Mix-subdivision Dynamic Terrain Visualization Algorithm

Xiaohui Zhao, Baodi Xie, Depeng Wan, Qingyun Wang
Department of Computer Sciences, Beijing Institute of Technology
Beijing, 100081, China
E-mail:kingzhaoliu@gmail.com

***Abstract*—Dynamic Terrain is becoming more and more important in ground-based simulation systems. In military simulation systems, craters and ruts can improve the reality. In this paper, a dynamic terrain visualization method based on quadtree and multi-resolution voxel is presented in order to realize the real-time rendering for realistic craters in battlefield. Quadtree is selected as our basic data structure and mix-subdivided according to the size of the terrain. Scene tree is recursive subdivided according to both the distance between the node and camera and error criterion. Vertex is removed to solve the cracks and linear interpolation to solve popping in the algorithm. We also implement the visualization of craters through combining our algorithm with the physical model of craters based on multi-resolution voxel. The implementation results prove that the method are feasible and efficient.**

Keywords-quadtreet; multi-resolution voxel; mix-subdivision; dynamic terrain

I. INTRODUCTION

As an important part of Terrain Simulation, Dynamic Terrain has become an increasingly important requirement in many fields, such as battlefield simulation, military training system, games. For Example, in battlefield simulation, we usually need simulate craters, tracks of tank, footprints and so on. Some simulation systems use picture of crater instead of real deformation of terrain surface, it's simple but it's useless to improve the realistic effect of simulation. So we present a method to implement dynamic terrain, especially the simulation of crater.

Based on quadtree, our algorithm part the terrain into a series of patches, then subdivide the patch into different LOD triangles. Then compute the number of patches the terrain has per side. According to the distance from the camera eye position to the current patch's center, every patch has corresponding LOD and is broken down into several triangles. At the same time, every triangle need keep neighbor information, by comparing the LOD of current triangle and its neighbor, we omit some vertices to eliminate crack between different LOD patch. In order to reduce popping, we interpolate several values from old value to the current true value. Furthermore, we compute terrain deformation by combining multi-resolution voxel with the physical model of craters to visualize the craters in the terrain.

In this paper, we mainly introduce optimized CLOD terrain algorithm and model of crater-terrain interaction and implement the visualization of craters.

II. PREVIOUS WORK

Some methods for dynamic terrain have been presented, such as ROAM(Real-time Optionally Adapting Meshes), QuadTree, GeoMipmap[1], View-Dependent Progressive Mesh, DEXTER-ROAM. These algorithms render terrain with multi-resolution regular meshes based on some error metric criteria.

Duchaineau[2] present an algorithm for constructing triangle meshes that optimizes flexible view-dependent error metrics, produces guaranteed error bounds, achieves specified triangle counts directly, and uses frame-to-frame coherence to operate at high frame rates for thousands of triangles per frame. The algorithm uses two priority queues to drive split and merge operations that maintain continuous triangulations built from pre-processed bintree triangles. The algorithm also introduces two additional performance optimizations: incremental triangle stripping and priority computation deferral lists. ROAM execution time is proportionate to the number of triangle changes per frame, which is typically a few percent of the output mesh size, so ROAM performance is insensitive to the resolution and extent of the input terrain. Dynamic terrain and simple vertex morphing are supported. But the algorithm is designed to put most of the workload on the CPU, so it is not adapted to the development of GPU.

Hoppe[3] presents the progressive mesh representation, it's a new scheme for storing and transmitting arbitrary triangle meshes. The representation addresses several practical problems in graphics: smooth geo-morphing of level-of-detail approximations, progressive transmission, mesh compression, and selective refinement. Hoppe also presents a new mesh simplification procedure for constructing a PM representation from an arbitrary mesh.

He[4] et al present a method for multi-resolution view-dependent real-time display of terrain undergoing on-line modification by extending ROAM (Real-time Optimally Adapting Meshes) with efficient hierarchy updates as terrain deforms, and using DEXTER (Dynamic Extension of Resolution) to provide only-where-needed memory efficient resolution extension. But their approach only united with fake properties of terrain deformation and did not combine the physical-based-model of terrain in the real world. They only dealt with relatively small scale terrain inputs.

III. CLOD TERRAIN ALGORITHM

In order to render terrain fast, we present a multi-resolution algorithm, based on the data structure of quadtree. We combine view-dependent with view-independent methods to decide which LOD the terrain block should select and subdivide the block to the finest level of detail. Next, we remove the Cracks by omitting vertex along the render region boundaries and reduce poppings by interpolating several values from old value to the current true value.

