Interactive Simulation and Visualization of Fluids with Surface Raycasting

Renan Teston Inácio, Tiago H. C. Nobrega, Diego D. B. Carvalho and Aldo von Wangenheim
LAPIX - Laboratory for Image Processing and Computer Graphics
Federal University of Santa Catarina
Florianópolis, Brazil.
e-mails: {zce,tigarmo,diegodbc,awangenh}@inf.ufsc.br

***Abstract*—We present a method to couple particle-based fluid simulation methods such as Smoothed Particle Hydrodynamics (SPH) and volume rendering in order to visualize the fluid. A volume is generated from the fluid's implicit density field so volume raycasting can be performed to render the surface on the GPU. The volume generation algorithm is also implemented in the GPU and is suitable to be used with both SPH simulations implemented on CPU and GPU. We compare different implementations of fluid simulation, demonstrating the modularity of the volume generation method and concluding that it can be used together with other particle-based simulation models and other volume rendering techniques.**

***Index Terms*—graphics processors; fluid simulation; sph; raycasting;**

I. Introduction

Simulation of physical phenomena can be employed in a wide area of applications, from engineering to health, and they have different requirements, ranging from precision to interactivity. Engineering applications typically simulate situations without user interference during calculations, while surgical simulators, for example, must support user interaction during the simulation. The possibility of user interaction improves the perception of reality, since in the real world we can observe the consequences of an action right on time.

In this work we propose the coupling of a fluid simulation based on a particle system with the raycasting algorithm to render the fluid simulation for interactive applications. The fluid's particles are represented by a set of spheres that have their movement calculated by the Smoothed Particle Hydrodynamics (SPH) method. The simulation is performed in a separate step from the visualization. To demonstrate the modularity of our approach we present a CPU and GPU implementation of the SPH method.

After calculating particles' positions, we arrange them all in a tridimensional grid and build a voxel model, where a region that has no particles represents an empty cell in the grid. The raycasting algorithm is applied in the volume comprised by this grid. In order to achieve an acceptable frame rate in our simulation we decided to use a raycasting implementation in the GPU.

In the second section we discuss fluid simulation and visualization. In the third section we describe the SPH method while the fourth section deals with our raycasting implementation and the volume generation from a set of particles. In the fifth session we show our results obtained in the SPH's CPU and GPU versions and perform a comparison between the results with and without raycasting. The last section contains our final remarks and future works.

II. Related Work

The scientific literature is rich in both fluid simulation using the *Smoothed Particle Hydrodynamics* model (henceforth called *SPH*) and in volume rendering through raycasting. Gingold and Monaghan[1] presented the SPH model in the 70's for astronomical modeling and since then various authors used it to model a variety of different, smaller-scale phenomena, such as arbitrary highly-deformable bodies[2], lava flows[3], [4], viscoelastic fluids[5], [4], and melting[6]. Müller et al.[7] used SPH to simulate fluids at interactive rates. Amada et al.[8] initially attempted to perform the SPH simulation on the GPU, an endeavor further pursued by Harada et al.[9], [10].

Visualization of the fluid can be accomplished by extracting the isosurface from the fluid's implicit function, evaluated on a discrete tridimensional grid, using Marching Cubes[11] and then rendering the generated polygons. In order to render the fluid with high quality the grid must be subdivided in very small cells, which will make the algorithm generate a lot of small polygons. The amount of these small polygons can be much higher than the amount of pixels on the screen, resulting on bottlenecks on polygon processing. Iwasaki et al.[12] used a point-based method to render the fluid surface and also created a method to create the volume grid on the GPU, but it requires multiple rendering passes.

Raycasting is used as a direct volume rendering technique in applications such as visualization of medical imaging[13], [14] and fluid simulation[9]. Volume raycasting can be accelerated by modern graphics hardware[15], [16], being adequate for interactive applications.

III. Interactive Simulation of Fluid Particles

The SPH model is an interpolation method in which the state of the fluid in a point in space is a function of the fluid's state in that point's immediate vicinity. The contribution that each point has around it is weighted by radial, symmetrical functions called smoothing kernels. The model we use is based on the work of Müller et al.[7] which does not enforce incompressibility, making the result less accurate but fast.

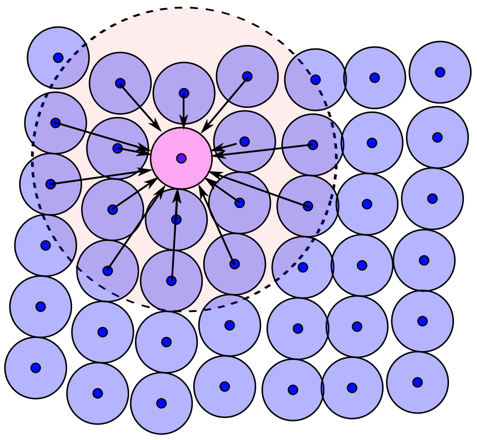

Figure 1. The shaded circular region contains the particles that exert an influence over the shaded central particle.

