Sparse Terrain Pyramids

Kenneth Weiss
Department of Computer Science
University of Maryland
College Park, Maryland 20742, USA
kweiss@cs.umd.edu

Leila De Floriani
Department of Computer Science
University of Genova
16146 Genova (Italy)
deflo@disi.unige.it

ABSTRACT

Bintrees based on longest edge bisection and hierarchies of diamonds are popular multiresolution techniques on regularly sampled terrain datasets. In this work, we consider *sparse terrain pyramids* as a compact multiresolution representation for terrain datasets whose samples are a subset of those lying on a regular grid. While previous diamond-based approaches can efficiently represent meshes built on a complete grid of resolution $(2^k + 1)^2$, this is not suitable when the field values are uniform in large areas or simply non-existent. We explore properties of diamonds to simplify an encoding of the implicit dependency relationship between diamonds. Additionally, we introduce a diamond clustering technique to further reduce the geometric and topological overhead of such representations. We demonstrate the coherence of our clustering technique as well as the compactness of our representation.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Hierarchy and geometric transformations*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Graphics data structures and data types*

General Terms

Multiresolution Terrain Models, Nested Triangle Meshes, Longest Edge Bisection, Diamond Hierarchies

1. INTRODUCTION

Digital terrains are generally defined by elevation values given at a discrete set of points distributed within a 2D domain. Since the sizes of current datasets can exceed one billion points [3, 12], a multiresolution approach is needed in order to allow extracting representations of the terrain at a level of detail which can be uniform or variable over the domain and based on a much smaller number of data points.

Triangulated Irregular Networks (TINs) are a common representation for terrains which explicitly encode the vertices of each triangle along with the field attributes. Since there are few restrictions on the locations of vertices within the domain, TINs allow a high degree of adaptability to features within the terrain, such as critical points and lines by using a minimum number of samples. However, this representation has a *geometric overhead*, due to the explicit storage of the coordinates, and a *topological overhead*, due to the explicit representation of the connectivity among the vertices.

Conversely, regularly sampled terrains have no geometric or topological overhead since the coordinates can be directly inferred from the position of the data points in a grid but suffer from a high *sampling overhead*, that is, the overhead related to the number of samples required to represent the features within the dataset, due to the rigid structure of such grids.

Quadtree representations are a typical compromise between the two representations since their vertices lie along a regular grid while allowing the representation to adapt to the features of the dataset. The major drawbacks with using quadtrees is that they produce discontinuous representations of the terrain, that is, surfaces with cracks. One way to solve the problem is to use restricted quadtrees, i.e., quadtrees in which neighboring blocks can differ by at most one level and for which the blocks are suitably triangulated [6, 21].

This has led to the development of multiresolution terrain models for points on a regular grid obtained through *Longest Edge Bisection (LEB)*, which generates a nested triangle mesh consisting of right triangles. Such a mesh is known as a *Hierarchy of Right Triangles (HRT)* and is obtained by subdividing a square domain into two right triangles along one of its diagonals. Each triangle in the mesh is generated by bisecting the right angle of its parent triangle. LEB meshes in fact have a higher degree of adaptability to the features of a domain than restricted quadtrees, since LEB subdivisions double rather than quadruple the elements of a cell. Furthermore, they have a higher representational power than quadtrees, i.e., the set of triangulations derivable from LEB subdivisions is a superset of those derivable from restricted quadtrees [4]. Compared with multiresolution terrain models based on irregular TINs, like the Multi-Triangulation (MT), they can be encoded implicitly and thus have smaller storage costs [5].

LEB meshes have been used mainly for terrain visualization in games, flight simulators, fly-throughs on terrains, etc. In our work, we focus on the use of such meshes for representing terrains in a Geographic Information System (GIS). This implies the need for updating the representation efficiently and for representing sparse regular grids, that is, situations in which the points are a subset of those contained in a regular grid.

However, current representations for LEB meshes in the form of a triangle bintree, or a hierarchy of diamonds (i.e., pairs of triangles sharing their longest edge and which need to be subdivided at the same time) are only suitable for LEB meshes built on complete regular grids, i.e., regular grids in which data are present at all

ACM GIS '08, November 5-7, 2008. Irvine, CA, USA.

the vertices. When modeling diamonds on a terrain where the complete hierarchy of diamonds is present, the diamonds are associated with the grid points and the mesh topology is implicitly encoded, thus resulting in an array representation of the height and the error values at the vertices of the grid. An issue here is to provide a representation when the field values are uniform in large areas, or simply not available. In this case, we will have a nested mesh in which the leaves have different sizes.

We have developed a representation for an LEB mesh based on an incomplete grid, that we call a *Sparse Terrain Pyramid (STP)*. In an STP, the mesh topology remains implicit in the model, but the set of diamonds in the mesh needs to be represented, incurring geometric overhead relating to the spatial location of each diamond. Whereas the height component of a vertex requires only a single value, its coordinates require two values, thus causing naive representations of such sparse diamond sets to require significantly more storage. Thus, a method to reduce this geometric overhead can greatly optimize such a representation.

To this aim, we introduce *super-squares*, a novel clustering structure over a hierarchical portion of the domain that implicitly indexes the diamonds contained within it. The geometric overhead incurred by representing the diamonds contained within a particular super-square is largely reduced. To the best of our knowledge, our STP approach is the first pointerless representation for sparse LEB meshes that provides random access to the mesh elements and efficiently supports general selective refinement queries.

The remainder of this paper is organized as follows. We review related work in Section 2. In Section 3, we discuss the diamond data structure and in Section 4, we discuss full and sparse terrain pyramids. In Section 5, we introduce our compact encoding and implicit clustering of spatially coherent diamonds. In Section 6, we introduce our diamond-based multiresolution representation for STPs. We then discuss our encoding of the extracted meshes in Section 7. Section 8 introduces an efficient technique to combine terrain datasets at different resolutions. In Section 9 we present experimental results and comparisons. Finally, we draw some concluding remarks in Section 10.

2. RELATED WORK

Multiresolution techniques based on LEB over regular terrains are popular as they allow better adaptability than quadtree representations and do not suffer from the presence of cracks in the meshes. All triangles in an LEB mesh have a single longest edge which is subdivided during a refinement operation. Crack-free meshes are guaranteed by allowing triangles to subdivide only when both triangles along the longest edge are the same size. Thus, refinements either require efficient neighbor finding operations on the triangles of the mesh, or a representation that directly encodes these corresponding pairs of triangles, also known as *diamonds*. The former techniques are typically represented as a pair of binary trees (bintrees) [7, 8, 9, 10, 2, 3] while, in the latter, the dependency relationship between diamonds can be encoded as a directed acyclic graph (DAG) [14, 15, 12]. Variable-resolution meshes are then extracted from the hierarchy through a process known as selective refinement, where a user-defined error metric guides the extraction of meshes from the model. In this work, we encode diamonds using a DAG whose arcs are determined from the implicit dependency relationship between diamonds. Recent surveys of these semi-regular multiresolution models can be found in [16, 19].