A. Data Structure

We choose quadtree as the basic data structure because the tree structure is good for frustum culling. For Convenience, we only consider the special case of terrain of size $(2^n+1)*(2^n+1)$, the horizontal and vertical number of vertices in the terrain block must be 2^n+1.

We pre-subdivide the total terrain to a series of patches evenly. Figure 1 shows the situation when the terrain is composed of 4x4 patches(5x5 vertices). The number of the patches is concerned with the total terrain size but it had better not exceed 17x17 in order to avoid excessive subdivision. The preprocessing construct the quadtree from root node to $\log_2 N$ level (N is the number of the patches), the levels are necessary and static, so the time for constructing the tree is reduced because we needn't construct these levels every frame. For example, Figure 2 shows the level 0 to level 2 are pre-constructed.

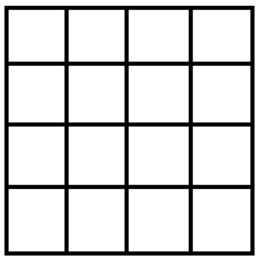


Figure 1 Pre-subdivision of the terrain

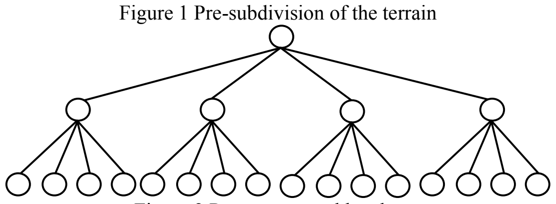
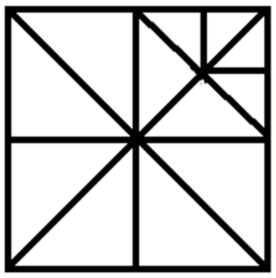


Figure 2 Pre-constructed level tree

Then, we continue to subdivide every patch unevenly according to which level of detail the patch need. The patch is represented by part of a 0-1 matrix which is composed of every little block's center entry set. 1 means the block needs subdivision, 0 means the oppsite, -1 means nothing. The quadtree matrix of a sample triangulation generated is shown in Figure 3.



$$\begin{bmatrix} -1 & -1 & -1 & -1 & 0 \\ -1 & 0 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 \\ -1 & 0 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Figure 3 Quadtree matrix of a patch

B. LOD Selection and Rendering

At run time, we choose appropriate subdivision level for every block that represents each part of the terrain to meet desired visual effect. In the paper, view-dependent and view-independent LOD algorithms are both used for terrain subdivision and coarsening.

View-dependent method is related with the distance from the center of block to the camera point. The longer the distance is, the lower the level is. Formula 1 shows the criterion. L is the distance, and d is the edge length of the block. C controls the global resolution.

$$\frac{l}{d} < C \tag{1}$$

View-independent method is related with the terrain surface roughness. The abrupt region has high level while the flat region has relative low level. Formula 2 shows the criterion. D is the edge length of the block. dh is the difference in height between the true elevation and the current height of the vertex. i is 1 to 6, represents the six points: the four midpoints of the block's edges and the two midpoints of the diagonals. The sum of the six errors determines the level of the block. When error > D_n, subdivide the block until to the desired resolution.

$$error = \frac{\sum_{i=1..6} dhi}{d} \tag{2}$$

C. Crack and Popping Removal

During the generation of the triangles we need to decide whether adjacent nodes are subdivided to the same level or not. When there are different levels between current node and its neighboring node, crack appears as Figure 4 shows. Instead of adding vertex or line, we use a relative simple method by omitting the center vertex at shared edges to remove the crack. Figure 5 shows the result. But at the same time we must guarantee that the levels of adjacent nodes differ by no more than 1. So we use the method in Rottger's paper to get it.

