行政院國家科學委員會專題研究計畫　成果報告

可程式化圖形處理器支援之雲與煙即時模擬與顯像技術之研究

計畫類別：個別型計畫
計畫編號：NSC93-2213-E-009-073-
執行期間：93 年 08 月 01 日至 94 年 07 月 31 日
執行單位：國立交通大學資訊工程學系(所)

計畫主持人：莊榮宏

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 95 年 1 月 28 日

摘要

我們提出了一個在繪圖硬體上進行流體模擬與顯像之架構。 流體的動態行為必需遵守納維-斯托克斯方程式(Navier-Stoke equations)，我們分別利用以網格為基礎(Grid-based)與基於粒子(Particle-based)的方式來進行計算。而計算完之流體結果則利用結合廖宏祥等人[10]與Harris等人[6]之兩步驟方式來計算光影效果。 最後，我們利用以繪圖硬體加速之容積顯像(Volume rendering)方式對以網格為基礎之流體進行顯像，而基於粒子之流體模擬結果則以廖宏祥等人[10] 所提出的Metaball貼圖資料庫來對粒子進行顯像。 所有的模擬與顯像計算都透過繪圖硬體來進行，因此可以將CPU的計算能力挪出來給應用程式中的其它計算使用。

關鍵字: 流體模擬，粒子系統，容積顯像，煙，雲

Abstract

We propose a framework for fluid simulation and rendering using graphics hardware. The dynamics of fluid is governed by the Navier-Stoke equations, and is solved separately by both grid and particle methods. A simplified lighting model that combines the work of Liao et al. [10] and Harris et al. [6] is used to capture the lighting effects. Hardware accelerated volume rendering is used to render the result of grid-based simulation. For particle-based method, metaball lighting combining with texture database and textured billboards proposed by Liao et al. [10] is used to render the particles. All the computations are performed in GPU, which releases the power of CPU for other applications.

Keywords: Fluid simulation, Particle system, volume rendering, Smoke, Cloud

1 Introduction

Natural phenomena, such as cloud and smoke, play important rules in the 3D applications such as flight simulation, virtual reality, and games. Simulating the natural phenomena requires the knowledge of computational fluid dynamics, thermodynamic, and other physics backgrounds. Usually, the computational cost for simulating the fluid effects is too heavy to be used in real-time applications. Many researches concentrate on reducing the computing time by using some heuristic rules or simplified physics models. Most of these works are restricted to simulate a certain type of fluid effects, such as smoke, fire, cloud, or water, and require lots of trial-and-error processes to tune the simulation results.

Recently, due to the great improvement in graphics hardware, lots of researches are devoted to translate the mathematical problems from CPU to GPU for speeding up the computation. In this project, we propose two frameworks for fluid simulation and rendering using the graphics hardware. For simulation, a grid-based fluid simulation based on the work of Harris et al. [6], and a two-stage method that maps the particle-based fluid simulation to GPU are proposed. A two-pass method that combines the works of Liao et al. [10] and Harris et al. [6] to render the simulation result is proposed.

2 Related Work

The simulation of natural phenomena can be simply divided into two parts: simulation and rendering. We briefly review the related work in this two categories.

Simulation: In general, the simulation of fluid requires solving the Navier-Stoke equations, which can be classified into two categories: the Eularian methods and the Lagrangian methods. In the Eularian methods, the fluid is described in terms of what takes place at a fixed point as the fluid flows by. In the Lagrangian methods, the fluid is described as particles, and are tracked in the simulation space.

Foster and Metaxas [4] is the first one to introduce the Navier-Stoke equations into computer graphics field. Stam enhanced the work of Foster and Metaxas [4] by introducing the semi-Lagrangian advection method, and a backward tracking method for ensuring the stability of fluid [15]. Fedkiw et al. simulated the smoke using the semi-Lagrangian method and added the vorticity confinement for turbulent effects [3]. Nguyen et al. simulated the fire by adding additional fire rules [13]. Dobashi et al. proposed a cellular automata method to simulate the cloud by considering only the boolean states for fast update [2].

