Realistic and Interactive Simulation of Rivers

Peter Kipfer
Havok
peter.kipfer@havok.com

Rüdiger Westermann
Computer Graphics & Visualization
Technische Universität München
westermann@in.tum.de

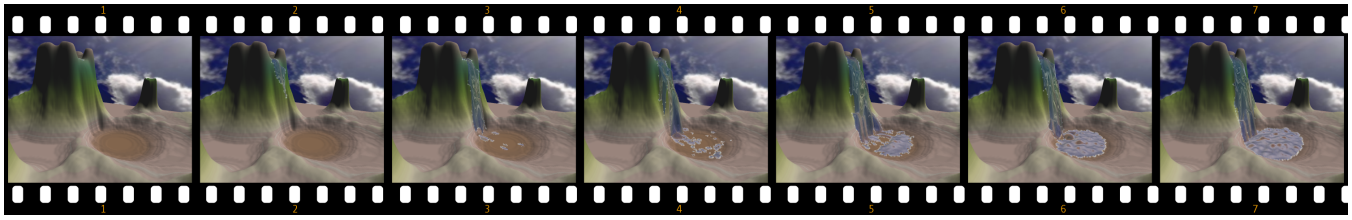

Figure 1: Simulation of a river cascade: Water flowing from a rock and filling the lake below (68 fps using 3000 particles).

ABSTRACT

In this paper we present interactive techniques for physics-based simulation and realistic rendering of rivers using Smoothed Particle Hydrodynamics. We describe the design and implementation of a grid-less data structure to efficiently determine particles in close proximity and to resolve particle collisions. Based on this data structure, an efficient method to extract and display the fluid free surface from elongated particle structures as they are generated in particle based fluid simulation is presented. The proposed method is far faster than the Marching Cubes approach, and it constructs an explicit surface representation that is well suited for rendering. The surface extraction can be implemented on the GPU and only takes a fraction of the simulation time step. It is thus amenable to real-time scenarios like computer games and virtual reality environments.

CR Categories: I.3.5 [Computational Geometry and Object Modeling]: Curve, surface, solid, and object representations—Physically based modeling; I.3.6 [Methodology and Techniques]: Graphics data structures and data types

Keywords: physics-based simulation, river flow, particle system, surface extraction

1 INTRODUCTION

Over the last decade, physics-based, yet interactive simulation and rendering of natural phenomena has been an active research area in computer graphics. For the simulation of fluid flow, besides Eulerian approaches using the Navier-Stokes equations, particle-based methods like Smoothed Particle Hydrodynamics (SPH) have been employed successfully. SPH is a Lagrangian approach for computational fluid dynamics, where the flow is modeled as a collection of particles that move under the influence of hydrodynamic and gravitational forces. While an Eulerian method obtains the solution relative to a fixed grid, in the Lagrangian framework a simulation grid is abandoned and the fluid flow equations are rewritten in terms of the concentration of advected particles.

According to SPH, a scalar quantity A is interpolated at locations $\vec{r}$ by a weighted sum of contributions from all particles:

$$A_S(\vec{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\vec{r} - \vec{r}_j, h)$$

where j iterates over all particles, m_j is the mass of particle j, $\vec{r}_j$ its position, ρ_j the density and A_j the field quantity at $\vec{r}_j$. The function $W(\vec{r}, h)$ is called the smoothing kernel with core radius h.

In this work, we aim at developing interactive methods for the simulation and rendering of water flowing over a height field using SPH. The flowing water can form rivers, or it can cluster into lakes. The water can spring from several sources, and its direction can be influenced both by terrain obstacles and external forces like wind and gravitation. In addition, due to effects like erosion or flooding the water surface may appear localized or it may cover large contiguous parts of the terrain. The mesh-free SPH method provides an effective tool for dealing with these requirements, and it is used in this work as the computational basis for modeling the motion of particles that simulate the flowing water.

To allow for interactive simulation and rendering of particle systems consisting of many thousands of elements, special care has to be taken on the simulation data structure and on the extraction of the renderable water surface. Therefore, we propose a grid-less data structure that efficiently handles neighborhood queries in sparse particle systems for real-time scenarios. In addition, we provide a solution for the efficient extraction of the surface suitable for the rendering of flowing water. Even though we target a sparse particle system to enable interactive simulation over large terrains, the water surface should be closed and it should provide a realistic look.

The paper is organized as follows: In the next section we review previous related work. Then, we give a description of the particle simulation system we employ. In Section 4 we show how the fluid surface is extracted and animated. Section 5 presents and discusses results before we conclude the paper and we give an outlook on future work.

2 PREVIOUS WORK

Lagrangian, particle-based fluid models as introduced to computer graphics by [5] have been extended by stable semi-Lagrangian advection [17] and level set methods to extract the actual free surface

of the liquid [4, 3]. The simulation of fluid-fluid interaction in multiphase fluids and bubbles has recently been studied in [15]. It is based on work by Greenwood et al. [7] and integrates the idea of using multiple fluid particle types and air particles to model realistic liquids.

The fluid model employed in these papers is based on the Smoothed Particle Hydrodynamics (SPH) model [6, 13, 11]. This model is very flexible and can be used for simulating deformable solid objects [1] or even lava [18] by integrating a diffusion equation to solve for the heat transfer in the fluid. We use a variation of the SPH model based on the work of Müller et al. [14] that provides good control at interactive simulation rates.

