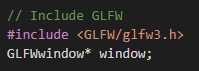
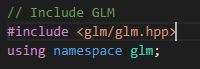
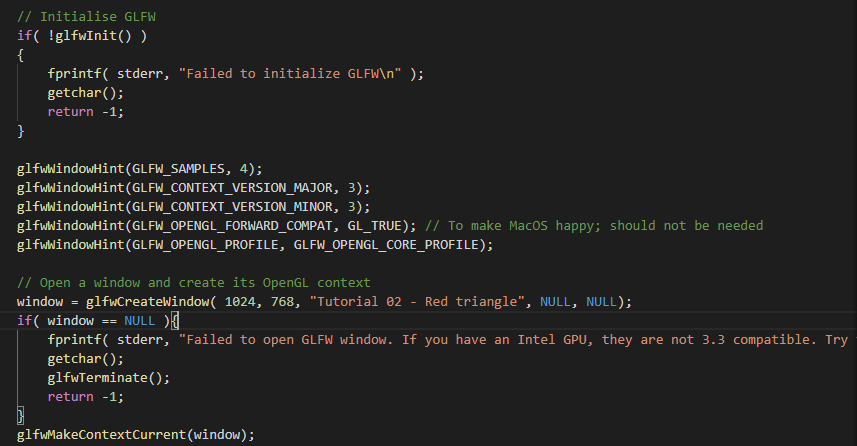
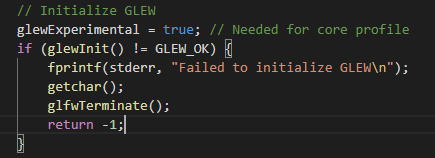
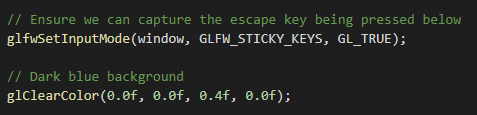
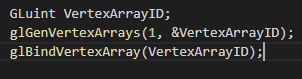
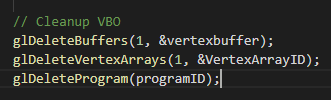
## Include một số thư viện cần thiết

* 
* 
* GLEW: cross-platform open-source dùng để load - link C/C++ lib (opengl extension nào phù hợp với platform nào)
* 
* GLFW: Open Source, multi-platform library cung cấp các high-level API của openGL (openGL ES) -> chủ yếu là dùng mở window, contexts, events, …
* 
* OpenGL Mathematics (GLM) is a header only C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specifications -> Những cái mat4, vec, ...

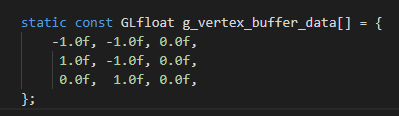
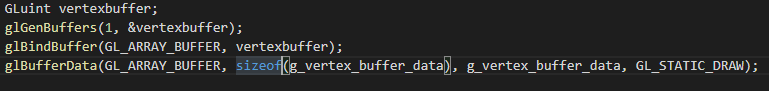
## Initialize

* Initialise GLFW
* 
* Initialize GLEW



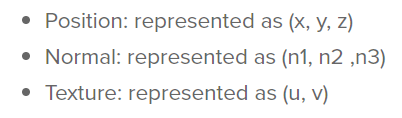
* Một số cái khác:
* 
* 
* VAO (Vertex Array Objects): là opengl object chỉ được tạo 1 lần trong toàn ctr và đóng vai trò là container cho VBOs
* 

## Nạp dữ liệu

* Khai báo:
* 
* 3 số liên tiếp ứng với tọa độ 1 đỉnh trong coordinate system
* Nạp vào GPU
* 
* Gluint: là 1 số uint nhưng kích thước cố định trên các platform khác nhau
* glGenbuffers: Tạo 1 vùng buffer và gán định danh (resulting identifier) vào vertexbuffer
* glBindBuffer: gán GL\_ARRAY\_BUFFER với vertexbuffer => để thay đổi vertexbuffer sẽ thao tác trên GL\_ARRAY\_BUFFER
* Cái nhìn tổng quan hơn: glGenbuffers tạo 1 buffer object trong GPU chứa unformatted data => The behavior of a buffer object depends on its binding points ( For example, when a buffer object's binding point is GL\_ARRAY\_BUFFER, it behaves as a Vertex Buffer Object - VBO)
* VBO là 1 loại object trong openGL. openGL object thường gắn với các thao tác:

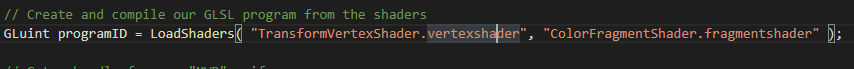


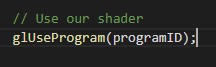
* VBO: Thông thường data được lưu trong VBO dưới dạng vector. VD, một đa giác sẽ gồm các loại data sau:

