Distributed VFX Architecture for SPH Simulation

Dieter Morgenroth
HDM Stuttgart
Nobelstraße 10
D 70569 Stuttgart

Daniel Weiskopf
University of Stuttgart
Visualisierungsinstitut
Allmandring 19
D 70569 Stuttgart

Bernhard Eberhardt
HDM Stuttgart
Nobelstraße 10
D 70569 Stuttgart

ABSTRACT

We propose an approach to simulating and rendering physically based fluid effects in a VFX production environment using a client/server architecture that is practical for distributed simulation resources and that can be seamlessly integrated into commercial 3D animation packages. The fluid simulation implements smoothed particle hydrodynamics (SPH). We extend the concept of surface particles by introducing blind particles that facilitate efficient direct raytracing of isosurfaces. We evaluate the performance of our approach with local simulation on CPUs and GPUs and distributed GPU simulation. We demonstrate the integration into the animation package 3ds Max and the VRay raytracer. The usability of the VFX production pipeline is assessed by a user study with VFX professionals and animation experts.

Keywords

Smoothed particle hydrodynamics, VFX pipeline, direct raytracing of implicit surfaces

1 INTRODUCTION

An important, however expensive and time-consuming part in CGI/VFX production is physically based simulation. There are many different and competing approaches, with respective advantages and disadvantages. In addition, to experiment with new VFX ideas, the integration of research results into VFX pipelines of production houses is a continuing challenge because the production pipeline adds requirements to the system that may turn good theory into complicated implementation. Many leading VFX production houses like ILM or CA Scanline use in-house solutions for physical simulation. A few specialized software vendors such as Next Limit Technologies offer solutions like Realflow [1] that couple to established animation packages. Recently, software packages that started as in-house solutions have been emerging on the market, like the fluid simulation software Naiad from Autodesk.

All current solutions, including the above, are either directly integrated into the animation package as extensions like Glu3D [2] from 3DAliens for Autodesk's 3ds Max, where the simulation can be run directly in the base package, or the simulation is run in an external application like Realflow and the results are then imported into the animation package as baked simulation data. A good example of a typical pipeline integration in a VFX environment is described by Lagergren [3], whose fluid simulation with a GPU implementation targets the production renderer in the SideFx Houdini system.

Here, we propose a client/server architecture for distributed VFX simulation that can be seamlessly used within an existing VFX pipeline. Similar to other system papers like the one by Parker et al. [4], we describe the design choices that have to be made for the integration of recent research advances into a production pipeline.

The proposed design pattern of this paper is applicable to many areas in the field of simulation. We have chosen fluid simulation as a concrete example because of its high relevance for special effects and since it also affects the rendering aspect of the production pipeline, thus demonstrating how we can address the integration of software components in all relevant stages of the pipeline.

We have designed a client/server fluid simulation system that allows artists to interactively setup simulations and change parameters inside their familiar animation package while the system executes the numerical calculation for the simulation on a remote system with specialized hardware.

We have chosen a particle-based method for fluid simulation because all major 3D packages have built-in particle systems that can handle simulated point data. The coupling to existing particle systems has the advantage that a variety of tools are already available to further

use of the simulation data, for example, for triggering the creation of splash particles.

Our second contribution is an enhanced direct raytracing technique for isosurfaces that extends the concept of surface particles by introducing blind particles. This reduces the amount of particles to be evaluated for rendering and the network bandwidth required for data exchange, which is the main bottleneck in client/server systems. As a side effect, this also speeds up direct raytracing methods since the rays can skip empty space inside the fluid.

As a case study, we have implemented our approach into the 3D package 3ds Max and used external computing capacity based on NVidia GPUs for the server side. With a user study with VFX professionals, we have evaluated the workflow in a production environment.

2 PHYSICAL BACKGROUND

For this paper, we use smoothed particle hydrodynamics (SPH) [5] for fluid simulation. A good overview of particle-based simulation methods can be found elsewhere [6].

Our implementation uses the kernel functions for density, viscosity, and pressure from Mueller et al. [7]. It employs constant particle size, but adaptive methods could be included as well [8, 9].

For tension forces, we apply the approach by Becker and Teschner [10] using cohesion forces. Integration of the dynamics of the particle system employs explicit first-order Euler integration because it delivers sufficient numerical quality at high computational speed. However, higher-order schemes could be easily used instead, if necessary. To achieve incompressibility, we use predictive-corrective incompressible SPH (PCISPH) [11].

