SIGRAD 2016
M. Hayashi (Editors)

Dynamic Creation of Multi-resolution Triangulated Irregular Network

Emil Bertilsson[†1] and Prashant Goswami[‡1]

[1]Blekinge Institute of Technology, Sweden

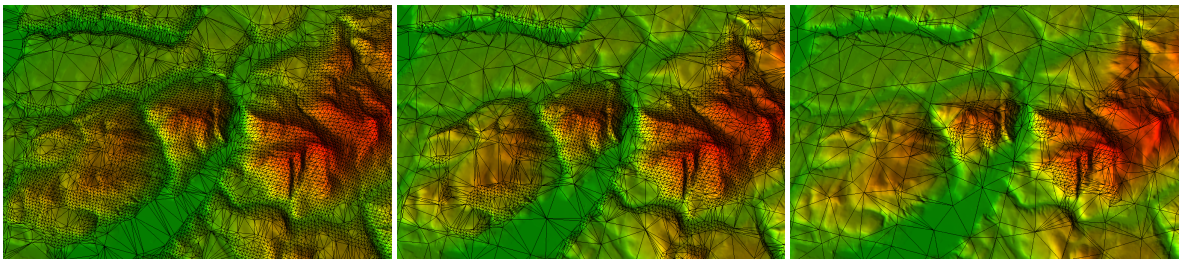

Figure 1: *Terrain mesh simplification achieved using the proposed algorithm in real time with pixel errors (from left to right) 0.5, 1.0 and 2.0.*

Abstract

Triangulated irregular network (TIN) can produce terrain meshes with a reduced triangle count compared to regular grid. At the same time, TIN meshes are more challenging to optimize in real-time in comparison to other approaches. This paper explores efficient generation of view-dependent, adaptive TIN meshes for terrain during runtime with no or minimal preprocessing. This is achieved by reducing the problem of mesh simplification to that of inexpensive 2D Delaunay triangulation and lifting it back to 3D. The approach and its efficiency is validated with suitable datasets.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics an Realism—Visible line/surface algorithms

1. Introduction

Terrain rendering can be found in a variety of applications and the detail of these terrain meshes has vastly increased over the years. Computer games and simulators often have heavy requirements on the visualization of the terrain, as well as to obtain a good resolution that can be presented without overwhelming the other parts of the application. Whether the focus lies on preprocessing level-of-detail (LOD) data or generating it during start up or in real-time, many different techniques have been developed to address the problem of representing and visualizing large quantities of terrain data.

These techniques can be divided into three major groups: regular meshes, semi-regular meshes and fully irregular meshes [PG07]. A regular mesh or digital elevation model (DEM) contains points sampled at equal distance and therefore same sized triangles. Though simple and easy to implement, the main disadvantage of purely regular triangulated meshes is the limited differentiation of the terrain features in triangulation. While there is data redundancy in areas of uni-

† emil.bertilsson91@gmail.com
‡ prashant.goswami@bth.se

form terrain, the mesh could be under represented in regions carrying high complexity.

In theory, fully irregular and semi-regular approaches differ in their capability to produce a minimal complexity mesh representation for any given error measurement. They utilize splitting, merging and subdivision to produce a triangle count closer to what is required to present the terrain features. While TIN methods can provide highly optimized meshes, most of them rely on offline preprocessing due to the heavy nature of computation involved.

When computing multi-resolution terrain models, visual artifacts can arise along the edges where areas of different resolutions meet. These inconsistencies in the mesh are handled by different forms of stitching the areas together [LKES09] or a structure that handles it automatically, providing an equal base for triangulation. Adjoining LODs are often constrained to differ in the tree by at most one level in order to avoid cracks and T-junctions [BGP09].

