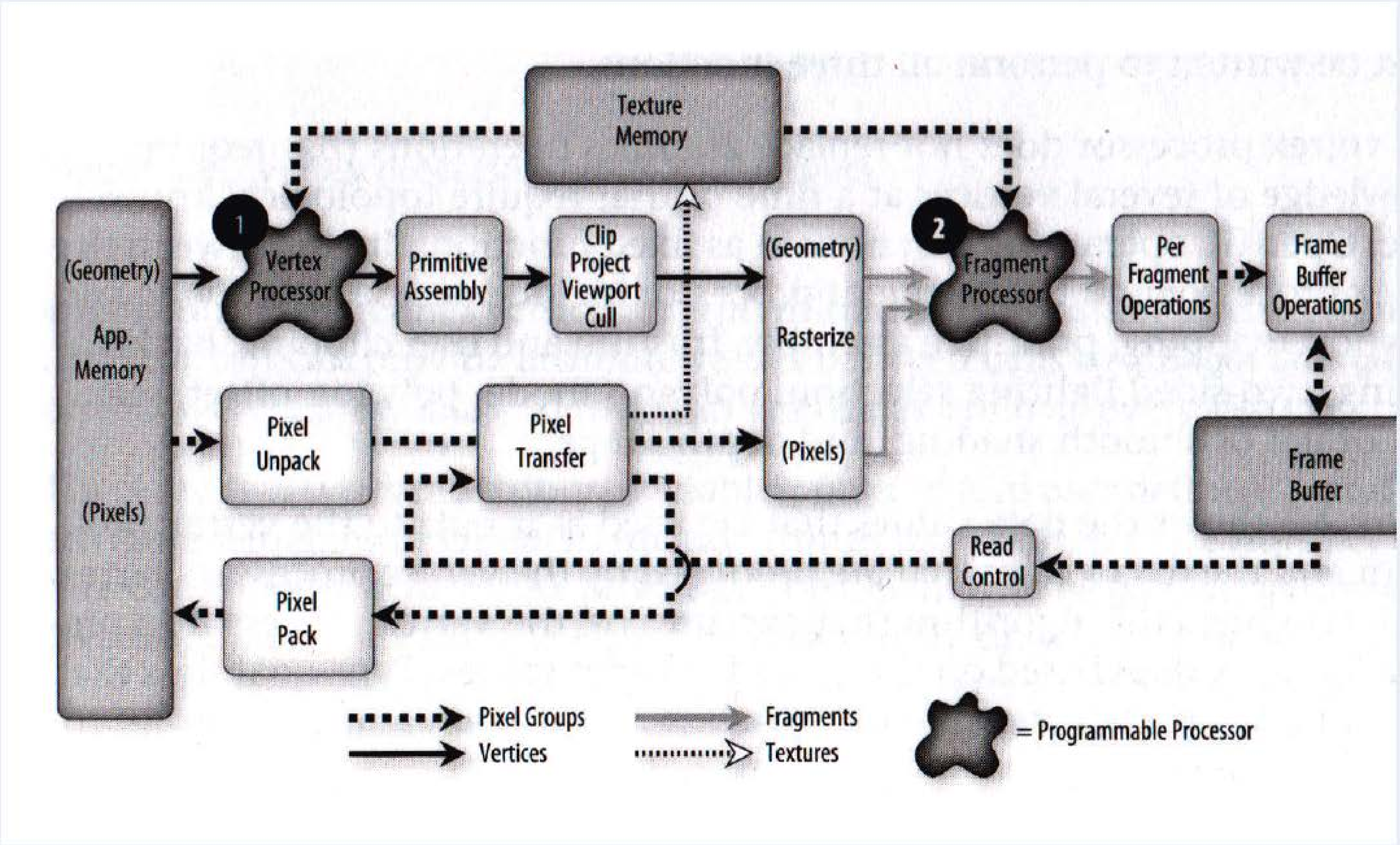
PIPELINE PROGRAMABLE

Pipeline fix



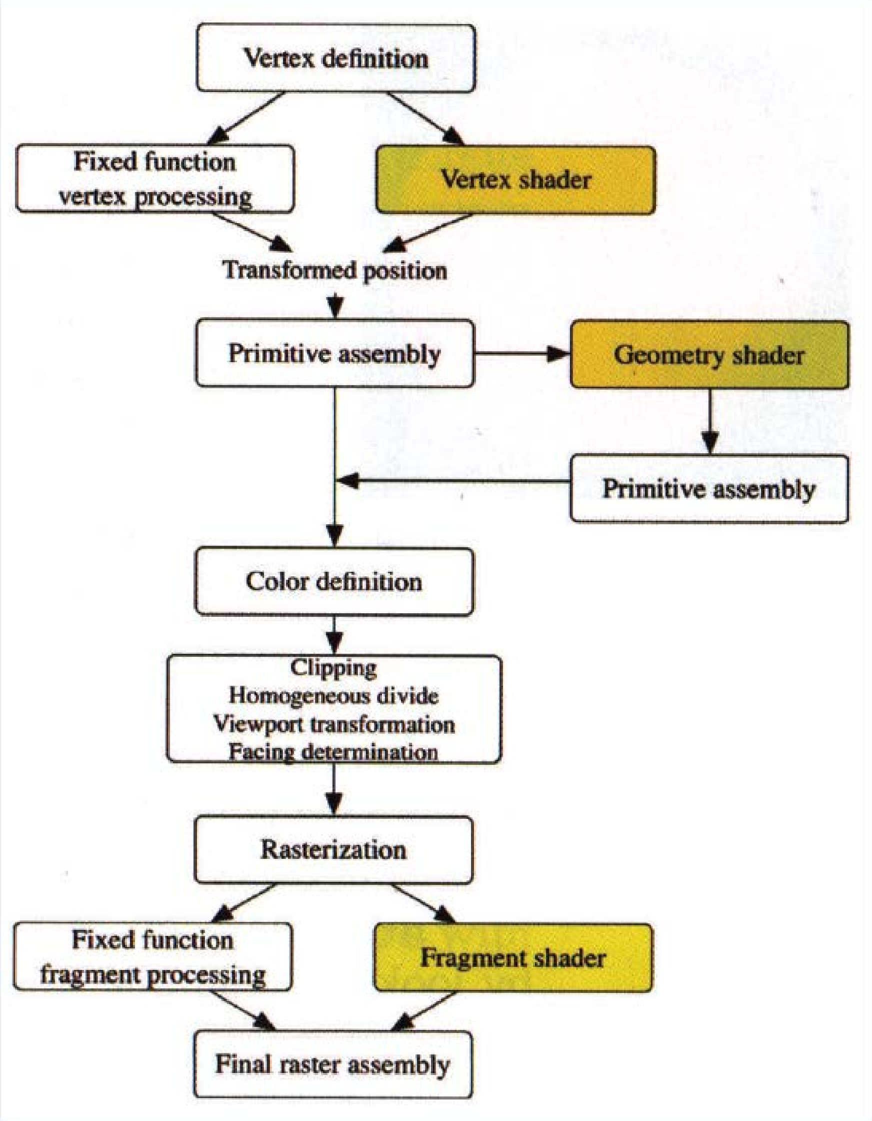
extreta de *OpenGL Shading Language*, R. J. Rost *et. al.*

Pipeline programable



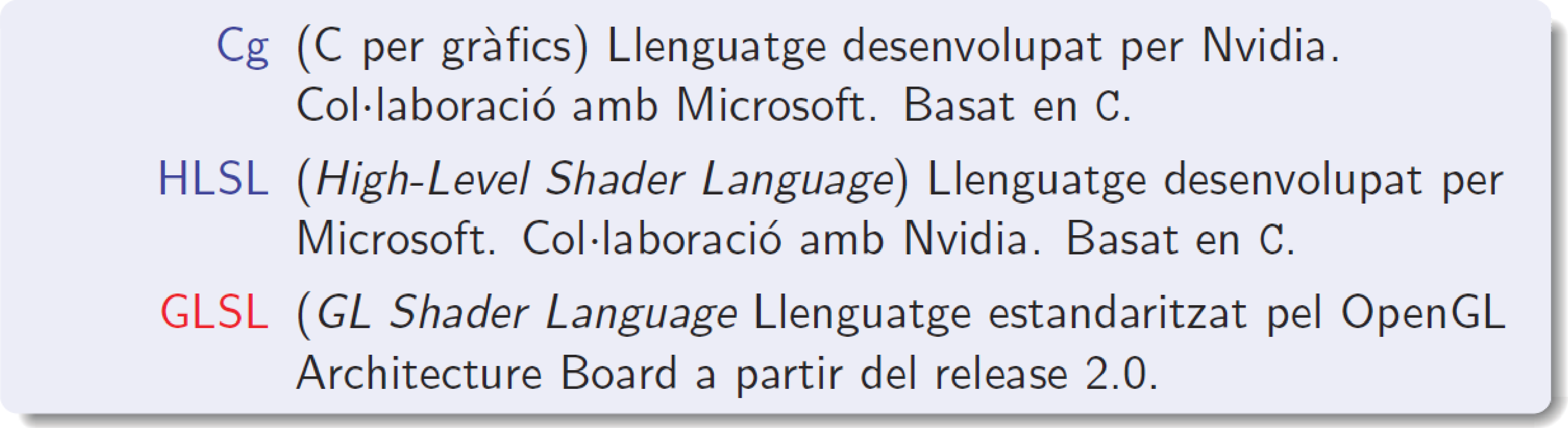
extreta de *OpenGL Shading Language*, R. J. Rost *et. al.*

