Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/070429

Real-time Subsurface Scattering for Particle-based fluids using Finite Volume Method

*Kyung-Kyu Kang and Dongho Kim**

Department of Digital Media, Soongsil University, Seoul, Korea

Received: 8 Dec. 2012, Revised: 17 Jan. 2013, Accepted: 20 Feb. 2013
Published online: 1 Jul. 2013

Abstract: We present a real-time subsurface scattering simulation to perform real-time rendering of translucent particle-based fluids. After particle-based fluid simulation, we immediately build voxelized fluids, called *Voronoi fluids*, with particle locations and neighbour lists using GPUs. And then, we perform a multiple subsurface scattering simulation over the Voronoi fluids with the diffusion equation (DE). We employ Finite Volume Methods to solve DE efficiently and rapidly with the voxelized fluids. In our implementation, DE is solved on GPUs with the diffusive source boundary condition and query function for computing the outgoing radiance. We demonstrate a milk demo that show our method can be performed in real-time with CUDA-based fluid simulation.

Keywords: Subsurface scattering, real-time rendering, particle-based fluids, diffusion equation, finite volume methods

1. Introduction

Subsurface scattering effects are important to improve the visual appearance in computer graphics. This natural phenomenon is a mechanism of light transport in a translucent object. When light photons are scattered by interacting with particles of material, we can observe this phenomenon occurred on the surface of translucent objects. However, scattering effects are complex to simulate for rendering images.

Real-time subsurface scattering simulation and rendering are still challenging works for interactive application such as 3D games. Many researchers have proposed real-time subsurface scattering rendering techniques for translucent materials [1,2,3]. These reserches are focused on rigid translucent objects, such as wax, candle, marble, jade, or human skin. And they need pre-processing time (maybe a few minutes) to generate inner mesh structures for radiance sampling. Some subsurface scattering rendering techniques for human skin are already available in famous game engines.

Recently, many video games employ particle-based fluids effects [4] which make games more realistic. They only render fluids' surface using simple methods, e.g. screen space rendering techniques [5]. The rendering results are a lack of optical phenomenon such as subsurface scattering. However, there are no available subsurface scattering rendering techniques for translucent fluids in real-time and without any inner mesh data.

We present a real-time subsurface scattering rendering method for particle-based fluids. Our method directly uses the particle's location as a radiant flux sampling point and performs a scattering simulation using the diffusion process [6]. The diffusion process describes isotropic multiple scattering effects in translucent materials by solving the diffusion equation. After the scattering simulation, we render the iso-surface of the fluids using screen space fluid rendering [5] with an outgoing radiance.

We voxelize particle-based fluids, called *Voronoi fluids*, with particle locations and neighbor lists using GPUs. This voxelization process is similar to the building process of the Voronoi diagram, and is parallelized easily to perform in real-time. For solving the diffusion equation over Voronoi fluids using Finite Volume Methods, we only need volumes of cells and areas of cells' surrounding faces from the voxelized fluids. So we can speed up the voxelization process with requiring simple data. In our implementaion, we use only nVIDIA CUDA [7] which performs this process in real-time.

This paper consists of the following contents: after the preview of the related works in chapter 2, we explain briefly on the diffusion equation in chapter 3, and derive the discrete diffusion equation and solve the equation. In

* Corresponding author e-mail: cg@su.ac.kr

chapter 4, our method is implemented with GPUs. Finally we discuss about the results and conclusion in chapter 6 and 7.

2. Related Work

2.1. Subsurface Scattering

Photorealistic subsurface scattering rendering can be performed with Monte-Carlo path tracing [8] or Photon Mapping [9]. They are physically accurate, but these methods usually require hours for rendering. Stam [10] introduces the diffusion equation to computer graphics for multiple scattering simulations for smoke and cloud. The diffusion equation is an approximation version of the volumetric radiative transfer equation with diffusion approximation [11]. Jensen and his colleagues [6] introduced a dipole method with the diffusion equation. They show realistic rendering results for homogeneous translucent materials, like skin and marble, in a few minutes. They also proposed how to acquire material's scattering properties. Arbree and his colleagues [12] improve the quality of results with the diffusive source boundary condition and the query function for computing outgoing radiance for diffusion process.

