DOI: 10.1111/cgf.13669
Eurographics Conference on Visualization (EuroVis) 2019
M. Gleicher, H. Leitte, and I. Viola
(Guest Editors)

Volume 38 (2019), Number 3

Interactive Visualization of Flood and Heavy Rain Simulations

D. Cornel[1], A. Buttinger-Kreuzhuber[1,2], A. Konev[1], Z. Horváth[1,2], M. Wimmer[2], R. Heidrich[3], and J. Waser[1]

[1]VRVis Forschungs-GmbH, Vienna, Austria [2]TU Wien, Vienna, Austria [3]RIOCOM, Vienna, Austria

Figure 1: *Visualization of smoothly interpolated flood simulation results and terrain defined on adaptive grids. (Left) Large-scale flooding of villages by a nearby river. (Right) Stormwater runoff in a mountainous region. High velocities are indicated by white foam.*

Abstract

In this paper, we present a real-time technique to visualize large-scale adaptive height fields with C^1-continuous surface reconstruction. Grid-based shallow water simulation is an indispensable tool for interactive flood management applications. Height fields defined on adaptive grids are often the only viable option to store and process the massive simulation data. Their visualization requires the reconstruction of a continuous surface from the spatially discrete simulation data. For regular grids, fast linear and cubic interpolation are commonly used for surface reconstruction. For adaptive grids, however, there exists no higher-order interpolation technique fast enough for interactive applications.

Our proposed technique bridges the gap between fast linear and expensive higher-order interpolation for adaptive surface reconstruction. During reconstruction, no matter if regular or adaptive, discretization and interpolation artifacts can occur, which domain experts consider misleading and unaesthetic. We take into account boundary conditions to eliminate these artifacts, which include water climbing uphill, diving towards walls, and leaking through thin objects. We apply realistic water shading with visual cues for depth perception and add waves and foam synthesized from the simulation data to emphasize flow directions. The versatility and performance of our technique are demonstrated in various real-world scenarios. A survey conducted with domain experts of different backgrounds and concerned citizens proves the usefulness and effectiveness of our technique.

1. Introduction

The ubiquitous use of computer simulations in flood and stormwater management creates a growing demand for expressive and efficient techniques to visualize the corresponding, often very large flood-related data (Figure 1). As interactivity is a key requirement for modern decision support tools, the performance of simulations becomes a crucial aspect. Flood simulation on triangular meshes is highly flexible with respect to approximating complex boundaries such as building walls or protection barriers. However, it is relatively slow for decision making, which often relies on ensembles of large-scale scenarios [WKS*14]. In contrast, shallow water simulation on rectangular grids provides a significantly higher performance. The grid structure is well suited for the GPU data model, which enables the parallelization of shallow water simulations [HPW*16]. For this reason, grid-based flood simulation using the finite-volume method is the state of the art in interactive flood management, and visualization of these results is an important task.

The results of shallow water simulations are discretely defined

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

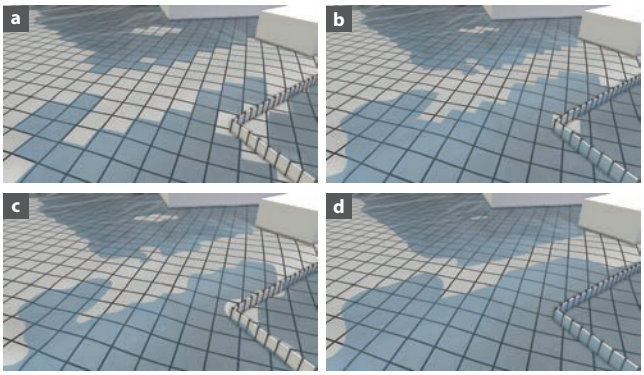

Figure 2: *Surface reconstruction with interpolation. (a) Nearest-neighbor and (b) bilinear interpolation lead to linear and angular shoreline features. A smoother shoreline is obtained by bicubic interpolation with (c) Catmull-Rom splines and (d) cubic B-splines.*

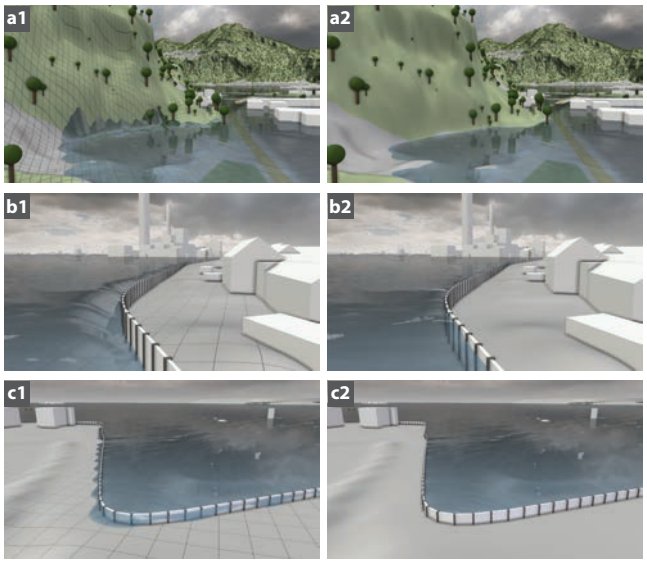

Figure 3: *Common artifacts with standard interpolation (left column) compared to our results (right column). (a) Climbing. (b) Diving. (c) Leaking.*

on rectangular grids in the form of height fields. Visualization of such height fields requires some sort of surface reconstruction by means of interpolation or approximation. The simplest form of interpolation, nearest neighbor, produces results that are far from being physically realistic (Figure 2a), which makes it practically useless for engineering or presentation purposes. Bilinear interpolation introduces C^0-continuity, but leads to straight or angular shoreline features (Figure 2b). For regular grids, C^1-continuity is achieved by bicubic interpolation (Figure 2c) or approximation (Figure 2d), which are fast enough for interactive applications.

To keep up with the increasing scale of simulated scenarios and the generated data, adaptive grids can be used. Currently, however, one has to choose between fast linear interpolation [KT09, Lia11] and offline smooth interpolation [BMA10, BOR14, GJAG14, FBHD17]. None of these algorithms provides a continuously differentiable surface reconstruction while also being efficient enough for interactive applications. The main challenge of adaptive grid interpolation is the absence of implicitly given relationships between neighboring cells of different resolution. Within a neighborhood required for third-order interpolation, a potentially infinite number of alternative cell arrangements is possible. Retrieving the neighborhood values is computationally hard, yet necessary to achieve smooth surface transitions between grid cells of different resolution. One contribution of this paper is a novel method for fast C^1-continuous third-order interpolation on adaptive grids.

When using a fast, but simple interpolator for surface reconstruction, such as usual bilinear or bicubic interpolation, the reconstructed water surface is not always plausible. This is particularly apparent at dry/wet boundaries. Here, two types of features can manifest, which we consider artifacts in the context of water visualization. First, if the height field is not defined everywhere, missing values at dry cells have to be deduced from existing data. Previous work [HWP*15, HPW*16] used the terrain elevation as water level at these dry cells, which can lead to uphill *climbing* of water (Figure 3a1) as well as *diving* towards walls (Figure 3b1). Second, wall boundaries are discretized on the rather coarse grid used for simulation and interpolation. Therefore, the interpolated extents of simulated inundation do not always match the detailed wall geometry used for visualization, leading to seeming *leaking* (Figure 3c1). During live sessions in the past, domain experts misinterpreted these artifacts as actual water propagation several times. We present a surface reconstruction for water height fields with boundary conditions that reduces these artifacts (Figure 3, right column) in both regular and adaptive grids.

The correct perception of flow behavior is important for the interpretation of water simulation results. However, it is hard to extract this information from a colored surface alone. The use of glyphs or other established flow visualization techniques could solve the problem, but would result in a cluttered visualization that draws all the viewer's attention. In nature, flow directions can often be identified easily by waves and patches of foam flowing on the water surface. We demonstrate the use of exaggerated waves and foam derived from simulation data as visual metaphors for the intuitive indication of flow behavior (Figure 1, right).

Although water rendering for entertainment purposes, e.g., for video games, is a well-researched topic, the interactive visualization of time-dependent flood simulation data is still a challenge. In previous work, the complex problem of water rendering has been simplified based on assumptions that certain data do not change. However, such assumptions do not hold for dynamic simulation data. For example, water bodies in video games are usually confined to a static triangle mesh or plane such that the water surface can be perfectly aligned to the scene geometry by an artist manually. This alleviates the need for expensive interpolation and dealing with artifacts, but prohibits the dynamic inundation of the scene. Likewise, using precomputed flow data for surface shading is sufficient for an aesthetic appearance. For the accurate visualization of flows in a dynamic velocity field, this simplification makes no sense. Therefore, we feel the need to set ourselves apart from these techniques

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

which are limiting in our case. Instead, we introduce more suitable techniques tailored to our particular application.

To summarize, this paper provides the following contributions:

- Continuous third-order and linear interpolation methods for height fields defined on adaptive grids
- A surface reconstruction for water height fields including correct treatment of boundaries to reduce reconstruction artifacts
- Water shading with waves and foam derived from simulation data for better depth and flow perception

The described techniques for the reduction of artifacts and for water surface shading are independent from the simulation grid and can be used for adaptive and regular grids alike. We have thoroughly evaluated the presented results in live sessions and with a user survey among 96 participants consisting of experts in the field of flood management as well as members of the general public. Benchmarks demonstrate the high performance of the proposed techniques in real-world scenarios.

2. Related Work

Interpolation. Fast interpolation of data defined on adaptive grids is of great interest in volume rendering, where data sets become large rapidly. Here, the common practice of interpolation is to map the data to multi-level textures and then facilitate trilinear hardware interpolation [WWH*00, KH02, BHMF08]. For adaptive-mesh-refinement grids, the stitching method is commonly used, which inserts special stitch cells with pre-defined interpolation behavior at level transitions [WKL*01, ME11, BST15]. For interpolation within quadtrees, Kim and Tsiotras [KT09] propose a bilinear interpolation scheme that uses bilinearly upsampled values in transition regions between levels. A similar scheme proposed by Liang [Lia11] uses an enclosing triangle for upsampling. All of these methods use a linear interpolator on the upsampled regions and thus yield C^0-continuous results. However, an error assessment by Kidner [Kid03] shows that higher-order interpolation of digital elevation models leads to a significantly more accurate surface representation than linear interpolation. The scheme proposed by Min and Gibou [MG06] uses a biquadratic interpolator, but does not consider continuous differentiability in transition regions.

