Parallel-optimizing SPH fluid simulation for realistic VR environments

Chen Huang1*, Jian Zhu2, Hanqiu Sun1* and Enhua Wu2,3

1 Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong
2 Computer Graphics and Multimedia Laboratory, Faculty of Science and Technology, University of Macau, Macau, China
3 State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

ABSTRACT

In virtual environments, real-time simulation and rendering of dynamic fluids have always been the pursuit for virtual reality research. In this paper, we present a real-time framework for realistic fluid simulation and rendering on graphics processing unit. Because of the high demand for interactive fluids with larger particle set, the computational need is becoming higher. The proposed framework can effectively reduce the computational burden through avoiding the computation in inactive areas, where many particles with similar properties and low local pressure cluster together. While in active areas, the computation is fully carried out; thus, the fluid dynamics are largely preserved. Here, a robust particle classification technique is introduced to classify particles into either active or inactive. The test results have shown that the technique improves the time performance of fluid simulation largely. We then incorporate parallel surface reconstruction technique using marching cubes to extract the surfaces of the fluid. The introduced histogram pyramid-based marching cubes technique is fast and memory efficiency. As a result, we are able to produce plausible and interactive fluids with the proposed framework for large-scale virtual environments. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS

active/inactive particle; neighbor search; SPH fluids; parallel HPMC; GPU; virtual environments

*Correspondence
Chen Huang; Hanqiu Sun, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong.
E-mail: hcmorning@hotmail.com; hanqiu@cse.cuhk.edu.hk

1. INTRODUCTION

Dynamic fluids are closely related to the nature world, examples are water, waves, fire, and gas. Physically plausible dynamic fluids simulation continues to be a hot research topic in computer graphics/virtual reality (VR) community. There are high demands for realistic fluid simulations in science fiction movies, three-dimensional (3D) computer games and interactive virtual environments. The animation of turbulent fluids can be successfully realized with either Eulerian or Lagrangian approach. Our paper focuses on the Lagrangian particle-based fluid solver. Smoothed-particle hydrodynamics (SPH), a Lagrangian approach, is relatively new for simulating fluids. Compared with the other physics-based simulation method, SPH shows great capability for simulating highly nonlinear and breaking fluids with small-scale features, as well as large-scale virtual scenes.

With the high demand for real-time application with large particle set, improving the efficiency of simulation becomes a really vital issue. As SPH needs a large number of particles for fine-detail fluids, the computational burden is very high when we need high-resolution flows and large scenes. In order to accelerate the animation and enable real-time application with a large particle set, researchers have contributed extensively in the literature. But, it remains an open problem. The bottleneck we face today is the huge gap between the massive particles and the computing units (CPU or graphics processing unit (GPU) thread processor) that we have. Even with the latest graphics card and the most up-to-date Compute Unified Device Architecture (CUDA) programming language with multithread technology, we can only execute several thousands of particles at one time. The other particles have to wait in the queue for computing vacancies for execution. The current parallel computing is logical parallel, and we still need hundreds or thousands of turns to finish one simulation step. Thus, increasing the turnover rate of each computing unit can speed up the whole simulation process.
Traditionally, each particle has to interact with all of its neighbors in each computational step. We should find a way to decrease the computation needed to increase the turnover rate of the computing units.

Visualizing the particle data is another challenge for computer graphics/VR researchers. It is crucial to have an efficient and high-quality surface rendering algorithm for real-time applications. Marching cubes (MC) [1] is a polygonization technique and is widely used for fluid surface extraction in the graphics society. The traditional MC is CPU-based and is very time-consuming in surface extraction. In the context of MC, the cell size significantly affects the quality of the rendering result and the computational time. Smaller cell size results in longer computation time and larger memory consumption. If higher resolution rendering results are needed, smaller grid cell size must be applied. Thus, a parallel and memory efficient MC is needed if we want to transfer the methodology for interactive applications.

In this paper, we focus on the interactive simulation and rendering of SPH fluids for virtual environments. We introduce a framework to implement the SPH simulation and rendering fully on GPU. Our framework can exploit the full computing power of the modern GPU and mitigate the computational needs of each simulation step in contrast to the previous methods. The framework can also produce realistic results using MC with small cell size. The main contributions of the work are summarized as follows:

- An activity-based particle property reuse algorithm is proposed for large-scale SPH. The algorithm can effectively decrease the computational burden of GPU and at the same time maintain the dynamics of the fluids.
- A parallel MC based on histogram pyramid (HPMC) is employed to polygonize the fluid surface with speed.
- A framework for SPH simulation and rendering for interactive VR environment is developed to be fully parallel and can be implemented on the latest GPU.