3 SYSTEM ARCHITECTURE

Our focus is good usability of efficient fluid simulation in a VFX environment. To overcome the computational bottleneck of simulating a huge amount of particles on the workstation itself, we outsource the computing power to an external GPU server. Still, the results of the simulation steps should be visible as soon as possible and steering the simulation parameters should reside in the 3D scene in the main package.

To achieve this level of interactivity, a client/server architecture is used; see Figure 1. The client resides in the commercial 3D package and sends simulation parameters to the server while requesting simulation results for specific frames. Decoupling of simulation computation and scene management has the benefit that several users can share specialized hardware. Recently,

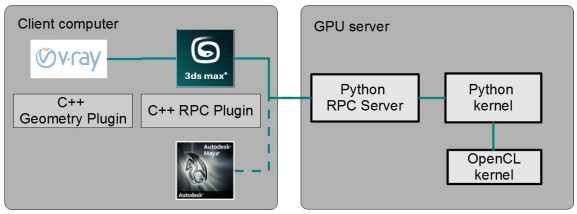

Figure 1: Architecture overview: outsourcing simulation tasks to remote hosts.

GPU blades like the Tesla workstations became affordable for smaller studios. Outsourcing simulation work to these specialized servers is a logical consequence.

The server runs on an external hardware and identifies the user and scene. The user can change parameters for the scene in the 3D package and then start a simulation for a specific frame range. The server will run the simulation in a background thread and store the results in a local disk cache. In the host application, the simulation is an operator in the particle simulation framework. The operator holds the simulation parameters and the current state of the particles as input parameters for the simulation and receives the particles over the network. We compute the SPH-related forces in the simulation server and allow the host application to add forces from the integrated particle system and then do one integration step for all forces. This opens the way for a variety of effects. With this seamless integration, for example, the interaction of the SPH simulation with forces coming from an inverse kinematics (IK) skeleton that is driven with motion capture data is possible.

In our current implementation, we integrated the client into 3ds Max. However, other host systems such as Autodesk Maya or SideFx Houdini would work equally well. For maximum flexibility, we integrated the client into the Particle Flow particle system as an operator using the Particle Flow SDK.

The communication between the client and the server is implemented via remote procedure calls (RPCs). The main problem in implementing a remote SPH fluid simulation is the huge amount of data to be transferred. A typical SPH-VFX shot uses several million SPH particles. The simulation data has to be stored at least for every frame. If the host system needs subframe precision to calculate secondary effects like e.g. additional spray particles, the data has to be saved even more frequently. This computed data has to be stored and streamed over the network. Hence, hard disk space and bandwidth rapidly become the critical factors and need careful decision making regarding file format and message protocol.

As network message protocol, we chose the msgpack [12] protocol, which is a bandwidth-efficient protocol for binary data for which implementations in C++ and Python exist. In addition, it allows

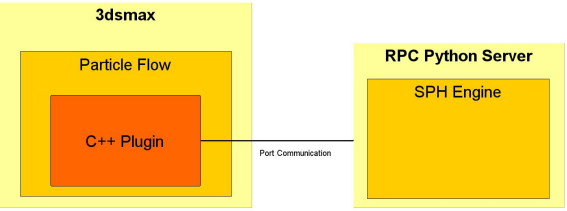

Figure 2: Remote procedure call architecture.

compressed communication with little overhead. The integration with 3ds Max was done using its event-driven particle system Particle Flow. Particle Flow can display all particle emitters and operators in a node-based editor visualizing a graph of nodes. Several data channels can flow through this graph. With the Particle Flow SDK, new channels and operators can be added to the system using C++. Figure 2 illustrates the remote procedure call architecture.

For our implementation, we added a new operator that holds the msgpack-rpc C++ client. This operator then sends remote procedure calls to the server. The client can request the particle count for a specific time. Our system exchanges position, velocity, and color for each particle. The color value will be written to the color channel in 3ds Max. We use it to send the particle type and particle density. We use float32 values for each array element. This leads to memory requirements of $(9 \times 4) = 36$ bytes per particle, which comes to about 3.6 MB per 100 k particles. This can be compressed to about 60 percent using zip compression. The operator unpacks the data and then sets both the particle count of the current Particle Flow graph and the positions and information about the particle type.

All performance-critical modules of the simulation were written as parallel code in OpenCL to run on both multi-CPU and GPU setups.

