OpenGL programming

- OpenGL 4 - The basis -

Jonathan Fabrizio

http://jo.fabrizio.free.fr

Version : Tue Feb 13 10:26:22 2024

OpenGL

OpenGL

Introduction

OpenGL

GL (1992) initially developed by SGI and now maintained by the Khronos Group

- Allows to render images thanks to rasterization
- OpenGL is a specification
- The previous version (N-1) of GL is free (OpenGL)
- A free implementation: MESA

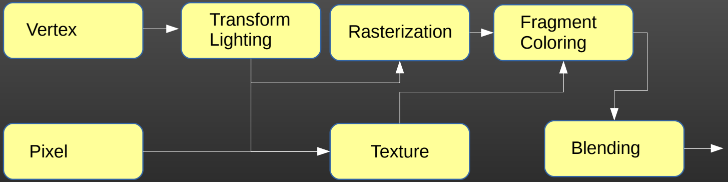
OpenGL

OpenGL:

- Does not Manage user interactions
- Does not manage shadows
- Initially embed Lambert and Gouraud models
  - From the version 3, does not manage lighting anymore
- Does not embed any physical model
- Does not manage collision detection
- Does not prepare cafe but may offer you a teapot

OpenGL: Old ages

Fixed graphical pipeline (OpenGL <2.0)



OpenGL: Old ages

```
glBegin();
  glVertex();
  glNormal();
  glColor();
glEnd();

glMatrixMode();
glPushMatrix();
glTranslated();
glFrustum();
```

```
glGenLists();
glNewList();
glEndList();
glCallList();
```

```
glShadeModel();
glEnable(GL_LIGHT
glEnable(GL_LIGHT
```

OpenGL: Old ages

```
glBegin();
  glVertex();
  glNormal();
  glColor();
glEnd();

glMatrixMode();
glPushMatrix();
glTranslated();
glFrustum();
```

Deprecated !

```
glGenLists();
glNewList();
glEndList();
glCallList();
```

```
glShadeModel();
glEnable(GL_LIGHT
glEnable(GL_LIGHT
```

OpenGL: Old ages

```
glBegin();
  glVertex();
  glNormal();
  glColor();
glEnd();

glMatrixMode();
glPushMatrix();
glTranslated();
glFrustum();
```

Deprecated !

```
glGenLists();
glNewList();
glEndList();
glCallList();
```

Deprecated !

```
glShadeModel();
glEnable(GL_LIGHT
glEnable(GL_LIGHT
```

OpenGL: Old ages

```
glBegin();
  glVertex();
  glNormal();
  glColor();
glEnd();

glMatrixMode();
glPushMatrix();
glTranslated();
glFrustum();
```

Deprecated !

```
glGenLists();
glNewList();
glEndList();
glCallList();
```

Deprecated !

```
glShadeModel();
glEnable(GL_LIGHT
glEnable(GL_LIGHT
```

Deprecated !

OpenGL: Old ages

```
glBegin();
  glVertex();
  glNormal();
  glColor();
glEnd();
```

Deprecated !

```
glMatrixMode();
glPushMatrix();
glTranslated();
glFrustum();
```

Deprecated !

```
glGenLists();
glNewList();
glEndList();
glCallList();
```

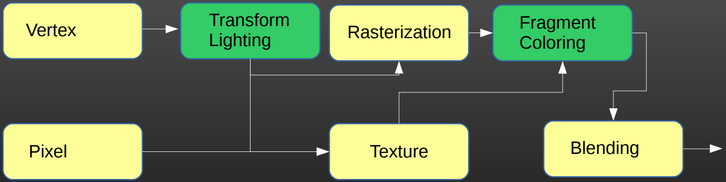
Deprecated !

```
glShadeModel();
glEnable(GL_LIGHT
glEnable(GL_LIGHT
```

Deprecated !

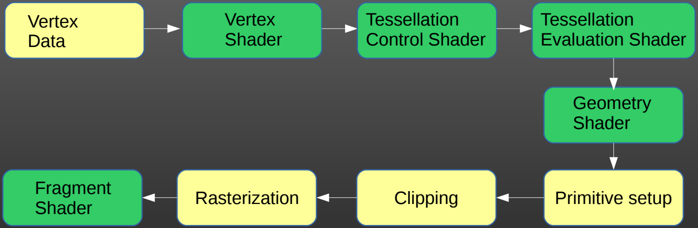
OpenGL 2.0: The end of the old ages

Fixed pipeline still accessible but introduction of vertex and fragment shaders.



OpenGL 3.0: The start of a new era

Exclusively programmable pipeline (OpenGL >=3.1)

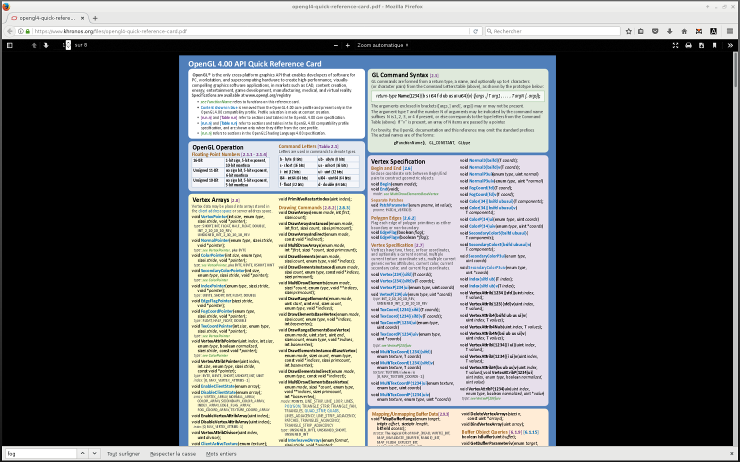


- Vertex shader
- Pixel shader
- Almost all data is GPU-resident

OpenGL 3.0: The start of a new era

- OpenGL 3.2
  - Geometry shader Allows modification/generation of shapes
- OpenGL 4.1
  - Tessellation-control shader
  - Tessellation-evaluation shader
- OpenGL 4.3/4.5
  - Compute shader

OpenGL: Respect the current specification


opengl4-quick-reference-card.pdf - Mozilla Firefox
opengl4-quick-refere...
https://www.khronos.org/files/opengl4-quick-reference-card.pdf
Rechercher
1 sur 8
Zoom automatique
fog
Tout surligner
Respecter la casse
Mots entiers

OpenGL 4.00 API Quick Reference Card

OpenGL® is the only cross-platform graphics API that enables developers of software for PC, workstation, and supercomputing hardware to create high-performance, visually-compelling graphics software applications, in markets such as CAD, content creation, energy, entertainment, game development, manufacturing, medical, and virtual reality. Specifications are available at www.opengl.org/registry

- *see FunctionName* refers to functions on this reference card.
- Content shown in blue is removed from the OpenGL 4.00 core profile and present only in the OpenGL 4.00 compatibility profile. Profile selection is made at context creation.
- [n.n.n] and [Table n.n] refer to sections and tables in the OpenGL 4.00 core specification.
- [n.n.n] and [Table n.n] refer to sections and tables in the OpenGL 4.00 compatibility profile specification, and are shown only when they differ from the core profile.
- [n.n.n] refers to sections in the OpenGL Shading Language 4.00 specification.

GL Command Syntax [2.3]

GL commands are formed from a return type, a name, and optionally up to 4 characters (or character pairs) from the Command Letters table (above), as shown by the prototype below:

```
return-type Name{1234}{b s i i64 f d ub us ui ui64}{v} ([args ,] T arg1 , . . . , T argN [, args]);
```

The arguments enclosed in brackets ([args ,] and [, args]) may or may not be present.

The argument type T and the number N of arguments may be indicated by the command name suffixes. N is 1, 2, 3, or 4 if present, or else corresponds to the type letters from the Command Table (above). If "v" is present, an array of N items are passed by a pointer.

