*All of these notes are derivative of **https://learnopengl.com** material, if not pulled directly from it.*

OPENGL

OpenGL by itself is not an API, but a **specification**, maintained and developed by the **Khronos Group**.
OpenGL is usually developed by graphics card manufacturers, specifically for their graphics cards.
OpenGL libraries are written in **C**.
If there is a bug, try updating your graphics drivers.

OpenGL specification of **version 3.3**:

Core Profile vs. Immediate Mode

Legacy OpenGL involved developing in immediate mode (otherwise known as the fixed function pipeline).

- Easy to use and understand, but…
- Lacked flexibility; developers had little control over calculations
- Inefficient

Immediate mode functionality was deprecated from 3.2 onwards.
Core-profile mode is a division of OpenGL's specification which removed all the old deprecated immediate mode functionality.

- Throws an error when using immediate mode functionality
- Very flexible and efficient
- More difficult to learn
- (En)forces understanding of graphics programming

Therefore, while core-profile mode is more difficult to learn, it is much more fruitful and can help expand one's understanding of graphics programming as a who le.

Why OpenGL 3.3 Over Newest Version (4.6)?

The newer versions will add extra features or slightly more efficient/useful ways to accomplish the same tasks, but the concepts and techniques will remain the same.
You may want to use a more recent version of OpenGL after gaining a solid foundation of graphics programming using OpenGL 3.3 .

- **NOTE:** Only the most modern graphics cards will be able to run an application using the most recent version of OpenGL. For that reason, lower versions are typically targeted, with higher version functionality being optionally enabled where necessary.

Extensions

Extensions are supported by OpenGL, and can offer more advanced or efficient graphics.
Allow developers to utilize new rendering techniques without waiting for OpenGL to include the functionality in a future vers ion.
However, it is important to check that the extension is supported by the hardware the application runs on.

State Machine

OpenGL is a large state machine:

- A collection of variables that define how OpenGL should operate.

The state of OpenGL is commonly referred to as the context.
Using OpenGL is a matter of changing its state by setting some options, manipulating some buffers, and then rendering using the current context.

- e.g. If we want to draw lines instead of triangles, we change the state of OpenGL by changing some context variable that sets how OpenGL should draw.

State-Changing functions will change the context and State-Using functions will perform some operation based on the current state.

Objects

OpenGL libraries are written in **C**.
Objects in OpenGL are simply an abstraction, since C does not support OOP.

- Object: *A collection of options that represents a subset of OpenGL's state.*

Example: An object that represents the settings of a drawing window, visualized as a C-like struct:

```
struct object_name {
    float  option1;
    int    option2;
    char[] name;
};
```

If we visualize OpenGL's context as a large struct, using an object would look like this:

```
struct OpenGL_Context {
    ...
    object_name* object_Window_Target;
    ...
};
```

```
// create object
unsigned int objectId = 0;
glGenObject(1, &objectId);
// bind/assign object to context
glBindObject(GL_WINDOW_TARGET, objectId);
// set options of object currently bound to GL_WINDOW_TARGET
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_WIDTH,  800);
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_HEIGHT, 600);
// set context target back to default
glBindObject(GL_WINDOW_TARGET, 0);
```

This is a **very common** workflow when using OpenGL:

1. **Create** an object and store a reference to it as an id (the real object's data is stored behind the scenes).
2. **Bind** the object (using its id) to the target location of the context.
3. **Set** the options of the target.
4. **Unbind** the object from the state by setting the current object id of the target to its default (in this case, 0).

Though the object is unbinded from the context, the options we set are stored in the object referenced by objectId and restored if the object is binded back to GL_WINDOW_TARGET.
You can define more than one object in the application, allowing you to situationally bind objects with specific settings when using OpenGL's state.

- ***Example:*** There are objects that act as container objects for 3D model data. Whenever we want to draw something like a house, we **bind** the object containing the model data that we want to draw (assuming we first created and set options for these objects).