Quasi-regular Tessellation of Hexagons

Branko Kaučič[1] and Borut Žalik[2]

[1] PEF, University of Maribor, Slovenia
[2] FERI, University of Maribor, Slovenia

An algorithm for quasi-regular hexagon tessellation of uniformly distributed points is presented. At first, the needed definitions and notations are introduced. Then, the algorithm for the tessellation, based on "laying-down the sticks" analogy, is given. At the end, the estimation of the algorithm time complexity is done.

Keywords: tessellation, gridded spatial data, computational geometry

1. Introduction

A tessellation is a process that surrounds us every day. It is an old problem in mathematics and in computational geometry (Preparata, 1985). Several tessellations are known, although only some of them are interesting in the computational geometry. The main aim of the tessellation is to tile (or divide) the space of the interest into smaller parts (tiles) upon which other problems could be solved more easily (e.g. geometric search, visualising only parts of the interest).

In the paper, a new algorithm for the tessellation of hexagons is presented. We believe that tessellation of hexagons is more efficient than other tessellations for some applications on digital elevation models (DEMs), as for example a fly-through problem. We will restrict ourselves to quasi-regular tessellation only.

The paper is divided into five sections. At the beginning, a tessellation of hexagons is explained. The algorithm of the tessellation is given in the next section, together with the details of the input data. In the fourth section, the idea of obtaining optimal tessellation and the worst case time complexity of the algorithm is discussed. In the last, fifth section, the paper is summarised.

2. Tessellation

In general, the tessellation is divided into the following groups:

- **a regular tessellation**, where the sides of tiles are all of the same length. Only three possible shapes of tiles exist: triangles, squares, and hexagons, where triangular and squared tiles are usually used in practice.
- **a quasi-regular tessellation**, where all tiles are of the same shape, but the lengths of the sides are not equal.
- **a semi-regular tessellation**, where several regular shapes are combined together.
- **a non-regular tessellation**, where the tiles have different lengths of their sides, and are differently shaped. The best known tessellation in practice are Voronoi diagrams (Mulmuley, 1994; Preparata, 1985), although in theory different ideas exist (Armstrong, 1988; Farmer, 1996; Wells, 1991).

Tessellations of squares are simple, because there is no need to apply any rotation to completely cover the plane. On the other side, tessellations of triangles produce gaps if they are not rotated. The same is true for hexagons.

2.1. Tessellation of Hexagons

As stated before, the paper considers a quasi-regular tessellation of hexagons. The tessellation of hexagons has some advantages against

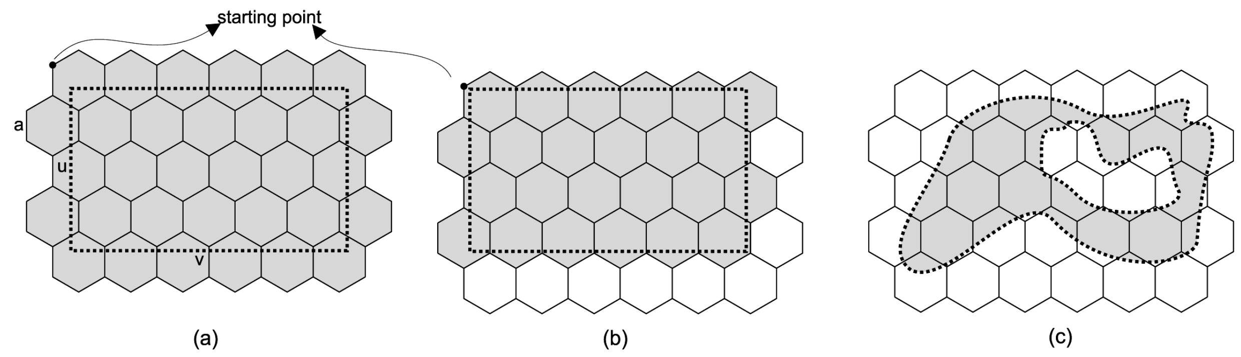

Fig. 1. Examples of tessellations.

other two regular tessellations (Kaučič, 2001a). Each vertex of the tile has the smallest number of neighbouring tiles, each tile has the smallest number of neighbouring tiles, the neighbourhood of the tile is determined only with edges of the tile and not with the combination of vertices and edges, and the number of tiles in the tessellation is the smallest under certain conditions.

