Synthesizing Periodic Tilings of Regular Polygons

Asla Medeiros e Sá
FGV EMAp, Rio de Janeiro, Brazil

Luiz Henrique de Figueiredo
IMPA, Rio de Janeiro, Brazil

José Ezequiel Soto Sánchez
IMPA, Rio de Janeiro, Brazil

***Abstract*—We present a simple representation for periodic tilings of the plane by regular polygons. Our approach is to represent explicitly a minimal subset of the vertices from which we systematically generate all vertices in the tiling by translations. We then deduce the edges and the faces using the constraint that all edges have the same length. Our representation can be used to synthesize tilings manually and automatically from images.**

***Index Terms*—tilings, tessellations, generative model**

I. Introduction

Tiling the plane with regular polygons is a fascinating subject, both mathematically and graphically [1], [2], [3], with a long history [4]. It is now almost exactly 400 years since the mathematics of tilings were first discussed in Kepler's book "Harmonices Mundi" of 1619.

A tiling of the plane by polygons is a subdivision of the plane into bounded polygonal regions that either are disjoint, share a vertex, or share an edge. Restricting the faces of the tiling to be regular polygons brings lots of rigidity while still allowing much interesting variety. For instance, there are 21 possibilities to arrange regular polygons around a vertex [4]. Of those, only the 15 types shown in Fig. 1 can actually occur as the neighborhood of a vertex in a tiling by regular polygons. If all vertices have the same type, then the tiling is *uniform* or *Archimedean* and only 11 such tilings are possible (Fig. 2). Besides other symmetries, these tilings are *periodic*, in the sense that they are invariant under two independent translations.

In this paper, we present a simple method to represent and compute periodic tilings of the plane by regular polygons. Our approach is to represent explicitly a minimal subset of the vertices from which we systematically generate all vertices in the tiling by translations. We then deduce the edges and the faces using the constraint that all edges have the same length. Our approach is reminiscent of the regular systems of points discussed in the classic book by Hilbert and Cohn-Vossen [5].

II. Translation Cells

A periodic tiling is invariant under two independent translations. The corresponding *translation vectors* t_1 and t_2 divide the plane into an infinite grid of parallelograms, called the *translation cells* (Fig. 3). Invariance means that everything in one cell of the grid is repeated exactly in all cells. A translation cell is also called a *fundamental domain* for the tiling.

The *basic translation cell* is the one containing the origin:

$$T_{0,0} = \{\lambda_1 t_1 + \lambda_2 t_2 : \lambda_1, \lambda_2 \in [0,1)\}$$

As illustrated in Fig. 3, this cell is *half-open*, that is, open at the sides not containing the origin.

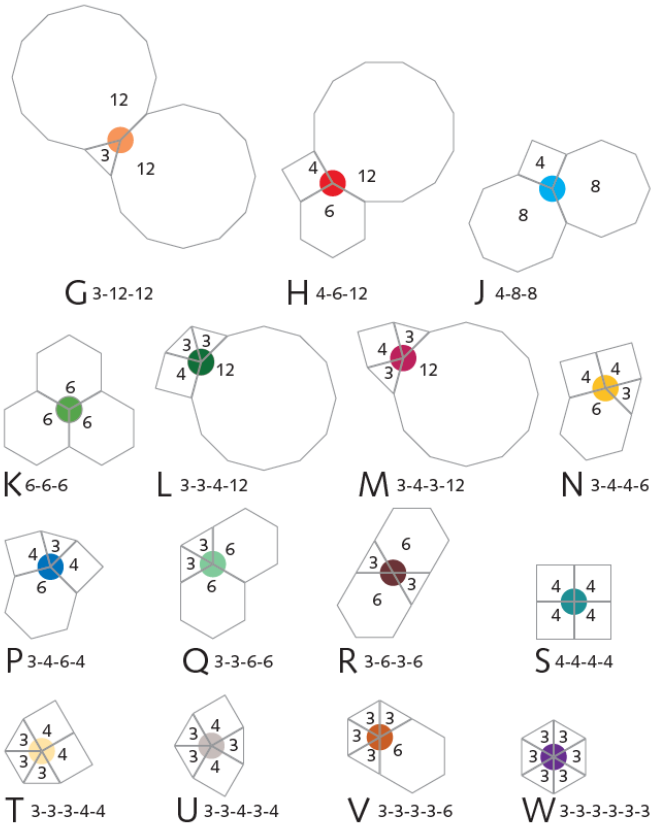


Fig. 1. The 15 vertex neighborhoods in tilings by regular polygons [6].

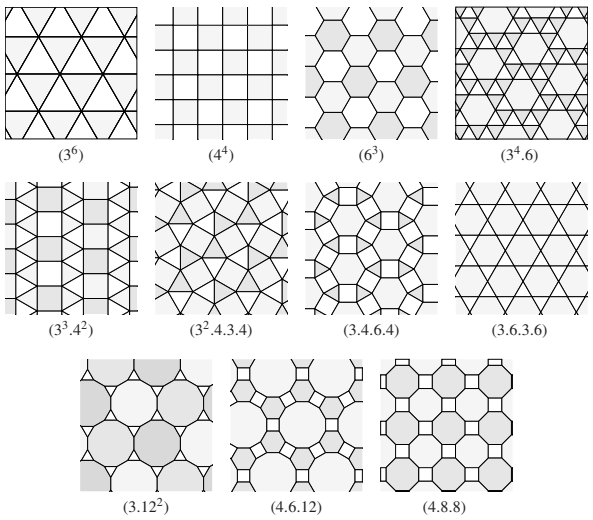


Fig. 2. The 11 uniform tilings of the plane (from [3]).

