In [1]:

```
import sys
sys.path.append('../')


import importlib
from  new_wallpaper_groups import groupings
from new_wallpaper_groups import slicing
importlib.reload(groupings)
importlib.reload(slicing)


from pdesign import canvas, shapes, lines, transforms
import numpy as np
from shapely.geometry import MultiLineString, LineString, Point, Polygon, Mul
from shapely.ops import unary_union, linemerge
import matplotlib.pyplot as plt

from shapely.geometry import box as Box
from shapely import affinity, ops

import noise


from scipy.spatial import Voronoi, voronoi_plot_2d


import svgwrite
```

In [2]:

```
#https://sgillies.net/2010/04/06/painting-punctured-polygons-with-matplotlib.

from matplotlib import pyplot
from matplotlib.path import Path
from matplotlib.patches import PathPatch
from numpy import asarray, concatenate, ones
from shapely.geometry import *

def ring_coding(ob):
    # The codes will be all "LINETO" commands, except for "MOVETO"s at the
    # beginning of each subpath
    n = len(ob.coords)
    codes = ones(n, dtype=Path.code_type) * Path.LINETO
    codes[0] = Path.MOVETO
    return codes

def pathify(polygon):
    # Convert coordinates to path vertices. Objects produced by Shapely's
    # analytic methods have the proper coordinate order, no need to sort.
    vertices = concatenate(
                    [asarray(polygon.exterior)]
                    + [asarray(r) for r in polygon.interiors])
    codes = concatenate(
                [ring_coding(polygon.exterior)]
                + [ring_coding(r) for r in polygon.interiors])
    return Path(vertices, codes)
```

In [18]:

```
picture = canvas.Canvas(paper_size=(11, 14), margin_percent=0.0, origin='corn
picture_bbox = Box(picture.bbox[0,0], picture.bbox[0,1], picture.bbox[1,0], p

dp = {
    "alpha":0.7,
    "linewidth":1.0*0.0393701*72,
    "clear":False,
}
```

```
<Figure size 792x1008 with 0 Axes>
```

In [4]:

```
m = 1

width = (11 - 3*m)/2
height = (17-2*m)

width, height, height/width
```

Out[4]:

```
(4.0, 15, 3.75)
```

In [5]:

```
3.625/4
```

Out[5]:

```
0.90625
```

In [ ]:

