

# The Tilings Package: Documentation

Andrew Stacey  
[loopspace@mathforge.org](mailto:loopspace@mathforge.org)

v2.0 from 2023/06/01

## Contents

<b>1 Introduction</b>	<b>1</b>
<b>2 Initialisation</b>	<b>1</b>
<b>3 Defining the Tiles</b>	<b>2</b>
3.1 Predefined Tiles: Penrose . . . . .	3
3.2 Predefined Tiles: Polykite . . . . .	5
<b>4 Using the Tiles</b>	<b>8</b>
4.1 Positioning . . . . .	8
4.2 Styling . . . . .	11
4.3 Placing Tiles Automatically . . . . .	12
<b>5 Deforming Paths</b>	<b>15</b>
<b>6 More Examples</b>	<b>19</b>

## 1 Introduction

The Tilings package is a TikZ library for drawing tiles and tilings, such as Penrose tilings (indeed, it was originally called the **Penrose** package). It currently supports natively the various Penrose tile sets – kite/dart, rhombus, and pentagon – and the aperiodical polykites, but has provision for defining new tiles. There are two main methods for their placement: one that automatically generates a tiling, and one that allows for “by hand” placement. Furthermore, the tiles themselves can be deformed and will still (hopefully!) fit together in the correct fashion.

## 2 Initialisation

This is actually several libraries. There is a core library called `tilings` which defines the main functionality for working with tiles and tilings but which doesn’t define any tiles itself. For that, there are currently two sublibraries:

- `tilings.penrose` for Penrose tiles,

- `tilings.polykite` for the aperiodical polykite tiles.

The sublibraries automatically load the core library.

So to use this package, load the `tikz` package and load one of the `tilings` libraries as a TikZ library. Specifically, your preamble should contain:

```
\usepackage{tikz}
```

and at least one of the following lines:

```
\usetikzlibrary{tilings}
\usetikzlibrary{tilings.penrose}
\usetikzlibrary{tilings.polykite}
```

The original version of this package concerned itself only with Penrose tiles so was named `penrose`. Since then it has been extended and is more general and so has been renamed `tilings`. Some commands have been renamed, but there is a wrapper library which aims to provide backwards compatibility with the original naming scheme. This is now the `penrose` TikZ library, meaning that old documents that use the `penrose` TikZ library should still work (please report any that don't on the [issue tracker](#)). New documents shouldn't be written using it.

### 3 Defining the Tiles

This package includes the capability to define new tiles. The command to do this is:

```
\DefineTile{<name>}{<sides>}{<vertices>}
```

For example,

```
\DefineTile {square}{a A a A}
{
  {0,0}
  {1,0}
  {1,1}
  {0,1}
}
```

The `<sides>` is a list of letters representing sides for matching rules and positioning. The first letter in the list corresponds to the edge between the first two vertices, through to the last side which corresponds to the edge between the last and first vertices.

Each letter used matches against its opposite case (so it is possible to create a side that matches against itself by using a symbol that  $\TeX$  regards as having the same upper as lower case; not all symbols work, I've used numbers and ! successfully). These are also used in path replacing, both a letter and its opposite case are replaced by the corresponding path (but in opposite directions) when the tile is “baked”.

The vertices can be written in cartesian form: `{<x>,<y>}`, or in polar: `{<angle>:<length>}` (with the angle in degrees). The vertex can be preceded by upto two + signs which have the same interpretation as in TikZ: the vertex

description is taken relative to the previously saved point with a single + additionally not updating the last saved point. The initial saved point is the origin, so the following defines a hexagonal tile.

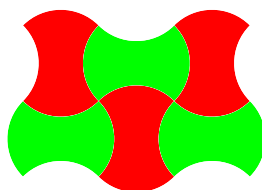
```
\DefineTile {hexagon}{a A a A a A}
{
  +{0:1}
  +{60:1}
  +{120:1}
  +{180:1}
  +{240:1}
  +{300:1}
}
```

The numbers are passed to the `fp` module of  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}3$  so can involve complicated expressions. Note that the trigonometric functions for angles in degrees are `cosd`, `sind`, and `tand`.

The tile must be baked via the command `\BakeTile` before it can be used.


```
\DefineTile {square}{a A a A}
{
  {0,0}
  {1,0}
  {1,1}
  {0,1}
}



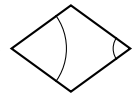
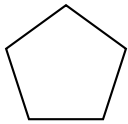
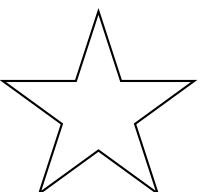
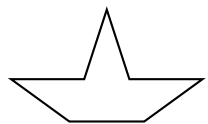
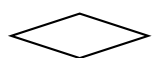


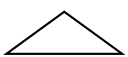

\begin{tikzpicture}
\path[save tiling path=a] (0,0) to[out=45,in=135] (1,0);
\BakeTile{square}
\pic[square,name=A,fill=green];
\pic[square,align with=A along a1 using 2,name=B,fill=red];
\pic[square,align with=A along A1 using 2,fill=red];
\pic[square,align with=A along A2 using 1,fill=red];
\pic[square,align with=B along a1 using 2,fill=green];
\pic[square,align with=B along a2 using 1,fill=green];
\end{tikzpicture}
```



### 3.1 Predefined Tiles: Penrose

The sublibrary `tilings.penrose` defines a slew of tiles based on Penrose tilings.

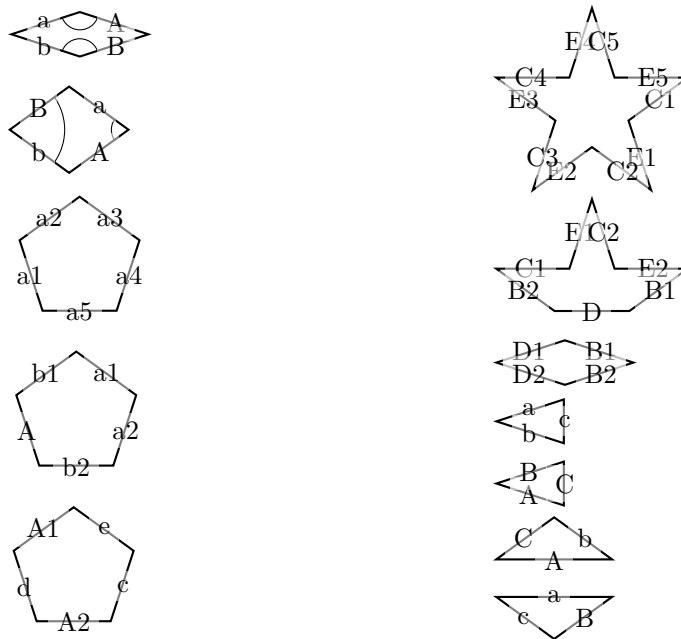
- kite 

- dart 
- thin rhombus 
- thick rhombus 
- pentagon 2, 3, and 5  The number signifies how many pentagons it goes next to.
- pentagram 
- boat 
- diamond 
- golden triangle 
- reverse golden triangle 
- golden gnomon 
- reverse golden gnomon 

The main tiles can have arcs drawn on them to illustrate the matching rules. The triangles and gnomon are not true Penrose tiles but rather can be used to build tilings so they do not have the arcs. The two types of each triangle and gnomon are actually different in that they have different matching rules. This is best illustrated by deforming the paths (see Section 5).

