The Visual Computer manuscript No.
(will be inserted by the editor)

Acquiring periodic tilings of regular polygons from images

José Ezequiel Soto Sánchez · Asla Medeiros e Sá · Luiz Henrique de Figueiredo

Full paper accepted in CGI 2019 / The Visual Computer. Last revised on March 28, 2019.

Abstract We describe how we have acquired geometrical models of many periodic tilings of regular polygons from two large collections of images. These models are based on a simplification of the representation recently proposed by us that uses complex numbers. We also describe an algorithm for deciding when two representations give the same tiling, which was used to identify coincidences in these collections.

Keywords tilings · tessellations · geometrical models

1 Introduction

Tiling the plane with regular polygons remains a fascinating subject [6,3,10,12], with a venerable history [5] going back 400 years to Kepler's book "Harmonices Mundi" of 1619. Yet, no complete classification of these tilings exists. Crucial to any classification is a representation that can be used to compare tilings for equality. We have recently proposed such a representation [19] but left deciding equality as future work.

In this paper, we review and simplify our original representation in section 2 and in section 3 give an algorithm that exploits the simplified representation for deciding when two representations give the same tiling. In sections 4 and 5 we describe in detail how we have acquired representations for the tilings in two large collections: the one in the catalog by Sá and Sá [20], which contains images of vertex constellations as dots, and the one by Galebach [4], which contains line drawings of tilings. We used our equality decision algorithm to find coincident tilings in these two collections.

José Ezequiel Soto Sánchez and Luiz Henrique de Figueiredo
IMPA, Rio de Janeiro, Brazil

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

2 Representing tilings

To represent a periodic tiling of the plane by regular polygons we focus solely on the translational symmetries of the tiling, purposely ignoring rotations and reflections. Our representation of a tiling contains three pieces of data: a small set of *directions* for the edges, two *translation vectors* that define a fundamental domain or *basic cell* for the tiling, and a set of *seeds* in the basic cell that represent each vertex in the tiling by translations. An example will be shown below. We have argued that this representation is simple to understand and to use to reconstruct the tiling [19].

We start by normalizing the scale, position, and orientation of the tiling by taking the edges to have unit length (all edges have the same length since they are sides of regular polygons) and choosing one of the vertices to be the origin and one of the edges to be horizontal. After this normalization, the edges of the tiling are aligned with the complex n-th roots of unity for $n \in \{1,2,3,4,6,8,12\}$. These are called the *basic directions*. Since there is only one tiling containing octagons (Fig. 1), we shall disregard $n = 8$ here. Thus, the edges of the tiling are aligned with the complex 12-th roots of unity, that is, the powers of $\omega = \exp(\frac{2\pi i}{12}) = (\sqrt{3}+i)/2$. We shall see how this reduction simplifies the representation.

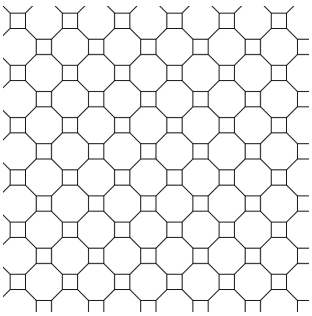

Fig. 1 The only periodic tiling of the plane containing octagons.

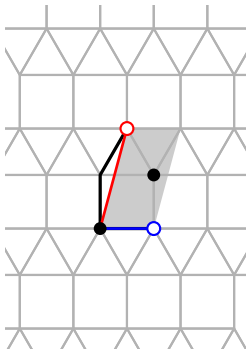

Fig. 2 A tiling and its representation data: translation vectors (in blue and red), basic translation cell (in gray), and seeds (in black) [19].

An example. Consider the tiling in Fig. 2. Our original representation [19] uses three basic directions for the edges: $\omega_1 = 1 = \omega^0$, $\omega_4 = i = \omega^3$, and $\omega_6 = \omega^2$. The translation vectors are $t_1 = \omega_1$ (in blue) and $t_2 = \omega_4 + \omega_6$ (in red). They define a parallelogram, the basic cell (in gray). The seeds are the vertices inside the basic cell: the origin and $\omega_1 + \omega_4$ (in black). Thus, according to our original scheme [19], the data representing this tiling is:

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

As suggested by this example, the key concept in our representation is to represent vertices and translation vectors using *integer linear combinations* of the basic directions. In our original scheme [19], the designer must choose an ordered set of non-redundant basic directions to achieve uniqueness of representation for vertices and translation vectors. Thus, the representation depends on that choice. Comparing representations based on different sets of basic directions is complicated, and was left as future work.

