In Proceedings Visualization 98, pages 19–26 and 515. IEEE Computer Society Press, 1998.

Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation

Renato Pajarola, ETH Zürich[1]

Abstract

Real-time rendering of triangulated surfaces has attracted growing interest in the last few years. However, interactive visualization of very large scale grid digital elevation models is still a hard problem. The graphics load must be controlled by an adaptive surface triangulation and by taking advantage of different levels of detail. Furthermore, the management of the visible scene requires efficient access to the terrain database. We describe a all-in-one visualization system which integrates adaptive triangulation, dynamic scene management and spatial data handling. The triangulation model is based on the restricted quadtree triangulation. Furthermore, we present new algorithms of the restricted quadtree triangulation. These include among others exact error approximation, progressive meshing, performance enhancements and spatial access.

Keywords algorithms, computer graphics, virtual reality, triangulated surfaces, terrain visualization, terascale visualization

1. Introduction

Interactive visualization of very large scale terrain data brings up a wealth of problems. The main problem in real-time graphics is rendering efficiency. To best exploit the rendering performance, the scene complexity must be reduced as much as possible without leading to an inferior visual representation. Therefore, the geometry simplification must be controlled by an approximation error threshold. Another way to increase efficiency is the use of different *levels of detail* (LODs) for different areas of the visible scene. Objects are displayed in lower resolutions – with higher approximation errors – the farther away they are from the view focus.

Large-scale terrains are usually too large to be displayed as a whole, even when using multiple LODs. For instance, the grid digital elevation model of Switzerland at 25 meter grid-resolution consists of more than 120 million triangles. Therefore, only a fraction of such an extensive model can be rendered at an interactive frame-rate. This partial scene, however, must be *updated dynamically* according to changes of the view parameters. Therefore, the data structure holding the terrain data must support spatial access. Additionally, an approximation error parameter is specified to indicate the correct LOD. Furthermore, incremental refinement must be supported. Efficient storage and compact topology is required to realize a small database and allow fast transmission.

The triangulation model is the core structure for every terrain surface visualization system. To achieve low complexity without negative impact on accuracy, the triangulation must be adaptive to the terrain surface characteristics. Furthermore, the triangulation model must provide means to extract surface representations at variable precisions so that multiple LODs can be supported. The following list briefly recalls the requirements for the visualization system, and in particular for the triangulation model. The list is not sorted by any priority, it is rather sorted by related topics.

1. fast access to large scene database
2. effective database storage management
3. variable resolution database access
4. terse topology representation
5. quick adaptive triangulation
6. dynamic visible scene management
7. multiresolution visualization
8. high-performance geometry rendering
9. continuous level of detail rendering

In Section 2 we briefly discuss similar terrain visualization approaches. Section 3 introduces the restricted quadtree triangulation. Two new construction algorithms are presented in Section 4, and exact error approximation is explained in Section 5. Performance issues are discussed in Section 7, and support of progressive meshing is described in Section 6. Sections 8, 9 and 10 discuss dynamic data handling, continuous LOD rendering, and storage. Experiments are presented in Section 11, and Section 12 concludes the paper with a comparison to other multiresolution triangulation models. More detailed information on all sections can also be found in [11,10].

2. Related work

In [8,9] the *restricted quadtree triangulation* (RQT) was first applied to terrain visualization. Their contribution was an efficient screen-space error metric calculation, and efficient scene culling and vertex selection according to this error metric. Furthermore, they presented a cumbersome triangle strip construction algorithm. A second application of the RQT to terrain visualization is described in [5]. They introduce a different notation for the same class of restricted quadtree meshes. The main contribution was a priority-queue driven mesh refinement and a combination of object- and screen-space error metric.

In contrast to [8,9] we provide two alternative vertex selection algorithms, and provide a more intuitive triangle strip construction method. Furthermore, we explain exact error approximation based on the RQT. Contrarily to [5] we present an error calculation that can also handle *non-monotonic* error metrics which is essential to extract exact approximations according to the L_∞-norm for a broader class of error metrics. Moreover, in extension to the previous applications, we combine the RQT with dynamic scene management, spatial access and progressive meshing in an integrated large scale terrain visualization system.

Other multiresolution triangulation models which can handle different classes of meshes are considered in the conclusion as well.

3. Restricted quadtree triangulation

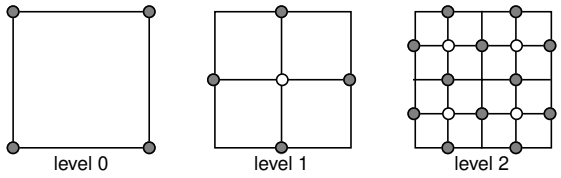
The RQT is an adaptive, hierarchical triangulation model. In [17] the restricted quadtree was used to triangulate a parametric surface. However, no algorithm was provided to efficiently build the restricted one from a plain quadtree. In addition to the algorithms in [14], we will provide two slightly different construction methods, one theoretically optimal in the sense of [13] and

1. Institute of Theoretical Computer Science
ETH Zentrum, 8092 Zürich, Switzerland
pajarola@inf.ethz.ch

an implementation-efficient one. Furthermore, we present a new triangle strip construction, progressive meshing and spatial access, all based on the restricted quadtree. Because of space limitations only preliminary details are presented in this introduction.

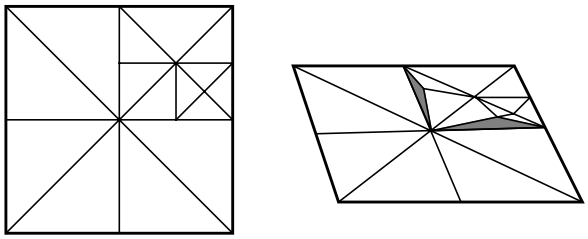
The assignment of grid points to levels in the quadtree hierarchy is shown in Figure 1, with the root being level 0. Given the grid size dimension $n = 2^k + 1$ and points $P_{i,j}; i, j = 0..n-1$, the points on lower levels $l \geq 0$ are given by $L_l = \{P_{i,j} | h_l = 0, 1, ..2^l; i, j \in \{0, d_l, ..h_l \cdot d_l\} \wedge P_{i,j} \notin L_{l-1}\}$. Note that the resolution $d_l = (n-1)/2^l$ is level dependent. The center-vertices L_l^{center} denote points of L_l which are located in the center of a quadtree block on level l-1 (*white* points in Figure 1). Let $\cup L_l = \cup_{k=0}^{l} L_k$ be all points down to level l, and $L_l(Q) \subseteq L_l$ be the vertices on level l of an unbalanced restricted quadtree Q.

