Malaysian Journal of Mathematical Sciences 13(S) December: 51–64 (2019)
Special Issue: Conference on Mathematics, Informatics and Statistics (CMIS2018)

MALAYSIAN JOURNAL OF MATHEMATICAL SCIENCES

Journal homepage: http://einspem.upm.edu.my/journal

Equilateral Monogonal Tessellation of Arbitrary Polygons - a Recursive Algorithm

Nesenbergs, K.

Institute of Electronics and Computer Science (EDI), Riga, Latvia

E-mail: krisjanis.nesenbergs@edi.lv
Corresponding author

ABSTRACT

Tessellation or tiling can be useful in multiple domains, such as computer graphics, physics, architecture etc. In this article the tessellation problem is explored and a recursive algorithm is proposed, implemented and validated for generating equilateral monogonal tessellations (including all semi-regular tessellations) in two dimensions inside any arbitrary simple polygon.

Keywords: Computational geometry, Equilateral tessellation, Monogonal tiling and Recursive algorithm.

1. Introduction

Tessellation or tiling can be defined as a dissection of an infinite plane into shapes of a finite area, see J. Scherphuis (2018). This can have practical applications in multiple domains - from triangulation in computer graphics (Ganapathy and Dennehy (1982)) to calculating domain walls in physics (Bazeia and Brito (2000)) and generating artistic patterns and modular assemblages in architecture (Celento and Harris (2011)). As such, algorithms for tessellation can have impact in unexpected areas.

In this paper a recursive tessellation algorithm is described, with original application in smart textiles, which might have applications in other fields.

A short introduction to the motivations for the proposed algorithm, followed by the definition of scope, and insight in state-of-art follows below. The proposed algorithm itself follows in section III. Finally, the results from implementing the algorithm as well as related conclusions and potential for future work are presented.

1.1 Motivation

The original motivation of the author is related to the search for the optimal wiring layout in textiles, as discussed in Nesenbergs (2016) and Nesenbergs and Selavo (2015), which could serve as a universal building material for prototypes of new smart clothing designs, such as Hermanis et al. (2013), Hermanis et al. (2015) or Ancans et al. (2017).

The need for all locations on the fabric to be reachable while using the minimum amount of material, and the grid of wiring to be resistant to both accidental damage and cutting by design while sewing the final garment dictate the properties of the wiring grid, which can be thought of as a tessellation of the two-dimensional plane, with emphasis on tessellation edges (wires) and vertexes (sensor connection points) instead of shapes/tiles themselves.

Even though the specific application is narrow, similar requirements for physical systems in other domains might also benefit from such tessellations and the proposed algorithm.

1.2 Scope

The tessellation problem is quite wide and multifaceted, thus a specific scope for the described algorithm, guided by the motivation above, is defined below:

1.2.1 Dimensionality

It is possible to tessellate space of any dimensionality, although the case of less than two dimensions is trivial. Usually research and practical applications are limited to 2D or 3D spaces, although higher dimensions are not unheard of. The simplest primitive for tessellating an n-dimensional space is called an n-simplex (Kozma and Szirmai (2015)), but more complex shapes can also be used. In this paper only two-dimensional tessellations will be explored.

1.2.2 Periodicity

A tiling with a repeating pattern is called periodic, and one without a repeating pattern is called non-periodic. A special case of non-periodic tilings without arbitrarily large periodic patches is called aperiodic - the best known examples of aperiodic tilings are Penrose tilings described in Steinhardt and Jeong (1996). However this paper is only concerned with periodic tiling patterns.

1.2.3 Symmetries

When talking about periodic tilings, an important concept is the symmetry of the tiling - the way how the pattern repeats.

