THE WELL STACKED PIZZA IN OPENGL

TECHNICAL MANUAL

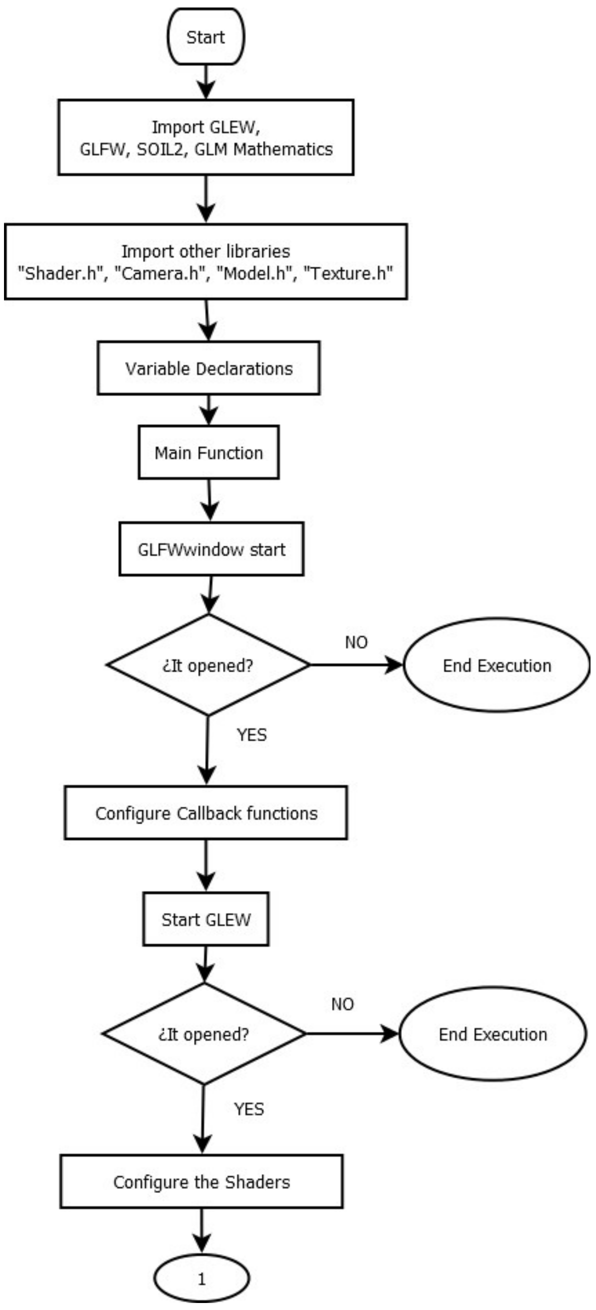
DANTE MOISÉS ARGÜELLO LEÓN
https://github.com/dargleon/thewellstackedpizza
June 8th, 2023

