


3D Real-Time Hydraulic Erosion Simulation using Multi-Layered Heightmaps

Alexander Maximilian Nilles¹  and Lars Günther¹ and Tobias Wagner^{1,2} and Stefan Müller¹

¹University of Koblenz, Institute for Computational Visualistics, Germany

²TOMRA Sorting GmbH, Germany

Abstract

We present a novel method for real-time 3D hydraulic erosion simulation of large-scale terrain. Existing hydraulic erosion methods based on heightmaps and the virtual pipes method are extended to multi-layered heightmaps that can represent more complex 3D features. Our new method for horizontal erosion is able to create overhangs, arches and to some extent caves by allowing water to erode horizontally adjacent bedrock, eventually splitting a column into two new columns. Additionally, we developed an iterative method for bedrock support check that efficiently prevents floating terrain and unrealistic overhangs by propagating bedrock connectivity while incorporating the weight of each column. We implement our method in CUDA, using only features that are also available in standard compute shaders. On a RTX 3070, the resulting method runs at approximately 6ms and 23ms per simulation step for resolutions of 2048^2 and 4096^2 , respectively.

CCS Concepts

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

1. Introduction

Terrain modeling is essential in computer graphics for applications like games, movies, and simulators. Real-world terrain forms through complex processes such as tectonics, hydraulic and aeolian erosion, and thermal weathering, making manual modeling challenging without specialized tools. Additionally, real-world terrain features complex 3D structures like overhangs, arches, and caves, beyond simple 2.5D heightmaps.

We focus on hydraulic erosion, a well-researched topic in computer graphics, with fast heightmap-based methods like MEI et al. [MDH07]. However, 3D hydraulic erosion has received less attention, with some exceptions like the *Arches* framework [PGGM09] and recent particle-driven methods [HMFF24].

We propose a novel 3D erosion approach, combining hydraulic erosion via the virtual pipes method with a 3D, multi-layered heightmap, resulting in a fast real-time GPU implementation. Our method generates detailed overhangs, arches, and given the right circumstances even caves. Unlike voxel grid methods, we achieve high resolutions and floating-point precision for height values. We support two ground materials: bedrock and sand. Bedrock, the structural support, collapses if unstable. Sand is stabilized to its talus angle and can be dissolved, deposited and softened by water. Bedrock can also be eroded by water, with our method enabling horizontal erosion, forming overhangs by splitting bedrock columns.

The paper is structured as follows: Section 2 covers related work,

Section 3 details our method, Section 4 provides implementation specifics, Section 5 presents results, and Section 6 concludes with future work ideas.

The full source code, scenes with their exact parameters, additional screenshots and videos are provided in the supplementary material and available open source on GitHub [NG24].

2. Related work

This section introduces relevant related work, focusing on hydraulic erosion simulation and the virtual pipes method. For a comprehensive overview of terrain modeling, see GALIN et al. [GGP*19]. For water modeling and fluid simulation methods, we refer to IGLESIAS [Igl04] and WANG et al. [WXL*24].

2.1. Virtual pipes method

The virtual pipes method, proposed in O'BRIEN and HODGINS [OH95], simplifies previous shallow water models [KM90]. By assuming vertical isotropy, it allowed fast simulations by discretizing fluid into columns connected by virtual pipes. Improvements include dividing columns vertically [MY97] and modeling waterfalls, rivers, and rapids [HW04]. Later, the method was brought to the GPU for real-time simulations [MFC06; MDH07].

BORGEAT et al. [BMPG11] introduced a multi-layer heightmap representation for fluid flow on geometry with overhangs. To improve performance, outflowing water could only go to the lowest

connected column. A similar approach by KELLOMÄKI [Kel13] supported only one additional layer for blocking geometry without the previous limitation on outflowing water. DAGENAIS et al. [DGV*18] extended this to multiple blockers and added a viscosity model. DAGENAIS et al. [DVG*18] improved this further by adding support for flooded passages by allowing pipe connections to non-neighboring cells.

2.2. 2D Hydraulic Erosion

MEI et al. [MDH07] proposed a real-time GPU hydraulic erosion simulation on heightmaps, integrating different early ideas from MUSGRAVE et al. [MKM89], CHIBA et al. [CMF98], and SUTHERLAND and KEYSER [SK06], and utilizing the virtual pipes method for water simulation. In contrast, BENES [Ben07] used the shallow-water equations to model erosion via a viscous regolith layer. ŠT'AVA et al. [ŠBBK08] combined these approaches, using the virtual pipes method and erosion/deposition model from MEI et al. [MDH07] while also simulating a regolith layer similar to BENES [Ben07]. They extended the method with multiple material layers, such as rock, soil, and sand, transformed by erosion and aging processes. KRIŠTOF et al. [KBKŠ09] later proposed 3D smoothed particle hydrodynamics (SPH) for hydraulic erosion, resulting in more accurate water behavior, though it was limited to heightmaps and could not generate overhangs.

2.3. Other notable 2D methods

Other methods focus on different erosion types, such as aeolian erosion for fast desert landscape simulation in PARIS et al. [PPG*19], later improved for real-time performance on the GPU [TK23; NGM24]. Recently, TZATHAS et al. [TGSC24] focused on procedural terrain generation using the stream power law and tectonic uplift for fast, consistent results.

2.4. 3D methods

BENEŠ et al. [BTHB06] and WOJTAN et al. [WCMT07] introduced 3D voxel grid-based hydraulic erosion using the Navier-Stokes equations, suitable for small scenes but slow. BEARDALL et al. [BFO*07] proposed interactive aeolian erosion for 3D goblin shapes using voxel grids. The *Arches* framework [PGGM09] achieves high resolutions with multi-layered heightmaps and an implicit surface representation, used for terrain sculpting and local erosion tools for artists. Recently, HARTLEY et al. [HMF24] proposed a particle-based erosion method for various terrain representations, supporting hydraulic, aeolian, and thermal erosion. However, it only supports a single material and cannot collapse unstable overhangs or floating terrain after erosion.

3. Our method

We extend real-time 2D hydraulic erosion methods using heightmaps to 3D multi-layered heightmaps, enabling 3D erosion and overhang formation while maintaining large horizontal resolutions and full floating-point height values. The virtual pipes method, known for high performance and GPU parallelism, is adopted here.

