Real-Time Visualization of Tire Tracks in Large Scale Dynamic Terrain

Yunan Zhang, Dong Wang, Nanming Yan and Yinghui Shang
Department of Control Engineering, Academy of Armored Force Engineering,
Beijing 100072, China
wangdongxy@hotmail.com

***Abstract*—In off-road simulation, the terrain is being modified as a result of its interaction with the vehicles. Previous methods just deal with relatively small scale terrain inputs. In this paper we describe a large scale dynamic terrain visualization method. The potentially visible portion of the terrain is cached in the form of texture arrays and is rendered from the GPU. The terrain deformation is generated in the fragment shader using the GPU-based terrain deformation algorithm. The deformed terrain heightmap texture is rendered to the corresponding layer in the texture array by the Framebuffer object extension. The CPU sends tiled quadtree flat grid, which was displaced by fetching the heights stored in the GPU Cache. The GPU cache is updated continuously as the viewpoint changes. Experimental results showed that the method is very efficient and suitable for applications of massive terrain tire tracks deformation.**

Keywords-dynamic terrain; texture array; tiled quadtree; terrain visualization

I. INTRODUCTION

The real-time visualization of the terrain plays an important role in computer graphics, three-dimensional geographic information system, virtual reality and simulation. Ground vehicle simulation belongs to the type of applications that reproduce real world human experience, and it is very important that the system supports user-simulator interaction.

We know that the presence of vehicles affects the soft terrain and tire-tracks will appear, explosions also deform the terrain. We refer to the type of terrain surface whose geometric and other properties may change during the process of the application as dynamic terrain [1].

Based on our previous work on terrain deformation algorithm, we use tiled block quadtree grid and texture array introduced in Shader Model 4.0 to render tire tracks effect in large terrain heightmap, such as 16k × 16k Puget Sound terrain data.

The remainder of this paper is organized as follows: In Section 2 we review some related work by previous researchers. Then, in Section 3, we review our previous terrain deformation technique. The large scale dynamic terrain rendering algorithm is explained in section 4. At the end of this paper, we give the results.

II. RELATED WORK

Although many existing terrain visualization algorithm focus on static terrain rending, there are still a few methods used for dynamic terrain.

Li and Moshell developed a model of soil that allows interactions between the soil and the blades of digging machinery [2]. Their technique is physically based and they arrive at their simulation formulation after a detailed analysis of soil dynamics. Sumner et al. [3] proposed an appearance-based solution for the display of dynamic terrains. They use a four step execution cycle to create a visually-convincing depiction of terrain surface interactivity. Their method needs to manually adjust rendering parameters to produce a visually-convincing image. The need for manual adjustments suggests that this technique may not be suited for an interactive system.

He et al. [1] extended the ROAM (Real-Time Optimally Adapting Mesh) algorithm that adds multi-resolution support. Their system DEXTER (Dynamic Extension of Resolution), dynamically extends the geometry hierarchy only where necessary. This method is a milestone in the dynamic terrain visualization. The problem with this approach is that the solution presented is only suitable for small, local updates deformations. Wang et al. [4] proofed the maximum extension of transition region based on DEXTER. The Dynamic Extension to ROAM was also extended to offer preservation of vertex properties and relationships with the use of a Direct Acyclic Graph (DAG) [5]. Cai et al. [6] presented a hybrid multi-resolution algorithm for dynamic terrain visualization method using strip masks.

With the development of the graphic hardware, in order to make use of the feature of the latest graphic process unit, Anthony S. et al. [7] presented a new GPU-based terrain deformation algorithm for dynamic terrain simulation. However, their approach needs vertex texture fetch operations twice each frame in the vertex pipelines; moreover, the method is relatively complicated that makes it suboptimal.

Wang et al. [8] improved the Anthony S' algorithm, but still using the brute force algorithm to render the deformed terrain heightmap texture. So the terrain area size to be deformed is also restricted.