Pipeline programable (amb GS)



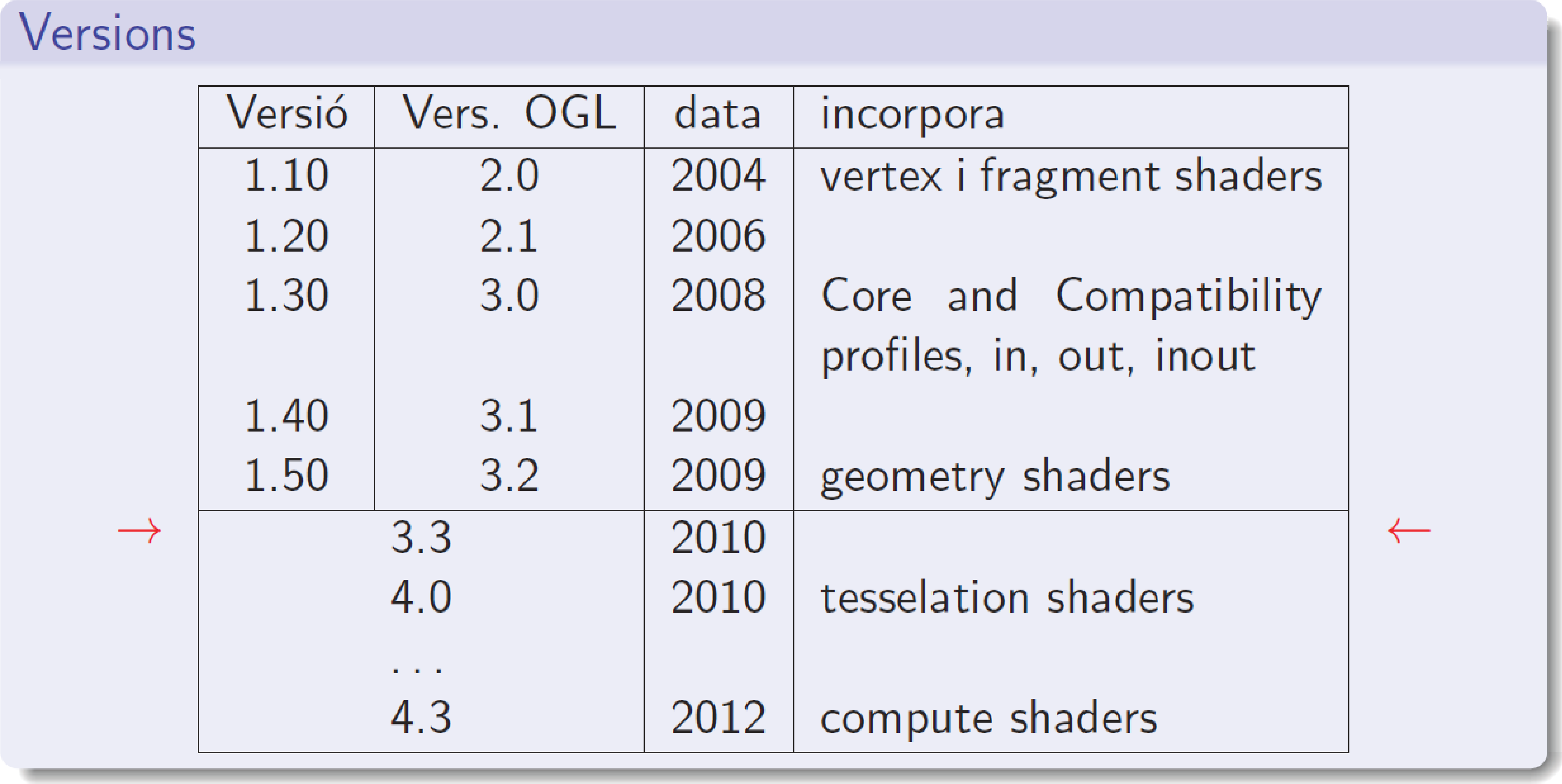
extreta de *Graphic Shaders*, M. Bailey & S. Cunningham

Llenguatges de programació shaders



- **Cg** (C per gràfics) Llenguatge desenvolupat per Nvidia. Col·laboració amb Microsoft. Basat en `C`.
- **HLSL** (*High-Level Shader Language*) Llenguatge desenvolupat per Microsoft. Col·laboració amb Nvidia. Basat en `C`.
- **GLSL** (*GL Shader Language* Llenguatge estandaritzat pel OpenGL Architecture Board a partir del release 2.0.

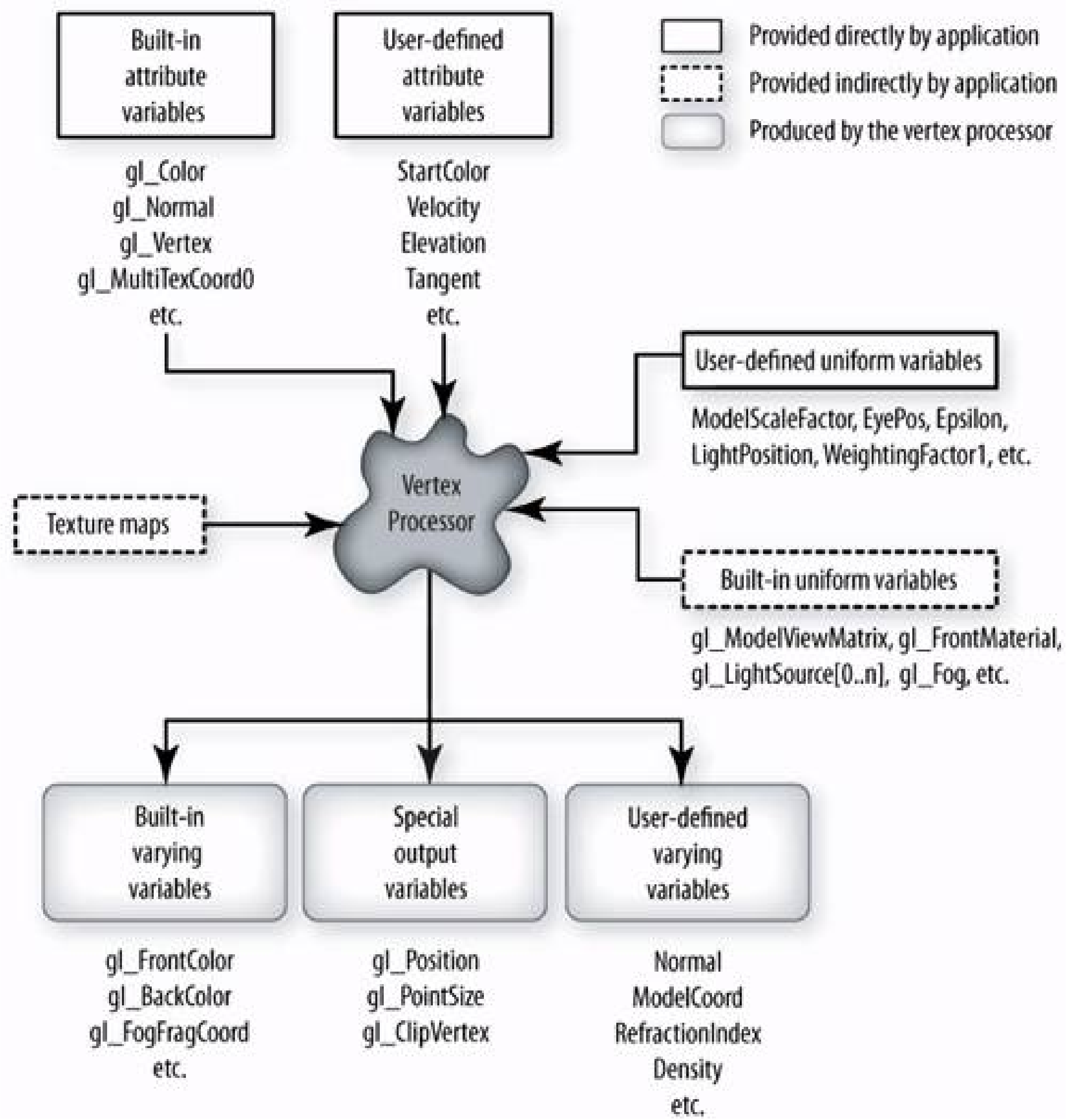
Versions



Versions

	Versió	Vers. OGL	data	incorpora	
	1.10	2.0	2004	vertex i fragment shaders	
	1.20	2.1	2006		
	1.30	3.0	2008	Core and Compatibility profiles, in, out, inout	
	1.40	3.1	2009		
	1.50	3.2	2009	geometry shaders	
→	3.3		2010		←
	4.0		2010	tesselation shaders	
	. . .				
	4.3		2012	compute shaders	

Vertex shader (compatibility)

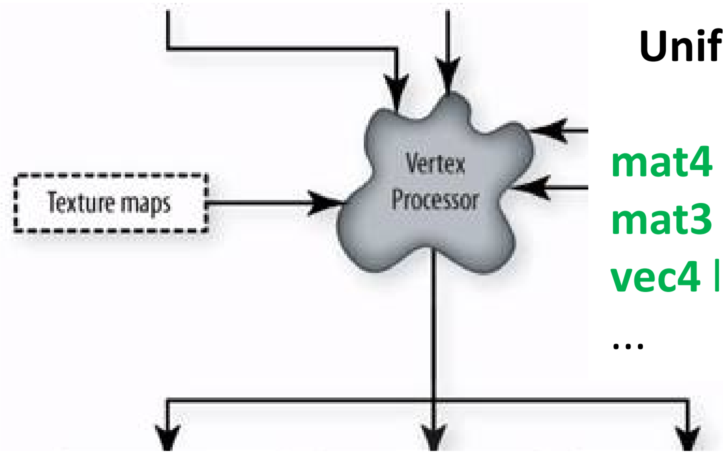


Built-in attribute variables
gl_Color
gl_Normal
gl_Vertex
gl_MultiTexCoord0
etc.
User-defined attribute variables
StartColor
Velocity
Elevation
Tangent
etc.
Provided directly by application
Provided indirectly by application
Produced by the vertex processor
User-defined uniform variables
ModelScaleFactor, EyePos, Epsilon,
LightPosition, WeightingFactor1, etc.
Texture maps
Vertex Processor
Built-in uniform variables
gl_ModelViewMatrix, gl_FrontMaterial,
gl_LightSource[0..n], gl_Fog, etc.
Built-in varying variables
gl_FrontColor
gl_BackColor
gl_FogFragCoord
etc.
Special output variables
gl_Position
gl_PointSize
gl_ClipVertex
User-defined varying variables
Normal
ModelCoord
RefractionIndex
Density
etc.