Further, at the tessellation of hexagons, the following interesting questions occur: With the given size of a hexagon, how an arbitrary area can be tessellated? What is the minimum number of tiles? How must the area be tessellated to reach the minimum? How many tiles are entirely in the area, and how many of them cover the area only partially? With a given point in the tessellated area, which cell contains it? In general, answers to these questions (especially to the second and third) present a hard problem. To illustrate the difficulty of given questions and the need for optimal tessellation, Fig. 1 shows two examples of tessellations of an arbitrary area.

The left side of the figure (a) shows the tessellation of the rectangular area with one of many optimal solutions (b). An optimal solution (position of the starting point) can be computed in a constant time. On the right side (c), more complex example with a hole is given. An optimal solution cannot be found so simply as in previous example.

So far, nothing was said about the representation of an area. In the continuation, only points arranged in the regular grid are considered. It turned out that the regular tessellation does not give the optimal results in some applications (e.g. fly-through problem). More suitable are quasi-regular hexagons, where some points of the regular grid lie on the border of the tile (Fig. 2 (a)).

Points located on edges, splitting two neighbouring tiles, are doubled in both tiles. In this manner, the direct connections between border points of neighbouring tiles exist (Kaučič, 2001a). Fig. 2 (b) shows how the dimension of the quasi-regular hexagon tile is defined. The values a and b can take any positive integer number, where a represents the number of points in upright side, and b is the number of points on the slanted side. However, to get the best approximation of the regular tessellation, b is determined by equation $b = ROUND(\frac{a}{\sqrt{2}})$.

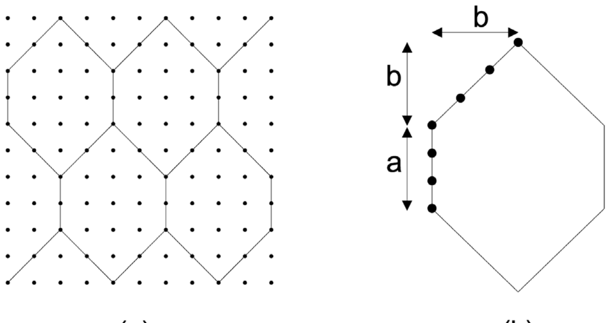

Fig. 2. (a) quasi-regular hexagons, (b) the hexagon tile.

3. Algorithm of the Tessellation

The motivation for the presented tessellation algorithm was a database of terrain elevations. Regular grids are usually in the form of a rectangle, while in our case the border of the regular grid represents the border of the land, which is

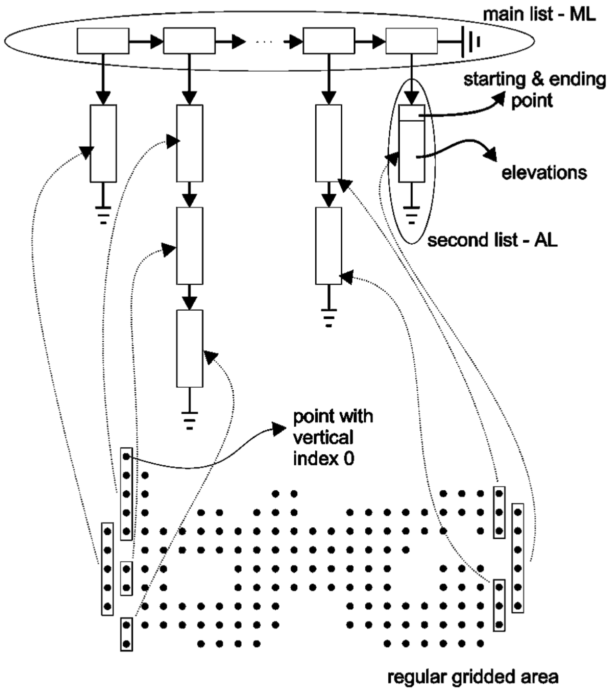

Fig. 3. (a) Input data structure.

far from being a rectangle. The data was then converted into a more appropriate form. Data structure consisting of a list of lists as shown on Fig. 3 (a) is used.