For the rest of the paper, we first review the related work in dynamic fluid simulation in Section 2. Following this, we present our GPU-based SPH fluid simulation including the proposed activity-based property reuse algorithm in Section 3. The surface reconstruction and rendering using GPU-based HPMC are presented in Section 4. We then have the experimental results and comparative discussions with other methods in Section 5. Finally, summary of the work goes to Section 6.

2. RELATED WORK

Fluid simulation has become a hot research topic since the publishing of papers [2–4]. Since then, researchers have endeavored to solve the Navier–Stokes equations that govern the motion of fluids for animation or engineering purpose. There are mainly two branches that are Eulerian and Lagrangian approaches on how to solve the governing equations in the computer graphics community. The Eulerian approach is a grid-based approach. It utilizes grid cells for storing fluid properties and computes the quantity change with time [4–7]. The Lagrangian approach is particle-based and solves the problem in a different way [8–10]. The fluid properties are stored in particles, and the governing equations are solved through tracking the motion of each particle.

2.1. Smoothed-particle Hydrodynamics Modeling

Smoothed-particle hydrodynamics is the most popular particle-based method for simulating fluids using numerical interpolation. It uses particle to store the properties such as density, velocity, position, and other attributes of the fluid. SPH has become an alternative to the Eulerian grid approach in the past decade. It was first introduced into the computer graphics community by Monaghan [8]. Müller et al. [11] popularized the method through deriving the interparticle forces from the Navier–Stokes equations in an interactive manner. However, it only works under very small time steps in order to maintain its incompressibility. Becker and Teschner [12] made the method weakly compressible by introducing the Tait equation. Later, Solenthaler and Pajarola [13] proposed the predictive-corrective incompressible SPH, which corrected the pressure term iteratively to make it incompressible with larger time step. The introduction of the adaptively sampled method with novel surface reconstructing by [9] improved the simulation speed through introducing adaptive particle sizes while still preserving the visual quality. Solenthaler and Gross [14] divided the simulation space into two parts and allocated more computational resources to a region where complex flows emerge. More visual effects such as fluid/object interactions, surface tension effects could be produced by SPH method, examples are [15–17].

2.2. Graphics Processing Unit Dynamic Simulation

The growth of the computational power of GPU is tremendous, and the graphics cards with the latest GPU are more affordable than in the past. An overview of fluid simulations on GPU could be found in review papers [18,19]. Lots of papers are published to exploit the computing power of GPU for SPH implementations. Amada et al. [20] used the GPU as an addition to CPU for the acceleration as the neighboring particle searching was carried out on the CPU, and the SPH physics attributes computation was conducted on GPU. The method suffered from frequent CPU–GPU data transfer and time-consuming neighbor list production on CPU. Kolb et al. [21] implemented SPH entirely on the
GPU, but they introduced an interpolation error because physical values at particles were based on one single particle. Kipfer et al. [22,23] used a uniform grid and sorting mechanism for contact detections on GPU. Harada et al. [24] presented a grid-based structure that employed bucket textures to represent a 3D grid. Limitations of his system are that it can only handle up to four particles within a grid cell, and too much memory is allocated to the buckets. Bayraktar et al. [25] presented a neighbor searching technique on GPU for fluid simulation. The main drawback of the method is that it is not as fast as the similar sorting-based methods because of the expensive grid map generation phase. Goswami et al. [26] introduced the interactive SPH simulation and rendering method on the GPU. Their method is totally parallel, and Z-indexing was used for decreasing memory miss rate. But still, the method did not exploit the full potential of the GPU computing power when many particles individually have many potential neighbors. Rustico et al. [27] further extended the SPH method to multi-GPU systems. But, the equipment expenditures were higher.

### 2.3. Fluid Rendering

Researchers have been exploring efficient and high-quality fluid rendering because of the high demand for rich visual experiences. For fast surface rendering techniques, metaball [28,29] was introduced as a very popular surface representing tool. Particles can be rendered at interactive rate using metaball on the GPU. Screen space-based rendering [30,31] were developed in recent years. The rendering speed is fast as we limit the computation to the visible area. But, blobby effects appear on fluid surfaces. GPU-based methods [29,32] are also introduced for fast rendering. For high-quality fluid rendering, MC [1,33] is an ideal tool for high-resolution rendering. But with smaller MC, more computing resource is needed. Advanced surface construction methods are introduced for better visual effects while using a small number of particles. Fraedrich et al. [34] proposed a high-quality volume rendering method for SPH data. It employs 3D texture-based volume rendering on the GPU to achieve efficient ray casting and enables high-quality rendering results. With the idea of refining a coarse simulation for high-quality rendering, researchers proposed to sample the simulation particles with a finer set of pseudorandom particles [35]. With a specially designed refining method, flickering could be eliminated [36].

