

**University of Petroleum and Energy Studies, Dehradun**

**Computer Graphics**

**Lab File**

Semester IV

By:

NAME – PRANJAL TRIPATHI

COURSE- B- TECH CSE (BAO)

BATCH – 01

ROLL NUMBER - R2142210964

SAI ID- 500096030

EXPERIMENT- 01

**OPEN GL:**

**1-Getting started with OpenGL:**

For rendering 2D and 3D Vector Graphics, the Open Graphics Library (OpenGL) is a cross-platform, cross-language (language-independent) API (use of polygons to represent image). The majority of the OpenGL API was developed for hardware.

Every function's expected output and outcome are detailed in the OpenGL standard, as well as how it should operate. The developers putting this specification into practise will then need to come up with a solution for how this function should work. The actual created versions of OpenGL are free to have multiple implementations as long as their outputs agree with the definition, as the OpenGL specification does not include implementation specifics (and are thus the same to the user).

**Design:** A set of functions that the client software may call is what is meant by this API. Although the functions are comparable to those of the C language, it is not language-specific.

**Development:** It is an evolving API and Chronos Group regularly releases its new version having some extended feature compare to previous one. Vendors of GPUs might also offer some extra functionality in the form of an extension.

**Associated Libraries:** The OpenGL utility library, a companion library, is included with the first edition. However, since OpenGL is a rather complicated procedure. Therefore, other libraries are introduced to make things simpler, such as the OpenGL Utility Toolkit, which is eventually replaced by free glut. GLEE, GLEW, and gliding were later added to the library.

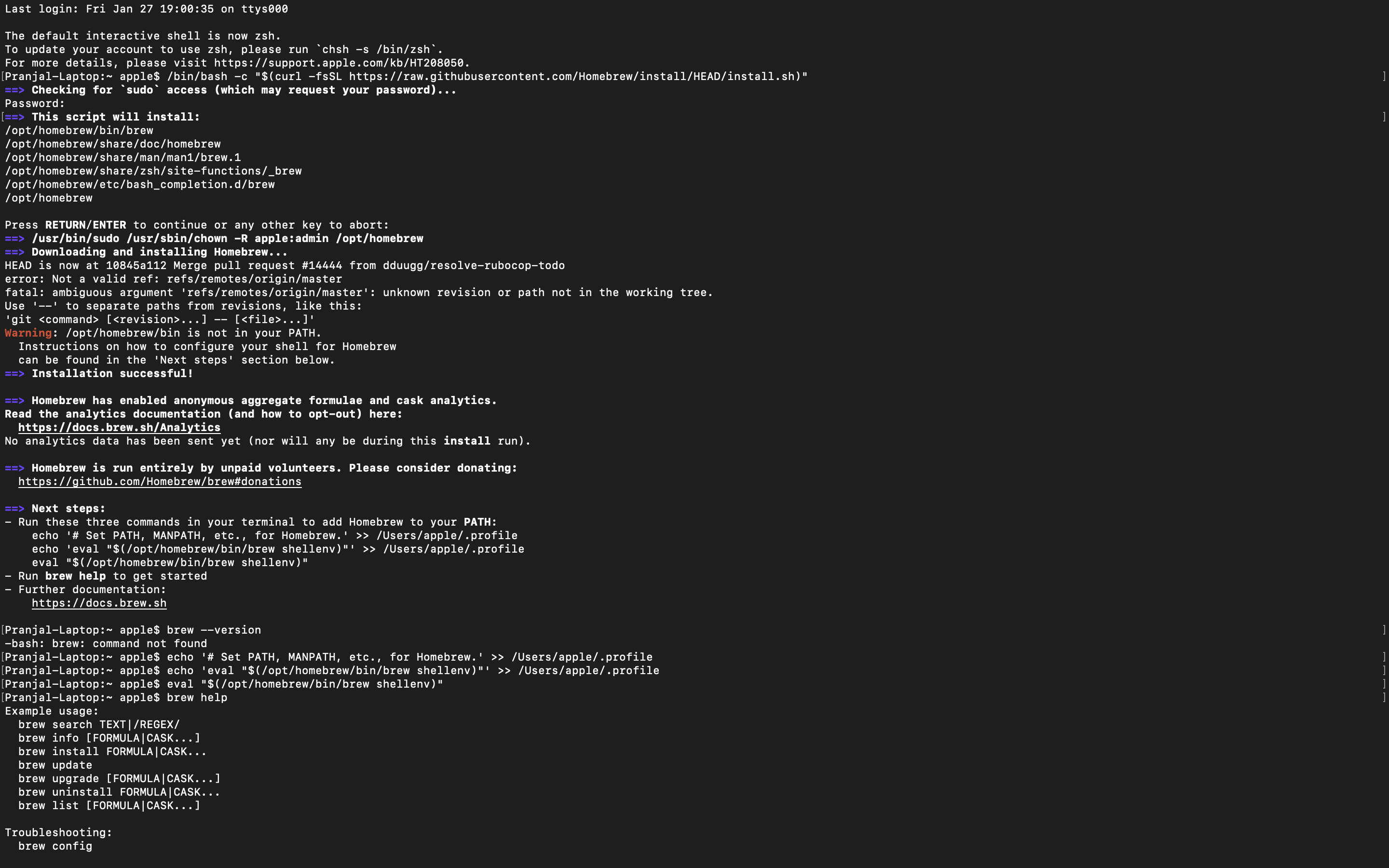
**Implementation: OpenGL** is implemented using Mesa 3D, an open source programmed. By utilizing Direct Rendering Infrastructure, it can do pure software rendering and hardware acceleration on BSD, Linux, and other systems.