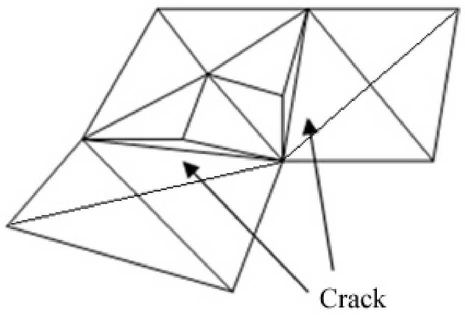


Figure 4 Crack

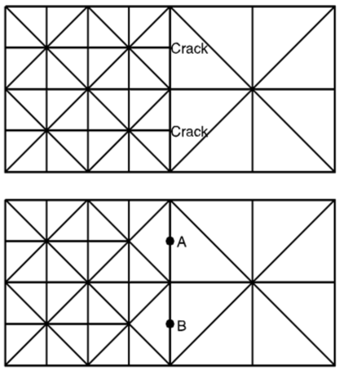


Figure 5 The removal of cracks

D. *Frustum Culling*

The main advantage of the tree data structure relies on the fact that if the parent node is culled so the child node needn't be examined any more when frustum culling as Figure 6 shows.

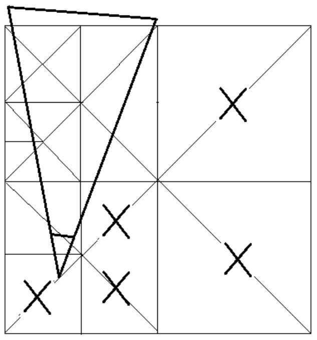


Figure 6 Frustum Culling

IV. Crater-Terrain Model And Visualization

Real terrain usually has many complex properties[5], such as rigidity, flowability, compressibility. So we take these elements into account when computing the deformation of the terrain surface. And we present a terrain model of multi-resolution voxel by using a series of voxel to represent the terrain.

The algorithm is:

a. Get the terrain height data. Initialization the voxel as Figure 7 shows.
b. Detect collision between the object and terrain surface, mark the direct and indirect affected block. Subdivide the voxel.
c. Compute the Deformation of the direct affected blocks according to different terrain properties.
d. Distribute certain deformation to indirect affected blocks what are in the collision edge.
e. Using Corrosion calculation to smoothing the surface of the transformative edge.

In step b, we detect collision between the object and terrain surface using bounding box. If bounding box of the terrain block and bounding box of the object are interactive, we mark the terrain block. And then update the height data of the block's vertex according to the deformation computation.

We classify the blocks into three types according to the projection of the object's bounding box on the terrain block. If the projection covers the whole voxel, it's fully collision voxel; if the projection cover the part of the voxel, it's partly collision voxel; as to the no collision voxel, if there are partly collision voxel as its neighbor, the voxel is edge voxel. Different types of voxel have different resolution as Figure 8 shows.

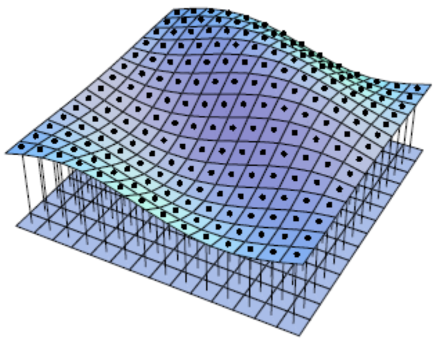


Figure 7 Initialization of the voxel

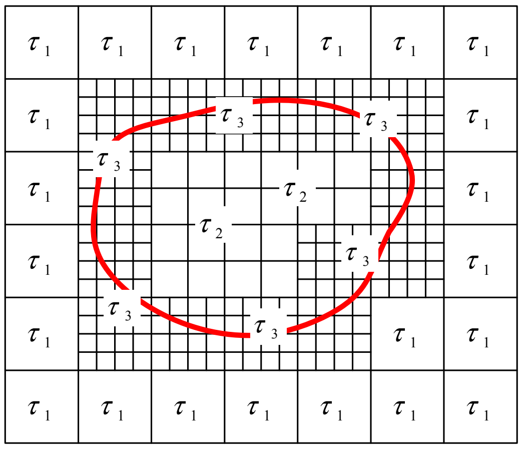


Figure 8 Multi-resolution voxel

τ_2 is the resolution of the fully collision voxel, $\tau_3$is partly collision voxel's, $\tau_1$is edge voxel's.

