Efficient Terrain Map generation

William Giffin and Matt Frey

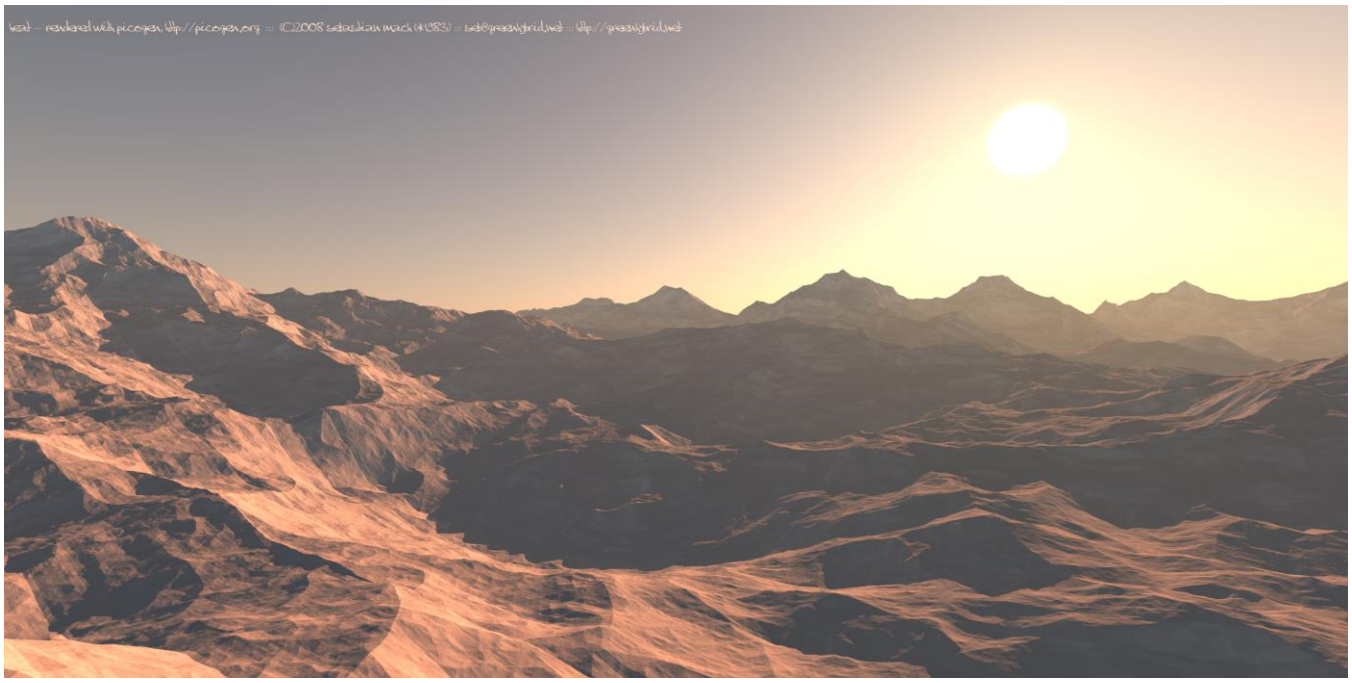

Fig. 1. Terrain map generation using a basic 2D quadtree to represent a 3D heightmap

Abstract— Develop an implementation of Thatcher Ulrich's "Rendering Massive Terrains using Chunked Level of Detail Control" [1], based off of a chunked Levels of Detail (LOD) quadtree algorithm. Implementation would be barebones, only a single 512 x 512 surface which would render with increasing detail as the camera moved closer to the terrain.

Index Terms—Level of Detail (LOD), QuadTree, Random Access Memory (RAM), Ideal Terrain Renderer (ITR), Heightmap

1 Introduction and Motivation

The purpose of this project is to research, implement, and demonstrate a continuous LOD mechanism for terrain/height maps. This will be done using a Quadtree system, C++, and OpenGL. The purpose (or problem being solved) is to enable a computer to handle higher resolution terrain/height map data without suffering a huge performance/processing hit. Throughout the next section of this paper, the specifics of a Quadtree will be explained: what it is, how it works, how it is represented in the data, and how it will ultimately be used to create a terrain scene.

