Pixel-Based Level of Detail on Hardware Tessellated Terrain Rendering

Mehmet Ömer Özek
HAVELSAN Inc.
Ankara, Turkey
mozek@havelsan.com.tr

Engin Demir
Computer Engineering Department
Cankaya University
Ankara, Turkey
engindemir@cankaya.edu.tr

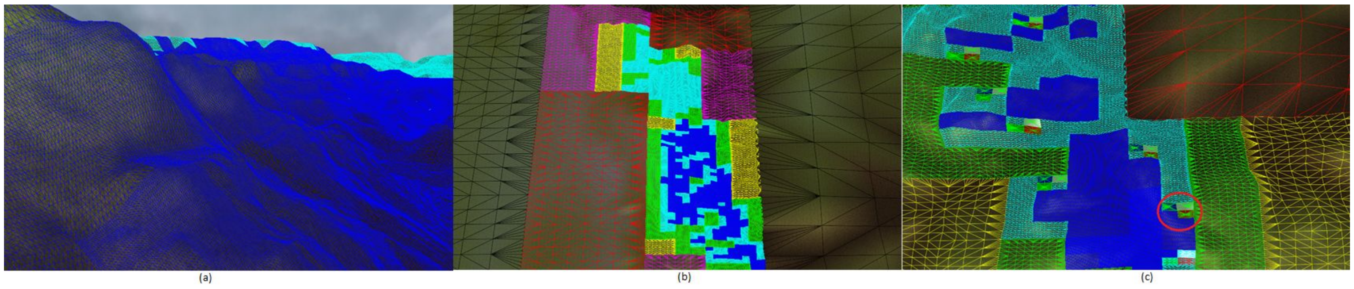

Figure 1: (a) Mesh terrain rendering using occlusion query (b) bird view of the scene in part a (c) quad-tree based visibility query of the mesh terrain.

ABSTRACT

As the GPU's processing power has been improving much faster than the CPU's. the terrain rendering algorithms have evolved to use the graphics hardware as much as possible. One of the recently developed GPU-based Level of Detail (LOD) algorithm is Continuous View Dependent Adaptive LOD using hardware tessellation. In this study, Continuous View Dependent Adaptive LOD using hardware tessellation is enhanced using three additional methods. First method is determining pixel-based LOD using occlusion query. Because of hidden surface culling, render time is reduced. Second method is determining pixel-based LOD using occlusion query that newly developed by using OpenGL and CUDA interoperability. Third method is extension of second method that includes quad-tree based query for each terrain node. Third method aims to increase rendering quality of partial rendered terrain node.

CCS CONCEPTS

•**Computing methodologies →Visibility; Mesh models;** *Graphics processors;*

KEYWORDS

Terrain Rendering, Hardware Tessellation, Determining LOD, Occlusion Query, Continuous View Dependent Adaptive LOD

ACM Reference format:
Mehmet Ömer Özek and Engin Demir. 2017. Pixel-Based Level of Detail on Hardware Tessellated Terrain Rendering. In *Proceedings of SIGGRAPH '17 Posters, Los Angeles, CA, USA, July 30 - August 03, 2017,* 2 pages.
DOI: 10.1145/3102163.3102218

1 INTRODUCTION

The simplest way to render a terrain is the brute force approach that every height in the heightmap data is represented with one vertex in the surface. However this is not feasible in real time terrain rendering for larger heightmap data in walktrough applicaitons [Cohen-Or et al. 2003]. Thus level of detail (LOD) algorithms produce a mesh of different complexity so that the on-screen triangle distribution is relatively equal while using only a subset of heightmap data. The LOD algorithm directly affects both the quality and the performance of the rendering. The aim of this work is to improve the performance of the visualization of the terrain and the LOD algorithm that directly affects the perception of reality. In this context, it is aimed to improve the performance of the continuous view dependent adaptive LOD algorithm using GPU tessellation [Wang et al. 2015].

2 OUR APPROACH

The baseline approach is selected as the adaptive LOD algorithm using GPU tessellation [Wang et al. 2015] which uses two phase LOD determination. The CPU based LOD is performed by meta quad tree traversal and selection of appropriate LOD for different areas of the terrain based on patch geometric world space error and distance to camera. The GPU based LOD is performed by hardware tessellation and selection of appropriate LOD for different areas of the terrain based on distance to camera [Wolff 2011].