FIGURE 1. Hierarchy levels



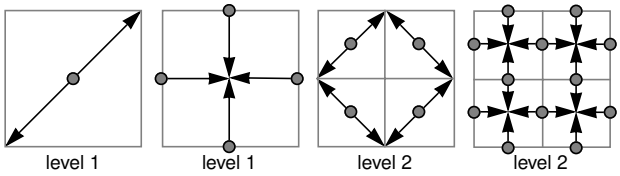
An example of the recursive triangulation of a plain quadtree is shown in Figure 2. However, *cracks* can occur in the corresponding three-dimensional surface as shown on the right. To guarantee a matching triangulation, cracks have to be avoided. In [17] this was granted by the restriction that adjacent quadtree blocks differ by at most one level in the quadtree hierarchy, and that every quadtree block is triangulated by eight triangles, or two triangles per boundary edge, unless the edge borders a larger block. However, this method may produce extra triangles, and provides no deterministic algorithm to convert a plain quadtree into a restricted.

FIGURE 2. Nonrestricted quadtree triangulation



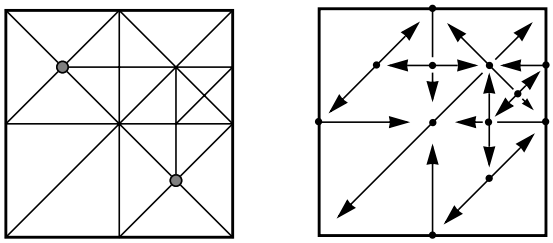
Another way is to represent the restriction of the RQT as a dependency graph on the grid, see also [9]. Every vertex depends on two other vertices of the same or the next higher level in the quadtree hierarchy. This means that if the vertex is selected for triangulation then the related must be selected too. The dependency graph for levels 1 and 2 is depicted in Figure 3. We use the notation $G_{dep} = (V_{dep}, E_{dep})$ to refer to the complete dependency graph, and includes all vertices $V_{dep} = \cup L_{\log 2(n-1)}$. The dependency relation is denoted by the directed edges set $E_{dep} = \{(P_1, P_2) | \text{ dependency from } P_1 \text{ to } P_2\}$.

FIGURE 3. Dependency graph



The dependency graph G_{dep} prevents cracks by consistently restricting the selection of points such that a matching triangulation results. Figure 4 shows the restricted version of the quadtree triangulation from Figure 2, with the dependency graph shown on the right. Using G_{dep}, any given set of points on the grid can now decisively be turned into a restricted quadtree by resolving the dependency rules. Note that the triangulation can efficiently be constructed because it is given implicitly, no geometric computations such as in-circle tests need to be preformed.

FIGURE 4. Restricted quadtree triangulation



4. Restricted quadtree construction

For the construction, we assume that the decision of selecting a point due to its approximation error can be determined by a threshold. The calculation of a suitable approximation error is described in Section 5. By collecting all vertices following the dependency graph the selection is completed to a restricted quadtree. Note that a quadtree data structure on a $(2^k+1) \times (2^k+1)$ grid can be implemented using index operations and recursion instead of using pointers and nodes. We call this an implicit quadtree defined on a grid.

In our first *top-down* algorithm we assume that the elevation points are kept in a region quadtree, and that an approximation error is associated to every point. Additionally, each node knows the maximal error of its own triangulation with respect to all points stored in that subtree. *SelectVertices()* selects all points of the current node satisfying the approximation threshold, and recursively traverses the given quadtree. For every selected vertex v, *ResolveDependencies()* adds the related points $\{p | (v, p) \in E_{dep}\}$ to the current selection, and follows the dependency graph.

The selection result *QSet* can be a new quadtree, or abstract in the sense that it is not another data structure but a selection of vertices in the original quadtree structure. Moreover, this restricted quadtree selection can even just be implicitly defined on the height field grid. For that, the procedures *set.insert()* and *set.contains()* can be implemented by setting and testing a flag for every point. The complexity of Algorithm 1 and the respective triangulation is linear in the size of the extracted restricted quadtree. Moreover, the complexity is theoretically optimal in the sense of [13] as the RQT is a multi-triangulation.

ALGORITHM 1. Recursive restricted quadtree construction

```
PROCEDURE SelectVertices(node: QTree; errmax: INT;
VAR set: QSet);
VAR v: Vertex; son: POINTER TO QTree;
BEGIN
    FOREACH v IN node.vertices() DO
        IF v.error >= errmax AND NOT set.contains(v) THEN
            set.insert(v);
            ResolveDependencies(v, set)
        ENDIF
    END;
    FOREACH son IN node.descendents() DO
        IF son.maxerror() >= errmax THEN
            SelectVertices(son, errmax, set)
        ENDIF
    END
END SelectVertices;

PROCEDURE ResolveDependencies(v: Vertex; VAR set:
QSet);
VAR dep: Vertex;
BEGIN
    dep := v.left();
    IF NOT set.contains(dep) THEN
        set.insert(dep);
        ResolveDependencies(dep, set)
    ENDIF;
    dep := v.right();
    IF NOT set.contains(dep) THEN
        set.insert(dep);
        ResolveDependencies(dep, set)
    ENDIF
END ResolveDependencies;
```

Theorem 4.1 *The number of calls to set.contains() and set.insert() of* Algorithm 1 *is linear in the size of the resulting restricted quadtree Q.*

Proof A node of the input quadtree is required if the subtree of that node contains a point that does not satisfy the approximation tolerance. All required nodes are visited once by *SelectVertices()*. The resulting restricted quadtree Q consists at least of all required nodes of the input quadtree. Thus we have $O(|Q|)$ calls to *set.insert()* and *set.contains()* by *SelectVertices()*.

For every selected point in *SelectVertices()*, *ResolveDependencies()* is called once. The recursion in *ResolveDependencies()* visits all vertices of Q at most four times as no more than four arcs of the dependency graph end in one vertex, see also Figure 3. Hence also here we have $O(|Q|)$ calls to *set.insert()* and *set.contains()* by *ResolveDependencies()*. □

Contrarily to the top-down method in [14] we do not predict and interpolate next-level vertices to guide recursive tree traversal, instead we use the exact approximation errors of Section 5. Also the restriction of the quadtree is handled differently, using the dependency relation E_{dep}.