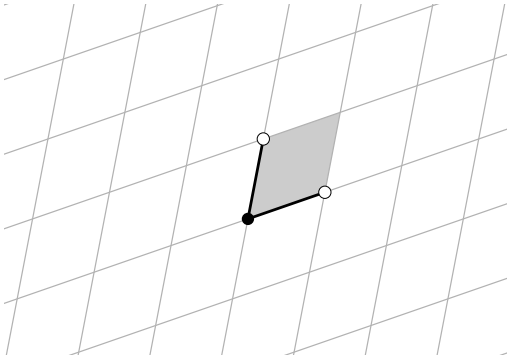


Fig. 3. Grid of translation cells, translation vectors at the origin (in black), and the basic translation cell (in gray).

Each translation cell T is a translation of the basic cell by *integer* multiples of t_1 and t_2 and so can be written

$$T = T_{n_1,n_2} = T_{0,0} + n_1 t_1 + n_2 t_2$$

with $n_1, n_2 \in \mathbf{Z}$. Different translation cells are disjoint, because they are half-open like $T_{0,0}$. Thus, the plane is the disjoint union of the translation cells, and each point in the plane belongs to exactly one translation cell.

Two points p and q in the plane are *equivalent* if they coincide when their translation cells are moved one on top of the other. This happens exactly when $p = q + n_1 t_1 + n_2 t_2$ for some $n_1, n_2 \in \mathbf{Z}$. In algebraic terms, p and q are equivalent exactly when they are in the same coset of the translation lattice $\mathbf{Z}t_1 + \mathbf{Z}t_2$, which is a discrete additive subgroup of $\mathbf{R}^2$.

Since the translation cells are bounded, there is only a finite number of vertices of the tiling in each translation cell. Moreover, no two vertices in a translation cell are equivalent. In fact, each translation cell contains exactly one representative of each equivalence class of vertices. We shall see how to reconstruct the whole tiling from the vertices in the basic cell.

III. Basic directions

Our representation of a tiling starts with an analysis of a sketch or an image of the tiling, like the ones in Fig. 2. Since the polygons are regular, all edges have the same length, which we may assume to be 1. We normalize the position and the orientation of the tiling by choosing one of its vertices to be the origin and one of its edges to be horizontal.

The first observation is that the edges are aligned with a small number of directions, because the angles between the edges around a vertex are limited to the internal angles of a regular polygon of 3, 4, 6, 8, or 12 sides, given the restrictions shown in Fig. 1. Since the edges have length 1 and there is a horizontal edge, it turns out that these directions are conveniently represented by complex numbers, more precisely by the roots of unity $\pm\omega_n$ and $\pm\overline{\omega}_n$ for $n \in \{1,2,3,4,6,8,12\}$, where $\omega_n = \exp(\frac{2\pi i}{n})$. These are called the *basic directions*. Of course, $\omega_1, \omega_2, \omega_4$ are more familiarly known as $1, -1, i$.

The next observation is that any two vertices in the tiling are connected by a path along the edges. Following the edges in the path expresses the path as an *integer* linear combination of the *positive* basic directions ω_n, $\overline{\omega}_n$ for $n \in \{1,3,4,6,8,12\}$. Note that $\omega_2 = -1 = -\omega_1$ is not needed since we have ω_1. (Henceforth, all basic directions are assumed positive.) The expression of a path as an integer linear combination of the basic directions is not always unique because some of the basic directions are linearly dependent over the integers. Here is a complete list of the integer linear dependence relations among the basic directions, up to negation and complex conjugation:

$$\omega_3 + \overline{\omega}_6 = 0, \quad -1 + \omega_6 + \overline{\omega}_6 = 0, \quad i - \omega_{12} + \overline{\omega}_{12} = 0$$

These relations simplify the expression of a path to use a subset of *non-redundant* basic directions, that is, those that are linearly independent over the integers.

IV. Data for representing tilings

The first ingredient in our representation of a tiling is a set of non-redundant basic directions. To find such a set, we list the basic directions that appear in the tiling and we use the relations above to eliminate redundant directions, if any. There is room for choice here. We always eliminate $\omega_3 = -\overline{\omega}_6$ and $\overline{\omega}_6 = 1 - \omega_6$, and we typically eliminate $i = \omega_{12} - \overline{\omega}_{12}$. After choosing and fixing a set of non-redundant basic directions, the paths connecting two vertices of the tiling have a *unique* representation as an integer linear combination of the basic directions. Since there is a vertex at the origin, every vertex is connected by a path to the origin and so has a unique representation as an integer linear combination of the basic directions in the tiling. This is the basis of our representation.

The next ingredient are the translation vectors t_1 and t_2. To find these vectors, we look for the two nearest vertices that are equivalent to the origin. The paths from the origin to these two vertices define the two translation vectors and so the basic cell. Like the vertices, the translation vectors have a unique representation as an integer linear combination of the basic directions in the tiling. Since $\pm t_1$ and $\pm t_2$ define the same grid of translation cells as t_1 and t_2, we may and shall assume that the x coordinates of t_1 and t_2 are non-negative. Thus, t_1 and t_2 always point to the right of the origin, as in Fig. 3. Even with this restriction, there may be room for choice here as well, as we shall see in the examples below.

The last ingredient is the set of vertices inside the basic cell. We call them *seeds* because all vertices in the tiling are translations of the seeds. Like every vertex in the tiling, each seed has a unique representation as an integer linear combination of the basic directions. This representation is found by following a path from the origin to each seed, but the representation does not depend on the actual path chosen. Unlike the previous ingredients, there is no room for choosing the seeds: they are exactly vertices inside the basic cell, which is determined by the translation vectors.