Recently many papers have presented algorithms for real-time rendering of translucent objects [1,2,3] with the diffusion process, but their goals are restricted only to rigid objects. They employ finite difference methods, finite element methods and finite volume methods to solve the diffusion equation within tetrahedral meshes as structured geometrical data. However, the building cost of the structure is expensive. So they are difficult to be applied on real-time applications of fluids. Screen space approaches [13,14] are also performed in real-time and show reasonable quality of results.

2.2. Particle-based fluids

Particle-based fluid simulation is a practical method for the representation of fluids in real-time rendering applications. Müller and his colleagues [4] introduced a real-time fluid simulation based on Smoothed Particle Hydrodynamics (SPH) to computer graphics community. After this research many researches employed the SPH-based methods for the real-time and off-line fluid simulations and brought enhancements and extensions. Recently, nVIDIA's particle demo in [15] shows that GPUs can handle massive particles using a GPU-based sorting library for constructing a partial hash table. nVIDIA PhysX [16] is a scalable game physics engine accelerated by multicore CPUs and GPUs for multi-platform interactive applications. This engine also provides real-time particle-based fluids. We obtain particle's locations from PhysX in our implementation.

Iso-surface rendering methods of particle-based fluids are also important. Müller and his colleagues [3] suggest an iso-surface extraction method using marching cubes [17] within a volumetric density field. This method becomes expensive in higher resolution of a grid and as increasing number of particles. Alternatively, screen space approaches use only locations of the camera-facing particles for speed-up and make 2.5D iso-surface meshes [18] without explicit meshing [5] which are suitable to interactive applications. We utilize the screen space fluids rendering technique [5] for our particle-based scattering simulation.

3. The Diffusion Theory

Subsurface scattering effects in translucent materials are complex to simulate accurately. The volumetric radiative transfer equation (VRTE) perfectly describes the scattering phenomenon in a randomly scattering medium [11]. This equation computes eventually the differential radiance $L(x, \omega)$ leaving at position x in direction ω. The radiance is evaluated with a scattering coefficient $\sigma(x)$, a absorption coefficient $\mu(x)$ and a phase function $p(\omega, \omega')$, the VRTE is following :

$$(\omega \cdot \nabla)L(x,\omega) = -\mu(x)L(x,\omega) + \sigma_s(x)\int_{4\pi} L(x,\omega')p(\omega',\omega)d\omega' + Q(x,\omega) \quad (1)$$

where $Q(x,\omega)$ is a light source function.

The diffusion equation is approximated version of VRTE by diffusion approximation defined by a simpler function with a near isotropic angular distribution: the scalar fluence $\phi(x)$ and the vector irradiance $E(x)$ [11,25]. The diffusion approximation is following,

$$L(x,\omega) \approx \frac{1}{4\pi}\phi(x) + \frac{3}{4\pi}E(x)\cdot\omega \quad (2)$$

where $\phi(x) = \int_{4\pi} L(x,\omega)d\omega$ and $E(x) = \int_{4\pi} L(x,\omega)\cdot\omega d\omega$.

The substitution of equation (2) into equation (1) yields the diffusion equation:

$$\nabla\cdot(\kappa(x)\nabla\phi(x)) - \mu(x)\phi(x) = Q(x) \quad (3)$$

where $\kappa(x) = [3((1-g)\sigma_s(x)+\mu(x))]^{-1}$ is a diffusion coefficient, g is an average cosine of scattering, and $Q(x)$ is an isotropic source function.

