UCRL-JC-127870
PREPRINT

ROAMing Terrain (*R*eal-time *O*ptimally *A*dapting *M*eshes)

M. Duchaineau
M. Wolinsky
D.E. Sigeti
M.C. Miller
C. Aldrich
M. Mineev

This paper was prepared for submittal to the
Institute of Electrical and Electronics Engineers Visualization '97
Phoenix, AZ
October 19-23, 1997

July 1997

ROAMing Terrain: Real-time Optimally Adapting Meshes

Mark Duchaineau*,† Murray Wolinsky* David E. Sigeti*
Mark C. Miller† Charles Aldrich* Mark B. Mineev-Weinstein*

Los Alamos National Laboratory*
Lawrence Livermore National Laboratory†

Abstract

Terrain visualization is a difficult problem for applications requiring accurate images of large datasets at high frame rates, such as flight simulation and ground-based aircraft testing using synthetic sensor stimulation. On current graphics hardware, the problem is to maintain dynamic, view-dependent triangle meshes and texture maps that produce good images at the required frame rate. We present an algorithm for constructing triangle meshes that optimizes flexible view-dependent error metrics, produces guaranteed error bounds, achieves specified triangle counts directly, and uses frame-to-frame coherence to operate at high frame rates for thousands of triangles per frame.

Our method, dubbed Real-time Optimally Adapting Meshes (ROAM), uses two priority queues to drive split and merge operations that maintain continuous triangulations built from preprocessed bintree triangles. We introduce two additional performance optimizations: incremental triangle stripping and priority-computation deferral lists. ROAM execution time is proportionate to the number of triangle changes per frame, which is typically a few percent of the output mesh size, hence ROAM performance is insensitive to the resolution and extent of the input terrain. Dynamic terrain and simple vertex morphing are supported.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation - Viewing Algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Geometric Algorithms, Object Hierarchies; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Virtual Reality.

Additional Keywords: triangle bintree, view-dependent mesh, frame-to-frame coherence, greedy algorithms.

1 INTRODUCTION

A complete system to display views of large datasets at high frame rates consists of components to manage disk paging of geometry and texture, level-of-detail (LOD) selection for texture blocks, LOD for triangle geometry, culling to the view frustum, and triangle stripping. This paper focuses on the last three of these components, which deal with in-memory geometry management.

During interactive visualizations, many types of geometric data compete for limited polygon budgets. Terrain remains one of the most challenging types because it is not naturally decomposed into parts whose complexity can be adjusted independently, and because the qualities required of a triangulation are view dependent. Classic geometry LOD optimization algorithms, such as those of Clark [1] and Funkhouser and Séquin [7], are not immediately applicable to terrain because they require independently adjustable parts. Traditional triangulation optimizations [14, 2, 15, for example], do not apply directly to terrain visualization because they do not admit flexible view-dependent error objectives, and they are much too slow to be used for each frame.

All of the existing algorithms that can interactively perform view-dependent, locally-adaptive terrain meshing [12, 11, 17, 19, 10], including ours, rely on a pre-defined multiresolution terrain representation that is used to build the adaptive triangle mesh for a frame. An example frame generated by the ROAM implementation is depicted in Figure 1, both with and without visible mesh edges. Figure 2 shows a birds-eye view of the domain mesh. The mesh is typical for ROAM: neighborhoods that are flat or distant are triangulated more coarsely than close or rough neighborhoods.

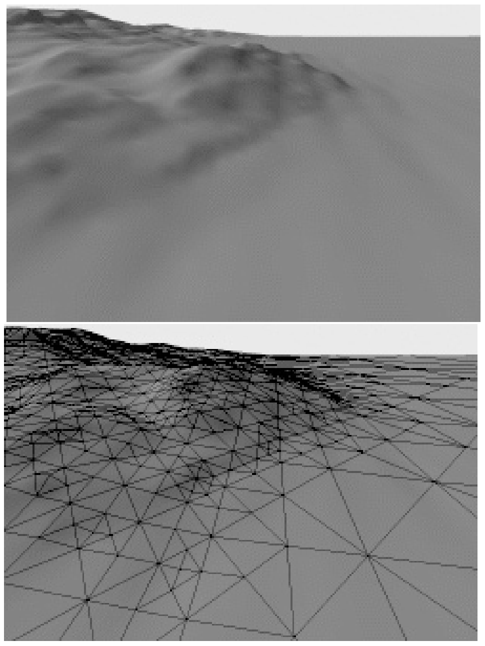
Figure 1: Example of ROAM terrain.

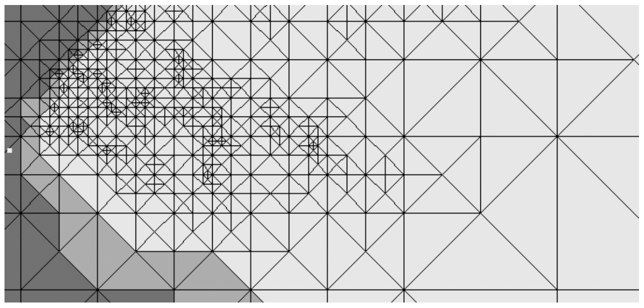
Figure 2: Triangulation for example frame, with eye looking right. Dark region is outside view frustum, light region is inside, and grey overlaps boundary.

