

# Programmable 2D Arrangements for Element Texture Design

HUGO LOI

Inria-LJK (UGA, CNRS)

and

THOMAS HURTUT

École Polytechnique de Montréal

and

ROMAIN VERGNE and JOELLE THOLLOT

Inria-LJK (UGA, CNRS)

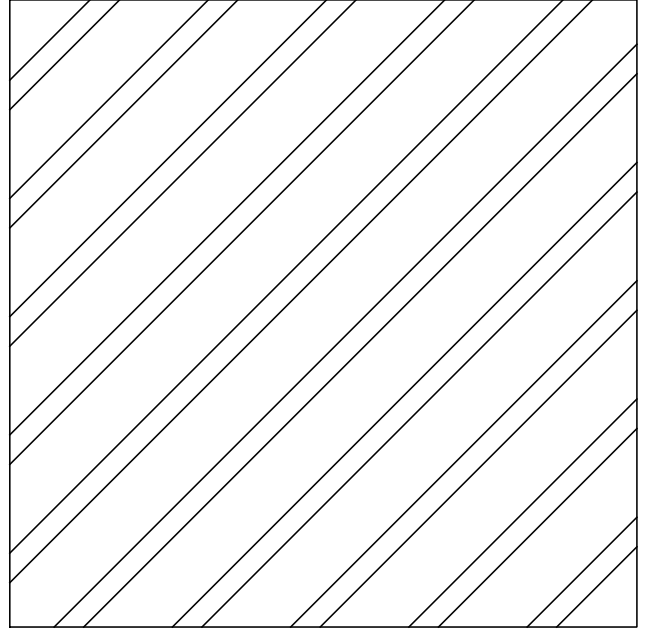
**Stripes**

&lt;1 sec

```

1 def stripes():
2     # Texture size
3     size = 2000
4
5     # Stripes angle=pi/4, widths=200 and 67
6     lines = StripesProperties(pi/4, size/10, size/30)
7
8     # Partition
9     tex = StripesPartition(lines)
10
11    # Export result
12    ExportSVG(tex, size)

```

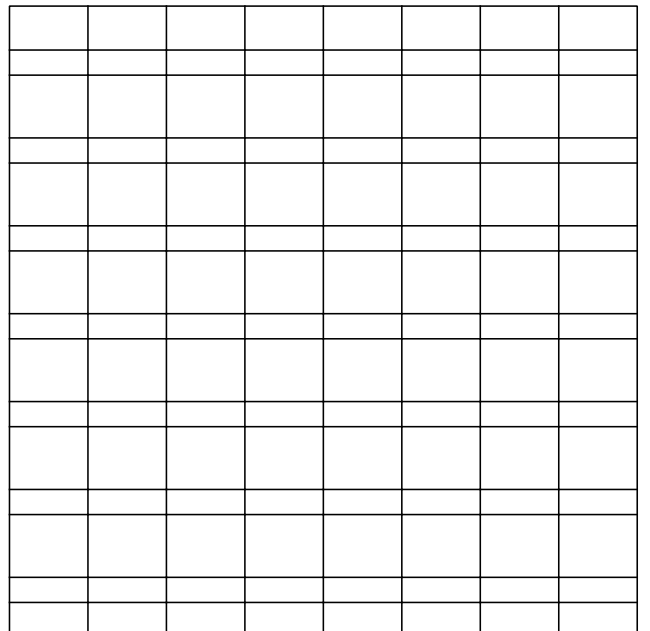
**Grid**

&lt;1 sec

```

1 def grid():
2     # Texture size
3     size = 2000
4
5     # Horizontal stripes angle=0, widths=200 and 80
6     lines1 = StripesProperties(0, size/10, size/25)
7
8     # Vertical stripes angle=pi/2, width=250
9     lines2 = StripesProperties(pi/2, size/8)
10
11    # Partition
12    tex = GridPartition(lines1, lines2, KEEP_OUTSIDE)
13
14    # Export result
15    ExportSVG(tex, size)

