Debris Flow Simulation-Oriented Three-Dimensional Scenery Modeling and Rendering

YE Jian1, 2, LIN Jiayuan1, TAO Heping1, CHEN Xiaqing1, CHEN Jingxiong1
1. Institute of Mountain Hazards and Environment, Chinese Academy of Sciences, Renmin Road, Chengdu 610041, China
2. Graduate University of the Chinese Academy of Sciences, 19A Yuquan Road, Beijing 100049, China

© Wuhan University and Springer-Verlag Berlin Heidelberg 2011

Abstract 3D (three-dimensional) process simulation is currently one of the most challenging fields of research on debris flow. Large scale terrain rendering is the most basic task of 3D scenery construction in debris flow simulation. As the major trigger for debris flow, rainfall will substantially enhance the realistic sense. Terrain and rainfall rendering in 3D debris flow simulations poses great challenges for numerical computation and graphical processing capability. In this paper, we propose to integrate GPU technology, LoD algorithms, and particle systems to realize 3D scenery modeling and rendering. The real-time LoD-based terrain modeling and rendering algorithm is presented first, and then a particle system-based rainfall scenery rendering method is implemented. Experimental results demonstrate that the 3D scenery rendered with the proposed approach exhibits sound performance and fair visual effects, which lays a solid foundation for the whole process simulation of debris flow disasters.

Keywords debris flow simulation; 3D scenery; GPU; LoD; particle system

CLC number P208

Introduction

Debris flow is a natural phenomenon characterized by high density water with mud and big gravel flowing down along a stream at high speeds in mountainous areas. It is one kind of solid-liquid two-phase fluid containing a lot of sediment, stones and huge gravel in a viscous laminar or diluted turbulent motion state. Debris flow is induced by many natural factors (geology, topography, hydrology, meteorology, soil, vegetation, etc.) and human factors.\[1\]

The occurrence of debris flow has often caused a lot of casualties and property loss. In the early morning of August 8, 2010, a super-large debris flow broke out in ZhouQu, GanSu province, which covered half of the town with 1765 people dead or missing. At the same time, it formed a big Avalanche Lake and induced other secondary geological disasters. In August of the same year, northern Pakistan suffered great rainstorms, which induced landslides and debris flow, and at least 63 people were killed.

In recent years, many researchers from different disciplines have focused their research on debris flow physical process, and its prediction, prevention and damage assessment. Three-dimensional(3D) debris flow simulation has attracted researchers from both the computer graphics and geological disaster do-
mains. A 3D visualization of the process of debris flow can not only improve the visual effect of observing debris flow, and contribute to identifying macro principles of debris flow occurrence, but also could lay a foundation for deep analysis of process data acquired from traditional 1D and 2D numerical simulations.[2]

The most basic task of the 3D debris flow simulation is to model and render the 3D realistic terrain in mountainous areas. The high requirements for numerical computation and graphic processing capability in 3D scenery construction are partly satisfied with the support of a Graphics Processing Unit (GPU). GPU has a computing power greatly exceeding the CPU, and supports more complicated calculations. Many previously unavailable real-time computing algorithms on the CPU currently have been realized on the GPU, which makes it possible to rapidly complete massive amounts of geometric and graphical computations in the 3D debris flow simulation [3]. On the other hand, the capability of large scale terrain rendering directly influences the real-time performance of debris flow process simulation and its visual effect of realistic 3D scenery. Therefore, a more efficient algorithm should be adopted to render the 3D terrain while the terrain information is extracted from the Digital Elevation Model (DEM) data to determine the mountain boundary.

In addition, as the majority of debris flows are triggered by strong rainfall, rainfall becomes an indispensable component in the 3D scenery of debris flow simulation. At present, rainfall rendering is mainly based on particle system model.[4] In order to realize the raining scenery, the number of particles is often more than tens of thousands, which results in the expansion of computation volume. Therefore, an optimized terrain rendering algorithm must be adopted to avoid the discontinuous and flickering feeling of 3D scenery on screen.