In the SPH model, a scalar quantity A is interpolated at a point in space $\boldsymbol{r}$ by a weighted sum of contributions from all points around $\boldsymbol{r}$:

$$A(\boldsymbol{r}) = \sum_{j=1}^{n} m_j \frac{A_j}{\rho_j} W(\boldsymbol{r} - \boldsymbol{r}_j, h), \tag{1}$$

where j iterates over the space around $\boldsymbol{r}$, W is the smoothing kernel and m and ρ represent mass and density, respectively. For a particle-based simulation, the summation is done over all the particles within a range h of the particle located at $\boldsymbol{r}$ (Figure 1). Particles beyond this radius exert zero influence over each other, and the value h is therefore known as the smoothing kernel's *support radius*. To simulate a fluid with the SPH model means to apply the interpolation equation to solve the Navier-Stokes equation of momentum:

$$\frac{\partial v}{\partial t} = \frac{-\nabla p + \rho \boldsymbol{f}_{ext} + \mu \nabla^2 \boldsymbol{v}}{\rho}. \tag{2}$$

In our implementation, the first stage of each simulation step is the definition of each particle's *neighborhood*, that is, the set of particles that are within h units of distance from each particle. Both our CPU and GPU implementation use a grid to accelerate the neighborhood query. Next, each particle's density is computed by replacing A with ρ in Equation 1 and by defining a smoothing kernel:

$$\rho(\boldsymbol{r}) = \sum_{j=1}^{n} m_j W_{poly6}(\boldsymbol{r} - \boldsymbol{r}_j, h),$$

where the W_{poly6} kernel has the following form[7]:

$$W_{poly6}(\boldsymbol{r}, h) = \frac{315}{64\pi h^9}\left(h^2 - |\boldsymbol{r}|^2\right)^3.$$

Therefore, the density computation is simply a summation of each neighbor particle's mass, weighted by the smoothing kernel. The steps that follow compute the pressure, viscosity and external forces components from Equation 2.

We implemented the fluid simulation on the GPU using the method developed by Harada et al.[9]. A grid is constructed on the GPU and the steps that follow use this grid to find the neighbor particles. The grid is stored as a RGBA texture, each color channel storing a particle. This leads to a limitation that each cell holds a maximum of 4 particles, requiring more grids to support more particles per cell. All the steps of the simulation runs on the fragment shader.

Note that in the work of Harada et al.[9] there are experiments with hundreds of thousands of particles and the processing time for each frame takes less than 100 milliseconds. In order to support that many particles the time step must be very small to keep the simulation stable, demanding more simulation frames per simulation time.

This work focus on interactive simulations, meaning that the simulation must be fast and have high response time. Ideally the simulated time would have to be the same as the processing time spent to execute the simulation step, i.e., having a ratio of simulated time per processing time at least equal to 1. However, higher values for the time step make the simulation less stable. For that reason we use a constant time step high enough for interactivity but that makes the choice of fluid parameters more strict to keep the simulation stable.

IV. Interactive Fluid's Surface Visualization

We use two approaches to visualize the fluid: rendering the SPH particles as spheres or rendering the implicit density field by volume raycasting. Figure 2 compares both approaches side by side. The notion of continuity given by the fluid's surface results in a more realistic visualization compared to the sphere rendering approach, although the latter is less time consuming. In this section we detail surface rendering.

In order to render the fluid surface a volume is generated from the SPH data after each step of simulation and stored as a 3D texture on the GPU. Both the volume generation and the raycaster algorithms are implemented as shaders and all the necessary data are stored in the video memory, so the whole process happens only on the GPU.

A. Volume Generation

After SPH simulation is performed we must obtain a scalar field to be used with a volume rendering technique. This scalar field can be discretized in a grid where each cell has its value $F(r)$ as the result of Equation 3. $F(r)$ is a sum over the particles where r is the position in world space of the center of the grid cell,W_{poly6} is the smoothing kernel and r_j and V_j are the position and volume of particle j.

$$F(r) = \sum_j V_j W_{poly6}(r - r_{j,}) \tag{3}$$

The values of the scalar field can be calculated on the fly when sampling it from a volume rendering technique or explicitly generated and stored in a discrete grid. The advantage of storing it on a discrete grid is that the sampling, on a GPU volume rendering implementation, is just a texture fetch with trilinear interpolation instead of fetching all the neighboring particles and applying the above equation. Another advantage is modularity: any GPU implementation of volume rendering