The edge labelling is as follows. For the **pentagon 5**, for example, the edges are all the same and the numbers are used to distinguish between them.





### 3.2 Predefined Tiles: Polykite

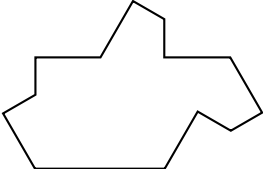
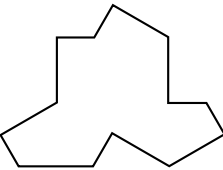
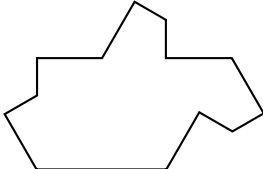
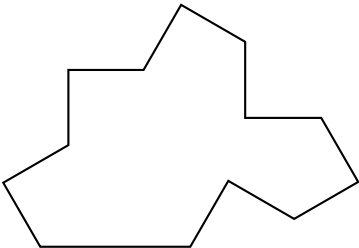
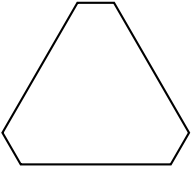

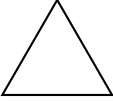
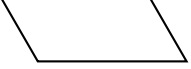
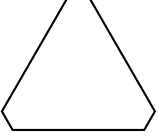

The sublibrary `tilings.polykite` defines a slew of tiles based on the aperiodical polykite tilings as described in [An Aperiodic Monotile](#) and [A Chiral Aperiodic Monotile](#). It also defines a wrapper around the `\DefineTile` command for creating one of the family of polykite tiles described in that article:

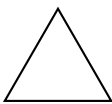
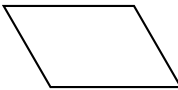




```
\DefinePolykiteTile{<name>}{<a>}{<b>}
\DefinePolykiteTile*{<name>}{<a>}{<b>}
\DefinePolykiteTile{aperiodical hat}{sqrt(3)/2}{1/2}
```

The polykite tiles have two types of edge corresponding to the two lengths. The starred option swaps the labels on these edges so that, for example, the hat and turtle tiles can be used in the same diagram. Predefined polykite tiles are the `aperiodical hat`, `aperiodical turtle`, `spectral hat`, and `spectral turtle`. The aperiodical versions can be swapped in for each other as they have the same edge labelling but the spectral versions have the opposite. (In actual fact, the aperiodical and spectral hats are the same.)

The equilateral polykite tile also exists as a `spectre` with alternating edge matching, as in the second paper [A Chiral Aperiodic Monotile](#).

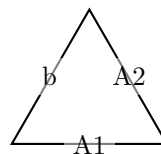
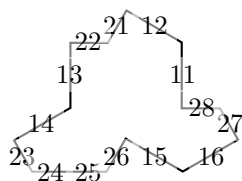
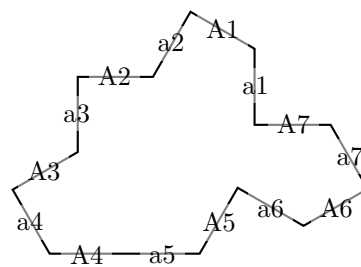
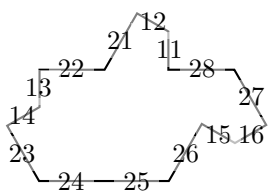
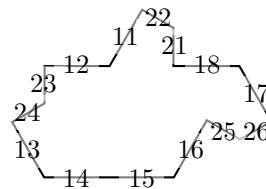
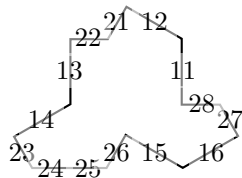


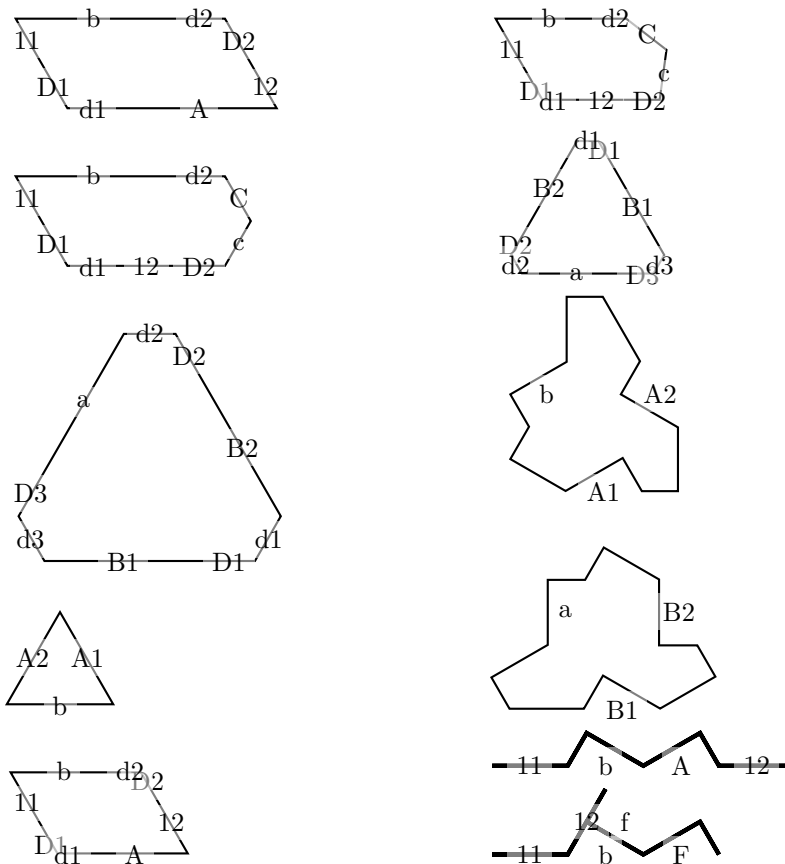
- Aperiodical Turtle 
- Spectral Hat 
- Spectral Turtle 
- Spectre 
- Meta Cluster H 
- Meta Cluster F 
- Meta Cluster T 
- Meta Cluster P 
- Super Cluster H 
- Super Cluster F 

- Super Cluster T 
- Super Cluster P 
- Subcluster H 
- Subcluster F 
- Subcluster T 
- Subcluster P 

The edges of the meta and super cluster tiles are actually multi-edges for the matching rules.

The subcluster tiles are a bit different to the others. The *H* and *T* tiles are actually equilateral triangles but they have edge deformations applied by default so that they have the appearance of the aperiodical hat tile. The *P* and *F* tiles have no area (so are not clipped by default) which means that the edge labels aren't fully clear as to which side they are on. They match the diagrams given in the [paper](#), with the *L* sides being the sides labelled 11 and 12.





