Middle East Technical University | Department of Computer Engineering

CENG 469

Computer Graphics II

Spring 2021-2022
Assignment 3
Terrain Rendering
(v1.0)

Due date: June 12, 2022, Sunday, 23:59

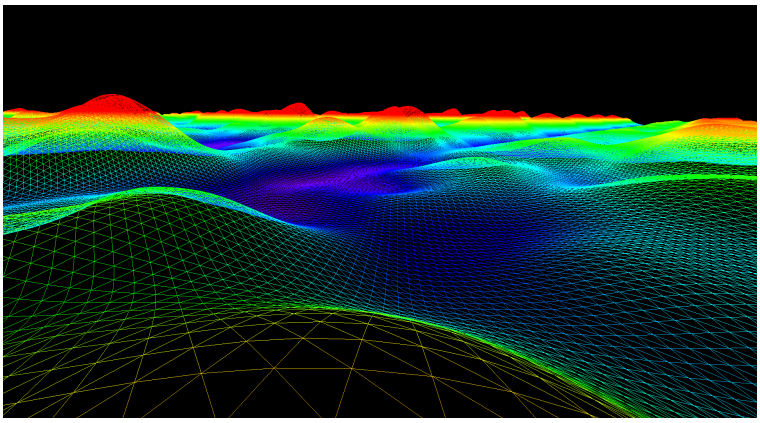

Figure 1: A sample rendering result for the assignment.

1 Objectives

In this assignment, you are going to implement Perlin noise and also work with the geometry shader of OpenGL to generate a terrain and travel on it via the car-like camera. The expected end product of this assignment is an OpenGL program which: *(i)* provides user interaction capabilities with keyboard and mouse buttons, and *(ii)* renders a scene which includes the items listed below:

- A generated terrain where its heights are calculated using Perlin noise, and colors are determined by the heights.
- The camera travels on the generated terrain as if it was a car. A sample frame where the camera is stopped on the peak of a mountain is shown in Figure 1.

The specifications are explained in Section 2. Note that you are free to implement your own design choices and extra features as long as you obey the requirements detailed in this assignment text. Design choices and extra features include but are not limited to:

- Adding a skybox to the scene, or integrating some mesh objects into the scene composition.
- The color map of the terrain, where the feeling of the height is still preserved.
- Applying textures/bump mapping to the terrain (but keeping the height feeling that would otherwise be kept via the change of colors).
- By composing the items above, you may generate your own imaginary planet and travel on it via the car (camera) by listening to the car's radio (music playing in a background app). A nice content for your blog post would be enabling the wireframe mode and listening to a Synthwave style music while driving on the magentaish-colored terrain, resulting in Outrun vibes.
- Some kind of acceleration/deceleration/high speed, or other small effects on camera for more immersive feeling of controlling a car. For instance, a small down and up movement on the camera when it comes to a full stop.

2 Specifications

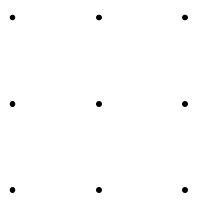

Figure 2: Generating a 2D 3x3 vertex grid in the vertex shader.

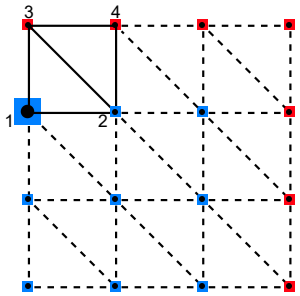

Figure 3: Generating a two-triangle grid cell in the geometry shader using one input vertex (bold blue), which is considered as the bottom-left corner of the cell. Blue vertices come from the vertex shader. This process is done for each of the blue vertices. Red vertices are additionally produced by the geometry shader. The resulting grid is composed of 3x3 cells.