The most commonly known symmetry is the isohedral or face-transitive symmetry, when the tiling has one (monohedral) or multiple (n-isohedral) polygons that are repeated through one of four Euclidean or rigid transformations also known as isometries. Depending on the amount and shape of these tiles, they form one of 17 possible plane symmetry or wallpaper groups, see Schattschneider (1978). There are a total of 81 isohedral types of planar tiling as described by Grünbaum and Shephard (1977a). Uniform tilings are isohedral tilings, that contain only regular figures. If no limit to the number of different vertex types is set there are infinite uniform tilings, basic types of which can be seen in Chavey (1989). When limited to only one type of vertex (isogonal or vertex-transitive tilings) there are only 11 uniform tilings left called Archimedian or semi-regular tilings as shown by Grünbaum and Shephard (1977b). One of these can be considered to be two different tilings, as they are mirror images of each other. Of these 11 only three consist of one type of regular tiles (regular triangles, squares, and regular hexagons) and are called regular tilings. The 11 Archimedian tilings can also be referred to as convex uniform tilings. These tilings are usually referred to by the list of numbers representing the number of edges of regular polygons surrounding a single vertex - e.g. "4.4.4.4" represents a vertex surrounded by 4 squares, and "4.6.12" represents a vertex surrounded by a square, a hexagon and a dodecagon. Grünbaum and Shephard (1977b) also describe additional concave uniform tilings, in which in addition to concave regular polygons also regular star-shapes are added.

As the original goal of the algorithm is concerned with vertexes and edges not tiles themselves, a more appropriate symmetry to discuss is the vertex symmetry or isogonality. This means, that there are one or more groups (equivalence classes) of vertices where each vertex can be transformed into another vertex the same Euclidean transformations as in the case of isohedral tilings. When there are multiple types of vertexes, these are called n-isogonal tilings, and in the case of single type of vertex these are called monogonal tilings. Grünbaum and Shephard (1978) proves that there are 91 types of normal plane isogonal tilings (or 93 if special tile or edge colorings are counted) of which 63 types are convex. All of these are based on 11 different "nets" one of which is in two enantiomorphic forms. These "nets" can be thought of as configurations or topologies of vertices which are connected with flexible strings. If these strings would be of equal length and pulled straight, forming regular rectangles, these nets would each form one of the 11 Archimedian tilings from above. Thus there are 11 "classes" of isogonal tilings, each of which contains also one or more monogonal tilings with the same vertex configuration. In this paper we will look at monogonal tilings.

1.2.4 Types of edges

The monogonal or 1-isogonal tilings can contain both straight and curved edges. As curves don't provide noticeable benefits for the field described in the motivation section, and increase the complexity, only straight edges will be in the scope of this paper.

The edges surrounding the vertex can also be of equal or different lengths. Equal length edges will form equilateral, but not necessarily regular, polygons. The previously mentioned 91 types of isogonal tilings, when restricted to straight and equal edges collapse into less types, of which there seems to be the 11 semi-regular tilings and at least additional 16 types of non-regular equilateral tilings, some of which have a degree of freedom allowing infinite number of specific equilateral tilings. These 16 types correspond to types IG23, IG25, IG28, IG31, IG33, IG38, IG39, IG41, IG43, IG55, IG78, IG79, IG81, IG84, IG86 and IG88 from the classification by Grünbaum and Shephard (1978). These are just the obvious ones, and some others might exist. An example of equilateral monogonal non-regular tiling IG88, can be seen in Figure 1.

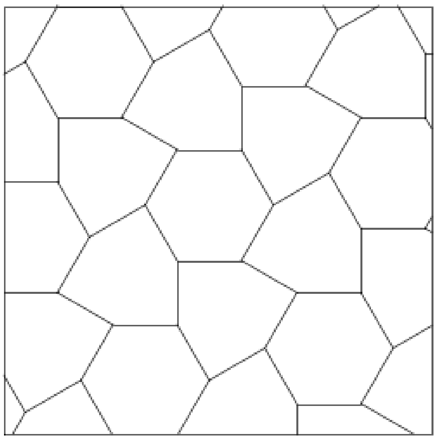

Figure 1: Equilateral monogonal non-regular tiling example with classification IG88.

The practical applications discussed in the motivation section would not benefit from different lengths of edges, as longer edges would introduce proportionally more risk of damage. Thus for the scope of the proposed algorithm only equal edges will be considered, resulting in equilateral monogonal tilings.

1.2.5 Out of scope considerations

There are some other considerations related to tessellations that do not impact the scope of this article. These include the specific cases of triangulation, e.g. connecting points on plane with triangles, while maximizing the minimum triangle angle as in Delaunay tessellations (Lee and Schachter (1980)), or quite the opposite - tessellating the plane based on equal distance to points on plane as in Voronoi diagrams (Aurenhammer (1991)).

