Explicación código OpenGL renderer

El ciclo del programa es el típico se crea en el main.cpp una window por SFML/Window, tras esto se inicia el contexto de OpenGL que sería el GLAD, y se crea una escena que en este caso se llama View además de una variable enfocada para el movimiento de cámara camera_pos y comienza el ciclo infinito de la ventana hasta que se haga un Evento detectado por SFML de cierre de ventana presionando la X de la misma. Adicionalmente también se han definido casos para si la ventana se pone en formato de pantalla completa y si se ha presionado una tecla dicho camera_pos que obtiene la cámara de la escena obtiene su posición y se la asigna y lo que se hace en los métodos de KeyPressed es usar dicha variable de copia de la posición de la cámara y tras modificar el vector camera_pos se le settea a la cámara gracias a que es hija de Entity.

Tras esto entramos en la Scene o View donde primero de todo se configuran valores iniciales par variables de View, como son angle, camera y las texturas con sus rutas correspondientes. Después, se define la configuración básica de la escena, las cullfaces y la profundidad y se settea el color del fondo a negro por el glClearColor. A continuación se compilan los shaders por los métodos de la clase Shaderer que a grandes rasgos lo que hacen es obtener los datos de shader de un archivo gracias a las funciones parse_shader() y el struct shader_code, que ayuda mucho a poder hacer return de los dos tipos de dato que se llenan en parse_shader() que son las dos partes del shader el vertex y el fragment. Luego dichos datos se compilan glShaderSource, se attachean los shader, se linkea el programa y luego se devuelve el id del programa de shaders.

Tras el linkeo de shaders descrito se asignan los valores a model_view_matrix y projection_matrix en GLSL para los archivos shader. Se setta la cámara en el 0,0,0 y tras esto se bindean las texturas, en el 0 y el 2 las texturas reales y en el hueco 1 se bindea la textura para la malla de elevación. Tras bindear las texturas se cargan las texturas en los modelos. Hablando de los modelos estos son hijos de Entity para simpleza al moverlos, adicionalmente a parte de getters presenta una Mesh que lo único que tiene son todos los datos típicos para carga de modelos en OpenGL como son los VBOs y sucedáneos para luego ser usados en su función load_mesh que pide un path para la mesh.

Sobre el proceso de load_mesh se comprueba si la escena es válida y si es válida se obtiene la primera mesh del objeto y se inicia el ciclo de OpenGL donde se van creando los buffers , añadiéndoles BufferData y finalmente bindeandolos, al llegar a los atributes debido a como están definidos en el shader de GLSL el hueco 0 corresponde a las coordenadas y el 1 a la textura, sobre la textura debido a como el dato devuelto es un aiVector3D pero solo los dos primeros huecos del vector 3 tienen valor así que se crea un vector de vec2 que copia las coordenadas de textura en x e y, luego eso es lo usado para definir el buffer data a un tamaño apropiado. Y finalmente se definen los índices.

Tras haber ya creado los modelos por el load_mesh() de las meses de los modelos en los mismos se setten sus posiciones iniciales fácilmente gracias a Entity y después se añaden al mapa de modelos, que debido a dificultades técnicas relacionadas con las mallas de elevación se ha

debido básicamente dejar de usar por lo que no voy a entrar mucho en detalle sobre ello. Pero su idea era actuar como manera de capar el bucle de renderizado en render().

Después, llegamos a el update donde se actualiza la variable local angle y luego se configura en los modelos otra vez gracias a como son parte de Entity. Finalmente, se entra en el bucle de renderizado donde como se ha dicho debido a la malla de elevación y su necesidad de variables uniform y como al settear las variables a menos que se eliminen se seguían aplicando a los otros modelos se decidió el hacer uso de un booleano en el shader del que depende si la glPosition es en base a la posición normal o la deformada por la malla de elevación.

A parte de eso el ciclo de renderizado es el típico donde se crea la matriz de transformación en base a el translate, rotate y la matriz inversa de la cámara.