Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

3D Terrain with Level of Detail

Written report for the module
BTI3041 – Project 2
by

Amar Tabakovic

Bern University of Applied Sciences
Engineering and Information Technology
Computer Perception & Virtual Reality Lab

Supervisor
Prof. Marcus Hudritsch

November 8, 2023

Abstract

Rendering terrains is a central task for video games, geographic information systems and simulators, but also computationally expensive. Optimizations, one of which is the level of detail (LOD), are necessary in order to ensure appropriate performances. This project aims to give an overview of the topic of terrain rendering. As part of this project, a demo terrain renderer (ATLOD) was implemented in C++ and OpenGL. ATLOD currently supports naive brute-force rendering, GeoMipMapping, TODO.

Contents

Chapter 1

Introduction

In the field of 3D computer graphics, rendering is one of the central tasks. Many practical applications of 3D computer graphics make use of terrains, such as flight simulators, open-world video games, and Geographic Information Systems (GIS) [1, p. 185]. At the same time, rendering large and constantly visible objects, such as the terrain, is computationally expensive and optimizations are necessary in order to avoid performance deficiencies.

One area which offers potential for optimizations is the *level of detail (LOD)* of objects. The concept of LOD is based on the idea that the farther away an object is, the fewer details are going to be visible to the human eye.

The problem of rendering terrains spawned numerous algorithms and approaches specifically for this purpose.

1.1 Goals of this Project

The main goal of this project is to gain an overview over the field of terrain LOD, in theory as well as in practice. This includes introducing the basics of terrain programming, analyzing and comparing existing approaches and algorithms, and researching what approaches are used in the real world. For this purpose, a terrain demo application (named *ATLOD*) is developed in C++ and OpenGL.

1.2 Structure of the Report

This report is structured as follows:

- Chapter 2 gives an overview of work that has already been conducted in the area of terrain LOD and relevant literature for this project.

- Chapter 3 Lists a few real-world examples where terrain LOD is being used, such as game engines or geographic information systems.
- Chapter 4 introduces the reader to various approaches for terrain LOD, beginning with some basic background information on terrain modelling and LOD approaches in general. Afterwards, a selection of algorithms are presented in detail, including ROAM, GeoMipMapping, Quadtrees, Geometry Clipmaps, CDLOD. The algorithms are presented in chronological order of their publication date.
- Chapter 5 describes ATLOD, the demo application for terrain rendering implemented as part of this project. An overview of the functionalities is given and the main approaches and design decisions are discussed.
- Chapter 6 TODO
- Chapter 7 TODO

Chapter 2

Existing Work and Literature

2.1 Research Articles and Publications

Terrain LOD is a well-researched topic and over the last three decades, numerous approaches have been published. Some important publications are the following:

2.2 Level of Detail for Computer Graphics

Level of Detail for Computer Graphics by Luebke *et al.* [1] is a reference book for the topic of LOD published in 2002. The book builds on top of years of research in the area of LOD and provides an overview to many LOD techniques. For this project, chapter 7 "Terrain Level of Detail" of the book is especially relevant, as it dives into the topic of LOD for terrains specifically. It covers basic approaches and techniques for terrain LOD, common problems that can arise during rendering of terrains and some solutions to them, and a catalog of terrain LOD algorithms.

2.3 Focus on 3D Terrain Programing

Focus on 3D Terrain Programming by Trent Polack [2] is a book on terrain programming published in 2002. Part one of the book introduces the reader to the basics of terrains, such as height maps and texturing. Part two then covers some more advanced topics, including a selection of terrain LOD algorithms. The presented LOD algorithms are

- ROAM by Duchaineau *et al.* [3],
- the quadtree-based algorithm described in "Real-Time Generation of Continuous Levels of Detail for Height Fields" by Röttger *et al.* [4],

- and GeoMipMapping by de Boer [5].

The book also includes demo source code in C++ and OpenGL.

2.4 Virtual Terrain Project

The *Virtual Terrain Project* [6] was a project run from 2001 to 2013 that consisted of a collection of software, information and resources on terrain modelling and rendering, including a large overview of publications and implementations related to terrain LOD algorithms.