- *William Giffin is with the Air Force Institute of Technology (RA). E-mail: giffin.208@osu.edu*
- *Matt Frey is with Fast Enterprises. E-mail: mefr3y@gmail.com*

The motivation behind this project was to explore the research and methods which sit as the foundation for all current game engine terrain systems (Unity, Havok, Unreal, etc). Ultimately this project's aim was to implement a portion of the framework (the quadtree and rudimentary visualization system) which could then be expanded at a later time (permitting) into a fully realized game engine. The second portion (creation of the game engine) quickly fell outside the scope of the projects timeline however the initial phase proved to be more than substantial for the amount of experience gained from the course.

2 The Problem

The problem being addressed by this project is the high cost of rendering large scale terrain fields. Applications for this problem include real world simulations, games, astronomical modelling for extra-terra planets and moons etc. Additional more advanced applications of this basic principle include procedurally generating terrain for 3D/VR/Augmented Reality systems or for advanced simulation and modelling for testing purposes of vehicles.

This particular project aims at the smallest portion of that problem by addressing an approach to handling the large amount data required by an ITR in hopes to setup an environment which mimics an infinite system within a finite space.

3 QuadTrees – An Introduction

"A QuadTree is a tree data structure in which each internal node has exactly four children. Quadtrees are the two-dimensional analog of octrees and are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. The data associated with a leaf cell varies by application, but the leaf cell represents a 'unit of interesting spatial information'." [4]. For the purposes of this project, the quadtree will be represented by a set of 4 triangles forming a triangle fan (Fig. 2.)

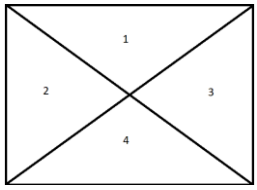

Fig. 2 the basic outline of a QuadTree fan

3.1 Quadtree setup

Each QuadTree fan will represent a certain LOD and if available, it will split into smaller subdivisions containing higher LOD's. When the highest LOD has been reached, the QuadTree will stop subdividing and remain static. This operation also works in reverse as lower LOD's are made through combining QuadTree children into a single QuadTree.

For the implementation portion of this project, the quadtree fan was replaced with a fan-square. (Fig. 2a), a version of the fan but instead of the four nodes being the corners, each node represents a smaller quadtree. This was done to enable a higher LOD detection resolution as compared to the single fan system and to save needless checks on sub tress. Finally the quadtree fan allowed for easier debugging, visualization, and data generation.

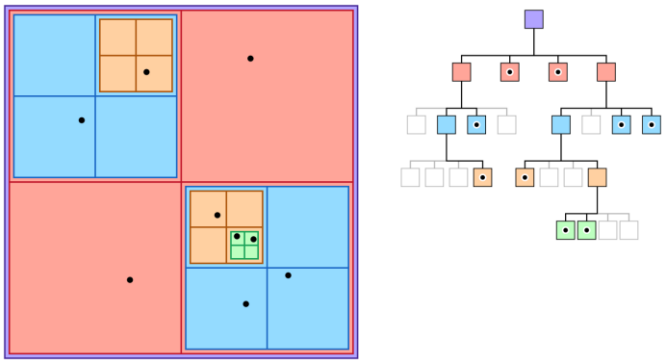

Fig. 2a The current Quadtree system. Each squad has a center with each point-end of the fan being another child quadtree

3.2 Q

Due to the square construction that has been chosen for this QuadTree, only perfect square heightmaps can be used for this LOD system. As a result, only heightmaps with sides which are powers of two will be used for this project, the biggest size being 512 x 512.

After the implementation phase was complete, the ideal size of the quadtree was reduced to 128x128. This was done purely for the demonstration purposes as it best shows the quadtree algorithm.