Typically, the companies who make graphics cards are the ones that create the real OpenGL libraries. When you purchase a graphics card, it supports the specific versions of OpenGL that were created for that card (series). When utilising an Apple system, the OpenGL library is maintained by Apple, and under Linux, there are a variety of versions of these libraries made by graphic providers and amateurs.

**2-Steps to install Open GL:**

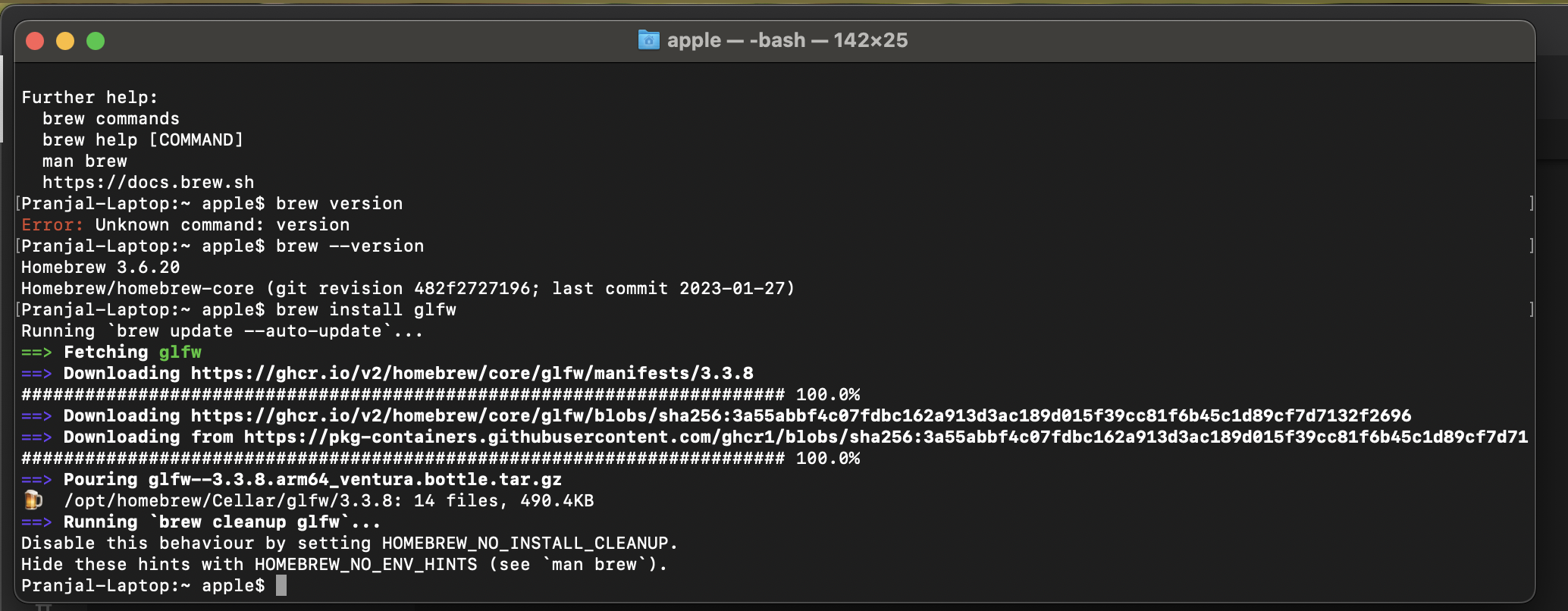
**STEP-01**

Install Brew :

