Vision, Modeling, and Visualization (2025)
B. Egger and T. Günther (Eds.)

Fast Rendering of Large-Scale Dynamic Multi-Layered Height Maps

A. M. Nilles[1] and S. Müller[1]

[1]University of Koblenz, Institute for Computational Visualistics, Germany

Abstract

In this paper, we develop two methods for fast visualization of fully dynamic large-scale multi-layered height maps (MLHMs). MLHMs are a fairly uncommon data structure in computer graphics, which has been used as an efficient representation of 3D terrain, among other applications. Recently, a 3D hydraulic erosion simulation that utilizes this data structure effectively, allowing for real-time simulation of large scale terrain, was developed, but the fast simulation was paired with slow visualization. We extend this previous work with two efficient visualization methods. Rendering the MLHM as boxes is done using ray tracing, while a smooth surface is rendered by ray marching an implicit surface generated by smoothing the MLHM, following previous work. Both techniques are accelerated using a hierarchical data structure built directly from the MLHM, which enables quadtree-like traversal using 2D DDA, where each 2D cell contains 3D axis-aligned bounding boxes (AABBs). This data structure is adapted to accelerate ray marching by appropriately padding AABBs with the smoothing radius. We further propose a soft shadow method and geometric ambient occlusion that work in tandem with this data structure. Our visualization is fast enough to support fully dynamic terrain in real-time, where simulation, creation of our data structure and visualization are done every single frame in real-time for terrain resolutions up to 4096^2*. For lower resolutions, it is possible to run expensive ray tracing and geometric ambient occlusion effects at full window resolution every frame with real-time or interactive frame rates.*

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing;*** *Volumetric models; Real-time simulation;*

1. Introduction

Multi-layered height maps (MLHMs) are a fairly rare data structure in computer graphics, which historically appeared under different names [PGGM09; SCC*10; GPF10; BMPG11], with the current name having been established more recently [DGV*18; DVG*18; SHVS18; NGWM24].

The most common use case for MLHMs is as an efficient volumetric terrain representation that serves as a natural extension of height maps with support for overhangs. It is often used in dynamic contexts such as erosion simulations [PGGM09; SCC*10; NGWM24] and fluid simulations [BMPG11; DGV*18; DVG*18], typically for interactive real-time applications.

Due to the relative obscurity of the data structure, there is a severe lack of research concerning efficient rendering methods for the visualization of MLHMs. While vast amounts of research exist around height maps, the standard height map rendering methods cannot be used for MLHMs because unlike height maps, they are fully volumetric representations. Previous work [PGGM09; NGWM24] mentions visualization briefly, with a sophisticated implicit surface representation of MLHMs in [PGGM09], however neither work goes into detail concerning efficient rendering, as they both mainly focus on interaction, modification and simulation of terrain.

Our work aims to fill this identified research gap by exploring two efficient rendering techniques. One renders the individual columns in an MLHM as boxes using ray tracing, and the other renders a smooth terrain using the implicit surface representation from [PGGM09] via ray marching. The proposed methods are tailored to support fully dynamic MLHMs. We use a recent 3D hydraulic erosion simulation [NGWM24] as our target application, meaning that every single column of the MLHM can change each frame. This is in contrast to [PGGM09], where changes are localized, which made continuation-based meshing algorithms viable.

The main contributions of this work are:

- a hierarchical 2D data structure for efficiently ray tracing and ray marching MLHMs via quadtree DDA traversal,
- a sparse ray marching method for MLHM implicit surfaces,
- a soft shadow algorithm applicable to sparse ray marching,
- and a geometric ambient occlusion method for MLHM terrains.

This paper is structured as follows: Section 2 discusses relevant related work on terrain visualization and MLHMs. Section 3 details the previous work that we used as a basis. We continue by explaining our method in Section 4, followed by our results in Section 5, where we compare quantitatively and qualitatively with the previous visualization method. Section 6 concludes with a summary, important limitations, and possible future work. Additional derivations and evaluation, source code, scenes, raw evaluation data and scripts that

DOI: 10.2312/vmv.20251227

generate our graphs are included in the supplementary material. The source code is also available open source on GitHub [NG25].

2. Related work

This section will introduce relevant related work, focusing mainly on height map-based terrain rendering as well as the MLHM data structure. Section 3 will go into more detail concerning the most relevant previous work on MLHM visualization [PGGM09; NGWM24].

2.1. Height map and terrain rendering

Early work on height map-based terrain rendering focused on level of detail (LOD) for rendering efficiency and out-of-core rendering, such as continuous LOD via geometry clipmaps [LH04] and GPU tessellation-based methods [KJCH15]. Common data structures include (semi)-regular grid structures and hierarchical data structures such as quadtrees or mipmapping [De 00; LP01] to dynamically adapt LOD based on pixel-level error metrics. More recently, summed-area tables have been proposed for multi-resolution terrain [LZW*21], while [ZWH*21] incorporate surface roughness and visibility analysis into LOD calculation and culling. Typical approaches to height map-based terrain rendering can also be extended to layered height maps as in [ŠBBK08], which can be simplified to a regular height map, where the layer information maps to a material.

Related to height map rendering, relief and per-pixel displacement mapping techniques use height fields to render detail without additional geometry. These methods work by computing ray-height field intersections. An early example is relief texture mapping [OBM00] which was later extended to multi-layer representations [PO06]. Other approaches use distance fields instead of height fields [Don05] or introduce dynamic step sizes via cone stepping [Dum06; PO07].

Most of the previous methods are tailored to static terrain. Lambers and Kolb [LK11] instead presented a bottom-up mesh refinement method for dynamic height maps. Some methods also explored ray tracing, such as GPU ray casting in [DKW09]. [TIS08] proposed maximum mipmaps alongside ray tracing for dynamic height maps. More recently, [DMHS15] developed a flexible architecture for height map ray tracing, demonstrating real-time feasibility, while [LLS16] focused on empty-space skipping in terrain ray casting to increase performance at scale.

2.2. Multi-layered height maps

MLHMs have been invented multiple times, with the goal of efficiently representing 3D volumes while supporting overhangs, which are not possible with regular height maps. For efficient 3D terrain representation, [PGGM09] proposed a "two dimensional grid of material stacks", while [SCC*10] independently proposed segmented height fields. Meanwhile, [GPF10] developed n-layer height maps for 3D reconstruction to support overhangs. Borgeat, Massicotte, Poirier, and Godin [BMPG11] extended the height map-based virtual pipes fluid simulation [OH95] to MLHMs for surgical training. Similarly, another such extension was developed in [DGV*18; DVG*18] based on previous work [Kel13]. Sarkar, Hampiholi, Varanasi, and Stricker [SHVS18] developed a MLHM data structure for 3D shapes that can be used with convolutional neural networks. Most recently, MLHMs were used for real-time 3D hydraulic erosion simulation [NGWM24].

3. Background

As outlined in Sections 1 and 2, MLHMs are not a very common term in computer graphics and thus not a well-defined data structure. This section will detail our definition of MLHMs and explain existing visualization methods.

3.1. Definition

Following [NGWM24], we define a MLHM as an $N_x \times N_y$ 2D grid of cells $\boldsymbol{i} = (i, j)$. Cells contain 1 to N_z columns $\boldsymbol{k} = (i, j, k)$ which are stacked from bottom to top. The lowest column extends downward to infinity. Each cell is size $l_c \times l_c$ horizontally and N_z is chosen to be small (8 or less).

