Rendering Snowing Scene on GPU

Jun Zhang, Xingquan Cai, Jinhong Li

College of Information Engineering
North China University of Technology
Shijingshan District, Beijing, China

alexmaya@163.com

***Abstract* - As the part of natural sceneries, snow scene is a familiar natural phenomenon in the north of China. In this paper, we present a new method to render snow scene on GPU based particle system. We simulate the whirling snowflakes in the sky and the snow deposition effect. Our method processes the birth and death of snow particles via index on CPU and uses a pair of Floating Point Textures on GPU to store the dynamic attributes of snow particles. Our method also updates the dynamic attributes of snow particles, and renders the system on GPU. We also provide a method based height field and establish some deposition rules to simulate the snow deposition. Finally, our experiments prove that our method is feasible and high performance. Our method can be used in computer animation, games, special movie effects, etc.**

Keywords – snowing scene;* snow deposition*; snow particle; GPU (Graphics Processing Unit)

I. INTRODUCTION

Most natural sceneries effects, such as clouds, fire, rain, snow, smoke, etc, are full of motion, full of chaos and full of fuzzy objects, and change with time past. These natural sceneries effects are routinely created in today's video games. So the simulation of natural sceneries becomes a hot topic in the research field of computer graphics.

Usually, the particle systems could be used to simulate these natural sceneries on CPU [1-3]. However, if the number of particles is above 10K, the particle system on CPU is difficult to run on real-time. It's required that plenty of particles more than 10K should be need in simulate system of photorealist natural sceneries effects. Today, with the development of GPU, we could deal with complex computing and program on GPUs.

As the part of natural sceneries, snow scene is a familiar natural phenomenon in the north of China. Snow scene in the simulation system could completely change the mood and appearance of the scene. So in this paper, we present a new method to render snow scene on GPU based particle system. We simulate the whirling snowflakes in the sky and the snow deposition effect. Our method processes the birth and death of snow particles via index on CPU and uses a pair of Floating Point Textures on GPU to store the dynamic attributes of snow particles. Our method also updates the dynamic attributes of snow particles, and renders the system on GPU. We also provide a method based height field and establish some deposition rules to simulate the snow deposition. Finally, our experiments prove that our method is feasible and high performance. Our method can be used in computer animation, games, special movie effects, etc.

In this paper, after exploring the related work of snow scene simulation, we provide our snow particle system method on GPU. Then we give the snow deposition simulation method based height field. And we also show the results using our method. Finally, we draw the conclusion and give the future work.

II. RELATED WORK

Snow scene simulation is a hot topic, so there are a lot of papers about simulating a snow scene. Reeves [4] first simulated falling snow with a particle system. However, the original simulation results are less realistic due to the incorporation of an over-simple set of rules. Nishita [5] adopted metaballs to model and render the scene of snow accumulation and to simulate light scattering through the snow. Nevertheless, their algorithm is not suitable for real-time simulation. Fearing [6] described a complex stability model to construct fallen snow by geometrical surfaces, and generated the most realistic falling snow scene to date. However, due to the huge scene data and the time-consuming modeling process, its rendering speed is quite low. Ohlsson [7] adopted a depth image method to calculate the amount of fallen snow on each mesh of the terrain and perform occlusion culling, thus they could render the scene of falling snow quite fast. However, this approach demands a great amount of pre-processing time and the system is not high performance. So we need to provide one efficient method to simulate the snow scene, including the whirling snowflakes in the sky and the snow deposition effect.