To summarize, each tiling is represented by three pieces of data: a set of non-redundant basic directions, two translation vectors, and a set of seeds in the basic cell. The translation vectors and the seeds are given as integer linear combinations of the basic directions. Let us now see some concrete examples of this data in action.

V. Two examples

Consider the hexagonal tiling in Fig. 4. The basic directions that appear in this tiling as vectors emanating from a vertex are $\pm\omega_1$, $\pm\omega_6$, $\pm\overline{\omega}_6$. Choose the non-redundant basic directions to be $\omega_1 = 1$ and ω_6. Place the bottom-right vertex of a hexagon at the origin. Then, the paths from the origin to the two nearest equivalent vertices are $\omega_6 + 1$ and $\omega_6 - \overline{\omega}_6$. After eliminating $\overline{\omega}_6 = 1 - \omega_6$, we get the translation vectors $t_1 = \omega_1 + \omega_6$ (in blue) and $t_2 = -\omega_1 + 2\omega_6$ (in red). The seeds are the vertices inside the basic cell: the origin and the point ω_6 (in black). We express the translation vectors and the seeds as integer linear combinations of the basic directions. Thus, the data for representing the hexagonal tiling is:

basic directions:	ω_1, ω_6
translation vectors:	$[1,1]$, $[-1,2]$
seeds:	$[0,0]$, $[0,1]$

This data is unique once we fix the basic directions, the origin, and the translation vectors, but variations are possible. For instance, we can choose the translation vector $t_1 = \overline{\omega}_6 + 1 = [2,-1]$ and keep the same seeds. Or we can keep the translation vectors but place the bottom-left vertex of a hexagon at the origin; the second seed then becomes $2\omega_6 = [0,2]$.

The hexagonal tiling is simple to understand because it has only two basic directions, which form a basis for the plane. This simplicity may be misleading. The tiling with triangles and squares in Fig. 5 uses *three* non-redundant basic directions: $\omega_1 = 1$, $\omega_4 = i$, and ω_6. These basic directions are not linearly independent over $\mathbf{R}$, but they are linearly independent over $\mathbf{Z}$. We place the bottom-left vertex of a square at the origin. The translation vectors are ω_1 (in blue) and $\omega_4 + \omega_6$ (in red). The seeds are the origin and $\omega_1 + \omega_4$ (in black). Thus, the data for representing this tiling is:

basic directions:	ω_1, ω_4, ω_6
translation vectors:	$[1,0,0]$, $[0,1,1]$
seeds:	$[0,0,0]$, $[1,1,0]$

Alternatively, if we place the top-right vertex of a square at the origin, then the second seed becomes $\omega_6 = [0,0,1]$.

Fig. 9 at the end of the paper shows more complex and more interesting examples of our representation of tilings.

VI. Synthesizing tilings

Given a set of non-redundant basic directions, the two translation vectors, and the seeds, all expressed as integer linear combinations of the basic directions, we can systematically generate all vertices of the tiling without repetition. Indeed, every vertex v has a unique representation as $v = s + n_1 t_1 + n_2 t_2$, where s is a seed and $n_1, n_2 \in \mathbf{Z}$. The set of all vertices is a regular system of points in the sense of Hilbert and Cohn-Vossen [5], the regularity coming from the two translations.

The key idea in our approach to synthesizing tilings is that we can find the edges and the faces *automatically* from the vertices (Fig. 6). Indeed, the edges emanating from a given vertex v go to the vertices lying at distance 1 from v. Moreover, these vertices are the nearest ones to v. Therefore, it is only necessary to search for these vertices in the cell c containing v and in the immediate neighborhood of c, that is, the eight adjacent cells $c \pm t_1$, $c \pm t_2$, $c \pm t_1 \pm t_2$. Thus, all vertices that can define an edge with v can be generated and checked in time that is linear in the number of seeds. Indeed, if there are n seeds in the tiling, then there are $9n$ candidate vertices, because every cell contains exactly n vertices. The candidate vertices are generated once per cell c and used to define all edges emanating from vertices in c (Fig. 6).

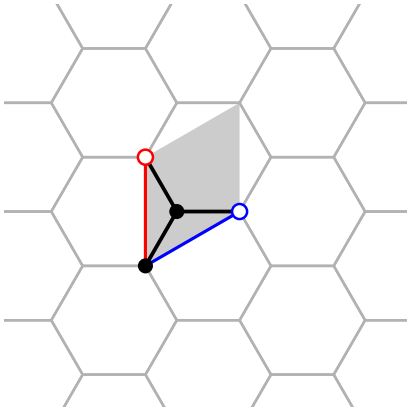


Fig. 4. Translation vectors (in color), basic translation cell (in gray), and seeds (in black) for the hexagonal tiling.

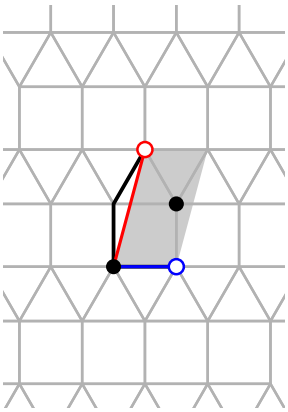


Fig. 5. Translation vectors (in color), basic translation cell (in gray), and seeds (in black) for the triangle-square tiling.

Finding the edges suffices for creating line renderings of the tiling (but note that the procedure above finds edges twice). For more interesting renderings and further processing, we need to find the faces of the tiling.

