Journal of Theoretical and Applied Information Technology
30th April 2013. Vol. 50 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ACCELERATE TERRAIN RENDERING USING PROGRAMMABLE GRAPHICS HARDWARE

[1]ZHISHENG MA, [2]MING SUN, [3]GUANGJUN SONG

[1]Lecturer, Computer Center, Qiqihar University, China

[2]Assoc. Prof., College of Computer and Control Engineering, Qiqihar University, China

[3]Prof., College of Computer and Control Engineering, Qiqihar University, China

E-mail: [1]xiaomageleiluo@163.com

ABSTRACT

Multiresolution representations are often used in terrain rendering systems to increase system performance. To avoid popping effects, these systems usually utilize geomorphing technique, which may be presented as the bottleneck of the system. With the recent advance of graphics hardware, that is, the programmability, we try to decrease the CPU load by shifting the geomorphing from CPU to GPU. But this doubles the data amount that needs to be transferred through graphics bus. By carefully organizing terrain data in compact format, we can reduce the data amount to about 40% of the original. These techniques help us to accelerate the rendering process about 30% to 100%, compared with standard OpenGL implementation.

Keywords: *geomorph, GPU computing, terrain rendering*

1. INTRODUCTION

Terrain rendering is an important part of many applications such as 3D games, flight simulations and geographical information systems. Due to the large amount of data, it often presents as a bottleneck of the whole system when high quality rendering is required.

In recent years, the graphics hardware has been experiencing a rapid progress. Now mainstream graphics adapters can process over tens of millions triangles per second, and most importantly, they allow programmers to write programs to control the rendering process, which was fixed before. This enables us to shift some computations from the CPU to the much faster GPU to decrease the CPU load and thereby increasing the performance of the whole system.

Another issue to be concerned is that current generation graphics adapters have rasterization performance that far outstrips the bus bandwidth available to feed them [1]. So transferring geometry data in a compact mode through graphics bus often gains better performance.

In this paper, we focus on a specific case of terrain rendering, geomorph in view dependant systems, and try to accelerate the rendering performance using current programmable graphics hardware. By shifting geomorph computation to GPU and transferring data in a compact format, we gain 30%-100% more performance compared with the standard OpenGL implementation.

This paper is organized as follows. We first review previous work on terrain rendering and recent progress of graphics hardware in section 2. Then we present our improvement on geomorphing rendering in section 3. Section 4 describes some implementation details and shows our experiment results. Then we make a conclusion in section 5.

2. RELATED WORK

Terrain rendering has been thoroughly investigated in the last decade [2, 3, 4, 5]. These algorithms seek to explore the potential of view dependent multiresolution geometric representation to reduce the scene complexity for an increased frame rate. We can roughly classify them into two categories: static level of detail algorithms and continuous level of detail algorithms. Static LOD algorithms represent terrain data in several fixed resolution level and divide each level into blocks offline. At runtime, blocks of appropriate level are selected based on viewing parameters. Continuous LOD algorithms can dynamically add or remove individual triangles according to viewing parameters or other time-varying criteria at runtime [3, 4, 5]. Compared with Static LOD, continuous LOD are more sophisticated and often need fewer

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

triangles to provide the same rendering quality, but consume more CPU resources. At the late '90s, when CPU resource is more available than GPU's, this type of algorithms are widely used. While with the recent rapid progress of graphics hardware, now it usually takes longer time to reject triangles than to render them out, so block based algorithms often gain more performance [6].

Geomorph is widely used in multiresolution rendering systems to reduce the popping effects that may occur when the geometry models switch between different resolutions. Instead of replacing the previous representation of models with the new one at a sudden, geomorph interpolates between the two successive representations according to time or other factors, so the geometry models can transit smoothly and cause no visual unpleasing.

Today, more and more modern graphics hardware adapts to programmable architecture and allows programmers to write programs or scripts to control the rendering process. Newly version 3D graphics APIs such as OpenGL and DirectX provide language level support to this programmability, say, OpenGL Shading Language and High Level Shading Language, respectively. NVIDIA Corporation also promotes a language, C for graphics, to facilitate 3D programming on their product.