```

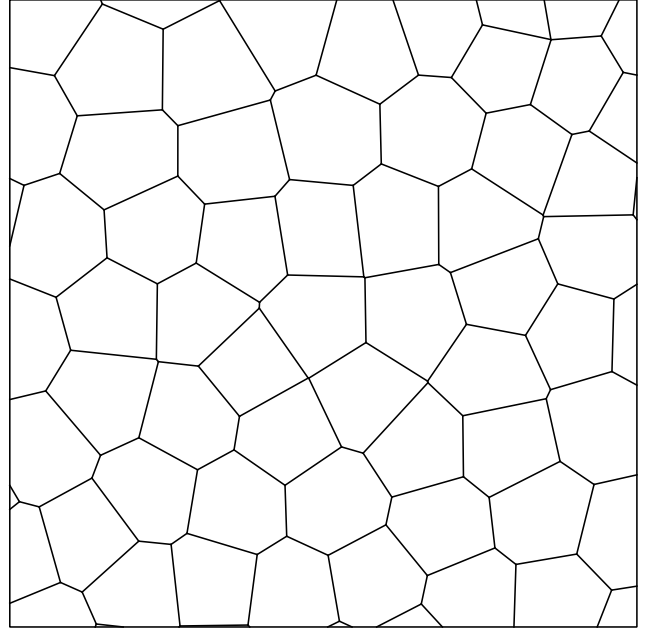


**Uniform**

2 sec

```

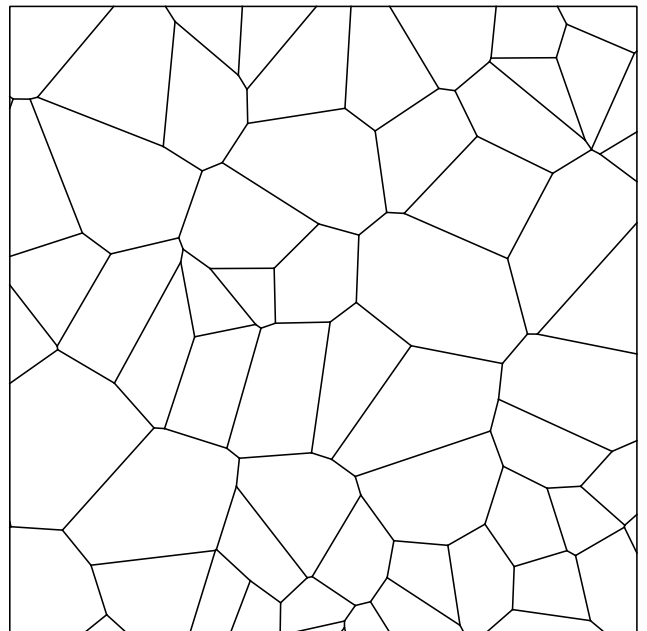
1 def uniform():
2     # Texture size
3     size = 2000
4
5     # Density ~= 50 faces
6     prop = IrregularProperties(50/(size*size))
7
8     # Partition
9     tex = UniformPartition(prop, KEEP_OUTSIDE)
10
11    # Export result
12    ExportSVG(tex, size)
    
```


**Random**

&lt;1 sec

```

1 def random():
2     # Texture size
3     size = 2000
4
5     # Density ~= 50 faces
6     prop = IrregularProperties(50/(size*size))
7
8     # Partition
9     tex = RandomPartition(prop, KEEP_OUTSIDE)
10
11    # Export result
12    ExportSVG(tex, size)
    
```



