Terrain rendering in games

Matej Mlejnek
matej.mlejnek@cg.tuwien.ac.at

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Vienna / Austria

Abstract

This paper describes terrain rendering algorithms and the state of the art in hardware-accelerated landscape engines. There are of course many terrain rendering research approaches and directions, but these are mostly either too new (that means, they are not implemented in any commercial engines yet) or their resource requirements are too high for today's computers. Therefore the paper focus on three methods which represent a good overview of terrain algorithms used by today's games and game engines, and finally discuss some games and game engines powered by them.

1 Introduction

Terrains, often called height fields, play an important role in new computer games as well as in the fast-growing domain of Geographic Information Systems (GIS). It is necessary to display different kinds of geographic based data sets on screen at interactive frame rates. Because of the inherent geometric complexity of terrains, this goal is often unachievable even with new generations of powerful graphics computers, unless the original height-field data is approximated in order to reduce the number of geometric primitives that need to be rendered without compromising visual quality: First, the terrain data cannot fit entirely in memory, so it needs to be loaded from disk. Second, the renderer must process a large number of small triangles corresponding to distant terrain.

2 Overview of terrain algorithms

A standart approach to triangle reduction is to construct multiresolution models representing the terrain. One posibility is to use discrete level-of-detail: the entire current resolution model is switched to a different resolution model based on the distance from the eye point, but the the switching is very noticeable, resulting in vertex popping. It is better to reduce triangles in a way to minimize visual impact. This is done by continuous level-of-detail. The terrain is changed by a small number of triangles at a time. Moreover, the change in the number of triangles is based on error measurements in screen space. The idea is that two triangles are reduced to one triangle if the height variation between the two triangles is smaller than a specified number of pixels.

Height data are typically represented in one of two types: grid or TIN. A grid is of the same form as the input array of height values. It is an array of height values at regularly spaced x and y coordinates. Since the sample points are regularly spaced,

it is possible to calculate the true (x,y) coordinates of a array entry from its array indices. Thus, the only storage needed is for the height values of the sample points. To obtain the height at (x,y) coordinates not in the sample set, one uses some form of interpolation.

The sample points of a TIN (Triangulated Irregular Network) are an arbitrary subset of the original data points. In order to specify this subset, the (x,y) coordinates (i.e. the indices in the input array) of each sample point in the TIN must be stored explicitly, in addition to the z-value. The sample points - without their height component - are triangulated (typically using a Delaunay triangulation) and linear interpolation is used to obtain the height at an arbitrary point (x,y) from the heights of the vertices of the triangle which contains the point.

Willis *et al.* [11] describe a hierarchical TIN data structure with for vertex morphing, along with a queue-driven top-down refinement procedure for building the triangle mesh for a frame. Specific effort is taken to avoid T-vertices. TINs allow variable spacing between vertices of the triangular mesh, approximating a surface at any desired level of accuracy with fewer polygons than other representations. However, the algorithms required to create TIN models are generally computationally expensive, prohibiting use of dynamically created TINs at interactive rates.

Lindstrom *et al.* [7] introduce a hierarchical quadtree technique. In order to reduce the projected screen space error, the height field is dynamically triangulated in a bottom-up fashion according to the distance to the point of view. Quad trees are very simple and efficient, sharing many of the design principles of the following algorithm (such as recursion).

Duchaineau *et al.* [1] present an algorithm (Real-time Optimally Adapting Meshes) based on a binary triangle tree structure. Here each patch is a simple right-isosceles triangle. Splitting the triangle from its apex to the middle of its hypotenuse produces two new right-isosceles triangles. The splitting is recursive and can be repeated on the children until the desired level-of-detail(LOD) is reached.

Hoppe [5] presents an algorithm based on Progressive Meshes, a relatively new technique for adding triangles to arbitrary meshes as you need more detail. With recent additions [6], this technique can also be applied to view-dependent triangulations, but requires complex data structures and therefore has high memory requirements.

2.1 Optimization criteria

There are various algorithms, all of which speed up rendering of terrain-data using various data structures and different optimization methods. Usually they are realized with respect to (all of) the following criteria:

- *Time required to achieve a given triangle count*:

  The algorithm's running time should be proportional to the number of triangle changes per frame, which can be only a few percent of the total mesh size.