III. GPU-BASED TERRAIN DEFORMATION ALGORITHM

First of all, translate the heightmap representing the original terrain height to the initial terrain depth texture directly, Fig.1(a). Secondly, the current vehicle depth texture is generated through a special modelview and projection transformation, Fig.1(b). Upon completion of the upper two render steps, we can get the current depth offset map through mathematical operation in the fragment shader, Fig.1(c). The depth offset map that represents the elevation offsets for each vertex in the terrain depth texture is due to the compression

978-1-4244-5540-9/10/$26.00 ©2010 IEEE

forces of the vehicle to the soft terrain. Then we can get the deformed terrain depth texture through a simple subtraction operation between the initial terrain depth texture and the depth offset map in the fragment program, Fig.1(d). Upon completion of this pass, all that is left to be done is just to do vertex texture fetch on the deformed terrain depth texture to generate deformed terrain.

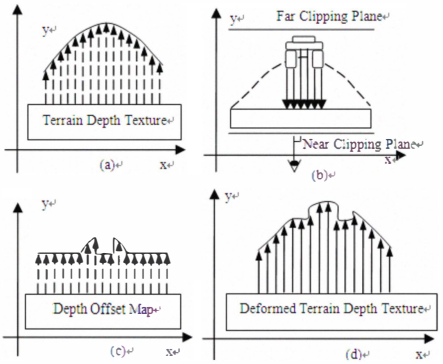

Figure 1. The dynamic terrain deformming algorithm

IV. LARGE SCALE DYNAMIC TERRAIN RENDERING

A. *Algorithm Overview*

We represent the large terrain data as a regular 2D grid of heights with a fixed post distance in X and Z directions. Fixedsize blocks are used as the base units of storage and transfer from the CPU to the GPU.

The potentially visible portion of the terrain is cached in the form of texture array and is rendered from the GPU. The GPU cache is updated continuously as the viewpoint changes.

The deformed terrain heightmap texture in texture array is updated through the FBO render to texture functionality.

The whole GPU Cached terrain is generated in the vertex shader through receiving the layer number of heightmap texture array and tiled block quadtree grid. The multi-thread scheme is used to scheduler terrain blocks between the hard disk and CPU memory and save the modified terrain heightmap texture data in hard disk.

Fig. 2 demonstrates the ruts leaves by a vehicle running in 16×16 terrain blocks.

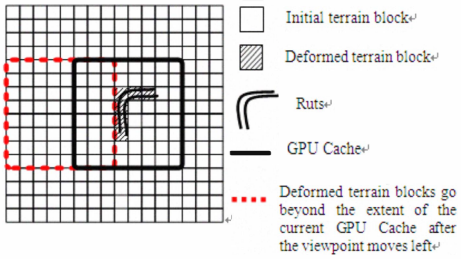

Figure 2. The sketch map of ruts leaved by a vehicle running in 16×16 terrain blocks

B. *Terrain block Representation*

The terrain is stored in main memory and sent to GPU as blocks. We use blocks of size 1025×1025.

The block is divided into a number of tiles, and each tile contains a fixed number of triangles, Fig. 3. The vertices of the tiles are organized in a $(2^k+1)\times(2^k+1)$ regular grid so that each vertex corresponds to one of the $(2^n+1)\times(2^n+1)$ element in the terrain block. All tiles are flat squares as they are passed to the 3D API. The vertices are then displaced in the vertex shader by fetching corresponding elevations from the heightfield vertex texture. Since all tiles consist of an equal number of vertices, a single vertex buffer and index buffer can be reused for all the tiles. Only a couple of uniform shader parameters are altered to render different tiles using the same vertex buffer and index buffer. The uniform parameters that must be set for each tile include a set of scale and bias factors. These are used to specify the area of the tile and its location on the terrain block.

Crack will appear on the borders between tiles of different level of detail. A number of different approaches can be used to prevent the dreaded holes in the terrain. We have chosen to use the simple, yet elegant solution of vertical skirts along the tiles borders in [9].