A simplified representation. We simplify the representation of tilings first by noting that the expressions of vertices and translation vectors in terms of basic directions translate into polynomials in ω with integer coefficients. The set of such expressions is traditionally denoted by $\mathbf{Z}[\omega]$. Our simplification is based on the key observation that $\{1, \omega, \omega^2, \omega^3\}$ is an additive basis for $\mathbf{Z}[\omega]$ over $\mathbf{Z}$, because the minimal polynomial of ω is $\omega^4 - \omega^2 + 1$. This allows us to express all powers ω^k for $k \geq 4$ as follows:

$$\begin{aligned}
\omega^4 &= -1 + \omega^2 &&= [-1,0,1,0] \\
\omega^5 &= -\omega + \omega^3 &&= [0,-1,0,1] \\
\omega^6 &= -1 &&= [-1,0,0,0] \\
\omega^7 &= -\omega &&= [0,-1,0,0] \\
\omega^8 &= -\omega^2 &&= [0,0,-1,0] \\
\omega^9 &= -\omega^3 &&= [0,0,0,-1] \\
\omega^{10} &= 1 - \omega^2 &&= [1,0,-1,0] \\
\omega^{11} &= \omega - \omega^3 &&= [0,1,0,-1] \\
\omega^{12} &= 1 &&= [1,0,0,0]
\end{aligned}$$

Thus, all polynomial expressions in ω reduce *uniquely* to cubic expressions. Therefore, we can represent all tilings using a *fixed* set of basic directions: $\{1, \omega, \omega^2, \omega^3\}$. This completely eliminates the choice of basic directions in our original scheme [19]. It also opens the door to the equality decision algorithm, as explained in the next section.

Every vertex in the tiling has a unique expression as an integer linear combination of the basic directions $\{1, \omega, \omega^2, \omega^3\}$. As before [19], this expression is found by following any path connecting the origin to the vertex, adding each edge in the path, and simplifying the sum using the equations above. Alternatively, the sum representing the path is a polynomial in ω and the simplified sum is the remainder of this polynomial modulo the minimal polynomial $\omega^4 - \omega^2 + 1$ (Fig. 3).

In our simplified scheme, the data for the tiling in Fig. 2 is:

basic directions:	1, ω, ω^2, ω^3
translation vectors:	$[1,0,0,0]$, $[0,0,1,1]$
seeds:	$[0,0,0,0]$, $[1,0,0,1]$

because $t_1 = \omega_1 = 1$, $t_2 = \omega_4 + \omega_6 = \omega^2 + \omega^3$, and the second seed is $\omega_1 + \omega_4 = 1 + \omega^3$. Although it contains one more coordinate for each element in this case (but not always), our representation is actually simpler because the set of basic directions is fixed and can be kept implicit. The only data are the coordinates of translation vectors and seeds. Uniform data simplifies further processing.

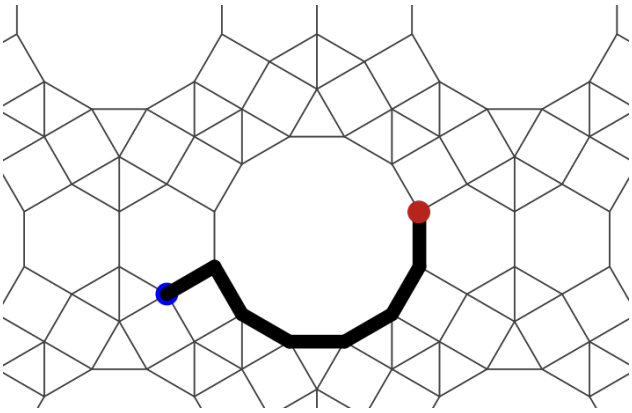

Fig. 3 Paths are polynomials in ω. This path from the blue origin to the red vertex is $\omega + \omega^{10} + \omega^{11} + \omega^0 + \omega + \omega^2 + \omega^3$, which reduces to $2 + 3\omega$. Thus, the red vertex has coordinates $[2,3,0,0]$ in $\mathbf{Z}[\omega]$.

3 Equivalent representations

Even with a fixed set of basic directions, a tiling can have several equivalent representations. Indeed, there is much room for choosing the translation vectors for a given translation grid. This choice affects the basic cell and so the seeds. There is also a little room for choosing which vertex is the origin, because not every tiling is vertex-transitive (the tiling

in Fig. 2 has two types of vertices), and a little room for choosing which edge is horizontal, because not every tiling is edge-transitive (the tiling in Fig. 2 has four types of edges). These two choices affect the seeds and the expression of the vertices as integer linear combinations of the basic directions.

Choice of translation vectors. When do two pairs of translation vectors determine the same translation grid? Take a pair of translation vectors t_1, t_2 and write them as integer linear combinations of the basic directions:

$$t_1 = a_{11}1 + a_{12}\omega + a_{13}\omega^2 + a_{14}\omega^3$$
$$t_2 = a_{21}1 + a_{22}\omega + a_{23}\omega^2 + a_{24}\omega^3$$

In matrix form we get

$$\begin{pmatrix} t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{pmatrix} \begin{pmatrix} 1 \\ \omega \\ \omega^2 \\ \omega^3 \end{pmatrix}$$

or, more concisely, $T = AW$.

Two pairs of translation vectors T and T' determine the same translation grid iff there is an invertible 2×2 integer matrix U such that $T' = UT$. (Note that U is invertible over $\mathbf{Z}$ and so has determinant ± 1. There are infinitely many such matrices U, with arbitrarily large entries.) Write $T = AW$ and $T' = A'W$ as above. Then, $T' = UT$ iff $A' = UA$, because the basic directions in W are linearly independent over $\mathbf{Z}$.