To generate the faces without repetition, we assign to each face its lowest leftmost vertex, which we call the *anchor* of the face (Fig. 7). Then, for each cell c, we generate the faces whose anchors are in c. Not all vertices in c are anchors. Some vertices are anchors to two faces.

To find the faces having a vertex v as anchor, first take the edges that emanate from v to the right, thus making angles with the horizontal in the interval $(-90°, 90°]$. There are at most three such edges, because the internal angles of the faces are at least 60°. If there is only one edge, then v is not an

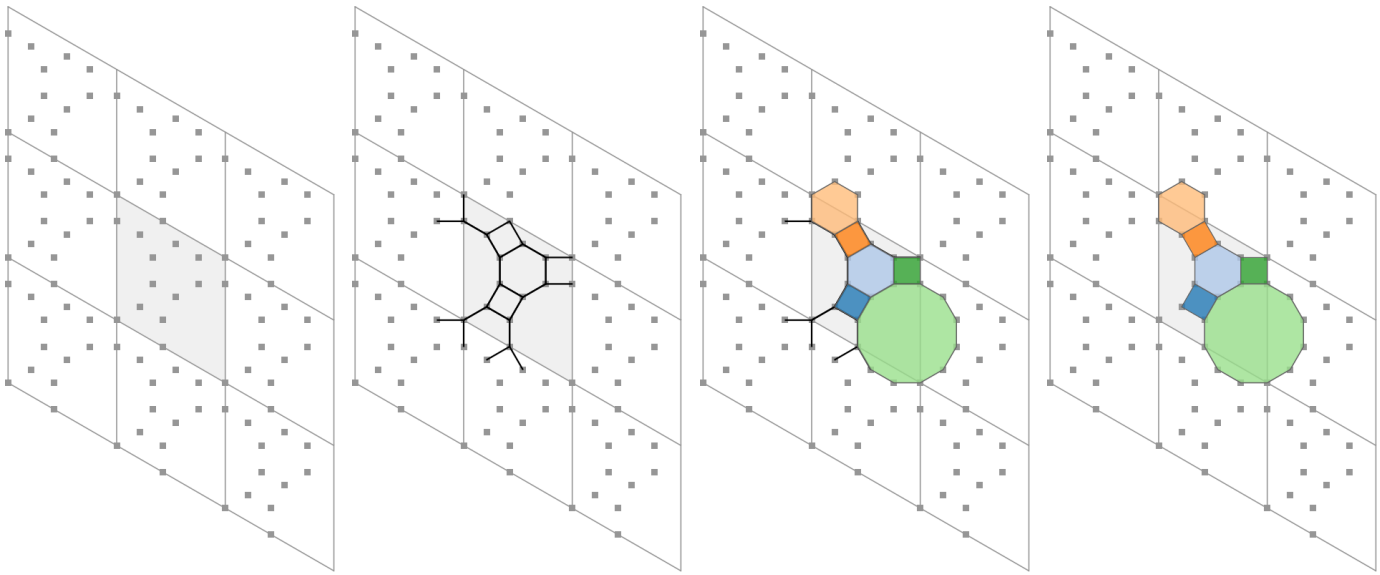


Fig. 6. Synthesizing a tiling near a cell: candidate vertices, edges, faces, patch.

anchor. Otherwise, there are two or three edges. Order the edges circularly around v, simply by comparing the y coordinates of their endpoints. Each pair of consecutive edges corresponds to a face having v as anchor, and so there are one or two such faces (Fig. 7). Each face is a regular polygon whose center o is found by intersecting the bisectors of the two edges corresponding to the face. The vertices of the face are those whose distance to o is the same as the distance from v to o. The same $9n$ candidate vertices are tested here. Finally, order the vertices of the face circularly around o, so that it can be drawn or further processed.

Repeating this procedure for each potential anchor v finds all faces anchored inside a cell c, which we call the *patch* of the tiling in c (Fig. 6). The complete tiling is the union of all patches. The patches are translated copies of each other and do not overlap. Patches are also called *translational units* [3].

To synthesize the tiling inside a given rectangular window in the plane, we first systematically visit the cells that intersect the window generating and storing its vertices. We also visit the immediate neighborhood of these cells. Cells are marked when visited to ensure being visited exactly once. Then we visit each cell again, finding the edges emanating from its vertices and the faces anchored in the cell, thus generating the patch in each cell. Both tasks use the vertices in the adjacent cells, which were generated in the first step.

Alternatively, we can process only the basic cell and translate the resulting patch to the other cells. If we store the result in terms of the basic directions, then the translation only involves integers, except perhaps for a final conversion to Cartesian coordinates for rendering. However, since the faces share vertices, care must be taken to avoid duplicating vertices. This may be relevant for modeling and further processing but probably not for rendering.

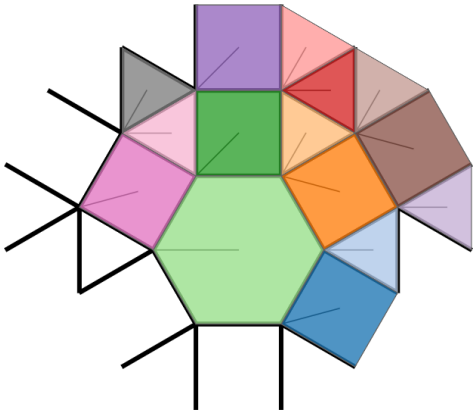


Fig. 7. Faces and anchors. A thin line joins the center of a face to its anchor.

VII. Extracting data from images

The analysis described in §IV to extract a representation from a sketch of the tiling can be automated to extract a representation from an image of the tiling. A fully automatic solution starts with image acquisition, noise removal, and corner detection to find the vertices. In the semi-automatic solution which we have implemented, the user selects the vertices directly on the image. In both cases, the coordinates of the vertices are acquired only approximately. As we explain below, these approximate coordinates suffice to find the basic directions, the translation vectors, and the seeds.