Chapter 3

Terrain LOD in Real-world Systems

3.1 Game Engines

3.1.1 Godot

Godot is a cross-platform game engine written in C#, C++ and its own scripting language GDScript. Terrains are supported in form of extensions developed by community members, which can be installed and used in Godot projects by game developers.

One such extension is Terrain3D by Cory Petkovsek [7] written in C++ for Godot 4. The LOD approach used in this extension is based on geometry clipmaps by Hoppe and Losasso [8]. The concrete implementation of the geometry clipmap mesh code was created by Mike J Savage [9].

Another extension for terrains is Godot Heightmap Plugin by Marc Gilleron [10] written in GDScript and C++. The extension uses a quadtree-based approach for terrain LOD.

3.1.2 Unity

Unity is another cross-platform game engine written in C# and C++, and has a built-in terrain system. The core engine source code of Unity is only accessible by owning an enterprise licence, therefore no information is given on which terrain LOD is used for the built-in terrain in Unity.

Nonetheless, there exists an open-source library for hierarchical LOD in Unity called HLODSystem developed by JangKyu Seo at Unity TODO citation . HLODSystem also supports terrains with its TerrainHLOD component, allowing for conversion from an Unity Terrain object to a HLOD mesh

with configurable parameters, such as chunk size and border vertex count. HLODSystem allows the developer to specify the mesh simplifier to be used and currently the only supported simplifier is UnityMeshSimplifier, an open-source mesh simplifier developed by TODO that utilizes the fast quadratic mesh simplification algorithm developed by TODO citation.

3.1.3 Unreal Engine

Unreal Engine is another cross-platform game engine written in C++ and features an integrated terrain system called the Landscape system. The technical documentation of Unreal Engine 5 mentions utilising GeoMipMapping for handling LOD for landscapes [11]. GeoMipMapping is a terrain LOD approach developed in 2000 by de Boer [5].

3.1.4 Frostbite

Frostbite is a closed-source game engine developed by DICE and is known for the *Battlefield* series. During the Game Developers Conference 2012, DICE presented the terrain system of *Battlefield 3*, which was developed with their Frostbite 2 engine. They mention a quadtree-based terrain LOD system and describe several optimizations regarding paging and streaming of terrain. TODO cite.

3.2 Geographic Information Systems

TODO: Check out Google maps, Google Earth, other GIS TODO: Maybe look at flight simulators (if information available)

Chapter 4

Algorithms and Approaches for Terrain LOD

4.1 Basics of Terrain LOD

4.1.1 Terrain Data Representation

Heightmaps

One way of representing terrains is using *heightmaps*. A heightmap is a $n \times n$-grid that contains the height value y for each (x, z)-position[1]. Positions are always spaced evenly in a grid-like manner, but the distance between any two neighboring positions is variable.

The main advantage of heightmaps is that they allow for very simple storage and manipulation of height data, e.g. in form of images, where low color values represent low areas of terrain and vice versa for high color values. For a grayscale image, up to 256 height values can be used and for an RGB image, more than 16 million height values are supported. Looking up a height value for a given (x, z)-position is easy as well, which consists of a simple lookup at the given position in the image. Figure 4.1 shows a 2000×2000 heightmap of the mountain Dom in Valais, Switzerland.

Triangulated Irregular Networks

An alternative to the heightmap is the *triangulated irregular network (TIN)* data structure. A TIN consists of a collection of 3D vertices, where the arrangement of vertices can be irregular. Figure 4.2 shows an example of a TIN.

[1] We always denote y for the up direction except if explicitly stated otherwise.

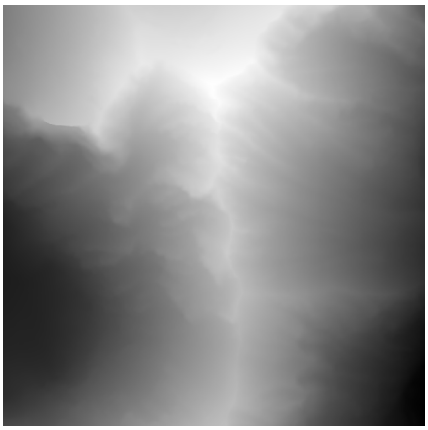

Figure 4.1: 2000×2000 heightmap of the mountain Dom in Valais, Switzerland retrieved from SwissTopo [12].