In [7]:

```
2*width + 3*m
```

Out[7]:

```
11.0
```

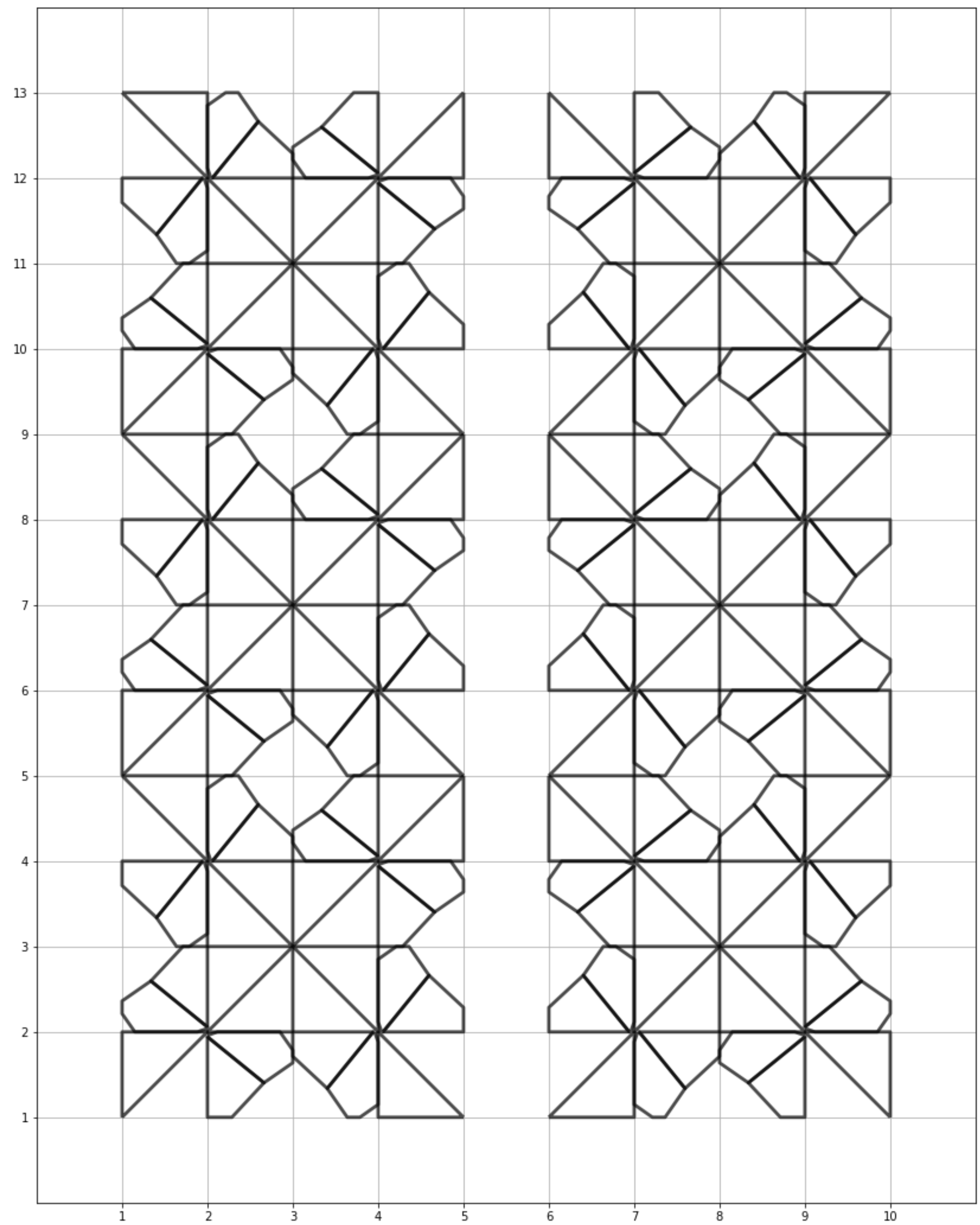
In [246…

```
m = 1.

width = (11 - 3*m)/2
height = (17-2*m)

width, height, height/width


n_doubles = 0


init_square_size = int(width/(2**n_doubles))



possible_trans = groupings.square_groups + groupings.rect_groups
n_inner_mixes = 0
n_out_mixes = 2

big_pattern = []


buffer = 1


for x in range(init_square_size):
    for y in range(init_square_size):

        pattern_bounds = Box(x,y,1+x, 1+y)

        n_pts = 7
        #n_pts = np.random.choice([30, 120])


        pts = np.dstack([np.random.uniform(x-buffer, x+1+buffer, n_pts), np.r
        vor = Voronoi(pts)

        reg = [r for r in vor.regions if -1 not in r and len(r)>0]
        pattern = [Polygon(vor.vertices[r])for r in reg]
        pattern = [p.intersection(pattern_bounds) for p in pattern]
        #pattern = [p for p in pattern if p.area>0.1]




        for _ in range(n_inner_mixes):
            group = possible_trans[np.random.randint(len(possible_trans))]
            pattern_bounds, pattern = groupings.slice_alias[group](pattern_bo


        big_pattern += pattern
        #big_pattern += [pattern_bounds]





pattern_bounds = Box(0,0,init_square_size, init_square_size)
```

```
for _ in range(n_out_mixes):
    group = possible_trans[np.random.randint(len(possible_trans))]
    pattern_bounds, big_pattern = groupings.slice_alias[group](pattern_bounds

for _ in range(n_doubles):
    pattern_bounds, big_pattern = groupings.square_to_square[np.random.randin



rs = width / (init_square_size*2**n_doubles)
center = (0,0)

big_pattern = [p for p in big_pattern if p.area>0.1]

#big_pattern = [affinity.scale(w, rs, rs, origin=center) for w in big_pattern


windows = []
#offsets = [(0,0)]
offsets = [(m,m), (m, m+width), (m, m+2*width), ]

for x,y in offsets:
    windows += [affinity.translate(o, x, y) for o in big_pattern]


mirrored = [affinity.scale(o, -1, 1, origin=(width/2, width/2)) for o in big_

offsets = [(2*m + width,m), (2*m + width, m+width), (2*m + width, m+2*width),
for x,y in offsets:
    windows += [affinity.translate(o, x, y) for o in mirrored]




windows = [w for w in windows if isinstance(w, Polygon)]


picture.make_canvas()
picture.add_grid(11, 14)
picture.plot_shapes(windows, **dp, color='black')
picture.fig
```

```
(11, 14)
```