C^1- or C^2-continuous local refinable splines defined on hierarchical data structures are often used for isogeometric analysis [DCL*08, BLE*14, LCKD16]. Fuchs et al. [FBHD17] use them for volume rendering and report runtimes of 38 milliseconds for around 256 000 cells. For unstructured data, interpolation with radial basis functions such as thin plate splines can be used. Hutchinson [Hut95] demonstrates an interpolation of rainfall data, resulting in a C^1-continuous surface. Smooth interpolation of scattered data can be accelerated by combining locally defined thin plates [Fra82]. With an evaluation of thin plate splines on the GPU, Beatson et al. [BOR14] report a runtime of 1.3 seconds for 128 000 data points. Kriging, a technique for geostatistical prediction, has also been used for the approximation of scattered geospatial data [Goo00, Ree00]. With GPU-based implementations, the processing of a few thousand data values takes several seconds [Che13, GJAG14]. Natural-neighbor interpolation for unstructured data yields C^1- or C^2-continuous surfaces [Bob08]. With an implementation optimized for the GPU, Beutel et al. [BMA10] report a runtime of 163 seconds for 186 million cells.

To our knowledge, there exists no continuously differentiable interpolation technique that can process adaptive data in the order of one hundred million cells interactively, i.e., in less than 30 milliseconds. Currently, this efficiency is only achieved by grid-based techniques such as regular bilinear or bicubic interpolation. However, the existing approaches are either not suited for adaptive grids or not continuously differentiable, which are given requirements in our application. Our proposed technique fills this gap by extending bicubic interpolation to adaptive grids with very little overhead.

Water visualization. The challenge of visualizing water bodies affects a variety of fields from engineering simulations to the entertainment industry. Often, the water visualization should not only be recognizable [KC14], but should also convey important characteristics such as depth, flow direction, or flow speed. This holds for engineering simulation tools [TUF, Riv, MIK], flood management decision support systems [Flo, Vis, LKT*17], and public communication [You, CKS*15, VGB*16]. Currently, the most popular ways to represent such properties are color mapping and glyphs.

For the realistic appearance of virtual water surfaces, waves can be generated by the superposition of wave functions [Tes01, Fin04, NSB13], wavefront tracking [JW15], or by particle-based simulations [YHK07, YNBH09, JW17]. Tile-based approaches map directional features onto surfaces using flow fields [Gri11, vH11, GH12]. Vlachos demonstrates their benefit for route guidance in a video game [Vla10]. Local flow behavior can be visualized using spray particles and foam [BSW10, CM10, DB12, KLCK17]. The perception of water depth can be influenced by appropriate shading methods [PA01, Bel04]. Visualizing simulation-based flow information with waves and foam was evaluated for non-professionals [GSH*15] and was judged intuitive and useful.

3. System Overview

We implement our findings in a decision support system that combines simulation, analysis, and visualization of flooding scenarios [Vis]. In the system pipeline, we use our proposed approach for interactive 3D visualization of dynamic water simulation and terrain data. The water flow is simulated with an existing, integrated GPU-based shallow water simulator [HPW*16] using the finite-volume method. The output of the simulation, which serves as input for our technique, is a collection of scalar and vector fields defined on a rectangular adaptive grid. For the adaptive grid, we use the data structure described by Liang [Lia11]. However, the techniques we present are largely independent of the used adaptive grid. Any grid-based data structure will suffice, provided that it facilitates a quick retrieval of the data value for the cell covering a given 2D position. The only necessary restriction is that neighboring cells, i.e., cells that share a vertex, must not differ by more than one level.

The input data fields include absolute *water levels* which are water elevations relative to sea level, *water depths* which are water elevations relative to the terrain, and 2D water velocity vectors. An additional terrain field represents ground elevations relative to sea level. In our system, all discretized data are cell-registered, meaning that data values are associated with cell centers. However,

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

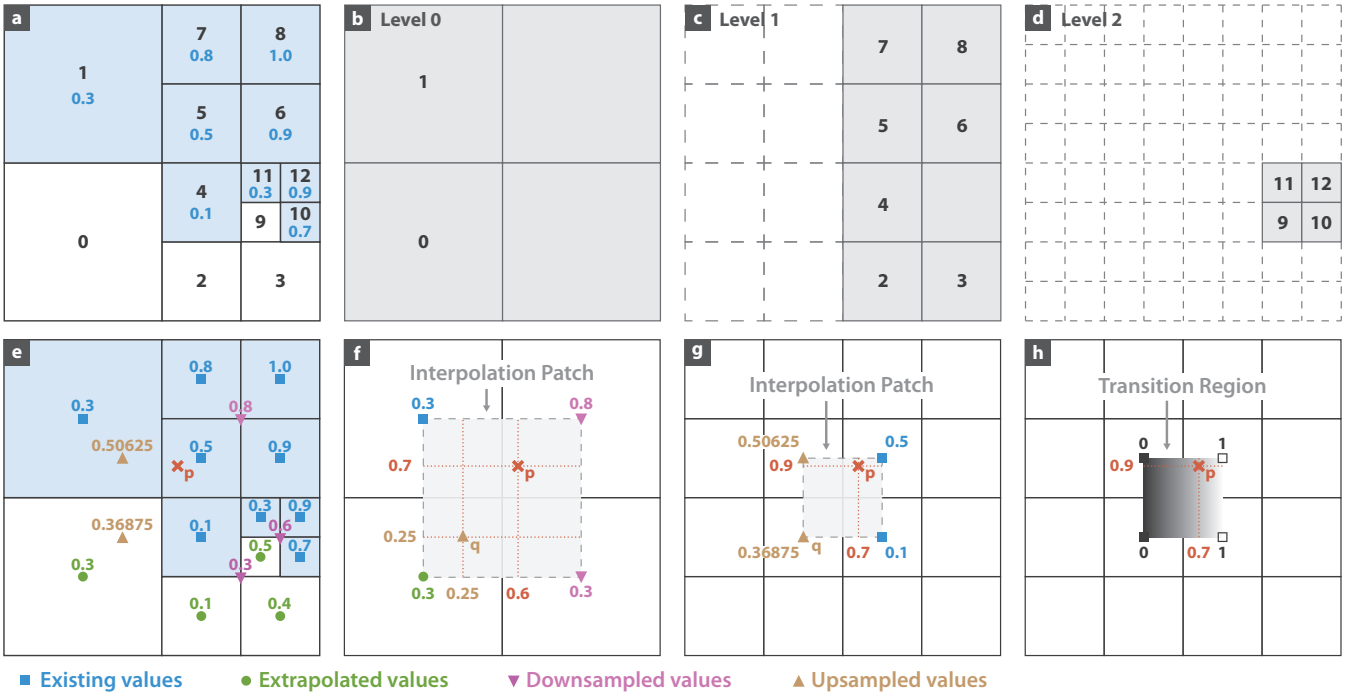

Figure 4: *Bilinear interpolation within an adaptive grid. (a) Enumerated cells of the adaptive grid (black). Height-field values (blue) defined on wet cells shaded in blue. Dry cells are not shaded. (b)–(d) Individual grid levels. (e)–(g) Missing values have to be reconstructed for the interpolation patches on the two levels. (h) Blending of individual interpolation results within the transition region across levels.*

a characteristic of the finite-volume method is that the resulting simulation data are average values over the entire cell rather than exact measurements at the cell center.

4. Adaptive Height-Field Reconstruction

The efficiency of interpolation on a regular grid stems from the trivial neighborhood relationship between cells. In a regular grid, the neighborhood of an interior cell is always symmetric and consists of eight neighbors. Data values of neighbors can be accessed easily, which makes operations defined on a larger neighborhood of cells simple. The challenge of interpolation on an adaptive grid is that neighborhood relationships between cells are not straightforward anymore. As the size of neighboring cells can differ by one level, there exists a large variety of arrangements with six to twelve direct neighbors of a cell. This prevents constant-time access to the data of neighbors and makes operations defined on cell neighborhoods—such as interpolation—complex tasks. However, adaptive grids can be interpreted as a set of sparse regular grids of the same world-space extents but different resolutions. We call these grids *levels* and enumerate them starting from 0 for the coarsest resolution. Figure 4a illustrates an adaptive grid with three levels that are shown individually in Figure 4b–d. On each of these conceptual regular grids, we can perform neighborhood operations efficiently again. This interpretation transforms the initially complex operation into multiple simple operations performed on the individual levels. Yet, it also introduces two new challenges, namely, how to reconstruct missing values on the individual levels, and how to combine the results of the individual operations to a final result. Kim and Tsiotras [KT09] perform bilinear interpolation on a local 2×2 grid with missing values reconstructed on the fly. We extend this approach to the 4×4 neighborhood required for bicubic interpolation, in which case the reconstruction of missing values is much more expensive. This is why we represent the individual levels with an actual sparse data structure to cache reconstructed values. As we aim for C^1-continuity of our reconstruction scheme, we also need to take care of level transitions, which has not been addressed in previous work.

In this section, we present a technique for the fast adaptive interpolation or approximation of cell-registered height fields. The main idea is to perform regular interpolation independently for neighboring levels and then stitch the regular regions together with blending functions. We obtain missing values on the individual levels by extrapolation, downsampling, and upsampling of existing values. The resulting reconstruction is identical to bilinear or bicubic interpolation in regular regions and continuous over level transitions. The described technique builds on well-known interpolation functions for regular grids that are interchangeable and independent from the additional challenges of adaptive grids that we tackle. For the sake of simplicity, we therefore start by explaining only bilinear interpolation in detail to highlight the important steps. The subsequent extension to bicubic interpolation then comes naturally. An interactive WebGL application demonstrating both techniques is available online with full source code [Sha].

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

4.1. Bilinear Interpolation

Bilinear interpolation operates on the 2×2 block of cells surrounding each interpolation position p. We call these cells the *neighborhood* of p in the following. Their centers span a square called an *interpolation patch* over which data values are interpolated. In an adaptive grid, however, the surrounding cells of p do not necessarily have the same size, but might belong to different levels. Yet, two neighboring cells can never differ by more than one level, which is why there are only two cases to distinguish. Either all four cells belong to the same level, in which case we perform usual bilinear interpolation on a locally regular grid, or the four cells belong to two consecutive levels. We now explain how to proceed in the second case with an example illustrated in Figure 4.