The question now is: Given two 2×4 integer matrices A and A', when is there an invertible 2×2 integer matrix U such that $A' = UA$? There is a classical algebraic tool that decides exactly this problem: the *Hermite normal form* of integer matrices [1]. The Hermite normal form is an integer analogue of the reduced echelon form and can be found with an integer version of Gaussian elimination. More precisely, if A is an $m \times n$ integer matrix, then there is an invertible $m \times m$ integer matrix U such that $H = UA$ is an $m \times n$ integer matrix in row-reduced form, called the Hermite normal form of A.

The key result relevant to our problem is that two pairs of translation vectors $T = AW$ and $T' = A'W$ determine the same translation grid iff the Hermite normal forms of A and A' coincide. This follows from the uniqueness of the Hermite normal form. We have thus solved the first problem.

Choice of horizontal edge. The second problem is how to deal with the choice of which edge is horizontal. Since every edge is aligned with a power of ω, any two edges differ in direction by a power of ω, and this corresponds to a multiplication by ω^k for some k. Therefore, we can compute the Hermite normal form of AM^k, where M is the matrix of the multiplication by ω in the basis $\{1, \omega, \omega^2, \omega^3\}$:

$$\omega \begin{pmatrix} 1 \\ \omega \\ \omega^2 \\ \omega^3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ \omega \\ \omega^2 \\ \omega^3 \end{pmatrix} = MW$$

We try all 12 possibilities, $k = 0, \ldots, 11$. If the Hermite normal forms of AM^k and A' coincide for some k, then the two representations use the same grid after a rotation.

Choice of origin. Once we have determined that the translation grids of two representations are the same, up to a rotation M^k, it remains to test whether the vertices are the same. This depends on the seeds. Two representations having the same translation grid and seed sets S_1 and S_2 of the same size give the same tiling iff there is a seed $s_0 \in S_1$ such that, for each seed $s_1 \in S_1$, $s_1 - s_0$ is mapped under M^k to a vertex generated by S_2. This test uses their coordinates in $\mathbf{Z}[\omega]$.

4 Acquiring tilings from images

We have acquired representations for the tilings in two large collections. The collection by Sá and Sá [20] contains 213 images of vertex constellations as colored dots, such as the one in Fig. 4, which is a 189×183 PNG image. The dots have diameter about 8 pixels. The collection by Galebach [4] contains 1351 images with low-resolution line drawings of tilings, such as the one in Fig. 5, which is a 768×576 PNG image. The lines are 1-pixel wide and have no antialiasing.

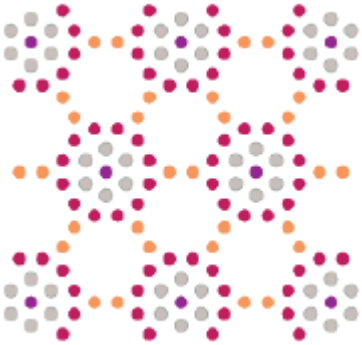

Fig. 4 Sample input image: tiling GMUW from Sá and Sá [20].

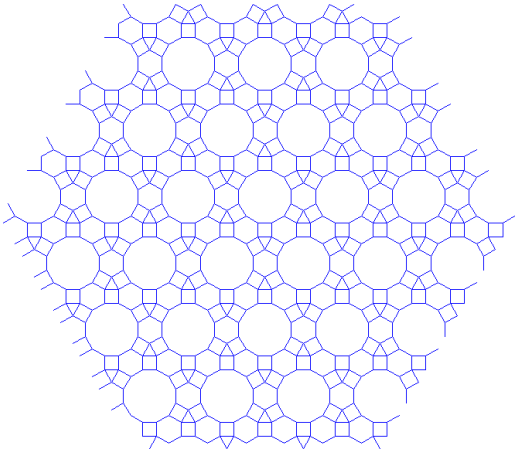

Fig. 5 Sample input image: the 2-uniform tiling 1 from Galebach [4].