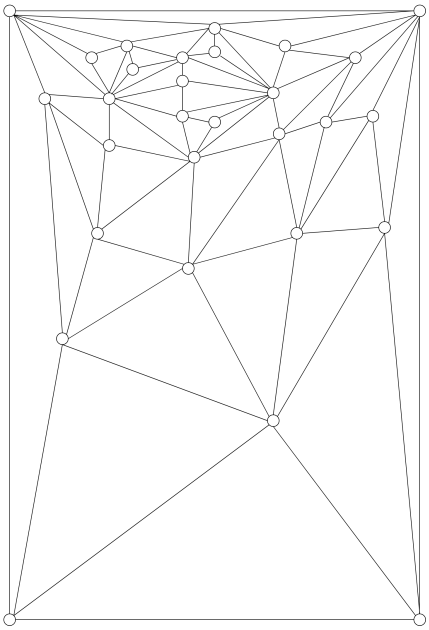

Figure 4.2: Example of a TIN. Note that the upper area represents a terrain area with many changes (e.g. mountains, hills, etc.), and the lower area represents an area with few changes (e.g. flat areas).

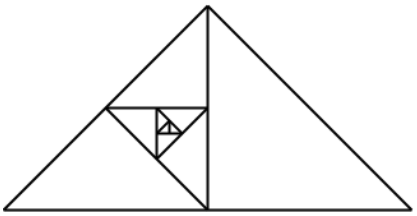

Figure 4.3: Example of a bintree.

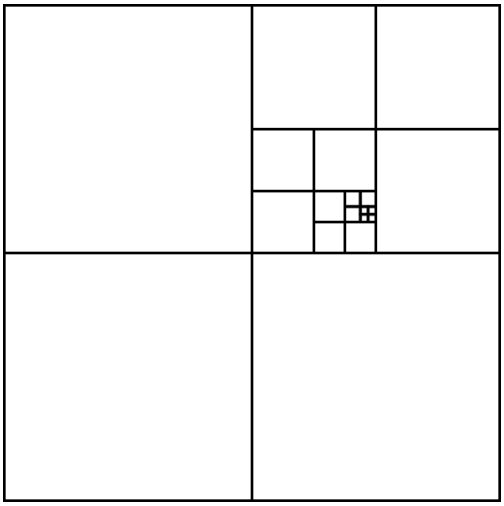

Figure 4.4: Example of a quadtree.

The main advantage of TINs is that fewer polygons need to be used for e.g. smooth terrain areas. Another advantage is that special terrain features can be modelled which are usually difficult to model with heightmaps, such as overhangs, cliffs and caves, The disadvantage of TINs, however, is that the full (x, y, z) coordinates need to be stored, whereas with heightmaps, only the height value y needs to be stored.

4.1.2 Bintrees and Quadtrees

Binary triangle trees (bintrees) and *quadtrees* are recursive data structures based on triangles and quads respectively. A bintree consists of up to two child triangles, both of which in turn also consist of up to two child triangles each, and so on, as shown in figure 4.3.

Quadtrees are structured similarly, with a quad consisting of up to four child quads, and each child quad consisting of up to four child quads, and so on. Figure 4.4 shows an example of a quadtree.

The main advantage of bintrees and quadtrees is that LOD can be mod-

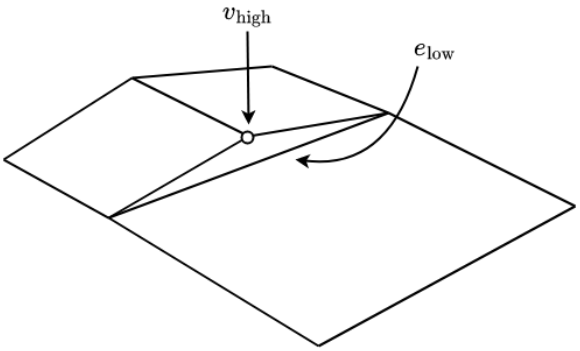

Figure 4.5: Example of a crack.

elled very naturally with them. Bintree/quadtree sections with few children correspond to a low LOD and vice versa for bintree/quadtree sections with many children.