The sampling position p, marked in red in Figure 4e, is covered by a level-1 cell. While its bottom neighbor belongs to the same level, the bottom left and left neighbors belong to level 0. On each of the two involved levels, we can imagine a 2×2 neighborhood of cells surrounding p, even if not all of the cells exist in the adaptive grid. We call these *imaginary neighborhoods*. The cell centers of each imaginary neighborhood again span a square interpolation patch, as shown for level 0 and level 1 in Figure 4f and Figure 4g, respectively. The values at the corners of the interpolation patches are the values given at the centers of the corresponding cells of the imaginary neighborhood. Within each interpolation patch, we can perform a usual bilinear interpolation of the corner values. However, some of the cells of both imaginary neighborhoods have missing values. Before interpolation, these missing values have to be reconstructed by *extrapolation*, *downsampling*, or *upsampling*.

Extrapolation. On level 0 in our example, only the bottom left and top left cells correspond to leaf cells of the adaptive grid shown in Figure 4a. This is indicated by unique cell indices shown in black. Of these two cells, only the top left cell is a wet cell that has a data value provided by the simulation, which is indicated by a blue square and a blue shade of the cell in Figure 4e. The bottom left cell is a dry cell and requires extrapolation, which is indicated by a green circle. To fill in missing values of dry cells, we average over the values of all neighboring wet cells, including diagonal neighborhood. In our application, this extrapolation is only used for absolute water levels to avoid climbing and diving at dry/wet cell boundaries, which is explained in detail in Section 5.

Downsampling. The top right cell in Figure 4f does not exist in the adaptive grid. Instead, this part of the adaptive grid is covered by the four level-1 cells labeled 5, 6, 7, and 8 in Figure 4a. Yet, we require a data value at the center of the level-0 cell for the interpolation patch on level 0. We therefore downsample the values of the four level-1 cells by averaging, which is indicated by a purple triangle pointing downward. The bottom right cell on level 0 also needs to be downsampled. In this case, however, not even all corresponding level-1 cells actually exist in the adaptive grid, but only three of them. The value of the last cell itself needs to be reconstructed by downsampling the four level-2 cells 9, 10, 11, and 12. It can be seen that downsampling is a recursive process. Furthermore, the cells 2, 3, and 9 covered by the bottom right level-0 cell are dry cells, meaning that extrapolation has to take place before downsampling.

Upsampling. With extrapolation and downsampling, all values of the interpolation patch on the coarse level can be reconstructed. On the fine level illustrated in Figure 4g, however, we also need to reconstruct missing values by upsampling, indicated by a brown triangle pointing upward. This is necessary for the bottom left and top left cells of the imaginary neighborhood, which are covered by the level-0 cells 0 and 1 in the adaptive grid. We focus on the bottom left cell with center q marked in both Figure 4f and Figure 4g. The value at q has to be reconstructed from surrounding values. These surrounding values have already been gathered for the interpolation patch on level 0, which is why we calculate the missing value by simple bilinear interpolation within this interpolation patch. We perform the same upsampling at the center of the top left cell.

In previous work, Kim and Tsiotras [KT09] performed downsampling and upsampling on the fly. However, during interpolation, reconstructed values are reused multiple times, which is why it makes sense to cache them. In Section 4.3, we explain how to store and update all values in a sparse texture hierarchy so that no reconstruction is required during rendering.

Once all required values are present, we perform bilinear interpolation at p on both levels separately. We denote the bilinear interpolation at p on the coarse (level 0) and fine (level 1) levels by $v_c(p)$ and $v_f(p)$, respectively. Since we need a single value, we combine the interpolation results of both levels such that discontinuities at the level transition are avoided. We achieve this with a convex combination of $v_c(p)$ and $v_f(p)$ over a *transition region*, illustrated in Figure 4h, which requires both values to be defined over the entire transition region. The transition region is therefore given by the intersection area of the interpolation patches of both levels, which is simply the interpolation patch of the fine level. In this region, the influence $i(p)$ of value $v_f(p)$ should be zero at corners where data values had to be upsampled, and one in the remaining corners. Influence values in between are calculated by bilinear interpolation. The final interpolation value at p is

$$v(p) = (1 - i(p))v_c(p) + i(p)v_f(p). \tag{1}$$

In our example given in Figure 4e–h, the relative positions of p within the interpolation patches of levels 0 and 1 are $(0.6,\ 0.7)$ and $(0.7,\ 0.9)$, respectively. This results in the individual values $v_c(p) = 0.51$ and $v_f(p) = 0.46975$. With the influence $i(p) = 0.7$, the final result is $v(p) = 0.481825$.

As stated above, our proposed interpolation technique is mostly independent from the interpolation function used for the individual regular levels. It simply blends the individual values together for a continuous surface with smooth transitions between different-sized cells. Bilinear interpolation is a simple and well-known technique ideally suited to introduce the idea. The resulting surface is C^0-continuous and preserves the given data values at cell centers. For many applications, this is a desired and sufficiently realistic representation of the height-field data. For water height fields, however, linear surface boundaries still lead to unpleasant visual artifacts such as perfectly straight or angular shorelines (Figure 2b).

4.2. Bicubic Reconstruction

For smoother interpolation of data defined on regular grids, cubic splines are a good choice [Ree00]. Compared to the 2×2 cell neighborhood considered for bilinear interpolation, bicubic interpolation

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

takes into account a 4×4 neighborhood of an interpolation position, including not only the four surrounding data points, but also their neighbors. The extended neighborhood allows for matching the derivatives in start and end points of the interpolation splines to neighboring interpolation patches, resulting in a smooth, C^1-continuous surface.

For adaptive cubic interpolation, the increased size of the imaginary neighborhood means that significantly more missing values than for bilinear interpolation have to be reconstructed. For bilinear interpolation, when sampling within a cell, only data of direct neighbors are needed, i.e., reconstruction of existing values on the corresponding level is limited to the 3×3 neighborhood around each cell. Now, we also need data of the direct neighbors' neighbors, which extends the region of values that need to be reconstructed to the 5×5 neighborhood around each cell. First, we need to extend the range of extrapolation. If none of the direct neighbors of a dry cell is a wet cell, we subsequently consider their neighbors as well, and average over the data value of all wet cells in this large neighborhood. Second, upsampling may be required outside the interpolation patch of the coarse level. For bilinear interpolation, we could conveniently obtain the value of q in Figure 4g by bilinear interpolation within the already existing interpolation patch of the coarse level illustrated in Figure 4f. For values of the direct neighbors' neighbors, however, we now have to construct separate interpolation patches. This, again, might also require extrapolation and recursive downsampling. In summary, while the individual reconstruction operations are the same as for bilinear interpolation, their combination in the larger neighborhood leads to a high number of required calculations. This makes it even more important to cache the reconstructed values.

Once all values are present, there exist many legitimate options for the cubic reconstruction filter. Mitchell and Netravali [MN88] define a family of cubic filters with a continuous parameter space ranging from exact interpolation of values by the Catmull-Rom spline to the smoothest approximation by the cubic B-spline. Interpolation preserves the given values at data points, which is often a desired property. Approximation of data values leads to an overall smoother and more natural result than cubic interpolation, as comparable in Figure 2. Moreover, approximation with the cubic B-spline can be implemented efficiently with only four instead of the usual 16 texture lookups for the 4×4 neighborhood of an interpolation position by exploiting hardware texture filtering [SH05]. The choice for either interpolation or approximation solely depends on the use case and the desired smoothness. Our reconstruction scheme is completely independent of the used cubic spline. The final interpolation value $v(p)$ in transition regions is again calculated by a convex combination of $v_c(p)$ and $v_f(p)$. However, to achieve global C^1-continuity, we need at least a cubic blending function $H(t)$, such as the cubic Hermite spline used in the popular *smoothstep* function,

$$H(t) = t^2(3-2t). \tag{2}$$

The final value is then

$$v(p) = (1 - H(i(p)))v_c(p) + H(i(p))v_f(p). \tag{3}$$

The derivative $H'(t)$ of $H(t)$ is continuous, meaning that the convex combination $v(p)$ of the C^1-continuous interpolation polynomials $v_c(p)$ and $v_f(p)$ itself is C^1-continuous within the transition region. Furthermore, $H'(t)$ vanishes on the edges of the transition region, i.e., $H'(0) = H'(1) = 0$. Thus, $v(p)$ is also C^1-continuous on the edges of the transition region, meaning that the height field is continuously differentiable over the whole adaptive grid.

4.3. Implementation

Our system uses the adaptive interpolation described above during rendering of the water and terrain height fields multiple times. Both height fields are sampled for triangulation, for surface effects, and coloring, amounting to many million interpolation operations each frame. It is therefore crucial that interpolation can be performed as efficiently as possible using hardware acceleration. This is why we focus on the GPU implementation here, although in our system, interpolation is also used for multiple CPU-side operations such as the water sampling near buildings to determine damage extents.

As described above, we can interpret the adaptive grid as a set of regular grids. For an efficient implementation, we adopt this conceptual representation and maintain a set of regular 2D textures created from the adaptive data structure. We store data values in these textures and update them whenever the input data fields change. For water, we store the absolute water level, the relative water depth, and the two-dimensional velocity for each cell. For the terrain, we store the elevation and an overlay value used for coloring the terrain with a user-defined transfer function. We use multiple texture levels for data of different grid levels, each having the same dimensions as its corresponding grid level. This is an existing functionality exposed by the OpenGL API. As a consequence, each cell of each individual grid level corresponds to one texel of the multi-level texture. Yet, the use of 2D textures is only partly due to the heavy optimization of texture accesses on GPUs.

More importantly, the individual texture levels also contain the cells of the imaginary neighborhoods that require reconstruction. This enables constant-time access to the cached values after an initial reconstruction step, which is crucial for the real-time applicability of our technique. Of course, allocating video memory for all cells of all levels would not be possible for large data sets, which is the reason for using adaptive grids in the first place. However, as bicubic interpolation within one level requires a 4×4 neighborhood of cells on that level, the set of all possibly needed reconstructed values is limited to the 5×5 neighborhood around each wet cell. With this information, we can greatly reduce the memory requirements by using sparse textures, which are a hardware feature widely supported by modern GPUs. Sparse textures allow for the definition of virtual 2D data fields, where memory is allocated only for manually defined memory pages. We allocate memory only for pages containing at least one cell of the 5×5 neighborhood around any wet cell. As the memory pages have a fixed size (64 KB on our system), the size of the texture's data type controls how many texels one page covers. To minimize memory waste, we minimize the number of texels covered by one page by packing all data fields into different channels of a single multi-channel texture.

We perform the reconstruction and storage of values in the sparse textures whenever input data fields change. We identify the cells that require reconstruction and iterate over these *reconstruction cells* with three different compute shaders in sequence. First, we

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

reconstruct absolute water levels inside dry cells by extrapolation of values from neighboring wet cells and set the relative water depths and velocities of the reconstruction cells to zero. Second, we calculate downsampled values of the four values on the next-higher level covered by each reconstruction cell. As this is a recursive process, we perform this step individually for each level from the finest to the coarsest. Finally, we calculate upsampled values for reconstruction cells covered by lower-level adaptive grid cells.