3.3 Detection of Subdivisions and Quadtree construstion

The final implementation of the quadtree subdivision is completely different than the proposed implementation from the proposal. This comes in part to a misunderstanding of the quadtree setup and in the previously mentioned change from the fan to quadtree fan setup. Overall the first portion of the subdivision system is in the quadtree point mesh generation (assuming that the point mesh is generated using a 2D array which generations a 2D matrix). After being given the 2D array of points, the Quadtree creates a "center" point (the position of the root) and proceeds to recursively add points into its children. This operation recursively iterates until there are no more points left to add to the quadtree. From there the levels of detail are assigned (the number of LOD's is given at run time by a pre-set variable) to the current level of children starting from 0 (root) all the way to n (the outermost child). During this time, a texture is assigned to a corresponding LOD which is later used during the visualization process.

3.4 QuadTree Components and Construction

As mentioned in the proposal document, the Quadtree as a data structure is built around:

1 Parent Node
4 Child Nodes
4 Neighbor Nodes
1 Center Point
4 Outer Points

For the implementation, certain components were found to be redundant or unnecessary for our particular application namely the neighbour nodes and outer points. These components were found to be redundant due to the change in visualization structure (fan to quadtree fan) and the camera – center point distance calculation which calculates the distance from a point on the mesh (directly below the camera) to the center points of the currently active quadtree nodes. This allowed for a general decrease in memory footprint.

3.5 Design Considerations

For the implementation, certain components were found to be redundant or unnecessary for our particular application namely the neighbour nodes and outer points. These components were found to be redundant due to the change in visualization structure (fan to quadtree fan) and the camera – center point distance calculation which calculates the distance from a point on the mesh (directly below the camera) to the center points of the currently active quadtree nodes. This allowed for a general decrease in memory footprint.

3.6 Design Considerations

As mentioned above the design and implementation of the project's systems is built around the Quadtree data structure. Thus the project was naturally limited to two dimensions as the adding in a third would require the use of an Octree (3 dimensional Quadtree) and was too complicated to implement given the time constraints of the semester. As a result, the project remained in 2D and all implementation specifics are built around this principle. This is being especially noted as the proposal indicated that the addition of a third dimension would be simple to add while in fact the contrary was found during the research phase of the project.

Further design considerations included how much memory would be used and the shape of the terrain mesh. The first is consideration is critical as the Quadtree system is based on recursion and thus there is a natural limit to the amount of LOD's which can be used in the program. The second consideration is techinically optional (based off of the research done by Thatcher Ulrich) but considered off limits again due to the time constraints of the semester.

3.7 Implementation methods

The methodological scope of this project is strictly software based using a windows computer and the Microsoft Visual Studios IDE to develop, test, and demo the project. It is important to note that Visual Studio's compiler is non GNU compliant and as such the code written for this project will only work on a Visual Studios compiler.

4 Results and Analysis of Results

The results of the project vs. the intended outcome exceeded expectations. The implemented Quadtree fan system correctly takes a runtime generated 2D matrix of floating point values (xy coordinates), correctly stores subdivides and inserts the points into the quadtree and

visualizes them according to their layer's corresponding LOD (Fig. 3)

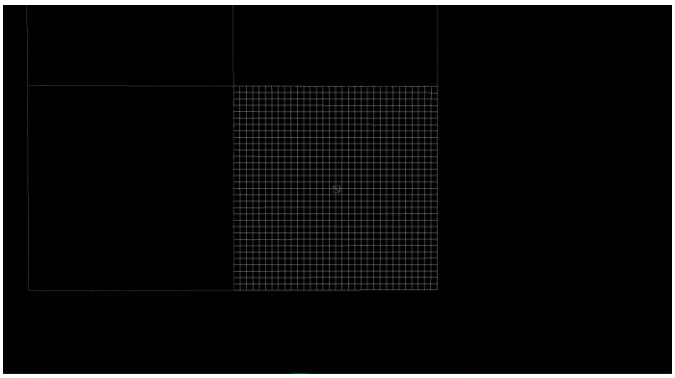

Fig. 3. An entire single Quadtree child being drawn at max LOD while the other children are being drawn at lowest LOD.