4.1.3 Potential Problems During Terrain Rendering

While terrain LOD algorithms dramatically improve the performance of terrain rendering, there are certain faults that can occur during rendering.

Cracks Cracks and holes in terrains can appear when a higher LOD terrain section is bordered by a lower LOD terrain section. The main problem is that when a vertex v_{high} of a higher LOD terrain section lies on the edge e_{low} of a lower LOD terrain section and the y coordinate of v_{high} is greater or less than the height of e_{low} at that point, the difference in height causes the crack to appear, as shown in figure 4.5. Cracks can be solved by either

- removing the vertex in question (in figure 4.5 vertex v_{high}),
- inserting more vertices so that the LOD,
- or by force splitting the mesh so that the LOD does not contain such vertices that can cause holes.

Popping The phenomenon of *popping* occurs when the camera is moving and the transition of the terrain's LOD level causes visual pops to appear. Popping decreases the realism of the terrain and should be as minimal as possible. Popping can be reduced by introducing *vertex morphing*, i.e. by animating the transition of one LOD level to the next seamlessly.

4.2 ROAM

ROAM (short for **R**eal-time **O**ptimally **A**dapting **M**eshes) is a terrain LOD algorithm developed by Duchaineau *et al.* [3] published in 1997. ROAM represents the terrain mesh using bintrees. The algorithm is mainly CPU-based, which can be attributed to the less developed state of GPU programming at the time.

4.2.1 General Idea

The central idea of the algorithm is to use temporal coherence: the mesh from a previous frame $\mathbf{T}_{f-1}$ is used to compute the mesh of the current frame $\mathbf{T}$, rather than building up the mesh from ground up for each frame. This is done using two priority queues: a split queue $\mathcal{Q}_s$ and a merge queue $\mathcal{Q}_m$. The split queue contains splittable triangle pairs () and the merge queue contains mergable triangle pairs (T, T_B). The elements of the priority queues are ordered by various geometric error metrics, which will be explained in the subsection "Error Metrics". ROAM is a greedy algorithm, meaning it will always compute the most optimal mesh for each frame.

4.2.2 Error Metrics

The error metrics used for the priority queues are the following:

Wedgies The so-called *Wedgies* are nested bounding volumes around triangles that are computed while building an initial mesh at the beginning of the algorithm. A wedgie is defined to contain the entire x and z extent[2] of a triangle and the height y including some additional space above and below the highest and lowest points, respectively.

Geometric Screen Distortion

Other Metrics Other priority metrics mentioned include

4.2.3 Reported Performance and Conclusion

Duchaineau et al. report that the runtime of ROAM is proportionate to the number of triangle changes.

[2]The original ROAM paper uses z for the up direction. For the sake of consistency with the rest of this report, we will use y as the up direction here.

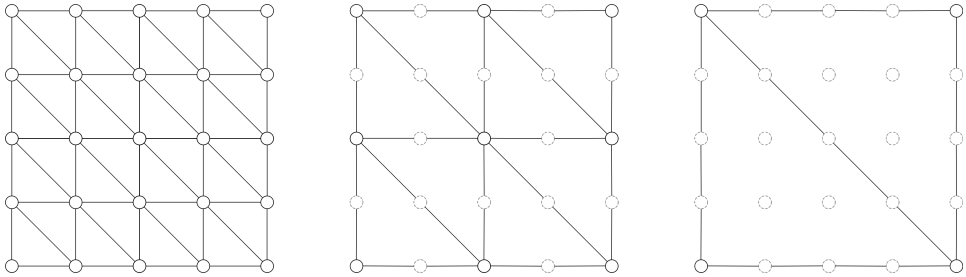

Figure 4.6: Example of a 5×5 block with the maximum LOD level of 2 (left), LOD level of 1 (middle) and minimum LOD level of 0 (right). The omitted vertices of the lower LOD blocks are shown here as dotted circles.

4.3 GeoMipMapping

Geometrical Mipmapping (GeoMipMapping) is a terrain LOD approach developed by de Boer [5] in the year 2000.

