Computing and Visualization in Science manuscript No.
(will be inserted by the editor)

Photorealistic Visualization and Fluid Animation - Coupling of Maya with a Two-Phase Navier-Stokes Fluid Solver

Peter Zaspel · **Michael Griebel**

Received: date / Accepted: date

Abstract We have coupled the three-dimensional solver for the two-phase incompressible Navier-Stokes equations NaSt3DGPF with Autodesk Maya. Maya is the industry standard software framework for the creation of three-dimensional animations. The parallel level-set-based fluid solver NaSt3DGPF simulates the interaction of two fluids like air and water. It uses high-order finite difference discretization methods that are designed for physics applications. By coupling both applications, we are now able to set up scientific fluid simulations in an easy-to-use user interface. Moreover, the rendering techniques provided by Maya allow us to create photorealistic visualizations for computational fluid dynamics problems and support the creation of highly visually realistic fluid simulations for animation movies. Altogether, we obtain an easy usable and fully coupled fluid animation toolkit for two-phase fluid simulations. These are the first published results of the full integration of a physics-oriented, high-order grid-based parallel two-phase fluid solver in Maya, at least to our knowledge.

Keywords Photorealistic visualization · computational fluid dynamics · user interface · multi-phase flows · water · animation

P. Zaspel
Institute for Numerical Simulation
University of Bonn, Germany
Tel.: +49-228-732748
Fax.: +49-228-737527
E-mail: zaspel@ins.uni-bonn.de

M. Griebel
Institute for Numerical Simulation
University of Bonn, Germany
Tel.: +49-228-733437
Fax.: +49-228-737527
E-mail: griebel@ins.uni-bonn.de

1 Introduction

Publishing research results for a broader community has always been a challenging task. Thus, in the domain of *scientific visualization*, we are interested in giving scientists an easy understandable representation of computational results. Let us exemplify this for this paper in the domain of computational fluid dynamics (CFD) or even more specific for liquid or two-phase fluid simulations with two interacting fluids like air and water. Standard applications for scientific visualization in CFD allow us to represent vector fields by glyphs and scalar fields by colored cut planes, isosurfaces or volume rendering. However, these representations are often still too detailed or complicated to illustrate research results to a broader community of non-scientists. This is why *photorealistic visualizations* (see Figure 1) may be used since people's visual reception practice is much better adapted to this kind of representation. Furthermore, we see that documentaries, commercials or popular journals increasingly create and use fluid simulations to transport their information or message. Obviously, the viewer is interested to see visually realistic animations/pictures with a high grade of physical accuracy. Consequently, we are now looking for a fluid simulation tool which is easy usable for everybody, has a high grade of physical realism and can create photorealistic visualizations.

There has been a broad development of *research* CFD codes specifically for animation and visualization oriented applications. Here, most *grid*-based fluid simulation engines model fluid behavior by the well-known three-dimensional incompressible Navier-Stokes equations. This approach was initially introduced to the computer animation community by Foster and Metaxas [13,14]. They discretized the equations using the finite difference approach on a staggered uniform grid. In [13], they especially focused on a one-phase fluid body with an additional free surface to obtain an ani-

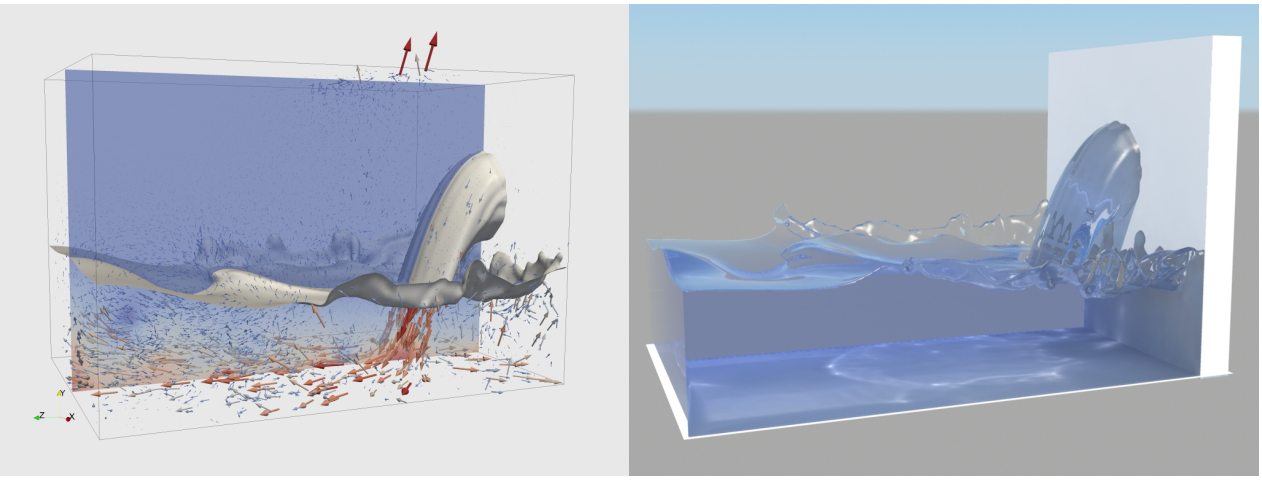

Fig. 1 Photorealistic visualizations facilitate the interpretation and comprehension of two-phase fluid simulations for non-scientists.

mation of a liquid including its surface. Furthermore, Stam [30] introduced semi-Lagrangian time advection schemes into fluid animation which resulted in stable simulations with large time steps, but with a considerably increased numerical diffusion. This issue of numerical diffusion was addressed in a wide number of publications by [10,29,28]. Foster and Fedkiw [12] as well as Enright et al. [9] combined the free surface representation by a level-set function with massless particles. This resulted in the *particle level-set function* which gave an improved liquid surface appearance and a better mass conservation. Enright et al. [8] used the fifth-order Weighted Essentially Non-Oscillatory scheme to discretize the level-set advection and reinitialization. Later on, several authors [19,24,22,25] covered the topic of visual improvements by adding particles. Kang et al. [20] were among the first to develop a two-phase Navier-Stokes solver for the computer graphics community which simulates two fluids like air and water at the same time. This approach leads to an improved realism, as the true interaction of the air and the water can be reproduced. Technically, they applied the *Ghost Fluid* method [11], to capture a discontinuous density jump, and the level-set method for the free surface representation with a third order ENO scheme to compute the transport term. Additional developments can be found in [32,18,23,25,22]. Furthermore, there is a wide variety of engineering- or numerics-oriented CFD literature. Here, we do not go into further details to be concise. However, even though the latest CFD research codes use strong numerical methods, there are no publications on their integration into an easy usable graphical animation interface.

On the other hand, standard engineering CFD packages like *Ansys CFD*, *FLUENT* or *Comsol Multiphysics* allow us to create fluid simulations using advanced CFD solvers in a graphical user interface. They can also generate scientific visualizations. Ansys CFD can even produce some sort of animations. But in general, these user interfaces are very complex and thus might not be suitable for e.g. documentary production. One can also use *animation* software from the computer graphics market, which sometimes provides easy-to-use fluid simulation integrations and supports the generation of photorealistic renderings. The industry standard software in this domain is *Autodesk Maya*. Here, *Maya Fluids* and *Glu3d* are fluid animation extensions that can be directly integrated into Maya. Additionally the *RealFlow* toolkit supports Maya. These fluid animation extensions use particle-based approaches like the *Smoothed Particle Hydrodynamics* (SPH) methods introduced by Monaghan [26]. *Autodesk Softimage* also has particle-based incompressible fluids. Thürey [33] integrated a different technique using a *Lattice-Boltzmann* fluid solver in the Open Source alternative for Maya, called *Blender*. Particle methods and Lattice-Boltzmann-like methods are fast but often physically not very accurate. Higher order grid-based fluid solvers at high resolutions typically outperform these other methods when it comes to visual and physical realism. The animation software toolkit *Houdini* allows to simulate liquids with a grid-base level-set method, which is an advanced technique. However, it is not perfectly clear which numerical methods it uses to simulate liquids. Consequently it might not be a good choice for scientific applications. Altogether, no existing software package fulfills our requirements.