In this paper, we propose to render terrain and rainfall based on Levels of Details (LoD) algorithm,[5] which greatly decreases geometric complexity of the 3D scenery by simplifying scenery details and effectively reducing graphical display data. Our method not only realizes efficient modeling and rendering of the 3D scenery of debris flow simulation, but also rapid and realistic rainfall rendering. Therefore, our work in this paper lays a sound foundation for simulating the whole process of debris flow in real time 3D scenery with GPU technique and particle system.

1 Terrain modeling and rendering

1.1 DEM-expressed terrain

Debris flow commonly occurs in mountainous areas, thus the mountain surface is the boundary for debris flow. Therefore, it is very important to factually reflect the terrain features of the mountain region where debris flow occurs. The terrain information is extracted from DEM data, from which we can easily calculate the elevation of the mountain and valley. DEM data contain high-resolution terrain information that general topographic maps cannot express, which makes it easy to conduct visualization research. More importantly, from DEM we can conveniently calculate the height of any point in the mountain region, which simplifies collision detection between terrain and debris flow.[6] In the motion process of debris flow, the fluid particles of debris flow will be projected onto a height map. If the height of fluid particles is lower than the mountain surface, we can anticipate collision occurrence in this place. In our experiment, we projected topographic height values into 3D space, and then read their space coordination triples \((X, Y, Z)\), with which the rugged mountain terrain will be painted and displayed on a computer screen using the OpenGL (Open Graphics Library).

1.2 Quadtree-based LoD algorithm

However, due to the complexity of 3D scenery debris flow simulation, we need to reduce the geometric complexity of 3D scenery to assure both picture fluency and dynamic characteristics while rendering and displaying the mountain terrain where debris flow occurs. Therefore, we reduce the number of triangles in each frame drawing to enhance the efficiency of the algorithm without obvious influence on visual effect.[7-8] This objective is achieved by inducing a LoD algorithm, which makes it possible for observers to focus on rendered 3D mountain scenery by following generated debris flow activities in real time.
During a dynamic display of debris flow mountain terrain, the more the viewer’s location approaches the mountain, the more details of the terrain will be shown. The picture of a 3D mountain will occupy more pixels on the screen. The farther the viewer’s location is from the mountain, the coarser the details will be shown and the picture of 3D mountain will occupy fewer pixels on the screen. At this moment, it is not necessary to accurately show the object using a lot of polygons or triangles. Therefore, we can use different kinds of precision to show a 3D mountain scenery based on the viewer’s location—in other words—how many pixels the object picture occupies on the screen and choosing different levels of the model for rendering depending on the viewer’s location. In this way, we can efficiently enhance the terrain rendering speed, and create a more realistic and more detailed terrain.\[9\]

In our experiment, we employ the hierarchical data structure of a quadtree to partition and reorganize original DEM data into different resolutions, based on which level of 3D terrains with different details are rendered. This quadtree-based LoD algorithm approximately fits for original DEM data and keeps a reasonable magnitude of the number of triangles that are needed to render in all hierarchies.\[10, 11\] The quadtree hierarchy is aimed at $(2^n + 1) \times (2^n + 1)$ regular mesh and the sampling interval must be uniform.\[12\] As shown in Fig.1, the square patch with a size of $5 \times 5$ vertices will be used as an example to illustrate the idea of quadtree LoD algorithm. The $5 \times 5$ patch is going to have three levels of detail, with level 0 being the most detailed and, level 1 being the less detailed; in this case, level 2 being the least detailed. Each square is a node of quadtree, and each node keeps some information in certain areas. In accordance with the relief degree of topographic conditions, the terrain is constantly partitioned into four equal areas until height variance in the square is relatively small. The deeper the partition depth of the terrain is, the greater the terrain resolution.