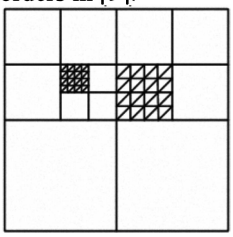

Figure 3. The tiles are mapped to the region of the heightmap using the scale and bias factors.

C. *GPU Cache representation using texture array*

One of the extensions introduced with Shader Model 4.0 is texture array in [10]. This extensions introduces the idea of one and two-dimensional texture array.

A Texture array is a collection of one- and two-dimensional images of identical size and format, organized in layers. Currently, this extension is not supported on the fixed-function fragment processing. Texture array can be accessed only using the programmable shaders.

The GPU Cache is stored as a texture array. The cache can be attached to a single texture unit and a height can be accessed on the fly by the GPU. Heights are accessed using three coordinates (s, t, p): p to select the block (also called a layer) and (s, t) to fetch the tiled height value from that layer. Each layer of the texture array can be updated randomly. We use a separate one dimensional int type array to store the layer IDs, Fig. 4. The LayerID[i]=i, i from 0 to N^2-1, where N means the size of the GPU Cache. The LayerID[i] is updated with new layer numbers when the GPU cache is updated. This process is explained in detail in Section 4.4. The unified architecture of SM4.0 provides fast access to the texture for all shader units.

With texture array any part of the terrain can be accessed by specifying the layer with p texture coordinate in case of a 2D texture array, thus providing the ease of selecting textures

in the programmable shaders rather than an intervention from CPU regarding the binding of correct texture. The same was possible with 3D textures, but the texture array can additionally be rendered to by binding them to a frame buffer object (EXT framebuffer object).

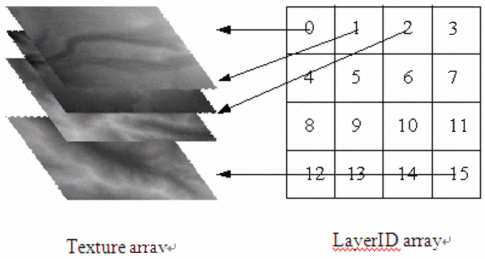

Figure 4. A 16 layer texture array(GPU Cache) with LayerID stored in one dimensional int type array

D. Caching

The GPU Cache contains N×N blocks, and the size N depends on the maximum visibility required at the highest resolution. We use N = 8 for our experiments, needing storage for 64 blocks on the GPU. We try to keep the GPU cache symmetric with respect to a reference point, which is the centre of the orthographic projection of the view frustum onto the ground, Fig. 5.

Lateral motion, pan, and tilt at a constant elevation bring in new data to the GPU cache .We use the position of the reference point in the cached terrain to trigger the data transfer. If the reference point goes outside the central 2×2 block of the GPU cache the cache is re-centered by bringing another row or column of blocks, discarding blocks on the other side of the cache, Fig. 5. We load the new blocks by overwriting the discardable blocks. The data from the CPU is loaded to selected layers of the texture array and the LayerID array is updated to rearrange the layer IDs on the GPU.

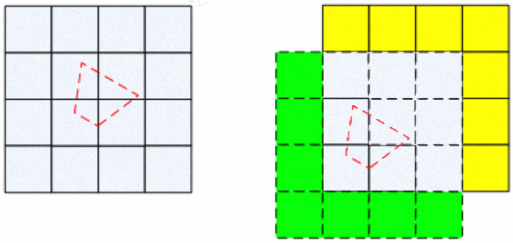

Figure 5. Lateral motion involve discarding an L-shaped region (yellow) and bringing in new blocks (green) from the CPU.

Different movement direction of the viewpoint in horizontal plane will result in different update of the LayerID number in the texture array. Fig. 6 demonstrates the LayerID's update after the viewpoint moves one block extent along the X axis.