Vertex shader (3.3 core)

Attributes (user-defined)

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



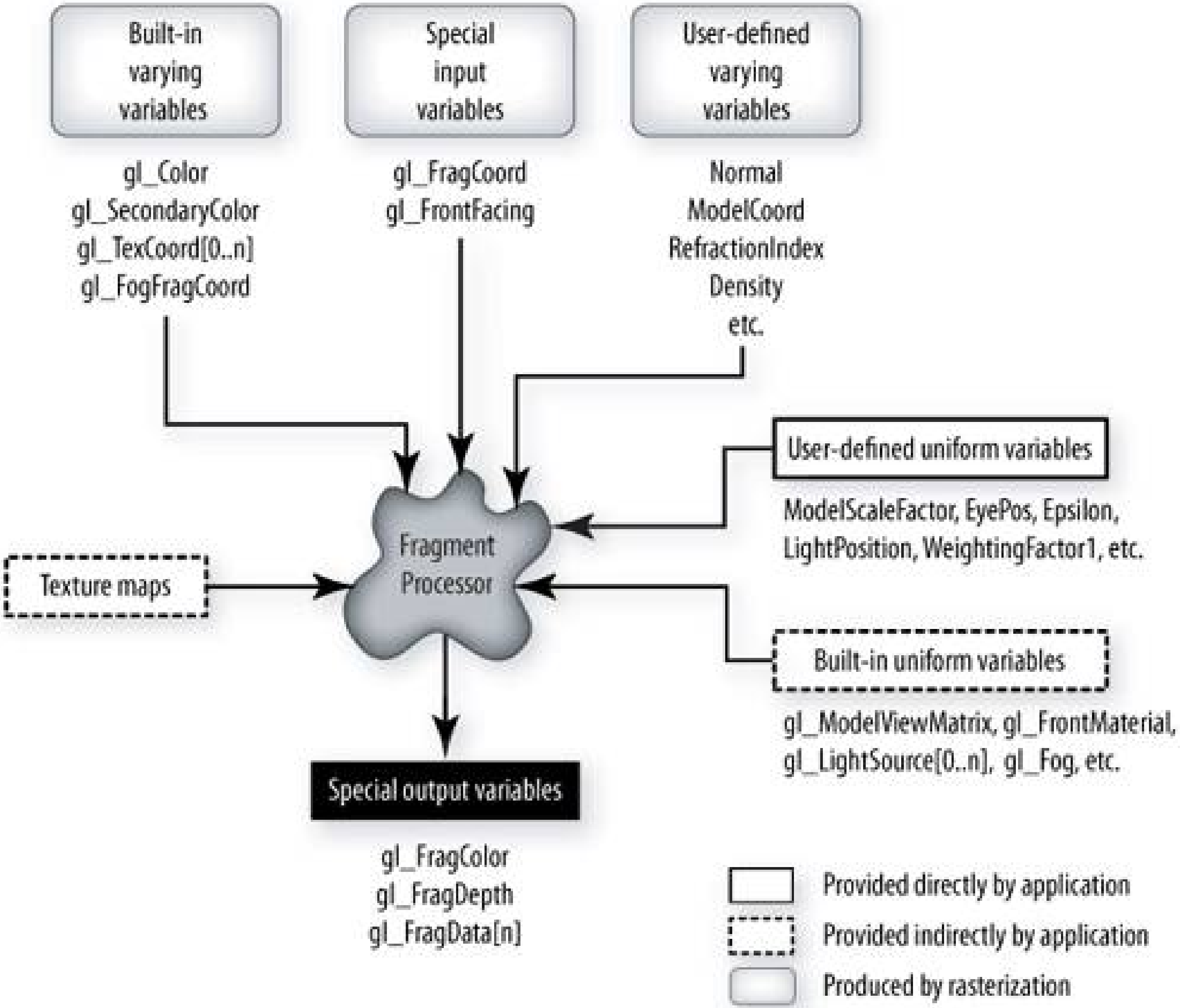
Uniforms (user-defined, read-only)

mat4 modelViewMatrix;
mat3 normalMatrix;
vec4 lightAmbient;
...

Outputs

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

Fragment shader (compatibility)



Built-in varying variables

gl_Color
gl_SecondaryColor
gl_TexCoord[0..n]
gl_FogFragCoord

Special input variables

gl_FragCoord
gl_FrontFacing

User-defined varying variables

Normal
ModelCoord
RefractionIndex
Density
etc.

User-defined uniform variables

ModelScaleFactor, EyePos, Epsilon,
LightPosition, WeightingFactor1, etc.

Texture maps

Fragment Processor

Built-in uniform variables

gl_ModelViewMatrix, gl_FrontMaterial,
gl_LightSource[0..n], gl_Fog, etc.

Special output variables

gl_FragColor
gl_FragDepth
gl_FragData[n]

Provided directly by application

Provided indirectly by application

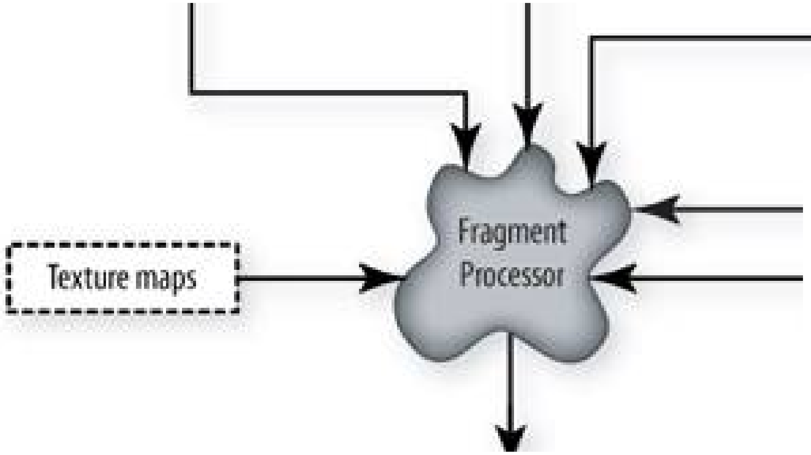
Produced by rasterization

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;
…

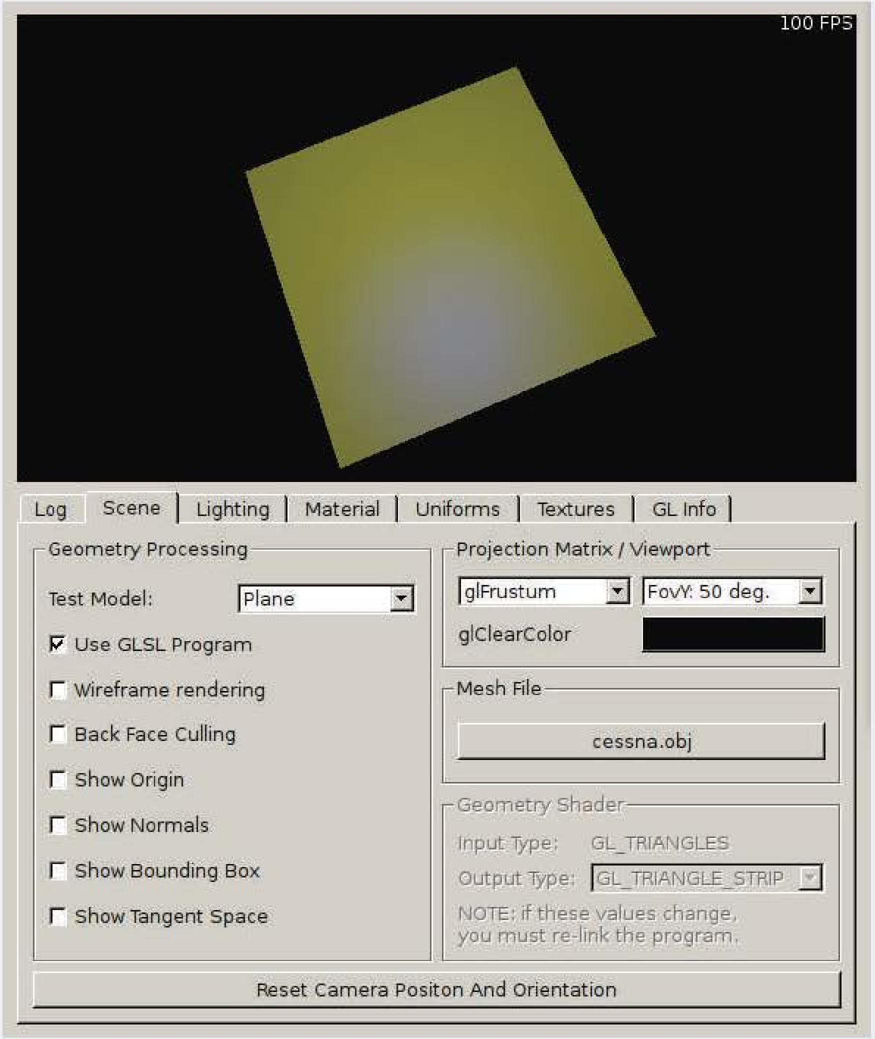
Outputs

float gl_FragDepth; // z in window space

vec4 fragColor;