## 3. Parallel-Optimizing Smoothed-Particle Hydrodynamics Simulation

As stated in the previous texts, SPH is a naturally parallel and computational intensive method. In order to accelerate the simulation, we must design a strategy to bring the whole computation onto the GPU. In our system, we develop the SPH fluid simulation fully on GPU. The main simulation loop on the GPU contains three steps, namely, neighbor data construction, density and force computations in which our proposed algorithm is applied, and time integration. The framework is illustrated in Figure 1.
3.1. Smoothed-particle Hydrodynamics Modeling

Smoothed-particle hydrodynamics defines the computing of a smooth continuous field \( A(r) \) from the discrete property \( A \) sampled at particle location \( r \) as

\[
A(r) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h)
\]

where \( r, m, \) and \( A \) are position, masses, and additional properties of the particle, respectively, and \( h \) is the smoothing length. \( W(r, h) \) is the kernel function that is typically a smooth, radial, symmetric, and normalized function with finite support. And, the equation uses \( j \) to iterate all the neighboring particles. For the standard simulation case, \( m \) and \( h \) are constant throughout the simulation. The gradient and Laplacian of the smoothed properties functions \( A(r) \) are computed by using the gradient and Laplacian of \( W \), respectively.

In SPH, the density of fluid is essential to compute pressure and forces. The density is usually used for computing \( \rho \) as expressed by Equation (2):

\[
\rho(r) = \sum_j m_j W(r - r_j, h)
\]

For the pressure term that is used in the force computations, we employ the Weakly Compressible Smoothed-particle Hydrodynamics (WCSPH) method [12]. The pressure and viscosity forces computing can be found in [8].

3.2. Neighbor Construction

Here, we demonstrate how the particles are processed for the followed steps. First, uniform grid method is employed for convenient particle storage and neighbor searching on GPU. This comes from the observation that one particle’s neighbors must reside in its 27 surrounding grid cells, if we have the grid cell size larger than the SPH finite support radius \( h \). The simulation space is discretized into a virtual uniform indexing grid in the X-axis, Y-axis, and Z-axis along each of the directions. Then, the Z-indexing technique [26] is employed to compute the Z-indices for each particle by accessing a lookuptable for fast lookup in parallel. After that, each particle is mapped to one grid cell. The Z-indexing step can be calculated in parallel on GPU.

Arrays are used to store the Z-indices of each particle and particle information after the uniform grid generation step. Then, we sort the arrays according to their Z-indices in ascending order. We have the efficient CUDA built-in radix sort for fast sorting. We also create two arrays with the name cellStart and cellEnd to specify the first and last position where the specific cell starts and ends in the sorted particle index array for each Z-indexed cell. The illustration of 2D SPH particle uniform grid generation and cell index sorting process is shown in Figure 2.

3.3. Activity-based Property Reuse

Although uniform grid method has improved the simulation performance significantly both on CPU and GPU, the computational burden of each thread processor is still very high for high-resolution simulation, especially for the common cases where large numbers of particles cluster together, such as deep water, sea waves, or slow water flows. The reason is that particles in inactive regions commonly have much more neighbors to interact than in active regions such as splashes, curly waves, as shown in Figure 3 for a 2D representation. For instance, particle 1 in red has more neighbors to interact than particle 2 in dark blue.

We propose an activity-based particle property reuse algorithm for large-scale simulations. From our experience, the activity level of a particle is mainly controlled by two aspects: pressure and neighbors. For incompressible or weakly compressible SPH fluids, each particle has a maximum neighbor number around a fixed number. This is due to the incompressible nature of fluids. If we discretize the domain, each grid cell contains particles close to a maximum capacity for incompressible fluids. We call this maximum capacity Cell-Particle-Number (CPN). First, particles with average particle number for each neighboring cell close to CPN are more likely to be inactive particles. This comes from the observation that active areas such as water surfaces, foams, and splashes all have a small particle number. So we conclude that...
If a particle is inactive, it is exempted from the time-consuming density and force computation step. If a particle is an active one, it should go through all the computations needed in that time step, and is assigned a negative value -1 indicating that it is an active particle. IC is crucial to control the inactive particles. 

