Vamshi Jandhyala

Islamic Geometric Patterns

✠

January 2020

A computational approach to the star patterns of the Islamic decorative tradition, after Bonner.

Star pattern construction

The goal of this handout is a computational pipeline that produces star patterns of the kind that decorate the walls of the Alhambra and the Imam mosque in Isfahan. The construction has three phases: build a translational unit that tiles the plane, choose a motif for each prototile within that unit, and decorate the line segments that result.

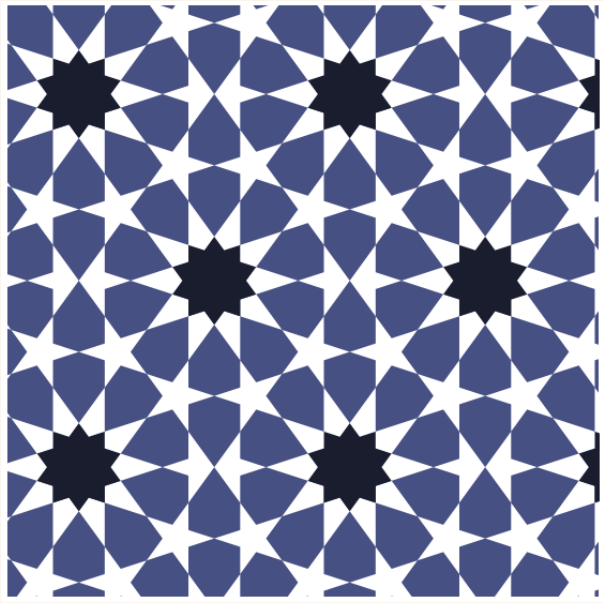
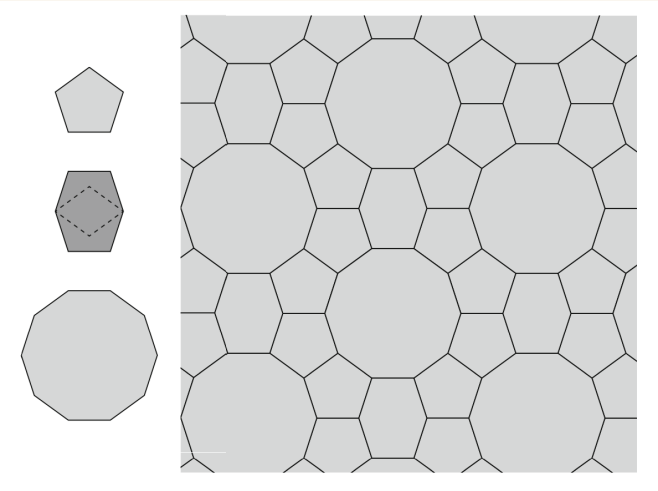


Figure 1: *
A target output: a tenfold star pattern.

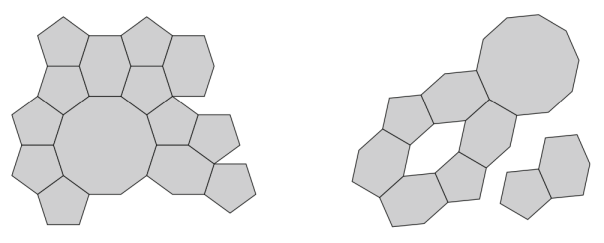
Patches and tilings

Let $\mathcal{T} = \{\mathcal{P}_1, \mathcal{P}_2, \dots\}$ be an infinite collection of polygons. We say $\mathcal{T}$ is a tiling of the plane if its polygons cover the plane with no gaps and no interior overlaps. The

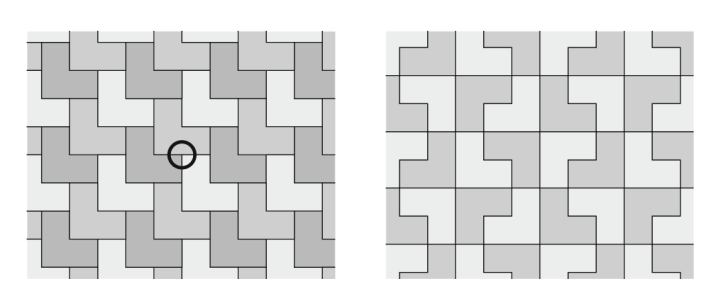
figure below shows a tiling by regular decagons, regular pentagons, and irregular barrel-shaped hexagons.