For the special case of fluids that can be represented by a height field alone, fast solutions exist [9]. They visualize a section of an infinite virtual fluid surface. The fluid surface is expected to cover the entire view and no interactions with solid terrain are considered. Keiser et al. [10] studied surfels for the rendering of the fluid surface, thereby integrating the visualization of fluids and solids as point sampled surfaces. As we will discuss in section 4, point splatting will not produce visually pleasing results for the particle systems we target. The surfels techniques just shift this problem to a finer scale and still require dense particle sets.

3 Particle System

SPH is an interpolation method for particle systems. Although the flow field quantities are only defined at discrete particle positions, SPH allows for the evaluation of these quantities at arbitrary locations in space. In the SPH model, the quantities are distributed in a local neighborhood of each particle using radial basis functions. The complexity therefore depends on the number of particles instead of the number of grid cells as in Eulerian approaches. A detailed discussion of the SPH model for physics-based simulation of fluid effects has been presented in [14].

3.1 Data Structure

To interpolate a quantity at a particular position in space using the SPH model, the set of all particles in close proximity to this position has to be determined. For large and dynamic particle sets, calculations and memory access operations involved in these queries quickly become the bottleneck in a simulation system. As a matter of fact, hierarchical acceleration structures are frequently employed.

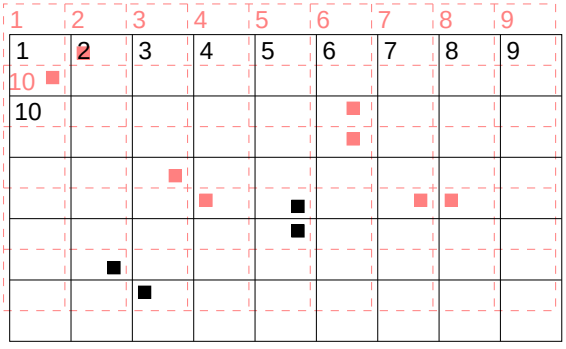

Figure 2: Collision detection in 2D using staggered grids. All interactions are found after linear search in both grids.

To verify the effectiveness of our new data structure to determine particle proximities, we have implemented a reference data structure - the octree-based approach by Vermuri et al. [20]. In this approach, a full octree is utilized to efficiently resolve proximity. References to particle sets are stored at the leaf nodes only. They are updated dynamically with respect to particle motion. The octree resolution is variable. If it is chosen such that the leaves have an edge-length of $2r$, with r being the maximum particle diameter, collision detection becomes linear in the number of particles.

For sparse particle sets, however, the cost for updating the data structure becomes the limiting factor. To continually allow for the efficient determination of non-empty leafs, in such a scenario every single particle movement causes the update of the entire hierarchy. Therefore, although providing linear cost collision detection, the constant factor of the algorithm is very large.

We propose a different collision detection data structure that offers a considerable speedup for sparse particle systems. The structure has a minimal memory footprint and remains static at runtime: We use a simple linear list that stores the particles. A virtual regular grid, that is identical to the leaf-level of the octree, now is used to associate a spatial bin with each particle. The bin indices (i_x, i_y, i_z) are combined to build a single unique 64 Bit identifier per bin $id = |0|i_z|i_y|i_x|$. We now sort the particle list according to this identifier. To determine the potential collision partners of a particle, we only need to check the left neighbor entries of the particle until we encounter an identifier $id \leq (id_{self} - 2)$ which signals that we have left the one-neighborhood in space. If we start the collision detection with the first particle entry in the sorted list, we are guaranteed to process each interaction exactly once. However, because of the structure of the identifier, this only works for neighbors in x-direction. Figure 2 shows the situation in 2D. We thus need a second linear list that is associated with a staggered grid to resolve neighborhood in y-direction. In order not to detect a collision found in the first grid a second time, for each collision resolved, the leading Bit of the identifier is set to 1 to mark particles for processing the successive list. After processing both lists in Figure 2, we have detected all possible collisions exactly once. In principle, we need an additional list (and associated staggered grid) for each spatial dimension that we have lost by linear searching. Our implementation of the collision system therefore uses a third list for the z-direction.

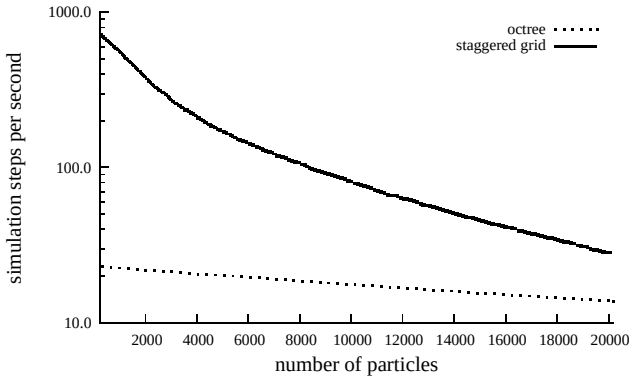

Figure 3: Performance of the octree and the staggered grid update driving the collision resolution algorithm.

Using the staggered grid approach, we have exchanged the octree update and traversal for a simple lookup in 3 linear lists. Of course, all three lists need to be sorted for each simulation step. We use the STL `std::sort` algorithm, that has a complexity of $O(n \log n)$. As there is a constant maximum number of particles per spatial bin, the list traversal is linear $O(3n)$. Both complexities add up. This clearly is more than $O(n)$ as proposed for the octree approach of Vemuri et al. [20], but as Figure 3 shows, for the number of particles we target in our system, the staggered grid approach introduces a much lower constant complexity. Especially for game applications, where a rather low number of particles is mandated by

multiple restrictions, the staggered grid outperforms the octree by more than an order of magnitude. Furthermore, the staggered grid is insensitive to spatial particle density and thus supports the sparse particle systems we employ. The performance profile in Figure 3 has been recorded while simulating the flow within the Hersbruck terrain dataset (see Figure 13) with carpet generation switched off.