4 SIMULATION

The Particle Flow framework employs an event-driven model, based on the concept of events and operators. Operators describe and modify particle properties such as speed and direction over a period of time. Individual operators can be combined into groups called events. Each operator provides a set of parameters that define particle behaviors during the event. Particle Flow continuously evaluates each operator and updates the particle system accordingly. Particles can be sent from event to event using tests. Tests let you connect events together in series. A test can check for certain properties of the particles. Particles that pass the test move on to the next event, while those that do not meet the test criteria remain in the current event [13]. We implemented the simulation code as an operator of Particle Flow. This decouples the SPH simulation from the computation of other forces that may act upon the particles.

Particle Flow has its own integration routine. To avoid 'double' integration a Particle Flow node can overwrite the default integrator with a custom implementation. By this, we can integrate the particles in our operator and make sure that boundary conditions and collisions are handled properly. By reading out the acceleration channel we can use the Particle Flow forces to affect the simulation. The acceleration due to the Particle Flow forces changes the velocity on the client side while the SPH acceleration is added on the server. The mass is defined as constant for all particles; therefore, we can simply add the accelerations:

$$a_{total} = a_{SPH} + a_{ParticleFlow} \tag{1}$$

We adopted the PCI SPH technique [11] with strategies for GPU implementation according to Goswami et al. [14]. PCI SPH offers good time/visual quality ratio, high robustness, and easy implementation. However, the choice of simulation algorithm should not affect the system architecture. One of the benefits of our approach is that maintaining and changing the simulation implementation is decoupled and transparent: it can be done without changes to the local software setup of the client workstations as long as the parameter set stays the same.

4.1 Neighbor Search

SPH simulation is well suited for a parallel implementation since the particles can easily be distributed to the computing units. Efficient neighbor search for each particle is the challenging part for parallel implementations. Our decision for an acceleration structure was based on the requirement that we need a memory-efficient structure that can run on the GPU. We also wanted a concept that can be extended to a multi-GPU scenario similar to the multi-GPU work by Valdez-Balderas et al. [15].

Ihmsen et al. [16] compared different neighbor search data structures for multi-core architectures with shared memory and found that using a regular grid for neighborhood search with zCurve sorting to be the fastest solution for parallel implementations. For sorting the particles into the cells, we use the algorithm described by Goswami et al. [14]. Each particle is assigned a single integer hash value based on its cell coordinates. For better memory caching, a space-filling curve numbering is employed instead of a simple hash value. After a key/value sort of this array and the particle indexes, all particles that belong to the same cell can be found in a continuous order. By executing binary search in the hash array, the particles for a particular cell can be found.

4.2 Collision Detection

Collision geometry is sent to the server with a RPC call. We send the polygon positions and a global transform matrix of the object for each frame. We generate a hash value for the polygons of each object based on the vertex positions. The polygons are cached on the simulation server and are only updated if the hash value changes. To handle collisions we intersect the particle trajectories with the triangles in both the PCI prediction step and the final integration step. If a collision is found, we correct the position of the particle to the intersection point and set its velocity to zero. The additional pressure correction iterations of the PCI method propagate the pressure back into the fluid and lead to visually satisfying results, according to our experiences. Therefore, more sophisticated boundary handling methods are not needed for our use cases.

4.3 Blind Particles

To reduce the amount of data required for rendering, we introduce the use of blind particles. Only particles near the surface contribute to the surface generation and the rendering process. The particles inside the volume are not of any interest for visualization. This observation was used before to accelerate raytracing. We also exploit this fact for optimized disk caching and reduction of bandwidth requirements. Recent works [17, 14, 18] identify the surface particles from the simulation and only use those for surface generation. [18] use surface particles for surface generation in a tesselation process based on Marching Cubes, but not for directly raytracing the surface. [18] store the surface information on the grid that they use for their Marching Cubes algorithm while we are tagging the underlying particles with the particle type property directly.

However, identifying only surface particles is not sufficient when used with a production renderer. Figure 3 (a) illustrates this problem. In this example, the inner red surface only emerges because the inner particles were deleted. This problem arises for commercial raytracers, because the core components of the raytracing algorithm cannot be changed by plugins. Therefore, we cannot decide if the surface is valid or not by the algorithm, especially if camera rays that start inside the fluid should be possible. A ray that enters a fluid and passes

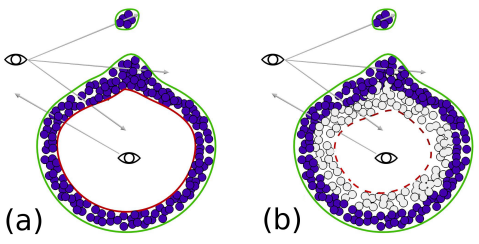

Figure 3: Surface generation without and with blind particles.