Out[246…

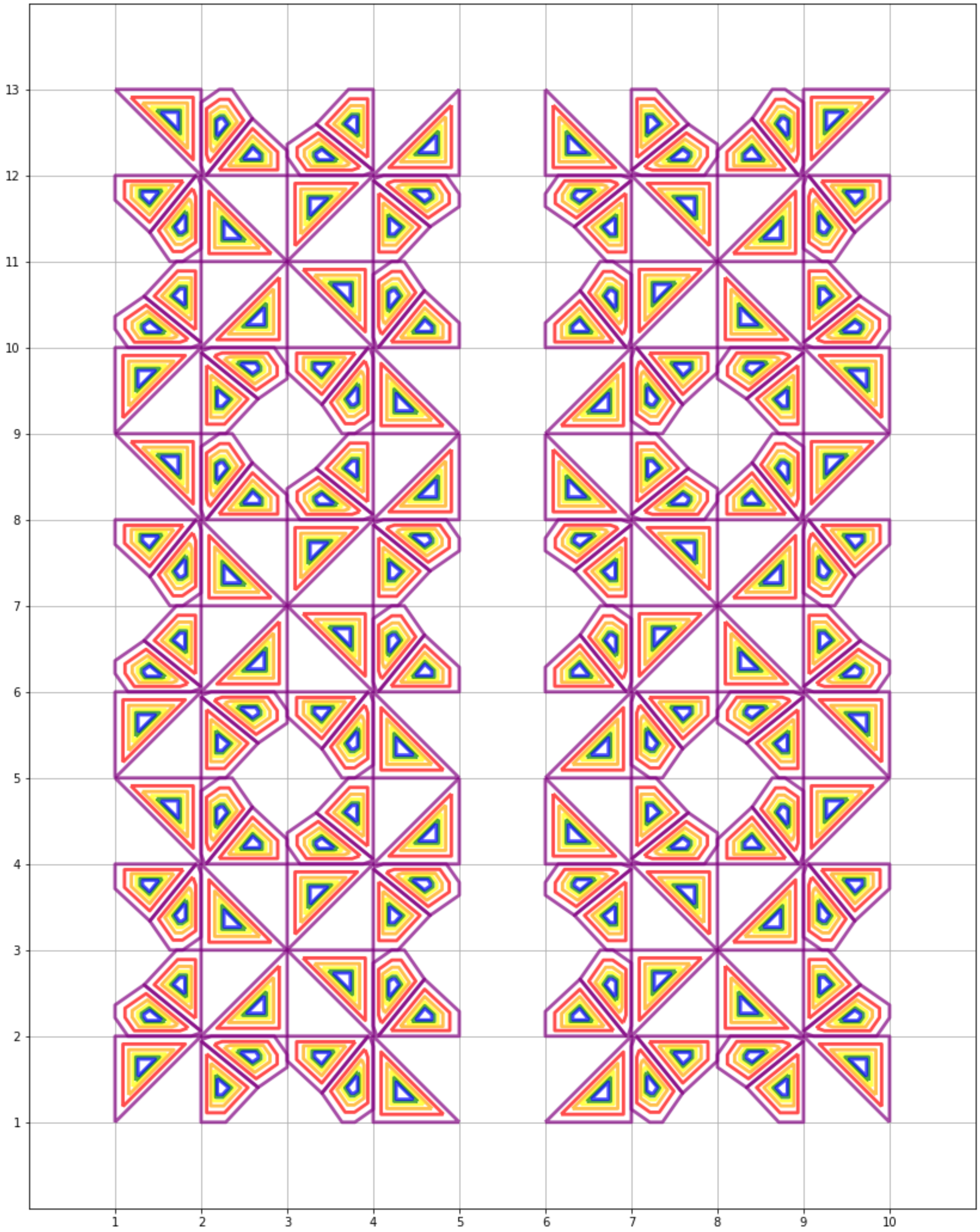


In [247…

```
colors = ['purple', 'red', 'orange', 'yellow', 'green', 'blue']

picture.make_canvas()
picture.add_grid(11, 14)

for s, c in zip(np.geomspace(1, 0.2, len(colors)), colors):

    picture.plot_shapes([affinity.scale(w, s, s, origin='centroid') for w in


#picture.plot_shapes(windows, **dp, color='black')
picture.fig
```

```
(11, 14)
```

Out[247…



In [248…

```
colors = ['purple', 'red', 'orange', 'yellow', 'green', 'blue']




for s, c in zip(np.geomspace(1, 0.2, len(colors)), colors):
    picture.make_canvas()

    picture.plot_shapes([affinity.scale(w, s, s, origin='centroid') for w in


#picture.plot_shapes(windows, **dp, color='black')
    picture.display_overlays(False)
    picture.fig.savefig("rainbow_stainglass/{}.svg".format(c))
    picture.fig
```

