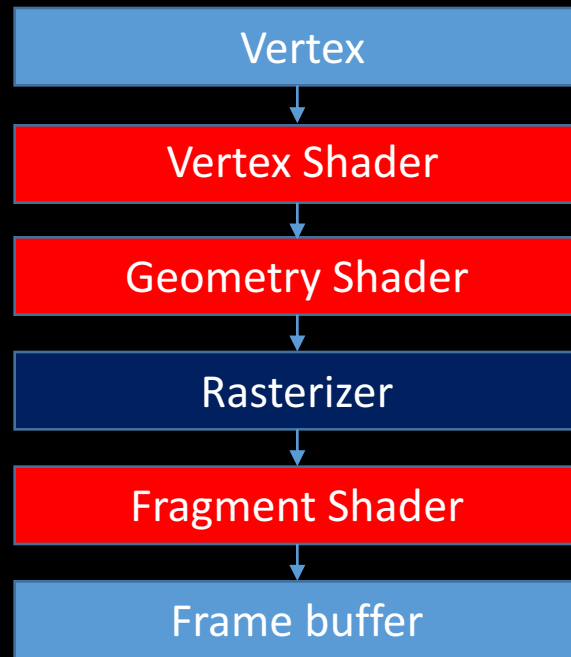


OpenGL shader & GLSL

2020 Introduction to Computer Graphics

OpenGL pipeline



Shader (我們可以自己寫, 存在 GPU 的東西)

- A program designed by users.
- Run in GPU pipeline.

Vertex Shader

- **Input:** Single vertex
- **Output:** Single vertex

Geometry Shader

- **Input:** One primitive
- **Output:** Can be more than one primitive

Fragment Shader

- **Input:** One pixel
- **Output:** One or no pixel

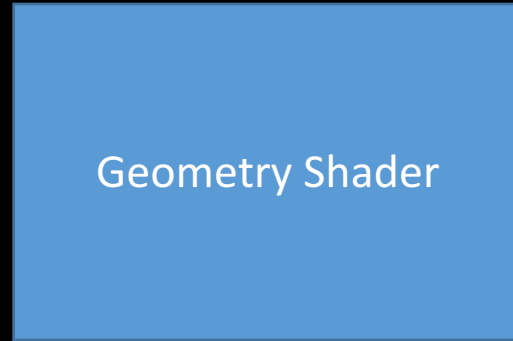
Shader

頂點、座標



- **Input:** Single vertex
- **Output:** Single vertex

顏色、材質



- **Input:** One primitive
- **Output:** Can be more than one primitive



- **Input:** One pixel
- **Output:** One or no pixel

Shader setting

→ 跟 OpenGL 语言要开一个 shader

- In the function : `createShader()` (defined in `shader.h`)
 - GLuint `glCreateShader` (GLenum `shaderType`); → *shader type*
 - Specifies the type of shader to be created and creates an empty shader object.
 - `shaderType` : `GL_COMPUTE_SHADER`, `GL_VERTEX_SHADER`, `GL_TESS_CONTROL_SHADER`, `GL_TESS_EVALUATION_SHADER`, `GL_GEOMETRY_SHADER`, `GL_FRAGMENT_SHADER`
 - void `glShaderSource` (GLuint `shader`, GLsizei `count`, const GLchar **`string`, const GLint *`length`);
 - Sets the source code in `shader` to the source code in the array of strings specified by `string`.
 - Ex : `string` = & `textFileRead("Shaders/example.vert")` → *是读进 写的 shader 的档案*
 - void `glCompileShader` (GLuint `shader`);
 - Compile the `shader`.

作業都做好了

```
void shaderInit() {  
    GLuint vert = createShader("Shaders/vertexShader.vert", "vertex");  
    GLuint frag = createShader("Shaders/fragmentShader.frag", "fragment");  
    program = createProgram(vert, frag);  
}
```

Shader setting

- In the function : `createProgram()` (defined in `shader.h`)
 - `GLuint glCreateProgram(void);`
 - creates a program object.
 - `void glAttachShader (GLuint program, GLuint shader);`
 - Attach the `shader` object to the `program` object.
 - `void glLinkProgram (GLuint program);`
 - Link this program
 - `void glDetachShader (GLuint program, GLuint shader);`
 - Detaches the `shader` object from the `program` object.

```
void shaderInit() {  
    GLuint vert = createShader("Shaders/vertexShader.vert", "vertex");  
    GLuint frag = createShader("Shaders/fragmentShader.frag", "fragment");  
    program = createProgram(vert, frag);  
}
```

OpenGL shader,
bind shader

Use program

```
void display() {  
    glUseProgram(program_id); → 现在使用那个program  
    /* Shader program effect in this block */  
    /* Pass parameters to shaders */  
    glUseProgram(0); → 现在program release掉  
    /* Pass 0 to stop the program */  
    glUseProgram(another_program_id);  
    /* Another shader program effect */  
    glUseProgram(0);  
}
```

Vertex Buffer Objects (VBO)

畫圖時有很多頂點標,
會存進 VBO (一個個 box)

- Since the vertex shader access only one vertex at one time, we use **Vertex Buffer Objects** to make the execution be faster. The advantage of using these buffered objects is that we can send a large amount of vertex data from system memory to GPU memory at one time instead of sending it once per vertex.