Our second algorithm operates iteratively *bottom-up*. Starting at the lowest level with the highest resolution, the algorithm iterates over all levels examining the particular points on each. But for the organization of the points in a matrix, the same presumptions hold as for Algorithm 1. During the inspection within a level a vertex is selected if its error exceeds the tolerance threshold, or if the point is marked from a dependency relation. Additionally, the two related vertices are marked accordingly.

In contrast to Algorithm 1, this bottom-up approach visits vertices only once. However, the marking mechanism has to be studied carefully such that no dependencies might be left out in any case. Consequently, no recursive resolution of the dependency graph must be performed, as well as no set-inclusion tests are needed to stop recursion. As in Algorithm 1, the selected vertices are stored in a query answer data structure, or form an implicit quadtree. The efficiency of Algorithm 2 results overall from a low number of simple operations computed per vertex.

ALGORITHM 2. Iterative restricted quadtree construction

```
PROCEDURE CollectVertices(field: Matrix; n, errmax: INT;
VAR set: QTree);
VAR l: INT; v: Vertex;
BEGIN
    l := log2(n-1);
    WHILE l >= 0 DO
        FOREACH v IN field.verticesOfLevel(l)
            IF v.error >= errmax OR v.marked THEN
                set.insert(v);
                v.left().marked := TRUE;
                v.right().marked := TRUE
            ENDIF;
            v.marked := FALSE
        END;
        DEC(l)
    END
END CollectVertices;
```

Theorem 4.2 *Algorithm 2 runs in time linear of the input data size, and no dependency relation of G_{dep} is omitted.*

Proof The while loop in Algorithm 2 iterates over all levels, and for each level the inner foreach loop must only visit points on that level. For a level l and the corresponding grid resolution d_l only the points on that level given by $\{P_{i,j} | (i \text{ DIV } d_l) \text{ MOD } 2 \neq 0 \vee (j \text{ DIV } d_l) \text{ MOD } 2 \neq 0\}$ are visited. Hence every vertex is visited exactly once which sums up to $O(n)$ for n input points.

To further guarantee consideration of all relevant dependencies of G_{dep}, the non center-vertices $L_l \backslash L_l^{center}$ have to be examined first within a level. This is sufficient because the dependencies of $L_l \backslash L_l^{center}$ point to L_l^{center}, and L_l^{center} point to vertices on level $l-1$, see also Figure 3. For the inner foreach loop this can be achieved by first visiting the points $\{P_{i,j} | i|_2^{d_l} \neq 0 \text{ XOR } j|_2^{d_l} \neq 0\}$ and afterwards $\{P_{i,j} | i|_2^{d_l} \neq 0 \wedge j|_2^{d_l} \neq 0\}$. (In this context, $k|_2^{d_l}$ abbreviates $(k \text{ DIV } d_l) \text{ MOD } 2$ in this context) □

In comparison to the algorithms in [14], we do not have to deal with insertion or merging of nodes, or examine neighboring nodes if we use an implicit quadtree.

5. Exact error approximation

We present an error metric in the object space to exactly control the approximation accuracy. Our error $e_T(P)$ for a Point P and a triangulation T is the point's euclidean distance $d(P,t)$ to the triangle $t \in T$ with $P_{z=0} \in \Delta(t)$. $\Delta(t)$ is the triangle domain in the x,y-projection $\mathbb{R}^2$. If $P_{z=0} \notin \Delta(t)$ then $d(P,t) = 0$. Therefore, the error of T is the maximal error of all points not in T, given by $e_T = \max_{\forall P \notin T}(d(P,t)) \wedge t \in T$. We combine the simplification idea of [4] with the partial order of G_{dep} to calculate $e_T(P)$.

The best triangulation not including L_l^{center} is $T_{l-1} = \{\text{RQT of } \bigcup L_{l-1}\}$, and without $L_l \backslash L_l^{center}$, it is $T_{l-1} = \{\text{RQT of } \bigcup L_{l-1} \cup L_l^{center}\}$. The *covering* of P are all triangles affected by the selection of P: $Cov(P \in L_l) = \{t \in T_{l-1} | \Delta(t) \cap P_{z=0} \neq \emptyset\}$. Given the inverse dependency relation $P \leftarrow Q \Leftrightarrow (Q,P) \in E_{dep}$, the transitive closure of P is defined as the set of vertices:

$$\zeta(P) = \{Q \in V_{dep} | \exists P_1, .., P_k; P \leftarrow P_1, .., P_k \leftarrow Q\} \cup \{P\}$$

Thus the approximation error of $P \in L_l$ is the maximum error with respect to *Cov(P)* of all points in $\zeta(P)$:

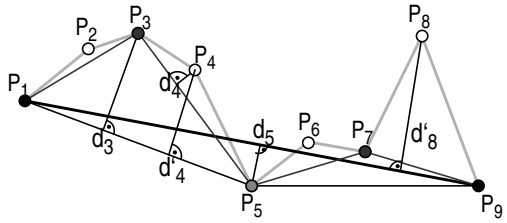
$$e_{T_{l-1}}(P) = \max_{Q \in \zeta(P), t \in Cov(P)}(d(Q,t)). \qquad \textbf{(EQ 1)}$$

Figure 5 illustrates a two-dimensional analog where the levels and the dependency graph are:

$$L_0 = \{P_1, P_9\}, L_1 = \{P_5\}, L_2 = \{P_3, P_7\}$$
$$L_3 = \{P_2, P_4, P_6, P_8\}$$
$$V_{dep} = \bigcup L_3$$
$$E_{dep} = \{(P_5, P_1), (P_5, P_9), (P_3, P_1), (P_3, P_5), ...\}$$

For example, P_5 is assigned the error d'_8 because this is the maximum error of all vertices in $\zeta(P_5) = \{P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$ to the line-approximation $\overline{P_1, P_9}$, especially $d'_8 > d_5$. On the next level however, the error of P_3 is exactly d_3, because no other vertices in $\zeta(P_3) = \{P_2, P_3, P_4\}$ have a bigger distance to $\overline{P_1, P_5}$.