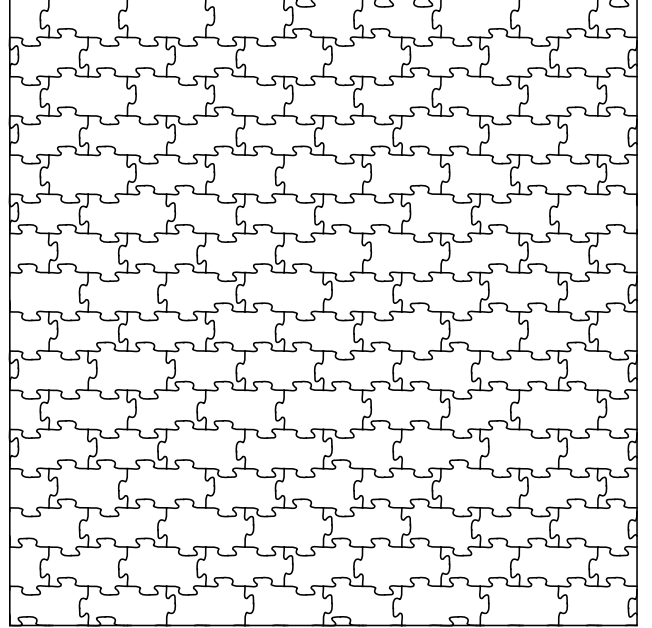
**Puzzle**

18 sec

```

1 def puzzle():
2     size = 2000
3     zig = ImportSVG("data/zig2.svg")
4
5     # Grid partition, including edge labels
6     lines1 = StripesProperties(0.0, size/16)
7     lines2 = StripesProperties(pi/2.0, size/16)
8     SetEdgeLabels(lines1, "h1", "h2")
9     SetEdgeLabels(lines2, "v1", "v2")
10    grid_tex = GridPartition(lines1, lines2, KEEP_OUTSIDE)
11
12    # Mapper: remove edges
13    def grid_to_wall(edge):
14        if ((HasLabel(edge, "v1") and HasLabel(IncidentEdges(
15            TargetVertex(edge)), "h1")) or
16            (HasLabel(edge, "v2") and HasLabel(IncidentEdges(
17                TargetVertex(edge)), "h2"))):
18            return Nothing()
19        return ToCurve(edge)
20
21    # Mapper: replace each edge by a curved line
22    def edge_to_curve(edge):
23        src_c = PointLabeled(zig, "start")
24        dst_c = PointLabeled(zig, "end")
25        src_v = Location(SourceVertex(edge))
26        dst_v = Location(TargetVertex(edge))
27        if Random(edge, 0, 1, 0) < 0.5:
28            return MatchPoints(zig, src_c, dst_c, src_v, dst_v)
29        else:
30            return MatchPoints(zig, src_c, dst_c, dst_v, src_v)
31
32    # Mapping operators
33    wall_tex = MapToEdges(grid_to_wall, grid_tex)
34    puzzle_tex = MapToEdges(edge_to_curve, wall_tex)
35
36    # Final texture
37    ExportSVG(puzzle_tex, size)