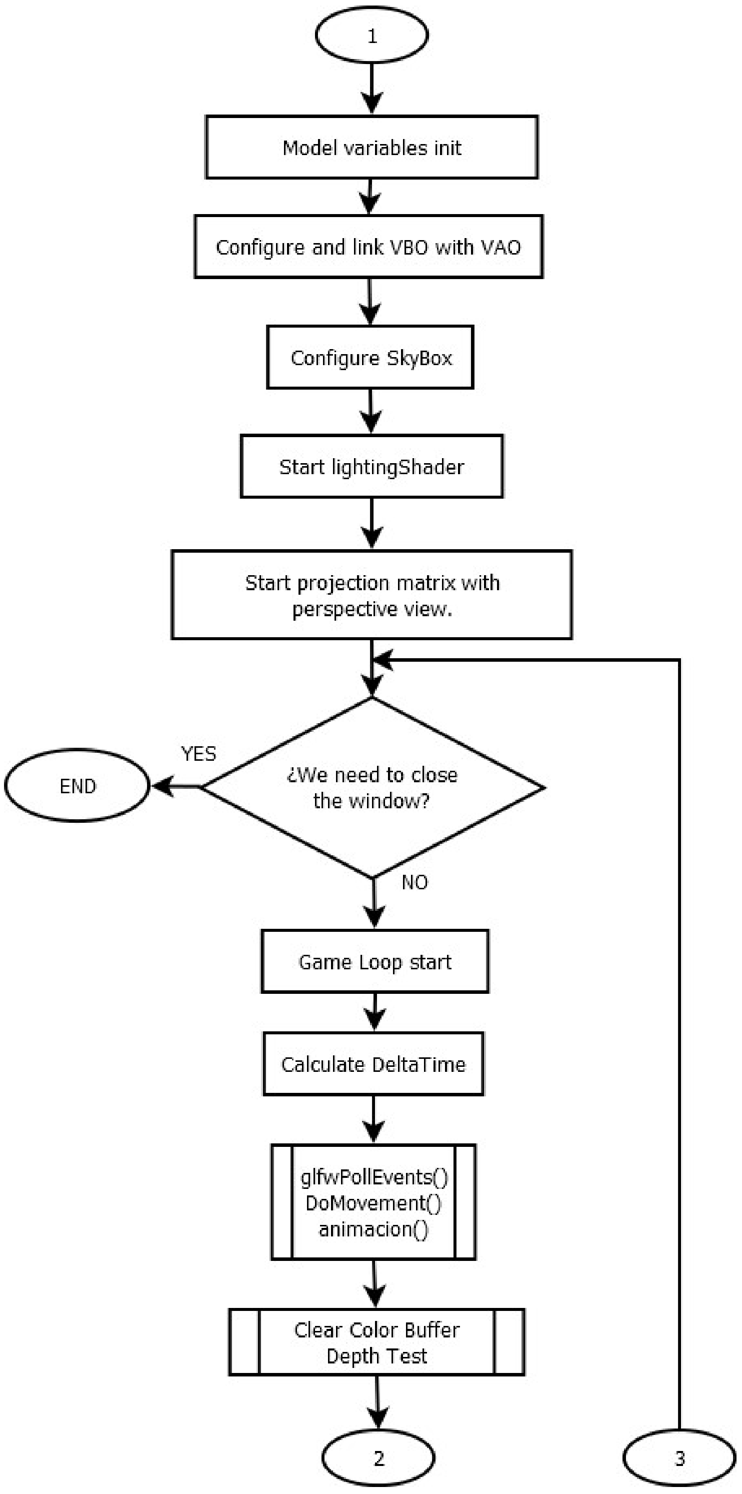
Technical Manual

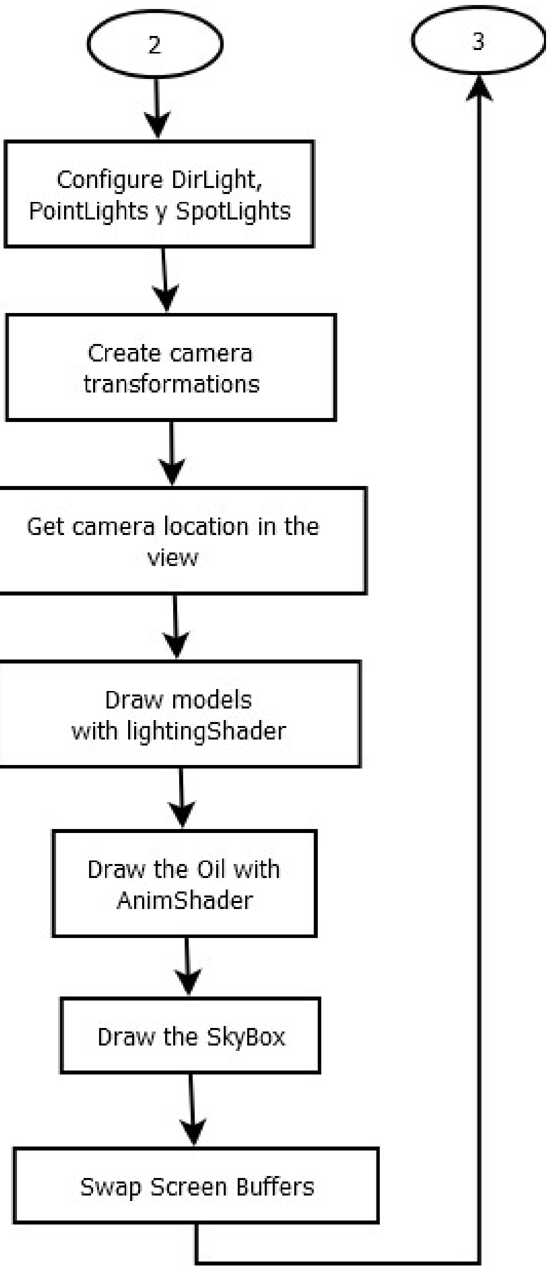
Goals

- Understand the elements that are part of an implementation of a virtual environment using OpenGL and C++
.