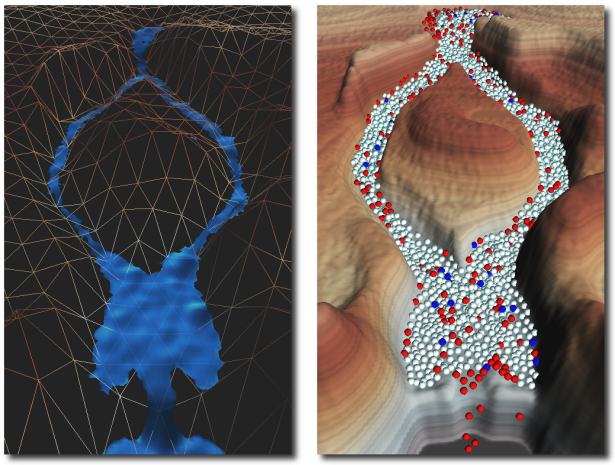

Figure 4: Only visible parts of the carpet are constructed (left image). Particle-terrain collisions (red) and frozen particles (blue) are treated special in our system (right image).

To resolve particle-particle collisions, we use a similar approach as Müller et al. [14] and compute a combination of an elastic and perfect inelastic collision [19]. For resolving the particle-solid collisions, we need a more precise approach than Müller et al. because the simple approach they use produces irregular motion artifacts, especially when dealing with particles of different radii. Because we are using sparse particle systems, these motion artifacts are very noticeable. We actually would need to compute the correct time of first contact between a moving sphere and triangles. A full implementation of this algorithm [2] however proved to be too expensive. We therefore use an iterative backtracking of the motion path using the bisection method to find a satisfactory contact position. At this point, the particle is bounced taking into account friction and other surface parameters. The particle path then is recomputed such that the particle travels the correct length for the given time step by considering the speed before and after the collision.

We further optimize the collision resolution, by introducing "frozen" particles. These particles do not move at all or have a motion vector that is too small to produce a particle-solid collision in the next simulation time step. They can therefore be excluded from the expensive particle-terrain collision test. The particle-particle collision test is performed all the time. Thus the frozen status is reset, if the particle gains speed or is hit by another particle. In Figure 4, frozen particles are colored blue. Red particles experienced a terrain collision. The frozen particle optimization is most valuable when particles start to stack, for example when filling a lake or some barrier is blocking the river. The efficiency of this optimization depends on the local speed and viscosity of the flow, the precision of the integrator, the particle size and terrain features. Therefore, we found it hard to give a consistent speedup factor for this optimization.

3.2 Simulation

We have implemented multiple integration schemes for the SPH equations. We found the embedded Runge-Kutta 3(2) scheme to be the best trade-off between speed and accuracy for the semi-Lagrangian method in the velocity advection step [8] to satisfy the Courant-Friedrichs-Lewy condition [1]. For interactive display, however, we need the particles to move at the same time increment and thus we use the second order Runge-Kutta scheme with fixed time step size for our simulations.

For the simulation of water flows over large terrains we do not enforce tight packing of the particles and we allow varying particle radii. The simulation and collision detection routines automatically handle this or can be easily modified. By specifying the minimum and maximum particle radius along with the emission rate, the user can therefore easily model a natural spring source emission. For our purposes, the surface extraction step is not straight forward. The relatively sparse particle distribution requires additional work to produce a closed surface.

4 Surface Construction

In previous SPH techniques for the simulation of fluid phenomena either point splatting, surface extraction using the Marching Cubes algorithm [12] or ray tracing of the implicit level set surface defined by the particle distribution have been utilized to visualize the fluid surface.

In point splatting, the particles used to model the flow are directly mapped onto renderable point primitives. While highly efficient, this approach can only provide the visual impression of a surface if the particle density is high enough. Thus, it is appropriate for the visualization of the surface of rather compact particle clusters, but it can not easily be applied in the current scenario to visualize rivers. Rivers tend to become very elongated, and they usually consist of only a single particle layer above the terrain and a rather sparse particle distribution within it. Splatting therefore would produce many holes in the water surface. Enlarged rendering primitives, on the other hand, do not solve the problem as they give a false impression of the water height above the terrain. Oriented point sprites (surfels) do not solve the problem but shift it to a finer scale. Finally, because there is no surface connectivity associated with the sprites, standard shading techniques may not be easily portable.

The marching cubes approach reconstructs the surface geometry from the particle distribution by sampling a distance field on a 3D grid. As a consequence, sampling artifacts may arise, which is usually avoided by sampling the field at the highest rate necessary to reconstruct all details. Then, however, the reconstruction process generates a highly tessellated surface, increasing computational and rendering load. As a consequence thereof, although many acceleration techniques exist the algorithm is considerably more expensive than simple point splatting. In the literature [14], a performance loss of about a factor of 4 is reported for the marching cubes approach compared to point splatting. To avoid the memory overhead that is introduced by sampling the entire simulation domain, it is typically restricted to a sub-area of the domain [4, 14, 15].