The novel approach presented in this paper creates a view-dependent multi-resolution TIN mesh on large models based on the user-specified tolerance and is updated throughout the execution of the application. To this end, we adopt the concept of dividing the terrain into larger regions called *patches* which are individually and independently triangulated. A vertex selection algorithm, oriented towards fast processing while approximating the mesh at a reasonable quality during runtime, combined with a Delaunay triangulation uses these patches to produce meshes during runtime. Other more complex and carefully created ways of approximating terrain can be found in [PAL02], [YLS05], [BG04], [EKT01], [TGM*06], [AW03], [HT97]. The main contributions and advantages of this method are:

1. Run-time generation of fully irregular multi-resolution meshes with no or minimal preprocessing.
2. Fast triangulation of a 3D view-dependent point selection by reducing it to a 2D Delaunay triangulation problem.
3. Independent patch processing without any mutual dependencies on the neighboring patches.

The system performs asynchronous LOD updating and supports GPU-based rendering. Delaunay triangulation was chosen as the basis due to its properties [Mus97] and simplicity, that allows for independent triangulation of points. The properties of Delaunay triangulation makes for a good basis for further parallelism in general, or delegation of the triangulation to the GPU.

2. Related Work

Terrain rendering with adaptive resolutions has been studied extensively, with different aims and angles. The earlier approaches [GGS95], [LKR*96], [Hop98], [CPS97], [XV96], [DFP95] have since been followed by more complex algorithms and more recently algorithms that can utilize the GPU to different extents. LOD techniques and multi-resolution meshes have received plentiful research and as such, many relevant papers have been published. We keep the discussion here more focused on techniques that are more related to ours.

The amount of redundant triangles created by these techniques vary on different circumstances and settings. A semi-regular mesh performs well but struggles where different terrain features meet. The transition from a low elevation area to a higher elevation area causes extra complexity due to edge following, which can give suboptimal subdivisions [CGG*03]. Fully irregular meshes struggle with the same kind of optimizations as semi-regular approaches mainly due to the arbitrary neighbours [PG07] inherent to the connection of nodes in a fully irregular mesh.

Many high performance techniques [CGG*03], [LP02], [SS09], [BGP09], [GMBP10] utilize offline precomputed LODs which are then combined at runtime to render the scene. This supplies a frame-to-frame good triangulation, achieved by performing splitting, merging and subdivision on the original mesh. The focus of this paper lies on triangulated irregular network (TIN) and generating view dependent meshes valid for a few frames up to potentially thousands of frames. The reader is referred to [DWS*97], [PG07], [SS09], [GH99] for a more in-depth analysis into other ways of achieving a multi-resolution terrain mesh.

In [LH04] geometric clipmapping for view dependent LOD is discussed and presented as a means to create a pyramidal structure to hold the cached nested regions. It operates on regular triangle grids on-the-fly without any preprocessing, using a refinement scheme of power-of-two to tessellate the terrain. Unlike the proposed algorithm in this paper, their algorithm generates intermediate areas to stitch together regions of different resolution, resembling [COL96] to ensure continuity. The technique has a number of advantages over similar algorithms, such that it can more efficiently compress and store the triangle pyramid during runtime.

Several papers discuss optimizing TIN meshes and different techniques to reduce the triangle count. One of the more interesting ones is BDAM [CGG*03], which deals with small triangle patches consisting of a few hundreds of triangles as primitives rather than single triangles. These patches, which are optimized TINs, are precomputed offline and are stored in a bintree, similar to [DWS*97]. To render the scene, a hierarchical view frustum culling is performed and the patches that are visible are then assembled. This is highly advantageous compared to other techniques that operate on a finer granularity level. However, this approach does require a lot more memory storage, and can only operate on static terrain meshes that have been preprocessed into small patches.

Another technique that utilizes the tree structure is QuadTIN [PAL02], where a quadtree hierarchy is generated over any TIN surface offline and an adaptive LOD is

stored along with the quadtree. Even though the algorithm can operate on any TIN surface, the mesh must be predefined so that the preprocessing pass can generate and store the quadtree. In [BG04], indexing the vertices in the TIN mesh is achieved in such a way that multi-resolution is achieved. By searching the mesh and defining an ordered index list in which consecutive triangles share an edge or a vertex, the mesh can be coarsened using a space-filling curve. The usage of space-filling curves is continuous and is oriented so that the highest index and the lowest index in adjacent triangles coincide.

