A HYBRID APPROACH FOR HIGH QUALITY REAL-TIME TERRAIN RENDERING AND OPTIMIZED A-PRIORI ERROR ESTIMATION

Falko Löffler, Sebastian Schwanke and Heidrun Schumann
Institute of Computer Science and EE, University of Rostock, Albert-Einstein-Strasse 21, Rostock, Germany

Keywords: Terrain rendering, Level of detail, Longest-edge-bisection, Refinement criteria, Cached geometry, Multi-core CPU.

Abstract: This paper describes a hybrid approach for high-quality real-time terrain rendering. The contributions of the work are twofold. First, a novel parallel preprocessing scheme for necessary a-priori error-bounds calculation based on the widely applied longest-edge-bisection approach is proposed, which exploits current multi-core CPU architectures. Compared to the common recursive and thus computationally expensive procedure, a significant performance increase can be achieved. Second, a novel method for view-dependent terrain rendering is described which combines the advantages of triangle-based CPU and patch-optimized GPU algorithms. We exploit frame-to-frame coherence by caching refined geometry on local VRAM in combination with an optimized update process. In contrast to patch-based methods, a substantial reduction of the number of primitives and rendering time can be achieved.

1 INTRODUCTION

In the last decade, many terrain rendering algorithms have been developed. Due to the close interrelation of rendering quality, rendering time and available resources, a compromise between these aspects has to be found (Löffler et al., 2009). This is a challenging task.

To achieve high-quality images in real-time, a multi-resolution data structure is required, which is generated in a preprocess. In particular, triangulated irregular networks (TIN) require an exhaustive preprocess and large storage. Thus, data structures on a semi-regular basis are commonly used. This leads to an improved balance in terms of preprocessing time and the number of primitives to be rendered. For this purpose, the *longest-edge-bisection* scheme is widely applied (e.g., in (Pajarola and Gobbetti, 2007)). It requires a recursive and therefore computationally expensive a-priori object-space error determination. The object space errors serve as refinement criteria for the view dependent triangle- or patch-based rendering process. For high-quality terrain rendering, small error tolerances have to be used. In the case of triangle-based algorithms this leads to the well-known CPU-GPU bottleneck. Patch-based algorithms reduce this bottleneck, but on the cost of a higher number of primitives to be rendered.

The goal of this work is two-fold. First, we accelerate the process of refinement criteria determination by exploiting multi-core CPU architectures. We therefore introduce a novel parallel a-priori error estimation scheme characterized in Section 3. As a result, preprocessing time could be decreased by several orders of magnitude. Second, we combine the advantages of triangle- and patch-based approaches by proposing a novel hybrid approach for view-dependent refinement (see Section 4). Given a particular error tolerance, we reduce the number of primitives and speed up the rendering process by exploiting frame-to-frame coherence. The results are presented in Section 5. Before going into detail, we reflect previous work in the next section.

2 RELATED WORK

In the last decade, many terrain rendering algorithms have been proposed. High-quality real-time rendering, especially terrain rendering, has to deal with very large data-sets. Hence, the usage of multi-resolution data structures is indispensable to gain interactive frame rates. Two types of data structures are

Löffler F., Schwanke S. and Schumann H. (2010).
A HYBRID APPROACH FOR HIGH QUALITY REAL-TIME TERRAIN RENDERING AND OPTIMIZED A-PRIORI ERROR ESTIMATION.
In *Proceedings of the International Conference on Computer Graphics Theory and Applications*, pages 233-238
DOI: 10.5220/0002824602330238

commonly used: First, techniques that are based on general, mainly unconstrained, triangulations (TIN), and second, semi-regular representations. Examples of the first category are described in (Puppo, 1998; Cignoni et al., 1997; Hoppe, 1998) and, e.g., the GPU-based approach of (Hu et al., 2009). These algorithms generate a high-quality terrain mesh with an adequate number of triangles (Evans et al., 2001). However, these algorithms require complex data structures and a complex preprocessing.