Harris implemented the semi-Lagrangian method on programmable graphics hardware and used it to simulate the cloud in real-time [6].

Lagrangian methods have the advantages that avoid the memory requirement of 3D grid in Eularian methods, and provide smooth motion trajectories. Desbrun and Gascuel simulated the deformation of soft bodies using smoothed particle hydrodynamics [1]. Müller et al. simulated the liquids using smoothed particle hydrodynamics, in which spatial partition is used to reduce the computation [12]. Premože et al. enhanced the particle-based method for liquid simulation, and could simulate the fluid flows like multifluids and multiphase flows [14].

Rendering: The rendering of natural phenomena is equvalent to rendering the participating media, which is generally a greate challenge. The works of Fedkiw et al. [3, 13] render the fluid effects by using photon map [8]. Dobashi et al. rendered the clouds using a two-pass method [2]. The first pass calculates the shadows and illumination of the clouds, and the second pass renders the cloud to form the final image. Harris and Lastra developed a system for rendering the static clouds [7]. The illumination of clouds are precomputed and the imposter is used to render the clouds in rum-time. Liao et al. proposed a framework for interactive rendering of cloud by using a shadow relation table (SRT) for fast illumination calculation and metaball lighting texture database (MLTDB) for final image composition [10].

3 Grid-Based Fluid Simulation on GPU

We propose a GPU based framework for fluid simulation and rendering based on the semi-Lagrangian method proposed by Stam [15]. The grid-based method can be used to simulate the large scale fluid without having too much detail, such as water and cloud.

A two-stage rendering method that combines the works of Liao et al. [10] and Harris et al. [6] is used to render the simulation result with illumination from the light source. The first stage computes the illumination of each voxel from the light source, and in the second stage, a hardware accelerated volume rendering is used to compose the voxel's image into the final image.

3.1 Fluid Simulation

The dynamics of fluid should satisfy the Navier-Stoke equations, which is derived from the conservation of mass and momentum. The system of Equations 1 and 2 is the most common form of Navier-Stoke equations.

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla) u - \frac{1}{\rho}\nabla p + \upsilon \nabla^2 u + f \qquad (1)$$

$$\nabla \cdot u = 0 \qquad (2)$$

where u, p, and ρ are the velocity, pressure, and density of fluid, respectly. t is the time step for advection, f is the external force applied to the fluid, and υ is viscousity coefficient.

We implement the work of Harris et al. [6] which is a GPU based implementation of the semi-Lagrangian method proposed by Stam [15]. The semi-Lagrangian method uses *Helmholtz-Hodge Decomposition* to ensure the satisfication of equation 2. And the simulation process is split into four steps: *advection*, *add force*, *diffusion*, and *proiection*.

The advection updates the attributes of fluid into next step. In order to ensure the stability of fluid dynamics, it uses a backward tracking method that tracks which part of fluid will come to current location after Δt time. Equation 3 explains the backward tracking method, and q can be any attributes of the fluid.

$$q(\vec{x}, t + \Delta t) = q(\vec{x} - \vec{u}(\vec{x}, t)\Delta t, t) \qquad (3)$$

The fluid may be interacted with other phenomena, such as wind, object boundaries, etc... These effects can be put into the fluid simulation by a additional force field, and affect the velocity of fluid as the following equation

$$\tilde{u}(x) = u(x) + \Delta t \cdot (F_1(x,t), F_2(x,t), ...)$$

where u is the fluid velocity after advection, and F_1, F_2, ... are all external force fields applied to the fluid. Also, the vorticity confinement proposed by Fedkiw et al. [3] is used to add extra turbulence to the fluid.

The third step adds the viscousity effect into the fluid, and is also computed by a backward tracking method. The equation used to compute the

viscousity diffusion is listed as follow

$$\left(I-\Delta t\cdot \upsilon\nabla^2\right)\tilde{u}\left(x\right)=u\left(x\right) \tag{4}$$

where u and $\tilde{u}$ are fluid velocity before and after applying the diffusion, I is an identity matrix. Harris et al. solves equation 4 using *Jacobi Iteration* [6].