Lindstrom et al. [14] introduce a bottom-up terrain simplification algorithm over regular grids through triangle fusion. They store large terrains out-of-core as rectangular blocks and use the dependency relationship between triangles to enforce conforming mesh updates. Duchaineau et al. [7] introduce a view-dependent top-down incremental selective refinement algorithm for LEB meshes. Frame-to-frame coherence is supported through the use of a dual queue system, where one queue holds mergeable triangles and the other holds refineable triangles. Evans et al. [8] present an efficient neighbor finding algorithm for LEB meshes using *location codes*, where triangles are encoded by their path through the binary tree. However, their approach has some inefficiencies in that they store an error value for each triangle rather than for each refinement vertex and that they use explicit pointers to children nodes.

Pajarola [18] applies the dependency relationship of [14] to create a crack-free restricted quadtree triangulation from quadtree subdivisions of a terrain. He introduces a top-down refinement as well as a bottom-up simplification algorithm over such datasets, and extracts the level and direction of a quadtree node directly from the coordinates of its center. Lindstrom and Pascucci [15] provide an out-of-core indexing scheme for diamond vertices based on a hierarchical Π-order space filling curve, which enables efficient cache-coherent access to the samples. Their error metric includes an approximation error as well as a nested bounding sphere radius for distance-dependent refinement (as first introduced by Blow [1]). In this scheme, the full 3D coordinates as well as the approximation error and bounding sphere radius of each point are explicitly encoded as 32 bit floating points numbers, requiring a total of 20 bytes per vertex. Recently, Gerstner [10] introduced an implicit optimally tight octagon-shaped bounding hierarchy for distance-dependent rendering based on a diamond's hierarchical neighbors. In this work, we introduce an implicit encoding of a diamond's *type* through the coordinates of its central vertex which allows us to find diamond vertices, parents, children and neighbors in $O(1)$ time. This encoding is similar to that of Pajarola [18] while allowing us to extract more information. Furthermore, we cluster related diamonds to exploit the spatial and hierarchical coherence among samples, thus enabling a compact encoding of sparse sets of samples.

Many optimizations have been developed to enable interactive rendering of large LEB meshes. Out-of-core paging systems enable interactive rendering of gigantic datasets (currently defined as having on the order of a billion or more points). Although block-based paging coupled with LRU cache replacement [18, 12] is the most common technique, empirical results presented in [15] indicate that block-based paging has worse performance (by an order of magnitude) than simple array based (row major) ordering, and that a hierarchical level based ordering (quadtree or Π-order) performs one or two orders of magnitude better. View-dependent refinement based on windowing [18], screen space error [14, 7] or nested bounding spheres [1, 15, 2, 3, 9] are typically used in conjunction with view-frustum culling to prioritize rendering to visible triangles. Triangle stripping is another optimization that better utilizes the graphics hardware [7, 18, 9].

More recently, an LEB subdivision model has been used as a spatial access structure for clusters of updates. Although this reduces the adaptability of such meshes, clustered updates take better advantage of the graphics hardware architecture. In [2, 3], the triangles of the bintrees store an index to a list of TIN triangles, which is generated in a fine-to-coarse preprocessing step from the original dataset via edge-contraction operations. Textured color maps are applied to the terrain using a corresponding tiled quadtree. Hwa et al. [12] present a similar technique using a DAG of diamonds to access clusters of updates whose vertices lie along the regular grid. The novelty of this method is to apply the diamond hierarchy to the height values as well as textured color maps. Since these techniques are relevant for interactive rendering of diamond hierarchies, they

can be incorporated into our STP approach. However, since many of these rendering optimizations require duplication of mesh geometry or rearranging the geometry into formats that make it difficult to access individual elements, they are not applicable to terrain processing algorithms such as visibility analysis, terrain morphology and dynamic updates to the terrain which are the focus of our approach. As such, we will not focus on interactive view-dependent rendering in this paper.

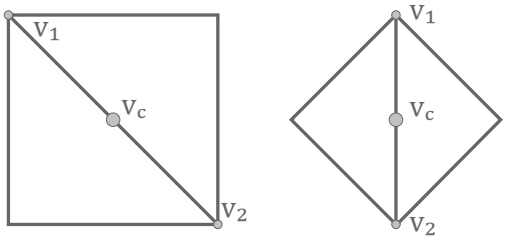
All the above methods are suitable only for representing an LEB mesh built from a regular grid where all the vertices are present, and thus the representations proposed there are either a full containment hierarchy or a complete hierarchy of diamonds, implicitly or explicitly encoded. Consequently, many of the above techniques exploit the regularity of the dataset to reduce the geometric and topological overhead of the multiresolution model. Since all vertices in a $(2^k + 1)^2$ terrain are present, they can be stored linearly and accessed using simple array notation or a more complicated indexing scheme [15, 20]. Furthermore, the implicit dependency relation can be used to locate parents, children and neighboring triangles, enabling pointerless representations. However, when samples are redundant, as in large bodies of water, or simply unavailable, this representation is no longer efficient.

To the best of our knowledge, the only method to encode an incomplete HRT generated through longest edge bisection has been proposed by Gerstner [9]. He introduces a compression method based on a linearization of the domain using a Sierpinski space-filling curve associated with the complete hierarchy of triangles. He uses a containment hierarchy as a multiresolution representation, and encodes in each node of the resulting partial tree the number of nodes that need to be skipped if the node is not to be refined. Variable-size relative pointers are used to indicate the number of bytes to skip if a node is not refined, and an average overhead of 3 bytes per vertex is reported. However, since each node is accessed via pointers stored in its parents, this representation does not provide random access to the data in the model. Gerstner [9] also introduces the notion of incorporating higher resolution blocks of samples into a coarse dataset. Although few details are given regarding how to accomplish this, he indicates that this requires the addition of some interpolated samples to align the new data with the hierarchy.

3. HIERARCHIES OF DIAMONDS

A mesh in which all cells are defined by the uniform subdivision of a cell into scaled copies of it is called a *nested mesh*. A special class of nested meshes are those generated by bisecting isosceles right triangles along their hypotenuse, and are denoted as *Longest Edge Bisection (LEB)* meshes. The *bisection rule* for a triangle **t** in such a mesh consists of replacing **t** with the two triangles obtained by splitting **t** along the line defined by the middle point of its longest edge **e** and the vertex of **t** opposite to **e**. When this rule is applied recursively to an initial decomposition of a square domain into two triangles sharing a diagonal of the square it generates two congruent classes of triangles, each with a single longest edge. We denote the *class* of triangles congruent to those sharing a diagonal of the base square as 0-*triangles*, and the triangles congruent to the ones obtained by splitting a 0-triangle as 1-*triangles*. The triangles obtained by splitting 1-triangles are congruent to 0-triangles. Observe that all triangle edges in these meshes are aligned with either the diagonal of an axis aligned square or an edge of such a square.

