Section 2 Create a Window in GLFW

In this section, we'll implement a window using GLFW. I'll explain the code step by step and provide the complete version at the end of this page.

Step 1 : Include our libraries and initialize GLFW

```
#include<iostream>
#include<glad/glad.h>
#include<GlFW/glfw3.h>

int main()
{
        //initialize GLFW
        glfwInit();
}
```

`glfwInit()` is used to set up the necessary resources and state for GLFW to function properly.

Here's what it does:

1. **Resource Allocation**: It allocates any internal resources needed by GLFW.
2. **Platform Initialization**: It initializes the platform-specific components, such as window creation and input handling.
3. **Error Handling**: It sets up the error handling mechanisms so that any issues encountered by GLFW can be reported.

Step 2 : Set up the version of OpenGL (in our case its version3.3) and Profile

```cpp
#include<iostream>
#include<glad/glad.h>
#include<GlFW/glfw3.h>

int main()
{
        //initialize GLFW
        glfwInit();
        // Set the version of OpenGL and Profile
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
        glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

}
```

the `glfwWindowHint()` is used to set hints for the next call to `glfwCreateWindow`. These hints specify the desired properties of the window and its context.

a `context` refers to an OpenGL or OpenGL Embedded systems context, which is essentially a container for all the state and resources needed for rendering with OpenGL.

Here's what each of these hints means:

1. `GLFW_CONTEXT_VERSION_MAJOR` **and** `GLFW_CONTEXT_VERSION_MINOR`:
    - These hints specify the desired major and minor version of the OpenGL context.
    - because we want version 3.3 the major version is 3 and the minor version is 3
2. `GLFW_OPENGL_PROFILE`:
    - This hint specifies which OpenGL profile to create the context for. The profile determines the features and behavior of the OpenGL context.
    - This requests a core profile context, which means that deprecated features of OpenGL are not available

Step 3: Create our Window and error check it

```cpp
#include<iostream>
#include<glad/glad.h>
#include<GlFW/glfw3.h>

int main()
{
	//initialize GLFW
	glfwInit();
	// Set the version of OpenGL and Profile
	glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
	glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
	glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

	//Create window
	GLFWwindow* window = glfwCreateWindow(800, 500, "Welcome to OpenGl", NULL, NULL);
	if (window == NULL)
	{
		std::cout << "Failed to create GLFW window" << std::endl;
		// Terminate GLFW
		glfwTerminate();
		return -1;
	}

}
```

Let's break down the statement `GLFWwindow* window = glfwCreateWindow(800, 500, "Welcome to OpenGl", NULL, NULL);` step by step:

1. `GLFWwindow*`:
   - This is a pointer to a `GLFWwindow` structure. The `GLFWwindow` structure represents a window and its associated OpenGL or OpenGL ES context.
2. `window`:
   - This is the variable name that will hold the pointer to the created window.
3. `glfwCreateWindow`:
   - This is the function call that creates a window and its associated context. It returns a pointer to a `GLFWwindow` structure.
4. **Parameters of** `glfwCreateWindow`:
   - `800`: The desired width of the window in screen coordinates.
   - `500`: The desired height of the window in screen coordinates.
   - `"Welcome to OpenGl"`: The initial, UTF-8 encoded window title.
   - `NULL`: The monitor to use for full-screen mode. Passing `NULL` means the window will be created in windowed mode.
   - `NULL`: The window whose context to share resources with. Passing `NULL` means the window will not share resources with any other window.

Step 4: Make GLFW Context using the created window and load GLAD

```cpp
#include<iostream>
#include<glad/glad.h>
#include<GlFW/glfw3.h>

int main()
{
	//initialize GLFW
	glfwInit();
	// Set the version of OpenGL and Profile
	glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
	glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
	glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

	//Create window
	GLFWwindow* window = glfwCreateWindow(800, 500, "Welcome to OpenGl", NULL, NULL);
	if (window == NULL)
	{
		std::cout << "Failed to create GLFW window" << std::endl;
		// Terminate GLFW
		glfwTerminate();
		return -1;
	}

	// Make the window's context current
	glfwMakeContextCurrent(window);
	//Load GLAD so it configures OpenGL
	gladLoadGL();

}
```

`glfwMakeContextCurrent(window);`

This function sets the OpenGL or OpenGL ES context of the specified window as current on the calling thread. Here's a detailed explanation:

1. **Function Purpose**:
   - `glfwMakeContextCurrent` sets the current context for the calling thread. This is crucial because OpenGL functions operate on the current context, and only one context can be current per thread at a time.
2. **Parameter**:
   - `window`: A pointer to a `GLFWwindow` structure. The context associated with this window becomes the current context.
3. **Behavior**:
   - Calling `glfwMakeContextCurrent(window);` makes the context of the specified window current for the calling thread.
   - If a context was previously current on the calling thread, it's detached.
   - Passing `NULL` instead of a window detaches the current context without making a new one current.

