Large-Scale Liquid Simulation on Adaptive Hexahedral Grids

Florian Ferstl, Rüdiger Westermann, and Christian Dick

Abstract—Regular grids are attractive for numerical fluid simulations because they give rise to efficient computational kernels. However, for simulating high resolution effects in complicated domains they are only of limited suitability due to memory constraints. In this paper we present a method for liquid simulation on an adaptive octree grid using a hexahedral finite element discretization, which reduces memory requirements by coarsening the elements in the interior of the liquid body. To impose free surface boundary conditions with second order accuracy, we incorporate a particular class of Nitsche methods enforcing the Dirichlet boundary conditions for the pressure in a variational sense. We then show how to construct a multigrid hierarchy from the adaptive octree grid, so that a time efficient geometric multigrid solver can be used. To improve solver convergence, we propose a special treatment of liquid boundaries via composite finite elements at coarser scales. We demonstrate the effectiveness of our method for liquid simulations that would require hundreds of millions of simulation elements in a non-adaptive regime.

Index Terms—Fluid simulation, finite elements, octree, multigrid

1 Introduction

Liquid simulation on regular grids has a long tradition in computer graphics. Such techniques are attractive because of the implicit encoding of topology by the grid, giving rise to efficient computational kernels for simulating the liquid's motion. Especially when paired with efficient solvers for the pressure Poisson equation, such as linear time-complexity geometric multigrid solvers [1], the projection step for enforcing the incompressibility constraint on the velocity field is possible at high rates. By using data structures like OpenVDB [2], regular grids can be dynamically restricted to the fluid domain, which reduces memory requirements in comparison to static grids that cover the whole simulation domain. This makes regular grids appealing in scenarios where violent flows with fragmentation and large deformations cover only a small area of the simulation domain. Since the number of computational elements is proportional to the liquid volume, in such scenarios a good ratio between resolution and memory is achieved.

On the other hand, keeping the number of computational elements proportional to the liquid body is not sufficient when large bodies of liquid are simulated at high resolution. This is demonstrated in Figure 1, where even a regular grid restricted to the liquid domain requires about 130 millions of cells, imposing severe memory requirements. Losasso and co-workers [3] addressed this problem via a first order accurate finite difference (FD) scheme on an adaptive octree grid. It supports an adaptive coarsening of the computational elements, yet it was shown recently that it introduces velocity oscillations due to the special treatment of hanging nodes at the coarse-to-fine grid interface [4]. Zhu et al. [5] introduced a FD scheme on a rectilinear grid that enlarges the cells in the far field surrounding a region of interest. This approach avoids hanging nodes, but it can less flexibly adapt the cell size locally due to the regular structure imposed by a rectilinear grid.

Unstructured grids, on the other hand, can adapt flexibly to local flow features and the changing liquid domain [6], [7], [8], [4]. However, they cannot take full advantage of efficient hierarchical solvers because no canonical coarse versions of such a grid exist in general. Furthermore, additional workload is required to generate an unstructured grid from a given set of vertices or regular cells, and to build the algebraic equations from the unstructured discretization.

In this work we shed light on the question whether techniques based on regular grids can be realized in much the same adaptive, and thus memory efficient way as techniques based on unstructured grids, yet keeping the computational advantages of a regular grid. We focus on the simulation of liquids at extreme effective resolutions, and we therefore strive for a solution that a) uses computational elements only where the liquid is present, b) uses ever coarser simulation resolutions in the liquid body to achieve an element number proportional to the surface area, c) inherits the efficient computational kernels of regular grids, and d) is amenable to geometric multigrid solvers to achieve high convergence rates. To the best of our knowledge, grid-based liquid simulation approaches combining all these properties have not been proposed so far.

• *Florian Ferstl, Rüdiger Westermann, and Christian Dick are with the Computer Graphics and Visualization Group, Technische Universität München, Germany.*
E-mail: ferstlf@in.tum.de, westermann@tum.de, dick@tum.de.

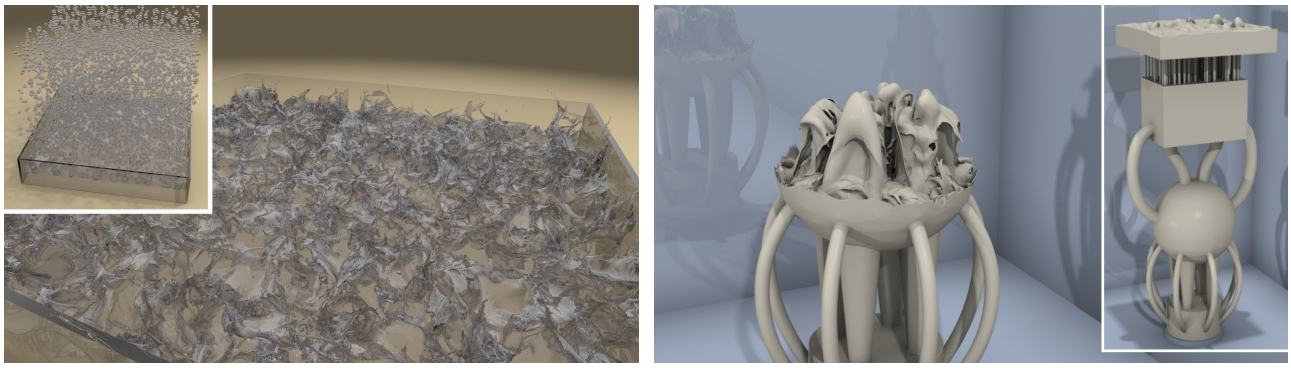

Fig. 1. Liquid flows using 34 (left) and 13 (right) million multi-resolution finite elements are simulated on an 8-core single node system with 64 GB main memory. A uniform hexahedral grid at the same effective resolution and restricted to the liquid domain would consist of about 130 million cells in both examples.

1.1 Contribution

To achieve our goals, we make the following specific contributions:

- We present a multiresolution hexahedral finite element method including a special treatment of hanging vertices, combined with a second order accurate representation of the free liquid surface.
- We make the number of finite elements proportional to the liquid boundary by using an octree grid that adapts spatially to the surface.
- We embed the finite element hierarchy into a geometric multigrid solver to achieve optimal convergence rates, and we use an adaptive variational formulation as well as element duplication to represent the domain at coarser scales.
- We propose a cell-based formulation for both the finite element method and the multigrid solver by exploiting that most elements share the same generic element matrix. This formulation enables us to alleviate memory bandwidth limitations, leading to speed-up factors between 2 and 3 compared to a stencil-based formulation.

Some achievements of our method are demonstrated in Figure 1. To demonstrate the different element resolutions that were used in the simulations, Figure 2 illustrates the octree grid structure for a part of the multi-stage water fountain on a 2D slice. The liquid surface steers the evolution of the hierarchical grid over time (see Figure 3). The surface is represented by means of a surface tracking method such as the particle level-set method by Enright et al. [9].

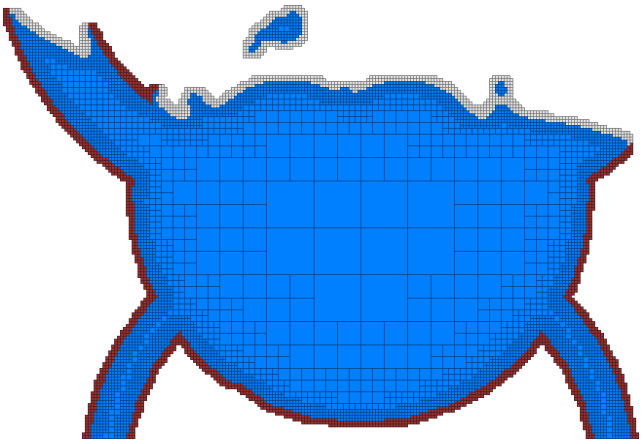

Fig. 2. Slice through the 3D octree grid the multi-stage water fountain in Figure 1 (right). A coarser version is shown for better readability. Cells are classified into fluid (blue), empty (gray), and solid (red). Note that the grid refinement is also affected by cells in front of and behind the shown layer of cells.

2 Related Work

Eulerian liquid simulation methods discretize the simulation domain using a fixed grid and compute the liquid's movement through this grid. In particular the use of regular hexahedral grid structures in combination with finite difference schemes to transform the governing equations into systems of difference equations has a long tradition in computer animation, see, for instance, [10], [11], [12], [13]. Let us also refer to the book by Bridson [14], which provides a thorough overview of grid-based approaches for liquid simulation. To efficiently generate detailed liquid surfaces from Eulerian simulations, surface tracking mechanisms making use of implicit level-sets and additional tracker particles [13], [15], [9] as well as explicit approaches using semi-Lagrangian advection of distance functions [16] and surface meshes [17], [18], [19] have been proposed. A good overview of the different tracking mechanisms used can be found in the course notes by Wojtan et al. [20].