For brevity, the OpenGL documentation and this reference may omit the standard prefixes. The actual names are of the forms:

glFunctionName(), GL_CONSTANT, GLtype

OpenGL Operation

Floating-Point Numbers [2.1.1 - 2.1.4]

16-Bit	1-bit sign, 5-bit exponent, 10-bit mantissa
Unsigned 11-Bit	no sign bit, 5-bit exponent, 6-bit mantissa
Unsigned 10-Bit	no sign bit, 5-bit exponent, 5-bit mantissa

Command Letters [Table 2.1]

Letters are used in commands to denote types.

b - byte (8 bits)	ub - ubyte (8 bits)
s - short (16 bits)	us - ushort (16 bits)
i - int (32 bits)	ui - uint (32 bits)
i64 - int64 (64 bits)	ui64 - uint64 (64 bits)
f - float (32 bits)	d - double (64 bits)

Vertex Arrays [2.8]

Vertex data may be placed into arrays stored in the client address space or server address space.

```
void VertexPointer(int size, enum type,
    sizei stride, void *pointer);
  type: SHORT, INT, FLOAT, HALF_FLOAT, DOUBLE,
    INT_2_10_10_10_REV,
    UNSIGNED_INT_2_10_10_10_REV
void NormalPointer(enum type, sizei stride,
    void *pointer);
  type: see VertexPointer, plus BYTE
void ColorPointer(int size, enum type,
    sizei stride, void *pointer);
  type: see VertexPointer, plus BYTE, UBYTE, USHORT, UINT
void SecondaryColorPointer(int size,
    enum type, sizei stride, void *pointer);
  type: see ColorPointer
void IndexPointer(enum type, sizei stride,
    void *pointer);
  type: UBYTE, SHORT, INT, FLOAT, DOUBLE
void EdgeFlagPointer(sizei stride,
    void *pointer);
void FogCoordPointer(enum type,
    sizei stride, void *pointer);
  type: FLOAT, HALF_FLOAT, DOUBLE
void TexCoordPointer(int size, enum type,
    sizei stride, void *pointer);
  type: see VertexPointer
void VertexAttribPointer(uint index, int size,
    enum type, boolean normalized,
    sizei stride, const void *pointer);
  type: see ColorPointer
void VertexAttribIPointer(uint index,
    int size, enum type, sizei stride,
    const void *pointer);
  type: BYTE, UBYTE, SHORT, USHORT, INT, UINT
  index: [0, MAX_VERTEX_ATTRIBS - 1]
void EnableClientState(enum array);
void DisableClientState(enum array);
  array: VERTEX_ARRAY, NORMAL_ARRAY,
    COLOR_ARRAY, SECONDARY_COLOR_ARRAY,
    INDEX_ARRAY, EDGE_FLAG_ARRAY,
    FOG_COORD_ARRAY, TEXTURE_COORD_ARRAY
void EnableVertexAttribArray(uint index);
void DisableVertexAttribArray(uint index);
  index: [0, MAX_VERTEX_ATTRIBS - 1]
void VertexAttribDivisor(uint index,
    uint divisor);
void ClientActiveTexture(enum texture);
```

```
void PrimitiveRestartIndex(uint index);
```

Drawing Commands [2.8.2] [2.8.3]

```
void DrawArrays(enum mode, int first,
    sizei count);
void DrawArraysInstanced(enum mode,
    int first, sizei count, sizei primcount);
void DrawArraysIndirect(enum mode,
    const void *indirect);
void MultiDrawArrays(enum mode,
    int *first, sizei *count, sizei primcount);
void DrawElements(enum mode,
    sizei count, enum type, void *indices);
void DrawElementsInstanced(enum mode,
    sizei count, enum type, const void *indices,
    sizei primcount);
void MultiDrawElements(enum mode,
    sizei *count, enum type, void **indices,
    sizei primcount);
void DrawRangeElements(enum mode,
    uint start, uint end, sizei count,
    enum type, void *indices);
void DrawElementsBaseVertex(enum mode,
    sizei count, enum type, void *indices,
    int basevertex);
void DrawRangeElementsBaseVertex(
    enum mode, uint start, uint end,
    sizei count, enum type, void *indices,
    int basevertex);
void DrawElementsInstancedBaseVertex(
    enum mode, sizei count, enum type,
    const void *indices, sizei primcount,
    int basevertex);
void DrawElementsIndirect(enum mode,
    enum type, const void *indirect);
void MultiDrawElementsBaseVertex(
    enum mode, sizei *count, enum type,
    void **indices, sizei primcount,
    int *basevertex);
  mode: POINTS, LINE_STRIP, LINE_LOOP, LINES,
    POLYGON, TRIANGLE_STRIP, TRIANGLE_FAN,
    TRIANGLES, QUAD_STRIP, QUADS,
    LINES_ADJACENCY, LINE_STRIP_ADJACENCY,
    PATCHES, TRIANGLES_ADJACENCY,
    TRIANGLE_STRIP_ADJACENCY
  type: UNSIGNED_BYTE, UNSIGNED_SHORT,
    UNSIGNED_INT
void InterleavedArrays(enum format,
```

Vertex Specification

Begin and End [2.6]

Enclose coordinate sets between Begin/End pairs to construct geometric objects.

```
void Begin(enum mode);
void End(void);
  mode: see MultiDrawElementsBaseVertex
```

Separate Patches

```
void PatchParameteri(enum pname, int value);
  pname: PATCH_VERTICES
```

Polygon Edges [2.6.2]

Flag each edge of polygon primitives as either boundary or non-boundary.

```
void EdgeFlag(boolean flag);
void EdgeFlagv(boolean *flag);
```

Vertex Specification [2.7]

Vertices have two, three, or four coordinates, and optionally a current normal, multiple current texture coordinate sets, multiple current generic vertex attributes, current color, current secondary color, and current fog coordinates.

```
void Vertex{234}{sifd}(T coords);
void Vertex{234}{sifd}v(T coords);
void VertexP{234}ui(enum type, uint coords)
void VertexP{234}uiv(enum type, uint *coords)
  type: INT_2_10_10_10_REV,
    UNSIGNED_INT_2_10_10_10_REV
void TexCoord{1234}{sifd}(T coords);
void TexCoord{1234}{sifd}v(T coords);
void TexCoordP{1234}ui(enum type,
    uint coords)
void TexCoordP{1234}uiv(enum type,
    uint *coords)
  type: see VertexP{234}uiv
void MultiTexCoord{1234}{sifd}(
    enum texture, T coords)
void MultiTexCoord{1234}{sifd}v(
    enum texture, T coords)
  texture: TEXTUREi (where i is
    [0, MAX_TEXTURE_COORDS - 1])
void MultiTexCoordP{1234}ui(enum texture,
    enum type, uint coords)
void MultiTexCoordP{1234}uiv(
    enum texture, enum type, uint *coords)
```

```
void Normal3{bsifd}(T coords);
void Normal3{bsifd}v(T coords);
void NormalP3ui(enum type, uint normal)
void NormalP3uiv(enum type, uint *normal)
void FogCoord{fd}(T coord);
void FogCoord{fd}v(T coord);
void Color{34}{bsifd ubusui}(T components);
void Color{34}{bsifd ubusui}v(
    T components);
void ColorP{34}ui(enum type, uint coords)
void ColorP{34}uiv(enum type, uint *coords)
void SecondaryColor3{bsifd ubusui}(
    T components);
void SecondaryColor3{bsifd ubusui}v(
    T components);
void SecondaryColorP3ui(enum type,
    uint coords)
void SecondaryColorP3uiv(enum type,
    uint *coords)
void Index{sifd ub}(T index);
void Index{sifd ub}v(T index);
void VertexAttrib{1234}{sfd}(uint index,
    T values);
void VertexAttrib{123}{sfd}v(uint index,
    T values);
void VertexAttrib4{bsifd ub us ui}v(
    uint index, T values);
void VertexAttrib4Nub(uint index, T values);
void VertexAttrib4N{bsi ub us ui}v(
    uint index, T values);
void VertexAttribI{1234}{i ui}(uint index,
    T values);
void VertexAttribI{1234}{i ui}v(uint index,
    T values);
void VertexAttribI4{bs ub us}v(uint index,
    T values);void VertexAttribP{1234}ui(
    uint index, enum type, boolean normalized,
    uint value)
void VertexAttribP{1234}uiv(uint index,
    enum type, boolean normalized, uint *value)
  type: see VertexP{234}uiv
```

Mapping/Unmapping Buffer Data [2.9.3]

```
void *MapBufferRange(enum target,
    intptr offset, sizeiptr length,
    bitfield access);
  access: The logical OR of MAP_{READ, WRITE}_BIT,
    MAP_INVALIDATE_{BUFFER, RANGE}_BIT,
    MAP_FLUSH_EXPLICIT_BIT,
```

```
void DeleteVertexArrays(sizei n,
    const uint *arrays);
void BindVertexArray(uint array);
```

Buffer Object Queries [6.1.9] [6.1.15]

```
boolean IsBuffer(uint buffer);
void GetBufferParameteriv(enum target,
```

source : www.khronos.org

Structure

CPU
RAM

GPU
Shaders (GLSL)
VRAM

OpenGL

General scheme

General scheme

Simplified scheme:

1. perform initialization,
2. compile shaders
3. initialization of data,
4. active the correct *programs* (*shaders*), and send data throw the graphical pipeline
5. get back the image

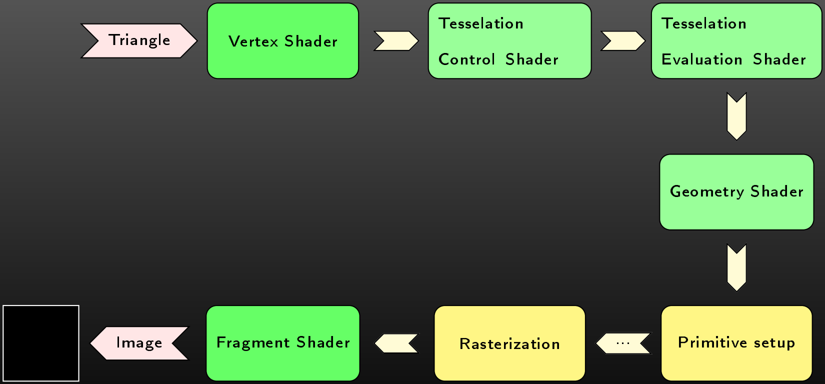
General scheme

1. Perform initialization OpenGL manages a lot of features but we have to initialize/activate them: *Z-buffer*, *Back face culling*...

