Fast Rendering of Particle-Based Fluid by Utilizing Simulation Data

Ren Yasuda [1] Takahiro Harada[2] and Yoichiro Kawaguchi [1]

[1]The University of Tokyo
[2]Havok

Abstract

This paper presents a novel algorithm for efficiently visualizing of particle-based fluid simulation with multiple refractions, especially smoothed particle hydrodynamics(SPH), using data and result of simulation itself. In general, particle-based fluid simulation and visualization are processed completely separated. The novelty of our method lives in combination of these two processes to avoid extra processes in visualization, and to mitigate computing cost of visualization.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—

1. Introduction

Real-time fluid simulation on the GPU, both grid-based and particle-based, is no longer difficult because of the growth of computing power and programmability of the GPU. There still are many challenging topics in simulating fluid with advanced methods in real-time, of course, but it is also true that "simple" real-time fluid simulation is no longer difficult. However, visualization of fluid is still left because of the difficulty of constructing a smooth surface from the discrete parameters of fluid. In particular, visualizing particle-based fluid with dynamically moving computational elements is more difficult than grid-based simulation with a static computational elements. So, several researchers are studying the visualization of particle-based simulation by using point-based, polygon-based, voxel-based approaches. Among these, voxel-based approach has an advantage which makes it possible to calculate multiple refraction using volume traversal by ray casting. Therefore, we used this approach.

Traditional voxel-based method is very simple. First, a density field in a 3D volume is constructed and then traverse the volume by ray casting. The density field is constructed by summing the density of particles, each particle is assumed to be surrounded by the same density distribution, where the density of the particle is modeled as a function of the distance from the center of the particle. However, each particle does not need to have the same density distribution as will be explained in this paper: particles in sparse areas should have a wide density field and vice versa.

For efficient visualization, we introduce "particle mask" for each particle. This parameter represents the grid occupancy of particles in six directions. Each particle deforms its density field on the basis of the "particle mask", and therefore we can avoid redundant computation for constructing density fields in a 3D volume.

2. Related Works

A typical approach of rendering metaballs, Marching Cubes Algorithm, is a polygon-based method and was proposed by Lorensen *et al.* [LC87]. However, it is difficult to initeractively render metaballs of polygon-based method with complex refractions, because they are represented as mesh data.

On the other hand, there also have been several studies about rendering of metaballs using methods other than the polygon-based method. Muller *et al*. proposed a screen-space rendering method of metaballs using iterative approximation of isosurface [MGE]. As their method is not for fluid visualization, it cannot calculate multiple refraction. Kanamori *et al.* computed ray-isosurface intersection using depth peeling and *Bezier Clipping* [KSN08]. However, multiple rendering passes are needed to detect the ray-isosurface

intersecting point, and therefore rendering with multiple refractions is difficult. Iwasaki *et al*. visualized the surface of particle-based simulation using a point-based method [IDYN06]. Their method is for surface generation, so details are lost.

3. Voxel-Based Vsualization Method

In this section, we discuss voxel-based visualization of particle-based fluid. Density field in a volume is constructed after fluid simulation, and then surface is extracted by ray casting and particles are visualized as fluid with complex refractions. We used a volume that had the same resolution as the fluid simulation grid for maximum efficiency;as described in Sec. 4.2.1. The density of the 3D volume is constructed by adding a density distribution of each particle to voxels within a certain distance from the center of the particle. Finally, the surface of the fluid is extracted by ray casting, a method that traverses a volume data at tiny intervals and detects surfaces using certain thresholds.More precise surface position can be obtained by using internal divisions. An benefit of this method is that, a normal vector can be easily obtained from the gradient at the detected position, and therefore visualizing volume with multiple refractions by computing refraction using a normal vector is comparatively easy. The threshold for detecting surfaces is set high enough that the minimum size of a drop attributed to only one particle was not smaller than the size of a voxel. This ensured that each particle added density higher than the threshold to at least one voxel and prevented drops from disappearing.

4. Algorithms

4.1. Smoothed Particle Hydrodynamics

We used smoothed particle hydrodynamics (SPH) for the fluid simulation. An SPH simulation stores the position, velocity, and density of each particle during the simulation and a grid is used to make the simulation efficient. The grid stores the indices of particles in each grid cell. It is used for accelerating neighboring searches during force and density computation. An SPH simulation is suited for parallelization, so there have been several studies on GPU-based implementation of SPH [AIY*04] [HKK07].

4.2. Density Field Construction

After simulating fluid by SPH, we construct a density field by adding the density distribution of each particle, to voxels within a certain distance, as described in Sec. 3. This process can be a bottle neck of the computation because each particle accesses many voxels in this process, and the massive number of memory accesses leads to high latency. Therefore, we separate the construction of the density field into two processes: rough density field construction and fine density field construction (Fig. 1). In addition, particle mask accelerates the construction of the fine density field further, with which the computation of particles in dense areas can be skipped. To skip density-adding operations in the construction of the fine density field, we define a voxel that has higher density than a threshold as "a saturated voxel". Voxels saturated by the construction of the rough density field are skipped in the construction of the fine density field. We discuss the construction of the rough density field and fine density field, the computation of particle mask, and voxel saturation in the followings.