We refer to the erosion in 2D heightmap methods as *vertical*

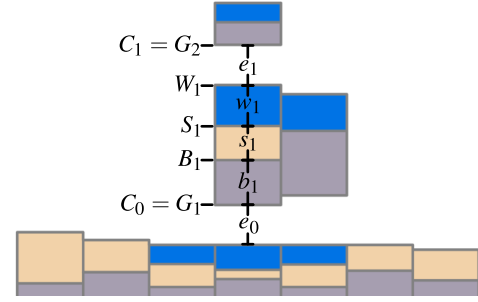


Figure 1: A diagram of our multi-layered heightmap structure with bedrock (gray), sand (brown) and water (blue), showing 7 cells horizontally. The center cell has 3 vertically stacked columns, annotated according to our notation in Section 3.1. Cell indices are omitted for clarity.

erosion, eroding downwards into a column. A 3D extension requires additional upward vertical erosion, eroding into columns from below. Sediment and sand transport need to be adapted to the new data structure. On its own, vertical erosion allows erosion in a pre-made 3D environment but cannot create new overhangs. To achieve this, we develop *horizontal erosion*, eroding into and splitting neighboring columns to create overhangs. Finally, to address degenerate rock formations, such as floating columns, we develop a *support check* to collapse unstable columns.

3.1. Terrain definition

The terrain is an $N_x \times N_y$ 2D grid of $l_c \times l_c$ sized cells $\mathbf{i} = (i, j)$. Each cell has up to N_z columns $\mathbf{k} = (i, j, k)$ layered vertically. Cells contain at least one bottom column extending infinitely downwards. We used $N_z = 8$ in our examples.

Each column stores the absolute bedrock height $B_{\mathbf{k}}$, relative sand height $s_{\mathbf{k}}$, relative water height $w_{\mathbf{k}}$, and the absolute ceiling height $C_{\mathbf{k}}$, which also serves as the floor $G_{(i,j,k+1)}$ of the column above. The lowest column's floor is $-\infty$ and the highest column's ceiling is $+\infty$. For each column, sand is placed on top of the bedrock and water on top of sand. Uppercase letters denote absolute quantities, while lowercase denotes relative quantities, transformed as follows:

$$S_{\mathbf{k}} = B_{\mathbf{k}} + s_{\mathbf{k}} \quad (1)$$

$$W_{\mathbf{k}} = S_{\mathbf{k}} + w_{\mathbf{k}} \quad (2)$$

$$b_{\mathbf{k}} = B_{\mathbf{k}} - G_{\mathbf{k}} \quad (3)$$

The relative height of the empty space above a column is:

$$e_{\mathbf{k}} = C_{\mathbf{k}} - W_{\mathbf{k}}. \quad (4)$$

Figure 1 illustrates our data structure.

3.2. Water and sand transport

Our method transports water, sediment (dissolved or dispersed sand), and sand (via slippage when exceeding the angle of repose) between adjacent columns using the virtual pipes framework. Originally designed for water, we extend this to handle sediment and sand.

Unlike 2D methods using backward Euler advection for sediment, we transport sediment via pipes due to linear interpolation being challenging on multi-layered heightmaps.

Each column connects via a pipe to the lowest eligible column in its 4-neighborhood, akin to BORGEAT et al. [BMPG11]. A neighboring column \mathbf{k}' is eligible if

$$B_{\mathbf{k}} < C_{\mathbf{k}'} \text{ and } e_{\mathbf{k}'} > 0. \quad (5)$$

This ensures fluid does not flow through bedrock and prevents filled columns from blocking flow to higher columns. Limiting to one pipe per neighboring cell balances performance. Columns are denoted as $L, U, R,$ and D for left, up, right, and down neighbors, respectively.

The virtual pipes method transports fluid based on static pressure. Each column computes positive flux through its pipes using acceleration:

$$\dot{f}(\mathbf{k}, \mathbf{k}') = \frac{g \cdot A \cdot (\min(W_{\mathbf{k}}, C_{\mathbf{k}'} - \max(W_{\mathbf{k}'}, S_{\mathbf{k}}))}{l_c}, \quad (6)$$

with gravity $g = -9.81 \frac{m}{s^2}$, cross-sectional pipe area $A = l_c^2$, and pipe length l_c . The maximum limits flow for steep cliffs as in KELLOMÄKI [Kel13], while the minimum prevents overfilling of columns. Then, the outflow flux for the next simulation step is computed as

$$\bar{f}^{t+\Delta t}(\mathbf{k}, \mathbf{k}') = K_e \cdot \max((1 - \delta_d \Delta t) f^t(\mathbf{k}, \mathbf{k}') - \Delta t \cdot \dot{f}(\mathbf{k}, \mathbf{k}'), 0), \quad (7)$$

with dampening factor $\delta_d = 0.01$ and K_e scaling based on available space:

$$K_e = \min(l_c^2 \cdot \frac{e_{\mathbf{k}'}}{\bar{f} \cdot \Delta t}, 1). \quad (8)$$

To prevent negative water heights from excessive outfluxes, we limit the final flux value, following MEI et al. [MDH07]:

$$K_w = \min\left(\frac{w_{\mathbf{k}} \cdot l_c^2}{\Delta t (\bar{f}_L + \bar{f}_U + \bar{f}_R + \bar{f}_D)}, 1\right), \quad (9)$$

$$f(\mathbf{k}, \mathbf{k}') = K_w \cdot \bar{f}(\mathbf{k}, \mathbf{k}'). \quad (10)$$

After calculating outflow fluxes, each column adjusts water volume based on inflow from all neighboring columns and its own outflow:

$$\Delta V(\mathbf{k}) = \Delta t (\sum f_{out} - \sum f_{in}) \quad (11)$$

Water height is updated as

$$w_{\mathbf{k}}^{t+\Delta t} = w_{\mathbf{k}}^t - \frac{\Delta V(\mathbf{k})}{l_c^2}, \quad (12)$$

which we clamp to prevent overfilling due to inflow potentially originating from multiple columns per neighbor cell, in contrast to outflow.