ShaderMaker

Exemple d'una plataforma per a experimentar



Geometry Processing

Test Model: Plane

☑ Use GLSL Program

□ Wireframe rendering

□ Back Face Culling

□ Show Origin

□ Show Normals

□ Show Bounding Box

□ Show Tangent Space

Projection Matrix / Viewport

glFrustum | FovY: 50 deg.

glClearColor ■

Mesh File

cessna.obj

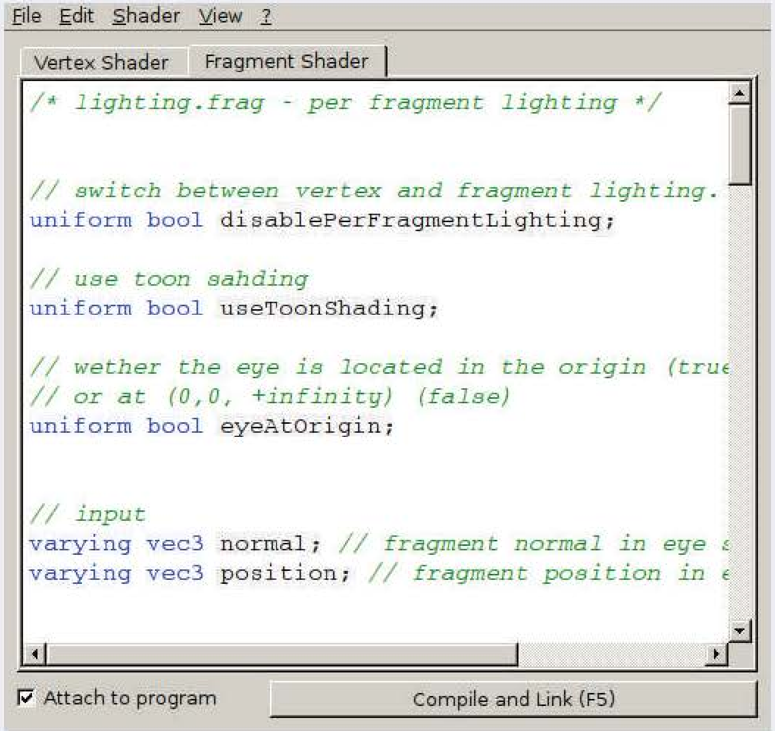
Geometry Shader

Input Type: GL_TRIANGLES

Output Type: GL_TRIANGLE_STRIP

NOTE: if these values change, you must re-link the program.

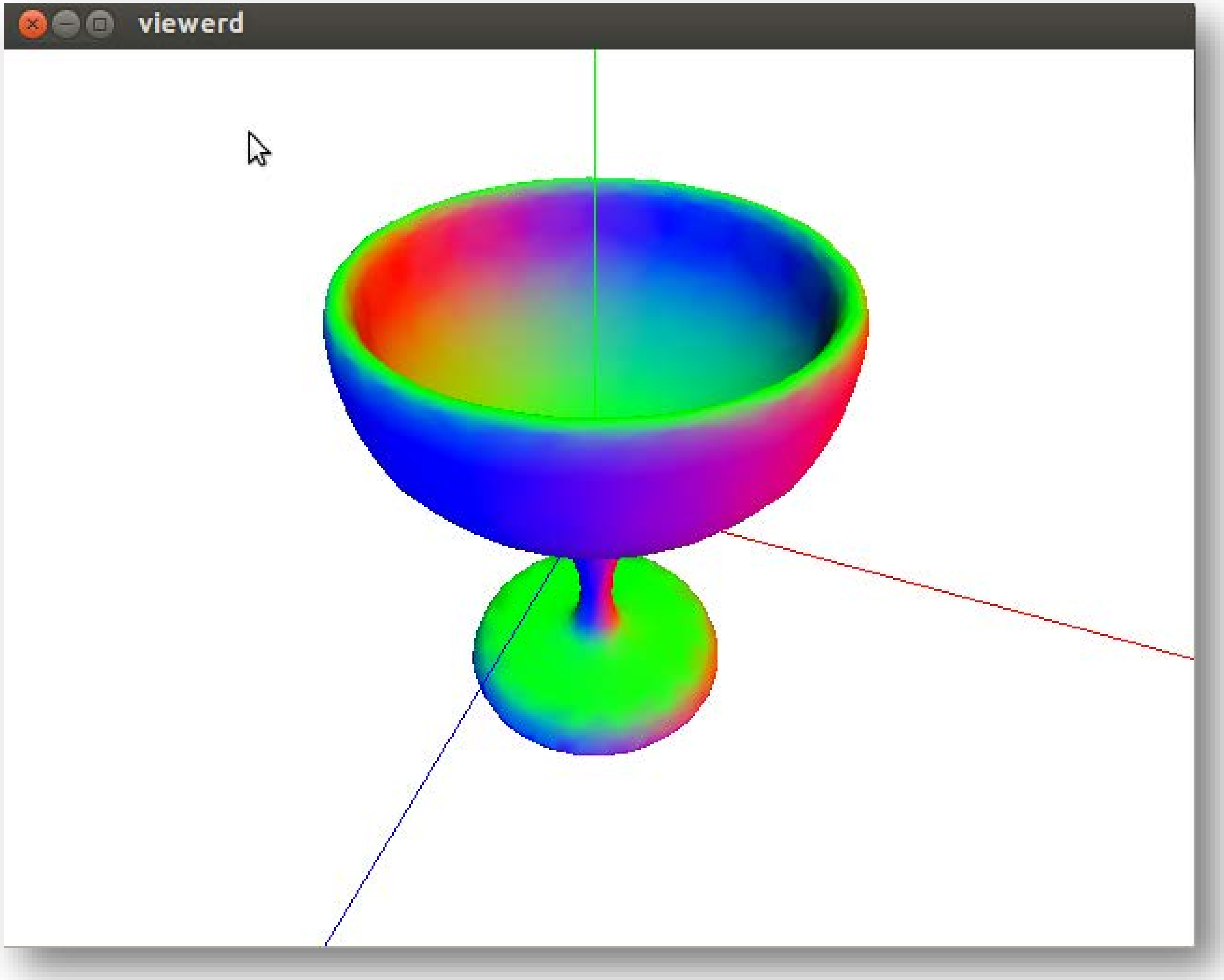
No permet provar shaders en core profile



```
/* lighting.frag - per fragment lighting */


// switch between vertex and fragment lighting.
uniform bool disablePerFragmentLighting;

// use toon sahding
uniform bool useToonShading;

// wether the eye is located in the origin (true
// or at (0,0, +infinity) (false)
uniform bool eyeAtOrigin;


// input
varying vec3 normal; // fragment normal in eye s
varying vec3 position; // fragment position in e
```