Travese all the voxel of resolution τ_2, get the deformation of every voxel,Δy_2. The new height of the voxel is computed by Formula 3.

$$y_2' = y_2 - \Delta y_2 \tag{3}$$

As to the voxel of resolution τ_3, their deformation is influenced by both the partly collision of the object and the distribution of the neighboring voxel of resolution τ_2. So their deformation is as Formula 4.

$$y_3' = y_3 - \Delta y_3 + k\sum \Delta y_2 \tag{4}$$

As to the voxel of resolution τ_1, their deformation is influenced by the distribution of the neighboring voxel of resolution τ_3. So their deformation is as Formula 5.

$$y_1' = y_1 + k\sum \Delta y_3 \tag{5}$$

After Deformation Distribution, the edge voxel maybe a little rough, we smooth the terrain by Corrosion calculation. Corrosion calculation reduces the height of the edge voxel and distributes them to out neighboring voxel to maintain the angle of tilt between neighboring voxels in certain range. Angle of tilt between two neighboring voxel V1 and V2 can be get by Formula 6. y1 and y2 is the height data, and d is the horizontal distance between the center of the two voxel.

$$\theta = \arctan(\frac{y_1 - y_2}{d}) \tag{6}$$

We set different threshold, θ_t , for different kind of terrain, shown in chart 1.

TABLE I. THRESHOLD FOR TERRAIN

Rock field	Sandfield	Grassland	Snowfield
0.01	0.8	0.88	1.57

Repeat to do corrosion calculation until $\theta \le \theta_t$.

V. RESULT

All screen shots shown here have been taken from the application running on a Intel Core i5 3.10G computer, 2GB RAM and NVIDIA GeForce GT 220 graphics card, under Windows 7, Visual Studio 2008, OpenGL environment, while running smoothly in real time.

One grassland and crater are simulated on the field caused by attack on it. Figure 9 shows the terrain generated by the mix-subdivision algorithm, and Figure 10 is the corresponding wireframe presentation. We show the crater in Figure 11 and Figure 12. The simulation of the application is fluent, and it's realistic.

The result indicates our method is more accurate and efficient the general algorithm based on quadtree.

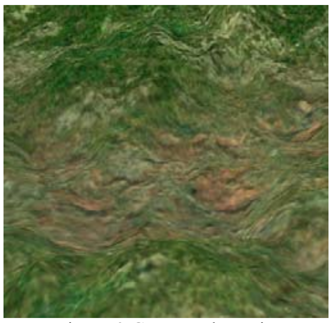


Figure 9 Generated terrain

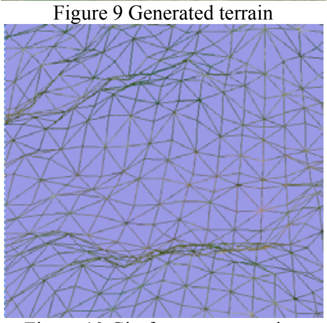


Figure 10 Gireframe presentation

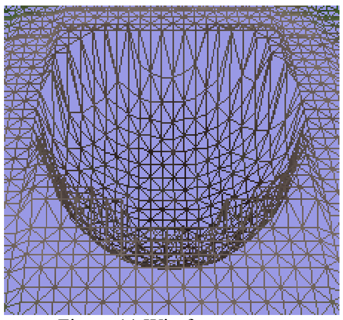


Figure 11 Wireframe crater

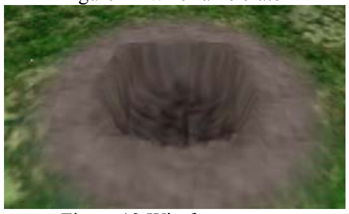


Figure 12 Wireframe crater

VI. CONCLUSION AND FUTURE WORK

An efficient mix-division terrain CLOD method is proposed based on quadtree what enables real-time visualization of large-scale terrain. An efficient terrain model is also given base on real terrain properties for crater interaction.

As a future possibility, we will balance the overhead between CPU and GPU, and optimize our dynamic terrain algorithm and physical-based-terrain-model to get more realistic simulation.

ACKNOWLEDGMENT

We would like to thank those who give us some advice and help. We would like to thank Virtual Terrain Project Website where we get good collections of terrain resources.

REFERENCES

[1] Willem H. de Boer. Fast Terrain Rendering Using Geometrical MipMapping, 2000

[2] Duchaineau M, Wolinsky M, et al. ROAMing Terrain: Real-time Optimally Adapting Meshes. In Proceedings of IEEE Visualization 1997, 1997: 81-88

[3] Hoppe H. View-dependent refinement of progressive meshes. In Proceedings of SIGGRAPH 1997, 1997: 189-198

[4] Huaiqing He, Chong Wang, et al. An Improved Approach on Visualization of Large-Scale Terrain Surface. In Proceedings of CGIV, 2007. 2007:481-488

[5] Xingquan,Cai, Jinhong Li, A-Real-time Visualization of Dynamic Terrain in Off-road Driving Simulation