Praktikum: Echtzeit Computergrafik

Assignment 4 – "*Interactive 3D Terrain Rendering*"

10 Points

Setup (1P)

- Copy your configuration parser class from the first assignment into "`<username>/GEDGame/projects/Game/src`" and add both .h and .cpp to the Game-Project.
- Edit the project dependencies of the "`Game`" project. It should depend on the projects "`Effects11`" and "`ResourceGenerator`". (While "`ResourceGenerator`" still depends on "`TerrainGenerator`").
- Since our template game project loads the texture "`debug_green.dds`" from your resource directory, it is necessary to convert the given JPEG image "`debug_green.jpg`" from "`external\textures`" to a DDS file in your resource directory. To do this simply add the line

  ```
  "$(OutDir)texconv" -o "$(OutDir)resources" -srgbi -f
  R8G8B8A8_UNORM_SRGB "..\..\..\..\external\textures\debug_green.jpg"
  ```

  to the NMake command line of your `ResourceGenerator`.
- Set the build configuration to "Release", set "`Game`" as start-up project and start it with or without debugger (F5 or Strg+F5). Visual Studio should now build your terrain generator tools, build your terrain resources, build the "`Effects11`" and "`Game`" projects and finally run the executable resulting from the "`Game`" project. If all of this works, the application should display a single rotating, lit and textured triangle.
- The framework project is now set up. Commit the changes to your repository.
- Make yourself familiar with the provided code of the game.cpp, GameEffect.h, Terrain.h and Terrain.cpp as well as the general structure of the program. Note the usage of the C++ keyword `extern` in GameEffect.h - you might want to integrate your configuration parser in the same way to make it accessible everywhere.
- In order to avoid regenerating all resources every time you rebuild your game, you may now unload the `ResourceGenerator` project from your solution. To do this right-click on this project in Visual Studio and choose "Unload Project".

Buffer Creation (7P)

A general hint: Whenever you encounter an unknown function, check the header files or MSDN for documentation (and remember your good old friend Google)! You can right-click on functions and select "Go to declaration" to instantly jump to the header file.

- Create a game.cfg in your solution directory containing the following information:

```
TerrainPath terrain_height.tiff terrain_color.dds terrain_normal.dds
TerrainWidth 800.0
TerrainDepth 800.0
TerrainHeight 200.0
```

Adapt your configuration parser accordingly.

- The content of the "game.cfg" should be parsed in the "InitApp()" function of the "game.cpp". We already prepared a call of "`DXUTFindDXSDKMediaFileCch()`" to find your game.cfg, check the documentation of this function!
- In "`Terrain::create`": Load the heightfield defined in your configuration file using `SimpleImage`. Comment out the code that creates a single triangle. Instead, create a vertex buffer for your terrain as described in the slides.
- In "`Terrain::create`": Create an index buffer for your terrain and fill it with the appropriate indices (use the Terrain member variable `indexBuffer`).
- In "`Terrain::create`": Use the "`DirectX::CreateDDSTextureFromFile()`" function for the "`device`" to load the terrain color texture from the DDS file into a shader resource view (`diffuseTextureSRV`).
- In "`Terrain::destroy`": Release the texture and the shader resource view.
- In "`Terrain::render`": Use the "`SetResource`" method of "`g_gameEffect.diffuseEV`" to bind the terrain diffuse texture to the effect variable.
- In "`Terrain::render`": Replace the "`Draw`" method with "`DrawIndexed`" of the passed ID3D11DeviceContext* to draw the terrain geometry as shared-vertex triangle list. Don't forget to bind the index buffer.

If the rendering is correct, you gain all 7 points. For every individual thing that is broken, one point is removed (e.g. wrong texture + terrain aligned vertically -> 5 points).

Questions (2P)

Create a section for assignment 4 in "`<username>/readme.txt`" and answer the following question:

- If the terrain geometry should be rendered using "`Draw()`" instead of "`DrawIndexed()`", what changes would be necessary to the vertex buffer? (1P)
- Given is a triangle by the vertices $A = (3, 0, 6)$; $B = (0, 2, 0)$; $C = (6, 0, 0)$. The following RGB-colors are assigned at each vertex: $c_A = (1, 0, 0)$; $c_B = (0, 1, 1)$; $c_C = (1, 0, 1)$. Calculate the color c_P at point $P = (2, 1, 2)$ by using barycentric interpolation. You can assume that P lies inside the given triangle.
  Hint: To calculate the area of a triangle XYZ you can use the formula $\frac{1}{2}\left\|\overrightarrow{XY} \times \overrightarrow{XZ}\right\|$. (1P)

General Grading Hints

Avoid producing memory leaks in CPU memory.

- Every "`new`" requires a "`delete`".

- In debug builds, a CPU memory leak leads to a “Detected memory leaks!” message in the “Output” log window. This message also contains the path to the source file and the line number of the respective allocation.

Avoid producing memory leaks in GPU memory.

- Every “`*Create*()`” requires a “`SAFE_RELEASE()`” (this works similar to `new` and `delete`!).
- Do not “`*Create*()`” any GPU resources outside these three functions or any functions called by them (unless you know EXACTLY what you’re doing).
- GPU memory leaks always result in a “non-zero reference count” error message box (which usually appears when the program terminates).
- To find GPU memory leaks, you must check for each and every “`*Create*()`” if the respective resource is also released. Hence try to not produce GPU memory leaks in the first place.

If your program exhibits memory leaks during a normal run, you will get a malus of **-1P** for the assignment.