Eurographics Symposium on Parallel Graphics and Visualization (2011)
T. Kuhlen, R. Pajarola, and K. Zhou (Editors)

GPU Algorithms for Diamond-based Multiresolution Terrain Processing

M. Adil Yalçın[†1], Kenneth Weiss[‡1] and Leila De Floriani[§2]

[1]University of Maryland, College Park, USA [2]University of Genova, Genova, Italy

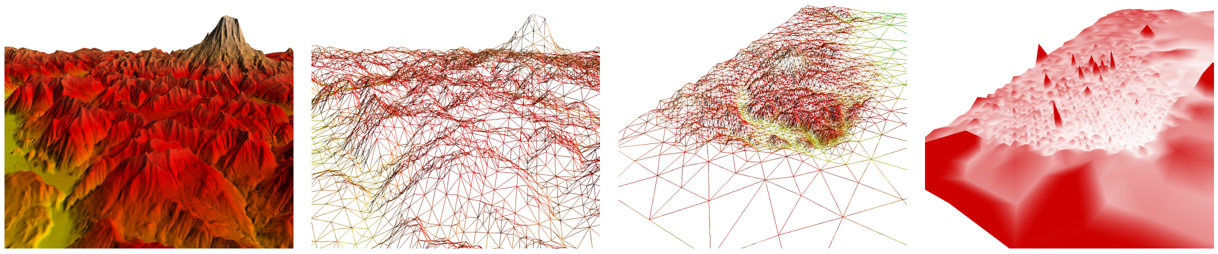


Figure 1: Left: Variable-resolution terrain extracted from the Puget Sound dataset at a fixed viewpoint (textured and wireframe). Right: Alternate view of the same mesh illustrating the view-dependent triangulation and error metric on the mesh vertices.

Abstract

We present parallel algorithms for processing, extracting and rendering adaptively sampled regular terrain datasets represented as a multiresolution model defined by a super-square-based diamond hierarchy. *This model represents a terrain as a nested triangle mesh generated through a series of longest edge bisections and encoded in an implicit hierarchical structure, which clusters triangles into* diamonds *and diamonds into* super-squares. *We decompose the problem into three parallel algorithms for performing: generation of the diamond hierarchy from a regularly distributed terrain dataset, selective refinement on the diamond hierarchy and generation of the corresponding crack-free triangle mesh for processing and rendering. We avoid the data transfer bottleneck common to previous approaches by processing all data entirely on the GPU. We demonstrate that this parallel approach can be successfully applied to interactive terrain visualization with a high tessellation quality on commodity GPUs.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations. I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types. I.3.m [Computer Graphics]: Misc.—Parallel rendering.

1. Introduction

Advances in computing hardware in recent years have shifted from improvements in the speeds of individual processors to increases in the number of cores on systems with multiple programmable processing units. This requires efficient parallel algorithms to take advantage of the increased computing power. This is especially the case for graphics hardware, where GPUs now have hundreds of cores powering thousands of lightweight threads that allow data and task parallelism in a highly programmable environment.

Interactive terrain rendering and processing has had continuing interest over the past several decades in many diverse fields including Geographic Information Systems (GIS), computer graphics, and scientific visualization (see [PG07] for a recent comprehensive survey). The increasing size of

† yalcin@cs.umd.edu
‡ kweiss@cs.umd.edu
§ deflo@disi.unige.it

available terrain datasets, made possible by improved sensing technologies, necessitates a multiresolution terrain representation from which *conforming*, i.e. crack-free, adaptive meshes can be interactively extracted and processed. Such extractions are driven by an application–specific predicate, referred to as a *selection criterion*, that can be tested against the nodes of the multiresolution model to determine if they contribute to the currently extracted mesh. A popular refinement operator for generating multiresolution models from regularly sampled datasets is *longest edge bisection*. In this scheme, isosceles right triangles are bisected along the midpoints of their hypotenuse into two similar triangles. Since pairs of triangles sharing a common longest edge, referred to as *diamonds*, need to be bisected concurrently to maintain a crack-free adaptive mesh, diamond-based representations have been a popular multiresolution model over such datasets. The use of the diamond primitive enables an implicit encoding of all hierarchical and geometric relationships among the elements of the hierarchy [Paj98, HDJ05, WD08].

Terrain datasets are typically provided as regularly sampled grids of elevation values. This regularity has led to the development of efficient parallel algorithms for processing, visualization and analysis. Many recent approaches make use of a CPU-resident coarse-grained multiresolution data structure, where each *macro* update on the CPU is associated with a *batched* update to the underlying triangulation data consisting of many triangles optimized for efficient streaming and rendering on the GPU [Pom00, Lev02, CGG*03, HDJ05, BGP09, LKES09]. These approaches typically require a hierarchical selection criterion in the form of an approximation error that has been *saturated*, i.e. the error associated with each node in the hierarchy must be greater than those of its descendants. This requires an expensive preprocessing stage, and can restrict the number of different meshes which can be extracted. Moreover, any modifications to the data can invalidate the error metric, requiring a new calculation over large portions of the dataset.

Here, we explore an alternative approach based on fine-grained updates to datasets that are entirely GPU-resident. To this aim, we use a diamond-based model and show how to generate the multiresolution terrain model and how to perform selective refinement on the model, thus extracting crack-free meshes. This constitutes the basis for a system for terrain modeling, analysis and rendering on the GPU. In particular, our system enables the use of an arbitrary (i.e. unsaturated) predicate as a selection criterion, as illustrated in Figure 1. We consider parallel rendering as a first application of our GPU-based multiresolution framework, although we intend to apply this framework to other terrain processing problems, such as parallel watershed and erosion analysis and visibility queries, on the extracted meshes.

Thus, we focus on the three primary problems: (a) parallel generation of the diamond-based multiresolution model, that is, computing the approximation error for each diamond; (b) parallel selective refinement, which requires the application of the selection criterion to the mesh elements in parallel followed by a parallel *closure* operation to ensure the extraction of crack-free meshes; and (c) generating and rendering the corresponding triangle mesh in parallel.