*{murray,sigeti,cha,mariner}@lanl.gov, Los Alamos, NM
†{duchaineau1,miller86}@llnl.gov, Livermore, CA

ROAM consists of a preprocessing component and several runtime components. The preprocessing component produces nested, view-independent error bounds bottom-up for a triangle bintree. At runtime, four phases of computation are performed per frame:

1. recursive, incremental update to view-frustum culling
2. priority update only for output triangles that can potentially be split/merged in phase 3
3. triangulation update using greedy split and merge steps driven by two priority queues (for splits and merges, respectively)
4. as-needed updates for triangle strips affected by the culling changes from phase 1 and the splits/merges from phase 3

We evaluate our method with respect to twelve criteria of general application as follows:

1. **Time required to achieve a given triangle count:** Our implementation can maintain an optimized mesh containing thousands of triangles at 30 frames per second for high-speed, low-altitude flights over rough terrain. The algorithm's running time is proportionate to the number of triangle changes per frame, which is typically a few percent of the total mesh size. Hence ROAM performance is insensitive to terrain database extent and resolution (with the caveat that we must have all the data in memory in the current implementation).
2. **Flexibility in choosing view-dependent error metrics:** ROAM uses maximum geometric distortion in screen space as a base metric and queue priority value. This metric can be enhanced in many ways: ensuring correct visibility along specified lines of site, providing correct terrain positions under objects, and eliminating back-facing detail are examples.
3. **Mesh representations (both pre-defined and run-time selected):** Our method uses a triangle bintree for pre-computation, and builds continuous adaptive meshes by incremental, fine-grained split and merge steps. The triangles are always a right-isosceles shape, and therefore avoid the numerical problems of thin, "slivery" triangles.
4. **Simplicity of algorithms:** ROAM is simple to understand and implement because it is centered on split and merge operations of bintree triangles whose structure naturally avoids the complex case proliferation and legitimacy rules for selectively refining and coarsening within irregular or even quadtree-structured multiresolution spaces, and no special efforts are required to avoid discontinuities (*cracks*) and thin triangles. The greedy priority-queue technique that drives the splits and merges provides a simple mechanism for incorporating an extensible set of error-metric enhancements.
5. **Quality of mesh at a given triangle count:** Our algorithm produces optimal meshes in the sense of minimizing the maximum error bound for *monotonic* bounds (bounds that don't get larger after a split operation). Of course this optimum is achieved over our particular choice of mesh space, and larger or better spaces of triangle meshes certainly exist. Our results in actual tests are qualitatively excellent.
6. **Direct control of triangle counts:** ROAM produces meshes with a specified triangle count directly. Algorithms that admit only error tolerances can only indirectly control triangle counts by hunting for the corresponding error threshold, a process that is either slow or unreliable. ROAM can operate to achieve specified tolerances, but triangle-count specification is preferable for many applications.
7. **Strict frame rates:** ROAM can progressively optimize from frame to frame and stop when the frame time is about to expire (although typically only a fraction of the frame time is required). Our motivating application, aircraft sensor stimulation, requires a strict frame rate governed by the equipment under test. In conjunction with processor locking or real-time operating systems on multi-processor graphics systems, our implementation provides a strict frame rate at the highest triangle counts that the graphics hardware can handle reliably.
8. **Guaranteed error bounds:** ROAM produces guaranteed bounds on geometric screen-space distortions. These bounds are obtained locally by a fast conversion from preprocessed world-space bounds to view-dependent screen-space bounds, and globally by the greedy nature of the optimization algorithm.
9. **Memory requirements:** We distinguish between preprocess data size and runtime size. Our preprocess size is equal to the number of raw, finest-level height samples plus a single "thickness" value per bintree triangle. The runtime data structures take up space proportionate to the output mesh size, which is minuscule compared to the preprocess data size.
10. **Dynamic terrain:** Since our preprocessing is fast and localized, the preprocessed data can be updated quickly when the terrain is changed by mud slides, explosions, etc.
11. **Reduced "popping":** ROAM's use of a screen-distortion metric and its tendency to make very few mesh changes per frame naturally reduces disturbing temporal artifacts (*pops*). In some cases it is desirable to reduce pops further by *vertex morphing*, which is easily incorporated in ROAM.
12. **General input meshes:** Although our motivating application and implementation focuses on terrain (in the form of a height field), ROAM's mesh structure applies to manifolds of arbitrary genus with boundary. A drawback of the bintree meshes we use, and more generally any subdivision surfaces, is that irregular input meshes must be approximated, albeit as closely as one likes [4].

In addition to these general criteria, ROAM was influenced by the specialized requirements of our motivating application, synthetic sensor stimulation for aircraft undergoing ground-based testing with hardware and human controllers in the loop. In this setting it is critical to query a runtime terrain data server for line-of-site (LOS) visibility tests, object positions, and so on in addition to the texture and triangle-mesh queries used for display to multiple sensors with differing viewpoints and requirements for accuracy and frame rate. Because of the approximations made to achieve high frame rates, the issues of consistency and correctness for the various queries becomes paramount. In earlier approaches, LOS and other queries are made with respect to a single current terrain approximation. This at least gives consistency to the various query responses, but requires a single "lowest common denominator" triangle mesh suitable for all views and sensor frame rates, thus degrading quality to unacceptable levels. The philosophy in ROAM is to give responses to LOS and position queries that are correct with respect to the finest-level input data, and constrain each of the view-dependent meshes to preserve the selected LOS visibilities and positions. The ROAM architecture efficiently supports this philosophy, and thus ensures consistency through correctness.

2 RELATED WORK

A general treatment of multiresolution LOD control is presented by Heckbert and Garland [8], while more specific surveys and references can be found in [3, 11, 19]. Triangle stripping cost models and techniques are presented by Evans *et al.* [5]. Greedy algorithms have been applied to terrain triangulation in the non-realtime, non-view-dependent setting [6, 15, for example]. A general treatment of subdivision surfaces and conversions from irregular meshes is given by Eck *et al.* [4], while a specific application of wavelet analysis to terrain LOD is given by Staadt *et al.* [16]. In the remainder of this section we give more detailed overviews of recent published work most closely related to ROAM.

Miller [12] uses a quadtree to preprocess a height field defined on a uniform grid. In a pre-processing phase, vertices at each quadtree level are computed using an approximate least-squares fit to the level below. For each frame at run time, a priority queue drives quadtree refinement top-down from the root, thus allowing specified triangle counts to be achieved directly. The priority for a quadtree element is a heuristic involving view-independent (error in surface) and view-dependent (screen-area coverage) components aimed at minimizing the squared error in output image pixel intensities. No advantage is made of frame-to-frame coherence, and only one type of error metric is developed. A special effort is made to ensure continuous triangulations, but "T-vertices" are allowed, where a vertex exists on only one side of an edge. Methods for overlaying of point, linear and areal features are also discussed.

