Jaba Makharashvili

Solar Galaxy Simulator

The application is a simple simulator for the solar system. It features the sun rotating in the middle and two planets that rotate around the sun. The planets also have their own moons which rotate around them. The application starts off as paused and can be unpaused by clicking escape. Of course, it can be paused again by clicking escape. The game does not take input and neither do the objects move when in paused state. Wasd is used for basic movement and the mouse is used to move around. The game will start in third person camera inheriting the moon object. F1 and F2 are used to turn on/off the wireframe mode. 1 is used for switching to first person camera, 2 is used for switching to static camera, 3 is used for switching to top-down camera. P switches to third person view of inherited object, F switches to First person view of inherited object. The object can be rotated around using the mouse. That is all for operating the game.

I will start off by going over lighting, I have implemented both phong and per vertex lighting. The difference between them is quite small as all it takes it to move the calculations from the vertex shader to the pixel shader. The pixel shader will apply the resulting color for every pixel as the name suggests. The current lighting system consists of Ambient, Diffuse, and Specular lighting. My main source for this was the book by Frank Luna. Ambient lighting is the easiest to apply as it is usually a set small constant number that exists in the scene. Ambient lighting is replacement for light that gets spread through other objects as this calculation is quite complex and taxing a cheap solution is to use a constant value for all objects that will give the same effect. Much less accurate but much cheaper to use. The next lighting, I implemented was diffuse which is also quite simple. Diffuse lighting is essentially lighting from a certain direction. It is calculated by getting the dot product of the normal and the light direction. The result of the dot product tells the shader if the light is facing the object. If it is not, then of course the light is not applied. Last one to be implemented was the Specular lighting which is the more complex of the three. We calculate diffuse lighting based on the diffuse material of the object and the direction of the Eye vector in the world. By changing where the camera is looking the specular lighting changes as well. The result can be seen below.

A picture containing text, dark, light

Description automatically generated

Here is another screenshot if we change the location of the camera which changes the Specular lighting on the objects. This is from behind the objects and thus the texture is much darker only lit up by the ambient lighting. As mentioned, the texture color is multiplied by the light color and thus it is mixed and the output comes out as expected. All 3 light source outputs are summed up before being multiplied by the texture color.

A picture containing text

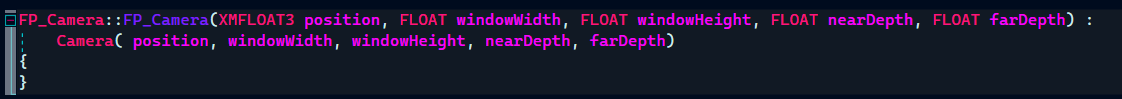
Description automatically generated

Next, the camera system. The application currently has 3 types of cameras. First person, third person, and a static one. Both first person camera and third person camera inherit from the static camera overriding its functions and adding a few new ones as well. This allows me to have a Camera class that is called main camera and I can easily switch between cameras by casting the camera to a camera class pointer and assigning it to the main camera object. I have different cameras set to 1-3 and by clicking a button I can switch between the different kind of cameras. I have a top-down camera which is just above the solar system and the look at vector is changed so it faces down the Y axis and the Up vector faces up the Z axis. For the first-person camera we handle rotation by altering the three axes. When we move the mouse vertically, we rotate the up and look vector by the right vector (called rotate pitch), and when we move the mouse vertically, we rotate all three basis vectors around the world Y-Axis. We use transform normal function instead of the standard normal function as this version is faster performance wise. Because we use only normal vectors, we can safely use the transform normal and get more optimization for the application. The code for rotating the camera can be found below.

Text

Description automatically generated

It is also important to note how the constructors are set up. The constructor passes in the values to the original Camera class constructor that it inherited from and in the camera class all member variables are initialized in the initializer list. This allows the inherited class to call the base constructor and just pass in all the values. This makes the code much cleaner and easier to maintain and avoids duplicating and writing the same code which is one of the code principles of OOP.



Text

Description automatically generated

The third person camera is slightly different as its variables are based on the object that it is attached to.

It has a different update function which takes in the position of the object that owns it and then calculates the look at vector by subtracting its own position from the position of the objects. This value is then normalized to get a normal vector pointing in the direction of the object. This way no matter where the object moves the third person camera will always be looking at the objects. After this we call the UpdateViewMatrix which in short calculates the right vector and up vector based on the look at vector. It first normalizes the look at vector then calculates the cross vector of look at vector and right vector which gives us a vector orthogonal to them and then we normalize that vector and we get the up vector this way. Then we get the cross product of the up and look at vector and get the Right vector. The right vector does not need to be normalized as the other two vectors are already normal and the resulting vector will be normal as well. After this we simply fill in the view matrix with the values of the three base vectors.