With the development of GPU, several forms of physical simulation have recently been developed for modern GPU. In 2003, Harris [8] has used GPU to perform fluid simulations and cellular automata with similar texture-based iterative computation. Green [9] describes a cloth simulation using simple grid-aligned particle physics, but does not discuss generic particle systems' problems, like allocation, rendering and deallocation. However their algorithm does not show the necessary properties to exploit the high frame-to-frame coherence of the particle system simulation. Recently, Schneider et al [10], Li et al [11], and Livny et al [12] have used GPU to render large scale terrain scene. Christopher et al [13] also provide the method of real-time mesh simplification using GPU. As GPU could deal with complex computing so fast, we want to implement particle system on GPU. So we

978-1-4244-6585-9/10/$26.00 ©2010 IEEE

provide a rendering snow scene method on GPU in this paper. Our method is based on the particle system, and we can simulate the whirling snowflakes in the sky and the snow deposition effect.

III. Snow Particle System on GPU

The following subsections describe the algorithm of our state-preserving snow particle system on GPU in detail. In this way, we can simulate the whirling snowflakes in the sky.

A. Snow Particle Data Storage

Position is one of the most important attributes of a snow particle. In our system, positions of all active particles are stored in a floating point texture with three color components that will be treated as x, y and z coordinates. Each texture is conceptually treated as a one-dimensional array, texture coordinates representing the array index. However, the actual textures need to be two-dimensional for the size restrictions of current hardware. The texture itself is also a render target, so it can be updated with the computed positions. As a texture cannot be used as input and output at the same time, we use a pair of these textures and a double buffering technique to compute new data from the previous values.

If other snow particle attributes, such as velocity, orientation, size, color, and opacity, were to be simulated with the iterative integration method, they would need texture double buffers as well.

B. Snow Particle Birth and Death

The birth of a snow particle requires associating new data with an available index in the attribute textures. Since allocation problems are serial by nature, this cannot be done efficiently with a data-parallel algorithm on the GPU. Therefore an available index is determined on the CPU via traditional fast allocation schemes. The simplest allocation method uses a stack filled with all available indices.

In our method, the particle's death is processed independently on the CPU and GPU. The CPU registers the death of a particle and adds the freed index to the allocator. The GPU does an extra pass over the particle data: The death of a particle is determined by the time of birth and the computed age. The dead particle's position is simply moved to invisible areas, e.g. infinity. As particles usually fade out or fall out of visible areas anyway at the end of their lifetime, the extra pass rarely really needs to be done. It is a basically clean-up step to increase rendering efficiency.

C. Update Snow Particles Attributes

The most important attributes of a snow particle are its position and velocity. So we just deal with snow particle position and velocity. The actual program code for the attributes simulation is a pixel shader which is used with the stream processing algorithm. The shader is executed for each pixel of the render target by rendering a screen-sized quad. The current render target is set to one of the double buffer attribute textures. The other texture of the double buffer is used as input data stream and contains the attributes from the previous time step. Other particle data, either from inside the attribute textures or as general constants, is set before the shader is executed.

1) Update Velocities:

There are several velocity operations that can be combined as desired: global forces (gravity, wind), local forces (attraction, repulsion), velocity dampening, and collision responses. For our GPU-based particle system these operations need to be parameterized via pixel shader constants.

Global and local forces are accumulated into a single force vector. The acceleration can then be calculated with Newtonian physics as (1). In (1), a is the acceleration vector, F is the accumulated force and m is the mass of the particle. If all particles have unit mass and forces have the same value, the accelerations can be used without further computation.

The velocity is then updated from the acceleration with a simple Euler integration in the form of (2). In (2), v is the current velocity, $\overline{v}$ is the previous velocity and Δt is the time step.

$$a = \frac{F}{m} \tag{1}$$

$$v = \overline{v} + a \cdot \Delta t \tag{2}$$

2) Update Positions:

Euler integration has already been used in the previous section to update the velocity by using the acceleration. The computed velocity can be applied to all particles in just the same way. We use (3) to update the position. In (3), p is the current position and $\overline{p}$ is the previous position.

$$p = \overline{p} + v \cdot \Delta t \tag{3}$$

D. Transfer Texture Data to Vertex Data

Before rendering snow particles, we should copy the snow particle data from the floating point texture to vertex data. Copying the particles data from a texture to vertex data is a hardware feature that is only just coming up in PC GPUs. OpenGL offers vertex textures with ARB_vertex_shader extension. OpenGL also provides two functions, glReadBuffer and glReadPixels. These functions could copy the particle data from the floating point texture to vertex data.