The resulting triangles can finally be rendered using graphics hardware. Due to the highly dynamic nature of the water surface, however, the surface has to be rebuilt in every simulation frame and does therefore not accommodate acceleration strategies like OpenGL display lists or GPU vertex arrays. To avoid the tessellation, the distance field can directly be traced along the rays of sight. This however is numerically even more complex and can also suffer from sampling artifacts because of the integration step size.

The mentioned techniques all share the common problem, that they approximate a level set surface by particle primitives. Because the particles usually define a spherical area of influence, the resulting surface always looks too blobby and gives the impression of an overly viscous fluid (see Figure 5 for a comparison to our method). Small scale effects like surface tension and capillary action are not reproduced appropriately. Recently, Wang et al. [21]

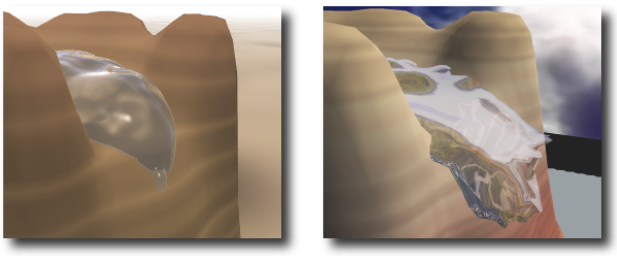

Figure 5: Left: traditional rendering of the implicit water surface using ray tracing. Right: real-time display of the water surface using our method.

described a complex system for handling those effects accurately at the expense of very long simulation times. For the application we target, small scale effects do not contribute to the simulation, but we aim to produce deformed drops as well as connected water surfaces at interactive rates. For this purpose, we propose the "carpet method": we build a surface representation that covers all particles and is supported by them against gravity. The carpet should only have the spatial extent of the particle cloud to allow efficient rendering.

4.1 Carpet Construction

The carpet can be thought of as a regular mesh that covers the entire domain. Two different approaches have been implemented to efficiently construct this mesh. In the first approach, the mesh only virtually exists, and it is constructed and rendered only at those positions where particles are present (see left image in Figure 6). A quadtree data structure is used to identify these parts efficiently:

① Particles are stored in the quadtree leaves. As it will be described below, this step is actually folded into the particle advection step.

② For each node, the maximum height values are pulled up the tree. These values define the virtual carpet surface.

③ Visible parts of the carpet surface are rendered by traversing the tree depending on whether the height of the fluid is above the height of the terrain.

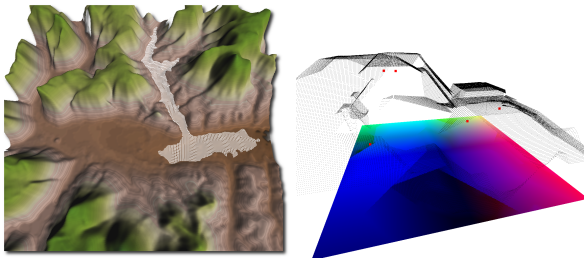

Figure 6: Our first carpet construction approach builds the surface only where particles are positioned (left image). The GPU construction approach splats the particles to a depth image of the carpet vertices (right image).

Let the carpet be initially flat and located below any terrain height value. The carpet consists of $n \times n$ virtual quads. This is the resolution of the carpets' surface during rendering. We now instantiate a full quadtree on these quads. The nodes of the tree store the absolute minimum terrain height in this domain, the current absolute carpet height and a velocity value. Then, we insert all particles into the tree leaves by storing their absolute height in the corresponding spatial bin using a **max** operation, and we thereby push the carpet to the level of the topmost particle. The velocity of this carpet part is set to zero. This is a very simple operation and has linear time complexity and zero memory footprint. Next, the inner nodes of the quadtree are updated by propagating the height values from bottom to top in log(height) steps. Because the height field of the terrain can be pre-processed, dynamic terrain changes can also be taken into account.

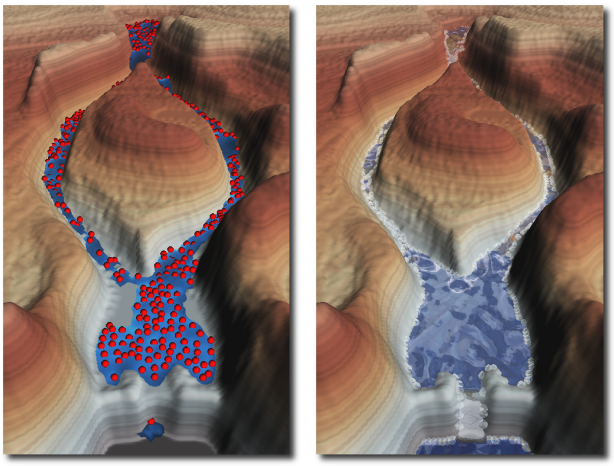

Figure 7: The carpet provides a closed water surface even for sparse particle systems (left image). The implicit classification of the quadtree node allows to render the carpet border differently (right image).

We can now render the carpet efficiently by traversing the quadtree. If we find a height value below the height of the terrain at an inner node, the recursion is aborted and nothing is drawn. If we reach a leaf node, we can be sure that it belongs to a visible part of the carpet. The corresponding quad is then rendered. Afterwards, the carpet velocity is accelerated by gravity and the carpet position is displaced downwards accordingly. If the supporting particle has left this part of the carpet in the upcoming frame, it will fall down until it reaches below the terrain surface. There, it will be excluded from the rendering step because of the quadtree traversal. If the supporting particle is still there, the quadtree pull in the second step of the algorithm will restore the correct height.