[Lo12], [KK02] aimed at using point insertions to triangulate randomly generated spatial points using Delaunay triangulation, with slightly different approaches. The algorithm presented in [Lo12] groups together points into cells of roughly equal sizes and triangulates the cells in parallel, without any data dependency between them. On the other hand, [KK02] does not utilize cells or tetrahedrons (and operates on two dimensions rather than three), but instead performs recursive checks after each insertion. But since the points used in the algorithm presented in this paper constitute a terrain, the points have properties that random points do not. Subsequently, simplifications can be made that render the more complex algorithms in [KK02] and especially [Lo12] less fit.

While the modern techniques operate at the level of meta units for more GPU-oriented rendering, simplification can still benefit from improved methods that are capable of reducing triangle counts for TINs at runtime without incurring significant overhead. Our method works in this direction and aims at bridging the aforementioned gap. By defining a distance metric for when the mesh is updated, we can cache our triangulation which in turn provides sufficient time to triangulate the irregular mesh during runtime.

3. Method

The presented approach relies on the standard rendering feature of view-dependent pixel error to define terrain resolution. However, one big difference from earlier works is the absence of the quadtree like hierarchical data structure. The terrain is divided into square regions called *patches* and each patch is triangulated individually, allowing patches to be calculated in parrallel. The concept of patches is not new and has been explored in [CGG*03], [LKES09], [LH04], among others. The absence of a tree structure to store the patches implies that all patches belong to the same level in the tree and have the same dimensions (though varying in the level of simplification). However, a hierarchical tree structure could additionally be imposed on top of the existing patches leading to further simplification of the mesh. This is recommended in cases where terrain size is very large. On the other hand, the tree structure constructs LODs for internal nodes and hence the preprocessing step would be necessary, should it be imposed on this algorithm.

The following sections will describe how points are selected inside a patch and how the shared border points are differentiated, before sending the points for Delaunay triangulation. Finally, the overall system will be explained in Sec. 4 and how the different parts are incorporated in the algorithm as well as how the algorithm is separated from the main thread.

3.1. Patch Points Selection

In order to achieve a more coarsened patch, fewer points are selected than what is available in the original dataset. Likewise, refinement is done by adding more points than what was present before refining the patch. In order to determine which points are required for each patch, the points in the original dataset are processed. The object-space ascent/descent of a terrain point with respect to its neighbors is projected onto the screen and measured against the error metric. However, where the ascent or descent is gradual, this approach may fail to include key points which may lead to oversimplification of the terrain. By also using the two closest points that have already been selected, these gradual differences are managed.

The point selection algorithm operates on single patches and computes the same border points for adjoining patches to achieve fully independent computation. This allows calculations in parallel as the process only requires access to the list of vertices and the position of the camera. In order to improve the quality of the triangulation, additional constraints were imposed where the middle point of a border was always added as well as restricting the maximum possible distance between selected points.

Algorithm 1 Point selection

```
1:  procedure CONSTRUCTPATCH(patch p, pixelError e)
2:      // Set of interior points included in the patch
3:      P := ∅
4:      // Set of border points included in the patch
5:      B := ∅
6:      for all indices i ∈ p do
7:          if i is a corner index then
8:              B.insert(i)
9:          else if i is a border index &
10:           (heightdiff(i − 1, i) > e ||
11:           heightdiff(i, B.back()) > e) then
12:             B.insert(i)
13:             // x: closest point from previous lines
14:         else if heightdiff(i − 1, i) > e ||
15:           heightdiff(i − width, i) > e ||
16:           heightdiff(i, P.back()) > e ||
17:           heightdiff(i, x) > e then
18:             P.insert(i)
19:     return P ∪ B
```

In Alg. 1, the step by step process is shown where the

function outputs a selection of indices based on the original dataset. The procedure $heightdiff(i, j)$ returns the absolute height difference between grid points i and j. The selection algorithm is an iterative process and tests whether or not the next point in succession will exceed the error metric when compared with other close by points. These points are the immediate neighbours taken from the previous column and row, as well as the closest selected point on the same row and previous rows. The selection process is visualized in Fig. 2.