If the adjacent nodes in different levels are not processed while rendering nodes, some vertices will be omitted and cracking problem will occur.\[13\] There are two ways of fixing the cracking problem. One way is to add vertices to the patch with the less detailed level so that the vertices in question will be of the same height as the corresponding vertices in higher detailed level. This approach is called edge addition method, which seems somewhat ugly (shown in Fig.2 (a)). It means that we have to do some rearranging of the patch by adding additional triangular fans. The other way of solving this problem is to omit vertices in the more detailed levels, which is called edge reduction method. This approach solves the cracking problem in a seamless and effortless way.\[14\] In our experiment, we adopt edge reduction method. As shown in Fig.2 (b), the subtle process of vertex omission and crack fixing is illustrated.

Because each level of the quadtree is regular mesh, the selected terrain patch should also be regular. The judgment of LoD levels is according to the distance between view location and the center of the current patch. Each level of detail has a distance range from the low point to the high. After observation distance is calculated, the corresponding LoD terrain data will be assigned to the patch.

1.3 Implementation and comparison

Take a height map of $512 \times 512$ mesh grids as the experimental data. While drawing terrain mesh with the non-simplified algorithm, the terrain is composed of $512 \times 512$ vertices, as shown in Fig.3 (a). Although...
this algorithm is simple and the rendered terrain provides the highest amount of details, it has fairly low efficiency. It always draws all of the mesh vertices without consideration of the distance between the terrain surface and the viewpoint. As we all know, it is a natural visual effect of human eyes that the clearness and details of objects in observation decrease as the distance between objects and viewpoint increases. As shown in Fig.3 (b), the LoD-based algorithm takes the view distance into account. The farther the terrain is from the viewpoint, the fewer terrain mesh vertices and details are drawn, and vice versa. Therefore, the latter algorithm has drawn fewer vertices as a whole and thus the rendering performance is substantially improved. On conditions of no distortion, the FPS (Frames Per Second) of rendering the same experimental terrain data is increased from the original 27 FPS to 89 FPS with the two algorithms, respectively.

2 Rainfall scenery rendering

Water is not only an important component of debris flow, but also its major triggering factor and transportation medium (source of driving force). Therefore, the occurrence of debris flow is closely connected with water. The water source that induces debris flow is mainly from atmospheric precipitation. The reality of rendering rainfall directly affects realistic expression of 3D virtual scenery of debris flow. For now, the particle system is considered a very effective graphical generation algorithm to simulate irregular fuzzy object. In this paper, we will use the particle system to render rainfall scenery.

2.1 Particle system model

The particle system defines a fuzzy object as a collection of particles consisting of thousands of randomly moving and irregularly distributed particles. Each particle has a certain life cycle and other properties (such as color, shape, size, and speed, etc). All of the particles are constantly changing their shapes and movement styles, by which the fuzzy object (3D scenery for instance) show dynamic variation in morphology and feature as a whole. The steps of using particle system to render rainfall scenery are listed as follows:

1) Generate new particles and add them to the system;
2) Each new generated particle must be assigned with the unique properties;
3) Any particle that has outlasted its lifespan must be declared dead;
4) Generated particles must be moving and changing according to their initialized attributes;
5) Display and render generated particles in current status.

2.2 Particle life cycle

While using the particle system to simulate rainfall scenery, there are always new particles being constantly generated and old particles dying. When the rain particles outlast their lifespan, they will be marked as vanished (or dead) and moved to an invisible area. After a while, the dead particles will be relaunched as new particles by the particle engine. The generation and death process of the particle system is shown in Fig.4.
Before rendering one frame, the particle system computes the rain particle attributes (position, speed, and, etc.) of the next frame based on particle motion equation, current position, and current speed. Then, rain particles of the next frame will be rendered into the frame buffer according to the new calculated position, speed, and other attributes. By this way, the rainfall rendering process is substantially accelerated.