The second approach for creating the carpet constructs the rendering geometry entirely on the GPU (see right image in Figure 6). We store the carpet vertices of the entire domain in two vertex array buffers and update them as follows:

① Drawing into the second buffer, fetch the vertices from the first buffer and accelerate them downwards according to gravity. Store height as z-value.

② Upload new particle positions and splat all particles on top of the displaced surface with z-test turned on.

③ Draw the target buffer as triangle strip array and exchange the two buffers for the next iteration step.

Because the first pass has set the current carpet height in the z-buffer, the particle splats in the second pass stack on top each other correctly and therefore push the carpet vertices to the topmost particle. If the supporting particle has left this part of the carpet

in the upcoming frame, the first pass will make it fall down until it reaches below the terrain surface. When the carpet is drawn in the third pass, these triangles will be efficiently culled by the early z-test.

For creating smooth particle shapes, we take advantage of the second pass to splat point sprites. The sprites can be textured with an arbitrary depth image. We provide several depth shapes that are additionally oriented by the splatting shader according to the direction of particle movement. This approach for constructing the carpet geometry therefore is implemented entirely on the GPU and thus minimizes the bus transfer to the upload of the particle positions.

4.2 Carpet Update

The carpet can be updated incrementally. The spatial data structure used in the SPH model and for collision resolution delivers the necessary information. If we project its leaves down in y-direction, we collapse each stack into a quad containing all particles in this part of the terrain (see Figure 8). The projected mesh is now used as the dual of the virtual carpet quadtree mesh. Moving a particle "below the carpet" therefore is integrated into the particle advection and collision resolution step by simply dropping the y-coordinate to determine the correct carpet bin. Because the quadtree is the dual of the spatial tree, a particle in a spatial bin will exactly support one carpet vertex.

4.3 Carpet Optimization and Rendering

In order to construct visually pleasant carpets for larger terrains, one would need to have a very large virtual quad surface and its corresponding quadtree. The storage requirements for the full quadtree would be bad as will be the traversal performance. However, for adequate collision detection as well as for a nice looking carpet surface, a good spatial resolution is mandatory. We therefore allow the quadtree leaves of the carpet to contain a whole set of spatial bins. Note that the hierarchical pull operation to build the quadtree is not affected by this.

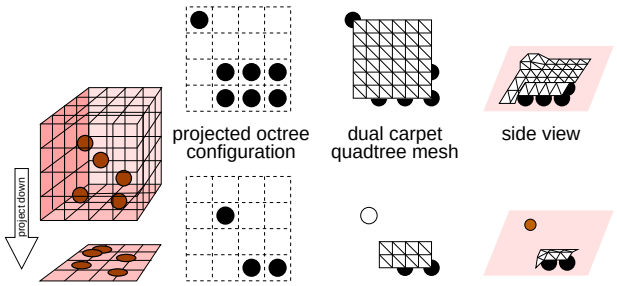

Figure 8: Coupling of simulation space and carpet quadtree structure and two example configurations for particle distributions within a quadtree leaf node.

When rendering a carpet quadtree leaf built by the first construction approach, we now have additional possibilities for the local shape of the quad. We can build a smoothed surface that minimizes the local curvature to account for surface tension or identify isolated particles as single drops to simulate spray. Figure 8 shows two example configurations. They are precomputed and stored in a lookup table. Note that the produced partial surface does not need to be connected. Disconnected particles can even be rendered using different primitives. In the lower row of Figure 8, we could choose to render the isolated particle as a point sprite textured with foam. This provides an implicit classification of the particles and gives subtle improvements of the carpet boundary (see Figures 7 and 12).

For a realistic behavior, it is only mandatory for the surface borders that are not supported by particles to reach below the terrain level. As we know the local absolute height and particle radius, this is easy to compute for the first construction approach without looking it up in the terrain map. The second carpet construction approach automatically accounts for this as it always draws the carpet on the entire domain. The carpet method differs from the surfels approach, in that the construction of a partial surface is not tied to a particle. In contrast to [10], we therefore do not need to carry pointers from surfels to particles and vice versa. The GPU carpet construction cannot change rendering primitives when drawing the carpet, but it offers a considerable performance gain. When using 20000 particles, the carpet can be constructed 55 times per second using the GPU method as opposed to 14 times using the more flexible CPU method.

Note that examining the particle neighborhood for one of the configurations is not limited to the inside of a carpet quadtree leaf. Because of the regular structure, neighbors can be accessed quickly. We can therefore also choose to match the configuration patterns using multiple quadtree leaves instead of spatial bins. The choice of the configuration implicitly classifies the particles.

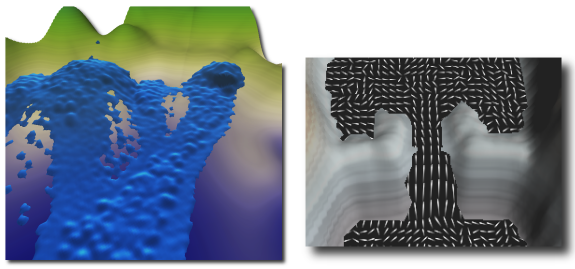

Figure 9: The carpet method allows to display elongated drops of water running over a hill (left image). A GPU shader calculates the local surface velocity and renders oriented streaks (right image).

