Praktikum: Echtzeit Computergrafik

Assignment 7 – *More Meshes!*

10 Points

In this assignment, we will load, place and render multiple meshes. First, we will add guns to the cockpit; then we will add buildings to the terrain. All these objects, including their positions and orientations, will be specified in the config file.

Resources (2P)

First, we need to add the new resources to our ResourceGenerator.

- Find the required resources (.obj and .png files) for the guns and for two other objects of your choice (e.g. buildings or trees) in `external/art` (Hint: The guns consist of 2 .obj files each). For each resource, add the appropriate line to the NMake script of the project `ResourceGenerator`, similarly to Assignment 6.
    - Hint: You can use the tool MeshLab, which is installed on the lab PCs, to look at meshes in the .obj format.
      Build `ResourceGenerator`, and verify the existence of the output files (.t3d and .dds) in your `resources` folders.
- Now add one "Mesh" line to your `game.cfg` for each mesh, like you did for the cockpit mesh in Assignment 6. If a texture does not exist (e.g. not all objects have glow textures), specify "-" (a single "minus" symbol) instead of a filename. Note: The normal maps will not be used for now.
- Extend `Mesh::create` so that it does not try to load the missing textures if the filename equals "-".

Background Info: If you skip the `Load*` and `device->Create*` calls, the Texture and Shader Resource View will be `nullptr`. Using a `nullptr` SRV (in `g_*EV->SetResource()`) is allowed; any accesses to it from the shader will return black (i.e. `float4(0, 0, 0, 0)`). This is exactly what we want if a glow or specular texture is missing, so no adjustments to the rendering code are necessary.

- Extend your config parser and your `InitApp()` function in `game.cpp` so that multiple "Mesh" lines can be read. Create a `new Mesh` for each mesh name read by the config parser, and store all the resulting mesh objects in a global dictionary of meshes (e.g. `std::map<std::string, Mesh*> g_Meshes`) so you can easily access them by their identifier (as stored in the config!).
- This is a good time to check (using e.g. the Visual Studio Debugger) that everything you did so far works as expected, i.e. that `g_Meshes` contains all the meshes specified in the config.
- Call `create()` for each `Mesh` object in `OnD3D11CreateDevice`, `destroy()` in `OnD3D11DestroyDevice`, and `SAFE_DELETE` all the meshes in `DeinitApp()`. Make sure your program produces no memory leaks on the CPU or the GPU.

- The `g_CockpitMesh` variable from Assignment 6 is not needed anymore now and should be removed.
    - Hint: The rendering code in `OnD3D11FrameRender`, which still uses this variable, can be commented out for now so that you can run your program again. (We will update this code lateron.)

Cockpit Objects (3P)

- In Assignment 6, you specified the required transformations for the cockpit mesh directly in the C++ code. Now, we will specify them in the config file instead. For example:

```
CockpitObject Cockpit 0.05 0 180 0 0 -16 42
```

  This line instantiates one "CockpitObject" (i.e. an object which is attached to the camera), which uses the mesh with the name "Cockpit". The following 7 numbers specify the scale (in all dimensions), rotation around the x axis, around the y axis, around the z axis, and translation in x, y and z direction.
  Add such a line to your `game.cfg` for the cockpit (if you used a different order of transformations in assignment 6, you will have to adjust the numbers accordingly), and add further lines to place each of the gun meshes. Don't worry about the correct transformations for the guns right now; we will adjust them later.
- Create a type (a `struct` or a `class`) to represent such an object definition, and extend your config parser to read in the objects. Store all the objects in a global `std::vector`. Check the results using the Debugger.
- Extend your rendering code in `OnD3D11FrameRender` so that it loops over all the objects. For each object, look up the corresponding mesh from `g_Meshes` and render it with the specified transformations. You should create one matrix for the scaling, one each for the rotations around the x, y and z axes, and one for the translation, and then multiply all the matrices together as done in the last assignment.
- Play around with the transform values for the guns until they are reasonably placed. Compare your results with the screenshots in `external\art\01-Cockpit\cockpit\screenshots`.
    - Hint: In MeshLab, select `Render->Show Axis` and `Render->Show Quoted Box` to show the orientation and size of an object in its local object coordinate system. This can help you find the necessary transformations.

Now, your result should look similar to the one from Assignment 6, but with ***MORE GUNS***!

Ground Objects (3P)

Now we want to place some "ground objects" on the terrain, i.e. objects which are attached to the terrain instead of the camera.

- Ground objects should be defined in the config file similarly to cockpit objects, e.g.:

```
GroundObject Barracks 1 0 0 0 100 10 -30
```

  The meaning of the parameters is the same as for cockpit objects.

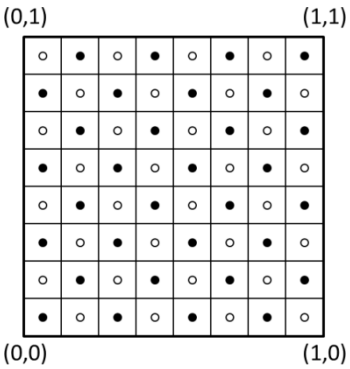
- Extend your config parser to parse these objects and store them.
    - Hint: You can either create a new type for ground objects, or extend the type you used for cockpit objects. Again, check with the Debugger that everything works as expected.
- Render all ground objects in `OnD3D11FrameRender` like you did for cockpit objects. The only difference is that ground objects do not move with the camera but have a fixed position in the world.
- For **each** of the objects you chose, add at least **two** instances with different positions and orientations to the config file (resulting in a total of at least four ground objects). Choose positions so that the objects are standing (roughly) on the terrain, and are visible from the original camera position.
    - Hint: Change your TerrainGenerator such that a fixed seed is used for the random engine. This way you will receive the same terrain each time you build it.

Finally, make sure once again that your program produces no memory leaks during a normal run, and that it works in both the Debug and Release configurations.

Questions (2P)

Create a section for assignment 7 in "`<username>/readme.txt`" and answer the following questions. You may also commit an image or pdf file containing the solution, but in this case include a reference to your solution file in the readme file.

- A 2D texture map consisting of white (white circles, value = *1*) and black (black circles, value = *0*) texture values is illustrated on the right. For this texture, compute the texture values at the texture coordinates $\left(\frac{1}{2},\frac{1}{2}\right)$ and $\left(\frac{4}{8}-\frac{1}{4}\cdot\frac{1}{8},\frac{4}{8}-\frac{1}{4}\cdot\frac{1}{8}\right)$ by bilinear interpolation. **(1P)**



- What color is seen when looking along a ray which first hits an object with color *(0.5, 0, 0, 0.5)*, then an object with color *(1, 1, 1, 0)*, and then an object with color *(0.25, 0, 0, 0.5)*? Assume a black background (color *(0 ,0 ,0 ,1)*). Here, the first three color components specify the Red, Green, and Blue components, respectively, and the fourth component specifies the opacity. Opacity ranges from *0* to *1*, where *0* indicates full transparency and *1* indicates full opacity. Use alpha-blending to compute the color. **(1P)**