→ 生成一個 buffer 物件

- Step 1 : Use **glGenBuffers()** to generate vertex buffer objects

```
void glGenBuffers ( GLsizei n, GLuint * buffers );
```

n : Specifies the number of buffer object names to be generated.

buffers : Specifies an array in which the generated buffer object names are stored.

- Step 2 : Use **glBindBuffer()** to bind the target buffer, which is GL_ARRAY_BUFFER here.

← void glBindBuffer (GLenum target, GLuint buffer);

target : GL_ARRAY_BUFFER 、 GL_TEXTURE_BUFFER 、

buffer : Specifies the name of a buffer object.

bind
buffer 到
OpenGL
中 的 緩衝區

buffer 的
type

buffer pointer

```
GLuint vboName;  
glGenBuffers(1, &vboName);  
glBindBuffer(GL_ARRAY_BUFFER, vboName);
```


Vertex Buffer Objects (VBO)

- Step 3 : Set up the data
- Step 4 : Use `glBufferData()` to copy the `data` into the `target`.

`void glBufferData (GLenum target, GLsizeiptr size, const GLvoid * data, GLenum usage);`

`target` : GL_ARRAY_BUFFER、GL_TEXTURE_BUFFER、.....

`size` : Specifies the size in bytes of the buffer object's new data store.

`data` : Specifies a pointer to data that will be copied into the data store for initialization, or NULL if no data is to be copied.

`usage` : Specifies the expected usage pattern of the data store. Ex: GL_STATIC_DRAW means the data store contents will be modified once and used at most a few times.

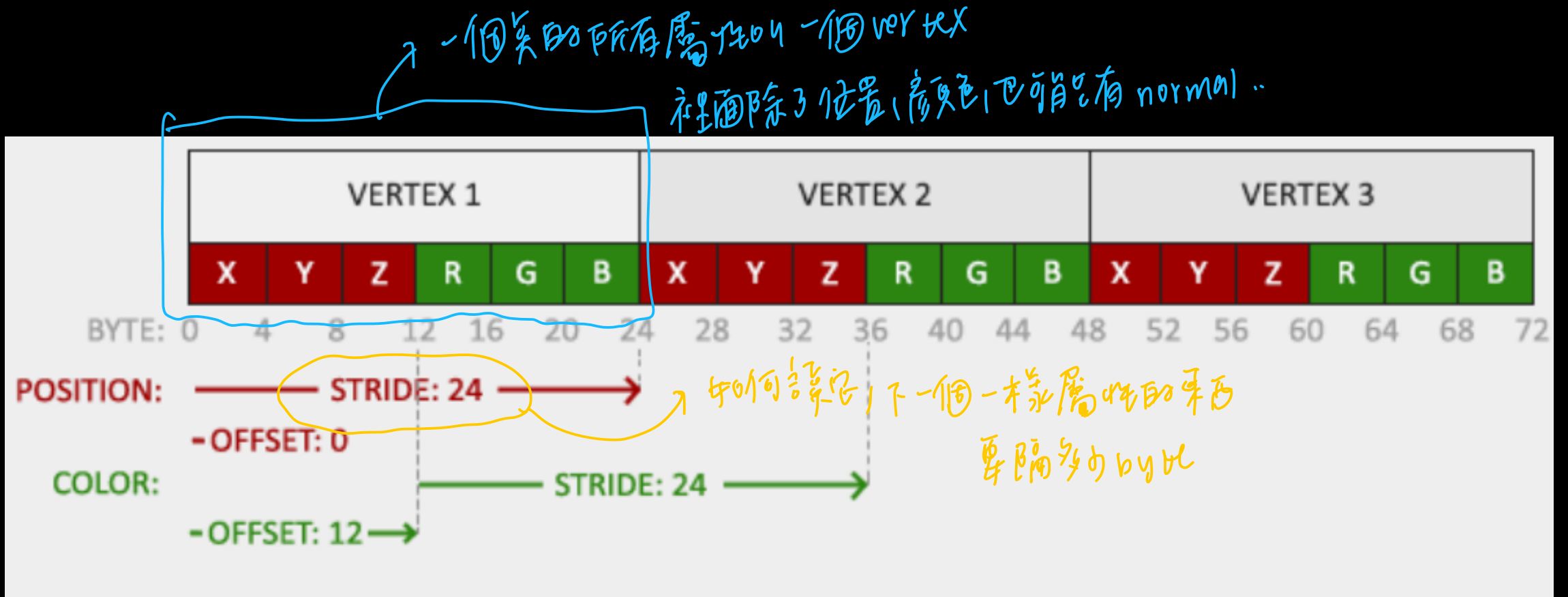
```
VertexAttribute *vertices;  
vertices = drawTriangle();  
glBufferData(GL_ARRAY_BUFFER, sizeof(VertexAttribute) * verticeNumber, vertices, GL_STATIC_DRAW);
```

