CEIG'08, Barcelona, Sept. 3–5 (2008)
L. Matey and J. C. Torres (Editors)

An Efficient Restricted Quadtree Triangulation based on the Hilbert Space-filling Curve

Y. Valle[1]

[1]University of the Informatics Sciences, Cuba

Abstract

Digital Terrain Models play an important role in a variety of applications such as games, virtual simulations and geographic information systems. Most of terrain visualization techniques are based on multiresolution models that reduce the number of rendered primitives using different levels of detail. In fact, the main problems are the big number of primitives to visualize and the transmission of large terrain datasets across the networks. The main contributions of this paper are focused on the first of this problems: an efficient implementation of a quadtree data structure for in-memory representation of terrain models is proposed, and a triangle strip generation algorithm based on the Hilbert space-filling curve for the fast rendering of the final multiresolution mesh.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Curve, surface, solid, and object representations

1. Introduction

The problem of mesh simplification and multiresolution surface triangulation has been widely studied over the last two decades. A number of approaches are based on the principle of Delaunay triangulation [vK97] to create Triangulated Irregular Networks (TINs) over irregularly spaced sets of points (see also [FMP96]). Another important contribution belongs to [Hop96] in which the progressive mesh representation is introduced, a new scheme for storing and transmitting arbitrary triangle meshes.

The most commonly used models over regular grids of points are based on quadtrees and binary triangle trees. Right-Triangulated Irregular Networks (RTIN) is presented in [EKT01]. Like [DWS*97], which uses a priority-queue driven triangulation based on the notion of longest side bisection, RTIN is based on the same binary triangle tree hierarchy, but mainly focused in the efficient representation of the hierarchy, and the fast traversal of the structure for neighbor finding.

The main contributions on Restricted Quadtree Triangulations (RQT) belongs to Lindstrom et al. [LKR*96] which introduces the vertex dependencies graph that can be used to prevent cracks and create triangulations at variable resolution, and to Pajarola [Paj98] which uses the dependencies relations presented in [LKR*96] to generate minimally triangulations over implicit region quadtrees, and reduces the storage cost effectively using both arithmetic an logical operations on large height-field grids. Another significant solution to quadtree based terrain rendering is proposed in [RHS98], which is mainly focused in a memory efficient representation of the quadtree data structure.

This paper addresses the problem of real-time visualization at different level of details of terrain datasets that entirely fit into the main memory. An efficient representation of a model based on a quadtree is proposed. The spatial data structure is implemented as a hierarchical Hilbert space-filling curve [ARR*97], which is the base of the technique proposed for fast triangle strip generation.

2. Quadtree construction

The Hilbert space-filling curve, as shown in Fig 1, it is widely considered to be tricky to work with. In this section we will show that this curve is very useful in quadtree-based terrain modeling and fast triangle strip generation.

In this case, the quadtree is implemented using index operations and recursion instead of pointers and nodes like

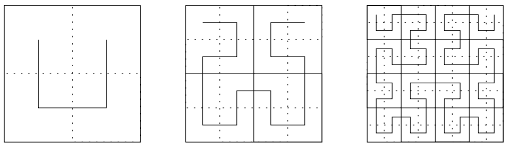

Figure 1: *Hilbert space-filling curves at levels* 1, 2 *and* 3.

in [Paj98]. In the proposed data structure, Fig 2, there is no need to store information about children, parents and neighbor squares.

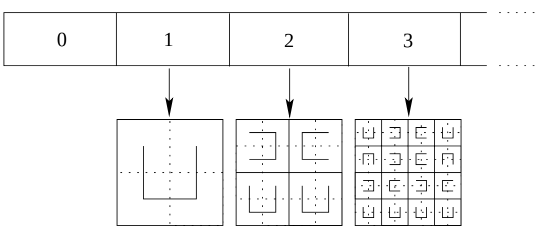

Figure 2: *The first three levels of the quadtree data structure.*

The starting point is a $(n \times n)$ regular grid, where $n = 2^k + 1, k \geq 1$, representing de height field (with corners at $(0,0)$, $(0,2^k)$, $(2^k,0)$, $(2^k,2^k)$), that is recursively subdivided at each step of the algorithm. Each quadtree node is represented as a segment of the Hilbert space-filling curve ("Hilbert pattern" from now on), Fig 2.