In summary, we maintain a sparse data structure of all given and intermediate values to separate the reconstruction of missing values from their use for surface reconstruction into two steps. An expensive, but infrequent update step is only necessary when the simulation data change, e.g., for playback of a scenario or for navigation in time. During highly interactive tasks, such as sketching and manipulation of simulation parameters or navigation in space, we can retrieve the cached values efficiently for fast rendering. For rendering, we create a triangle mesh with view-dependent level of detail on the fly using recursive tessellation [LJL13]. We provide runtime benchmarks for both update and rendering in Section 7.

5. Artifact Removal

The reconstruction of height-field data by interpolation is not always plausible. For example, a reconstructed water surface should have a smooth shoreline that touches the terrain. Furthermore, the water surface should extend to walls that are positioned on dry cells. Here, we assume that the water at dry/wet boundaries near walls should be locally flat. In order to propagate height values to obstacles in a realistic manner, we include boundary conditions for the spline interpolation by properly extending the height field in dry regions. An incorrect extension of height-field values leads to unnatural shapes at dry/wet boundaries that suggest false water propagation, which is why we consider them artifacts. At slopes, for example, if missing water levels of dry cells are simply set to the terrain elevation, water climbs uphill (Figure 3a1). In proximity of walls, if the water levels on the wet side of the wall are smoothly interpolated to the lower terrain elevation of dry cells on the other side, the water surface sharply declines towards the wall (Figure 3b1). Here, the water surface might also be interpolated through the wall, leading to ostensible leaking (Figure 3c1).

These artifacts are not specific to adaptive grids or to our proposed interpolation method. Even in the case of traditional bilinear interpolation within a regular grid, these artifacts will occur. The main issue for diving and climbing is an incorrect extension of the water height field to dry grid cells. Leaking is caused by the finite resolution of the simulation domain and the consequent discretization of boundary conditions for simulation, which neglects more precise information. In theory, leaking could be avoided with an ideal interpolator that accounts for dynamically drawn barrier lines, e.g., for sandbags and dam lines, which can have arbitrarily many vertices. However, even if such an interpolator could be constructed, the computational effort of such high-resolution interpolation would likely be unfeasible for interactive applications. We therefore focus on removing the most prominent artifacts that appear in practice.

Climbing and diving artifacts are different manifestations of the same problem, which is the interpolation of absolute water levels across dry/wet boundaries. In dry cells, no absolute water level is given. However, to interpolate within a 4×4 neighborhood around a given position, values have to be set for these cells. A seemingly good choice is to use the terrain elevation. This is consistent with wet cells, where the absolute water level is the sum of the terrain elevation and the relative water depth. However, interpolating from the water level to the terrain elevation looks wrong every time we expect the water surface to extend horizontally to a higher obstacle. Climbing occurs if the water height-field interpolation connects a higher terrain with the lower water levels. Diving at walls occurs if the interpolation connects the higher water level on the wet side of the wall with the lower terrain elevation on the dry side. Our solution to this problem is already given by the extrapolation of missing values explained in Section 4.1. For all dry cells, we extrapolate the absolute water level of all neighboring wet cells and average them. Thus, we extend the geometry of water surfaces by one cell towards dry cells, leading to nearly horizontal intersections with obstacles (Figure 3a2 and Figure 3b2).

Apart from climbing and diving situations, this extension also affects the water surface in shallow regions. This is not desired, because here it gives a false impression of the extents of inundation. We address this issue by a distinction between the surface geometry and the surface visibility during rendering. There, we generate a temporary triangle mesh from the height field, which is rasterized on the pixel grid of the screen into *fragments* by the GPU. We displace the vertices of this triangle mesh using absolute water levels, but decide visibility for each fragment using the relative water depth. For the visibility, we first introduce a threshold for the minimum water depth to display, which will typically be around one millimeter. Such a threshold is generally helpful for rendering water on top of a terrain, because it prevents z-fighting. We then calculate the relative water depth of each fragment and compare it to this threshold. If the water depth is less than the threshold, the fragment is discarded, which will happen at the dry/wet boundaries of interest. This prevents the depiction of wrong water extents in shallow regions.

A second issue caused by extending the water surface towards the terrain is floating water below the terrain if the water surface geometry intersects with the terrain. In our application, this is unpleasant, because the scene is often viewed from below, for example, for the inspection of sewer networks. Fortunately, we can also fix this issue with the threshold comparison introduced above. As already mentioned in Section 4.1, we extrapolate absolute water levels in dry cells, but not the relative water depths, which we set to zero. We can therefore calculate the relative water depth of a fragment in two different ways: Either by interpolating it directly from the simulation data, or by interpolating the absolute water levels and terrain elevations and calculating their difference. We interpret the first way as a ground truth provided by the simulation that leads to visually correct results most of the time, except in climbing situations. In this case, however, the difference between the absolute water level and the terrain elevation will be negative. Thus, calculating the relative water depth in both ways and comparing the smaller of the two values to the threshold allows us to preserve the correct extends of inundation while also removing water below the terrain.

Leaking artifacts as shown in Figure 3c1 can also be removed

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

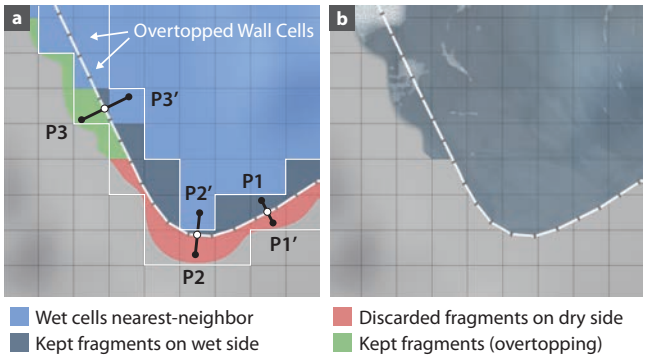

Figure 5: *Removal of leaking artifacts at walls. (a) Comparison of a wetness value for each fragment p_i and mirror position p_i'. If higher at p_i', p_i is discarded (red), unless the cell containing p_i touches an overtopped wall cell (green). (b) Result without leaking.*

by a per-fragment operation that determines on which side of a wall the fragment is located. These artifacts occur at wall boundaries, which are rasterized on the adaptive grid for the simulation. While the minimum cell size of the adaptive grid is usually one to five meters, a floodwall itself only has a thickness of approximately 15 centimeters. It is therefore not adequately represented by the rasterization, which is illustrated in Figure 5a by a white border surrounding the wall cells. Here, wet cells are colored in light blue using nearest-neighbor interpolation to demonstrate that most of the time, the wet cells output by the simulation do not even reach the wall. Bicubic interpolation is therefore needed to extend the water surface to the wall, while at the same time introducing the leaking artifact by extending the surface too much. Our approach to prevent this is to consider the exact geometry of the wall line to determine the location of the fragment relative to the wall, therefore introducing high-resolution data that was not available to the simulation and grid-based interpolation. We explain this approach with the help of Figure 5a. For now, we assume that the line geometry is already accessible from within the fragment shader.

We consider three different fragments at positions p_1, p_2, and p_3 contained by dry cells. Starting with p_1, we need to decide whether it is on the flooded side of the wall. We accomplish this by comparing an interpolated wetness value (dry $= 0$, wet $= 1$) at p_1 with the analogous value at the mirror position p_1' on the other side of the wall. We obtain p_1' by reflecting p_1 through its closest point on the wall line. At both p_1 and p_1', we use our proposed interpolation to smoothly interpolate the binary information whether a cell is wet (1) or not (0). Unless the wall is flooded from both sides or has been overtopped, this comparison value is significantly higher on the flooded side of the wall. For p_1, we keep the corresponding fragment. For p_2, its mirror position p_2' is closer to a wet cell. Therefore, the interpolated wetness value is higher at p_2' and the fragment at p_2 has to be discarded. However, if the wall is flooded from both sides or has been overtopped, the water surface should be visible on both sides of the wall. To test for this case, we check if the cell containing the fragment in question has a wet neighbor that is either a wall boundary cell or is on the same side of the wall. If this is the case (see p_3), the fragment must not be discarded.

To calculate the point closest to a fragment on any wall line, we need access to the line geometry in the fragment shader. We rasterize all wall lines on the adaptive grid in a pre-processing step. For each cell, we collect the indices of all line segments of all lines that intersect its 8-neighborhood. The sequences of segment indices of all cells are stored contiguously in a buffer. For each cell, the length and start offset of the sequence is stored. 2D vertex positions of all segments are stored contiguously in a buffer. During rendering, we iterate over all segment indices of the corresponding cell collected before. This is relatively inexpensive, as the vast majority of cells has no line segments in the proximity at all, while most of the remaining cells have less than five. The test also terminates early if the distance to the closest point on any line is greater than twice the square root of the minimum cell size (meaning the maximum distance of influence of a single cell using bicubic interpolation).

6. Water Flow Perception

If the simulation domain is slowly flooded, it is very obvious where the water is flowing. Once the terrain is inundated, however, flow behavior such as direction and speed are no longer visible from the water surface itself. Common visualization techniques such as glyphs convey this information in a very expressive way, but add to the complexity of the visualization. This is a concern, for example, if visualizations of simulation results need to be communicated to the general public. While the appearance and behavior of water are well-known and are intuitively recognized [KC14], more abstract visualization techniques are not. In situations where only subtle hints should be provided, animated waves and foam can be used as intuitive and less obtrusive alternatives [Vla10, GSH*15]. In contrast to previous work, we do not focus primarily on physically plausible water rendering. Instead, we use waves and foam as visual metaphors for flow directions and high velocities. The size of waves and the intensity of foam are deliberately exaggerated such that they are also visible from an overview perspective. Also, flooding scenarios usually span several hours or days, but are replayed in time lapse. Waves moving at physically plausible speeds as given by the simulation would not be perceivable at all in this application.

Fast tile-based approaches for small waves rely on the subdivision of the water surface into tiles, within which pre-computed wave textures are rotated and shifted according to a low-resolution flow texture. However, the pre-computed wave texture prevents dynamic wave patterns based on an underlying flow field and leads to obvious repetitions. Alternatively, for open waters, large waves can be generated by superposition of individual wave functions, which are not well-suited for locally varying flow behavior. We add directional waves synthesized from the simulated velocity field by combining the two approaches. Within tiles of different sizes corresponding to different wavelengths, we generate individual waves by a wave function that extends the simple sine function proposed by Finch [Fin04]. The extensions allow us to control the chaotic behavior and adjust the amplitude of waves according to simulation data. We provide the used formulae in Appendix A. Replacing pre-computed wave textures by these dynamic waves removes the major disadvantages of tile-based approaches. By using multiple layers of tiles of different resolution, we are also able to visualize both principal flow directions and small-scale local behavior such as turbulence.