☑ Attach to program

Viewer

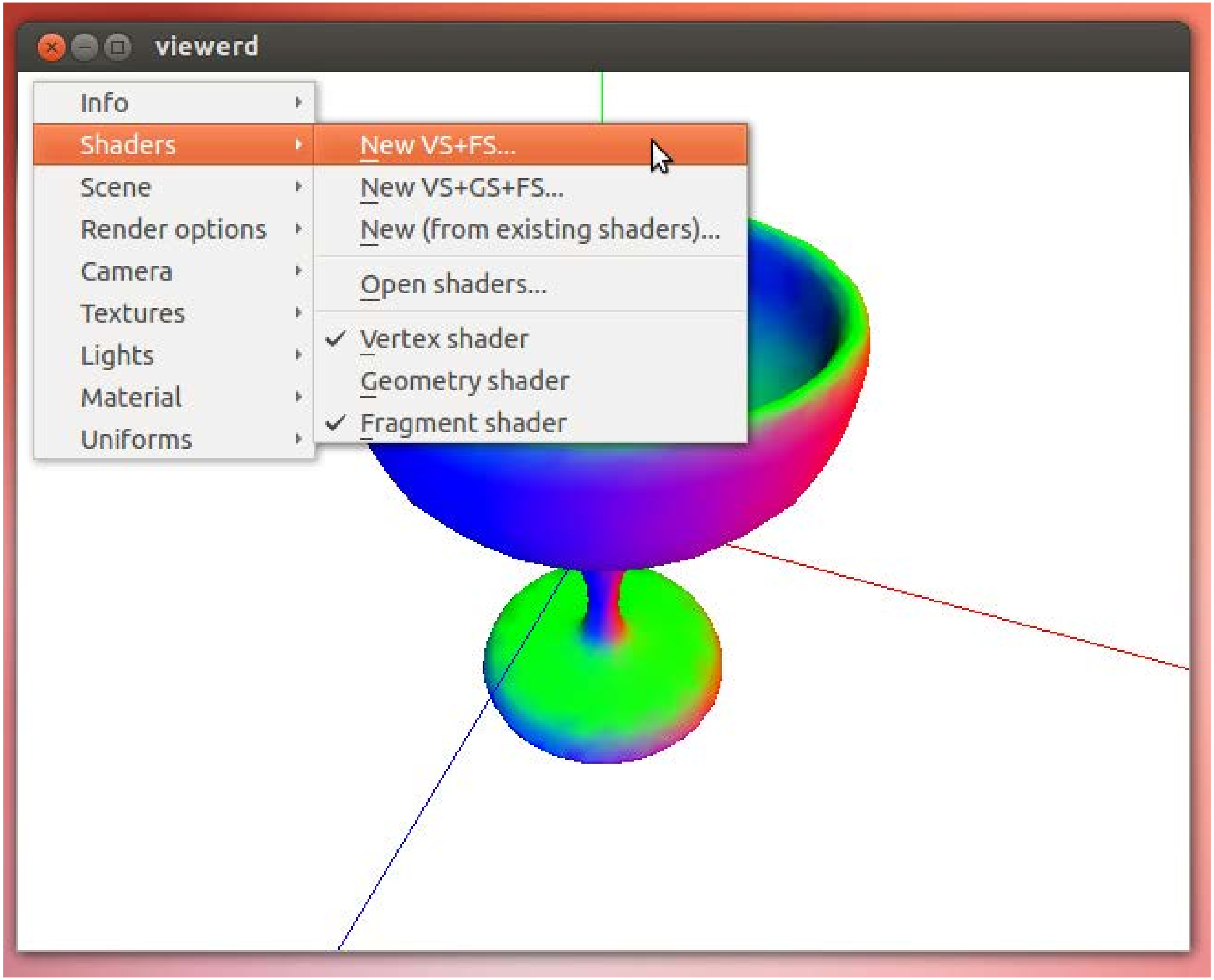


Viewer

- 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

Viewer


viewerd
Info
Shaders
Scene
Render options
Camera
Textures
Lights
Material
Uniforms
New VS+FS...
New VS+GS+FS...
New (from existing shaders)...
Open shaders...
✓ Vertex shader
Geometry shader
✓ Fragment shader

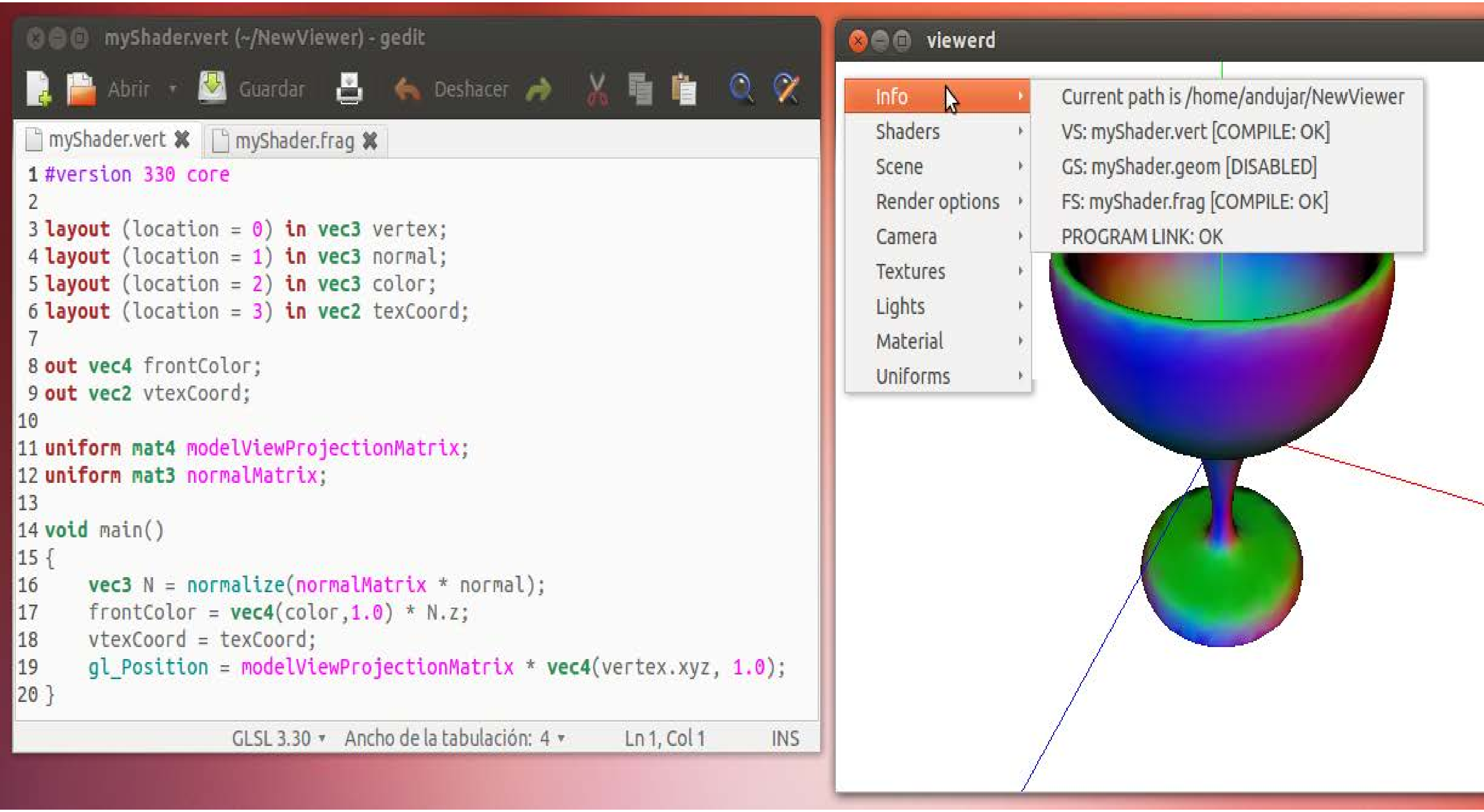
Viewer

Workflow per resoldre cada exercici:

- cd shaders/
- /assig/grau-g/viewer
- Crear/obrir els shaders:
  - Crear shaders des de l'inici:
Shaders → New VS+FS… (usa una plantilla per defecte).
  - Crear els shaders basant-se en shaders existents:
Shaders → New (from existing shaders)…
  - Obrir shaders existents:
Shaders → Open shaders…

Cada cop que guardeu un shader, es carregarà automàticament

Viewer

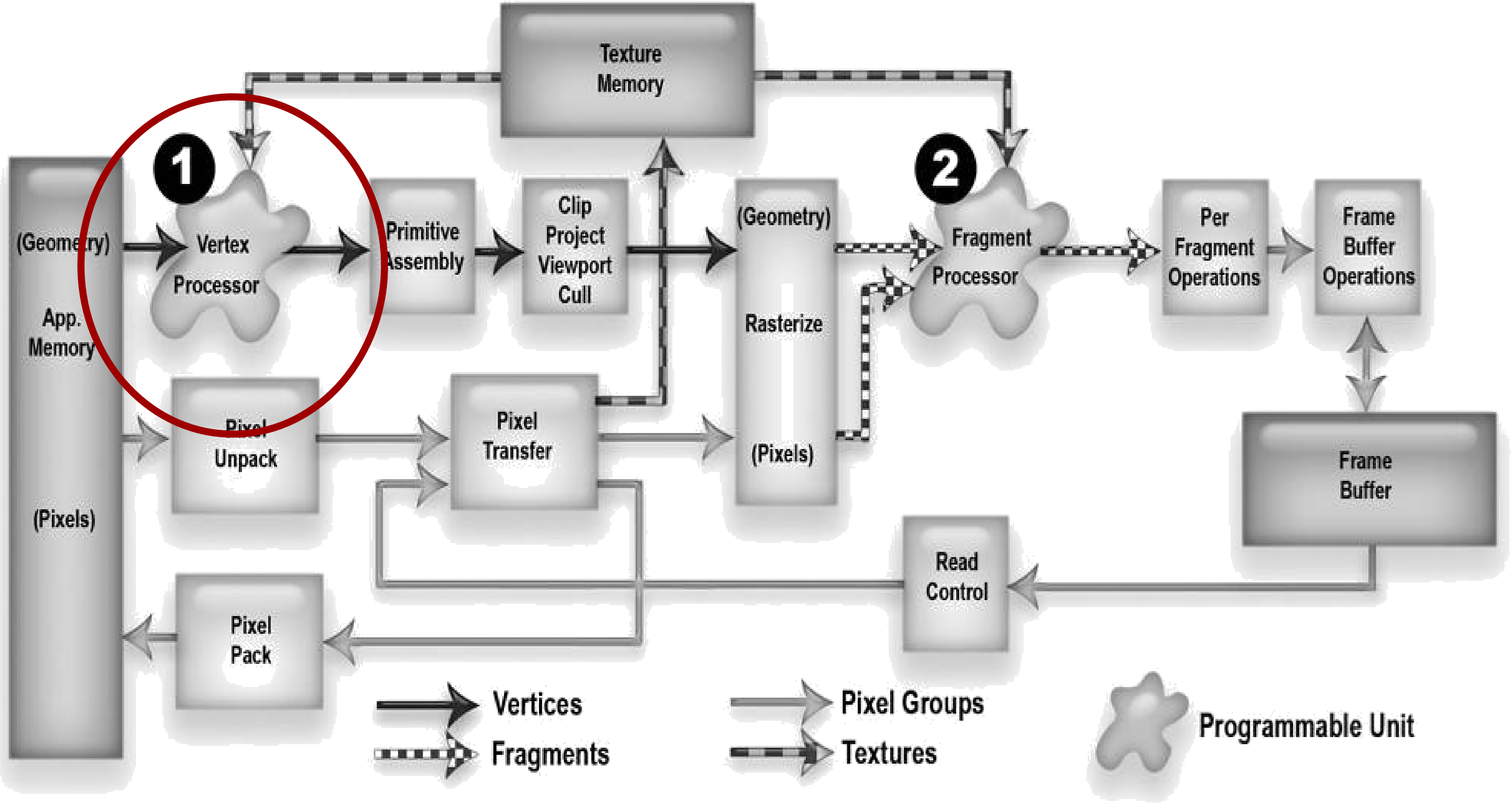


```glsl
#version 330 core

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;
out vec2 vtexCoord;

uniform mat4 modelViewProjectionMatrix;
uniform mat3 normalMatrix;

void main()
{
    vec3 N = normalize(normalMatrix * normal);
    frontColor = vec4(color,1.0) * N.z;
    vtexCoord = texCoord;
    gl_Position = modelViewProjectionMatrix * vec4(vertex.xyz, 1.0);
}
```

```
Current path is /home/andujar/NewViewer
VS: myShader.vert [COMPILE: OK]
GS: myShader.geom [DISABLED]
FS: myShader.frag [COMPILE: OK]
PROGRAM LINK: OK
```