- *Large reduction in the number of polygons to be rendered*:

  Typically, the surface grid is decimated by several orders of magnitude with no or little loss in image quality, accommodating interactive frame rates for smooth animation.

- *Flexibility in choosing view-dependent error metrics (user-specified)*:

The metric can be enhanced in many ways (e.g. ensuring correct visibility along specified lines of sight, providing correct terrain positions under objects, and eliminating back-facing detail). The parameterization allows for easy variation of the balance between rendering time and rendered image quality.

- *Terrain structure (Mesh representations, both pre-defined and run-time selected)*
- *Simplicity of algorithms*:

  The method should be simple to understand and implement.

- *Strict frame rates*:

  The algorithm should provide a guaranteed frame rate at the highest triangle counts that the graphics hardware can handle reliably.

- *Memory requirements*
- *Dynamic terrain*

  The preprocessed data should be updated quickly when the terrain is changed by mud slides, explosions, etc., and these dynamic changes to the geometry should be made with little computational cost.

- *Reduced popping artifacts*
- *Continuous changes between different surface LODs*

  The number and distribution of rendered polygons should change smoothly between frames, affording maintenance of consistent frame rates.

2.2 Real-time, continuous LOD-rendering of height fields

The algorithm by Lindstrom *et al.* [7] can be divided into two parts: a coarse-grained (block-based) simplification of the height field mesh geometry that is done to determine which discrete level-of-detail models are needed, followed by a fine-grained retriangulation of each LOD model in which individual vertices are considered for removal. The algorithm ensures that no errors are introduced in the coarse simplification beyond those that would be introduced if the fine-grained simplification were applied to the entire mesh. Both steps are executed for each rendered frame, and all evaluations involved in the simplification are done dynamically in real-time, based on the location of the viewpoint and the geometry of the height field. The surface corresponding to the height field before simplification is represented as a symmetric triangle mesh. The smallest mesh representable using this triangulation, the primitive mesh, has *3x3* vertices, and successively larger meshes are formed by grouping smaller meshes in a *2x2* array configuration (see figure 1). Therefore, for any level l in this recursive construction of the mesh, the vertex dimensions xdim and ydim are $2l+1$. Lower resolution blocks can be obtained by discarding every other row and column of four higher resolution blocks.

A block represents 8 triangles, 9 verices and 16 edges (figure 1a), all sharing the center point of the block. The simplification of the vertices in a block refers to reducing the number of triangles in the block by removing vertices that are unnecessary based on measurements of screen space heights. Typical terrain data sets have a large number of vertices. Performing vertex simplification on all vertices each frame is very expensive and will prevent real-time frame rates. Instead, groups of vertices can be simplified at once by analysing the screen space height of the bounding boxes of blocks that contain those vertices (discussed later).

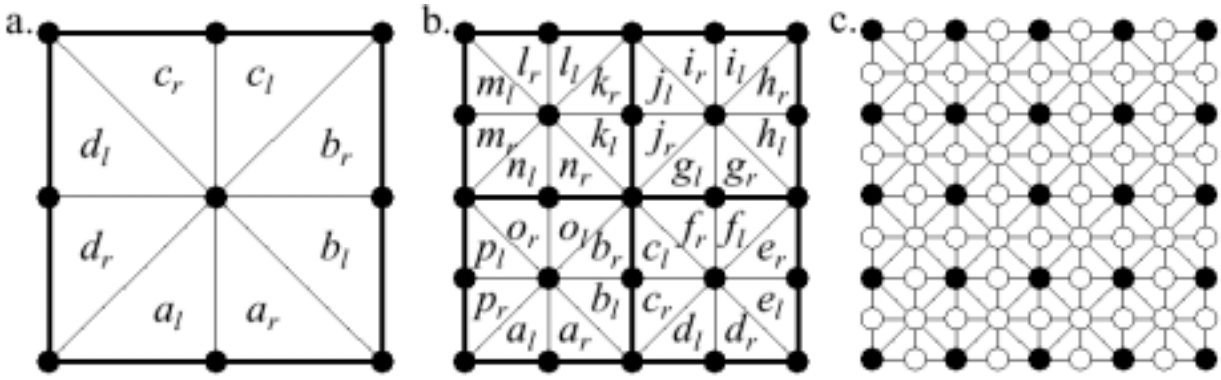

Figure 1: Triangulation of uniform height fields of (a) 3x3 and (b) 5x5 vertices. (c) Lowest level vertices (unfilled).