Columns contain a layered stack of materials with bedrock at the bottom, followed by sand, water and air. For $N_z = 1$, this is equivalent to the layered height map in [ŠBBK08] and extends this previous work naturally for $N_z > 1$. Stacking multiple columns on top of each other is what allows for air gaps and thus overhangs and similar structures, which is what makes the MLHM fully volumetric and differentiates it from (layered) height maps.

Our version of MLHM is different from the original in [PGGM09], which was a two-dimensional grid of material stacks without any strict ordering. This decision was made in [NGWM24] because the strict ordering allowed to adapt existing simulation methods for layered height maps to the volumetric MLHM. Specifically, our implementation uses the same data layout as [NGWM24] and stores four values per column k: the absolute height of the bedrock surface, sand and water thickness and the absolute ceiling height, which is the lower bound of the bedrock of column $k+1$.

3.2. Previous visualization

The main types of visualization found in [NGWM24; PGGM09] include direct rendering of the underlying box shapes (Figure 1, top row), as well as smooth surface rendering (Figure 1, bottom row). Box visualization is useful as it represents the underlying data structure without abstraction, allowing for clear insights into the erosion simulation. Smooth surfaces on the other hand are the representation of the terrain surface that the MLHM approximates.

In [NGWM24], the authors focused purely on the simulation and only briefly touched on how they visualized their results, however, the entire source code was provided, allowing for further insights.

Box visualization (`BoxVis`) was implemented by first filling a draw-list with all existing columns, which are then rendered in a single draw call as points. These points are expanded into boxes in the geometry shader. The approach suffers from huge overdraw, as the majority of the side surfaces of each column are occluded. This visualization is orders of magnitudes slower than the erosion simulation [NGWM24].

The smooth surface visualization (`SmoothVis`) is built on top

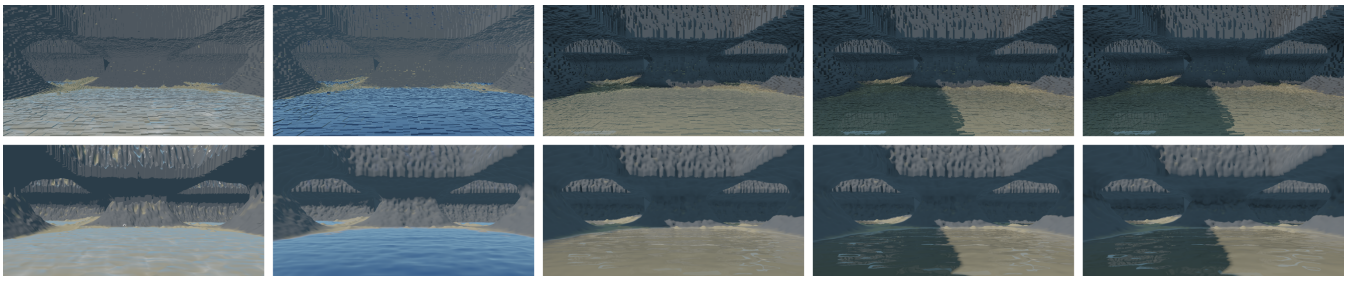


Figure 1: *The* `Arches` *scene rendered using box visualization (top row) and smooth surface visualisation (bottom row). From left to right: previous method [NGWM24], followed by* `low`*,* `mid`*,* `high` *and* `max` *presets using our method (see Section 5).*

of the `BoxVis`. Top and bottom vertices of each box are modified by interpolation with neighboring columns. Which neighboring columns are used depends on a number of rules that determine the highest possible overlapping column for the top surface and the lowest possible one for the bottom. Further rules prevent nonbilateral interpolation for edge cases. The advantage of this approach is that many of the side surfaces of adjacent columns are identical and can be discarded as they are guaranteed to be invisible. This `SmoothVis` method is thus significantly faster than the `BoxVis`. However, it is slow relative to the simulation at high resolutions. Furthermore, this method is unable to smooth the entire terrain, as side surfaces are not smoothed. This leads to visual artifacts in situations where columns that can be interpolated are next to ones that are unable to be interpolated [NGWM24].

While more sophisticated mesh generation and interpolation rules could salvage these approaches, it would likely lead to highly dynamic triangle counts per column and convoluted branching code, which is why we looked elsewhere for a cleaner solution.

3.3. Implicit surface representation

A much more sophisticated approach to `SmoothVis` was already proposed in [PGGM09]. They combine a skeleton function g that is 1 inside the boxes of a material or set of materials $\mathcal{M}$ and 0 outside with a simple box smoothing kernel h, where $\boldsymbol{p}$ and $\boldsymbol{q}$ are 3D positions and σ is the smoothing radius:

$$g(\boldsymbol{p}) = \begin{cases} 1 & \text{if } \boldsymbol{p} \in \mathcal{M}, \\ 0 & \text{otherwise} \end{cases}, \tag{1}$$

$$h(\boldsymbol{q}) = \begin{cases} 1 & \text{if } \max_i(q_i) < \sigma \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

The convolution function is defined as

$$i(\boldsymbol{p}) = (h * g)(\boldsymbol{p}) = \int_{\mathbb{R}^3} g(\boldsymbol{p}) h(\boldsymbol{p} - \boldsymbol{q}) d\boldsymbol{q} = \mathcal{V}_{\mathcal{M}}, \tag{3}$$

which simplifies to a function that measures the volume $\mathcal{V}_{\mathcal{M}}$ of the terrain material boxes inside the smoothing kernel as shown in Figure 3a. 50% of the box volume $\mathcal{V}_{\Omega} = 8\sigma^3$ is used as the isosurface, forming the implicit surface [PGGM09]:

$$f(\boldsymbol{p}) = 2\frac{\mathcal{V}_{\mathcal{M}}}{\mathcal{V}_{\Omega}} - 1. \tag{4}$$

In [PGGM09], f is used for the interactive editing of terrain. Visualization is done by generating a mesh from f using standard methods [BB97], which is not further specified. As their method involves local editing tools, we assume that they use a continuation-based approach that only updates the mesh locally in regions of change, yielding good CPU performance even at high resolutions.

Polygonizing the implicit surface allows the use of standard rasterization methods tailored to real-time visualization. However, it offers numerous challenges when used in context of a large-scale erosion simulation such as [NGWM24]. During each simulation step, all columns in the scene can change at the same time. Thus, the entire mesh has to be generated every frame, which is expensive for large resolutions. Continuation-based methods are difficult to parallelize on the GPU as they introduce the need to track voxels that have already been visited in parallel. This can lead to dynamic, branch-diverging and cache inefficient code, while also requiring dynamic amounts of memory. Other approaches require choosing a fixed 3D voxel resolution and handling all voxels in parallel. This is problematic because a high vertical resolution is required to capture the detail of the MLHM, quickly leading to billions of voxels. This issue is why [PGGM09; NGWM24] used a MLHM in the first place. While not necessarily impossible to do in real-time, we chose to abandon the idea of mesh generation for these reasons, in favor of ray tracing.

4. Our method

Our visualization has been built directly on top of the previous work [NGWM24], enabling real-time visualization during simulation. While our implementation is tailored to this data structure, it can be modified to support other MLHMs such as in [PGGM09]. Our main requirement is low vertical complexity, which translates to a low number of columns per cell. Extensions that can handle higher complexity are feasible and will be discussed in Section 6.