VERTEX SHADERS

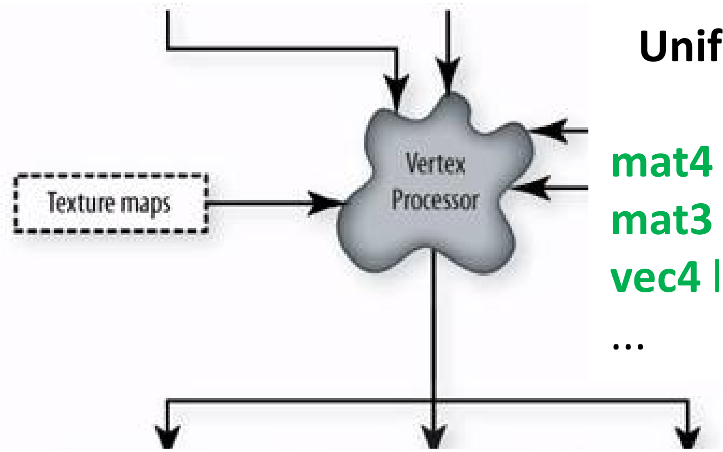
Vertex shaders



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 eye space)

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).

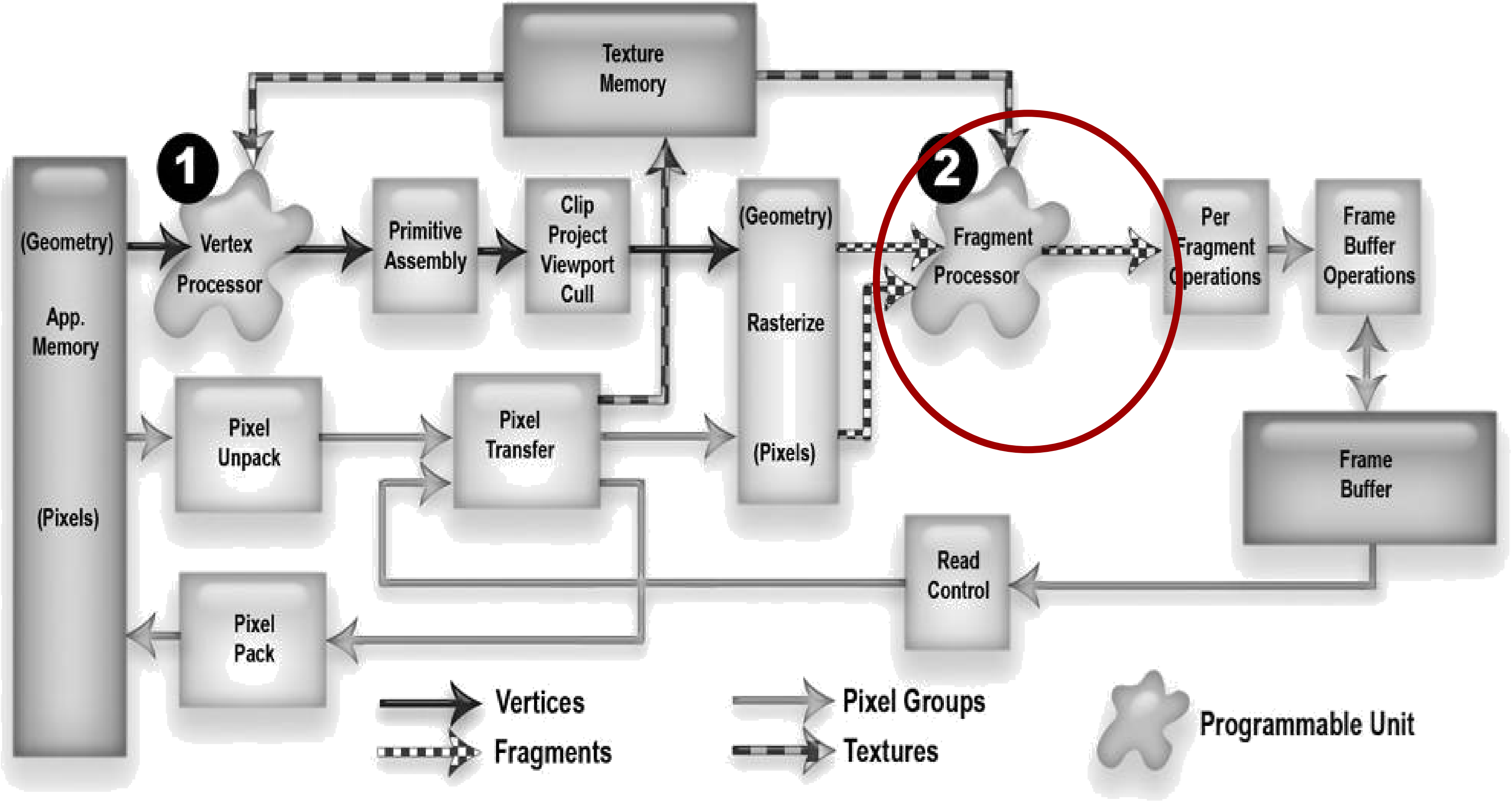
Normalment ho farà multiplicant el vèrtex per la matriu modelViewProjectionMatrix.

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 shaders

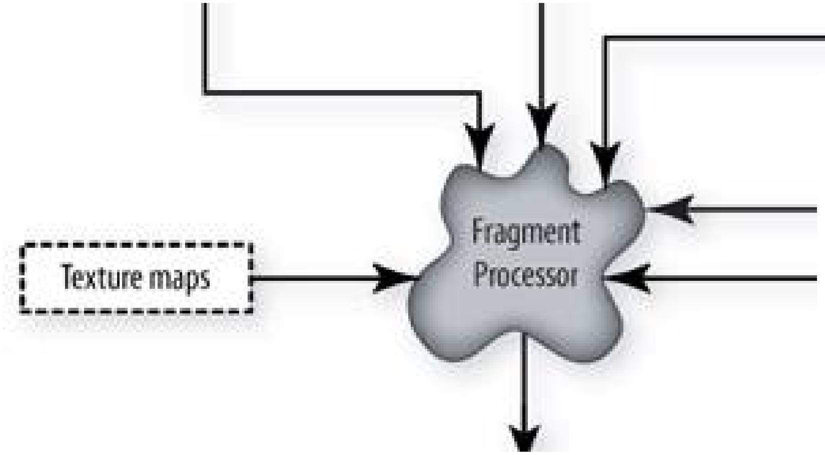


Fragment shader (3.3 core)

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)