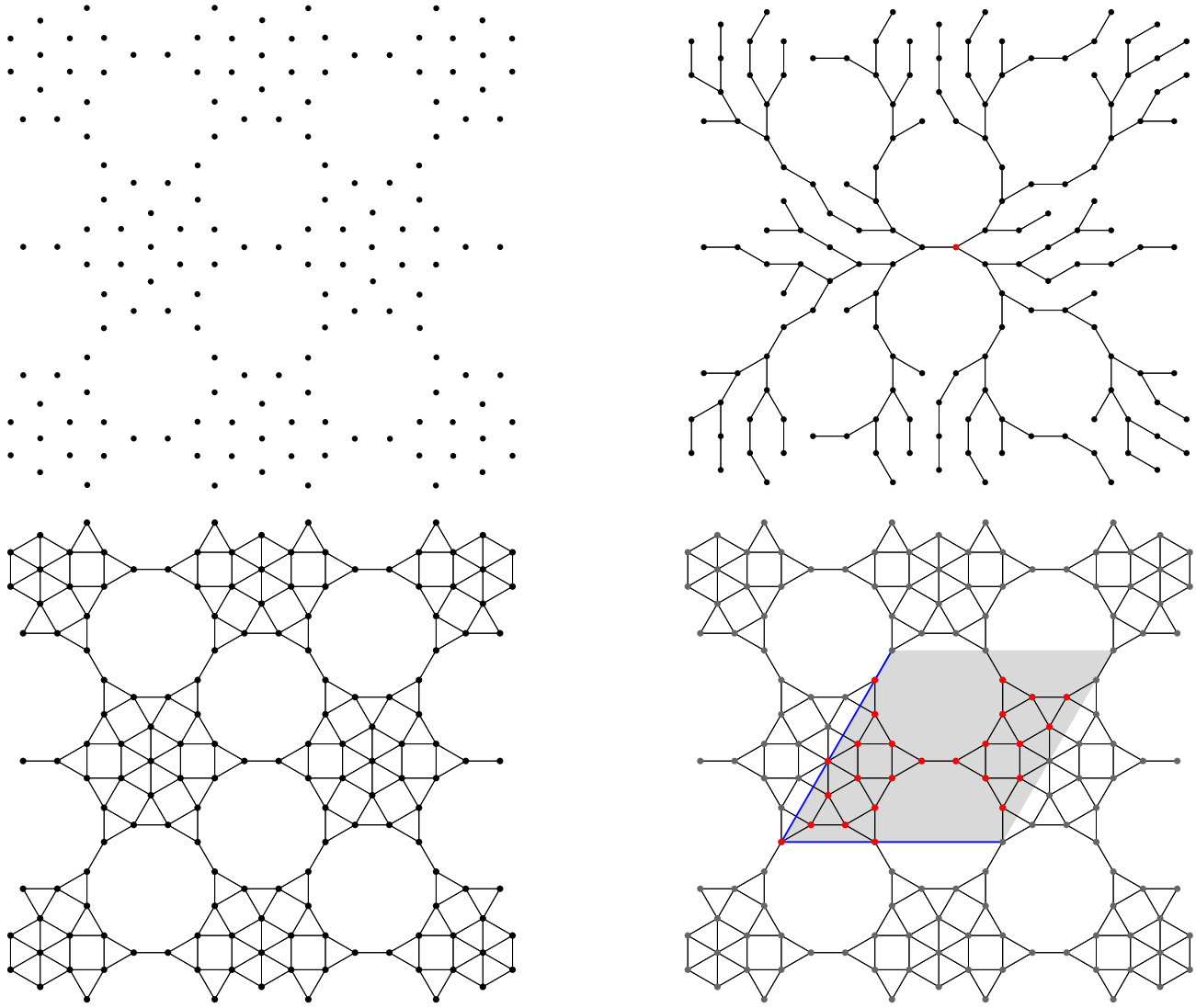

Fig. 6 Our pipeline. From top to bottom, left to right: vertices extracted from image; spanning tree; stars; basic cell, translation vectors, and seeds.

We followed the pipeline below for acquiring representations of tilings from images, as suggested before [19].

1. Find approximate coordinates for the vertices in the tiling.
2. Correct the vertices so that the edges have unit length and are aligned with the basic directions.
3. Find the edges.
4. Find the translation vectors.
5. Find the seeds.
6. Find minimal translation vectors and seed sets.

We give below complete details for each step. Fig. 6 illustrates this acquisition pipeline for the input image in Fig. 4.

The pipeline relies on our simplified representation of the vertices as points in $\mathbf{Z}[\omega]$, which are found in step 2. As mentioned before [19], the vertices are the central elements of the tiling. Edges and faces are easily deduced from the vertices, since the tiling is composed of regular polygons. This approach is reminiscent of the regular systems of points discussed by Hilbert and Cohn-Vossen [8].

Acquiring vertices. The goal of this step is to find approximate coordinates for the vertices in the image. This is the only step that depends on the nature of the input. We use standard image processing to identify and extract the vertices from the input image.

For vertex constellations, we found the vertices by finding the centroids of the connected components in the image, using *regionprops* in MATLAB. We had to rotate a few images to ensure that every image had a horizontal edge.

For line drawings, we convolved the input image with a 3×3 kernel of ones. In this new image, we selected the pixels having value at least four. The corresponding pixels

in the original image surround a vertex, where at least three lines meet. Recall that lines are 1-pixel wide and have no antialiasing. We then found the vertices by finding the centroids of the connected components in the new image, as before.

Finally, we ensured that all implied edges have approximately unit length by scaling the extracted points about their centroid with a scale of $1/d$, where d is the distance from the origin to its nearest neighbor. The origin is the point nearest to the centroid of the point cloud.

Correcting vertices. The approximate (x,y) coordinates for the vertices found in the first step are corrected in the second step by enforcing the restriction that the edges have unit length and are aligned with the basic directions. The output of this step are exact coordinates in $\mathbf{Z}[\omega]$ for the vertices in the tiling. When needed, the corresponding Cartesian coordinates are easily found from the Cartesian coordinates of the basic directions $\{1,\omega,\omega^2,\omega^3\}$, which are known exactly to any precision required.

We build a spanning tree T for the vertices using front propagation as follows. The tree T and the front F start with the origin, which is the vertex closest to the centroid of the vertices. While F is not empty, we remove a vertex $v \in F$ and find all vertices w not in T such that $1-\delta < d(v,w) < 1+\delta$. The edge vw is added to the tree T and the vertex w is added to the front F. Ideally, we seek those w such that $d(v,w) = 1$, but we need to use a tolerance δ because the vertex coordinates coming from the first step are approximate. The comfortable value $\delta = 0.35$ worked for all 1564 vertex clouds. This large tolerance attests to how robust this step is.