Lindstrom *et al.* [11] choose the same space of continuous triangle-bintree meshes as we do. They do not recognize the simple bintree structure nor the split and merge operations that we present, and so must take special care to maintain mesh continuity. They obtain high frame rates for large output meshes using a bottom-up vertex-reduction methodology enhanced by an elegant block-LOD-reduction algorithm. The block-LOD updates are incremental, taking advantage of frame-to-frame and object-space coherence. However, the fine-grained vertex deletion is performed bottom-up, which tends to limit performance compared to our fine-grained incremental mesh updates. Also, although they use the same base metric of geometric screen-space distortion, the block-based optimizations appear to make it difficult to enhance this metric. They provide no guaranteed error bounds (only level-to-level displacements are considered in the fine-grained LOD reduction). A tolerance parameter is used to control the algorithm, but the lack of guaranteed bounds precludes guaranteed success in attaining the requested accuracy. A triangle-count parameter is not considered, and would be difficult to incorporate with their block-LOD algorithm (this precludes support for strict frame rates with maximum triangle counts). Memory requirements are similar to ROAM, and dynamic terrain is supported. Vertex morphing is not supported, and only height maps are considered. A recursive triangle-stripping method is described, but requires corner-turning and does not take advantage of frame-to-frame coherence.

Willis *et al.* [17] describe a hierarchical triangulated-irregular-network (TIN) data structure with "near/far" annotations for vertex morphing, along with a queue-driven top-down refinement procedure for building the triangle mesh for a frame. No automatic procedure is given to build the precomputed TIN hierarchies and morphing annotations. No consideration is given to controlling errors or triangle counts. Specific effort is taken to avoid T-vertices. No advantage is taken of frame-to-frame coherence. Memory requirements are higher per preprocessed multiresolution element than in ROAM. The method applies to general base (coarsest-level) triangle meshes. The vertex-morphing capabilities are powerful and unique (we consider only simple split/merge animations).

Xia *et al.* and Hoppe [19, 10, 18] give similar methods for interactive, fine-grained LOD control of general TIN input meshes based on view-dependent refinement of preprocessed *progressive mesh* representations [9]. Both algorithms allow flexible view-dependent error metrics, and have demonstrated backface detail reduction based on nested Gauss-map normal bounds. Xia *et al.* use a base metric derived from the edge-collapse operations inherent to progressive meshes, which gives only a loose heuristic estimate of geometric or parametric screen-space distortions. Hoppe gives a metric that separates non-directional and normal-direction errors, and mentions the possibility of modifying this to measure errors in approximating non-linear texture-coordinate mappings. Progressive meshes are most naturally refined by undoing the edge-collapse operations in exactly the reverse order of the preprocess collapse sequence. Complex legitimacy rules are required to allow different refinement orders. Only tolerance parameters are given to the view-dependent refinement. In Xia *et al.*, special consideration is taken to avoid thin triangles. Hoppe uses a feedback mechanism to perform rough frame-rate regulation. Nested bounds are provided by analyzing neighborhoods defined by the binary vertex trees formed during the preprocessing. Although some consideration is given to frame-to-frame coherence, execution times are still proportional to the full output mesh size. As a performance enhancement, Hoppe considers traversing the active vertices once over the course of multiple frames, but indicates that this complicates frame-rate regulation. Progressive-mesh preprocessing is organized as a global optimization process, and thus is too slow to support dynamic terrain. Animation of the edge collapse/expand operations reduces popping similar to vertex morphing, but introduces degenerate thin triangles temporarily in the process.

3 OVERVIEW

In the remainder of this paper we describe the ROAM mesh representation, optimization algorithm, error metrics, performance enhancements and results. In this section, we give an overview of what follows.

More fundamental than the ROAM algorithm itself is the dynamic mesh representation based on triangle bintrees. In Section 4 we define triangle bintrees and describe the split and merge operations that are used to maintain continuous meshes while adding and removing one vertex at a time. Animation of the splits and merges is presented as a simple method for obtaining temporal continuity. Applicability to surfaces with general topology is considered.

Given these fundamentals of the mesh representation, Section 5 goes on to describe the dual-queue incremental optimization algorithm. A simplified version of the algorithm is given first that uses only a single "split" priority queue. We explain why this top-down algorithm produces optimal bintree-based triangulations for monotonic priorities. A second "merge" priority queue is added to enable incremental optimizations for time-varying priorities.

ROAM uses view-independent, preprocessed error bounds to facilitate the computation of view-dependent error bounds. In Section 6, the metrics, preprocessing and view-dependent conversions are given. In world space, we rely on nested bounds using a single number per bintree triangle to define a "pie-wedge" bound shape. These *wedgies* are readily used to compute per-triangle bounds on screen-space distortions in the projected "up" direction. We also use wedgies to facilitate computations like view-frustum intersections and LOS tests. We consider various enhancements to the basic screen-space geometric distortion metric.

For ROAM to operate at high frame rates, several performance enhancements are needed. These are described in Section 7: incremental view-frustum culling, incremental T-stripping, deferring priority recomputations, and progressive optimization. Results from experiments with our implementation are given in Section 8. These show the effect of the performance enhancements as well as the overall behavior of the algorithm.

4 MESH REPRESENTATION

4.1 Triangle Bintree

Just as the square-shaped quadtree has a triangle-quadtree counterpart, the familiar rectangle-shaped bintree [13] has a little-known triangle-shaped counterpart. Figure 3 shows the first few levels of a triangle bintree. The root triangle, $T = (v_a, v_0, v_1)$, is defined to be a right-isosceles triangle at the coarsest *level* of subdivision, $\ell = 0$. At the next-finest level, $\ell = 1$, the children of the root are defined by splitting the root along an edge formed from its *apex vertex* v_a

to the midpoint v_c of its *base edge* (v_0, v_1). The *left* child of T is $T_0 = (v_c, v_a, v_0)$, while the *right* child of T is $T_1 = (v_c, v_1, v_a)$. The rest of the triangle bintree is defined by recursively repeating this splitting process.