To optimize memory and computations, we utilize existing fluid flux values for sediment transport. We calculate a sediment flux scale:

$$K_r = \min\left(\frac{r_{\mathbf{k}} \cdot l_c^2}{\Delta t (f_L + f_U + f_R + f_D)}, 1\right), \quad (13)$$

where $r_{\mathbf{k}}$ denotes the dissolved sediment in a column. K_r functions similarly to K_e . Sediment values are updated as in Equations (11) and (12), scaled by the K_r of the source column.

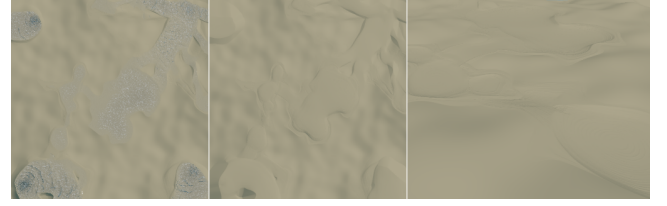


Figure 2: *The RiversSource scene, featuring purely vertical erosion of sand using four water sources. After a while, we let the water evaporate to show sedimentation patterns and terrain surface below the water. Despite our simplified approach to regolith, the terrain is appropriately smoothed and interesting sedimentation patterns are visible.*

Sand transport is handled similarly. Positive slippage values are computed based on the angle of repose, using the same neighboring columns determined for outflow flux:

$$\bar{a}(\mathbf{k}, \mathbf{k}') = \min\left(\max\left(\frac{S_{\mathbf{k}} - S_{\mathbf{k}'} - \tan(\theta) \cdot l_c}{8}, 0\right), \frac{e_{\mathbf{k}}}{4}\right). \quad (14)$$

Divisions by 8 and 4 prevent oscillations and overfilling. The angle of repose θ is linearly interpolated between 0° and 30° based on the average water depth of the columns, with the lowest angle at a depth of one. This mimics a regolith layer at no cost, unlike ŠT'AVA et al. [ŠBBK08].

Similar to water and sediment, slippage values are scaled to prevent negative sand heights:

$$K_a = \min\left(\frac{s_{\mathbf{k}}}{\bar{a}_L + \bar{a}_U + \bar{a}_R + \bar{a}_D}, 1\right), \quad (15)$$

$$a(\mathbf{k}, \mathbf{k}') = K_a \cdot \bar{a}(\mathbf{k}, \mathbf{k}'). \quad (16)$$

Sand heights are updated accordingly:

$$s_{\mathbf{k}}^{t+\Delta t} = s_{\mathbf{k}}^t - (\sum a_{out} - \sum a_{in}), \quad (17)$$

which we clamp to prevent overfilling.

We implemented reflecting and outflow boundaries. Reflecting boundaries are handled by omitting pipes to neighboring cells outside the grid, creating an invisible wall. This prevents any outflow to or inflow from those neighbors. For outflow boundaries, all outflow quantities are calculated with the bedrock height of each column clamped to the border, while sand, water, and sediment outside the grid are set to zero. This setup treats the border as a sink, removing matter from the scene. The boundary mode can be set for slippage independent of the one used for water and sediment.

3.3. Vertical erosion

Vertical erosion in our method involves the dissolution of sand into sediment and the deposition of sediment as sand as in MEI et al. [MDH07]. Bedrock erosion is also considered, serving as a second material layer. ŠT'AVA et al. [ŠBBK08] employed a similar dissolution model across all material layers. Our approach instead converts bedrock into sand, which can then be dissolved into sediment later.

MEI et al. [MDH07] define sediment transport capacity κ using

a sediment capacity constant κ_C , local tilt angle α , and horizontal velocity \mathbf{v} as

$$\kappa(\mathbf{k}) = \kappa_C \cdot \sin(\alpha_{\mathbf{k}}) \cdot \|\mathbf{v}_{\mathbf{k}}\|. \quad (18)$$

Velocity is computed based on the flux values of our pipe model:

$$\mathbf{v}_{\mathbf{k}} = \frac{1}{2 \cdot l_c^2} \begin{pmatrix} f_R - f_L \\ f_U - f_D \end{pmatrix}. \quad (19)$$

The local tilt angle is ambiguous in our data structure. We determine it using central differencing of the four neighbor columns found during outflow flux computation, utilizing absolute sand heights.

There are several issues with Equation (18). As noted by MEI et al. [MDH07], perfectly flat terrain cannot be eroded using this equation, prompting the use of a remapping of $\sin(\alpha)$ with a user-specified minimum threshold, which we also implement. Another concern is the vertical isotropy, which affects accuracy, especially for erosion at the bottom of deep water columns. MOULD and YANG [MY97] already address this, but introducing more columns and pipes is costly. Instead, we multiply Equation (18) by a factor that reduces capacity with water depth:

$$1 - \tanh\left(w_{\mathbf{k}} \cdot \frac{2}{w_{\max}}\right) \quad (20)$$

where w_{\max} is a user-defined soft maximum water depth.

In our multi-layered method, velocities in flooded passages are often zero. To resolve this without extending the method as in DAGENAIS et al. [DVG*18], we apply the same remapping used for the tilt angle to $\|\mathbf{v}\|$. We additionally interpolate to zero when water height is below 1 mm to avoid erosion without the presence of water.

Additionally, the capacity equation's encouragement of steep, thin, and deep ridges is problematic for bedrock erosion which does not utilize slippage. We mitigate this by interpolating $\sin(\alpha)$ towards zero for values in $[0.6, 1]$ using smooth Hermite interpolation. This adjustment complements horizontal erosion, which specifically targets steep angles.

With our modified capacity model, dissolution and deposition of sand happens as in MEI et al. [MDH07]. Given a user-defined sand dissolving constant κ_s and sediment deposition constant κ_d , sand dissolves when there is less sediment than capacity, otherwise, sediment is deposited. For bedrock, we use a bedrock erosion constant κ_b without the capacity model. Bedrock erosion is scaled by velocity, tilt angle, and water depth. We linearly interpolate bedrock erosion towards zero as sand height in the cell exceeds 1cm.