After the diffusion, a projection is required to ensure the simulation result satisfies the conservation of mass. Equation 5 and equation 6 are used to achieve this goal.

$$\nabla^2 q\left(x\right)=\nabla u\left(x\right) \tag{5}$$

$$\tilde{u}\left(x\right)=u\left(x\right)-\nabla q\left(x\right) \tag{6}$$

Also, equation 5 can be solved by an iterative process using the *Jacobi Iteration*.

3.2 Rendering

The result of simulation is rendered in a two-pass method. The illumination of the voxels are computed in the first pass, and a hardware accelerated volume rendering is performed in second pass to take the illumination from voxels to eye into account.

The first pass calculates the illumination of each voxel in the simulation space. For each light ray, the illumination of point $\vec{x_n}$ on the light ray can be computed by the following equations

$$\begin{aligned} L\left(\vec{x_n},\vec{\omega}\right) &= L\left(0,\vec{\omega}\right)T\left(0,D_n\right) \\ &\quad +\int_0^{D_n} g\left(\vec{x}\left(s\right)\right)T\left(s,D_n\right)ds \end{aligned} \tag{7}$$

$$g\left(\vec{x}\right)=K_s\int_{4\pi}P\left(\vec{\omega},\vec{\omega}'\right)L\left(\vec{x},\vec{\omega}'\right)d\vec{\omega}' \tag{8}$$

where D_n is the distance between point $\vec{x_n}$ and the light source, $\vec{\omega}$ is the incident angle of light ray, $L(0,\vec{\omega})$ is the radiance of the light source, $P(\vec{\omega},\vec{\omega}')$ is the phase function, and K_s is the scattering coefficient. To capture the incident radiance of a voxel, it generally requires to integrate over all incident directions. We use the simplified model proposed by Harris et al. [6] that considers only the forward scattering, and assume that the phase function is a constant. Equation 8 can be rewritten as

$$g\left(\vec{x}\right)=\frac{\gamma K_s P\left(\vec{\omega}_l,\vec{\omega}_l\right)L\left(\vec{x},\vec{\omega}_l\right)}{4\pi} \tag{9}$$

The second pass renders the simulation result from eye position with the illumination. The outgoing radiance for each voxel in the eye ray direction can be computed using equation 10

$$S_k=\frac{K_s P\left(\vec{\omega},\vec{\omega}_l\right)L_k}{4\pi} \tag{10}$$

We use hardware accelerated volume rendering method that generates the textured slices in a back-to-front order to render the volume [5]. For each slice, the outgoing radiance of voxels on the slice can be computed using equation 10. And a simple back-to-front composition with alpha blending is performed that blends all the slices to the final image.

4 Particle-Based Fluid Simulation on GPU

Although the grid-based method yields resonable results for fluid simulation, it usually requires high resolution to capture the detail of fluid effects. Also, the numerical dissipation may occur in grid-based method smears out the flow. Particle method has the advantage of smooth motion for highly dynamics fluids, and is able to capture more details than the grid-based method.

4.1 Two-Stage Fluid Simullation

In particle-based method, the Navier-Stoke equations are solved on the particles. Equation 2 which describes the conservation of mass can be omitted by requiring constant number of particles and constant mass of particle. Moreover, the advection term $-\left(u\cdot\nabla\right)u$ in equation 1 can be removed since the particles always move with the fluid. The density, pressure, viscousity, and external forces of a particle in current step are accumulated by the weighting sum of all its neighboring particles using smoothed particle hydrodynamics (SPH) [11]. These accumulated values are used to advect the particle into next step.

We split the traditional particle-based method into two stages: the first stage accumulates the attributes of all particles into an intermediate grid using SPH. In the second stage, each particle is advected into next step using the attributes associated with the particle itself and the intermediate grid that the particle lies.