2.3 Particle motion equation

Without consideration of the role of the wind, the main external forces acting on the falling rain particles are mostly gravity and air resistance.

\[ F_r = G_g - F_a = mg - F_a = ma \]  

(1)

Where \( F_r \) is resultant force, \( G_g \) is force of gravity, \( F_a \) is force of air resistance.

Thus, the acceleration of rain particles is:

\[ a = \frac{mg - F_a}{m} \]  

(2)

Updating speed of rain particles is:

\[ V_t = V_{t-1} + a \Delta t \]  

(3)

Where \( \Delta t = 1 / f_R \) and \( f_R \) is refreshing frequency.

The location of rain particles in time \( t \):

\[ S_t = S_{t-1} + \frac{2V_{t-1} + a \Delta t}{2} \Delta t \]  

(4)

2.4 Particle initialization and texture mapping

The formation of rain particles is one random process. Rain particles are generated by the particle engine at the top of the screen. Before rain particles are launched, their attribute values should be initialized including location, lifespan, initial speed, color, transparency, size, quality, air resistance, gravity acceleration, etc. Corresponding to the attributes of rain particles, the RPARTICLE data structure is defined as follows:

struct RPARTICLE {
float R_Life; // lifespan of rain particles
Vertex3 R_Position; // location of rain particles
Vertex3 R_Velocity; // initial speed of rain particles
Vertex3 R_Forces; // initial forces of rain particles
int R_Size; // size of rain particles
float R_Mass; // mass of rain particles
Vertex3 R_Color; // color of rain particles
float R_Slucency; // transparency of rain particles
float R_Friction; // air resistance of rain particles
};

In order to realize continuous and realistic effects of rainfall, texture mapping technology is adopted to project a flare texture image onto the foreground of 3D terrain scenery. As mentioned before, the rain particles are launched by the particle engine at the top of the screen. While the rain particles are being launched, their \( Y \) coordinates have been determined, but their \( X, Z \) coordinates are random. In order to allow the rain particles to be seen in the process of rendering them, the flare texture image should parallel with the \( XOZ \) plane. We can shorten a little bit of the \( X \)-axis length of the rain particle size to make the flare texture look like a real raindrop, which greatly
enhances visual effect and reality of the rainfall scenery.

3 Experiment and analysis

The software used in the experiment of 3D terrain and rainfall rendering includes Windows XP, Visual Studio 2005, and OpenGL. The experiment is conducted on an ordinary Personal Computer. Hardware configuration is listed as follows:

- CPU: Intel E8400, 3.0 GHz 1333 MHz FSB, 6MB L2 cache;
- Graphics card: NVIDIA GeForce 8800GT, 512M memory size;
- Total host memory: 2.0 GB.

In order to make the rainfall effect look more realistic, a cylinder dome sky with dark clouds drifting is added to the rainfall scenery. Our experiment is carried out on a 512×512 topographic map in DEM format. To prove the superiority of real-time rendering performance of the particle system based on LoD algorithm, we compare the result with that of rainfall rendering with non-simplified particle system. The FPS comparison of the two particle systems is shown in Fig.5.

Experiment results demonstrate that the particle system based on LoD algorithm has better performance than that non-simplified algorithm. When the number of particles reaches 400000, the rendering speed of LoD-based particle system algorithm can still reach as high as 30 FPS.

The real-time rendered 3D scenery is shown in the Fig.6. After the LoD optimization algorithm is applied, the adjacent edges among parcels of the terrain for experiment keep good effect. The rendered 3D scenery has sound visual effect without the sense of flickering. After 10000 particles are added to the particle systems, the experiment not only realizes the rainfall scenery required by debris flow simulation, but the FPS of rendering scenery maintains more than 55 (generally speaking, 30 FPS in 3D scenery rendering will meet the interactive requirement). Experiment results demonstrate that the proposed algorithm is effective for simplifying geometric complexity of 3D scenery without giving rise to obvious distortion. Therefore, the rendered 3D scenery (terrain and rainfall) can be used as natural background for the whole debris flow process simulation.