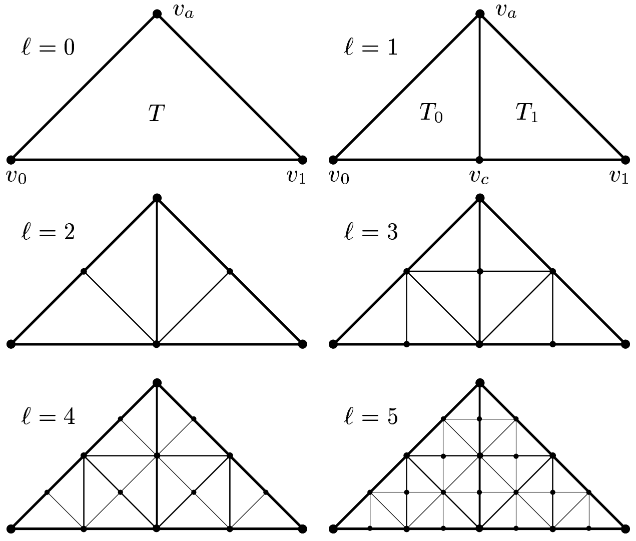

Figure 3: Levels 0–5 of a triangle bintree.

4.2 Dynamic Continuous Triangulations

Meshes in world space are formed by assigning world-space positions $w(v)$ to each bintree vertex. A set of bintree triangles forms a continuous mesh when any two triangles either overlap nowhere, at a common vertex, or at a common edge. We refer to such continuous meshes as *bintree triangulations* or simply *triangulations*. Figure 4 shows a typical neighborhood about a triangle T within a triangulation. We define T_B to be the *base neighbor* sharing base edge (v_0, v_1), T_L to be the *left neighbor* sharing left edge (v_a, v_0), and T_R to be the *right neighbor* sharing right edge (v_1, v_a).

A key fact about bintree triangulations is that neighbors are either from the same bintree level ℓ as T, or from the next finer level $\ell+1$ for left and right neighbors, or from the next coarser level $\ell - 1$ for base neighbors. All of these possible relationships are depicted amongst the triangles in Figure 4.

When T and T_B are both from the same level ℓ, we refer to the pair (T, T_B) as a *diamond*. A simple *split* operation and its inverse, *merge*, are depicted in Figure 4 for a triangulation containing a diamond. Split replaces triangle T with its children (T_0, T_1), and triangle T_B by its children (T_{B0}, T_{B1}). This split operation introduces one new vertex v_c at the diamond center, resulting in a new, continuous triangulation. If triangle T does not have a base neighbor T_B, only T is split into its children. Merging can be applied to diamond (T, T_B) when the children of T and T_B (if T_B exists) are all in the triangulation. In this case, we say (T, T_B) is a *mergeable diamond* for the triangulation.

An important fact about the split and merge operations is that any triangulation may be obtained from any other triangulation by a sequence of splits and merges.

Splits and merges can be animated using *vertex morphing* to provide a simple form of temporal continuity. For a time interval $t \in [0, 1]$, consider the split of diamond (T, T_B) shown in Figure 4. Instead of moving v_c immediately to its new position $w_c = w(v_c)$, let it move linearly over time from the unsplit base-edge midpoint $w_m = (w(v_0) + w(v_1))/2$ as $w_a(t) = (1 - t)w_m + tw_c$. Merges can be animated in a similar way.

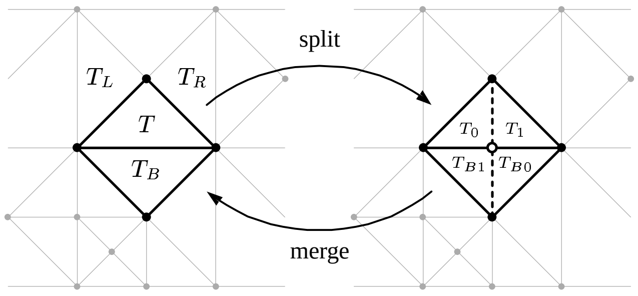

Figure 4: Split and merge operations on a bintree triangulation. A typical neighborhood is shown for triangle T on the left.

A triangle T in a triangulation cannot be split immediately when its base neighbor T_B is from a coarser level. To force T to be split, T_B must be forced to split first, which may require further splits in a recursive sequence. A case requiring a total of four splits is depicted in Figure 5. Such *forced splits* are needed for the optimization algorithm in Section 5.

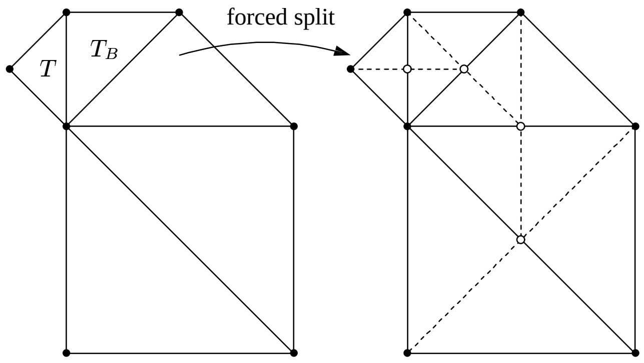

Figure 5: Forced splitting of triangle T.

Base meshes of more than one triangle can be used to represent manifold surfaces of arbitrary genus with boundary. If the base mesh can be tiled with diamonds, then the split and merge operations may be used freely as in the case with a single root triangle. For terrain, a typical base mesh is a single diamond.

5 DUAL-QUEUE OPTIMIZATION

The split and merge operations provide a flexible framework for making fine-grained updates to a triangulation. No special efforts are needed to avoid cracks or T-vertices. This section presents a greedy algorithm that will drive the split and merge process. The idea is simple: keep priorities for every triangle in the triangulation, starting with the base triangulation, and repeatedly do a forced split of the highest-priority triangle. As shown next, this process creates a sequence of triangulations that minimize the maximum priority (which is typically an error bound) at every step. The only requirement to ensure this optimality is that priorities should be *monotonic*, meaning a child's priority is not larger than its parent's. Adding a second priority queue—for mergeable diamonds—allows the greedy algorithm to start from a previous optimal triangulation when the priorities have changed, and thus take advantage of frame-to-frame coherence.