Our multi-layered model allows erosion of the bedrock above a column when $W_{\mathbf{k}} \approx C_{\mathbf{k}}$. This increases the ceiling and adds the removed bedrock to the sand layer. This effect is controlled by κ_b and scaled by velocity. No tilt angle is computed for the underside of bedrock columns, instead we use the user-defined minimum tilt angle. Erosion strength linearly interpolates based on the space between water surface and ceiling, fully active when no space is available and inactive when $e_{\mathbf{k}} \geq \kappa_r$ (typically set to $1 - 2m$).

3.4. Horizontal erosion

Vertical erosion alone cannot create new overhangs. We propose a horizontal erosion method that splits bedrock columns to form new

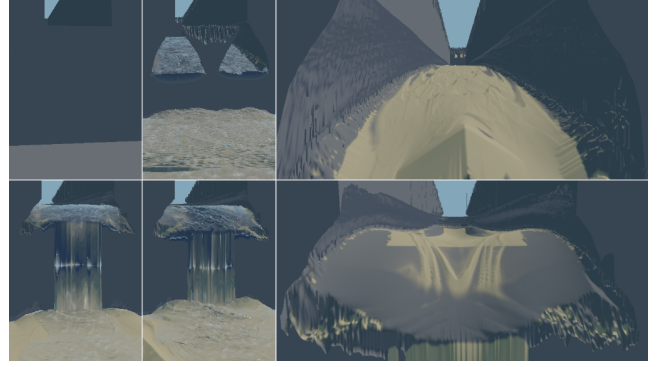


Figure 3: Top row: The *ChasmRain* scene shows a chasm eroded by rain, forming overhangs and a triangular ridge that collapses after further erosion, leaving flow-like sedimentation patterns after evaporation. Bottom row: The *ChasmSource* scene features a water source inside the chasm and no rain, yielding very different results. The initial chasm is 126 cells wide in a 2048^2 scene.

columns within a cell. This process is driven by water in neighboring cells, rather than within the same cell or column as in vertical erosion. Managing the number of columns in a cell is crucial, as excessive splitting can quickly reach the limit N_z without generating interesting terrain. To mitigate this, we structure our method in a way that minimizes unnecessary splits and we provide mechanisms to recover from unfavorable splits.

We iterate over columns \mathbf{k}' of the four neighboring cells of column \mathbf{k} . The overlap bounds between bedrock in \mathbf{k} and water in \mathbf{k}' are calculated as

$$\sigma(\mathbf{k}, \mathbf{k}') = \begin{pmatrix} \max(G_{\mathbf{k}}, S_{\mathbf{k}'}) \\ \min(B_{\mathbf{k}}, W_{\mathbf{k}'}) \end{pmatrix}. \quad (21)$$

Erosion strength is computed if $\sigma.y > \sigma.x$ as

$$\Delta D(\mathbf{k}, \mathbf{k}') = \kappa_b A_{\sigma} \cdot \sin^*(\beta(\mathbf{k}, \mathbf{k}')) \cdot \|\mathbf{v}_{\mathbf{k}'}\| \cdot \kappa_n(\mathbf{k}, \mathbf{k}'), \quad (22)$$

where κ_b is a user-defined constant, $A_{\sigma} = l_c(\sigma.y - \sigma.x)$ is the overlap area, and $\beta(\mathbf{k}, \mathbf{k}')$ is the tilt angle:

$$\beta(\mathbf{k}, \mathbf{k}') = \begin{cases} \frac{\pi}{2} & F_{\mathbf{k}} > S_{\mathbf{k}'} \\ \text{atan2}(\min(B_{\mathbf{k}}, C_{\mathbf{k}'}) - S_{\mathbf{k}'}, l_c) & \text{otherwise} \end{cases}. \quad (23)$$

For columns floating with respect to their neighbor, we use the maximum angle. The minimum ensures appropriate clamping at overhangs. \sin^* denotes a sine remapping using smooth Hermite interpolation from 0 to 1 for values in $[0.9, 1]$, targeting steep slopes only. κ_n weights erosion based on the dot product between the vector towards the neighbor and the normalized velocity, calculated as

$$\kappa_n(\mathbf{k}, \mathbf{k}') = \left(\frac{1}{2} - \frac{\langle \mathbf{i}' - \mathbf{i}, \frac{\mathbf{v}_{\mathbf{k}'}}{\|\mathbf{v}_{\mathbf{k}'}} \rangle}{2} \right). \quad (24)$$

ΔD is computed for all neighboring columns and we determine the highest damage ΔD_{\max} and its overlap σ_{\max} , discarding the other damage values to simplify the algorithm. This updates the height

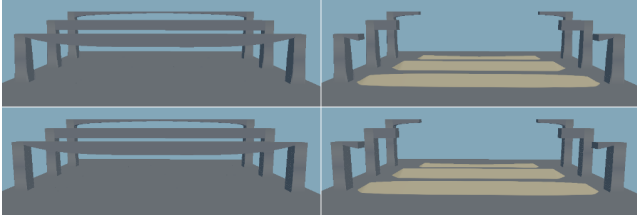


Figure 4: The Bridges scene at resolutions 256^2 (top) and 1024^2 (bottom). Behavior is consistent with resolution and bridge type.

damage $D_{\mathbf{k}}$ of the column as

$$D_{\mathbf{k}}^{t+\Delta t} = D_{\mathbf{k}}^t + \frac{\Delta t}{l_c^2} \Delta D_{\max}. \quad (25)$$

Height damage accumulates over time and triggers column splitting at $\sigma_{\max,y}$ (i.e. the water surface or top of the bedrock if submerged) once surpassing a split threshold D_{split} . Post-split, the lower column has ceiling $\sigma_{\max,y}$ and consists solely of bedrock. D_{split} is subtracted from its bedrock height and added as sand, available for subsequent erosion. The upper column resets its damage to 0, while the lower retains its pre-split damage, minus D_{split} . Splitting only occurs if the cell has fewer than N_z columns. Columns with no bedrock can be generated, which are removed by our support check.

To reduce splits further, we introduce damage recovery. $D_{\mathbf{k}}$ decreases by $\kappa_r \cdot \Delta t$ per step, with κ_r as a recovery constant (typically 0.005). Bedrock shrinks by the same amount from its top, possibly eroding it completely without splitting, keeping column count low. This mimics vertical erosion, justifying our decision to fade out vertical erosion for steep slopes.

