

# Using Mesh Shaders for Continuous Level-of-Detail Terrain Rendering

Matthias Englert

Tinman 3D

Fulda, Germany

me@tinman3d.com

## ABSTRACT

Using a commercial terrain engine [Englert 2012] as an example, we discuss both advantages and disadvantages of the continuous level-of-detail approach [Lindstrom and Pascucci 2001] to terrain-rendering while comparing our implementation with other hybrid [Dick et al. 2010] and implicit [Jad Khoury and Riccio 2018] approaches. We show how mesh shader and inline ray-tracing features of DirectX 12 Ultimate can be used to remedy those disadvantages.

## KEYWORDS

terrain rendering, continuous level-of-detail, right-triangulated irregular network, mesh shader, inline ray-tracing

### ACM Reference Format:

Matthias Englert. 2020. Using Mesh Shaders for Continuous Level-of-Detail Terrain Rendering. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '20 Talks)*, August 17, 2020. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/3388767.3407391>

## 1 INTRODUCTION

We describe the essential stages of a general-purpose pipeline for continuous level-of-detail terrain-rendering in real-time: using on-the-fly data processing and/or offline processing, terrain data from arbitrary sources is transformed into a uniform hierarchical data representation (UHDR). At run-time, the UHDR is transformed into a view-dependent right-triangulated irregular network (RTIN) [Lindstrom and Pascucci 2001], a fixed-size vertex buffer (VB) and a set of texture atlases (TAs).

The hierarchical structure of the RTIN is used to perform spatial queries, such as picking, horizon culling and frustum culling. The VB encodes vertex data in a way that resolves floating-point precision issues; this allows to use terrain maps up to a size of  $2^{30} + 1$  by  $2^{30} + 1$  samples.

We describe a scheme for arranging vertices in the VB which provides near-optimal cache locality for the applied algorithms, both on the CPU and the GPU. We introduce the concept of palette-based material tokens (PBMT), which are stored in the VB and can be used as a replacement for traditional texture splat-maps.

The TAs are used to provide unique texture data for the terrain, with a maximum virtual texture dimension of  $2^{30}$  by  $2^{30}$  texels. For rendering, the RTIN is transformed into a contiguous triangle-based GPU primitive. View-dependent hardware tessellation techniques are used to implement displacement-mapping for the PBMT.

We present implementation details for various aspects of the pipeline, using different GPU feature levels for comparison: DirectX 9, DirectX 10 (geometry shaders, texture arrays), DirectX 11 (hull, domain and compute shaders), DirectX 12 Ultimate (amplification and mesh shaders, inline ray-tracing). We show that recent GPU features can be used to move critical pipeline steps onto the GPU, which removes bottlenecks like CPU triangulation and upload of triangle-based GPU primitives.

## 2 TERRAIN PIPELINE

In contrast to recent implicit approaches [Jad Khoury and Riccio 2018], our approach uses an explicit hierarchical data structure on the CPU in the form of an extended directed acyclic graph (xDAG), to encode the RTIN that represents the terrain mesh.

Unlike other hybrid approaches [Dick et al. 2010], we solely rely on triangles for rendering, thus requiring hardware rasterization or hardware ray-tracing.

These design decisions impose certain challenges for a terrain-engine [White 2008], for which we describe viable solutions:

- Using a vertex pool to avoid dynamic allocations
- Maintaining cache locality in the vertex pool
- Fixed-path xDAG traversal without arithmetic
- Mirroring data to the GPU incrementally
- Using NO\_OVERWRITE semantic to avoid GPU stalls
- Using meshlets instead of triangle-based GPU primitives

Having a triangle-based explicit data structure on the CPU allows for versatile features, such as spatial analysis on the CPU (picking, collision detection, visibility determination), support of downgrade GPU feature levels (e.g. DirectX 9, WebGL) and general-purpose utilization in content pipelines (e.g. export of terrain tiles).

### 2.1 Overview

The *Data Processing* stage performs on-the-fly processing of terrain data from arbitrary sources and/or streams in data resulting from offline processing, producing content in the UHDR, which is accessed by the subsequent pipeline stage.

The *Mesh Refinement* stage transforms the UHDR into a view-dependent triangle mesh (xDAG), a fixed-size vertex buffer (VB) and a set of texture atlases (TAs). The VB encodes vertex data in a way that resolves floating-point precision issues; this allows to use terrain maps up to a size of  $2^{30} + 1$  by  $2^{30} + 1$  samples. The VB may

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGGRAPH '20 Talks, August 17, 2020, Virtual Event, USA

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7971-7/20/08.

<https://doi.org/10.1145/3388767.3407391>

**Table 1: Pipeline stages per GPU feature level**

Stage	DX9	DX10	DX11	DX12
Data Processing	CPU	CPU	CPU	CPU
Mesh Refinement	CPU	CPU	CPU	CPU
Batching	CPU	CPU	CPU	GPU
Triangulation	CPU	CPU	CPU/GPU	GPU
Tessellation	n/a	GPU	GPU	GPU

include palette-based material tokens, which can replace traditional texture splat-maps. The TAs are used to provide unique texture data for the terrain, with a maximum virtual texture dimension of  $2^{30}$  by  $2^{30}$  texels.