3.2. Border Points Selection

In many cases, not all patches will require new triangulation when navigating across the mesh. If patches are selectively updated without caring for dependencies, incoherent borders are inevitable where one patch is updated while an adjacent patch is not. Since the border points are shared between adjacent patches, the logic for selecting border points was separated from the internal selection process. This allowed us to ensure continuous transitions without visual artifacts. The extent of the overlap between two patches is limited to a single row of border points.

Unlike the points located inside the patches, the border points only test their height differences with other border points from the original height dataset that lie along the same edge. This is required so as to produce identical border selection for two patches sharing a border. By associating the patches with the position of the camera when the update request was sent, the pixel error for producing the border points for any adjacent patch can always be reproduced. By doing so, high resolution patches can have seamless transitions to low resolution patches. This also implies that the border points will remain in the same state until both patches that share that border receive a new update request.

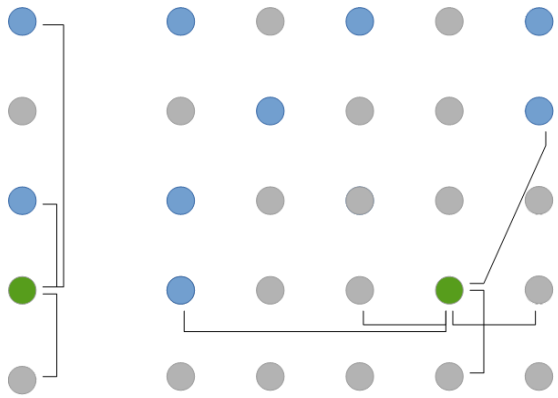

Figure 2: *Visualization of the selection process applied to a border (left image) and the interior of a patch (right image). The current point (green) is affected by its immediate neighbours as well as the closest selected points (blue).*

Fig. 2 shows how the selection algorithm operates on points belonging to a patch. In the figure, the unselected points are marked as grey, selected points as blue and the currently processed point as green. As discussed in both this section and Sec. 3.1, the current point is affected by its immediate neighbours as well as the closest selected points.

3.3. Delaunay Triangulation

Once the points in the patch are obtained from the selection process described in Sec. 3.1 and Sec. 3.2, the next step is to triangulate them. Delaunay triangulation operates by selecting triangles such that the minimum angle is maximized, so as to avoid creating thin triangles. The triangulation is applicable to points with and without internal structure, enabling triangulation for any form of point input data. A Delaunay triangulation has a circle associated with each triangle, such that no circle contains points from another triangle. To this end, techniques such as 3D Delaunay triangulation could be applied. However, given the large execution times as demonstrated in [Lo12], it would no longer allow real-time mesh simplification. We make the observation that the terrain datasets are in fact 2.5D and running the expensive 3D Delaunay triangulation could be avoided. Further, the points in a patch are originally picked from DEM and hence no two points have identical 2D coordinates. This implies that we can project the selected points of a patch on the $x-z$ plane dropping their height values, triangulate them using 2D Delaunay method, finally giving the points back their y values, lifting them back to 3D.

With this, the problem is simplified to two dimensions which reduces the complexity of the algorithm while still achieving a good triangulation, see also Fig. 5. This implementation strategy allowed calculation with 2D lines and circles, rather than 3D lines and spheres. Delaunay triangulation of the selected points in each patch is carried out in parallel using multi-core resources, thereby further reducing the overhead.