Given an LEB mesh Σ, the pair of triangles sharing a common longest edge form a *diamond*. We denote the longest edge as the *spine* of the diamond with *spine vertices* $\mathbf{v}_1$ and $\mathbf{v}_2$. The midpoint $\mathbf{v}_c$ of the spine is called the *central vertex* of the diamond. Note that a diamond can be uniquely identified by its spine or, equivalently, by its central vertex. Since each triangle has a single longest edge, both triangles in a diamond belong to the same class of triangles. As a consequence, there are two different congruence *classes* of diamonds: those with spines aligned with square diagonals (*0-diamonds*, see Figure 1(a)), or with square sides (*1-diamonds*, see Figure 1(b)). The domain of a diamond is thus an axis-aligned square (0-diamond) or a square that is rotated 45 degrees (1-diamond).



(a) 0-Diamond (b) 1-Diamond

Figure 1: The two classes of diamonds. In both cases, the *spine* is the internal edge with extreme vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ and the *central vertex* $\mathbf{v}_c$ is located at the midpoint of the spine.

Given a square domain containing regularly sampled terrain data, let us consider the collection T of all the triangles generated from the initial subdivision of the square domain through the longest edge bisection rule. If we consider the collection Δ of all diamonds associated with the longest edges of the triangles in T, the containment relation between the triangles in T induces a parent-child dependency relation over set Δ. A diamond δ' is a *parent* of another diamond δ if and only if some triangle in δ is created by the splitting of a triangle in δ'. If δ' is a parent of δ, then δ is called a *child* of δ'. For a diamond δ, we denote the set of its children diamonds as *Children*(δ), and the set of its parent diamonds as *Parents*(δ). Note that for any diamond δ, *Children*(δ) consist of the diamonds whose spines coincide with the four external edges of δ, and *Parents*(δ) consists of the two diamonds whose central vertices coincide with the right-angled vertices of δ. Since non-spine edges are always smaller than spine edges (by a factor of $\sqrt{2}$), the transitive closure of the parent-child relation defines a partial order on Δ which can be described using a DAG.

4. TERRAIN PYRAMIDS

Given an LEB mesh Σ generated by using all the vertices V of a terrain data set, we call the set Δ of all diamonds, whose central vertices have coordinates at the points of V, along with their parent-child dependency relation a *Full Terrain Pyramid* (FTP). Thus, Δ includes all points of V except the four corner vertices of the domain. The *root diamond* is the diamond with vertices at the corners of the domain. Those diamonds whose central vertices belong to the domain boundary are called *boundary* diamonds and all other diamonds are *non-boundary* diamonds. Since the central vertex of all 0-diamonds lies at the center of a square, 0-diamonds are never boundary diamonds.

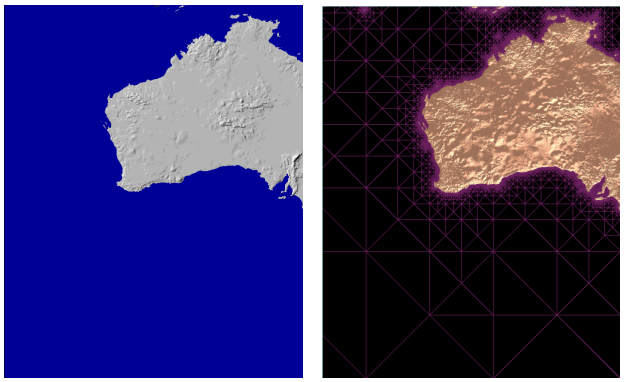
The *depth* of a diamond δ within a terrain pyramid is the length of a path from the root diamond to δ in the DAG, i.e. the number of subdivisions required to obtain δ. The *level* of an i-diamond δ_i is the number of i-diamonds above δ_i along any path to the root diamond and is denoted as $\text{LEVEL}(\delta_i)$. Since the class of diamonds cycles with every two subdivisions, the level of a diamond is its depth divided by 2, and its class is its depth modulo 2. If

the dimensions of the grid are $(2^k + 1)^2$, then there are k levels in the hierarchy, and k is the MAXLEVEL of the hierarchy. Finally, the *scale* of a diamond δ is defined in terms of its level as $\text{SCALE}(\delta) = \text{MAXLEVEL} - \text{LEVEL}(\delta)$.

An FTP built on a grid of dimensions $m = (2^k + 1)^2$ can be implicitly encoded by just storing the height and error values at the vertices of the grid. For each vertex **v**, the error associated with **v** is the error computed for approximating the original scalar field in the diamond δ having **v** as its central vertex. Specifically, in a preprocessing step, we assign to **v** the maximum of the interpolation error of all samples in the domain of its associated diamond δ. Since vertices of a diamond δ at level ℓ (and depth $d = 2\ell + i$) cannot contribute to the interpolation error, while points along an edge of δ belong to at most two diamonds at depth d and internal points of δ belong only to δ at depth d, the error evaluation requires $O(k \cdot m)$ operations, where k is the maximum level of the hierarchy and m is the number of points in the hierarchy. The parent-child relation in a full terrain pyramid is implicit, since it can be inferred directly from the encoding of a diamond's central vertex (see Section 5.2).

A *Sparse Terrain Pyramid (STP)* consists of a subset Δ_∂ of the diamonds in the full hierarchy Δ subject to the constraint that if a diamond δ belongs to Δ_∂, then Δ_∂ contains all diamonds which are predecessors (ancestors) of δ in Δ_∂ with respect to the parent-child relation. These are all the diamonds which belong to a path from the root diamond to δ in the DAG.

Sparse terrain pyramids are important, for example, when not all the data in a volume data set are available and instead of having a full grid, we have the data points at the vertices of a quadtree subdivision of the domain, or when the portion of data of interest is small compared to the full dataset, as, for instance, when only certain portions of the dataset are available at a higher resolution. Furthermore, when the terrain is locally oversampled, e.g., samples covering a large body of water, we can accurately interpolate these values from samples at a higher resolution. Figure 2 shows a zero approximation error sparse representation of a 6000×4800 sample tile from the gtopo30 dataset [22]. For this dataset, an STP requires less than $1/6$ of the original samples since flat regions do not need to be subdivided to the highest resolution to obtain an accurate approximation.



(a) GTOPO30 Tile (b) Sparse Pyramid

Figure 2: Terrains covering large flat regions such as oceans are oversampled by a regular grid (a). A zero error sparse representation of this terrain (b) requires less than $1/6$ of the samples from the original dataset. Image (a) courtesy of USGS [22].

The main challenge in representing such sparse datasets is to efficiently encode the coordinates of its diamonds. Whereas the coordinates of diamonds in an FTP can be implicitly determined such a representation is impractical for an STP, where much of the data is non-existent or redundant. However, explicit representations for the diamonds can be inefficient as well since they require an encoding of the spatial location of each vertex. By exploiting the spatial coherence between the diamonds in a partial hierarchy, we can achieve a compromise in terms of the explicit and implicit storage of such a representation.