For each edge vw, we store for w its coordinates in $\mathbf{Z}[\omega]$, which are given by $v+\omega^k$, where ω^k is the direction closest to vw. (Not having to deal with edges from octagons here significantly helps robustness, since the candidate edges are better separated.) Thus, if $v = [a_1,a_2,a_3,a_4]$ and $\omega^k = [b_1,b_2,b_3,b_4]$, as given in section 2, then $w = [a_1+b_1,a_2+b_2,a_3+b_3,a_4+b_4]$. The origin is of course $[0,0,0,0]$ in $\mathbf{Z}[\omega]$.

The spanning tree T is just a tool for systematically visiting all vertices and finding their coordinates in $\mathbf{Z}[\omega]$. These coordinates for a vertex v correspond to the unique path in T from the origin to v. The tree T is not needed in the other steps and is discarded.

Finding edges. We find the edges of the tiling by finding the *star* of each vertex v, that is, the list of vertices w such that vw is an edge of the tiling, ordered circularly around v (Fig. 7). Although this finds each edge twice (once as vw and once as wv), having the stars of the vertices is useful when finding the translations vectors, as we shall see.

Since each edge vw is aligned with some ω^k, we represent the star of v by the ordered list of the exponents that appear in these edges (Fig. 7). To avoid testing all possible pairs of

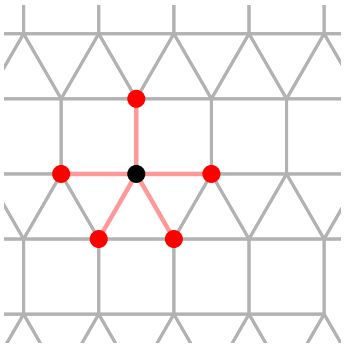

Fig. 7 The red vertices form the star of the black vertex. The edges are $\omega^0,\omega^3,\omega^6,\omega^8,\omega^{10}$ and the star is represented by the list $[0,3,6,8,10]$.

vertices to find the edges, we store the vertices in a hash table indexed by their coordinates in $\mathbf{Z}[\omega]$, found in the previous step. Then, to find the edges around a vertex v, we simply test whether $v+\omega^k$ is in the hash table. If so, we add k to the star of v. We test $k = 0,\ldots,11$ and so the list of exponents is already ordered. Note again that $v+\omega^k$ is computed in $\mathbf{Z}[\omega]$, using the reductions given in section 2.

Finding translation vectors. In this step, we find two translation vectors that define a basic cell for the tiling.

A translation vector t must induce an automorphism of the tiling seen as a graph, that is, it must send every vertex v to another vertex $v' = v+t$ and it must respect edges. However, this is true only if the vertex cloud is infinite because there are no automorphisms induced by translations on a finite cloud. When the vertex cloud is finite, we must settle for a *partial* automorphism: some vertices and edges will be sent outside the cloud, and some vertices and edges in the cloud will not be the image of any vertex or edge. Therefore, even efficient algorithms for graph automorphisms [15,11] cannot find translations, only rotations and reflections.

Note that the problem of finding translation vectors is *not* the standard cloud registration problem in computer vision, which deals with *two* clouds of points [7]. In that context, the problem is solved by finding the best rigid transformation that maps one cloud to the other, in the sense of least squares. However, we are looking for *exact* solutions, not *approximate* ones. Exact solutions are possible because the vertices have exact integer coordinates in $\mathbf{Z}[\omega]$, and so do translation vectors. Our solution is exact in this sense; it does not use the (x,y) coordinates of the vertices, which are necessarily approximate, since they involve $\sqrt{3}$.

To find the translation vectors, we need to analyze the translational symmetries of the vertex cloud. We try all possible translations and pick the best ones. Here are the details.

A translation vector must send the origin to another vertex *of the same type* in the tiling. A necessary condition for two vertices to have the same type is that they have the same star, hence the computation in the previous step. To each vertex v

having the same star as the origin, we assign a score to the translation vector t that sends the origin to v:

$$\text{score}(t) = \#((V \pm t) \cap V)$$

Here, V is the vertex cloud and $V \pm t$ is the result of translating V by t in both directions. A vertex w is counted in this score iff $w+t$ or $w-t$ is a vertex $w' \in V$ and w and w' have the same star. This score measures how much of the tiling is preserved under translation by t. We use coordinates in $\mathbf{Z}[\omega]$ to test whether $w \pm t \in V$ with the hash table.

We order the pairs $(\text{score}(t), \text{length}(t))$ lexicographically in order of decreasing score and increasing length, and select the top two translation vectors that are linearly independent. Including lengths in this selection helps to find a small basic cell.

Finding seeds. The seeds are the vertices inside the basic cell determined by the translation vectors t_1 and t_2 found in the previous step. The basic cell is $\{\lambda_1 t_1 + \lambda_2 t_2 : \lambda_1, \lambda_2 \in [0,1)\}$. Note that this cell is *half-open*, that is, open at the sides not containing the origin.

For each vertex v in the tiling, we find its coordinates (λ_1, λ_2) in the basis (t_1, t_2) by solving a standard 2×2 linear system. This step uses the (x, y) coordinates of v, t_1, t_2. Then v is a seed iff $-\varepsilon \leq \lambda_1, \lambda_2 \leq 1 - \varepsilon$. This tests ensures that we find all seeds, including the ones on the closed sides of the basic cell, but not the ones on its open sides. The value $\varepsilon = 10^{-6}$ worked well in all cases.