4.1. Box ray tracing

The MLHM defined in Section 3.1 can be viewed as a 2D uniform grid data structure containing 3D box primitives. While typical ray tracing applications would use a 3D uniform grid instead, their traversal algorithms can be leveraged for our data structure. This only requires projecting the 3D camera rays into the 2D grid and traversing the uniform grid using *i.e.* the DDA (digital differential

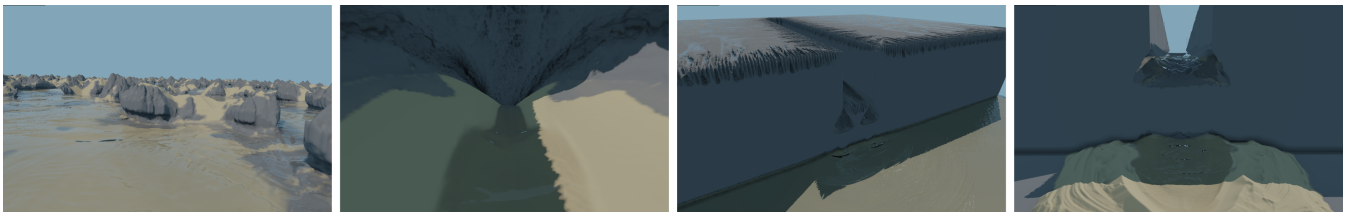


Figure 2: *Smooth surface visualization rendered using our ray marching approach at* `max` *preset. Scenes from left to right:* `RiversRain`*,* `Cave`*,* `ChasmRain` *and* `ChasmSource`*, each at* 2048^2 *terrain resolution.*

analyzer) algorithm. For each 2D cell, all contained columns are then intersected using a 1D Ray-AABB (axis-aligned bounding box) test. The entry and exit point along the other 2 dimensions are already provided by DDA. Columns are exactly as wide as the cells and do not overlap, so the closest hit for the entire ray is the closest hit of the first cell that has intersections.

This simple algorithm is already faster for `BoxVis` than the naive rendering method in [NGWM24]. By using the MLHM directly as a 2D data structure, the low vertical scene complexity is leveraged and building a 3D data structure is avoided, saving memory. However, the MLHM contains at least one column per cell, which means that intersection tests happen for every cell that is traversed, which is inefficient. This is in contrast to typical ray tracing applications, where the majority of cells are usually empty. In the past, this caused uniform grids to be faster than octrees [FTI86]. In our case, we can however benefit tremendously from constructing a hierarchical data structure.

4.2. Hierarchical acceleration data structure

We build a hierarchy of increasingly lower resolution versions of the MLHM, down to a resolution of 4×4. Level 0 contains the actual scene columns at full resolution in half-precision format (Figure 3a). We choose a bottom-up approach, where a coarse column in level l with vertical index k is merged using the columns of the same vertical index in 2×2 cells on level $l-1$ (Figure 3b). Resolution halves on each axis and l_c doubles with every level. Merging is done by picking the minimum of the lower bound and maximum of the higher bound of the respective parent columns. Starting with level 1, we additionally halve N_z every two levels to reduce vertical complexity on coarser grids. Excess columns on the finer level are merged into the highest coarser column. On the GPU, this process is fast enough to build the data structure from scratch after every simulation step.

To support shadow rays, refractions and underwater rendering, we need to be able to intersect with only solids, with solids and air combined, as well as with solids and water combined. The hierarchy has to provide AABBs for all these cases, which can be done with just 4 values per column: minimum air level $a_{\min}$, minimum bedrock level $b_{\min}$, maximum sand level $s_{\max}$ and maximum water level $w_{\max}$. Minimum and maximum refer to the operation used for merging columns. For example, $a_{\min}$ is merged for level 1 by choosing the minimum of the absolute water height among the columns that are being merged, which is where the air layer begins. The remaining levels can then use the minimum of $a_{\min}$ of the columns being merged on the previous level. With these four values, we can construct bounding volumes for the three required cases: $[b_{\min,k}, s_{\max,k}]$ (solids), $[b_{\min,k}, w_{\max,k}]$ (solids and water) and $[a_{\min,k-1}, s_{\max,k}]$ (solids and air).

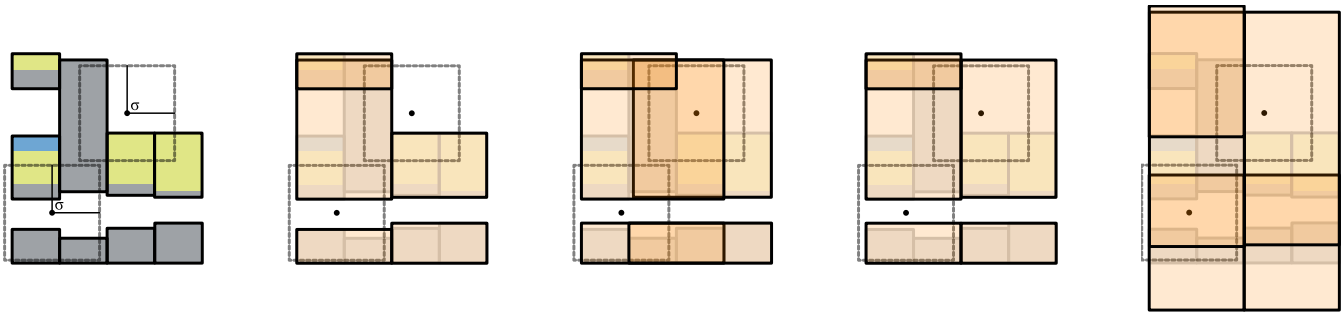
The data structure can be traversed analogously to octrees by adapting the DDA algorithm. Starting at the coarsest level, if the ray intersects a coarse AABB, traversal continues on the next finer level, otherwise traversal continues on the current level. The most crucial part of traversal is the backtracking to coarser levels. We implemented this with a stack that keeps track of the distance along the ray at which the current cell is exited. This is pushed when the ray moves down. If the ray misses the contents of a cell, the stack is popped in a loop until the distance on the stack is larger than the current distance, moving to a coarser level each step.

A stack costs memory, can be cache unfriendly and can lead to excessive jumps through the hierarchy, which is part of the reason why simple uniform grids have historically outperformed octrees [FTI86]. We thus also implemented a simpler method using a miss counter, set to 0 every time we step up or down in the hierarchy. It is incremented whenever the columns in a cell are missed. Once it exceeds a threshold N_{miss}, we jump to the next coarser grid. This requires no stack and causes fewer jumps through the hierarchy. We use $N_{\text{miss}} = 8$, except for the finest level where we set it to $N_{\text{finemiss}} = 16$.

4.3. Smooth surface ray marching

`SmoothVis` requires ray marching, as it is defined using an implicit surface (see Section 3.3). The coarse columns as defined in Section 4.2 are not valid AABBs of the implicit surface due to the smoothing, which can extend the terrain surface beyond the confines of the coarse AABBs of the hierarchy (see Figures 3a and 3b). We thus incorporate the smoothing radius σ into the hierarchy, which involves padding heights by $\pm\sigma$ and merging a $2(1+\frac{\sigma}{l_c}) \times 2(1+\frac{\sigma}{l_c})$ neighborhood of columns (see Figures 3c to 3e). This is only necessary for level 1 of the hierarchy as the other levels are guaranteed to be correct when generated from valid AABBs. Only the vertical bounds of the AABBs are affected by this, it is not necessary to increase the horizontal bounds due to the padded neighborhood considered for merging the columns.

