Implementation of a Real-time Streaming-based Terrain Level of Detail System

Degree programme: BSc in Computer Science
Thesis advisor: Prof. Marcus Hudritsch
Expert: Dr. Eric Dubuis

Rendering large landscapes is not only interesting from a visual standpoint, but also from a technical one. How do large landscapes get rendered efficiently with high frame rates? How does the underlying data get managed effectively without overloading the available memory? How does something like Google Earth work under the hood? In this thesis, a large-scale terrain renderer was developed that can render the entire Earth.

Introduction

Large-scale terrain rendering is an important task for various practical applications of computer graphics, such as video games, simulation systems and geographic information systems (GIS). It is also a difficult task due to the sheer size and the constant visibility of terrains, making the naive approach of rendering every point of the terrain impractical. For this reason, there exist several **Level of Detail (LOD)** algorithms, which remove detail from the terrain the further away it is from the camera. Besides the efficient rendering, the handling of terrain data is also a central aspect. Massive terrain datasets, such as the Earth, span multiple terabytes and cannot fit into the memory of today's personal computers. The terrain renderer must therefore support **streaming**, the concept of loading and offloading terrain data dependent on the camera movement.

Implementation

Technologies and Data

The system was implemented with C++17 and OpenGL 4. Numerous smaller helper libraries were used, such as Dear ImGui, GLM, STBI, libcurl, libwebp, and more, but the bulk of the system was written from the ground up. The terrain data, which includes satellite imagery for texturing and heightmaps, is served by web APIs from Maptiler. The data is served with the XYZ tiling scheme, which organizes the data in a hierarchical manner, and is available up to level 14.

Rendering

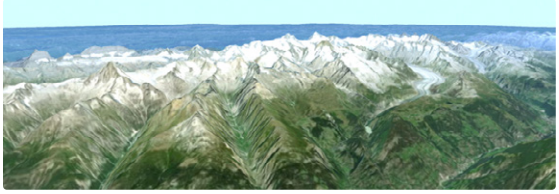
The LOD algorithm is based on Chunked LOD, which organizes the terrain into a quadtree, a tree where each node has four children. The actual rendering is done with heightmap-displacement in the vertex shader, which requires only a constant set of vertex and index buffers to be defined globally and allows the usage of the same mesh for every quadtree node. The vertex shader also perfroms the geo-projection to the WGS84-ellipsoid, giving the terrain its familiar elilpsoidal shape. Cracks between adjacent terrain meshes with different LOD levels are hidden by rendering a skirt around the mesh. View-frustum culling and horizon culling prevent loading and rendering areas which lie outside of the viewing area and beyond the horizon respectively.

Streaming and Caching

Streaming in terrain data involves reading data from the disk or fetching data from the web API, which introduces latency and results in hiccups. To circumvent this, a multithreaded architecture based on message-passing was used. The main thread sends out requests to worker threads, which then load terrain data in the background while the main thread continues rendering. After loading the data into memory, the workers send the data back to the main thread for rendering. Terrain data which has not been rendered in a while should be deallocated to make room for new data. For this, a least-recently used (LRU) cache is used to store terrain data in memory, which evicts unused terrain data as soon as new data comes in. A similar mechanism is used for the disk cache.

Results

The terrain rendering system performs well, yielding over 60 FPS on a 2020 MacBook Air. The system allows the user to browse the Earth seamlessly and its performance settings can be configured. The source code is published under the MIT licence. Potential improvements and future work include supporting multiple data providers with fallbacks and extending the system into an SDK.



The Swiss Alps in Valais.

Amar Tabakovic
078 784 89 87
tabakovicamar1@gmail.com