General scheme

2. Compile the *shaders* and link the *programs*

Shaders are software programs that run directly into the graphical pipeline. They are linked into a *program*



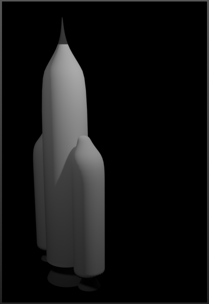
They are written in *GLSL*

General scheme

Initialize data

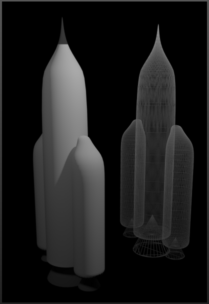
General scheme

Initialize data



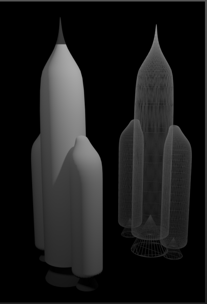
General scheme

Initialize data



General scheme

Initialize data



Store mesh into a buffer (the VBO[a])

- vertices
- colors
- texture coordinates
- normal vectors
- ...

Explain how data is organized into buffers

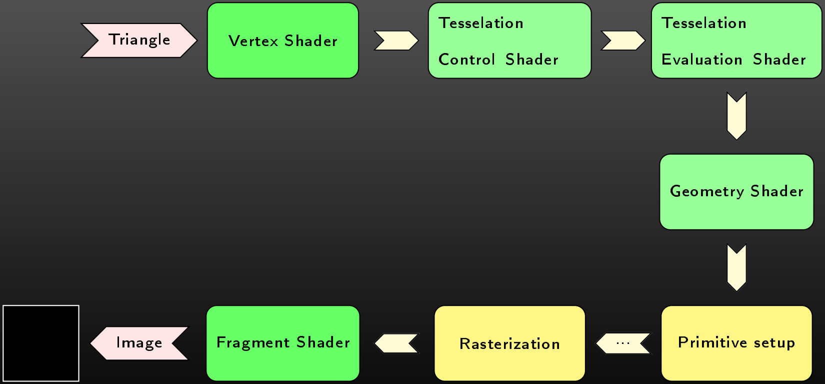
[a]Vertex Buffer Object

General scheme

4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline.

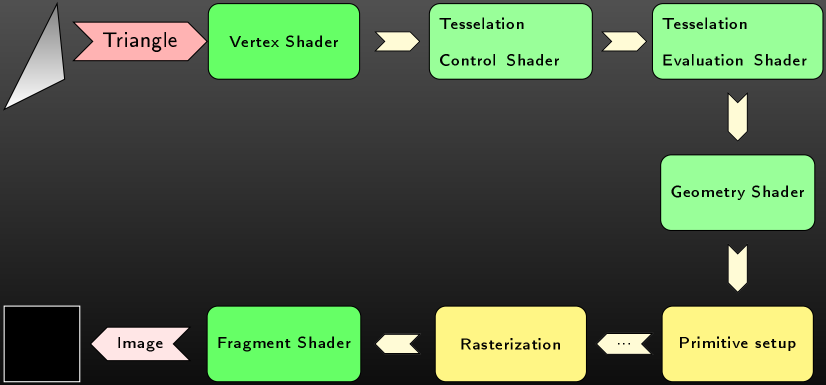
General scheme

4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline:
Activate the correct *program*.



General scheme

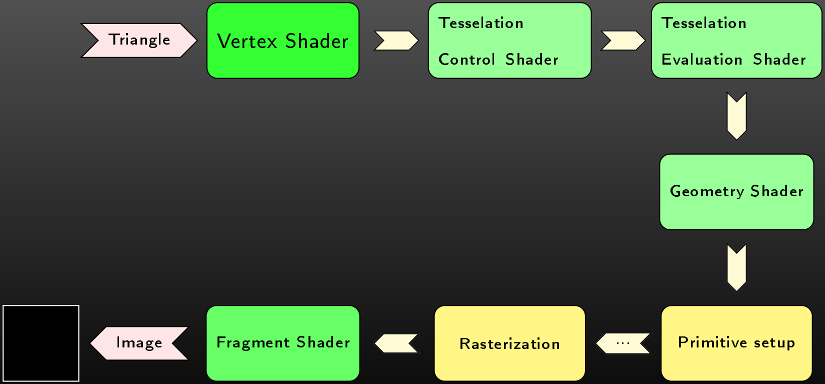
4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline:
Send the data of the mesh



General scheme

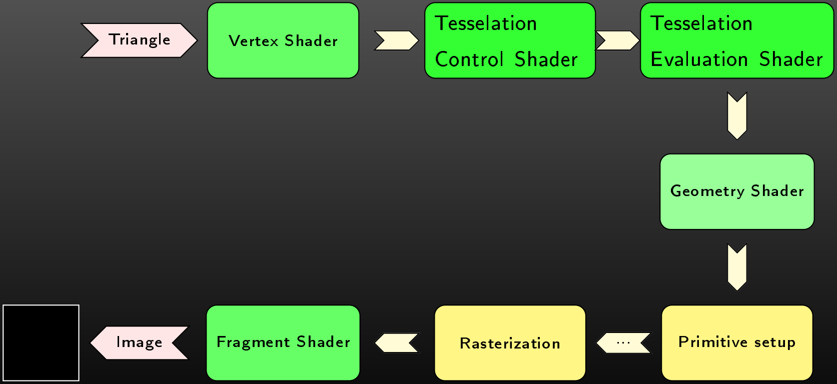
4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline:

Vertex Shader - Coordinate system conversion to prepare the projection



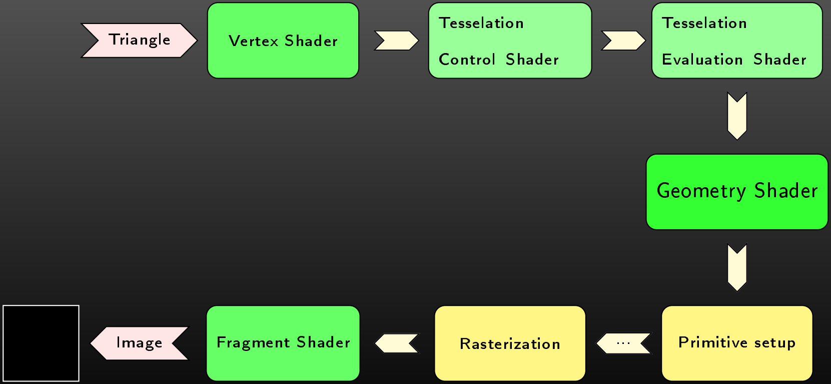
General scheme

4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline:
Tesselation Shaders - Refine the mesh (Optionnel)



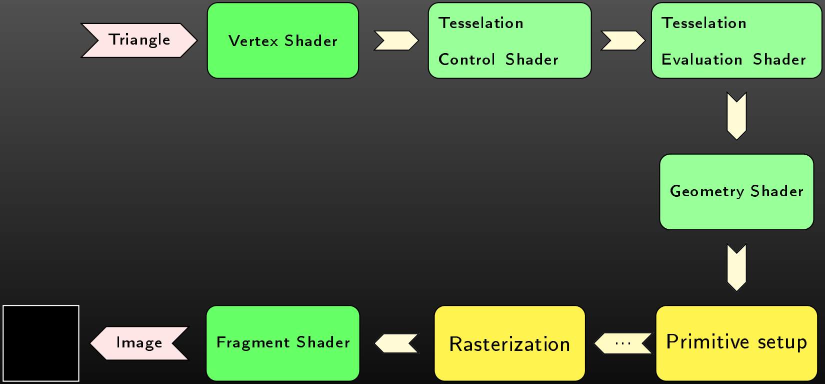
General scheme

4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline:
Geometry Shader - Change the type/refine primitives (Optionnel)



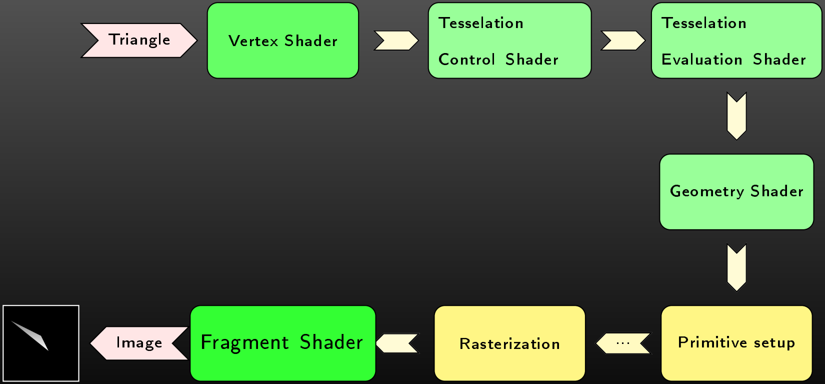
General scheme

4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline:
Primitive setup/Rasterization - Prepare to draw.