Software Flow Chart




1
Model variables init
Configure and link VBO with VAO
Configure SkyBox
Start lightingShader
Start projection matrix with perspective view.
¿We need to close the window?
YES
END
NO
Game Loop start
Calculate DeltaTime
glfwPollEvents()
DoMovement()
animacion()
Clear Color Buffer
Depth Test
2
3


2
3
Configure DirLight, PointLights y SpotLights
Create camera transformations
Get camera location in the view
Draw models with lightingShader
Draw the Oil with AnimShader
Draw the SkyBox
Swap Screen Buffers

Gantt diagram

TASK	TIME FRAME	MARCH 2023					APRIL 2023			
TABLE MODELING	16/MAR - 30/MAR			🟩	🟩	🟩				
CHAIR MODELLING	30/MAR - 4/APR						🟩			
AUTODESK MAYA STUDY	4/APR - 18/MAY						🟩	🟩	🟩	🟩
GRILL MODELLING	10/APR - 3/MAY							🟩	🟩	🟩

TASK	TIME FRAME	MAY 2023																									
AUTODESK MAYA STUDY	4/APR - 18/MAY	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩							
GRILL MODELING	10/APR - 3/MAY	🟩	🟩	🟩																							
COUCH MODELING	5/MAY - 12/MAY					🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩														
UPLOADING OBJECTS ON GITHUB	12/MAY - 14/MAY												🟩	🟩	🟩												
LONG GRILL MODELING	12/MAY - 15/MAY												🟩	🟩	🟩	🟩											
INDOOR DEVELOPMENT	13/MAY - 20/MAY													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩						
SMALL GRILL MODELING	15/MAY - 16/MAY															🟩	🟩										
PIZZA SHOWCASE MODELING	16/MAY - 17/MAY																🟩	🟩									
USING SETS OF OBJECTS	17/MAY- 19/MAY																	🟩	🟩	🟩							
INTEGRATION OF OBJECTS	17/MAY - 25/MAY																	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	
OUTDOOR DEVELOPMENT	20/MAY - 22/MAY																				🟩	🟩	🟩				
DEVELOPMENT OF ANIMATIONS	22/MAY - 24/MAY																						🟩	🟩	🟩		
SKYBOX	25/MAY - 25/MAY																									🟩	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

TASK	TIME FRAME	JUNE 2023							
PREP. TABLE MODELLING	2/JUNE - 3/JUNE		🟩	🟩					
TABLE WITH DOORS MODELLING	3/JUNE - 5/JUNE			🟩	🟩	🟩			
PIZZA OVEN MODELLING	3/JUNE - 5/JUNE			🟩	🟩	🟩			
CASH REGISTER MODELLING	6/JUNE - 8/JUNE						🟩	🟩	🟩
MICROWAVE OVEN MODELLING	6/JUNE - 8/JUNE						🟩	🟩	🟩
NEW ANIMATIONS	8/JUNE								🟩
RELEASE	8/JUNE								🟩
		1	2	3	4	5	6	7	8

Project Scope

The modelling and texturing of the Pizza Parlour "The Well Stacked Pizza" present in the video game GTA: San Andreas is proposed, following the models and reference images presented; It will be a matter of detailing the interior of the Pizza Parlour as it is an environment that can be divided into 2 rooms, the dining room and the kitchen, both have many elements; So, for the kitchen, it is planned to correctly model the objects Small Table, Chair, Armchair, Long Table and the Pizza Display Case. While, from the kitchen, he plans to model the Long Grill, the Small Grill, the Fryer, the Preparation Table, and the Pizza Oven. As well as the incorporation of at least 5 animations and more models that complement the interior and exterior setting.