FIGURE 5. Error calculation, 2D analog to restricted quadtree



A triangulation which satisfies the error tolerance ε everywhere on the input data is called an ε-approximation. Using $e_T(P)$ of Equation 1 we can construct an ε-approximation even for other non-monotonic error metrics in contrast to [5], and solve the exact approximation problem of restricted quadtrees mentioned in [2]. The algorithms presented in [14] cannot provide an exact ε-approximation either for non-monotonic error metrics because they do not take into account errors of lower level vertices in $\zeta(P)$.

6. Progressive meshing

Progressive meshing denotes the construction and transmission of a triangulation such that an intermediate triangulation can be updated easily given only a few new elements such as points, edges or triangles, see also [7]. Preferably, an update affects the topology only nearby the new elements and can be computed in constant time. Most hierarchical triangulations [2,3] support progressive meshing.

A *breadth-first* traversal of the restricted quadtree Q builds up the progressive mesh sequence of vertices. However, on level l the vertices $L_l^{center}(Q)$ must be transmitted first to guarantee a RQT. Using $\leq$ as the ordering relation the progressive mesh sequence is:

$$L_0(Q) \leq L_1^{center}(Q) \leq L_1(Q) \backslash L_1^{center}(Q) \leq ..$$
$$\leq L_{\log 2(n-1)}(Q) \backslash L_{\log 2(n-1)}^{center}(Q) \quad . \quad \textbf{(EQ 2)}$$

Each update (vertex insert) splits exactly two triangles up into four, and can be computed in constant time. A *vertex location* operation is needed to select the right insertion point using $O(\log n)$ time in a quadtree.

Lemma 6.1 *For a given restricted quadtree Q, the vertex ordering of Equation 2 provides a progressive, matching triangulation.*

Proof This ordering takes into account all dependency relations of Q. No vertex is used before its related vertices are. Therefore, any sub-quadtree $Q_l = \{\bigcup_{k=0}^{l} L_k(Q)\}$ is a restricted quadtree. □

A restricted quadtree ε-approximation Q_ε can also incrementally be updated to a smaller approximation error δ by adding an *error-range* quadtree $Q_{\Delta_{\varepsilon,\delta}}$ ($\Delta_{\varepsilon,\delta} = \varepsilon - \delta$). For $Q_{\Delta_{\varepsilon,\delta}}$ we select vertices according to an error-range instead of comparing to a threshold. Q_ε equals to $Q_{\Delta_{\infty,\varepsilon}}$, with $\Delta_{\infty,\varepsilon} = \infty - \varepsilon$. A progressive mesh sequence is then:

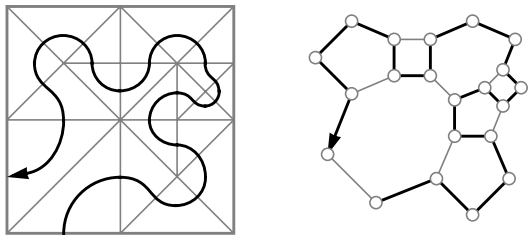
$$Q_{\varepsilon_0}, Q_{\Delta_{\varepsilon_0,\varepsilon_1}}, ..; \varepsilon_0 > \varepsilon_1 = \varepsilon_0 - \Delta_{\varepsilon_0,\varepsilon_1} > \varepsilon_2 = \varepsilon_1 - \Delta_{\varepsilon_1,\varepsilon_2} > ... .$$

Transmitting each quadtree's vertices ordered as in Equation 2 results in a smooth progressive meshing. To improve efficiency, Q_{Δ_i} can be inserted as a whole. Traversing $Q_{\varepsilon_{i-1}} = Q_{\varepsilon_0} + \sum_{k=1}^{i-1} Q_{\Delta_k}$ and Q_{Δ_i} *depth-first* gives a linear time update complexity $O(|Q_{\Delta_i}|)$.

7. Fast triangulation

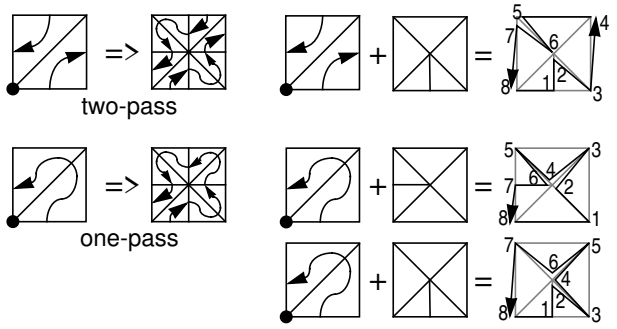
To enhance performance of the RQT, we introduce a *triangle strip* construction that is more intuitive than the one presented in [9], and that operates recursively on the quadtree data structure. Figure 6 shows the idea of a triangle strip for a RQT. We recursively circle counter-clock wise around the quadtree block centers and output every triangle once with alternating orientation. A triangle strip can also be regarded as a *space filling curve* visiting all triangles, or as a *Hamiltonian path* in the dual graph of the triangulation, see Figure 6 on the right.

FIGURE 6. RQT triangle strip



For each quadtree block, we can specify the corner-vertex where the triangle strip path enters to the right (inlet) and leaves the block on the left (outlet). Therewith, all blocks can be characterized by two rotation invariant path types shown in Figure 7, together with the hierarchical decomposition. To join the triangle strips of two quadtree blocks, the first and last edge of a path must be on the block border. Some triangulation rules for different block types are depicted in Figure 7 as well. With this triangulation scheme it is possible to create a triangle mesh in linear time.

FIGURE 7. Triangle strip decomposition



Theorem 7.1 *A triangle strip for the restricted quadtree Q can be constructed in time linear in the size of Q.*

Proof The quadtree traversal is limited by the one- and two-pass recursion schemes. Only the two-pass nodes are visited exactly twice, once for each inlet-outlet path. Therefore, the number of nodes processed is at most twice the number of nodes in the quadtree. Thus we visit O(|*Q*|) nodes.

Now, for each inlet-outlet path we have to show that the corresponding sequence of vertices can be created in constant time. Note that a quadtree block with only two triangles is triangulated in its ancestor block, see also Figures 2 and 7. Therefore, every visited block has a center-vertex called *c*. The other vertices are numbered counter-clock wise from the inlet = 0 to the outlet = *k*. Each triangle strip sequence of vertices for an inlet-outlet path consists of:

$$\{0, 1, c, [\forall i, 1 < i < k-1 : i, c], k-1, k\}$$

The size of such a sequence is limited by15 if all vertices are needed for triangulation. Therefore, each inlet-outlet triangle strip sequence can be generated in constant time. □

Besides the triangulation, also the construction of the restricted quadtree can be optimized. After calculating the approximation errors as described in Section 5, we maximize it on behalf of all vertices in the transitive closure:

$$e(P) = \max_{Q \in \zeta(P)}(e(Q)) . \qquad \textbf{(EQ 3)}$$

Consequently, dependencies are already encoded in the error values, and $(P, R) \in E_{dep}$ has not to be resolved because R is automatically selected if P is, since $e(R) \geq e(P)$ from Equation 3. Therefore, the construction of a restricted quadtree is simplified to the selection of all vertices satisfying the requested error threshold ε.