Our system provides a coherent and integral fine-grained approach multiresolution terrain processing directly on the GPU using the OpenCL API. Although current implementations of OpenCL have not been fully optimized compared to shader languages (e.g. HLSL, GLSL) or platform specific languages such as CUDA, our system achieves real time performance on $2k \times 2k$ datasets and interactive performance on $4k \times 4k$ datasets using commodity GPUs. We expect these results to improve as the OpenCL language matures.

The remainder of the paper is organized as follows. In Section 2, we review related work, and in Section 3, we review a multiresolution terrain model built on diamonds. In Section 4, we provide an overview of our approach. In Section 5, we describe the generation of the diamond hierarchy. In the next three sections, we describe the three phases of our selective refinement approach, namely, the error evaluation on the diamonds (Section 6), the closure of the dependency relation (Section 7) and the generation of the corresponding triangle mesh (Section 8). We discuss performance results in Section 9. Finally, in Section 10, we draw some concluding remarks and discuss current and future work.

2. Related work

In this section, we review related work on multiresolution terrain processing of large datasets and on parallel processing algorithms for such datasets.

Multiresolution modeling of terrains has been an active area of research over the past several decades, intially in GISs and in terrain rendering for the development of geobrowsers, video games and flight simulators. The various approaches have been based on irregular triangle meshes (TINs) and more recently on nested triangle hierarchies built on regularly sampled data sets. We refer the reader to [LRC*02] for a survey of approaches to multiresolution terrain modeling, and to [PG07] for a comprehensive survey of approaches based on regular nested triangle hierarchies.

The early work on multiresolution terrain models based on nested triangle hierarchies [LKR*96, DWS*97, Paj98] utilized a CPU-based algorithm for serialized fine-grained updates to the terrain. Meshes can be extracted in a bottom-up manner [LKR*96], by simplifying the mesh at full resolution; in a top-down manner, by refining a coarse base mesh [Paj98]; or incrementally, through refinements and simplifications on a previously extracted mesh [DWS*97].

Based on the observation that modern GPUs enable rendering of pixel-sized triangles, later works [Pom00, Lev02, CGG*03, HDJ05, BGP09] focus on batched updates to a coarse-grained multiresolution model. In this case, the multiresolution hierarchy consists of a collection of *macro* elements, each of which corresponds to a set of triangles at a finer resolution. This set can consist of a triangulation of a regularly [Pom00, BGP09], semi-regularly [Lev02] or irregularly [CGG*03] sampled dataset. These batched triangulations are typically computed during an offline preprocessing step, where highly optimized triangle strips can be generated for efficient transfer to the GPU. Recent approaches have focused on working directly with compressed data [GMC*06, DSW09, LC10] and on spreading the processing to multiple processors and displays [GMBP10].

Most parallel approaches utilize a *saturated* selection criterion [OR98, Paj98], i.e. one in which the error at a node is guaranteed to be greater than those of its descendants. This requires the selection criterion to be calculated offline and can limit the types of possible metrics. Saturated distance-dependent metrics have also been studied, where a *bounding* hierarchy is associated to each element [BAV98, Blo00, LP02, Ger03]. Only the nodes within the bounded region must be refined, while those outside this region can be safely culled.

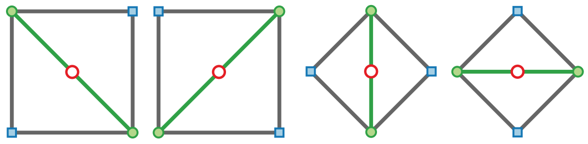
Ji et al. [JWLL05] present a GPU-resident LOD approach for rendering crack-free adaptive geometry images [GGH02] in the form of stitched triangulated restricted quadtrees [VHB87, SS92]. However, they achieve crack-free meshes through the use of a saturated error metric on the quadtree refinement.

We propose a fine-grained GPU-based approach implemented in OpenCL that evaluates each node at runtime and applies a parallel closure operation on the mesh elements to ensure the extraction of crack-free meshes. Thus, arbitrary predicates can be used for the selection criterion.

3. A hierarchy of diamonds

A mesh in which all elements are defined by the uniform subdivision of its elements into scaled copies is called a *nested mesh*. A special class of nested triangle meshes are those generated by the *Longest Edge Bisection (LEB)* operator, in which a triangle is bisected along the midpoint of its longest edge and the vertex opposite this edge [Riv84]. When this rule is applied recursively to the pair of triangles decomposing a square domain, this generates a *containment hierarchy* composed entirely of isosceles right triangles.

In many applications, such as terrain processing, we are interested in *conforming*, i.e. crack-free, meshes, since cracks in the mesh correspond to discontinuities in functions defined on the mesh vertices. However, bisections only generate conforming meshes when both triangles sharing a common longest edge bisect concurrently. This pair of triangles is referred to as a *diamond* [DWS*97], and the shared longest edge is referred to as its *spine* (see Figure 2).



(a) 0-diamond configurations (b) 1-diamond configurations

Figure 2: Four possible diamond configurations. The *spine* (green edge) of a 0-diamond (a) is a diagonal of a square while that of a 1-diamond (b) is an axis-aligned edge. The *central vertex* (hollow red circle) of a diamond is located at the midpoint of its spine, while those of its two *parents* are the two vertices not incident to the spine (blue).

A diamond is subdivided by bisecting both of its triangles. This adds a single vertex at the midpoint of its spine, which we refer to as its *central vertex*. Diamond subdivision defines a direct dependency relation on the diamonds, where a diamond δ_p is a *parent* of another diamond δ_c if δ_c contains a triangle generated during the bisection of a triangle in δ_p.