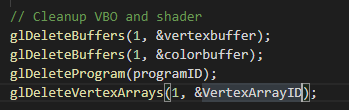


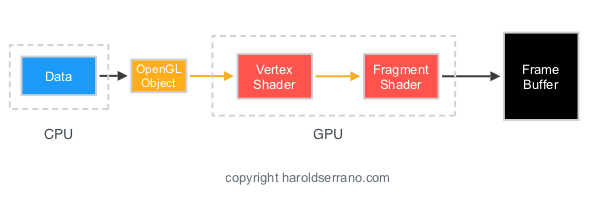
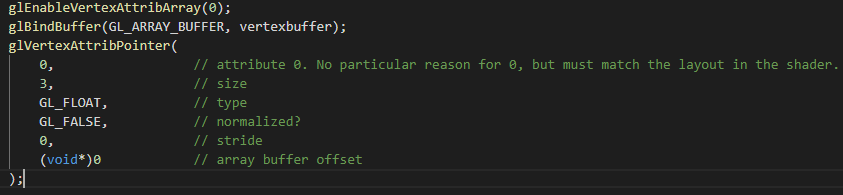
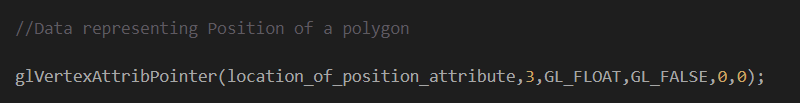
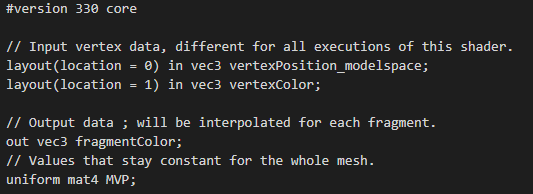
## Shader

* Load shader:

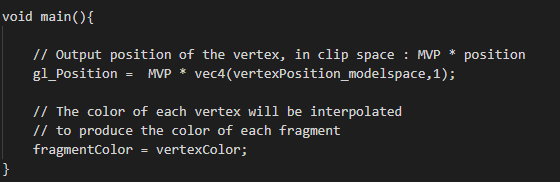


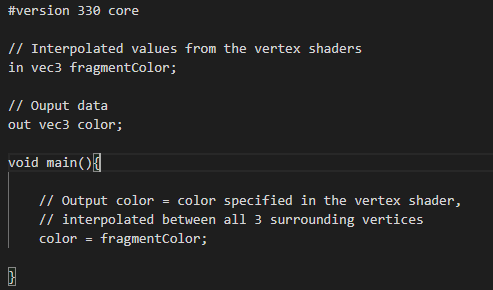




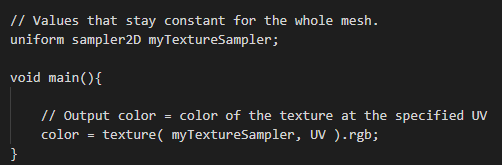
* Connect VBO với shader:
* 
* Là cặp chương trình hoạt động trong GPU, có nhiệm vụ rendering the polygon mesh lên màn hình.
* Mặc dù VBO ở trong GPU nhưng chưa có cách nào để connect VBO với shader program => tạo connect bằng function **glVertexAttribPointer()**
* 
* 
* glEnableVertexAttribArray: enables the generic vertex attribute array specified by *index*. By default, all client-side capabilities are disabled, including all generic vertex attribute arrays. If enabled, the values in the generic vertex attribute array will be accessed and used for rendering when calls are made to vertex array commands such as [glDrawArrays](https://www.khronos.org/registry/OpenGL-Refpages/es2.0/xhtml/glDrawArrays.xml) or [glDrawElements](https://www.khronos.org/registry/OpenGL-Refpages/es2.0/xhtml/glDrawElements.xml)
* Size: Vì ví dụ trên cần dữ liệu tọa độ => second param là 3 ~ biểu thị 3 trục tọa độ x, y, z
* Type: vertex data type
* Vertex Shader
* 
* #version => khai báo sử dụng opengl 3 syntax
* Vec3: là vector 3 components trong glsl
* layout(location = 0): chỉ cái vùng buffer khi **glVertexAttribPointer của** vertexPosition\_modelspace. Mỗi vertex có thể có nhiều attribute: color, coordinate, ...

=> Specify vùng nhớ nào tương ứng với attribute nào ( layout(location = 0): refers to the buffer we use to feed the vertexPosition\_modelspace attribute. Each vertex can have numerous attributes : A position, one or several colours, one or several texture coordinates, lots of other things. OpenGL doesn’t know what a colour is : it just sees a vec3. So we have to tell him which buffer corresponds to which input. We do that by setting the layout to the same value as the first parameter to glVertexAttribPointer. The value “0” is not important, it could be 12 (but no more than glGetIntegerv(GL\_MAX\_VERTEX\_ATTRIBS, &v) ), the important thing is that it’s the same number on both sides.)

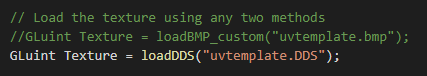
* “vertexPosition\_modelspace” could have any other name. It will contain the position of the vertex for each run of the vertex shader.
* “in” means that this is some input data
* 
* gl\_Position là một trong những built-in variables => xác định vị trí
* fragmentColor là output cho fragment shader nhận vào.
* Fragment shader



* Texture
* Trong fragment shader(uniform giống kiểu biến global trong glsl):



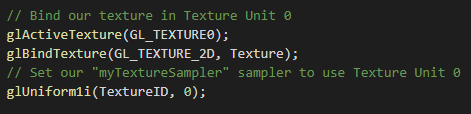
* Load texture vào



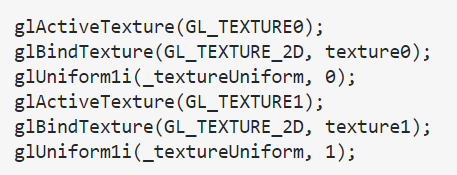
* Lấy textureID ứng với biến myTextureSampler (type: uniform) trong FragmentShader

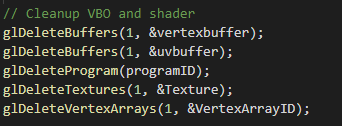


* Bind texture với texture unit 0 (có nhiều unit bởi vì shader có thể có nhiều texture). glUniform1i gắn cái TextureID (ứng với myTextureSampler) vào texture unit0



* Khi có nhiều texture:





-

## Tạo Model, View và Projection matrices