IC has two types of value: (I) IC > 0 means the particle is an active particle in the previous time step, and we can reuse the properties for this time step; (II) IC < 0 means the particle is previously an active particle, or an inactive particle that has to correct its properties in this time step. We set the particle’s IC to IDLE to indicate its inactive feature, and correct its data by performing the full computation. We can see the activity-based property reuse technique in the simulation framework as illustrated in Figure 1.

Two main advantages are associated with this practical method. First, the computational burden of inactive particles decreases significantly; thus, the performance of the simulation improves. The reason is that inactive particles simply reuse the properties from the previous time step, and the time-consuming neighbor searching step is avoided. The speedup is obvious if many particles are classified as inactive particle. Second is the dynamics of the fluid preserve. Active areas have an ACT value larger than the threshold \(T\), and the property update are carried out as usual. So, no computational error is introduced there. And commonly, we are more interested in the active areas than the inactive ones. Thus, our method certainly does not affect the visual quality of the simulation result.

### 3.4. Density and Force Computations

To compute the density of each particle, we have to compute Equation (2). We use our activity-based property reuse technique in this process. From the equation, we know that we have to search for the neighbors and compute their contribution to the current particle. Because the simulation domain is divided into a 3D uniform grid, the neighbors of the current particle all lie in its immediate neighbor cells (see Figure 2 for a 2D representation). So, similarly, for our 3D SPH simulation, we only need to query 27 cells for each particle. If each particle has to interact with all neighbors, the computational need would be very high. Our technique decides which particles are inactive ones and reuses the data of this particle from its previous time step. Thus, we avoid the time-consuming neighbor searching for inactive particles. After we have the newly computed densities of each particle, pressure can be updated in parallel using the weakly compressive equation as in [12].

Then, each particle repeats the same procedure as the density computing in the previous texts for force computing. After that, we can obtain the acceleration of the particle at this time step and calculate the new position using an explicit integration scheme accordingly.
3.5. Compute Unified Device Architecture Programming

With regard to the hardware, the GPU creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps. And each warp is scheduled for execution according to a warp scheduler. Actually, there is only one warp in each block in execution concurrently. The other warps are called active warps and are waiting in the queue for execution. From this point of view, we learn that the threads in execution concurrently are still very limited. This also tells why we should reduce the computation needs of particles, which are executed on thread processors.

The latest CUDA language allows flexible array usage on GPU. Particle properties such as position, velocity, mass, and color field are all stored separately in their own array indexed by the particle index in the global memory on the GPU. The reason is that not all the properties are updated in the same time. It reduces data transfer time compared with storing all the properties in one single array using a particle structure. New arrays with particle index and cell index information are created for storing uniform grid and Z-index sorting results. The cellStart and cellEnd information as well are recorded in two separate arrays for fast neighbor searching. Constant parameters such as gravity, smoothing radius, smoothing kernel, and mass are stored in the constant memory on GPU for fast data access. In our development, we employ the Z-indexing technique [26]. Z-indexing decreases the cache miss rate and thus improves the time performance of simulation. The fluid simulation pipeline is illustrated in Algorithm 1. Our proposed activity-based property reuse technique (red color in Algorithm 1) is integrated into the simulation framework as described in the previous texts.

4. SURFACE RECONSTRUCTION AND RENDERING

4.1. Surface Definition

Our surface definition is based on the paper [8], where the surface is defined as an isosurface of a scalar field:

\[ C_s = \sum_j m_j \frac{1}{\rho_j} W(r - r_j, h) \]  

(4)

The field is called color field, and it is a normalized density field that smooth out the scalar value of 1 at each particle’s position. A properly chosen threshold value decides a smooth surface that coats the particles. The color field computation does not introduce extra neighbor searching needs as it can be integrated into the density computation step.

4.2. Marching Cubes based on Histogram Pyramid

Marching cubes is a common technique to visualize the scalar field. The original MC by [1] is CPU-based and very time-consuming to process the data set. Later, the data parallel nature of MC is exploited to reduce the processing time. The early GPU-based MC algorithms still needs CPU for preprocessing. The CPU step largely affects its time efficiency. The scan-based MC method is totally parallel, but its application is limited by its large memory consumption. Here, we employ a parallel HPMC [33] to accelerate the surface construction process. The method is totally parallel and memory efficiency.