In the fine-grained (vertex-based) simplification step, many smaller triangles are removed and replaced with fewer larger triangles. Conceptually, at the beginning of each rendered frame, the entire height-field dataset at its highest resolution is considered. Wherever certain conditions are met, a triangle/co-triangle pair (with the same base edge) is reduced to one single triangle and the resulting triangle and its co-triangle (if one exists) are considered for further simplification in a recursive manner. Figure 1a and 1b illustrate the lowest level pairs. The conditions under which a triangle pair can be combined into a single triangle are primarily described by the amount of change in slope between the two triangles. If the error is smaller than a given threshold τ, the triangles may be fused. If the resulting triangle has a co-triangle with error smaller than the τ (1), this pair is considered for further simplification. This process is applied recursively until no further simplification of the mesh can be made. This scheme typically involves a reduction of an already simplified mesh, and the resulting errors are not defined with respect to the highest resolution mesh, but rather relative to the result of the previous iteration in the simplification process. However, empirical data indicates that the effects of this approximation are negligible. Whether the error is acceptable and the triangle may be fused, is decided by the followig equation:

$$\frac{\delta_v}{\|v-e\|}\sqrt{1-\left(\frac{(v-e)\cdot\widehat{n}_v}{\|v-e\|}\right)^2} < \kappa, \tag{1}$$

where e is the viewpoint, v the mesh vertex, $\widehat{n}_v$ its normal, δ_v its neighborhoods residual error, and $\kappa = 2\tau\, tan\,\frac{\Phi}{2}$ accounts for field-of-view angle Φ and pixel tolerance τ.

By obtaining a conservative estimate of whether certain groups of vertices can be eliminated in a block (block-based simplification), the mesh can often be decimated by several factors with little computational cost. If it is known that the maximum amount of change in slope of all lowest-level vertices in a block falls within a threshold τ, those vertices can immediately be discarded, and the block can be replaced with a lower resolution block, which in turn is considered for further simplification.

2.3 Progressive meshes

Hoppe [5] uses only a single operation for triangle mesh simplification: the edge collapse. In the usual case, shown in the figure 2, an edge collapse eliminates two triangles

from a mesh. Continuous level-of-detail primitives are created by starting with a triangle mesh and performing edge collapses until no more legal collapses exist. In choosing which edge to collapse, the simplification algorithm used to create continuous level-of-detail primitives always attempts to pick the edge whose collapse will have the least visual impact on the mesh. Each edge collapse creates an intermediate representation of the mesh which is one of the levels of detail at which the mesh can be rendered. The representation of the mesh that exists when no more legal edge collapses exist is referred to as the base mesh.

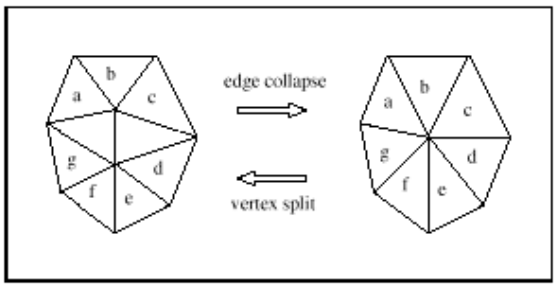

Figure 2: An edge collapse removes two triangles from a mesh.

The edge collapse is a reversible operation. Its inverse is called a vertex split. A vertex split undoes the edge collapse; it adds the same two faces back to the mesh. This reversibility allows rendering a continuous level-of-detail primitive in any of its intermediate representations. Stated more plainly, it is possible to render a version of the triangle mesh with any number of triangles (give or take one since edge collapses and vertex splits usually destroy or create two triangles) that lies between the number of triangles in the original mesh and the number of triangles in the base mesh.