This dependency relation defines a multiresolution model, referred to as a *hierarchy of diamonds*, and can be modeled as a *Directed Acyclic Graph (DAG)* whose root is the diamond subdividing the square domain; whose nodes are the diamonds; and whose arcs encode the parent-child relationships. Each diamond δ has two parent diamonds, whose central vertices coincide with the two vertices of δ that are not incident to its spine, and it has four children diamonds, whose spines coincide with δ's boundary edges.

A diamond δ's *depth* is the length of a path from the root of the DAG to δ. Diamonds at an even depth have spines that are aligned with the diagonal of an axis-aligned square, and are referred to as *0-diamonds*, while those at an odd depth have spines that are aligned with an edge of such a square and are referred to as *1-diamonds*. The set of diamonds at successive depths within the hierarchy define a *level* of resolution. Each level of resolution can be tiled by *super-squares* [WD08, WD09], structured patterns of edges within the hierarchy (see Figure 3). The correspondence between diamonds and edges, via their spines, and between edges and vertices of the hierarchy, via their unique midpoints, enables efficient processing algorithms and encoding schemes for hierarchical subsets of diamonds, edges and vertices within the hierarchy. Due to the regularity of the subdivision scheme, all geometric and hierarchical relationships can be implicitly determined from the coordinates of a diamond's central vertex in terms of scaled offsets along directions separated by 45° angles [Paj98, HDJ05, WD08]. We refer the reader to [WD10] for more details on these hierarchical structures and their applications.

Diamond hierarchies can be used to extract conforming triangle meshes of uniform or variable resolution. Each such mesh uniquely corresponds to a *closed* cut of the DAG de-

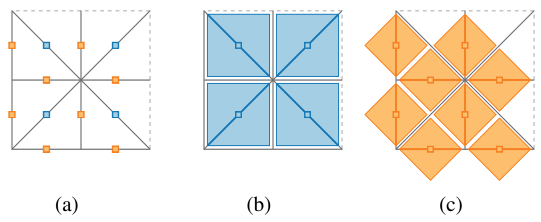


Figure 3: Super-squares are structured sets of edges tiling each level of resolution within a hierarchy of diamonds. Each super-square contains twelve edges (a), corresponding to four 0-diamonds (b) and eight 1-diamonds (c).

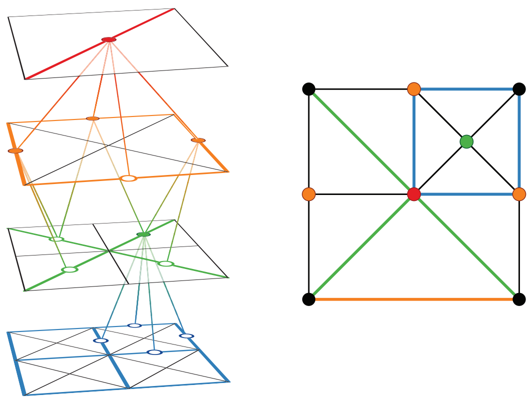


Figure 4: A conforming mesh (right) extracted from a hierarchy of diamonds corresponds to a *closed* cut of the DAG describing its dependency relation (left). Triangles in the mesh correspond to subdivided parents (filled circles) of unsubdivided diamonds (unfilled circles).

scribing the diamond dependency relation, separating subdivided and unsubdivided diamonds, i.e. if a diamond δ is subdivided, then both of its parents are as well. A diamond belongs to the *active front* of the cut if it is not subdivided, but at least one of its parents is. Each such subdivided parent contributes a single triangle to the mesh. Figure 4 illustrates a closed cut of the diamond dependency relation (left) and its corresponding extracted mesh (right).

When used as a multiresolution model for a terrain dataset, a diamond hierarchy can be implicitly encoded as an array of elevation values at the coordinates of a $(2^N+1)^2$ grid, where N is the maximum level of resolution. Typically, an *approximation error* is associated with each diamond in the hierarchy to indicate its level of approximation to the underlying elevation grid. Due to the correspondence between diamonds and grid points, via their central vertices, this *error field* can be encoded as an array of resolution $(2^N+1)^2$.

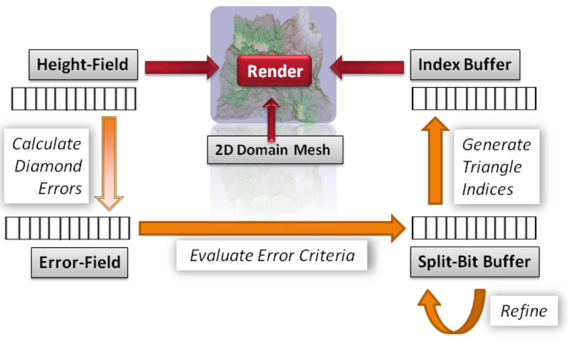


Figure 5: System overview

4. Overview

In this section, we provide an overview of our terrain processing framework (see Figure 5). All phases are run in parallel on the GPU. The CPU only needs to create the GPU buffers, to maintain the tasks run on the GPU, and to initialize the static data such as the height values and normal maps, if required. In a preprocessing phase, we generate the multiresolution model in parallel. Due to the implicit nature of the hierarchical and geometric relationships within the hierarchy of diamonds model, we only need to compute the approximation error associated with each diamond.

Hierarchy generation: This phase calculates per-diamond approximation errors for a given height field. The error values are stored in an *error field* buffer.

Mesh extraction is performed in three phases, each of which runs entirely on the GPU in a data-parallel fashion, and uses the output from previous phases. Although we demonstrate the workflow for an interactive terrain rendering application, these three stages are common to all multiresolution terrain processing workflows, such as visibility computation or morphological analysis.

Evaluation of the selection criterion: This phase tests the selection criterion against each diamond in the hierarchy. The result of each test indicates whether the diamond should be subdivided (i.e. if it fails the test). The output of this phase is written to a binary-valued buffer that we call the *split-bit* buffer.