2.1. The quadtree data structure

As shown in Fig 2, the quadtree is stored in a one dimensional array that at each position stores a bidimensional array of Hilbert patterns. Each index in the one dimensional array represents a level (from 1 to number of levels, position 0 remains empty) in the quadtree, and each matrix stores all the nodes in corresponding level. The algorithm 1 shows how the recursive subdivision is achieved.

The CreateQuadtree procedure starts calculating the number of levels in order to create the one dimensional array. Lines 4 to 6 set the corresponding bidimensional array to each level. After that, the starting Hilbert pattern, Fig 1, is created and stored in the (1×1) matrix at level 1, besides the direction to follow by the space-filling curve for further triangle strip creation(ie. the direction of the north-west Hilbert pattern in Fig 1 at level 2 is South). Both attributes, the Hilbert pattern orientation and the direction, can be represented with values from 0 to 3, needing only 2 bits per attribute. Finally, at line 8, the recursive subdivision is started.

The DownSplit procedure takes four parameters: the size of the squares that will be created in the next level, and the grid row, grid column, and level of the father square. In

Algorithm 1 Quadtree creation

```
1:  procedure CreateQuadtree
2:      $levels \leftarrow \lfloor \log_2 n \rfloor$
3:      $qt \leftarrow array[levels+1]$
4:      for $i \leftarrow 1, levels$ do
5:          $qt[i] \leftarrow array[2^{i-1}, 2^{i-1}]$
6:      end for
7:      $qt[1][0,0] \leftarrow Data(Down, East)$
8:      DownSplit($\lfloor n/2 \rfloor, \lfloor n/2 \rfloor, \lfloor n/2 \rfloor, 1$)
9:  end procedure

10: procedure DownSplit(size, r, c, level)
11:     if $\lfloor size/2 \rfloor \neq 0$ then
12:         $r_{nw} \leftarrow r - size/2$
13:         $c_{nw} \leftarrow c - size/2$
14:         $r_m \leftarrow \lfloor r_{nw}/size \rfloor$
15:         $c_m \leftarrow \lfloor c_{nw}/size \rfloor$
16:         $qt[level+1][r_m, c_m] \leftarrow Data(Right, South)$
17:         RightSplit($size/2, r_{nw}, c_{nw}, level+1$)
            ...
18:         $qt[level+1][r_m, c_m] \leftarrow Data(Down, East)$
19:         DownSplit($size/2, r_{sw}, c_{sw}, level+1$)
            ...
20:         $qt[level+1][r_m, c_m] \leftarrow Data(Down, North)$
21:         DownSplit($size/2, r_{se}, c_{se}, level+1$)
            ...
22:         $qt[level+1][r_m, c_m] \leftarrow Data(Left, Father.Dir)$
23:         LeftSplit($size/2, r_{ne}, c_{ne}, level+1$)
24:     end if
25: end procedure
```

this procedure, a Down Hilbert pattern is subdivided in four patterns that are stored in the next level, Fig 2. Before to execute the procedures at lines 19, 21 and 23, the values of the row and column in the grid and in the corresponding matrix for de south-west, south-east and north-west children have to be calculated. The calculation is done similar to lines 12 to 15: $[r_{sw} = r + size/2, c_{sw} = c_{nw}]$, $[r_{se} = r_{sw}, c_{se} = c + size/2]$, and $[r_{ne} = r_{nw}, c_{ne} = c_{se}]$. The r_m and c_m values are updated for each child. Procedures RightSplit, LeftSplit and UpSplit are very similar to DownSplit.

An important advantage of this quadtree representation is that each Hilbert pattern represents the new point introduced at each step of the recursive subdivision, that's why the information of the smallest squares is not explicitly stored in the structure. Given the position of a Hilbert pattern in the structure, the corresponding information in the height field (x,y coordinates and the height value) can be retrieved by simple mathematical operations. In a similar way the corners of the four quadrants represented by each Hilbert pattern can be found.

The procedure GetSWChild in the algorithm 2 returns the south-west child for the Hilbert pattern at the valid position $[level][r_m, c_m]$. In a similar way, references to the neighbors and father nodes of a given Hilbert pattern are calculated using simple mathematical operations, and there is no need to store that information in the quadtree data structure.