```

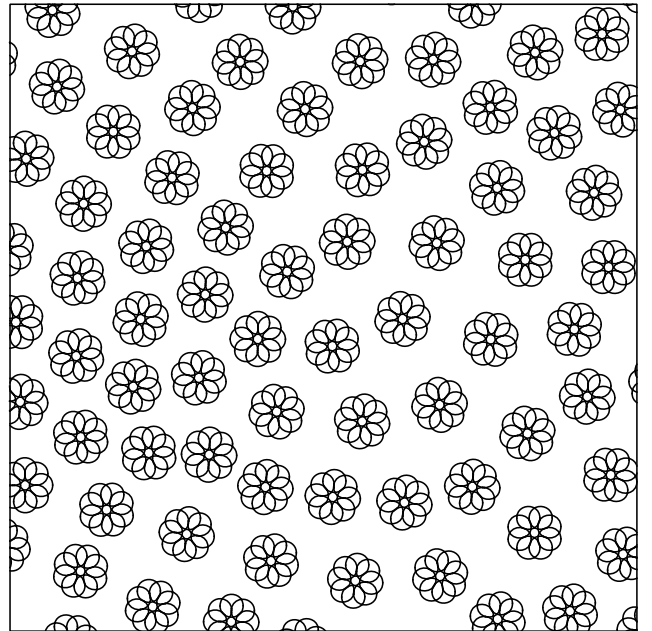
**Rosettes**

36 sec

```

1 def rosette():
2     size = 2000
3     line = ImportSVG("data/line.svg")
4     circle = ImportSVG("data/circle.svg")
5
6     # Uniform partition (around 70 faces)
7     props = IrregularProperties(70/(size*size))
8     part_tex = UniformPartition(props, KEEP_OUTSIDE)
9
10    # Mapper: create a star in each face
11    def face_to_star(face):
12        rot = Random(face, 0.0, 2.0*pi, 0)
13        elem1 = MatchPoint(line, BBoxCenter(line), Centroid(face))
14        elem2 = Append(Rotate(elem1, rot), Rotate(elem1, rot+pi/2))
15        elem3 = Append(elem2, Rotate(elem2, pi/4))
16        return elem3
17
18    # Mapper: place a circle on each vertex
19    def vertex_to_circle(vertex):
20        if (len(IncidentEdges(vertex)) > 1):
21            return Nothing()
22        return MatchPoint(circle, BBoxCenter(circle), Location(
23            vertex))
24
25    # Mapping operators
26    stars_tex = MapToFaces(face_to_star, part_tex)
27    flowers_tex = MapToVertices(vertex_to_circle, stars_tex)
28
29    # Final texture
30    ExportSVG(flowers_tex, size)

```

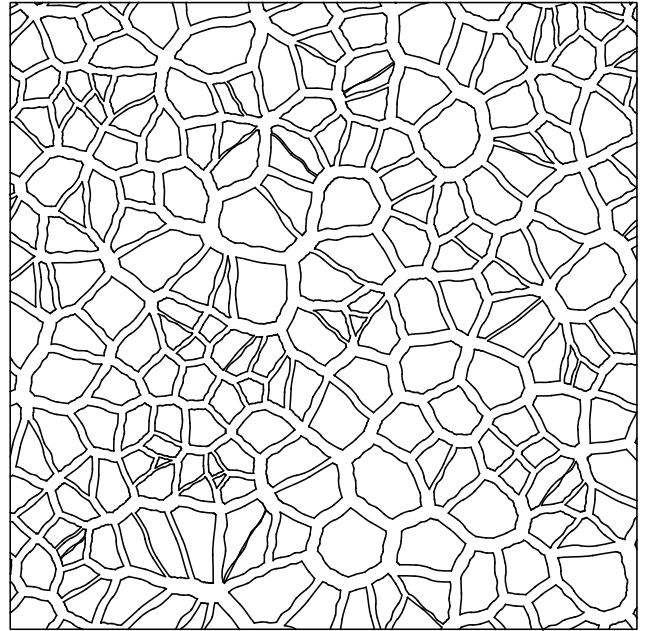


**Cracks**

35 sec

```

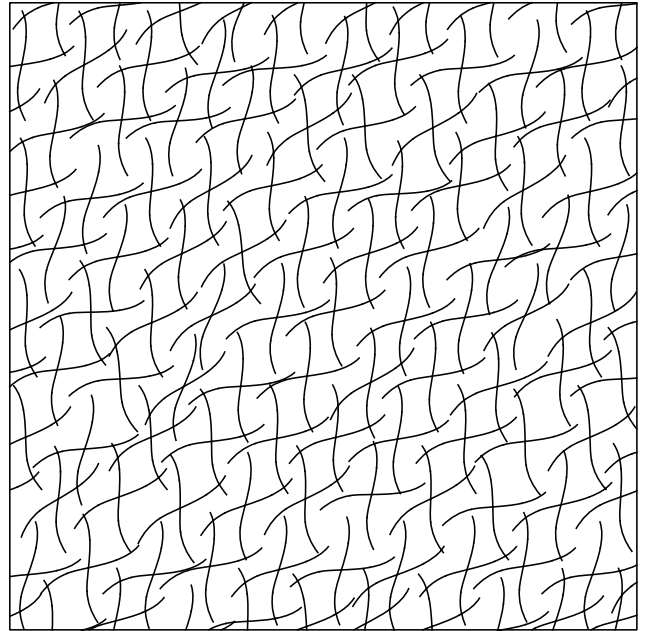
1 def cracks():
2     size = 2000
3     lines = [ImportSVG("data/line1.svg"),
4               ImportSVG("data/line2.svg"),
5               ImportSVG("data/line3.svg")]
6
7     # Random partition (around 200 faces)
8     props = IrregularProperties(200/(size*size))
9     part_tex = RandomPartition(props, KEEP_OUTSIDE)
10
11    # Mapper: rescale each face
12    def rescale_face(face):
13        return Scale(Contour(face), 0.8)
14
15    # Mapper: replace each edge by a random curved line
16    def edge_to_curve(edge):
17        line = lines[floor(Random(edge, 0, len(lines), 0))]
18        src_c = PointLabeled(line, "start")
19        dst_c = PointLabeled(line, "end")
20        src_v = Location(SourceVertex(edge))
21        dst_v = Location(TargetVertex(edge))
22        return MatchPoints(line, src_c, dst_c, src_v, dst_v)
23
24    # Mapping operators
25    reduced_tex = MapToFaces(rescale_face, part_tex)
26    cracks_tex = MapToEdges(edge_to_curve, reduced_tex)
27
28    # Final texture
29    ExportSVG(cracks_tex, size)
    
```