Marching cubes is a naturally parallel technique, because each cell can be processed independently. The challenge comes from the racing problem when writing triangle information onto the memory. The early GPU implementations, such as scan-based methods, allocate the same amount of memory for each cell to store the possible triangle information. The processing is faster than the original CPU-based method, but more than 80% of the memory space is wasted. Because most of the cells are empty and not all the cells contain the same number of triangles. So a scheme should be proposed to efficiently store the triangle information onto memory in parallel. The scheme should: (i) decide the number of triangles produced, so the exact amount of memory can be allocated on GPU; and (ii) a unique index for each cell to store its triangle information in the memory.

Marching cubes is a common technique to visualize the scalar field. The original MC by [1] is CPU-based and very time-consuming to process the data set. Later, the data parallel nature of MC is exploited to reduce the processing time. The early GPU-based MC algorithms still needs CPU for preprocessing. The CPU step largely affects its time efficiency. The scan-based MC method is totally parallel, but its application is limited by its large memory consumption. Here, we employ a parallel HPMC [33] to accelerate the surface construction process. The method is totally parallel and memory efficiency.

Marching cubes is a common technique to visualize the scalar field. The original MC by [1] is CPU-based and very time-consuming to process the data set. Later, the data parallel nature of MC is exploited to reduce the processing time. The early GPU-based MC algorithms still needs CPU for preprocessing. The CPU step largely affects its time efficiency. The scan-based MC method is totally parallel, but its application is limited by its large memory consumption. Here, we employ a parallel HPMC [33] to accelerate the surface construction process. The method is totally parallel and memory efficiency.
memory space on GPU with this information. Then, if we want to store the triangle information exactly into the right place, we can traverse the histogram pyramid from top level to base level to find the right place for storing the triangle information. The technique is quite suitable for MC parallelization, and we can see the demonstration in Figure 4.

4.3. Discussion

The proposed MC process consists of five main steps. First, we transfer the original SPH data onto the GPU memory and use a texture to store the base-level information. The base level stores the number of triangles each cell contains from the original data. The numbers are stored in the corresponding texel. Second, we compact the data using HPMC, and the results are stored in texture memory for fast access. Third, after the compact process is finished, we can allocate memory for the data set on GPU memory. Then, the triangle information is written to the memory using HPMC traversal as described in Section 4.2. Finally, the vertex and normal data from the previous steps can be used by OpenGL for rendering purpose.

In contrast to [33] that used shader to construct and build HP, we use the up-to-date CUDA parallelism. [33] faced the huge memory usage problem as the shader treats each HP level in the same way. As a result, each level has to use the same memory size as the base level. In our CUDA programming, we use the fast texture memory for HP data storage, and each layer is dealt with individually. Each level has sufficient memory that right fits its requirement. Thus, a faster surface construction speed is achieved, and the memory usage is largely reduced.

5. FLUID SIMULATION RESULTS

We have developed and tested both SPH simulation and rendering on a platform with the following hardware and software configuration: 2.0G Hz Dual-Core Intel E7350, 2.0GB main memory, and NVIDIA Geforce 9500M GS with 512M on-board memory; Windows 7 32bit, CUDA 4.2, OpenGL, and GLSL.

The supporting radius of SPH and the grid cell size have close relationship. Generally, set the supporting radius $h$ equals to the grid cell size $l$ has the optimal performance on GPU simulation. In this paper, we have $h=l$ in all our test cases. Different kinds of fluids have different CPN. For example, water has a CPN around 6. So for water, average particles in adjacent cells are around. That means when simulating water, each particle has to make about 162 computations to find out the density or force of this particle. Our method reduces the computation in inactive areas, and massive computing needs for inactive particles are avoided.

From our description in Section 3.3, the threshold $T$ and $IC$ play important roles in the process. With larger $T$ value, more particles are classified as inactive particles. The computational needs are reduced, and the simulation is simplified as more properties are inherited rather than computed. With larger IC value, inactive particles can be idle for more cycles. More errors would be introduced with larger IC value. From our experiments, we find that $T=1.4$ and $IC=10$ for water has a nice tradeoff between quality and speed.

To test the visual quality of our algorithm, we integrated HPMC as discussed in Section 4 for rendering. We set $T=1.4$ and $IC=10$ for water. Although our method speeds up the simulation process by introducing some inaccuracy in computation, the rendering results show that it is worth doing it. We compare the visual results of the same frame with and without our proposed activity-based property reuse method. We can see that although there is some small visual difference between the traditional simulation method and our optimized method, the simulation results are largely plausible and realistic (see Figure 5 for a comparison). More fluid rendering results are illustrated in Figures 8 and 9. We can see that the visual quality is plausible and realistic.