`gladLoadGL()`

The `gladLoadGL` function is used to load all the OpenGL function pointers, making them available for use in your application. This is necessary because OpenGL functions are not directly accessible by default; they need to be loaded at runtime.

Step 5: Specify Viewport ,background color

```cpp
#include<iostream>
#include<glad/glad.h>
#include<GLFW/glfw3.h>

int main()
{
	//initialize GLFW
	glfwInit();
	// Set the version of OpenGL and Profile
	glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
	glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
	glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

	//Create window
	GLFWwindow* window = glfwCreateWindow(800, 500, "Welcome to OpenGl", NULL, NULL);
	if (window == NULL)
	{
		std::cout << "Failed to create GLFW window" << std::endl;
		// Terminate GLFW
		glfwTerminate();
		return -1;
	}

	// Make the window's context current
	glfwMakeContextCurrent(window);
	//Load GLAD so it configures OpenGL
	gladLoadGL();

	// Specify the viewport of OpenGL in the Window
	// In this case, the viewport goes from x = 0, y = 0, to x = 800, y = 500
	glViewport(0, 0, 800, 500);

	// Specify the color of the background
	glClearColor(0.07f, 0.13f, 0.17f, 1.0f);

	// Clean the back buffer and assign the new color to it
	glClear(GL_COLOR_BUFFER_BIT);

	// Swap the back buffer with the front buffer
	glfwSwapBuffers(window);

}
```

`glViewPort()`

The `glViewport` function tells OpenGL where to draw on the screen. Think of it as setting the area of your window where your graphics will appear.

Parameters

- `x` **and** `y`: These are the coordinates of the bottom-left corner of the area where you want to draw. Usually, you start at (0, 0), which is the bottom-left corner of the window.
- `width` **and** `height`: These define how wide and tall the drawing area should be

What is a Viewport?

The viewport is like a picture frame. It defines the part of the window where your OpenGL drawings will show up.

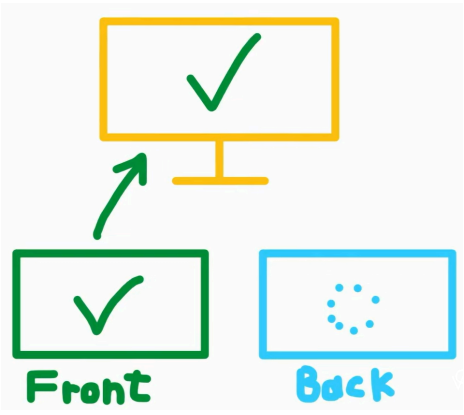
---

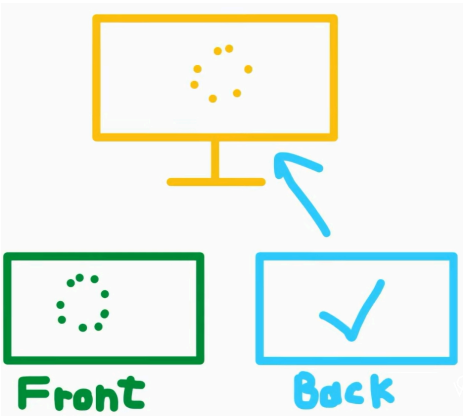
before explaining `glClearColor()` , `glClear()` and `glfwSwapBuffers()` we need to understand how screens display images and how these screens changes Frames.

In OpenGL, frames are typically displayed from left to right

OpenGL uses a technique called **double buffering** to render smooth animations and avoid flickering. Here's how it works:

1. **Front Buffer**:
    - The front buffer is the buffer that is currently being displayed on the screen. This is what the user sees.
2. **Back Buffer**:
    - The back buffer is where all the rendering commands are directed. You draw your scene to the back buffer, not directly to the screen.
3. **Swapping Buffers**:
    - Once rendering is complete, the back buffer is swapped with the front buffer. This swap is usually done using a function like `glfwSwapBuffers(window);` in GLFW.
    - The back buffer becomes the front buffer (now visible on the screen), and the front buffer becomes the back buffer (ready for the next frame of rendering).