Transitive closure: Since conforming meshes correspond to closed subsets of the diamond hierarchy with respect to the dependency relation, we apply a *closure* operation to the split-bit buffer. For each diamond δ with a split-bit equal to TRUE, we mark the split-bit of all of its ancestors as TRUE.

Generating triangle indices: This phase reads the split-bit buffer and generates triangles for the diamonds belonging to the active front. Specifically, when a diamond's split-bit is FALSE, a triangle is emitted for each of its parents whose split-bit is TRUE.

The selective refinement query is defined by a selection criterion that guides the mesh extraction. The latter typically incorporates the approximation error and can include other terms, such as a distance-based component. In addition to queries based on approximation error, we have implemented selection criteria for extracting meshes at uniform resolution as well as view-dependent queries, where the selection criterion depends on the distance from the viewpoint.

The triangulation module runs entirely on the GPU and processes a 2D *domain mesh* buffer, which provides the x- and z-coordinate for each vertex to the OpenGL shader, while the elevation value indexed at this location provides the y-coordinate. We use an *index buffer* to store the three indices for each generated triangle.

As illustrated in Figure 5, we use five GPU buffers in our system, all of which are stored in linear (1D) arrays. In the discussions below, the number of samples is identical to the number of diamonds.

Height field buffer: stores the elevation values, typically requiring 16-bits per sample.

Error field buffer: stores the approximation error of each diamond, requiring 16 bits per sample.

Split-bit buffer: stores a single bit per diamond. Each diamond uniquely corresponds to a single sample.

Index buffer: stores triangle mesh index data. We process diamonds using the super-square construct. There are 48 indices reserved for each super-square (as shown in Section 8), resulting in 4 indices per sample. Since the index data is encoded using 32-bit integers, this buffer requires 128 bits per sample.

2D domain mesh buffer: stores the 2D coordinates of the basic domain layout, and is used only for the rendering phase. Each point uses two 16-bit integers, resulting in an overhead of 32-bits per sample.

Our parallel kernels require no communication between threads other than atomic write operations, required in some phases for updating shared memory positions. The methods can be therefore considered as *embarrassingly parallel*, and the parallelism introduced does not add computational overhead with respect to a serial implementation of the algorithms.

Our implementation uses the OpenCL API for the parallel GPU algorithms and OpenGL for GPU rendering. However, this system can be easily extended to other parallel/multi-core architectures due to the low communication between threads and limited access to shared memory. The atomic operations are only used in the error generation phase, which we did not optimize, and in the transitive closure operations, where at most two threads can write to a shared memory location in a single pass.

5. Hierarchy generation

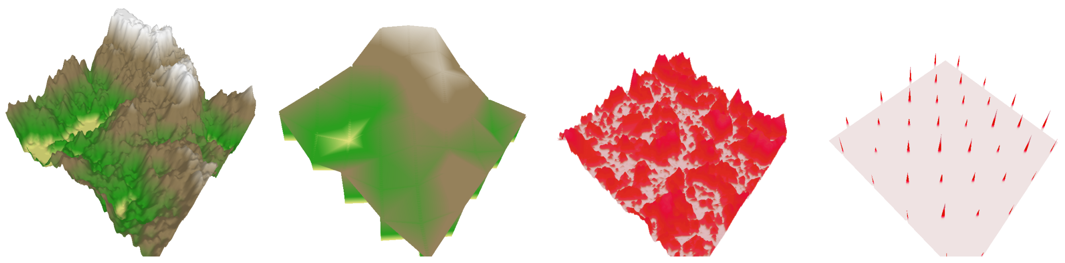
Due to the implicit nature of the hierarchy of diamonds representation, we only need to evaluate the approximation errors for the diamonds to generate the full multiresolution terrain representation.

The approximation error of a triangle t in the hierarchy, also referred to as the total error of t [LRC*02, WD10], is the maximum distance from t's plane in R^3 of a sample whose domain lies within that of t. The approximation error associated with a diamond δ is the maximum of the approximation errors of its two triangles. Since each diamond is in one to one correspondence with its central vertex, per-diamond errors can be stored in an additional *error field* buffer with the same resolution as the height field. This phase requires read access to the height field buffer and read-write access to the error field buffer.

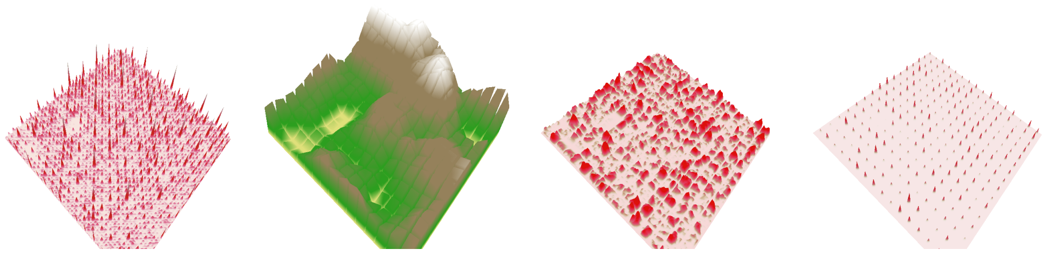
Our parallel algorithm computes the errors of all the diamonds at a given depth in a single pass. The entire process is applied to each depth of the hierarchy, thus, if the maximum level of resolution is N, this step requires $2 \cdot N$ passes. Figure 6 displays elevation and error values on a synthetic terrain dataset for selected depths (namely, $d = 5$ and $d = 8$).

In each pass, we process all samples in the domain. For each vertex v in a given pass at depth d, we first find its containing diamond δ at depth d (as described below). We then compute the interpolated height value $\overline{h}(v)$ obtained from linear interpolation of the triangle vertices (see Figures 6b and 6f). The interpolated values are used to generate the absolute interpolation error for each vertex in the domain (see Figures 6c and 6g). Finally, atomic MAX operations on the diamond's central vertex reduce these interpolation errors to the correct value. Thus, after completing a pass at depth d, the error values for all diamonds at depth d are computed and stored in their corresponding central vertex positions (see Figures 6d and 6h). As illustrated in Figure 6, the interpolation errors tend to decrease at greater depths in the hierarchy. Figure 6e shows the final error buffer values over the entire domain.