****

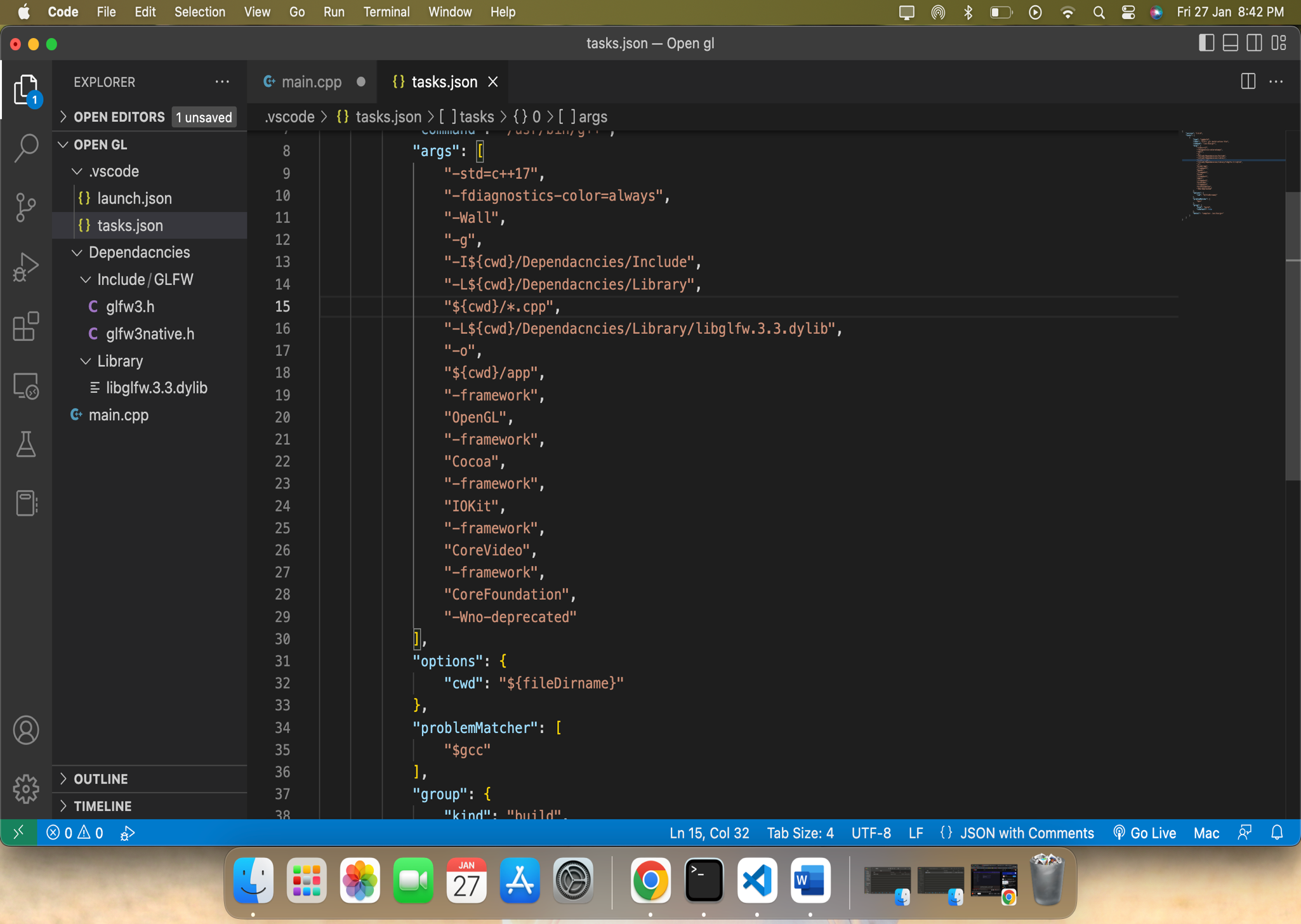
**Step-02**

Install Brew GLFW :