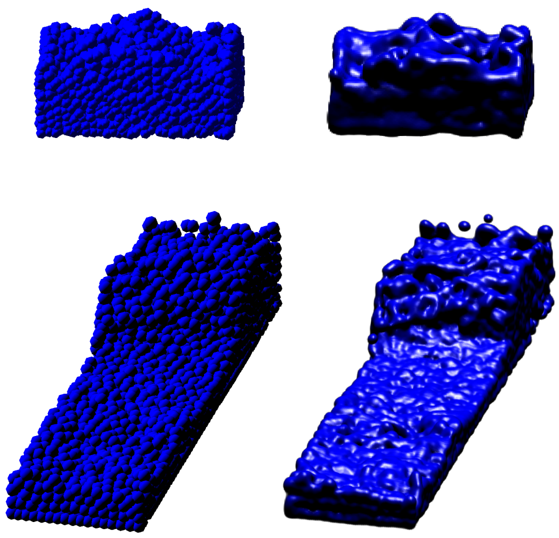

Figure 2. Comparison between rendering the particles' spheres and the surface rendered from the generated volume. These pairs of pictures were taken on the same frame on a simulation with 4096 particles.

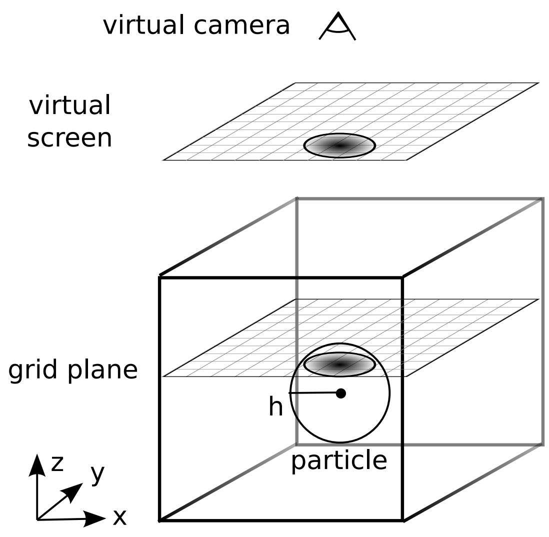

Figure 3. A disk is formed by the projection of a particle on a grid plane[12]. The virtual camera is set so when the disk is sent as a point sprite the generated fragments on the virtual screen correspond to the grid cells of the grid plane being processed. This must be done for every grid plane in order to build the volume grid.

or isosurface extraction can be used without modification, assuming its input already is a tridimensional texture.

We use a similar approach that is used in the work of Iwasaki et al.[12]. Their idea is to create texture-mapped disks for each intersection between the grid planes and the particles. These disks are then projected on the corresponding layer of the volume grid as shown in Figure 3. On the fragment shader a parcel of $F(r)$ is calculated for each fragment inside the disk. The particles are sent as point sprites and this rendering process is repeated for each grid plane. The result of each rendering is a volume layer that is transferred to the main memory, so the frame buffer can be cleared and the process repeated. Since the frame buffer can hold 4 scalars ($RGB\alpha$) per pixel, it is possible to calculate 4 grid planes per rendering.

The main disadvantage of the method described by Iwasaki et al.[12] is that it makes multiple rendering passes. For each rendering pass the intersecting disks must be sent and the result copied to the computer main memory. In order to find the intersecting particles a clustering step must be performed. Since a particle intersect multiple grid planes, the particle must be sent again to the GPU on each rendering pass. Another efficiency issue is the transfer of the frame buffer into the main memory. When using a volume rendering technique on the GPU, it is desirable that the volume is stored on the video memory, so the generated volume would have to be transferred to the GPU on the end.

We use OpenGL and GLSL to implement the volume generation algorithm on the GPU. The algorithm's inputs are the particles' position and volume. The particles are sent as point sprites positioned according to their position in world space and their volume are sent as the color of the point sprite. The result is a 3D texture with $F(r)$ evaluated for each voxel.

The method we propose runs entirely on the GPU and is implemented with OpenGL and GLSL. All the particles are sent as point sprites and in a single rendering step a 3D texture is filled with the generated volume. We use the geometry shader to generate all the disks for each particle and then write the results of the fragment shader directly onto a 3D texture. This is accomplished by sending only one disk, as a point sprite, for each particle. Then on the geometry shader additional sprites are created for that particle, i.e. the other disks of the particle intersection with the grid planes. Working with point sprites instead of quads is advantageous because only one vertex is sent to the GPU instead of four. Also, the sprite's texture coordinates can be automatically generated to be used in the fragment shader.