Algorithms of the second category simplify the preprocess on the cost of a higher number of triangles to be rendered. Most of these algorithms are based on the simple, but powerful subdivision scheme of *longest–edge–bisection* (Duchaineau et al., 1997; Pajarola, 1998; Lindstrom et al., 1996; Röttger et al., 1998; Evans et al., 2001). This scheme recursively splits an isosceles right triangle (vl, va, vr) at the midpoint vm of its hypotenuse (vl, vr) into two triangles (vl, vm, va) and (va, vm, vr). As described in (Lindstrom and Pascucci, 2002), such a mesh can be represented as a direct acyclic graph (DAG), the vertex graph. The mesh vertexes define the node set, whereas the edge set is defined by all directed edges (va, vm) that correspond to a triangle split. The graph is described by a recursive indexing scheme. During the preprocess, each vertex is associated with the refinement criteria including the bounding sphere error radius and the radii of its child nodes (also referred to as nested error saturation to guarantee crack-freeness).

In general, two types of rendering algorithms are applied: First, those working at the triangle/vertex primitive level and those at clusters of primitives (patches). Approaches of the first category need fewer triangles but lead to the CPU-GPU bottleneck. To overcome this bottleneck, approaches of the second category use patches optimized for GPU throughput requiring just a few CPU instructions (Cignoni et al., 2005). Hence, more geometry has to be rendered. For instance, (Levenberg, 2002) dynamically extracts triangle clusters during view-dependent rendering and caches the geometry on the GPU. Following this approach, various GPU-oriented multi-resolution structures have been proposed, for instance (Pomeranz, 2000; Cignoni et al., 2003; Hwa et al., 2004; Bösch et al., 2009). Especially (Cignoni et al., 2003) combines the advantages of semi-regular structures and TIN for high performance terrain rendering. For an excellent overview, we refer to (Pajarola and Gobbetti, 2007).

Triangle-based and patch-based approaches as well require an adequate a priori error estimation. Therefore, we introduce our error estimation scheme in Section 3 before discussing our new hybrid terrain rendering method in Section 4.

3 A-PRIORI ERROR ESTIMATION

Using common recursive a-priori error estimation schemes for parent-child relationship determination requires the bottom-up traversal of the vertex graph hierarchy node-by-node. In order to accelerate this process, we take advantage of recent multi-core CPU architectures. However, the challenge is to determine the parent-child relationships in parallel. Our novel solution is to harness these relationships from diamond entities. Diamonds can be characterized consisting of triangles of the same level, which share their adjacent hypotenuse (see, i.e., (Hwa et al., 2004)). A diamond is associated with a center vertex (the *diamond vertex*), the diagonal edge, and one quadrilateral face. The parents and children of a diamond can be described using a special indexing scheme (see Section 3.1), which we adapt to the vertex graph structure since each vertex in the vertex graph represents a diamond. Parent-child relationships of two consecutive levels can now be extracted, if those of the coarser level are known (see Section 3.2). The novelty is that we are enabled to consider all vertexes of one level independently of each other. Thus, we determine the refinement criteria by sequentially treating the levels bottom-up combined with their inner-level determination in a parallel way.

3.1 Basic Indexing Scheme

Following the indexing scheme of (Hwa et al., 2004), the face of a diamond d is described by the diamond ancestors $a_0, \ldots, a_4$ (see Figure 1). The hypotenuse

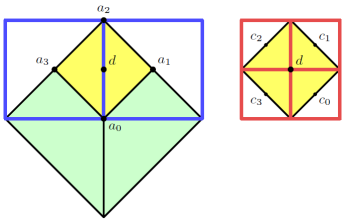

Figure 1: A diamond d (yellow) is shown with its ancestors (left) and its children (right)(Hwa et al., 2004). The green ancestor is a_0, the two parents are a_1 and a_3 (blue outline). ds children are $c_{0\ldots3}$ (red). Note that the orientation is rotated by 45° each level.

of the two adjacent triangles, is characterized by the ancestors a_0 and a_2 and split by d. Using the locations

of the ancestors, the locations of the child diamonds c_i with $i = 0,\ldots,3$ can be determined as follows:

$$c_i = (a_i + a_{(i+1) \mod 4})/2 \quad (1)$$

Using this information, the computation and saturation of the refinement criteria of a diamond d as described in (Lindstrom and Pascucci, 2002) is possible. However, only implicit access is feasible due to the recursive indexing scheme. Consequentially, we introduce how to solve parent-child relationships with regard to an explicit position of a vertex in the vertex-graph in the next section.