Because a carpet vertex smoothly falls down if the supporting particle moves away, rapid changes of the carpet occur only if a particle flows into this region. This is consistent with the observed behavior of water and leads to elongated traces of water drops if the particle moves sufficiently fast (see Figure 9). This allows us to run the simulation more efficiently using considerably less particles. We can also show surface waves by using a procedural gravity force component. For the analysis of the flows, we have implemented a GPU shader to visualize the primary movement direction and speed of the underlying particles using a variation of the Oriented Line Integral Convolution method [22].

The carpet method efficiently extracts the top flow surface. For visualization of rivers this is ideal. The method reaches its limits only with phenomena like waterfalls or fountains but remains applicable in non-extreme cases. Figure 12 shows water falling down from a rock and filling a small lake below. The carpet provides a visually pleasant hull for the cascade. The carpet method however is not volume conserving which poses a potential problem during the filling of the lake. However this effect is totally dominated by the river flow motion and is not noticeable. When using the GPU construction method, additional care has to be taken when drawing the resulting carpet, as the carpet resolution and the terrain resolution need not match. Very low parts of the carpet therefore can disappear temporally below the terrain because of z-fighting. The shader used for rendering the carpet can suppress this effect by comparing carpet and terrain height and adding a small offset to the carpet in relevant areas.

5 Results

In the following, we present performance results for two typical applications of our particle system. The experiments were run on a machine equipped with AMD Athlon64 2.2 GHz processor and *n*VIDIA GeForce 6800 GT graphics card with 256 MByte video memory. Previous work by Müller et al. showed interactive rendering of SPH models at 5 fps for 3000 particles [14]. For this number of particles, we obtain a speedup of 13.6 using the GPU carpet (see figure 12).

The first example demonstrates the capability of the carpet to provide a convincing river simulation even on large and complex terrains. In Figure 11, we placed a source in the upper right corner in the bed of the Colorado river in the Grand Canyon dataset. The flow reached all the way down to the lake in the lower left. In order to provide an interactive display, we use a rather sparse particle system. The carpet nevertheless faithfully fills the canyon. The simulation uses up to 8000 particles but maintains a performance of 26 fps when all particles are released.

In the second application, the carpet is used for interactive visualization of a flooding simulation. In Figure 13, the dam of the small hydro-electric power plant in the foreground is assumed to be broken. Apart from the water level in the local housings, it is interesting to see whether the flood reaches the larger city of Hersbruck in the center of the simulation region and which infrastructure elements get blocked. The dataset has 25 meters ground resolution and 1 meter height resolution. The challenge for the simulation here is that there is only a very small slope for the water to run into the larger valley. We therefore had to use many very small particles and integration step sizes for a good simulation. The more expensive particle-terrain collision routine we are using allows for this. Despite the large number of particles, the carpet method provides an interactive visualization. It consumes less than 10% of the frame time. The simulation uses up to 20000 particles and maintains a performance of 12.7 fps when all particles are released. As Figure 13 shows, the flood blocks a major road running through the valley, then it flows into the river in the middle of the larger valley without reaching the city.

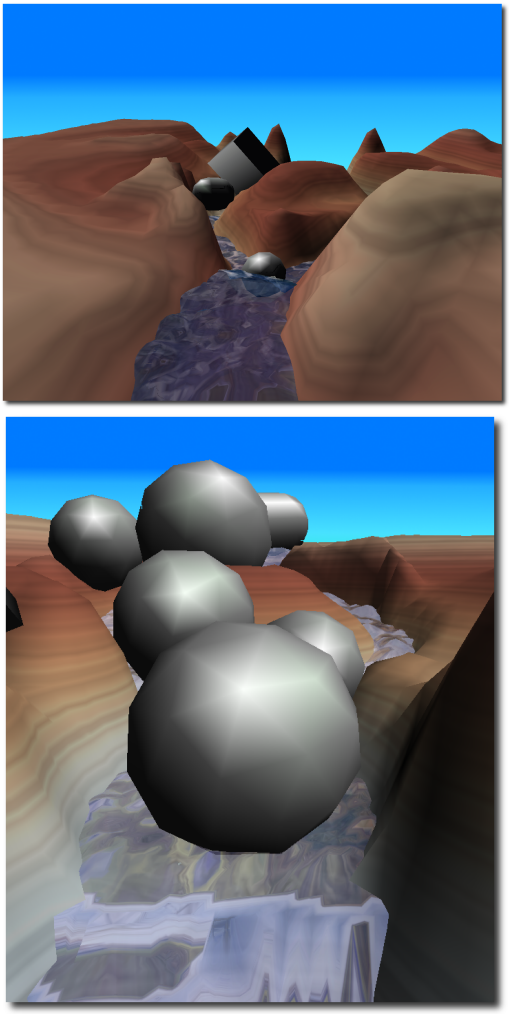

Figure 10: Coupling SPH and rigid body simulation allows for adding blocking obstacles to the river.

6 Conclusion

We have presented a grid-less acceleration structure and two fast surface construction methods for SPH-style particle systems that are especially suited for the display of water flowing in rivers. The grid-less technique uses sorted lists obtained from staggered grids to resolve neighbor queries for the particle systems we target more than an order of magnitude faster than hierarchical octree-based approaches. The carpet structure does not require a dense particle set to produce a closed surface like it is needed for point splatting approaches. It offers an explicit surface description that can be constructed incrementally at much faster rates than using the marching cubes approach. Because the carpet construction is very efficient, one can afford to visualize every simulation time step which results in an interactive tool for particle-based flow simulations.