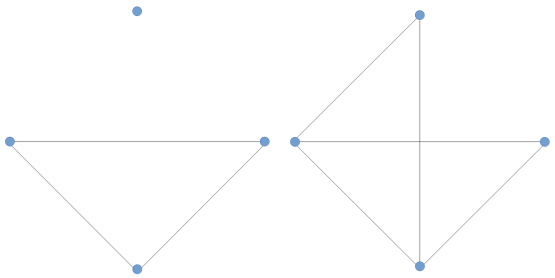

Figure 3: *Identical angles between points that can cause the algorithm to create conflicting triangles.*

A slight downside to the structure of points when creating TINs collected from a DEM was observed in instances where four points formed a square. The angles of the two hypotenuse lines would then be identical. Depending on the

order in which these points were processed, an overlap could occur which is illustrated in Fig. 3. This problem was mitigated by randomly offsetting the 2D points sent to the triangulation, thereby eliminating these identical angles.

In order to obtain a smoother representation at coarse LODs, the unselected neighbours could be used to interpolate the height data of a selected point. Doing so would however require updating to the vertex buffer during the update step, rather than keeping it static with changes localized to the index buffer.

4. System

While the entire pipeline could be carried out in the main thread, multi-threading is employed to make use of the multi-core architecture. The main thread is responsible for visibility checks, handling user input and rendering. A concurrent asynchronous thread is launched that manages the patches that require an update, selecting the points required to approximate the mesh in its current state, Delaunay triangulation and updating the index buffer. With this architecture, the main thread can be kept unblocked and the frame rates are more consistent. The entire procedure is outlined in Alg. 2.

The asynchronous thread starts off by determining which of the patches require new triangulation (line 3), by projecting the largest difference between points onto the screen. The difference value is then compared with the previous value from when the patch was last updated, to calculate the error metric for the patch and to ensure border consistency. If the difference does not exceed the user-defined threshold, the patch is excluded from the update. Only patches that are visible within the camera frustum are handled by the asynchronous thread.

After determining the patches that require an update, the point-selection process for each patch is launched as explained in Sec. 3.1 and Sec. 3.2. The list of patches is also sorted (line 12) to ensure that the patches closest to the camera are updated first. Each thread is given access to a region within the vertex list that is used to triangulate the patches and by sorting which set of boundaries each thread has access to, the order is rearranged.

The varying separation between the selected vertices could result in lost detail in patches with lower resolution, as pixels could receive different normals between updates, resulting in inconsistent lighting. To overcome these lighting issues, the normals are written into a texture and applied to the pixels in the pixel shader. The vastly increased end result can be observed in Fig. 4, where an area further away from the camera is examined more closely. This strategy is similar to normal mapping [TWBO03] but since all vertices have world space coordinates, these coordinates can be utilized to sample the appropriate normal from the texture without needing tangent plane to convert the coordinates to model space.

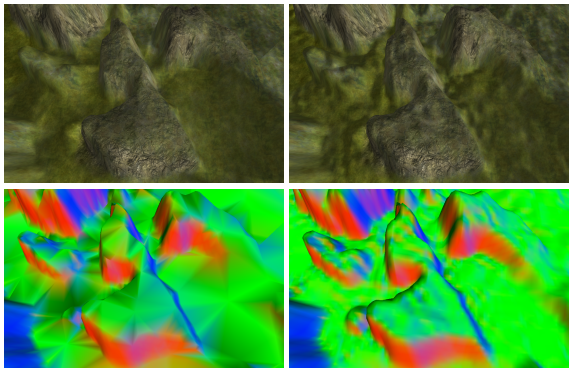

Figure 4: *View of coarsened patches where the top left image shows normals on a vertex level and the top right image shows normals applied on a pixel level. Bottom images display the normals used to apply light to the scene.*