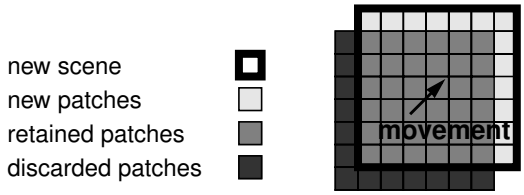
Theorem 7.2 *The set* $Q = \{P | e(P) \geq \varepsilon\}$ *is a restricted quadtree, given the error* $e(P)$ *is calculated as in Equation 1 and maximized according to Equation 3.*

Proof Assume that Q is not a restricted quadtree, then $\Rightarrow \exists P \in Q : (P, R) \in E_{dep} \wedge R \notin Q$ and $R \notin Q \Rightarrow e(R) < \varepsilon$. However, because of $P \in \zeta(R)$ and Equation 3 we have $e(R) \geq e(P)$. Because of $P \in Q \Rightarrow e(P) \geq \varepsilon$ we conclude that $e(R) \geq \varepsilon$ in contradiction to the assumption. □

8. Dynamic scene management

We use a *windowing* concept as shown in Figure 8 to visualize very large terrain databases, because we do not assume that the whole terrain database is in main memory. Furthermore, we do not restrict access to a per file basis as in [6,15,16]. Our visible scene always represents a window onto the world. A partition of the visible scene into *rectangular patches* as shown in Figure 8, efficiently supports scene updates. Furthermore, the database reloads can be composed from fixed sized range-queries.

FIGURE 8. Dynamic scene update



Besides supporting culling of invisible patches (Figure 16), this *scene map* can also be used to assign different levels of detail (LODs) to different patches (Figure 17). Refer to Section 9 for details on LOD handling. The scene map is not updated for every small variation of view position, view frustum coordinates or LOD distribution. However, an update occurs only when the variation exceeds a specific threshold. This approach of *deferred cumulative updates* helps to reduce the data management costs significantly without severe loss of display quality. Additionally, it provides further means to tailor the visualization system to available resources.

The dynamic scene manager maintains each terrain patch in a quadtree data structure. Whenever an update is scheduled, a RQT reflecting the new situation is extracted and provided for rendering the patch at the requested LOD. The restricted quadtree is also the basic transfer and query unit used for reloading or updating a patch from the terrain database.

9. Continuous LOD rendering

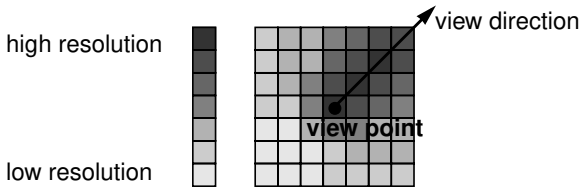
Continuous LOD stands for three different aspects of multiresolution visualization:

1. representation of a scene in an (almost) unlimited number of different LODs,
2. display different parts of a scene at different resolutions without discontinuities in between,
3. smooth changes between different LODs of the same scene.

The first aspect is handled by the restricted quadtree triangulation described in Section 3. Very important is that the different LODs do not have to be precomputed in advance and stored redundantly along with the original full-resolution model as it is the case in [15,16].

For the second aspect, an independent LOD is assigned to each terrain patch, using the object-space error metric of Section 5 that defines the approximation accuracy. The LOD is assigned based on relative distance to viewpoint and view direction, as shown in Figure 9, and demonstrated in a graphical example in Figure 15. After a terrain patch is loaded for the first time, LOD updates are performed incrementally. Note that unused vertices are never deleted until the complete patch is thrown out of the scene map.

FIGURE 9. LOD distribution



Avoiding discontinuity, between regions of different terrain complexity, is another requirement of aspect two of continuous LOD rendering. This is a hard problem for multiresolution triangulations [1,3], however, efficiently solved for grid-based inputs by the RQT. Nevertheless, neatly stitching together independent terrain patches of different resolutions is another problem. In [8] discontinuities were allowed between quadtree blocks, whereas [15,16] solved the problem by consistently triangulating the borders of all patches at the same resolution for all LODs. A matching triangulation between two patches can be achieved by mutually exchanging the missing border vertices, which can be quite expensive for sophisticated triangulations such as [3].

Also using the RQT, modification of the triangulation is not avoidable to match adjacent, triangulated regions but it is simple and efficient. The RQT T of a patch can be modified by insertion of the missing border vertices V_b. The new set of verti-

ces is then $V = V_T + V_b + V_{b_{dep}}$, consisting of the old vertices V_T, the missing border vertices V_b and the related vertices $V_{b_{dep}} = \{P | \exists R \in V_b : (R, P) \in E_{dep} \wedge P \notin V_T \wedge P \notin V_b\}$. Note that the vertices $V_{b_{dep}}$ can be interpolated without affecting the approximation precision of T, and V_b can only improve the accuracy. The related vertices $V_{b_{dep}}$ can be constructed or selected while traversing the quadtree Q for the insertion of V_b. V_b and $V_{b_{dep}}$ which guarantee a matching triangulation between two patches are marked to be included in the triangulation. Otherwise, they could be left out in the construction of the modified triangulation.

Aspect number three of continuous LOD rendering calls for smooth changes between LOD updates within the visible scene. In [7] the term *geomorph* was coined to denote morphing between different LODs of the same terrain segment. The RQT is well adapted to morphing. A new vertex v always lies in the middle of an edge between two vertices v_1 and v_2 further up in the dependency graph. Therefore, the interpolation can be calculated as $v' = 0.5(v_1 + v_2)$. The linear blending function is $f(v', v, t) = (1.0 - t)v' + tv$, with $t = 0.0..1.0$, and it is dependent from the distance to the observer. Morphing can be applied directly to the geometries used for rendering without retriangulation of the quadtree itself.