The texture coordinates are values inside the range $[0, 1]$ where the extreme values represent the extremities of the sprite, therefore the center of the disk is the coordinate $(0.5, 0.5)$. While the 2D texture coordinates of the sprite are calculated automatically, we need to calculate on the geometry shader a third coordinate that varies along the grid planes, i.e. are the same for a whole disk. The size of the sprite is defined to be the radius of the kernel smoothing length, so the distance to the center of the particle can be calculated in terms of world coordinates. On the fragment shader this distance is used to evaluate the kernel and calculate the corresponding parcel of the equation 3.

With the depth test disabled and the blending equation set to sum the fragments, the particle contributions are accumulated and the fragments are composed on the corresponding volume layer, previously specified on the geometry shader. The result is $F(r)$ evaluated for each voxel.

Note that the position of the particles are specified as point sprites, but their position and volume don't necessarely

Table I
AVERAGE TIMES (IN SECONDS) TAKEN BY EACH STAGE IN A SIMULATION FRAME, WITH RAYCASTING FOR THE RENDERING.

Raycasting, 1024 Particles	Fluid Simulation	Volume Generation	Raycasting	Rendering	Total
CPU Simulation	0.022	0.0075	0.0076	0.00051	0.038
CPU Simulation %	59.0%	19.6%	20.1%	1.4%	100%
GPU Simulation	0.0029	0.020	0.034	0.0013	0.058
GPU Simulation %	5.1%	34.3%	58.3%	2.3%	100%

Raycasting, 2048 Particles	Fluid Simulation	Volume Generation	Raycasting	Rendering	Total
CPU Simulation	0.042	0.0098	0.0078	0.00097	0.061
CPU Simulation %	69.4%	16.1%	12.9%	1.6%	100%
GPU Simulation	0.0025	0.021	0.034	0.0053	0.063
GPU Simulation %	4.0%	33.5%	54.0%	8.5%	100%

Raycasting, 4096 Particles	Fluid Simulation	Volume Generation	Raycasting	Rendering	Total
CPU Simulation	0.078	0.013	0.0078	0.00062	0.099
CPU Simulation %	78.3%	13.3%	7.8%	0.6%	100%
GPU Simulation	0.0034	0.024	0.034	0.0056	0.067
GPU Simulation %	5.1%	35.4%	51.1%	8.4%	100%

Raycasting, 8192 Particles	Fluid Simulation	Volume Generation	Raycasting	Rendering	Total
CPU Simulation	0.19	0.037	0.010	0.00087	0.24
CPU Simulation %	79.7%	15.6%	4.3%	0.4%	100%
GPU Simulation	0.0075	0.033	0.032	0.0014	0.074
GPU Simulation %	10.2%	45.0%	42.8%	1.9%	100%

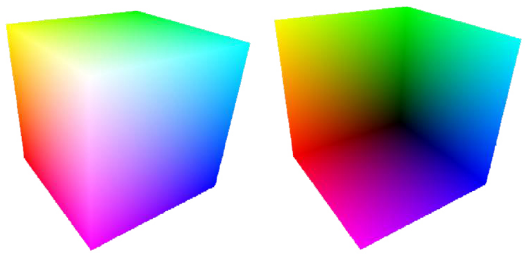
Figure 4. Rasterization of front and back faces of the volume's bounding box.

need to be stored and transferred from the main memory. Considering a SPH simulation implemented on the GPU that stores the position and volume of the particles in textures, we can setup a Frame Buffer Object with that texture, read its contents into a Pixel Buffer Object and then use it as a Vertex Buffer Object. That way the point sprites can be drawn using data already available on the video memory. Since both input and output are located on the GPU, the volume generation effectively bridges the simulation and rendering with no significant transfer overhead.

B. Raycasting

The product of the volume generation is a 3D texture which can be rendered by volume rendering techniques such as raycasting. Our raycaster is based on the work of Kruger and Westermann[15] and Scharsach[16].

The rays' start and end coordinates are determined by rasterizing the front and back faces of the volume's bounding box in world space. The normalized coordinates of the corners of the box are encoded in the color channel as RGB values. The resulting images look like Figure 4. Each pixel of the image generated from the front faces represent the normalized volume coordinates where the corresponding ray enters the volume. The other image has the exit information of the rays. The rays are then marched on the fragment shader with early ray termination when the pixel is opaque enough, according to a predefined threshold.

Table II
AVERAGE TIMES (IN SECONDS) TAKEN BY EACH STAGE IN A SIMULATION FRAME, WITH SPHERES FOR THE RENDERING.