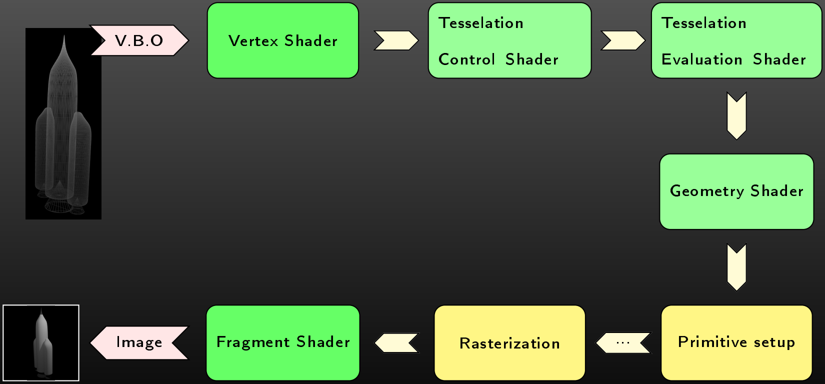
General scheme

4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline:
Fragment Shader - Draw a fragment.



General scheme

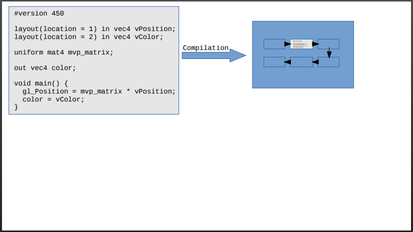
4. Activate the correct *progam* (with *shaders*) then send data (buffers) to the graphical pipeline:
Get back the image.



General scheme

GLSL: The language used to write shaders

- Compiled for your video card.

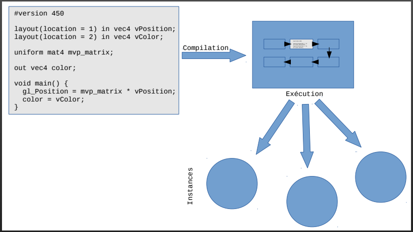


```
#version 450

layout(location = 1) in vec4 vPosition;
layout(location = 2) in vec4 vColor;

uniform mat4 mvp_matrix;

out vec4 color;

void main() {
  gl_Position = mvp_matrix * vPosition;
  color = vColor;
}
```

Execution/Instances

Shaders execution

- Single Instruction Multiple Instances



```
#version 450

layout(location = 1) in vec4 vPosition;
layout(location = 2) in vec4 vColor;

uniform mat4 mvp_matrix;

out vec4 color;

void main() {
  gl_Position = mvp_matrix * vPosition;
  color = vColor;
}
```

Shaders execution

Many threads (but they all do the same)

```
...             ...             ...             ...             ...
if (...) {      if (...) {      if (...) {      if (...) {      if (...) {
  ...             ...             ...             ...             ...
  a=a*c; ⊗        a=a*c; ←        a=a*c; ⊗        a=a*c; ←        a=a*c; ←
  ...             ...             ...             ...             ...
}               }               }               }               }
...             ...             ...             ...             ...
```

GLSL

Language presentation

GLSL - Variables

Native data types

- scalar (limited): bool, int, uint, float, double
- vectors: bvecn, ivecn, uvecn, vecn, dvecn (n=2..4)
- matrices: matn, matnb, dmatn, dmatnm (n/m=2..4)
- samplers/images

Access

- vectors: ivec4 t ; t[2]/t.r/t.rgb/t.rgba/t.xy/t.xyz...
- operators: Matrix multiplication

Structures

- Arrays: vec3[5][2] multidim;
- Structures:

```
struct Light
{
  vec3 eyePosOrDir;
  bool isDirectional;
} variableName;
```

GLSL - instructions

- stream: if/switch
- loop: for/while/do while
- functions: int fun(int i)
- prepro: #define...
- and a lot of functions (clamp...)

Shaders - Data

- Shared among all instances (uniform)
  - variables *uniform*
  - UBO
  - SSBO
  - Textures (sampler)
  - Images
- specific for one instance
  - VBO
- Outputs
  - FBO
- Communication between shaders
  - in/out
  - shared

Shaders - Data

- Shared among all instances (uniform)
  - variables *uniform*
  - UBO
  - SSBO
  - Textures (sampler)
  - Images
- specific for one instance
  - VBO
- Outputs
  - FBO
- Communication between shaders
  - in/out
  - shared

Uniform

- Shared among all the instances
- Read-only on the GLSL side

On the CPU side
The address of the variable must be known (or queried `glGetUniformLocation()`)
Then you can assign the content: `glUniform*(location, value);`

On the GPU side
Declaration

```
uniform int v;
layout(location = 1) uniform float t;
```

Buffers

block of memory

- Declaration:

```
GLuint buffer_id;
glGenBuffers(1, &buffer_id);
```

- Activate/deactivate

```
glBindBuffer(--TYPE--, buffer_id);
glBindBuffer(--TYPE--, 0);
```

- Allocation :

```
glBufferData(...);
```

- Writing/modification:

```
glBufferData(...)
glMapBuffer(...)/glUnMapBuffer(...)
```

- destruction

```
glDeleteBuffers(1, &buffer_id);
```

Buffers

Base of FBOs, UBOs, TextureBuffer...

UBO: Uniform Buffer Object

Usage:

- Groups multiple *uniforms*
- Used in different programs and updated outside of the program.

Limitations:

- Few kb per blocks.
- Limited number of activated buffers simultaneously (GL_MAX_??_UNIFORM_BLOCKS).
- Limited (total) number of uniform buffers (GL_MAX_UNIFORM_BUFFER_BINDINGS).
- read only, on the GPU-size (in shaders)
- Restricted to possible types in GLSL
- Pre-defined length.

UBO: Uniform Buffer Object

Example:

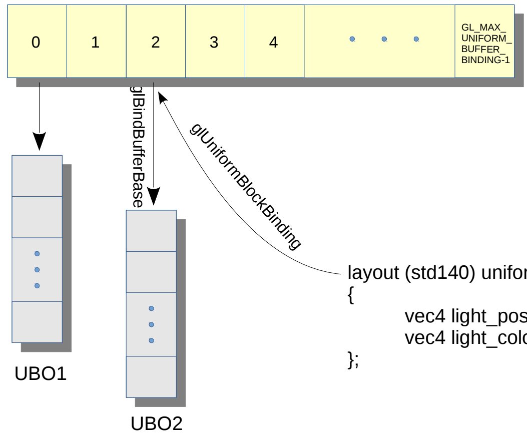
- We can substitute:

```
uniform vec4 light_position;
uniform vec4 light_color;
```

by:

```
layout (std140) uniform shader_data
{
  vec4 light_position;
  vec4 light_color;
};
```

UBO: Uniform Buffer Object



```
layout (std140) uniform shader_data
{
        vec4 light_position ;
        vec4 light_color ;
};
```

UBO: Uniform Buffer Object

Creation and activation:

- GLuint buffer_id;
  glGenBuffers(1, &buffer_id);
  glBindBuffer(GL_UNIFORM_BUFFER, buffer_id);

Bindind:

- glBindBufferBase(GL_UNIFORM_BUFFER, binding_point_index, buffer_id);

binding_point_index:

- set up in shader code (declaration)
- or set up by glUniformBlockBinding(program, uniform_bloc_index (glGetUniformBlockIndex), binding_point_index);

Usage:

- As any other buffer.