Algorithm 2 South-west child

```
1: procedure GETSWCHILD($level, r_m, c_m$)
2:     return $quadtree[level+1][r_m*2+1, c_m*2]$
3: end procedure
```

3. Fast triangle strip generation

The speed at which triangulated surfaces can be displayed is decisive in almost all scientific visualization techniques (see [AHMS94]). Triangle strip generation based on space-filling curves has been widely used to generate fast an efficient triangulations over hierarchical data structures (see [Paj98], [VdFG99]). The technique proposed in this section is based on the Hilbert space-filling curve, and the information needed for its construction is stored explicitly in the model.

In the algorithm 1, once each Hilbert pattern is created, an object space error like in [POF04] is calculated (ie. after line 17) and stored for further extraction of the multiresolution triangulation. Due to the fact that our recursive subdivision starts with a down-oriented Hilbert pattern, the beginning of the space-filling curve is in the north-west region, while the end is at the north-east area. Following the curve, for each node in the structure the vertices are added to the triangle strip according to some rules, Fig 3.

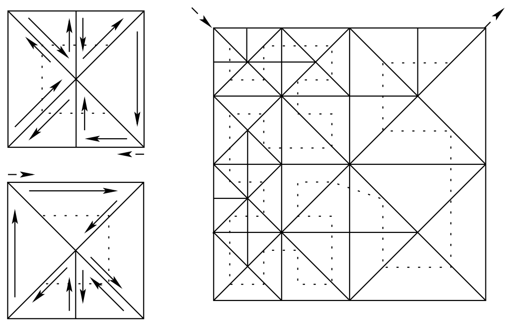

Figure 3: *Left: Triangulation of left and right oriented Hilbert patterns. Right: A triangulated surface created following the space-filling curve (dotted lines).*

Converting a general polygon mesh to a single triangle strip is generally not possible. In order to describe a complete object, either several strips has to be created, or a degenerated strip (see [Paj98]) as in Fig 3, which contains zero-area triangles that the processing software or hardware will discard. Fig 4 shows a rendered landscape with the superimposed Hilbert curve.

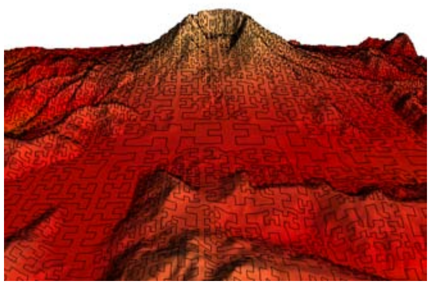

Figure 4: *Rendered sample landscapes: Fragment of the original Puget Sound height field.*

The quadrants to be rendered are selected in a similar way to [RHS98] guaranteeing that the level difference of adjacent blocks is less than or equal to one. The starting point of the space-filling curve is found by recursively descending the north-west region of the quadtree until a marked(indivisible) quadrant is reached. Then, following the directions stored in the structure, each Hilbert pattern is triangulated in a continues way.

4. Results

In order to show the effectiveness of the proposed model for fast an efficient terrain rendering, some screen shoots were taken from the application running in a simple Toshiba Satellite L20-273 with a 1.40 GHz Intel Celeron M360 processor and a 64 MB ATI RADEON XPRESS 200M Series (0x5A62) graphics card. For visualization purposes, we used an object space error metric that is defined in terms of the vertical distance between points in quadrants at different levels, and sample data that describes the Great Canyon and Puget Sound areas, USA, with the elevation data artificially scaled in order to exaggerate the elevation changes.

Table 1 shows the number of points included in the triangle strip for a 1025×1025 fragment of the original Great Canyon height field with different error threshold tolerance. Instead of $3N$ points needed to draw N triangles, about the 80% are used by the triangle strip (TS).

Table 1: *Points in the triangle strip*

Error	0 meters	2 meters	5 meters
Triangles	524288	503763	260529
Points in TS	1310720	1237622	633832

Fig 5 shows a sample landscape rendered using our algorithm and the superimposed triangulation generated following the corresponding Hilbert space-filling curve. In this simple example, the levels of details clearly depends only on surface roughness. In a future work, we will consider the distance of the point of view for the generation of multiple levels of details in order to decrease the number of rendered primitives. Moreover, we will compare our proposal with other approaches in order to evaluate its effectiveness.