**By example limitations: bimodal hatching**

7 sec

```

1 def exemplar_based_a():
2     size = 2000
3     hatch = ImportSVG("data/hatch.svg")
4
5     # Grid partition, including face labels
6     lines1 = StripesProperties(0, size/10)
7     lines2 = StripesProperties(pi/2, size/20)
8     SetFaceLabels(lines1, "h1", "h2")
9     SetFaceLabels(lines2, "v1", "v2")
10    grid_tex = GridPartition(lines1, lines2, KEEP_OUTSIDE)
11
12    # Mapper: place 2 hatches in one face on two
13    def face_to_hatches(face):
14        if ((HasLabel(face, "v1") and HasLabel(face, "h1")) or
15            (HasLabel(face, "v2") and HasLabel(face, "h2"))):
16            elem1 = Scale(hatch, Random(face, 2.6, 2.7, 1))
17            elem2 = Rotate(elem1, pi/7+Random(face, -pi/12, pi/12, 2))
18            elem3 = Rotate(elem1, pi/2+Random(face, -pi/12, pi/12, 3))
19            elem4 = Append(elem2, elem3)
20            w = BBoxWidth(face)/7
21            h = BBoxHeight(face)/7
22            pos = Centroid(face)+Point(Random(face, -w, w, 4),
23                                       Random(face, -h, h, 5))
24            return MatchPoint(elem4, BBoxCenter(elem4), pos)
25        return Nothing()
26
27    # Mapping operator
28    hatch_tex = MapToFaces(face_to_hatches, grid_tex)
29
30    # Final texture
31    ExportSVG(hatch_tex, size)
    
```



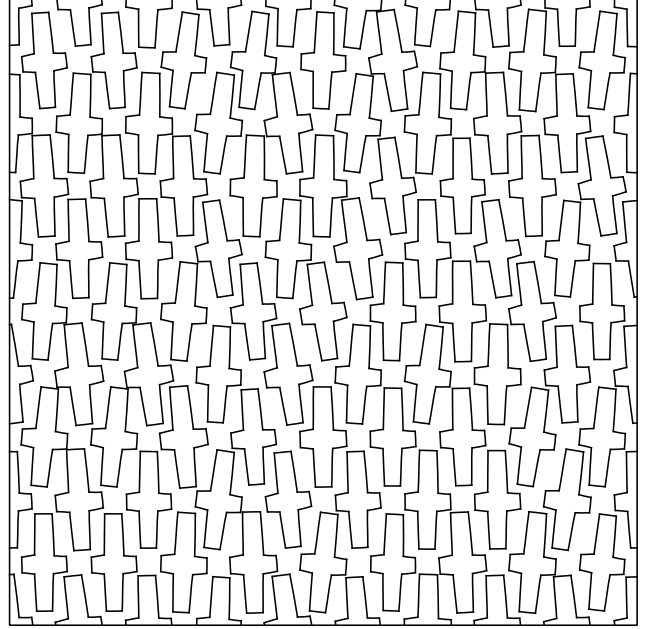
## By example limitations: regular crosses

