Computer Graphics

Assignment 3 – 3D maze exploration

Familiarize yourself with the 3D lab exercises on Viewing and 3D transformations, projection and lighting. You can of course use any programming session recording and any 3D base programs and or helper classes that have been added to the course website. Understand how these work and add them as needed into your own code or base your project on them.

The 3D scene

The program allows the user to “walk” around in a 3D scene. The scene should have walls and/or boxes that the user can walk in between. They can fence off the area or be set up as a maze or in any other way. The user should not be able to walk or see into or through these wall objects. The scene should also have at least one other object that has some kind of motion. It can rotate in place or move between points, just that it happens at a speed independant from the frame rate (use deltaTime). The user should not be able to walk into or through these objects either (at least one type of them – you can then add different types and do whatever you want with them). It would be preferred to try colliding with a circular perimeter around these objects, but with a square perimeter on the boxes/walls.

It's sufficient to always walk on a level floor/ground.

Collision detection and response

Collision detection in this project will be much simpler than in the 2D game. There's no need to respond by bouncing or changing any motion vectors. Just check distance from the camera's eye point. You can do it on x and z coordinates independantly if colliding with walls or boxes parallel to the axis and you can do it with a simple distance function if using a circular collision area. Nothing more complex than this is expected. The response only needs to move the camera/player's position to the edge of the boundary. If done correctly the player should slide smoothly along the objects rather than stop abruptly.

Drawing 3D objects

The extra objects can be pretty much anything you want. They don‘t need to by anything more than an instance of one of the base shapes, transformed in some way or combined to form a different object. Really just a sphere or diamond is perfectly sufficient but adding a little something extra by hand can be interesting and cool too. They can also be models loaded with 3rd party file loaders or made by yourself by hand, in code or in a file but we‘ll focus on mesh loading and rendering in a later assignment so it‘s not necessary to spend much time on this now. The main thing is to place the objects in the scene and have control over them.

Lighting (not the main focus in this assignment)

The main thing here is to make sure all the normals on the objects are correct. There will be a parallel assignment focusing on the shaders, where you will do the lighting calculations. To begin with, use a very simple shader that you‘re sure works, like the one given in the base project. Make sure all your objects look right using this shader, then add more interesting lights. This project does not take lighting into account, only that all objects are defined correctly with good normals.

Extra points!

To make the program more interesting you can add anything you think of. One way to get good marks is to make sure everything in the base project is cleanly implemented, smooth and glitch-free. Here are some ideas of interesting things to implement for other extra points.

Viewing (extra points)

The program could offer the user different ways to look at the scene, for example top down, third person chase and first person view. The whole drawing area could be rendered using either first person or third person chase views and the user could be able to switch between the two by pressing the button 'v'. The top down view might then always be visible in a smaller area in the top right corner of the viewing area.
To implement this you will need to keep track of where the user is and how he's oriented. This can be done using an instance of the Camera class but that instance of the Camera class can only be used to display the First Person view. When displaying the other views you can access the data from the First Person instance and set up a new position orientation using the look operation. You can do all this in your update code and then call setShaderMatrices for the camera you're using each time in the display code.
When drawing into several viewports you have to draw the entire scene again, the only difference being the Viewport you set up in the beginning and the camera you use to set the ModelView matrix. A good way would be to draw the main view first into the entire window/screen. Then you set up a smaller viewport in the corner which covers part of the first viewport. In order to overwrite that area you will have to call glClear(GL_DEPTH_BUFFER_BIT). Don't clear the COLOR_BUFFER because then you will clear everything you've drawn already. Make sure you start by drawing something that covers the entire viewport to get a background (or make sure the colors well defined enough to look good as an overlay). That will either have to be further away than the rest of the scene or you can do glDisable(GL_DEPTH_TEST), then draw the background, then call glEnable(GL_DEPTH_TEST) again.