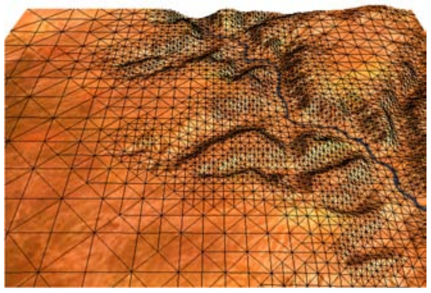

Figure 5: *Rendered sample landscapes: Fragment of the original Great Canyon height field.*

Like some other works (see [EKT01] and [RHS98]), the main contribution of this proposal is focused on the efficiency in memory representation of the model. Let M a $(n \times n)$ regular grid, where $n = 2^k + 1, k \geq 1$, representing a height field. A classical quadtree (represented using nodes and pointers) built over M have $k+1$ levels and

$$Q = \sum_{i=0}^{k} 2^i \tag{1}$$

nodes. Besides at least two bytes per height value in the regular grid, for each node in the quadtree information about error, descendants, parent and neighbors have to be stored. That's why the size of each node q is at least

$$S_n = s_p(d_q + v_q + f_q + e_q) \tag{2}$$

with s_p the size of a pointer to a node, d_q, v_q, f_q and e_q the number of descendants, the number of neighbors, the father and the associated error respectively.

Due to the fact that in our quadtree representation each Hilbert pattern represents four quadrants, we only need k levels. Therefore, the number of Hilbert patterns stored is

$$Q = \sum_{i=1}^{k} 2^{i-1} \tag{3}$$

Besides at least two bytes per height value, for each Hilbert pattern only four bytes per error and four additional bits, two for the kind of Hilbert pattern and two for the direction to follow in the triangle strip generation, are needed. Calculations done for some regular grids shows that our representation takes about the 6% of the total memory needed by a classical representation using nodes and pointers.

References

[AHMS94] Arkin E. M., Held M., Mitchell J. S. B., Skiena S.: Hamiltonian triangulations for fast rendering. In *ESA '94: Proceedings of the Second Annual European Symposium on Algorithms* (London, UK, 1994), Springer-Verlag, pp. 36–47.

[ARR*97] Asano T., Ranjan D., Roos T., Welzl E., Widmayer P.: Space-filling curves and their use in the design of geometric data structures. *Theoretical Computer Science 181*, 1 (1997), 3–15.

[DWS*97] Duchaineau M. A., Wolinsky M., Sigeti D. E., Miller M. C., Aldrich C., Mineev-Weinstein M. B.: ROAMing terrain: real-time optimally adapting meshes. In *IEEE Visualization '97* (1997), pp. 81–88.

[EKT01] Evans W. S., Kirkpatrick D. G., Townsend G.: Right-triangulated irregular networks. *Algorithmica 30*, 2 (2001), 264–286.

[FMP96] Floriani L. D., Marzano P., Puppo E.: Multiresolution models for topographic surface description. *The Visual Computer 12*, 7 (1996), 317–345.

[Hop96] Hoppe H.: Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 99–108.

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

[Paj98] Pajarola R. B.: Large scale terrain visualization using the restricted quadtree triangulation. In *IEEE Visualization '98* (1998), Ebert D., Hagen H., Rushmeier H., (Eds.), pp. 19–26.

[POF04] Perez M., Olanda R., Fernandez M.: Visualization of large terrain using non-restricted quadtree triangulations. In *Computational Science and Its Applications Ű ICCSA* (2004), Springer Berlin H., (Ed.), pp. 671–681.

[RHS98] Roettger S., Heidrich W., Slussallek P.: Real-time generation of continuous levels of detail for height fields. In *Proceedings of 6th International Conference in Central Europe on Computer Graphics and Visualization* (1998), pp. 315–322.

[VdFG99] Velho L., de Figueiredo L. H., Gomes J.: Hierarchical generalized triangle strips. *The Visual Computer 15*, 1 (1999), 21–35.

[vK97] van Kreveld M. J.: Algorithms for triangulated terrains. In *Conference on Current Trends in Theory and Practice of Informatics* (1997), pp. 19–36.