The GPU-based carpet construction method has been evaluated on a complex terrain where the sparse particle system employed provides interactive simulation frame rates despite the expensive particle-terrain collision resolution. In a second experiment, we used a massive particle set for accurate environmental simulation. The carpet method allows us to visualize both the sparse and the massive particle system very efficiently. Because we extract an explicit surface representation, rendering effects using GPU shaders are straight forward to apply [16].

6.1 Future Work

We are about to integrate rigid body simulation into our system to allow for simulating obstacles blocking the river. The SPH model and the rigid bodies influence each other, so the objects in figure 10 will get washed away by the flow as soon as enough water has accumulated behind it.

We would like to extend our system to support surfaces with dynamic anisotropic properties to model small scale effects in the spirit of [21]. As the carpet is only constructed in regions where particles are, we can easily render it into an offscreen texture using additive blending to create a dynamic “wet-map” that can be taken into account when integrating particle positions. An example would be the modulation of surface friction. The carpet coverage can also be helpful to trigger erosion effects.

Acknowledgments

The Hersbruck dataset appears by courtesy of Bayerisches Landesvermessungsamt München. The Grand Canyon dataset is from the Large Geometric Models Archive at Georgia Institute of Technology.

APPENDIX A

This is the GLSL code of the splatting shader for the GPU carpet construction.

```
uniform sampler2D splatTex;
uniform vec4 remapFragPos;


void main(void)
{
 // calculate splat height
 float splat = texture2D(splatTex,gl_TexCoord[0].xy).x;
 float depth = gl_TexCoord[1].y + splat;

 // calculate object-space vertex position and
 // draw to depth image
 gl_FragColor.xz = (gl_FragCoord.xy-remapFragPos.xy) *
                   remapFragPos.zw;
 gl_FragColor.y = depth;
 gl_FragColor.w = 1.0;
 gl_FragDepth = depth;
}
```

REFERENCES

[1] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: a new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 61–76, New York, NY, USA, 1996. Springer-Verlag New York, Inc.

[2] David H. Eberly. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*. Morgan Kaufmann, 2000.

[3] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744, New York, NY, USA, 2002. ACM Press.

[4] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM Press.

[5] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[6] R. Gingold and J. Monaghan. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–398, 1977.

[7] S. T. Greenwood and D. H. House. Better with bubbles: enhancing the visual realism of simulated fluid. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 287–296, New York, NY, USA, 2004. ACM Press.

[8] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerische Simulation in der Strömungsmechanik*. Vieweg Verlag, 1995.

[9] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive animation of ocean waves. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 161–166, New York, NY, USA, 2002. ACM Press.

[10] Richard Keiser, Bart Adams, Dominique Gasser, Paolo Bazzi, Philip Dutré, and Markus Gross. A unified lagrangian approach to solid-fluid animation. In *Eurographics Symposium on Point-Based Graphics (2005)*, pages 125–133, 2005.

[11] M.B. Liu and G.R. Liu. *Smoothed Particle Hydrodynamics: A Meshfree Particle Method*. World Scientific Publishing, 2003.

[12] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (SIGGRAPH 87 Proceedings)*, pages 163–169, 1987.

[13] J. Monaghan. Smoothed particle hydrodynamics. *Annual Reviews of Astronomy & Astrophysics*, 30:543–574, 1992.

[14] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[15] Matthias Müller, Barbara Solenthaler, Richard Keiser, and Markus Gross. Particle-based fluid-fluid interaction. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 237–244, New York, NY, USA, 2005. ACM Press.

[16] Simon Premoze and Michael Ashikhmin. Rendering natural waters. *Computer Graphics Forum*, 20(4):189–200, 2001.

[17] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[18] Dan Stora, Pierre-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret, and Jean-Dominique Gascuel. Animating lava flows. In *Graphics Interface (GI'99) Proceedings*, pages 203–210, June 1999.

[19] Paul A. Tipler. *Physics For Scientists and Engineers, Vol. 1: Mechanics, Oscillations and Waves, Thermodynamics*. W. H. Freeman, fourth edition, 1999.

[20] B. C. Vemuri, Y. Cao, and L. Chen. Fast collision detection algorithms with applications to particle flow. *Computer Graphics Forum*, 17(2):121–134, 6 1998.

[21] Huamin Wang, Peter J. Mucha, and Greg Turk. Water drops on surfaces. *ACM Trans. Graph.*, 24(3):921–929, 2005.

[22] Rainer Wegenkittl, E. Gröller, and Werner Purgathofer. Animating flow fields: Rendering of oriented line integral convolution. In *CA '97: Proceedings of the Computer Animation*, page 15, Washington, DC, USA, 1997. IEEE Computer Society.

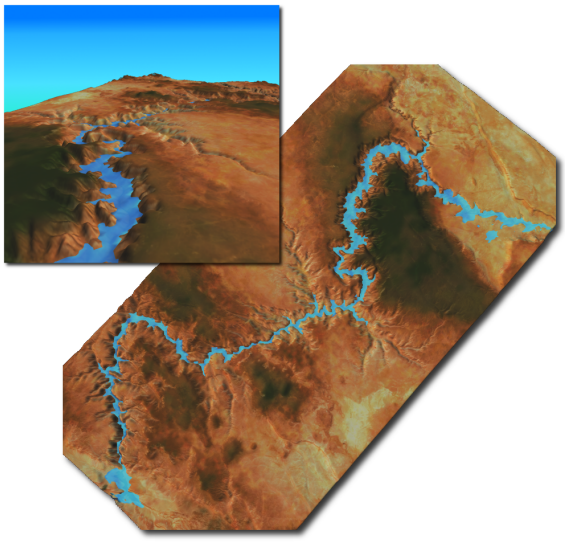

Figure 11: The carpet method on large complex terrain. Even though the particle system is sparse, the carpet provides a closed water surface (particle size exaggerated compared to real world scale). We use 8000 particles at 26 fps.