4.3.1 General Idea

The central idea of GeoMipMapping is its analogy to texture mipmapping: just like how textures of far away objects are rendered using lower resolution texture mipmaps, terrain areas that are far away from the camera should also be rendered with a lower resolution mesh. This is achieved by splitting up the terrain into so-called *blocks* (also called *patches*) of a fixed width $2^n + 1$ for an arbitrary n. At each LOD level, the number of vertices on one side of a block is $2^l + 1$, where $0 \leq l \leq n$ is the current LOD level[3]. For example, a 5×5 terrain block contains $2^2 + 1 = 5$ vertices on one side at the maximum LOD level 2, $2^1 + 1 = 3$ vertices at LOD level 1 and $2^0 + 1 = 2$ vertices at the minimum LOD level 0, as shown in figure 4.6.

[3]The original paper uses 0 to denote the maximum LOD level and vice-versa for the minimum LOD level.

Chapter 5

ATLOD: A Terrain Level of Detail (Renderer)

This chapter describes *ATLOD* (short for **A** **T**errain **L**evel **o**f **D**etail (Renderer)), the demo terrain rendering application.

5.1 Preliminaries

5.1.1 Used Technologies

ATLOD is written in C++17 and OpenGL 4.2. For compiling build files, CMake (minimum version 3.5) is used. ATLOD uses the following third-party libraries:

- GLM: The *OpenGL Mathematics (GLM)* library provides functionality for the mathematics of graphics programming, such as classes for vectors, matrices and perspective transformations.
- GLEW: The *OpenGL Extension Wranger Library (GLEW)* is an extension loading library for OpenGL.
- GLFW:
- STB: STB is a collection of header-only libraries developed by Sean Barrett TODO cite. ATLOD uses `stb_image.h` for loading images of heightmaps and textures.

The source code is hosted on GitHub on the repository AmarTabakovic/3d-terrain-with-lod and is licensed under TODO.

5.1.2 Chosen Algorithms

The chosen algorithms and their reasons for implementing them are the following:

Naive brute-force algorithm The naive brute-force algorithm consists of simply reading in all height values from the heightmap as vertices and rendering them directly to the screen. The main reason for implementing the naive brute-force algorithm is to motivate the usage of terrain LOD algorithms by showing the difference in performance compared to the optimized LOD approaches.

De Boer's GeoMipMapping The GeoMipMapping

Algorithms Not Chosen

ROAM Duchaineau's ROAM was not chosen to be implemented due to its CPU-heavy nature, which makes it not suitable for today's highly performant GPUs.

5.2 Architecture

TODO High-level class diagram

Chapter 6

Results

Chapter 7

Discussion and Conclusion

7.1 Further Work

7.2 Reflexion

Bibliography

[1] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., USA, 2002.

[2] Trent Polack and Willem H. de Boer. *Focus on 3D Terrain Programming*. Premier Press, 2002.

[3] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In *Proceedings of the 8th Conference on Visualization '97*, VIS '97, pages 81–88, Washington, DC, USA, 1997. IEEE Computer Society Press.

[4] Stefan Röttger, Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel. Real-time generation of continuous levels of detail for height fields. *Journal of WSCG*, 6(1-3), 1998. URL: `http://hdl.handle.net/11025/15845`.

[5] Willem H. de Boer. Fast terrain rendering using geometrical mipmapping. In *The Web Conference*, 2000. URL: `https://api.semanticscholar.org/CorpusID:7296927`.

[6] Virtual terrain project. `https://vterrain.org`.

[7] Cory Petkovsek. Terrain3d. `https://github.com/TokisanGames/Terrain3D`, 2023.

[8] Frank Losasso and Hugues Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. *ACM Trans. Graph.*, 23(3), 2004. `doi:10.1145/1015706.1015799`.

[9] Mike J Savage. Geometry clipmaps: simple terrain rendering with level of detail. `https://mikejsavage.co.uk/blog/geometry-clipmaps.html`, 2017.

[10] Marc Gilleron. Godot heightmap plugin. `https://github.com/Zylann/godot_heightmap_plugin`, 2023.

[11] Epic Games. Landscape Overview — Unreal Engine 5.3 Documentation. https://docs.unrealengine.com/5.3/en-US/landscape-overview/.

[12] Federal Office of Topography swisstopo. swissALTI3D. https://www.swisstopo.admin.ch/en/geodata/height/alti3d.html.