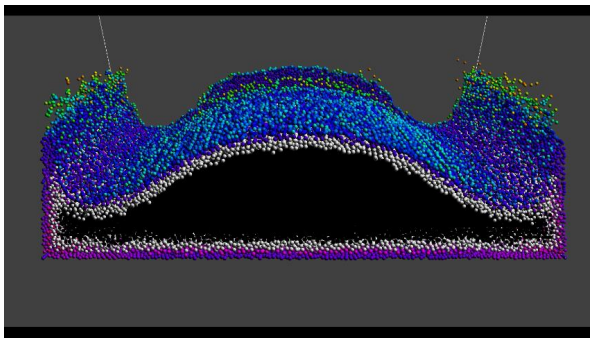

Figure 4: SPH particles color-coded by particle type. Black: particles to be omitted, white: blind particles, colored: surface particles color-coded by pressure.

the surface particles will generate a surface on the inside of the boundary particle hull. For the ray, it could also be a small splash particle or a thin fluid stream.

To solve this problem, we introduce blind particles. We not only identify the surface particles but also a thin layer of particles tagged as 'blind' around them. See Figure 3 (b). Blind particles prevent the generation of the inner surface. The rest of the particles is only relevant during simulation. Omitting particles would normally create new surfaces inside the fluid where they are missing. The blind particles will tell the surface generation algorithm to not generate a surface where their field value would normally create an isosurface. For identifying the surface particles, we adopt the method described by Goswami et al. [14]. A blind particle is tagged as blind if at least one neighbor in a threshold distance was tagged as a boundary particle. Figure 4 shows a typical scenario with blind particles, surface particles, and particles that are to be omitted.

4.4 Implementation Details

Similar to Goswami et al. [14], we use the zCurve approach for parallel implementation on the GPU. We temporarily store the neighborhood of each particle in memory for each timestep to reuse the information in the prediction/correction loop.

For the simulation code, we chose OpenCL over CUDA because of the ability to run it efficiently on CPUs as well. For the host code, we chose the Python OpenCL integration [19]. We experienced that kernels run at the same speed independent of whether host code is written in C++ or Python.

The advantage of a Python/OpenCL-based framework is optimal portability. The code ran on all combinations of Windows/Linux and GPU/CPU with only little modifications. However, some optimizations that would be possible for GPU only code were sacrificed for CPU compatibility. For example, the use of shared local memory was avoided in the kernel code.

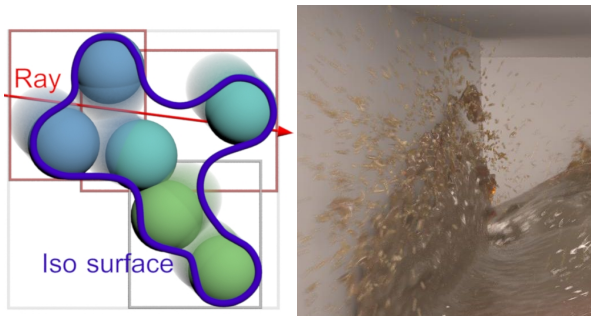

Figure 5: BVH for motion-blurred isosurface.

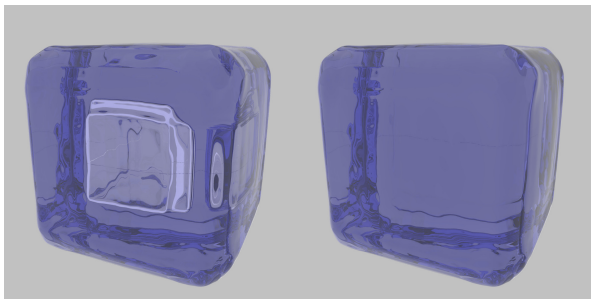

Figure 6: Rendering of a particle cube where inner particles were omitted: (left) without blind particles and (right) with blind particles.