Finding minimal translation vectors. The translation vectors, the basic cell, and the set of seeds found in the previous steps may not be the smallest possible. This is due to the size and shape of the vertex cloud. Nevertheless, if we have succeed in reproducing almost all vertices in the cloud by translating the seeds inside the basic cell along the translation vectors, then we have acquired the tiling correctly. In particular, we can generate arbitrarily large vertex clouds for the tiling.

We find minimal translation vectors and seed sets by running steps 3–5 again on a synthetic vertex cloud generated in a 5×5 grid of translation cells and insisting that the candidate translations reproduce *all* vertices in the 3×3 subgrid.

5 Results

We applied the pipeline described in section 4 to the tilings by Sá and Sá [20] and by Galebach [4]. For each tiling, we verified how well the tiling reconstructed from our acquired representation matched the original input data. More precisely, we computed how many corrected vertices appeared in the vertex cloud obtained by translating the seeds along the translation vectors to cover the input cloud.

Table 1 Verification results for the two collections of tilings.

Sá and Sá:	% success	100	99	98	96	49
	# tilings	195	9	7	1	1
Galebach:	% success	100	99			
	# tilings	1346	5			

The results are shown in Table 1, which should be read as follows: In the collection of 213 tilings by Sá and Sá, 195 tilings were perfectly reconstructed, 9 tilings had at least 99% success but less than 100%, 7 tilings had at least 98% success but less than 99%, 1 tiling had at least 96% success but less than 97%, 1 tiling had at least 49% success but less than 50%. In the collection of 1351 tilings by Galebach, the results are even better, probably because their clouds are larger: 1346 tilings were perfectly reconstructed and the remaining 5 tilings had at least 99% success.

In summary, these results show over 99% success overall for the tilings by Sá and Sá: all but one tiling were perfectly reconstructed except possibly for a few vertices (6 in the worst case). For the tilings by Galebach, over 99% of the tilings were perfectly reconstructed with no exceptions; the five remaining tilings were perfectly reconstructed except for a few vertices (3 in the worst case). In both collections, the exceptional vertices that were not reconstructed are typically at the fringe of the original cloud and have been acquired erroneously because they have very few neighbors and so the wrong star.

The only example where our algorithm found an incorrect translation cell is the tile named PTU6 by Sá and Sá (Fig. 8). This failure is due to input data that is insufficient to recognize the full periodicity of the tiling, even for humans. Despite appearances, the input image (top) does not have enough information to allow our algorithm to find the correct vertical translation, because the neighborhood of the center hexagon is never fully replicated in the image. The hexagons in the top and bottom rows are equivalent under a vertical translation, but they are not equivalent to the hexagons in the middle row. This is a subtle point even for a human: notice how the yellow vertices are distributed differently around the hexagons in the center column. Our algorithm did find one correct translation (bottom).

Finding coincidences. We used the equality decision algorithm described in section 3 to detect coincidences in the two collections. To avoid testing all pairs of models, we only tested pairs with the same number of seeds and basic translation cells of the same area. These are necessary but not sufficient conditions for two representations to define the same tiling. We tested 364 potential coincidences, having the same number of seeds, area, and Hermite normal form. We found 143 coincidences, some of which are not obvious because they involve rotations. Fig. 9 shows an example.

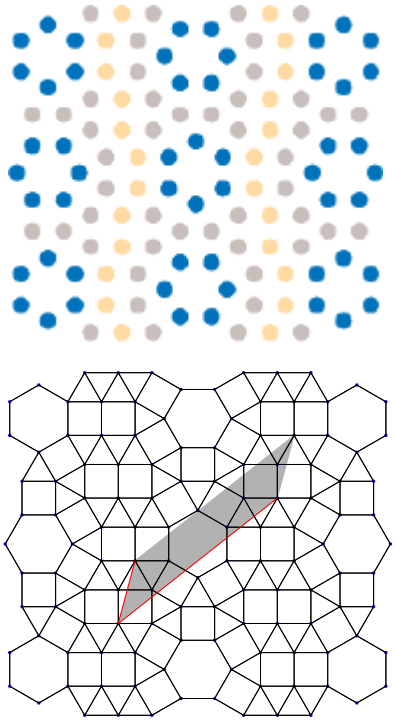

Fig. 8 The input image does not contain sufficient information for reconstructing the tiling. The neighborhood of the center hexagon is not fully replicated in the image.