To change the background color, we use `glClearColor(Red float, Green float, Blue float, alpha value as float);` to specify our desired color. Next, we clear the back buffer and apply our color values using `glClear(GL_COLOR_BUFFER_BIT);`. Finally, we swap the back buffer with the front buffer using `glfwSwapBuffers(window);` to display our changes.

Step 6: Create a loop

We create a loop in graphical applications, like the one using GLFW, to continuously update and render the scene.

```cpp
#include<iostream>
#include<glad/glad.h>
#include<GlFW/glfw3.h>

int main()
{
	//initialize GLFW
	glfwInit();
	// Set the version of OpenGL and Profile
	glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
	glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
	glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

	//Create window
	GLFWwindow* window = glfwCreateWindow(800, 500, "Welcome to OpenGl", NULL, NULL);
	if (window == NULL)
	{
		std::cout << "Failed to create GLFW window" << std::endl;
		// Terminate GLFW
		glfwTerminate();
		return -1;
	}

	// Make the window's context current
	glfwMakeContextCurrent(window);
	//Load GLAD so it configures OpenGL
	gladLoadGL();

	// Specify the viewport of OpenGL in the Window
	// In this case, the viewport goes from x = 0, y = 0, to x = 800, y = 500
	glViewport(0, 0, 800, 500);

	// Specify the color of the background
	glClearColor(0.07f, 0.13f, 0.17f, 1.0f);

	// Clean the back buffer and assign the new color to it
	glClear(GL_COLOR_BUFFER_BIT);

	// Swap the back buffer with the front buffer
	glfwSwapBuffers(window);

	while (!glfwWindowShouldClose(window))
	{

	    // Poll for and process events
	    glfwPollEvents();

	}

	// Terminate GLFW, clearing any resources allocated by GLFW.
	glfwDestroyWindow(window);

	// Terminate GLFW before ending the program
	glfwTerminate();
	return 0;


}
```

Inside the loop, we use `glfwPollEvents();` to handle events:

• `glfwPollEvents` checks for triggered events such as key presses, mouse movements, clicks, window resizing, and more.

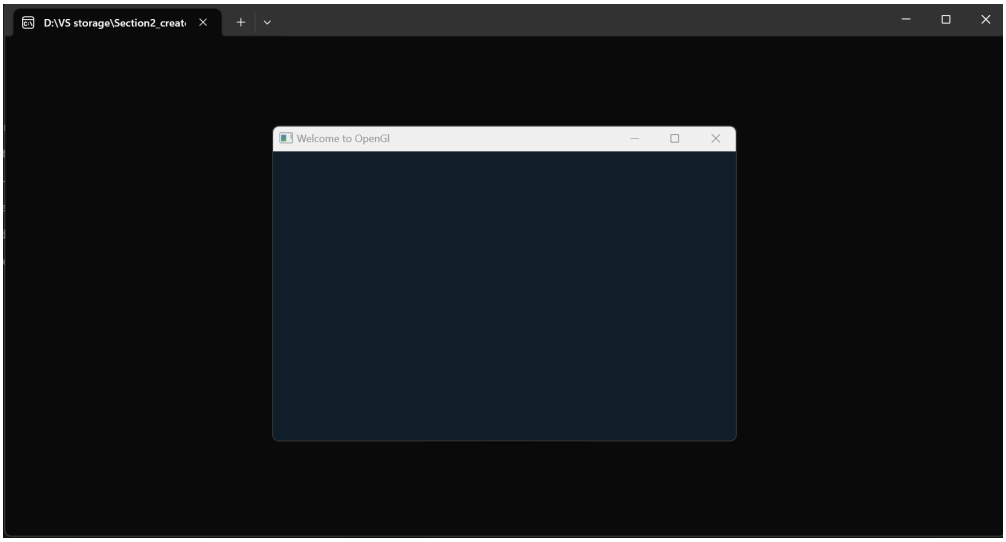
• It then processes these events, updating the window's state and calling any set-up callback functions for specific events.

Without this function, the window would be unresponsive.

finally we Clear any resources allocated by GLFW using `glfwDestroyWindow(window)`

and terminate GLFW using `glfwTerminate()` .

OUTPUT:



ASSIGNMENT 1

Exercise 1. Change the color of the window to a shade of Orange

Exercise 2. Change the size of the window to width = 400, height = 225

Exercise 3. Change the name of the window to “I made this! “