6 sec

```

1 def exemplar_based_b():
2     size = 2000
3     cross = ImportSVG("data/cross.svg")
4
5     # Grid partition, including face labels
6     lines1 = StripesProperties(0, size/10)
7     lines2 = StripesProperties(pi/2, size/18)
8     SetFaceLabels(lines1, "h1", "h2")
9     SetFaceLabels(lines2, "v1", "v2")
10    grid_tex = GridPartition(lines1, lines2, KEEP_OUTSIDE)
11
12    # Mapper: place a cross in one face on two
13    def face_to_hatches(face):
14        if ((HasLabel(face, "v1") and HasLabel(face, "h1")) or
15            (HasLabel(face, "v2") and HasLabel(face, "h2"))):
16            elem1 = Scale(cross, Random(face, 1.35, 1.45, 2))
17            elem2 = Rotate(elem1, Random(face, -pi/20, pi/20, 4))
18            return MatchPoint(elem2, BBoxCenter(elem2), BBoxCenter(
19                face))
20        return Nothing()
21
22    # Mapping operator
23    hatch_tex = MapToFaces(face_to_hatches, grid_tex)
24
25    # Final texture
26    ExportSVG(hatch_tex, size)

```



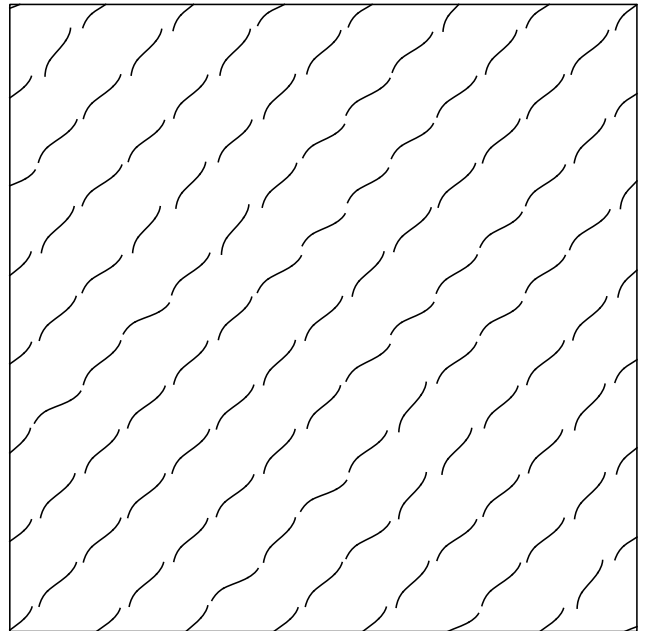
## By example limitations: regular curves

4 sec

```

1 def exemplar_based_c():
2     size = 2000
3     hatch = ImportSVG("data/hatch.svg")
4
5     # Grid partition
6     lines1 = StripesProperties(pi/4, size/10)
7     lines2 = StripesProperties(pi/2+pi/4, size/10)
8     grid_tex = GridPartition(lines1, lines2, KEEP_OUTSIDE)
9
10    # Mapper: place a curved line in each face
11    def face_to_hatch(face):
12        src_p = BBoxCenter(hatch)
13        dst_p = Centroid(face)
14        elem1 = Scale(hatch, Random(face, 1.3, 1.4, 0))
15        elem2 = Rotate(elem1, pi/4+Random(face, -pi/12, pi/12, 1))
16        return MatchPoint(elem2, src_p, dst_p)
17
18    # Mapping operator
19    hatch_tex = MapToFaces(face_to_hatch, grid_tex)
20
21    # Final texture
22    ExportSVG(hatch_tex, size)

