Real-Time GPU-Based Visualization of Tile Tracks in Dynamic Terrain

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

***Abstract*—In conventional vehicle driving simulation systems, the tracks effect does not alter the terrain surface topology that the tires interact with. In this paper, we propose a dynamic terrain visualization method based on modern Graphic Processing Unit features: vertex texture fetch, framebuffer object extension and shader technology. First, the height map to the initial terrain depth texture directly. Next, the vehicle depth texture is generated through a special render pass. Then in the fragment program the depth offset map is computed. We proceed by generating the terrain depth texture that represents the whole deformed terrain surface. At last, we use it as displacement map to create the final image of the tire tracks leaved by a moving vehicle. We demonstrate our algorithm by simulating a ground vehicle traveling on soft terrain. The experiment proves that the method is feasible and efficient.**

Keywords-terrain visualizatio; dynamic terrain; tire simulation; hardware acceleration

I. INTRODUCTION

The real-time visualization of the terrain plays an important role in computer graphics, three-dimensional geographic information systems, virtual reality and 3D games. Many excellent algorithms are proposed to realize the large scale terrain rendering. Along with the terrain visualization technology progressing, high quality and reality ground vehicle driving simulation system is more desired than before. Dynamic terrain has become an increasingly important requirement for realistic ground-based simulation systems. However, nowadays the vehicle-terrain interaction is absent in most driving simulation systems. They only animate the tire tracks by texturing without changing the elevation of the affected terrain vertexes. The vehicle-terrain interaction is not only important to improve the reality of driving simulation system, but also to the training value of the battle simulation system.

In this paper, our research is focused on the visualization of terrain changes due to the terrain-vehicle interaction. By means of the features of modern Graphic Processing Unit-framebuffer object (FBO), vertex texture fetch and shader technology, we provide a GPU-based algorithm for tracks visualization. Our algorithm only needs to perform vertex texture fetch operation one time each frame to reflect topographical changes, and the efficiency of our algorithm is improved. We demonstrate our algorithm by implementing a real-time driving simulation in which a ground vehicle leaves tracks on soft terrain.

The remainder of this paper is organized as follows: In Section 2 we review some related work by previous researchers. Then, in Section 3 and 4, we introduce our rendering technique and give an implementation of it. 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 [3]. Their technique is physically based and they arrive at their simulation formulation after a detailed analysis of soil dynamics. Sumner et al. [4] 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. [5] 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. [6] 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) [7]. Cai et al. [8] 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. [9] presented a new GPU-based 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.

978-1-4244-4507-3/09/$25.00 ©2009 IEEE

III. TERRAIN DEFORMATION ALGORITHM

A. Algorithm Overview

First of all, translate the height map representing the original terrain height to the initial terrain depth texture directly. Secondly, the current vehicle depth texture is generated through a special modelview and projection transformation. Upon completion of the upper two render steps, we can get the current depth offset map through mathematical operation in the fragment shader. The depth offset map that represents the elevation offsets for each vertex in the terrain depth texture is due to the compression 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. In order to produce continuous tracks, the deformed terrain depth texture in the current frame is also used as the initial terrain depth texture of the next frame. 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.

The pseudo-code of our GPU-based dynamic terrain visualization algorithm is as follows:

```
begin
initialize terrain depth texture
while no exit signal do
begin
generate vehicle depth texture
generate terrain depth offset map
update terrain depth texture
generate deformed terrain
end
end
```

The following sections cover each step in greater detail.

B. Initial terrain depth texture

Terrain height data is generally stored as a grayscale image in which each pixel represents a height value. Dark colors represent a lower elevation, and lighter colors represent a higher elevation. In our method, the 2D height map is bound to the color attachment of the FBO; as a result the range of the generated texture element is from 0 to 1. Through the GL_ARB_texture_float extension we can make the color component of the terrain depth texture 32-bit floating format, which would be used for fetching. With the floating-point internal formats, no normalization or clamping takes place.

Let Us represent the height map's elevation data and let Uv represent the corresponding data in the terrain depth texture. The above rendering pass can be treated as a transformation function, Us→Uv. Through the offscreen rendering of the FBO, the size of our terrain depth texture is not limited to our window.