Text

Description automatically generated

Graphical user interface

Description automatically generated

Next is the player pawn class which inherits from the BaseObectOBJ. BaseObjectOBJ is a simple class that contaits the world matrix, Render function, and the mesh data. The constructor takes in the mesh data which can be loaded in from an OBJ file. It has dedicated functions for retrieving or changing the world matrix. It can be either retrieved as a 4x4 float or an actual matrix. Setting it up this way allows us to not expose the member variable which is considered a good practice to avoid accidentally changing the value. All member variables in the project have m\_VariableName same type of naming convention to have consistency.

Text

Description automatically generated

The player pawn constructor sets up a first-person camera, a third person camera, and all the other variables needed for the pawn. It provides two functions to switch between first person camera and third person camera. Graphical user interface, text

Description automatically generated

The first person camera’s position is set to the position of the object in the update function. We move the object and that moves the camera. There are four functions for changing the rotation and the position. We change them and then in the update function we call XMStoreFloat which applies all the rotations that we need. Text

Description automatically generated

The third person camera always sets its LookAt vector to point to the object. The z value of the position is set to z value of the player pawn minus the distance we want the camera to be away from the player. Text

Description automatically generated

The application also has a logging system in place which allows for easier debugging. The debugging uses spdlog library to setup a logger and then through macros it can be easily accessed to print out to a cmd window. In the initialize of the application there is specific code set so that we can have a cmd window open alongside the application. It also has different level of warning which changes the color of the message. The logger library is set to git sink which means that if the git repository is imported it will also setup the spdlog library for you.

Text

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

The application also has its own input system setup. Inputs are loaded as events in a queue and once resolved they are removed from the queue. The mouse input system also calculates the difference in Y and X between frames which is used for calculating how much the screen should rotate during first person camera or when rotating the player pawn. The input is received as a message from WndPrc function.

Text

Description automatically generated

We also can get if a key was pushed down by checking its 30th bit using the bit variable << Graphical user interface, text

Description automatically generated with medium confidence

The 30th bit is 0 or 1 based on if the key was pressed or not. The application also has a GameTime manager class which keeps count of the time and when clicked on escape the game pauses and as the rotation of all objects and movement of all objects is based on time everything stops including the input manager until we unpause the game.

I have also developed a base shader class which makes the code much tidier and cleaner as the shader can be initialized and shaders can be created through only one call. The vertex and pixel shader are setup separately to allow for flexibility in case they are written in different files or it was decided to mix different pixel/vertex shaders. Also multiple retrieve/set functions throughout the application are set as inline functions which means the complier will copy paste the code into the application. This is used when the function is very short and can save time/performance through simply replacing the function with the code.

Text

Description automatically generated

There is also a math library which provides a function for normal averaging so that user does not have to do it by hand. The function needs vertices, indices , and their count. It gets the indices and vertices of each triangle, computes the face normal for the triangle and then each vertex is added the normal value. After going through each triangle it keeps adding up the normals and we get an average of each triangle that shares each vertex.

Text

Description automatically generated

Feature list:

* Logging system
* Phong and per vertex lighting
* Diffuse, Specular, and ambient lighting
* Game timer class
* Custom input class
* Smart inheritance-based camera system which allows easy switching between the cameras
* Classes for setting up shaders
* Classes for setting up normal objects, obj objects, and a player pawn class which inherits from obj object
* DDS texture loading
* Manual object loading through manually provided vertices and normal
* OBJ loading
* Math library which also provides a function to calculate vertex normal average. Which removes the need to manually calculate normals. By providing vertices and indices it calculates the normal average.

Refelctive log:

Week 1:

Downloaded the framework and spent majority of the time familiarizing myself with the framework. It was quite daunting to see how much of a difference there is drawing

a single triangle in DirectX compared to OpenGL. There's around 500 lines of code just to draw a single triangle. Of course, using modern OpenGL is also a similar experience

but OpenGL still feels like it’s a lot more beginner friendly. I was not able to attend class due to my self-isolation so I had to do a lot of online researching to figure out

how some of the code worked but I have a good understanding of the code now.