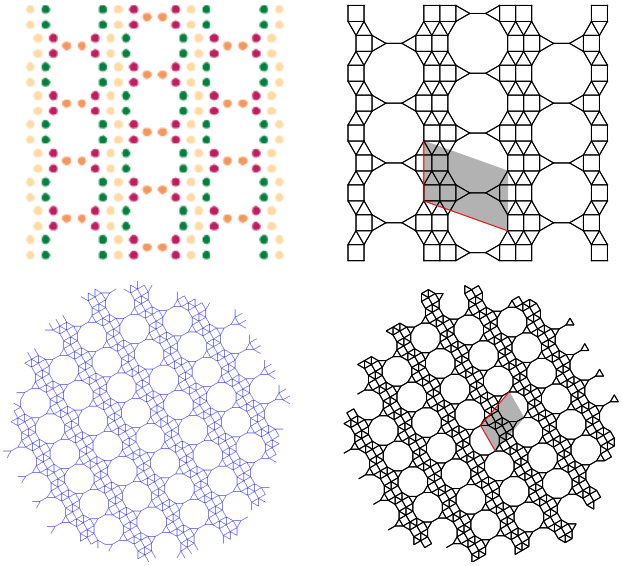

Fig. 9 Coincidences in the two catalogs: the tiling in the top row is the same as the tiling in the bottom row, after a rotation. Input images on the left, reconstructed tilings on the right.

After a rotation, the tiling GLMT2 in the collection by Sá and Sá [20] is the same as the 4-uniform tiling 37 and the 4-archimedean, 4-uniform tiling 16 in Galebach [4].

6 Related work

This paper is a significant continuation of our recent work [19], where we proposed a representation for tilings but used it mainly for synthesis and rendering. In hindsight, that representation is suitable for designing and acquiring tilings manually, because of the designer can freely choose a set of non-redundant basic directions. After discarding the only tiling containing octagons (Fig. 1), we have simplified the representation to use the natural additive basis of $\mathbf{Z}[\omega]$ as a fixed set of basic directions. This simpler and more elegant representation is a key tool in the automatic acquisition of tilings from images, while remaining easy and natural for manual acquisition.

An important class of tilings by regular polygons are k-uniform tilings, whose vertices form k equivalence classes with respect to the symmetries of the tiling. The enumeration of all k-uniform tilings for a given value of k is far from trivial. Lenngren [12] reviewed the most important steps in the investigation of k-uniform tilings, summarizing the efforts found in literature on classifying tilings by regular polygons, and pointing out directions of research in the area.

The collection of tilings by Galebach [4] is the state of the art in the classification of k-uniform tilings [21]. Only low-resolution images with line drawings of tilings are available at that site, but unfortunately not vertex coordinates or code.

Efforts in producing catalogs on tilings of regular polygons are recurrent in the literature. In 1989, Chavey [2] collected results and drawings of regular tilings, arguing that classifications and theoretical results on the topic are scattered across several papers. Chavey provides drawings of 165 tilings of regular polygons, arranged by vertex orbits and labelled according to vertex types. Recently, Wikipedia included a catalog of 564 tilings [22] that cite Chavey [2] and Galebach [4] as sources. The SVG images in Wikipedia contain coordinates that could be extracted and used in our pipeline. However, the tilings in Wikipedia are a subset of those by Galebach [4] and have fewer vertices.

Liu et al. [14] carefully reviewed the field of computational symmetry, including the long history of symmetry detection algorithms, dating back to 1932. They mention that detection of reflection symmetry used to dominate the field of symmetry detection in computer vision. Their review classifies methods by the types of symmetries detected, as well as the type of method used for detection. We stress that we are interested only in translations, even though periodic tilings of regular polygons have several types of symmetries. Thus, our problem falls into the class of lattice detection algorithms. Liu et al. [14] classified a few papers as lattice detection. All of them base their search for translation direction in the frequency space. Our approach is based on voting schemes,

which are more frequently used to detect rotations and reflections. We search for global translational symmetry although we have necessarily only local information in hand.

Liu et al. [13] investigate the problem of automatically inferring the lattice structure of near-regular textures in real-world images. The first step toward automatic texel discovery is the detection of repeated interest points present in the image. The key trade-off is to extract enough interest points to reveal some repeated structure reliably without overwhelming the subsequent lattice finder with false positives. Liu et al. [13] report good performance for their algorithm by testing it on the data used in Symmetry Detection from Real-World Images Competition 2013. The problem that we solve here is different for two main reasons: (1) we seek translations in $\mathbf{Z}[\omega]$ and search in a few directions; (2) our data is composed of several interlocked lattices that share same translation grid. This would most probably be a challenging dataset for previous algorithms.

Kaplan [9] presented a method for rendering Islamic patterns on top of a tiling of the plane. The method is based on Hankin's "polygons-in-contact" technique, which dates back to 1920's. The method can be directly applied on top of the periodic tilings by regular polygons that we have acquired, achieving analogous results. All the derived products, such as 3D printings of ornaments, can also be produced out of the tilings acquired here, as well as their duals.

Sá et al. [18] described how, given a volume boundary, to find a low-density internal mesh that is 3D-printable. Their method exploits the relationship between primal and dual tessellations, and has two steps, The first step defines a cell complex partition for the internal space of the volume. The second step applies the *Skin4Skeleton* algorithm, which uses the dual cell complex to produce a 3D-printable cell-complex mesh with a parametrized thickness. The same idea is useful for producing an offset to a polygonal tiling with parameterized thickness, and so can be applied for pre-processing tilings in order to send them to a laser cutter machine.

Nasri and Benslimane [16] proposed a modeling method to automatically generate periodic Moorish geometric patterns. Their approach is to construct a periodic pattern using isometric transformations of its template motif. They identify the shapes vocabulary characterizing the Moorish style by analyzing a dataset of historical Moorish patterns. The research context of their paper shares the same background as the studies and classifications available in literature for tilings of regular polygons. This fact is not surprising since the symmetry structure underlying both are the same. In the case of periodic tilings of regular polygons, the pattern structure is so rigid that there is little room for playing with shape grammars, but the overall structure of this paper on Moorish patterns has the same flavor as the one proposed here, in the sense that it extracts shape knowledge out of a collection of given images in order to permit synthesis of periodic patterns.