Ray marching uses the same traversal algorithm from Section 4.2. At level 0, the algorithm switches from ray tracing AABBs to ray marching on the fine grid, where $f(\boldsymbol{p})$ is evaluated every step.



(a) *Initial terrain* **(b)** *Initial AABBs* **(c)** *Horizontal padding* **(d)** *Horizontal clipping* **(e)** *Vertical padding*

Figure 3: *Creation of level 1 of our acceleration data structure. The figures show a side view. Figures 3a and 3b visualize steps that occur for box and smooth visualizations, the remaining figures are specific to smooth visualization with smoothing radius* σ*. Two boxes with radius* σ *are shown, both of which are filled with terrain by at least 50%,* i.e. *their centers are inside the implicit surface. For our smooth visualization method, all points that are part of the implicit surface, such as the two centers shown in this figure, need to be inside the AABBs. Initial AABBs are created in a* 2×2 *neighborhood and are not sufficient as the centers lie outside of them. Padding horizontally with* σ *incorporates a larger neighborhood, which is clipped back to the previous horizontal extents. This solves the problem for one center point, while the other one requires additional vertical padding with* σ*.*

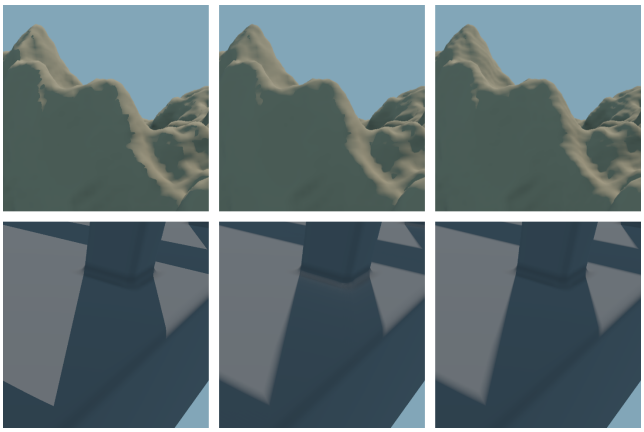


Figure 4: *Comparison of normals (top) and shadows (bottom). The top row shows forward differencing on the left, central differencing in the middle and central differencing with reduced spacing on the right. Accuracy to the terrain surface increases from left to right, leading to fewer artifacts with harsh shadow edges. The bottom row shows hard shadows on the left, pure soft shadows and accompanying light leaks in the middle, and the combination on the right.*

4.4. Gradient calculation

The analytical gradient ∇f has numerous issues, which we outline in the supplementary material alongside its derivation. We thus calculate normals numerically using either forward or central differences depending on the desired quality. This is more expensive than the analytical calculation as it involves 3, respectively 6, additional evaluations of the implicit surface. Smoothness of the normal can be controlled by choosing a different smoothing radius $\sigma_{\boldsymbol{n}}$ for normal calculation, as well as by changing the spacing h used in the finite difference. Using $h = 0.5\sigma_{\boldsymbol{n}}$ leads to normal vectors that are more faithful to the terrain surface but less smooth with low $\sigma_{\boldsymbol{n}}$, while $h = \sigma_{\boldsymbol{n}}$ is smoother for low $\sigma_{\boldsymbol{n}}$. The differences are shown in Figure 4.

4.5. Adaptive step size

We propose an adaptive ray marching step size based on a heuristic H which is derived in the supplementary material:

$$H(\Delta\mathcal{V}) = \min\left(\frac{\Delta\mathcal{V}}{4\sigma^2}, \sqrt{3}\left(2\sigma - \left(8\sigma^3 - \Delta\mathcal{V}\right)^{\frac{1}{3}}\right)\right). \tag{5}$$

This serves as a replacement for a dynamic step size via an approximate signed distance field, as the analytical gradient is not suitable for that. The heuristic takes as input a target volume change, *i.e.* the volume change needed to reach the isosurface, and outputs a safe ray marching step size Δt.

The adaptive step size for point $\boldsymbol{p}$ is then calculated as

$$\Delta t(\boldsymbol{p}) = H(4\sigma^3 - i(\boldsymbol{p})), \tag{6}$$

where $4\sigma^3$ is the isosurface. Due to using a heuristic, the actual isosurface is never reached, which is why we use 99% of the isosurface when checking for intersections. The intersection point is then refined by performing one more step, *i.e.*

$$\boldsymbol{q} = \boldsymbol{p} + H(4\sigma^3 - i(\boldsymbol{p})) \cdot \boldsymbol{d}. \tag{7}$$

4.6. Shading

Previous work [NGWM24] used simple Phong Shading. Shadows, ambient occlusion or other advanced effects were not implemented. A pseudo-refraction effect was used, which intersects refracted rays with an infinite plane spanned by the sand level of the column that the ray hit, *i.e.* it ignores the actual scene geometry and is extremely cheap to do. The artifacts of this method were hidden by procedural normal mapping of the water surface and rudimentary caustics generated by animated noise textures.

In our method, we use a standard physically-based shading model

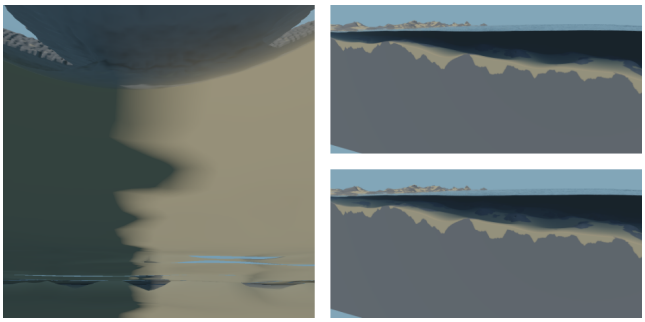


Figure 5: *Left:* `Arches` *scene rendered from underwater using our method with visible total internal reflection and Snell's window. Right: Comparison of cheap (top) and expensive (bottom) absorption model. The cheap model causes high absorption even in shallow water when viewed from the side.*

with image-based lighting [KG13]. Shadows, reflections and refractions are trivially supported using secondary rays. We implement perfect reflections and refractions, glossy ones only reflect the environment. A smoothing radius of one cell ($\sigma = \sigma_{\boldsymbol{n}} = l_c$) is used throughout the method. Secondary rays are computed at full resolution and every frame without any denoising, upscaling or temporal antialiasing, leaving room for performance improvements.

Inspired by soft shadow methods based on signed distance fields [TCKB22], we propose an approach compatible with our method. For a shadow ray $\boldsymbol{p} = \boldsymbol{o} + t \cdot \boldsymbol{d}$, we calculate the shadow as

$$\gamma_s = \min_t \left(\max \left(\frac{\frac{1}{2} - \frac{i(\boldsymbol{p})}{8\sigma^3}}{\min\left(\frac{\alpha_s \cdot t}{l_c}, \beta_s\right)}, 0 \right) \right). \tag{8}$$

This is only updated when traversal has reached the finest resolution where $i(\boldsymbol{p})$ is available, *i.e.* close to the surface. The maximum size of the penumbra is limited by the smoothing radius σ, which is why $\beta_s = 0.5$ is used to avoid visible cut-off. Growth of the penumbra with distance is controlled by $\alpha_s = 0.01$. This implementation suffers from light leaks due to the division by distance, which is fixed by setting $\gamma_s = 0$ if the shadow ray hits any geometry. See Figure 4 for a comparison between hard and soft shadows.