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

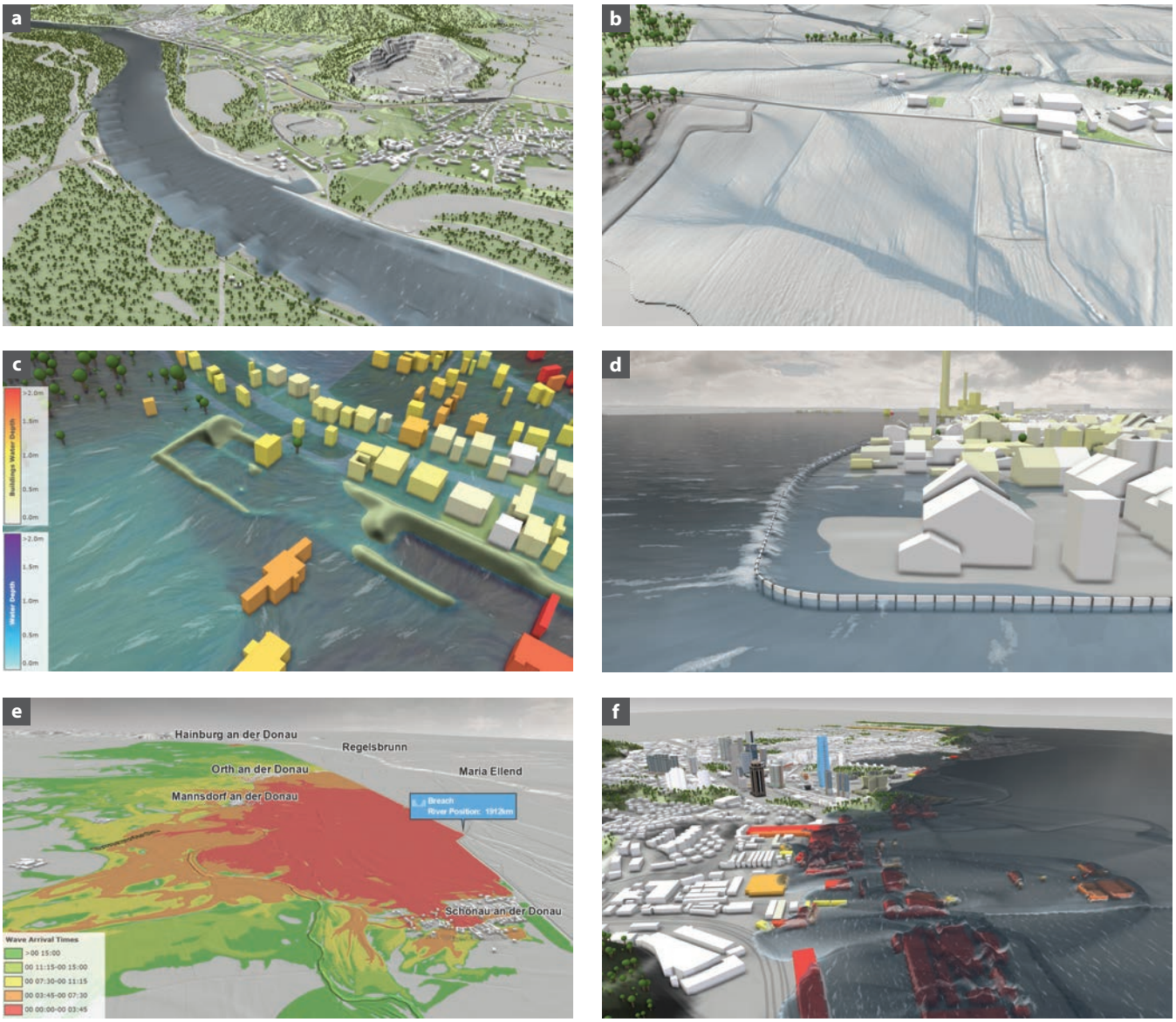

Figure 6: *Results of our proposed technique. (a) Large-scale visualization of a river valley (**C1**). (b) Stormwater runoff on a high-resolution grid (**C3**). (c) Coloring of water and buildings by water depth. (d) Failing floodwall in an urban scenario (**C5**). Foam indicates current overtopping regions. (e) Terrain overlay of wave arrival times after a dike breach (**C6**). (f) Tsunami impact on a city (**C7**).*

Based on these tiles, we also generate foam on the water surface to highlight areas of high velocity and significant changes in the velocity, i.e., high magnitudes in the velocity's gradient field. Additionally, we add foam along high waves to increase their visibility. Instead of using a wave function within the tile, we now use an animated cellular-noise function [Wor96] producing a bubble-like pattern. We use wave and velocity information to modulate the visibility of this function. As waves and foam dynamically change with the simulation data over time, we refer to the accompanying video in the supplementary material and also available online [Vid] for a demonstration.

Finally, we include visual hints of the water depth by using a transfer function [Bel04] for water color and opacity, such that inundated structures remain visible and depth information is preserved. We also apply a depth-dependent box blur [HSC*05] to refractions such that objects in deep water regions are blurred and distorted more than in shallow regions.

7. Results

In this section, we demonstrate the results of our approach in various typical use cases for flood management. Based on the working task and the specific scenario, these cases differ significantly in terms of

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

Case	Extents	Cell Sizes	# Cells	# Wet	T. Data	W. Data	Update	Terrain	Water	Artifacts	Flow Vis.
C1	17.0×19.4 km^2	3…96 m	22.58 M	0.75 M	1553 MB	1933 MB	57.8 ms	4.9 ms	3.1 ms	0.4 ms	2.7 ms
	"	"	"	5.32 M	"	"	68.8 ms	"	9.6 ms	0.7 ms	2.6 ms
C2	9.8×4.0 km^2	3 m	4.31 M	0.62 M	159 MB	211 MB	7.0 ms	2.1 ms	6.7 ms	0.1 ms	0.4 ms
C3	2.0×2.1 km^2	0.5 m	17.15 M	15.12 M	625 MB	822 MB	36.5 ms	1.7 ms	7.7 ms	0.7 ms	2.1 ms
C4	4.2×5.7 km^2	2 m	6.01 M	5.95 M	221 MB	293 MB	14.5 ms	2.7 ms	3.3 ms	0.7 ms	5.0 ms
C5	4.1×5.1 km^2	5 m	1.00 M	0.34 M	37 MB	57 MB	2.5 ms	1.7 ms	4.9 ms	0.5 ms	1.9 ms
C6	31.9×13.4 km^2	3…96 m	12.51 M	3.31 M	1067 MB	1553 MB	59.7 ms	5.8 ms	8.1 ms	0.5 ms	1.6 ms
C7	9.4×6.6 km^2	3 m	7.29 M	3.55 M	268 MB	355 MB	11.8 ms	2.2 ms	5.3 ms	0.3 ms	2.4 ms

Table 1: *Benchmarks for case studies. Columns from left to right: Name of the use case, extents of the simulation domain, cell sizes of individual levels, number of cells, number of wet cells, terrain data size, water data size, water data update time, terrain rendering time, water rendering time with tessellation and surface shading, artifact removal time, visualization time of flow properties with waves and foam.*

Case	Nearest Neighb.	Bilinear	Catmull-Rom	B-spline
C1	0.044 m	0.029 m	0.023 m	0.037 m
C2	0.503 m	0.126 m	0.101 m	0.173 m
C3	0.075 m	0.036 m	0.028 m	0.044 m
C4	1.177 m	0.345 m	0.273 m	0.481 m
C5	0.280 m	0.240 m	0.210 m	0.277 m
C6	0.199 m	0.182 m	0.166 m	0.208 m
C7	0.307 m	0.133 m	0.121 m	0.149 m

Table 2: *Accuracy of surface reconstruction. The root mean square error between the terrain surface and the original digital elevation model expresses the mean deviation from the ground truth.*

simulation grid extents, grid cell sizes, and the share of wet and dry cells. We show exemplary results of various scenarios in Figure 6 and provide the scenario parameters in Table 1 together with runtime benchmarks. For more use cases and animated results, we refer to the accompanying video [Vid]. For benchmarking, we used a system with an Intel Core i7-6700K 4 GHz CPU, 64 GB RAM, and an Nvidia GTX 1080 Ti GPU. The rendering resolution is 1920×1080.

The first use case **C1** involves the modeling of river floods of an outstanding magnitude (e.g., 100-year floods) for flood risk assessment. For this task, engineers need to frequently navigate in space and time and switch between overview and detail perspectives to investigate flood risks on a region, village, or infrastructure level. Figure 6a shows the Danube river in the Marchfeld region in Austria, with a simulation domain spanning many villages over an area larger than 300 km^2. For comparison, we provide timings for the initial state and after seven days of flooding. In this case, the large scale of the data makes the use of an adaptive grid essential. As use case **C2**, we consider a second, small-scale scenario with a regular grid. When modeling river floods, only a fraction of cells of the simulation grid is wet, which makes the amount of inundation visible from the extents of the water surface.

In contrast, stormwater and surface runoff modeling operates on high-resolution grids where almost all cells are wet. Use case **C3** is an open-air hydrological laboratory in Petzenkirchen, Austria, shown in Figure 6b. Use case **C4** shown in Figure 1, right, is a mountainous region in Tyrol, Austria. Here, it is important to visualize where water collects and forms small streams. Showing all wet surfaces at full opacity, which in the case of rainfall is the entire scene, would hide this information. This is why we reduce the opacity of the rain layer if it is below one millimeter.

The interactive planning of urban protection measures requires smaller regions and coarser grids (3–5 m resolution) to enable on-the-fly simulation. Our use case **C5** shown in Figure 6d is located in the city of Cologne, Germany. Here, a floodwall is failing and foam along the wall indicates the locations of overtopping. In such scenarios, data updates and rendering need to be fast to support fully interactive sketching of protection measures, for example.

In use case **C6**, we demonstrate the analysis of breach scenarios of a dike more than 50 km long along the Danube, again in the Marchfeld region. Engineers define different breach locations and their structure along the dike, and then navigate in both time and space to analyze the simulation results, e.g., to identify damage to villages and important infrastructure, and to check wave arrival times for evacuation planning. Figure 6e shows wave arrival times interpolated with our proposed method and visualized as terrain overlay according to a user-defined transfer function. Based on the simulation results, the engineers identify good locations for local protection barriers to protect parts of villages where possible, which are sketched as lines directly on the terrain. The robustness and construction details of these barriers are then evaluated in additional simulation runs within our system.