5. COMPACT ENCODING OF DIAMONDS

The ingredients of a hierarchy of diamonds are the diamonds and the parent-child dependency relation. For full terrain pyramids, where the central vertices of diamonds can be determined from the representation (e.g. using array notation) this creates an entirely implicit representation. However, sparse terrain pyramids require more explicit schemes. We thus begin by introducing *super-squares* as a method of clustering spatially coherent diamonds. This significantly reduces the geometric overhead of an STP representation. We then introduce a method to determine all geometric and hierarchical elements of a diamond entirely from the binary representation of its central vertex.

5.1 Clustering Diamonds with Super-Squares

Grouping pairs of triangles into diamonds is a powerful abstraction that enables the efficient extraction of conforming meshes from an LEB mesh. We introduce a further abstraction within the hierarchy of diamonds representation, the *super-square*, which enables a compact representation for sparse terrain pyramids. A super-square is a highly symmetric structured set of edges within the hierarchy that tile each level of resolution of the LEB mesh. Figure 3 shows the repeating structure of super-squares at two different scales.

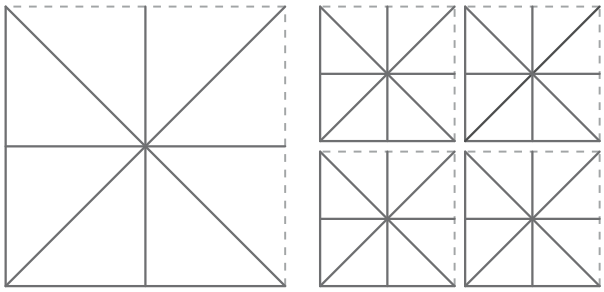
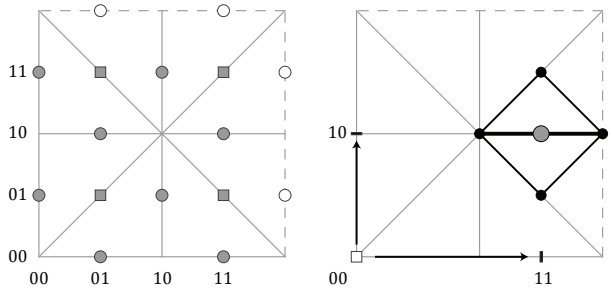


Figure 3: Super-squares are sets of edges that tile each level of resolution within the hierarchy. The super-squares on the right are one level of resolution higher than the one on the left.

If we consider a square domain (without any edges), the edges of a super-square are formed in two stages. First, by the diagonal edges joining the center of the square to its four corners, and second, by the edges joining the midpoint of each of the four edges of the square to the two closest square corners, and to the square's center, yielding a total of 16 edges. Recall that there is a one to one correspondence between diamonds and edges of an LEB mesh through the diamond's spine (or alternatively, through its central vertex). Thus a super-square corresponds to two consecutive levels in a containment triangle hierarchy being formed by four adjacent 0-diamonds which are in turn subdivided into 1-diamonds.

To ensure that each edge in the LEB mesh is only associated with a single super-square, we adopt the common convention used for quadtrees [20] that a super-square uses *half-open* intervals, i.e.

it contains the edges on its lower boundaries but not the edges on its upper boundaries. Thus, we consider any edge of a super-square σ whose endpoints are both on an upper boundary of σ to belong to another super-square. In other words, if the center point of a spine lies on one of the upper boundaries of σ, then that diamond belongs to a neighboring super-square. Consequently, of the 16 original edges, a super-square only contains the 12 edges that satisfy the half-open criteria and consists of 4 square diagonals (0-diamonds) and 8 square edges (1-diamonds). This is illustrated in Figure 4(a), where the solid lines with filled squares indicate the spines and central vertices of 0-diamonds, the solid lines with filled circles indicate the spines and central vertices of 1-diamonds and the dashed lines with hollow circles indicate diamonds belonging to neighboring super-squares due to the half-open interval convention.



(a) Diamonds in a Super-Square (b) Diamond as offset

Figure 4: (a) Correspondence between super-squares and diamonds: each diamond's spine coincides with an edge (solid lines) of a single super-square. (b) Alternatively, a diamond's central vertex lies at the midpoint of an edge of a super-square.

We associate a unique *type* τ with each diamond based on the super-square edge with which its spine coincides. If we consider the lower left corner of a super-square to be its local origin, the type corresponds to the offset of the midpoint of each edge from this origin, and is represented as an ordered pair. Figure 4 illustrates these offsets as binary numbers along the x- and y-axes. This association allows us to simplify operations on the diamond mesh and to create lookup tables for a diamond based on its type.

5.2 Representing Diamonds

Using a few simple bit operations on the binary representation of a diamond's central vertex, we can directly extract properties of a diamond, such as its type, scale and corresponding super-square. A diamond δ's hierarchical and geometric *elements* such as the coordinates of its vertices and the central vertices of its parents and children are then implicitly determined from δ's scale and type using scaled offsets from δ's central vertex.

Let δ be a diamond with central vertex $\mathbf{v}_c$ and let b_x and b_y denote the binary representation of the x and y coordinates of $\mathbf{v}_c$. The coordinates encode three pieces of information. First, the scale of the diamond, $s = \text{SCALE}(\delta)$ is encoded by the minimum of the number of trailing zeros in b_x and b_y. Next, the two bits at positions $s+1$ and $s+2$ from the right of b_x and b_y encode the diamond type, $\tau(\delta)$. Furthermore, the rightmost bit of each component of τ determines the class of δ. By the definition of $\text{SCALE}(\delta)$ they cannot both be zero; if they are both non-zero, δ is a 0-diamond, otherwise, δ is a 1-diamond.

Finally, by clearing the type bits of δ, we can evaluate the origin of the super-square σ to which δ is associated, i.e. the coordinates of the lower left corner of σ. Thus, an alternate interpretation of a diamond's type τ is that it represents the unscaled offset of δ's central vertex from the origin of its associated super-square σ. We note that the coordinates of the central vertices of all diamonds associated with a given super-square σ agree on all bits but the type bits (i.e. those at positions $s+1$ and $s+2$).