4.1.1 Force Accumulation

The goal of the first stage is to accumulate all the fources of fluid for advecting the particles into next step. For each point g_i on the intermediate grid, the accumulated density from all particles can be computed using SPH as

$$\rho_{g_i} = \sum_j m_j W\left(r_{g_i} - r_j, h\right) \tag{11}$$

where ρ_{g_i} is the density on the grid point g_i, m_j is the mass of particle j, and $W\left(r_{g_i} - r_j, h\right)$ is a kernel function to describe the contribution of the attribute relative to the distance.

The pressure force on the grid point g_i can be accumulated from all particles using SPH as follow

$$f_{g_i}^{pressure} = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W\left(r_{g_i} - r_j, h\right)$$

and $\nabla W\left(r_{g_i} - r_j, h\right)$ is the gradient of kernel function, p_j is the pressure of particle j. From the Ideal-Gas Equation and assume that the temperature T is constant, the pressure of particle can be simplified to be only affected by the density

$$p = k\rho$$

thus the pressure force can be rewritten to

$$f_{g_i}^{pressure} = -\sum_j m_j k \nabla W\left(r_{g_i} - r_j, h\right) \tag{12}$$

where k is a constant depending on the temperature.

The viscousity force on the grid point g_i can be achieved by accumulating the laplacian of the particle velocities using SPH, which is the following form

$$f_{g_i}^{viscousity} = \mu \sum_j m_j \frac{u_j}{\rho_j} \nabla^2 W\left(r_{g_i} - r_j, h\right) \tag{13}$$

The μ is a constant to control the amount of viscousity force, and $\nabla^2 W\left(r_{g_i} - r_j, h\right)$ is the laplacian of SPH kernel function.

4.1.2 Particle Advection

The second stage updates the velocity and position of particles using the forces in the intermediate grid. For each particle, tri-linear interpolation is performed to get the total force on the intermediate grid at the particle's current position. The velocity of the particle is updated using the interpolated force on the intermediate grid

$$v_i = \frac{f_g \Delta t}{m_i} \tag{14}$$

where v_i, m_i are velocity and mass of the particle i, and f_g is the force value achieved by tri-linear interpolation of the intermediate grid, and Δt is the step time. Finally, the position of particle i in the next step can be computed by

$$\tilde{x}_i = x_i + v_i * \Delta t \tag{15}$$

4.1.3 Vorticity Confinement

We use the vorticity confinement method proposed by Fedkiw et al. [3]. The vorcitiy can be computed by the curl of fluid velocity

$$\omega = \nabla \times u$$

and the force due to the vorticity confinement can be computed by the following equation

$$f_{vorticity} = \varepsilon\left(N \times \omega\right) \tag{16}$$

where ε is a constant used to control the amount of vorticity force introduced to the fluid, and $N = \frac{\nabla|\omega|}{|\nabla|\omega||}$ is a vector pointing from low vorticity to high vorticity.

We compute the vorticity of fluid in the intermediate grid, and the value will be carried by the particles. In the first stage, the vorticity and the partial gradient of velocity are accumulated using SPH. The vorticity on the intermediate grid in next step can be computed by the accumulated values.

$$\tilde{\omega}_g = \omega_g + \nabla \times u_g \tag{17}$$

Furthermore, the curl operator can be replaced by the following form

$$\nabla \times u_g = \left(\frac{\partial u_{g_z}}{\partial y} - \frac{\partial u_{g_y}}{\partial z}\right)\hat{x} + \left(\frac{\partial u_{g_x}}{\partial z} - \frac{\partial u_{g_z}}{\partial x}\right)\hat{y} + \left(\frac{\partial u_{g_y}}{\partial x} - \frac{\partial u_{g_x}}{\partial y}\right)\hat{z} \tag{18}$$

and all the partial differentiation terms on intermediate grid can be accumulated in the first stage. The force due to vorticity confinement can be derived from equation 16.