3.5. Support Check

Our algorithm can create degenerate columns with thin or absent bedrock, floating columns, and unrealistically large overhangs. To address this, we propose a support check algorithm with phases:

1. Stability initialization
2. Stability propagation
3. Column collapse

Stability initialization sets a stability value $p_{\mathbf{k}}$ for each column. The lowest column in each cell is always stable and set to ∞ , all others are set to 0. Stability propagation iteratively determines actual column stability over typically 10 steps per simulation step. After a user-defined total number of propagation steps, columns are collapsed and the process repeats, starting with initialization.

3.5.1. Stability Propagation

A single stability propagation step updates stabilities of all columns except the lowest, based on the columns in the 4-neighborhood. Column mass $m_{\mathbf{k}}$ is calculated as:

$$m_{\mathbf{k}} = l_c^2 (\rho_b b_{\mathbf{k}} + \rho_s s_{\mathbf{k}} + \rho_w w_{\mathbf{k}}) \quad (26)$$

with densities $\rho_b = 2600$, $\rho_s = 1600$, and $\rho_w = 1000 \frac{\text{kg}}{\text{m}^3}$ for bedrock, sand, and water, respectively. We define the maximum bedrock

support mass as $\kappa_p = 60000 \frac{\text{kg}}{\text{m}^2}$. Using the overlapping bedrock surface area, a column's maximal support is calculated as:

$$\Delta B = \min(B_{\mathbf{k}'}, B_{\mathbf{k}}) - \max(F_{\mathbf{k}'}, F_{\mathbf{k}}) \quad (27)$$

$$\bar{p}_{\mathbf{k}} = \max_{\mathbf{k}' \in N_4} \left(\min \left(l_c \kappa_p \frac{\Delta B^2}{B_{\mathbf{k}} - F_{\mathbf{k}}}, p_{\mathbf{k}'} \right) \right) \quad (28)$$

The minimum ensures that a neighbor cannot support more than its current stability. We implement two boundary modes: *border support* assumes infinitely large bedrock with infinite stability outside boundaries. Otherwise, no neighbors exist out-of-bounds.

Using the mass and maximal support, stability is updated:

$$p_{\mathbf{k}}^{t+\Delta t} = \max(p_{\mathbf{k}}^t, \bar{p}_{\mathbf{k}} - m_{\mathbf{k}}). \quad (29)$$

Thus, stability can only increase, starting from an initialized value of 0. Our stability propagation is aimed at efficiency and is at best physically plausible. However, it effectively detects floating bedrock and unrealistic, large overhangs connected to ground through thin sections. Behavior can be tweaked using κ_p and a strict maximum overhang size can be imposed by limiting the number of iterations. The delayed collapse mechanism introduced provides significant performance gains, but for immediate collapse detection, all iterations can be executed in a single step if required.

3.5.2. Column collapse

After sufficient stability propagation steps, columns with $p_{\mathbf{k}} \leq 0$ collapse, in addition to columns with bedrock thickness below a threshold b_{\min} . This process reverts splits that are close in proximity, as well as fully eroded columns, lowering column count. The collapsing column's bedrock becomes sand. Water, sand and sediment are merged with the column below.

Additionally, columns with $C_{\mathbf{k}} - B_{\mathbf{k}} < 0.05 D_{\text{split}}$ are merged with the column above. Sand is converted to bedrock and the height of the upper column is adjusted accordingly. This can only happen due to petrification of sand, for example when a column is buried under sand and ensures that no space is wasted on columns that are not needed anymore.

3.6. Water sources and aging

To enable hydraulic erosion, water is added to the terrain. Following MEI et al. [MDH07], we increase the topmost column's water level by $\Delta t \cdot \kappa_{\text{rain}}$ at the start of each simulation step, modulated with time-varying discrete noise for randomness. Users can place up to four noise-modulated circular water sources with adjustable radius, strength, and flux acceleration. These sources add water to the topmost column, but could be extended to a 3D position in the future.

Evaporation is unnecessary with outflow borders, but with reflecting boundaries, it prevents overflow. Unlike MEI et al. [MDH07], who remove a water percentage each step, we take a more realistic approach and subtract $\Delta t \cdot \kappa_{\text{evap}}$ from the water level each step. This is done after the transport step, which enables water accumulation even if $\kappa_{\text{evap}} \geq \kappa_{\text{rain}}$. Evaporation is interpolated to 1% of its strength if the water level is near the ceiling to accommodate flooded passages.

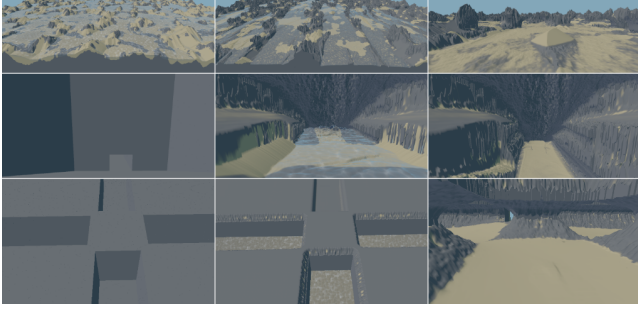


Figure 5: *Top: RiversRain scene with heavy rain which is decreased over time. The last image is a closeup of the middle of the scene, showing a collapsed structure. Middle: Cave scene showing cave formation based on a initial guide cut. Bottom: Arches scene showing arch generation without initial overhangs. The first image in each row shows the scene pre-simulation and for the last image, water was evaporated fully.*

Sand petrifies into bedrock over time, influenced by the petrification constant κ_{petr} and sand height. Each step, we move $\kappa_{\text{petr}} \cdot \Delta t \cdot s_{\mathbf{k}}$ from sand to bedrock. This simplifies the approach of ŠT’AVA et al. [ŠBBK08], which involved timestamps and additional material layers.