Software Methodology

An incremental methodology is used, with a focus on bottom-up development; This is because the project gives the opportunity to segment progress in different areas, making somewhat small commits, but ensuring progress.

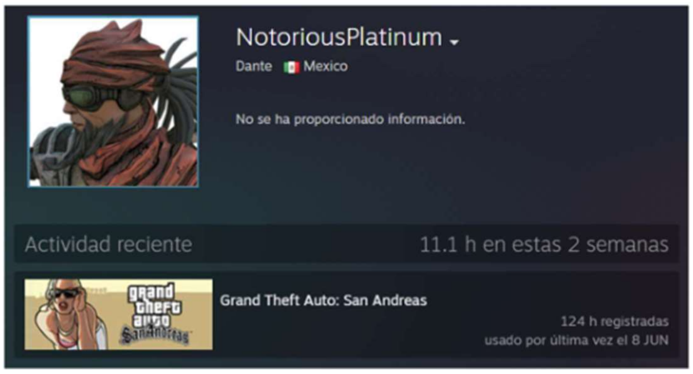
In the Incremental methodology, software development builds the final product progressively. At each incremental stage, new functionality is added, allowing you to see results faster compared to the waterfall model. The software can be started even before it is fully completed and is generally much more flexible than other methodologies.

The development approach is bottom-up, since the first advances consist in the development of objects, which are individual elements within the entire project and that are at the base of the modelling; Subsequently, the structures were modelled maintaining the scale of the objects, and with the structures made, sets of individual objects are incorporated as in the case of the "MesasYSillas" or "MesasYSilliones" objects that will be seen in the code documentation.

With the individual objects and the sets of objects placed in the structures, it is possible to start carrying out the integration actions, which are part of the "up" development since they required the other bases. Within the integration actions, we can see the separation of some objects in order to animate them or the incorporation of lighting for all the models in a single scene. The nature of the Incremental methodology is appreciated in each of the GitHub commits since they are all possible to compile to see the respective progress.

Developing Time

Regarding the hours used for development, it was proposed from the beginning to keep the video game GTA: San Andreas running in the background, since the number of hours used is counted on the Steam platform. It was necessary to have the video game running as it served as a reference to make detailed observations while modelling, and during programming to make spatial locations. Below is the time spent in-game based on the Steam platform.


NotoriousPlatinum
Dante Mexico
No se ha proporcionado información.
Actividad reciente 11.1 h en estas 2 semanas
Grand Theft Auto: San Andreas
124 h registradas
usado por última vez el 8 JUN

Before starting the project, 48 hours of play had been registered, so if we make the difference of 124 hours - 48 hours, we are left with the fact that the project took us approximately 76 hours of modelling and part of the programming.

For the complete programming part, it is established that 25 hours of animation development, lighting and logic were used.

Code Documentation

The code makes use of the GLEW, GLFW, GLM Mathematics and SOIL2 libraries.

The code begins by declaring the function prototypes it will use, as well as the variables needed for window and camera management. There is a vector with the position of each of the PointLights.

Another of its starting vectors is the so-called vertices that contains the vertices of the 6 faces of a cube made up of triangles. This cube is necessary when we want to graphically represent the position of the PointLights.

We also declare the float and boolean animation variables, which are:

Float:

- movSilla, will contain the displacement of an individual chair in the X axis.
- movCanasta, will contain the displacement of the basket of the fryer un the Y axis.
- movHornoPizz, will contain the displacement of the interior of the Pizza Oven in Z axis.
- rotPuertaPrincipal, will contain the radians of rotation in the Y axis.
- rotPuertaC1, will contain the radians of rotation in the Y axis.
- rotPuertaC2, will contain the radians of rotation in the Y axis.

- rotPizzas, will contain the radians of rotation on the Y axis.
- rotMicroondas, will contain the radians of rotation on the Y axis.
- rotMesa, will contain the radians of rotation on the Y axis.
- tiempo, needed for animation by Oil Shader.
- speed, needed for animation by Oil Shader.

Boolean, they are in charge of activating or deactivating their respective animations:

- sillaAnim
- puertaAnim
- puertaC1Anim
- puertaC2Anim
- pizzasAnim
- canastaAnim
- mesaAnim
- microondasAnim
- hornoPizzAnim

The DeltaTime is necessary, to homogenize the speed of the movements.