In second stage, the forces due to the vorticity confinement are added simply as the other forces to compute the particle velocities, and the vorticity ω is interpolated and carried by the particles.

4.2 Particle Rendering

Since the density of fluid is accumulated into the intermediate grid, the illumination of fluid from the light source can be easily computed using the first-pass rendering method described in Sec. 3.2. After we get the illumination of all grid points, the illumination of each particle is achieved using tri-linear interpolation of illumination of the grid points.

In the second pass, the particles are rendered using the metaball lighting texture database (MLTDB) proposed by Liao et al. [10]. The particle density $D_{particle}$ and the angle between light and eye rays $\alpha_{L,E}$ are used to fetch a texture from the MLTDB, and the texture is multiplied by the illumination of particle before rendering. A back-to-front order for rendering particles is required to ensure that it yields correct result.

5 Implementation

In this section, some implementation details about how to simulate and render on the programmable graphics hardware are addressed.

5.1 Grid-Based Fluid Simulation

The most intuitive way to describe the 3D volume in graphics hardware is the 3D texture. But the run-time update of the 3D texture is a greate challenge which usually requires multiple passes. To speed up the run-time update process, 2D tiled texture that flattens all slices of the 3D texture into a 2D texture is used, and only one single rendering pass is needed to update all the voxels. Since the computation is performed in 3D space, an additional texture which maps the texture coordinate in 3D volume space into the 2D tiled texture is required.

All the grid-based simulations are performed in the pixel shaders. Four pixel shaders which correspond to the four steps of the simulation are implemented. Another pixel shader is used to perform the Jacobi Iteration. For each step, one single quad that covers the entire space of the 2D tiled texture is rendered, and the pixel shaders compute and output the results of this step. Jacobi Iteration is performed in the same way.

5.2 Particle-Based Fluid Simulation

To implement particle system on the graphics hardware, we store the attributes of all particles in the textures. Each texel on the texture represents a particle in the particle system. The intermediate grid is represented as the 2D tiled textures in the grid-based method. The kernel, gradient, and laplacian of the kernel are precomputed and stored as 1D textures for run-time lookup.

To implement the first stage of our simulation method on programmable graphics hardware with the vorticity confinement, the first stage is split into two steps. The first step accumulates all attributes of particles into the intermediate grid, and the second step compute the vorticity on the grid.

Since the attributes of the particles are stored in the textures, we need one pixel shader to fetch the position of particles. For each particle, the vertex shader fetches the particle position stored in the texture , and translates the particle to the correct position before rendering. The pixel shader fetches the kernel values according to the distance between the particle and the intermediate grid point. Particle attributes are then accumulated to the intermediate grid using the kernel values. After the accumulation, one pixel shader which renders a quad to cover all intermediate grid texture is used to update the vorticity value on the intermediate grid using Equation 18.

The second stage is used to advect attributes of particles. One pixel shader is used. A quad that covers all particle textures is rendered. For each pixel, the pixel shader fetches data from particle textures and from intermediate grid textures with linear interpolation, advects the attributes of particle using data from the intermediate grid, and finally write back to the particle textures.

5.3 Rendering

In the two-pass rendering method, the first pass computes the illumination of voxels in the volume. To compute the illumination of voxels, the knowledge of the oeder which light rays traverse through the voxels is necessary. We modify the shadow relation table (SRT) proposed by Liao et al. [10] to record the order of voxels when the light rays traverse through the volume. The orig-

inal SRT is an array of linked lists to record the voxels in the light ray direction. To satisfy the GPU computation, the SRT is converted into the 2D tiled texture we used to represent the 3D volume, and for each voxel, the traverse order from the light source and the index of previous voxel where the light ray is traversed before this one are stored.

During the first-pass rendering, an iterative rendering process is performed to render the texture quads of volume N times, where N is the maximum depth in SRT. In each iteration, a pixel shader is used to fetch the illumination on the previous voxel, and update the illumination of current voxel if the iteration number is equal to the traverse order on the SRT texture.