5 RENDERING

Our contribution for the rendering of fluid surfaces is the integration of current fluid rendering algorithms in a VFX pipeline. Often, rendering is a step separate from animation and simulation. To adapt to this separation we designed the rendering routines as a geometry plugin for 3ds Max from Autodesk and the raytracer VRay from Chaosgroup. This renderer is available for all major 3D packages and is widely used in VFX companies in Europe. In general, there are two ways that are used to render fluid surfaces. One approach is to generate a triangle mesh that is then rendered with the standard triangle pipeline. Akinci et al. [18] proposed an efficient parallel implementation for the surface reconstruction. The other approach is direct raytracing of the surface where the intersection with the surface is found for every ray [20]. [21] accelerated direct rendering of fluid surfaces on the GPU by sampling the particle values to a perspective grid. We implemented a surface reconstruction algorithm and a direct raytracing routine and compare both solutions.

In the preparation phase, we fill the particles into a bounding volume hierarchy (BVH), taking the radius of each particle into account. Our implementation differs from other current implementations [22] regarding that if motion blur is required, we use both the position at the beginning of the motion blur interval and at the end to define the leaf nodes. Each particle carries velocity information; the ray request from the raytracing system has time information. With this information, each sample can calculate the exact positions of the particles in the intersected BVH leaves that were generated for the full frame length. Figure 5 shows how the BVH is used for rendering motion-blurred isosurfaces.

As a first step, we find the intersected leaf nodes. We then sort those nodes by their intersection point along the ray. We then march along the ray from the first intersection point into bounding boxes and evaluate a level-set function for the particles in the leaf node. We implemented both the simple Blinn blob and the surface function according to Zhu and Bridson [23]. If the sign of the result changes from one evaluation to the next, then the surface lies between those marching steps. The accurate position is found via binary search.

We omit the intersection if most particles involved in the calculation of the level set are flagged as blind particles. This robustly handles all secondary rays like shadow rays, reflection/refraction rays, and rays for indirect illumination. Also rays that start inside the fluid are handled correctly. Figure 6 compares rendering with and without blind particles.

Direct raytracing is very fast for opaque materials. For ray marching near the boundaries, only outer leaf nodes of the BVH have to be considered and only a few particles add to the level-set function. Once the first intersection is found, the function is finished and may return. The slightly longer raytracing times compared to intersection with polygons are compensated with the much faster preparation times per frame, since no marching cubes [24] or similar meshing phases are needed for explicit isosurface extraction. Especially in cases where huge water masses are simulated, but the camera captures just a fraction of them, the direct rendering method can speed up rendering substantially. An important advantage of the direct raytracing approach is the handling of 3D motion blur. Mesh-based rendering methods have to calculate multiple meshes per frame to obtain clean motion blur in order to compensate for changes in the mesh topology. Modern raytracers can render motion blur based on velocity information per vertex, but this approach can not consider topology changes e.g. merging of fluid drops. For direct raytracing, it does not matter if the topology changes inside the time frame of the motion blur. In Figure 5, the isosurface for a single point in time is drawn as seen by a specific ray sample. For each sample, we calculate the exact position of the surfaces at the requested point in time.

However, the advantage of faster rendering times is lost for transparent materials like water. Here, ray marching has to continue through the material. Also reflections inside the fluid add to the rendering times. Here, the caching strategies VRay provides for polygons outperform direct raytracing. To allow for polygon-based ray-

Scene Stepsize dt	Dambreak 20k 0.01s	Dambreak 125k 0.01s	Dambreak 1.25M 0.01s
Local Intel i7	0h:01m:27s	0h:13m:29s	2h:54m:08s
Local NVidia GTX 570M	0h:00m:48s	0h:05m:06s	1h:09m:28s
Local NVidia GTX 680	0h:00m:19s	0h:02m:16s	0h:29m:34s
Server/client GTX 680	0h:00m:20s	0h:02m:19s	0h:30m:17s

Table 1: Simulation time for 100 frames.

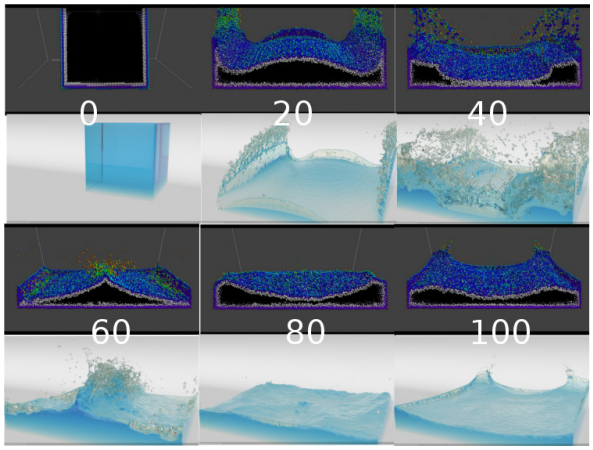

Figure 7: Different time steps of our dambreak simulation.

tracing we also implemented a multi-threaded marching cubes algorithm to create polygons.