A *patch* is a finite collection of non-overlapping polygons whose union is a single connected region with no internal holes.



When two polygonal tiles meet, a vertex of one polygon may lie somewhere along an edge of the other. Such an arrangement is a *T-junction*. A patch or tiling is *corner-to-corner* if it contains no T-junctions. We will need to watch out for tilings that are not corner-to-corner when constructing motifs.



Periodic tilings

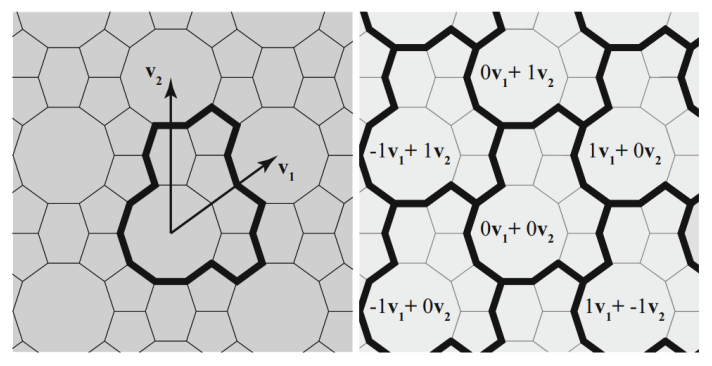
A *rigid motion* of the plane is a symmetry of a tiling if every transformed tile lies directly atop some untransformed tile. A tiling is *periodic* if its symmetry group includes

translations in two non-parallel directions; these translations form a two-dimensional family through which the tiling repeats across the entire plane.

Periodicity implies redundancy. A periodic tiling can be reduced to:

- a finite patch of tiles, called a *translational unit*;
- two vectors $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$.

The full tiling is recovered by stamping translated copies of the unit by $av_1 + bv_2$ for all integers a, b. The translational unit must be *non-redundant* (no tile is a copy of another translated by v_1 or v_2) and *maximal* (enough tiles for the construction to leave no holes).



Representing a translational unit

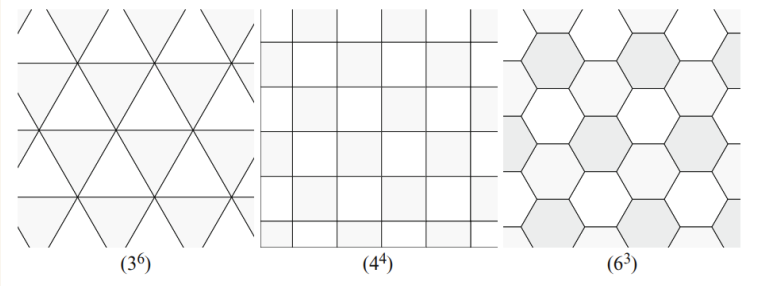
A translational unit is stored as:

- an array of k distinct *prototiles*, each in a convenient local coordinate system;
- an array of *placed prototiles* — pairs of (index into the prototile array, 3×3 similarity matrix that places the prototile in its position).

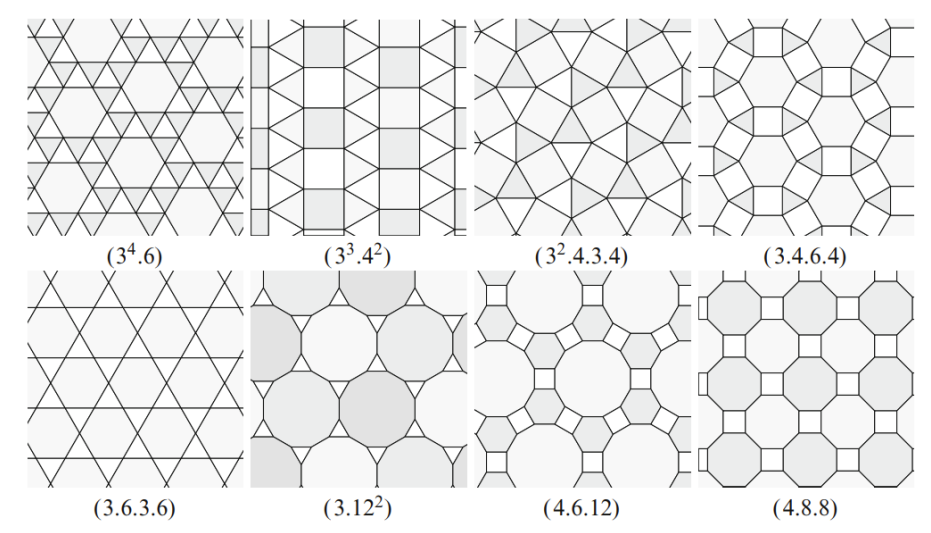
Each unique prototile shape is filled with a motif once; the resulting motifs are then placed in the final pattern using the stored transformation matrices.