10. Storage and retrieval

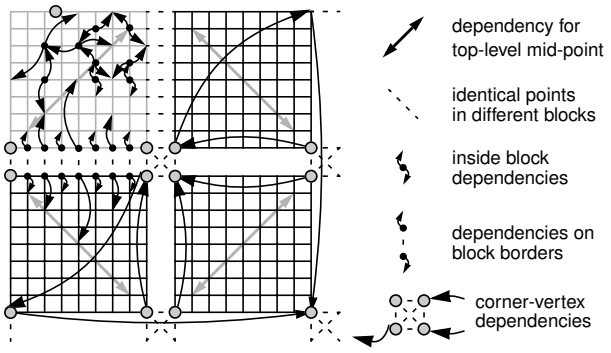
The scene management of Section 8 requires that the terrain database supports rectangular range queries with LOD restrictions, and returns a restricted quadtree of elevation points. Therefore, the database must conform to the same assignment of points to levels L_l, dependency graph G_{dep}, and relation of points to quadtree nodes. This is defined by the dependency direction for the top-level vertex in L_1^{center} of the root quadtree region.

Among all possible data structures for maintaining a grid digital elevation model we describe one here that fits best with Algorithm 2 of Section 4, and that provides fast access to the elevation data. The input data space is partitioned into blocks of $(2^k+1) \times (2^k+1)$ grid points. This partitioning provides efficient spatial selectivity and supports physical clustering on external storage. An implicit restricted quadtree is superimposed on every block as shown in Figure 10. Note that vertices *inside* of a block have dependencies within the same block, whereas dependencies of border vertices also symmetrically point to the adjacent block (*overlapping* dependencies). Finally, the corner-vertices' dependencies point to other corner-vertices of blocks as depicted in Figure 10.

Spatial access is performed by the following steps for a query with region R, error tolerance ε, and using the index structure S:

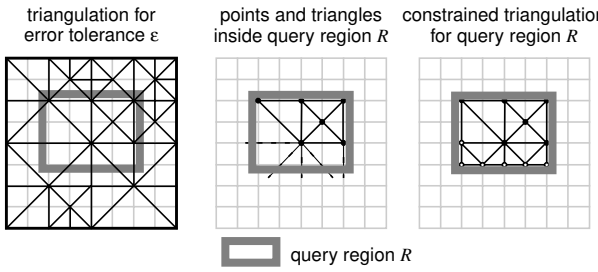
1. find all blocks b in S intersecting R using the spatial index: $B = \{b \in S | b \cap R \neq \emptyset\}$
2. apply bottom-up restricted quadtree construction algorithm to $\forall b \in B$,
   check each point against error threshold ε and intersection with R,
   keep track of *overlapping* dependencies as well
3. apply bottom-up or top-down restricted quadtree construction to the remaining corner-vertices

FIGURE 10. Height field partitioning



Special attention has to be paid to the border ∂R of the query region R. The border ∂R constrains the selection to include all vertices needed to map ∂R on edges of maximal length which conform to the RQT, see Figure 11. Note that an incremental query does not have to deal with the border problem because ∂R was already consistently triangulated by the first query of R.

FIGURE 11. Range query constraints



Furthermore, the transfer from the terrain database to the visualization is very efficient. The triangulation topology and the dependency graph are given implicitly by the restricted quadtree. Moreover, not even the spatial location must be transmitted because it is given by the point's position in the quadtree. The transmission is reduced to a traversal of the restricted quadtree where for every point only its height and error values are sent. Furthermore, dependencies can be resolved in the errors as shown in Equation 3, simplifying the triangulation for the visualization.

11. Experiments

This experimental section is not intended as a comparative study of available mesh compression methods, however, these experiments are meant to demonstrate the excellent compression potential of the restricted quadtree triangulation. Note that for this experimentation we used an object space error metric where the approximation error of a point on level L_l is defined as its vertical distance to the triangulation of its quadtree block on level L_{l-1}.

Table 1 shows the compression results of different error tolerance thresholds (0, 1, 2, 5 and 10 meters) for two topologically very different terrain samples. *Albis* represents a rather smooth hilly region and *Matterhorn* a very mountainous area. Both samples have an original grid resolution of 25 meters and an altitude accuracy of one meter. As can be seen from Table 1, already a very small error tolerance of one or two meters results

in a drastic reduction of the number of points used to approximate (triangulate) the surface.

TABLE 1. Compression results

	0 meter		1 meter		2 meter		5 meter		10 meter	
	points	*%*	*points*	*%*	*points*	*%*	*points*	*%*	*points*	*%*
Albis	25921	100	19773	76	11636	45	4664	18	1789	7
Matterhorn	25921	100	22492	87	17156	66	10492	40	6367	25

The visual fidelity of different error thresholds and compression ratios is depicted in Figure 12. Although the mesh complexity itself is reduced drastically, as seen in Figures 12-a, 12-b, 12-c and 12-d, the visual fidelity of a moderate error tolerance of 5 meters, see Figures 12-e and 12-f, is still extremely good. Even at a 10 meter error threshold with high compression ratios, significant visual details are retained, see Figures 12-g and 12-h. Furthermore, Gouraud shading and texturing improve the visual impression of the terrain, see also Figure 14, and allow the use of high error tolerances.

We also performed some rendering and scene update tests that are reported in Tables 2 and 3. The frame rates were measured using the *SGI Performer Toolkit*™ statistics, and our test scene was a hilly terrain, 52km × 52km large. The results were obtained on an SGI Inigo2 High Impact10000. From Table 2 it can be seen that scene culling according to the view-frustum (60° viewing angle) does not influence the rendering performance a lot because the non-visible triangles are already efficiently culled by the underlying graphics system. However, the incremental dynamic scene update benefits a lot from culling on the scene management level. Furthermore, rendering performance almost doubles when using different LODs because the adaptive RQT lowers the triangle count significantly, see Table 2. Figure 13 shows the test scene in full resolution and adaptively triangulated. Note that despite the massive reduction of the triangle count no significant loss of detail is observable.