This is why we now present a coupling of the three-dimensional two-phase solver for the incompressible Navier-Stokes equations NaSt3DGPF with Autodesk Maya. The fluid solver NaSt3DGPF was originally developed for physics applications such as the simulation of water ways [4], droplet deformations [5], flows through porous media [34] or coating processes. It uses the level-set method to distinguish the two fluid phases and WENO/ENO schemes for high-order level-set and velocity transport space discretizations and improved mass conservation, see [5]. Consequent-

ly, the solver is optimized for high accuracy of the approximated Navier-Stokes solution.

As a result of the proposed coupling the whole production pipeline for the creation of fluid simulations including photorealistic visualizations can now be controlled from within Maya's three-dimensional user interface which is easy usable for a broader community. Furthermore, since Maya is the industry standard for digital artists in movie production, we imagine not just to apply our coupling for visualization and a facilitated simulation setup but to use it for classical *fluid animation* or even *fluid special effects*. The applied fluid solver is able to compete with most of the latest CFD codes for movie production and even overcomes their functionality e.g. in the domain of the application of WENO schemes even for convective terms.

The contributions of this article are as follows:

- One can now set up high quality fluid simulations from within an easy-to-use three-dimensional user interface
- Photorealistic visualizations can be created for fluid simulations computed by NaSt3DGPF. This greatly facilitates the explanation of CFD research results to non-scientists.
- We present the successful coupling of a physics-oriented research software code and an industrial standard software computer graphics animation framework.
- To our knowledge, the first integration of a parallel grid-based truly two-phase solver for the three-dimensional Navier-Stokes equations in Autodesk Maya is achieved.
- Digital artists can now use Maya to create two-phase flow simulations with high visual realism and numerical accuracy.

The remainder of this article is organized as follows. In **Section 2** we present our two-phase solver for the incompressible Navier-Stokes equations. This includes a short summary of the model equations and the numerical solution methods. **Section 3** briefly introduces the application framework of Autodesk Maya as well as its programming interface. In **Section 4** we describe and discuss the details of our coupling and sketch some ideas of the implemented components. **Section 5** shows visualization and animation results that were created using our coupling. **Section 6** concludes the main ideas of this article and gives a short outlook on future work.

2 NaSt3DGPF - A solver for the 3D two-phase Navier-Stokes equations

2.1 Model equations

The model equations for two-phase incompressible flows with surface tension in NaSt3DGPF (see [31,7,5]) are based on on the Navier-Stokes equations. Following the notation of Croce et al. [5], they can be expressed by the momentum equation

$$\rho(\phi)\frac{D\mathbf{u}}{Dt} + \nabla p = \nabla\cdot(\mu(\phi)\mathbf{S}) - \sigma\kappa(\phi)\delta(\phi)\nabla\phi + \rho(\phi)\mathbf{g} \quad (1)$$

and the continuity equation

$$\nabla\cdot\mathbf{u} = 0\,. \quad (2)$$

Here, the fluid velocity $\mathbf{u}$ and pressure p are time- and space-dependent with the material derivative $\frac{D\mathbf{u}}{Dt} := \partial_t\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}$. Volume forces are given as $\mathbf{g}$. By introducing the level-set function ϕ, here a signed distance function with

$$\phi(\mathbf{x},t)\begin{cases} <0 & \text{if } \mathbf{x}\in\Omega_1 \\ =0 & \text{if } \mathbf{x}\in\Gamma_f \\ >0 & \text{if } \mathbf{x}\in\Omega_2 \end{cases} \qquad \text{and} \qquad |\nabla\phi| = 1\,,$$

we can formulate a domain-dependent fluid density ρ and viscosity μ for two fluid phases Ω_1 and Ω_2 by

$$\begin{matrix}\rho(\phi) := \rho_2 + (\rho_1-\rho_2)H(\phi) \\ \mu(\phi) := \mu_2 + (\mu_1-\mu_2)H(\phi)\end{matrix} \quad H(\phi) := \begin{cases} 0 & \text{if } \phi<0 \\ \frac{1}{2} & \text{if } \phi=0 \\ 1 & \text{if } \phi>0\,. \end{cases}$$

Thus, the free surface Γ_f (e.g. the water surface) is modeled discretely by $\Gamma_f(t) = \{\mathbf{x} : \phi(\mathbf{x},t) = 0\}$. It is thus given implicitly as isosurface for the zero level-set of ϕ. The curvature κ of the free surface, the surface tension coefficient σ and the Dirac functional δ are included in the surface tension force term which is based on the *Continuum Surface Force* (CSF) scheme. The tensor $\mathbf{S}$ in the diffusion term is given by $\mathbf{S} := \nabla\mathbf{u} + \{\nabla\mathbf{u}\}^T$. Interested readers are referred to [5] and [7] for more detailed introductions to the above model equations.

2.2 Numerical solution techniques

In the solver NaSt3DGPF, the two-phase Navier-Stokes equations are discretized with the finite difference method on a staggered uniform grid. For a coupled solution of the momentum equation (1) and the continuity equation (2) over time, the projection approach introduced by Chorin [1] is applied. The method starts by a time discretization of the momentum equation with a skipped pressure gradient term. This allows us to compute a velocity field for a new time step. To enforce incompressibility, i.e. to satisfy the continuity equation, a pressure Poisson equation is solved. Finally a pressure correction is added to the velocity field which removes the divergence as required by equation (2).

The outlined method is combined with a level-set transport mechanism to get a full simulation for two-phase flows. A sketch of the resulting solution approach is given by the following time-discrete algorithm:

For each computational time step n with step size δt do:

1. compute intermediate velocity field $\mathbf{u}^*$

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\delta t} = -(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n + \mathbf{g} + \frac{1}{\rho(\phi^n)} \nabla \cdot (\mu(\phi^n)\mathbf{S}^n) - \frac{1}{\rho(\phi^n)} \sigma \kappa(\phi^n) \delta(\phi^n) \nabla \phi^n$$

2. transport level set function

$$\phi^* = \phi^n - \delta t \, (\mathbf{u}^n \cdot \nabla \phi^n)$$

3. reinitialize level set function by solving

$$\partial_\tau d + sign(\phi^*)(|\nabla d| - 1) = 0, \; d^0 = \phi^*$$

4. solve the pressure Poisson equation with $\phi^{n+1} = d$

$$\nabla \cdot \left(\frac{\delta t}{\rho(\phi^{n+1})} \nabla p^{n+1} \right) = \nabla \cdot \mathbf{u}^*$$

5. apply velocity correction

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\delta t}{\rho(\phi^{n+1})} \nabla p^{n+1}$$

Note that this algorithm only demonstrates the method for the case of a first-order Euler time discretization. In our implementation of the solver, time discretization methods of second or third order such as the Adams-Bashforth or the Runge-Kutta method are used. Space discretizations of the convective terms are done using fifth-order WENO, ENO or VONOS schemes. The simulation code provides a Jacobi-preconditioned BiCGStab solver and an algebraic multi-grid solver [16] for the linear system of the discretized Poisson equation.