1.3 State-of-art

Even though many tessellation algorithms exist, most of them are related to 3D surface tessellation for modelling software and games as evidenced by Hamann and Tsai (1996) and Tanemura et al. (1983).

Although some research, such as Delgado-Friedrichs (2003), targets similar tessellation problems, there are no algorithms for equilateral monogonal tiling of arbitrary polygons known to the author. Such an algorithm is proposed below.

2. Proposed algorithm

In order to generate the tessellation grid, the algorithm takes a starting configuration, including a seed point p_0, and generates all edges exiting this vertex, while executing the same recursive algorithm for each of the new vertex points p_i at the ends of those edges. Full source code of the algorithm implementation in *Python 3.6* is available at `https://github.com/krisjanis-nesenbergs/Tessellator/`. Several non-trivial details and considerations as well as configuration specifics related to the tessellation algorithm are detailed below.

2.1 Global configuration

In order for the algorithm to function, the specific tessellation configuration must be set, which is unchanging during the recursive iterations of the algorithm:

First, the specific tessellation method is set with three parameters - list of angles α_i surrounding a vertex ($\bar{A} = \alpha_0, \ldots, \alpha_n$), and two transition arrays ($\overline{T_c}$ and $\overline{T_{cc}}$), to find the angle at which to start the tessellation on the new vertex, if it was drawn as a result of traversing a specific angle from the previous vertex (any choice can be made for the first seed vertex). These transition arrays are of the same length n as $\bar{A}$ and at each index i contain the index of the next angle in $\bar{A}$ for the specific transition. There are two transition arrays instead of one - one for clockwise angle traversing around a vertex and one for counter-clockwise. On most semi-regular tessellations, the order of angles is the same in both directions, but for "4.6.12" as well as other non-semi-regular tessellations this order differs. For some tessellation methods multiple transition arrays are possible due to the fact that they have multiple degrees of freedom, such as several angles with the same value. An example of these parameters for all 11 of semi-regular tessellations are displayed in Table 1.

Table 1: An example of tessellation method parameters for all semi-regular tessellations.

Tessellation	$\bar{A}$ (in degrees)	$\overline{T_c}$	$\overline{T_{cc}}$
3.3.3.3.3.3	60,60,60,60,60,60	1,2,3,4,5,0	1,2,3,4,5,0
3.3.3.3.6	60,60,60,60,120	1,3,2,0,4	1,3,2,0,4
3.3.3.4.4	60,60,60,90,90	1,2,0,4,3	1,2,0,4,3
3.3.4.3.4	60,60,90,60,90	1,3,2,0,4	1,3,2,0,4
3.4.6.4	60,90,120,90	0,3,2,1	0,3,2,1
3.6.3.6	60,120,60,120	0,3,2,1	0,3,2,1
3.12.12	60,150,150	0,2,1	0,2,1
4.4.4.4	90,90,90,90	1,2,3,0	1,2,3,0
4.6.12	90,120,150	**0,2,1**	**0,1,2**
4.8.8	90,135,135	0,2,1	0,2,1
6.6.6	120,120,120	1,2,0	1,2,0

Additionally, to check if a newly calculated vertex is actually new, not calculated from another recursive path, an insignificantly small Δ must be specified to detect if two vertex coordinates are the same - as the floating point calculations are not precise two coordinates can be considered the same if they differ by less than Δ.

Finally, as only equilateral tessellations are calculated by the algorithm, the edge length l must also be defined.

2.2 Recursion termination conditions

Even though theoretical tessellations of plane are infinite, real world algorithms require some sort of termination conditions. The specific algorithm supports two termination conditions, of which at least one must be set - either maximum number of iterations I_{max} or boundary polygon B which is provided as an array of points describing its vertexes. B must have non-zero area.

Thus, if I_{max} is set, each recursive iteration is counted and if the count exceeds I_{max} no new iterations are initialized. Using I_{max} as a boundary condition can lead to incomplete tessellation, thus use of the boundary polygon is advised. If B is set, then p_0 must be within this polygon and each new point must also fall within the polygon, otherwise it is discarded and not processed by the algorithm further, thus limiting the number of recursive iterations.