Algorithm 2 Multi-resolution TIN system

```
1:  procedure UPDATEMESH( )
2:      //Camera position when initiating update
3:      determineOutdatedPatches(camPos)
4:      //Set of visible (outdated) patches
5:      P := ∅
6:      for all patches p ∈ outdatedPatchList do
7:          P.append(constructPatch(p, p.pixelError))
8:      //Set of 2D vertices used in the Delaunay step
9:      V := ∅
10:     for all indices i ∈ P do
11:         V.append(vertices[i])
12:     sortByDistance()
13:     for all patches p ∈ sortedList do
14:         //Triangle list
15:         T := ∅
16:         for all vertices v ∈ p do
17:             triangulate(v, V, T, p)
18:         mapToIndexBuffer(T)
```

The triangulation uses the selected 2D points and computes the patches both independently and in parallel. Once the triangulation has been completed for a patch, it is mapped into the corresponding area in the index buffer. The index buffer has a certain amount of elements allocated for each patch to ensure no data is overwritten and that no additional information is required to determine where to write the data. This does however mean that memory is allocated to manage a fully refined patch for every single area, thereby utilizing an abundant allocation scheme which in almost all cases wastes some memory. Although allocating more memory than required, culling, data retrieval and data management in general is simplified with this approach.

The result of the triangulation is cached, since the mesh is often valid for multiple frames (ranging from a few to thousands) depending on the camera movement. No tree structure for the patches was implemented, thereby forcing frustum checks against each patch. This could be improved by keeping a hierarchical data structure, that performs top-down visibility check. On the other hand, the current arrangement enables the algorithm to eliminate preprocessing completely, wherein the patches are adaptively triangulated on-the-fly.

5. RESULTS

The proposed algorithm was implemented in C++ using DirectX 11 and its programmable shader language HLSL. The tests were conducted on a 3.20GHz Intel Xeon processor with a NVIDIA Quadro 4000 graphics card.

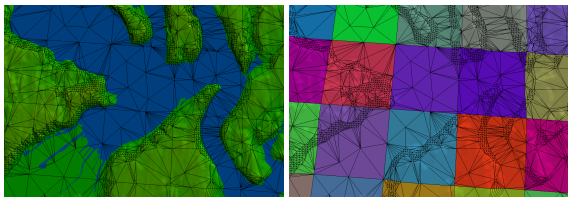

Figure 5: *Terrain mesh simplification achieved using the proposed algorithm in real time, where the triangulation can be seen on top of the rendered terrain (left image) along with patch association (right image).*

In terms of required preprocessing before the technique can be utilized, the presented algorithm requires very little time. Comparing with other TIN algorithms [BG04], [CGG*03], [HT97], [PAL02] that process up to the entire mesh outside of runtime, the presented algorithm only requires the vertex buffer holding the terrain mesh in its original format before it can be launched. The height data can be read from a height map or generated procedurally. Furthermore, no visual artifacts or inconsistencies are produced between different LODs, see also Fig. 5. For each dataset, the presented tests below are all based on the same conditions and the same predefined movement across the terrain, from a ground level perspective. The average frames rates for various dataset with increasing user-specified pixel error are presented in Tab. 1. The variation of the mesh quality with the pixel error is shown in Fig. 1.

A correlation between the number of triangles produced and the time spent on triangulation can be observed when comparing Fig. 6 and Fig. 7, where a lower percentage spent on triangulation corresponds to a lower triangle count. The graphs concern the same tests and are measured over the Puget Sound model, with four different pixel errors. The main aspect that is affected by the number of points is the geometrical calculations in the Delaunay triangulation. Not only is the algorithm forced to process a large amount of points for each triangle in dense patches, scarce patches will require many iterations before the slowly growing Delaunay circle can locate a final point to a triangle. Another reason why the higher error metrics achieve an overall lower triangles per second count, is the linear scalability of the selection process combined with the non-linear scalability of the triangulation process. The selection process still operates on the same data while the triangulation process operates on whatever amount data it receives. Furthermore, a lower amount of triangles are generated for higher error values which reduces the relative time spent triangulation, thereby resulting in a lower triangle per second value.