Regular grids are appealing because they give rise to efficient numerical stencils and fast computational solvers. In particular multigrid methods [21] have become popular in computer animation due their linear time-complexity in the number of computational elements. The potential of geometric multigrid schemes for projecting the velocity field to a diver-

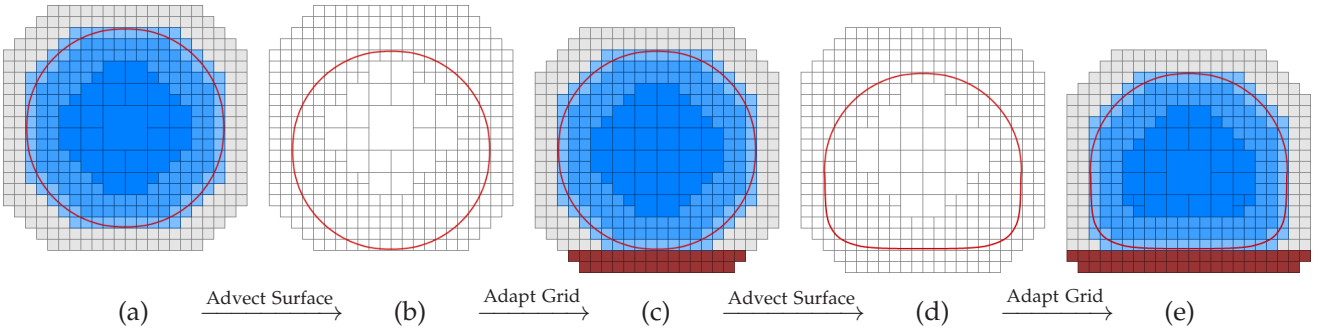

Fig. 3. Illustration of the evolution of the adaptive octree grid for the simulation of a falling water drop: (a,c,e) Grid and liquid boundary (red curve) in three consecutive time steps. Solid cells only appear in the grid when the liquid's surface comes close to an obstacle. Three different shades of blue indicate fluid cells intersected by the surface, fluid cells within the refinement band around the surface, and interior fluid cells. (b,d) Surface and grid (with unclassified cells) after surface advection, but before grid adaptation.

gence free state has been employed by McAdams et al. [1], and Chentanez and Müller [22], [23]. Only few approaches have addressed adaptivity in liquid simulations using regular grids. Varying resolutions have been accommodated via an octree grid [3], by enforcing incompressibility locally from a divergence-free field on a coarse resolution grid [24], and by using simultaneously a coarse grid for representing the liquid body and a fine grid near the surface [23]. The most recent approach by Zhu et al. [5] makes use of a rectilinear grid to fade out the grid resolution from a prescribed focus region.

Instead of using regular grids, in a number of previous approaches unstructured grids have been used to enforce a sharp representation of the liquid interface and thus to effectively restrict the computational workload to the liquid domain [25], [26], [6], [7], [8]. The grids are either re-meshed to the liquid boundary in every simulation step, or they are used in Arbitrary Lagrangian-Eulerian (ALE) methods where the grid vertices are moved according to the fluid motion [27], eventually requiring re-meshing when deformations become large. Recently, Ando and co-workers [4] proposed the integration of a tetrahedral background grid into FLIP. Here, a quasi regular, yet adaptive tetrahedralization was constructed from the dual of a multi-level hexahedral grid. FLIP [28], [29], is a hybrid method employing an auxiliary background grid for velocity projection and using particle-based transport. We mention FLIP here because the use of an adaptive regular grid as proposed in our work is promising for reducing the number of required FLIP particles in the liquid interior.

3 Creeping Octree Grid

We start with a description of the spatially adaptive octree grid used in our method. The grid is constructed in a way that does not require any initial domain specification. It is sufficient to define the size h (i.e., the side-length) of the smallest hexahedral cells to be used in the simulation. All grid cells of size $2^k h$ are aligned on an imaginary lattice with lattice points $(2^k h)\mathbb{Z}^3$, where $k = 0, 1, \ldots$ is referred to as the octree level of the respective cells.

Based on these imaginary lattices, we build the grid such that only the liquid body is covered by grid cells, except for an additional thin refinement band around the liquid boundary (both the fluid/air and fluid/solid interface). The enclosing space is *not* represented. At the beginning of the simulation, the grid is constructed from a given surface describing the initial liquid body. During the simulation, in each time step, the octree grid is adapted to the current surface of the liquid, as described below. As a result, the simulation grid seems to 'creep' along with the liquid boundary, as illustrated in Figure 3.

The adaptivity of the octree grid is controlled as follows. In a narrow band around the liquid boundary, which we refer to as the refinement band, we require all simulation cells to be at the finest resolution. The radius $r := \omega h$ ($\omega \in \mathbb{N}$) of the refinement band has to be specified. We enforce that the liquid boundary never leaves the refinement band, and thus is always represented on the finest resolution level. As a consequence, the maximum time step that can be simulated is coupled to r. While it is desirable in principle to make ω large in order to reduce time step constraints, this can significantly increase the number of grid cells. We have found that $\omega = 2$ results in a good trade-off between time step constraints and memory requirements, and have used this value in all of our experiments. The interior of the fluid is filled by coarse cells, with the restriction that the level difference between neighboring cells sharing at least one common vertex is at most one. This results in a restricted octree grid as proposed by Popinet [30]. In this way, the fluid interior is covered by a minimum number of cells, while the grid resolution decreases smoothly with increasing distance to the surface.

In the first simulation step, a restricted octree grid

is built from an analytical or triangle mesh representation of the given surface of the initial liquid body, with the requirement that all cells intersected by the surface are on the finest octree level.

In each simulation step (including the first, in order to build the refinement band), the following steps are performed to adapt the grid to the current liquid boundary:

1) Creation of the refinement band: The set of cells intersected by the fluid surface is determined. The set is then successively extended in ω iterations, such that in each iteration the finest level cells in the 26-neighborhood of each cell of the set are added to the set. If in this process we encounter finest level cells that do not exist yet, we create them. If we encounter cells that are not on the finest octree level, we split them and apply further splits recursively to keep the octree grid restricted. In this way, we create one layer of the refinement band (on both sides of the liquid boundary) in each iteration. The resulting refinement band has a minimum width of $(2\omega + 1)h$ at any given point. Cells outside of the liquid body that do not belong to the refinement band are deleted.
2) Grid coarsening: As many cells as possible are merged in the fluid interior, while keeping the octree grid restricted.
3) Cell classification: The cells of the new grid are re-classified: Cells in the interior of solids are classified as *solid*, cells of which at least one vertex is lying inside the liquid boundary as *fluid*. All remaining cells are labeled *empty*. Note that per construction, all non-finest-level cells are fluid cells and lie completely within the liquid boundary. As an optimization, we utilize that all solid cells remain solid when the obstacles do not move.

The union of all fluid cells now defines the actual computational domain for the finite element discretization. The other cells are required only in the advection steps of the simulation. Consequently, all fluid cells and all vertices that are adjacent to at least one fluid cell (referred to as fluid vertices) carry a full set of simulation quantities. All other cells and vertices are light-weight and carry only a small subset of these quantities, in particular including extrapolated velocities and level-set values required for advection.

To represent the octree grid, an unbalanced octree data structure is used. Since we do not use an initially specified domain, the height of this octree varies over the course of the simulation according to the dynamically changing extent of the liquid.

3.1 Obstacles

Solid obstacles are handled in a similar way as proposed by Houston et al. [31] for generating a compact level-set representation. Obstacles are initially voxelized with respect to the imaginary finest level lattice such that a run-length encoded binary voxelization is created. This representation introduces a negligible memory footprint (compared to the final simulation grid and the additional data structures used by the solver), yet it allows us to efficiently test whether a finest level cell is inside a solid obstacle or not. It is important to note that the solids' voxelization is decoupled from the simulation grid, and that only those solid parts are represented in the simulation grid which are located in the refinement band around the liquid boundary.

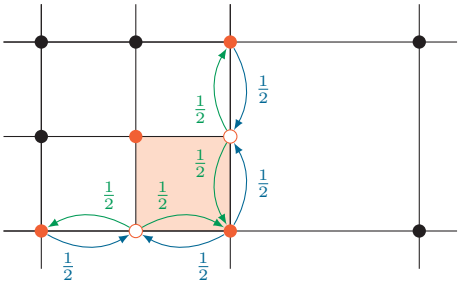

Fig. 4. Handling of hanging vertices (2D case), illustrated for the shaded element: The unknowns at hanging vertices (empty orange dots) are substituted (blue arrows) by linear interpolation from the unknowns at non-hanging vertices (filled orange dots). The equations at hanging vertices are then distributed (green arrows) to non-hanging vertices using the same weights, corresponding to the interpolation of values at hanging vertices in the test functions.