We propose a geometric ambient occlusion (AO) method by measuring the terrain volume in a box of radius σ_{ao}, offset by σ_{ao} along the surface normal. The AO value is the square of the percentage to which this box is filled by terrain. We use $\sigma_{ao} \in [1,2]m$. This means that AO gets more expensive with finer resolution (lower l_c) because more cells need to be iterated.

Refraction and reflection coefficients are calculated using the Schlick approximation [Maj21], which was extended to support correct underwater behavior such as total internal reflections [Lag13] (Figure 5, left). We did not implement full volumetric lighting. Light intensity is approximated using Beer's Law in two different modes. The cheap absorption mode uses twice the distance that the camera ray travelled through water, which assumes that all light is coming from the camera. The expensive version approximates the distance that the light travelled through the water by casting a ray directly upwards, which is added to the camera-based distance. This is only expensive for `SmoothVis` as a ray pointing upwards can be resolved inside a single cell without the full hierarchy traversal for `BoxVis`. In the smooth case, this simplification is not correct because the smoothing radius breaks the vertical isotropy of the water column, requiring ray casting. We use separate extinction coefficients for red, green and blue. For a comparison of both variants see Figure 5.

The material for shading is a weighted sum of the volume fractions of each material in relation to the volume of the union of materials the ray intersects. Previous work instead used the visible surface [PGGM09], which is more complicated to calculate.

5. Results

We evaluate our box and smooth visualizations quantitatively against the previous approach in [NGWM24], as well as against different versions of our own method. Four quality presets are used for this:

`Low` is the closest to the previous work (no secondary rays, no AO).
`Mid` enables cheap water absorption, single reflections and refractions, soft shadows and AO with a radius of $1m$.
`High` uses expensive water absorption and enables soft shadows and AO for secondary rays.
`Max` increases the AO radius to $2m$ and uses central differencing for normal calculation.

Evaluation is done on an RTX 4080 GPU using scenes provided by the previous work [NGWM24] (see Figures 1 and 2). The scenes are $256m \times 256m$ at various grid resolutions, with cell size l_c ranging from $1m$ down to $6.25cm$ at 4096^2 resolution.

5.1. Runtime analysis

First, we evaluated both types of visualization at $720p$ window resolution on the `Arches` (1024^2) and `Cave` (2048^2) scenes. Both scenes were warmed up by running the simulation until the arches, respectively cave, had formed. Results can be seen in Figure 6. For `BoxVis`, our new method is significantly faster than the inefficient previous approach for both scenes. With the feature-similar `low` preset, our method is $18\times$ faster in `Arches` and $40\times$ faster in `Cave`. The `max` preset, which uses geometric AO with a large radius and calculates up to 7 rays per pixel is still significantly faster at above 60 FPS including the simulation time, outperforming simple Phong Shading in the previous work. For `SmoothVis`, the performance of the previous method is comparable to our `low` preset, which in turn has similar performance to the `max` preset of `BoxVis`. The previous method again scales worse with terrain resolution and is outperformed by the `low` preset in `Cave`, where its performance is similar to our `mid` preset instead, which uses up to 4 rays per pixel. 60 FPS are possible with the `mid` preset in `Arches` and almost possible in `Cave` while running the simulation.

Our second test is done on the `RiversRain` scene at multiple terrain resolutions and compares $720p$ with $1080p$ window resolutions (Figure 7). The graphs use a linear scale for the x-axis with respect to the number of cells. We measured the frame time with the simulation paused, when the scene contains no water yet, and compare it with the frame time during simulation. For the `low` preset, the difference between these two times is the time required to run the simulation (up to $10ms$ at 4096^2) and create the data structure

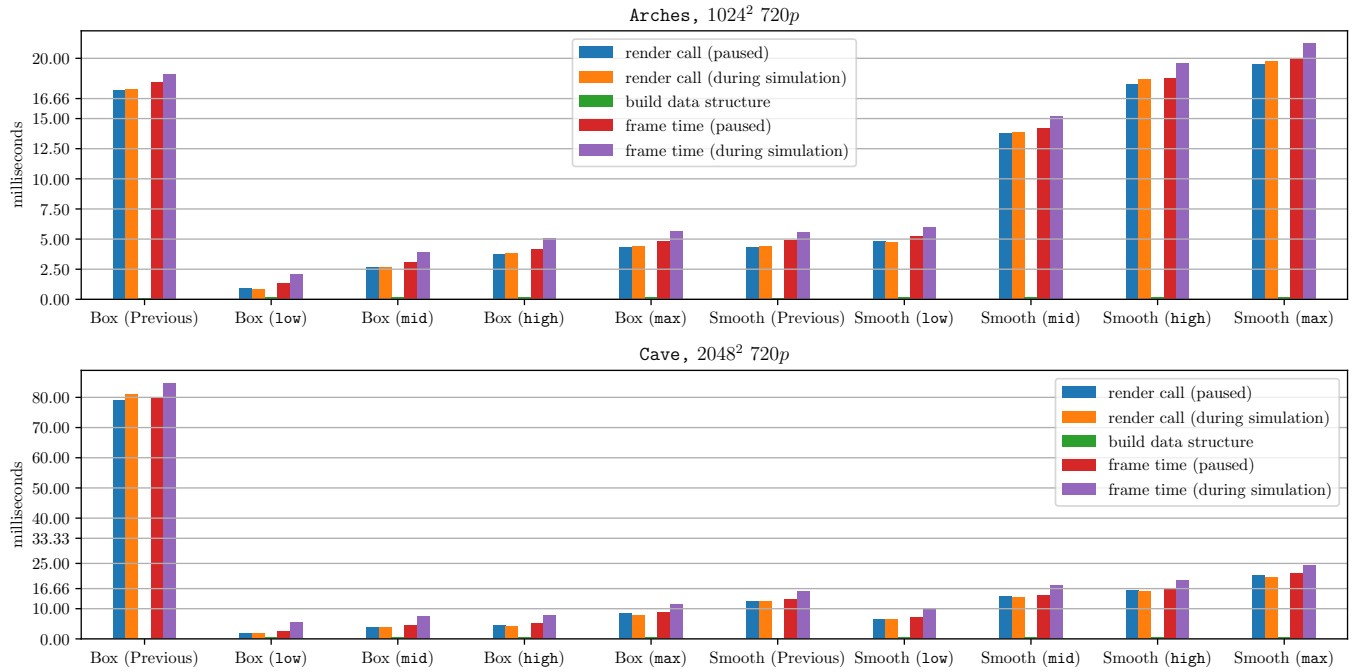


Figure 6: *Performance measurements for the* `Arches` *(1024², top) and* `Cave` *(2048², bottom) scenes at 1280 × 720 pixel resolution. We measure the previous methods for box and smooth visualization [NGWM24] alongside our method with four different quality presets. The pure render call that produces the image is measured alongside the full frame time, once with simulation paused and once measured during simulation, where simulation time is included. We also measure the data structure creation time, which in the case of the previous method refers to creating the draw list. Measurements were taken after 70000, respectively 30000 warm-up simulation iterations.*

(see Figure 8). The other presets additionally include the impact of the water shading. `Mid` and `high` presets thus have identical frame times without water. The `low` presets' frame time grows logarithmically with cell count, while the previous methods have linear growth. However, for dynamic scenes the data structure build time grows linearly (see Figure 8), but remained negligible relative to frame time up to 4096^2. The geometric AO used in other presets has quadratic growth with the inverse cell size. We halve the cell size when doubling the resolution for each axis, so growth is linear with cell count. With simulation paused, AO with a radius of $1m$ at $l_c = 6.25cm$ (4096^2) accounts for 20% at best up to 40% of frame time. At $2m$ radius (`max` preset), AO evaluates 64^2 grid cells per shaded intersection, requiring 60% of frame time. Except for the previous `SmoothVis` method, which is almost unaffected by window resolution, all methods show linear growth with pixel count.