5.1 Split Queue

Suppose that every bintree triangle T is given a monotonic priority $p(T) \in [0,1]$. As triangulation $\mathbf{T}$ is built top-down, we shall maintain a priority queue $\mathcal{Q}_s$ containing all of the current triangles in $\mathbf{T}$. The top-down greedy algorithm is the following:

```
Let $\mathbf{T} =$ the base triangulation.
For all $T \in \mathbf{T}$, insert $T$ into $\mathcal{Q}_s$.
While $\mathbf{T}$ is too small or inaccurate {
    Identify highest-priority $T$ in $\mathcal{Q}_s$.
    Force-split $T$.
    Update split queue as follows: {
        Remove $T$ and other split triangles from $Q_s$.
        Add any new triangles in $\mathbf{T}$ to $Q_s$.
    }
}
```

This greedy algorithm produces optimal triangulations at every step. Consider any other triangulation $\mathbf{T}'$ that has a lower maximum priority than $\mathbf{T}$. Clearly $\mathbf{T}'$ must contain only descendents of all the triangles that were forced to be split while building $\mathbf{T}$. Because the force-split operation makes the minimum necessary refinements to preserve continuity, $\mathbf{T}'$ can not contain any ancestors to the triangles in $\mathbf{T}$. Finally, because $\mathbf{T}'$ has a lower priority, it must contain only descendents of at least one triangle in $\mathbf{T}$. Therefore, $\mathbf{T}'$ has a higher triangle count than $\mathbf{T}$ and so $\mathbf{T}$ is optimal. The total number of splits and merges performed by the top-down algorithm is proportionate to N, the number of triangles in the final triangulation $\mathbf{T}$.

5.2 Merge Queue

Now suppose that we are given time-varying priorities $p_f(T) \in [0,1]$ for frames $f \in (0,1,\ldots)$, and the problem is to build optimal triangulations $(\mathbf{T_0}, \mathbf{T_1}, \ldots)$. If these priorities are changing slowly and smoothly, then the optimal triangulations for any two consecutive frames will tend to be similar to one another. In this case, performance will be enhanced if we use triangulation $\mathbf{T_{f-1}}$ as a starting point to build triangulation $\mathbf{T_f}$. This is done by maintaining a second priority queue, $\mathcal{Q}_m$, that contains all the mergeable diamonds for the current triangulation. The priority for a mergeable diamond (T, T_B) is set to the maximum of its two triangles' priorities, $\max\{p_f(T), p_f(T_B)\}$. The incremental greedy algorithm is the following:

```
If $f = 0$ {
    Let $\mathbf{T} =$ the base triangulation.
    Clear $Q_s$, $Q_m$.
    Compute priorities for $\mathbf{T}$'s triangles and diamonds, then
        insert into $Q_s$ and $Q_m$, respectively.
} otherwise {
    Continue processing $\mathbf{T} = \mathbf{T_{f-1}}$.
    Update priorities for all elements of $Q_s$, $Q_m$.
}
While $\mathbf{T}$ is not the target size/accuracy, or the maximum split
    priority is greater than the minimum merge priority {
    If $\mathbf{T}$ is too large or accurate {
        Identify lowest-priority $(T, T_B)$ in $\mathcal{Q}_m$.
        Merge $(T, T_B)$.
        Update queues as follows: {
            Remove all merged children from $Q_s$.
            Add merge parents $T$, $T_B$ to $Q_s$.
            Remove $(T, T_B)$ from $\mathcal{Q}_m$.
            Add all newly-mergeable diamonds to $\mathcal{Q}_m$.
        }
    } otherwise {
        Identify highest-priority $T$ in $\mathcal{Q}_s$.
        Force-split $T$.
        Update queues as follows: {
            Remove $T$ and other split triangles from $Q_s$.
            Add any new triangles in $\mathbf{T}$ to $Q_s$.
            Remove from $Q_m$ any diamonds whose chil-
                dren were split.
            Add all newly-mergeable diamonds to $\mathcal{Q}_m$.
        }
    }
}
Set $\mathbf{T_f} = \mathbf{T}$.
```

The incremental greedy algorithm produces an optimal mesh $\mathbf{T_f}$ that has the same priority as if the top-down algorithm had been performed on the base mesh. The incremental algorithm does not generally produce optimal meshes during the intermediate steps (for example, T_{f-1} is usually not optimal for frame f), but it does reach optimality using the smallest possible number of split/merge operations applied to $\mathbf{T_{f-1}}$. The total number of splits and merges performed is proportionate to ΔN, defined as the number of triangles from $\mathbf{T_f}$ and $\mathbf{T_{f-1}}$ that are not in common. In the worst case, ΔN can be $N_{f-1} + N_f$. Situations like this are easily detected: there will be a large number of triangles and diamonds whose priorities are between the minimum merge priority and the maximum split priority. The remedy in this case is to fall back on the top-down algorithm, which can be accomplished by initializing $\mathbf{T}$, $\mathcal{Q}_s$ and $\mathcal{Q}_m$ as though $f = 0$.

6 ERROR METRICS

This section describes the various error metrics and bounds that are used to compute queue priorities. In the remainder of this paper we restrict our attention to height maps. Specifically, we assume that the vertex-to-world-space mapping $w(v)$ is of the form $w(v) = (v_x, v_y, z(v))$, where (v_x, v_y) are the domain coordinates of the vertex v, and $z(v)$ is the height at v. We denote the affine height map for a bintree triangle T to be $z_T(x,y)$. We also assume that camera coordinate systems and perspective transforms are given for each frame. We will continue to use the triangle-neighborhood notation from Section 4.

6.1 Nested World-Space Bounds

For height-map triangulations, a convenient bound per triangle T is a *wedgie*, defined as the volume of world space containing points (x,y,z) such that $(x,y) \in T$ and $|z - z_T(x,y)| \le e_T$ for some wedgie *thickness* $e_T \ge 0$. We refer to the line segment from $(x,y,z-e_T)$ to $(x,y,z+e_T)$ as the *thickness segment* for v. Nested wedgie bounds are built bottom-up, starting with $e_T = 0$ for all T at finest level $\ell_{\max}$. The wedgie thickness e_T for a parent triangle is defined in terms of its children's wedgie thicknesses, e_{T_0} and e_{T_1}. The tightest nested wedgie bounds are given by the formula

$$e_T = \max\{e_{T_0}, e_{T_1}\} + |z(v_c) - z_T(v_c)| \qquad (1)$$

where $z_T(v_c) = (z(v_0) + z(v_1))/2$. Note that this computation is fast and localized, which facilitates dynamic terrain. A univariate example of nested wedgies is illustrated in Figure 6, along with the chain of wedgies that depend on a particular vertex v.