SSBO: Shader Storage Buffer Objects

Usage:

- Large blocks of data
- R/W access
- Size not necessarily pre-defined

Limitations:

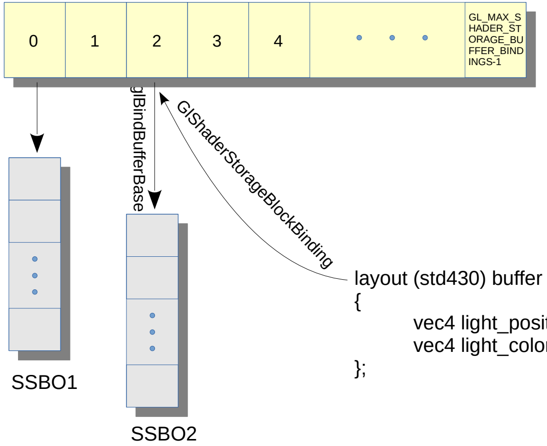
- Since OpenGL 4.3
- Max number of SSBOs (GL_MAX_SHADER_STORAGE_BUFFER_BINDINGS)
- Limite de taille (16Mo garanti, en pratique pas -> taille de la mémoire libre) (GL_MAX_SHADER_STORAGE_BLOCK_SIZE)
- Limited number of simultaneous activated buffers (L_MAX_???_SHADER_STORAGE_BLOCKS). +GL_MAX_COMBINED_SHADER_STORAGE_BLOCKS
- Restricted to allowed types in GLSL.
- In theory, a bit slower than UBOs.

SSBO: Shader Storage Buffer Objects

Definition in the shader:

```
layout (std430, binding = 1) buffer shader_data
{
  vec4 light_position;
  vec4 light_color;
};
```

SSBO : Shader Storage Buffer Objects


0
1
2
3
4
GL_MAX_SHADER_STORAGE_BUFFER_BINDINGS-1
glBindBufferBase
GlShaderStorageBlockBinding
SSBO1
SSBO2

```
layout (std430) buffer shader_data
{
        vec4 light_position ;
        vec4 light_color ;
};
```

SSBO: Shader Storage Buffer Objects

Creation and activation:

- GLuint ssbo_id;
  glGenBuffers(1, &ssbo_id);
  glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo_id);

Binding:

- glBindBufferBase(GL_SHADER_STORAGE_BUFFER, binding_point_index, ssbo_id);

binding_point_index:

- Set up in the code of the shader binding = xx
- or set up in by glShaderStorageBlockBinding(program, storage_bloc_index (glGetProgramResourceIndex), binding_point_index);

Usage:

- As any other buffer.

SSBO and UBO: data alignment

Be aware: padding may be added in UBOs or SSBOs (GPU side.).
The issues due to padding added to data in RAM is well know, there are also many rules on the padding (GPU-side).

CPU side

```
struct line {
    GLfloat old_pos[4];
    GLfloat old_color[4];
    GLfloat new_pos[4];
    GLfloat new_color[4];
};
```

GPU side

```
struct Line {
    vec4 old_pos;
    vec4 old_color;
    vec4 new_pos;
    vec4 new_color;
};
layout(std430, binding = 2) buffer line_buffer
{
    Line line_list[NB_PARTICLES];
};
```

SSBO and UBO: data alignment

Padding GPU side :

CPU side

```
struct line {
  GLfloat pos[3];
  GLfloat color[3];
  GLfloat prop1;
  GLfloat prop2
};
```

GPU side

```
struct Line {
  vec3 pos;
  *Padding 1 byte*
  vec3 color;
  float new_pos;
  float new_color;
};
```

The two structures do not match anymore.

SSBO and UBO: data alignment

Check for data alignment (GPU side)

CPU side

```
struct line {
  GLfloat pos[3];
  GLfloat prop1;
  GLfloat color[3];
  GLfloat prop2
};
```

GPU side

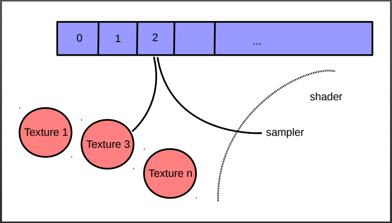
```
struct Line {
  vec3 pos;
  float prop1;
  vec3 color;
  float prop2;
};
```

The two structures match. Take care about the order of the fields.

Textures

- Declaration
  - glGenTextures(1, &id);
- Activate/deactivate
  - glBindTexture (...);
- Allocation
  - glTexStorage2D();//Best
  - glTexImage2D();//Declaration of the bitmap.
- Fill
  - glTexImage2D();
  - glTexSubImage2D();
  - also texture buffers
- Delete
  - glDeleteTextures (...);

Textures



- To use a texture, you have to bind this texture on a *texture unit* (glActivateTexture()).
- You must set the *sampler* of the *shader* to work on the correct *texture unit*

Textures

```
GLint tex1_loc = glGetUniformLocation(prog, "tex1_sampler");
glUniform1i(tex1_loc, 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, id_tex1);

GLint tex2_loc = glGetUniformLocation(prog, "tex2_sampler");
glUniform1i(tex1_loc, 1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, id_tex2);
```

```
uniform sampler2D tex1_sampler;
uniform sampler2D tex2_sampler;

...
vec4 texel = texture2D(tex1_sampler, interpolated_uv_position)
           +texture2D(tex2_sampler, interpolated_uv_position);
...
```

Textures

Attention :

- For a 2D texture, the origin is bottom-left
- The first bind sets the type of the texture (GL_TEXURE_2D, GL_TEXTURE_CUBE_MAP...). This state cannot be changed ever.
- To activate a *texture unit*: `glActiveTexture(GL_TEXTURE0 + i);` instead of `glActiveTexture(GL_TEXTUREi);` because there are not enough constants (until GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS)

Images

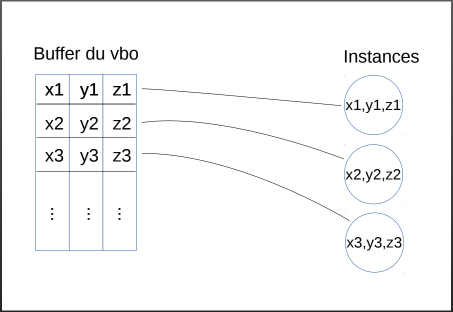
```
glBindImageTexture(image_unit_in_id0, texture_in_id0, 0, GL_FALSE, 0,
                   GL_READ_ONLY, GL_RGBA8UI);
glBindImageTexture(image_unit_out_id1, texture_out_id, 0, GL_FALSE, 0,
                   GL_WRITE_ONLY, GL_RGBA8UI);
```

```
#version 430 core

layout (local_size_x = 16, local_size_y = 16) in;

layout (rgba8ui, binding = 0) readonly uniform uimage2D input_image;
layout (rgba8ui, binding = 1) writeonly uniform uimage2D output_image;

void main(void) {
     ivec2 pos = ivec2(gl_GlobalInvocationID.xy);
     uvec4 result = imageLoad(input_image, pos)/2;
     imageStore(output_image, pos, result);
}
```

VBO

- Declare/free: glGenBuffers()/glDeleteBuffers()
- Activate: glBindBuffer(GL_ARRAY_BUFFER, ...)
- Copy data glBufferData()/glMapBuffer()
- Link with your shader glVertexAttribPointer()
- Activate link/data sending glEnableVertexAttribArray()
- Encapsulated in a *Vertex Array Object* - VAO ! **(it is mandatory)**

VBO


Buffer du vbo
x1 y1 z1
x2 y2 z2
x3 y3 z3
⋮ ⋮ ⋮
Instances
x1,y1,z1
x2,y2,z2
x3,y3,z3

VBO

```
glVertexAttribPointer(location, nb_comp, type, normalize, stride, offset);
```

xyzxyzxyzxyz...xyz
rgbrgbrgbrgb...rgb
stststststst...st

```
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);%//xyz
```

stride = 0 => consecutive data

VBO

```
glVertexAttribPointer(location, nb_comp, type, normalize, stride, offset);
```

xyzxyzxyzxyz...xyzrgbrgbrgbrgb...rgbstststststst...st

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                      0, (void*)0);%//xyz
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
                      0, (void*)3*nb_vertices*sizeof(GLfloat)); //rgb
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE,
                      0, (void*)2*3*nb_vertices*sizeof(GLfloat)); //st
```

xyzrgbstxyzrgbst...xyzrgbst

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                      8*sizeof(GLfloat), (void*)0);%//xyz
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
                      8*sizeof(GLfloat), (void*)3*sizeof(GLfloat)); //rg
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE,
                      8*sizeof(GLfloat), (void*)6*sizeof(GLfloat)); //s
```

Note: warning, `glVertexAttribIPointer (...) != glVertexAttribPointer (..., GL_INT, ...)`

VBO/VAO

Vertex Array Object
Encapsulate VBOs (**mandatory un OpenGL 4 !!!**)

```
glGenVertexArrays(...);
glBindVertexArray(...);
glEnableVertexAttribArray(...);
glBindVertexArray(...);
```

glDeleteVertexArrays(...)
VAO 0 -> static VAO ; Default value for `glDisableVertexAttribArray();`

VBO

```
glBindVertexArray(vao_id);
glDraw...
glBindVertexArray(0);
```

VBO

Many questions to answer:

- Only one VBO or many VBOs for the same mesh
- Interleaved data or sequential blocks
- glBufferData()/glMapBuffer() ?