7 Conclusion

We have described a uniform representation for periodic tilings of regular polygons that simplifies our original representation [19]. We have also described in detail a pipeline to acquire such representations from images, which we used on two large collections of tilings [4,20]. This pipeline is quite robust and can be easily adapted to handle other kinds of images, by using suitable image processing to extract vertices. Previously [19], we had the user select the vertices directly on the image, which could even be photographs with mild perspective distortions. The pipeline can also be used to handle vector graphics, such as SVG and PostScript, as long as it is possible to extract vertex coordinates, even in different coordinate spaces.

The data acquired from the two collections of tilings will be available at our project web page [17] for further research in the topic. The data are JSON files containing the name of the tiling and the coordinates in $\mathbf{Z}[\omega]$ of its translation vectors and seeds. These coordinates are very small integers (less than 15 in absolute value). The file for the whole collection by Sá and Sá [20] has about 64K bytes; the file for the collection by Galebach [4] has about 343K bytes.

Acknowledgements We thank Sá and Sá and Galebach for making their collections of tilings freely available at their websites. The first author is partially supported by a CNPq doctoral scholarship. The third author is partially supported by a CNPq research grant. This research was done in the Visgraf Computer Graphics laboratory at IMPA. Visgraf is supported by the funding agencies FINEP, CNPq, and FAPERJ, and also by gifts from IBM Brasil, Microsoft, NVIDIA, and other companies.

Compliance with ethical standards Conflict of Interest: The authors declare that they have no conflict of interest.

References

1. Bradley, G.H.: Algorithms for Hermite and Smith normal matrices and linear Diophantine equations. Mathematics of Computation **25**, 897–907 (1971)
2. Chavey, D.: Tilings by regular polygons. II. A catalog of tilings. Computers & Mathematics with Applications **17**, 147–165 (1989)
3. Conway, J.H., Burgiel, H., Goodman-Strauss, C.: The symmetries of things. AK Peters (2008)
4. Galebach, B.: *n*-uniform tilings. probabilitysports.com/tilings.html
5. Grünbaum, B., Shephard, G.C.: Tilings by regular polygons. Mathematics Magazine **50**, 227–247 (1977)
6. Grünbaum, B., Shephard, G.C.: Tilings and patterns. W. H. Freeman (1989)
7. Hartley, R.I., Zisserman, A.: Multiple view geometry in computer vision. Cambridge University Press (2004)
8. Hilbert, D., Cohn-Vossen, S.: Geometry and the imagination. Chelsea (1952)
9. Kaplan, C.S.: Islamic star patterns from polygons in contact. In: Proceedings of the Graphics Interface 2005, pp. 177–185 (2005)
10. Kaplan, C.S.: Introductory tiling theory for computer graphics. Synthesis Lectures on Computer Graphics and Animation **4**(1), 1–113 (2009)

11. Kawarabayashi, K.i., Mohar, B.: Graph and map isomorphism and all polyhedral embeddings in linear time. In: STOC'08, pp. 471–480. ACM (2008)
12. Lenngren, N.: k-uniform tilings by regular polygons. Tech. Rep. U.U.D.M. project report 2009:23, Uppsala University (2009)
13. Liu, S., Ng, T., Sunkavalli, K., Do, M.N., Shechtman, E., Carr, N.: PatchMatch-based automatic lattice detection for near-regular textures. In: Proceedings of ICCV 2015, pp. 181–189 (2015)
14. Liu, Y., Hel-Or, H., Kaplan, C.S., Gool, L.J.V.: Computational symmetry in computer vision and computer graphics. Foundations and Trends in Computer Graphics and Vision **5**(1-2), 1–195 (2010)
15. McKay, B.D., Piperno, A.: Practical graph isomorphism, II. Journal of Symbolic Computation **60**, 94–112 (2014)
16. Nasri, A., Benslimane, R.: Parametric shape grammar formalism for Moorish geometric design analysis and generation. Journal on Computing and Cultural Heritage **10**, 1–20 (2017)
17. Soto Sánchez, J.E., Sá, A.M., de Figueiredo, L.H.: Periodic tilings of regular polygons. www.impa.br/~cheque/tiling/
18. Sá, A.M., Echavarria, K.R., Arnold, D.: Dual joints for 3d-structures. The Visual Computer **30**(12), 1321–1331 (2014)
19. Sá, A.M., de Figueiredo, L.H., Soto Sánchez, J.E.: Synthesizing periodic tilings of regular polygons. In: Proceedings of SIBGRAPI 2018, pp. 17–24. IEEE Computer Press (2018)
20. Sá, R., Sá, A.M.: Sobre malhas arquimedianas. Editora Olhares (2017)
21. The On-Line Encyclopedia of Integer Sequences: A299780. oeis.org/A299780
22. Wikipedia: Euclidean tilings by convex regular polygons. en.wikipedia.org/wiki/Euclidean_tilings_by_convex_regular_polygons