TABLE 2. Frame rates

	number of triangles	frames per second
full resolution	86000	20
culled scene	20000	21
adaptive triangulation	8000	38

Our visible scene consisted of 13 × 13 patches, each 4km × 4km large and adaptively triangulated using the RQT, see also Figure 13 on the right. The dynamic scene update performance was tested with a velocity of 1000 meters per second, which is triple super-sonic and much faster than needed by any aircraft simulation. Our dynamic scene management updated 65 patches in 10 seconds in real-time. Of the 10 seconds only 1570 milliseconds[1] were used for all dynamic scene updates together. Note that this update includes loading 9922 vertices from the terrain database server over the network, constructing 65 restricted quadtrees, resolving map inconsistencies and generating a triangle strip for every updated patch. The average update time for one patch was only 24 ms and had to process 152 vertices for loading, quadtree construction, and triangle strip generation.

TABLE 3. Scene updates

65 patches in 10 seconds	CPU time	number of vertices
average	24 ms	153
total	1570 ms	9922

12. Conclusion

The RQT has several advantages over other triangulations. Regular grid triangulations are not adaptive to the terrain structure. Maintaining a triangulation for every LOD does not support continuous LOD rendering and the storage costs are very high. More sophisticated multiresolution triangulation models [2] replace a number of triangles by more and smaller triangles to refine the approximation, see also [1,13]. However, extraction and incremental refinement of a triangulation are rather complex for on-demand requirements. Furthermore, the topology representation is very complicated compared to the RQT. The same arguments hold for hierarchical triangulation models as described in [3]. Furthermore, no graphics optimizations are known. Therefore, from the requirements introduced in Section 1 number 4, 5 and 8 are not met. Moreover, the triangulation topology and hierarchy is complicated, and uses much more space than the restricted quadtree. Spatial access is not as efficient as well because topology and vertices have to be extracted from the database. Hence also the demands 1 and 2 cannot be fulfilled effectively.

The visualization systems in [8,9] do not consider the problem of very large scale terrains. Therefore, the problems 3, 6 and 9 of dynamic scene management and the database aspects 1 and 2 are ignored. The proposal of [15] does not focus on the dynamic scene and database aspects too. Furthermore, it is lacking effective solutions to the triangulation requirements 4, 5 and 8. In [6] no multiresolution approach is followed at all. In contrast to other triangulation and visualization models our approach fully satisfies al requirements listed in Section 1. In particular it supports many exceptional features in one multiresolution triangulation model that were also suggested in other publications:

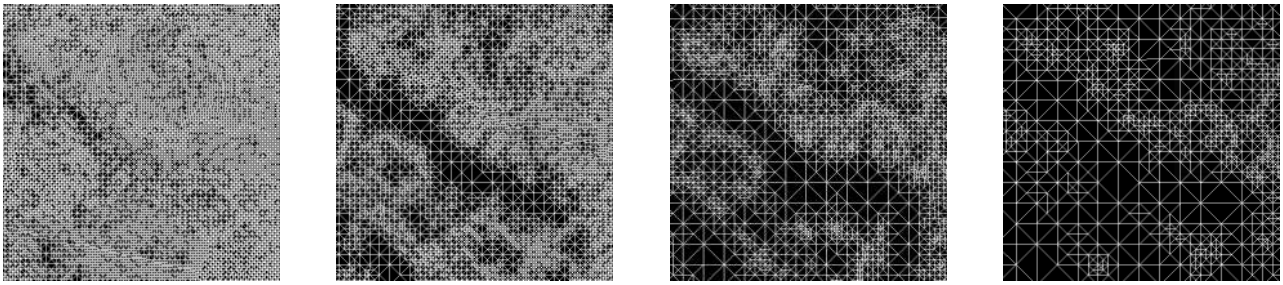
- progressive transmission [7]
- smooth, continuous LOD transitions [9]
- fast access and manipulation of triangulation [2,3]
- real-time triangulation [9,13]
- LOD constrained, spatial access [13]
- efficient storage costs, mesh compression [2, 7]

Most of the presented concepts and solutions are incorporated into the newest version of the ViRGIS[2] project [12]. ViRGIS mainly consists of a prototypical client-server system which allows interactive, three-dimensional visualization and exploration of large scale terrains, including texturing, see also Figure 14.

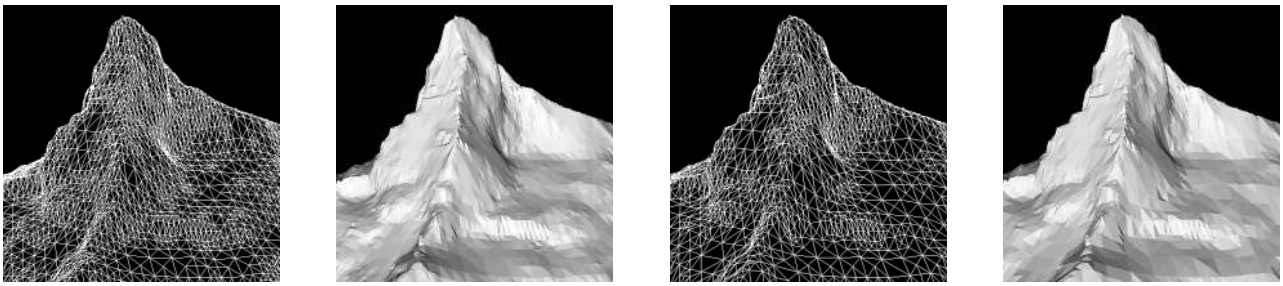
1. real-world time

2. ViRGIS - Virtual Reality and Geoinformation Systems http://www.inf.ethz.ch/personal/pajarola/virgis.html

FIGURE 12. Triangulated surfaces [DHM©]



a) 1 meter max. error, 76% b) 2 meter max. error, 45% c) 5 meter max. error, 18% d) 10 meter max. error, 7%



e) 5 meter max. error, 40% f) 5 meter max. error, 40% g) 10 meter max. error, 25% h) 10 meter max. error, 25%

Copyrights

References

[1] M. de Berg and K. Dobrindt. "On levels of detail in terrains". In *11th Symposium on Computational Geometry*, pages C26–C27. ACM, 1995.

[2] L. De Floriani, P. Marzano, and E. Puppo. "Multiresolution models for topographic surface description". *The Visual Computer*, 12(7):317–345, August 1996.

[3] L. De Floriani and E. Puppo. "Hierarchical triangulation for multiresolution surface description". *ACM Transactions on Graphics*, 14(4):363–411, 1995.

[4] D. H. Douglas and T. K. Peucker. "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". *The Canadian Cartographer*, 10(2):112–122, December 1973.

[5] 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 *Proceedings Visualization 97*, pages 81–88. IEEE, Computer Society Press, Los Alamitos, California, 1997.