E. Render Snow Particles

The snow particles can be rendered as point sprites, triangles or quads. If the snow particle is rendered as triangles or quads, the particle may have three vertices or more vertices. So it is required that we should recompute the vertices position of particle before rendering. To avoid this overhead, our implementation uses point sprites.

IV. Snow Deposition

Snow deposition plays an important role in the simulation of snow scene. After a snowflake falls and reaches the ground, it reaches a stable site and gets solidified.

To simulate the dynamic snow deposition, we first need to generate the height field h(x, y) of the ground. Here the

ground includes the terrain, buildings, some man-made snow fences, etc.

Therefore, the height field is represented in a discrete form $h(x_i, y_j)(i = 0, 1, \cdots, m; j = 0, 1, \cdots, n)$. After a snowflake falls to the height $h(x_i, y_j)$, it reaches the site (x_i, y_j) and accumulates there. After plenty of snowflakes deposit at the same site, the height of this site is increased and updated.

In order to simulate the realistic snow deposition scene, we establish some deposition rules.

1) When the current height is equivalent and below to all the height of neighbors, just as Fig. 1 shows, the snow particle is stable deposition and updates the position.

2) When the current height is above all the height of neighbors and all the height of neighbors is equal, just like Fig. 2, the snow particle will roll to the four positions of neighbors in the same probability, 25%.

3) When the current height is above and equivalent to the height of neighbors, just as Fig. 3 shows, but the height of neighbors is not the same, and the number of lower neighbors is n, the snow particle will roll to the lower positions in the same probability, 1/n.

In this way, we can implement the snow deposition.

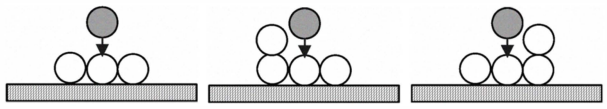

Figure 1. Current height is equivalent and below to all the height of neighbors.

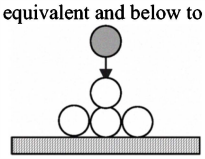

Figure 2. Current height is above all the height of neighbors and all the height of neighbors is equal.

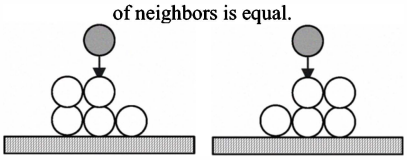

Figure 3. Current height is above and equivalent to the height of neighbors and the height of neighbors is not the same.

V. RESULTS

We have implemented our algorithm. Our implementations are running on a Intel PIV 2.8GHz computer with 1GB RAM, and NVIDIA GeForce7650 graphics card with 256M RAM, under Windows XP, Visual C++6.0, OpenGL and Cg environment, while running smoothly in real time. The rendering has a real viewport size of 1024×768.

A. Compare with CPU Particles Method

We implement a particle system on CPU and a particle system on GPU to simulate flowing magma. The particle just has the gravity and we do not concern other forces and the collision. At the same number of particles, we note the rendering frame rate. In order to ensure the objectivity of the experiment data, we sample continuous 1000 frames, note the FPS and compute the average FPS. In our experiment, when the number of particles is 100,000, the FPS of PS on GPU is above 60 fps. But at the similar condition, the FPS of PS on CPU is below 18 fps. When the number of particles is 200,000, the FPS of PS on GPU is 36 fps, but the FPS of PS on CPU is below 8 fps. All these prove that PS on GPU is higher performance than PS on CPU.