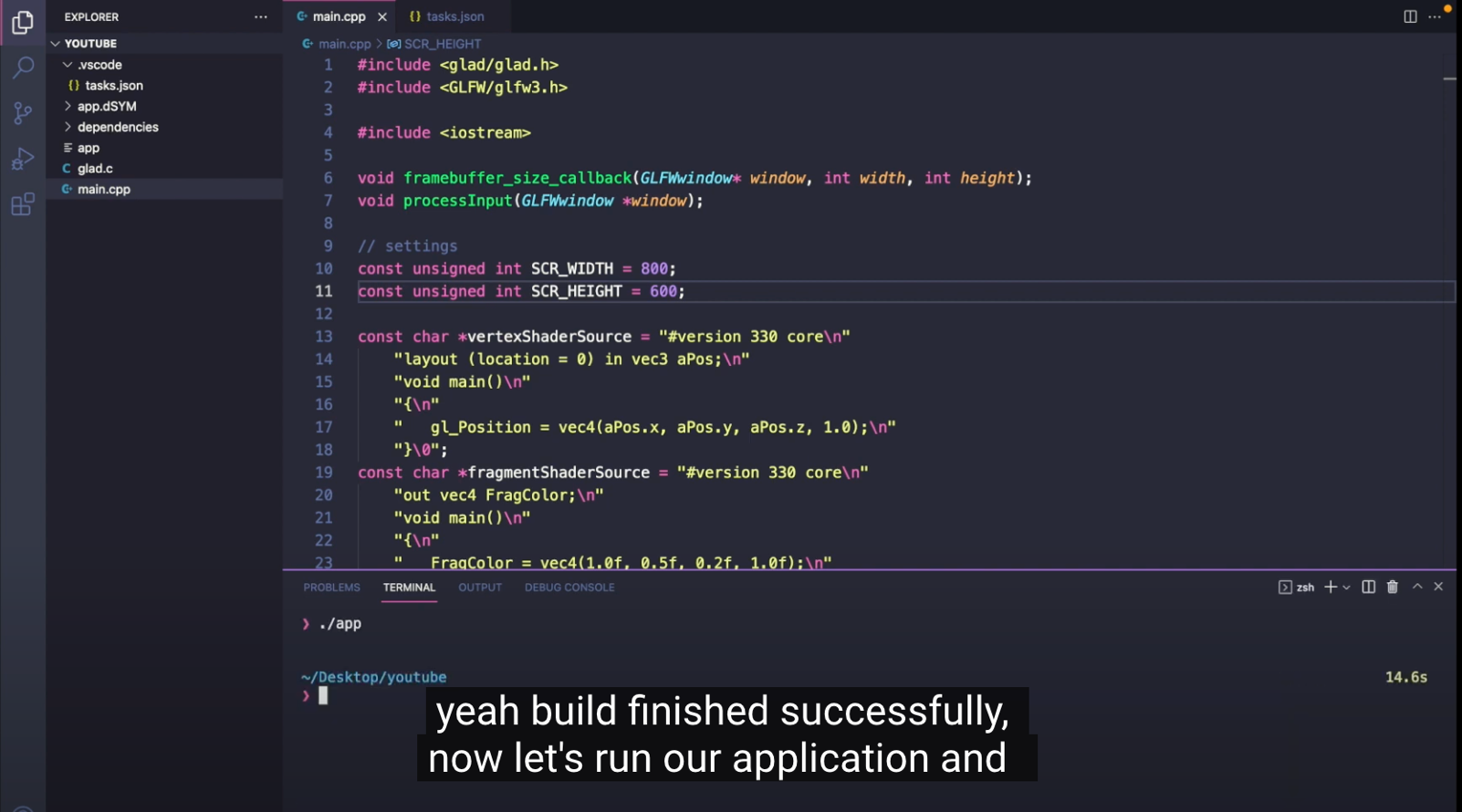
**STEP-03**