## 4 Using the Tiles

### 4.1 Positioning

The main way to use the tiles is via the `pic` syntax that was introduced in TikZ3.0. These are mini-drawings that are node-like in style, but a little more geared towards repeatable *drawings* than boxes containing text.

There are two ways in TikZ to specify the `pic` type: either as the “contents” of the `pic` or as the argument to the `pic type` key. Each of the tiles comes with a shorthand key which specifies the `pic type` and also invokes the keys `every tile pic` and `every <tile type> pic`. That is, the key `dart` calls the `every tile pic` and `every dart pic` keys and specifies the `pic type` as `dart`.

The tiles can be placed using standard TikZ methods. One important thing to note is that by default, `pics` are like `nodes` in that they only respond to ambient translations, and not to rotations and scaling. To make them notice this, use the key `transform shape` or specify the transformation to the `pic` directly. If the shortcut keys are used to specify the tiles, this can be put in the `every tile pic` style.

TikZ `pics` can be named, using the `name=<name>` key or using the `(name)` syntax (see the TikZ documentation for more details). When a tile has been named then it can be used for positioning other tiles. Each edge is assigned a



label and a new tile can be aligned with an old one along a matching edge (a matches with A and so on).

To align a tile with an existing one, use the following key:

```
align with=<tile> along <edge>
```

where `<tile>` is the name given to an existing tile, and `<edge>` is the label on the existing tile. If the tile being placed has edges that are identical (for example, pentagon 5), this syntax extends to

```
align with=<tile> along <edge> using <number>
```

to specify which of the edges on the new tile to use.

Lastly, it is possible to reflect a tile either in use or in definition in which case the edges might be backwards so the key can be modified with the word `back` as in:

```
align with=<tile> back along <edge>
```

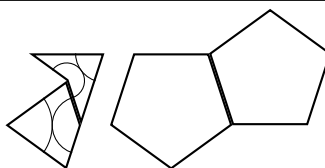
and similarly with `using` ....

A similar key can be used to place the tile so that an edge starts and ends at specified coordinates:

```
align between=<coordinate> and <coordinate> using <edge>
```

Here, the `edge` has to be specified in full since the start and end coordinates don't contain the information about which edge should stretch between them. To obtain the equivalent of `back along` then simply swap the coordinates.

```
\begin{tikzpicture}
\pic[kite,name=tile];
\pic[dart,align with=tile along c];
\pic[pentagon 5,name=ptile,at={(3,0)}];
\pic[pentagon 3,align with=ptile along a1];
\end{tikzpicture}
```

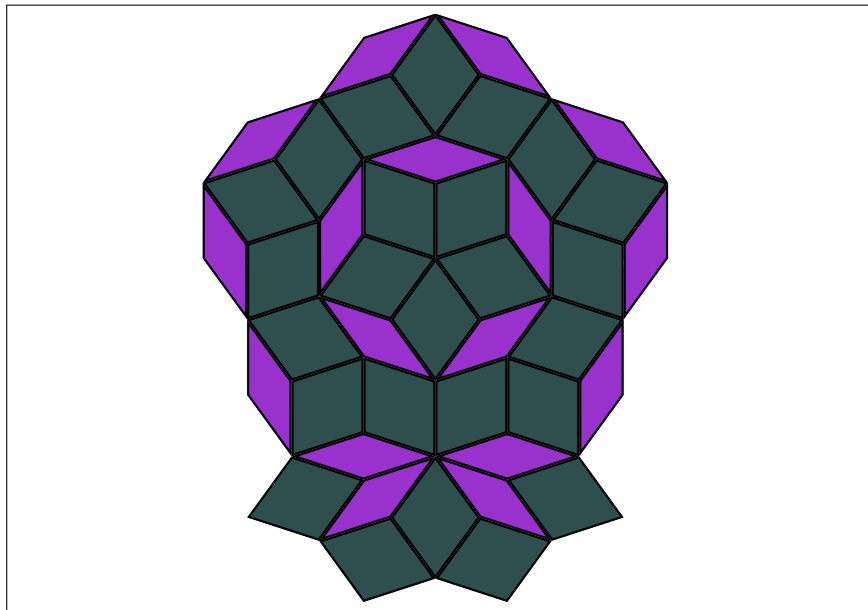


With judicious use of loops, quite complicated pictures can be rendered using simple code. (Note that the `transform shape` is *not* needed to apply the transformations needed to place a tile using this syntax.)

```

\begin{tikzpicture}[
  every rhombus/.style={
    draw=black,
    ultra thick,
  },
  every thin rhombus/.style={
    every rhombus/.try,
    fill=thinRhombus,
  },
  every thick rhombus/.style={
    every rhombus/.try,
    fill=thickRhombus,
  },
  every circle arc/.style={
    draw=circleArc
  },
  every long arc/.style={
    draw=longArc
  }
]
\pic[rotate=18,thick rhombus,name=a0];
\foreach[evaluate=\k as \kmo using int(\k-1)] \k in {1,...,4}
{
  \pic[thick rhombus,name=a\k,align with={a\kmo} along A];
}
\foreach \k in {0,...,4}
{
  \pic[thin rhombus,name=b\k,align with={a\k} along B];
  \pic[thick rhombus,name=c\k,align with={b\k} along A];
  \pic[thick rhombus,name=d\k,align with={b\k} along a];
  \pic[thick rhombus,name=e\k,align with={c\k} along A];
  \foreach \l/\a in {{0/b},{1/B}}
  {
    \pic[thin rhombus,name=f\k\l,align with={e\k} along \a];
  }
}
\pic[thin rhombus,name=g0,align with={f10} along a];
\pic[thin rhombus,name=g1,align with={f21} along A];
\foreach \l/\a in {{0/a},{1/A}}
{
  \pic[thick rhombus,name=h\l,align with={g\l} along \a];
}
\pic[thick rhombus,name=i,align with=g0 along B];
\foreach \l/\a in {{0/a},{1/A}}
{
  \pic[thick rhombus,name=j\l,align with=i along \a];
}
\end{tikzpicture}

```



## 4.2 Styling

The tiles can be styled, either directly or using various keys. When a tile is placed then there are various actions that happen with associated keys. The actions and keys are:

1. A scope is started and the following keys are tried.
  - (a) `every tile scope`
  - (b) `every <name> scope` where `name` is the tile name (not the pic name)
  - (c) `this tile scope`
2. The tile path is used with the intention of it being a clipping path. The keys for this action are:
  - (a) `every tile clip` by default this is set to `clip`
  - (b) `every <name> clip`
  - (c) `this tileclip`
3. A set of keys are used to allow arbitrary code to be executed before the tile path is rendered. These are:
  - (a) `every tile before path`
  - (b) `every <name> before path`
  - (c) `this tile before path`
4. The tile path is rendered. The keys for this action are:
  - (a) `every tile`
  - (b) `every <name>`

- (c) `this tile`
  - (d) `pic actions`
5. A set of keys are used to allow arbitrary code to be executed after the tile path is rendered. These are:
- (a) `every tile after path`
  - (b) `every <name> after path`
  - (c) `this tile after path`

The `pic actions` are any actions given directly to the tile, as in `\pic[draw,thin rhombus];`. The kite, dart, and rhombus tiles also have arcs drawn on them which are inserted via the `every <name> after path` keys, and these are themselves styled using `every circle arc` and `every long arc`. The names come from the way the arcs look on the rhombus shapes.