Our subsurface scattering simulation only supports highly scattering, non-emissive, heterogeneous particle -based fluids. With equation (3), we can remove the isotropic source function term $Q(x)$ and the equation defines the radiant flux $\phi(x)$ within fluids Ω,

$$\nabla\cdot(\kappa(x)\nabla\phi(x)) - \mu(x)\phi(x) = 0, x \in \Omega. \quad (4)$$

On surface $\partial\Omega$ of fluids we use the diffusive source boundary condition [12] given by

$$\phi(x) + 2A\kappa(x)(\nabla\phi(x)\cdot n(x)) = \frac{4}{1-F_{dr}}q(x), x \in \partial\Omega \quad (5)$$

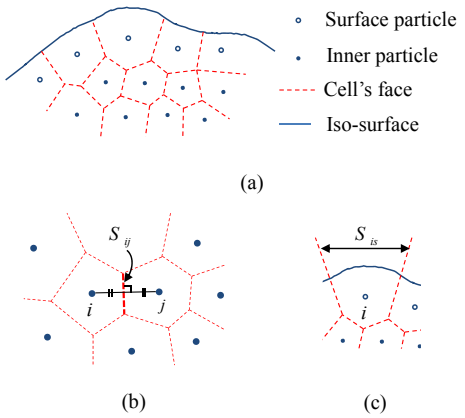


Figure 1: Illustration of Voronoi fluids and its cells with particle's locations. Particle-centered volumes, called cells, fill the inside of the fluid (a). Inner cells are made by a similar way to a Voronoi diagram (b), and Surface cells are located beneath the iso-surface (c).

where $q(x) = \int_{2\pi} L_i(x,\omega_i)(n(x)\cdot\omega_i)F_t(\omega_i)\,d\omega_i$ is the diffused incoming light at surface x. F_{dr} is the diffuse Fresnel reflectance and F_t is the Fresnel transmittance [6]. $A = (1+F_{dr})/(1-F_{dr})$ is a Fresnel boundary term and $n(x)$ is the surface normal at x.

The outgoing radiance on surface can be computed with the query function as described by [12] :

$$L_o(x,\omega_o) = \frac{F_t(\omega_o)}{4\pi}\left[\left(1+\frac{1}{A}\right)\phi(x) - \frac{4}{1+F_{dr}}q(x)\right], x\in\partial\Omega \quad (6)$$

where ω_o is the outgoing direction from the surface x.

4. Diffusion Equation Solver for Particle-based Fluids

We solve the diffusion equation, the boundary condition and the query function for the outgoing radiance in a discrete volumetric domain. We use voxelized fluids called *Voronoi fluids* with non-uniform volumes, *cells*, which are similar with Voronoi diagram [19]. The cell-based structure as the discrete domain can be built with particles' geometrical data. Each cell x has its own optical properties, the diffusion coefficient $\kappa(x)$ and the absorption coefficient $\mu(x)$, and store the radiant fluence $\phi(x)$ after solving the diffusion equation.

4.1. Voronoi Fluids

In Figure 1(a), we illustrate Voronoi fluids and cells made with particle's radius and location from fluids. These looks similar to Voronoi diagram and Voronoi cells except that it has the iso-surface of fluids as boundary. Cells beneath the iso-surface are called *surface cells*, others are called *inner cells*. Cells have their *neighbor faces* between neighbor cells as shown in Figure 1(b). The surface cells have one *surface face* that is boundary of the cell on the iso-surface cells as shown in Figure 1(c). We can evaluate cells' volume V and faces' area S from this structure.

4.2. Discretized Diffusion Equation

We solve the diffusion equation by discretizing the divergence operator $\nabla\cdot$ and the gradient operator ∇ in equation (4) using Finite Volume Methods [3,24]. First we derive an integration version of Equation (4) over a cell as scattering domain:

$$\int_{v_i}\nabla\cdot(\kappa(x)\nabla\phi(x))\,dv - \int_{v_i}\mu(x)\phi(x)\,dv = 0 \quad (7)$$

where V_i is volume of the cell x_i. We integrate the first term to get the volume average and can apply the divergence theorem to equation (7), this yields following:

$$\oint_{s_i}\kappa(s)(\nabla\phi(s)\cdot n(s))\,ds - V_i\mu(x_i)\phi(x_i) = 0, \quad (8)$$

where s_i is all faces of the cell i and $n(s)$ is a normal vector on the face. Equation (8) can be reformed by dividing the cell's volume,

$$\frac{1}{V_i}\oint_{s_i}\kappa(s)(\nabla\phi(s)\cdot n(s))\,ds - \mu(x_i)\phi(x_i) = 0. \quad (9)$$

In Equation (9), the gradient of the radiant flux on faces is approximated with face's area and difference between neighbors' flux,

$$\nabla\phi(s)\cdot n(s) \approx S_{ij}(\phi(x_j) - \phi(x_i)) \quad (10)$$

where S_{ij} is the face's area, cell j is the neighbor of cell i.