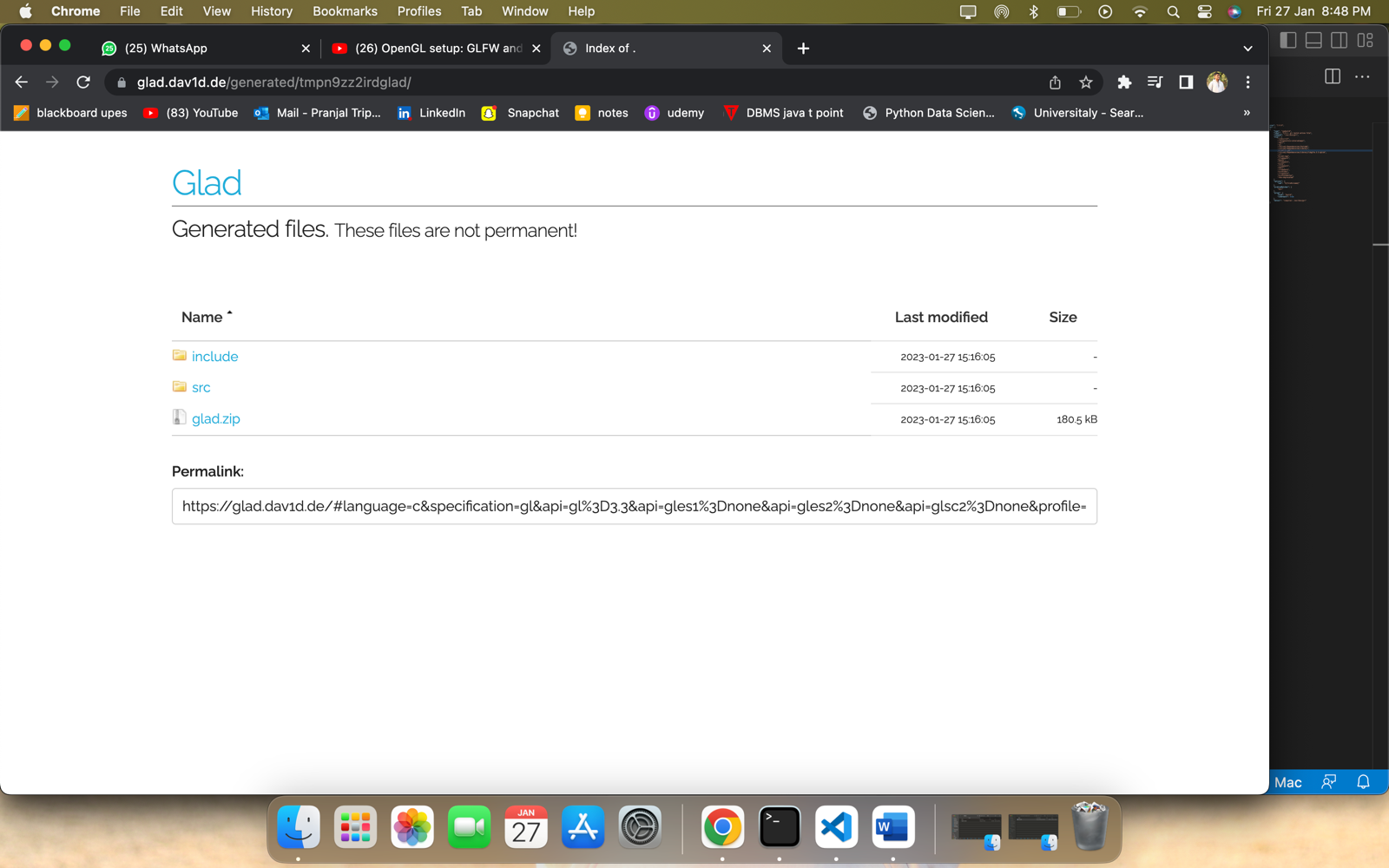
Setup Brew GLFW IN VS code



**STEP-04**