Regular and Archimedean tilings

The simplest tilings of the plane are corner-to-corner tilings by congruent regular polygons. There are exactly three: by equilateral triangles, by squares, and by regular hexagons.



We name these by the period-delimited list of polygon sizes met around each vertex: $(3.3.3.3.3.3) = (3^6)$, $(4.4.4.4) = (4^4)$, and $(6.6.6) = (6^3)$. If we relax the constraint to allow more than one polygon shape, but still require every vertex to be described by the same word, we obtain the eight *Archimedean* tilings; together with the three regular ones they form the eleven canonical tilings used in Islamic design.



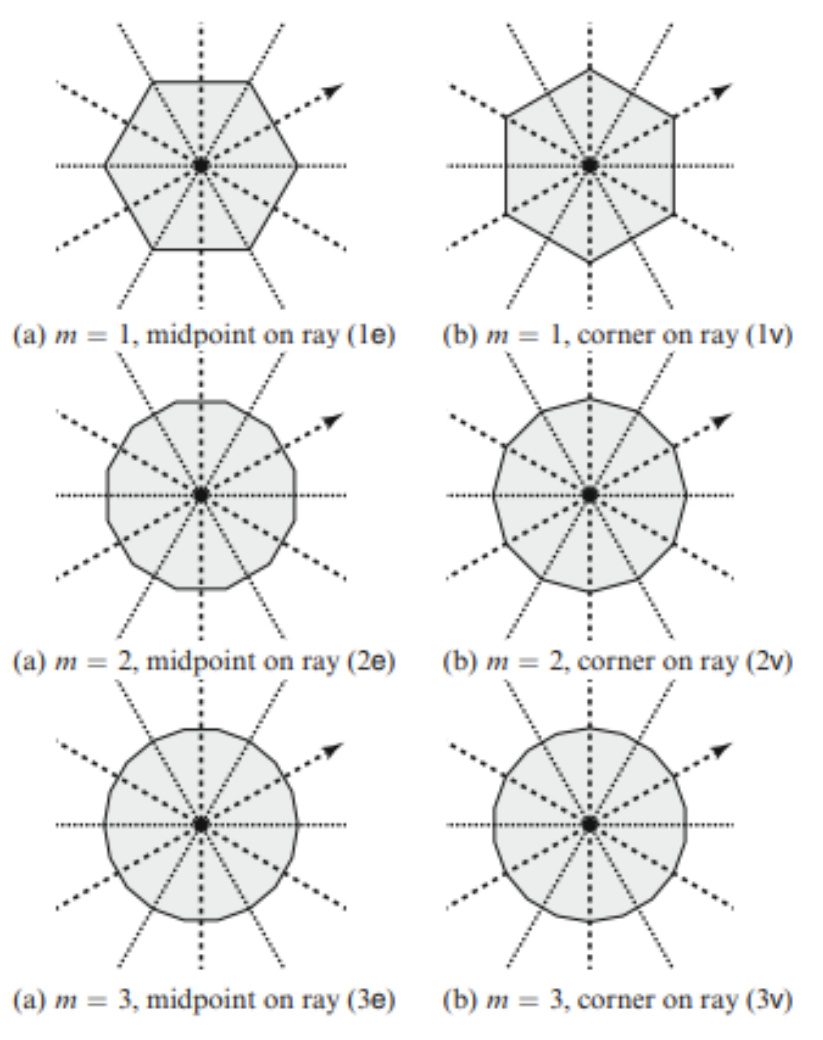
Axis-based construction of tilings

This technique identifies points where regular polygons may be centred, then scales the polygons until they link up to form a tiling. Take the regular tiling (6^3). Each hexagon centre is the focal point of $360°/6 = 60°$ rotational symmetry and is crossed by six lines of reflection.

Generally, any point that acts as the centre of a $360°/n$ rotational symmetry is an *n-fold axis*. Each hexagon centre is a sixfold axis; hexagon corners are threefold axes; edge midpoints are twofold axes.