2.3 Single recursive iteration

Each recursive call receives several parameters describing the specific iteration. First - a seed point p which is equal to p_0 in the first recursive call, and other generated vertex points in latter iterations. Additionally a starting angle α_s for the iteration is provided, as each vertex might have a different relative angle in regards to the coordinate system. Finally, transition information is provided, describing how the current vertex point must be constructed based on how it was reached. This consists of two parameters - direction of processing τ which is equal to either 0 (clockwise) or 1 (counter clockwise), and current angle index i_s in $\bar{A}$ for specific tessellation configuration.

With this configuration the algorithm first increases the current iteration count and checks if it does not exceed I_{max}. If it does - stop the current iteration. Otherwise for each angle α_i in $\bar{A}$ starting from i_s and wrapping around the array (where at k^{th} step $i = (i_s + k) \mod n$) the following steps are taken:

2.3.1 Calculating absolute angle

As the angles in $\bar{A}$ are relative angles, an absolute angle α for the current iteration must be calculated. To do that a running sum α_Σ of all relative angles used in this recursive call is kept ($\alpha_\Sigma = \alpha_\Sigma + \alpha_i$). From this $\alpha = \alpha_\Sigma + \alpha_s$.

2.3.2 Calculate coordinates of a new vertex

Coordinates for a new vertex p' are calculated using α_i in radians and the coordinates of p with simple trigonometry as seen in Equation (1).

$$\begin{aligned} p'_x &= p_x + l * cos(radians(\alpha_i)) \\ p'_y &= p_y + l * sin(radians(\alpha_i)) \end{aligned} \tag{1}$$

2.3.3 Verify vertex point is new

The set of all processed vertexes $\bar{P}$ is checked. If $p' \in \bar{P}$ the vertex point has already been processed and the algorithm can continue to the next angle α_{i+1} if there are unprocessed angles left.

2.3.4 Verify vertex point is inside boundary

If p' is in the interior of B (including on the edge), the point is a valid tessellation vertex and can serve as a seed point for the next recursion step. Otherwise, a point of $p_\times$ is found between line (p, p') and the polygon B and the resulting edge $(p, p_\times)$ is added to the set of final tessellation edges $\bar{E}$ ending on the edge of the tessellated figure. In some cases, if B is not convex, there might be several intersection points with the boundary and if it is an even number of intersections, p' will still fall within the interior of B. In this case the recursion continues as normal, but all partial edges defined by intersection points and end points p and p' that are on the interior of B are added to $\bar{E}$ as shown in Figure 2.

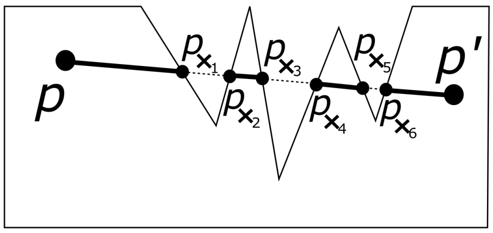

Figure 2: Resulting tessellation edges on intersections with concave boundary.

2.3.5 Good vertex is found

When a good vertex point p' is found it is added to $\bar{P}$ and the edge (p, p') is added to $\bar{E}$ (Except for cases where it was intersecting the boundary and added as intersection fragments in the previous step).

2.3.6 Call the next recursive iteration

For the next recursive iteration, the set of parameters $[p, \alpha_s, \tau, i_s]$ is replaced with a new set $[p', \alpha'_s, \tau', i'_s]$. Of these p' is already known. The new starting angle α'_s is the opposite angle of the α that lead to this vertex, thus it is calculated in Equation (2):

$$\alpha'_s = (\alpha + 180) \mod 360 \tag{2}$$

The new direction of processing is always the opposite of the current one, thus $\tau' = \neg\tau$.

Finally, the new current angle index i'_s is determined by using the correct element of the appropriate transition array as shown in Equation (3):

$$\begin{aligned} if \tau = 0, then i'_s = \overline{T_c}[i] \\ if \tau = 1, then i'_s = \overline{T_{cc}}[i] \end{aligned} \tag{3}$$

The new set of parameters is then used to call the next recursive iteration until some recursion termination condition is met.