Method 1 extends the baseline and additionally computes visibility using occlusion query. By this way, CPU-based LOD creates less number of patches by eliminating the ones that are not visible. Invisible patches are rendered with low LOD and hence the performance has improved. As seen in Figure 1(a), with the occlusion query, almost all patches of the wireframe has rendered in full detail but in Figure 1(b) the invisible patches are rendered with low detail due to the quad tree partitioning. If the camera position has changed, in order to display all patches in the scene, the previous visibility testing for a patch is used. In GPU-based LOD, inner tessellation levels are determined using the previous scene visibility results. OpenGL occlusion query is utilized to determine patches in the scene. The occlusion query is started while the corresponding patch is rendered and finally the query is finalized. As these operations are performed in parallel, the occlusion query can be performed efficiently. After all patches are computed, they are stored in the visibility table and for each patch, screen pixel density is computed according to the ratio of total number of visible pixels to surface area. While inner tessellation levels are determined using the ratio, pixel density may have small changes continuously and it may show a hysteresis behaviour as shown in Figure 2 to soften the changes in the walktrough applications.

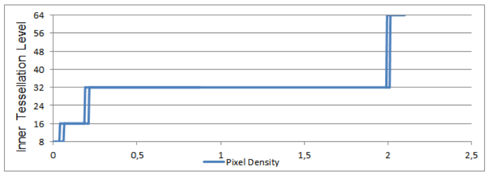

Figure 2: Hysteresis change of pixed density and inner tessellation level

In Method 2, a novel occlusion query processing approach is introduced and implemented on the GPU using OpenGL and CUDA. Method 2 extends Method 1 with the proposed occlusion query and uses a SSBO [1] (Shader Storage Buffer Object) to store counts of pixels in the fragment shader. The reason of storing a SSBO is because (i) write operations are atomic (ii) data size is adaptively changing (iii) data size can be larger than maximum size of the standard buffer object. Because of the performance bottleneck of the atomic add operations in fragment shader, the SSBO are mapped to CUDA memory using OpenGL and CUDA interoperability. Then during the summation of counts in the SSBO, the parallel summation algorithm in CUDA is used as proposed by Harris [Harris 2008]. Then counts are mapped from CUDA memory in order to determine inner tessellation levels.

Method 3 improves the rendering quality of Method 2 by performing visibility tests by diving each visible patch into quad cells. In Figure 1 (c), the patch under the red circle has only one visible cell which is rendered in higher LOD but the remaining three cells are rendered in lower LOD. By this way, even the number of triangles increases in the visible scene, the number of triangles decreases in the overall scene. This method is expected to improve the quality of rendering but may slow down the computation compared to Method 2 as the computation has more cells to be rendered by the GPU.

[1] https://www.opengl.org/wiki/Shader_Storage_Buffer_Object

Table 1: Performance results for the first scenario

Method	Rendering Time (ms)	Rendered patches	Generated triangles
Baseline	18.97	2505	1282648
1	4.70	478	126032
2	3.73	478	126032
3	4.06	618	125221

Table 2: Performance results for the second scenario

Method	Rendering Time (ms)	Rendered patches	Generated triangles
Baseline	18.67	2399	1228398
1	1.94	196	38941
2	1.81	196	38941
3	1.91	231	38618

3 EXPERIMENTAL RESULTS

Experimental results are performed on a PC with Intel Core i5-4200H CPU @ 2.80Ghz, 12 GB RAM, and NVIDIA GeForce GTX 950M using Windows 8.1 Pro 64Bit, Visual Studi 2013 C++, and CUDA 7.5. Hardware tessellation requires OpenGL 4.0 or a newer version. and the SSBO in the fragment shader, used in Method 2 and 3, requires OpenGL 4.3 or a newer version. The dataset used in the experiments is the grand canyon (80km x 40km). Two basic scenarios are tested such that in the first scenario fewer occluded regions appears whereas in the second one in the middle of the visualization most of the terrain is occluded and a small part of the terrain is rendered during walktrough the scene.

All methods are compared using render time (in milliseconds), number of rendered patches, and generated triangles. Table 1 and 2 show the performance comparisons of methods under distinct two scenarios where quad tree based partitioning ratio is set to 10 and SSBO buffer size for each patches is 32. The baseline approach is shown to perform poorly due to the rendering of patches that are invisible. The proposed methods for pixel based LOD solutions based on hardware tessellation and occlusion query shown to outperform the baseline. In the experimental results with the increasing level of inner tessellation, it is observed that the performance difference between the proposed methods is getting smaller as the rendering time is increasing, the performance improvement due to the occlusion query turns out to be dominated by the other costs.

REFERENCES

D. Cohen-Or, Y.L. Chrysanthou, C.T. Silva, and F. Durand. 2003. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics* 9, 3 (2003), 412–431. DOI: http://dx.doi.org/10.1109/TVCG.2003.1207447

Mark Harris. 2008. *Optimizing Parallel Reduction in CUDA.* Technical Report. nVidia. http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

D. Wang, Q. Zhu, Y. Xia, D. Liu, W. Li, and L. Zhang. 2015. Adaptive LOD terrain rendering with GPU tessellation. *Journal of Computational Information Systems* 11, 20 (2015), 7489–7496. DOI: http://dx.doi.org/10.12733/jcis15787

David Wolff. 2011. *OpenGL 4.0 Shading Language Cookbook.* Packt Publishing.