4 Finite Element Fluid Simulation

To simulate the liquid, we solve for its motion on the fluid domain Ω as dictated by the incompressible Navier-Stokes equations

$$\dot{\boldsymbol{u}} = -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{\mu}{\rho}\Delta \boldsymbol{u} + \boldsymbol{g} - \frac{1}{\rho}\nabla p \tag{1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2}$$

where $\boldsymbol{u}$ is the fluid velocity, ρ the density, μ the viscosity, p the internal pressure, and $\mathbf{g}$ the gravity force. Note that ρ and μ are constant. We solve the pressure Poisson equation

$$\frac{1}{\rho}\Delta p = \frac{1}{\Delta t}\nabla \cdot \boldsymbol{u}^* \qquad \text{on } \Omega \tag{3}$$

$$p = p_A \qquad \text{on } \Gamma_A \tag{4}$$

$$\boldsymbol{n} \cdot \nabla p = \boldsymbol{n} \cdot \frac{\rho}{\Delta t}\left(\boldsymbol{u}^* - \boldsymbol{u}_W^{t+\Delta t}\right) \qquad \text{on } \Gamma_W \tag{5}$$

in each time step to project the intermediate velocity $\boldsymbol{u}^*$ to the space of divergence free functions according to $\boldsymbol{u}^{t+\Delta t} = \boldsymbol{u}^* - \frac{\Delta t}{\rho}\nabla p$ in order to enforce (2). $\Gamma_{A(ir)}$ and $\Gamma_{W(all)}$ denote the fluid/air and fluid/solid

interface parts of the liquid boundary $\partial\Omega$, respectively ($\partial\Omega = \Gamma_A \uplus \Gamma_W$). On Γ_A the pressure p_A is prescribed as a Dirichlet boundary condition (4). On Γ_W, the velocity $\boldsymbol{n} \cdot \boldsymbol{u}_W^{t+\Delta t}$ in normal direction $\boldsymbol{n}$ to the boundary, and optionally the velocity $\boldsymbol{\tau} \cdot \boldsymbol{u}_W^{t+\Delta t}$ in tangential direction $\boldsymbol{\tau}$ to the boundary for the next time step $t + \Delta t$ is prescribed. The normal component is enforced via (5) as a Neumann boundary condition for the pressure. The optional tangential component can be enforced by setting $\boldsymbol{\tau} \cdot \boldsymbol{u}^{t+\Delta t}$ to the respective value after each projection step [32].

The liquid's surface is captured using the level-set method [33], i.e., the interface location is represented by the zero-contour of a signed distance function ϕ. Additionally, following [9], we augment the level set with marker particles to improve mass conservation.

To obtain a consistent discretization on the spatially adaptive octree grid, we employ a finite element method (FEM). Each *fluid* cell is a finite element. We use an equal-order interpolation scheme, where both $\boldsymbol{u}$ and p are discretized via element-wise tri-linear basis functions. Accordingly, all degrees of freedom are located at the grid vertices. At the hanging vertices at octree level transitions, however, a special treatment procedure is required: Their values are constrained by linear/bilinear interpolation along the edge/within the face with respect to which the vertex is hanging (see Figure 4 for an illustration). This ensures that the resulting weak solution is C^0 continuous across the entire domain. Furthermore, as will be demonstrated below, the specific choice of basis functions yields a discretization of the pressure Poisson equation that is exceptionally well suited for a multigrid scheme, as it results in very good convergence rates.

Let $\boldsymbol{v}$ and q denote the vector and scalar valued test functions corresponding to $\boldsymbol{u}$ and p, respectively. By testing (1) against $\boldsymbol{v}$ and (3) against q (i.e., by multiplying with the test function and integrating over Ω), we obtain the following weak form of the continuous problem:

$$\int_\Omega \dot{\boldsymbol{u}} \cdot \boldsymbol{v} = -\int_\Omega \boldsymbol{u} \cdot \nabla \boldsymbol{u} \cdot \boldsymbol{v} - \int_\Omega \frac{\mu}{\rho} \left(\nabla \boldsymbol{u} : \nabla \boldsymbol{v}^T\right) + \int_\Omega \boldsymbol{g} \cdot \boldsymbol{v} - \int_\Omega \frac{1}{\rho} \nabla p \cdot \boldsymbol{v} \tag{6}$$

$$\int_\Omega \frac{1}{\rho} \nabla q \cdot \nabla p = -\frac{1}{\Delta t} \int_\Omega q \nabla \cdot \boldsymbol{u}^* + \frac{1}{\Delta t} \int_{\Gamma_W} q \boldsymbol{n} \cdot \left(\boldsymbol{u}^* - \boldsymbol{u}_W^{t+\Delta t}\right) \tag{7}$$

For a thorough overview of different FEM weak forms for the Navier-Stokes equations let us refer to the book by Gresho [34].

Applying the finite element discretization leads to the following equations ($\boldsymbol{u}$, p, etc. now are vectors containing the respective values at the grid vertices):

$$M\dot{\boldsymbol{u}} = C(\boldsymbol{u})\boldsymbol{u} + K\boldsymbol{u} + \boldsymbol{g} - Gp \tag{8}$$

$$Lp = -\tfrac{1}{\Delta t} D\boldsymbol{u}^* + \tfrac{1}{\Delta t} B(\boldsymbol{u}^* - \boldsymbol{u}_W^{t+\Delta t}) \tag{9}$$

1: $\phi^{t+\Delta t} := \text{Advect}(\phi^t)$
2: Adapt octree grid
3: Reinitialize $\phi^{t+\Delta t}$
4: $\boldsymbol{u}' := \text{Advect}(\boldsymbol{u}^t)$
5: $\boldsymbol{u}' := \boldsymbol{u}' + \Delta t M^{-1} \boldsymbol{g}$
6: Solve for $\boldsymbol{u}^*$: $(I - \Delta t M^{-1} K)\boldsymbol{u}^* = \boldsymbol{u}'$
7: Solve for p: $\Delta t\ Lp = -D\boldsymbol{u}^* + B(\boldsymbol{u}^* - \boldsymbol{u}_W^{t+\Delta t})$
8: $\boldsymbol{u}^{t+\Delta t} := \boldsymbol{u}^* - \Delta t M^{-1} Gp$
9: Extrapolate $\boldsymbol{u}^{t+\Delta t}$

Fig. 5. Algorithm for time integration according to equations (8) and (9). I denotes the identity matrix. Note that the mass matrix M is lumped, thus M^{-1} is a diagonal matrix.

M is the (lumped) mass matrix, $C(\boldsymbol{u})$ the (FEM) convection operator, K the diffusion operator, G pressure gradient operator, L the Laplace operator, D the divergence operator, and B a boundary operator corresponding to the boundary term in (7).

To perform the time integration of the derived equations we use an explicit Euler scheme with operator splitting as proposed by Stam [12]. The resulting procedure, which is performed in each time step, is listed in Figure 5.

The listing shows that the octree grid is adapted after the level-set has been advected, but before it is re-initialized. In this way, every new vertex that is introduced during octree adaptation is immediately assigned a correct value for ϕ, instead of an interpolated value. Level-set re-initialization is performed only in every tenth time step [33].

We use semi-Lagrangian advection to handle the advection of the level-set as well as the convective term ($C(\boldsymbol{u})$ is not required). During advection, trilinear interpolation is used to retrieve samples within the adaptive hexahedral grid. Furthermore, an implicit Euler scheme is used to handle the diffusive term with unconditional stability.

4.1 Pressure Boundary Conditions

The boundary conditions (4, 5) for the pressure Poisson equation are handled via a particular second order accurate FEM discretization, while at the same time the symmetry of the resulting system matrix as required by the multigrid solver is preserved. For this method, the integration domain Ω in the weak form (6, 7) is restricted with subgrid accuracy to the part of the domain that is actually covered by the fluid (rather than being aligned to the hexahedral grid). Considering each single finite element, this means that the support of the basis functions is restricted to the portion of the element that is covered by fluid. Note that the finite element grid is not modified, and that the degrees of freedom of the finite elements remain at the grid vertices.

This restriction of the integration domain is sufficient to handle the pressure Neumann boundary condition on the fluid/solid interface Γ_W with subgrid accuracy. Thus, no special treatment as in finite difference schemes has to be performed, e.g., by using the variational approach by Batty et al. [35].

In contrast, for the pressure Dirichlet boundary condition on the fluid/air interface Γ_A, further considerations are necessary. In finite difference schemes, this condition can be imposed with subgrid accuracy by using so-called ghost fluid methods [36]. These methods introduce additional ghost pressure values in the air, and linear dependencies among the pressure values around the boundary to enforce the Dirichlet boundary condition at the actual surface position. However, when this procedure is directly applied to the FEM formulation (7), the boundary term cannot be restricted from Γ to Γ_W (as has been done in (7)) since $q = 0|_{\Gamma_A}$ no longer holds, resulting in an asymmetric contribution to the system matrix.

A way to handle the pressure Dirichlet boundary condition on the fluid/air interface with subgrid accuracy that we have found to work very well in our formulation is the group of Nitsche methods [37]. In these methods, all vertices of a finite element are full degrees of freedom, and a carefully chosen penalty term is added to the weak form such that the Dirichlet boundary condition is enforced at the actual surface position in a variational sense.

In particular, we have adapted the method developed by Dolbow et al. [38] for general Poisson problems, which achieves second order accuracy [39]. It maintains symmetry of the system matrix by constructing the penalty term such that the aforementioned asymmetric boundary term is counterbalanced. Leaving the detailed derivation and convergence proof to [38], we extend (7) according to:

$$\begin{aligned}\int_\Omega \frac{1}{\rho}\nabla q\cdot\nabla p - \int_{\Gamma_A}\left(\frac{1}{\rho}q\boldsymbol{n}\cdot\nabla p + \frac{1}{\rho}p\boldsymbol{n}\cdot\nabla q - \alpha qp\right)\\ = -\frac{1}{\Delta t}\int_\Omega q\nabla\cdot\boldsymbol{u}^* + \frac{1}{\Delta t}\int_{\Gamma_W} q\boldsymbol{n}\cdot(\boldsymbol{u}^* - \boldsymbol{u}_W^{t+1})\\ \underbrace{-\int_{\Gamma_A}\left(\frac{1}{\rho}p_A\boldsymbol{n}\cdot\nabla q + \alpha q p_A\right)}_{=0 \text{ if } p_A=0}\end{aligned} \tag{10}$$

In principle, (10) is constructed by testing (3) against q, (4) against αq, (4) against $-\frac{1}{\rho}(\boldsymbol{n}\cdot\nabla q)$ and then by summing up the resulting three equations. The method requires a constant stabilization parameter $\alpha = \frac{s}{\rho}\frac{\text{Area}(\Gamma_A)}{\text{Volume}(\Omega)}$ per cell, where $\text{Area}(\Gamma_A)$ is the area of the fluid/air interface and $\text{Volume}(\Omega)$ the volume of the fluid in the cell. s depends on the finite element type used and has been set to $s = 5$ in all of our experiments. If $p_A = 0$, the last term in (10) vanishes. Note that for cells which do not contain any part of Γ_A (i.e., which are fully covered by fluid), the weak form (10) is equivalent to (7).