要傳遞的 buffer data 的



為何畫它

Vertex Buffer Objects (VBO)



Implementation in OpenGL

```
struct VertexAttribute{ GLfloat position[3]; };
```

```
VertexAttribute *vertices;
```

```
GLuint vboName;
```

```
glGenBuffers(1, &vboName); //generate 1 buffer
```

```
glBindBuffer(GL_ARRAY_BUFFER, vboName);
```

```
glBufferData(GL_ARRAY_BUFFER, sizeof(VertexAttribute) * vertices_length,  
vertices, GL_STATIC_DRAW);
```

現在只有一個vertex，是1

Vertex Attribute Pointer

把弄好的 data
弄到 GLSL (GPU) 中

- We can use **glVertexAttribPointer()** to link the vertex buffer with the vertex shader input.

```
void glVertexAttribPointer ( GLuint index, GLint size, GLenum type, GLboolean normalized,  
GLsizei stride, const GLvoid * pointer);
```

index : Specifies the index of the generic vertex attribute to be modified.

size : Specifies the number of components per generic vertex attribute.

type : Specifies the data type of each component in the array. Ex: GL_FLOAT

normalized : Specifies whether fixed-point data values should be normalized or not.

stride : Specifies the byte offset between consecutive generic vertex attributes.

pointer : Specifies a offset of the first component of the first generic vertex attribute in the array in the data store of the buffer currently bound to the GL_ARRAY_BUFFER target. The initial value is 0.

Vertex Attribute Pointer

```
glEnableVertexAttribArray(0);
```

→ 開個通道

```
glVertexAttribPointer(0,
```

用哪個通道

3, 一次存取幾個東西

```
GL_FLOAT,
```

```
GL_FALSE,
```

→ 要不要做 normalize (一般沒有都寫 false, 要的話再自己做大 normalize)

```
sizeof(VertexAttribute), // stride
```

→ offset

```
(void*)(offsetof(VertexAttribute, position)));
```

OpenGL

```
layout(location = 0) in vec3 in_position;
```

“我每次存取 3 個 data”

GLSL (vertex shader)