One drawback of the described format for triangle meshes is that it does not contain an explicit representation of mesh discontinuities. For example, if a mesh has a sharp crease, vertices that lie on the crease should have different normals for triangles on different sides of the crease. The only way to accomplish this in the described mesh format is to duplicate such crease vertices; two vertices are created that have the same three-dimensional coordinates but different shading parameters. This creates a problem that simplification programs must address: Because triangles on different sides of the crease use different vertices, they are not actually connected to each other. A simplification program that does not address this problem will cause the triangles on different sides of the crease to tear away from each other as the simplification proceeds. The simplification part used to create continuous level-of-detail primitives should solve this problem by merging coincident vertices before simplification begins. Therefore, it should use a data structure that allows a vertex to have different shading parameters in different triangles.

Deciding which edges to collapse during mesh simplification is computationally expensive. Fortunately, this work can be done entirely in a preprocessing step. The calculation done at runtime just involves deciding how many triangles to render and performing the edge collapses or vertex splits necessary to get the model into the configuration that contains that many triangles. Edge collapses and vertex splits are simple, fast operations. Frame-to-frame to coherence usually causes the time required to perform these operations to be insignificant. If an object is moving gradually towards or away from the camera, the number of triangles used to render it in each frame will change gradually; not many vertex splits or edge collapses will need to be performed per frame.

The screen-space error metrics is derived from the refinement criterion used by Lindstrom *et al.* [7] (see also equation 1), where the denominator $\|v-e\|$ is an estimate of the z coordinate of the vertex v in screen space. This denominator is replaced with the linear functional $L_{e,\vec{e}}(v) = (v-e)\cdot\vec{e}$ which computes this z coordinate directly ($\vec{e}$ is the viewing direction). The screen-space error criterion is $\delta_v > \kappa\, L_{\acute{e},\vec{e}}(v)$, in which the point $\acute{e}$ is either the current viewpoint e or the anticipated future viewpoint e+*gtime* Δe depending on whether $\Delta e\cdot\vec{e}$ is negative or positive respectively (i.e. viewer moving backwards or forwards).

2.4 Real-time Optimally Adapting Meshes (ROAM)

The ROAM [1] algorithm uses a triangle bintree to handle the terrain. The root triangle is defined to be a right-isosceles triangle at the coarsest level of subdivision. The children of the root are defined by splitting the root along the edge formed from its apex vertex to the midpoint of its base edge. The rest of the triangle bintree is defined by recursively repeating the splitting process.

A set of bintree triangles forms a continuous mesh when any two triangles either overlap nowhere, at a common vertex, or at a common edge. Such continuous meshes are referred to as bintree triangulations. Figure 3 shows a typical neighborhood about a triangle T within a triangulation. A key fact about bintree triangulations is that neighbors are either from the same bintree level l as T, from the next finer level *l+1*, or from the next coarser level *l-1*. When T and T_B are both from the same level l, the pair ($T;T_B$) is referred to as a diamond. A simple split operation and its inverse, merge, are depicted in the figure 3 for a triangulation containing a diamond.

Split replaces triangle T with its children ($T_0;T_1$), and triangle T_B by its children ($T_{B0};T_{B1}$). This split operation introduces one new vertex at the diamond center, resulting in a new, continuous triangulation. If triangle T does not have a base neighbor T_B, only T is split into its children. Merging can be applied to diamond ($T;T_B$) when the children of T and T_B (if T_B exists) are all in the triangulation. In this case, ($T;T_B$) is a mergeable diamond for the triangulation. An important fact about the split and merge operations is that any triangulation may be obtained from any other triangulation by a sequence of splits and merges.

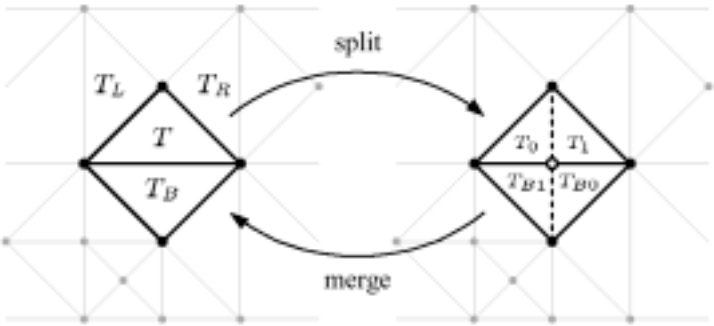

Figure 3: Split and merge operations on a bintree triangulation.