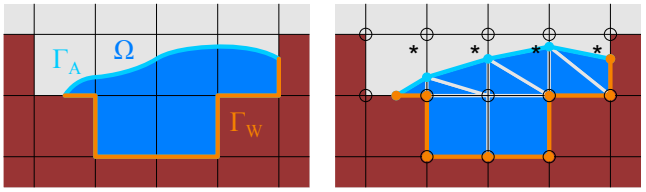

Fig. 6. Subgrid accurate treatment of boundary conditions (for illustration purposes shown in 2D). Left: The computational domain Ω is restricted to the portion of the grid cells that is actually covered by the fluid. Right: In cells intersected by the fluid/air interface (cells marked with a star), Ω, Γ_A, and Γ_W are tetrahedralized/triangulated by employing a pre-computed lookup table. The degrees of freedom of the finite elements (black circles) remain located at the grid vertices.

In order to apply the extended weak form (10), in grid cells that are intersected by the fluid/air interface, the calculation of volume and surface integrals must be restricted to the fluid domain. For this purpose, we triangulate/tetrahedralize the boundary and the fluid domain in these cells, as illustrated in Figure 6 (right). The integration problem has been similarly addressed in [40], [41]. We then evaluate the surface and volume integrals using numerical quadrature on each of the resulting simplices to obtain the respective element matrices.

For the triangulation/tetrahedralization, we have derived an extended, marching-cubes-style lookup table [42]. It provides pre-calculated triangulations/tetrahedralizations for any given configuration of level-set values (positive/non-positive) at the eight vertices of a fluid cell. Note that in our implementation, solids are represented on a per-cell basis. Let $\Omega_c \subset \Omega$ denote the part of a cell's domain that is covered by the fluid. Then, for each configuration, the lookup table contains the following three components:

- A set of triangles that triangulates the fluid/air interface $\partial\Omega_c \cap \Gamma_A$ within the cell (native marching-cubes).
- For each of the six cell faces a set of triangles that triangulates the potential fluid/solid interface $\partial\Omega_c \setminus \Gamma_A$ (used only if a respective neighbor cell is solid).
- A set of tetrahedra that tetrahedralizes the fluid domain Ω_c within the cell (up to ten tetrahedra are required per cell).

The lookup table is constructed such that the triangulation and tetrahedralization are consistent, i.e., each triangle always corresponds to a face of a tetrahedron. In order to avoid near-zero values in the system matrix (due to cells containing almost zero fluid), we enforce the generated vertices on the edges of a cell to have a minimum distance of $0.01h$ to

the cell's vertices inside the liquid body. Note that any remaining degenerated triangles/tedrahedra do not cause problems, because their contribution to the integral in (10) vanishes. For the numerical quadrature on the triangles and tetrahedra, we use a three-point Gaussian and an eight-point Newton-Cotes quadrature rule [43], respectively. For the evaluation of the surface integrals, the face normals of the triangles are used as the normal of the boundary.

The simple example in Figure 7 demonstrates the accuracy achieved by our approach. Considering the performance, the subgrid accurate treatment of boundary conditions does not have a significant impact on the total computing time of our method. The time required for the computation of the element matrices for the cells that are intersected by the fluid/air interface is typically less than 5% of the time required for one complete time step. From an implementation point of view, it is worth noting that the computation of the element matrices, including triangulation/tetrahedralization and quadrature can be performed independently for each cell, and thus can be easily parallelized.

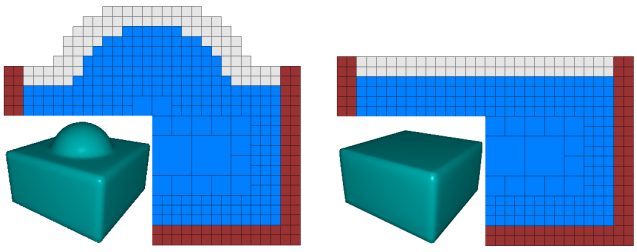

Fig. 7. Demonstration of second order accurate boundary conditions for a perturbed liquid in a box: A bump in the surface (left) is smoothed out after several seconds of simulation (right). The resulting surface is perfectly planar. Note that the surface is tracked by a level-set without particles in this experiment, and that the blue color indicates fluid cells, rather than depicting the actual fluid domain.

5 Multigrid Solver

The pressure Poisson equation is solved in each time step with an efficient geometric multigrid solver, which exhibits linear time-complexity in the number of unknowns [21]. Its efficiency is crucial, since with increasing grid size, the pressure solve quickly becomes the major performance bottleneck when less efficient solvers are used.

A major challenge in the design of geometric multigrid solvers is the treatment of complex shaped domain boundaries (here, the *fluid* domain boundary) on the successively coarser scales. In contrast to common multigrid schemes on simple shaped (e.g., rectangular) domains, we have to use modified processes for the construction of the coarse grid hierarchy and the algebraic operators on these grids. Our scheme relies on a specific strategy to generate the coarse grids incrementally from the fine grid, and uses a variational principle to generate the transfer and coarse grid operators. In this way our solver can handle complex fluid boundaries in a highly efficient manner.

In particular, we address a problem that specifically arises when the simulation domain is composed of multiple (locally) separated components (for instance, separated fluid branches, drops or splashes). In such situations, separated domain components might get represented by the same coarse grid cell, when a canonical coarse grid hierarchy is used. As a consequence, the current error cannot be represented very well on the coarse grids, which in turn causes the multigrid convergence to break down. This is demonstrated in Section 7. We address this problem by duplicating coarse grid cells at the same geometric position, with each instance corresponding to a separated fluid component, similar to [44], [45], [46], [47]. Conceptually, we build a distinct coarse grid for each separated fluid component, with the coarse grids being allowed to overlap. This strategy alleviates the convergence problem and introduces only little extra cost when building the multigrid hierarchy.

In each simulation step, the multigrid hierarchy is rebuilt and the corresponding coarse grid operators are recomputed. Note that the multigrid hierarchy is completely separate from the octree data structure that is used for the representation of the adaptive octree grid. Cell duplication is controlled by means of a connectivity graph G_l per multigrid level, which is constructed on the finest multigrid level $l = 0$ from the set of fluid cells, and on the coarser levels $l = 1, \dots, l_{max}$ by propagating the previous level graph to the current level along with the grid. The nodes of each graph G_l correspond to the cells of the grid on the respective level, and edges in the graph represent physical connections among these cells. Edges only exist between nodes corresponding to adjacent cells that share at least one common vertex. Since the graphs are only required during construction of the hierarchy, they do not need to be stored permanently.

Multigrid hierarchy construction starts by creating the finest level connectivity graph G_0. For each *fluid* cell a node is created, and two nodes are connected if the respective fluid cells share at least one common vertex.

After initialization of the finest level graph, we build the multigrid hierarchy successively in a fine-to-coarse process. This process stops at the coarsest level l_{max}, as soon as the remaining number of cells is less than 1000.

In the coarsening procedure, we have to consider that the finest level grid is an adaptive octree grid consisting of cells of different size. On the transition from multigrid level l to level $l + 1$, we therefore combine only cells of size $2^l h$, but copy all larger cells to the next level.

The construction of multigrid level $l+1$ is divided into the following three steps:

1) Creation of coarse grid cells: Considering an imaginary lattice with lattice points $(2^{l+1}h)\mathbb{Z}^3$, for each cell of this lattice, we determine the connectivity components of the subgraph of G_l induced by the grid cells of size $2^l h$ within this lattice cell. For each connectivity component, a coarse grid cell of size $2^{l+1}h$ is created, which subsumes the grid cells corresponding to the respective subgraph. In this way, multiple coarse grid cells might be created at the same location. Then, all grid cells with size $\geq 2^{l+1}h$ are copied from the level l to level $l+1$.
2) Propagation of the connectivity graph: G_{l+1} is obtained by collapsing each subgraph of G_l that corresponds to a set of merged cells into a single node.
3) Creation of shared vertices on the coarse grid: At each vertex position, the subgraph of G_{l+1} corresponding to all cells that are incident to this position is investigated. A shared vertex is generated for each connectivity component of this subgraph.

An example illustrating the coarse grid hierarchy construction according to the specified rules is demonstrated in Figure 8. Hanging vertices are not shown, because they are eliminated before the multigrid hierarchy is constructed, as described in Section 6. It is worth noting that due to the particular coarsening strategy, no additional hanging vertices are created in the coarse grid. Furthermore, it can be observed that duplicate vertices only occur on levels $l \geq 1$ and duplicate cells only on levels $l \geq 2$.