6.2 Geometric Screen Distortion

With textured triangulations it is natural to separate geometric screen-space distortions from color distortions. We assume in this paper that colors for a surface point are accurately represented by

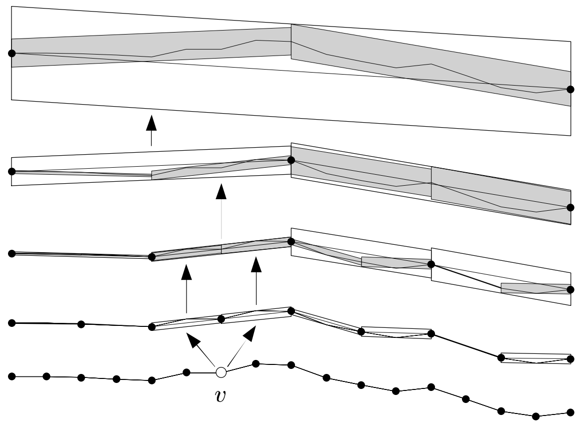

Figure 6: Nested wedgies for 1-D domain with dependents of v.

the texture. The remaining image errors can be represented as purely geometric distortions: the distance between where each surface point should be in screen space and where the triangulation places the point. Over the whole image we measure the maximum of these pointwise distortions. This is the base metric for ROAM.

Formally, let $s(v)$ be the correct screen-space position for a domain point v, and $s_T(v)$ be the approximate position from triangulation T. We define the pointwise geometric distortion at v to be $\mathbf{dist}(v) = \|s(v) - s_T(v)\|_2$. For the whole image, we define the maximum distortion to be $\mathbf{dist}_{\max} = \max_{v \in V} \mathbf{dist}(v)$ where V is the set of domain points v whose world-space positions $w(v)$ are within the view frustum.

In practice an upper bound is computed for the maximum distortion. For each triangle T in the triangulation, a local upper bound on distortion is obtained by projecting T's wedgie into screen space, as shown in Figure 7. The bound is defined as the maximum length of the projected thickness segments over all $v \in T$. These local bounds are monotonic, and will be used to form queue priorities. The maximum split-queue priority provides an upper bound on maximum distortion. If a wedgie extends behind the near clipping plane, the triangle's priority is set to an artificial maximum value and the distortion-bound computation is skipped.

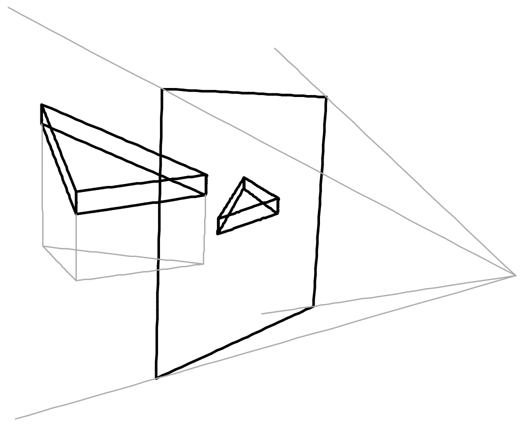

Figure 7: Distortion bound by projecting wedgie to screen space.

Because of the peculiarities of the perspective transform, the maximum projected wedgie thickness does not always occur at one of the triangle vertices. This leads to the following upper-bound computation. Let (p, q, r) be the camera-space coordinates of a point $w(v)$ without perspective, and without loss of generality assume the perspective projection is of the form $s = (p/r, q/r)$. The screen-space distortion at $v \in T$ is bounded by projecting the thickness segment at v. Let (a, b, c) be the camera-space vector corresponding to world-space thickness vector $(0, 0, e_T)$. The screen-space distortion at v is bounded by

$$\widehat{\mathbf{dist}}(v) = \left\| \frac{p+a}{r+c} - \frac{p-a}{r-c}, \frac{q+b}{r+c} - \frac{q-b}{r-c} \right\|_2 \qquad (2)$$

This can be rewritten as

$$\widehat{\mathbf{dist}}(v) = \frac{2}{r^2 - c^2} \left((ar - cp)^2 + (br - cq)^2 \right)^{1/2} \qquad (3)$$

It is straightforward to show that the minimum of $r^2 - c^2$ and the maximum of $(ar - cp)^2 + (br - cq)^2$ occur at the corner vertices of T (although not generally the same corner). An upper bound on $\widehat{\mathbf{dist}}(v)$ can thus be obtained by substituting these minimum and maximum values into Equation 3.

6.3 Line-of-site Correction

So far, the queue priority was derived solely from the screen-distortion bound for a triangle T. This priority can be modified to ensure that selected lines of site are correctly occluded or not. A simple method to do this is to change the priorities for any triangles whose wedgies intersect the LOS. By setting these priorities to an artificial maximum value, splits will be made preferentially in a manner sufficient to ensure correct visibility along the LOS. This method tends to promote more triangle splits than necessary, although this excess is typically small in practice compared to the overall triangle count. An example of LOS correction is given in Section 8.1.

6.4 Other Metrics

We briefly mention other possible metric/priority variations that are compatible with ROAM:

Backface detail reduction: Using nested Gauss-map normal bounds (as in [18, 10]), priorities can be set to minimum for triangles whose subtree of triangles are all back-facing.

Normal distortions: For specular highlights determined by normal vectors interpolated from the vertices, priority should be given to triangles having large normal distortions where the highlights occur.

Texture-coordinate distortion: For curved mappings from surface domain to texture coordinates, priority should be added proportionate to the size of the screen-space position distortions associated with the texture-coordinate approximation.