For fluid simulation using grid-based method, the volume rendering method using graphics hardware is performed. The 3D texture coordinate is translated into 2D tiled texture space, and the linear interpolation is performed by fetching two texels on the 2D tiled texture with the previous and next Z-slices, and interpolates these two values linearly according to the exact Z coordinate. The lighting is calculated according to Equation 10, and the back-to-front composition with alpha blending ensures that the result image is correct.

For fluid simulation using particle method, the illumination on the intermediate grid is interpolated into the particles using the method described in the second stage of simulation. A GPU-based back-to-front sorting of particles proposed by Kipfer et al.[9] is performed, and followed by the rendering of the particles using the textured billboard fetched from MLTDB.

6 Examples

We choose two examples to demostrate our frameworks: *cloud* and *smoke*. The cloud changes slowly and has complex rules that are strongly sensitive to the altitude, and is simulated by the grid-based fluid simulation framework. The dynamics of smoke changes quickly and has fast and smooth motion, which is quite suitable for particle-based framework.

6.1 Cloud

We use the cloud rules proposed by Harris et al. [6]. Two cloud quantities are considered, which are the amount of water q_c and vapor q_v. A threshould called saturation vapor mixing ratio q_{vs} is used to control when the vapor is transitted to water. q_{vs} is affected by the temperature and pressure. The temperature and pressure decrease as the altitude raises, and thus decreases the q_{vs}. Equation 19 reveals the relation between q_{vs}, the temperature T, and pressure p.

$$q_{vs}(T,p) = \frac{380.16}{p} exp\left(\frac{17.67T}{T+243.5}\right). \quad (19)$$

The rules to update the amount of water q_c and vapor q_v are listed as follow:

$$\begin{aligned} \nabla q'_v &= min(q_{vs} - q'_v, q'_c), \\ q_v &= q'_v + \nabla q'_v, \\ q_c &= q'_c - \nabla q'_v. \end{aligned} \quad (20)$$

There are no rules to decrease the amount of water q_c in the work of Harris et al. [6], and we add an extra rule as shown in Equation 21: when the amount of water q_c exceeds a threshould q_{cs}, the water is eliminated as it becomes the rain.

$$q_c = min(q_c, q_{cs}) \quad (21)$$

The weight and buoyancy are also taken into account for raising and lowering the water and vapor as in the work of Harris et al. [6].

6.2 Smoke

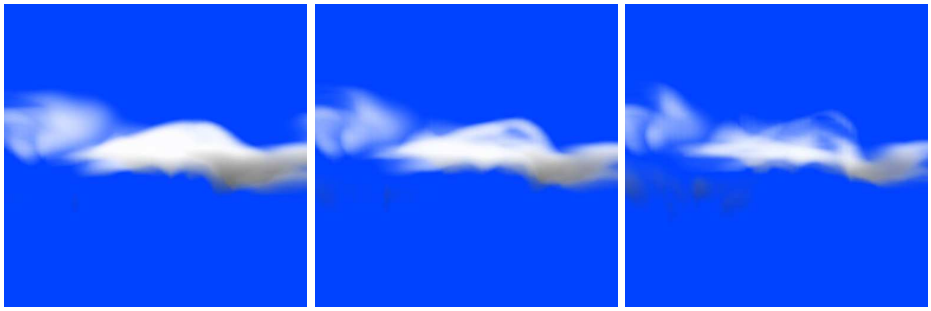
The smoke is implemented using the proposed particle-based fluid simulation framework. An emitter is used to emit the particles into the simulation space with an initial speed. No extra rules is used to simulate the particles in the space. Fig. 4 illustrates the smoke simulated and rendered using the proposed framework.

7 Results

We have implemented our works on PC with Pentium 4 3.2GHz CPU and NVidia GeForce 6800 graphics card.

Fig. 1 shows the cloud images in six frames. The light source is placed at the top of simulation space, and the resolution of simulation space

(a) Frame 43 (b) Frame 44 (c) Frame 45



(d) Frame 46 (e) Frame 47 (f) Frame 48

Figure 1: Screenshot of cloud images.