B. Rendering snowing scene on GPU

We implement a snow particle system on GPU to simulate snowing scene. The particle also just has the gravity and we do not concern other forces. We can modify the number of snow particles in scene to simulate the light snow and the heavy snow. As Fig. 4 shows the light snow in our scene, there are 5,000 snow particles. And Fig. 5 shows the heavy snow scene, there are 15,000 snow particles in our scene. Our system is running smoothly.

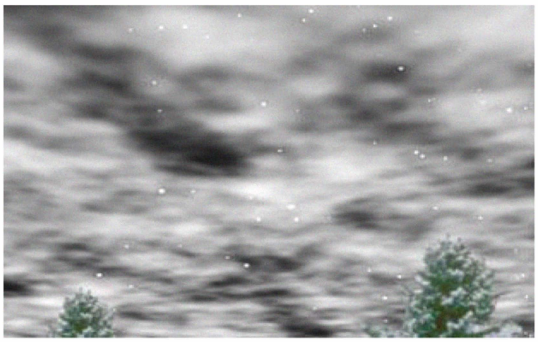

Figure 4. Light snow in our scene.

Figure 5. Heavy snow in our scene.

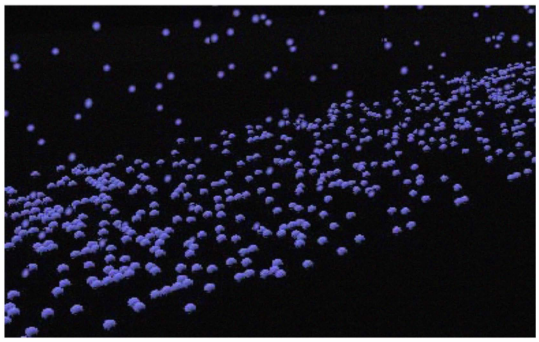

Figure 6. Snow deposition simulation after a short time.

C. *Snow Deposition Effects in our scene*

In order to display our method clearly, we use some small balls to simulate the snow deposition. Fig. 6 shows the snow deposition simulation after a short time. And Fig. 7 shows snow deposition simulation after a long time.

We also have used our method to simulate the snow deposition effect. Fig. 8 shows the snow deposition in our implementation scene. Because we add the wind in our scene, so the snow on the terrain is not plate.

Our method has also been used in the practical projects, and the practical projects have the high performance.

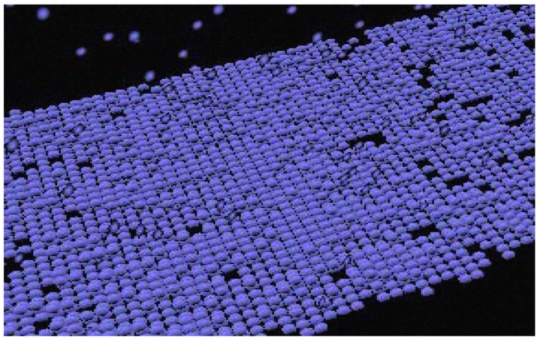

Figure 7. Snow deposition simulation after a long time.

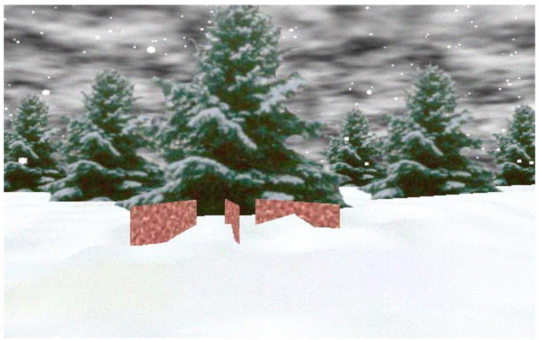

Figure 8. Snow deposition simulation in our implementation.