6 RESULTS

6.1 Performance Results

We provide performance numbers for the simulation and the rendering components of our system. All tests use a dambreak simulation designed by us. Figure 7 illustrates different time steps of the simulation; also see the accompanying video. We use different volume quantities for the SPH simulation keeping the kernel radius at 4 cm and time step at 0.01 s (with 20 k, 125 k, or 1.25 M particles). The test environment was a mobile workstation equipped with an NVidia GTX 570 M GPU and Intel I7 CPU (2.0 GHz) and a desktop workstation with an NVidia GTX 680. In the client/server configuration, the mobile workstation was the client with 3ds Max and the desktop workstation was the simulation server. The network was a 1 GBit Ethernet connection.

Table 1 documents the simulation times for the different hardware platforms and SPH quantities. The simulation times include transfer of the particle data between client and server for each frame and, therefore, may be slower than times presented in other SPH real-time papers. However, the difference between running the server on the same machine (“local" in Table 1) and running over the network was only about 2 percent. The advantage of having access from the mobile workstation to the computing power of the GTX 680 card in the desktop workstation, which resulted in about twice the speed, by far compensated the communication overhead. Especially, time-critical productions may benefit from the flexible use of external compute resources.

Scene		Dambreak 20 k	Dambreak 200 k
Marching Cubes	opaque	0:36	2:46
	transp.	0:39	2:27
Direct raytracing	opaque	0:18	2:01
	transp.	0:55	6:18
Direct raytracing blind	opaque	0:18	1:57
	transp.	0:55	4:43

Table 2: Rendering times (min:sec).

Table 2 shows the rendering times on the mobile workstation for the small and medium-sized dambreak simulations. The rendering times were recorded for frame 40 of the simulation, where a large number of active particles are present. As shown in Figure 8, there are other time steps of the simulation that have much fewer active particles. In those cases, savings from direct raytracing are even more pronounced than for frame 40 of the animation.

Finally, Figure 9 compares rendering times between the blind particle method and the conventional method. The direct raytracing implementation is especially helpful in the setup phase for low-resolution test rendering. Especially for small cropped render tests without antialiasing, direct raytracing outperforms methods that need preparation steps with meshing. This allows fast iterations in the lighting phase.

Longer preparation times for polygon generation pay off in more complex scenes with multiple ray bounces. For each ray sample, the scalar field of the isosurface has to be evaluated multiple times for the binary search in order to find the intersection point. This is a costly operation. For small resolutions without antialiasing and only single ray bounces, this is still faster than surface reconstruction. Antialiasing schemes dramatically increase the rendering time for direct raytracing while only moderately increase the rendering time for polygons. This can be seen in Table 2 for frame 40 of our dambreak simulation. All tests were performed with an image resolution of 1280×720 pixel and fixed-rate antialiasing. Our conclusion is that direct raytracing should be used in the setup phase, when small resolution test renders are made or in situations where only

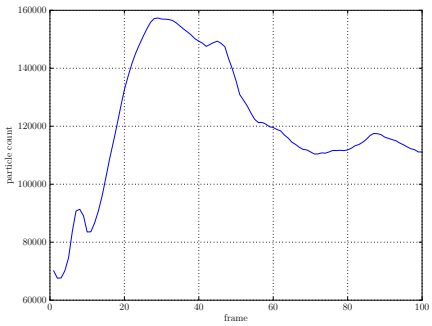

Figure 8: Particle count of active particles (boundary and blind particles) of the 200 k dambreak simulation.

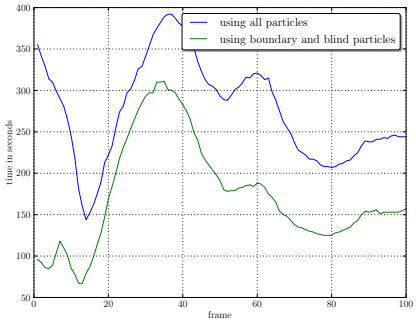

Figure 9: Rendering times for the blind particle and conventional methods for the 200 k dambreak simulation.

portions of the simulation volume are seen by the camera.