This approach requires efficient evaluation of the unique diamond containing a given vertex at a specified depth in the hierarchy. When processing 0-diamonds, the central vertex position can be determined by adding a diagonal offset (equal to half the diamond's edge length) to the lower-left corner of the diamond, which can be found by conceptually diving the domain into axis-aligned blocks of size diamond-edge-length, i.e. by using the MOD operator. Similarly, when processing 1-diamonds, the same logic can be applied after rotating the domain of 45 degrees, to align the diamonds with the coordinate axes (similar to the *quincunx* quadtrees of [Heb98, LP02]).

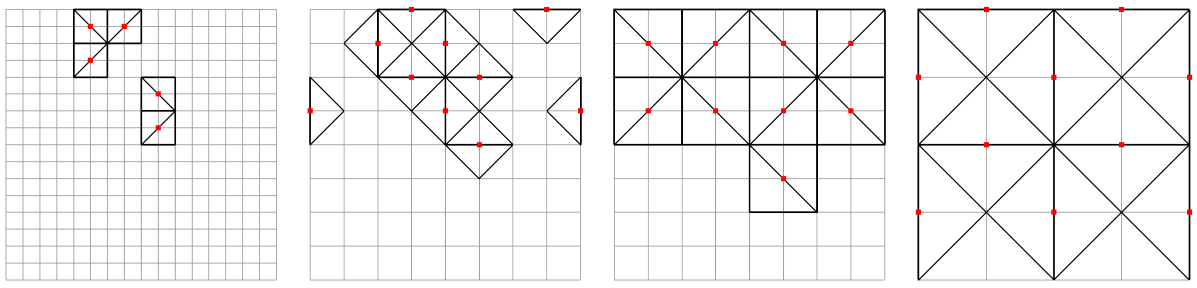


(a) Height field (b) Depth 5 - Interpolation (c) Depth 5 - Difference (d) Depth 5 - Diamond errors



(e) All - Diamond errors (f) Depth 8 - Interpolation (g) Depth 8 - Difference (h) Depth 8 - Diamond errors

Figure 6: The error metric (e) computation for depths $d = 5$ (b-d) and $d = 8$ (f-h) for the procedurally generated terrain dataset in (a).



(a) Depth=6 (b) Depth=5 (c) Depth=4 (d) Depth=3

Figure 7: Example diamonds that failing the selection criterion at descending depths within a diamond hierarchy.

6. Evaluation of selection criterion

Once the error field is computed, we initialize the diamond refinement by testing each diamond against the selection criterion and storing the result in the *split-bit* buffer. This phase is run in parallel over each super-square in the hierarchy and requires a single pass over the collection of diamonds in the hierarchy.

Our selection criterion can be an arbitrary predicate, and typically involves terms related to a diamond's approximation error as well as its distance to an object of interest. The former is evaluated at the granularity of super-squares and is independent of the viewpoint. For each super-square, its twelve diamond's error values are read from the error field and are compared against a given threshold error value. For the latter, we consider the scaled absolute distance of a diamond's central vertex from the viewpoint. In this context, failing the selection criterion means that the diamond requires subdivision, i.e. its split-bit should be set to TRUE. After the evaluation of all diamonds in the super-square, a single bit value for each of the twelve diamonds is written back to the split-bit buffer as a two byte block of memory. Figure 7 illustrates this algorithm over the diamonds from depths $d = 6$ to $d = 3$ on a small example dataset.

We also incorporate a geometry culling operation that removes diamonds behind the frustum's near plane, or outside a given view angle. Thus, significantly reducing the processing of invisible mesh elements in the remaining phases.

7. Transitive closure refinements

Recall from Section 3 that a conforming triangle mesh corresponds to a closed cut of the diamond dependency graph. However, after applying an arbitrary selection criterion to the diamonds in the hierarchy, the split-bit buffer will not generally correspond to a closed set of refinements.

Thus, we must ensure that the transitive closure of the set bits in the split-bit buffer is satisfied. That is, for each diamond δ whose split-bit is set, we propagate the split-bits up the hierarchy by setting the split-bits of all ancestor of δ.

As in the previous phase, we achieve parallelism through the granularity of the super-square construct. However, in this phase, we require a bottom-up traversal within each level of resolution of the hierarchy. In each pass, we read the split-bit fields of each super-square and update the parent bits accordingly. Each of the twelve diamond *types* within a super-square have different parent configurations of containing super-squares. The parent diamond of a diamond can reside in the same super-square; in neighboring super-squares at the same level or resolution; or, in a super-square at a lower resolution. However, since all super-squares are identical, there are only twelve such cases to consider.

We minimize the number of read-write accesses to neighboring split-bit buffers by initially caching these updates in local memory. After all such updates have completed, we update the neighboring super-square split-bits using atomic OR operations. With this approach, each bit needs to be evaluated at most twice since a super-square can update another neighboring super-square on the same level and the second pass in the same level ensures that these changes propagate to the next higher level of super-squares.

Figure 8 shows an example of the parallel transitive closure algorithm on an initial set of split-diamonds are some diamonds in depths $d = 7$ and $d = 6$ of the hierarchy. A composite diagram of the final set of diamonds that are split are shown in Figure 8g, and its conforming mesh generated by this set of subdivided diamonds is shown in Figure 8h. As noted in [WD08], the number of new diamonds that are split after this phase is bounded by a relatively small size.

8. Generation of the triangle mesh

The triangle mesh is generated in parallel by processing each split-bit in a single pass. Using super-squares as the unit of data parallelism, the maximum number of triangles that can be generated is reduced from 24 (i.e. two triangles for each of the twelve diamonds) to 16 since there is a maximum of eight split-bits that can be set after the closure operation.