4 Conclusion

In recent years, the 3D debris flow process simulation is one of the most challenging research fields. Debris flow usually occurs in mountainous areas and is triggered by strong rainfall. Therefore, the undulating terrain and rainfall modeling and rendering are very important for realistic 3D scenery construction in debris flow simulation, which has very high numerical computation and graphical processing capability requirements. On one hand, we apply the LoD algorithm to reorganize DEM-expressed terrain and its geometric complexity has been greatly reduced. On the other hand, we employ LoD-based particle system to render rainfall. The 3D scenery rendering speed was substantially improved. Experimental results have demonstrated the efficiency and visual effect of our proposed approach for 3D scenery modeling and rendering, which lays a solid foundation for the whole debris flow process simulation.
However, the rainfall physical model adopted in this experiment is relatively simple and fails to take into account wind effects and the permeating phenomenon of surface water. In future research, we will construct a new rainfall scenery model that considers these factors. We will also take advantage of the parallel computing capability of GPU technology to establish a liquid-solid two-phase flow model and thereby achieve more efficient simulation of the whole debris flow process in realistic 3D scenery.

References

[1] Tang B X (1994) The flash flood debris flow and landslide and these mountain disasters control [M]. Beijing: Science Press

[2] Liu X, Wang X, Wang G (1999) Three-dimensional visualization of the process simulation of debris flow based on GIS [J]. Advance in Water Science, 10(4): 388-392 (in Chinese)

[3] Zhang S (2009) CUDA, high performance computing of GPU (in Chinese) [M]. Beijing: China Water Power Press

[4] William T R (1983) Particle systems—A technique for modeling a class of fuzzy objects [J]. ACM Computer Graphics, 17(3): 359-376

[5] Zhang W, Su H (2004) Research on LoD algorithm of 3-D models and its implementation with OpenGL [J]. Computer Engineering and Design, 25(10): 1790-1796 (in Chinese)

[6] Yang Z, Ke X, Peng Q (2008) Realistic modeling and rendering of multi-phase flow catastrophic scenes [J]. Journal of Computer-Aided Design & Computer Graphics, 20(8): 1023-1032 (in Chinese)

[7] Hoppe H (1998) Smooth view-dependent level-of-detail control and its application to terrain rendering [C]. IEEE Visualization 1998 Conference, Los Alamitos, USA

[8] Lindstrom P, Pascucci V (2002) Terrain simplification simplified: a general framework for view-dependent out-of-core visualization [J]. IEEE Transactions on Visualization and Computer Graphics, 8: 239-254

[9] Lindstrom P (2000) Out-of-core simplification of large polygonal models [C]. SIGGRAPH 2000: 27th International Conference on Computer Graphics and Interactive Techniques, New Orleans, USA

[10] Peter L, David K, William R, et al. (1998) Real-time, continuous level of detail rendering of height fields [C]. SIGGRAPH 96: 23rd International Conference on Computer Graphics and Interactive Techniques, New Orleans, USA

[11] Slasallek P, Seidelh P (1998) Real-time generation of continuous levels of detail for height fields [C]. Winter School of Computer Graphics Conference 1998, Plzen, Czech. Republic

[12] Cignoni P, Ganovelli F, Gobbetti E, et al. (2003) Planet-sized batched dynamic adaptive meshes (P-BDAM) [C]. The 14th IEEE Visualization 2003 (VIS’03), Seattle, USA

[13] Pajarolar R, Antonijuan M, Mlario R (2002) QuadTIN: quadtree based triangulated irregular networks [C]. The Conference on Visualization ’02, Boston, USA

[14] Trent P (2002) Focus on 3D terrain programming [M]. Ohio: Premier Press

[15] Lan H, Zhou C (2007) A literature review on debris flow constitutive model and its dynamic simulation [J]. Journal of Engineering Geology, 15 (03): 314-321 (in Chinese)