```

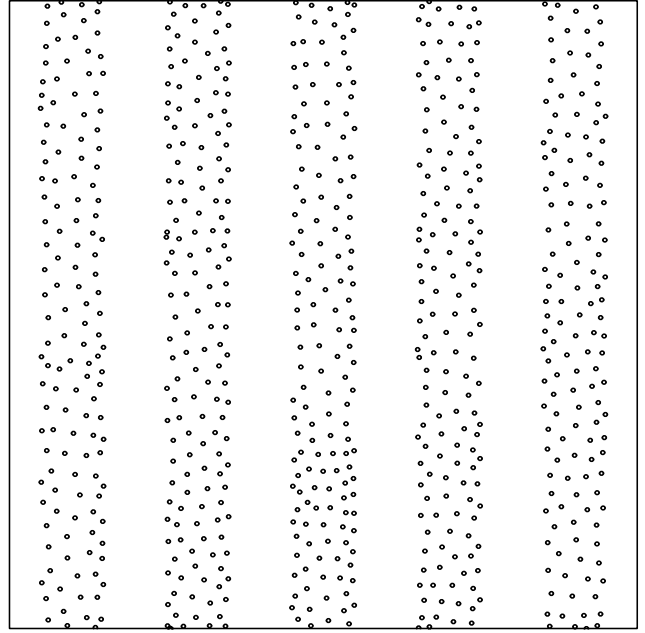


## By example limitations: clusters

42 sec

```

1 def exemplar_based_d():
2     size = 2000
3     circle = ImportSVG("data/circle.svg")
4
5     # Stripes partition, including face labels
6     lines = StripesProperties(pi/2, size/10)
7     SetFaceLabels(lines, "v1", "v2")
8     stripes_tex = StripesPartition(lines)
9
10    # Mapper: place a circle in each face
11    def face_to_circle(face):
12        return Scale(MatchPoint(circle, BBoxCenter(circle),
13                               Centroid(face)), 0.02)
14
15    # Mapper: creates a dot pattern in one face on two
16    def face_to_circles(face):
17        if HasLabel(face, "v2"):
18            props = IrregularProperties(100/(BBoxWidth(face)*
19                                           BBoxHeight(face)))
20            part = UniformPartition(props, CROP_ADD_BOUNDARY)
21            circ = MapToFaces(face_to_circle, part)
22            return circ(face)
23        return Nothing()
24
25    # Mapping operator
26    texture = MapToFaces(face_to_circles, stripes_tex)
27
28    # Final texture
29    ExportSVG(texture, size)
    
```