A crucial part of the level-set method lies in the conservation of the distance property $|\nabla \phi| = 1$. Unfortunately this property is disturbed after the transport of the level-set function in step 2 of the algorithm. This is why a so-called reinitialization of the level-set function is performed in step 3. Here, a partial differential equation of Hamilton-Jacobi type is numerically solved in artificial time τ to recover the distance property.

The fluid solver NaSt3DGPF is able to perform simulations in user-defined geometries with arbitrarily shaped obstacles. A flag technique is used to mark solid computation cells and to apply appropriate boundary conditions on the resulting obstacles.

A major strength of the fluid solver is its parallelization. On can perform simulations on large clusters of workstations or on supercomputers. For that, the domain decomposition approach of Schwarz is used. Its basic idea is to divide the computational domain into parts that can be processed independently on a parallel computer. Furthermore, some boundary data must be exchanged between the processors, if needed. The implementation of the solver is based on C++ and uses the *Message Passing Interface* (MPI) as parallelization framework.

Other features of NaSt3DGPF include a large-eddy turbulence model of Smagorinsky, passive transport of species and the simulation of energy (thus temperature) based on the Boussinesq approximation [7].

2.3 User interface of the original solver

To better understand the requirements for a coupling between Maya and the fluid solver, we need to characterize the user interface of NaSt3DGPF. The original fluid simulator comes with a command line interface. It uses text-based configuration files that describe simulation parameters. The overall geometry of a scene is either defined by basic geometric objects like boxes/spheres that can be combined with Boolean operations or by pre-discretized geometric objects. The latter are given in external files: To perform the flagging for solid objects, boundaries or initial conditions for the fluids in the discretization grid, a Boolean value per cell is saved. The value indicates, whether the cell is inside or outside the geometric object. This type of discretization is often called *voxelization* in computer graphics.

A preprocessor of NaSt3DGPF generates the necessary three-dimensional data fields from the configuration files, which are then used as initial conditions for the parallel solver. During the solution process, a subset of the raw simulation data can be written to data files for given computational time steps.

After completion of a calculation, the saved raw simulation data of the original fluid solver can be post-processed and transformed to data formats used by visualization applications like *Paraview* and *Tecplot*. These applications allow scientific visualizations.

3 Autodesk Maya - A software framework for computer animation

Maya is a well-known software framework for the creation of computer-generated images and animations. It has a user interface for the interactive construction of three-dimensional scenes. Geometrical objects, which are described by polygonal meshes or *NURBS*, can be modeled and animated in various ways. Textures or materials are applied to the objects' surfaces to define their appearance. Different sorts of lights create a realistic illumination of the scene. Cameras can be placed, as if it would be a real film set. The final images are created by a *renderer* like *Mental Ray*, which is supplied with Maya.

3.1 Base components

Following the description in [15], the software framework of Maya is composed of three layers:

1. the graphical user interface
2. the script interpreter
3. the dependency graph

Each action in the *user interface* is directly translated to a script command. Maya uses its own scripting language, the *Maya Embedded Language (MEL)*. The *script interpreter* handles all these commands. Most of the commands are designed to modify the *dependency graph*, which is the low-level representation of the animation scene. Each shape, object, material, camera, geometrical transformation, etc. is described by a single *node* in the graph. By connecting these nodes one can set up an arbitrarily complex three-dimensional scene.

3.2 Developer framework

There are two main concepts to extend the functionality of Maya: *scripting* and *plugins*.

Scripting was originally based on the script language MEL. During the last years *Python* has also been included as script language. It now gains more and more importance for Maya developers. The whole user interface is based on scripting and extending Maya with new menus, windows, etc. can be also done with scripts. In addition to this, scripts help to automatize recurring actions.

Plugins collect extensions that are implemented based on a C++ application programming interface (API) for Maya. The API allows writing new nodes for the dependency graph and thus is a powerful way to extend the core functionalities. Additionally one can implement new so-called *commands* for the MEL language that are basically C++ functions which allow dependency graph modifications. Precompiled plugins are used to distribute commercial extensions for Maya.

4 Coupling Maya and NaSt3DGPF

By coupling Maya with NaSt3DGPF, one is able to control the full fluid simulation pipeline from within Maya's user interface. In this section, we will outline this coupling. After sketching the broad idea of the designed workflow within Maya, we give a deeper insight into the implemented software components and point out some of the details. We end with a short design discussion.

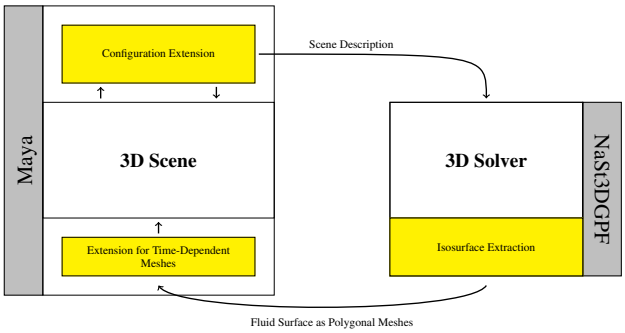

Fig. 2 The necessary components for a coupling between Maya and NaSt3DGPF (new components are highlighted in yellow)

4.1 Workflow

At the beginning of the animation process, the user selects a simulation domain which is shaped like a rectangular box. Arbitrary geometrical objects, given by polygonal meshes, can then be placed inside this domain. They are included as solid obstacles in the fluid simulation after a selection and application of the appropriate menu option. Fluid objects are defined in the same way. Boundary conditions like inflow-/ outflow-regions or slip-/noslip-boundaries are applied by selecting the designated areas with a special geometrical object. Most of the fluid solver parameters, including density, viscosity, gravity or grid resolution can be accessed by dialog controls. Furthermore, all necessary parameters for a standard air-water-simulation are already preset. The fluid solver application is launched by a menu option. Necessary discretizations/voxelizations of geometry objects are performed fully-automatic in the background. During the solution process, the solver generates polygonal meshes of the free surface between the fluid phases (e.g. the water surface) for a specified animation frame rate. They are then automatically loaded back into Maya. The last step of the fluid animation pipeline is a photorealistic rendering of the free surface meshes.

4.2 Necessary components

Based on the outlined workflow and the description of the user interface of NaSt3DGPF in Section 2.3, one can now more rigorously formulate the components that are necessary for the coupling of Maya and the fluid solver. We need a *configuration extension* for the graphical user interface of Maya which allows setting up the fluid simulation and which creates a scene description for the fluid solver. Additionally an *isosurface extraction* extension for NaSt3DGPF is necessary. As part of the simulation algorithm, it computes the appropriate polygonal meshes of the fluid surface for the computational time steps in the simulation that will be used as animation frames. Finally, Maya needs an *extension for*

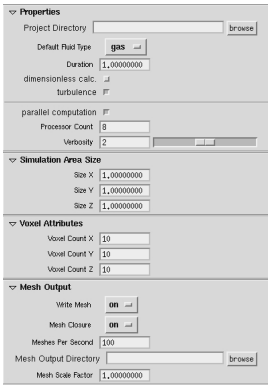

Fig. 3 Small part of the *attribute editor template* shown on the right-hand side of Maya's user interface after selecting the graphical representation of the `SimulationArea` node

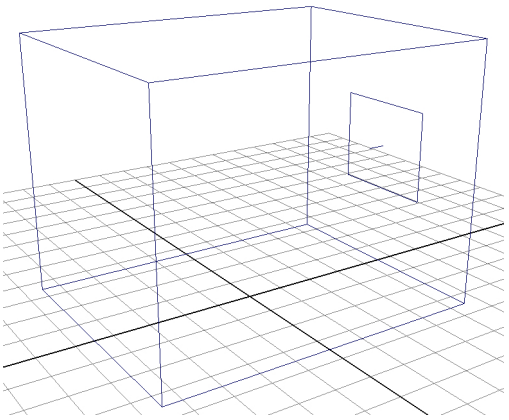

Fig. 4 Graphical representation of the `SimulationArea` node in the three-dimensional scene. An inflow boundary is attached to the right-hand side of the simulation domain.

time-dependent meshes which automatically loads appropriate meshes based on a given animation frame number. Figure 2 sums up the above mentioned components.