We start by choosing the vertex closest to their barycenter as the origin. We place this vertex in a queue. We process each vertex v in the queue as follows. We find the vertex not in the queue that is closest to v. Let d be its distance to v. Then we find all vertices not in the queue whose distance to v is at most to $1.15d$. This handles imprecisions in vertex acquisition and

mild distortions in images. For each selected vertex w, we find the basic direction ω that is closest to the direction vw and correct vw to be ω. Then we add w to the queue and repeat. After the queue is exhausted, we have built a tree connecting all vertices whose edges are aligned with the basic directions. Every vertex then has a unique path to the origin and so is given as an integer linear combination of the basic directions. Finally, we remove redundant directions as before.

The next step is finding the translation vectors. We simply test all possibilities. For each vertex v not the origin o such that the vector ov points to the right, we compute a score for the translation vector ov as the number of vertices that are (nearly) translated by ov to other vertices. The two translations with the highest scores are the translation vectors for the tiling. Since the translations move the origin to a vertex, they are also given by integer linear combinations of the basic directions.

Having found the translation vectors, we have the basic cell, and the seeds are the vertices inside the basic cell, as before. Thus, the data for representing the tiling is complete.

Fig. 10 shows three examples of this process in action. In Fig. 10a, the user selected all vertices in an image of one of our synthesized tilings, but the user did not do it very carefully. Nevertheless, the tiling is correctly reconstructed. Fig. 10b shows that the tiling can be reconstructed by selecting a small, representative fraction of vertices. Fig. 10c shows that a tiling can be reconstructed from a photograph even if it has perspective distortion, which affects edge lengths.

VIII. Discussion

Since the only symmetries represented in our data are the translations, our data may be longer than if rotations and reflections were incorporated, as in the database given by Kaplan [3]. In particular, our basic translation cell is typically larger than the fundamental domains that can be replicated by rotations and reflections. Also, complex tilings may need a large number of seeds. On the other hand, our data is by design simpler to understand, extract, and code. It is just symbolic and integer data. The only numerical data are the Cartesian coordinates of the basic directions, which are known exactly to any precision required. They are known a priori and are not explicitly part of the data.

The simplicity of our data makes it feasible to extract it automatically from an image, as described in §VII. Having to represent other symmetries, such as rotations around certain vertices and reflections about certain edges, would make this task harder to automate. Moreover, having other symmetries complicates generating vertices and faces without repetition.

Our representation is robust: any data that is correct but does not strictly conform to our constraints can be easily fixed. Indeed, we can eliminate redundant directions and redundant seeds, we can invert any translation vector that does not point to the right, and can ensure that the seeds are inside the basic cell by moving seeds that are outside using translations. These tasks are all essentially symbolic, not numerical.

The uniqueness of the representation of vertices in terms of the basic directions has the numerical benefit that the Cartesian coordinates of vertices are always computed using the same expression and so always yield the same floating-point numbers.

Unfortunately, the geometric computations required in our method use the Cartesian coordinates of vertices, not their integer representation in terms of basic directions. The Cartesian coordinates are not exact because the Cartesian coordinates of basic directions are typically irrational numbers. Nevertheless, the discrete nature of tilings allows us to set comfortable numerical tolerances for these geometric computations. For instance, the vertices at distance 1 from a given point are those whose numerical distances are in a wide interval, such as $[0.9, 1.1]$. This tolerance more than suffices: the actual numerical errors are in the last few bits.

The use of complex numbers to represent basic directions is mostly a convenience, but it does allow us to use symbolic names for the basic directions, and more importantly, to find and express integer dependence relations between them.

Although we haven't explored this yet, our representation and synthesis method allows high-quality direct rendering of tilings with shaders. First, we compute the patches for the basic cell and for its immediate neighbors once, offline even. Then, given a point p which we need to color in a shader, we find the translation cell that contains p and move p to its equivalent point q in the basic cell. Finally, we find which face in the stored patches contains q and return its color or texture value. To draw edges of a given width w, just check whether q is within w of an edge and return an appropriate edge color. There is room for endless artistic variation [7].

IX. Related work

The general approach given by Kaplan [3] is well suited for representing all the varieties of isohedral tilings, given a parametrization of the tile borders and its own adjacency. However, that approach becomes too complex when we try to generalize it for periodic tilings having more than one tile.

Delgado-Friedrichs [8] describes a general data structure for representing any periodic tiling accurately with a graph symbol and a pair of adjacency functions between "chambers", a triangulation of the original tiles such that each triangle includes exactly one edge of the tiling. Again, the graph symbol and the chamber labeling gives all the symmetries of the tiling. However, the rendering of such symbols in large scale is costly, since the classification of each point relies only in the neighboring chambers. Our translation-only approach gives a fast method for classifying an arbitrary point and for drawing the tiling inside arbitrarily large regions of the plane.

Our approach is closer in spirit to that of Ostromoukhov [9]. He explains how to produce a representation of a plane ornamental pattern, such as an Islamic pattern, by manually analyzing its structure starting from a sketch or an image of the pattern. He also shows how to use the analytical representation to synthesize a drawing. A key point in Ostromoukhov's approach is strand analysis, which represents how the symmetries of the pattern behave in a fundamental region. Our approach with seeds is similar, but much simpler, since it only requires translational symmetry.

X. Conclusion

No complete classification exists for periodic tilings of the plane by regular polygons. We believe that a simple data representation, such as the one presented here, will enable the comparison and classification of existing tilings and a systematic search for new tilings. This is our long-term goal.