Since all vertices of the diamond mesh have integral coordinates, and there are only 12 types of diamonds, these offsets can be precalculated and accessed via a lookup table. We do so by keeping track of vertices and parent-child relationships while using a subdivision scheme such as the one given in [17]. All unscaled offsets $\vec{f}$ have components $f_i \in \{-1, 0, 1\}$. The actual coordinates of an element $\mathbf{p}$ of δ at unscaled offset $\vec{f}$ can be computed at runtime as:

$$\mathbf{p} = \mathbf{v}_c + 2^{Scale(\delta)} * \vec{f}.$$

As an example, consider a diamond δ whose central vertex $\mathbf{v}_c$ has coordinates $(28, 8)$. We first look at the binary representation of its coordinates, $b_x = 011100_2$ and $b_y = 001000_2$. Its scale, type and corresponding super-square can be determined from $\mathbf{v}_c$ as follows: b_x has two trailing zeros and b_y has three trailing zeros, so $\text{SCALE}(\delta) = \min(2, 3) = 2$. The diamond type is determined by bits 3 and 4 of b_x and b_y and is thus $\tau(\delta) = (11_2, 10_2) = (3, 2)$ (see Figure 4(b)). Since the set of rightmost bits of τ contains a single zero, δ is a 1-diamond. Finally, by clearing bits 3 and 4 of b_x and b_y, we find the coordinates of the super-square to which δ is associated, namely, the one at scale 2 whose origin is located at coordinates $(010000_2, 000000_2) = (16, 0)$. To find an element at offset $(-1, 0)$ from $\mathbf{v}_c$, we first scale the offset by $2^{Scale(\delta)} = 4$ and add the result to $\mathbf{v}_c$ to obtain: $(28, 8) + 4 * (-1, 0) = (24, 8)$.

6. STP DATA STRUCTURE

We utilize the super-square notion in several ways when modeling multiresolution terrains. First, it is the conceptual model of our multiresolution scheme, simplifying the extraction of diamond type and scale from a diamond's central vertex (as introduced in Section 5). Super-square clustering also enables an efficient encoding of sparse terrain pyramids and variable-resolution triangle meshes extracted from an STP. Finally, super-square clustering reduces the dependency of our representation from the resolution of its originating dataset. We discuss the first application in this section, and the latter two applications in the sections that follow.

Due to the transitive closure requirement in an STP, there is a high degree of hierarchical coherence in addition to the spatial coherence among samples associated with a given super-square. Namely, if a diamond is required to satisfy a given error criterion, it is likely that its neighbors, parents and children are also necessary. Consequently, of the 12 diamonds per super-square, it is common for an STP representation to average 9 or more diamonds per super-square, with many super-squares containing all 12 diamonds. The non-full super-squares are typically those that are close to the boundary of the domain or those containing leaf nodes.

This observation leads us to a representation for a sparse terrain pyramid that exploits the coherence within super-squares. We use the 1-1 correspondence between the central vertices of diamonds and super-square edges to store information about vertices, such as their associated height values, within super-squares. Thus, super-squares encode a map from grid points to diamonds. E.g., given a vertex whose corresponding super-square is σ, and whose type is τ, then $\sigma[\tau]$ corresponds to the diamond whose central vertex lies on edge τ of σ.

An implementation of this mapping must carefully balance storage space and computational speed. When speed is the most impor-

tant factor, a super-square's internal map can be implemented using an array of 12 elements. If instead we would like to minimize storage space e.g. as a file format, then a bit field of 12 bits is sufficient to track and locate the diamonds. As a compromise, a tree or hash map can be used to access the diamonds in a super-square. We note that since the number of diamonds per super-square is strictly limited to 12, all of these operations can be handled in $O(1)$ time. For our implementation, super-squares in memory use a 12 element array to speed up access to elements, while super-squares on disk use bit flags to index the individual diamonds.

Each vertex requires 4 bytes: the elevation value is stored as a 2 byte signed integer and the approximation error for its associated diamond is quantized using 2 bytes into the range $[0, 1]$. For each super-square, we need only the spatial location of its lower left corner as well as a bit-flag indicating which diamonds are present. This requires a total of 6 bytes per super-square: 2 bytes for each coordinate and one bit corresponding to each of the 12 possible diamonds (represented using two bytes). We eliminate the need to store a super-square's scale by storing all super-squares from the same scale together. The set of super-squares at the same scale can then be indexed by its scale.

This partitioning of super-squares by their scale has an additional advantage as it provides a means of reducing the dependency on the resolution of the original dataset. Since the sparse pyramid structure stores the associated elevation and error values of diamonds within super-squares, it is self contained and thus no longer requires the original DEM dataset. An application of this is to augment a dataset with samples from a corresponding dataset of higher resolution, which we discuss in Section 8.

Finally, although they are not diamonds, we apply the transformation of Section 5 to the four corner vertices of the domain. The lower left corner maps to a super-square with coordinates $(0, 0)$ whose scale is determined by the number of bits used by the machine to represent coordinates (e.g. 32 for 4 byte integers). The other three corners map to a super-square at location $(0, 0)$ at a scale of MAXLEVEL $+ 1$.

Discussion.

Super-square clustering of diamonds is an efficient representation of an STP when the super-squares have a high density i.e., average number of diamonds per super-square, but due to the geometric and topological overhead of the super-square representation a full hierarchy is more efficient when the number of omitted samples is not sufficiently smaller than the number of samples in the original dataset.

Recall that an FTP built on a grid of dimension $(2^k+1)^2$ requires storage of a scalar value and an error value at each vertex, which are implicitly associated with the central vertex of each diamond and are encoded as arrays. If $|\Delta|$ is the number of diamonds in the full hierarchy, then a full representation requires $4 * |\Delta|$ bytes. Our super-square based STP requires 6 bytes per super-square, and 4 bytes for each represented diamond. Thus, the size of a super-square based STP is $6*|\sigma|+4*|\delta|$, where $|\sigma|$ is the number of super-squares and $|\delta|$ is the number of diamonds in the multiresolution model. Observe that this representation only requires the storage of elevation values for diamonds with children, and thus does not require any storage for the diamonds at the highest resolution.

Thus, a super-square based STP requires less storage than the FTP when

$$6 * |\sigma| + 4 * |\delta| < 4 * |\Delta|.$$

Let $\alpha = |\delta|/|\sigma|$ be the average super-square density, then substituting α for σ and simplifying, we have

$$(4 + \frac{6}{\alpha}) * |\delta| < 4 * |\Delta|,$$

and thus, in terms of the super-square density, our sparse pyramid representation yields a compressed dataset when we have fewer than $(\frac{4}{4+6/\alpha})$ of the diamonds as the FTP. Table 1 lists these percentages for integral values of α.

On the other hand, an explicit STP representation without super-squares would require 8 bytes per diamond, i.e. 4 bytes for the coordinates of the central vertex in addition to the 2 bytes each for the elevation and error values. Thus a super-square based STP is more compact than an explicit STP when:

$$6 * |\sigma| + 4 * |\delta| < 8 * |\delta|,$$

or, substituting σ with α, when $\alpha > 1.5$.

7. ENCODING EXTRACTED MESHES

Crack-free variable-resolution triangle meshes are extracted from an STP using a process called *selective refinement*, which is performed by traversing the STP hierarchy in a top-down manner while preserving the dependency relation. This is done through a cut of the (implicit) DAG, where the set of diamonds along the *active front* of the cut represent the current state of the refinement process. This process is guided by a user-defined predicate called the *error criterion*, which can be based on factors such as the approximation error, height value, location, distance to the view point and projected screen error of each diamond. To ensure that extracted meshes do not contain cracks, the selective refinement process requires all of a diamond δ's parents to subdivide before it can update δ. Although, in the worst case, this requirement can trigger subdivisions recursively up to the root of the hierarchy, the cost of such non-local subdivisions is amortized over the set of diamonds in the neighborhood of δ.