By substitution Equation (10) into Equation (9), we can get the final version of a discretized diffusion equation as following:

$$\frac{1}{V_i}\sum_j^n S_{ij}(\kappa(x_j)\phi(x_j) - \kappa(x_i)\phi(x_i)) - \mu(x_i)\phi(x_i) = 0. \quad (11)$$

The discretized diffusion equation is performed in inner cells for computing its radiance. However, in surface cells, the diffusion with neighbor cells and the boundary condition with incoming light on the iso-surface at a same time [3]. We compute the diffusive source boundary condition as inserting a term S into Equation (9)

$$\frac{1}{V_i}(I+S) - \mu(x_i)\phi(x_i) = 0, \quad (12)$$

where $I = \oint_{s_{in}}\kappa(s)(\nabla\phi(s)\cdot n(s))\,ds$ is a sum of diffused flux through neighbor faces as equation (9) and it can be

discretized in Equation (11). S is same as I but through the surface face:

$$S = \oint_{s_{is}} \kappa(s)\left(\nabla\phi(s)\cdot n(s)\right) ds, \tag{13}$$

where s_{is} is the surface face of the surface cell i. Incoming flux on the surface are evaluated with the diffusive source boundary condition (Equation (5)) as following:

$$S = \oint_{s_{is}} \frac{1}{2A}\left(-\phi(x) + \frac{4}{1-F_{dr}} q(x)\right) ds, \tag{14}$$

and its discretized version :

$$S = \frac{1}{2A}\left(-\phi(x_i) S_{is} + \frac{4}{1-F_{dr}} q(s_{is})\right), \tag{15}$$

where S_{is} is the surface face's area and $q(s_{is})$ is total diffused incoming lights on the surface face of cell i.

We use the relaxation scheme as in [1,2,10] to solve Equation (11) and Equation (12) on the Voronoi fluids. We start by initializing the value of the flux at cells to zero, and iterate the following equations until reach a user defined number of iterations.

Inner cells:

$$\phi_{t+1}(x_i) = \frac{\sum_j^n (1/S_{ij})\kappa(x_j)\phi_t(x_j)}{w_i\kappa(x_i) + \mu(x_i)\sqrt[3]{V_i}}, w_i = \sum_j^n 1/S_{ij} \tag{16}$$

Surface cells:

$$\phi_{t+1}(x_i) = \frac{\sum_j^n (1/S_{ij})\kappa(x_j)\phi_t(x_j) + \frac{4}{1-F_{dr}} q(s_{is})/2A}{w_i\kappa(x_i) + \mu(x_i)\sqrt[3]{V_i} + \frac{S_{is}}{2A}} \tag{17}$$

After the radiant flux $\phi(x_i)$ at every cell is determined, and we can compute the outgoing radiance with Equation (6).

5. Implementation

We implemented our real-time scattering algorithm with CUDA [7] and OpenGL Shader Language (GLSL) [20]. Our test system has Intel i7-2600K CPU and two GPUs, nVIDIA geForce GTX 680, geForce GTX 580. The geForce GTX 580 is dedicated to a CUDA-based fluids simulator and the geForce GTX 680 used for CUDA and GLSL. Our subsurface scattering rendering stars after fluids simulation. We use nVIDIA's PhysX [16] as a particle-based fluids simulator. After fluids simulation, particle's locations are transferred to OpenGL's Vertex Buffer Objects.