[6] K. Graf, M. Sutter, J. Hagger, and D. Nüesch. "Computer graphics and remote sensing – a synthesis for environmental planning and civil engineering". In *Proceedings EUROGRAPHICS 94*, pages C13–C22. IEEE, 1994. also in Computer Graphics Forum Vol. 13, Nr. 3.

[7] H. Hoppe. "Progressive meshes". In *Proceedings SIGGRAPH 96*, pages 99–108. ACM SIGGRAPH, 1996.

[8] D. Koller, P. Lindstrom, W. Ribarsky, L. F. Hodges, N. Faust, and G. Turner. "Virtual GIS: A real-time 3D geographic information system". In *Proceedings Visualization 95*, pages 94–100. IEEE, Computer Society Press, Los Alamitos, California, 1995.

[9] 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. ACM SIGGRAPH, 1996.

[10] R. Pajarola. *Access to large scale Terrain and Image Databases in Geoinformation Systems*. PhD thesis, Dept. of Computer Science, ETH Zürich, 1998.

[11] R. Pajarola. "Large scale terrain visualization using the restricted quadtree triangulation". Technical Report 292, Dept. of Computer Science, ETH Zürich, 1998. ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/2xx/292.ps.

[12] R. Pajarola and P. Widmayer. "Virtual geoexploration: Concepts and design choices". *International Journal of Computational Geometry and Applications*, pages –, 1998.

[13] E. Puppo. "Variable resolution terrain surfaces". In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 202–210, 1996.

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

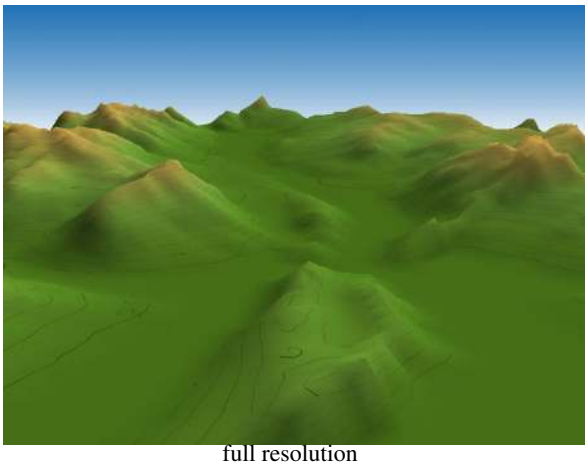
[15] M. Suter. *Aspekte der interaktiven real-time 3D-Landschaftsvisualisierung*. PhD thesis, Remote Sensing Laboratories, University of Zürich, 1997.

[16] M. Suter and D. Nüesch. "Automated generation of visual simulation databases using remote sensing and GIS". In *Proceedings Visualization 95*, pages 86–93. IEEE, Computer Society Press, Los Alamitos, California, 1995.

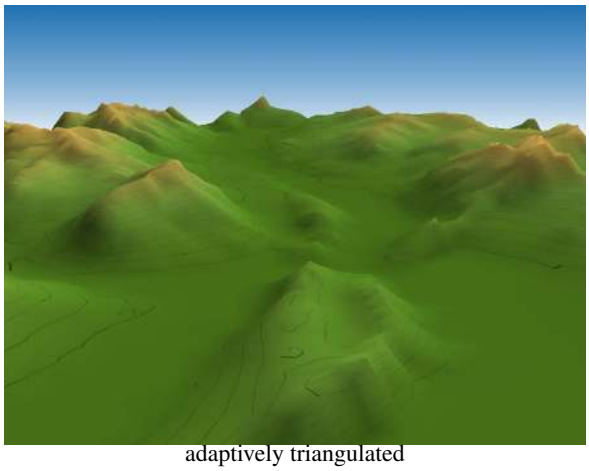
[17] B. Von Herzen and A. H. Barr. "Accurate triangulations of deformed, intersecting surfaces". In *Proceedings SIGGRAPH 87*, number 4 in ACM Journal Computer Graphics, pages 103–110. ACM SIGGRAPH, 1987.

Color plates

FIGURE 13. [DHM©]



full resolution



adaptively triangulated

FIGURE 14. Textured terrain visualization [DHM©], [SAT©]

FIGURE 16. Scene culling [DHM©]

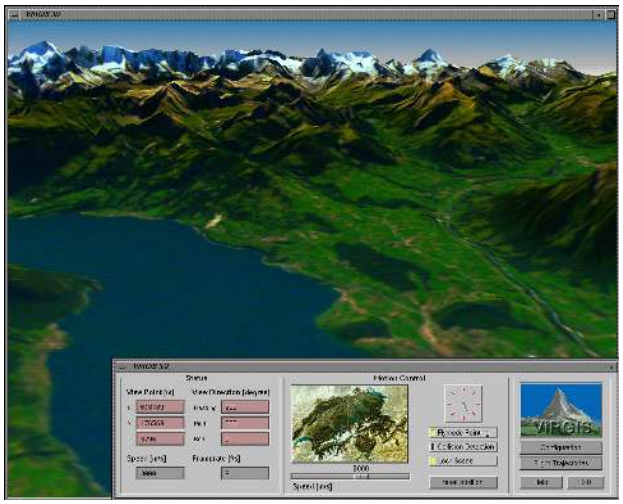


FIGURE 15. View-centered LOD strategy [DHM©]

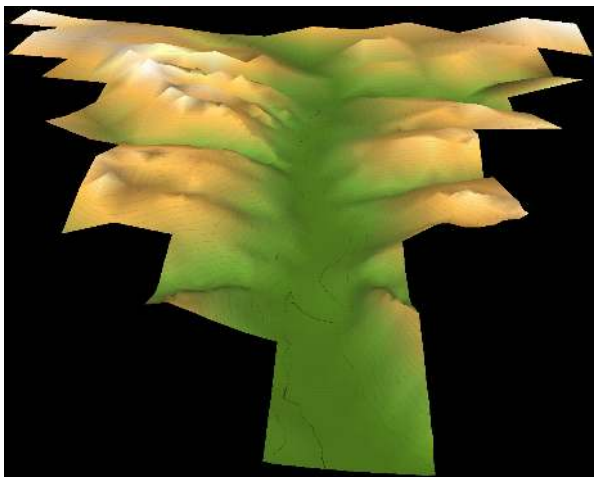


FIGURE 17. Scene map [DHM©]