Best practices

- Uniform data forma as much as possible.
- Reduce as possible the length of the data.
- When possible, use indexed access.

VBO

VBOs can be filled:

- on CPU side, as any other buffer (GL_ARRAY_BUFFER)
- on GPU side, using a compute shader for example.

It is possible to invoke glBindBuffer(GL_SHADER_STORAGE_BUFFER, ...) and later glBindBuffer(GL_ARRAY_BUFFER, ...) over the same buffer.

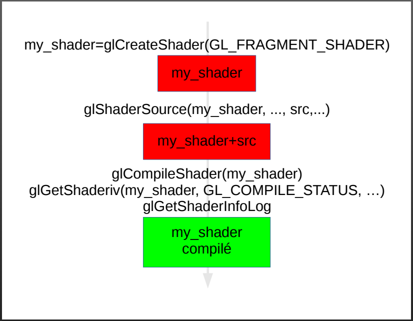
Shaders - Communication between shaders

- in/out
- shared

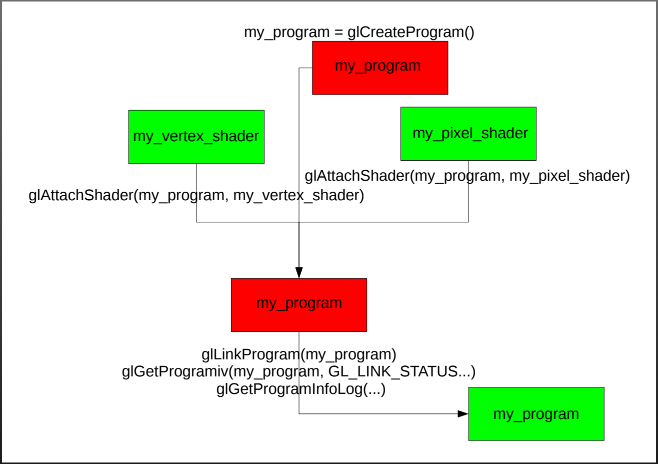
Shaders - data

- Shared among all instances (uniform)
  - variables *uniform*
  - UBO
  - SSBO
  - Textures (sampler)
  - Images
- specific for one instance
  - VBO
- Outputs
  - FBO
- Communication between shaders
  - in/out
  - shared

Shaders: compiler


my_shader=glCreateShader(GL_FRAGMENT_SHADER)
my_shader
glShaderSource(my_shader, ..., src,...)
my_shader+src
glCompileShader(my_shader)
glGetShaderiv(my_shader, GL_COMPILE_STATUS, …)
glGetShaderInfoLog
my_shader compilé

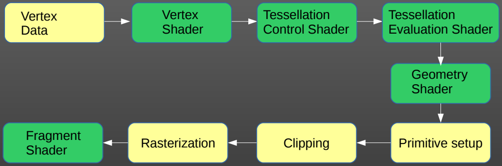
Shaders: Linker

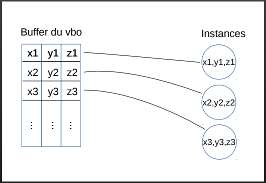

my_program = glCreateProgram()
my_program
my_vertex_shader
my_pixel_shader
glAttachShader(my_program, my_pixel_shader)
glAttachShader(my_program, my_vertex_shader)
my_program
glLinkProgram(my_program)
glGetProgramiv(my_program, GL_LINK_STATUS...)
glGetProgramInfoLog(...)
my_program

After the link it is possible to invoke `glDetachShader()` and `glDeleteShader()`

Activation: `glUseProgram(...)`

Vertex Shader





Vertex Shader

- Vertex shader inputs:
  - gl_Vertex: vertex position
  - gl_Color: vertex color
  - gl_Normal: vertex normal
  - gl_MultiTexCoordn: texture coordinate n
  - gl_SecondaryColor
  - gl_FogCoord

Vertex Shader

- Vertex shader inputs:
  - gl_Vertex: vertex position
  - gl_Color: vertex color
  - gl_Normal: vertex normal
  - gl_MultiTexCoordn: texture coordinate n
  - gl_SecondaryColor
  - gl_FogCoord

Deprecated !

Vertex Shader

- Vertex shader inputs:
  - gl_Vertex: vertex position
  - gl_Color: vertex color
  - gl_Normal: vertex normal
  - gl_MultiTexCoordn: texture coordinate n
  - gl_SecondaryColor
  - gl_FogCoord

Deprecated !

Deprecated !

Vertex Shader

- Vertex shader output:
  - gl_Position: position (*the clip-space output position of the current vertex*) of the vertex
  - gl_PointSize
  - gl_ClipDistance[]

  - gl_FrontColor/gl_BackColor: color of the vertex
  - SecondaryColor...
  - gl_TexCoordn: n texture coordinate
  - gl_FogFragCoord
  - gl_ClipVertex

Vertex Shader

- Vertex shader output:
    - gl_Position: position (*the clip-space output position of the current vertex*) of the vertex
    - gl_PointSize
    - gl_ClipDistance[]

    - gl_FrontColor/gl_BackColor: color of the vertex
    - SecondaryColor...
    - gl_TexCoordn: n texture coordinate
    - gl_FogFragCoord
    - gl_ClipVertex

Deprecated!

Vertex Shader

in/out usage
example:

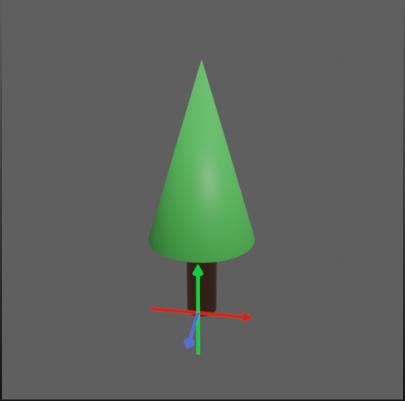
```
#version 450

layout(location = 1) in vec4 vPosition;
layout(location = 2) in vec4 vColor;

uniform mat4 model_view_matrix;
uniform mat4 projection_matrix;

out vec4 color;

void main() {
  gl_Position = projection_matrix * model_view_matrix * vPosition;
  color = vColor;
}
```

Vertex Shader - Coordinate systems

- The object in its own coordinate system
- The *model* matrix sets the object in the scene coordinate space
- The *view* matrix sets the complete scene in the camera coordinate space
- The *projection* matrix sets the complete scene in the *clip space*
- Then the coordinates are divided by w to sets them into the *normalized device coordinates and project them onto the screen*

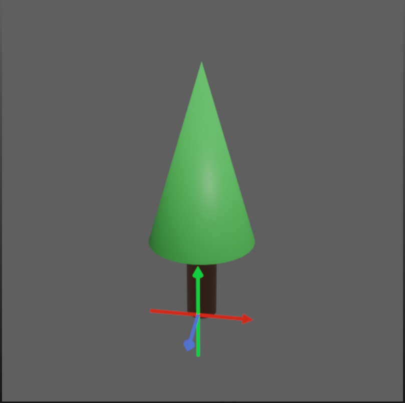
Vertex Shader - Coordinate systems

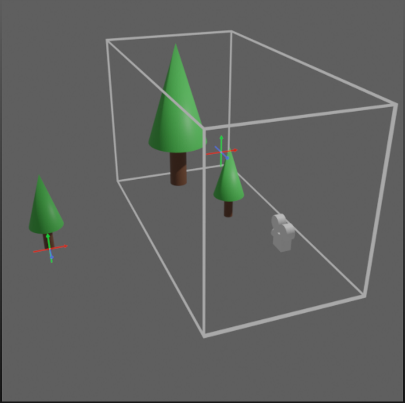
The object in its own coordinate system



Vertex Shader - Coordinate systems

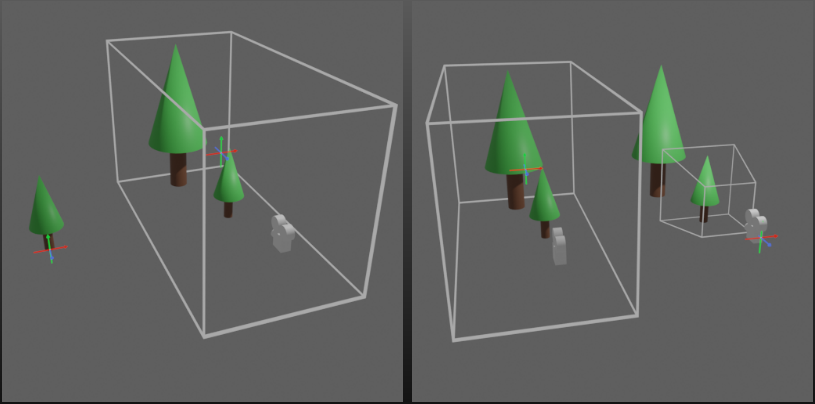
The *model* matrix sets the object in the scene coordinate space