The split and merge operations provide a flexible framework for making fine-grained updates to a triangulation. No special efforts are needed to avoid cracks or T-vertices. The idea is simple: keep priorities for every triangle in the triangulation, starting with

the base triangulation, and repeatedly do a forced split of the highest-priority triangle. This process creates a sequence of triangulations that minimize the maximum priority (which is typically an error bound) at every step. The only requirement to ensure this optimality is that priorities should be monotonic, meaning a child's priority is not larger than its parent's. Adding a second priority queue for mergeable diamonds, allows the greedy algorithm to start from a previous optimal triangulation when the priorities have changed, and thus take advantage of frame-to-frame coherence.

2.5 Performance enhacements

The following optimization algorithms can be additionaly applied to any described method to refine their performance using the default metric:

- *backface detail reduction*:

  Priority can be set to minimum for triangles whose children are back-faced.

- *object positioning*:

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

- *atmospheric obscurance*:

  Priorities can be decreased when fog reduces visibility.

- *view frustum culling*:

  Because large parts of the terrain will not be visible (i.e., not inside the view frustum) from a certain camera point, they need not to be rendered and therefore should be culled away early to prevent unnecessary calculations.

- *progressive optimization*:

  The triangulation optimization has to be stopped when the frame time is about to expire in order to maintain a strict frame rate.

2.6 Comparison

A thorough comparison of terrain algorithms would require the implementation of them and compare them empirically. However, this would be rather time-consuming. Therefore, the algorithms are compared only theoretically. All three algorithms include a preprocessing component and a runtime component.

ROAM, Lindstrom and progressive meshes (PM) support vertex morphing. A PM elementary mesh operation, i.e. vertex split or edge collapse, involves two vertices. Therefore, PM is required to morph two vertices in such an operation. On the other hand, a ROAM and Lindstrom elementary mesh operation only involves a single vertex. Thus, vertex morphing is less computationally expensive for ROAM and Lindstrom.

A ROAM elementary mesh operation is easily checked for validity. A split operation for a triangle T is valid if T is neither a leaf or previously split. A merge operation for a triangle T is valid if it has previously been split. The positions of the resulting vertices from these operations are given from the height field and require no computations. A validity test for Lindstrom and PM elementary mesh operation requires more computations. A vertex split is a parametrized operation depending on the vertex activity.

The conditions under which a Lindstrom triangle pair can be combined into a single triangle are described by the amount of change in slopebetween the two triangles.

ROAM and Lindstrom automatically avoids slivers i.e. thin triangles since all triangles are right-isosceles. Slivers can introduce aliasing effects in texture maps, which reduce the visual quality of the terrain. The minimum angle in any triangle is $\Pi/4$ radians. PM have to make additional tests to avoid slivers. Therefore compactness of the triangle has to be computed. In addition, finding the compactness requires the computation of the lengths of the three triangle edges and the area of the triangle. Morphing two vertices during a PM elementary mesh operation introduces slivers temporarily in the mesh. This cannot be avoided and such an operation may result in aliasing artefacts.

Although all three algorithms exploit frame coherence, ROAM execution time is proportional to the number of triangle changes per frame, while Lindstrom and PM execution time is proportional to the full output mesh size.

The space of triangle meshes that can be produced by ROAM is only a subset of the space of Lindstrom and progressive meshes. ROAM produce optimal triangulation within the restricted space of triangulations, but it does not produce optimal triangulation in the space of all possible triangulations. Progressive meshes produce triangle meshes with 50-75% of the triangles produced by Lindstrom.

3 GAMES AND ENGINES

Only a few current and upcoming games give us a realistic perception of the natural beauty of the outdoors (Tribes 1, Tribes 2, Tread Marks, Outcast, Myth, ...). These games have taken 3D action gaming to the next level with the inclusion of incredibly detailed worlds upon which the story and action are played out. The next section describes two of them as well as the game (also terrain rendering) engines which power them.

3.1 The NetImmerse 3D Game Engine

NetImmerse is an object-oriented C++ software toolkit for creating real-time games and other interactive 3D content for Windows, Windows NT, Xbox, Gamecube, and PlayStation2 platforms. For typical game-development projects, the toolkit can save much development time.