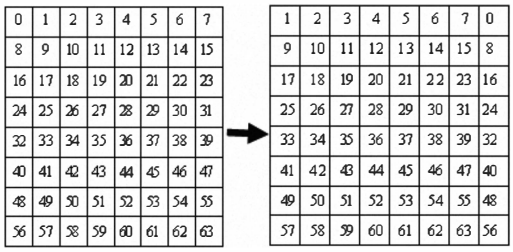

Figure 6. The LayerID array in GPU Cache is updated as the viewpoint moves one block extent along X axis

E. Level of Detail

As noted earlier, by halving the scale parameters, the area covered by a tile is reduced to a quarter. Thus, a tile can be divided into four smaller tiles with half the scale. If we start with one tile covering the entire terrain block and then recursively divided it into four children until the tiles match the heightfield resolution, we have effectively created a quadtree of tiles. We will use this to control the level of detail of the terrain block. We recursively evaluate each tile and decide whether to render it or to divide it into four smaller tiles. Depending on our evaluation criteria and the tile size, we can reduce the total number of triangles rendered and still get a nice terrain.

We take an approach like [11,12], and limit our refinement to be solely based on the viewer' position. This method uses a simple distance-based evaluation criterion so that distant tiles cover more area than tiles closer to the camera. Obviously this approach would look awkward if steep mountains jutted out of the ground with few vertices to represent its peak. However, natural terrains are well suited since a smooth gradient of vertices usually make up the elevation data, i.e. a mountain peak doesn't consist of a single vertex.

The distance-based evaluation criterion is given by the simple test:

$$\frac{l}{d} < C \qquad (1)$$

The value l in (1) is the distance from the center of the tile in a terrain block to the camera, and d is the world-space extent of a single triangle. If the test fails, the tile is rendered. If the test succeeds, the tile is divided into four smaller tiles and the recursion continues.

F. Render to different texture array layer

In order to update the deformed terrain heightmap texture in texture array, we will use FBO render to different texture layers functionality. The current EXT_framebuffer_object specification allows for 16 attachment points (GL_COLOR_ATTACHMENT0_EXT to GL_COLOR_ATTACHMENT15_EXT) each of which can point to a separate texture attached to it. However, the number you can render to depend on whether you are running on hardware and drivers. On our NVIDIA Geforce 9500 hardware the total number of color attachments will be

a max of 8 buffers. Therefore we will set up 8 different framebuffer objects to make the total GPU Cache deformable.

The movement of the viewpoint will lead to the update of the LayerID array and the GPU Cache content. After the update process completed, the LayerID of the block that the vehicle is currently running in is computed through the local block offset in the GPU Cache blocks, VehicleInLayerIDNumber=f_LayerID[iLocal_XStride+iLocal_ZStride*8]. Using the LayerID number, we can identify which framebuffer object to bind and which color attachment to attach.

G. *Large scale terrain data scheduler scheme*

First of all, neither the render to texture operation of the FBO nor the texture array extension support the compressed texture format. If we load in all terrain data of the 16k×16k Puget Sound data in its initial format, the memory will overflow. Secondly, as the movement of the viewpoint will result in the deformed heightmap textures leave the current GPU Cache, we need save the modified terrain data in hard disk at right time.

Aim at upper problems we designed multi-thread scheme. The main thread makes the view frustum culling, computes the level of detail of blocks and renders the scene data. The assistant thread scheduler terrain blocks between the CPU memory and hard disk dynamically and save the modified terrain heightmap texture data in hard disk.

V. THE EXPERIMENT RESULT

We performed our experiments on an Nvidia 9500 GTX using OpenGL and GLSL shaders on Windows with a Pentium Dual-Core CPU running at 2.5 GHz. We use the Puget Sound data, consisting of a 16385×16385 grid of 16-bit heights.

The horizontal grid post is 0.3m, and the deformable terrain area size is 2.5km×2.5km. The largest height value is 65536×0.003=196.608m.

We simulate one scene that a hummer moves in soft soil. We show the colored terrain surface and the wire terrain. Fig. 7, Fig. 8 are some screenshots from our algorithm. All data sets were rendered to a 1024×768 view port. The simulation runs at approximately 150 frames per second.