2.4 Example parameter visualization

To help visualize, the algorithm, Figure 3 shows an example for "4.6.12" configuration, where $\bar{A} = \{a_0 = 90°, a_1 = 120°, a_2 = 150°\}$ and the algorithm is approaching vertex p' from vertex p. In this example if $\tau = 0$, then $\tau' = 1$, $i'_s = \overline{T_{cc}}[i] = [0, 1, 2][1] = 1$ representing 120° to be drawn in counter-clockwise direction from α'_s. Otherwise, $\tau' = 0$, $i'_s = \overline{T_c}[i] = [0, 2, 1][1] = 2$ representing 150° to be drawn in clockwise direction from α'_s.

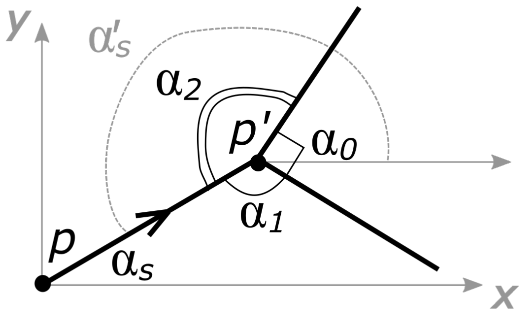

Figure 3: Visualization of example algorithm parameters for "4.6.12" configuration.

2.5 Additional considerations

Some additional considerations might be important for specific applications and implementations of the algorithm.

When checking for intersection points between the new edge and the bounding polygon, the new edge might fully or partially coincide with an edge of the polygon leading to and infinite number of intersection points. This can lead to the exception to the rule, that even number of intersections will lead to the new point being inside of the boundary polygon, thus additional checks are needed in such cases.

Because checking the list of processed points can be resource intensive it is advised to use specialized data structures for this list, e.g. a hash table using representation of the significant numbers of x and y coordinates of the point as a key could be a feasible solution. Because the calculation of equality of two points requires the Δ and coordinates might differ by a miniscule amount, ordinary hashing algorithms for the point object are not applicable, as the same point might hash to different values.

3. Results

The proposed algorithm was implemented in *Python 3.6* and validated by tessellating arbitrary simple polygons (both concave and convex) with all 11 semi-regular tessellations (See example in Figure 4), as well as other equilateral monogonal tessellations (see example in Figure 5).

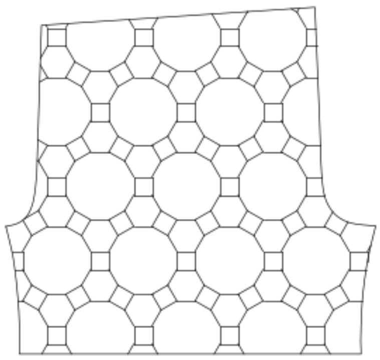

Figure 4: Example algorithm result using 4.6.12 tessellation on an arbitrary simple polygon.

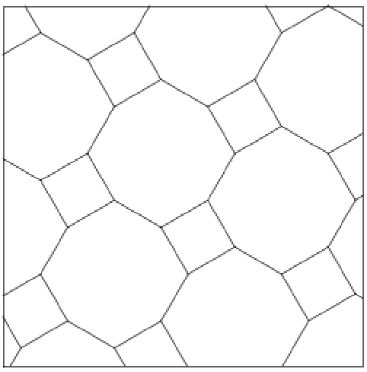

Figure 5: Example non-semi-regular tessellation with vertex angles 90, 120 and 150 degrees.

The speed of algorithm implementation was validated on a modern laptop. A task of triangulating (Tessellation *3.3.3.3.3.3*) an A4 page ($210mmx297mm$) using triangle edge length of $10mm$ took 0.9 seconds, generating 2217 tessellation edges in $\bar{E}$ while calling the recursive function 4067 times.

Of the total run time almost 15% were spent on intersection checking with boundaries, and 11% for checking if the point/edge are new instead of already generated through another recursion path, thus these checks are prime suspects for speed optimization of the implementation.

The source code of the implemented algorithm is available for attributed use in the link in Section 2.