4.3 Configuration extension

Functionally speaking, the configuration extension has to represent the parameters of a fluid simulation scene in some graphical and easily usable way. Additionally it is necessary to convert this representation into data known by NaSt3DGPF.

4.3.1 Data representation

We use Maya *dependency graph nodes* to represent the data structures which collect the fluid simulation parameters. There is one main node, the `SimulationArea` node, which contains the most important parameters of a simulation as e.g. the grid resolution, domain sizes, viscosities and densities of the two simulated fluids, volume forces and numerical methods for time and space discretizations with appropriate preset values.

In addition, there are nodes, which are attached to polygonal objects, that define the representation of this object in the fluid simulation scene. By attaching a `SimulationSolidShape` node to a polygonal mesh, the shape of the mesh is used as solid obstacle in the fluid simulation. The node stores the boundary conditions which will be applied to the obstacle. On the other hand, the `SimulationFluidShape` node allows the use of a mesh as initial fluid shape (e.g. a sphere represents the initial shape of a liquid drop) and collects parameters as the initial velocity of the fluid, the fluid type or the initial pressure.

Finally, inflow boundary conditions (i.e. Dirichlet boundary conditions) and outflow boundary conditions (i.e. Neumann boundary conditions) are represented by the `SimulationInflowBoundary` node and the `SimulationInOutBoundary` node which store e.g. the location and (in the case of an inflow boundary) the inflow velocities. We also allow the setup of *arbitrarily* shaped inflow boundaries (i.e. not only rectangular inflows) by intersecting a plane of the special `SimulationCustomInflowBoundary` node with a polygonal shape, which will be explained further in paragraph 4.3.3.

All the above mentioned nodes have some sort of graphical representation in Maya. While they all have a so-called *attribute editor template* (see Figure 3) which is basically a property dialog sheet that allows the easy modification of the node's data, some of them also have an actual shape in the three-dimensional scene representation. The `SimulationArea` is thus represented by a rectangular box in the scene, which can be resized with the mouse. Nodes for the inflow / outflow boundary conditions are also represented by resizable rectangular shapes (see Figure 4).

4.3.2 Converter for fluid simulation data

The conversion process between the simulation data in Maya and the representation for NaSt3DGPF is performed in a single Maya *command*. It basically reads the data of all the relevant nodes and creates geometrical information from the graphical description of the fluid simulation. We have chosen to pre-discretize, thus to voxelize, all the fluid/solid geometries and the geometry of the inflow/outflow boundaries. The output of the conversion process is an ASCII file in the scene description format of NaSt3DGPF and several files which represent the voxelized geometries (see Figure 5). Note that the conversion method checks each parameter for valid input ranges such that non-functional configurations are in general prevented or corrected before a simulation is computed.

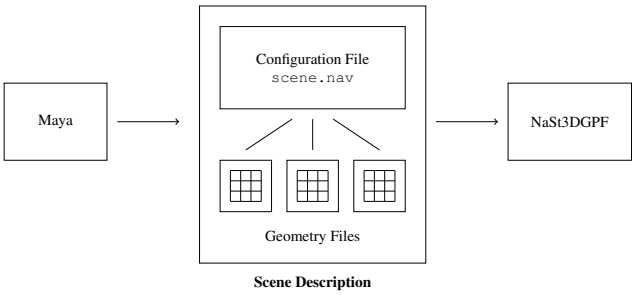

Fig. 5 A full scene description for NaSt3DGPF is composed of an ASCII configuration file and several files containing voxelized geometries.

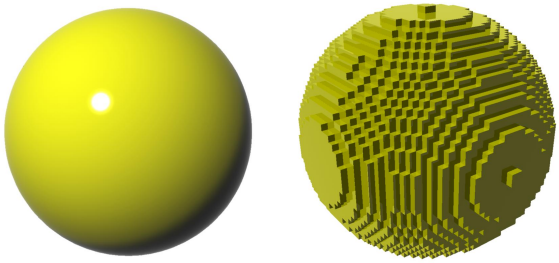

Fig. 6 Example of a low-resolution voxelization (right-hand side) of a sphere geometry based on a polygonal mesh (left-hand side)

The voxelization was implemented directly on top of the Maya API. Here, we apply the existing ray intersection framework of Maya and use, in contrast to methods presented by Kaufman and Décoret [21,6], the classical voxelization approach to count ray intersections: It is well-known that, given a closed polyhedron, one can detect from a given point in space whether this point lies inside or outside of the polyhedron by counting ray intersections. If a ray from this point has an uneven number of intersections with the polyhedron, it lies inside, otherwise it is outside. As pre-build geometries (i.e. geometries from large geometry databases) are often not fully watertight we cast several rays per cell in random directions and use majority decision to find out whether this cell is inside a polygonal mesh or outside. In Figure 6 one can find a low-resolution voxelization example.

4.3.3 Inflow boundaries for two-phase flows

At this point, we would like to highlight one quite technical feature which is representative for the challenges we faced during the implementation: The correct realization of inflow boundaries in two-phase fluid simulations. As mentioned previously, we allow the user to define inflow boundaries using a `SimulationInflowBoundary` node. The graphical representation of this node is an axis parallel plane which can be arbitrarily placed either at the boundaries of the simulation domain or on solid obstacles. Inflows are mathematically defined by Dirichlet boundary conditions on the solid boundary cells which effectively set a (fixed) velocity, the inflow velocity, for these cells. Figure 7 shows this on the left-hand side for the profile of a rectangular inflow boundary for the case of a *single-phase* flow simulation. Here, solid simulation cells are shaded with a line pattern. Cells with a Dirichlet boundary condition are indicated by an arrow.

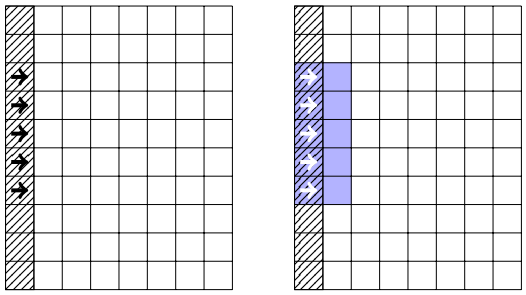

Fig. 7 Profile of a boundary cell definition for single-phase inflows (left-hand side) and a first attempt to define a liquid inflow in domain filled with gas (right-hand side). The liquid phase is shaded blue, the gas phase is white. Solid cells have a line pattern and inflow cells are marked by an arrow.

Let us now assume that we would like to define a liquid inflow, while the remaining domain is in the gas phase. Consequently, we have to make sure, that the inflow boundary is actually in the liquid phase. This can be accomplished by the configuration shown on the right-hand side of Figure 7. The blue cells are in the liquid phase while the white cells are in the gas phase. We have to set the non-solid cell layer in front of the Dirichlet boundary cells to the liquid phase since we would otherwise just define an inflow condition for the (white) gas phase.

Using this definition of a two-phase inflow boundary in connection with gravity forces pointing from the top of the figure to the bottom would result in liquid inflow boundary which would vanish over time. A reason for this lies in the gravity-based transport of the liquid phase: It moves down. Consequently, the inflow boundary is no longer covered by the one-cell layer liquid phase and thus becomes an inflow boundary for the gas phase. This drawback is common to all grid-based level-set implementations of two-phase flow simulations, which do not explicitly overwrite the cell layer in front of the Dirichlet boundary cells for each time step. However, such an approach could cause mass balance problems.