跟 GLSL 定義 input data

Unbind the VBO

- Use `glBindBuffer()` with the buffer set to zero to unbind the target buffer.

```
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

用完 unbind 操作

物体多, 渲染复杂程序, VBO可简化流程

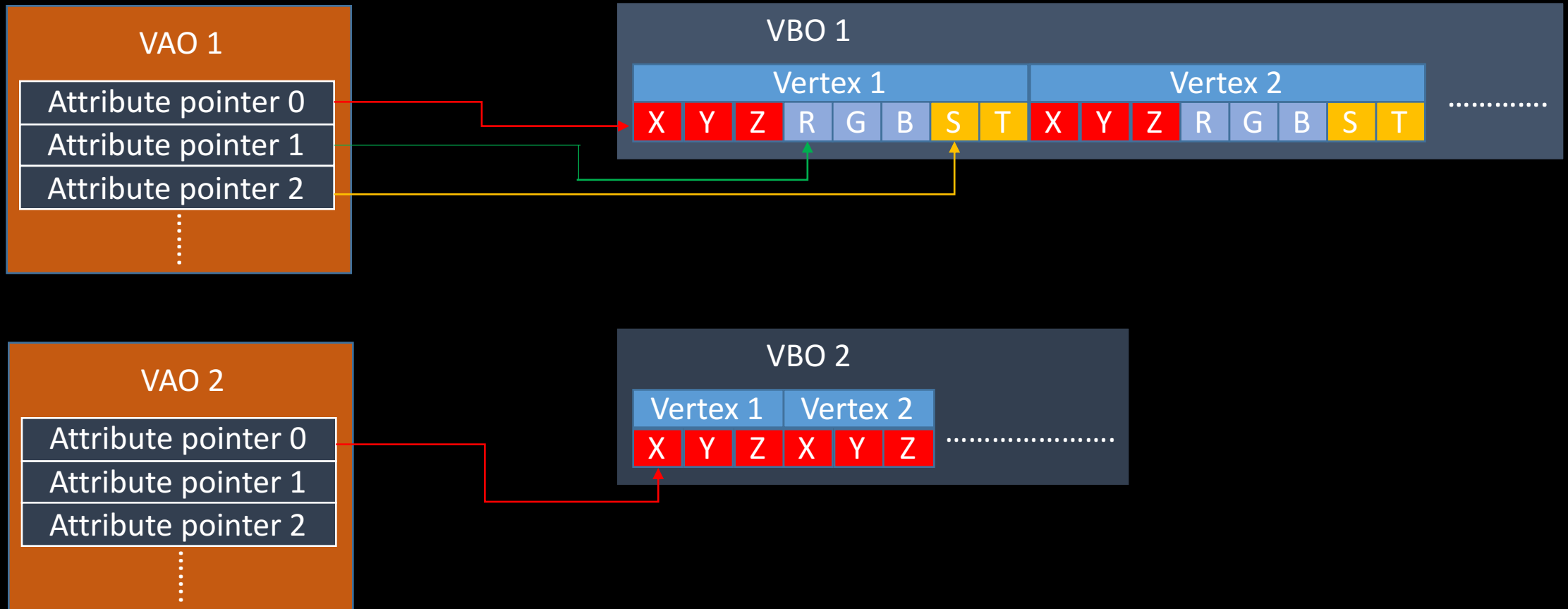
Vertex Array Object (VAO)

用VBO再把VBO串起来, 比较有序整理

(VAO中有多个VBO)

- If you want to render more than one objects, you have to repeat above steps (slides 8 ~14).
→very troublesome
- Use VAO(Vertex Array Object) to handle this problem.
- First, you have to set up all the VAOs with its corresponding VBO, including all VertexAttribPointer. After that, every time you want to render a certain object, you just need to bind its VAO.

Vertex Array Object (VAO)



Vertex Array Object (VAO)

- Step 1 : Use **glGenVertexArrays()** to generate vertex array objects

void **glGenVertexArrays** (GLsizei n, GLuint * arrays);

n : Specifies the number of vertex array object names to be generated.

arrays : Specifies an array in which the generated vertex array object names are stored.

- Step 2 : Use **glBindVertexArray()** to bind a vertex array object.

void **glBindVertexArray** (GLuint array)

array : Specifies the name of the vertex array to bind.

先生成VAO後，
後面生成的VBO都會
預設綁定之下

create一個VAO
然後把VAO
bind起來

```
GLuint VAO;  
glGenVertexArrays(1, &VAO);  
glBindVertexArray(VAO);
```

Vertex Array Object (VAO)

- Step 3 : Setting up its corresponding VBO, for example :
 - `glBindBuffer(GL_ARRAY_BUFFER, VBO);`
 - `glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);`
 - `glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);`
 - `glEnableVertexAttribArray(0);`
- Step 4 : Use `glBindVertexArray(0)` with the array's name set to zero to unbind the array object.

`void glBindVertexArray (GLuint array)`

Ex: `glBindVertexArray(0)` means to unbind the VAO previously bound.

When Rendering

資料處理好後 如何拿東西來畫

→ 要用哪個 VAO 來畫

- Step 1 : Use **glBindVertexArray(VAO)** to bind the VAO you want.
- Step 2 : Use **glDrawArrays()** to render primitives from vertex array data.

void **glDrawArrays()** (GLenum mode, GLint first, GLsizei count);

mode : Specifies what kind of primitives to render. Ex: GL_POINTS, GL_LINES, GL_TRIANGLE_STRIP.....

first : Specifies the starting index in the enabled arrays.

count : Specifies the number of indices to be rendered.

- Step 3 : Remember to unbind the VAO. (**glBindVertexArray(0)**)

*Every time you want to render another object, you just need to bind another VAO.

直接畫

Data Connection - Uniform

寻找在 program object 中
uniform variable "Projection" 的位置

GLSL 中的一个 type

不会用到的东西、不会在 2 种 Shader 中

共用的东西会写成 uniform

4x4 矩阵 → 较小的 array 直接把这个 matrix
传递

[否则的话像前几页 ppt 一样就写成 layout)

```
GLfloat pmtx[16]; //getP(), getV()
```

```
glGetFloatv(GL_PROJECTION_MATRIX, pmtx);
```

```
GLint pmatLoc = glGetUniformLocation(program, "Projection");
```

location

```
glUseProgram(program);
```

```
glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, pmtx);
```

```
glUseProgram(0);
```

uniform 的 type

在这个 program 会传入一个参数叫
Projection

OpenGL

```
uniform mat4 Projection;
```

GLSL (vertex shader)

在 shader 中直接用 uniform 接

修改 uniform 变量的值

是 4x4 矩阵

GLSL 用一行数字来描述

Data Connection - Texture

會 bind texObj 到現在 active 的 texture 上