The "main list" (ML) stores pointers to the "lists of arrays" (ALs) of data. At the second level, AL contains all points of the corresponding column. Each element of AL represents a connected set of points and consists of a starting point, an ending point, and elevations. The starting point represents a vertical index of the first point in the connected set of points, and the ending point represents a vertical index of the last point in the same set of points. The most upper points in the area have their vertical index set to 0, and empty columns that represent a region without points have NULL pointers in ML.

The algorithm for the tessellation of hexagons works with a sweep traversal over a tessellated area and in each step behaves as if the sticks were being laid-down. Each sweep step is associated with a column in the input data structure. For each column, the algorithm lays-down the sticks of appropriate length under each other until the end of points in the current column is reached.

Let us observe Fig. 3 (b). Suppose, there are enough points in the first column to lay-down at least two consecutive tiles. The first starting point in the first AL (marked with an arrow) is taken as the starting point of the tessellation. The upper stick used is shorter than the lower stick (we name them "short stick" (S|s) and

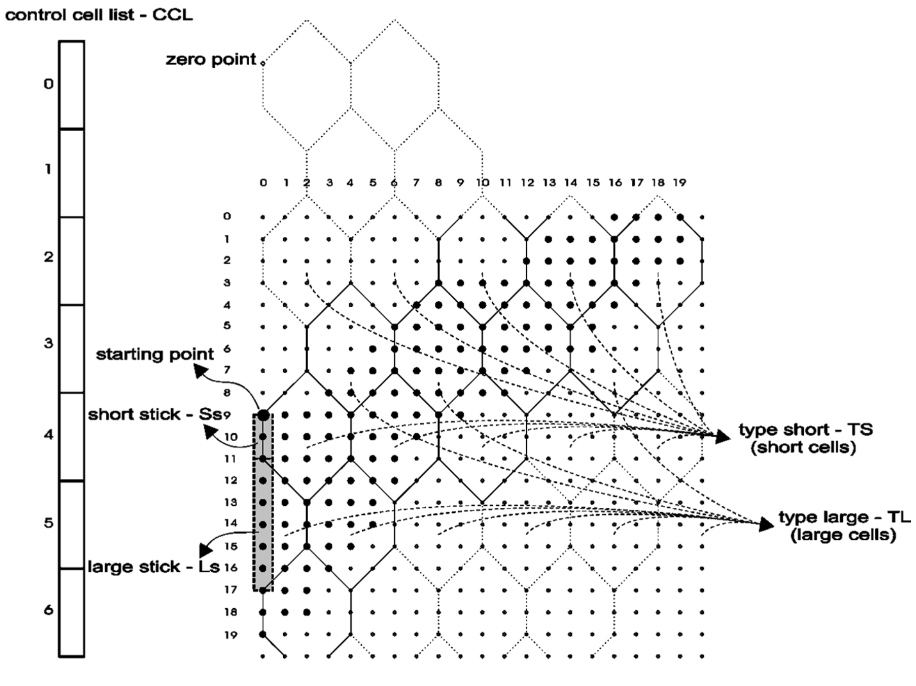

Fig. 3. (b) Division of cells on short and large.

"large stick" (Ls)). Accordingly to that, are the hexagons in the same row of type short (TS), or type large (TL). An additional array (Fig. 3 (b)) is used to control the tessellation process, named "control cell list" (C|CL). It contains identifiers of tiles and the information about the row type (TS or TL). All even entries are of type TS and the odd entries are of type TN.

At each step of the algorithm, S|s and Ls have to be updated. The update depends on the state of the tessellation process. Four states are possible (Fig. 4):

- **left side**, where all entries in C|CL, representing the cells of TS are removed. Some of them are replaced with new identifiers in the next step.
- **slanted-up**, where the length of S|s increases by 2, and the length of Ls decreases by the same value. In this state, the tessellation process stays at most $b-1$ times, except if the end of ML is reached.
- **peak** is similar to the first state, but this time it is applied on the cells of TL.
- **slanted-down**, where the opposite actions of the state "slanted-up" are performed.