Our way to overcome this issue is based on a fixation of the level-set values around the inflow boundary. Figure 8 outlines this approach. On the left-hand side of the figure, one can now observe solid cells at the boundaries of the inflow region. The right-hand side of the figure presents the front view of the rectangular inflow. Here, one can clearly see our approach: An inner boundary cell layer around the

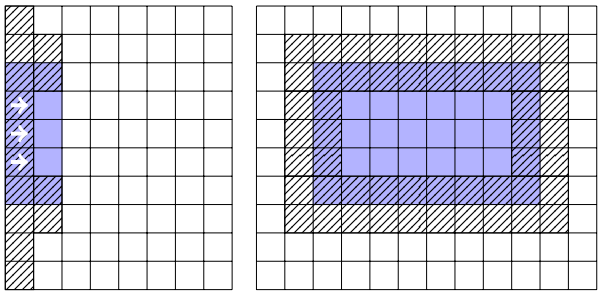

Fig. 8 Modified version of of the inflow boundary (left-hand side: profile view, right-hand side: front view). The inflow is now stabilized and will no longer vanish based on gravitational forces.

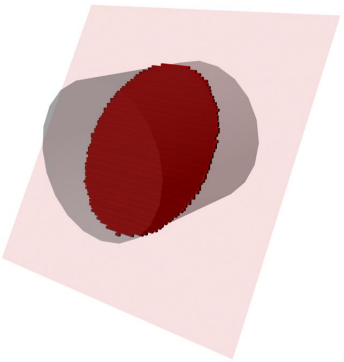

Fig. 9 A voxelization along a plane through a polygonal object (here: a cylinder) is used to construct arbitrarily shaped inflow boundaries.

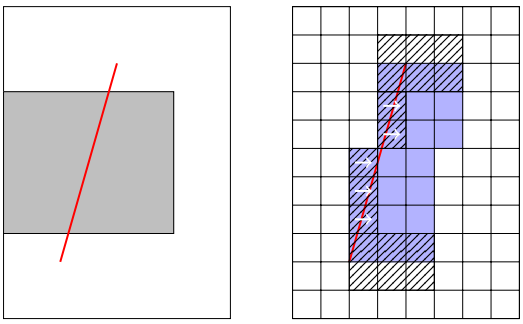

Fig. 10 Left-hand side: The red line is the 2D-version of the inflow plane of the `SimulationCustomInflowBoundary`. The gray shaded object is the polygonal mesh which is used to define the actual inflow region as cut plane. Right-hand side: This is the required cell configuration for a stabilized two-phase inflow.

inflow region is now set as solid cells in the liquid phase. The outer cell layer is composed by solid cells which are flagged as gas cells. By this construction, we are able to stabilize the inflow boundary.

In the more general case of arbitrarily shaped inflow boundaries, the graphical representation of the `SimulationCustomInflowBoundary`, which is a customly placed and rotated plane, will be intersected with an arbitrary polygonal object. The cut plane between both objects becomes the inflow boundary. It can be defined by voxelizing the polygonal object along the inflow plane, see Figure 9 for an example showing a voxelization of a cylinder along a non-axis-aligned plane. An appropriate inflow boundary for such a non-planar inflow is suggested in Figure 10 which has on the left-hand side a simplified version of the problem and on the right-hand side an appropriate cell configuration for the solver. Note that we have to use here a double-layer of liquid cells since we need at least one liquid fluid cell at the thinnest part of the inflow in the plane surface normal direction. While defining stabilizing solid cells for *rectangular* inflows is quite easy, it is not as simple for a voxelized, rotated cut-plane with an arbitrary shape. The quite technical details of this task are however beyond the scope of this paper and might be described elsewhere. A successful realization of a circular inflow to a tank of water is presented in Figure 1.

4.4 Isosurface extraction

In the original version of the fluid solver NaSt3DGPF, the state of the simulation is saved in a binary data file for a subset of the computational time steps. These binary files, each describing one time step, are converted to a visualization file format and finally one can visualize e.g. the free surface in a program like *Paraview*. The drawback of this approach for fluid animation and photorealistic visualization is evident: Binary data files for full high-resolution three-dimensional simulations often take hundreds of megabytes just for one time step. If one simulates a longer period of time, the required disk space grows dramatically and reaches tens or even hundreds of gigabytes. Additionally, storing binary files and converting them to a second format later on to create a visualization even doubles the necessary disk space.

This is why we now only store the kind of data that we really need for our visualizations: This is the free surface as polygonal mesh. Thus, instead of three-dimensional flow data, only the two-dimensional surface data needs to be stored for each animation frame. This results in a substantial reduction in storage. Consequently, we have implemented an isosurface extraction method into the fluid solver which writes polygonal meshes during the fluid simulation. To this end, we employ the well-known *Marching Cubes algorithm* of Cline and Lorensen [2] in the improved version described by Montani et al. in [27]. As output file format we utilize the well-known *Wavefront OBJ* format for polygonal meshes.

One nice property of the Marching Cubes algorithm is that it is highly parallel. We thus can perform the isosurface extraction in parallel for each subdomain of the parallelized fluid solver without any data communication. It is even possible to store the extracted surfaces in independent files which avoids a parallel data output into one file.

We only have to take care of appropriately connecting the meshes when they are loaded into Maya.

4.5 Extension for time-dependent meshes

Loading the generated fluid surface meshes into Maya is a non-trivial operation. Here, we face two major difficulties: The still large amount of generated data and the lack of a light-weight mechanism in Maya to load time-dependent meshes. Even though we do not store the full three-dimensional fluid simulation data explicitly we still generate a large amount of high-resolution polygonal meshes over time. Loading these meshes into Maya at the same time would normally inhibit an efficient workflow due to the lack of enough main memory. Even though the existing so-called *proxy* in Maya should allow loading only data from storage when it is necessary, it turned out to be not flexible enough[1] in the case of time-dependently loaded meshes in the Wavefront OBJ format.

We thus implemented our own component to load time-dependent meshes. From the user's point of view, there is now a standard mesh (node), which can be transformed, scaled, sharpened, smoothed, etc. like every other mesh. The only difference is that it loads different mesh geometries given by Wavefront OBJ files for different time frame numbers from the file system. Consequently, we only have one mesh at a given point of time loaded into memory which makes this method very light-weight in terms of memory usage. By doing this, we can also include the reconnection of the meshes, which were written parallelly into different files, such that they look like one mesh for the user.

4.6 Design discussion

The design we have chosen to implement the presented coupling is build on three basic principles:

1. Keep the coupling as light-weight as possible.
2. Profit from the strengths of the coupled software packages.
3. Try to stay as interoperable as possible.

The first principle caused us to use files as interaction base for both components. One alternative approach can be achieved by putting the whole fluid simulation code directly into a Maya plugin. This may facilitate and optimize data transfers. However, since the fluid solver is also a stand-alone program, the maintenance of the plugin, which is then no longer part of the original code trunk, could become a major issue. Another way to couple both applications can be a network data exchange for both applications. The drawback of this design would be the largely increased complexity of an appropriate client-server system.

Based on the second principle, we decided to implement the voxelization inside the Maya user interface and the free surface extraction inside NaSt3DGPF. As already mentioned, we could directly profit from the line intersection framework of Maya for this process. We are aware that GPU-rasterization-based approaches could be a lot faster than the applied method. On the other hand, the choice of doing the free surface extraction was very much driven by the previously mentioned massive storage requirements of our simulations. If this would not be a problem, we could imagine to transfer the volumetric data which could allow even more interesting fluid effects in Maya based on volumetric rendering techniques.