```
glUseProgram(program);  
glActiveTexture(GL_TEXTURE0);  
glBindTexture(GL_TEXTURE_2D, texObj);  
GLint texLoc = glGetUniformLocation(program, "Texture");  
glUniform1i(texLoc, 0);  
/* draw objects */  
glBindTexture(GL_TEXTURE_2D, 0);  
glUseProgram(0);
```

用某個 program

要用哪個 texture, 用哪個 texture

貼圖 data

integer
glUniform1i(texLoc, 0);
/* draw objects */

binding 不需, 這個也不用寫

用這個 texture, GL_TEXTURE0 會初始化為 0
用這個和 前一個 texture 差不多

OpenGL

```
layout(binding = 0) uniform sampler2D Texture;  
in vec2 texcoord;  
out vec4 outColor;  
void main() { outColor = texture2D(Texture, texcoord); }
```

不用寫

texture type

vec2

GLSL (fragment shader)

return texcoord 座標上的 Texture 顏色

GLSL Syntax

- Basic Variable Types

- vec2, vec3, vec4, ... (2-d, 3-d, 4-d... array)
- mat2, mat3, mat4, ... (2x2, 3x3, 4x4...)
- float, int, bool, ...
- sampler2D, ... \rightarrow texture type

- Basic Functions

- max, min, sin, cos, pow, log, ...
- dot, normalize, reflect, ...
- transpose, inverse, ...

Vertex Shader

• must have `gl_Position`

vertex shader 一定要有這個

算完之後要用它接起

最終結果

從 vertex shader
寄到
fragment shader

是一個
4x4 矩陣

/* Example of vertex shader */

#version 430

版本

layout(location = 0) in vec3 position; → 有個 attribute 位置來

uniform mat4 Projection;

uniform mat4 ModelView;

out vec3 color; // to fragment shader

void main() {

`gl_Position` = Projection * ModelView * `vec4(position, 1.0);`

color = vec3(1.0, 0.0, 0.0);

}

要接起來

用這個把我的 position 轉成座標

直接給它值就會寄到 fragment shader

Fragment Shader

- **must have** a out vec4 **for** color buffer