Week 2:

Setup a small solar system with a yellow sun in the middle and two planets of different sizes rotating around the sun. The two planets have small white moons rotating around them.

I learned a lot and got a deeper understanding of how matrix transformations work and how changing the order of transformations effects the object. I had to setup different vertex

buffers for every object as they all had different colors and sizes and it was more practical to specify size in the vertex buffer rather than scale them in the update method. In the

future planning on setting up a 100 asteroid belt bonus task from today using instancing. After this week I decided to dedicate my project to this solar system theme. I will be making

either a sci-fi game or a simulator where I'll have a solar system running that will be as realistic as I can get it.

Week 3:

Did a lot of experimenting with the shader for the most of the session. Added a pyramid as well under the solar system for the time being. I also decided to make a diary in the actual code as

a txt file as GitHub will save it and it's good to fill out every week after work. I also will be updating my twitter with weekly updates about how the project is progressing. Shaders are

not easy but they're not as hard as I thought either. I will be spending the week trying to learn more about the shaders.

Week 4:

Started working on normals. I am trying to implement vector normal avaraging according to the book introduction to DirectX. Current output is just a blank screen as I have not been able to finish

setting up normals and thus I will try my best to finish it before the next tutorial. I will have to implement a mesh class that will contain all the indices, vertices, triangle count, vertex count,

etc. I also might have to implement my own vector class for math purposes, but it depends on how it will work once I have implemented the mesh class. Since my current output is a blank screen heres my

tweet on my account which shows my last working output. https://twitter.com/JMakharashvili/status/1453736737876758531?s=20

Week 5:

Implemented normal avaraging and tried to merge vertex and index buffer into one method. Currently the engine compiles but it does not draw properly.

Judging from the output there’s a problem with either the index buffer or vertex buffer and I will be trying to get this fixed before next week. Otherwise

Lighting has been implemented properly and just need to fix the bug. The normal avaraging is in Math3D.h and takes in an index buffer, a vector buffer,

and the triangle count.

Week 6:

I have finally been able to resolve the normal issue which was changing WORD buffer to UINT buffer. The issue has been haunting me for 3 weeks now but I have finally fixed it

and properly updated the lighting model to work well. Next goal will be to catch up to all the tutorials over the weekend. Update 2: I have finally added a BaseObject class which

makes the code a lot easier to edit and use. If you need to render and object its one Render() call and you make an object in one call by passing all the information it needs to setup

the index and vertex buffers and normals. The base object is very flexible and can take any real object. The code is much smaller now which is always a good thing. Once I make game

objects they will inherit from the BaseObject as it will only contain what every game object needs. Next goal is to catch up to the tutorials.

Week 7:

Caught up to the tutorials. Added a camera class and obj loading. obj loading looks fine and the camera class works as intended. Also added a base object class for obj based objects.

goal for next week is to implement proper obj files for the planets and make the game look more like a solar system.

Week 8:

Added GameObjectOBJ which allows me to instantly setup an object with one call of the constructor and this saves a lot of space and is much easier to read. I also have setup the planets

to rotate around a sun object and it looks a lot more like a solar system. I have majority of the week trying to setup the first-person camera. I'm currently working on the mouse input

needed to get the angle for camera rotation.

Week 9:

I have setup my own input class and have found a way to translate mouse movement into an angle. After the help of Frank Lunas book I have made progress on the first-person camera and was

finally able to finish it this week. I spent the whole week on this so I’m quite proud of the result.

Week 10:

I have developed a logging system for the game. Using spd log a cmd window I can output the messages easily through a macro. I had to do a lot of research on how to open a cmd window along

with an application window but at the end I Was able to find the code for it. I also setup github to manually import the library along with my own git repository making setup much easier.

Week 11:

Majority of the week went into developing the camera system. I Was confused about how to switch between different cameras without writing crazy amount of code and finally realized I could cast

first person and third person camera to camera as they both inherit from it and then use a MainCamera object which passes all the information to constant buffer. this way I can switch between

the cameras through a single call.

Week 12:

Spent the week developing a playerpawn class which has a third person and a first-person Camera. The player pawn inherits from the baseobjectobj class and can switch between the two cameras.

I have trouble figuring out how to use the third person camera.

week 13:

Last week and finished up the whole project. Third person camera now works by setting the camera behind the object and making sure the look at vector is always pointing at the player pawn.

The first-person camera position is always set to where the player is we move the camera by moving the player this insures it’s impossible for the first-person camera and the pawn to ever

separate.