One other point is important to note about the tiles. They are actually clipped against themselves. This ensures that the tiles do not overstep their bounds and so when placed alongside each other then they do not go over each other. In practical terms, this means that if drawn then the line width is half that which might be expected (but when placed next to another tile, the two halves combine to the expected width).

### 4.3 Placing Tiles Automatically

There is a way to specify certain tilings using *Lindenmayer systems*. In brief, this takes a seed pattern and successively applies *substitution rules* which are meant to simulate replacing a tile by smaller versions of it (and others from the same set). This is implemented for the Penrose tiles and the various clusters related to the aperiodical polykite tile.

The Penrose rules are *positional* in that when replacing a Penrose tile by smaller versions of it the new tiles can be placed solely based on information about the position and orientation of the original tile. This is true also for the *super cluster* tiles of the aperiodical polykite, but not for the meta cluster or subclusters.

The meta and subclusters work on a different system. The structure that is built up is that of a *tree* with a root tile. That root tile is positioned, and then the other tiles are placed next to previously placed tiles. This is a slower process and can mean that the overall shape is different to what might be expected, but from the user's perspective it is the same.

The user command is one of:

```
\TilingDecomposition{<type>}{<level>}{<seed>}
\pic{decomposition={<type>}{<level>}{<seed>}};
```

where `<type>` is one of:

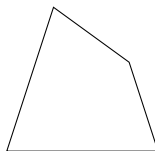
- `kite` for the kite and dart tiling,
- `rhombus` for the rhombus tiling,
- `pentagon` for the pentagon tiling.
- `ktriangle` for the triangular decomposition used to form the kite and dart tiling but with the individual triangles

- `rtriangle` for the triangular decomposition used to form the rhombus tiling but with the individual triangles.
- `supercluster` for the super cluster substitution system that works on a positional basis.
- `cluster` for the cluster substitution system that uses the neighbour information. The choice of super, meta, or subcluster is determined by the value of the key `cluster type`.

The `<seed>` is a “word” that will be used to initiate the Lindenmayer system. The key letters in the alphabet for the kite/darts and rhombuses are `T`, `t`, `G`, and `g`. These actually correspond to the two triangles and two gnomons. For the pentagons, the key letters are `P`, `Q`, `R`, `G`, `B`, `D`. These correspond to the three pentagons, the pentagram, the boat, and the diamond. For the clusters, the key letters are `H`, `T`, `P`, and `F`. Other permitted letters are `[`, `]`, `s`, `f`, `+`, `*`, `-`, `_`, `>`. These refer to various transformations (for details, see the implementation).

The `<level>` controls how far to take the iteration. The code is not particularly optimised for speed, and once `<level>` gets to about 5 or 6 then we are at the “make a cup of tea while compiling” stage, depending on the processor.

```
\begin{tikzpicture}[
  every tile/.style={draw},
  tiling step=2cm,
]
\TilingDecomposition{kite}{0}{T}
\end{tikzpicture}
```



The two types of substitution rules – positional and neighbourly – are rendered slightly differently. The positional tiles are drawn directly but the neighbour ones are drawn via the `\pic` mechanism. This is to exploit the existing code for aligning against existing tiles. This means that the `\pic` styling options are available for the latter type of decomposition but not for the former. For the former, only the following keys are used:

```
every tile
every <tile type>
```

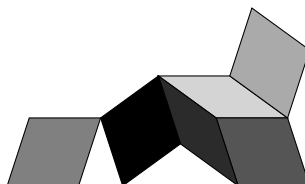
There are, however, some additional styles which allow styling the tiles by their number: a count is kept of the number of tiles and each tile knows its own number. Specifically, two keys are tried:

1. `tile <number>`, and
2. `tile={<number>}{<total>}`

A word of warning is in order on the second of these. The `<total>` is not guaranteed to be correct. It is done by a quick count at the start of the process and counts those letters which *might* result in a rendered tile. Not every letter in the resulting word actually does. Nevertheless, this can be used to style a tile based on what proportion of tiles have been rendered.

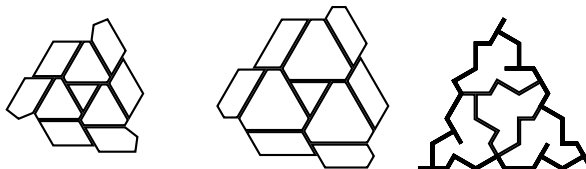
Lastly, `tiling step` is used to control the size of the resulting picture.

```
\begin{tikzpicture}[
  every tile/.style={draw},
  tiling step=4cm,
  tile/.code 2 args={
    \pgfmathsetmacro\tint{100*#1/#2}
    \pgfkeysalso{fill=black!\tint}
  }
]
\TilingDecomposition{rhombus}{3}{T}
\end{tikzpicture}
```



Here's the super, meta, and subcluster combinatorial decompositions. These decompositions also define a style `first tile` which is applied to the initial tile (that isn't placed next to an existing tile).

```
\begin{tikzpicture}[
  every tile/.style={draw},
  first tile/.style={
    transform shape
  },
]
\TilingDecomposition[scale=.25,cluster type=super
  cluster]{cluster}{1}{H}
\TilingDecomposition[xshift=3cm,scale=.5,cluster type=meta
  cluster]{cluster}{1}{H}
\TilingDecomposition[xshift=5cm,cluster
  type=subcluster]{cluster}{1}{H}
\end{tikzpicture}
```

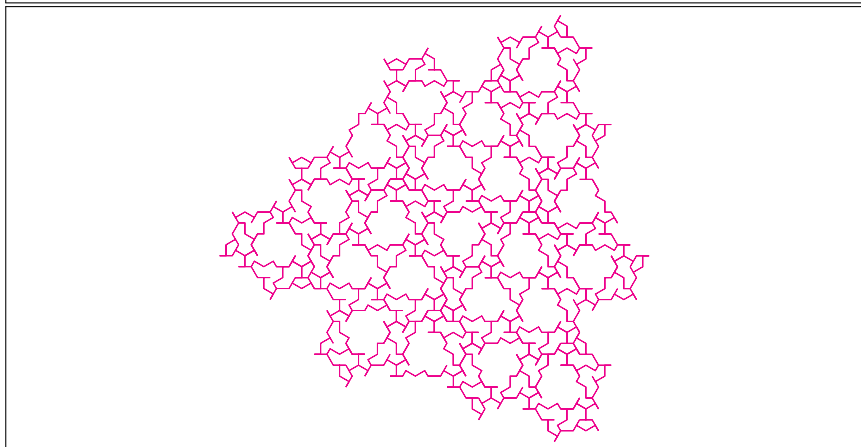


Here is a subcluster decomposition showing just the  $F_0$  and  $P_0$  tiles.

```

\begin{tikzpicture}[
  every tile/.style={},
  every tile pic/.style={},
  every subcluster F/.style={draw=magenta, ultra thin},
  every subcluster P/.style={draw=magenta, ultra thin},
]
\pic[
  first tile/.style={transform shape},
  cluster type=subcluster,
  scale=3
] {decomposition={cluster}{3}{H}};
\end{tikzpicture}

```



## 5 Deforming Paths

This package provides the ability to deform the edges of the tiles. The tiles are built from labelled paths together with their reverses. By changing these paths, one can get a wide variety of different tiles with the same fundamental matching rules. Indeed, by using asymmetric paths, the matching rules can be enforced without the need for additional decoration.