- **main()**

In the main function we have the initialization of GLFW, and the creation of a GLFWwindow object where we specify our window size and the name of the window. In the case that it cannot be opened, the message "Failed to create GLFW window" will be displayed and the execution will end.

If the creation of the window object is successful, the window will be defined as our current context and a frame buffer will be created. For GLFW it is necessary to define the interrupt functions, which are called KeyCallBack and MouseCallBack, which we will explain later. To have a more comfortable handling of the camera, when creating the window, the disabling of the cursor is specified as long as the movement of the mouse does not take us out of the created window.

In addition to GLFW, we need to enable GLEW to load OpenGL extensions dynamically. An important detail is that in our code we enable the experimental property of GLEW in order to be able to use pointers and extensions; if GLEW startup failed, a notification message is displayed, and execution terminates. After enabling GLEW and GLFW, we set the dimensions of our viewport to match the size of the window.

In our main function, we also have the declaration of our Shaders that will be used for lighting, drawing and animation of our objects, we have:

- lightingShader that makes use of lighting.vs and lighting.frag
- lampShader which makes use of lamp.vs and lamp.frag
- SkyBoxShader which makes use of SkyBox.vs and SkyBox.frag
- Anim that makes use of anim.vs and anim.frag

With our Shaders declared, the next step is to call the objects by means of variables of type Model, the following structure for the models was decided.

For the structure, we have the following models:

- Interior. It corresponds to the interior structure of the pizzeria, with its internal advertising and without a wall.
- Fachada. It corresponds to the exterior structure of the pizzeria, with its sidewalk and fences and with 4 walls, one of which is shared with the interior of the pizzeria. It is worth mentioning that the Facade structure “covers” the Interior structure when it is drawn.
- LogosFachada. Because these Logos are transparent, they are kept in separate models to be drawn with their alpha channel.
- BasesLamp. Contains the lamp bases inside the pizzeria.
- Lamps. It contains the glasses of the lamps inside the pizzeria, because its transparency is necessary, they were modeled separately.

We have another block corresponding to the Doors, which consists of:

- PuertaPrin. It is the main door of the Pizzeria.
- PuertaC1. It is the interior door that leads to the dining room.

- PuertaC2. It is the interior door that leads to the kitchen.

In the block corresponding to the dining room, we have:

- Silla. It is a single chair that was modelled separately due to its animation implementation.
- MesasYSillas. It is a set of small table and chair objects, because due to the repetition of the individual table and chair objects, it was decided to handle them as a single object.
- MesasYSillones. It is a set of objects of armchairs and long tables, because due to the repetition of the individual table and chair objects, it was decided to handle them as a single object.

In the block corresponding to the kitchen, we have:

- Freidora. It is the structure of the Fryer.
- Aceite. It is the plane that represents the Oil in the Fryer.
- Canasta. It is the basket of the Freidra.
- Parrilla. It is a Long Grill that goes in the corner of the kitchen.
- ParrillaChica. It is a Small Grill that goes between the Long Grill and the Deep Fryer.
- VitStructure. It is the metal structure of the pizza cabinet.
- VitGlass. It is the glass of the pizza showcase, it was modeled separately due to its drawing with transparency.
- VitPizzas. They are the 3 levels of Pizzas in the Showcase.
- MesaPreparación. It is a pizza preparation table, it is placed against the right wall in the kitchen.
- MesaConPuertas. It is a table model with a door, this table supports the Pizza Oven.
- PuertaMCP. It is the second door of the TableWithDoors, it was modeled separately to be animated.
- HornoStructure. It is the exterior structure of the Pizza Oven.
- HornoInside. It is the plate inside the Pizza Oven, it was modeled separately to be animated.

- CajaRegistradora. It is a cash register that goes in the same bar as the Pizza Showcases.
- MicroondasHorno. It is a Microwave Oven without a door.
- MicroondasPuerta. It’s the Microwave door.

Those are all the objects that we need to declare, after that, we have the skyBoxVertices vector that contains the vertices of the SkyBox.