Our real-time scattering system consists of the following process (see Figure 2). After obtaining particle locations from the fluids simulator, we build the Voronoi fluids and compute the diffused incoming light ($q(s_i)$). Cells' volume (V_i), faces' area (S_{ij} and S_{is}), and the incoming light as input, and then, we solve the diffusion

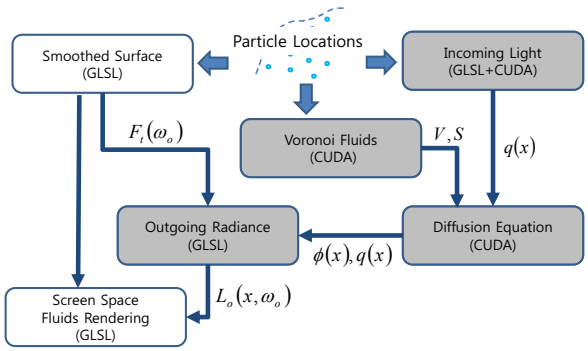


Figure 2: Overview of our subsurface scattering method. Gray boxes are our proposed process, and white boxes are the fluids rendering process based on the screen space fluids rendering [5].

equation (Equation 16, 17) with a few iterations. When the surface rendering time, we get the Fresnel transmittance ($F_t(x)$) on surface x from the rendering process and compute the outgoing radiance ($L_o(x,\omega_o)$ in Equation (6)) with an eye position.

5.1. Building Voronoi Fluids

The Voronoi fluids look similar to the 3D Voronoi diagrams, and a building process also similar to its. However, the building high quality 3D Voronoi diagrams takes a few seconds [21]. Since we need only volumes of cells and areas of faces in the Voronoi fluids, we use a simple method to compute those geometrical data for a real-time performance.

We, first, make a neighbor particle list on a particle to build the particle-centered cell's surround planes. We collect particles located nearer than the threshold distance from the particle using the CUDA-based hashing technique [15]. Then, we can get surrounding planes at cell i defined with a middle point between a neighbor particle's location and the particle i's location, and a normal vector pointing toward the neighbor. The surrounding planes are used for computing cell's volume and face's area.

We use a simple heuristic approach to build Voronoi fluids which start to make uniform samples within a bounding sphere of the cell (see Figure 3). And then, we do a *cut-off process* which is cutting off samples if they are located outside of the surrounding planes. When we make samples for neighbor faces, we use a 2D rectangle on surrounding plane instead of the bounding sphere. The number of live simples represents actual volume or area. After computation of the face's area, if the area is zero that face is removed from the list of the neighbor faces of the cell.

Computation method the surface area on the surface cells is similar to the cell's volume and face's area case. We

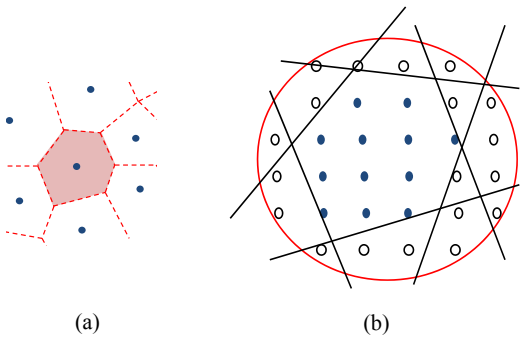


Figure 3: Illustration of evaluation a volume of a cell (a) and the cut-off process (b). A cell has six neighbor particles (a), and six surround planes (black lines in (b)). The number of live samples (blue solid dots in (b)) represents the approximated volume of the cell after the cut-off process with surround planes. The red circle in (b) represents a bounding sphere.

make samples on the surface on a particle-centered sphere, and do the cutting off process. And, we count live samples and can get an approximated area.

This approach is simple to implement with CUDA, but the performance is totally dependent on the number of initial samples. We use 64 samples for volume and surface area processes, and 36 samples for the face area process.

5.2. Diffusion Scattering Process

The diffused incoming light $q(s_i)$ is evaluated with a light-view depth map including particle IDs. We count how many pixels cover the surface cell from the map, and translate the number to the incoming light. Using Voronoi fluids V, S and the diffused incoming light $q(s_i)$, the diffusion equation can be computed with the Equation (16), (17). The light-view depth map is made by GLSL, the diffusion equation is computed on CUDA.

After a few iterations, we can get the final radiant fluence $\phi(x_i)$ on every cell. The fluxes are stored in VBO of cells, and used as input of the outgoing radiance shader. This shader renders spheres at cell's positions with cell's flux encoded to its color, and computes the outgoing radiance with surface normal from a smoothed surface map. Finally, the outgoing radiance is stored as texture for the final rendering step.