In our last use case **C7** shown in Figure 6f, the impact of a tsunami on a city is modeled, which results in high water velocities and complex waves. Inundated buildings are colored by their estimated damage according to a user-defined transfer function.

For all use cases, we also assessed the accuracy of our proposed surface reconstruction, provided in Table 2. The established procedure for this task is to calculate the root mean square error between the reconstructed surface and a high-resolution ground truth [Ree00, Kid03], for which we use the original digital elevation model of the terrain defined on a regular grid. Even for adaptive height fields, we could reproduce the results of Kidner [Kid03] showing that third-order interpolation results in a more accurate surface reconstruction than linear or nearest-neighbor interpolation. As expected, B-spline approximation has a higher deviation from the ground truth than exact cubic interpolation (Catmull-Rom), as it

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

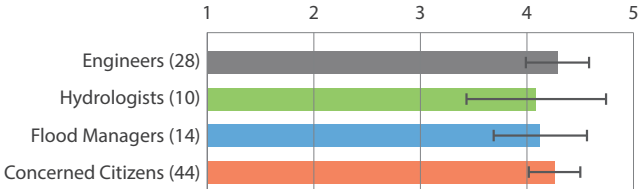

Figure 7: *Average rating results per group of participants with 95 % confidence intervals.*

does not preserve values at data points for the sake of a smoother surface.

8. Evaluation

We conduct our research in close collaboration with domain experts working in the field of flood management, hence evaluation of our results is a continuous process. In evaluation sessions prior to this work, shortcomings in the previously used visualization of water surfaces have emerged, which triggered our research for the solutions presented in this work. These solutions were evaluated qualitatively by domain experts in live sessions which focused on the differences to previously used techniques. To strengthen this with quantitative results, we conducted a survey among experts in various fields of work as well as the general public.

8.1. Live Sessions

During two separate live sessions of about one hour each, we demonstrated our solutions in different real-world flood and heavy rain scenarios to four flood management experts from two different organizations. The first expert works for the flood protection center of Cologne, Germany, a leading institution for integrated flood management in Europe [Sta]. Her typical tasks in the field of urban flood and stormwater management include heavy rain modeling, response planning, and public communication. The other three experts are engineers at the consulting agency RIOCOM [RIO] focusing on flood risk management in Austria for over 20 years. Their typical tasks include flood risk analysis and the creation of flood risk maps as well as the design of protection measures, for which they use our decision support system Visdom. For evaluation, we showed the experts various prepared flooding scenarios in Visdom and asked them to fulfill specific tasks. The first task was the subjective assessment of the usefulness and quality of our proposed techniques for the use cases discussed in Section 7. For the second task, we showed the experts the same scenario with different interpolation methods as well as with and without common interpolation artifacts. We asked them to compare the different visualizations and indicate their preference. Finally, we asked them to identify local flow directions as well as deep and shallow regions of inundated regions from an overview perspective.

After being shown results of nearest-neighbor, bilinear, and bicubic interpolation as well as B-spline approximation, all four experts concurred that B-spline approximation led to the most aesthetic results and was also the best surface reconstruction to use for their working tasks. The engineers said that the rectangular structures of nearest-neighbor interpolation were unpleasant to look at and carried no valuable information. The stormwater expert liked to have several options. She preferred the smooth B-spline approximation for planning work, presentations, and public communication, as it results in natural surfaces and people are more familiar with continuous regions than with blocks. However, she would use nearest-neighbor interpolation for comparability with results of other software and for communication with other domain experts.

Our strategies to avoid climbing, diving, and leaking artifacts were well received by all experts. One expert initially perceived leaking artifacts as a visualization feature to indicate seeping of water through leaky barriers. The experts agreed that our visualization without leaking artifacts was unambiguous in that regard. Likewise, they appreciated the removal of climbing and diving artifacts, stating that it leads to a more realistic depiction of the water surface. While climbing just looked wrong, one expert said that diving artifacts could be misinterpreted as a large approaching wave.

All four experts welcomed our visualization of flow directions and high velocities with animated waves and foam. They called the use of waves and foam as visual metaphors aesthetic and intuitive and said they were good indicators of the flow strength and direction. All experts were able to quickly identify principal flow directions in an already inundated area from an overview perspective. When shown arrow glyphs instead of waves to visualize the velocities, one expert stated they were useful for still images, but she would prefer the waves for animation and videos.

8.2. Online Survey

For a quantitative evaluation of our proposed techniques, we conducted an online survey among 96 participants.

Participants. We reached out to experts in various fields of work related to flood management with the help of our collaboration partners and provided them with a questionnaire. As one important task of flood management is public communication, we also evaluated our results with members of the general public. Specifically, we asked the users of the commercial flood alert system Pegel-Alarm [Peg] for participation, assuming that this user group consists of people particularly concerned about floods. For the presentation of the survey results, we grouped the domain experts roughly by profession, for which we asked them to state their profession and areas of responsibility in the survey. In summary, the 96 participants included 28 experts working in civil and hydraulic engineering, 10 scientists in the field of hydrology, 14 flood management experts of public authorities, and 44 concerned citizens (Figure 7).

Questionnaire. We asked all participants of our survey to fill out a questionnaire that is still available online [Sur]. It contained eight short videos (37 s to 1 min 55 s) showing visualizations of real-world scenarios within the Visdom decision support system [Vis]. For each video, the participants were asked to focus on the flood visualization, which is why almost all user interface elements of the application were hidden. In detail, the videos showed river floods in rural and urban regions (**Q1**), heavy rains and stormwater runoff (**Q2**), floods caused by floodwall overtopping (**Q3**), by a floodwall breach (**Q4**), and by dike breaks (**Q5**), the interactive planning of a short-term object protection with sandbags (**Q6**), the interactive planning of

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

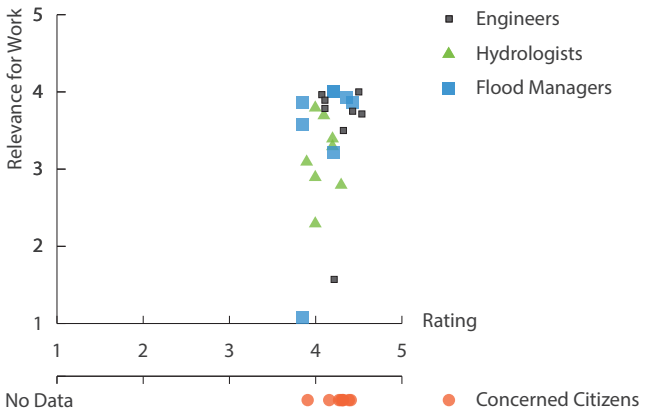

Figure 8: *Evaluation results per group of participants. Each data point represents the average result of one question within the group.*

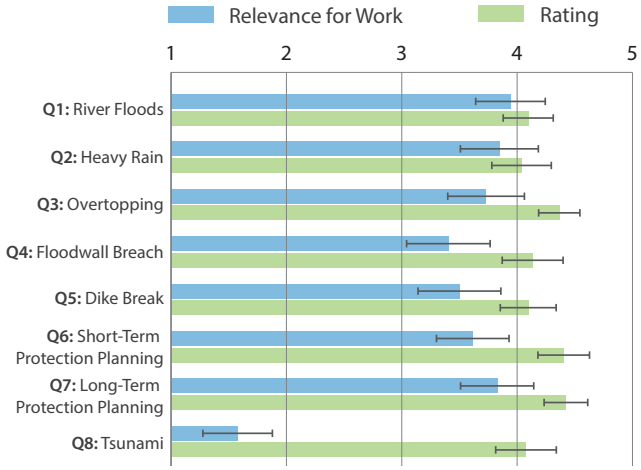

Figure 9: *Evaluation results per question averaged over the results of all domain experts with 95 % confidence intervals.*

long-term protection measures with mobile floodwalls and retention basins (**Q7**), and the impact of a tsunami on a city (**Q8**). After each video, the participants assessed the visualization of the flooding on a scale of 1 (very bad) to 5 (very good). The questionnaire distributed among the domain experts additionally asked to state the relevance of the visualization for their work-related areas of responsibility on a scale of 1 (not relevant) to 5 (very relevant). Below each video, additional comments could be provided.

Results. We provide the detailed results of the survey in the supplementary material. In Figure 7, we give an overview of the groups of participants and their average rating over all questions. On average, our visualizations were well received by all groups and were rated *good*. The favorable ratings of both domain experts and concerned citizens suggest a high suitability of our visualizations for both technical tasks and public communication.

In Figure 8, we relate the average rating of each question to the average relevance for the participants' work, separated by group. For the group of concerned citizens, we collected no data on the relevance for their work. Engineers and flood managers assessed the visualizations as both *good* and *relevant* for their work, which we see as a particularly important result of this evaluation, as the experts in these fields are the primarily intended users of our proposed techniques. As most of the scenarios included in the survey deal with flood management tasks, the hydrologists expectedly considered not all of them relevant for their work, but still good. As most of our participants are located in Central Europe, the relevance of one scenario was considered very low by all groups, which is the modeling of a tsunami (**Q8**). Yet, even visualizations not particularly relevant have been rated highly by the participants.

Figure 9 shows the average relevance and rating by all experts for each question. This reveals that the visualizations in the context of planning tasks (**Q6** and **Q7**) have been rated best. We attribute this to the interactivity shown in the videos, which demonstrates the benefit of expressive water visualization for interactive tasks.

Besides the quantitative evaluation, the additional comments by some of the participants of the survey provide valuable qualitative statements. In general, they received our visualizations very well and praised them as aesthetic, realistic, and descriptive. Eight participants saw a benefit of our visualizations for specific applications, most of all for public communication, but also for planning and coordination. One participant saw little benefit of our visualization over established 2D methods. 13 participants criticized the chosen color of water and the lack of contrast to the terrain color in the videos. While these colors can be changed freely by the user within our application (e.g., transfer-function-based coloring in Figure 6c), we consider the study of proper default values an important aspect for future research.

Three participants explicitly stated that the waves and foam helped them identifying flow directions, one participant still could not identify them all the time. One participant made us aware that on mobile devices, the flow directions were not discernible at all, which is an issue we have to address in the future.

Six participants praised the combination of the visualization of water with other visualization techniques, in particular the coloring of buildings according to their inundation. Nine participants mentioned combinations with other visualizations that could improve the result, such as driftwood, street names, detailed buildings, and a time line. Our application supports most of these visualizations, which have not been shown in order to draw the participants' attention to the flood visualization for evaluation. Yet, synergies between different visualization techniques and the decision which ones to always show are interesting aspects that we will further investigate.