The last principle caused us to stay with more generic data formats: Obviously, the Wavefront OBJ format is neither the most efficient nor the most storage space saving mesh data format. Nevertheless, we also want to be able to use the mesh extraction mechanism as a stand-alone feature in the fluid solver. To our knowledge, the OBJ format is very wide-spreaded in all sorts of animation and rendering toolkits. So we can also use several other applications to perform the actual rendering. Our voxelization data format could also be optimized by switching to a more sparse representation. Anyway, we never ran into storage problems for the initial voxelization data.

5 Visualization and animation results

In the following section, we will present some visualization and animation results which are based on the proposed coupling of Maya and NaSt3DGPF. This includes results which point out the easy setup up of highly complex fluid simulations, the strength of our system to create photorealistic visualizations for physics applications and the application of the coupling for fluid animation in movie production.

All simulations were performed for a combination of pure water and air at a temperature of 20°C. Volume forces were set to standard gravity. While Autodesk Maya and the applied renderer Mental Ray run on a single workstation, we submit the fluid simulation tasks to a compute cluster. The cluster system consists of 128 compute nodes, each with two single-core Intel Xeon 3.2 GHz processors and 4-6 GB of main memory. In practice we normally use up to 64 processors for single compute tasks. Presented calculation timings in this section reflect the time necessary for the simulations on the cluster. The full animation sequences may be seen in the video accompanying this paper, which is available as Online Resource 1.

[1] This is at least valid for the software version of Maya (2011) that was available during the software development for this paper.

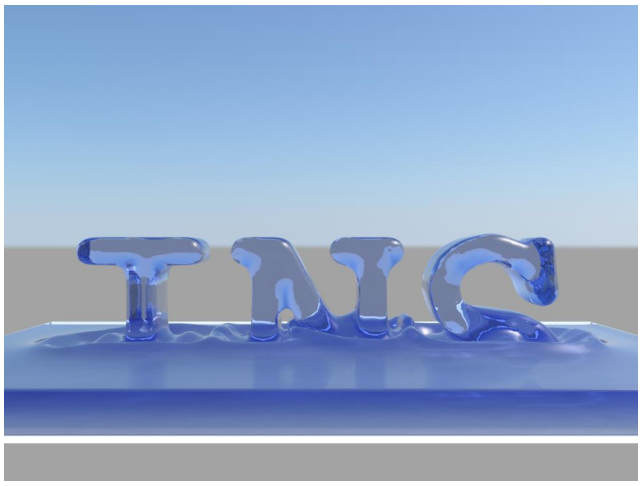

Fig. 11 The letters *I N S* are modeled as a liquid body initial condition. Over time, they vanish in the water volume and create some waves (domain size: $10\,m \times 4\,m \times 6\,m$, resolution: $200 \times 80 \times 120$ cells).

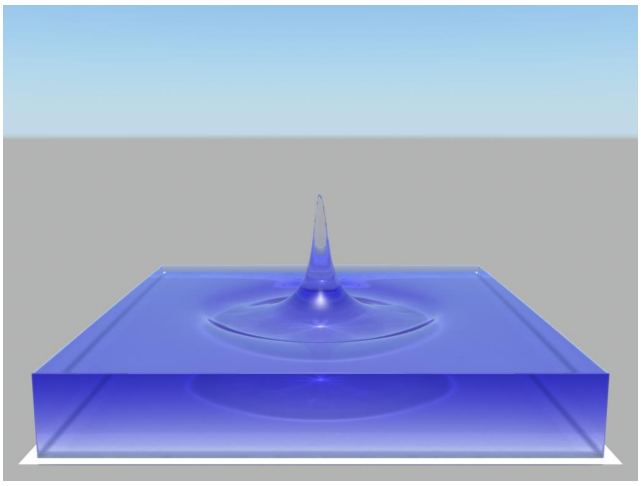

Fig. 12 A fully simulated drop impact in a water basin creates a realistic liquid jet (domain size: $5.76\,cm \times 3.72\,cm \times 5.76\,cm$, resolution: $180 \times 120 \times 180$ cells, drop diameter: $0.576\,cm$).

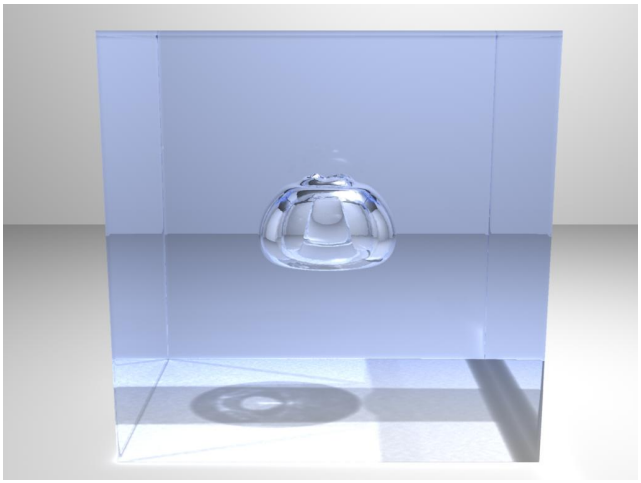

Fig. 13 A large air bubble in water gets ring-shaped while rising (domain size: $20\,cm \times 20\,cm \times 20\,cm$, resolution: 100^3 cells, initial bubble diameter: $6\,cm$).

5.1 Easy fluid simulation setup

A major advantage of the integration of the fluid simulation pipeline into Maya is that we are now able to set up fluid animations for highly complex base geometries with just a few mouse clicks. This is possible since we can use arbitrarily complex polygonal meshes as fluid and solid geometries.

Our first example outlines the ability of modeling complex fluid initial conditions for the two fluid phases. In this artificial case, we take the letters *I N S* as water volumes which disappear over time in a water tank (see Figure 11). We defined the liquid tank as a cuboid. It took us a few seconds to model the letters in Maya. The overall simulation setup can be done in a few minutes. Furthermore, in the supplementary video material, we show the full setup of this simulation scene to prove the easiness of our approach. We did compute this simulation of 5 seconds, thus 125 frames at 8.25 minutes per animated frame on 32 processors for a quite high resolution.

5.2 Photorealistic visualization of scientific results

We are now able to highlight fluid simulation results for scientific applications in a broadly understandable and visually clear way. This will help to transport physics simulation results.

A first example to demonstrate this is the simulation of a droplet impact: If a drop falls into a basin of water, one can typically observe a jet of liquid that leaves the water body after the impact. The NaSt3DGPF solver can simulate this effect. It is modeled by constructing a water basin and a water drop that is initial located above the basin. During the simulation, one can observe the impact and finally the fully realistic creation of a water jet (see Figure 12). We create a visualization with 10x slow motion and a total of 250 frames. The simulation time per frame is about 5.12 minutes when computed in parallel with 64 processors.

Another interesting two-phase flow problem is the behavior of an air bubble in water, cf. [5]. Figure 13 presents the visualized result. A simulation time of 0.4 seconds, thus 200 animated frames for an animation at 20x slow motion, can be computed by 32 parallel processes within 5.63 minutes per frame.