Resolution	Performance (ms)			
	Simulation	Lighting	Rendering	All (ms / fps)
$32 \times 32 \times 32$	26.65	13.32	39.98	66.64 / 15.00
$64 \times 32 \times 32$	53.31	26.65	39.98	93.30 / 10.71
$64 \times 32 \times 64$	93.30	39.99	39.98	153.41 / 6.52
$12 \times 32 \times 64$	186.60	66.64	39.98	279.90 / 3.57

Table 1: Cloud performance results in different resolutions

is $64 \times 32 \times 32$. To test the performance relative to the simulation resolution, we decompose the computing time into simulation, lighting, and rendering parts, which are shown in Table 1 lists the results. As the resolution increases, the time for simulation and lighting also increase. The time needed for rendering is constant since we use the view-aligned slices for volume rendering, and is independent of the volume resolution. Fig. 2(a) to Fig. 2(c) are cloud images in three different simulation resolutions, more detail can be generated using higher resolution. Table 2 shows the simulation time needed in different Jacobi iterations. The more Jacobi iterations are performed, the better numerical results can be achieved, in the cost of longer computing time. In our experiment, $30-35$ iterations are enouth to produce the visually acceptable result. By controlling the value of saturation vapor mixing ratio

Resolution	Number of Jacobi Iterations	Simulation Time (ms)	All (ms / fps)
$64 \times 32 \times 32$	15	26.66	79.97 / 12.50
	30	39.99	93.30 / 10.71
	45	54.38	106.64 / 9.37
$64 \times 32 \times 64$	15	53.31	119.96 / 8.34
	30	79.97	146.61 / 6.82
	45	106.62	173.27 / 5.77

Table 2: Cloud performance results in different Jacobi iterations

q_{vs} and the source of vapor, the weather such as sunny, cloudy, and overcast can be simulated easily. Fig. 3 demostrates the cloud in different weather conditions.

The smoke is implemented and rendered using the particle-based framework. Fig. 4 illustrates the smoke in different grid resolutions, and table 3 is the performance result. 1024 particles is used to simulate the smoke. The higher resolution, the more detail of vorticity can be generated, but af-

(a) $32 \times 32 \times 16$ (b) $64 \times 32 \times 32$ (c) $128 \times 32 \times 64$

Figure 2: Cloud images in different resolutions

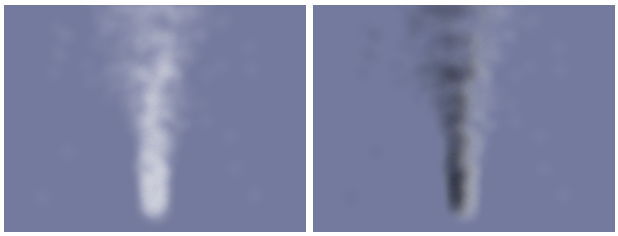
(a) Sunny (b) Cloudy (c) Overcast

Figure 3: Clouds in different weather conditions

fects the simulation and lighting speed. According to our experiments, resolution of $32 \times 32 \times 16$ is enough to produce visually acceptable result of smoke with smooth vorticity details. Our current implementation does not optimize the pixel shaders, and due to the slow rasterization speed of the graphis hardware, the performance can achieve only interactive rate with about 1 to 13 FPS. Fig. 5(a) and Fig. 5(b) show the smoke rendered with and without lighting, the light source is placed at the right side of the smoke. With the lighting, the detail turbulent effects can be seen clearly due to the contrast of illumination. The lighting calculation requires additional rendering iterations for computing light propagation, and thus affects the performance.