9. Conclusion

In this paper, we present a real-time technique to visualize adaptive water height fields without misleading interpolation artifacts. The adaptive reconstruction scheme treats different levels of the adaptive grid as separate height fields that are combined into a globally C^1-continuous height field. It is efficient enough to be used in interactive applications even for very large data sets. Thus, it fills the gap between fast linear interpolation and slow smooth interpolation. The removal of artifacts counters shortcomings of grid-based interpola-

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

tion of water height fields by two means. First, by extrapolating the water surface at dry/wet boundaries in a physically plausible way, and, second, by incorporating detailed geometric information in the surface reconstruction that is not available in the regular height field data. A visualization of important flow properties by waves and foam is applied to the water surface to support the interpretation of simulation results. The resulting visualizations have been positively evaluated by domain experts and the general public.

Currently, our proposed surface reconstruction is restricted to data structures with at most one level difference between neighboring cells. Future work is necessary to investigate how to remove this restriction. After reconstruction with bicubic interpolation or approximation, height-field structures not aligned with the interpolation grid often exhibit staircase artifacts. These artifacts might be reduced by using rotationally invariant reconstruction filters instead of a dimension-wise application of the cubic filter. We therefore consider the combination of fast bicubic interpolation and the computationally more expensive thin plate splines a possible direction for future research. The feedback provided by the participants of our survey indicates varying effectiveness of preset visualization parameters as well as specific combinations of visualization techniques. This issue also requires further research and evaluation.

10. Acknowledgments

VRVis is funded by BMVIT, BMDW, Styria, SFG and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174) which is managed by FFG. We thank RIOCOM, SOBOS, the Stadtentwässerungsbetriebe Köln, AöR, and all participants of our online survey.

References

[Bel04] BELYAEV V.: Real-time rendering of shallow water. In *GraphiCon Proceedings* (Moscow, 2004), GraphiCon Scientific Society, pp. 1–6. 3, 9

[BHMF08] BEYER J., HADWIGER M., MÖLLER T., FRITZ L.: Smooth mixed-resolution GPU volume rendering. In *Proc. Fifth Eurographics/IEEE VGTC Conference on Point-Based Graphics* (Aire-la-Ville, 2008), Eurographics Association, pp. 163–170. 3

[BLE*14] BROVKA M., LÓPEZ J. I., ESCOBAR J. M., CASCÓN J. M., MONTENEGRO R.: Construction of polynomial spline spaces over quadtree and octree T-meshes. *Procedia Engineering 82* (2014), 21–33. 3

[BMA10] BEUTEL A., MØLHAVE T., AGARWAL P. K.: Natural neighbor interpolation based grid DEM construction using a GPU. In *Proc. 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems* (New York, 2010), ACM, pp. 172–181. 2, 3

[Bob08] BOBACH T.: *Natural Neighbor Interpolation - Critical Assessment and New Contributions*. PhD thesis, TU Kaiserslautern, 2008. 3

[BOR14] BEATSON R., ONG W., RYCHKOV I.: Faster fast evaluation of thin plate splines in two dimensions. *Journal of Computational and Applied Mathematics 261* (2014), 201–212. 2, 3

[BST15] BOROVIKOV D., SOKOLOV I. V., TÓTH G.: An efficient second-order accurate and continuous interpolation for block-adaptive grids. *Journal of Computational Physics 297* (2015), 599–610. 3

[BSW10] BAGAR F., SCHERZER D., WIMMER M.: A layered particle-based fluid model for real-time rendering of water. *Computer Graphics Forum 29*, 4 (2010), 1383–1389. 3

[Che13] CHENG T.: Accelerating universal kriging interpolation algorithm using CUDA-enabled GPU. *Computers & Geosciences 54* (2013), 178–183. 3

[CKS*15] CORNEL D., KONEV A., SADRANSKY B., HORVÁTH Z., GRÖLLER E., WASER J.: Visualization of object-centered vulnerability to possible flood hazards. *Computer Graphics Forum 34*, 3 (2015), 331–340. 3

[CM10] CHENTANEZ N., MÜLLER M.: Real-time simulation of large bodies of water with small scale details. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, 2010), Eurographics Association, pp. 197–206. 3

[DB12] DUPUY J., BRUNETON E.: Real-time animation and rendering of ocean whitecaps. In *SIGGRAPH Asia Technical Briefs* (New York, 2012), ACM, pp. 15:1–15:3. 3

[DCL*08] DENG J., CHEN F., LI X., HU C., TONG W., YANG Z., FENG Y.: Polynomial splines over hierarchical T-meshes. *Graphical models 70*, 4 (2008), 76–86. 3

[FBHD17] FUCHS F. G., BARROWCLOUGH O. J. D., HJELMERVIK J. M., DAHL H. E. I.: Direct interactive visualization of locally refined spline volumes for scalar and vector fields. *arXiv e-prints abs/1707.01170* (2017), 1–11. 2, 3

[Fin04] FINCH M.: Effective water simulation from physical models. In *GPU Gems*, Fernando R., (Ed.). Addison-Wesley, Boston, 2004, pp. 5–29. 3, 8, 15

[Flo] FloodViz - Visual analytics for assessment and interpretation of simulated river flooding. https://www.gri.msstate.edu/research/floodviz/ (last visited on April, 9th 2019). 3

[Fra82] FRANKE R.: Smooth interpolation of scattered data by local thin plate splines. *Computers & Mathematics with Applications 8*, 4 (1982), 273–281. 3

[GH12] GONZALEZ-OCHOA C., HOLDER D.: Water technology of Uncharted. Game Developers Conference, 2012. 3, 15

[GJAG14] GUTIÉRREZ DE RAVÉ E., JIMÉNEZ-HORNERO F. J., ARIZA-VILLAVERDE A. B., GÓMEZ-LÓPEZ J. M.: Using general-purpose computing on graphics processing units (gpgpu) to accelerate the ordinary kriging algorithm. *Computers & Geosciences 64* (2014), 1–6. 2, 3

[Goo00] GOOVAERTS P.: Geostatistical approaches for incorporating elevation into the spatial interpolation of rainfall. *Journal of Hydrology 228*, 1 (2000), 113–129. 3

[Gri11] GRIMES B.: Making and using non-standard textures: Manipulating UVs through color data in Portal 2. Game Developers Conference, 2011. 3, 15

[GSH*15] GROTTEL S., STAIB J., HEYER T., VETTER B., GUMHOLD S.: Real-time visualization of urban flood simulation data for non-professionals. In *Workshop on Visualisation in Environmental Sciences (EnvirVis)* (Aire-la-Ville, 2015), Eurographics Association, pp. 37–41. 3, 8

[HPW*16] HORVÁTH Z., PERDIGAO R. A., WASER J., CORNEL D., KONEV A., BLÖSCHL G.: Kepler shuffle for real-world flood simulations on GPUs. *The International Journal of High Performance Computing Applications 30*, 4 (2016), 379–395. 1, 2, 3

[HSC*05] HENSLEY J., SCHEUERMANN T., COOMBE G., SINGH M., LASTRA A.: Fast summed-area table generation and its applications. *Computer Graphics Forum 24*, 3 (2005), 547–555. 9

[Hut95] HUTCHINSON M. F.: Interpolating mean rainfall using thin plate smoothing splines. *International Journal of Geographical Information Systems 9*, 4 (1995), 385–403. 3

[HWP*15] HORVÁTH Z., WASER J., PERDIGÃO R. A. P., KONEV A., BLÖSCHL G.: A two-dimensional numerical scheme of dry/wet fronts for the saint-venant system of shallow water equations. *International Journal for Numerical Methods in Fluids 77*, 3 (2015), 159–182. 2

[JW15] JESCHKE S., WOJTAN C.: Water wave animation via wavefront parameter interpolation. *ACM Transactions on Graphics 34*, 3 (2015), 27:1–27:14. 3

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

[JW17] JESCHKE S., WOJTAN C.: Water wave packets. *ACM Transactions on Graphics 36*, 4 (2017), 103:1–103:12. 3

[KC14] KRYVEN M., COWAN W.: What does water look like? In *Proceedings of the Workshop on Computational Aesthetics* (New York, 2014), ACM, pp. 53–56. 3, 8

[KH02] KÄHLER R., HEGE H.-C.: Texture-based volume rendering of adaptive mesh refinement data. *The Visual Computer 18*, 8 (2002), 481–492. 3

[Kid03] KIDNER D. B.: Higher-order interpolation of regular grid digital elevation models. *International Journal of Remote Sensing 24*, 14 (2003), 2981–2987. 3, 10

[KLCK17] KIM J. H., LEE J., CHA S., KIM C. H.: Efficient representation of detailed foam waves by incorporating projective space. *IEEE Transactions on Visualization and Computer Graphics 23*, 9 (2017), 2056–2068. 3

[KT09] KIM B., TSIOTRAS P.: Image segmentation on cell-center sampled quadtree and octree grids. In *Wavelet Applications in Industrial Processing VI* (Bellingham, 2009), vol. 7248, SPIE, pp. L:1–L:9. 2, 3, 4, 5

[LCKD16] LI X., CHEN F., KANG H., DENG J.: A survey on the local refinable splines. *Science China Mathematics 59*, 4 (2016), 617–644. 3

[Lia11] LIANG Q.: A structured but non-uniform cartesian grid-based model for the shallow water equations. *International Journal for Numerical Methods in Fluids 66*, 5 (2011), 537–554. 2, 3

[LJL13] LEE H., JEONG Y., LEE S.: Recursive tessellation. In *ACM SIGGRAPH Asia Posters* (New York, 2013), ACM, p. 16:1. 7

[LKT*17] LESKENS J. G., KEHL C., TUTENEL T., KOL T., HAAN G. D., STELLING G., EISEMANN E.: An interactive simulation and visualization tool for flood analysis usable for practitioners. *Mitigation and Adaptation Strategies for Global Change 22*, 2 (2017), 307–324. 3

[ME11] MORAN P., ELLSWORTH D.: Visualization of AMR data with multi-level dual-mesh interpolation. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 1862–1871. 3

[MG06] MIN C., GIBOU F.: A second order accurate projection method for the incompressible Navier-Stokes equations on non-graded adaptive grids. *Journal of Computational Physics 219*, 2 (2006), 912–929. 3

[MIK] MIKE FLOOD - Toolbox for professional flood modellers. https://www.mikepoweredbydhi.com/products/mike-flood (last visited on April, 9th 2019). 3