Silhouette edges: Specific emphasis can be placed on triangles whose normal bounds indicate potential back-face to front-face transitions.

View frustum: Wedgies outside the six clipping planes can be given minimum priority, as discussed in Section 7.1.

Atmospheric obscurance: Wedgie priorities can be decreased when fog reduces visibility.

Object positioning: To correctly position objects on terrain, the priorities of triangles under each object can be artificially increased.

Clearly this list can be extended to suit a variety of applications.

7 PERFORMANCE ENHANCEMENTS

In this section we describe enhancements that enable the ROAM algorithm to operate at high frame rates for triangulations consisting of thousands of triangles. The first three optimizations decrease the respective computation times for their subtasks by more than a factor of ten. The fourth ensures strict frame rates.

7.1 View-Frustum Culling

We assume the view frustum is defined as the intersection of six halfspaces. Each triangle in the bintree (down to the current triangulation) is given an IN flag for each of the six halfspaces, and an overall label of OUT, ALL-IN or DONT-KNOW, defined as follows: IN is set when the wedgie is entirely inside the halfspace, OUT is given when the wedgie is entirely outside at least one halfspace, ALL-IN is given if all IN flags are set, and DONT-KNOW is given if not OUT or ALL-IN.

Updating these flags and labels from frame-to-frame is efficiently handled by a recursive bintree traversal. If a triangle T was labeled OUT or ALL-IN for the previous frame, and these labels are correct for the current frame, then the subtree for T does not need to be updated and recursion terminates. Otherwise, T inherits its IN flags from its parent and rechecks its wedgie against the halfspaces not marked IN, setting new IN flags if appropriate. If the wedgie is entirely outside any of these halfspaces, T and all its children are marked OUT. If all IN flags are set, T and all its children are marked ALL-IN. Otherwise T is marked DONT-KNOW and recursion continues to its children.

7.2 Incremental T-Stripping

Significant performance gains result from organizing triangles into *strips*, although optimum stripping is a difficult problem [5]. We consider only non-generalized strips (no "vertex swapping"). We use a simple, sub-optimal, incremental approach that yields average strip lengths of around four to five triangles. As triangles are split, merged or change view-culling status, minimal re-linking of strips is performed. Deleting a triangle from a strip causes the strip to be deleted (for a singleton strip), shortened on the end, or split in two. New triangles are first inserted as singleton strips, which we subsequently attempt to glue to each of the neighboring strip ends.

7.3 Deferring Priority Recomputation

The screen-distortion priorities of the triangles change as the viewing position changes, typically in a slow and smooth manner. Recalculating priorities of all triangles for every frame is too costly. Instead, priorities are recomputed only when they potentially affect a split/merge decision.

Given a velocity bound on the viewpoint, bounds can be obtained for screen-distortion priorities over time (i.e. a time-dependent bound). Also, the crossover priority (defined as the maximum split-queue priority when the incremental split/merge process is complete) changes slowly from frame to frame (typically around 1% change). Recomputation of a triangle can safely be deferred until its priority bound overlaps the crossover priority. A deferral list is kept for each of the next few dozen frames. Only the triangles on the current frame's deferral list must have priorities recomputed. If time allows, additional triangles may be recomputed in subsequent deferral lists. After recomputation, the triangle is placed on the deferral list farthest ahead that will provide safe recomputation scheduling.

7.4 Progressive Optimization

To ensure strict frame rates, triangulation optimization should stop when the alloted frame time is about to expire. The ROAM algorithm readily supports this because optimization processing and stripping updates occur one split/merge at a time. Of course, early termination yields non-optimal triangulations. However, because the split/merge steps are performed in decreasing order of importance, the partial work done is optimal in the sense that we have gotten as close to true optimal as time permits while maintaining the specified triangle count. We refer to this stepwise processing as *progressive optimization*. Priority recomputations can also be limited based on time available. The only phase of ROAM not amenable to progressive completion is view-frustum culling, which fortunately requires only a small fraction of the frame time and is completed before priority recomputation and mesh optimization.

8 RESULTS

Performance figures were measured on an Indigo2 Silicon Graphics workstation with Maximum Impact graphics hardware and a single R10000 processor. These figures were obtained simulating a fighter aircraft in terrain avoidance mode flying at high speed over very hilly terrain[1]. With all incremental features of the algorithm turned on and 3000 triangles rendered the total time per frame is approximately 30 milliseconds. Of this, 5 milliseconds is spent doing the view-frustum culling, 5 milliseconds calculating the queue priorities, 5 milliseconds splitting/merging the triangles in the mesh and 15 milliseconds outputting the triangle strips.

Turning off the priority recomputation deferral increases the time calculating the queue priorities to about 43 milliseconds resulting in a total frame time of 68 milliseconds. Turning off incremental stripping further increases the frame time to 140 milliseconds. Finally, turning off incremental split/merge optimization increases the time to 210 milliseconds.

On a single R10000 processor Silicon Graphics Onyx with an Infinite Reality graphics board, performance improves so that 6000 triangles can be rendered at 30 frames per second. The subjective quality of the terrain geometry is very good. In a high-speed terrain-following scenario over rough terrain with 3000 terrain triangles in the mesh, silhouettes appear realistically complex and stable and there are virtually no popping artifacts. In a similar flight over less rough but still mountainous terrain 1500 triangles suffice to eliminate perceptible popping. Since the system is capable of producing meshes with 6000 triangles or more at 30 frames per second, ROAM appears to completely eliminate perceptible popping artifacts for even the most stringent flight regimes.

To understand this qualitative assessment better, we measured "pop" sizes and average number of splits and merges for the same high-speed, low-altitude loop over very rough terrain. A histogram of pop sizes in pixels (for a 1000×1000 image with 3000 triangles per frame), is shown in Figure 8. On average, only 43.2 splits and merges are performed per frame. In other words, less than 3% of the triangles change, and these few, scattered pops almost all measure under 1.5mm on a typical 21 inch workstation monitor.