The transfer and coarse grid operators are constructed as follows: We use full-weighting for the interpolation operators I_{l+1}^l, i.e., values on level l are tri-linearly interpolated from values on level $l+1$. More precisely, considering the occurrence of duplicate coarse grid cells in our advanced multigrid hierarchy, each cell is merged into exactly one coarse cell on the next coarser level, and the cell's vertices exactly interpolate from and restrict to this coarse cell's vertices (see Figure 9). It is important to note that this scheme also works at the fluid boundary where coarse cells are only partially 'filled' by cells on the finer level, in that the constructed interpolation operator always has full rank.

Starting from the linear equation system $Lp = b$ on the finest level (9), the restriction operators R_l^{l+1} and coarse grid operators L_l ($L_0 \equiv L$) are built from fine to coarse according to

$$R_l^{l+1} = (I_{l+1}^l)^T, \tag{11}$$

$$L_{l+1} = R_l^{l+1} L_l I_{l+1}^l, \tag{12}$$

with (12) being known as Galerkin-based coarsening. Note that the construction of the operators is purely algebraic, i.e., boundary conditions are automatically incorporated on the coarse levels. The construction of the operators follows a variational principle [48], in that by applying the computed coarse grid correction, the error is minimized. In particular, in combination with a converging smoother such as Gauss-Seidel relaxation, it can be shown that the norm of the error is monotonically decreasing with each further iteration of the solver, i.e., the convergence of the solver is guaranteed [49]. As a side note, let us mention that from a mathematical point of view, the resulting coarse grid operators can be interpreted as composite finite element discretizations [50] of the continuous problem on the coarse grids.

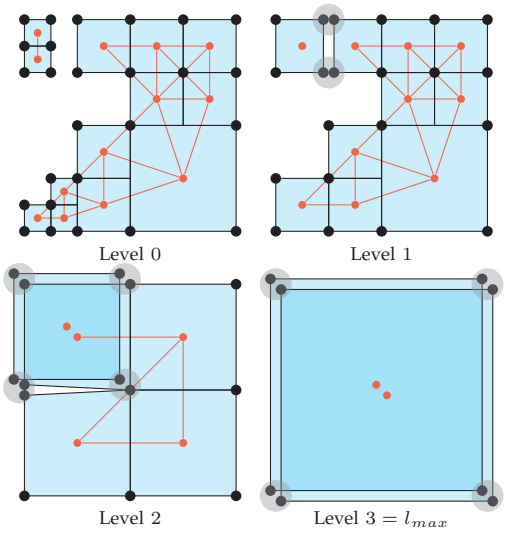

Fig. 8. A 2D example for a multigrid hierarchy. Circles shaded gray indicate duplicate vertices sharing the same geometric location, overlapping cells indicate duplicate cells. Cell connectivity graphs G_l are depicted in red. Note that for demonstration purposes, the level 0 grid does not strictly adhere to the rules specified in Section 3, i.e., not all boundary cells are on the finest octree level.

Our solver traverses the multigrid hierarchy according to the standard V-cycle scheme. On each level of the hierarchy, we perform two pre-smoothing and two post-smoothing (multi-color) Gauss-Seidel relaxation steps. On the coarsest level l_{max}, the linear system is solved using a Jacobi-preconditioned conjugate gradient solver. The multigrid solver can be used as a direct solver as well as a preconditioner for a conjugate gradient solver, which improved its performance even further in our experiments. When used as a preconditioner, we perform one V-cycle per CG iteration.

In our experiments, we stop the multigrid solver when the norm of the residual $r = b - Lp$ has been reduced by a factor of 10^{-6} (i.e., when $\|r_i\|_2 / \|r_0\|_2 \leq 10^{-6}$, where r_i denotes the residual after the ith V-cycle or CG iteration).

6 Implementation Details

As a consequence of the increasing gap between CPU and memory speed, main memory access has become the major bottleneck in numerical simulation on today's PC architectures, both in terms of limited memory throughput and high memory access latencies.

In grid-based fluid simulation based on the projection method, computing time is known to be dominated by the run-time of the linear solver for the pressure Poisson equation. In particular, for a geometric multigrid solver, the most frequent and most time-consuming operations are the pre- and post-smoothing relaxation steps as well as the residual computation step on each level of the multigrid hierarchy. In each of these steps, the numerical stencil at each vertex, consisting of (approximately) $3^3 = 27$ floating point values, has to be accessed, leading to a significant amount of memory traffic when this stencil is fetched from main memory.

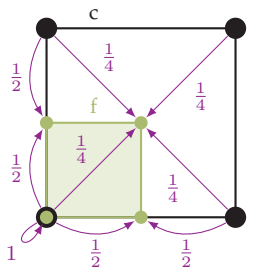

Fig. 9. Galerkin-based coarsening with our cell-based formulation (2D case): The vertices of a fine grid cell f (green) interpolate from the vertices of its associated coarse grid cell c (black) using linear/bilinear weights (purple). The matrix of a coarse grid cell is computed as a sum of matrices derived from its associated fine grid cells. The contribution of each fine grid cell is obtained by substituting the fine grid unknowns by interpolation from the coarse grid unknowns, and distributing the equations from the fine grid vertices to the coarse grid vertices using the same weights.

6.1 Cell-based FEM and Multigrid Formulation

To alleviate the aforementioned memory bottleneck, our implementation is built upon a cell-based formulation of the finite element method and the geometric multigrid solver. This cell-based formulation is leveraged by our particular design choices of using a hexahedral discretization, tri-linear finite elements and Galerkin-based coarsening. More specifically, these choices allow us to compute the FEM operators, including in particular the Poisson operator and its corresponding multigrid coarse grid operators on a per-cell basis (rather than a per-vertex basis). During the computation, the numerical stencil at each vertex is then assembled on-the-fly from the matrices of the incident cells. Due to the regular hexahedral shape of the elements in our adaptive grid, for each FEM operator, (almost) all elements share the same generic element matrix (except for scaling with the cell size). These generic element matrices are pre-computed analytically. On the finest level, the only exception are elements that are intersected by the fluid/air interface. Only for those we numerically compute and store the non-generic element matrices resulting from the second-order accurate boundary conditions.

For the coarse grid versions of the Poisson operator, the matrix of each cell is assembled from the matrices of its associated fine grid cells by means of Galerkin-based coarsening (see Figure 9). This matrix is equal to the generic Poisson operator element matrix (except for scaling with the cell size), if (a) the cell is completely 'filled' by fine grid cells and (b) all of these cells' matrices are generic. As a consequence, on the coarse levels, we only have to compute and store the matrices of those cells for which one of these two conditions is not met.

Since the generic element matrices can be assumed to reside in cache memory, the amount of main memory traffic is reduced considerably. Compared to an implementation where the numerical stencil at each vertex is fetched from main memory, we observe a speed-up factor for the multigrid solver between 2 and 3—despite of the increased number of floating point operations due to the on-the-fly assembly of the stencil. As an additional benefit, the maximum memory footprint during the course of a time step is reduced by 20% to 30%.

It is worth noting that the size of the 27-point (in average) FEM Poisson stencil should not be considered a major performance drawback per se (compared to a size of only 7 for the commonly used uniform grid FD Poisson stencil). Because the performance bottleneck is the memory throughput resulting from fetching pressure values rather than arithmetic throughout, the large vertex overlap among neighboring stencils alleviates the impact on performance to a certain extent. This is especially true in a parallel implementation. Consider a uniform grid on a rectangular domain: Although the FEM stencil is larger than the FD stencil by a factor of $27/7 \approx 4$, the memory throughput increases only by a factor of $9/5 \approx 2$ (the size ratio of the stencils' 2D footprints) when iterating over the vertices of the grid due to cache coherence. In a simple experiment, we could actually verify this factor of roughly 2 between FEM and FD on our test system.

To efficiently handle hanging vertices occurring in the adaptive hexahedral discretization, we eliminate these vertices directly on the finest level, similar to the approach proposed in [51]. This is achieved by observing that a hanging vertex is lying inside an edge or face of a neighboring cell, with the unknown at the hanging vertex being determined by linear or bilinear interpolation from the edge or face's vertices. Note that in a restricted octree, these vertices are guar-

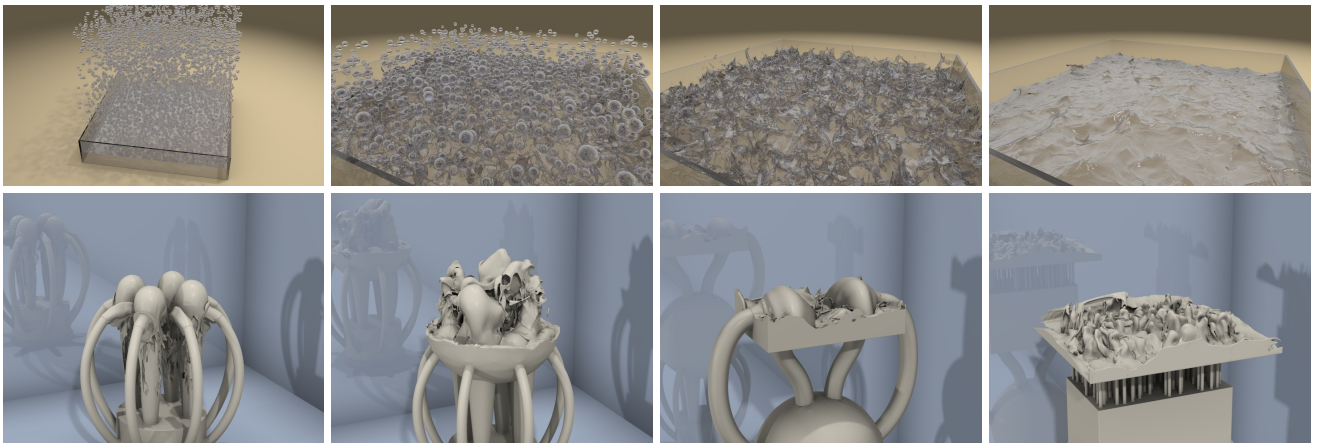

Fig. 10. Two large-scale liquid simulations: Top: 4000 drops of water fall into a rectangular basin. Each drop has an extent of up to 30 finest octree level cells. Bottom: A liquid is pumped into a multi-stage water fountain.