Internally, the `tiling` library uses the `spath3` package for storing and manipulating the paths.

To create a new edge path, use the key `save tiling path=<edge>` where `<edge>` is a symbol used in the edge description of a tile. There are no constraints on the size of the path as all paths are scaled and transformed to fit the tiles. Once the edge paths have been specified, they are welded together into the tiles using the following command:

```
\BakeTile{<name>}
```

Here, `<name>` is one of the names of the tiles. This has global effect, as does the definition of the edge paths. Paths, both sides and tiles, can be cloned via:

```

\tikzset{clone tiling side path={target}{source}}
\tikzset{clone tile path={target}{source}}

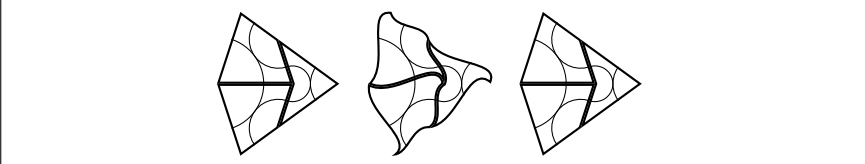
```

and restored with the same command (but names switched).

```

\begin{tikzpicture}
\pic[draw,dart,name=dart];
\pic[draw,kite,align with=dart along c];
\pic[draw,kite,align with=dart along C];
\tikzset{clone tile path={original kite}{kite}}
\tikzset{clone tile path={original dart}{dart}}
\path[save tiling path=a] (0,0) to[out=-30,in=100] (1,0);
\path[save tiling path=c] (0,0) to[out=-40,in=140] (1,0);
\BakeTile{kite}
\BakeTile{dart}
\pic[xshift=2cm,draw,dart,name=dart];
\pic[draw,kite,align with=dart along c];
\pic[draw,kite,align with=dart along C];
\tikzset{clone tile path={kite}{original kite}}
\tikzset{clone tile path={dart}{original dart}}
\pic[xshift=4cm,draw,dart,name=dart];
\pic[draw,kite,align with=dart along c];
\pic[draw,kite,align with=dart along C];
\end{tikzpicture}

```



With deformed tiles there is no guarantee that the inner arcs on the Penrose tiles will match up perfectly.

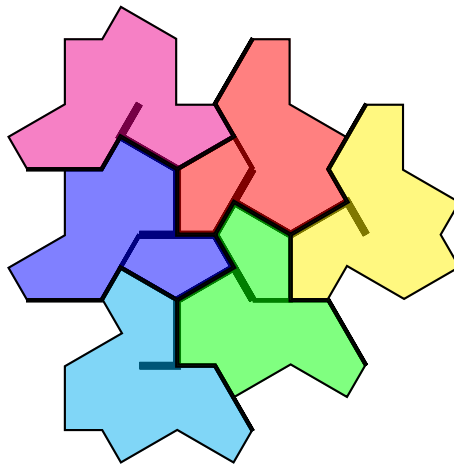
Note that the subcluster tiles from the polykite subpackage are actually already deformed tiles.



```

\begin{tikzpicture}[
  every subcluster F/.style={draw, ultra thick},
  every subcluster P/.style={draw, thick},
  every subcluster H/.style={draw, fill, fill opacity=.5}
]
\pic[subcluster F, name=F1, this tile/.style={line width=1mm}];
\pic[subcluster F, align with=F1 along 12 using 1, name=F2, this
tile/.style={line width=1mm}];
\pic[subcluster F, align with=F1 along 11 using 2, name=F3, this
tile/.style={line width=1mm}];
\pic[subcluster F, align with=F1 along F,name=F1-2];
\pic[subcluster F, align with=F1 along f,name=F1-3];
\pic[subcluster F, align with=F2 along F,name=F2-2];
\pic[subcluster F, align with=F2 along f,name=F2-3];
\pic[subcluster F, align with=F3 along F,name=F3-2];
\pic[subcluster F, align with=F3 along f,name=F3-3];
\pic[subcluster H, align with=F1 along b using 2, fill=red];
\pic[subcluster H, align with=F2 along b using 2, fill=green];
\pic[subcluster H, align with=F3 along b using 2, fill=blue];
\pic[subcluster H, align with=F1-2 along b using 1, fill=yellow];
\pic[subcluster H, align with=F2-2 along b using 1, fill=cyan];
\pic[subcluster H, align with=F3-2 along b using 1, fill=magenta];
\end{tikzpicture}

```

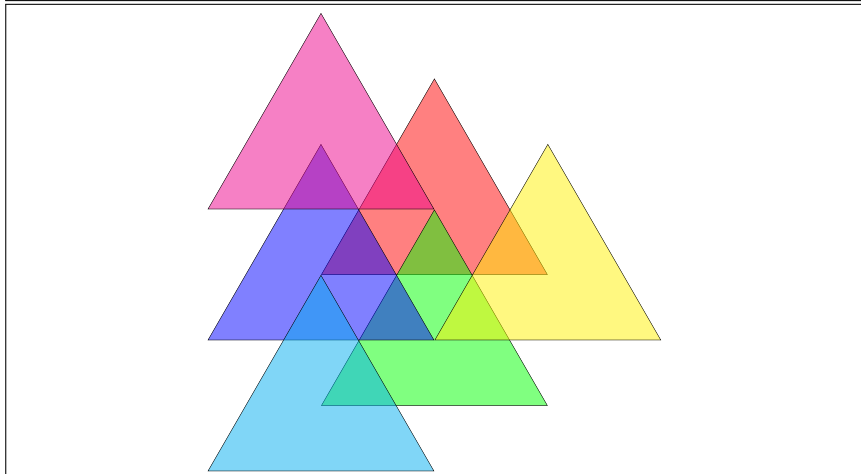


```

\begin{tikzpicture}[
  every tile pic/.style={},
  % every subcluster F/.style={draw, ultra thick},
  % every subcluster P/.style={draw, thick},
  every subcluster H/.style={draw, fill, fill opacity=.5}
]
\path[save tiling path=a] (0,0) -- (1,0);
\path[save tiling path=b] (0,0) -- (1,0);
\path[save tiling path=f] (0,0) -- (1,0);
\BakeTile{subcluster H}
\BakeTile{subcluster T}
\BakeTile{subcluster P}
\BakeTile{subcluster F}

\pic[subcluster F, name=F1, this tile/.style={line width=1mm}];
\pic[subcluster F, align with=F1 along 12 using 1, name=F2, this
  tile/.style={line width=1mm}];
\pic[subcluster F, align with=F1 along 11 using 2, name=F3, this
  tile/.style={line width=1mm}];
\pic[subcluster F, align with=F1 along F,name=F1-2];
\pic[subcluster F, align with=F1 along f,name=F1-3];
\pic[subcluster F, align with=F2 along F,name=F2-2];
\pic[subcluster F, align with=F2 along f,name=F2-3];
\pic[subcluster F, align with=F3 along F,name=F3-2];
\pic[subcluster F, align with=F3 along f,name=F3-3];
\pic[subcluster H, align with=F1 along b using 2, fill=red];
\pic[subcluster H, align with=F2 along b using 2, fill=green];
\pic[subcluster H, align with=F3 along b using 2, fill=blue];
\pic[subcluster H, align with=F1-2 along b using 1, fill=yellow];
\pic[subcluster H, align with=F2-2 along b using 1, fill=cyan];
\pic[subcluster H, align with=F3-2 along b using 1, fill=magenta];
\end{tikzpicture}