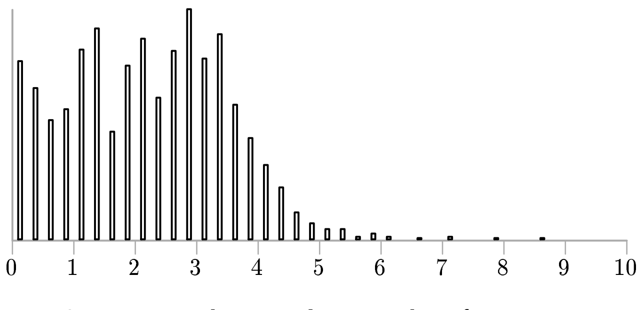

Figure 8: Histogram showing relative number of pops versus pop size (in pixels) totaled over the full test flight.

[1] Specifically, we use the USGS 1-degree DEM data (90 meter post spacing) for Northern New Mexico. We test on a circular flight 300 meters above the Jemez mountains, with trajectory smoothing, and camera facing in the direction of flight.

8.1 Line-of-site Example

An example of LOS visibility correction is shown in Figure 9. In this example, 8 triangles were “stolen” from the surrounding terrain to provide the correct occlusion along the line of site to the centermost box.

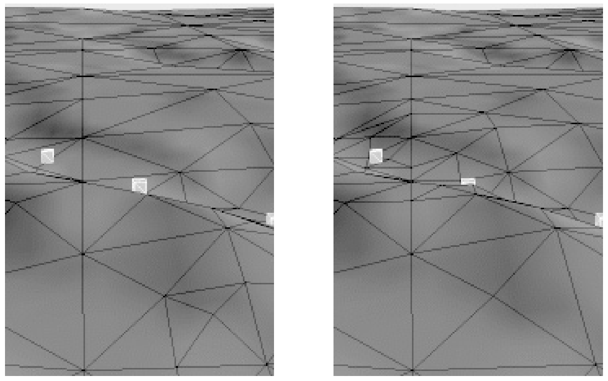

Figure 9: Left side is before LOS correction, right is after.

9 CONCLUSION

We have presented ROAM, an algorithm for optimizing triangulations given view-dependent error metrics. The method has been implemented and provides high-quality triangulations with thousands of triangles at high frame rates. A novel dual-queue incremental optimization process was combined with fast, localized preprocessing and several runtime performance enhancements, including incremental view-frustum culling, incremental stripping, priority-computation deferral and progressive optimization.

Critical future issues include management of geometry- and texture-loading from disk, and optimization of texture-block LOD.

Acknowledgments

This work was performed on U.S. Department of Energy Contract W7405-ENG-36, funded jointly by the U.S. Navy and Air Force. We thank James Reus for his assistance in the early stages of this work. We also thank our sponsors, who have provided substantial input and feedback during these investigations: in particular, we thank Kevin Hannon, Frank Griffo, Ted Wilson, Lenart Clark, Don Jackson, Dr. Shah Mahmood, David Purdue, Tom Joyner, Pamela Stewart and David Blake from the Air Combat Environment Test and Evaluation Facility at the Naval Air Warfare Center, Patuxent, River, MD; and Keem Thiem and Dr. John Fong from the Electronic Combat Integrated Test Program at the Air Force Flight Test Center, Edwards Air Force Base, CA. We also thank Kenneth Lee, Robert Webster, Ronald Pistone, Lynn Cline and Fatima Woody at Los Alamos National Laboratory for their valuable assistance. Finally we thank the staffs of the Advanced Computing Laboratory at Los Alamos National Laboratory and the Visualization Laboratory at Lawrence Livermore National Laboratory for sharing their expertise and for the generous use of their equipment.

References

[1] James H. Clark. Hierarchical geometric models for visible surface algorithms. *CACM*, 19(10):547–554, October 1976.

[2] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *SIGGRAPH ’96 Proc.*, pages 119–128, August 1996.

[3] Daniel Cohen-Or and Yishay Levanoni. Temporal continuity of levels of detail in delaunay triangulated terrain. In *Proc. Visualization ’96*, pages 37–42. IEEE Comput. Soc. Press, 1996.

[4] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH ’95 Proc.*, pages 173–182, August 1995.

[5] Francine Evans, Steven Skiena, and Amitabh Varshney. Optimizing tirangle strips for fast rendering. In *Proc. Visualization ’96*, pages 319–326. IEEE Comput. Soc. Press, 1996.

[6] Robert J. Fowler and James J. Little. Automatic extraction of irregular network digital terrain models. *Computer Graphics (SIGGRAPH ’79 Proc.)*, 13(2):199–207, August 1979.

[7] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (SIGGRAPH ’93 Proc.)*, pages 247–254, 1993.

[8] Paul S. Heckbert and Michael Garland. Multiresolution modeling for fast rendering. In *Proc. Graphics Interface ’94*, pages 43–50, Banff, Canada, May 1994. Canadian Inf. Proc. Soc.

[9] Hugues Hoppe. Progressive meshes. In *SIGGRAPH ’96 Proc.*, pages 99–108, August 1996.

[10] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH ’97 Proc.*, August 1997.

[11] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH ’96 Proc.*, pages 109–118, August 1996.

[12] Mark C. Miller. *Multiscale Compression of Digital Terrain Data to meet Real Time Rendering Rate Constraints*. PhD thesis, University of California, Davis, 1995.

[13] Hanan Samet. *Applications of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[14] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH ’92 Proc.)*, 26(2):65–70, July 1992.

[15] Cláudio T. Silva, Joseph S. B. Mitchell, and Arie E. Kaufman. Automatic generation of triangular irregular networks using greedy cuts. In *Proc. Visualization ’95*. IEEE Comput. Soc. Press, 1995.

[16] Oliver Staadt, Markus Gross, and R. Gatti. Fast multiresolution surface meshing. In *Proc. Visualization ’95*, pages 135–142. IEEE Comput. Soc. Press, July 1995.

[17] Lee R. Willis, Michael T. Jones, and Jenny Zhao. A method for continuous adaptive terrain. In *Proc. IMAGE VII Conference*, June 1996.

[18] J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Trans. on Visualization and Computer Graphics*, 3(2), 1997.

[19] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *Proc. Visualization ’96*, pages 327–334. IEEE Comput. Soc. Press, 1996.