anteed to be non-hanging. By replacing all hanging-vertices with the corresponding non-hanging vertices, each cell can always be associated with 8 non-hanging vertices (which may be different from the cell's geometric vertices). The vertex/cell incidence relationship is then determined accordingly. To obtain a cell's element matrix from the generic element matrix, we first substitute the unknowns at the cell's hanging vertices with the respective unknowns at its associated non-hanging vertices, and then distribute the equations at the hanging vertices to the non-hanging vertices, using the same weights as are used for linear/bilinear interpolation (see Figure 4). Since there are 8 possible positions of a cell with respect to its (imaginary) parent cell in the octree, and since for each position at most 6 of the 8 vertices can be hanging, there are at most $8 \cdot 2^6 = 512$ hanging vertex configurations, corresponding to at most 512 adapted generic element matrices. These matrices are pre-computed and stored in a look-up table, which is then used to assemble the numerical stencils on-the-fly. The elimination of hanging vertices leads to a speed-up factor of about 2 compared to continuously handling the hanging vertices on-the-fly during the computation.

6.2 Parallelization

To exploit the performance available on today's multi-core CPU architectures, we have parallelized all parts of our algorithm where cells or vertices can be processed independently. Only the adaptation of the octree grid, the construction of the multigrid coarse grid hierarchy (not including the computations of the coarse grid operators), and the re-initialization of the level set function using a fast marching technique [33] are running sequentially. In terms of run-time on a single core, more than 90% of the algorithm are parallelized.

For the parallelization of the multi-color Gauss-Seidel relaxation step, a vertex coloring defining subsets of independent vertices is required. For our adaptive octree grid, we construct this vertex coloring independently for each multigrid level l as follows: For each vertex, we define its level as the maximum octree level of its incident cells. Considering each subset of vertices on the same level l_v ($l \leq l_v \leq l_{\max}$), these vertices are lying on a uniform grid with spacing $2^{l_v}h$, and none of the vertices is incident to a cell larger than $2^{l_v}h$. Therefore, this subset can be further partitioned into subsets of independent vertices using 8 colors. As a consequence, on multigrid level l, at most $8(l_{\max} - l + 1)$ colors are required to partition the vertices into subsets of independent vertices.

All per-cell and per-vertex quantities are stored in linear arrays, which are accessed by means of cell and vertex indices. In order to allow for an efficient, parallel 'traversal' of the cells and vertices of the adaptive octree grid and the coarse grid hierarchy, the cells and vertices of each level are consecutively enumerated in a particular order, which is determined as follows. First, all fluid cells and fluid vertices are located prior to non-fluid cells and vertices, respectively. Considering that the non-fluid cells and vertices carry a reduced set of simulation quantities, this allows us to store all quantities contiguously in memory. Second, considering the parallelization of the Gauss-Seidel relaxation, the fluid vertices on each multigrid level are ordered according to their vertex color as defined above. In this way, the simulation quantities of each vertex subset are lying consecutively in memory. To maintain this particular order after adaptation of the octree grid, we re-enumerate the cells and the vertices in each simulation step, and re-order the linear arrays of per-cell and per-vertex quantities accordingly.

7 Results

To demonstrate the effectiveness of our approach, we have run two large-scale simulations with up to 34 million elements, which are shown in Figure 10. In

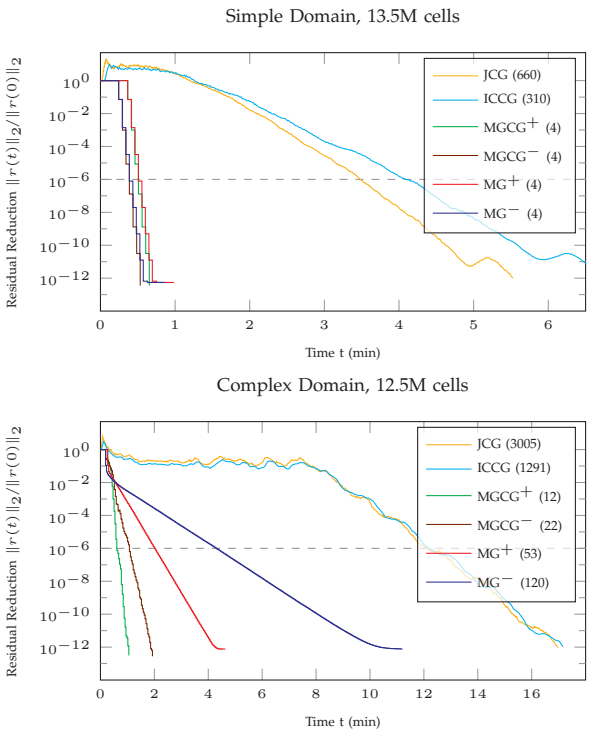

Fig. 11. Solver convergence over time. In parentheses are given the number of iterations (V-cycles or CG iterations) that were required to achieve a residual reduction by 10^{-6} (dashed line). As can be seen, the MGCG$^+$ solver outperforms the JCG/ICCG solvers roughly by a factor of 8 to 12. Furthermore, the second example demonstrates the necessity of the cell-duplication strategy in complex scenarios, where we can observe a speed-up of almost a factor of 2 when comparing MGCG$^+$ to MGCG$^-$ and MG$^+$ to MG$^-$.

addition, we have analyzed the convergence behavior of our solver in several scenarios and compared it to a conjugate gradient solver. All simulations and tests were carried out at double floating point precision on a single workstation equipped with two Intel Xeon E5-2643 processors with a total of eight cores running at 3.3 GHz and 64 GB of RAM.

7.1 Convergence Study

In Figure 11 we showcase the performance of our multigrid solver for two scenarios exhibiting different complexities of the simulation domain. In particular, we compare our multigrid solver, used as a direct solver (MG) and as a preconditioner for the conjugate gradient method (MGCG), to conjugate gradient solvers with Jacobi (JCG) and incomplete Cholesky IC(0) preconditioners (ICCG). To demonstrate the effectiveness of our cell duplication strategy in the construction of the coarse grid hierarchy, we have also included the convergence behavior of our multigrid solver using a standard coarse grid hierarchy, which is obtained by merging cells purely based on their geometric position without considering the connectivity between cells. Multigrid variants with cell duplication are marked with a $^+$, whereas variants without cell duplication are marked with a $^-$. In order to guarantee a fair comparison, the CG solvers are optimized for low memory traffic in exactly the same way (matrix-free) as our MG solver based on the on-the-fly assembly of the numerical stencils, and also run in parallel. For the IC(0) preconditioner, forward and backward substitution are parallelized using the existing vertex coloring scheme. Furthermore, we have included the initialization times of the solvers (e.g., the construction of the coarse grid hierarchy and the computation of the coarse grid operators for the multigrid solvers). Note that the MG solvers without cell duplication require slightly less time for initialization, because no connectivity graphs have to be built. When analyzing the IC(0) preconditioned CG solver, it is also important to note that in our scenario each CG iteration with IC(0) takes almost three times longer than with its Jacobi counterpart. As a consequence, although IC(0) requires less than half of the iterations, the achieved solver times are roughly identical.

In the first scenario ('Simple Domain') we solve for the pressure in a solid cube that is almost completely filled with liquid. Since the liquid covers a convex domain, no cell-duplicates are created. Therefore, the coarse grid hierarchy generated by our approach is identical to a standard hierarchy, thus the direct MG solvers exhibit the same stable convergence rate of $\|r_{i+1}\|_2/\|r_i\|_2 = 0.03$. The very high convergence rate is not further increased by using the MG solver as a preconditioner for the CG solver. As a consequence of the slightly higher initialization time required for creating a coarse grid hierarchy with connectivity analysis, in this scenario the MG solvers using the standard hierarchy are slightly faster. Compared to the commonly used Jacobi and IC(0) preconditioned CG solvers, all MG variants are about 6 times faster.