Grid Resolution	Number of Particles	Average FPS
$16 \times 16 \times 16$	512	13.39
$16 \times 16 \times 16$	1024	7.46
$16 \times 32 \times 16$	512	7.46
$16 \times 32 \times 16$	1024	3.75
$32 \times 32 \times 32$	512	1.97
$32 \times 32 \times 32$	1024	0.98

Table 3: Smoke performance results in different configurations



(a) Without lighting (b) With lighting

Figure 5: Smoke with and without lighting

8 Conclusion

We have implemented GPU based frameworks for grid-based and particle-based fluid simulation and rendering. The grid-based simulation is based on the semi-Lagrangian method proposed by Stam [15] and the GPU version of Harris et al. [6]. A two-stage particle-based simulation method is proposed to simulate the fluid using SPH. The first stage accumulates the attributes of fluid into an intermediate grid, and the second stage advects the particles using the accumulated attributes stored on the intermediate grid. A two-pass rendering method is used to render the result of simulation. The first pass computes the illumination of all grid voxels using the SRT texture. For grid-based simulation, a hardware accelerated volume rendering using 2D tiled texture is used to render the final image. For particle-based simulation, the illumination on the grid voxels are interpolated into particles and a back-to-front order to render all particles using textured billboard fetch from the MLTDB. All the computations are in the GPU which maximizes the parallelism of pixel shaders. The examples have shown that our framework can achieve the interactive rate at about 1-70 FPS according to the resolution for

(a) $16 \times 16 \times 16$ (b) $32 \times 32 \times 16$ (c) $64 \times 64 \times 32$

Figure 4: Smoke in different resolution

simulation.

References

[1] M. Desbrun and M.-P. Gascuel. Smoothed Particles: A New Paradigm for Animating Highly Deformable Bodies. In *Proceedings of The 6th Eurographics Workshop on Animation and Simulation*, pages 61–76, 1996.

[2] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A Simple, Efficient Method for Realistic Animation of Clouds. In *Proceedings of SIGGRAPH*, pages 19–28, 2000.

[3] R. Fedkiw, H.W. Jensen, and J. Stam. Visual Simulation of Smoke. In *Proceedings of ACM SIGGRAPH*, pages 15–22, 2001.

[4] N. Foster and D. Metaxas. Modelling the Motion of a Hot, Turbulent Gas. In *Proceedings of ACM SIGGRAPH*, pages 181–188, 1997.

[5] M. Hadwiger, C.R. Salama, K. Engel, J.M. Kniss, A. Lefohn, and D. Weiskopf. Real-Time Volume Graphics. *ACM SIGGRAPH Course Note*, 2004.

[6] M.J. Harris, W.V. Baxter, T. Scheuermann, and A. Lastra. Simulation of Cloud Dynamics on Graphics Hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 92–101. Eurographics Association, 2003.

[7] M.J. Harris and A. Lastra. Visual Simulation of Clouds. In *Proceedings of Eurographics*, pages 76–84, 2001.

[8] H.W. Jensen and P.H. Christensen. Efficient Simulation of Light Transport in Scenes with Participating Media Using Photon Maps. In *Proceedings of ACM SIGGRAPH*, pages 311–320, 1998.

[9] P. Kipfer, M. Segal, and R. Westermann. UberFlow: A GPU-Based Particle Engine. In *Proceedings of Graphics Hardware*, 2004.

[10] H.-S. Liao, T.-C. Ho, J.-H. Chuang, and C.-C. Lin. Fast Rendering of Dynamic Clouds.

[11] J.J. Monaghan. Smoothed Particle Hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30:543–574, 1992.

[12] M. Müller, D. Charypar, and M. Gross. Particle-Based Fluid Simulation for Interactive Applications. In *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation*, 2003.

[13] D.Q. Nguyen, R. Fedkiw, and H.W. Jensen. Physically Based Modeling and Animation of Fire. In *Proceedings of ACM SIGGRAPH*, 2002.

[14] S. Premŏz, T. Tasdizen, A. Lefohn, and R.T. Whitaker. Particle-Based Simulation of Fluids.

[15] J. Stam. Stable Fluids. In *Proceedings of SIGGRAPH*, pages 121–128, 1999.