Our method consistently outperforms the previous box and smooth visualizations at high terrain resolutions for both $720p$ and $1080p$ using `low` and `mid` presets. In particular, `low` presets scale extremely well with terrain resolution, with `BoxVis` at $1080p$ requiring only $3.5ms$ to render a 4096^2 terrain. Thus, If the simulation is paused, rendering alone can be done at 500 FPS ($720p$) or 280 FPS ($1080p$). This translates to over 60 FPS when including simulation and data structure build time each frame. The more complex smooth surface visualization, which extends the ray tracing method with ray marching using an implicit surface formulation [PGGM09], is slower, but still manages to reach over 40 FPS at $1080p$ or over 50 FPS at $720p$. Here, pure rendering is possible at 150 FPS ($720p$) or 80 FPS ($1080p$). With lower terrain resolutions, more advanced shading effects can be supported in real-time. For example, a box visualization at $1080p$ and 2048^2 grid resolution can be rendered at over 120 FPS with one reflection, one refraction, shadows and AO for primary and secondary rays. `High` and `max` presets outperform the previous `BoxVis` but can only beat the previous `SmoothVis` for low window resolutions, which is mainly due to AO. Without AO, all presets across both visualizations outperform the previous work at high terrain resolutions.

In our development environment (RTX 3070 and GTX 1660 Ti Max-Q), the simpler miss counter-based backtracking lead to 20–30% faster rendering compared to stack-based backtracking. However, the opposite was true on the newer, more powerful evaluation hardware, where stack-based backtracking is 20–30% faster. We thus omitted further analysis. Both approaches are clearly viable depending on the specific hardware and a broader range of GPUs would have to be examined to reach further conclusions.

Our last test involves viewpoint dependence of rendering performance, which was done in the `ChasmRain` scene (2048^2, 720p). For all methods, performance improves if less surface pixels are visible. However, the previous `SmoothVis` can only slightly benefit from this, while our methods show very large improvements with beneficial camera poses. In particular, our methods benefit from a

side facing, shallow viewpoint just as well as from zooming far out, while the previous `BoxVis` benefits to a lesser degree.

The supplementary material PDF contains further evaluation on additional scenes as well as auxiliary figures regarding backtracking and viewpoint dependence.

5.2. Visual comparison

Figure 1 shows the previous methods and our methods with all four presets side-by-side for a single scene. For `BoxVis`, the main differences between [NGWM24] and our `low` preset are water shading, direct lighting model and the image-based lighting used in our method. While the previous work at first glance appears to have refractions, these are a simple fragment shader trick and the method does not allow to actually see the underwater terrain. This is however possible using our method by enabling refractions. Another problem of the previous method are pixel-level artifacts caused by overlapping box geometries.

The addition of shadows, AO, refraction and reflection with our method has clear advantages, as they allow for better depth perception and highlight details that would otherwise be shaded in a uniform color, as can be seen in the previous method (Figure 1, lower left). Geometric AO is viewpoint-independent, temporally stable and noise-free. The soft shadow implementation works well, as all light leaks can be avoided by combining with hard shadows. While the penumbra size is limited, it does grow according to distance to the occluder as shown in Figure 4 and it avoids the harsh pixel edges of hard ray traced shadows at practically no cost to runtime. We can render the scene from underwater, capturing effects such as total internal reflection (Figure 5). Approximating the distance through the water to the light by sending a ray upwards works well enough for our use case, however adding features such as light rays or caustics using typical real-time rendering techniques could enhance visuals further.

Lastly, by using the implicit surface formulation from previous work [PGGM09], we drastically improve on the prior `SmoothVis` method [NGWM24], which is unable to render smooth terrain on side surfaces. This is a very common failure case for terrain with overhangs, as can be seen in Figure 1. The terrain thus looks like a combination of smooth terrain and boxes. These problems are entirely removed with our ray marching method, which renders terrain that has been smoothed in all three dimensions equally. Figure 2 showcases this in additional scenes from the previous work [NGWM24], rendered smooth using the `max` preset.

6. Conclusion and future work

In conclusion, we have successfully created two methods suitable for real-time visualization of fully dynamic MLHMs up to a grid resolution of 4096^2 and window resolution of 1920×1080, depending on the method and frame rate requirements. Our methods were evaluated on the 3D hydraulic erosion simulation presented in [NGWM24]. Their simulation is very fast, requiring only $10ms$ on an RTX 4080 to simulate a 4096^2 grid and less than $2ms$ for 2048^2 and lower. However, the previous work was unable to visualize their results in real-time at high resolutions, which is made possible using our work.

At 4096^2, our box visualization method based on ray tracing is faster than the simulation itself, which demonstrates that the proposed 2D hierarchical data structure and associated traversal algorithm scale even better with terrain resolution than the simulation itself. The data structure is fast to build every frame as we tailored it to the output of the simulation. This allows us to run ray traced/marched reflections, refractions, shadows and geometric AO every frame at the full resolution while still reaching real-time performance, depending on the exact visualization method and screen resolution. However, our visualization remains slower than the simulation at lower terrain resolutions because it is additionally dependent on screen resolution. In addition to vastly improving the performance of the previous naive visualization implementations that originally came with the simulation, the implicit surface formulation used for smooth visualization allows for much higher visual quality due to proper surface smoothing in all cases, unlike the previous method [NGWM24].

We introduced an AO method tailored to the MLHM data structure and a soft shadow method that is specific to ray marching using our acceleration data structure. While the first method can also be used in rasterization and produces visually stable results, its performance implications limit it to be used in situations where cell size l_c is similar to the AO radius. The soft shadow method does not introduce any noticeable performance cost over regular ray traced shadows, and while not physically accurate, it is a clear improvement over hard shadow edges, producing plausible results.

For future work, the geometric AO method should be replaced with standard methods used in real-time graphics, as its performance impact is unreasonably large for finely detailed terrain, so much so that it dominates the runtime complexity. On the other hand, ray tracing and ray marching performance using our method is already very good, but it would be possible to boost performance significantly by using real-time ray tracing techniques from video games. This includes using a lower resolution for secondary rays, (temporal) upscaling, denoising, ray reconstruction or screen-space alternatives. As we can already reach good frame rates while using the full resolution every frame, this is a logical next step to further lower the visualization overhead compared to simulation time. This would also open up the potential to have ray traced glossy reflections and global illumination.