Construct a 30–60–90 triangle from a hexagon centre, the midpoint of one of its edges, and the corresponding vertex; we call this triangle a *flag*. Label the corners A (the 30° corner, at hex centre), B (the 90° corner, at edge midpoint), and C (the 60° corner, at hex vertex). On each axis we place a regular polygon whose orientation is summarised by an integer multiplier and a corner-or-edge marker.

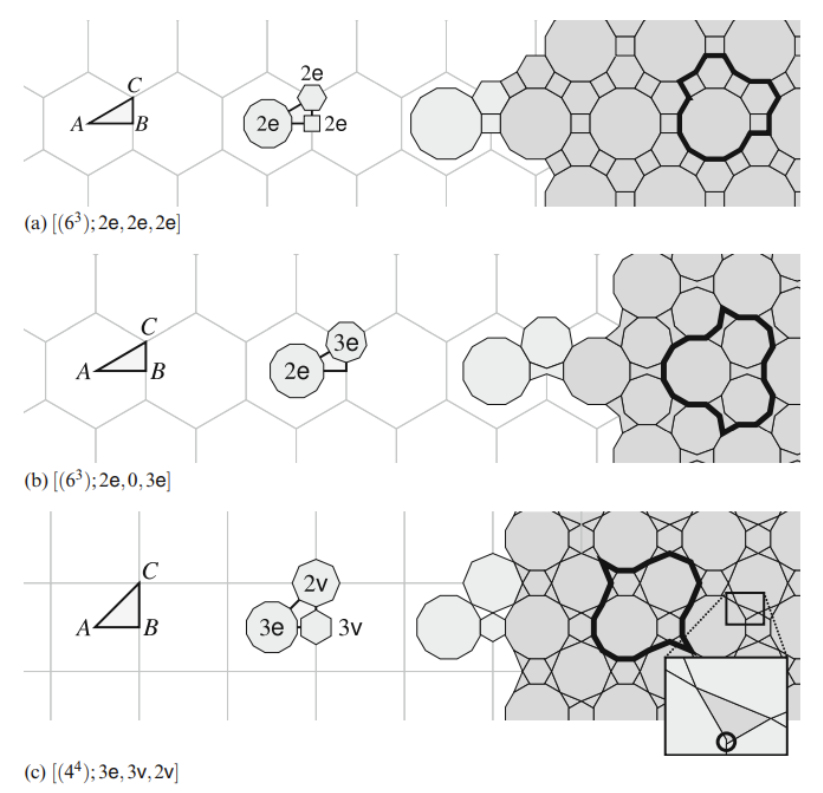
Letting m_A, m_B, m_C be the multipliers and $o_A, o_B, o_C \in \{e, v\}$ the orientations, the construction is denoted $[(6^3); m_A o_A, m_B o_B, m_C o_C]$. To complete the tiling we scale the polygons until they meet, filling residual holes with irregular tiles.



(a) $m = 1$, midpoint on ray (1e) (b) $m = 1$, corner on ray (1v)

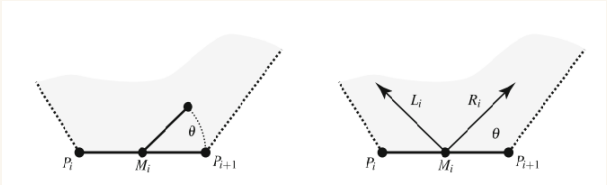
(a) $m = 2$, midpoint on ray (2e) (b) $m = 2$, corner on ray (2v)

(a) $m = 3$, midpoint on ray (3e) (b) $m = 3$, corner on ray (3v)



Motifs

Given a periodic tiling, we elaborate a line-based motif inside each unique tile. Choose a contact angle $\theta \in (0°, 90°)$. At the midpoint of each edge of a tile draw two rays into the interior, each making an angle θ with the edge. If the corners of an n-gon are $P_1, \ldots, P_n$ then we denote the two rays at the midpoint M_i of P_iP_{i+1} by L_i (left ray) and R_i (right ray).



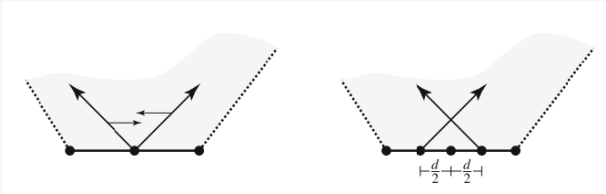
The fivefold-system tradition uses three named contact angles, called the *acute*, *median*, and *obtuse* families: 72°, 54°, and 36° respectively.