```
/* Example of fragment shader */  
#version 430
```

```
in vec3 color; //from vertex shader
```

```
out vec4 frag_color;
```

```
void main() {
```

```
    frag_color = vec4(color, 1.0);
```

```
}
```

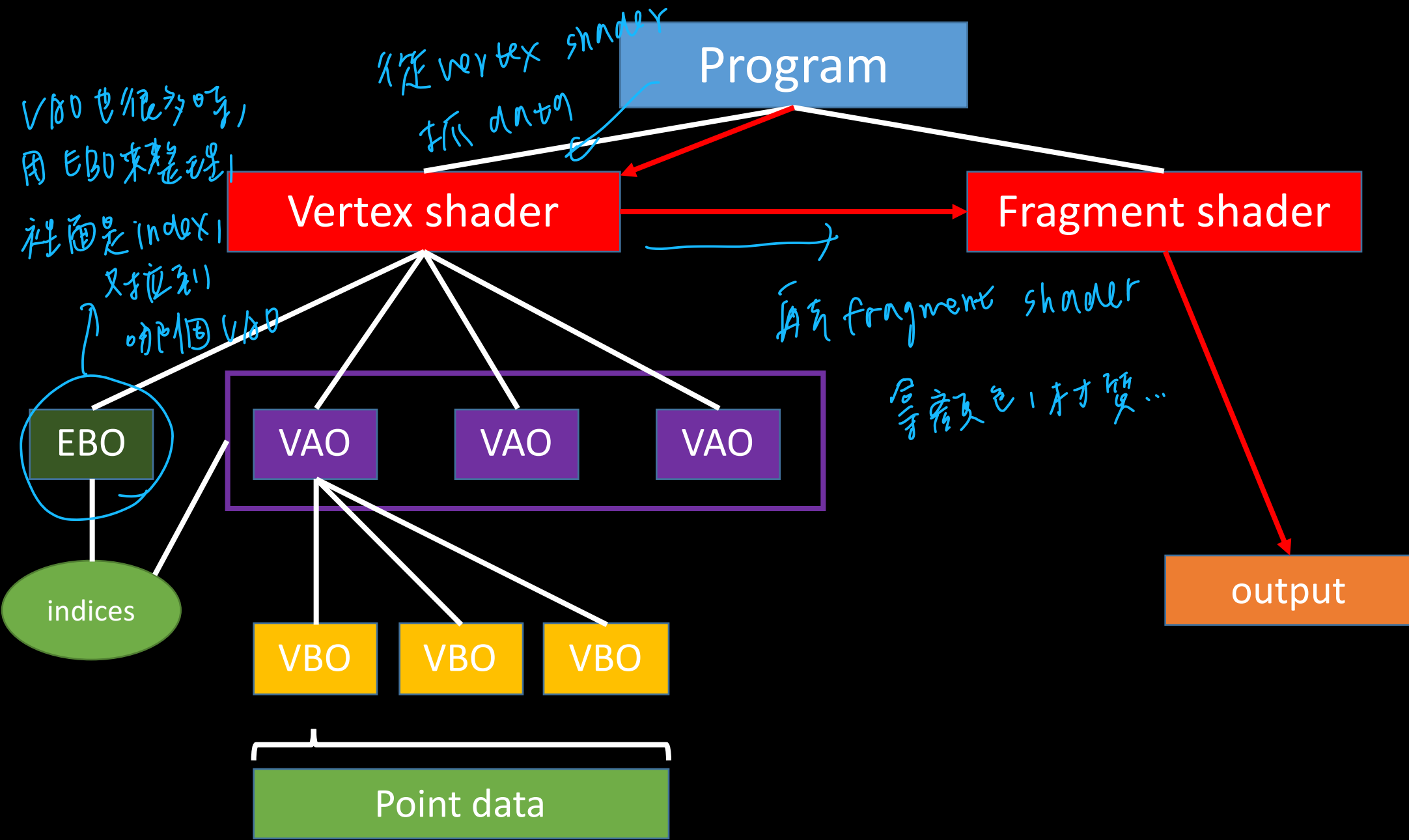
output

要是4維向量

是最後這個

想呈現的東西

第4維是透明度

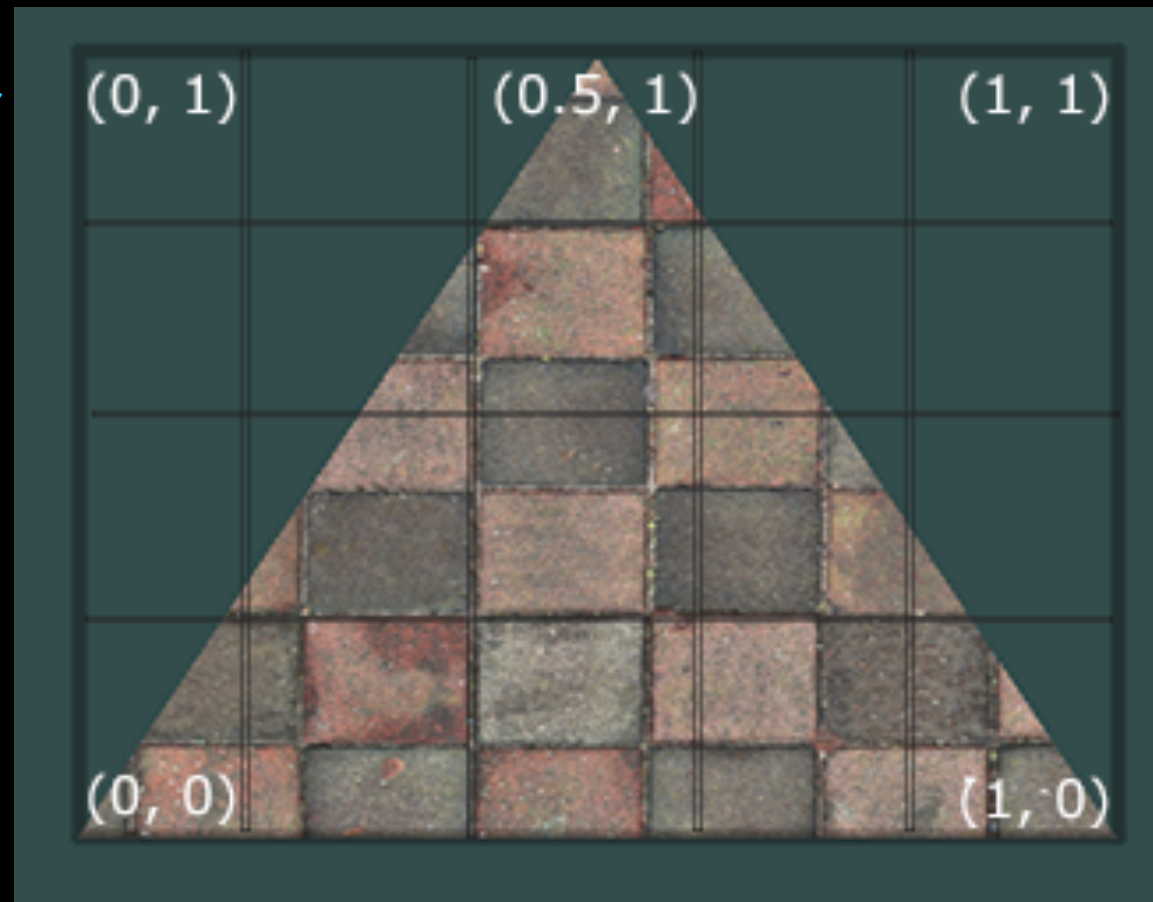


Texture in OpenGL

2014年1月14日 OpenGL 2.0 教程

Texture coordinate

texture
的采样



How to load and bind a texture

- void `glEnable`(GLenum cap);
Use `GL_TEXTURE_2D` to enable texture
- Use FreeImage library to **load** and **free** texture memory
- void `glGenTextures`(GLsizei n, GLuint * textures);
Takes as input how many textures we want to generate and stores them in a **unsigned int array**
- void `glBindTexture`(GLenum target, GLuint texture);
Bind a named texture to a texturing target
- void `glTexImage2D`(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid * data);
Generate a two-dimensional texture image

load 進來後
去生成 texture

enable 要用這個圖
生成 texture 空間