Spheres, 1024 Particles	Fluid Simulation	Rendering	Total
CPU Simulation	0.021	0.013	0.034
CPU Simulation %	62.5%	37.5%	100%
GPU Simulation	0.0015	0.0077	0.0091
GPU Simulation %	16.1%	83.9%	100%

Spheres, 2048 Particles	Fluid Simulation	Rendering	Total
CPU Simulation	0.083	0.046	0.129
CPU Simulation %	64.4%	35.6%	100%
GPU Simulation	0.0024	0.017	0.019
GPU Simulation %	12.3%	87.7%	100%

Spheres, 4096 Particles	Fluid Simulation	Rendering	Total
CPU Simulation	0.080	0.044	0.124
CPU Simulation %	64.4%	35.6%	100%
GPU Simulation	0.0033	0.033	0.036
GPU Simulation %	9.1%	90.9%	100%

Spheres, 8192 Particles	Fluid Simulation	Rendering	Total
CPU Simulation	0.199	0.091	0.291
CPU Simulation %	68.5%	31.5%	100%
GPU Simulation	0.0058	0.060	0.066
GPU Simulation %	8.9%	91.1%	100%

V. RESULTS

The experiments were performed on a computer with an Intel Core 2 Duo P8400 2.26GHz as the processor and a NVIDIA GeForce 9800M GS with 512MB GDDR3 VRAM as the GPU. The operating system used was Linux 2.6.31 with the proprietary NVIDIA driver version 190.29. The resolution of the generated volume in the CPU simulation is 128x128x128, while in the GPU it is 128x512x512. In all experiments the raycaster rendered a 512x512 image.

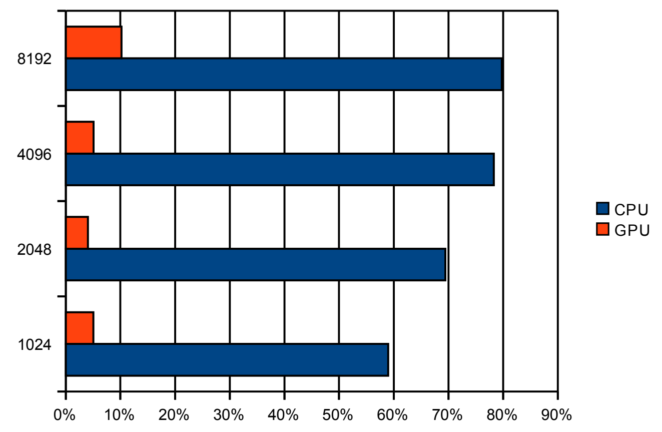

Figure 5. Percentage of time taken up by the fluid simulation when rendered with raycasting, with different number of particles.

Table I and Table II show the time taken by different stages in a simulation frame, respectively with raycasting and spheres for visualization. The "Fluid Simulation" stage takes up the majority of the frame when the stage is performed on the CPU, while the visualization-related processes ("Volume Generation", "Raycasting" and "Rendering") dominate the frame time when the whole simulation is performed on the GPU (Figure 5). This is to be expected because the fluid simulation is a fairly parallelizable process and hence very suited for the GPU.

Since the visualization portion dominates the frame, we experimented with performing more than one fluid-simulation step per simulation frame. For example: instead of performing one simulation step with $\Delta t = 0.01s$, we can perform 10 steps and use $\Delta t = 0.001s$ for each. The graph on Figure 6 shows that, for a simulation with 8192 particles, although 10 simulation steps are performed instead of only one, the total simulation time roughly only doubles.

Figure 7 shows the visual difference between each scenario. The screenshots were taken at the same simulation time for both the single $\Delta t = 0.01s$ simulation step and the ten $\Delta t = 0.001s$ simulation steps, i. e., the the bottom row calculated more simulation frames. A smaller time step tends to yield a more stable simulation, as depicted by the smaller "splashes".

The video[1] shows a simulation of SPH running on CPU inside an invisible box. When the user moves the box a force is applied on the particles and they react in real time. This is important because the simulation must not be only fast, but also have a high response time. The video also contains a comparison in real time of the simulation with both single and multiple time steps, in order to compare performance impact and stability of these approaches.

VI. Final remarks and future works

Analyzing the experiments and the results we can conclude that:

[1]http://150.162.202.1/~renan/sibgrapi2010/fluid_video.avi

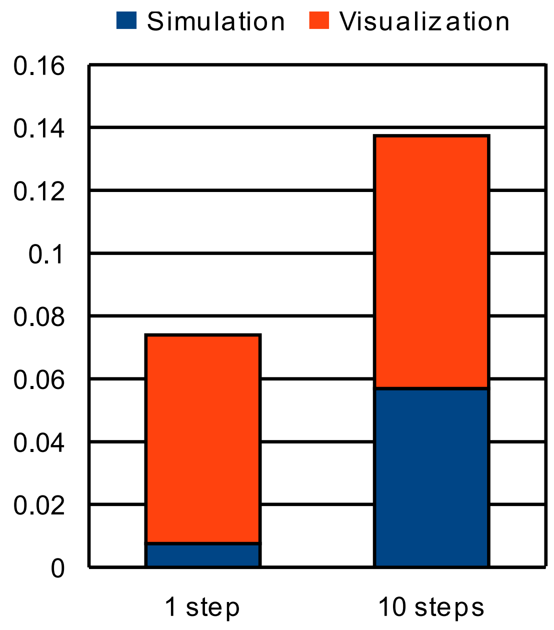

Figure 6. Average time taken up by the fluid simulation on the GPU with raycasting, with both the 1-step and the 10-step configurations and 8192 particles simulated.