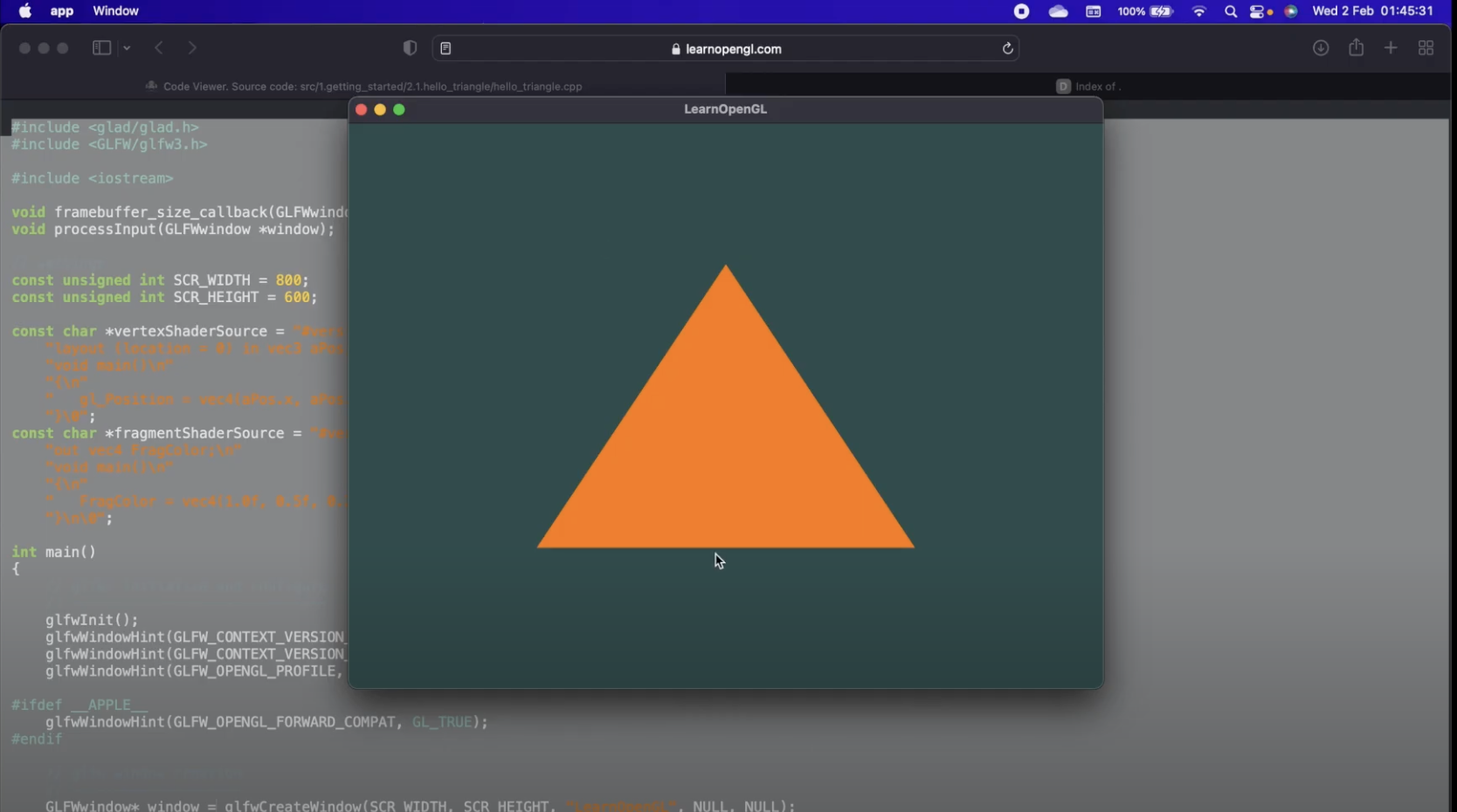
Download GLAD Open GL:





