Assignment 3 – 3D program with selected camera views

Familiarize yourself with the first 3D lab exercise on Camera and Lighting and 3D transformations. An updated Camera class and more helpful actions can be found in Assignment 3 Code Base under "other material". Understand how these work and add them as needed into your own code.

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 type of object (and more than one instance of that type) 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 ad 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.

Drawing 3D objects

The extra objects can be model files loaded with libGDX's classes. They can be model files loaded with 3rd party file loaders or they can be made by yourself by hand, in code or in a file. The main thing is to place them in the scene and have control over them.

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 be 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.

Viewing

The program should offer the user three different ways to look at the scene, top down, third person chase and first person view. The whole drawing are should be rendered using either first person or third person chase views and the user should be able to switch between the two by pressing the button 'v'. The top down view should always be visible in a smaller area in the top right corner of the viewing area.

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 lookAt operation. You can do all this in your update code and then call setModelViewMatrix or setMatrices 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. 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.setMatrices (or setModelViewMatrix).

The top down view
This view can use the eye point from camFirstPerson as a lookAt 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.lookAt and camera.setMatrices 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 lookAt 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.
The base code includes an Arrow class that draws an arrow object centered in (0,0,0) and with a fixed orientation and size. You can use this as the main character if you want, but make sure to set it's material and translate, rotate and scale it. ***Can you use camFirstPerson's local coordinate frame and glMultMatrix to do this without having to backtrack rotations?*** *glMultMatrix takes the same type of variable as glLoadMatrix but adds the transformation rather than overwrite it.*

Lighting
The main thing here is to make sure all the normals are correct. Then it's enough to have maybe two static lights that light up the scene enough to get the 3D feel and so that there's some lighting on all sides. It's better if it's not too evenly lit, so that you can see that different sides of objects are lighting up differently. Otherwise don't spend time on this. We'll cover it later.