Since the element of the terrain depth texture is from zero to one, in the process of inverse transformation, from Uv→Us, we need a scaling factor to obtain the initial terrain elevation data.

C. Vehicle Depth Texture

After finishing the initial terrain depth texture, we can get the vehicle depth texture through following render state.

1. Create FBO and bind the texture to the depth attachment.

2. Set the orthographic projection mode and configure the viewport to an m× n area of the render target.

3. Translate the viewing position to the center of the m× n region, and viewpoint position is under the lowest elevation point of the terrain. Rotate the viewing direction to be perpendicular to the ground plane, looking upward.

First Execute the application-specific simulation model and collision detection method to sink the vehicle, then render the vehicle depth texture to the depth attachment of the FBO. The configuration of the rendering state is shown in Fig. 1.

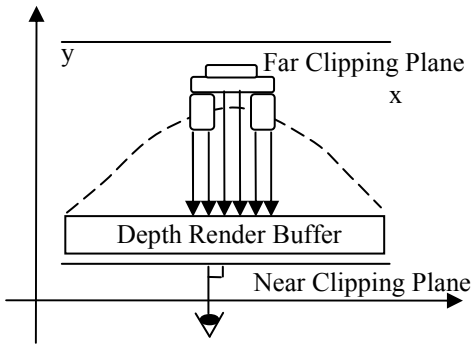

Figure 1. The camera configuration for generating vehicle depth texture

D. Depth Offset Map

After finishing the vehicle depth texture computation, we can calculate the elevation offset for each height in the original terrain depth texture through comparison. The elevation offsets result from the compression forces of the vehicle to the soft terrain. With the camera below the lowest elevation point of the terrain, only the vertices correlated to object geometry that penetrate the terrain surface are processed. We can store the result into another m× n render target called terrain depth offset map.

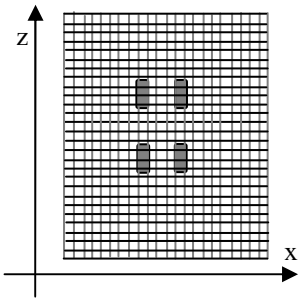

Figure 2. The pixel value in the four rectangles represents tires depth under the terrain surface, and others are zero.

In the fragment program, subtracting the vehicle depth texture in section 3.3 from the initial terrain depth texture in the section 3.2, we can get the depth offset map. Each pixel in the offset map has a one-to-one mapping to the initial terrain depth texture (Fig. 2).

E. Update Terrain Depth Texture and Render deformed terrain

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. 3, Fig. 4). For producing continuous tracks, the deformed terrain depth texture in current frame is also regarded as the initial terrain depth texture of the next frame.

During this step the deformed terrain depth texture is available in the video memory and accessible as textures from the preceding passes. Upon completion of this pass, all that remains to be done is to do vertex fetching on the deformed terrain depth texture to generate deformed terrain.

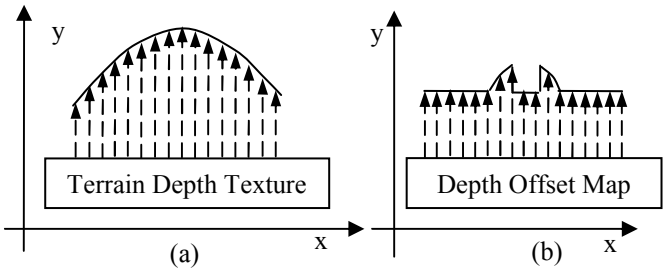

Figure 3. Image of generated terrain with initial terrain depth texture (a) as displacement map and with depth offset map (b) as displacement map.

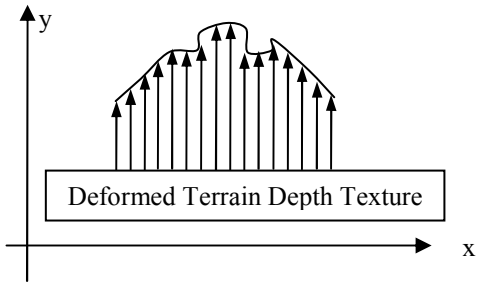

Figure 4. Deformed terrain with vertex texture displacement

The vertex shader samples the vertex texture using the texture coordinates defined in an m× n rectangular plane grid, which is stored in an indexed triangle strip format and cached through Vertex Buffer Object (VBO). The data from the vertex texture then displace the height of the vertexes so as to generate the deformed terrain.