```

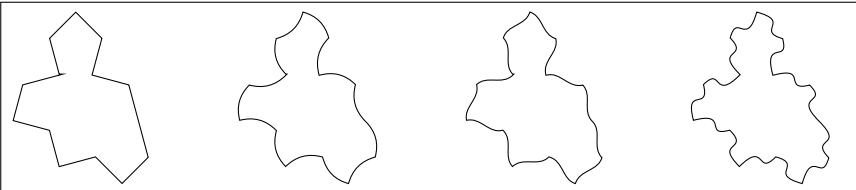


The spectre tile, from [A Chiral Aperiodic Monotile](#), is a natural for deformation as that can be used to force aperiodicity.

```

\begin{tikzpicture}[
  every spectre/.style={
    draw,
    thick,
  },
  every tile pic/.style={scale=.5,rotate=105},
]
\pic[spectre];
\path[save tiling path=a] (0,0) to[out=30,in=150] (1,0);
\BakeTile{spectre}
\pic[spectre,at={(3,0)}];
\path[save tiling path=a] (0,0) to[out=30,in=210] (1,0);
\BakeTile{spectre}
\pic[spectre,at={(6,0)}];
\path[save tiling path=a] (0,0) .. controls +(30:1) and +(210:.75)
  .. (1,0);
\BakeTile{spectre}
\pic[spectre,at={(9,0)}];
\end{tikzpicture}

```



## 6 More Examples

Let's set some aesthetically pleasing shapes. Note that I'm using the TikZ external library in creating this document so to deform the paths in the examples I need to add the deformation code to every picture (otherwise the externalisation skips it). I've saved it into a key `deform tiles` to avoid overwhelming the example code. In a document without externalisation then adding the following code will achieve the same effect.

```

\begin{tikzpicture}
\path[save tiling path=a] (0,0) to[out=-30,in=100] (1,0);
\path[save tiling path=b] (0,0) to[out=0,in=140] (1,0);
\path[save tiling path=c] (0,0) to[out=-40,in=140] (1,0);
\BakeTile{thin rhombus}
\BakeTile{thick rhombus}
\BakeTile{golden triangle}
\BakeTile{reverse golden triangle}
\BakeTile{golden gnomon}
\BakeTile{reverse golden gnomon}
\BakeTile{kite}
\BakeTile{dart}
\end{tikzpicture}

```

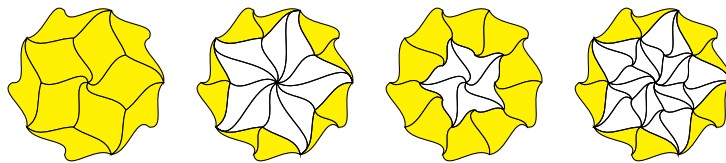
Styling the first tile. Note that as the pattern is formed by repeating two

different initial seeds 5 times, there are 10 “first tiles” in each overall pattern.

```

\begin{tikzpicture}[
  deform tiles/.try,
  every tile/.style={draw},
  tile 1/.style={fill=yellow},
]
\foreach \tp/\pos in
  {rhombus/0cm,rtriangle/2.5cm,kite/5cm,ktriangle/7.5cm}
{
\begin{scope}[xshift=\pos]
\foreach[evaluate=\k as \mk using {\k+Mod(\k,2)},evaluate=\k as
  \ax using {Mod(\k,2) == 0 ? "T" : "t"}] \k in {0,...,9} {
\begin{scope}[rotate=\mk*36]
\TilingDecomposition{\tp}{1}{\ax}
\end{scope}
}
\end{scope}
}
\end{tikzpicture}

```



A more detailed decomposition, with more and more tinting applied to each tile. Roughly half of the counted tiles are rendered, and the ordering in which they are rendered is not at first an obvious one (though it is in general from “outside in”).

Note that the key `tint fill colour` is not a TikZ native. It is defined as:

```

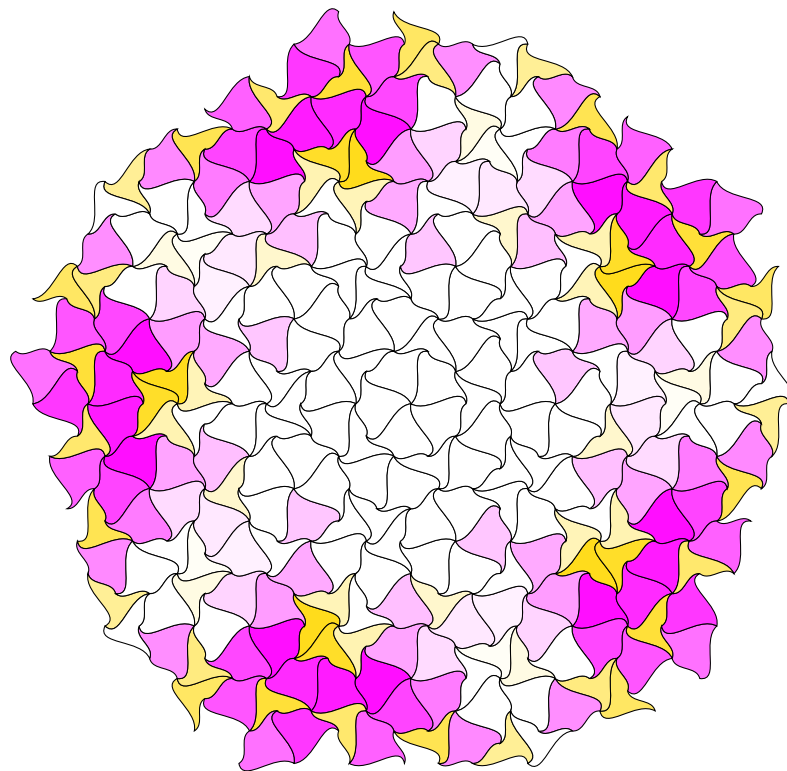
\makeatletter
\begin{tikzset}
  tint fill colour/.code={%
    \edef\@temp{%
      \def\noexpand\tikz@fillcolor{\tikz@fillcolor!#1}%
      \noexpand\tikz@adoption{%
        \noexpand\pgfsetfillcolor{\tikz@fillcolor!#1}%
      }%
    }%
    \@temp
  }
\end{tikzset}
\makeatother