To get started with drawing objects, we must first link the VAO and the VBO to ensure that the vertex data is stored in the VBO and the position and normal attributes are set in the VAO, in order to have access to the vertex data. during rendering. We also configure a VAO and a VBO with the data from the SkyBox and load their faces with the push_back function in the faces vector, in order to map them to the cube.

Just before starting the main cycle of the program, a projection matrix is generated using a perspective view.

When starting the main cycle, the DeltaTime of the current frame is calculated, since it is necessary to constantly calculate it for it to have correct effects. The glfwPollEvents(), DoMovement() and animation() interrupt functions are reviewed to review changes in the peripherals and perform the respective actions.

The colorbuffer is also cleared before each new frame is drawn and the depth buffer and depth test are enabled to store the distance information of objects from the camera correctly.

It is also necessary to configure the lights, for this the lightingShader is started providing it with the position information of the camera, so that the scenes are drawn correctly. In the code it was decided to configure a DirectionalLight with ambient and low diffuse components, the specular a little high. Its address faces the facade.

It was decided to have 6 PointLights, 1 of them slightly acclimating the kitchen, another represents the sun facing the façade and the rest simulate the entry of the sun through the windows of the interior room.

It was decided to have 4 SpotLights, 2 of them simulate the yellow bulb that keeps the pizzas hot in their display cases, the other 2 are the lighting provided by the ceiling lamps inside the kitchen.

The shininess was specified with the value 16.0f, this property affects the way the specular component of lighting is calculated on objects, so that value will not give much shine, but they will look good.

Subsequently, the view matrix of the camera is calculated, and the view and projection matrices are passed to the lightingShader using the corresponding uniform variables. A normalized “model” array is also declared and will be used to apply model transformations to the objects. Before model loading, it is necessary to update the view matrix with respect to the position and orientation of the camera.

Using the lightingShader, all the objects are drawn, except the Oil, since that is drawn with the Anim Shader. It is necessary to specify that transparency is activated for LogosFachada; and that for Lamps and VitGlass it is drawn with colorAlpha, to generate the transparencies, to finish the use of lightingShader glBindVertexArray is used.

To finish with a Frame, the SkyBox is drawn using the SkyBoxShader where the cubic mapping of the textures is used for its drawing and the view is modified to achieve an unattainable depth, therefore it is necessary to draw it at the end. For the frame change, a swap of the buffers is done.

Given the case that the drawing cycle is terminated, that is, by closing the window, then all vertex arrays and buffers are cleared, GLFW is terminated with the glfwTerminate() function, and 0 is returned. All the above is with respect to the function main.

Animation and Camera Movements

The functions that are called by main() before drawing a frame and that handle mouse and keyboard interrupts to have effects on animations and/or camera movement are as follows.

- **DoMovement()**

This function allows movement in the X and Z axes of the camera using the keys:

- W or Up, to go in -Z
- S or Down, to go in +Z
- To the Left, to go to +X
- D or Right, to go in -Z

It makes use of the “camera.h” library to access the ProcessKeyBoard function of the object declared camera, in order to perform said functions on the position of the camera and it is normalized with the deltaTime.

- **animacion()**

This function reads the values of the boolean variables that activate or deactivate an animation, depending on the following cases.

puertaAnim. If on, causes the Front Door to rotate by +0.05 radians in the Y axis, up to 90°; if it is deactivated it rotates in the opposite direction until it reaches 0° (the original position of the door).

PuertaC1Anim. If on, causes Gate C1 to rotate by +0.05 radians in the Y axis, up to 90°; if it is deactivated it rotates in the opposite direction until it reaches 0° (the original position of the door).