EXEMPLES

VS per defecte al viewer

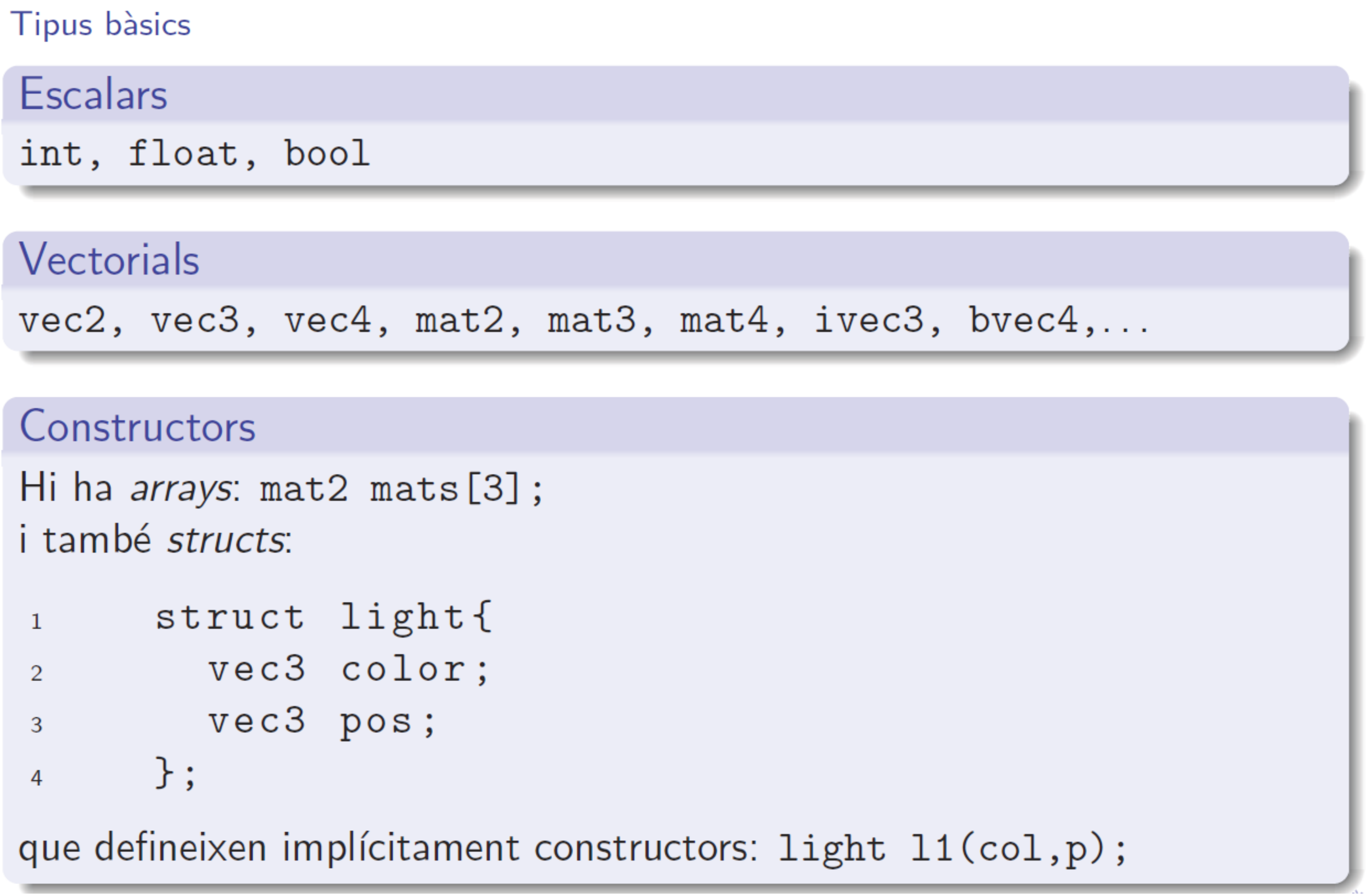
```
#version 330 core
layout (location = 0) in vec3 vertex;
layout (location = 1) in vec3 normal;
layout (location = 2) in vec3 color;
layout (location = 3) in vec2 texCoord;
out vec4 frontColor;
out vec2 vtexCoord;
uniform mat4 modelViewProjectionMatrix;
uniform mat3 normalMatrix;
void main() {
  vec3 N = normalize(normalMatrix * normal);
  gl_Position = modelViewProjectionMatrix * vec4(vertex.xyz, 1.0);
  frontColor = vec4(color,1.0) * N.z;
  vtexCoord = texCoord; }
```

FS per defecte al viewer

```
#version 330 core

in vec4 frontColor;
out vec4 fragColor;

void main()
{
	fragColor = frontColor;
}
```

LLENGUATGE GLSL

Elements del llenguatge GLSL



Tipus bàsics

Escalars

`int, float, bool`

Vectorials

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

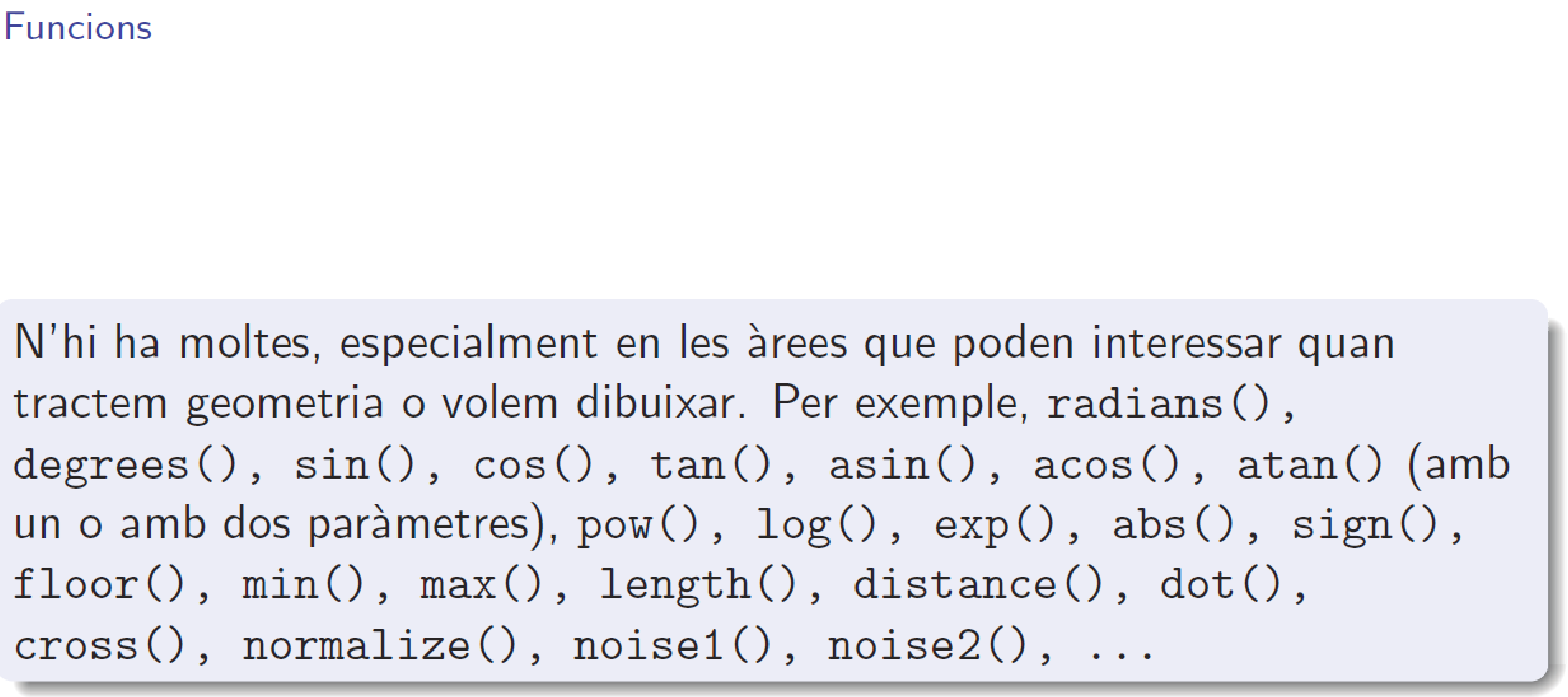
Constructors

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

```
    struct light{
      vec3 color;
      vec3 pos;
    };
```

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

Elements del llenguatge GLSL

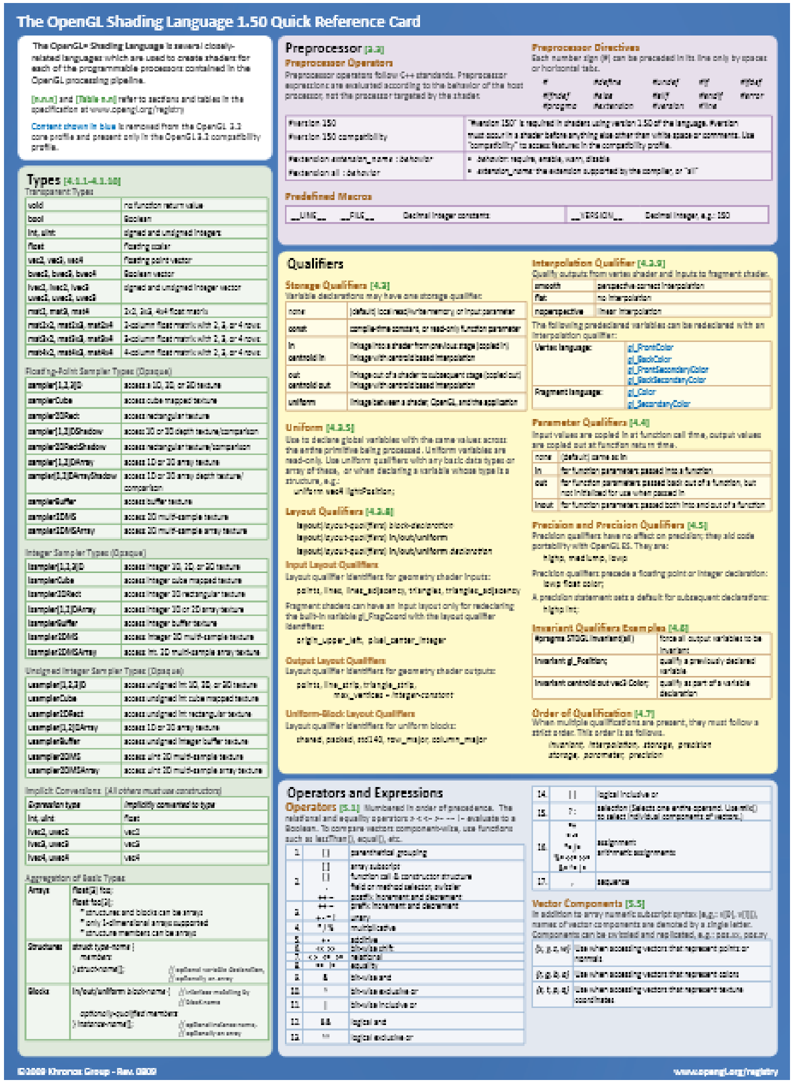


Funcions

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()`, ...

