AN IMPLICIT AND GPU BASED APPROACH FOR SOLVING CRACKS IN TERRAIN VISUALIZATION

Cao Wei[1] Duan Fuzhou[1] Gong Huili[12] Zhao Wenji[12]

(1 Capital Normal University Beijing China 100048,

2 Key Lab of 3D Information Acquisition and Application MOE, Beijing China 100048）

Efficient interactive visualization of large-scale terrain has played an important role in many application domains, such as geology, digital city, tourism, disaster relief and military command. However with the birth of varieties of advanced equipments and technologies in spatial data acquisition, the terrain resolution is higher and the storage is larger which force the terrain visualization face to great challenges. Because the huge storage of terrain will contain hundreds of thousands of samples to construct the terrain mesh, the size of which has greatly exceeded the rendering capabilities of commodity PCs. So when rendering large scale of terrain, the frame rates are always very low and many interactive operations could not be completed smoothly which will bring bad experiences to clients. To solve the problems, researchers present some solutions that first partition the large terrain into small tiles and then make specific simplifications to each tile and finally render them respectively. These approaches improve the performance of visualization but will induce terrain cracks and decrease the fidelity of the scene. Though some algorithms are proposed to fix or stitch the cracks, they haven’t achieved absolutely success either by the complexity or by the bad effect.

In this paper, an easy to implement approach will be introduced, and the effect to solve cracks of this approach is perfect. The premise of our approach is that the size of height fields is equal to $(2^n+1)\times(2^n+1)(n>1)$. The approach is processed as follows:

1) Partition the terrain into triangular tiles by triangular bisection, and the newly produced triangular tiles will construct a triangle bintree. The large triangular tile is at the high level, and the small triangular tile is at the low level. The lowest level is 0.
2) Split each triangular tile produced at step 1 into same amount of triangles, and the amount is equal to $2^m(m<=n)$ which will divide the edge of each tile into same sections.
3) At runtime we use terrain simplification rules to select specific tiles with different size and force the neighboring tiles that are either from the same level L, or from the next lower level L-1 for the left and right neighbors, or from the next higher level L+1 for the

base neighbors. This will force the neighbor tiles to be connected seamlessly.

As in the above process, we just use an implicit method to solve the cracks, and there will not be any extra cost involved while implementation. So the performance will not be affected.

Another improved aspect of our approach is that we utilize the programmable pipeline of GPU (Graphics Processing Unit) to accelerate the rendering. The process is as follows:

1) We transform the normal terrain data into a single channel floation-point texture, so that each texel will contain a height value. And we call this texture as vertex texture.
2) Send the texture to GPU so that the height data will reside in video memory rather than in system memory and the limited storage of system memory will be saved a lot.
3) To obtain the height value, we just use the row-column value for the specific texel and choose a proper sampler to fetch the value from the vertex texture. So we just store the row-column value as integer for each sample point in system memory which will save more storage than storing the whole value as (X, Y, Z).
4) To obtain the real plane value (X, Y) for each sample point, we also utilize the row-column value and just make some transformations such as shifting and scaling for the up-left specified point.
5) While rendering we arrange the mesh in triangle strips so that we can obtain spatial and temporal coherency, and the vertices data can be efficiently reused on subsequent frames.

The above operations of step 3 and 4 are used to be done by CPU, but in our approach all these operations are handled in GPU and processed in vertex shader. We will write the specific operation code in HLSL (High Level Shader Language) which is compiled on GPU. So the high performance of GPU is utilized effectively and the great pressure on CPU will alleviate a lot.

We choose the Island of Hawaii terrain datasets with size of 2049×2049 for testing on Intel Core2 Duo CPU with 2GB memory and ATI Radeon HD 2400 PRO with 256M video memory. From the results we could not see any cracks, and each tile is seamlessly connected together. The interactive frame rates are stable and have reached to more than 35pfs which is twice of the minimum frame rates 16pfs that the eyes demand.

Keywords—Terrain rendering; LOD (level of detail); crack; GPU