5.3. Rendering

We employ a screen space fluids rendering technique with a blurred depth map using GPUs [5]. This technique is the fastest and easy to implement, so it is suitable with interactive applications like games. We run only essential steps of this technique for real-time rendering. The surface is illuminated based on BRDF [22] with just adding the outgoing radiance.

6. Results

We simulate milk-like fluids for testing our subsurface scattering method. As seen in Figure 4 (a), we can generate visually feasible results using our rendering technique in real-time with an outgoing radiance map (Figure 5 bottom-right in (b)) and a BRDF rendered image (Figure 5 bottom-left in (b)). The outgoing radiance map is blurred for removing an artificial sharp boundary between cells.

Figure 5 shows our subsurface scattering rendering results of a milk demo and compares results with subsurface scattering effects (Figure 5 (a)) and without them (Figure 5 (b)). The milk demo was performed at about 24 FPS with 15,625 particles including the fluid simulation. Table 1 also shows the statistics of this demo scene. The building time of Voronoi fluids, *Finding*

Number of particles	Finding neighbors	Volume and area	Diffusion process	Rendering	Avg. FPS
15,625	10ms	32ms	2ms	2ms	24
31,250	11ms	64ms	2ms	2ms	13
46,875	17ms	92ms	2ms	2ms	10

Table 1: Computation time. The average FPS includes a fluid simulation with 30 diffusion iterations.

Number of Particles	Screen Resolution	VBO (KB)	FBO (KB)	CUDA (KB)	Total (MB)
15,625	1200x1024	1,464	110,400	24,671	133.3
	800x600	1,464	43,125	24,671	67.6
31,250	1200x1024	2,929	110,400	49,085	158.6
	800x600	2,929	43,125	49,085	92.9
46,875	1200x1024	4,394	110,400	73,499	183.8
	800x600	4,394	43,125	73,499	118.1

Table 2: Memory usage.

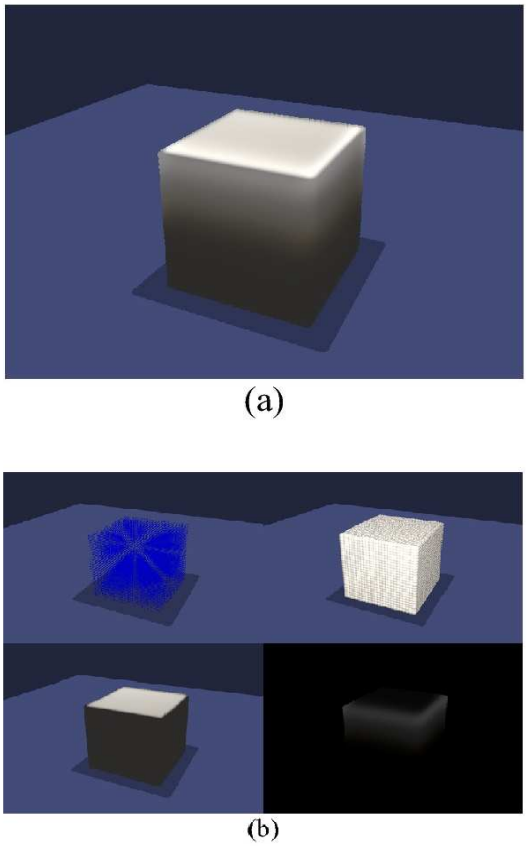


Figure 4: A final image of our rendering method (a), particle locations as blue points (top-left in (b)) and their spheres (top-right in (b), a shaded smoothed-depth map (bottom-left in (b)), and an outgoing radiance map (bottom-right in (b)).

