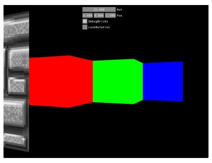
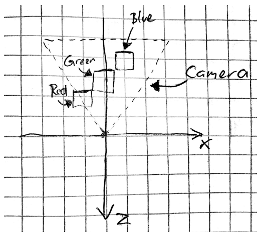
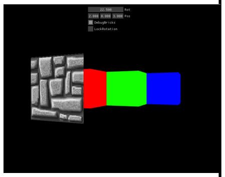
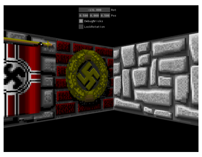
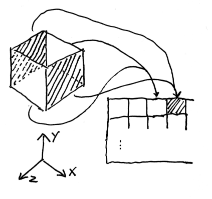
Game Programming: Exercise 6: 3D rendering and scene graphs

Learning objectives	Learning objectives ● Use the view transform to position the camera in world coordinates ● Create a simple FPS controller which moves the camera using WASD and mouse input ● Create a simple 3D model with texture mapping ● Parse a level defined in a JSON file using RapidJSON. **Handing in**: Create a zip-version of source-files, header-files and resources (CMakeFileLists.txt, .json and png). We will build your project using CMake, so make sure it works before hand-in (Note: If you make sure to keep all files in the same directory it should work without you needing to change the CMakeFileLists.txt). Do not submit SimpleRenderEngineProject files. Changes to SimpleRenderEngineProject are not allowed.
4-1 	**First Person Controller Position and rotation** The goal here is to be able to move the camera by modifying the view transform. The initial scene looks like this:  ● FirstPersonController::update() should update the camera view transform based on the position and rotation variable. You can set the view transform using either camera->lookAt(…) or camera->setViewTransform(…). ● Verify that your implementation works, by changing the values in the GUI interface. o The position is in world space coordinates. Changing the x,z values should move the camera along the floor. Changing the y-value should move the camera up/down. o Changing the rotation should rotate the camera around its current position. Positive rotations should result in counter clockwise rotation around the y axis.

4-2 	**First Person Controller Movement** ● When one of A,W,S,D-keys is held down the movement should happen relative to the current direction of the camera. The movement must be performed in FirstPersonController::update() and must use delta time (to make sure that the movement is independent of movement speed). o Hint: You need to distinguish between key down and key up events. ● Mouse movement events should change the rotation when the relative x is changed. (See https://wiki.libsdl.org/SDL_MouseMotionEvent) ● Enable “mouse lock” by uncommenting the lines in the Wolf3D constructor
4-3 	**Create procedural wall with texture mapping** The structure of the input texture: The input texture contains tiles of size 64x64 separated with s 1 pixel outline. The wall textures are indexed from upper left corner in the following order: [0][0d] [1][1d] [2][2d] [3][3d] [4][4d] [5][5d] [6][6d] [7][7d] [8][8d] [9][9d] … Where each tile exists in two versions: normal and dark. Normal textures should be used for faces in the xy-plane and dark tiles should be used for faces in the yz-plane.  Modify the member function Wolf3D::addCube(). Currently it draws one quad (using two triangles) between -0.5 and 0.5 in the xy-plane (with z = 0.5). ● First create a cube (with a side length of 0.5) instead of just a single side. Note that the number of texture coordinates has to match the number of vertex positions. You don’t have to create the top and bottom faces. Move around the cube to verify it looks correct from all sides. ● Take the parameter x,z into account and make sure that the cube is centered at (x,0.0,z). Look around – you should now be inside a small room with two wall segments on each side. ● Change the texture mapping, such that the correct tile as described above (use the parameter type and make sure that the faces in the yz-plane are dark).
4-4 	**Parse JSON** ● Implement WorldMap::loadMap() such that the data of the world map object is loaded from the JSON file (instead of being hardcoded)
4-5	**Create floor and ceiling (Optional)** ● Based on floor and ceiling color (defined in the JSON-file and WorldMap) create one large quad for ceiling and another one for floor.