Consequently, an efficient implementation of selective refinement requires fast access to the diamonds in the active front as well as their ancestors and descendants. Recall that each non-boundary diamond contains the two triangles sharing the longest edge of the diamond. Also, each triangle in the front is only associated with a diamond after its LEB parent has subdivided. Thus, diamonds in an active front require one bit to determine the presence or absence of each of their triangles. Since each bit also corresponds to the subdivision status of that triangle's parent diamond, this encoding enables efficient updates to diamonds in the active front.

We encode an active front of a selective refinement query in terms of super-squares as well. Each super-square in such a front requires 7 bytes: 3 bytes to represent the bit flags (e.g. 1 bit for each of the 24 possible triangles) in addition to the 4 bytes for the coordinates of its origin. Empirically, we found that these active front super-squares average between 5 and 7 triangles, and thus, our representation requires between $1 - 1.4$ bytes per triangle in the active front. In contrast, a similar diamond-based representation would require 4 bytes to index each diamond in addition to the 2 bits of bookkeeping. Although a more efficient (4 bit per tree branch) code was introduced in [9] for bintrees, we note that our method enables random access to the elements, and does not require the use of a finite automaton to track the level of diamonds.

8. MERGING CORRESPONDING STPS

In this Section, we consider the problem of merging two STPs at different resolutions, covering portions of the same domain, where

α	1	2	3	4	5	6	7	8	9	10	11	12
$\frac{4}{(4+6/\alpha)}$	40%	57%	67%	73%	77%	80%	82%	84%	86%	87%	88%	89%

Table 1: Relationship between the super-square density $\alpha = |\delta|/|\sigma|$ and the percentage of total diamonds that can be represented by a sparse pyramid while still being more compact than the full hierarchy.

corresponding samples do not necessarily map to the same coordinates. We observe that the resolution of a dataset is a bottom-up distinction, that is, it is determined by the minimum distance between samples. However, corresponding datasets are aligned in a top-down rather than a bottom-up manner, i.e. their roots correspond to the same diamond. Thus, a reinterpretation of super-squares in a top-down manner would enable the alignment of datasets of different resolutions. Since a diamond's scale is a bottom-up characteristic and its level is a top-down characteristic, this requires a method to represent super-squares by their level rather than by their scale.

Recall that the super-square structure clusters together those diamonds whose coordinates agree on all but two bits, i.e. the bits corresponding to their diamond type τ, and that all the bits to the right of these bits are zero. Consequently, the origin of a super-square σ at scale s has at least $s+2$ trailing zeros in each of its coordinates, and σ can be unscaled by shifting its coordinates to the right by $s+2$ bits. Unscaled super-square origins at a particular level are thus a subset of the points of a regular grid.

Since the scale of a super-square is a function of the level of its diamonds as well as the resolution of the dataset, i.e. MAXLEVEL, a super-square based STP representation can store the unscaled coordinates of the origins of its super-squares and rescale them at runtime. Let σ be a super-square at scale s whose unscaled origin is located at point $\mathbf{p}$. Then the central vertex $\mathbf{v}_c$ of a diamond with type τ can be calculated as: $\mathbf{v}_c = \mathbf{p} \ll (s+2) + \tau \ll s$, where $\ll$ denotes a bitwise left-shift operator on each component of its left operand. Super-squares can then be partitioned by either their level or their scale, corresponding to top-down or bottom-up representations, respectively.

Thus, aligning two top-down super-square based STP representations is accomplished by simply setting the maximum level of the lower resolution dataset to that of the higher resolution dataset. Specifically, let A and B be two terrain pyramids where dataset A has a resolution of $(2^\jmath + 1)^2$ and dataset B has a resolution of $(2^k + 1)^2$, such that $\jmath < k$. Then MAXLEVEL(A) is $\jmath$ and MAXLEVEL(B) is k. A top-down super-square based STP representation of dataset A can be aligned with dataset B by simply increasing MAXLEVEL(A) to k.

An advantage of this unscaled representation is that the dataset is no longer dependent on the resolution of the original DEM and can be dynamically rescaled. Furthermore, this unscaled representation requires the same amount of storage as the scaled super-square based STP representation. However, since this requires rescaling the super-squares at runtime, the unscaled representation has a slight (but constant) computational overhead to the scaled representation.

When merging two corresponding STPs, we typically need to insert additional vertices to maintain the transitive closure of the resultant STP. Each of these new vertices requires an elevation as well as an error value. For the elevation values, we recursively interpolate the value from the two endpoints of its associated diamond's spine. This is guaranteed to terminate at the four corners of the domain, but will typically terminate much earlier. Since we want to ensure that the diamonds from the new dataset are reached, we set the error of the new points to the maximum possible error.

Due to the large number of shared ancestors within the sparse pyramid, the number of points necessary to ensure transitive closure is often quite small relative to the number of diamonds being inserted, but depends on the location to which it is inserted. For example, let S be an empty STP with MAXLEVEL $= 30$, e.g. its equivalent FTP would have a resolution of $(2^{30}+1)^2$. Adding a 2x2 block of samples to S requires only a few hundred samples to maintain transitive closure, while adding a 1025^2 block of highest resolution samples to S has an overhead of less than 1% e.g. whereas the 1025^2 block contains 1,050,625 samples, an empty sparse pyramid of size $(2^{30}+1)^2$ with this block added has fewer than 1,060,000 diamonds. Furthermore, adding a 4097^2 block from the same DEM requires fewer than 0.1% additional samples to maintain transitive closure.

9. EXPERIMENTAL RESULTS

In this Section, we present experiments on the sparse pyramids structure and compare our sparse representation to that of Gerstner [9]. We performed our experiments on several regular datasets, including DEMs of Mt. Marcy, the Grand Canyon, Devils Peak, San Bernardino, two versions of the Puget Sound at different resolutions, and a tile of the gTopo30 covering a portion of Australia (see Figure 2). Datasets whose dimensions are not $(2^k+1)^2$ were embedded into the smallest containing virtual grid of dimensions $(2^k+1)^2$.

A typical application of sparse pyramids is to represent multiresolution terrains extracted from an FTP where the error ϵ of any diamond in the STP is less than some threshold approximation error. Table 2 summarizes the sizes of sparse diamond pyramids extracted from the dataset testbed with a range of uniform errors.

Figure 5 shows the average density (diamonds/super-square) of the various datasets from Table 2. As can be seen from the table, when encoding a super-square based STP of uniform error less than one percent, there are between 5 and 12 diamonds per super-square on average, and for errors less than 0.1 percent, the average is between $9-11$ diamonds for most datasets. Note also, that the average density of super-squares increases as the error threshold decreases.