A first step in this program is an algorithm for deciding whether two representations define the same tiling. This involves normalizing the set of basic directions (a symbolic task), deciding whether two pairs of translations vectors define the same grid (a numerical task), and deciding whether the two sets of seeds are equivalent (a matching task). A robust solution for this problem will be the theme of our future work.

Our initial motivation for this work and our short-term goal is to convert all the drawings in the catalog [6] to our representation. There are over 200 drawings, all made manually using precise geometric constructions with a CAD program. Fortunately, the materials for the book already include images of vertex constellations as dots, like the ones shown in Fig. 8. These images can be processed fully automatically as described in §VII, because it is simple to process such images to find the centers of the dots as approximate positions for the vertices. This conversion is already under way.

Acknowledgment

The second author is partially supported by a CNPq research grant. The third author is partially supported by a CNPq doctoral scholarship.

References

[1] B. Grünbaum and G. C. Shephard, *Tilings and patterns*. W. H. Freeman, 1989.

[2] J. H. Conway, H. Burgiel, and C. Goodman-Strauss, *The symmetries of things*. AK Peters, 2008.

[3] C. Kaplan, *Introductory tiling theory for computer graphics*. Morgan and Claypool, 2009.

[4] B. Grünbaum and G. C. Shephard, "Tilings by regular polygon," *Mathematics Magazine*, vol. 50, no. 5, pp. 227–247, 1977.

[5] D. Hilbert and S. Cohn-Vossen, *Geometry and the imagination*. Chelsea, 1952.

[6] R. Sá and A. Medeiros e Sá, *Sobre malhas arquimedianas*. Editora Olhares, 2017.

[7] J. M. Sullivan, "Conformal tiling on a torus," in *Proceedings of Bridges 2011: Mathematics, Music, Art, Architecture, Culture*, R. Sarhangi and C. H. Séquin, Eds. Tessellations Publishing, 2011, pp. 593–596.

[8] O. Delgado-Friedrichs, "Data structures and algorithms for tilings I," *Theoretical Computer Science*, vol. 303, no. 2, pp. 431–445, 2003.

[9] V. Ostromoukhov, "Mathematical tools for computer-generated ornamental patterns," in *Electronic Publishing, Artistic Imaging, and Digital Typography*, ser. Lecture Notes in Computer Science, vol. 1375. Springer, 1998, pp. 193–223.

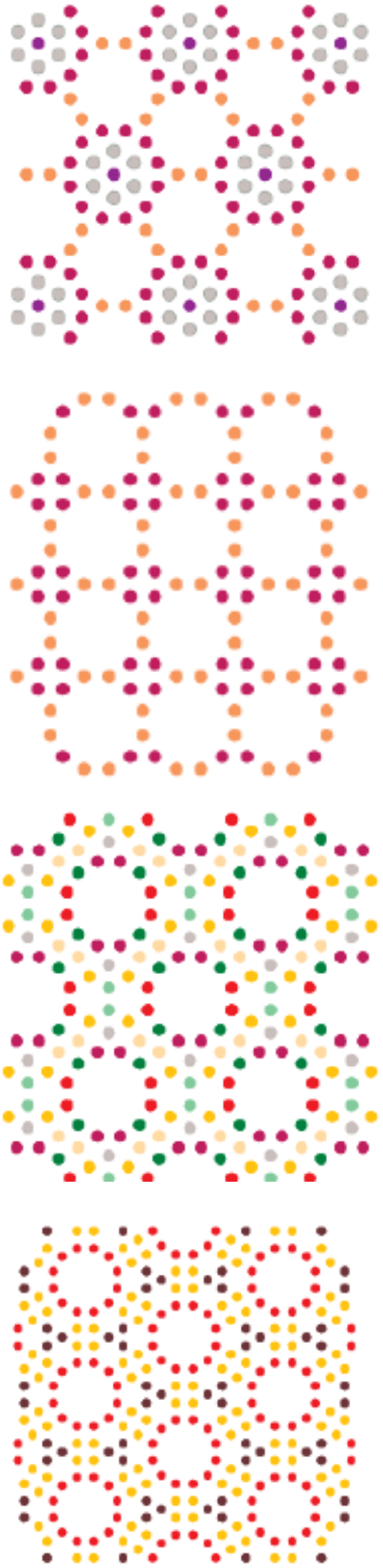
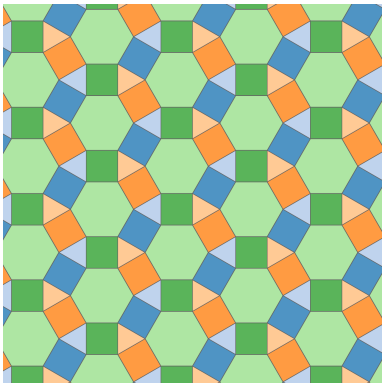
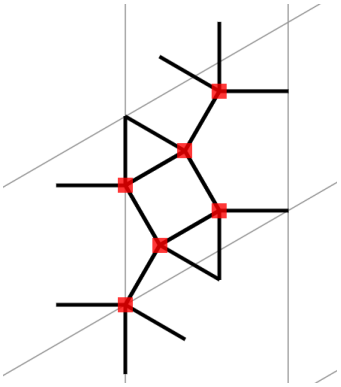


