Ray3D Documentation

Framebuffer Size

- Game.cpp → Init_Window()
- Framebuffer size could be different from the window's size

```
glfwGetFramebufferSize(window, &framebufferWidth, &framebufferHeight);
```

- This retrieves the size of the window's framebuffer - a collection of buffers *(e.g., color, depth, and stencil buffers)* used to render images to the screen.

```
glfwSetFramebufferSizeCallback(window, FramebufferResizeCallback);
```

- Setting a callback function that GLFW will call whenever the window's framebuffer is resized. This helps in dynamically adjusting the viewport when the window size changes.

```
inline static void FramebufferResizeCallback(GLFWwindow* window, int fbw,
{
    glViewport(0, 0, fbw, fbh);
}
```

- It updates the OpenGL viewport to match the new dimensions of the framebuffer.

GL_DEPTH_TEST: Depth testing is crucial for rendering 3D scenes correctly, ensuring that objects that are closer to the camera obscure objects behind them.

GL_CULL_FACE: Face culling allows OpenGL to skip rendering certain faces of objects based on their orientation.

glCullFace(GL_BACK): OpenGL culls the back faces *i.e.* the faces that are facing away from the polygon.

glFrontFace(GL_CCW): Ensuring that any polygons whose vertices are ordered counterclockwise will be considered the front face.

GL_BLEND: It helps in rendering transparent objects.

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA): *GL_SRC_ALPHA* means the source color is multiplied by its alpha value. *GL_ONE_MINUS_SRC_ALPHA* means destination color is multiplied by 1 minus the source's alpha.

glPolygonMode(GL_FRONT_AND_BACK, GL_FILL): This will render polygons as filled shapes.

---

glm::lookAt()

```
//Syntax
glm::mat4 lookAt(const glm::vec3& eye, const glm::vec3& center,
```

```
    const glm::vec3& up);

//Code
viewMatrix = glm::lookAt(camPos, camPos + camFront, worldUp);
```

- ***eye:*** Position of the camera (camPos)
- ***center:*** point that the camera is looking at (camPos + camFront)
- ***up:*** up direction of the world (worldUp)

---

glVertexAttribPointer

- It maps your vertex buffer data to the vertex shader's attributes.

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

//Example
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex),
    (GLvoid*)offsetof(Vertex,   position));

layout (location = 0) in vec3 v_position;  //Code from VertexShader.glsl
```

- ***index:*** Location of the vertex attribute in your shader program. Likein above example, the index would be 0, since *position* is bound to location 0.
- ***size:*** No. of components we are providing like in above example, since we are passing a *'vec3',* we pass in size 3.
- ***type:*** Data type
- ***normalized:*** Normailzed to range [0, 1] for unsigned, and [-1, 1] for signed.
- ***stride:*** No. of bytes b/w consecutive vertex attributes.
- ***pointer:*** Offset in the buffer where the first component of the attribute data is located. It is passed in bytes, so it essentially a memory offset.