4. Implementation details

We implemented our method on the GPU using CUDA with 9 kernels. Each kernel is launched with one thread per cell. Every thread reads the number of columns from a buffer (8-bit integer per cell) and iterates over all columns of its cell, avoiding unnecessary threads and simplifying our parallel implementation. Buffers are pre-allocated to the full $N_x \times N_y \times N_z$ size, avoiding re-allocations. We use 32-bit integers and floats unless stated otherwise. Our buffers require 84 bytes per column and one additional byte per cell, totaling 673 bytes per cell at $N_z = 8$. No CUDA exclusive features are used, so our method can be implemented in regular compute shaders on any modern graphics card.

The first kernel adds rain and source water. Transport is handled by two kernels to avoid race conditions: the first iterates over the neighborhood of each column, starting with the lowest to allow early exit. It computes outflow flux, sediment flux scale, slippage values, and tilt angle, storing them together with the z-index of the respective neighbor column (8-bit integer). The second kernel collides outflow and inflow flux values by checking whether the z-index of the neighboring column matches the current column, calculates velocities and updates height and sediment values. Evaporation and petrification are also handled here.

Next, Horizontal and vertical erosion are applied, split into three kernels to avoid race conditions. The first kernel iterates over neighbor columns starting from the highest and exits early for a direction if $W_{\mathbf{k}'} \leq F_{\mathbf{k}}$. It finds σ_{max} and ΔD_{max} , increases the column damage and stores σ_{max} in a buffer. The second kernel checks if damage exceeds the threshold and splits by inserting a new column if needed while keeping all columns in ascending order. The third kernel

applies vertical erosion and damage recovery locally within each cell.

The final three kernels handle the support check algorithm as described in Section 3.5. The stability propagation kernel resolves race conditions on the stability values by alternating updates using a checkerboard pattern, fitting well with our iterative approach.

4.1. Visualization

Our work focused on simulation, so we developed only a basic real-time visualization. We use OpenGL to allocate buffers for bedrock, sand, water, and ceiling/floor height values, mapping them to CUDA. After each simulation step, we fill an index buffer with all existing column indices using an atomic counter. The visualization draws a point for each column, expanded to cuboids in the geometry shader. Basic normal mapping, volumetric effects, refraction, reflection, and caustics are applied to the water using time-varying noise functions. Cuboid rendering results in blocky visuals and poor performance due to many hidden fragments and z-fighting artifacts.

We improve this visualization by interpolating the top and bottom surfaces of each cuboid. This checks the 8-neighborhood of cells and averages up to four height values from connecting columns per vertex if the neighboring bedrock overlaps with the center column. Top vertices interpolate with the highest column, and bottom vertices with the lowest. If a higher or lower column exists that overlaps the same neighbor, no interpolation is applied with that neighbor for the top, respectively bottom vertices.

This method hides many side faces, allowing us to skip emitting them in the geometry shader, resulting in faster rendering. However, determining which faces to discard is challenging, and our heuristic misses some faces that can be omitted. Side faces remain blocky, and the visualization is much slower than the simulation. A more advanced method, like in *Arches* [PGGM09], which computes an implicit representation and generates a smooth mesh, would be better.

We include a user interface for changing most parameters interactively, even during the simulation. Resolution and l_c changes require a reset. Scenes can be saved and loaded via the UI. The scenes used in this paper are included in the supplementary material.

5. Results

We evaluate our method on scenes with resolutions from 256^2 to 4096^2 . All scenes span $256m \times 256m$ with l_c set appropriately. Except for *Cave*, all scenes start with a single column per cell and no initial overhangs. Cave generation required a preformed guiding cave to be cut into the terrain, as our method does not cover the majority of the complex natural processes that lead to cave formation.

Performance was measured on an Nvidia RTX 3070 with 8 GB VRAM. At 4096^2 , we reduced N_z to 4 due to VRAM limits. Four layers are enough for our scenes, but we used eight in all other scenes for safety. 6 GB VRAM is sufficient for simulating our largest scene, the rest of our scenes require less than 4 GB each. We summarize average simulation time after 50,000 steps in Table 1. Simulation time is predicted to grow by a factor of 4 with resolution doubling,

| Scene | Resolution | Time/Step [ms] |
|---------------|-------------------|----------------|
| Bridges | 256 ² | 0.17 |
| ChasmRain | 256 ² | 0.23 |
| ChasmRain | 512 ² | 0.44 |
| ChasmSource | 512 ² | 0.49 |
| Bridges | 1024 ² | 1.44 |
| Arches | 1024 ² | 1.50 |
| ChasmRain | 1024 ² | 1.61 |
| ChasmSource | 1024 ² | 1.53 |
| Cave | 2048 ² | 5.77 |
| ChasmRain | 2048 ² | 6.00 |
| ChasmSource | 2048 ² | 5.89 |
| RiversRain | 2048 ² | 6.16 |
| RiversSource | 2048 ² | 5.46 |
| RiversSource* | 2048 ² | 4.53 |
| RiversRain | 4096 ² | 23.00 |

Table 1: Performance in milliseconds per simulation step across various scenes and resolutions. We used $N_z = 8$ except for the highest resolution, where we set it to 4. The scene marked with an asterisk ran with horizontal erosion and support check kernels disabled.

but our method overperforms slightly, achieving better than real-time performance up to 2048² (6ms per step with $\Delta t = 50ms$). Even 4096² runs in real-time at 23ms per step. At 2048², our fastest scene needs $\approx 5.5ms$ as it generates no extra columns. The slowest scene, generating the most extra columns, is about 10% slower. Horizontal erosion and support check on the fastest scene add about 1ms, which is about 20% of a simulation step. Transport is the most expensive part of our method, taking up approximately 50% of a step.

Our simplified method for the regolith layer works well, generating smooth sand surfaces under lakes and rivers (Figure 2). We produce interesting sedimentation and flow-like patterns on the sand surface (Figures 2 and 3). Overhang generation is easy and adapts to the scene conditions, as seen in Figure 3, comparing global rainfall to placing a water source inside a chasm using the same terrain. Our support check is evaluated on three bridge types (Figure 4). The most stable bridge starts thick and thins out, the second is uniformly thick, and the least stable grows thicker in an arch. Results match expectations, and parameters can be tweaked for different overhang lengths.