Level-of-Detail: *NetImmerse* supports multiple levels of detail for any object in a scene, including direct import of Discreet 3DStudio MAX and MultiGen Creator levels of detail.

Continuous Level of Detail: *NetImmerse's* continuous level-of-detail primitive is based on Hoppe's [5] progressive mesh system. It allows the content developer to create one model at the highest level-of-detail and, at run-time, the object's mesh is scaled to an appropriate level. This level-of-detail is tuneable and the run-time processing is minimal. Numerical design limited white paper [8] fully describes this feature.

Terrain Structure: The terrain option creates the terrain skin during the game. The only input for the terrain is simple, regularly gridded elevation data such as DEM, which describes the topography of the terrain, and some information, which is geographically related, such as satellite images and Land Use Land Cover (LULC) data, which describes the type of vegetation. The DEM is a 2D array of uniformly

sampled height values of the planetary surface. Each value represents the measurement of height in meters relative to the sea level at that specific position. The satellite image is a collection of 2D array of uniformly sampled emission values from the planetary surface. The measurement is conducted at the specified infrared and visible frequencies. By connecting the points in the DEM, a polygon-based representation of the terrain is generated. In real-time, the terrain generator tessellates the database with continuous level-of-detail.

Figure 4: Screenshot from Dark age of Camelot.

3.2 Starsiege: Tribes 1,2

Starsiege *Tribes* [2] is an online-only game of fast-paced squad warfare. The game is played seamlessly between indoor and outdoor environments where terrain features are extremely strategic to the success of a mission. Long-distance hills are commonplace and enemies may hide behind hills to avoid detection.

Figure 5: Screenshot from Tribes 1.

Height Maps: The heights in *Tribes 1* are stored on a regular 8m square grid. *Tribes 2* grid size is selectable by mission.

The Engine: The *Tribes 1* terrain engine uses the Quad tree algorithm designed by Lindstrom [7]. For *Tribes 2*, a new approach to screen error based on edge traversal was implemented. This approach makes up for the limitations of currently published algorithms, including texturing with a bin-tree approach and seaming up edges between squares in quadtree algorithms.

Texturing has been the single biggest headache with landscapes in the game - allowing the mission editor to select a texture for every square and dynamically generate the combination textures for squares that are at a lower detail level, as well as automatically texturing the terrain based on vertex material. This engine is also not frame-coherent, requiring a rebuild of the Quad tree each frame. This is required for their view metric and clipping code.

References

[1] Mark Duchaineau, Murray Wolinksy, David E. Sigeti, Mark C. Miller, Charles Aldrich, Mark B. Mineev-Weinstein. ROAMing Terrain: Real-time Optimally Adapting Meshes. *In Visualization '97 Proceedings*, pages 81-88, 1997. `http://www.llnl.gov/graphics/ROAM`

[2] Dynamix Page, 2001. `http://www.Dynamix.com`.

[3] Gamasutra Features Archive, 2001. `http://www.Gamasutra.com`.

[4] GameDev Programming Archives, 2001. `http://www.GameDev.net`.

[5] Hugues Hoppe. Progressive meshes. *In SIGGRAPH '96 Proceedings*, pages 99-108, 1996. `http://www.research.microsoft.com/~hoppe`

[6] Hugues Hoppe. View-dependent refinement of progressive meshes. *In SIGGRAPH '97 Proceedings*, 1997. `http://www.research.microsoft.com/~hoppe`

[7] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A.Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. *In Proceedings of ACM SIGGRAPH 96*, pages 109-118, 1996. `http://www.cc.gatech.edu/gvu/people/peter.lindstrom/papers/siggraph96`

[8] Dave Eberly, Director of Engineering, Numerical Design, Ltd. The NetImmerse Terrain System, 2001. `http://www.ndl.com/terrainwhitepaper.html`.

[9] Outcast Engine Technology Presentation for GDC, 2000 `http://www.appeal.be/products/page1/Outcast_GDC/outcast_gdc_1.htm`.

[10] Outcast official page, 2001. `http://www.Outcast-thegame.com`.

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