3.2 Our General Scheme

We must identify the four ancestors of a vertex in the vertex graph to adapt the indexing scheme described above in order to determine parent-child relationships. However, three problems for arbitrary vertex positions can be identified: First, the orientation of a diamond alternates from level to level. Second, different levels form different diamond sizes and dependent on that, different ancestor permutations exist. To accomplish this, a computation rule for ancestor identification is required.

For that reason, we position special *diamond kernels* with a kernel configuration k and the kernel position $\mathbf{k_p}$ on appropriate vertexes. We address the first problem by categorizing each level l into an $even = (n \mod 2 == 0)$ and an $odd = (n \mod 2 == 1)$ level, where $2*n$ levels exist with regard to height field sizes of $2^n + 1$. Hence, we introduce two different diamond kernels for both level types. The configurations are illustrated in Figure 2.

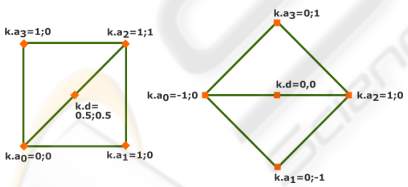

Figure 2: Configuration of the diamond kernels for *even* (left) and *odd* (right) levels, respectively.

To address the second and third problem, we determine the distance $s = 2^{l/2}$ of parent and child diamond vertexes and introduce bias b in dependence on l. b represents the specific permutation of the ancestors. As it can be seen, the ancestor permutation of the split edge a_0, a_2 can be neglected, since the vertex order is only relevant for rendering in the sense of back-face culling. Hence, we derive two cases: $b = 1$ represents a horizontally, $b = 0$ a vertically aligned diamond.

b can be determined as follows:

$$b = \left\{ \begin{array}{ll} (k_{p.x} + k_{p.y}) \mod 2 & l = even \\ k_{p.y} \mod 2 & l = odd \end{array} \right\} \quad (2)$$

With the help of these parameters, the locations of the ancestor a_i and the diamond vertex d can now be determined as follows, where `makeIdx` realizes the mapping of a position to an index:

$$j = (i + b) \mod 4 \quad (3)$$

$$a_i = \text{makeIdx}((\mathbf{k_p} + \mathbf{k.a_j}) * s) \quad (4)$$

$$d = \text{makeIdx}(\mathbf{k_p} * s + \mathbf{k.d} * s) \quad (5)$$

Using this approach, we can determine the refinement criteria in a bottom-up fashion by applying the *diamond kernels* in parallel for each level. This is achievable due to fact that the kernel attributes only depend on the current level and the position of the kernel (see an example in Figure 3).

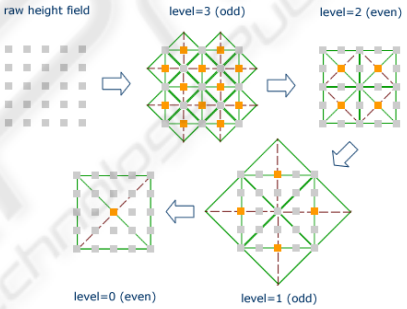

Figure 3: Processing sequence of a 5x5 height-field. Starting at the finest level, for each diamond on a level the refinement criteria are computed and saturated by maximizing the error over the child vertexes. The vertexes on a level are processed in parallel.

4 OUR HYBRID APPROACH

The key idea is now to combine the advantages of triangle- and patch-based approaches for view-dependent rendering. Our procedure is as follows: First, we construct a semi-regular multi-resolution data structure of patches in a preprocess (see Section 4.1) based on the refinement criteria (see Section 3). During rendering, we select appropriate patches. These patches are then dynamically triangulated (see Section 4.2) and cached on the GPU for frame-to-frame-coherent reuse. This commonly requires an extensive cache validity evaluation for subsequent frames. Hence, Section 4.3 introduces our update process including a simple but efficient cache validity evaluation. This results in a reduction of the vast triangle workload given a particular error tolerance during the rendering process.