4. Conclusions

The proposed algorithm is capable of producing all equilateral monogonal tessellations, including the semi-regular tessellations. This can be used not only for tessellating a plane, but for tessellating any simple convex or concave polygon.

With addition of an array of edge lengths for each vertex the algorithm could be extended to generate any monogonal tessellation without the limit of equilateral edges. In the use-cases where the overhead introduced by recursion is not acceptable, the algorithm can also be flattened by the use of standard recursion flattening methods.

Acknowledgement

The author would like to thank Mr. Jaap Scherphuis for help on clarifying tiling concepts and being an insightful discussion partner on the subject.

References

Ancans, A., Rozentals, A., Nesenbergs, K., and Greitans, M. (2017). Inertial sensors and muscle electrical signals in human-computer interaction. In *Information and Communication Technology and Accessibility (ICTA), 2017 6th International Conference on*, pages 1–6. IEEE.

Aurenhammer, F. (1991). Voronoi diagrams-a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405.

Bazeia, D. and Brito, F. (2000). Tiling the plane without supersymmetry. *Physical review letters*, 84(6):1094.

Celento, D. and Harris, E. (2011). Potentials for multi-dimensional tessellations in architectural applications. In *Integration Through Computation: Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, pages 308–313.

Chavey, D. (1989). Tilings by regular polygons-ii: A catalog of tilings. *Computers & Mathematics with Applications*, 17(1-3):147–165.

Delgado-Friedrichs, O. (2003). Data structures and algorithms for tilings i. *Theoretical Computer Science*, 303(2-3):431–445.

Ganapathy, S. and Dennehy, T. G. (1982). A new general triangulation method for planar contours. *ACM Siggraph Computer Graphics*, 16(3):69–75.

Grünbaum, B. and Shephard, G. (1978). The ninety-one types of isogonal tilings in the plane. *Transactions of the American Mathematical Society*, 242:335–353.

Grünbaum, B. and Shephard, G. C. (1977a). The eighty-one types of isohedral tilings in the plane. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 82, pages 177–196. Cambridge University Press.

Grünbaum, B. and Shephard, G. C. (1977b). Tilings by regular polygons. *Mathematics Magazine*, 50(5):227–247.

Hamann, B. and Tsai, P.-Y. (1996). A tessellation algorithm for the representation of trimmed nurbs surfaces with arbitrary trimming curves. *Computer-Aided Design*, 28(6-7):461–472.

Hermanis, A., Cacurs, R., Nesenbergs, K., Greitans, M., Syundyukov, E., and Selavo, L. (2015). Wearable sensor grid architecture for body posture and surface detection and rehabilitation. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, pages 414–415. ACM.

Hermanis, A., Nesenbergs, K., Cacurs, R., and Greitans, M. (2013). Wearable posture monitoring system with biofeedback via smartphone. *Journal of Medical and Bioengineering Vol*, 2(1).

Scherphuis, J. (2018). Jaapsch.net/tilings. [Online; accessed 14-July-2018].

Kozma, R. T. and Szirmai, J. (2015). New lower bound for the optimal ball packing density in hyperbolic 4-space. *Discrete & Computational Geometry*, 53(1):182–198.

Lee, D.-T. and Schachter, B. J. (1980). Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242.

Nesenbergs, K. (2016). Architecture of smart clothing for standardized wearable sensor systems. *IEEE Instrumentation & Measurement Magazine*, 19(5):36–64.

Nesenbergs, K. and Selavo, L. (2015). Smart textiles for wearable sensor networks: Review and early lessons. In *Medical Measurements and Applications (MeMeA), 2015 IEEE International Symposium on*, pages 402–406. IEEE.

Schattschneider, D. (1978). The plane symmetry groups: their recognition and notation. *The American Mathematical Monthly*, 85(6):439–450.

Steinhardt, P. J. and Jeong, H.-C. (1996). A simpler approach to penrose tiling with implications for quasicrystal formation. *Nature*, 382(6590):431.

Tanemura, M., Ogawa, T., and Ogita, N. (1983). A new algorithm for three-dimensional voronoi tessellation. *Journal of Computational Physics*, 51(2):191–207.