Regular polygons

For a regular n-gon inscribed in a unit circle, intersect R_i with L_{i+1} for each i; the intersection points C_i together with the segments R_iC_i and C_iL_{i+1} produce an n-pointed star inside the polygon. Computing one C_i analytically and obtaining the others by rotation is sufficient. When n is large and θ is small the star fills the polygon sparsely; the standard remedy is to propagate the rays one extra step — intersect R_i with L_{i+2} — which adds an extra layer of geometry. A simple heuristic: do this whenever $n \geq 6$.

Two-point patterns

Two-point patterns place pairs of rays at points equidistant from each edge midpoint, separated along the edge by a distance $d > 0$. Construct L_i and R_i as before, then displace each by $d/2$ along the edge in opposite directions. The remainder of the matching process is unchanged, except we never use the new intersection between L_i and R_i. Setting $\theta = 45°$ produces squares centred on edge midpoints; other angles produce rhombs.



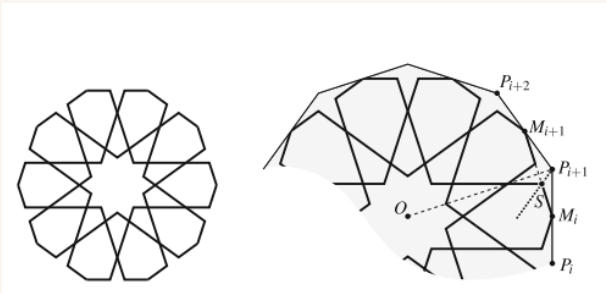
Rosettes

The eye chunks the elements of an ornamental design into larger super-elements. One such device, the *rosette*, consists of an n-pointed star surrounded by two layers of geometry: a ring of n kite quadrilaterals and a ring of shield-shaped hexagons. Rosettes pervade Islamic design.

A simple way to construct an n-pointed rosette inside a regular n-gon hinges on locating the *shoulder*, S. Two regularity conditions fix S:

- The outer edges of the shield hexagons should align to form the outline of a regular n-gon inscribed in the tile. This forces S to lie on the line M_iM_{i+1} joining adjacent edge midpoints.
- The four outer shield edges should have the same length. This forces S to lie on the angle bisector of $\angle OP_{i+1}P_i$.

The intersection of the line and the ray fixes S. Requiring the sides of the shield to be parallel to its axis of symmetry then determines the rest of the rosette.