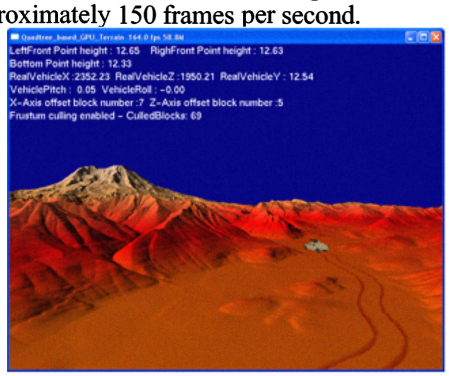

Figure 7. The hummer deforms the terrain dynamically as it moves in large scale terrain and carves tire tracks.

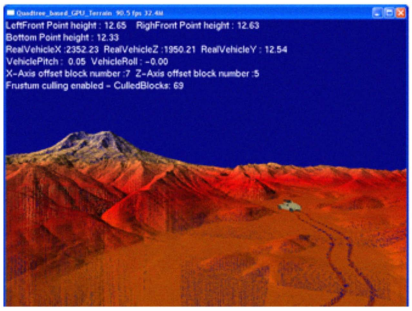

Figure 8. The wireframe mode, using tiled block quadtree to realize LOD rendering, vertical skirt to avoid cracks.

VI. CONCLUSION

In this paper we presented a multiresolution rendering method of tire tracks in large scale dynamic terrain. It is important that terrain deformation and rendering are all manipulated on the GPU. Experiments show that our method is valid and effective.

ACKNOWLEDGMENTS

This work was partially supported by The Armament Kit Advanced Research (No. 623010102).

REFERENCES

[1] Y. He, J. Cremer, and Y. Papelis, "Real-time extendible-resolution display of on-line dynamic terrain," In: Proceedings of the 2002 Conference on Graphics Interface. Calgary, Alberta, Canada, 2002

[2] Li, X. and Moshell, J. M, "Modeling soil: realtime dynamic models for soil slippage and manipulation," in Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH), 1993

[3] Sumner, R. W., O'Brien, J. F., and Hodgins, J. K, "Animating sand, mud, and snow," Computer Graphics Forum, vol. 18, 1999, pp. 17-28.

[4] Wang Linxu, Li Sikun, Pan Xiaohui, "Real time visualization of dynamic terrain," Chinese Journal of Computers, 2003, pp.1524–1531

[5] Guojun Chen, Jing Zhang, Xiaoli Xu, "Real-time visualization of tire tacks in dynamic terrain with LOD," In: Proceedings of International Conference on E-Learning and Games. LNCS, vol. 4469, 2007, pp. 655–666. Springer-Verlag

[6] Xinquan Cai, Fengxia Li, Haiyan Sun, "Research of dynamic terrain in complex battlefield environments," Pan et al. (Eds.). Proceedings of International Conference on E-Learning and Games. LNCS, vol. 3942, 2006, pp. 903–912. Springer-Verlag

[7] Anthony s. Aquilio, Jeremy C. Brooks, Ying Zhu, "Real-time GPU-based simulation of dynamic terrain," ISVC 2006, LNCS, vol. 4291, 2006, pp. 891-900. Springer, Heidelberg

[8] Dong Wang, Yunan Zhang, Peng Tian, and Nanming Yan, "Real-time GPU-based visualization of tile tracks in dynamic terrain," The International Conference on Computational Intelligence and Software Engineering (CiSE), 2009, pp. 1-4, IEEE Press, Wuhan China

[9] Harald Vistnes, "GPU Terrain Rendering." In Game Programming Gems 6. Charles River Media, 2006

[10] Shiben Bhattacharjee, Suryakant Patidar, P. J. Narayanan., "Real-time rendering and manipulation of large terrains," Sixth Indian Conference on Computer Vision, Graphics & Image Processing, 2008, pp. 551-559

[11] Losasso F, Hoppe H. Geometry clipmaps, "Terrain rendering using nested regular grids," In Proceedings of ACM SIGGRAPH '04, 2004

[12] William E. Brandstetter, "Multi-resolution deformation in out-of-core terrain rendering," University of Nevada, 2007