The last example of a photorealistic visualization of a scientific research result comes from the domain of water ways construction, in which the coupled solver was intensively used [4]. Here, we show how water behaves, when it discharges from a dam construction and pours into a water basin. The water tank behind the dam construction is initially filled up and drains over time (see Figure 14). This

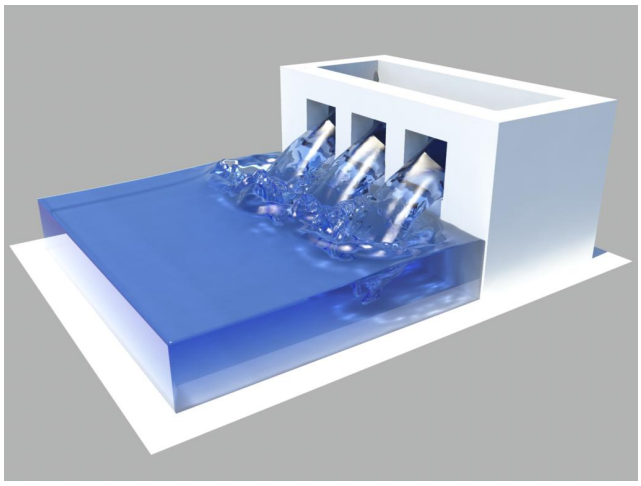

Fig. 14 Water discharges from a dam construction and pours into a water basin (domain size: $24m \times 14m \times 30m$, resolution: $120 \times 70 \times 150$).

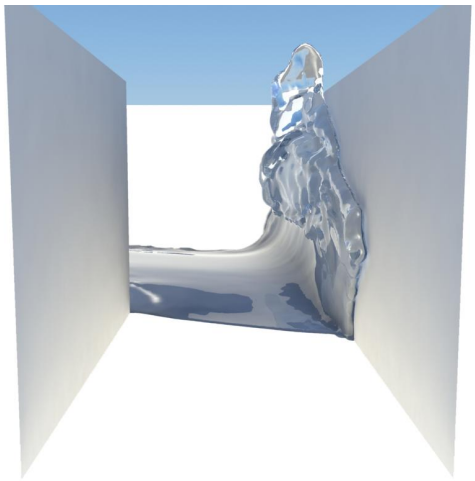

Fig. 15 Water flows through an alley splashing against a wall (domain size: $24m \times 14m \times 30m$, resolution: $120 \times 70 \times 150$).

example is simulated over a time period of 8.5 seconds. We animate 212 frames with a computation time of 1.05 minutes per frame using 64 parallel processes.

5.3 Animations for movie production

In our first example for movie animation, we show water that flows through a small model of an alley. By doing this, it splashes against a wall. This kind of animation is often used for special effects in disaster films. In the final animation (see Figure 15), we use a 4x slow motion for a simulated time of 1.47 seconds. We thus compute 147 frames with a calculation time of 5.65 minutes per frame by 64 processes.

The second case study focuses on a breaking wave that splashes against a lighthouse, cf. [24]. We combine in this simulation most of our effects: Breaking waves can only be simulated by high quality fluid simulation engines. The lighthouse is a complex geometry which can be easily integrated into the fluid simulation with Maya. We also use an additional droplet particle system for improved visual appearance. We create the wave by a water column that is initially placed at one boundary of the simulation domain. When the column collapses, it creates a wave that travels across the domain. The simulated time is 7 seconds, animated in real-time. We thus get 175 frames. On 64 processors, it takes us in average 16 minutes to simulate one frame. In Figure 16 one can observe the realistic development of the wave and the splash particles. When the wave hits the lighthouse, additional splashes are created, which follow the simulated gas phase velocity field. This makes the results even more realistic.

6 Conclusions and Future Work

In this paper, we presented the successful coupling of a physics-oriented fluid solver with an industrial standard animation software framework. The applied parallel fluid simulator NaSt3DGPF is able to approximate the two-phase Navier-Stokes equations. A coupling of the solver and Autodesk Maya includes an easy-to-use interface to set up fluid simulations in Maya, efficient extensions for the fluid solver to extract the fluid surface and new components for Maya to automatically load the data created by NaSt3DGPF for fluid animation. Consequently, we can set up fluid simulations, create photorealistic fluid visualizations and make highly physically realistic animations including two-phase flows available. We also showed a large variety of different complex fluid visualization results.

In the future, we plan to integrate even more features of NaSt3DGPF into Maya. The most important one will be the fluid-structure interaction which was recently developed in [3]. We also intend to design techniques to control the fluid during the simulation from within Maya. Another task is to port parts or even the whole fluid solver NaSt3DGPF to the GPU, cf. [17]. It is evident that this will speed up the simulation process a lot. Obviously, the presented system design can also be adapted to other applications like solid mechanics or acoustics.

Acknowledgements This work was supported in parts by the Sonderforschungsbereich 611 *Singular phenomena and scaling in mathematical models* funded by the *Deutsche Forschungsgemeinschaft*.

References

1. Chorin, A.J.: Numerical solution of the Navier-Stokes equations. Mathematics of Computation **22**(104), 745–762 (1968)
2. Cline, H.E., Lorensen, W.E.: Marching cubes: A high resolution 3d surface construction algorithm. SIGGRAPH Comput. Graph. **21**(4), 163–169 (1987). DOI http://doi.acm.org/10.1145/37402.37422

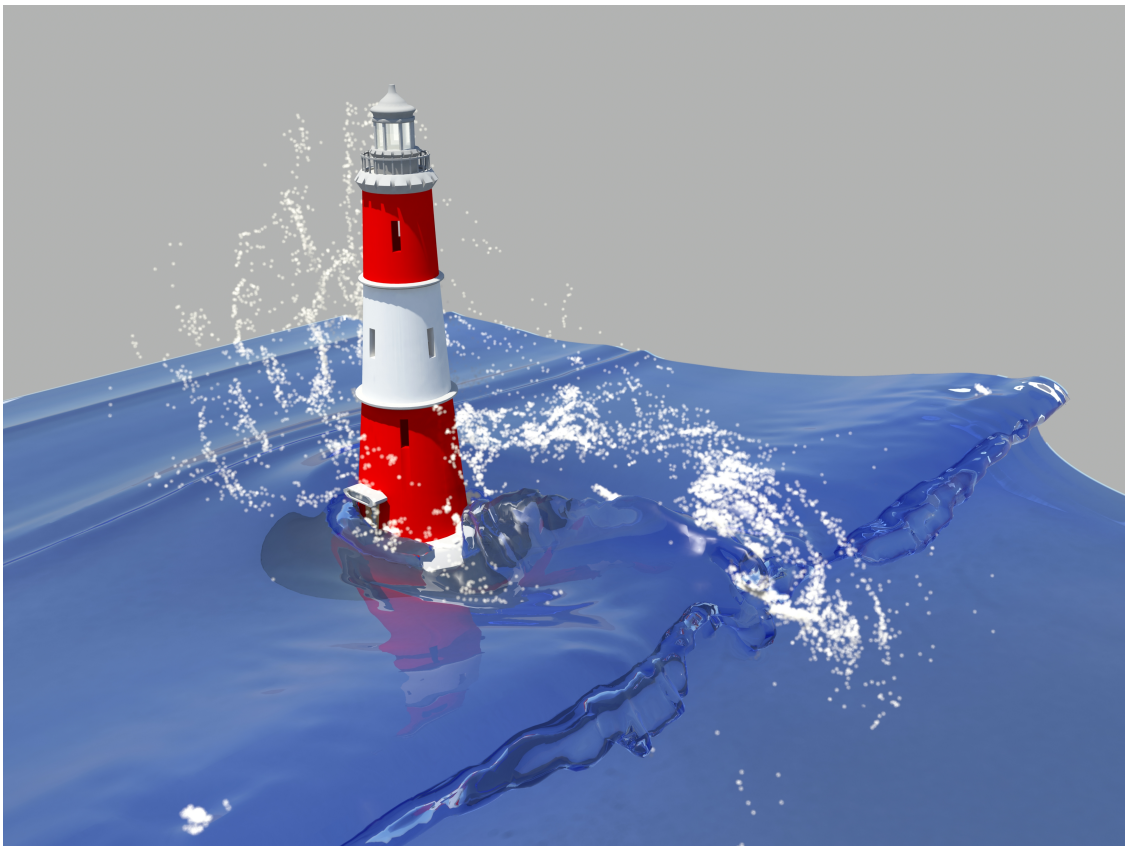

Fig. 16 The water wave hits a lighthouse. Splashes are created based on the implemented particle system (domain size: $60\,m \times 12\,m \times 25\,m$, resolution: $480 \times 96 \times 192$ cells, water surface height: $2\,m$).