3. ALGORITHM DESCRIPTION

Terrain can be seen as a continuous function over X-Z plane, and the terrain data we concern is the values of the function sampled on regular grid over the X-Z plane. Each node of the grid can be represented using two integers indicating the row and column number of the grid where it resides, we call these two numbers grid coordinate. We also record the color of each sampling points, and the normal, if lighting effect is required.

For static scene or the scene that geometry models do not change frequently, display lists are preferred to increase rendering performance. But as long as multiresolution representations of models are used, especially when the network scenarios are concerned, in which the terrain data is transferred over network in a progressive way, the scene may be updated very frequently, and even worse, with the utilization of geomorph, geometry may change every frame, thus display lists are no longer suitable. We have to find other ways to increase the rendering performance.

We observe that, in geomorph rendering system, the vertex morphing operation presents as bottleneck of the system since lots of floating-point operations are involved. To overcome this bottleneck, we shift the morphing operation from CPU to GPU. That is, instead of doing morphing on CPU and then transferring the result of calculation to GPU as the input of graphics rendering pipeline, we transfer both the properties of original samples and those of target samples to GPU and perform morphing operation in vertex shader. This step frees the CPU load but doubles the amount of data that needs to be transferred from main memory to GPU, as a result, the bus bandwidth between them appears to be new the bottleneck of the system. To further improve system performance, we take advantage of the regularity of terrain data and represent terrain data in a compact mode while transferring. As mentioned above, the terrain data used in our system including positions, colors and normals of each samples. In usual representation, each position and normal contains three floating-points, which is a large fraction of data compared to color representation that takes three bytes. So we will concentrate the compact representation of these two components.

3.1 Compact Representation of Vertex Positions

Since the terrain data is sampled on a regular grid, we can separate the position of each sample into two parts, sample's location and sample's value, corresponding to the location on the 2D grid and the height field, respectively. The sample's locations are represented using grid coordinate. We also divide the grid into blocks as most static LOD algorithms do. Each block is identified by the grid coordinate of its left-top corner, which we called block identifier. Then each sample's location can be identified by the identifier of the block that it belongs to and its offset in the block relative to the left-top corner of the block. We call this offset block coordinate. We can constrain the size of the block to less than 256x256, so the block coordinate can be represented using two bytes, one for row, and the other for column. Rendering is performed in a blockwise manner. For each block, we transfer its identifier only once, since it is shared by all samples that lie in it. And then, each height field and its corresponding block coordinate are transferred to GPU, in which they are assembled to the usually-used three floating-point format. When geomorphing applies to terrain data, since the corresponding samples' locations are the same, we only need transfer another height field to GPU instead of both height field and block coordinate.

If the height field is represented using 32-bits IEEE floating-point, for a block contains n samples'

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

location, we only transfer n*4 bytes for original height field, n*4 bytes for target height field, 2*n bytes for block coordinate and 2 bytes for block identifier, instead of 3*n*4 bytes for original position and 3*n*4 bytes for target position. So about 50% bandwidth is saved.

The x and z coordinate of the vertex assembled in GPU are a sample's grid coordinate instead of its real coordinate. Fortunately, since the terrain is regular sampled, its grid coordinate and its real coordinate only differ by a scale factor. We can integrate the scale factor into model transform matrix. Since the matrix will always apply to vertices that enter the graphics rendering pipeline, this integration will not introduce additional calculation.

3.2 Compact Representation of Vertex Normals

Deering [7] argued that the usual three floating-points representation for normal far exceed the actual requirement for rendering purpose. He arrived at the result that an angular density of 0.01 radians between normals gave results that were not visually distinguishable from finer representations. This meant only 100,000 normals that distributed over the unit sphere were needed. By extensively taking advantage of the symmetry property of the unit sphere, Deering used 17 bits to represent these normals, 11 bits to index into a look-up table and 6 bits to indicate one symmetric region. We take Deering's idea but implement it in an easier way, by taking graphics hardware programming and the special characteristic of terrain data into consideration.