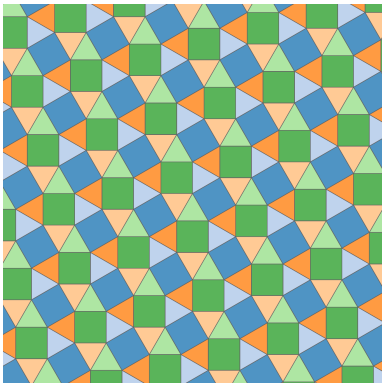
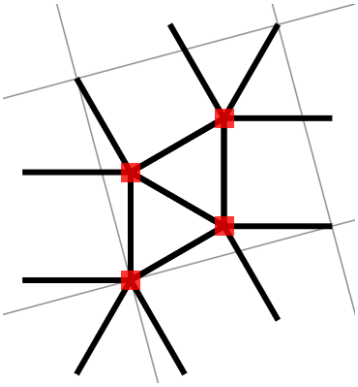
Fig. 8. Vertex constellations [6].

basic directions:
ω_1, ω_6, ω_{12}, $\overline{\omega}_{12}$

translations vectors:
$[1,1,1,0]$, $[-1,2,1,-1]$

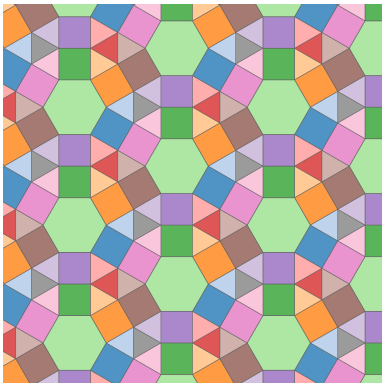
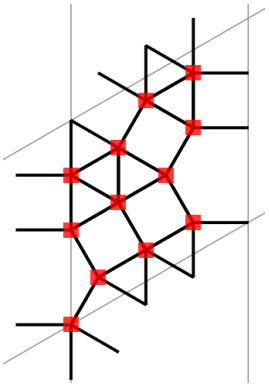
seeds:
$[0,0,0,0]$, $[0,1,0,0]$, $[0,1,1,0]$, $[-1,2,0,0]$,
$[-1,3,1,0]$, $[-1,2,1,0]$

basic directions:
ω_1, ω_4, ω_6, ω_{12}

translations vectors:
$[1,0,0,1]$, $[-1,1,1,0]$

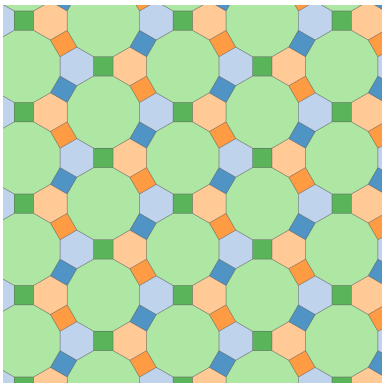
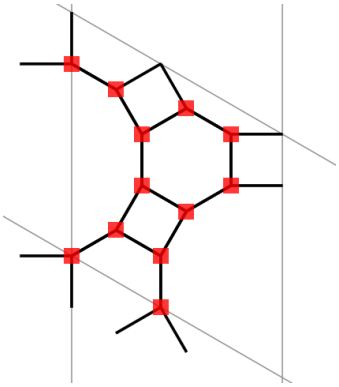
seeds:
$[0,0,0,0]$, $[0,1,0,0]$, $[0,0,0,1]$, $[0,1,0,1]$

basic directions:
ω_1, ω_6, ω_{12}, $\overline{\omega}_{12}$

translations vectors:
$[1,1,2,0]$, $[-1,2,2,-2]$

seeds:
$[0,0,0,0]$, $[0,1,0,0]$, $[0,1,2,0]$, $[-1,2,0,0]$,
$[-1,3,2,0]$, $[-1,2,2,0]$, $[-1,2,1,-1]$,
$[-1,3,3,-1]$, $[0,1,1,0]$, $[-1,2,1,0]$, $[-1,2,2,-1]$,
$[-1,3,2,-1]$