In our current system, each triangle is indexed using three vertex indices, so the index buffer size contains 48 (i.e. 3×16) indices per super-square. Since the GPU discards degenerate triangles without further rendering, triangles that remain at zero index do not add much of a processing penalty other than the increased size of the buffer that is processed.

A triangle of a diamond δ belongs to the mesh when the δ's split-bit value is FALSE and the parent's split-bit is TRUE. Access to the split-bits of parent super-squares is handled similarly to the transitive closure operation (as described in Section 7), although there are two main differences. First, the parent values are not updated, so the split-bit access is read-only. Second, we need some additional bookkeeping to track the processed triangles. For example, we need the coordinates of the vertices of the currently processed triangle.

Interestingly, given the above method for generating the triangle indices from the split-bit buffer, we can analyze the refinement operation visually. When a diamond is not refined, some of the diamonds with FALSE split-bits which should have been set to TRUE to satisfy the transitive closure create additional triangles, which intersect existing triangles. By using the blending mode of the rendering pipeline we can detect these overlaps (as in Figure 9a, where overlaps appear as black regions) The rendering with refinement applied (Figure 9b) illustrates that the transitive closure process corresponds to a conforming triangulation covering the entire domain.

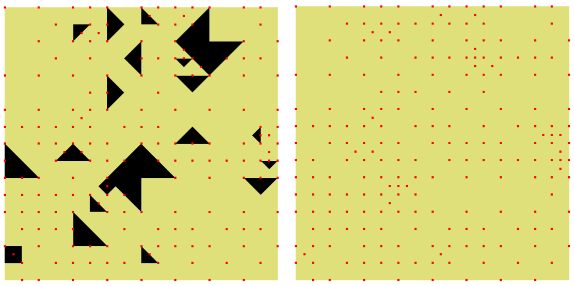
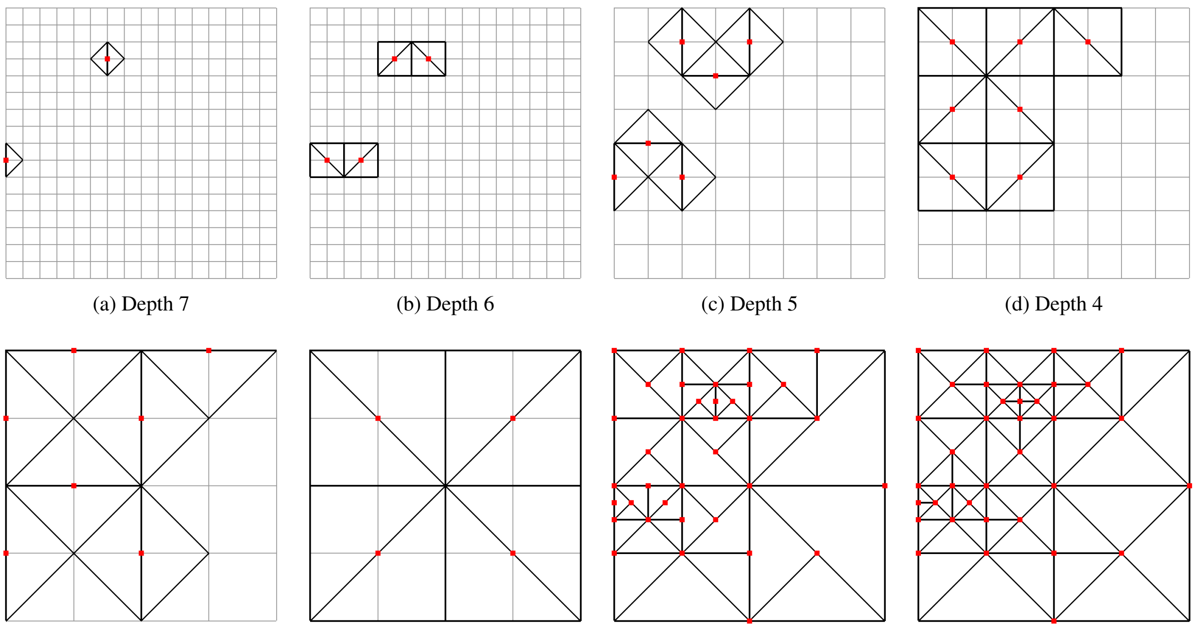

(a) Mesh from initial graph (b) Mesh from closed graph

Figure 9: Visualizing the triangulation associated with a split-bit buffer before (a) and after (b) performing the transitive closure operation. Black triangles in (a) correspond to regions with overlapping triangles, which are not present in the conforming mesh. The conforming mesh also has more subdivided diamonds (red dots).

9. Performance results

In this section we present timing results of our proposed methods works on commodity GPUs for each phase of the framework from our sample implementation. Our implementation uses OpenCL 1.0 for the parallel GPU algorithms and OpenGL 3.3 for rendering. The same data buffers are shared across different GPU APIs and access across the OpenCL and OpenGL APIs is synchronized using explicit `wait` commands. High resolution timings are measured using the OpenCL event object profiling capabilities. All ex-


(a) Depth 7 (b) Depth 6 (c) Depth 5 (d) Depth 4

(e) Depth 3 (f) Depth 2 (g) Split-Diamonds Overlayed (h) Conforming Mesh

Figure 8: Starting with a few deep diamonds ($d = 7$ (a) and $d = 6$ (b)), the refinement sets the split-bits of their ancestor diamonds at shallower levels (c-f). (h) The conforming mesh associated with these split diamonds.

periments are run on a test machine with a NVIDIA GeForce GTX 260 GPU (216 shader cores, 877 MB GDDR3), running Windows 7 OS (64-bit).