OpenGL Quick Reference card

https://www.khronos.org/files/opengl-quick-reference-card.pdf



The OpenGL Shading Language 1.50 Quick Reference Card

The OpenGL® Shading Language is several closely-related languages which are used to create shaders for each of the programmable processors contained in the OpenGL processing pipeline.

[n.n.n] and [Table n.n] refer to sections and tables in the specification at www.opengl.org/registry

Content shown in blue is removed from the OpenGL 3.2 core profile and present only in the OpenGL 3.2 compatibility profile.

Types [4.1.1-4.1.10]

Transparent Types

void	no function return value
bool	Boolean
int, uint	signed and unsigned integers
float	floating scalar
vec2, vec3, vec4	floating point vector
bvec2, bvec3, bvec4	Boolean vector
ivec2, ivec3, ivec4 uvec2, uvec3, uvec4	signed and unsigned integer vector
mat2, mat3, mat4	2x2, 3x3, 4x4 float matrix
mat2x2, mat2x3, mat2x4	2-column float matrix with 2, 3, or 4 rows
mat3x2, mat3x3, mat3x4	3-column float matrix with 2, 3, or 4 rows
mat4x2, mat4x3, mat4x4	4-column float matrix with 2, 3, or 4 rows

Floating-Point Sampler Types (Opaque)

sampler[1,2,3]D	access a 1D, 2D, or 3D texture
samplerCube	access cube mapped texture
sampler2DRect	access rectangular texture
sampler[1,2]DShadow	access 1D or 2D depth texture/comparison
sampler2DRectShadow	access rectangular texture/comparison
sampler[1,2]DArray	access 1D or 2D array texture
sampler[1,2]DArrayShadow	access 1D or 2D array depth texture/comparison
samplerBuffer	access buffer texture
sampler2DMS	access 2D multi-sample texture
sampler2DMSArray	access 2D multi-sample array texture

Integer Sampler Types (Opaque)

isampler[1,2,3]D	access integer 1D, 2D, or 3D texture
isamplerCube	access integer cube mapped texture
isampler2DRect	access integer 2D rectangular texture
isampler[1,2]DArray	access integer 1D or 2D array texture
isamplerBuffer	access integer buffer texture
isampler2DMS	access integer 2D multi-sample texture
isampler2DMSArray	access int. 2D multi-sample array texture

Unsigned Integer Sampler Types (Opaque)

usampler[1,2,3]D	access unsigned int 1D, 2D, or 3D texture
usamplerCube	access unsigned int cube mapped texture
usampler2DRect	access unsigned int rectangular texture
usampler[1,2]DArray	access 1D or 2D array texture
usamplerBuffer	access unsigned integer buffer texture
usampler2DMS	access uint 2D multi-sample texture
usampler2DMSArray	access uint 2D multi-sample array texture

Implicit Conversions (All others must use constructors)

Expression type	Implicitly converted to type
int, uint	float
ivec2, uvec2	vec2
ivec3, uvec3	vec3
ivec4, uvec4	vec4

Aggregation of Basic Types

Arrays	float[3] foo; float foo[3]; * structures and blocks can be arrays * only 1-dimensional arrays supported * structure members can be arrays

Structures	struct type-name {
	members
	} struct-name[]; // optional variable declaration, // optionally an array

Blocks	in/out/uniform block-name { // interface matching by // block-name
	optionally-qualified members
	} instance-name[]; // optional instance name, // optionally an array

Preprocessor [3.3]

Preprocessor Operators

Preprocessor operators follow C++ standards. Preprocessor expressions are evaluated according to the behavior of the host processor, not the processor targeted by the shader.

Preprocessor Directives

Each number sign (#) can be preceded in its line only by spaces or horizontal tabs.

#	#define	#undef	#if	#ifdef
#ifndef	#else	#elif	#endif	#error
#pragma	#extension	#version	#line	

#version 150 #version 150 compatibility	"#version 150" is required in shaders using version 1.50 of the language. #version must occur in a shader before anything else other than white space or comments. Use "compatibility" to access features in the compatibility profile.
#extension extension_name : behavior #extension all : behavior	• behavior: require, enable, warn, disable • extension_name: the extension supported by the compiler, or "all"

Predefined Macros

__LINE__	__FILE__	Decimal integer constants	__VERSION__	Decimal integer, e.g.: 150

Qualifiers

Storage Qualifiers [4.3]

Variable declarations may have one storage qualifier.

none	(default) local read/write memory, or input parameter
const	compile-time constant, or read-only function parameter
in centroid in	linkage into a shader from previous stage (copied in) linkage with centroid based interpolation
out centroid out	linkage out of a shader to subsequent stage (copied out) linkage with centroid based interpolation
uniform	linkage between a shader, OpenGL, and the application

Uniform [4.3.5]

Use to declare global variables with the same values across the entire primitive being processed. Uniform variables are read-only. Use uniform qualifiers with any basic data types or array of these, or when declaring a variable whose type is a structure, e.g.:

```
uniform vec4 lightPosition;
```

Layout Qualifiers [4.3.8]

```
layout(layout-qualifiers) block-declaration
layout(layout-qualifiers) in/out/uniform
layout(layout-qualifiers) in/out/uniform declaration
```

Input Layout Qualifiers

Layout qualifier identifiers for geometry shader inputs:

points, lines, lines_adjacency, triangles, triangles_adjacency

Fragment shaders can have an input layout only for redeclaring the built-in variable gl_FragCoord with the layout qualifier identifiers:

origin_upper_left, pixel_center_integer

Output Layout Qualifiers

Layout qualifier identifiers for geometry shader outputs:

points, line_strip, triangle_strip,
max_vertices = integer-constant

Uniform-Block Layout Qualifiers

Layout qualifier identifiers for uniform blocks:

shared, packed, std140, row_major, column_major

Interpolation Qualifier [4.3.9]

Qualify outputs from vertex shader and inputs to fragment shader.

smooth	perspective correct interpolation
flat	no interpolation
noperspective	linear interpolation

The following predeclared variables can be redeclared with an interpolation qualifier:

Vertex language:	gl_FrontColor gl_BackColor gl_FrontSecondaryColor gl_BackSecondaryColor
Fragment language:	gl_Color gl_SecondaryColor

Parameter Qualifiers [4.4]

Input values are copied in at function call time, output values are copied out at function return time.

none	(default) same as in
in	for function parameters passed into a function
out	for function parameters passed back out of a function, but not initialized for use when passed in
inout	for function parameters passed both into and out of a function

Precision and Precision Qualifiers [4.5]

Precision qualifiers have no effect on precision; they aid code portability with OpenGL ES. They are:

highp, mediump, lowp

Precision qualifiers precede a floating point or integer declaration:

lowp float color;

A precision statement sets a default for subsequent declarations:

highp int;

Invariant Qualifiers Examples [4.6]

#pragma STDGL invariant(all)	force all output variables to be invariant
invariant gl_Position;	qualify a previously declared variable
invariant centroid out vec3 Color;	qualify as part of a variable declaration

Order of Qualification [4.7]

When multiple qualifications are present, they must follow a strict order. This order is as follows.

invariant, interpolation, storage, precision
storage, parameter, precision

Operators and Expressions

Operators [5.1]

Numbered in order of precedence. The relational and equality operators > < <= >= == != evaluate to a Boolean. To compare vectors component-wise, use functions such as lessThan(), equal(), etc.

1.	( )	parenthetical grouping
2.	[ ] ( ) . ++ --	array subscript function call & constructor structure field or method selector, swizzler postfix increment and decrement
3.	++ -- + - ~ !	prefix increment and decrement unary
4.	* / %	multiplicative
5.	+ -	additive
6.	<< >>	bit-wise shift
7.	< > <= >=	relational
8.	== !=	equality
9.	&	bit-wise and
10.	^	bit-wise exclusive or
11.	|	bit-wise inclusive or
12.	&&	logical and
13.	^^	logical exclusive or
14.	||	logical inclusive or
15.	? :	selection (Selects one entire operand. Use mix() to select individual components of vectors.)
16.	= += -= *= /= %= <<= >>= &= ^= |=	assignment arithmetic assignments
17.	,	sequence

Vector Components [5.5]

In addition to array numeric subscript syntax (e.g.: v[0], v[i]), names of vector components are denoted by a single letter. Components can be swizzled and replicated, e.g.: pos.xx, pos.zy

{x, y, z, w}	Use when accessing vectors that represent points or normals
{r, g, b, a}	Use when accessing vectors that represent colors
{s, t, p, q}	Use when accessing vectors that represent texture coordinates

©2009 Khronos Group - Rev. 0809

www.opengl.org/registry

EXEMPLE: 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);
}
```

MISC

Configuració de **gedit**

- Activar syntax highlighting per GLSL 3.30:

```
mkdir  ~/.local/share/gtksourceview-3.0/
mkdir ~/.local/share/gtksourceview-3.0/language-specs
cp   /assig/grau-g/glsl330.lang  ~/.local/share/gtksourceview-3.0/language-specs/
mkdir  ~/.config/
mkdir  ~/.config/gedit/
mkdir  ~/.config/gedit/snippets/
cp /assig/grau-g/glsl.xml    ~/.config/gedit/snippets/glsl.xml
```

o directament:

/assig/grau-g/gedit-config

- Activar el plugin “snippets” del gedit (**Preferences→Plugins→Snippets**)
- El plugin del gedit fa que defs[TAB] s’expandeixi a les declaracions de tots els uniforms que envia el viewer