The second scenario ('Complex Domain') corresponds to the pressure solve in the fully filled fountain scenario shown in Figure 1. In contrast to the first scenario, now our advanced coarse grid hierarchy exhibits a significant number of cell duplicates due to the complex shape of the fluid domain, formed by a multitude of fine branches and separated domain fragments. For this scenario, our direct MG solver achieves a stable convergence rate of 0.81 and 0.92 for the advanced and the standard coarse grid hierarchy, which clearly demonstrates the effectiveness of our cell duplication strategy. Note that the decrease of the convergence rates compared to the first scenario is fully in-line with multigrid theory, considering the very complex shape of the simulation domain. Here, when using MG as a preconditioner, the convergence behavior is significantly improved, leading to an average residual reduction of 0.33 and 0.53 per

Scene (Time Step)	#Fluid Cells (Uncoarsened)	Max. Domain Resolution	Timings (min)					Memory Usage (GB)			
			T_{Total}	T_{ϕ}	T_{Octr}	T_{Proj}	T_{Other}	Total	Sim	MG	Octree
Drops (first step)	30.5M (131M)	1024^3	4.25	1.28	1.16	1.41	0.40	25.4	18.0	1.1	6.3
Drops (longest step)	33.3M (129M)	1024^3	5.10	1.77	1.38	1.46	0.49	24.8	17.2	1.2	6.4
Drops (last step)	12.0M (124M)	1024^3	1.34	0.28	0.26	0.69	0.11	6.7	4.6	0.5	1.6
Fountain (beginning)	13.4M (129M)	$1024^2 \times 3072$	2.89	0.33	0.30	2.13	0.13	8.6	5.9	0.5	2.2
Fountain (last step)	34.1M (391M)	$1024^2 \times 3072$	4.53	0.69	0.95	2.55	0.34	22.4	14.8	1.2	6.4

TABLE 1
Number of elements, timings and memory footprints for distinct time steps of the two simulations depicted in Figure 10. From top to bottom, these time steps correspond to the images in Figure 10 (top row, 1st), Figure 1 (left), Figure 10 (top row, 4th), Figure 1 (right) and Figure 10 (bottom row, 4th).

CG iteration, respectively. Compared to the standard Jacobi and IC(0) preconditioned CG solvers, the MG preconditioned CG solver is 12 times faster.

Since the performance advantages of MG over CG are generally weaker at smaller resolutions, we have further tested the second scenario at several smaller resolutions. Here, MG as a preconditioner still proved to be significantly faster, e.g., by a factor of 6 at a resolution of 1M cells, and a factor of 3 at a resolution of 250k cells.

7.2 Performance Analysis

Table 1 lists timings and memory footprints for distinct time steps for the scenes depicted in Figures 1 and 10. The first column '#Fluid Cells' shows the number of fluid cells in the respective state of the adaptive octree grid. To demonstrate the efficacy of the adaptiveness of the grid, we have also included in parentheses the number of fluid cells that would be required without coarsening in the liquid interior. Our experiments demonstrate (last row of the table) that by using the octree grid, the number of fluid cells is reduced by up to 90% compared to using an uncoarsened octree grid. 'Max. Domain Res.' indicates the resolution of a uniform hexahedral grid that would have to be used instead of our creeping grid in order to run the simulation using the same cell size as in the refinement band around the liquid boundary. In the next columns, we specify the total computation time T_{Total}, which is composed of the time spent for level set advection and velocity extrapolation (T_{ϕ}), adaptation of the octree grid (T_{Octr}), the projection step including construction of the multigrid hierarchy, the construction of the coarse grid operators, and the pressure solve (T_{Proj}), and the time spent in the remaining parts of the simulation (T_{Other}). In the remaining columns, we specify the maximum memory footprint of our simulation during the course of the time step ('Total'), which is composed out of three major parts. First, the memory required for all simulation quantities stored on a per-cell or per-vertex basis ('Sim'), such as u, u^*, p, ϕ, the cell classification flags, the non-generic matrices of the FEM and coarse grid operators, as well as the required temporary CG variables of the MG preconditioned CG solver. Second, the memory required for the topological encoding of the coarse grid hierarchy of the multigrid solver ('MG'), not including the finest level (i.e., the adaptive octree grid). Third, the memory required for the topological encoding of the adaptive octree grid ('Octree').

The timings given in the table refer to our parallel implementation. Compared to the algorithm running on a single core, we achieve a speed-up factor between 5 and 6 on our eight core workstation.

7.3 Octree vs. Uniform Grid

Coarsening the grid in the liquid interior does inevitably influence the simulation result compared to running the simulation on an uncoarsened grid (corresponding to a uniform grid). To demonstrate that our coarsening strategy does result in very little differences, we have performed a comparison for a dambreak scenario, which is depicted in Figure 12. The same setup was run twice: First, using our octree grid as proposed and second, using the same grid with the difference that no cells are merged in the liquid interior. Hence the only difference between the two simulations is the size of the cells in the liquid interior (in the octree case cells are coarsened up to size $16h$). As can be seen, the major characteristics of the flow are maintained almost exactly. There are only minor differences in surface details and small splashes, which naturally increase over the course of the simulation due to the accumulation of small errors.

8 Conclusion and Future Work

To the best of our knowledge, our work presents the first multigrid approach for liquid simulation on a spatially adaptive octree grid using a hexahedral finite element discretization. By using an octree simulation grid in combination with finite elements, the number of simulation elements can be made proportional to the liquid boundary area, and oscillatory velocity modes at level transitions can be avoided. The use of

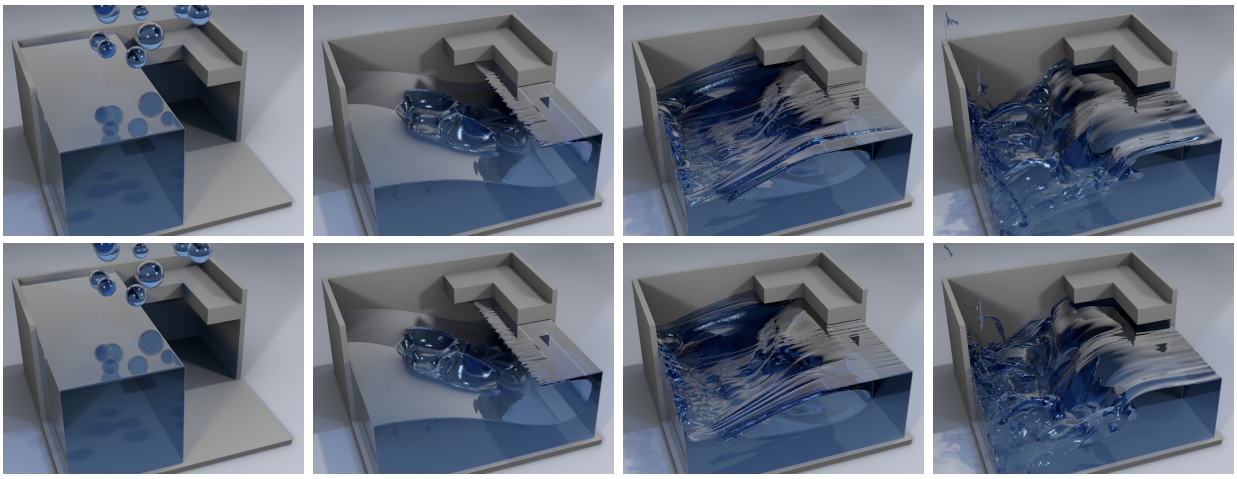

Fig. 12. Two dambreak simulations: Top: With coarsening in the fluids interior (650k to 1.15M cells). Bottom: Without coarsening (3.75M cells) the grid is equivalent to a uniform grid ($\approx 256^3$). Note that the unevenness in the splash coming back off the wall is owed to the random placement of level set particles around the surface.

a hexahedral discretization of the simulation domain severely simplifies the construction of the adaptive simulation grid, including a consistent hierarchy of coarse grids for a geometric multigrid solver, and gives rise to regular numerical stencils and efficient realizations of multi-level matrix assembly and multigrid solve. Our experiments have demonstrated excellent convergence rates of the multigrid solver used to enforce incompressibility, enabling efficient simulations of complicated liquid domains consisting of millions of simulation elements.

A limitation of the current method is the time step constraint (CFL number $= \omega$) that it imposes. This constraint is a result of our particular choice of grid refinement strategy, i.e., the use of a fixed-width refinement band. In a next step it will be interesting to explore more sophisticated refinement strategies in order to loosen the time step constraints and/or to reduce the number of required cells in the grid. One possibility would be to locally vary the width of the refinement band based on the velocities in a close vicinity around the surface.

ACKNOWLEDGMENT

This work was supported by the European Union under the ERC Advanced Grant 291372 SaferVis - Uncertainty Visualization for Reliable Data Discovery.

REFERENCES

[1] A. McAdams, E. Sifakis, and J. Teran, "A parallel multigrid poisson solver for fluids simulation on large grids," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010, pp. 65–74.

[2] K. Museth, J. Lait, J. Johanson, J. Budsberg, R. Henderson, M. Alden, P. Cucka, D. Hill, and A. Pearce, "OpenVDB: an open-source data structure and toolkit for high-resolution volumes," in *ACM SIGGRAPH Courses*, 2013, pp. 19:1–19:1.

[3] F. Losasso, F. Gibou, and R. Fedkiw, "Simulating water and smoke with an octree data structure," *ACM TOG*, vol. 23, no. 3, pp. 457–462, 2004.

[4] R. Ando, N. Thürey, and C. Wojtan, "Highly adaptive liquid simulations on tetrahedral meshes," *ACM TOG*, vol. 32, no. 4, pp. 103:1–103:10, 2013.

[5] B. Zhu, W. Lu, M. Cong, B. Kim, and R. Fedkiw, "A new grid structure for domain extension," *ACM TOG*, vol. 32, no. 4, pp. 63:1–63:12, 2013.