Since the overhead per super-square is 6 bytes, our super-square based STP representation has an overhead of less than 1 byte per vertex when the average super-square density is greater than 6, an approaches 0.5 bytes per vertex as the average density approaches 12. Compared to the average of 3 Bytes per vertex in [9], our method is 3-6 times more space efficient. Furthermore, the stack-based method of [9] does not provide random access to its vertices and requires up to $O(\sqrt{|\delta|})$ extra storage in memory, where $|\delta|$ is the total number of diamonds.

Sparse terrain pyramids are suitable for representing datasets extracted from DEMs using arbitrary selective refinement criteria. These include polygonal regions such as squares and circles (see Figure 6) as well as polylines. Additional error functions include distance or view dependent criteria as well as samples relevant to specific contour lines or ranges of contour lines within the datasets.

When adding higher resolution components to a sparse pyramid, we are effectively raising the resolution of the mesh to that of the higher resolution component, as discussed in Section 8. We simulate a situation where a higher resolution component is available

Dataset	Dims	10% Error			1% Error			.1% Error			0% Error		
		$\|\sigma\|$	$\|\delta\|$	size	$\|\sigma\|$	$\|\delta\|$	size	$\|\sigma\|$	$\|\delta\|$	size	$\|\sigma\|$	$\|\delta\|$	size
San Bern	128×128	222	867	4.7K	1.2K	8K	39K	1.4K	14K	63K	1.4K	16K	71 K
Devil's Peak	165×301	275	1.5K	7.3K	2.8K	19K	89K	3.3K	32K	145K	3.3K	32K	145K
Grand Canyon	1280×640	3.5K	16K	83K	28K	172K	836K	62K	556K	2.5M	62K	556K	2.5M
Mt. Marcy	1201×1201	1.3K	6K	32K	13K	68K	340K	63K	426K	2M	101K	939K	4.2M
Puget Sound 1k	1025×1025	710	3.3K	17K	41K	219K	1.1M	82K	761K	3.4M	86K	1M	4.3M
Puget Sound 4k	4097×4097	754	3.5K	18K	75K	354K	1.8M	880K	5.5M	26M	1.2M	9.7M	44M
Australia	4800×6000	18K	99K	490K	20.1K	110K	546K	50.1K	262K	1.3M	528K	4.78M	21M

Table 2: Number of super-squares ($|\sigma|$) and diamonds ($|\delta|$) as well as disk size(sz) in KB = 1024 Bytes or MB = 1024^2 Bytes) for sparse pyramids with different uniform errors. Note that the values for *Puget Sound 4k* dataset for 0% error are actually for 0.03% error rather than 0% error.

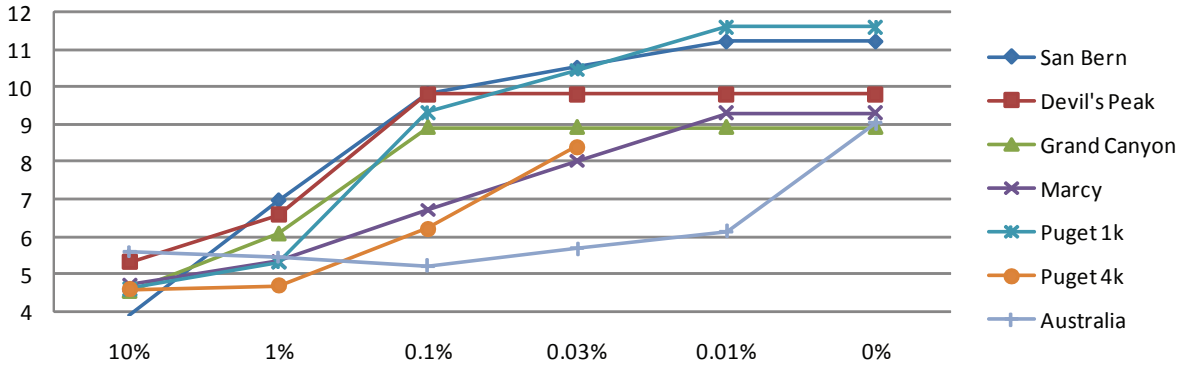


Figure 5: Average density of super-squares (vertical axis) with uniform error (horizontal axis). Derived from Table 2 as ($|\delta|/|\sigma|$).

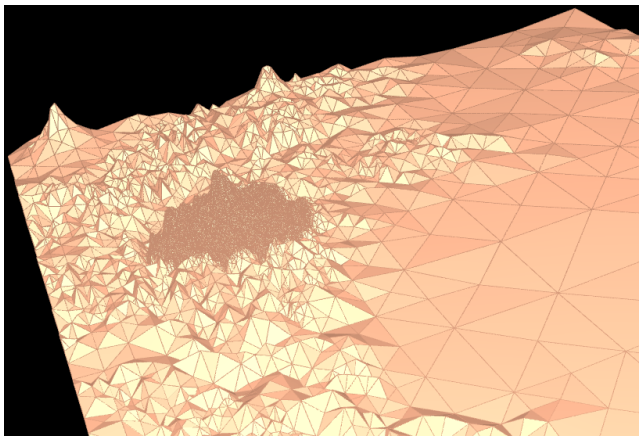


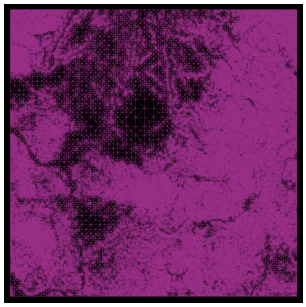
Figure 6: Circular region of interest (ROI) from Puget Sound 1k dataset. Values inside the ROI have 0 error while those outside the ROI have an approximation error less than 10%. This STP has 3600 super-squares with an average of 10.3 diamonds per super-square.

by using the two corresponding Puget Sound datasets, whose resolutions are 1025^2 and 4097^2, respectively. First, we extracted a sparse pyramid of uniform error less than 1% from the Puget Sound 1k dataset (see Figure 7(a)). This STP had 82 K super-squares and 761 K diamonds (density = 9.3), Next, we add to it a square ROI of side length 580 pixels from the Puget Sounds 4K dataset (see Figure 7(b)). This STP had 29 K super-squares and 341 K diamonds (density = 11.6). The combined sparse pyramid had 108 K super-squares and 1.08 M diamonds (density = 9.95). The number of shared samples between the two datasets is only 21 K or around 2% of the samples in the resultant dataset. Figure 7 illustrates a terrain extracted from this sparse pyramid.

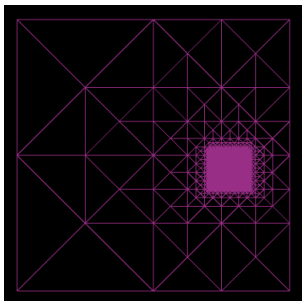
10. CONCLUDING REMARKS

We have introduced a compact representation for terrains encoded as sparse pyramids of diamonds. Our encoding of diamonds allows the recovery of all local mesh geometry and topology from the coordinates of the central vertex of each diamond. Furthermore, our super-square clustering of the diamonds reduces the geometric overhead by implicitly indexing up to 12 diamonds. Super-squares also enable an efficient encoding of an active front of a selective refinement query. We demonstrated the effectiveness of this clustering over a wide range of datasets and error criteria and discussed situations where a full pyramid would be more appropriate.