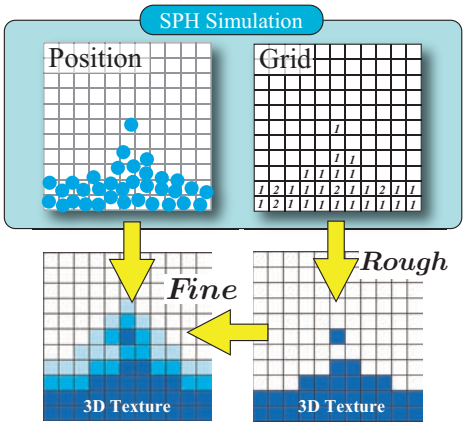

Figure 1: *Flow of constructing density field*

4.2.1. Rough Density Field Construction

First, we construct the rough density field. Because we ensure that each particle adds density higher than the threshold to at least one voxel, voxels including one or more particles will certainly be saturated after the construction of the whole density field. Fortunately, the SPH grid has the number of particles in each grid cell. Therefore, we set the resolution of the volume to the same as the SPH grid and saturate voxels including one or more particles first; we call this process rough density field construction. Each voxel first loads the number of particles from the corresponding grid cell of SPH. If the number is other than zero, the voxel saturates; if the number is zero, the density of the voxel is set to zero.

4.2.2. Fine Density Field Construction

The rough density field does not have detailed information of the surface although it knows which grid is already saturated. Therefore, each particle adds value to the neighboring cells to calculate the details. However, because it does not have to add value to a saturated cell, the density of voxels are loaded at first and operations are skipped at saturated voxels. Although a lot of computation is skipped because of the rough density field, constructing the fine density field requires a lot of memory access. This is because each particle

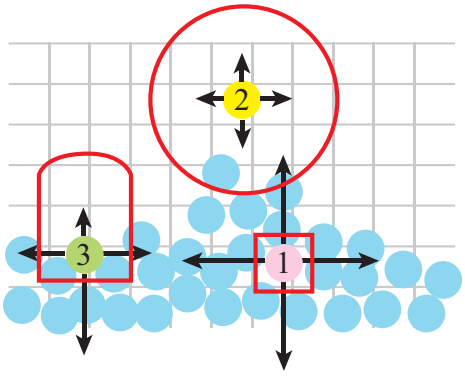

Figure 2: *Particle mask (2D for simplicity, so particle mask only has four parameters). Black arrows represent the particle mask of corresponding directions, and red curved line represents the extent to which a particle has impact.*

has to load the value in the rough density field from all cells within the effective radius at least once. Memory access is much expensive than arithmetic on the GPU. To reduce the memory access, we introduced *particle mask*.

Particle mask includes six parameters, each of which corresponds to +x, -x, +y, -y, +z, and -z. The particle mask is used to skip the computations by deforming a density distribution of each particle as illustrated in Fig. 2. There are three highlighted particles in different positions in the Fig. 2. Particle 1 is an underwater particle, and so each component of particle mask (represented as a black arrow) has the maximum value. Particle 2 does not have any neighbor in its effective radius, therefore all components of particle mask have the minimum value. Particle 3 exists on the surface, so the downward direction has the maximum value, the upward direction has the minimum value, and the left and right directions have medium values. The density distribution of each particle is deformed according to the particle mask in order to reduce the computation in the direction of higher particle mask.

4.2.3. Computation of Particle Mask

Although neighboring searches are needed for the computation of particle mask, additional memory accesses are not required because this process is set in the process of computing density of the SPH simulation, which also has the neighboring search process. The +x component of particle i $[P_i]_{px}$ is calculated as follows.

$$[P_i]_{px} = \sum_j \frac{k}{[r_{ij}]_x / ([r_{ij}]_y)^2}, \qquad (1)$$

where r_{ij} is a vector from particles j to i, $[r_{ij}]_x$ and $[r_{ij}]_y$ represent the x and y components of r_{ij}, respectively, and k is a coefficient.