We present timing results in Table 1 for the various phases of our algorithm on a 513×513 procedurally generated terrain using libnoise library [Bev] and for the Puget Sound dataset [UT] in resolutions $1k \times 1k$, $2k \times 2k$ and $4k \times 4k$. The results are measured in milliseconds (ms), except for rendering, which is measured in overall frames-per-second (fps). Initialization, refinement and mesh index generation are considered as dynamic phases, since they need to be run each time we generate a new terrain mesh. The performance and timings are observed to be stable when the viewpoint flies over the terrain, avoiding the terrain boundaries, thus we only present representative timings for the general performance.

Phase		**513**	**1025**	**2049**	**4097**
Gen. Hier.	*Total time*	260	1,118	4,615	20,401
	Root-Depth	62.16	249.1	999.7	3,997
	Max-Depth	0.46	1.81	7.12	33.41
Evaluate	*View-Dep.*	0.07	0.19	0.63	12.39
Closure	*Max. Time*	2.27	6.02	20.41	76.1
	View-Dep.	0.90	1.17	1.82	3.41
Gen. Index	*View-Dep.*	0.17	0.52	2.16	51.72
Total time (View-Dep.)		1.14	1.89	4.60	67.52
Render (fps)	*Dynamic*	150	95	41	5
	Static	780	220	62	10

Table 1: Results in ms, except rendering row in FPS

9.1. Hierarchy generation

Although the generation of the approximation errors is likely to be only performed a single time for a given static terrain dataset, we found that this calculation can be significantly accelerated using our parallel diamond-based algorithm, which scaled over multiple processing cores efficiently, as shown in Table 1 under the *Generate Errors* heading. We report the total time to generate the entire array of errors over all depths as well as the timings for updating the root diamond and those at the maximum depth, which depends on the resolution of the dataset and ranged from 17 to 23, respectively. It can be observed that every doubling of the domain resolution increases the total processing time by approximately a factor of four, demonstrating linear growth. Updating the root diamond requires the most time, since all the writes are practically serialized to the same shared memory location using atomic updates. At higher depths, this bottleneck is reduced since fewer work items write to the same memory and we achieve significantly better parallelism.

9.2. Evaluation of selection criterion

For these experiments, we used a view-dependent error criterion, as described in Section 6. Since this phase is applied in a single pass to all diamonds in the hierarchy, it initializes the split-bit buffer in time that is linearly in the size of the terrain dataset, as also shown by the results in Table 1. Although the selected error criterion, including the frustum culling, requires many arithmetic operations for each diamond, this phase is the fastest of the three dynamic phases.

9.3. Transitive closure

We present two timings for the refinement phase in Table 1: the maximum observed time, using a hierarchy initialized to a fully-split state, and the timing of refinement on a partially-split hierarchy, initialized with view-dependent criterion.

We note that the performance of this phase is affected by the complexity of the split-bits set in the previous phase. As an optimization, we can skip super-squares whose split-bits disabled, since they cannot affect their parents. Thus, as the number of unsubdivided diamonds increases, this phase achieves better performance.

When using a view-dependent selection criterion, we expect that many diamonds in regions outside the viewpoint will nor require refinement, and thus the view-dependent metric better characterizes the expected performance for interactive terrain rendering. In general, the closure step is currently the slowest phase in our system.

9.4. Generation of triangle indices

Terrain mesh index generation requires a single pass over the whole split-bit buffer, and the timing presents the total time of this single pass. The implementation includes an optimization which detects super-squares in which all split-bits are set and discards them (unless they are at the lowest depth), since they cannot add triangles to the mesh. The result in Table 1 indicates that the mesh generation time also scales linearly with dataset size. After every doubling of domain edge size, this phase runs four times slower.

9.5. Overall performance in dynamic index updating

The overall performance is displayed in two ways in Table 1. The first focuses on the dynamic phases of our framework, while the second focuses on the use of our system in interactive terrain visualization. The *Total time (View-dep.)* row presents timing results of the three parallel phases for view-dependent mesh generation (i.e. all phases except rendering and hierarchy generation). The performance is expected to scale linearly by the number of samples in the dataset, and the results show some deviations from the expected theoretic performance. We note that in our proposed system, generation of a view-dependent mesh takes less than 5 milliseconds for terrains of size $2k \times 2k$, which contain four million samples in total.

In our rendering tests, we both render the whole terrain at the highest resolution as a single mesh (static), and render a new generated terrain mesh from the current view-point of the camera (dynamic). The results show the average frame rates observed. The performance of the dynamic phases is close to rendering performance, while still allowing interactive visualization of terrains and generating a new mesh for every frame. We note that a new mesh does not have to be generated each-frame, since the computations can be spread out over multiple frames by distributing the phases and data ranges. This will in turn generate a mesh with fewer triangles which has a lower vertex processing cost than a full resolution triangulation of terrain.

10. Concluding remarks

We have introduced parallel algorithms for generating, querying and rendering a multiresolution terrain model using an entirely GPU resident hierarchy of diamonds that exploits the data parallelism enabled through the super-squares primitive. We have demonstrated interactive querying and rendering of large datasets on commodity GPUs that require the CPU only for calling the GPU kernels.

A major advantage of our approach which we are planning to explore in our future work is its ability to apply arbitrary predicates to control the extraction of conforming meshes from the hierarchy. Thus this approach can be beneficial in common GIS applications including visibility computations, viewshed and horizon computations of data points at any intermediate (uniform or variable) resolution, or morphological analysis, such as computing the regions of influence of the critical point at different resolutions.

A limitation of the current approach is that it must process the entire multiresolution terrain dataset, since it considers terrains sampled at all the vertices of a regular grid. There are however applications in which elevation values are available only at a subset of the vertices of the grid, or that only a subset of vertices have a meaningful elevation (like in terrain data sets containing flat regions or planetary data sets with large areas corresponding to the ocean). For these applications, we are currently investigating the use of adaptive representations of the terrain dataset such as those presented in [WD08]. This can also be helpful in dynamically updating the dataset when new data becomes available.