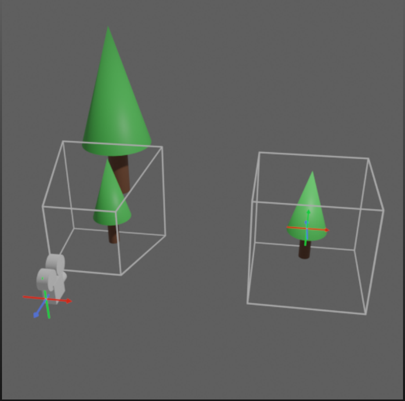
Vertex Shader - Coordinate systems

The *view* matrix sets the complete scene in the camera coordinate space



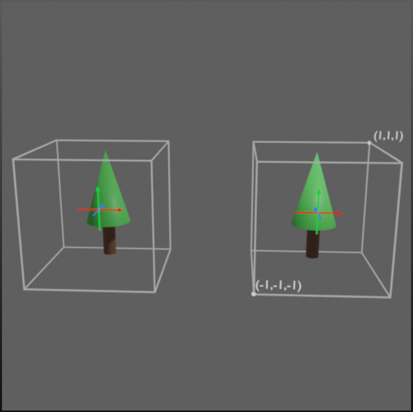
Vertex Shader - Coordinate systems

The *projection* matrix sets the complete scene in the *clip space*



Vertex Shader - Coordinate systems

Then the coordinates are divided by w to sets them into the *normalized device coordinates and projected onto the screen*

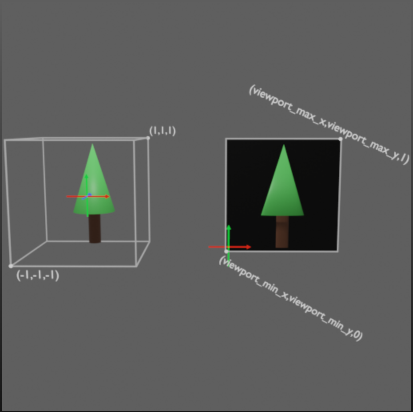


$$v_{clip} = \begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} \quad (1)$$

$$v_{NDC} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \\ w_{clip}/w_{clip} \end{pmatrix} = \begin{pmatrix} x_{NDC} \\ y_{NDC} \\ z_{NDC} \\ 1 \end{pmatrix} \quad (2)$$

Vertex Shader - Coordinate systems

and projected onto the screen


(1,1,1)
(-1,-1,-1)
(viewport_max_x,viewport_max_y,1)
(viewport_min_x,viewport_min_y,0)

$$v_{screen} = \begin{pmatrix} \frac{width}{2}(x_{NDC}+1)+off_x \\ \frac{height}{2}(y_{NDC}+1)+off_y \\ \frac{1}{2}z_{NDC}+\frac{1}{2} \end{pmatrix} \quad (3)$$

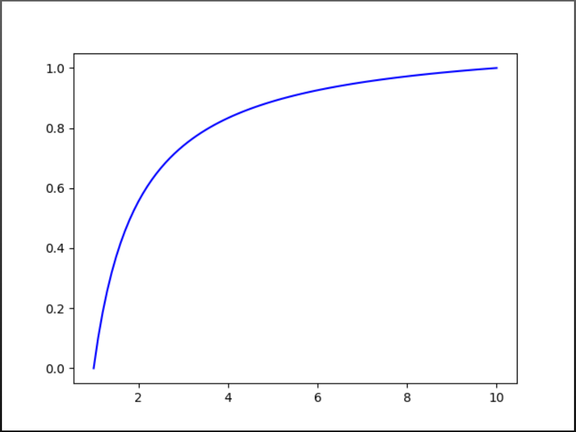
Vertex Shader - la précision du z-buffer

```
def f(i,A):
    j=-(i)
    B= np.array([[0.0],[0.0],[j],[1.0]])
    #z_clip=A.dot(B)[2]
    z_NDC=(A.dot(B)/A.dot(B)[3])[2]
    z_screen= z_NDC/2.0+1/2.0;
    return z_screen
```

Avec une matrice de projection orthogonale, z_{screen} est linéaire.
Avec une matrive de projection perspective, z_{screen} n'est pas linéaire.

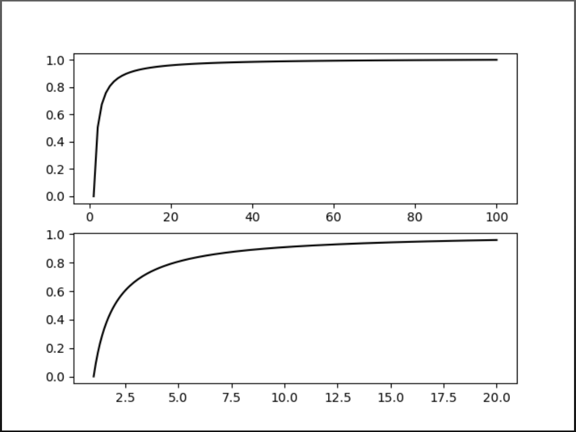
Vertex Shader - la précision du z-buffer

near plane = 1, far plane = 10, $z_{screen} = f(z, A_{perspective})$



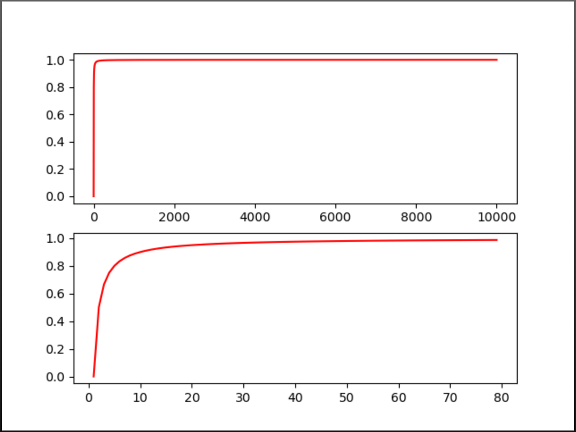
Vertex Shader - la précision du z-buffer

near plane = 1, far plane = 100, $z_{screen} = f(z, A_{perspective})$

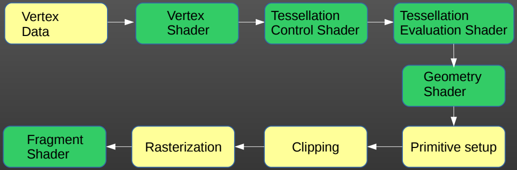


Vertex Shader - la précision du z-buffer

near plane = 1, far plane = 10 000, $z_{screen} = f(z, A_{perspective})$



Fragment shader



(The fragment shader is Optionnal)

Fragment shader

- Fragment shader input:
  - gl_FragCoord: location (in screen) of the fragment (pixel)
  - gl_FrontFacing
  - gl_PointCoord

  - gl_Color : the color of the vertex
  - Secondary color
  - gl_TexCoordn
  - gl_FogFragCoord

Fragment shader

- Fragment shader input:
  - gl_FragCoord: location (in screen) of the fragment (pixel)
  - gl_FrontFacing
  - gl_PointCoord

  - gl_Color : the color of the vertex
  - Secondary color
  - gl_TexCoordn
  - gl_FogFragCoord

Deprecated !

Fragment shader

- Pixel shader output:
  - gl_FragDepth: depth
  - gl_FragColor: fragment's color
  - gl_FragDatan

Fragment shader

- Pixel shader output:
  - gl_FragDepth: depth
  - gl_FragColor: fragment's color
  - gl_FragDatan

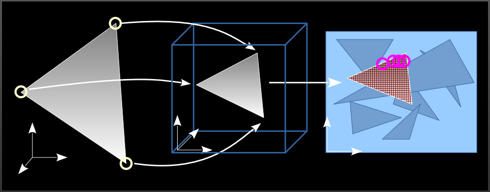
Deprecated !

Fragment shader

in/out usage
Example:

```
#version 450

in vec4 color;
out vec4 output_color;

void main() {
   output_color = color;
}
```

Value of color interpolated!