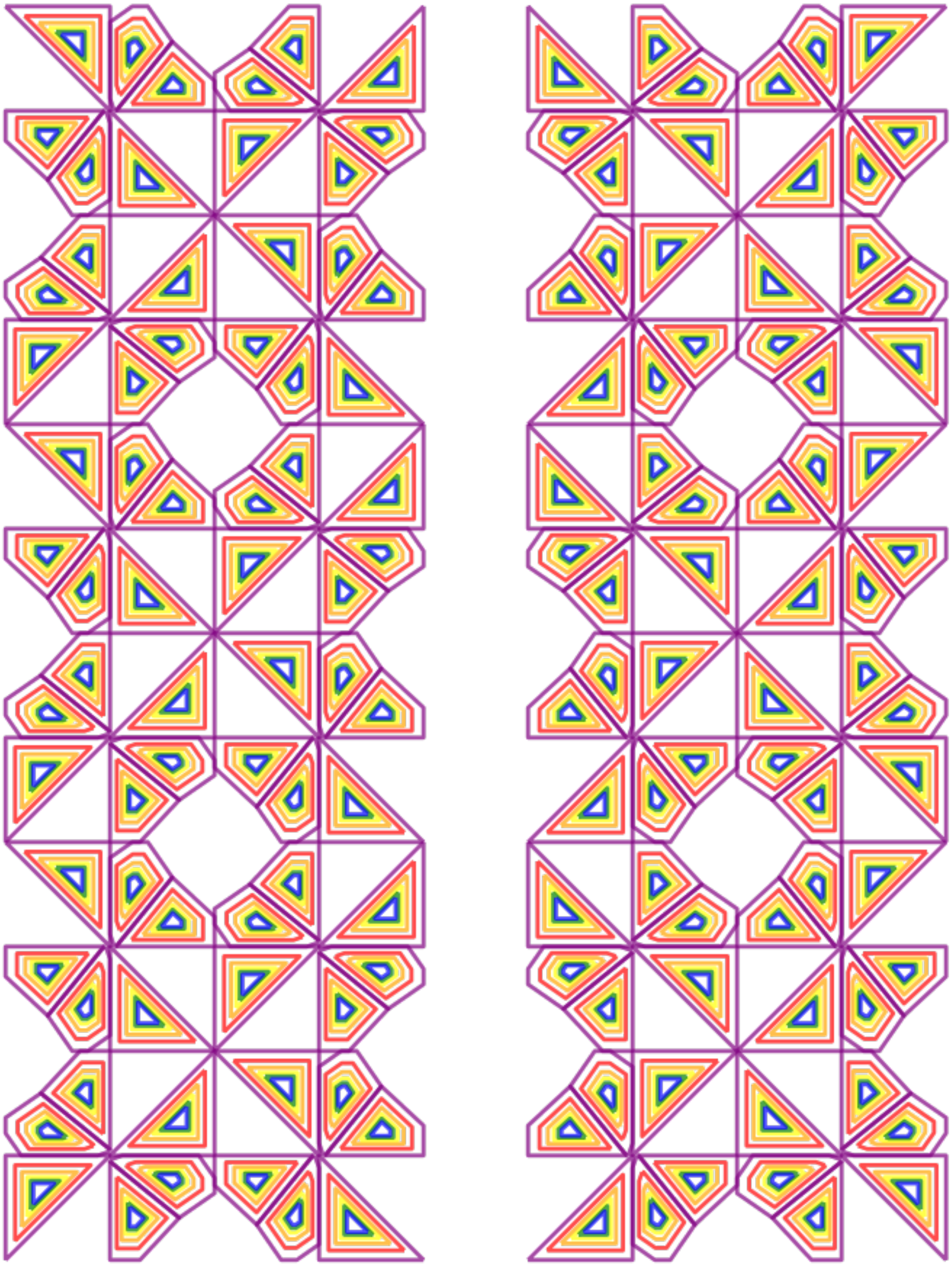
```
(11, 14)
(11, 14)
(11, 14)
(11, 14)
(11, 14)
(11, 14)
```

In [249…

```
colors = ['purple', 'red', 'orange', 'yellow', 'green', 'blue']

picture.make_canvas()
picture.add_grid(11, 14)

for s, c in zip(np.geomspace(1, 0.2, len(colors)), colors):

    picture.plot_shapes([affinity.scale(w, s, s, origin='centroid') for w in


#picture.plot_shapes(windows, **dp, color='black')
picture.display_overlays(False)
picture.fig.savefig("rainbow_stainglass/all.svg")
picture.fig
```

```
(11, 14)
```

Out[249…



In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
In [181…
In [ ]:
In [ ]:
In [ ]:
```