3. Croce, R.: Numerische Simulation der Interaktion von inkompressiblen Zweiphasenströmungen mit Starrkörpern in drei Raumdimensionen. PhD thesis (2010)
4. Croce, R., Engel, M., Strybny, J., Thorenz, C.: A parallel 3d free surface Navier-Stokes solver for high performance computing at the german waterways administration. In: The 7th Int. Conf. on Hydroscience and Engineering (ICHE-2006). Philadelphia, USA (2006)
5. Croce, R., Griebel, M., Schweitzer, M.A.: Numerical simulation of bubble and droplet-deformation by a level set approach with surface tension in three dimensions. International Journal for Numerical Methods in Fluids **62**(9), 963–993 (2009). DOI 10.1002/fld.2051
6. Décoret, X., Eisemann, E.: Fast scene voxelization and applications. In: I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games, pp. 71–78. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1111411.1111424
7. Dornseifer, T., Griebel, M., Neunhoeffer, T.: Numerical Simulation in Fluid Dynamics, a Practical Introduction. SIAM, Philadelphia (1998)
8. Enright, D., Fedkiw, R., Ferziger, J., Mitchell, I.: A hybrid particle level set method for improved interface capturing. J. Comput. Phys. **183**, 83–116 (2002). DOI 10.1006/jcph.2002.7166. URL http://portal.acm.org/citation.cfm?id=641282.641285
9. Enright, D., Losasso, F., Fedkiw, R.: A fast and accurate semi-lagrangian particle level set method. Computers and Structures **83**, 479–490 (2003)
10. Fedkiw, R., Stam, J., Jensen, H.W.: Visual simulation of smoke. In: SIGGRAPH 2001: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 15–22. ACM, New York, NY, USA (2001). DOI http://doi.acm.org/10.1145/383259.383260
11. Fedkiw, R.P., Aslam, T., Merriman, B., Osher, S.: A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). Journal of Computational Physics **152**(2), 457–492 (1999). DOI 10.1006/jcph.1999.6236. URL http://dx.doi.org/10.1006/jcph.1999.6236
12. Foster, N., Fedkiw, R.: Practical animation of liquids. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 23–30. ACM, New York, NY, USA (2001). DOI http://doi.acm.org/10.1145/383259.383261
13. Foster, N., Metaxas, D.: Realistic animation of liquids. Graphical models and image processing: GMIP **58**(5), 471–483 (1996). URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.4000
14. Foster, N., Metaxas, D.: Modeling the motion of a hot, turbulent gas. In: SIGGRAPH 1997: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pp. 181–188. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1997). DOI 10.1145/258734.258838. URL http://dx.doi.org/10.1145/258734.258838
15. Gould, D.: Complete Maya Programming: An Extensive Guide to MEL and the C++ API. Elsevier (2003)
16. Griebel, M., Metsch, B., Oeltz, D., Schweitzer, M.A.: Coarse grid classification: A parallel coarsening scheme for algebraic multigrid methods. Numerical Linear Algebra with Applications **13**(2–3), 193–214 (2006)
17. Griebel, M., Zaspel, P.: A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations. Computer Science - Research and Development **25**(1–2), 65–73 (2010). DOI 10.1007/s00450-010-0111-7

18. Hong, J.M., Kim, C.H.: Discontinuous fluids. In: SIGGRAPH 2005: ACM SIGGRAPH 2005 Papers, pp. 915–920. ACM, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1186822.1073283
19. Hong, J.M., Lee, H.Y., Yoon, J.C., Kim, C.H.: Bubbles alive. In: SIGGRAPH 2008: ACM SIGGRAPH 2008 papers, pp. 1–4. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1399504.1360647
20. Kang, M., Fedkiw, R., Liu, X.D.: A boundary condition capturing method for multiphase incompressible flow. J. Sci. Comput. **15**(3), 323–360 (2000)
21. Kaufman, A., Shimony, E.: 3d scan-conversion algorithms for voxel-based graphics. In: SI3D '86: Proceedings of the 1986 workshop on Interactive 3D graphics, pp. 45–75. ACM, New York, NY, USA (1987). DOI http://doi.acm.org/10.1145/319120.319126
22. Lee, H.Y., Hong, J.M., Kim, C.H.: Interchangeable SPH and level set method in multiphase fluids. The Visual Computer **25**(5), 713–718 (2009). URL http://dx.doi.org/10.1007/s00371-009-0339-z
23. Losasso, F., Shinar, T., Selle, A., Fedkiw, R.: Multiple interacting liquids. In: SIGGRAPH 2006: ACM SIGGRAPH 2006 Papers, pp. 812–819. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1179352.1141960
24. Losasso, F., Talton, J.O., Kwatra, N., Fedkiw, R.: Two-way coupled SPH and particle level set fluid simulation. IEEE Transactions on Visualization and Computer Graphics **14**(4), 797–804 (2008). DOI 10.1109/TVCG.2008.37
25. Mihalef, V., Metaxas, D.N., Sussman, M.: Simulation of two-phase flow with sub-scale droplet and bubble effects. Comput. Graph. Forum **28**(2), 229–238 (2009)
26. Monaghan, J.J.: Smoothed particle hydrodynamics. Annual review of astronomy and astrophysics **30**, 543–574 (1992). DOI 10.1146/annurev.aa.30.090192.002551
27. Montani, C., Scateni, R., Scopigno, R.: A modified look-up table for implicit disambiguation of marching cubes. The Visual Computer **10**(6), 353–355 (1994). URL http://www.crs4.it/vic/cgi-bin/bib-page.cgi?id='Montani:1994:MLT'
28. Selle, A., Fedkiw, R., Kim, B., Liu, Y., Rossignac, J.: An unconditionally stable MacCormack method. J. Sci. Comput. **35**(2-3), 350–371 (2008). DOI http://dx.doi.org/10.1007/s10915-007-9166-4
29. Selle, A., Rasmussen, N., Fedkiw, R.: A vortex particle method for smoke, water and explosions. In: SIGGRAPH 2005: ACM SIGGRAPH 2005 Papers, pp. 910–914. ACM, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1186822.1073282
30. Stam, J.: Stable fluids. In: Proceedings of SIGGRAPH 1999, Computer Graphics Proceedings, Annual Conference Series, pp. 121–128 (1999)
31. Sussman, M., Smereka, P., Osher, S.: A level set approach for computing solutions to incompressible two-phase flow. J. Comput. Phys. **114**, 146–159 (1994). DOI 10.1006/jcph.1994.1155. URL http://portal.acm.org/citation.cfm?id=182683.182718
32. Takahashi, T., Fujii, H., Kunimatsu, A., Hiwada, K., Saito, T., Tanaka, K., Ueki, H.: Realistic animation of fluid with splash and foam. Comput. Graph. Forum **22**(3), 391–400 (2003)
33. Thuerey, N.: Fluid simulation with blender. Dr. Dobbs Journal (2006)
34. Verleye, B., Croce, R., Griebel, M., Klitz, M., Lomov, S., Morren, G., Sol, H., Verpoest, I., Roose, D.: Permeability of textile reinforcements: Simulation, influence of shear, validation. Composites science and technology **68**(13), 2804–2810 (2008)