Figure 5 shows our method on a terrain initialized with procedural noise eroded by heavy rainfall, alongside scenes producing arches and a cave. Rounded overhangs and arches form in scenes initialized with rectangular bedrock blocks and no overhangs. Cave generation was guided by cutting a small rectangular guiding shaft through the bedrock.

Compared to voxel grid methods, a 512³ grid matches the column count of a 4096² grid with up to 8 layers in our method. Most cells contain a single column, rarely reaching the maximum, so less columns have to be processed. In contrast to a voxel grid, we can handle larger horizontal resolutions matching typical game terrain heightmaps and use full floating-point precision for height values. For example, a 1km vertical span using 512 voxels means each voxel

is 2 meters high. While voxels allow for more overhangs, this is unnecessary for large-scale terrain erosion. Our method is faster than typical voxel grid methods as we use the simple pipe model instead of solving the incompressible Navier-Stokes equations and we only have to process existing columns.

Lastly, our visualization has limitations. The limited interpolation reveals the box shape of the data structure, and z-fighting causes flickering artifacts in some scenes, as visible in Figure 4 when zooming in. Interpolation can stretch sand and water along steep cliffs (see Figure 3). Worst of all, our current visualization can take up to 10 times longer per frame than a single simulation step.

6. Conclusion and future work

In conclusion, our novel GPU 3D erosion method leverages multi-layered heightmaps to extend existing techniques, introducing innovations like horizontal erosion and bedrock support check. Demonstrated on an RTX 3070, it achieves real-time performance up to 4096² resolution, potentially scaling to 8192² with high-end GPUs. The method facilitates easy generation of overhangs and arches, with limited cave generation. It recovers efficiently from unrealistic terrain features like floating structures. By employing multi-layered heightmaps with full floating-point precision, we effectively handle high-resolution scenes, far surpassing the resolutions that would be feasible real-time in voxel grid methods.

Future directions include exploring tiled simulation as in ŠT’AVA et al. [ŠBBK08] for exceptionally large terrains as seen in open-world games or elevation data from satellites, enhancing user interaction through a more intuitive parameterization, and integrating with modeling tools like the *Arches* framework [PGGM09] for better interactivity and scene authoring. Our performance leaves enough room for more advanced algorithms, such as reducing vertical isotropy and adding more pipe connections as in [MY97; MFC06; Kel13; DGV*18] or treatment of flooded passages [DVG*18]. Integrating more accurate regolith models [Ben07; ŠBBK08] and sediment advection [MDH07; ŠBBK08] is feasible as well. Alternatively, the virtual pipes model could be replaced with SPH for proper 3D fluid flow, as proposed in KRISTOF et al. [KBKŠ09]. HARTLEY et al. [HMF24] have recently shown how to perform particle-based erosion on many terrain representations, including layered heightmaps, so this is certainly possible.

Our current implementation needs a faster real-time visualization that produces smoother results and should be extended with import and export capabilities using standard mesh formats. Importing meshes would be feasible using depth-peeling to build the multi-layered heightmap and mesh generation for exporting can be adapted from PEYTAIVIE et al. [PGGM09].

Ultimately, our method provides a robust and high-performance 3D erosion simulation for large-scale terrain, with many possibilities for future enhancements in functionality and integration into existing tools for various applications.

References

- [Ben07] BENES, BEDRICH. “Real-Time Erosion Using Shallow Water Simulation”. *Workshop in Virtual Reality Interactions and Physical Simulation “VRIPHYS” (2007)*. Ed. by DINGLIANA, JOHN and GANOVELLI,