VI. Conclusion and Future Work

Snow scene is a familiar natural phenomenon in the north of China, and could change the mood and appearance of the scene. So in this paper, we present a new method to render snow scene on GPU based particle system. We simulate the whirling snowflakes in the sky and the snow deposition effect. Our method processes the birth and death of snow particles via index on CPU and uses a pair of Floating Point Textures on GPU to store the dynamic attributes of snow particles. Our method also updates the dynamic attributes of snow particles, and renders the system on GPU. We also provide a method based height field and establish some deposition rules to simulate the snow deposition. Finally, our experiments prove that our method is feasible and high performance. Our method can be used in computer animation, games, special movie effects, etc.

As a future possibility, we are working on using our method to implement other complex natural phenomenon, and developing the method to implement the effect with wind and snow particles.

Acknowledgment

This work was supported by PHR(IHLB) Grant (PHR20100509, PHR201008202), Funding Project of Beijing Outstanding Academic Human Resource Program (No. 20081D0500200114), Funding Project of Beijing Municipal Education Committee (No. KM201010009002), and Funding Project of North China University of Technology (No. NCUT2009008). We would like to thank those who care of this paper and our projects. Also, we would like to thank everyone who spent time on reading early versions of this paper, including the anonymous reviewers.

References

[1] Wang Changbo, Wang Zhangye, Peng Qunsheng, "Real-time Snowing Simulation", *The Visual Computer*, vol. 22, no. 5, pp. 315-323, 2006.

[2] Liu Xiaoping, Yu Ye, Chen Hao, et al, "Real-time simulation of special effects in navigation scene", *Journal of Engineering Graphics*, no. 3, pp. 44-49, 2007.

[3] Guan Yu, Zou Lin-can, Chen Wei, Peng Qun-sheng, "Real Time Waterfall Simulation Based Particle System", vol. 16, no. 11, pp. 2471-2474, 2004.

[4] Reeves W.T., "Particle Systems-Technique for Modeling a Class of Fuzzy Objects", In: Proceeding of SIGGRAPH 1983, 1983.

[5] Nishita T., Iwasaki H., et al, "A modeling and rendering method for snow by using metaballs", Computer Graphics Forum, Vol.16, No.3, pp. 357–364, 1997.

[6] Fearing P., "Computer modeling of fallen snow", In Proceedings of SIGGRAPH 2000, pp. 37-46, 2000.

[7] Ohlsson P., Seipel S., "Real-time rendering of accumulated snow", In Proceedings of SIGRAD 2004, pp.25-32, 2004.

[8] Harris M., "Real-Time Cloud Simulation and Rendering", PhD thesis, University of North Carolina at Chapel Hill, 2003.

[9] Green Simon, "Stupid OpenGL Shader Tricks", http://developer.nvidia.com/docs/IO/8230/GDC2003_OpenGLShaderTricks.pdf, 2003.

[10]Schneider J., Westermann R., "GPU-Friendly High-Quality Terrain Rendering", *Journal of WSCG*, vol. 14, pp.49-56, 2006.

[11]Li Sheng, Ji Junfeng, et al, "High performance navigation of very large-scale terrain environment", *Journal of Software*, vol. 17, no. 3, pp. 535-545, 2006.

[12]Livny Y., Kogan Z., and El-Sana J., "Seamless Patches for GPU-based Terrain Rendering", In Proceedings of WSCG 2007, pp. 201-208, 2007.

[13]Christopher D. and Natalya T., "Real-time Mesh Simplification Using the GPU", In Proceedings of Symposium on Interactive 3D Graphics (I3D), 2007.

[14]Latta L., "Building a Million Particle System", In Proceedings of Game Developers Conference 2004, 2004.

[15]Kolb A., Latta L., et al, "Hardware-based Simulation and Collision Detection for Large Particle Systems", In Proceedings of Graphics Hardware 2004, pp. 123-132, 2004.

[16]Tang Yong, Zhang Qian, "Real-time modeling and rendering on he realistic of snow scenes", Chinese Computer Science, vol.37, no.4, pp. 289-292, 2010.