The first person view
Here you will have an instance of the Camera class, let's call it camFirstPerson. You can do incremental changes on this camera and use this camera as the main source for the user's position. Make sure you keep doing the incremental changes here even when not using this camera for viewing. In the display code simply call camFirstPerson.setShaderMatrices.

The top down view
This view can use the eye point from camFirstPerson as a center (look at) point and it's own eye point will be straight above that. Add as much as you want to the y-coordinates, depending on how much area you want to see and how narrow/wide the lens should be. The up vector can just always be the same, parallell to the z-axis, very likely (0,0,1) or (0,0,-1).
In the display code you have to call Camera.look and camera.setShaderMatrices in order to set it up again, as this camera will not have any incremental changes, but instead follow the incremental changes of camFirstPerson.

The 3rd person chase view
This is where it get's interesting. Here you will also use the eye point from the camFirstPerson as the center point. The eye point for this camera can be calculated in different ways, depending on what outcome you want.
One way is to always move it back along the camFirstPerson's n-vector and up be some amount. This will make the camera feel like it's stuck on a pole to the “main character”. The camera will spin

around when you rotate and it will move when you move.
Another way is to let it follow motion rather than orientation. Keep track of where the main view was moving and add that vector to the amount you're dragging behind by. Then normalize the drag vector to some length. This will feel like the camera is being pulled along by an elastic band.
This looks the most natural but will take a bit of experimentation to get just right. When displaying the 3rd person chase view you will also have to draw a "character" where the first person view would otherwise be. Use the camFirstPerson's eye point as a center point to draw an object that represents the main character. It must also be oriented in the direction the first person view is looking. You can make some graphics class that draws a directional object (we might make an arrow or similar in class, and you can use that) centered in (0,0,0) and with a fixed orientation and size. You can then use this as the main character, but make sure to set it's material and translate, rotate and scale it. Can you use camFirstPerson's local coordinate frame and ModelMatrix.addTransformation to do this without having to backtrack rotations?

Make a bit of a game out of it (extra points)

Can those moving objects somehow interact with the player. Can he eliminate them somehow or can they eliminate him? Add a timer to get things done more quickly. Add extra levels, and/or read your levels from a file, or generate them randomly.

Grading

Grading is not an exact science here, but you can keep the following guidelines in mind.
A bare minimum implementation that still does everything in the base description will give a grade of around 8. That means you have some boxes and they generally stop the player from moving. There might be some glitches in special cases but in general, when you hit the side of a box you slide along it and don't see into it. Serious glitches and weirdness happening in the program can still affect the gradenegatively, just as clean and good implementations can have a positive effect.
If you make sure your implementation is glitch free and that everything feels smooth, this will be closer to 9 and very little needed to make that a 10. Just an interesting scene, the level either read from a file or built nicely, some beginning and end or such. Maybe a bit more maneuverability of the camera, while still just keeping it on the level floor. So basically an implementation of just the base description can give full marks (10) if it's smooth, glitch free and just a bit more than absolutely the bare minimum objects.
If you have your base implementation for an 8 you can instead implement the changable point of view and map view, as described above and that will be a 10 if implemented so that it works, even if the collision is still a bit glitchy at times.
If you instead implement a little game logic or interaction with the level or the objects, that can also take your grade to full marks. There's no one right thing to do and if the program looks and feels good and smooth, and implements all the base requirements it will probably get full marks.

Implementing something like boxes that are not parallell to the axis (diagonal collisions), especially if it takes you off the floor and into full 3D is above and beyond the requirements here, for those who want to try for extra points above the full marks. Also any of the other additions added to an already glitch free and smooth program will probably take you there.

Anything you think of and isn‘t mentioned will still probably impress and benefit your grade. Make sure you mention what you‘ve done and how it can be experienced, in order for it to be considered in the grade.

To summarize, the full base implementation can give 8-10 depending on how bare minimum, smooth and/or glitchy it is. Adding on to that will always work in your favor.