4.1 Patch LOD Construction

In an a-priori step, we construct a multi-resolution hierarchy of patches. A cut through this hierarchy represents a coarse grain triangulation of the terrain, in that each precomputed patch triangle represents a regular sub-triangulation. Therefore, it is necessary to derive patch-related refinement criteria, which we derive using the per-vertex refinement criteria (see Section 3) by estimating the maximum error of the patch as a consequence. Thus, the patch-related criteria are also nested. Our hierarchical patch structure is based on a *diamond-graph* as proposed in (Hwa et al., 2004) since this data structure guarantees crack-freeness in combination with the nesting properties. In this way, each diamond is associated with vertex attributes of two triangular patches with an adjacent hypotenuse. Using this structure, we can use the algorithm of (Lindstrom and Pascucci, 2002) to select an appropriate patch during rendering.

4.2 View-dependent Triangulation

Selecting a patch naively represents a regular sub-triangulation. Hence, we dynamically refine the selected patch and therefore adapt the algorithm of (Lindstrom and Pascucci, 2002) for triangular patches over a regular grid of the size $2^n + 1$ (see Section 2). The resulting strip is transferred to the graphics board. According to (Lindstrom and Pascucci, 2002), we perform the refinement of patches and the rendering concurrently. However, the refinement from scratch of each visible patch in each frame is a tedious task.

Hence, we store triangulation results on local VRAM for frame-to-frame coherent reuse, which enables us to fully exploit the rendering performance of recent GPUs and to reduce CPU-GPU transfer. Though, the necessary validity re-evaluation requires expensive CPU workload due to the per-vertex evaluation. We therefore introduce our update process including an optimized error evaluation scheme in the next section.

4.3 Update Process

In subsequent frames, our update process determines for each selected patch if a cached triangulation is still valid leading to its reuse, or not, which results in a view-dependent triangulation. We first describe some basics for determining the appropriate triangulation during the rendering process, which we adapt to our update mechanism.

During the rendering process, the cut, i.e. the level of mesh detail with regard to given refinement criteria (including all so-called active vertexes) has to be determined (Puppo, 1998). To decide the activity state of each vertex v_i, the screen-space error tolerance τ can be evaluated in dependence on the associated object-space error δ_i, the vertex position p_i, the bounding sphere error radius r_i and a perspective projection factor λ as follows (Lindstrom and Pascucci, 2002):

$$\tau < \lambda \frac{\delta_i}{\| \mathbf{p_i} - \mathbf{e} \| - r_i} \tag{6}$$

Hence, all vertexes on the cut - the vertex frontier - and "above" are active and thus part of the triangulation, whereas the other vertexes are inactive. This means that this cut is valid for subsequent view-points, if the activity state of all vertexes does not change. However, this requires both storing and expensively evaluating the entire vertex frontier. To avoid this, we confine the validity evaluation to checking the new view-point against a *sphere of validity* determined during the view-dependent triangulation of the patch.

The utilized error metric (see Equation 6) is isotropic and can be interpreted as blowing up a spherical region $s_i = (\mathbf{p}_i, r_i^s)$ around each vertex v_i where r_i^s represents the distance between the view-point and the vertex v_i with respect to Equation 6. Hence, we rearrange the equation to:

$$r_i^s = \frac{\delta_i \lambda}{\tau} + r_i \tag{7}$$

As a consequence, the vertex is active, if the view-point $\mathbf{e}$ is located in that sphere, inactive otherwise. The *region of validity* of a cut can now be described as the *difference* of the following two regions (see Figure 4): First, the *intersection* of all spheres of all active vertexes. If the view-point leaves this region, one of the vertexes becomes inactive. Second, the *union* of the spheres of all inactive vertexes. If the view-point enters this region, one of the vertexes becomes active.

However, this region is very expensive to evaluate. Hence, we simplify this region to a sphere characterized by the current view-point $\mathbf{e}$ and the minimal distance ε between $\mathbf{e}$ and any sphere s_i as follows:

$$\varepsilon = \min_{i=0}^{n} | \, r_i^s - \| \mathbf{p_i} - \mathbf{e} \| \, | \tag{8}$$

The resulting *sphere of validity* is an upper estimate and thus we guarantee that a triangulation is always valid, if subsequent view-points $\mathbf{e}'$ are inside this sphere. Therefore, we only need to evaluate, if $\varepsilon > \| \mathbf{e} - \mathbf{e}' \|$.