Our geometry plugin supports all VRay render elements. Being able to deliver all requested render element passes is an important factor for successful integration in today's production pipelines.

6.2 User Study

We assessed the usability and effectiveness of our system conducting a user study with VFX professionals. The study is mainly of qualitative nature, accompanied by quantitative and qualitative questionnaires. In addition to gathering information to improve the usability of our system (as part of a formative process), we wanted to test two hypotheses: "SPH solver integration into Particle Flow improves the workflow in 3ds Max" (Hypothesis 1) and "Direct raytracing is suitable for VFX production" (Hypothesis 2).

Our study was designed to test realistic work environments and tasks. Therefore, we recruited specialists for fluid-related VFX. Due to the highly specialized user group, the number of participants was small (three). Therefore, we have chosen a qualitative user study design: we used the think-aloud method [25] in addition to the questionnaires to obtain maximum feedback from each individual. According to Nielsen [26], three expert users can be sufficient for a think-aloud study. A screen and audio recording was later transcribed into text and selected comments were then categorized into the phases 'initial setup', 'tweaking simulation', 'tweaking surfacing', 'tweaking render settings', and placed in a review document.

All participants had more than 3 years of professional experience with 3D software and spent more than 6 hours per day working with 3ds Max. All participants were working on fluid-related VFX projects in their jobs at the time of the user study. Since VFX professionals with a fluid background and 3ds Max experience are scattered over the world, the user study was performed remotely with screen sharing sessions. The participants temporarily installed our plugin on their workstations. After an introductory session of about 30 minutes, we asked them to design a dambreak-like animation. The task was easily explained and can be solved in the short amount of time available. As an additional requirement, they were asked to add additional forces from 3ds Max to the simulation like e.g. a vortex to explore the coupling with 3ds Max forces. The participants were instructed to think aloud while they work to protocol their first impressions.

After the test, we asked the participants to fill in an online survey with questions about the usability of the system. Included in these were questions about different areas of the system like *"Integrating fluid solver in Particle Flow allows me to achieve a greater variety of effects than a standalone solver."* and the 10 standard System Usability Scale (SUS) questions [27]. The duration of the test was between 30 and 45 minutes.

All participants agreed that the overall workflow is better if the fluid simulation is integrated in the 3D content creation software in contrast to standalone fluid simulation applications. The usability was also rated very good in the SUS questionnaire. This is attributable to the fact that the solver blended into the interface for which they were experts. In particular, all users appreciated the flexibility that the integration into Particle Flow offered. We also asked about the subjective opinion on the visual quality of direct raytracing of fluid blobs in contrast to meshing approaches. All agreed that direct raytracing offers superior image quality; no participant considered it slow. All would consider it for their next project. The detailed questions and results of the questionnaires can be found in the supplementary material.

The participants were also asked for unstructured feedback. Some of the representative feedback includes: "The integration of an SPH solver into a particle system is basically interesting. It often happens in daily work that you have to add small fluid simulations on top of existing particle simulations, where you don't want to setup a big system. For example liquid spurts in battle scenes, where a complete fluid simulation would

be too much, but still small effects are needed." Or: "This should actually in theory kill the performance because we are raytracing into an isosurface and that is something you shouldn't do. Sure enough it is going slightly slower but testament to the quality of the mesh it's intersecting – even at the low settings – it is really not." Or: "It is really a dream to be able actually stop a render this quickly and go back to your settings, and change them, tweak them, press f9 to re-render and it is there.... Seriously, on a daily basis I have to wait 15 minutes between stopping a render and releasing all memory possible from the computer it takes minutes and then I make the one small change like 'I need two more of this' and press f9 and wait 10 minutes before the fist bucket is on screen."

In summary, the user study provides a preliminary indication that our approach provides a useful integration of SPH simulation in the VFX workflow (Hypothesis 1) and that direct raytracing can be suitable for certain aspects of VFX production (Hypothesis 2).

7 CONCLUSION AND FUTURE WORK

Our approach shows that external and distributed computing resources can be integrated in the established 3D workflow of VFX production companies seamlessly using commercial software packages allowing interactive sessions. In our user study, we confirmed the good utility of our approach for domain experts. We have presented an approach to reducing the memory footprint of particle caches without visual difference. This is especially important if the data has to be transferred over network. The idea of blind particles is independent from the simulation concept used and can be easily integrated in existing pipelines with savings in both rendering time and storage requirements. The performance tests confirmed that the distributed simulation leads to negligible communication overhead.

