Graphics – Work for Part 2, Week 7

This week, you should update the Camera class to your framework and see if you can fly around the terrain that you implemented last week. As before, the way I suggest to do it here is just that – a suggestion. Feel free to implement it in any way you wish.

Updating the Camera Class

The DirectXFramework you have been using so far incorporates a very basic Camera class. A possible updated Camera.h and Camera.cpp have been provided for you in the file Week7_Files.zip. Replace the existing files in your Visual Studio solution with these new files.

There are two changes you now need to make to your code:

- In DirectXFramework.cpp, add the following line to the end of the Update method (after the call to _sceneGraph->Update):

  ```
  _camera->Update();
  ```

  This will ensure that the View Matrix is updated with any changes to the camera.

- In DirextXApp::CreateSceneGraph, set the initial position of the camera, e.g.

  ```
  GetCamera()->SetCameraPosition(0.0f, 1000.0f, -1000.0f);
  ```

Flying a camera around the terrain

Now we can try flying the camera around the terrain. This tutorial uses a keyboard, but if you want to use a game controller, I have put a class that does this in the files for this week (GamePadController. I will leave looking at this as a task for you. You will need to make changes to the class so that you can perform actions when game pad controls are used, but the basic logic is included.

Regardless of how you handle input, I put the code to do this in DirectXApp::UpdateSceneGraph.

To read the state of a key, we could put in message handlers for WM_KEYDOWN, etc, but that will not allow us to determine the state of a particular key at a particular point in time. Instead, we can make a call to the GetAsyncKeyState function. This is defined as :

```
short GetAsyncKeyState(int virtualKey);
```

where virtualKey is the virtual key code of the key we want to test. A full list of virtual key codes can be found at: https://msdn.microsoft.com/en-us/library/windows/desktop/dd375731(v=vs.85).aspx

If the function succeeds, the return value specifies whether the key was pressed since the last call to GetAsyncKeyState, and whether the key is currently up or down. If the most significant bit is set, the key is down, and if the least significant bit is set, the key was pressed after the previous call to GetAsyncKeyState.

So if we just want to test if a particular key is pressed, we can test for a negative result from the function (since a short is a signed value). For example:

```
if (GetAsyncKeyState(VK_UP) < 0)
{
	GetCamera()->SetForwardBack(1);
}
```

This will move the camera forward by one if the cursor up key is pressed.

Of course, you can use whatever keys you want. You will want to adjust the pitch of the camera for it to go up or down and the yaw of the camera for it to turn left or right.

Right now, as you fly around the terrain, you will see that you can fly into it, i.e. fly underground. In most cases, this would not be desirable, so the first thing we will look at when looking at collision detection is how to know if we are colliding with the ground. To do this, we need to know the height of the terrain at any value of X and Z. We will look at this after we have looked more at how to texture the terrain.

Following an Object

If you want your camera to follow an object, that is more involved. I will leave the detail of implementing this to you, but the general process is as follows:

- Create a new node class for a moveable object. This should implement similar methods to the Camera class above (SetPitch, MoveForwardBack, etc). When it is updated, it should calculate its world transformation matrix in a similar way to the mechanism used to update the view matrix in the Camera class.

  Note. You could inherit from one of your existing node classes to create the new moveable node, but I found it useful to inherit from SceneGraph. Then I could just make any nodes children of the moveable node and provide movement functionality to different nodes as a result.

- Add methods to your new node class to retrieve the forward, right and up vectors for the node and the position of the node.

- In your Camera class, add a new method that specifies a node to follow and the offset from that node's position to use for the Camera's position.

- In the Camera's Update method, instead of calculating forward, right and up vectors for the camera, retrieve the vectors from the node object that the camera is following as well as its position and then use those to calculate the camera position using the node's position and offset from the node. Then you can use this camera position and the vectors retrieved from the node object to calculate the view matrix.