**Step-05**

Final Output for sample program



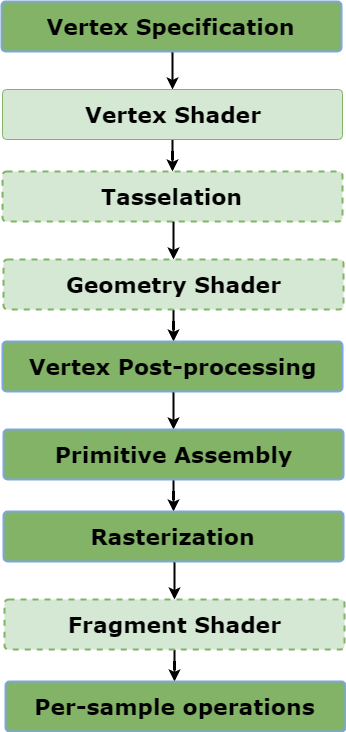
**3- OPEN GL PIPELINE**

**Vertex Specification :** List an ordered list of the vertices that characterise the boundaries of the primitive in vertex specification. Additional vertex properties, such as colour and texture coordinates, can be defined in addition to this. Later, the pipeline sends this data down and modifies it.

**Vertex Shader :** Vertex Shader now accepts the above-described vertex specification. A GLSL application called Vertex Shader is used to alter vertex data. Calculating each vertex's final vertex position is the vertex shader's main objective. The GPU examines a vertex once (or three times for a triangle) before executing a vertex shader. The vertex shader will therefore run a million times, one for each vertex, if the scene has a million vertices. Calculating the final placements of the vertices in the scene is the primary responsibility of a vertex shader.

**Tessellation : This step is optional. Primitives are tessellated, or separated into a smoother mesh of triangles, at this stage.**

**Geometry Shader : Additionally optional is this shader stage. The task of the geometry shader is to produce zero or more output primitives from a single input primitive. Geometry shader will display a string of triangles if a triangle strip is given as a single primitive. By producing numerous primitives from a single input, Geometry Shader can delete primitives or tessellate them. Primitives can be changed into other types using geometry shaders. Point primitives, for instance, can transform into triangles.**



**Vertex Post Processing : This stage has a fixed function, meaning that the user has little to no control over it. Clipping is the most significant component of this stage.**

**Fragment Shader :** **Although not essential, this stage is employed 96% of the time. The colour of each piece that the viewer sees on the screen is determined by this user-written GLSL application.**

**Rasterization :** In this process, this is a crucial phase. A fragment is what rasterization produces.

**Primitive Assembly :** This stage collects the vertex data into a ordered sequence of simple primitives(lines, points or triangles).

**Thank You !**