	Size of dataset		
Pixel error	256 × 256	512 × 512	1024 × 1024
0.25	3765	2657	1264
0.5	3938	2818	1232
1.0	3953	2868	1290
2.0	4035	2904	1378

Table 1: *The average frames per second (fps) for various dataset sizes as a function of the user-defined pixel error.*

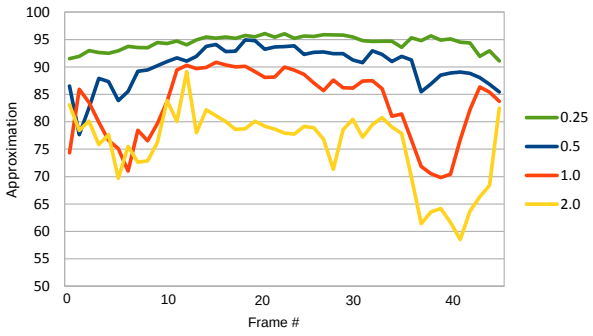

Figure 6: *The percentage of the algorithm that is spent on triangulation across the simulation with different error metrics.*

The dimensions of the patches also affect the performance of the algorithm, where patches that are too large process too many points and patches that are too small increase the overhead. As shown in Tab. 2, the most suited patch dimension is 32 × 32 and was therefore used in previous tests. When determining the suitable patch dimension, the tests were conducted on the Puget Sound model with a pixel error of 0.5.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a novel LOD-based approach for TIN simplification that is capable of generating and stitching various resolutions at runtime. Our approach can employ simplification by leveraging 2D Delaunay triangulation while simultaneously reducing triangle count by

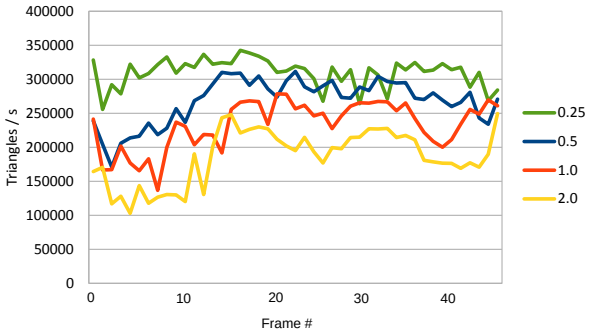

Figure 7: *The performance across the simulation with different error measurements.*

Patch dim	Max	Min	Average
8 × 8	3.45	0.41	1.94
16 × 16	1.57	0.43	0.90
32 × 32	1.19	0.19	0.68
64 × 64	2.45	0.25	1.01

Table 2: *The maximum, minimum and average number of seconds that the algorithm took to complete one update.*

examining the features based on a user-defined error metric. The technique uses multi-core resources to process selection of vertices representing the terrain patches and to triangulate them in parallel.

In terms of future work, there are a few aspects of which to focus on, such as delegating the entire triangulation to the GPU and adding a postprocessing step to improve the quality of the triangulation. Our algorithm was designed around parallelism so triangulating on the GPU is an interesting extension to investigate. If a more complex approximation technique is used, then focus should still revolve around fast processing, without sacrificing the approximation quality.

References

[AW03] AMARATUNGA K., WU J.: Wavelet triangulated irregular networks. *International Journal of Geographical Information Science 17*, 3 (2003), 273 – 289. 2

[BG04] BARTHOLDI J. J. I., GOLDSMAN P.: Multiresolution indexing of triangulated irregular networks. *IEEE Transactions on Visualization and Computer Graphics 10*, 4 (2004), 484 – 495. 2, 3, 6

[BGP09] BÖSCH J., GOSWAMI P., PAJAROLA R.: RASTeR: Simple and efficient terrain rendering on the GPU. In *Proceedings EUROGRAPHICS Areas Papers, Scientific Visulization* (2009), pp. 35 – 42. 2

[CGG*03] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Bdam - batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (2003), 505 – 514. 2, 3, 6

[COL96] COHEN-OR D., LEVANONI Y.: Temporal continuity of levels of detail in delaunay triangulated terrain. pp. 37 – 42. 2