- The so-called voxelization of the particles fits as a good approach in coupling a particle-based fluid simulation technique with different visualization techniques. Assuming the output of the simulation can be represented as particle positions with properties to define a implicit field, it can be used with this method. The generated volume could be rendered as a surface through a mesh generation algorithm (Marching Cubes, NURBS) or oher direct volume rendering techniques.
- The prototypes allow real time user's interaction. The effects of a fluid vs. solid collision are visually represented just as they occur.
- The results achieved an acceptable interactivity level, considering the time step constraints discussed at the end of Section 3. The rendering by raycasting did not penalize the execution time in a relevant manner, so the appeal obtained in the visual aspect is worth it.

Using a volume as an intermediate data between the fluid simulation and the rendering decoupled the logic between the two steps. This decoupling helped in coding many tests with different parameters, allowing the developer to deal only with local updates in the prototypes' source code.

In every test that the simulation was performed in the CPU, the rendering was not the most time consuming task. In fact the simulation took around 60% or more. Analyzing the results of the raycasting experiments, it can be seen that in the 1024 particles experiment the CPU version took less time than the GPU in the experiments with 1024 and 2048 particles. However, we can observe that in the following experiments the opposite occurred, with the GPU version executing a time step in less than one third of the time of the CPU version in the 8192 particles' experiment.