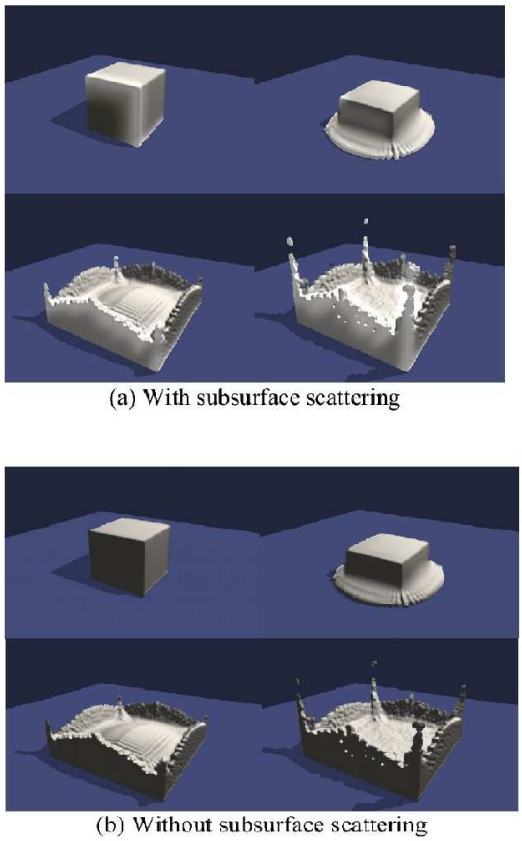


Figure 5: Milk demo with 15,625 particles at 24 FPS. Sequences with subsurface scattering effects (a) and same sequences without them (b).

neighbors and *Volume and area* columns in Table 1, is a bottleneck, with about 90% of total computation of our whole rendering process.

Our method uses various types of main memory, Vertex Buffer Objects (VBO), Frame Buffer Objects (FBO), and arrays for CUDA (*CUDA* column in Table 2). Since they are allocated on the GPU's memory space, data of those memory types can be shared by the CUDA-OpenGL interoperability. Table 2 shows the VBO and CUDA usage depend on the number of particles, and the FBO usage depends on the screen resolution.

7. Conclusion

We have presented the real-time subsurface scattering rendering technique for particle-based fluids with view-dependent diffusion approach. Our whole process is computed with CUDA and GLSL on the GPU in real-time. Our method is suitable for real-time application like video games. Our whole scattering steps are run on GPUs in real-time, so we can also apply our technique to any interactive applications containing fluids effects.

Previous diffusion methods have been accelerated by using various geometrical hierarchy structures. Most of them are built in pre-processing time, because of the costly building time. These acceleration methods are not suitable to real-time fluids system. We hope to find suitable techniques for our real-time system.

Our future works will include handling anisotropic scattering phenomenon in real-time, which can make translucent liquids more realistic. An anisotropic diffusion scattering rendering method [23] still is a challenging work in real-time rendering because of the large computing time. We also want to apply our algorithm to mesh-based objects. We hope that if any fluid simulator can fit into the objects with particles properly, we can use our scattering rendering method with those objects easily.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0015620).

References

[1] J. Wang, S. Zhao, X. Tong, S. Lin, Z. Lin, Y. Dong, B. Guo, H.-Y. Shum, Modeling and Rendering of Heterogeneous Translucent Materials using the Diffusion Equations, ACM Trans. Graph., **27(1)**, 9 (2008).

[2] Y. Wang, J. Wang, N. Holzschuch, K. Subr, J.-H. Yong, B. Guo, Real-time Rendering of Heterogeneous Translucent Objects with Arbitrary Shapes, Computer Graphics Forum **29(2)**, 497-506 (2010).

[3] D. Li, X. Sun, Z. Ren, S. Lin, Y. Tang, B. Guo, and K. Zhou, TransCut : Interactive Rendering of Translucent Cutouts, IEEE Transactions on Visualization and Computer Graphics **19(3)**, 484-494 (2013).

[4] M. Müller, D. Charypar, and M. Gross, Particle-based Fluid Simulation for Interactive Applications, In Proc. ACM SIGGRAPH/Eurographics Sypm. Comp. Anim. 2003, 154-159 (2003).

[5] W. J. van der Laan, S. Green, and M. Sainz, Screen Space Fluid Rendering with Curvature Flow, In I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, 91-98 (2009).

[6] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan, A Practical Model for Subsurface Light Transport, In Proc. ACM SIGGRAPH 2001, 511-518 (2001).