[MN88] MITCHELL D. P., NETRAVALI A. N.: Reconstruction filters in computer graphics. *ACM SIGGRAPH Computer Graphics 22*, 4 (1988), 221–228. 6

[NSB13] NIELSEN M. B., SÖDERSTRÖM A., BRIDSON R.: Synthesizing waves from animated height fields. *ACM Transactions on Graphics 32*, 1 (2013), 2:1–2:9. 3

[PA01] PREMOŽE S., ASHIKHMIN M.: Rendering natural waters. *Computer Graphics Forum 20*, 4 (2001), 189–200. 3

[Peg] PegelAlarm Bürgerservice, Hochwasser-Warndienst. https://www.pegelalarm.at (last visited on April, 9th 2019). 11

[Ree00] REES W. G.: The accuracy of digital elevation models interpolated to higher resolutions. *International Journal of Remote Sensing 21*, 1 (2000), 7–20. 3, 5, 10

[RIO] RIOCOM - flowing competence. http://riocom.at/en/ (last visited on April, 9th 2019). 11

[Riv] RiverFlow2D - Two-dimensional combined hydraulic and hydrologic flexible-mesh model. http://www.hydronia.com/riverflow2d/ (last visited on April, 9th 2019). 3

[SH05] SIGG C., HADWIGER M.: Fast third-order texture filtering. In *GPU Gems 2*, Pharr M., (Ed.). Addison-Wesley, Boston, 2005, pp. 313–329. 6

[Sha] Shadertoy - Adaptive grid interpolation. http://shadertoy.com/view/WsXXRf (last visited on April, 9th 2019). 4

[Sta] Stadtentwässerungsbetriebe Köln. http://www.steb-koeln.de/ (last visited on April, 9th 2019). 11

[Sur] Survey to evaluate a novel 3D flood and stormwater visualization. https://goo.gl/forms/DqrLM6cCkroWLs9q2 (last visited on April, 9th 2019). 11

[Tes01] TESSENDORF J.: Simulating ocean water. *Simulating Nature: Realistic and Interactive Techniques, ACM SIGGRAPH Course #47 Notes* (2001), 3:1–3:19. 3

[TUF] TUFLOW - Numerical engines for simulating free-surface water flow. http://www.tuflow.com/ (last visited on April, 9th 2019). 3

[VGB*16] VAN ACKERE S., GLAS H., BEULLENS J., DERUYTER G., DE WULF A., DE MAEYER P.: Development of a 3D dynamic flood WebGIS visualisation tool. *International Journal of Safety and Security Engineering 6*, 3 (2016), 560–569. 3

[vH11] VAN HOESEL F.: Tiled directional flow. In *ACM SIGGRAPH Posters* (New York, 2011), ACM, p. 19:1. 3

[Vid] Flood and stormwater modelling - Hydrodynamic modelling with VISDOM. https://www.youtube.com/watch?v=GBP97uc7eTk (last visited on April, 9th 2019). 9, 10

[Vis] Visdom - Combining simulation and visualization. http://visdom.at (last visited on April, 9th 2019). 3, 11

[Vla10] VLACHOS A.: Water flow in Portal 2. Advances in Real-Time Rendering in 3D Graphics and Games II course, ACM SIGGRAPH, 2010. 3, 8

[WKL*01] WEBER G. H., KREYLOS O., LIGOCKI T. J., SHALF J. M., HAGEN H., HAMANN B., JOY K. I., MA K.-L.: High-quality volume rendering of adaptive mesh refinement data. In *Proc. Vision Modeling and Visualization Conference* (Berlin, 2001), Aka GmbH, pp. 121–128. 3

[WKS*14] WASER J., KONEV A., SADRANSKY B., HORVÁTH Z., RIBIČIĆ H., CARNECKY R., KLUDING P., SCHINDLER B.: Many Plans: Multidimensional ensembles for visual decision support in flood management. *Computer Graphics Forum 33*, 3 (2014), 281–290. 1

[Wor96] WORLEY S.: A cellular texture basis function. In *Proc. 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, 1996), ACM, pp. 291–294. 9

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMANN K., ERTL T.: Level-of-detail volume rendering via 3D textures. In *Proc. IEEE Symposium on Volume Visualization* (New York, 2000), ACM, pp. 7–13. 3

[YHK07] YUKSEL C., HOUSE D. H., KEYSER J.: Wave particles. *ACM Transactions on Graphics 26*, 3 (2007), 99:1–99:8. 3

[YNBH09] YU Q., NEYRET F., BRUNETON E., HOLZSCHUCH N.: Scalable real-time animation of rivers. *Computer Graphics Forum 28*, 2 (2009), 239–248. 3

[You] You Are Here: Mapping how sea level rise and flooding will affect your home. http://seagrant.gso.uri.edu/you-are-here/ (last visited on April, 9th 2019). 3

Appendix A: Wave Synthesis

Waves are usually synthesized by superposition of individual wave functions with different wavelengths and amplitudes. The parameters are often defined manually by artists or are controlled by probabilistic models. In contrast, we derive wave properties directly from the velocity field output by our simulation. The individual steps of our approach are illustrated in Figure 10. For multiple wavelengths, the spatial domain is subdivided into tiles over which an average velocity is calculated (Figure 10a). According to the velocity, sine waves are oriented inside the tile (Figure 10b), which leads to discontinuities at tile borders. This process is performed for four

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.

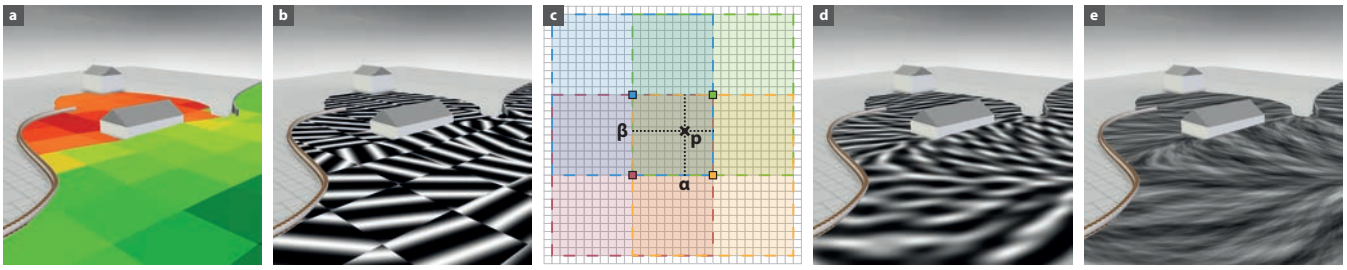

Figure 10: *Overview of tile-based wave synthesis. (a) Average velocity of each tile for a single wavelength, normalized for display. (b) Sine waves of a single wavelength oriented towards flow direction. (c) Layout of four overlapping tiles around sample position p for seamless blending of values between tiles. (d) Blended waves of a single wavelength. (e) Superposition of sine waves of multiple wavelengths for vertex displacement.*

partially overlapping tiles (Figure 10c) to get four different wave heights. The final wave height is interpolated between these four wave heights, which eliminates the discontinuities (Figure 10d). The wave heights of all considered wavelengths are summed up to a final displacement value (Figure 10e) applied to triangle vertices of the water surface.

Our approach combines tile-based wave synthesis by texture advection [Gri11, GH12] with dynamic superposition of the simple wave functions used by Finch [Fin04] to avoid texture-related artifacts. For n different wavelengths $\lambda_i = r_i 2^i \lambda$, we define a tile over which a wave function is evaluated. λ is the shortest considered wavelength and $r_i \in [0.8, 1.0]$ is a random value preventing exact frequency doubling. We chose $n = 5$ and $\lambda = 0.5$ m empirically. Given a world-space position $p = (x, y)$ at time step t, we define a tile $T = [x_0, x_1] \times [y_0, y_1]$:

$$(x_0, y_0) = 2\lambda_i \left\lfloor \frac{(x,y)}{2\lambda_i} \right\rfloor, \quad (x_1, y_1) = (x_0, y_0) + (2\lambda_i, 2\lambda_i). \tag{4}$$

This tile is shaded in blue in the upper left of Figure 10c. Additionally to this tile, we consider the green tile shifted horizontally by λ_i, the red tile shifted vertically by λ_i, and the orange tile shifted both horizontally and vertically by λ_i, for which we evaluate the wave function separately. Over each of these tiles, we require an averaged velocity v_i as well as the average flow direction $d_i = v_i / \|v_i\|$.

The wave height at world-space position $p = (x, y)$ at time step t for n different wavelengths is calculated as

$$H(x,y,t) = \sum_{i=0}^{n-1} A_i W_i(x,y,t) \tag{5}$$

with the wave amplitude

$$A_i = \frac{\|v_i\|}{\|v_{\max}\|} \frac{w_i}{\sum_{j=0}^{n-1} w_j}, \quad w_i = \left(\frac{1}{2} + 1\right)^{-i}, \tag{6}$$

and wave function

$$W_{i,T}(x,y,t) = 2\left(\frac{\sin(-\theta_i \omega_i + t\varphi_i + r_T 2\pi) + 1}{2}\right)^2 - 1 \tag{7}$$

for each tile T. $\theta_i = \langle d_i, (x,y) \rangle$ is the distance of the wave traveled along the average flow direction and $\omega_i = 2\pi/\lambda_i$ is the wave frequency. $\varphi_i = \min(r_i \omega_i, \pi/2)$ is the phase constant, which controls the traveling speed of the wave. $r_T \in [-0.1, 0.1]$ is a random phase offset of tile T that is used to introduce wave irregularities between neighboring tiles. Amplitude A_i depends on the average velocity v_i normalized by a maximum velocity $v_{\max}$, which is either provided by the user or derived from the velocity field. As a result, waves are higher in high-velocity regions, and there are no waves in regions at rest.

Tiling introduces discontinuities at the tile borders (see Figure 10b), which need to be treated to avoid artifacts in the visualization. This is why for each (x,y,t), we evaluate the four wave functions $W_{i,T}(x,y,t)$ in the four tiles covering $p = (x,y)$ and blend them such that the weight of each tile smoothly fades from one at its center to zero at its borders. Figure 10c illustrates the blending in the overlapping region of the four tiles at p, which is a bilinear interpolation with the horizontal and vertical weights α and β, respectively:

$$\alpha = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{x}{2\lambda_i} 2\pi\right), \quad \beta = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{y}{2\lambda_i} 2\pi\right). \tag{8}$$

The blended result $W_i(x,y,t)$ for a single wavelength is shown in Figure 10d. The final wave height $H(x,y,t)$ over all wavelengths used for vertex displacement is shown in Figure 10e.

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.