The *Batching* stage exploits nested bounding geometries [Lindstrom and Pascucci 2001] in the xDAG in order to apply culling techniques, such as horizon culling and frustum culling. Then, render batches are generated for the relevant terrain parts. Each render batch corresponds to a GPU draw command, which may be direct or indirect, depending on the GPU feature level.

The *Triangulation* stage traverses the xDAG and generates indexed triangle-based primitives for the generated terrain batches, which are submitted to the GPU for rasterization. When run on the CPU, index data must be uploaded to the GPU for each frame, using DISCARD semantic, to avoid stalling. This consumes valuable bandwidth and enforces sequential processing, which quickly becomes a bottleneck.

The *Tessellation* stage applies view-dependent subdivision to the triangulation for displacement mapping. Since the *Mesh Refinement* stage delivers a crack-free triangle mesh at fine resolution with tangent spaces and world-to-pixel size coefficients, the subdivision algorithm may be simplistic, for example a basic 1-to-4 triangle split. In practice, two levels of subdivision are sufficient in most cases.

### 3 IMPLEMENTATION DETAILS

Supplementary pseudo-code (see *Algorithms.cs*) is provided to demonstrate several implementation details.

#### 3.1 UHDR

We outline the structure of the UHDR for rectangular and planetary terrains, including a storage partitioning scheme based on the level-of-detail of individual terrain samples for efficient retrieval of terrain data.

#### 3.2 xDAG

We present the structure of the xDAG and list the algorithms that need to be performed on it, giving details for some of the most important ones. We give details regarding vertex pooling and the use of spatial buckets in order to maintain cache locality.

#### 3.3 Culling

Horizon culling and view frustum culling are typically applied on the CPU, since the *Data Processing* stage uses the results in order to fetch new texture data only for the visible terrain parts.

For DX12, we show how on-the-fly visibility culling can be performed in amplification shaders: The hierarchical structure of the xDAG is mirrored to the GPU, in the form of bounding volume hierarchies. Then, inline ray-tracing is used to perform spatial queries for visibility determination. This approach is advantageous when rendering the same scene into different views, for example shadow-map cascades.

#### 3.4 Batching

Traditionally, render batches are generated by the CPU and submitted to the GPU through draw commands. For DX12, we describe how to encode render batches, upload them to the GPU and process them in an amplification shader. This way, the CPU only needs to issue a single draw command and the GPU can process the render batches in parallel.

#### 3.5 Triangulation

We show how the xDAG can be traversed in order to generate contiguous triangle-strip ( $\sim 1.61$  indices per triangle) and triangle-list (3 indices per triangle) primitives. For DX11, we describe the terminal-triangle primitive ( $\sim 0.25$  indices per triangle), which is expanded to a triangle-list on the GPU, using a compute shader. For DX12, we introduce the sector-list primitive ( $< 0.01$  indices per triangle), which is expanded to meshlets on the GPU. With sector-lists, required CPU work for triangulation becomes negligible and GPU processing becomes highly parallel.

#### 3.6 PBMT

We describe the encoding of the PBMT and present CPU and GPU algorithms for performing filter operations on them.

#### 3.7 Tessellation

For DX10, we describe a simple 1-to-4 triangle subdivision scheme which can be implemented in a geometry shader for up to 2 subdivision levels. For DX11, fixed-function hardware tessellation is used, via hull and domain shaders. For DX12, we show how to use a variant of the DX10 subdivision scheme in the mesh shader, to replace the fixed-function hardware tessellation stage.

### 4 EXAMPLES

For demonstration, we show two terrains:

- Rectangular game-like terrain with PBMT, based on a terrain map of 1,048,577 by 1,048,577 samples
- Planetary Earth terrain with TAs (satellite imagery), based on a terrain map of 1,073,741,825 by 1,073,741,825 samples.

### REFERENCES

- Christian Dick, Jens Krüger, and Rüdiger Westermann. 2010. GPU-Aware Hybrid Terrain Rendering. In *Proceedings of IADIS Computer Graphics, Visualization, Computer Vision and Image Processing 2010*. 3–10.
- Matthias Englert. 2012. *Tinman 3D SDK - Real-time Terrain*. <https://www.tinman3d.com>
- Jonathan Dupuy Jad Khoury and Christophe Riccio. 2018. *Adaptive GPU Tessellation with Compute Shaders*. <https://github.com/jdupuy/opengl-framework/tree/master/demo-isubd-terrain>
- Peter Lindstrom and Valerio Pascucci. 2001. Visualization of Large Terrains Made Easy. *Proc. IEEE Visualization 2001*. <https://doi.org/10.1109/VISUAL.2001.964533>
- Matthew White. 2008. Real-Time Optimally Adapting Meshes: Terrain Visualization in Games. *Int. J. Computer Games Technology* 2008 (01 2008). <https://doi.org/10.1155/2008/753584>