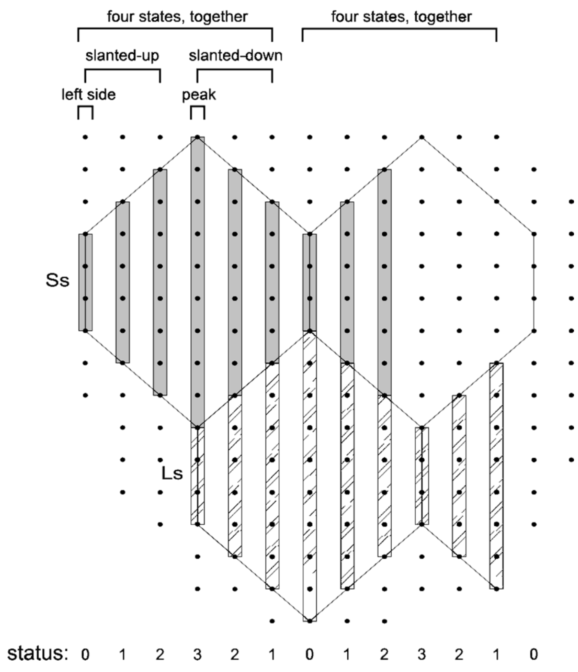

Fig. 4. Four states of the tessellation process.

Within those states, an additional value *status* is updated. As shown in Fig. 4, it starts with value 0 and increases by 1 in state "slanted-up", while in the state "slanted-down" it decreases by 1.

```
LDS ALGORITHM:
  input : a, b, ML
{
 //initialization
 COMPUTE(zero, C|CL, S|s, Ls, status);
 while not end of ML
  //slanted-up
  while status < $b-1$ and not end of ML
   COMPUTE(index, boundary, type);
   if start = boundary then
    if CONTAINS(C|CL, $index-1$) then
     UPDATE(C|CL,new identifier);
    endif
   endif
   if CONTAINS(C|CL,index) then
    UPDATE(C|CL,new identifier);
   endif
   //laying-down the sticks
   COMPUTE(first stick);
   while bottom not reac|hed
    case type of
     TS: lay-down current stick within S|s;
       COMPUTE(next stick);
       if CONTAINS(C|CL, $index+1$) then
        UPDATE(C|CL,new identifier);
       endif
     LS: // ...similar to TS
    COMPUTE(type, index);
   endwhile
   COMPUTE(S|s, Ls, status);
  endwhile
  //peak
  if not end of ML then
   REMOVE(C|CL,cells of TL);
  endif
  //slanted-down
   ...similar to slanted-up
  //left side
   ...similar to peak
 endwhile
}
```

Fig. 5. A pseudocode of the LDS algorithm.

At each step of the algorithm, three things have to be determined. For each starting point in AL it must be known to which C|CL entry it belongs (*index*), which is the nearest upper boundary point of the hexagon containing this starting point (*boundary*), and how long the first stick is. Then, at each step for each AL element the algorithm lays-down the sticks in the following way. The length of the first stick is calculated first. Then, the starting point is checked if it lies on the boundary of two cells. If true and necessary, C|CL is updated. The sticks of lengths S|s and Ls are then laid-down until the ending point is reached. At each stick, the entry index in C|CL is computed and checked if C|CL already contains the tile (its identifier). If true, the tile already contains some points from previous steps. If false, the identifier of a new tile is added in C|CL. The pseudocode in Fig. 5 illustrates the algorithm named "LDS algorithm".

Some computations are only indicated. For example, COMPUTE(S|s,Ls,*status*) represent the computation (update) of variables S|s, Ls, and *status*. All details can be found in (Kaučič, 2001b). With the algorithm in Fig. 5, the necessary information for the final construction of the hexagon tessellation is obtained. To use the hexagon tessellation efficiently (e.g. at geometric search, a flight-through problem), the algorithm in the second phase uses all given data and stores the tessellation. Points are inserted into the hexagons, the tiles that have their points only on their borders are removed, and the relationship network between neighbouring tiles is established.

4. The Worst Case Analysis

Finding the optimum tessellation with the minimum number of tiles in straight approach remains a hard problem (see Section 2). However, if the starting point of the tessellation is moved around to cover all points of the tile that contains the first (initial) starting point, all possible tessellations are covered. And, because the number of points in a tile remains much smaller than the total number of points ($k \ll n$), the optimum is found in $O(n)$, which represents the time complexity of the LDS algorithm.