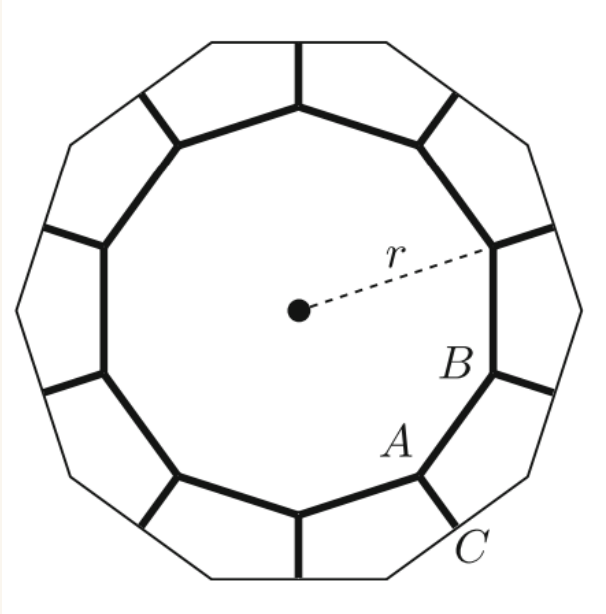


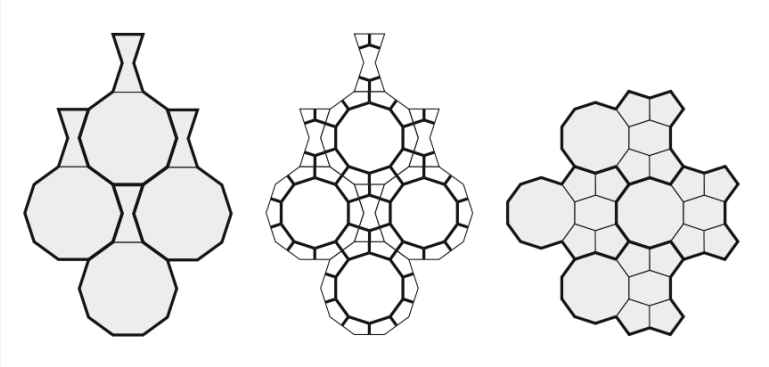
The rosette dual

Rather than overlay a rosette construction on a finished motif tiling, we can build the inevitability of rosettes into the underlying template tiling. The *rosette dual* replaces each regular n-gon ($n \leq 6$) with a concentric, smaller n-gon, rotated so that its corners point at the edge midpoints of the original tile, and with n radial line segments joining

corners to edge midpoints. The radius is chosen so that the internal side length is exactly twice the radial segment length; this guarantees that adjacent radial edges in neighbouring duals fuse to form tile edges of the same length as the internal edges.

The result is a new tiling that, when motifs are drawn at a single global contact angle, produces consistent rosettes.





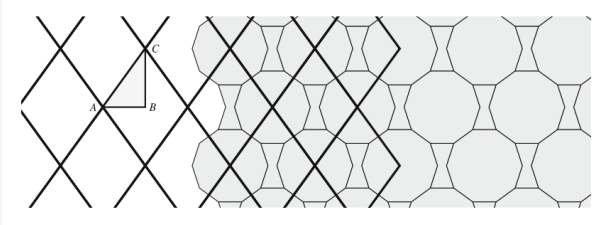
Decoration and filling

Treating each motif as a planar map of vertices, edges, and faces, the simplest decoration is to assign a colour to each face. Many traditional schemes are two-colourings: half the faces share one background colour, the other half share a foreground palette, and no two like-coloured faces are adjacent. Every vertex in our tile planar maps is the meeting point of either two or four edges, which is sufficient for two-colourability. The colouring proceeds by depth-first search from any boundary face.

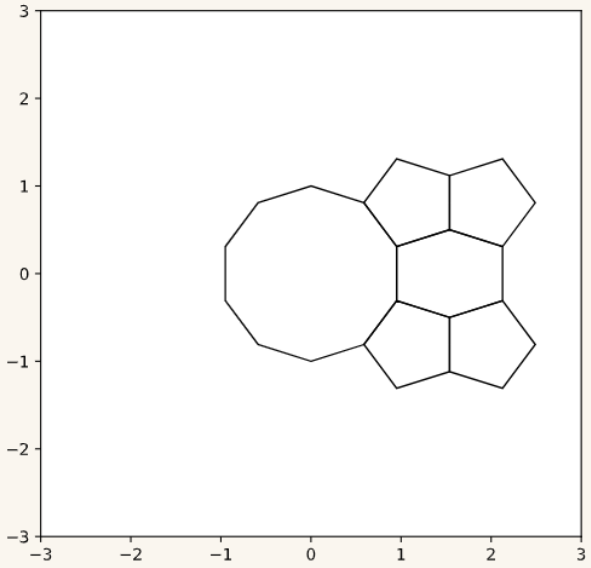
Implementation in Python

Rhomb tiling and the rosette-dual translational unit

Begin with a tiling of the plane by rhombs with interior angles 72° and 108°. A quarter of such a rhomb — a 36–54–90 triangle — functions as a flag. Both 36° and 54° are multiples of 18°, the angle between adjacent lines of reflection through the centre of a decagon. We can therefore place regular $10m$-gons at the 36° and 54° corners.

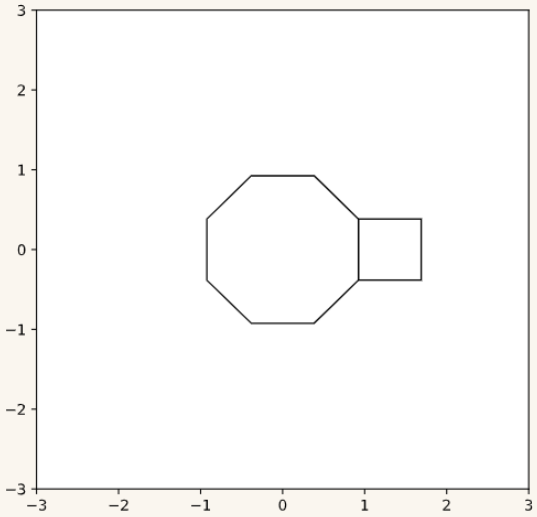


The translational unit consists of a regular decagon and a bow-tie hexagon. After applying the rosette dual we obtain a unit with one decagon, four pentagons, and a barrel-shaped hexagon.