PuertaC2Anim. If on, causes Gate C2 to rotate by +0.05 radians in the Y axis, up to 90°; if it is deactivated it rotates in the opposite direction until it reaches 0° (the original position of the door).

sillaAnim. If it is active, it causes a displacement on the X axis of +0.1 until it reaches 0.8f, to simulate that someone is going to sit down, but if it is deactivated, the displacement is given inversely until it reaches its original position.

pizzasAnim. If it is active, it causes a 0.3f rotation on the Pizzas in the Showcase about the Y axis, when it is not active, it does not rotate.

aceiteAnim. If active it sets a speed to 25.0f which will affect its animation per Shader specified in the Anim shader, when not active the speed is 0.

canastaAnim. If it is active, it causes a displacement on the Y axis of +0.025 until it reaches 0.5f, to simulate that the oil basket has been raised, but if it is deactivated, the displacement is given inversely until it reaches its original position.

mesaAnim. If active, causes a rotation of -0.5f about the Y axis until it reaches -90°, in the MCPDoor model to simulate a table door being opened, when inactive it returns to its original position.

microondasAnim. If it is active, it causes a rotation of -0.5f about the Y axis until it reaches -90°, in the MicrowaveDoor model, to simulate that the microwave door is opened, when inactive it returns to its original position.

hornoPizzaAnim. If it is active, it causes a rotation of -0.5f about the Y axis until it reaches -90°, in the MicrowaveDoor model, to simulate that the microwave door is opened, when inactive it returns to its original position.

- **KeyCallBack()**

This function reads the keys in order to do the following:

Space or ESC, exits the window.

Key 1. Activates or deactivates puertaAnim.

Key 2. Activates or deactivates puertaC1Anim.

Key 3. Activates or deactivates puertaC2Anim.

Key 4. Activates or deactivates sillaAnim.

Key 5. Activates or deactivates pizzasAnim.

Key 6. Enables or disables aceiteAnim.

Key 7. Activates or deactivates canastaAnim.

Key 8. Activates or deactivates tableAnim.

Key 9. Activates or deactivates microwaveAnim.

Key P. Activates or deactivates hornoPizzaAnim.

The effects of the numeric keys are related to the animacion() functions

- MouseCallBack()

This interrupt function is responsible for capturing the mouse movement events in the window, calculating the displacement and sending that displacement to the camera object in order to update its orientation.

Shaders Used

modelLoading

- modelLoading.vs takes care of computing the final position of the vertex in clip space based on its original position and the transformation matrices.

- modelLoading.frag displays a texture based on the fragment's texture coordinates and discards fragments whose alpha component is less than 0.1

lighting

- lighting.vs performs the necessary transformations on the vertices, calculates the final position of the vertex in clip space, and sends additional information such as normal, world space position, and coordinates.
- lighting.frag calculates the lighting for the current fragment, taking into account directional light, point lights, and spot lights. It also applies the color and texture of the material to the fragment and allows the activation of transparency according to the value of activateTransparency.

anim

- anim.vs This shader is used to animate the Fryer Oil, it applies a wave deformation along the Y axis to the vertices of the object. This deformation is performed as a function of time and the distance from the vertex to the origin. The texture coordinates remain unchanged.
- anim.frag This fragment shader samples a texture based on the fragment's texture coordinates and discards fragments with very low transparency. The result is the colour of the texture, or full transparency if the fragment has been discarded.

SkyBoxShader

- SkyBox.vs This vertex shader applies projection and view transformations to the vertex position and assigns the vertex coordinates to the TexCoords output variable. The transformed position is assigned to gl_Position to be used in the rendering pipeline.
- SkyBox.frag This fragment shader samples a texture cubemap using the texture coordinates of the fragment and assigns the sampled color as the final color of the fragment.

Other Libraries

- Shader.h

The Shader class encapsulates the logic for loading, compiling, linking, and using vertex and fragment shaders in OpenGL.

- Camera.h

This camera class encapsulates the logic to process user input and compute the matrices and vectors needed to drive the camera view in OpenGL. It can be used to implement the functionality of a camera in a 3D scene.

- Model.h

The model class contains the logic for loading models and their respective textures in OpenGL.

- Texture.h

The texture class is used for the representation of the skybox, since it allows the correct mapping in the textures of a cube.

Conclusion

With this project, I realized that there can be great complexity in the development of virtual environments, and that it takes a lot of time to develop, so it is very important to use version control and a development methodology as well as define the costs of production, sales, and profits to achieve profitability. It is also observed that the creation of these products requires a good hardware capacity to process the vertex meshes in agile times. The use of OpenGL and libraries such as GLFW and GLEW facilitate the implementation of 3D graphics and take advantage of hardware graphics processing capabilities in an abstraction layer. Having documentation ready while it is developed is important, since it will facilitate the task of programming for future changes.