In future work, our implementation could be extended to a multi-GPU system to be employed in scalable hardware environments such as GPU clusters on the server side. Our generic client/server architecture could be extended to other fields of physically based simulation. Similarly, other commercial 3D packages could be integrated with our system.

8 ACKNOWLEDGMENTS

This work was partly supported by "Kooperatives Promotionskolleg Digital Media" at Stuttgart Media University and the University of Stuttgart.

9 REFERENCES

[1] Next Limit. Realflow product website. `http://www.realflow.com`.

[2] 3D Aliens. Glu3d product website. `http://3daliens.com/joomla`.

[3] M. Lagergren. GPU accelerated SPH simulation of fluids for VFX. Report LiU-ITN-TEK-A–10/044–SE, Dept. Sci. Tech., Linköping University, 2010.

[4] E. G. Parker and J. F. O'Brien. Real-time deformation and fracture in a game environment. In *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Anim.*, pages 156–166, 2009.

[5] J. J. Monaghan. Smoothed particle hydrodynamics. *Ann. Rev. Astron. Astrophys.*, 30:543–574, 1992.

[6] B. Adams and M. Wicke. Meshless approximation methods and applications in physics based modeling and animation. In *Eurograph. 2009 Tutorials*, pages 213–239, 2009.

[7] M. Mueller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Anim.*, pages 154–159, 2003.

[8] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26(3):48, 2007.

[9] H. Yan, Z. Wang, J. He, Xi Chen, C. Wang, and Q. Peng. Real-time fluid simulation with adaptive SPH. *Comput. Anim. Virt. Worlds*, 20:417–426, 2009.

[10] M. Becker and M. Teschner. Weakly compressible SPH for free surface flows. In *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Anim.*, pages 209–217, 2007.

[11] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. *ACM Trans. Graph.*, 28(3):40:1–40:6, 2009.

[12] Furuhashi S. msgpack website. `http://msgpack.org`.

[13] Autodesk. Autodesk 3ds Max userguide. `http://docs.autodesk.com/3DSMAX/15/ENU/3ds-Max-Help/index.html`.

[14] P. Goswami, P. Schlegel, B. Solenthaler, and R. Pajarola. Interactive SPH simulation and rendering on the GPU. In *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Anim.*, pages 55–64, 2010.

[15] D. Valdez-Balderas, J. M. Domínguez, B. D. Rogers, and A. J. C. Crespo. Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters. *J. Parallel Distrib. Comput*, 2012. doi 10.1016/j.jpdc.2012.07.010.

[16] M. Ihmsen, N. Akinci, M. Becker, and M. Teschner. A parallel SPH implementation

on multi-core CPUs. *Comput. Graph. Forum*, 30(1):99–112, 2011.

[17] Y. Zhang. Adaptive sampling and rendering of fluids on the GPU. In *Proc. Symp. Point-Based Graph.*, pages 137–146, 2008.

[18] G. Akinci, M. Ihmsen, N. Akinci, and M. Teschner. Parallel surface reconstruction for particle-based fluids. *Comput. Graph. Forum*, 31(6):1797–1809, 2012.

[19] A. Kloeckner, N. Pinto, Y. Lee, B.C. Catanzaro, P. Ivanov, and A. Fasih. PyCUDA: GPU run-time code generation for high-performance computing. *CoRR*, abs/0911.3456, 2009.

[20] J. C. Hart. Ray tracing implicit surfaces. *ACM SIGGRAPH 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, pages 1–16, 1993.

[21] R. Fraedrich, S. Auer, and R. Westermann. Efficient high-quality volume rendering of sph data. *IEEE Trans. Vis. Comput. Graph.*, 16(6):1533–1540, 2010.

[22] O. Gourmel, A. Pajot, M. Paulin, L. Barthe, and P. Poulin. Fitted BVH for fast raytracing of metaballs. *Comput. Graph. Forum*, 29(2):281–288, 2010.

[23] Y. Zhu and R. Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.

[24] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Comput. Graph.*, 21(4):163–169, 1987.

[25] C. H. Lewis. Using the “Thinking Aloud" method in cognitive interface design. Tech. report RC-9265, IBM, 1982.

[26] J. Nielsen. Estimating the number of subjects needed for a thinking aloud test. *Intl. J. Human-Comput. Stud.*, 41(3):385–397, 1994.

[27] J. Brooke. SUS: a quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability Evaluation in Industry*. Taylor and Francis, 1996.