Following the initial visualization and a small movement of the camera, the system correctly changes the visualization based off distance from the camera reference position (the small cube in the center of Fig. 3) and redraws the correctly visualization with dynamically recalculated LOD's based off of the reference position (Fig. 4)

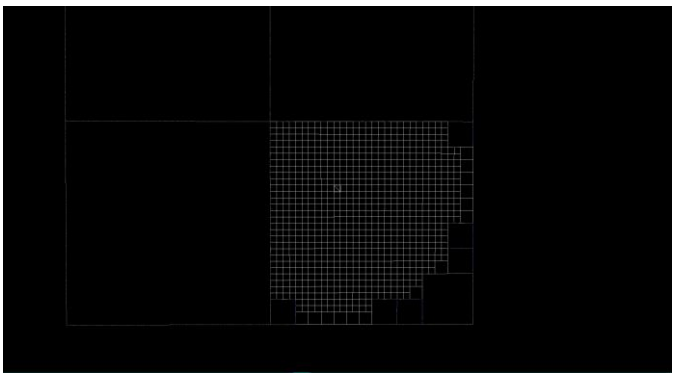

Fig. 4. A redraw of the LODs based off of a small movement of the camera. XY position of camera over terrain indicated by small white square

Finally the system correctly toggles the textures and displays them according to the position based LODs (Fig. 5)

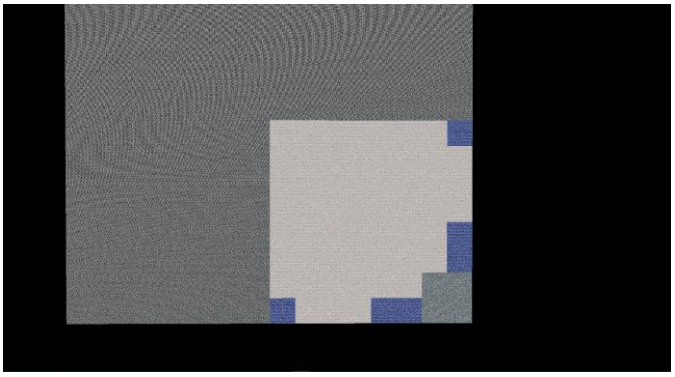

Fig. 5: Fig. 4 but with the textures toggled on

Moving the camera reference position further shows that this system works all positions of the map, further controls manipulate the distance at which the highest LOD is drawn (Fig. 6)

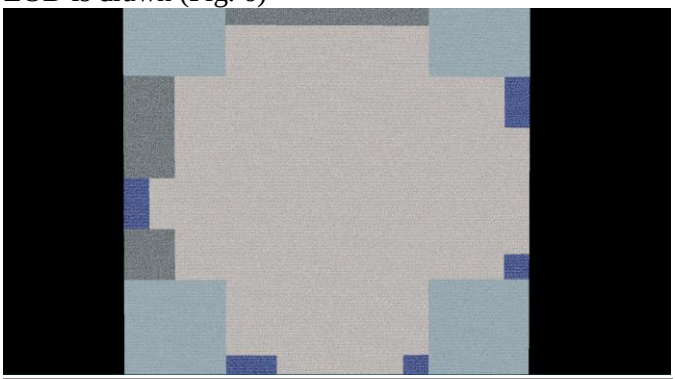

Fig. 6. The reference position moved toward the center of the terrain

5 Conclusion

The aim of this project was to implement a methodology specified by Thatcher Ulrich in which a terrain mesh's level of detail can dynamically change based off of user position. The approach was to utilize the Quadtree data structure to represent the terrain mesh in the attempt to change the LOD based off of user position. Finally, all objectives of the project were accomplished as seen above.

Acknowledgments

The authors wish to thank Chris Brough, Thatcher Ulrich, Dr. Han Wei Shen, and Dr. Jian Chen,

References

[1] T. Ulrich, "Rendering terrains using chunked detail control", http://tulrich.com/geekstuff/sig-notes.pdf, 2002.

[2] C. Brough, "Quadtree Level of Detail for Heightmaps", http://chrisbrough.com/project/2012/04/Quadtree-Level-of-Detail-for-Heightmaps, 2012.

[3] Gamastura, “Continous LOD Terrain Meshing with using adaptive Quadtrees”,http://www.gamasutra.com/view/feature/131841/continuous_lod_terrain_meshing_.php?page=1, 2000.
[4] Wikipedia,“Quadtree”, *https://en.wikipedia.org/wiki/Quadtree*