The arithmetic computation in the vertex shader to generate the dynamic terrain is as follows:

$$Y_{deformed\_terrain} = Y_{nerar\_plane} + V_{deformed\_terrain} \times \left(Y_{far\_plane} - Y_{nerar\_plane}\right) \quad (1)$$

In Equation (1), $Y_{deformed\_terrain}$ is the deformed terrain height, Y_{nerar_plane} is the near clipping plane, Y_{far_plane} is the far clipping plane, and $V_{deformed\_terrain}$ is the deformed terrain depth texture.

Finally the vertexes are transformed to screen space by multiplication by the modelview projection matrix. In this manner, the terrain is deformed to realistically reflect the terrain topographic changes.

IV. IMPLEMENTATION RESULT

We have implemented the algorithm on a Pentium(R) Dual-Core 2.5GHz Processor with NVIDIA GeForce9 graphics that supports Shader Model 3.0 under Windows XP. We use VC++.Net 2003 and OpenGL environment. Our implementation uses GLSL for shader programming. All data sets are rendered to a 1024×768 viewport. The initial terrain height map and the render target are 1024×1024.

Figure 5. The terrain geometry is changed due to the compression forces of the vehicle to the soft terrain.

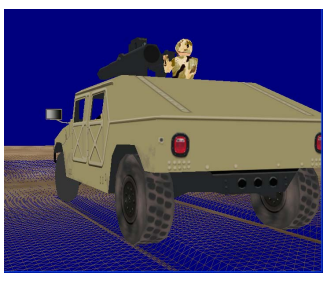

Figure 6. The wire mode of the deformed terrain

Figure 7. The vehicle deforms the terrain dynamically as it moves and carves tire tracks.

Fig. 5, Fig. 6 and Fig. 7 are some screenshots from our algorithm. We use a military vehicle model created by Full Sail student Stephen Carter [10]. The algorithm is implemented as a research prototype with no code tuning or low-level code optimization, and view-frustum culling is also absent. The simulation runs at approximately 105 frames per second.

V. Conclusion and Continuing Work

In this paper we have presented a simple and effective algorithm for interactive tire tracks rendering in dynamic terrain. Our approach makes full use of the modern features of GPU and dramatically reduces the communication traffic between CPU and GPU. The dynamic updated terrain depth texture is generated through FBO, and the whole deformed terrain is generated through vertex texture fetch operation. As a result, our method achieves a high frame rate and is suitable for interactive 3D applications.

Our method has some limitations. Our approach does not simulate soil erosion. This means that our method is quite effective for clay-like ground materials. Future research includes (1) extend our method to include granular materials by integrating erosion mode and (2) integrate our method with large-scale visualization algorithm.

Acknowledgments

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

References

[1] A. Shamir, V. Pascucci, C. Bajaj: Multiresolution dynamic meshes with arbitrary deformations. In: Proceeding IEEE Visualization'00, pp. 423—430. (2000)

[2] Losasso F, Hoppe H. Geometry clipmaps: terrain rendering using nested regular grids. In Proceedings of ACM SIGGRAPH '04, (2004)

[3] 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)

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

[5] 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)

[6] Wang Linxu, Li Sikun, Pan Xiaohui: Real Time Visualization of Dynamic Terrain. Chinese Journal of Computers (2003)1524–1531

[7] Guojun Chen, Jing Zhang, Xiaoli Xu e: Real-Time Visualization of Tire Tracks in Dynamic Terrain with LOD. In: Proceedings of International Conference on E-Learning and Games. LNCS, vol. 4469, pp. 655–666. Springer-Verlag (2007)

[8] 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, pp. 903–912. Springer-Verlag (2006)

[9] Anthony s. Aquilio, Jeremy C. Brooks, Ying Zhu: Real-Time GPU-Based Simulation of Dynamic Terrain. ISVC 2006, LNCS, vol. 4291, pp. 891-900. Springer, Heidelberg (2006)

[10] Richard S, Wright J r: OpenGL SuperBible (Third Edition). Addison-Wesley Publishing (2004)