- FABIO. The Eurographics Association, 2007. ISBN: 978-3-905673-65-4. DOI: [10.2312/PE/vriphys/vriphys07/043-050](https://doi.org/10.2312/PE/vriphys/vriphys07/043-050) 2, 7.
- [BFO*07] BEARDALL, M., FARLEY, M., OUDERKIRK, D., et al. “Goblins by spheroidal weathering”. *Proceedings of the Third Eurographics Conference on Natural Phenomena*. NPH’07. Prague, Czech Republic: Eurographics Association, 2007, 7–14. ISBN: 9783905673494 2.
- [BMPG11] BORGEAT, LOUIS, MASSICOTTE, PHILIPPE, POIRIER, GUILAUME, 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, 3.
- [BTHB06] BENEŠ, BEDŘICH, TĚŠÍNSKÝ, VÁCLAV, HORNÝŠ, JAN, and BHATIA, SANJIV K. “Hydraulic erosion”. *Computer Animation and Virtual Worlds* 17.2 (2006), 99–108. DOI: [10.1002/cav.772](https://doi.org/10.1002/cav.772).
- [CMF98] CHIBA, N., MURAOKA, K., and FUJITA, K. “An erosion model based on velocity fields for the visual simulation of mountain scenery”. *The Journal of Visualization and Computer Animation* 9.4 (1998), 185–194. DOI: [10.1002/\(SICI\)1099-1778\(1998100\)9:4<185::AID-VIS178>3.0.CO;2-2](https://doi.org/10.1002/(SICI)1099-1778(1998100)9:4<185::AID-VIS178>3.0.CO;2-2).
- [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 2, 7.
- [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](https://doi.org/10.1016/j.cag.2018.08.005) 2, 4, 7.
- [GGP*19] GALIN, ERIC, GUÉRIN, ERIC, PEYTAUVIE, ADRIEN, et al. “A Review of Digital Terrain Modeling”. *Computer Graphics Forum* 38.2 (2019), 553–577. DOI: [10.1111/cgf.13657](https://doi.org/10.1111/cgf.13657) 1.
- [HMFF24] HARTLEY, MARC, MELLADO, NICOLAS, FIORIO, CHRISTOPHE, and FARAJ, NOURA. “Flexible terrain erosion”. *The Visual Computer* (June 2024). ISSN: 1432-2315. DOI: [10.1007/s00371-024-03444-w](https://doi.org/10.1007/s00371-024-03444-w) 1, 2, 7.
- [HW04] HOLMBERG, NATHAN and WÜNSCHE, BURKHARD C. “Efficient modeling and rendering of turbulent water over natural terrain”. *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*. GRAPHITE ’04. Singapore: Association for Computing Machinery, 2004, 15–22. ISBN: 1581138830. DOI: [10.1145/988834.988837](https://doi.org/10.1145/988834.988837) 1.
- [Igl04] IGLESIAS, A. “Computer graphics for water modeling and rendering: a survey”. *Future Generation Computer Systems* 20.8 (2004). Computer Graphics and Geometric Modeling, 1355–1374. ISSN: 0167-739X. DOI: [10.1016/j.future.2004.05.026](https://doi.org/10.1016/j.future.2004.05.026) 1.
- [KKBŠ09] KRIŠTOF, P., BENEŠ, B., KŘIVÁNEK, J., and ŠT’AVA, O. “Hydraulic Erosion Using Smoothed Particle Hydrodynamics”. *Computer Graphics Forum* 28.2 (2009), 219–228. DOI: [10.1111/j.1467-8659.2009.01361.x](https://doi.org/10.1111/j.1467-8659.2009.01361.x) 2, 7.
- [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/11025/69732> 3, 7.
- [KM90] KASS, MICHAEL and MILLER, GAVIN. “Rapid, stable fluid dynamics for computer graphics”. *SIGGRAPH Comput. Graph.* 24.4 (Sept. 1990), 49–57. ISSN: 0097-8930. DOI: [10.1145/97880.97884](https://doi.org/10.1145/97880.97884) 1.
- [MDH07] MEI, XING, DECAUDIN, PHILIPPE, and HU, BAO-GANG. “Fast Hydraulic Erosion Simulation and Visualization on GPU”. *15th Pacific Conference on Computer Graphics and Applications (PG’07)*. 2007, 47–56. DOI: [10.1109/PG.2007.15](https://doi.org/10.1109/PG.2007.15) 1–5, 7.
- [MFC06] MAES, MARCELO M., FUJIMOTO, TADAHIRO, and CHIBA, NORISHIGE. “Efficient animation of water flow on irregular terrains”. *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*. GRAPHITE ’06. Kuala Lumpur, Malaysia: Association for Computing Machinery, 2006, 107–115. ISBN: 1595935649. DOI: [10.1145/1174429.1174447](https://doi.org/10.1145/1174429.1174447) 1, 7.
- [MKM89] MUSGRAVE, F. K., KOLB, C. E., and MACE, R. S. “The synthesis and rendering of eroded fractal terrains”. *SIGGRAPH Comput. Graph.* 23.3 (July 1989), 41–50. ISSN: 0097-8930. DOI: [10.1145/743334.743372](https://doi.org/10.1145/743334.743372).
- [MY97] MOULD, DAVID and YANG, YEE-HONG. “Modeling water for computer graphics”. *Computers & Graphics* 21.6 (1997). Graphics in Electronic Printing and Publishing, 801–814. ISSN: 0097-8493. DOI: [10.1016/S0097-8493\(97\)00059-9](https://doi.org/10.1016/S0097-8493(97)00059-9) 1, 4, 7.
- [NG24] NILLES, A. M. and GÜNTHER, L. *CUDA 3D Hydraulic Erosion Simulation with layered stacks*. <https://github.com/Clocktown/CUDA-3D-Hydraulic-Erosion-Simulation-with-Layered-Stacks>. 2024 1.
- [NGM24] NILLES, ALEXANDER, GÜNTHER, LARS, and MÜLLER, STEFAN. “Real-Time Desertscape Simulation with CUDA”. *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP*. INSTICC. SciTePress, 2024, 34–45. ISBN: 978-989-758-679-8. DOI: [10.5220/00123156000036602](https://doi.org/10.5220/00123156000036602).
- [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](https://doi.org/10.1109/CA.1995.393532) 1.
- [PGGM09] PEYTAUVIE, A., GALIN, E., GROSJEAN, J., and MERILLOU, S. “Arches: a Framework for Modeling Complex Terrains”. *Computer Graphics Forum* 28.2 (2009), 457–467. DOI: [10.1111/j.1467-8659.2009.01385.x](https://doi.org/10.1111/j.1467-8659.2009.01385.x) 1, 2, 6, 7.
- [PPG*19] PARIS, A., PEYTAUVIE, A., GUÉRIN, E., et al. “Desertscape Simulation”. *Computer Graphics Forum* 38.7 (2019), 47–55. DOI: [10.1111/cgf.13815](https://doi.org/10.1111/cgf.13815) 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, 3, 6, 7.
- [SK06] SUTHERLAND, BEN and KEYSER, JOHN. “Particle-based enhancement of terrain data”. *ACM SIGGRAPH 2006 Research Posters*. SIGGRAPH ’06. Boston, Massachusetts: Association for Computing Machinery, 2006, 96–es. ISBN: 1595933646. DOI: [10.1145/1179622.1179732](https://doi.org/10.1145/1179622.1179732) 2.
- [TGSC24] TZATHAS, PETROS, GAILLETON, BORIS, STEER, PHILIPPE, and CORDONNIER, GUILLAUME. “Physically-based analytical erosion for fast terrain generation”. *Computer Graphics Forum* 43.2 (2024), e15033. DOI: [10.1111/cgf.15033](https://doi.org/10.1111/cgf.15033) 2.
- [TK23] TAYLOR, BRENNEN and KEYSER, JOHN. “Real-Time Sand Dune Simulation”. *Proc. ACM Comput. Graph. Interact. Tech.* 6.1 (May 2023). DOI: [10.1145/3585510](https://doi.org/10.1145/3585510) 2.
- [WCMT07] WOJTAN, CHRIS, CARLSON, MARK, MUCHA, PETER J., and TURK, GREG. “Animating corrosion and erosion”. *Proceedings of the Third Eurographics Conference on Natural Phenomena*. NPH’07. Prague, Czech Republic: Eurographics Association, 2007, 15–22. ISBN: 9783905673494 2.
- [WXL*24] WANG, XIAOKUN, XU, YANRUI, LIU, SINUO, et al. “Physics-based fluid simulation in computer graphics: Survey, research trends, and challenges”. *Computational Visual Media* (Apr. 2024). ISSN: 2096-0662. DOI: [10.1007/s41095-023-0368-y](https://doi.org/10.1007/s41095-023-0368-y) 1.