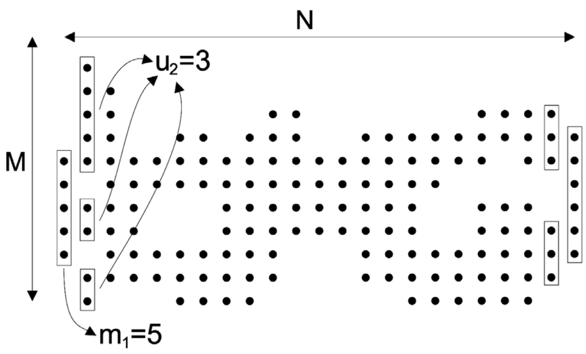

Fig. 6. Computing time complexity.

Let us consider the worst case of the LDS algorithm, too. Let N and M, $M \leq N$ denote the number of boundary points, as shown in Fig. 6.

Let m_i denote the number of points in column i, where $0 < i < N$, and $0 < m_i < M$. Let u_i denote the number of connected sets of points (the number of elements in AL), where $0 < u_i < \lfloor \frac{M}{2} \rfloor$ and let u be the average number of them, $u = (1/N) \sum_i u_i$. Time complexity of the LDS algorithm obviously depends on N and u. Then, the worst case occurs if all u_i take maximal values, i.e. $\forall i : u_i = \lfloor \frac{M}{2} \rfloor$. The worst time complexity is therefore

$$O(NM) = Nu = N\lfloor \frac{M}{2} \rfloor = O(N^2).$$

Such case occurs when between the points in the same column exists a gap in the size of one point. In general, $u \ll \frac{M}{2}$ and steps in the same column are performed only a few times, resulting in the total complexity $O(un)$.

5. Conclusion

In the paper, the process of the tessellation is outlined. A special care is devoted to the quasi-regular tessellation, and an algorithm for the tessellation, named LDS algorithm, is given. The paper is concluded with an idea of obtaining the optimum, and the worst case analysis.

To the knowledge of authors, no published algorithms for the tessellation of hexagons on an arbitrary area exist. Moreover, because the tessellation of hexagons has some interesting properties, it will soon become important in GIS society.

References

[1] ARMSTRONG M. A., (1988), *Groups and Symetry*, Springer-Verlag, New York.

[2] FARMER D. W., (1996), "Groups and Symmetry — a Guide to Discovering Mathematics", *Mathematical World, American Mathematical Society*, Vol. 5.

[3] KAUČIČ B., ŽALIK B., (2001a), "Comb Grid — Another Approach for Real-time Terrain Flythroughs", Proceedings of the GIS Research UK, 9th annual conference GISRUK 2001, Kidner D., Higgs G. (eds), 18th-20th April 2001, University of Glamorgan, Wales, 212–217

[4] KAUČIČ B., ŽALIK B., NOVAK F., (2001b), "Tessellation of quasi-regular hexagons", IJS Technical Report 8378, Institute Jožef Stefan, Ljubljana.

[5] MULMULEY K., (1994), *Computational Geometry — An Introduction Through Randomized Algorithms*, Prentice Hall, New Jersey.

[6] PREPARATA F. P., SHAMOS M. I., (1985), *Computational geometry — an introduction*, Springer-Verlag, New York.

[7] WELLS D., (1991), *The penguin dictionary of Curious and Interesting GEOMETRY*, Penguin books, London.

Received: June, 2001
Accepted: September, 2001

Contact address:
Branko Kaučič
PEF, University of Maribor
Koroška c. 160, SI-2000 Maribor, Slovenia
Phone: +386 2 2293 639
Fax: +386 2 2518 180
e-mail: branko.kaucic@uni-mb.si

Borut Žalik
FERI, University of Maribor
Smetanova ul. 17, SI-2000 Maribor, Slovenia
e-mail: zalik@uni-mb.si

Since 2001. BRANKO KAUČIČ is a PhD student at the Faculty of EE & CS, University of Maribor, Slovenia. His main interests are in computational geometry, terrain visualisation and terrain visibility. He is currently a teaching assistant at the Faculty of Education, University of Maribor, Slovenia.

BORUT ŽALIK is an associate professor in the department of Computer Science, Faculty of EE & CS, University of Maribor, Slovenia. He received his BSc in Electrical Engineering in 1985, MSc and PhD in Computer Science, both from the University of Maribor in 1989 and 1993. He is also a visiting senior research fellow at De Montfort University, U.K. His research interests include geometric modelling, computational geometry, GIS, and multimedia applications.