```
glEnable(GL_TEXTURE_2D);  
glGenTextures(1, &texture);  
glBindTexture(GL_TEXTURE_2D, texture);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
```

How to load and bind a texture

贴图比较小,但要贴的范围

很大,要如何贴图(重复贴)

(贴图大小与范围的比例)

- void `glTexParameteri`(GLenum target, GLenum pname, GLint param);
- Texture wrapping
 - Texture coordinates usually range from (0,0) to (1,1) but if we specify coordinates outside this range, the default behavior of OpenGL is to **repeat** the texture images
 - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);`
 - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);`
- Texture filtering
 - Texture coordinates do not depend on resolution but can be any floating point value, thus OpenGL has to figure out which texture pixel to map the texture coordinate to
 - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);` → 取最近的像素颜色
 - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);`

把load完的贴图 bind 起来

How to use

- Step 1 : Use **glActiveTexture(GL_TEXTURE0)** to activate the texture unit
Texture unit GL_TEXTURE0 is always by default activated
- Step 2 : Use **glBindTexture (GL_TEXTURE_2D, texture_name)** to bind the texture which you want to use.
- Step 3 : Use **glTexCoord2f(s, t)** to define the point's texture coordinate
- Step 4 : give the point (usually using glVertex3f or glVertex3d)

LoadTexture(basistexture, "basis.jpg");

```
glActiveTexture(GL_TEXTURE0);  
glBindTexture(GL_TEXTURE_2D, basistexture);  
glBegin(GL_POLYGON);  
glNormal3d(sin(i / edge * 2 * pi), 0, cos(i / edge * 2 * pi));  
glTexCoord2f(0.0f, 0.0f);  
glVertex3d(radius * sin(i / edge * 2 * pi), 0, radius * cos(i / edge * 2 * pi));  
glActiveTexture(0);
```

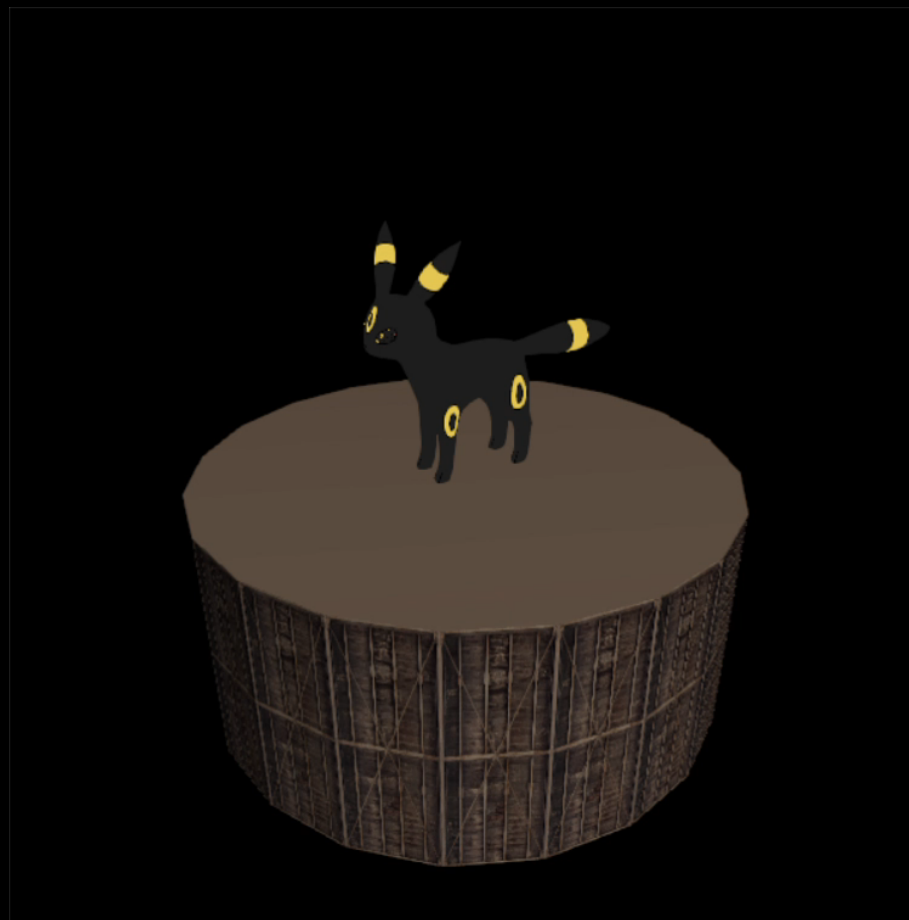
開始畫圖

加贴图座標

表示要畫在哪

Texture you load before

Homework 2 - Music Box



Homework 2

- Goal :

1. Using GLSL to draw

2. Using OpenGL to draw

用之前写的 code + 贴图 (贴图)

- Some parameters will be used (define in Object.h)

- Position : model->positions

- Normal : model->normals

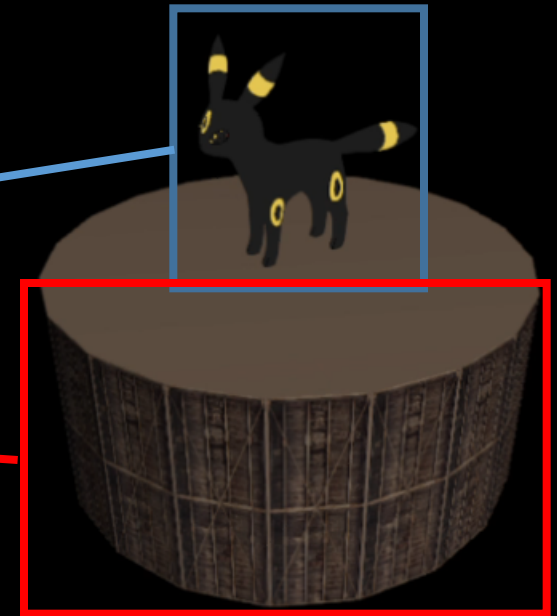
- Texcoords : model->texcoords

这个不用管 (用不到), 但要写

给 vertex shader, 再传给 fragment shader

有些要用的
参数定义在这