To test the speedup in SPH simulation, we compare our method with activity-based property reuse to the original GPU-based Z-indexing method [26]. We compare the frames per second (FPS) of key frames from frame 1 to 10000, using a water cube of 60 K particles. The testing results are shown in Figure 6. The green line is the FPS of original Z-indexing method without optimization. The red line is the FPS of our algorithm with $T=1.4$ and

Figure 4. Histogram pyramid: (top) compact and (bottom) traversal processes.
IC = 10. The blue line is the FPS of our algorithm with $T = 1.68$ and IC = 10. The orange line is the FPS of our algorithm with $T = 1.4$ and IC = 12. From the figure, we see that our method obviously outperforms the original method with a speedup factor between 1.3 and 1.8. Larger $T$ has better time performance than smaller $T$ (blue and red line), but the fluid quality is somewhat affected. Larger IC also results in better time performance, but the impact is not as obvious as the same percentage increase in threshold $T$ (orange and red line). The rendering result of this water cube example is illustrated in Figure 7.

We also tested our algorithm with different fluid scenes. Under the same $T$ and IC configuration, the water jet with obstacle scene (Figure 5) and without obstacle (Figure 9) obtained a maximum speedup factor of 1.41 and 1.35, respectively, in simulation. The maximum speedup for the two examples is achieved when the fluid becomes stable. The experimental results have shown that our algorithm actually performs better comparing with the original GPU-based Z-indexing method. The reason for the improved performance is obviously from the decreasing in computation for inactive particles. Thanks to our algorithm, we reach 21 fps for water cube rendering case in Figure 8 and 23 fps for water jet case in Figure 9.

6. SUMMARY

In this paper, we focus on the parallel-optimizing simulation and rendering of SPH fluids for interactive virtual
environments. A new activity-based property reuse algorithm is presented to improve the performance of fluid simulations on GPU. The visual quality of the fluid dynamic is not obviously affected. The simulation framework can produce plausible results with invisible sacrifice in quality. Because of the parallel nature of our algorithm, it is capable to put all the simulation and rendering on GPU in parallel. Our simulation reaches interactive rate with large particle sets and achieves better performance compared with the original GPU-based approaches without computation optimization. With the fast HPMC technique, we achieve real-time and high-quality simulation and rendering. If threshold $T$ and IC are properly chosen, an optimized trade-off between efficiency and quality can be
achieved. Our framework can be easily implemented on most platforms with a parallel computing device for interactive virtual environments. Our future work will focus on the integration of our activity-based property reuse technique with nonuniform particle SPH method to achieve better time performance for virtual environment applications.

ACKNOWLEDGEMENTS

The work was supported by RGC research grant (ref. 416212), UGC direct grant for research (no. 2050485), National Natural Science Foundation of China (Grant No. 61272326), and research grant of University of Macau.

REFERENCES


**AUTHORS’ BIOGRAPHIES:**

**Chen Huang** is a PhD candidate at the Department of Computer Science and Engineering, Chinese University of Hong Kong. He obtained his Bachelor of Engineering degree in 2009 from Zhejiang University, P. R. China. His research interests include physics-based fluid simulation, multi-fluid interaction, and soft-fabrics dynamics.

**Jian Zhu** is currently a postdoctoral research fellow at the University of Macau. He received his PhD and Master’s degree in software engineering from the University of Macau in 2013 and 2008, respectively, and his BSc degree in Mathematics from Sun Yat-sen (Zhongshan) University in 2005. His research interests include physics-based computer animation, graphics hardware acceleration, and so on.

**Hanqu Sun** is an associate professor at the Chinese University of Hong Kong. Her research interests include virtual reality, interactive graphics/animation, real-time hypermedia, virtual surgery, mobile image/video synopsis and navigation, and touch-enhanced simulations. Sun has an MS in electrical engineering from the University of British Columbia and a PhD in computer science from the University of Alberta, Canada. Contact her at hanqu@cse.cuhk.edu.hk.
Enhua Wu graduated from Tsinghua University in 1970 with his Bachelor’s degree, and he received his PhD degree (CS) from the University of Manchester, UK, in 1984. Since 1985, he has been researching at the Institute of Software, Chinese Academy of Sciences, and currently serves as the director of the CG group in the State Key Lab of Computer Science. He is also a professor in the University of Macau since 1997. His research interests include computer graphics, computer-aided design, virtual reality, and graphic hardware techniques. He is a member of CCF, IEEE, and ACM.