- A 2D static uniform vertex grid should be generated/placed on the X-Z plane via the vertex shader. Static means the terrain is not moving together with the camera (as was previously discussed in the class), but stays still in its coordinate in the world. An illustration of this uniform grid is given in Figure 2, where the dots show the vertices.
- A geometry shader should be implemented which will then work on each of the vertices of the grid and produce the triangles of the terrain. An example illustration is shown in Figure 3.
- Perlin noise should be implemented which will then be queried for each vertex of the newly produced triangles using its world coordinates to set the height of that vertex before the geometry shader ends.
- The height information should be passed to the fragment shader, which uses this height information to look up and set the color of that fragment from the color map. Two color maps are shared with you in **hw3_support_files.zip** file.
- The camera should behave like a car that moves on the terrain and go in the direction of the Gaze vector when some speed input is provided. To have a car-like camera, the Gaze vector should remain parallel to the terrain underneath the camera.
- The program should have the following user controls:
  - **W and S buttons** should increase and decrease, respectively, the camera's movement speed. The camera can also go backwards, behaving like a car in reverse gear. It should also be fully stoppable using those buttons to, for example, be able to watch a scenery by stopping on top of a mountain. The speed should be 0 initially.
  - **Right Mouse Button:** Moving the mouse while pressing and holding the Right Mouse Button should make the car change its course in the desired direction: The camera's Gaze direction should be changed accordingly.
  - **E and Q buttons** should increase and decrease, respectively, the vertex count of each axis of the terrain grid. Initially, the terrain should be composed of 1000x1000 vertices distributed uniformly. This means the parameter's value is initially 1000. Each tap on the buttons should change this amount by 100, down to a minimum of 100x100. There is no upper value limit. In Figure 2, this parameter is set to 3.
  - **T and R buttons** should increase and decrease, respectively, the coordinate value that the terrain spans on the X-Z plane of the world by 1. Initially, the terrain should span from $x \in$ [-30, 30] and $z \in$ [-30, 30] coordinates uniformly, with the center of it corresponding to the origin. This means the parameter's initial value is 30. Tapping the buttons should change this amount by 1, down to the minimum of 1, where $x \in$ [-1, 1] and $z \in$ [-1, 1]. There is no upper limit. Note that decreasing/increasing this parameter without touching E and Q buttons results in visually higher/lower resolution terrain, as the same vertex count is distributed to a smaller/larger world area, respectively. The camera might go outside into the blackness of the space when the boundaries set by this parameter are reached. This is expected, not a problem.
  - **G and F buttons** should be used to increase and decrease, respectively, a scale parameter that the Perlin noise value is multiplied by before setting it as the height.
  - **L button** should toggle the wireframe mode.

- You are advised to use the **sampleGL.zip** file shared to you on ODTUClass course page as the template code and build your homework code on top of it.
- There is no frames-per-second (FPS) constraint in the homework but your animation should be seen as continuous and smooth. Please test your code on Inek machines before submitting, as the animations may slow down due to the performance capacity of the Inek machines.
- **Blog Post:** You should write a blog post that shows some output visuals of your work, and explains the difficulties you experienced, and interesting design choices you made during the implementation phase. You can also write about anything you want to showcase or discuss. Below we provide some discussion ideas:
  - A discussion on how you have placed and oriented the camera on the terrain.
  - Discussions on what kind of extra features you have implemented and how you have implemented them, if any.
  - An analysis of **FPS** (frames-per-second) of your code when the user-controlled parameters are changed.
  - An analysis of outputting a triangle strip composed of 2 triangles from the geometry shader versus outputting a triangle strip composed of one-row-of-the-terrain-grid many triangles.

  The deadline for the blog post is 3-days later than the deadline for the homework. You can submit your code before the deadline, and finalize the blog post during the 3-day late submission period without losing late days. You consume your late days only if you submit your "code" late. However, submitting the blog post more than three days later will incur grade deductions.

3 Regulations

1. **Programming Language:** C/C++. You also must use gcc/g++ for the compiler. Any C++ version can be used as long as the code works on Inek machines.
2. **Changing the Code Template:** We provide you a code template (sampleGL.zip on ODTUClass). You are free to edit, rename or delete any file from the given code template as long as you comply with the submission rules below.
3. **Additional Libraries:** GLM, GLEW, and GLFW are typical libraries that you will need (these are already included in the sampleGL source file). You should not need to use any other library except for reading images. But if you still want to use some other library, please first ask about it in the ODTUClass forum of the homework.
4. **Submission:** Submissions will be done via ODTUClass. You should submit your blog post link to "Blog Links" forum on ODTUClass, using the title "HW3 Blog Post". For code submission, create a "**tar.gz**" file named "hw3.tar.gz" that contains all your source code files, texture images and meshes (if any), and a Makefile. The executable should be named as "**main**" and should be able to be run using the following commands (any error in these steps may cause a grade deduction):

tar -xf hw3.tar.gz
make
./main

5. **Groups:** All assignments are to be done individually.

6. **Late Submission:** You can submit your codes up to 3 days late. Each late day will be deduced from the total 7 credits for the homeworks of the semester. However, if you fail to submit even after 3 days, you will get 0 regardless of how many late credits you may have left. If you submit late and still get zero, you cannot claim back your late days. You must e-mail the assistant if you want your submission not to be evaluated (and therefore preserve your late day credits).

7. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0 from the homework. You can discuss algorithmic choices, but sharing code between students is strictly forbidden. Please be aware that there are "very advanced tools" that detect if two codes are similar.

8. **Forum:** Any updates/corrections and discussions regarding the homework will be on ODTUClass. You should check it on a daily basis. You can ask your homework related questions on the forum of the homework on ODTUClass.

9. **Grading:** Your codes will be evaluated on Inek machines. We will not use automated grading, but evaluate your outputs visually. Implementing the following will get you 100 points from the homework part: A program that correctly implements the user-interaction capabilities and renders a smoothly-animated scene composition where there is a generated terrain whose heights are calculated using Perlin noise, and whose colors indicate the height information; and a camera that follows the terrain as if it was a car travelling on the terrain. Note that you should test your code on an Inek machine before submission, as the frame rate might not be as high as your home computer if you have a powerful PC, and might cause hangs during the animation. This might require reducing/optimizing per frame calculations in the code. Blog posts will be graded by focusing on the quality of the content.