```

```

\begin{tikzpicture}[
  deform tiles/.try,
  every tile/.style={draw},
  every kite/.style={fill=reverseGoldenTriangle},
  every dart/.style={fill=goldenTriangle},
  tile/.code 2 args={
    \pgfmathsetmacro\tint{100*(1 - 1.5*#1/#2)}
    \pgfkeysalso{tint fill colour=\tint}
  }
]
\foreach[evaluate=\k as \mk using {\k+Mod(\k,2)},evaluate=\k as
  \ax using {Mod(\k,2) == 0 ? "T" : "t"}] \k in {0,...,9} {
  \begin{scope}[rotate=\mk*36]
  \TilingDecomposition[tiling step=5cm]{kite}{4}{\ax}
  \end{scope}
}
\end{tikzpicture}

```

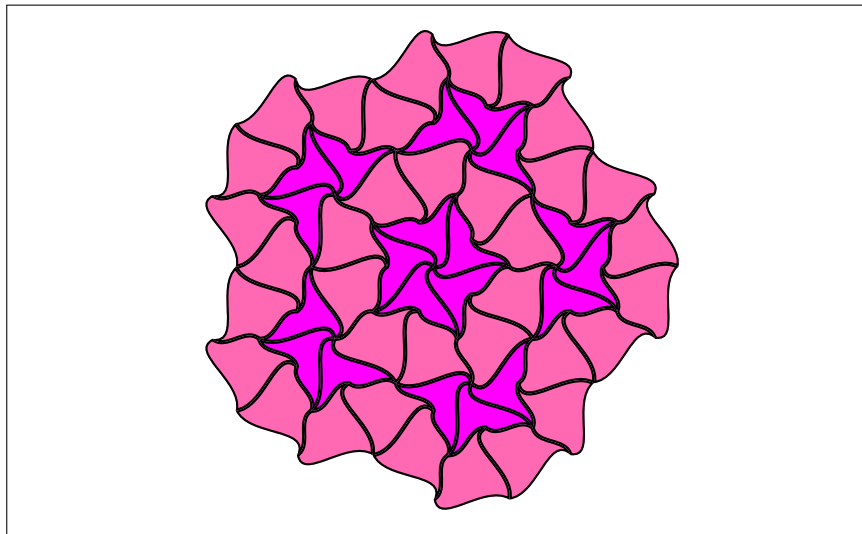


An example with “manual placement”.

```

\begin{tikzpicture}[
  deform tiles/.try,
  every tile/.style={
    draw=black,
    ultra thick,
  },
  every kite/.style={
    fill=kite,
  },
  every dart/.style={
    fill=dart,
  },
  every circle arc/.style={
    draw=circleArc
  },
  every long arc/.style={
    draw=longArc
  }
]
\pic[dart,name=a0];
\foreach[evaluate=\k as \kmo using int(\k-1)] \k in {1,...,4} {
  \pic[dart,name=a\k,align with={a\kmo} along a];
}
\foreach \k in {0,...,4} {
  \foreach \l/\e/\ee in {0/c/a,1/C/A} {
    \pic[kite,name=b\l\k,align with={a\k} along \e];
    \pic[dart,name=c\l\k,align with={b\l\k} along \ee];
    \pic[kite,name=d\l\k,align with={c\l\k} along \e];
  }
  \pic[kite,name=e\k,align with={c0\k} along C];
  \pic[dart,name=f\k,align with={c0\k} along a];
  \foreach \e in {c,C} {
    \pic[kite,name=g\k,align with={f\k} along \e];
  }
}
\end{tikzpicture}

```

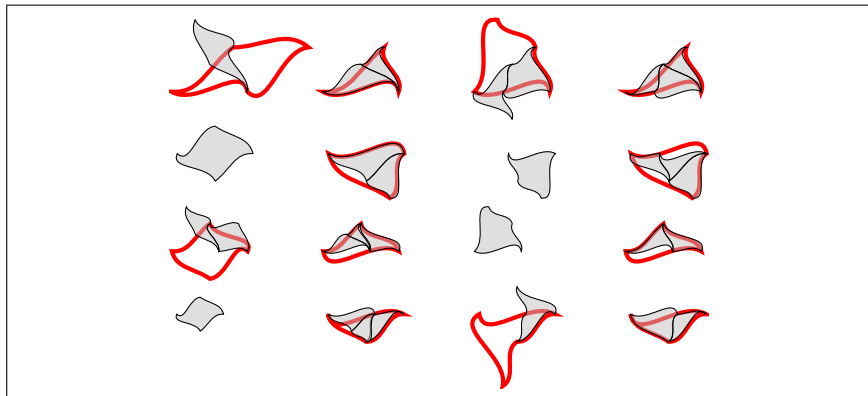


The decomposition rules for the Lindenmayer system can be illustrated by drawing each tile together with the result of one decomposition superimposed on top.

```

\foreach \ax in {T,t,G,g} {
\begin{tikzpicture}[
  deform tiles/.try,
]
\foreach \tp/\pos in
  {rhombus/0cm,rtriangle/2cm,kite/4cm,ktriangle/6cm}
{
\begin{scope}[xshift=\pos]
  \TilingDecomposition[every path/.style={draw=red,ultra
    thick}]{\tp}{0}{\ax}
  \TilingDecomposition[every path/.style={fill=gray!50,fill
    opacity=.5,draw=black}]{\tp}{1}{\ax}
\end{scope}
}
\end{tikzpicture}
}