Lastly, the scalability of our method, *i.e.* resolutions above 4096^2, could be improved. While this would also require further work on the simulation, the data structure creation for our visualization method starts to become an issue due to scaling linearly with cell size, instead of logarithmically. Linear time complexity is inherent to the algorithm, but further code optimizations could increase performance enough to delay the problem to much larger resolutions. Lastly, the main restriction of our method lies in its requirement for low vertical complexity. This matches the specific use case of terrain rendering but may not apply to other areas of research using this type of data structure. Thus, an extension of the acceleration data structure to additionally consider the third dimension would be useful, *i.e.* by introducing links to the contained columns of each AABB.

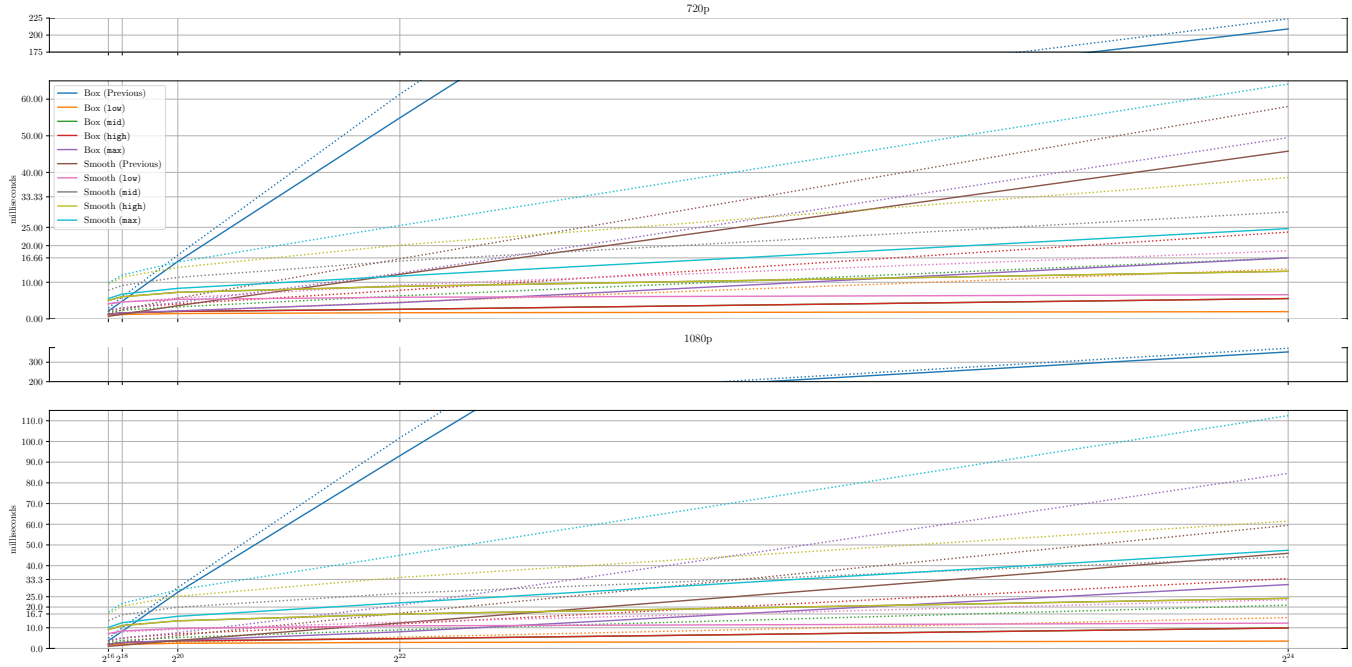


Figure 7: *Frame time with simulation paused in the* `RiversRain` *scene, where the scene contains no water yet (filled lines) and frame time during simulation where the terrain fills with water (dotted lines, includes simulation time). The x-axis is linear with cell count.*

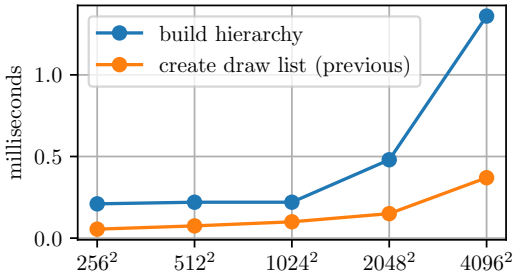


Figure 8: *Build time of our hierarchical data structure versus the draw list from the previous method [NGWM24] in* `RiversRain`*. The x-axis is logarithmic with respect to cell count.*

References

[BB97] BLOOMENTHAL, JULES and BAJAJ, CHANDRAJIT. *Introduction to implicit surfaces*. Morgan Kaufmann, 1997 3.

[BMPG11] BORGEAT, LOUIS, MASSICOTTE, PHILIPPE, POIRIER, GUILLAUME, and GODIN, GUY. "Layered Surface Fluid Simulation for Surgical Training". *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2011*. Ed. by FICHTINGER, GABOR, MARTEL, ANNE, and PETERS, TERRY. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, 323–330. ISBN: 978-3-642-23623-5 1, 2.

[De 00] DE BOER, WILLEM H. "Fast terrain rendering using geometrical mipmapping". *Unpublished paper* (2000), 7. URL: https://www.flipcode.com/archives/geomipmaps.pdf 2.

[DGV*18] DAGENAIS, F., GUZMÁN, J., VERVONDEL, V., et al. "Real-time virtual pipes simulation and modeling for small-scale shallow water". *Proceedings of the 14th Workshop on Virtual Reality Interactions and Physical Simulations*. VRIPHYS '18. Delft, The Netherlands: Eurographics Association, 2018, 45–54 1, 2.

[DKW09] DICK, CHRISTIAN, KRÜGER, JENS, and WESTERMANN, RÜDIGER. "GPU Ray-Casting for Scalable Terrain Rendering". *Eurographics 2009 - Areas Papers*. Ed. by EBERT, D. and KRÜGER, J. The Eurographics Association, 2009. DOI: 10.2312/ega.20091007 2.

[DMHS15] DÜBEL, STEVE, MIDDENDORF, LARS, HAUBELT, CHRISTIAN, and SCHUMANN, H. "A Flexible Architecture for Ray Tracing Terrain Heightfields". Aug. 2015 2.

[Don05] DONNELLY, WILLIAM. "Per-pixel displacement mapping with distance functions". *GPU gems* 2.22 (2005), 3 2.

[Dum06] DUMMER, JONATHAN. "Cone step mapping: An iterative ray-heightfield intersection algorithm". 2.3 (2006), 4. URL: http://www.lonesock.net/files/ConeStepMapping.pdf 2.

[DVG*18] DAGENAIS, FRANÇOIS, VERVONDEL, VALENTIN, GUZMÁN, JULIÁN E., et al. "Extended virtual pipes for the stable and real-time simulation of small-scale shallow water". *Computers & Graphics* 76 (2018), 84–95. ISSN: 0097-8493. DOI: 10.1016/j.cag.2018.08.005 1, 2.

[FTI86] FUJIMOTO, AKIRA, TANAKA, TAKAYUKI, and IWATA, KANSEI. "ARTS: Accelerated Ray-Tracing System". *IEEE Computer Graphics and Applications* 6.4 (1986), 16–26. DOI: 10.1109/MCG.1986.276715 4.