[7] NVIDIA, CUDA 4.2, http://developer.nvidia.com/category/zone/cuda-zone, (2012)

[8] H. Li, F. Pellacini, K. E. Torrance, A Hybrid Monte Carlo Method for Accurate and Efficient Subsurface Scattering, In Proc. Rendering Techniques 2005 (Eurographics Symposium on Rendering), 283-290 (2005).

[9] H. W. Jensen, P. Christensen, Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps, In Proc. ACM SIGGRAPH 1998, 311-320 (1998).

[10] J. Stam, Multiple Scattering as a Diffusion Process, Eurographcis Rendering Workshop 1995, 41-50 (1995).

[11] A. Ishimaru, Wave Propagation and Scattering in Random Media, Academic Press (1978).

[12] A. Arbree, B. Walter, and K. Bala, Heterogeneous Subsurface Scattering using the Finite Element Method, IEEE Comp. Graph. Appl. **17(7)**, 956-969 (2011).

[13] K.-K. Kang, D. Kim, Screen Space Rendering of Translucent Liquids, The Journal of Future Game Technology **2(2)**, 159-164 (2012).

[14] A. Munoz, J. I. Echevarria, F. J. Seron, and D. Gutierrez, Convolution-Based Simulation of Homogeneous Subsurface Scattering, Computer Graphics Forum, **30(8)**, 2279-2287 (2011).

[15] NVIDIA, Particles Demo, GPU Computing SDK 4.2, http://developer.nvidia.com/cuda/ cuda-toolkit (2012).

[16] NVIDIA, PhysX 3.2, http://developer.nvidia.com/physx (2012).

[17] E. W. Lorensen, H. E. Cline, Marching cubes: A High Resolution 3D Surface Construction Algorithm, In Proc. ACM SIGGRAPH 1987, 163-169 (1987).

[18] M. Müller, S. Schirm, and S. Duthaler, Screen space meshes, In Proc. ACM SIGGRAPH/Eurographics Sypm. Comp. Anim 2007, 9-15 (2007).

[19] F. Aurenhammer, Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure, ACM Computing Surveys **23(3)**, 345-405 (1991).

[20] OpenGL Shading Language, http://www.opengl.org/documentation/glsl/ (2012).

[21] D.-M. Yan, W. Wang, B. Lévy, and Y. Liu, Efficient Computation of 3D Clipped Voronoi Diagram, In GMP'10 Proccedings of the 6th international conference on Advances in Geometric Modeling and Processing, 269-282 (2010).

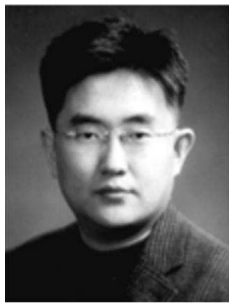
[22] R. L. Cook, K. E. Torrance, A Reflectance Model for Computer Graphics, Comupter Graphics (SIGGRAPH '81) **15(3)**, 1307-316 (1981).

[23] W. Jakob, A. Arbree, J. T. Moon, K. Bala, and S. Marschner, A Radiative Rransfer Framework for Rendering Materials with Anisotropic Structure, ACM Trans. Graph. **29(4)**, 53 (2010).

[24] R. Eymard, T. R. Gallouet, R. Herbin, The Finite Volume Method Handbook of Numerical Analysis, Vol. VII, 713-1020 (2000).

[25] W. Jakob, A. Arbree, J. T., Moon, K. Bala, and S. Marschner, Expanded Technical Report of A Radiative Transfer Framework for Rendering Materials with Anisotropic Structure, Cornell University, http://www.cs.cornell.edu/projects/diffusion-sg10/diffusion-sg10-tr.pdf, (2010).

Kyung-Kyu Kang is a Ph.D. candidate student at MAGIC Lab., in the Department of Digital Media at Soongsil University, Seoul, Korea. He received his Master degree in Media Engineering at Soongsil University in 2006. His interests include real-time rendering algorithms and physically-based simulation.



Dongho Kim is a professor in the Department of Digital Media at Soongsil University, Seoul, Korea. He obtained B.Sc. degree from Seoul National University and Master degree from KAIST, Korea. He received Ph.D. degree in Computer Science at the George Washington University in 2002. His research interests include real-time rendering, animation, game engineering, digital contents, and media art.