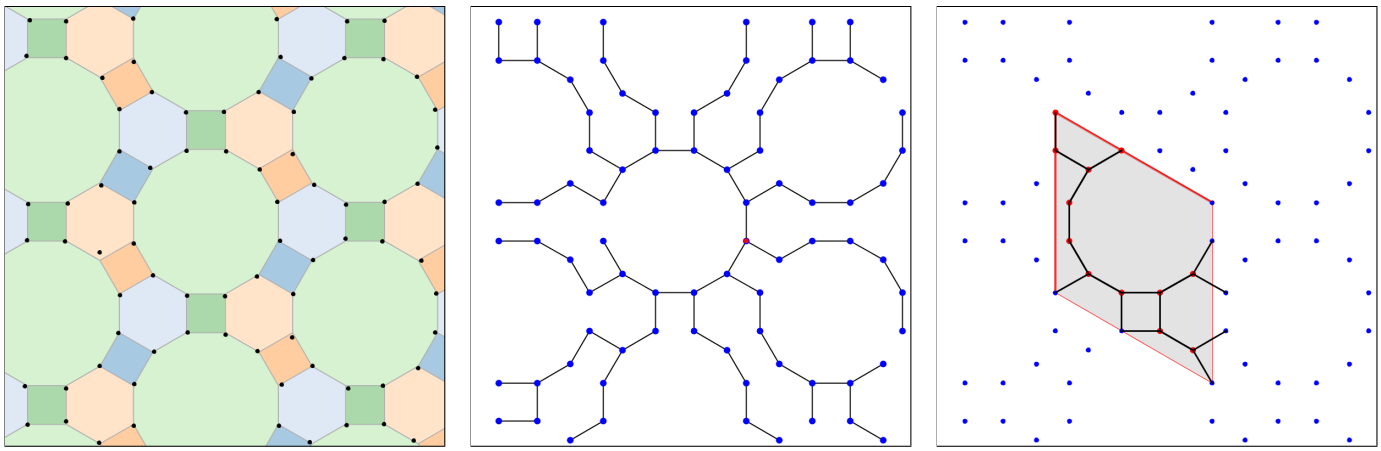



basic directions:
ω_1, ω_6, ω_{12}, $\overline{\omega}_{12}$

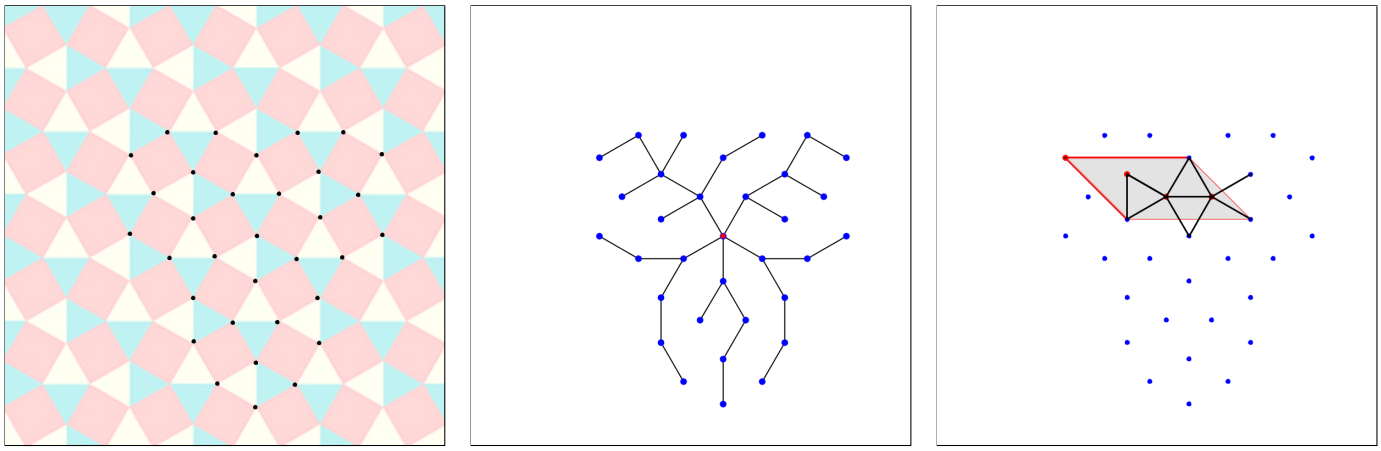
translations vectors:
$[-1,2,3,-3]$, $[2,-1,0,3]$

seeds:
$[0,0,0,0]$, $[0,0,1,0]$, $[0,1,1,0]$, $[0,1,2,-1]$,
$[-1,2,2,-1]$, $[-1,2,2,-2]$, $[0,1,1,1]$, $[0,1,2,1]$,
$[0,0,1,1]$, $[0,0,0,2]$, $[0,1,3,-1]$, $[0,1,3,0]$

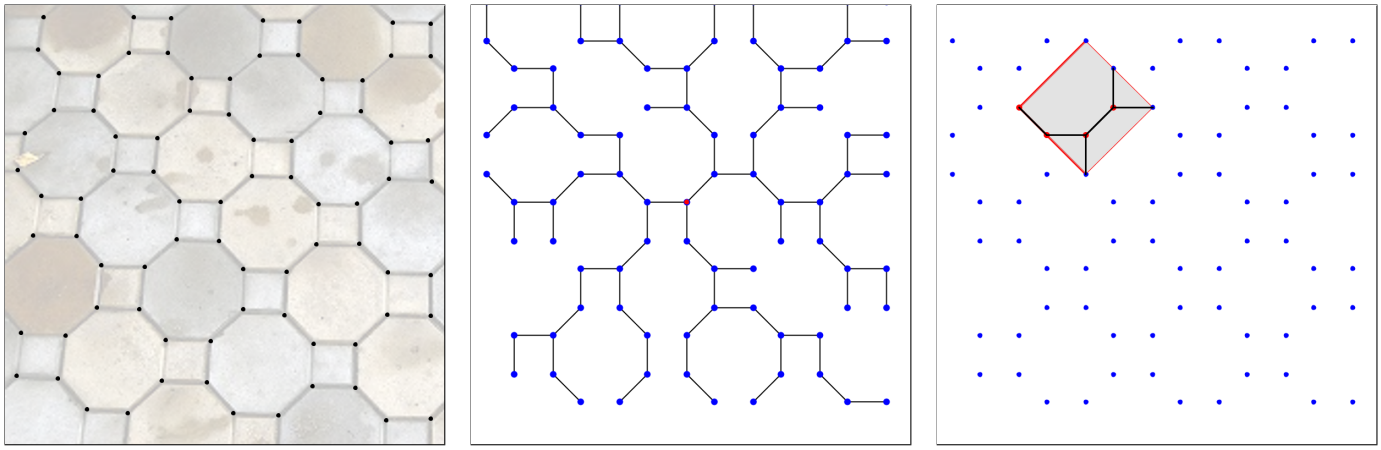
Fig. 9. Some examples: tiling, seeds, representation data. The origin is the lowest leftmost seed.



(a) A synthesized tiling; user selected all vertices, but not very carefully.



(b) A tiling from an image; user selected only a fraction of vertices.



(c) A tiling from a photograph, with perspective distortion.

Fig. 10. Tiling data extracted from a image: vertices selected by user on image, corrected vertices and tree, basic cell and seeds.