For terrain data, it is easy to deduce that their normals are surely distributed over the upper part of the unit sphere. We can parameterize this hemisphere over a square region to create a normal texture (Fig. 1), serving as a look-up table, then each normal is represented by its coordinate in the normal texture. To recover the normals' three components, we perform looking-up operations in GPU using texture mapping hardware. Since the parameterization is continuous and has no singularity, we can interpolate over normal texture coordinates before the looking-up operation while performing rasterization, this helps us to achieve high quality images close to those that shaded by phone model.

This differs from Deering's algorithm since Deering cared about more general cases instead of terrain data, which meant the normals of the geometry models might be distributed over the whole unit sphere. Singularity is unavoidable when a sphere is parameterized to a square region. So interpolation is impossible.

By experiments, we find that it is enough to use a 256x256 normal texture to render high quality images that not visually distinguishable from finer representations, which means we can represent a normal by two bytes. This result also coincides with Deering's conclusion, since he argued to represent normals that distributed over the whole unit sphere using 17 bits, and we represent normals over hemisphere using 16 bits.

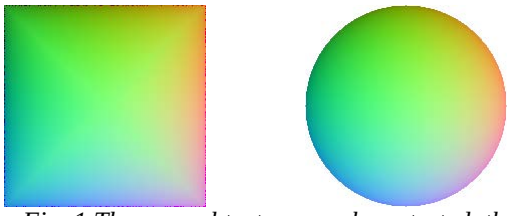
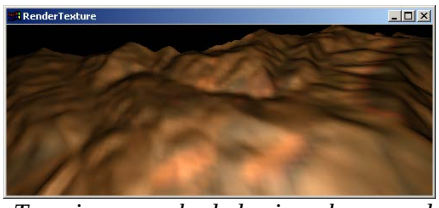
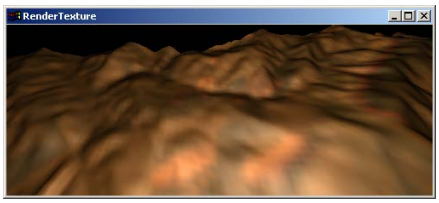


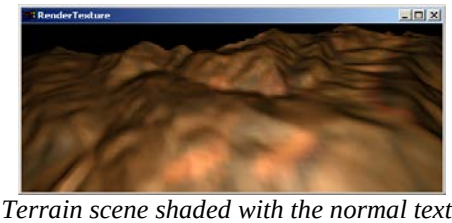
Fig. 1 The normal textures we have tested, the left one is created by the parameterization proposed by Shirley, and the right one is created by orthotic projecting the hemisphere along y–axis. The x, y, z component of the normal is represented by the r, g, b color respectively



a. Terrain scene shaded using phone model



b. Terrain scene shaded with the orthotic projection normal texture



c. Terrain scene shaded with the normal texture that parameterized by Shirley's method. Slight artifact can be seen at the left-bottom of the image

Fig. 2 Images that shaded with different normal textures

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Another issue is about the mapping function between the hemisphere and the square. We test two functions, one was introduced by Shirley [8] and another is simply orthotic projecting the hemisphere to the square along y-axis. The parameterization results are shown in Figure 1. Using them as normal textures, we render terrain scene and compare the rendering result with phone shading (Fig. 2). We find that orthotic projecting normal texture works well while Shirley's introduce slightly artifacts where height field changes rapidly. This may be caused by the stretch at the corners of the square. Orthotic projection is non-uniform, since that normals that parallel or near parallel to y-axis are over sampled while those that perpendicular or near perpendicular to y-axis are under sampled. But this seldom affects the rendering result, since the normals of the terrain data tends to distribute along y-axis and with the only exception at cliffs.

The normal map is transferred to GPU when the system starts up. While rendering, each normal is represented using two bytes instead of three floating-point numbers. This drastically decreases the data amount that needs to be transferred.