Alternatively, our approach could be coupled with a batch updated triangulation structure to reduce the processing of lower levels of the hierarchy. Thus, each macro triangle generated during our refinement process can be replaced by a batch of triangles. This approach can help bridge the gap between the performance of our system and the state of the art batched update approaches [Pom00, HDJ05, BGP09, LKES09].

As future work, we are investigating the use of geometry shaders to emit triangles on a per-super-square basis. This can eliminate the index and domain mesh buffers from our pipeline, thereby reducing the memory requirements and mesh extraction times.

Acknowledgments

We would like to thank the reviewers for their many helpful suggestions. This work has been partially supported by the National Science Foundation under grant CCF-0541032.

References

[BAV98] Balmelli L., Ayer S., Vetterli M.: Efficient algorithms for embedded rendering of terrain models. In *Proceeding International Conference on Image Processing (ICIP)* (1998), vol. 2, pp. 914–918. 3

[Bev] Bevins J.: Libnoise. http://libnoise.sourceforge.net. 8

[BGP09] Bösch J., Goswami P., Pajarola R.: Raster: Simple and efficient terrain rendering on the GPU. In *EG 2009 - Areas Papers* (2009), Ebert D., Krueger J., (Eds.), Eurographics Association, pp. 35–42. 2, 3, 9

[Blo00] Blow J.: Terrain rendering at high levels of detail. In *Proceedings of the Game Developers Conference* (2000). 3

[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

[DSW09] Dick C., Schneider J., Westermann R.: Efficient geometry compression for GPU-based decoding in realtime terrain rendering. *Computer Graphics Forum 28*, 1 (2009), 67–83. 3

[DWS*97] Duchaineau M., Wolinsky M., Sigeti D. E., Miller M. C., Aldrich C., Mineev-Weinstein M. B.: ROAMing terrain: real-time optimally adapting meshes. In *Proceedings IEEE Visualization* (Phoenix, AZ, October 1997), Yagel R., Hagen H., (Eds.), IEEE Computer Society, pp. 81–88. 2, 3

[Ger03] Gerstner T.: *Top-Down View-Dependent Terrain Triangulation using the Octagon Metric*. Tech. rep., Institut für Angewandte Mathematik, University of Bonn, 2003. 3

[GGH02] Gu X., Gortler S., Hoppe H.: Geometry images. In *Proceedings ACM Siggraph* (2002), ACM Press, pp. 355–361. 3

[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. 3

[GMC*06] Gobbetti E., Marton F., Cignoni P., Di Benedetto M., Ganovelli F.: C-BDAM – Compressed Batched Dynamic Adaptive Meshes for terrain rendering. *Computer Graphics Forum 25*, 3 (2006), 333–342. 3

[HDJ05] Hwa L., Duchaineau M., Joy K.: Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 355–368. 2, 3, 9

[Heb98] Hebert D.: Cyclic interlaced quadtree algorithms for quincunx multiresolution. *Journal of Algorithms 27*, 1 (1998), 97–128. 5

[JWLL05] Ji J., Wu E., Li S., Liu X.: Dynamic LOD on GPU. In *Proceedings Computer Graphics International* (2005), IEEE Computer Society, pp. 108–114. 3

[LC10] Lindstrom P., Cohen J. D.: On-the-fly decompression and rendering of multiresolution terrain. In *Proceedings of ACM Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 65–73. 3

[Lev02] Levenberg J.: Fast view-dependent level-of-detail rendering using cached geometry. In *Proceedings IEEE Visualization* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 259–266. 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, 9

[LKR*96] Lindstrom P., Koller D., Ribarsky W., Hodges L. F., Faust N., Turner G. A.: Real-time continuous level of detail rendering of height fields. In *Proceedings ACM SIGGRAPH* (August 1996), pp. 109–118. 2

[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. 3, 5

[LRC*02] Luebke D., Reddy M., Cohen J., Varshney A., Watson B., Huebner R.: *Level of Detail for 3D Graphics*. Computer Graphics and Geometric Modeling. Morgan-Kaufmann, San Francisco, 2002. 2, 5

[OR98] Ohlberger M., Rumpf M.: Adaptive projection operators in multiresolution scientific visualization. *Visualization and Computer Graphics, IEEE Transactions on 4*, 4 (Oct-Dec 1998), 344–364. 3

[Paj98] Pajarola R.: Large scale terrain visualization using the restricted quadtree triangulation. In *Proceedings IEEE Visualization* (1998), Ebert D., Hagen H., Rushmeier H., (Eds.), IEEE Computer Society, pp. 19–26. 2, 3

[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

[Pom00] Pomeranz A.: *ROAM using surface triangle clusters (RUSTiC)*. Master's thesis, U.C. Davis, 2000. 2, 3, 9

[Riv84] Rivara M.: Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering 20*, 4 (1984), 745–756. 3

[SS92] Sivan R., Samet H.: Algorithms for constructing quadtree surface maps. In *Proc. 5th Int. Symposium on Spatial Data Handling* (1992), pp. 361–370. 3

[UT] U.S.G.S., The University of Washington: Puget sound terrain dataset. http://www.cc.gatech.edu/projects/large_models/ps.html. 8

[VHB87] Von Herzen B., Barr A. H.: Accurate triangulations of deformed, intersecting surfaces. In *Proceedings ACM SIGGRAPH* (New York, NY, USA, 1987), ACM, pp. 103–110. 3

[WD08] Weiss K., De Floriani L.: Sparse terrain pyramids. In *Proceedings ACM SIGSPATIAL GIS* (2008), pp. 115–124. 2, 3, 7, 9

[WD09] Weiss K., De Floriani L.: Supercubes: A high-level primitive for diamond hierarchies. *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Visualization 2009) 15*, 6 (November-December 2009), 1603–1610. 3

[WD10] Weiss K., De Floriani L.: Simplex and diamond hierarchies: Models and applications. In *EG 2010 - State of the Art Reports* (Norrköping, Sweden, 2010), Hauser H., Reinhard E., (Eds.), Eurographics Association, pp. 113–136. 3, 5