```
vector<float> positions;  
vector<float> normals;  
vector<float> texcoords;
```



Homework 2 (配分)

1. createShader, createProgram (5%)
2. Setup VAO, Setup VBO of vertex positions, normals, and texcoords (20%)
3. draw the basis and make the side of basis with texture (10%)
 - #There is no requirement for the top and bottom colors
 - #use OpenGL to draw , not need to use shader
4. pass projection matrix, and view matrix and trigger by Uniform (use getP() and getV()) (5%)
also pass modeltexture to shader and trigger by Uniform (20%)

texture, P、V 要用 uniform 传
5. vertex shader (10%)
 - #normal will not be used in this homework, but you have to receive from bind buffer in the vertex shader and pass to fragment shader
6. fragment shader (10%)
7. report (20%)
8. # Bonus (5%) Using shader to do anything you want is welcomed ☺

Homework 2 (report)

- Please specify your name and student ID in the report.
- Explain in detail how to use GLSL by taking screenshots.
(first create program ,second create VAO and VBO, third bind together.....etc.)
- Describe the problems you met and how you solved them.
- Explain what you do for the Bonus. (optional)

Homework 2 (繳交規則)

1. DeadLine: 2020/ 11 / 30 23: 59:59
2. Penalty of 10% of the value of the assignment per late week.

If you submit your homework late, the score will be discounted.

submit between (12/1 - 12/7) : Your final score * 0.9

submit between (12/8 - 12/14) : Your final score * 0.8

submit after 12/14 : Your final score * 0.7

Restrictions !!

作業只允許 layout | uniform | in | out

- Your GLSL version should \geq #version 330 版本要 ≥ 330 ，不然有些東西不兼容
- Deprecated shader syntaxes are not allowed, e.g. attribute, varying
- You are only allowed to use VBO and/or VAO when rendering model 不能用 for 環
- You are only allowed to pass uniform data to shader using glUniform* series function
- Using built-in uniform variables in shader is forbidden!
 - (That is, you **cannot** use gl_ModelViewMatrix or gl_NormalMatrix ...etc)
 - The only gl_XXX term should be in your shader code is gl_Position.

Upload Format

1. If your uploading format doesn't match our requirement, there will be penalty to your score. (-5%)
2. Please hand in the whole **project file** and **report** (.pdf) as STUDENTID_Name.zip to e3 platform.
e.g. 0716XXX_王小明.zip

#project file要載下來就可以demo

Reference

- <https://learnopengl.com/Advanced-OpenGL/>
- <https://learnopengl.com/Getting-started/Textures>
- [https://www.khronos.org/opengl/wiki/Built-in_Variable_\(GLSL\)](https://www.khronos.org/opengl/wiki/Built-in_Variable_(GLSL))