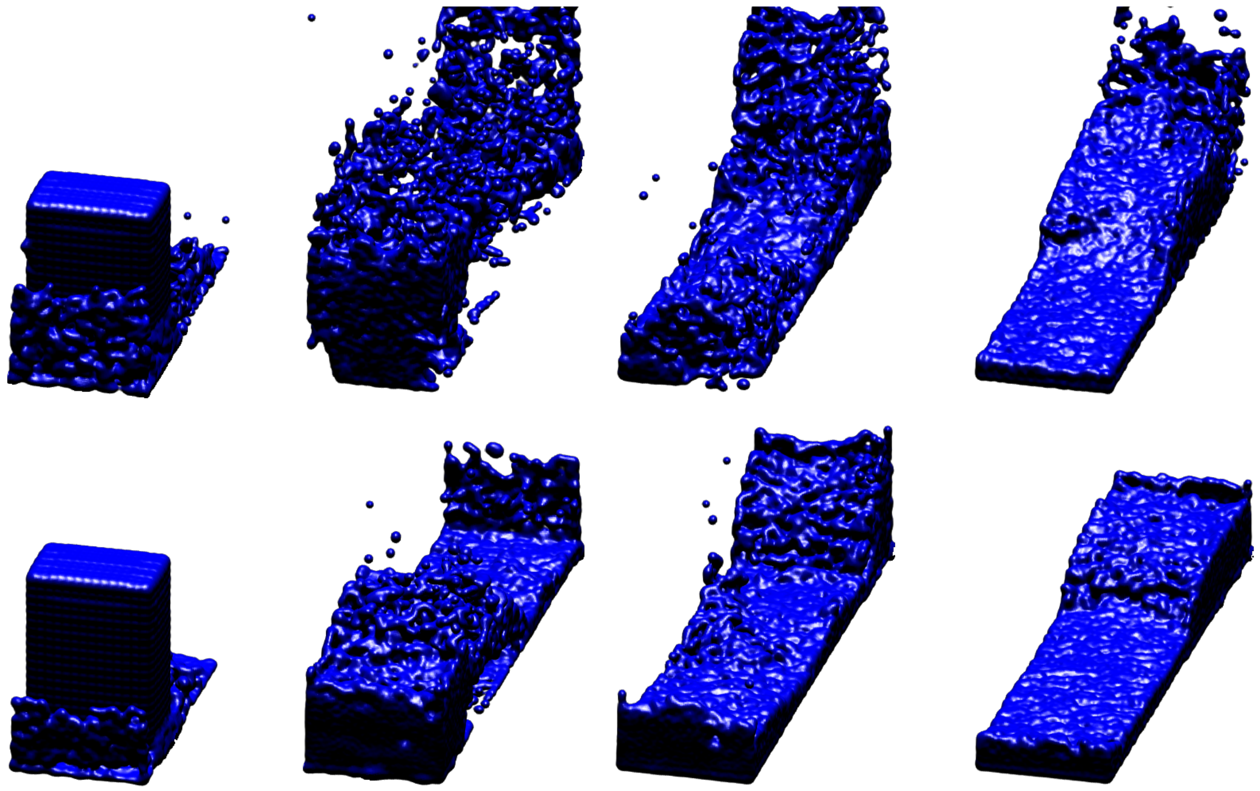

Figure 7. Comparison between one step of fluid simulation with $\Delta t = 0.01s$ (top row) and ten steps with $\Delta t = 0.001s$ (bottom row).

The SPH method runs more efficiently in the GPU due to characteristics of this processor, which can perform the same instructions for different data inputs at the same time. The method is parallel by its own nature in which each particle's position can be calculated separately for a given simulation step. When more particles are applied on a simulation, this difference tends to grow. The raycasting algorithm runs efficiently in the GPU for the same reason, because every ray task can be calculated in separate.

The experiments' pictures demonstrate the difference in the plasticity obtained through the raycasting algorithm. The fluid has a more appealing aspect in comparison with the spheres representation, as can be observed in Figure 9, even if we compare the 8192 spheres representation with the 1024 particles simulation using raycasting. If a simulation application requires more accuracy, it makes sense applying more particles to represent the fluid motion. More particles means that the fluid is more subdivided for the simulation. On the other hand if the application claims for visual effect, e.g. video games, the developer could adapt it in a fluid simulation with less particles (less memory consuming solution) to achieve a more interesting visual result. The transfer function of the raycaster also affects the visual quality as can be observed in Figure 8, where the fluid was rendered with lower opacity on the surface.

For future works we intend to simulate fluid's contact with more complexes solid structures and to take advantage of SPH's inner characteristic in providing the variation in physical quantities during its own calculations[17]. We also aim to take advantage on the raycasting transfer function to codify through colors the fluid's physical condition such as pressure and velocity that are not observed just in the surface visualization.

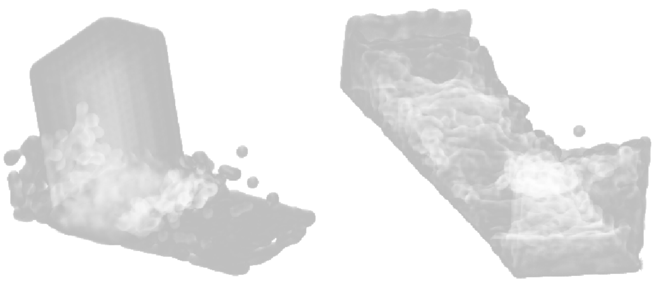

Figure 8. Raycasting the fluid surface by using a translucent transfer function.

Finally, we could try to apply the voxelization of the model on solids and tissues' simulation in real time, considering the proper adaptations. The raycasting algorithm provides a relevant enhancement of the visual characteristics and since it presents a regular behavior representing volatile objects like fluid's particles, it seems reasonable to think about adapting it to models that do not suffer of such unpredictability regarding their spatial parameters.