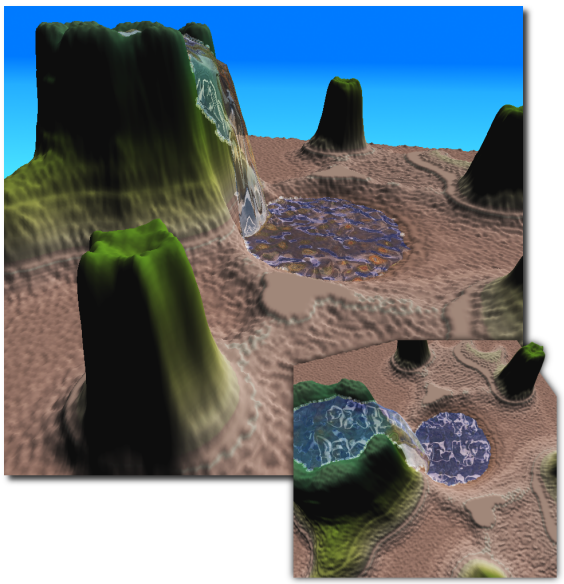

Figure 12: The carpet method used for visualizing a cascade. The method gives sensible results even for this extreme case. (68 fps using 3000 particles)

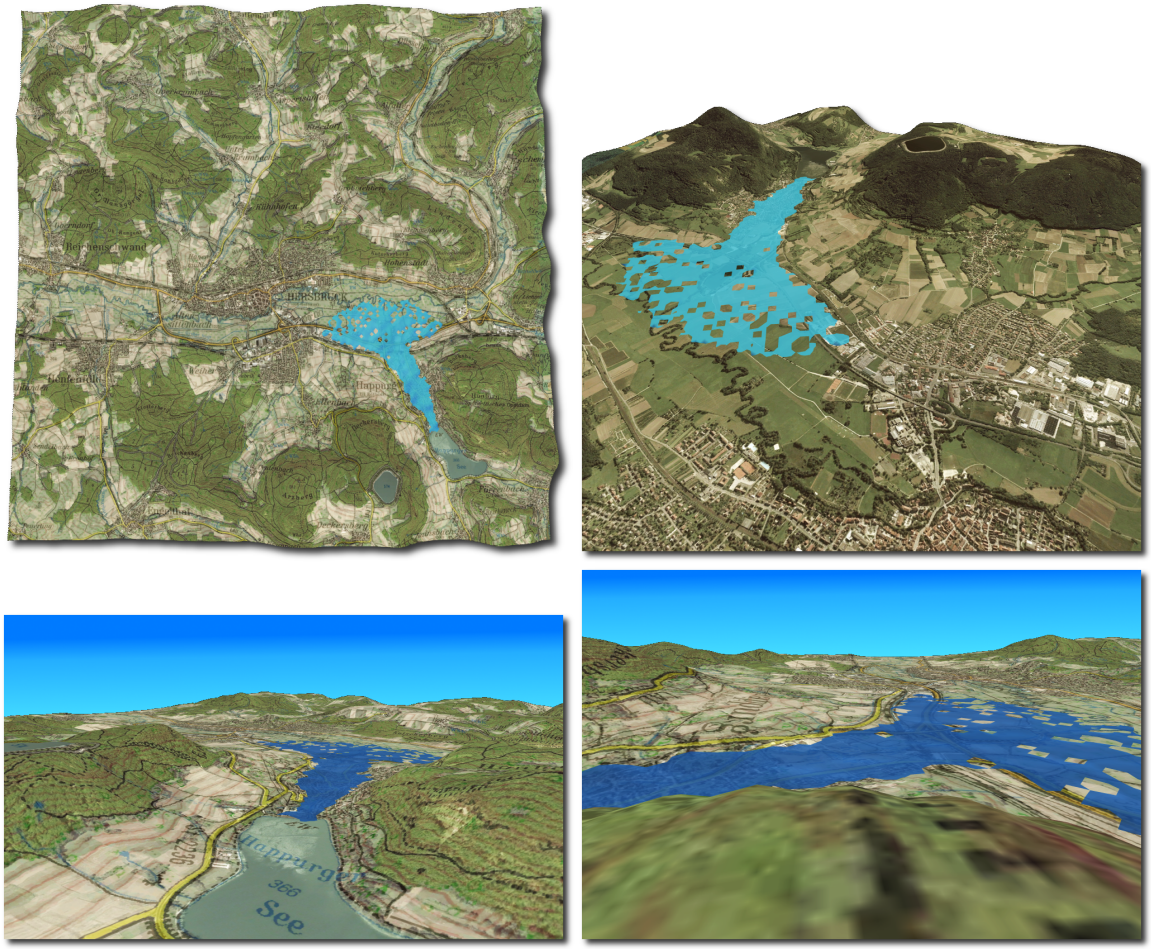

Figure 13: Flooding simulation after an assumed dam breach. The simulation uses 20000 particles and a 4:1 mapping of spatial nodes to the carpet quadtree. Performance: 12.7 fps.