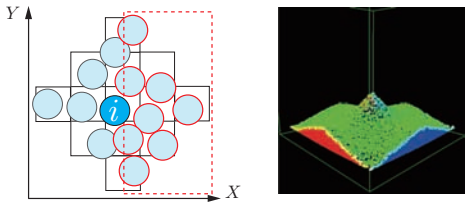

Figure 3: *Computation of particle mask. Left) Particles used for computing positive x component of particle mask of particle i. Right) Colored SPH particles visualized using particle mask.*

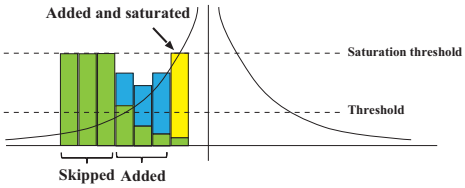

Figure 4: *Atomic operation of a particle and surrounding voxels. Green bars are values already added by other particles. Atomic operation not executed if value of voxel is over saturation threshold. If value of voxel exceeds saturation threshold, value is clipped.*

4.2.4. Voxel Saturation

If a voxel is already saturated, accumulation of the density to the voxel can be skipped. The threshold we used here to define voxel saturation is not the same as the threshold used for defining the surface at ray casting. This is because in ray casting, the value at a point is obtained by linear interpolation of values on grid points. Therefore, if values of voxels are clipped by the ray-casting threshold, the ray would detect the surface at a point different from that in the case of using the saturation threshold; this leads to apparent deterioration. We chose a threshold about five times larger than the threshold used for the surface definition for our cases. If a smaller threshold is used, the quality of the surface decreases, although the computation decreases. There is a trade off between the quality of the surface and the computational burden.

4.3. Ray casting

Finally, we visualize the density field by ray casting, described by Crane *et al.* [CLT07]. The maximum reflaction depth is set to 4 in the results shown in this paper. This makes it possible to see water through water, which has never before been accomplished in real-time.

5. Results and Conclusions

The proposed method was implemented on a PC with Core 2 Duo 3.0 GHz CPU and Nvidia GeForce GTX 280, using

Figure 5: *Left) Real-time rendering of 27,000 fluid particles with* $64*64*64$ *grid; image size of figures is* $512*512$*. Even though fluid simulation is iterated 5 times with each frame, this program runs at about 30 fps, including density field construction, rendering, and simulation time. Calculating multiple refractions enables to see water through water. Right) Rendering of 1,000,000 particles with* $192*192*192$ *grid. This is not real-time simulation (1.8sec/frame), but our method makes the density field construction 2.5 times faster. Background image from <www.debevec.org>.*

	8,000	27,000	64,000	125,000
Sim1	14,533.0	33,058.3	48,941.9	72,054.3
Sim2	7,687.7	12,788.7	16,180.1	19,993.7

Table 1: *Computing performance of density field with* $64*64*64$ *grid*

OpenGL, Cg, and CUDA. The real-time SPH simulation including visualization of the proposed method is shown in Fig. 5.

A comparison of the computational time of constructing the density field between a simulation that used a density field construction method without and with the proposed method is shown in Table 1 as Sim1 and Sim2, respectively. We can see that the proposed method reduces the computational cost of density field construction, which is expensive compared to that of SPH simulation. The proposed method reduces the computation cost at least by half with 8,000 particles and the difference grows as the number of particles increases. When 125,000 particles were used, the computation was more than 3.5 times faster than one without our method.

This paper proposed a new method for accelerating the construction of a density field by using the results of SPH and visualized SPH with multiple refractions. There are several points which should be addressed in future works. First, the rendering speed is strongly connected to the resolution of the output image. To improve the speed, the GPU-based octree introduced by Sun *et al*. [SZS*08] should be of help. Second, our method is fully grid-based, so the minimum size of a particle cannot be smaller than the size of a voxel. This restriction would be circumvented by separating particles in sparse areas from other dense particles and calculating isosurface directly from their positions.

Acknowledgements

This research was supported by Core Research for Evolution Science and Technology (CREST) of Japan Science and Technology Agency(JST).

References

[AIY*04] AMADA T., IMURA M., YASUMURO Y., MANABE Y., CHIHARA K.: Particle-Based Fluid Simulation on GPU. In *ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH* (2004).

[CLT07] CRANE K., LLAMAS I., TARIQ S.: Real-time simulation and rendering of 3d fluids. *GPU Gems 3* (2007), 633–674.

[HKK07] HARADA T., KOSHIZUKA S., KAWAGUCHI Y.: Smoothed particle hydrodynamics on GPUs. In *Computer Graphics International* (2007), pp. 63–70.

[IDYN06] IWASAKI K., DOBASHI Y., YOSHIMOTO F., NISHITA T.: Real-Time Rendering of Point Based Water Surfaces. *LECTURE NOTES IN COMPUTER SCIENCE 4035* (2006), 102.

[KSN08] KANAMORI Y., SZEGO Z., NISHITA T.: GPU-based Fast Ray Casting for a Large Number of Metaballs. In *Computer Graphics Forum* (2008), vol. 27, Blackwell Synergy, pp. 351–360.

[LC87] LORENSEN W., CLINE H.: Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM New York, NY, USA, pp. 163–169.

[MGE] MULLER C., GROTTEL S., ERTL T.: Image-Space GPU Metaballs for Time-Dependent Particle Data Sets.

[SZS*08] SUN X., ZHOU K., STOLLNITZ E., SHI J., GUO B.: Interactive relighting of dynamic refractive objects.