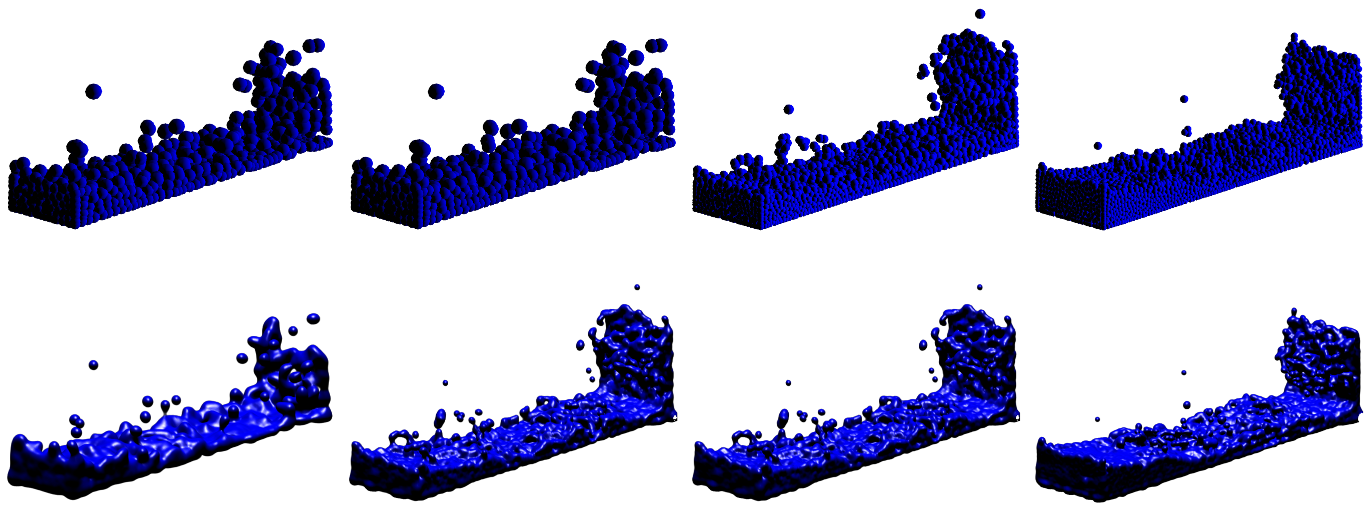

Figure 9. From the left to the right, surface renderings of the same simulation step using 1024, 2048, 4096 and 8192 particles.

References

[1] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics - theory and application to non-spherical stars," *Mon. Not. Roy. Astron. Soc.*, vol. 181, pp. 375–389, November 1977. [Online]. Available: http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=1977MNRAS.181..375G

[2] M. Desbrun and M.-P. Gascuel, "Smoothed particles: a new paradigm for animating highly deformable bodies," in *Proceedings of the Eurographics workshop on Computer animation and simulation '96*. New York, NY, USA: Springer-Verlag New York, Inc., 1996, pp. 61–76.

[3] D. Stora, P.-O. Agliati, M.-P. Cani, F. Neyret, and J.-D. Gascuel, "Animating lava flows," in *Graphics Interface*, Jun 1999, pp. 203–210. [Online]. Available: http://www-evasion.imag.fr/Publications/1999/SACNG99

[4] A. Paiva, F. Petronetto, T. Lewiner, and G. Tavares, "Particle-based viscoplastic fluid/solid simulation," *Computer-Aided Design*, vol. 41, no. 4, pp. 306–314, april 2009.

[5] S. Clavet, P. Beaudoin, and P. Poulin, "Particle-based viscoelastic fluid simulation," in *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. New York, NY, USA: ACM, 2005, pp. 219–228.

[6] A. Paiva, F. Petronetto, T. Lewiner, and G. Tavares, "Particle-based non-newtonian fluid animation for melting objects," in *19th Brazilian Symposium on Computer Graphics and Image Processing*, Manaus, AM, October 2006, pp. 78–85. [Online]. Available: http://www.matmidia.mat.puc-rio.br/tomlew/attachments/melt_sibgrapi.pdf

[7] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 154–159.

[8] T. Amada, M. Imura, Y. Yasumuro, Y. Manabe, and K. Chihara, "Particle-based fluid simulation on gpu," *ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH 2004 Poster Session*, 2004.

[9] T. Harada, S. Koshizuka, and Y. Kawaguchi, "Smoothed particle hydrodynamics on gpus," in *Computer Graphics International*, 2007, pp. 63–70.

[10] ——, "Sliced data structure for particle-based simulations on gpus," in *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. New York, NY, USA: ACM, 2007, pp. 55–62.

[11] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, vol. 21, no. 4. New York, NY, USA: ACM Press, July 1987, pp. 163–169. [Online]. Available: http://dx.doi.org/10.1145/37401.37422

[12] K. Iwasaki, F. Yoshimoto, Y. Dobashi, and T. Nishita, "Real-time rendering of point-based water surfaces," in *Proceedings of Computer Graphics International 2006*, 2006, pp. 102–114.

[13] M. Levoy, "Display of surfaces from volume data," *Computer Graphics and Applications, IEEE*, vol. 8, no. 3, pp. 29–37, May 1988. [Online]. Available: http://dx.doi.org/10.1109/38.511

[14] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1988, pp. 65–74. [Online]. Available: http://dx.doi.org/10.1145/54852.378484

[15] J. Kruger and R. Westermann, "Acceleration techniques for gpu-based volume rendering," in *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. Washington, DC, USA: IEEE Computer Society, 2003. [Online]. Available: http://dx.doi.org/10.1109/VIS.2003.10001

[16] H. Scharsach, "Advanced gpu raycasting," in *Proceedings of the 9th Central European Seminar on Computer Graphics*, May 2005. [Online]. Available: http://www.cg.tuwien.ac.at/hostings/cescg/CESCG-2005/papers/VRVis-Scharsach-Henning.pdf

[17] T. Nobrega, D. D. B. Carvalho, and A. von Wangenheim, "Simplified simulation and visualization of tubular flows with approximate centerline generation," in *Proceedings of the 22nd IEEE Symposium on Computer-Based Medical Systems (CBMS)*, 2009.