Octagon-and-square translational unit

The second unit pairs a regular octagon with a square attached on one side, sharing edge length.



Defining the translational units

Each point is a NumPy array of shape 1×2. Each n-gon is an array of shape $n \times 2$. Affine transformations are 3×3 matrices acting on homogeneous coordinates.

```
from math import sin, cos, pi, sqrt
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as patches
from numpy.linalg import inv
from functools import reduce

identity = np.identity(3)

def rotate(angle):
    cosa, sina = cos(angle*2*pi/360), sin(angle*2*pi/360)
    return np.array([[cosa, -sina, 0],
                     [sina, cosa, 0],
                     [0,    0,   1]])

def shift(tx, ty):
    return np.array([[1, 0, tx],
                     [0, 1, ty],
                     [0, 0, 1]])

def scale(factor):
    return np.array([[factor, 0, 0],
                     [0,     factor, 0],
                     [0,     0,     1]])

def regularPolygon(n):
    return np.array([np.array([cos(2*pi*i/n), sin(2*pi*i/n)])
                     for i in range(n)])

def transform(polygon, tm):
    polygon_homo_T = np.hstack([polygon, np.ones((len(polygon), 1))]).T
    return (np.dot(tm, polygon_homo_T).T)[:, :2]

sin18, cos18 = sin(pi/10), cos(pi/10)
sin36, cos36 = sin(pi/5), cos(pi/5)
```

```
sin30, cos30 = sin(pi/6), cos(pi/6)
sin54, cos54 = sin(3*pi/10), cos(3*pi/10)

pentagon = regularPolygon(5)
decagon = regularPolygon(10)
barrel_hexagon = np.array([
    np.array([ sin18 + 2*sin18**2, 0]),
    np.array([ sin18,          sin36]),
    np.array([-sin18,          sin36]),
    np.array([-sin18 - 2*sin18**2, 0]),
    np.array([-sin18,         -sin36]),
    np.array([ sin18,         -sin36])])
```

Drawing the motif inside a polygon

The function below takes a polygon and a contact angle and returns a list of polygons — the motif tessellation of the original tile. Each output polygon is tagged as either a *boundary* polygon (`"b"`) or one of two *interior* polygons (`"i1"`, `"i2"`). For polygons with $n > 6$ we propagate the rays one extra step to fill the interior more densely.

```
def motif(polygon, angle):
    polygons = []
    n = len(polygon)
    cosa, sina = cos(angle*2*pi/360), sin(angle*2*pi/360)
    M, L, R = [], [], []
    for i in range(n):
        m_x, m_y = 0.5*(polygon[i] + polygon[(i+1) % n])
        r_x, r_y, _ = reduce(np.dot, [
            shift(m_x, m_y), rotate(angle), shift(-m_x, -m_y),
            np.array([polygon[(i+1) % n][0], polygon[(i+1) % n][1], 1]).T])
        l_x, l_y, _ = reduce(np.dot, [
            shift(m_x, m_y), rotate(-angle), shift(-m_x, -m_y),
            np.array([polygon[(i+1) % n][0], polygon[(i+1) % n][1], 1]).T])
        M.append(np.array([m_x, m_y]))
        L.append(np.array([l_x, l_y]))
        R.append(np.array([r_x, r_y]))

    IP1, IP2 = [], []
    for i in range(n):
        t1, _ = np.dot(inv(np.array([R[i] - M[i],
                                     M[(i+1) % n] - L[(i+1) % n]]).T),
                       (M[(i+1) % n] - M[i]).T)
        IP1.append(M[i] + t1*(R[i] - M[i]))
        if n > 6:
            t2, _ = np.dot(inv(np.array([R[i] - M[i],
                                         M[(i+2) % n] - L[(i+2) % n]]).T),
                           (M[(i+2) % n] - M[i]).T)
            IP2.append(M[i] + t2*(R[i] - M[i]))

    int_polygon = []
    if n <= 6:
        for i in range(n):
            int_polygon += [M[i], IP1[i]]
            polygons.append((np.array([M[i], polygon[(i+1) % n],
                                       M[(i+1) % n], IP1[i]]), "b"))
        polygons.append((np.array(int_polygon), "i1"))
    else:
        for i in range(n):
```