```



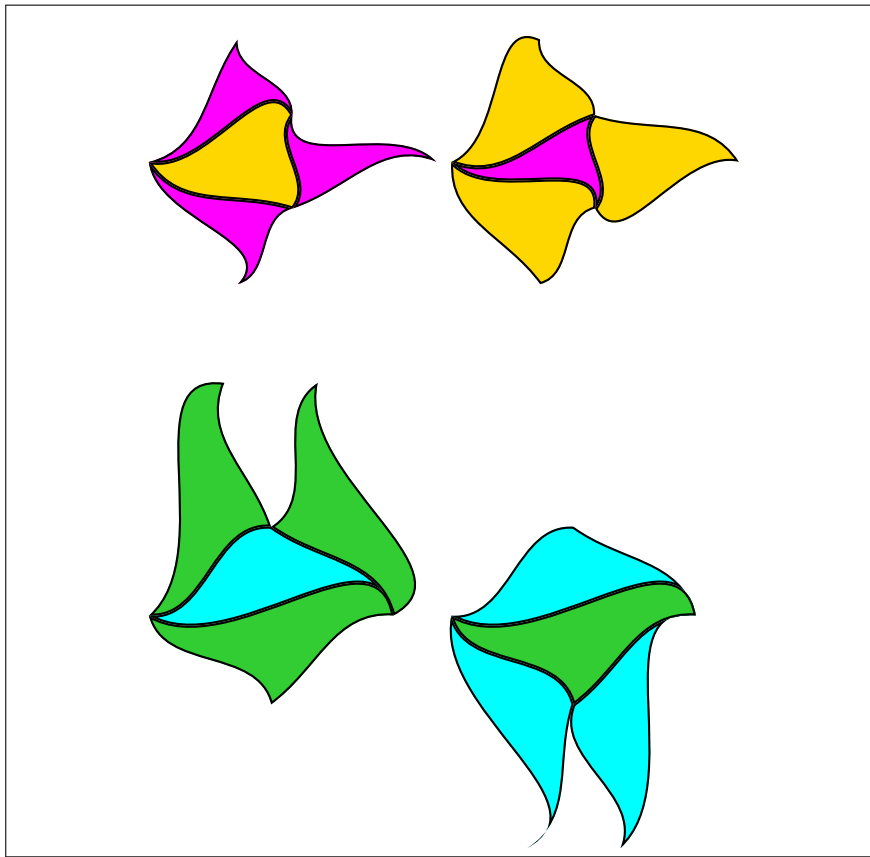
The tiles can make interesting forms by themselves.



```

\begin{tikzpicture}[
  deform tiles/.try,
  scale=2,
  every tile pic/.style={
    transform shape,
  },
  every golden triangle/.style={
    draw=black,
    ultra thick,
    fill=goldenTriangle,
  },
  every reverse golden triangle/.style={
    draw=black,
    ultra thick,
    fill=reverseGoldenTriangle,
  },
  every golden gnomon/.style={
    draw=black,
    ultra thick,
    fill=goldenGnomon,
  },
  every reverse golden gnomon/.style={
    draw=black,
    ultra thick,
    fill=reverseGoldenGnomon,
  },
]
\pic[golden triangle,name=a];
\pic[reverse golden triangle,align with=a along a];
\pic[reverse golden triangle,align with=a along b];
\pic[reverse golden triangle,align with=a along c];
\begin{scope}[xshift=2cm]
\pic[reverse golden triangle,name=a];
\pic[golden triangle,align with=a along A];
\pic[golden triangle,align with=a along B];
\pic[golden triangle,align with=a along C];
\end{scope}
\begin{scope}[yshift=-3cm]
\pic[golden gnomon,name=a];
\pic[reverse golden gnomon,align with=a along C];
\pic[reverse golden gnomon,align with=a along b];
\pic[reverse golden gnomon,align with=a along A];
\begin{scope}[xshift=2cm]
\pic[reverse golden gnomon,name=a];
\pic[golden gnomon,align with=a along c];
\pic[golden gnomon,align with=a along B];
\pic[golden gnomon,align with=a along a];
\end{scope}
\end{scope}
\end{tikzpicture}

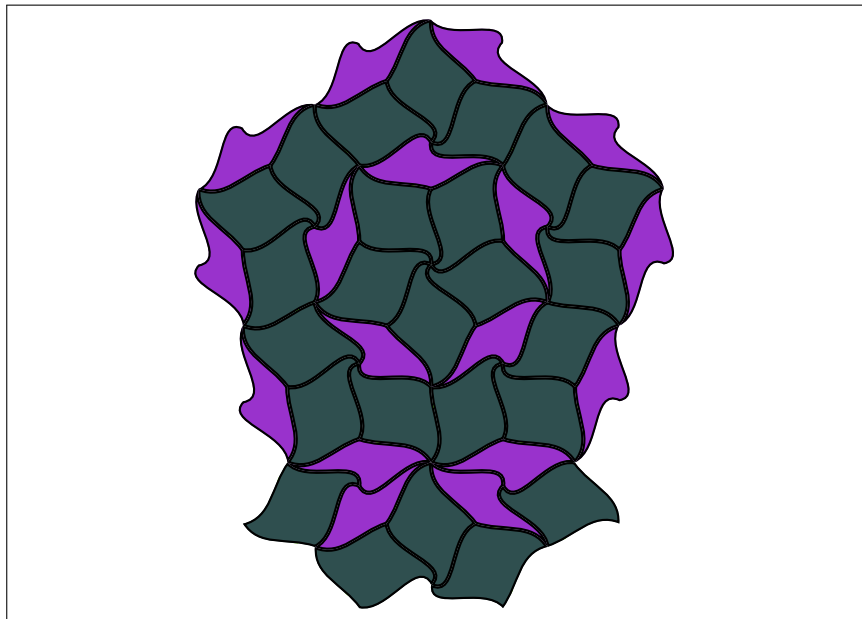
```



```

\begin{tikzpicture}[
  deform tiles/.try,
  every rhombus/.style={
    draw=black,
    ultra thick,
  },
  every thin rhombus/.style={
    every rhombus/.try,
    fill=thinRhombus,
  },
  every thick rhombus/.style={
    every rhombus/.try,
    fill=thickRhombus,
  },
  every circle arc/.style={
    draw=circleArc
  },
  every long arc/.style={
    draw=longArc
  }
]
\pic[rotate=18,thick rhombus,name=a0];
\foreach[evaluate=\k as \kmo using int(\k-1)] \k in {1,...,4}
{
  \pic[thick rhombus,name=a\k,align with={a\kmo} along A];
}
\foreach \k in {0,...,4}
{
  \pic[thin rhombus,name=b\k,align with={a\k} along B];
  \pic[thick rhombus,name=c\k,align with={b\k} along A];
  \pic[thick rhombus,name=d\k,align with={b\k} along a];
  \pic[thick rhombus,name=e\k,align with={c\k} along A];
  \foreach \l/\a in {{0/b},{1/B}}
  {
    \pic[thin rhombus,name=f\k\l,align with={e\k} along \a];
  }
  \pic[thin rhombus,name=g0,align with={f10} along a];
  \pic[thin rhombus,name=g1,align with={f21} along A];
  \foreach \l/\a in {{0/a},{1/A}}
  {
    \pic[thick rhombus,name=h\l,align with={g\l} along \a];
  }
  \pic[thick rhombus,name=i,align with=g0 along B];
  \foreach \l/\a in {{0/a},{1/A}}
  {
    \pic[thick rhombus,name=j\l,align with=i along \a];
  }
}
\end{tikzpicture}

```



Lastly, here's an example that generates full page patterns.

```

\foreach \tp/\lvl in {rhombus/5,rhombus/6,kite/5,kite/6}
{
\tikzset{external/export next=false}
\begin{tikzpicture}[
every tile/.style={draw},
remember picture,
overlay,
scale=20
]
\coordinate (a) at (current page.center);
\begin{scope}[shift={(a)}]
\foreach[evaluate=\k as \mk using {\k+Mod(\k,2)},evaluate=\k as
\ax using {Mod(\k,2) == 0 ? "T" : "t"}] \k in {0,...,9} {
\begin{scope}[rotate=\mk*36]
\TilingDecomposition{\tp}{\lvl}{\ax}
\end{scope}
}
\end{scope}
\end{tikzpicture}
\newpage
}

```