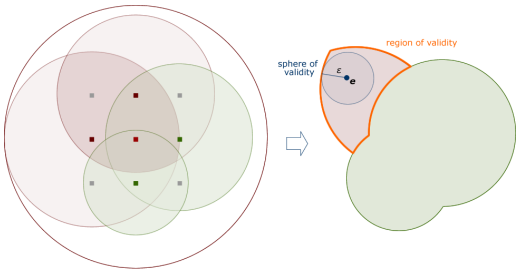

Figure 4: Red represents the spheres of the active vertexes and green those of the inactive vertexes with respective *region of validity* and *sphere of validity* for a view-point.

5 RESULTS

We implemented our approaches using C++ and OpenGL. We measured the performance of our novel a-priori error estimation scheme with 1k, 2k, 4k and 8k height fields. Two different hardware configurations were utilized including a single-core Pentium 4 3 GHz and a quad-core 2,83 GHz CPU in order to evaluate hardware parallelism. The respective preprocessing times are shown in Table 1. The results clearly show that using our parallel error estimation scheme decreases preprocessing times in any case. On both configurations, our approach diminishes preprocessing time linearly with increasing height field sizes. On the single-core environment our approach accelerates the processing by factor 4-5, whereas exploiting hardware parallelism, the factor even rises up to 10-12. As a result, our approach is well-suited for on-the-fly height field processing.

The results of our hybrid approach have been captured on the quad-core configuration equipped with a GeForce GTX 280, using 1k, 2k and 4k height field dimensions and a screen-resolution of 1280x800. We compared our hybrid method with a representative pre-generated patch-based method with a static patch size of 65 according to (Bösch et al., 2009). Rendering and dynamic tessellation were driven concurrently (Lindstrom and Pascucci, 2002). The results are reported in Table 2. As figured out, our approach significantly reduces the triangle count per frame (by up to 84%) with regard to the minimum error tolerance. With small error tolerances in general, high resolution patches must be selected, which results in a vast amount of rendered primitives. In contrast, an entire significant performance gain is achieved using our method, particularly with increasing error tolerances. This is due to the frequent cache reuse capability of our algorithm. In terms of minimum tolerances, the necessary high re-triangulation effort affects rendering performance and caching capabilities.

Table 1: Performance comparison of preprocessing times in seconds in regard to the hardware configurations. Note that *rpcs* means the straight forward recursive computation scheme, while *ppcs* represents our novel parallel computation scheme.

CPU	Height field size							
	1k		2k		4k		8k	
	rpcs	*ppcs*	*rpcs*	*ppcs*	*rpcs*	*ppcs*	*rpcs*	*ppcs*
Single – Core	1,43	0,28	5,17	1,18	20,03	4,71	65,3	16,9
Quad – Core	0,58	0,05	2,27	0,20	9,18	0,76	36,8	3.18

Table 2: Results of the *quad-core* test environment with regard to a patch-based implementation compared to our hybrid approach. The table shows the average frame rate fps and the average number of rendered triangles per frame num_Δ.

Error tolerance	Height field size					
	1k		2k		4k	
	fps	num_Δ	fps	num_Δ	fps	num_Δ
	Patch-based approach					
0,5	284	1082k	143	2732k	82	4642k
1,0	286	1079k	191	1885k	121	3001k
2,0	291	1037k	261	1223k	212	1522k
	Our hybrid approach					
0,5	299	947k	172	665k	113	758k
1,0	427	472k	343	343k	274	339k
2,0	570	175k	566	147k	497	136k

6 CONCLUSIONS

We presented a novel approach for a-priori error estimation scheme exploiting hardware parallelism which is widely applicable. Furthermore, we proposed a novel hybrid approach for view-dependent refinement. This algorithm incorporates the advantages of both patch-based and triangle-based data structures. In more detail, we suggested selecting appropriate patches from a predetermined patch hierarchy and refining them on a per-triangle basis. Additionally, an optimized cache validity evaluation for reuse of geometry persisting in the GPU memory was introduced. The results reported improved preprocess time, reduction of triangles per frames as well as a gain in rendering performance.