[6] N. Chentanez, B. E. Feldman, F. Labelle, J. F. O'Brien, and J. R. Shewchuk, "Liquid simulation on lattice-based tetrahedral meshes," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007, pp. 219–228.

[7] M. K. Misztal, K. Erleben, A. Bargteil, J. Fursund, B. B. Christensen, J. A. Bærentzen, and R. Bridson, "Multiphase flow of immiscible fluids on unstructured moving meshes," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2012, pp. 97–106.

[8] P. Clausen, M. Wicke, J. R. Shewchuk, and J. F. O'Brien, "Simulating liquids and solid-liquid interactions with Lagrangian meshes," *ACM TOG*, vol. 32, no. 2, pp. 17:1–17:15, 2013.

[9] D. Enright, F. Losasso, and R. Fedkiw, "A fast and accurate semi-Lagrangian particle level set method," *Computers and Structures*, vol. 83, no. 6-7, pp. 479–490, 2005.

[10] M. Kass and G. Miller, "Rapid, stable fluid dynamics for computer graphics," in *Proc. SIGGRAPH*, 1990, pp. 49–57.

[11] N. Foster and D. Metaxas, "Realistic animation of liquids," *Graphical Models and Image Processing*, vol. 58, no. 5, pp. 471–483, 1996.

[12] J. Stam, "Stable fluids," in *Proc. SIGGRAPH*, 1999, pp. 121–128.

[13] N. Foster and R. Fedkiw, "Practical animation of liquids," in *Proc. SIGGRAPH*, 2001, pp. 23–30.

[14] R. Bridson, *Fluid Simulation for Computer Graphics*. A K Peters, 2008.

[15] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, "A hybrid particle level set method for improved interface capturing," *Journal of Computational Physics*, vol. 183, no. 1, pp. 83–116, 2002.

[16] A. W. Bargteil, T. G. Goktekin, J. F. O'Brien, and J. A. Strain, "A semi-Lagrangian contouring method for fluid simulation," *ACM TOG*, vol. 25, no. 1, pp. 19–38, 2006.

[17] C. Wojtan, N. Thürey, M. Gross, and G. Turk, "Deforming meshes that split and merge," *ACM TOG*, vol. 28, no. 3, pp. 76:1–76:10, 2009.

[18] T. Brochu, C. Batty, and R. Bridson, "Matching fluid simulation elements to surface geometry and topology," *ACM TOG*, vol. 29, no. 4, pp. 47:1–47:9, 2010.

[19] N. Thürey, C. Wojtan, M. Gross, and G. Turk, "A multiscale approach to mesh-based surface tension flows," *ACM TOG*, vol. 29, no. 4, pp. 48:1–48:10, 2010.

[20] C. Wojtan, M. Müller-Fischer, and T. Brochu, "Liquid simulation with mesh-based surface tracking," in *ACM SIGGRAPH Courses*, 2011, pp. 8:1–8:84.
[21] W. Hackbusch, *Multi-Grid Methods and Applications*. Springer, 1985.
[22] N. Chentanez and M. Müller, "A multigrid fluid pressure solver handling separating solid boundary conditions," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2011, pp. 83–90.
[23] ——, "Real-time Eulerian water simulation using a restricted tall cell grid," *ACM TOG*, vol. 30, no. 4, pp. 82:1–82:10, 2011.
[24] M. Lentine, W. Zheng, and R. Fedkiw, "A novel algorithm for incompressible flow using only a coarse grid projection," *ACM TOG*, vol. 29, no. 4, pp. 114:1–114:9, 2010.
[25] B. E. Feldman, J. F. O'Brien, B. M. Klingner, and T. G. Goktekin, "Fluids in deforming meshes," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005, pp. 255–259.
[26] B. M. Klingner, B. E. Feldman, N. Chentanez, and J. F. O'Brien, "Fluid animation with dynamic meshes," *ACM TOG*, vol. 25, no. 3, pp. 820–825, 2006.
[27] C. W. Hirt, A. A. Amsden, and J. L. Cook, "An arbitrary Lagrangian-Eulerian computing method for all flow speeds," *Journal of Computational Physics*, vol. 135, no. 2, pp. 203–216, 1997.
[28] J. U. Brackbill and H. M. Ruppel, "FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions," *Journal of Computational Physics*, vol. 65, no. 2, pp. 314–343, 1986.
[29] Y. Zhu and R. Bridson, "Animating sand as a fluid," *ACM TOG*, vol. 24, no. 3, pp. 965–972, 2005.
[30] S. Popinet, "Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries," *Journal of Computational Physics*, vol. 190, no. 2, pp. 572–600, 2003.
[31] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth, "Hierarchical RLE level set: A compact and versatile deformable surface representation," *ACM TOG*, vol. 25, no. 1, pp. 151–175, 2006.
[32] D. L. Brown, R. Cortez, and M. L. Minion, "Accurate projection methods for the incompressible Navier-Stokes equations," *Journal of Computational Physics*, vol. 168, no. 2, pp. 464 – 499, 2001.
[33] S. Osher and R. Fedkiw, *Level set methods and dynamic implicit surfaces*. Springer, 2003.
[34] P. M. Gresho, R. L. Sani, and M. S. Engelman, *Incompressible Flow and the Finite Element Method: Isothermal Laminar Flow*. Wiley, 2000.
[35] C. Batty, F. Bertails, and R. Bridson, "A fast variational framework for accurate solid-fluid coupling," *ACM TOG*, vol. 26, no. 3, pp. 100:1–100:7, 2007.
[36] F. Gibou, R. P. Fedkiw, L.-T. Cheng, and M. Kang, "A second-order-accurate symmetric discretization of the poisson equation on irregular domains," *Journal of Computational Physics*, vol. 176, no. 1, pp. 205–227, 2002.
[37] J. Nitsche, "Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind," *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, vol. 36, no. 1, pp. 9–15, 1971.
[38] J. Dolbow and I. Harari, "An efficient finite element method for embedded interface problems," *International Journal for Numerical Methods in Engineering*, vol. 78, no. 2, pp. 229–252, 2009.
[39] I. Harari and J. Dolbow, "Analysis of an efficient finite element method for embedded interface problems," *Computational Mechanics*, vol. 46, no. 1, pp. 205–211, 2010.
[40] C. Min and F. Gibou, "Geometric integration over irregular domains with application to level-set methods," *Journal of Computational Physics*, vol. 226, no. 2, pp. 1432–1443, 2007.
[41] J. L. Hellrung Jr., L. Wang, E. Sifakis, and J. M. Teran, "A second order virtual node method for elliptic problems with interfaces and irregular domains in three dimensions," *Journal of Computational Physics*, vol. 231, no. 4, pp. 2015–2048, 2012.
[42] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proc. SIGGRAPH*, 1987, pp. 163–169.
[43] A. Stroud, *Approximate calculation of multiple integrals*. Prentice-Hall, 1971.
[44] M. J. Aftosmis, M. J. Berger, and G. Adomavicius, "A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries, AIAA 2000-0808," in *Proc. 38th AIAA Aerospace Sciences Meeting and Exhibit*, 2000.
[45] J. Teran, E. Sifakis, S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw, "Creating and simulating skeletal muscle from the visible human data set," *IEEE TVCG*, vol. 11, no. 3, pp. 317–328, 2005.
[46] M. Nesme, P. G. Kry, L. Jeřábková, and F. Faure, "Preserving topology and elasticity for embedded deformable models," *ACM TOG*, vol. 28, no. 3, pp. 52:1–52:9, 2009.
[47] R. Crockett, P. Colella, and D. Graves, "A Cartesian grid embedded boundary method for solving the Poisson and heat equations with discontinuous coefficients in three dimensions," *Journal of Computational Physics*, vol. 230, no. 7, pp. 2451–2469, 2011.
[48] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed. SIAM, 2000.
[49] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*. Academic Press, 2001.
[50] F. Liehr, T. Preusser, M. Rumpf, S. Sauter, and L. O. Schwen, "Composite finite elements for 3D image based computing," *Computing and Visualization in Science*, vol. 12, no. 4, pp. 171–188, 2009.
[51] W. Wang, "Special bilinear quadrilateral elements for locally refined finite element grids," *SIAM Journal on Scientific Computing*, vol. 22, no. 6, pp. 2029–2050, 2000.

Florian Ferstl received his MSc in computer science 'with distinction' from the Technische Universität München in 2011. Since 2012, he is a PhD student at the Technische Universität München in the Computer Graphics and Visualization Group headed by professor Rüdiger Westermann. His research interests include physics-based simulation of fluids, finite element methods, multigrid methods, interactive flow visualization and GPU computing.

Rüdiger Westermann studied computer science at the Technical University Darmstadt, Germany. He pursued his Doctoral thesis on multiresolution techniques in volume rendering, and he received a PhD in computer science from the University of Dortmund, Germany. In 2002, he was appointed the chair of computer graphics and visualization at the Technical University Munich. His research interests include scalable simulation and visualization algorithms, GPU computing, real-time rendering of large data, and uncertainty visualization.

Christian Dick is a PostDoc in the Computer Graphics and Visualization Group at the Technische Universität München, Germany. He received a diploma in computer science in July 2007 and a PhD in January 2012, both from Technische Universität München. His research is focussed on interactive simulation and visualization methods, including physics-based simulation of deformable objects and fluids, simulation and visualization in the context of medical applications, as well as the visualization of very large scientific data sets.