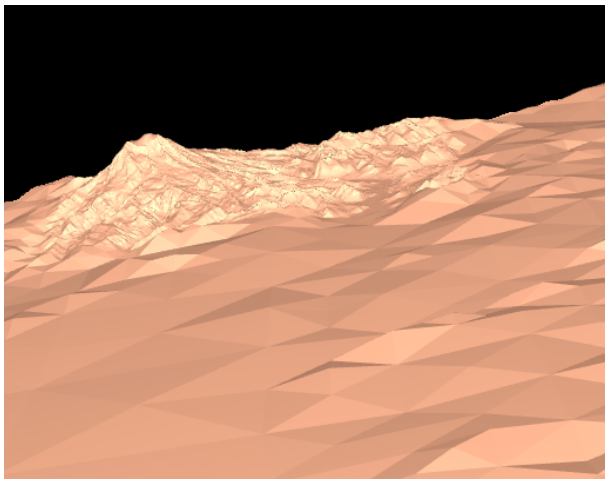
Compared to the sparse representation of Gerstner [9], our representation supports random access as well as general selective refinement queries and does not require keeping track of an automaton or stacks of height values. Finally, the overhead of our super-square based representation is less than 1 byte per vertex compared to 3 bytes in [9].



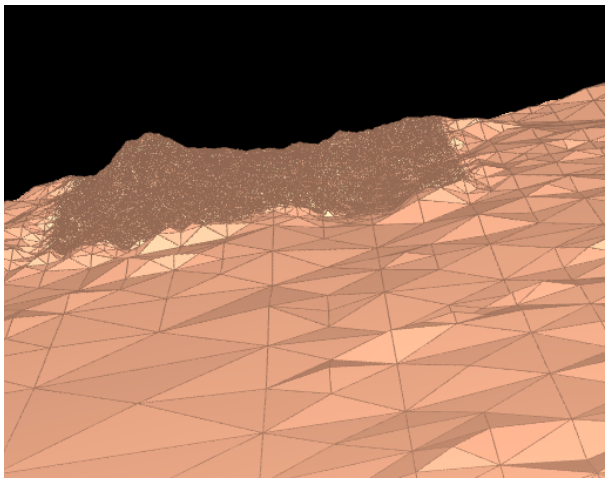
(a) Puget Sound 1k at 1% error.



(b) Puget Sound 4k with ROI.



(c) Combination of Puget datasets



(d) Combination of Puget datasets with wireframe

Figure 7: Planar projection of (a) the Puget sound 1k dataset with 1% error and (b) the Puget sound 4k dataset with square ROI. (c,d) Merged datasets after upscaling the lower resolution dataset. Shown without (c) and with (d) wireframe to highlight the size of the mesh elements.

We are currently researching an out-of-core access structure for the super-square based STP representation. For future work, we are considering using super-squares to compress the height values and approximation errors of their contained diamonds. For example, we are considering the feasibility of adding a base elevation or error value to a super-square to enable compression of the associated diamond values. Another compression method for the errors might be to quantize them on a level by level basis as in [11], where error components are quantized to 6 bits.

Additionally, for sparse pyramids that are static, we would like to apply perfect spatial hashing [13] to super-squares to further reduce the geometric overhead of our representation.

Another direction of future work is to generalize the super-square clustering technique to higher dimensions. This would be beneficial in modeling multiresolution volumetric and time-varying scalar fields, or interval volumes of volumetric grids.

11. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their many helpful suggestions. This work has been partially supported by the National Science Foundation under grant CCF-0541032, by the MIUR-FIRB project SHALOM under contract number RBIN-04HWR8 and be the MIUR-PRIN project on modeling of scalar fields and digital shapes. All datasets are based on data released by the US Geological Survey. Puget Sound data is courtesy of [15]. Grand Canyon dataset is courtesy of Chad McCabe. Australia dataset is from the GTopo30 dataset [22]. San Bernardino, Devil's Peak and Mount Marcy datasets were provided to the authors by Paola Magillo.

12. REFERENCES

[1] J. Blow. Terrain Rendering at High Levels of Detail. *Proc. of the 2000 Game Developers Conference*, 2000.

[2] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. BDAM- Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization. *Computer Graphics Forum*, 22(3):505–514, 2003.

[3] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM). *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, 2003.

[4] L. De Floriani and P. Magillo. Multi-resolution mesh representation: models and data structures. In M. Floater, A. Iske, and E. Quak, editors, *Principles of Multi-resolution Geometric Modeling*, Lecture Notes in Mathematics, pages 364–418, Berlin, 2002. Springer Verlag.

[5] L. De Floriani, P. Magillo, and E. Puppo. VARIANT: a system for terrain modeling at variable resolution. *Geoinformatica*, 4(3):287–315, 2000.

[6] L. De Floriani, E. Puppo, and P. Magillo. Applications of computational geometry to Geographic Information Systems. In J. R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 7, pages 333–388. Elsevier Science, 1999.

[7] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: real-time optimally adapting meshes. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization'97*, pages 81–88, Phoenix, AZ, October 1997. IEEE Computer Society.

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

[9] T. Gerstner. Multi-resolution visualization and compression of global topographic data. *GeoInformatica*, 7(1):7–32, 2003.

[10] T. Gerstner. Top-down view-dependent terrain triangulation using the octagon metric. Technical report, Institut für Angewandte Mathematik, University of Bonn, 2003.

[11] B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings IEEE Visualization 2002*, San Diego, CA, October 2002. IEEE Computer Society.

[12] L. Hwa, M. Duchaineau, and K. Joy. Real-Time Optimal Adaptation for Planetary Geometry and Texture: 4-8 Tile Hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, pages 355–368, 2005.

[13] S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics (TOG)*, 25(3):579–588, 2006.

[14] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. Real-time continuous level of detail rendering of height fields. In *Proceedings SIGGRAPH'96*, pages 109–118, August 1996.

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

[16] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann, San Francisco, 2002.

[17] J. M. Maubach. Local bisection refinement for n-simplicial grids generated by reflection. *SIAM Journal on Scientific Computing*, 16(1):210–227, January 1995.

[18] R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *Proceedings IEEE Visualization'98*, pages 19–26, Research Triangle Park, NC, October 1998. IEEE Computer Society.

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

[20] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Elsevier, 2006.

[21] R. Sivan and H. Samet. Algorithms for constructing quadtree surface maps. *Proc. 5th Int. Symposium on Spatial Data Handling*, pages 361–370, 1992.

[22] U.S. Geological Survey. Global 30 arc second elevation data. `http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html`.