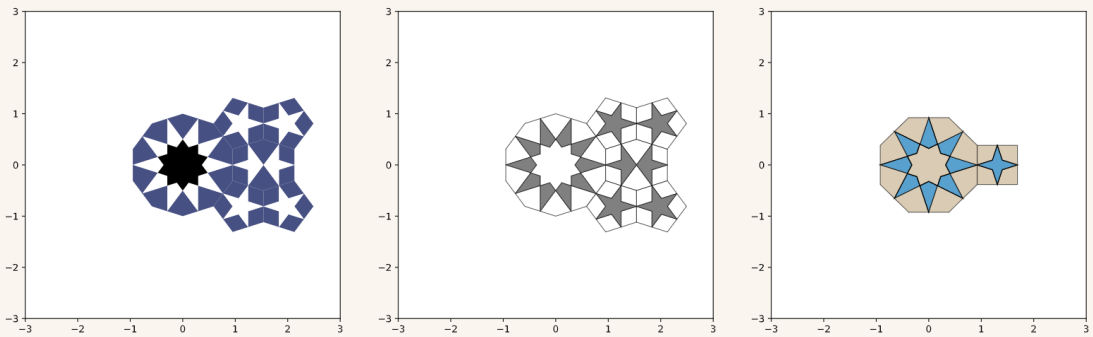
```
            int_polygon += [IP1[i], IP2[i]]
            polygons.append((np.array([M[i], IP1[(i-1) % n],
                                       IP2[(i-1) % n], IP1[i]]), "i1"))
            polygons.append((np.array([M[i], polygon[(i+1) % n],
                                       M[(i+1) % n], IP1[i]]), "b"))
        polygons.append((np.array(int_polygon), "i2"))

    return polygons
```

Stamping the tiling

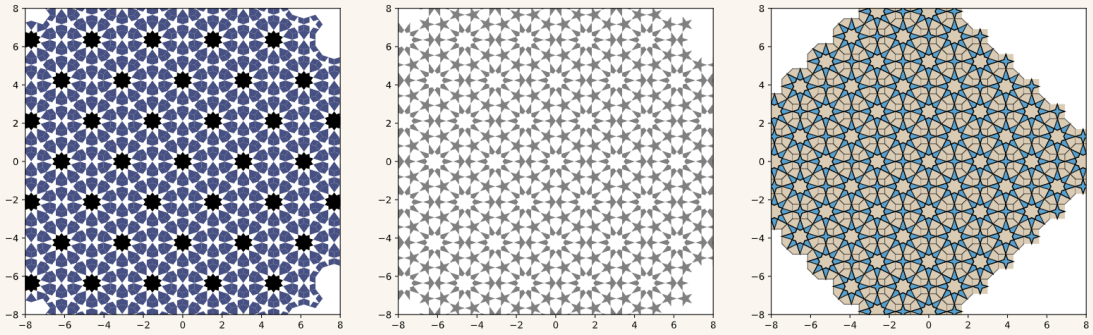
```
def plot_polygons(axes, polygons, styles=None,
                  style_default={"facecolor": "None",
                                 "linewidth": 1,
                                 "edgecolor": "black"}):
    for i, p in enumerate(polygons):
        poly = patches.Polygon(p, **(styles[i] if styles else style_default))
        poly.set_transform(axes.transData)
        axes.add_patch(poly)
    return axes

def plot_translation_units(axes, translation_unit,
                           tiling_pos=[(0, 0)], show_motif=False):
    polygons_transforms, v1, v2 = translation_unit
    polygons, styles = [], []
    for i, j in tiling_pos:
        shiftx, shifty = i*v1 + j*v2
        S = [[1, 0, shiftx], [0, 1, shifty], [0, 0, 1]]
        for polygon, tr, (angle, style) in polygons_transforms:
            if not show_motif:
                polygons.append(transform(transform(polygon, tr), S))
            else:
                for p, ty in motif(polygon, angle):
                    polygons.append(transform(transform(p, tr), S))
                    styles.append(style[ty])
    plot_polygons(axes, polygons, styles)
```

Results

The decagonal translational unit, with two different decoration styles applied to the same motif, and the octagonal one:



Tiled across the plane:



References

- Bonner, Jay. *Islamic Geometric Patterns: Their Historical Development and Traditional Methods of Construction*. Springer, 2017.