We see the scope of future work in extending our scheme for out-of-core processing. Moreover, we would further harness heterogeneous hardware platforms in general (for instance, OpenCL). This includes the a-priori error estimation and view-dependent refinement. A further point of interest is to adapt our hybrid method to TINs due to the possible reduction of primitives to be rendered.

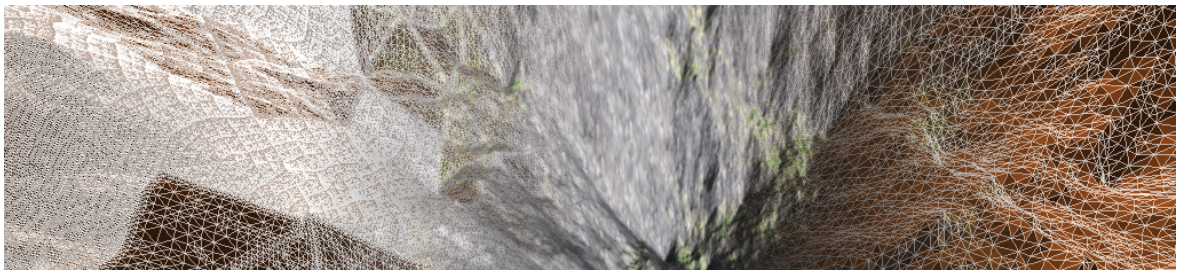

Figure 5: Comparison of preprocessed and dynamic triangulation of patches. left: static pre-generated patch triangulation; middle: textured representation; right: dynamic triangulation generated by our hybrid approach. Notice the significant reduction of triangle count on the right side.

REFERENCES

Bösch, J., Goswami, P., and Pajarola, R. (2009). RASTeR: Simple and Efficient Terrain Rendering on the GPU. In *Eurographics 2009 - Areas Papers*, pages 35–42. Eurographics Association.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2003). BDAM - batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22:505–514.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2005). Batched multi triangulation. *IEEE Visualization, 2005. VIS 05*, pages 207–214.

Cignoni, P., Puppo, E., and Scopigno, R. (1997). Representation and visualization of terrain surfaces at variable resolution. In *The Visual Computer*, pages 50–68.

Duchaineau, M. A., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C., and Mineev-Weinstein, M. B. (1997). Roaming terrain: real-time optimally adapting meshes. In *IEEE Visualization*, pages 81–88.

Evans, W., Kirkpatrick, D., and Townsend, G. (2001). Right-triangulated irregular networks. *Algorithmica*, 30(2):264–286.

Hoppe, H. (1998). Smooth view-dependent level-of-detail control and its application to terrain rendering. *Visualization Conference, IEEE*, 0:35.

Hu, L., Sander, P., and Hoppe, H. (2009). Parallel view-dependent refinement of progressive meshes. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 169–176. ACM New York, NY, USA.

Hwa, L. M., Duchaineau, M. A., and Joy, K. I. (2004). Adaptive 4-8 texture hierarchies. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 219–226, Washington, DC, USA. IEEE Computer Society.

Levenberg, J. (2002). Fast view-dependent level-of-detail rendering using cached geometry. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 259–266, Washington, DC, USA. IEEE Computer Society.

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F., Faust, N., and Turner, G. A. (1996). Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118, New York, NY, USA. ACM.

Lindstrom, P. and Pascucci, V. (2002). Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254.

Löffler, F., Rybacki, S., and Schumann, H. (2009). Error-Bounded GPU-Supported Terrain Visualisation. In *WSCG'09 Communication Papers Proceedings*, pages 47–54. University of West Bohemia.

Pajarola, R. (1998). Large scale terrain visualization using the restricted quadtree triangulation. *Visualization Conference, IEEE*, 0:19.

Pajarola, R. and Gobbetti, E. (2007). Survey on semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23(8):583–605.

Pomeranz, A. (2000). *ROAM Using Surface Triangle Clusters (RUSTiC)*. PhD thesis, UNIVERSITY OF CALIFORNIA.

Puppo, E. (1998). Variable resolution triangulations. *Computational Geometry: Theory and Applications*, 11(3-4):219–238.

Röttger, S., Heidrich, W., and Seidel, H.-P. (1998). Real-time generation of continuous levels of detail for height fields. pages 315–322.