Generation3D

March 22, 2024

1 Project: 3D generation

1.1 Abstract

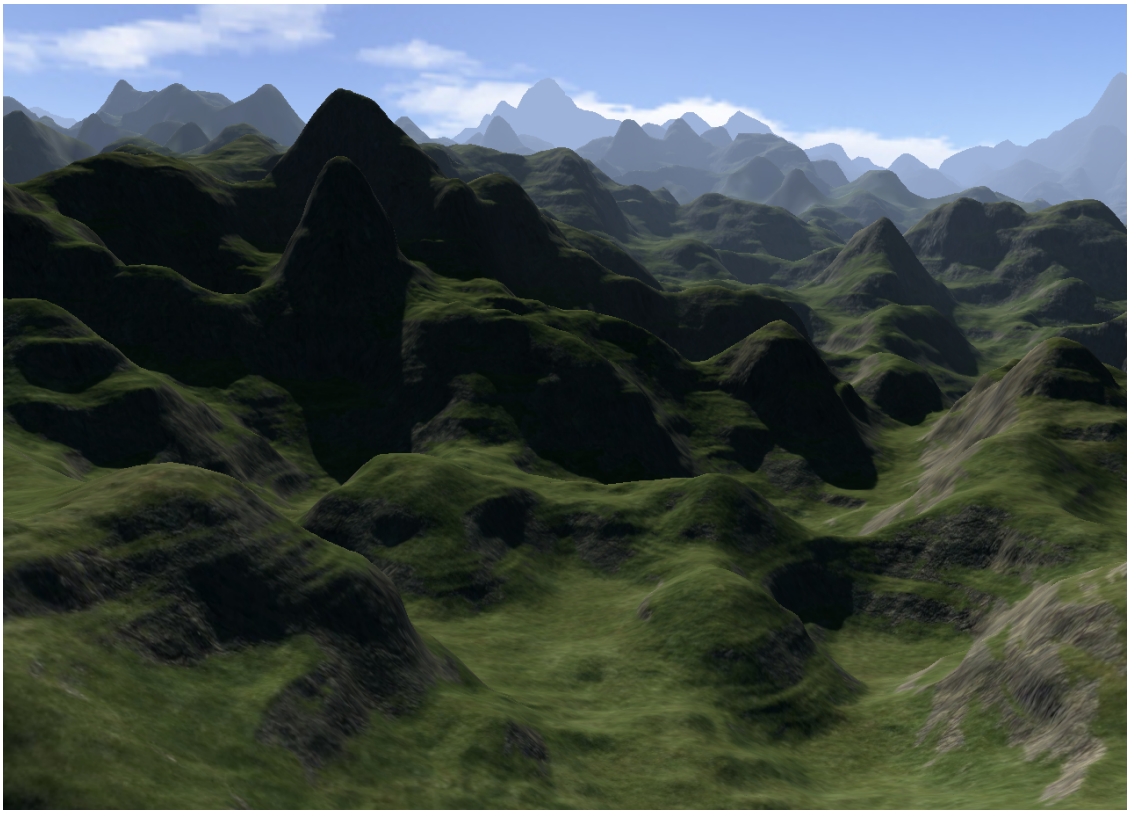
The purpose of this project is to address various issues regarding 3D generation techniques used in the video game industry. Moreover, to realize this project, the use of Unreal Engine, Unity, or other engines is forbidden. The only accepted graphical APIs are SFML and/or imgui and OpenGL. We want your work to be representative of what you might be asked to do in large industries and to understand the issues behind the magic tools.

In order to increase your knowledge of the video game industry and the problems that programmers have encountered in the history of this industry, the game you will make will be in 3D.

1.2 Project description

1.2.1 Required work

In this project, you will work with procedural generation and OpenGL. Your mission is to create a 3D viewer that will display a 3D terrain generated procedurally.



We are able to move the camera in fly mode and to visualize object with different settings. We need to have a viewer with UI to change The settings and apply them dynamically. We want to see the number of primitives and framerate.

1.2.2 Some improvements

Range filter With this filter, we want to be able to filter points outside the range to generate the terrain.

Median filter and/or Gaussian filter We want to be able to manage settings of these filters to apply them on the generated terrain.

Smooth gradient You could use other algorithms to smooth the gradinent to be more realistic.

Erosion with water drop To make the terrain more realistic, we suggest you to use an algorithm that will generate water drop on the map like the rain to simulate realistic erosion. You will find many articles about it on the internet.

Create rivers and lakes To make the landscape beautiful and alive, you could add some rivers and lakes with the according textures to make it realistic.

Performances You could use many algorithms to improve your performances to display objects faster, to compute faster, etc…

1.3 Technicals requirements

1.3.1 Language and APIs

To realize this video game, you will have to use exclusively C++. Of course you could use SFML or imgui. You will be able to rely on the latest standards of the C++20 language. If the standard library does not suit you, you can create your own container or others. You will use OpenGL only to manage rendering of 3D objects.

For SFML, it is a very well documented library and very easy to understand.

1.3.2 Architecture

You start from scratch, and you are in a group. Many people are going to touch the code, so within your team, you need to use a common language: the UML. Plan ahead, anticipate and rely on design patterns to set up extensible and maintainable architectures. This part is the most important part of the project and if it is well organized, doing the rest will be a breeze. This part will have to be presented at the presentation of your project. Do not hesitate to document its evolution and the reason for your choices.

1.3.3 Rendering

For resources (sprites), you can use free or your own sprites to embellish your game. Try to optimize the resources.

1.3.4 Memory

Unreal Engine has a strong point, that of managing your objects to destruction. Now, this is no longer possible, so you have to be careful to memory, and to optimize it. You can use APIs like OPTICK C++ to profile your game.

1.3.5 Versioning

For the versioning of your source code, you could use Git, SVN, Perforce or others CVS but ensure that your work flow is correctly managed. Try to have clean branches with correct commit names to work fluently with your colleagues. You could try to set a pull request mechanism to force you to review code elaborated by your colleagues.

1.3.6 Build system

You need to have a cross-platform build system to generate your solution on Mac, Windows or Linux. To generate your solution, you will use CMake. With CMake the build system will be independant from the target. When you will elaborate your build system, you will need to think how to manage your libraries or package them to have a clear vision of your architecture and an optimized compilation.