[GPF10] GALLUP, DAVID, POLLEFEYS, MARC, and FRAHM, JAN-MICHAEL. "3D Reconstruction Using an n-Layer Heightmap". *Pattern Recognition*. Ed. by GOESELE, MICHAEL, ROTH, STEFAN, KUIJPER, ARJAN, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 1–10. ISBN: 978-3-642-15986-2 1, 2.

[Kel13] KELLOMÄKI, TIMO. "Interaction with dynamic large bodies in efficient, real-time water simulation". *Journal of WSCG* 21.2 (2013), 117–126. ISSN: 1213–6964. URL: http://hdl.handle.net/11025/6973 2.

[KG13] KARIS, BRIAN and GAMES, EPIC. "Real shading in unreal engine 4". *Proc. Physically Based Shading Theory Practice* 4.3 (2013), 1 6.

[KJCH15] KANG, HYEONGYEOP, JANG, HANYOUNG, CHO, CHANG-SIK, and HAN, JUNGHYUN. "Multi-resolution terrain rendering with GPU tessellation". *The Visual Computer* 31.4 (Apr. 2015), 455–469. ISSN: 1432-2315. DOI: 10.1007/s00371-014-0941-6 2.

[Lag13] LAGARDE, SÉBASTIEN. *Memo on Fresnel equations*. Apr. 2013. URL: https://seblagarde.wordpress.com/2013/04/29/memo-on-fresnel-equations/ 6.

[LH04] LOSASSO, FRANK and HOPPE, HUGUES. "Geometry clipmaps: terrain rendering using nested regular grids". *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. Los Angeles, California: Association for Computing Machinery, 2004, 769–776. ISBN: 9781450378239. DOI: 10.1145/1186562.1015799 2.

[LK11] LAMBERS, MARTIN and KOLB, ANDREAS. "Dynamic terrain rendering". *3D Research* 1.4 (Nov. 2011), 1. ISSN: 2092-6731. DOI: 10.1007/3DRes.04(2010)01 2.

[LLS16] LEE, EUN-SEOK, LEE, JIN-HEE, and SHIN, BYEONG-SEOK. "A bimodal empty space skipping of ray casting for terrain data". *The Journal of Supercomputing* 72.7 (July 2016), 2579–2593. ISSN: 1573-0484. DOI: 10.1007/s11227-015-1522-9 2.

[LP01] LINDSTROM, P. and PASCUCCI, V. "Visualization of large terrains made easy". *Proceedings Visualization, 2001. VIS '01.* 2001, 363–574. DOI: 10.1109/VISUAL.2001.964533 2.

[LZW*21] LI, SHI, ZHENG, CHUANKUN, WANG, RUI, et al. "Multi-resolution terrain rendering using summed-area tables". *Computers & Graphics* 95 (2021), 130–140. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2021.02.003 2.

[Maj21] MAJERCIK, ZANDER. "The Schlick Fresnel Approximation". *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*. Ed. by MARRS, ADAM, SHIRLEY, PETER, and WALD, INGO. Berkeley, CA: Apress, 2021, 109–114. ISBN: 978-1-4842-7185-8. DOI: 10.1007/978-1-4842-7185-8_9 6.

[NG25] NILLES, ALEXANDER MAXIMILIAN and GÜNTHER, LARS. *CUDA 3D Hydraulic Erosion Simulation with Layered Stacks*. https://github.com/Clocktown/CUDA-3D-Hydraulic-Erosion-Simulation-with-Layered-Stacks/tree/VMV2025. 2025 2.

[NGWM24] NILLES, ALEXANDER MAXIMILIAN, GÜNTHER, LARS, WAGNER, TOBIAS, and MÜLLER, STEFAN. "3D Real-Time Hydraulic Erosion Simulation using Multi-Layered Heightmaps". *Vision, Modeling, and Visualization*. Ed. by LINSEN, LARS and THIES, JUSTUS. The Eurographics Association, 2024. ISBN: 978-3-03868-247-9. DOI: 10.2312/vmv.20241211 1–9.

[OBM00] OLIVEIRA, MANUEL M., BISHOP, GARY, and MCALLISTER, DAVID. "Relief texture mapping". *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. USA: ACM Press/Addison-Wesley Publishing Co., 2000, 359–368. ISBN: 1581132085. DOI: 10.1145/344779.344947 2.

[OH95] O'BRIEN, J.F. and HODGINS, J.K. "Dynamic simulation of splashing fluids". *Proceedings Computer Animation'95*. 1995, 198–205. DOI: 10.1109/CA.1995.393532 2.

[PGGM09] PEYTAVIE, A., GALIN, E., GROSJEAN, J., and MERILLOU, S. "Arches: a Framework for Modeling Complex Terrains". *Computer Graphics Forum* 28.2 (2009), 457–467. DOI: https://doi.org/10.1111/j.1467-8659.2009.01385.x 1–3, 6–8.

[PO06] POLICARPO, FABIO and OLIVEIRA, MANUEL M. "Relief mapping of non-height-field surface details". *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. I3D '06. Redwood City, California: Association for Computing Machinery, 2006, 55–62. ISBN: 159593295X. DOI: 10.1145/1111411.1111422 2.

[PO07] POLICARPO, FABIO and OLIVEIRA, MANUEL M. "Relaxed cone stepping for relief mapping". *GPU gems* 3.3 (2007), 409–428 2.

[ŠBBK08] ŠT'AVA, ONDŘEJ, BENEŠ, BEDŘICH, BRISBIN, MATTHEW, and KŘIVÁNEK, JAROSLAV. "Interactive terrain modeling using hydraulic erosion". *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '08. Dublin, Ireland: Eurographics Association, 2008, 201–210. ISBN: 9783905674101 2.

[SCC*10] STUETZLE, CHRISTOPHER S, CHEN, ZHONGXIAN, CUTLER, BARBARA, et al. "Computer simulations and physical modelling of erosion". *7th International Conference on Physical Modelling in Geotechnics (ICPMG 2010), Zurich*. 2010, 20–24. ISBN: 9780429088094 1, 2.

[SHVS18] SARKAR, KRIPASINDHU, HAMPIHOLI, BASAVARAJ, VARANASI, KIRAN, and STRICKER, DIDIER. "Learning 3D Shapes as Multi-Layered Height-maps using 2D Convolutional Networks". *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018 1, 2.

[TCKB22] TAN, YU, CHUA, NICHOLAS, KOH, CLARENCE, and BHOJAN, ANAND. "RTSDF: Real-time Signed Distance Fields for Soft Shadow Approximation in Games". *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS - Science and Technology Publications, 2022, 302–309. DOI: 10.5220/0010996200003124 6.

[TIS08] TEVS, ART, IHRKE, IVO, and SEIDEL, HANS-PETER. "Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering". *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*. I3D '08. Redwood City, California: Association for Computing Machinery, 2008, 183–190. ISBN: 9781595939838. DOI: 10.1145/1342250.1342279 2.

[ZWH*21] ZHANG, LEI, WANG, PING, HUANG, CHENGYI, et al. "A Method of Optimizing Terrain Rendering Using Digital Terrain Analysis". *ISPRS International Journal of Geo-Information* 10.10 (2021). ISSN: 2220-9964. DOI: 10.3390/ijgi10100666 2.