[CPS97] CIGNONI P., PUPPO E., SCOPIGNO R.: Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer 13*, 5 (1997), 199 – 217. 2

[DFP95] DE FLORIANI L., PUPPO E.: Hierarchical triangulation for multiresolution surface description. *ACM Transactions on Graphics (TOG) 14*, 4 (1995), 363 – 411. 2

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D., MILLER M., ALDRICH C., MINEEV-WEINSTEIN M.: Roaming terrain: real-time optimally adapting meshes. IEEE Computer Society Press, pp. 81 – 88. 2

[EKT01] EVANS W., KIRKPATRICK D., TOWNSEND G.: Right-triangulated irregular networks. *Algorithmica 30*, 2 (2001), 264 – 286. 2

[GGS95] GROSS M. H., GATTI R., STAADT O.: Fast multiresolution surface meshing. pp. 135 – 142. 2

[GH99] GUMHOLD S., HÃIJTTNER T.: Multiresolution rendering with displacement mapping. ACM, pp. 55 – 66. 2

[GMBP10] GOSWAMI P., MAKHINYA M., BÖSCH J., PAJAROLA R.: Scalable parallel out-of-core terrain rendering. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization* (2010), pp. 63–71. 2

[Hop98] HOPPE H.: Smooth view-dependent level-of-detail control and its application to terrain rendering. IEEE Computer Society Press, pp. 35 – 42. 2

[HT97] HUANG S.-J., TSENG D.-C.: Construction of multiresolution terrain models using hierarchical delaunay triangulated irregular networks. vol. 4, pp. 1999 – 2001 vol.4. 2, 6

[KK02] KOLINGEROVÃĄ I., KOHOUT J.: Optimistic parallel delaunay triangulation. *The Visual Computer 18*, 8 (2002), 511 – 529. 3

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 769 – 776. 2, 3

[LKES09] LIVNY Y., KOGAN Z., EL-SANA J.: Seamless patches for gpu-based terrain rendering. *The Visual Computer 25*, 3 (2009), 197 – 208. 2, 3

[LKR*96] LINDSTROM P., KOLLER D., RIBARSKY W., HODGES L., FAUST N., TURNER G.: Real-time, continuous level of detail rendering of height fields. ACM, pp. 109 – 118. 2

[Lo12] LO S. H.: Parallel delaunay triangulation in three dimensions. *Computer Methods in Applied Mechanics and Engineering 237-240* (2012), 88 – 106. 3, 4

[LP02] LINDSTROM P., PASCUCCI V.: Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 239 – 254. 2

[Mus97] MUSIN O.: Properties of the delaunay triangulation. ACM, pp. 424 – 426. 2

[PAL02] PAJAROLA R., ANTONIJUAN M., LARIO R.: Quadtin: quadtree based triangulated irregular networks. IEEE Computer Society, pp. 395 – 402. 2, 6

[PG07] PAJAROLA R., GOBBETTI E.: Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer 23*, 8 (2007), 583 – 605. 1, 2

[SS09] SCHWARZ M., STAMMINGER M.: Fast gpu-based adaptive tessellation with cuda. *Computer Graphics Forum 28*, 2 (2009), 365 – 374. 2

[TGM*06] TANG T., GONG H., MO Y., DUAN F., ZHAO W.: Dynamic data retrieval and distance decay of triangulated irregular network(tin) in three dimensional visualizations. *Geographic Information Sciences 12*, 1 (2006), 21. 2

[TWBO03] TASDIZEN T., WHITAKER R., BURCHARD P., OSHER S.: Geometric surface processing via normal maps. *ACM Transactions on Graphics (TOG) 22*, 4 (2003), 1012 – 1033. 5

[XV96] XIA J. C., VARSHNEY A.: Dynamic view-dependent simplification for polygonal models. pp. 327 – 334. 2

[YLS05] YANG B., LI Q., SHI W.: Constructing multi-resolution triangulated irregular network model for visualization. *Computers and Geosciences 31*, 1 (2005), 77 – 86. 2