4. EXPERIMENT RESULTS

We have implemented several versions of terrain rendering system for comparison (Table 1 and Table 2). All the versions are implemented using OpenGL and Cg, if vertex shader programs and/or pixel shader programs are needed. The OpenGL versions use data arrays for efficiency (VertexArray, ColorArray and NormalArray) instead of display lists, since we assume terrain data updates frequently when geomorphing is applied, so precompiling schemes will not work. Another to mention is that the version of Cg we use does not support byte format while transferring, so we have to represent block coordinates, normal texture coordinates and colors using short int (two bytes) format. This doubles the bandwidth needed to transfer those data, but we still get improvement on performance.

Table 1 and 2 shows comparison among OpenGL standard implementation, Cg implementation with data transferred in normal format and Cg implementation with data transferred in compact format. The first row is bytes per vertex that needs to be transferred from CPU to GPU. The second row is time consumed to render each frame. The third row is triangle count that the system processes per second. The data are obtained on a Pentium IV 2.4 GHz PC, with a GeForce FX5600 Ultra graphics card. The terrain data we use contains 2,000,000 triangles.

Table 1 Statistics when render the scene with color

	OpenGL	Normal Cg	Compact Cg
Bytes per sample	15	36	24
Time per Fram (sec/frame)	0.2407	0.1432	0.1178
Throughput (M tri/sec)	8.309	13.966	16.978

Table 2 Statistics when shading the scene with light

	OpenGL	Normal Cg	Compact Cg
Bytes per sample	27	60	32
Time per Frame (sec/frame)	0.2923	0.2582	0.2000
Throughput (Mtri/sec)	6.842	7.746	10.000

Table 1 compares rendering performance when the terrain is rendered with color. Standard OpenGL implementation morph height values and colors in CPU, so CPU presents as a bottleneck. Cg implementations perform vertex morphing in GPU, then the bottleneck appears on graphics bus. So when we transfer data in compact format, we see improvement on performance.

Table 2 compares rendering performance when the terrain is shaded by light. We still need the color information and serve it as the corresponding sample's material. The normal texture is selected into graphics memory just like ordinary texture maps and interpolation scheme are applied. In pixel shader, we perform texture mapping and obtain each pixels's normal for lighting calculation. We can see improvement of performance over standard OpenGL version.

5. CONCLUSION

In this paper, we concentrate on acceleration techniques of those terrain rendering systems that multiresolution representation and geomorphing are concerned. Our algorithm has following features:

1) We perform vertex morphing in GPU to decrease CPU load.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2) By dividing terrain data into blocks, we can represent the samples' location in the local coordinate relative to each block, so each coordinate can be identified by one byte.

3) We create a normal texture and use normal coordinates to represent normals. Since the normals of terrain data are restrict to the upper part of the unit sphere by nature, there is no singularity on normal texture and we can interpolate over it freely.

REFERENCES:

[1] H. MORETON, "Watertight Tessellation Using Forward Differencing," *In SIGGRAPH / Eurographics Workshop on Graphics Hardware*, 2001, pp. 25-32.

[2] John S. Falby, Michael J. Zyda, David R. Pratt and Randy L. Mackey, "NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation," *Computers and Graphics*, Vol. 17, No.1, 1993, pp. 65 - 69.

[3] Peter Lindstrom, David Koller, William Ribarsky, and Larry F. Hodge, "Real-Time, Continuous Level of Detail Rendering of Height Fields", *ACM SIGGRAPH 96 Proceedings*, August 1996

[4] M. A. Duchaineau, M.Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing Terrain: Real-time Optimally Adapting Meshes", *IEEE Visualization '97*, 1997, pp. 81-88.

[5] H. Hoppe, "Smooth view-dependent level-of-detail control and its application to terrain rendering", *IEEE Visualization '98*, October 1998

[6] W. de Boer, "Fast Terrain Rendering Using Geometrical MipMapping", October 31, 2000 http://www.flipcode.com/tutorials/geomipmaps .pdf

[7] M. Deering, "Geometry Compression", *Proc. of SIGGRAPH'95*, 1995, pp. 13-20.

[8] Peter Shirley and Kenneth Chiu, "A low distortion map between disk and square", *Journal of Graphics Tools*, Vol. 2, No. 3, 1997, pp. 45-52.