Shaders : exemples

Compute the lighting in each vertex

Vertex shader

```
#version 450

layout(location = 1) in vec3 vPosition;
layout(location = 2) in vec3 vNormal;

uniform mat4 model_view_matrix;
uniform mat4 projection_matrix;
uniform vec3 object_color;
uniform vec3 light_position;

out vec3 color;
vec3 normal;
vec3 light_dir;

void main() {
  gl_Position =
    projection_matrix
    * model_view_matrix
    * vec4 (vPosition, 1.0);
  normal = normalize(vNormal);
  light_dir = normalize(light_position-vPosition);
  color = clamp((dot(normal,light_dir))*object_color, 0, 1);
}
```

Fragment shader

```
#version 450

in vec3 color;
out vec4 output_color;

void main()
{
  output_color = vec4(color, 1.0);
}
```

Shaders : exemples

Compute the lighting in each fragment

Vertex shader

```
#version 450

layout(location = 1) in vec3 vPosition;
layout(location = 2) in vec3 vNormal;

uniform mat4 model_view_matrix;
uniform mat4 projection_matrix;
uniform vec3 object_color;
uniform vec3 light_position;

out vec3 color;
out vec3 normal;
out vec4 light_dir;

void main() {
  gl_Position =
            projection_matrix
            * model_view_matrix
            * vec4 (vPosition, 1.0);
  light_dir = normalize(light_position-vPosition);
  normal = normalize(vNormal);
  color = object_color;
}
```

Fragment shader

```
#version 450
in vec3 color;
out vec4 output_color;
in vec3 normal;
in vec3 light_dir;

void main() {
  output_color = vec4(
     clamp(
       (dot(normal, light_dir))*color
       ,0.0, 1.0
     )
   ,1.0);
}
```

Shaders: examples

Criticism:

- The use of vec3
- normalize/ glVertexAttribPointer (normal = normalize(vNormal);)
- multiplication projection_matrix * model_view_matrix
- clamp/max

OpenGL

OpenGL: Types and conventions

b integer 8 bits - signed char - GLbyte
s integer 16 bits - short - GLshort
i integer 32 bits - long, int - GLint, GLsizei
f real 32 bits - float - GLfloat, GLclampf
d real 64 bits - double - GLdouble, Glclampd
ub integer unsigned 8 bits - unsigned char - GLubyte, GLboolean
us integer unsigned 16 bits - unsigned short - GLushort
ui integer unsigned 32 bits - unsigned long - GLuint, Glenum, Glbitfield
v vector
Types linked: int (glsl) GLint (OpenGL), float (glsl) et GLfloat (OpenGL)...

OpenGL: Error detection

GLenum glGetError(void);
Returns the current value of the error stat of OpenGL environment
possible values:

- GL_NO_ERROR,
- GL_INVALID_ENUM,
- GL_INVALID_VALUE,
- GL_INVALID_OPERATION,
- GL_STACK_OVERFLOW,
- GL_STACK_UNDERFLOW,
- GL_OUT_OF_MEMORY.

glGetError reset the stat to GL_NO_ERROR

OpenGL : Interaction

Interaction with OpenGL

- glEnable(GLenum cap)
- glDisable(GLenum cap)

In practice

OpenGL is low level and dedicated to image computation.

- no window management
- no user interaction / IHM
- ...

Useful tools:

- glut/SDL/Qt/...
- glew
- ...

Glut (freeGlut)

Window management and OpenGL initialization

```
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize (800,600);
  glutFullScreen();
glutInitContextVersion (4, 5);
glutCreateWindow (char*...);
```

Glut (freeGlut)

Events/interactions

```
glutDisplayFunc(display); (-> glutPostRedisplay();)
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutKeyboardUpFunc(keyboardup);
glutSpecialFunc(special);
glutSpecialUpFunc(specialup);
glutMouseFunc(mouse);
glutMotionFunc(motion);
glutPassiveMotionFunc(motion);
glutIdleFunc(idle_func);
glutTimerFunc(timer);
```

Glut (freeGlut)

Register a callback on paint event

```
glutDisplayFunc(display);
```

Loop events

```
glutMainLoop();
```

Fires paint event

```
glutPostRedisplay();
```

Classical mistake:

```
void display(void){
   ...
   glutPostRedisplay();
}
```

Glut (freeGlut)

Register a callback on paint event

```
glutDisplayFunc(display);
```

Loop events

```
glutMainLoop();
```

Fires paint event

```
glutPostRedisplay();
```

Classical mistake:

Nooooo !

```
void display(void){
  ...
  glutPostRedisplay();
}
```

GLEW

```
glewExperimental = GL_TRUE;
GLenum err = glewInit();
if (GLEW_OK != err) {
  std::cerr << "GlewInit␣fails:␣" << glewGetErrorString(err) << std::endl;
  return false;
}
return true;
```

Example

Example - Initialization

```
void init_gl(void) {
  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_LESS);
  glDepthRange(0.0, 1.0);
  glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
  glEnable(GL_CULL_FACE);
  glCullFace(GL_FRONT);
  glFrontFace(GL_CCW);
  glClearColor(0.0,0,0,0);
  ...
}

init_glut(argc, argv);
init_glew();
init_gl();
```

Example - Declaration

```
glGenVertexArrays(1, &vao_id);
glBindVertexArray(vao_id);

glGenBuffers(1, &vbo_position_normal_earth_id);
glBindBuffer(GL_ARRAY_BUFFER, vbo_position_normal_id);
glBufferData(GL_ARRAY_BUFFER, nb_vertex*sizeof(GLfloat), vertex_list, GL_STATIC_
glVertexAttribPointer(vertex_position_location, 3, GL_FLOAT, GL_FALSE, 6*sizeof(
glEnableVertexAttribArray(vertex_position_location);

glVertexAttribPointer(vertex_normal_location, 3, GL_FLOAT, GL_FALSE, 6*sizeof(G
glEnableVertexAttribArray(vertex_normal_location);

glGenBuffers(1, &vbo_texture_coord_earth_id);
glBindBuffer(GL_ARRAY_BUFFER, vbo_texture_coord_id);
glBufferData(GL_ARRAY_BUFFER, nb_vertex/3*2*sizeof(GLfloat), uv_list, GL_STATIC_
glVertexAttribPointer(uv_location, 2, GL_FLOAT, GL_FALSE, 2*sizeof(GLfloat), 0);
glEnableVertexAttribArray(uv_location);

glBindVertexArray(0);
```

Example - Texture

```
texture_bitmap = load_image("texture_monde.tga");
GLint texture_location;

glGenTextures(1, &earth_texture);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, earth_texture);

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, texture_bitmap->iw, texture_bitmap->ih, 0

glTexParameteri(...);

earth_texture_location = glGetUniformLocation(program->program_id, "tex_sampler'
glUniform1i(earth_texture_location, 0);
```

Example - Vertex Shader (code must be improved!)

```
#version 420

in vec3 vertex_postion;
in vec3 vertex_normal;
in vec2 vertex_uv;

uniform mat4 model_view;
uniform mat4 projection;
uniform vec3 light_position;

out vec4 interpolated_color;
out vec2 interpolated_uv_position;

void main ()
{
  vec4 light_dir = vec4(light_position-vertex_position.xyz, 1);
  float coef = dot(normalize(vertex_normal.xyz),  normalize(light_dir.xyz));
  coef = clamp(coef, 0, 1);
  interpolated_color =  vec4(vec3(1, 1, 1)*coef, 1.0);
  gl_Position =  projection * model_view * vec4 (vertex_position, 1.0);
  interpolated_uv_position = vertex_uv;
}
```

Example - Fragment Shader (code must be improved!)

```
#version 420

uniform sampler2D tex_sampler;

in vec4 interpolated_color;
in vec2 interpolated_uv_position;

out vec4 output_color;

void main()
{
    vec4 texel = texture2D(tex_sampler, interpolated_uv_position);
    output_color =  interpolated_color * vec4(texel.rgb, 1.0);;
}
```

Example - Start drawing

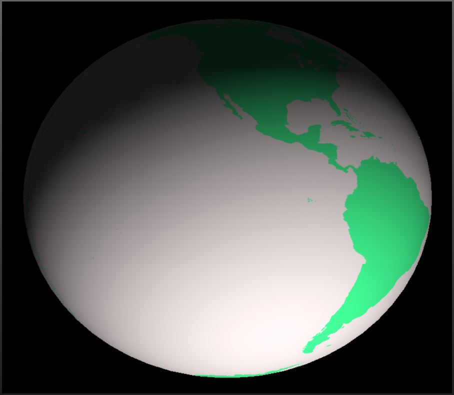
```
glBindVertexArray(vao_id);
glDrawArrays(GL_TRIANGLES, 0, vertex*3);
glBindVertexArray(0);
```

Example - Variation

```
    ...
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, &vbo_position_shape_id);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,
                    my_shape.nb_faces*3*sizeof(GLuint),
                    my_shape.triangles_list,
                    GL_STATIC_DRAW);
...

...
glDrawElements(GL_TRIANGLES, my_shape.faces*3, GL_UNSIGNED_INT, 0);
...
```

- indexed access (different from obj format)

Result



To be continue...

Conclusion

- Low level
  - Vulcan < OpenGL < Unity
- Portable (+WebGL, OpenGL ES...)