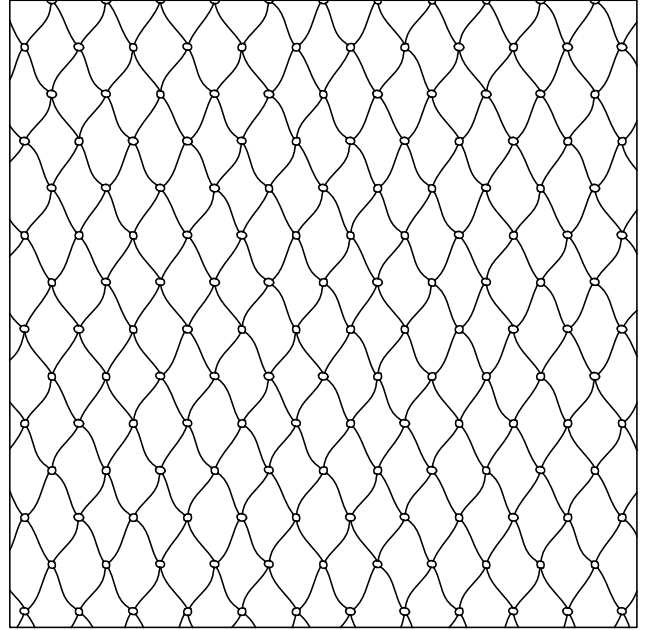


## Teaser: example a

25 sec

```

1 def teaser_a():
2     # Load multiple curve and blob shapes
3     curves = [ImportSVG("data/curve1.svg"),
4               ImportSVG("data/curve2.svg"),
5               ImportSVG("data/curve3.svg"),
6               ImportSVG("data/curve4.svg")]
7     blobs = [ImportSVG("data/blob1.svg"),
8              ImportSVG("data/blob2.svg"),
9              ImportSVG("data/blob3.svg"),
10              ImportSVG("data/blob4.svg")]
11     size = 2000
12
13     # Mapper: replace each edge by a random curve
14     def edge_to_curve(edge):
15         curve = curves[floor(Random(edge, 0, len(curves), 0))]
16         src_c = PointLabeled(curve, "bottom")
17         dst_c = PointLabeled(curve, "top")
18         src_v = Location(SourceVertex(edge))
19         dst_v = Location(TargetVertex(edge))
20         return MatchPoints(curve, src_c, dst_c, src_v, dst_v)
21
22     # Mapper: replace each vertex by a random blob
23     def vertex_to_blob(vertex):
24         blob = blobs[floor(Random(vertex, 0, len(blobs), 0))]
25         return MatchPoint(blob, BBoxCenter(blob), Location(vertex))
26
27     # Grid Partition
28     lines1 = StripesProperties(pi/3, 150)
29     lines2 = StripesProperties(-pi/3, 150)
30     grid_tex = GridPartition(lines1, lines2, KEEP_OUTSIDE)
31
32     # Mapping operators
33     curve_tex = MapToEdges(edge_to_curve, grid_tex)
34     blob_tex = MapToVertices(vertex_to_blob, grid_tex)
35
36     # Combining
37     final_tex = Union(blob_tex, Outside(curve_tex, blob_tex, CROP))
38
39     # Final texture
40     ExportSVG(final_tex, size)
    
```



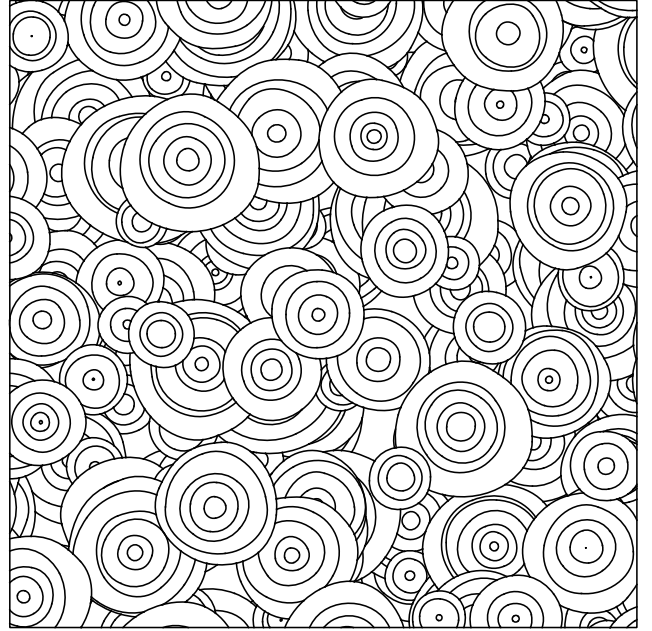
## Teaser: example b

2 min, 9 sec

```

1  # create a texture with non-overlapping concentric circles
2  def circles_in_circles(density):
3      cir = [ImportSVG("data/circle1.svg"),
4              ImportSVG("data/circle2.svg"),
5              ImportSVG("data/circle3.svg")]
6
7      # Uniform partition
8      props = IrregularProperties(density)
9      part = UniformPartition(props, KEEP_OUTSIDE)
10
11     # Mapper: place circles in each face
12     def face_to_circles(face):
13         all_circ = Nothing()
14         cur_scale = Random(face, 0.3, 0.8, 0);
15
16         # Perturb shape and scale for each circle
17         i = 0
18         while cur_scale > 0.0:
19             circle = cir[floor(Random(face, 0, len(circles), i))]
20             cur_circ = Scale(MatchFace(circle, face), cur_scale)
21             cur_circ = Rotate(cur_circ, Random(face, 0, 2*pi, i+10))
22             all_circ = Append(all_circ, cur_circ)
23             cur_scale = cur_scale - Random(face, 0.09, 0.2, i+20)
24             i = i+1
25
26         return all_circ
27
28     return MapToFaces(face_to_circles, part)
29
30 def teaser_b():
31     size = 2000
32
33     # Create a first layer of circles
34     tex = circles_in_circles(15/(size*size))
35
36     # Combine layers
37     for i in range(0, 9):
38         tmp = circles_in_circles(15/(size*size))
39         tex = Union(tmp, Outside(tex, tmp, CROP))
40
41     ExportSVG(tex, size)

```



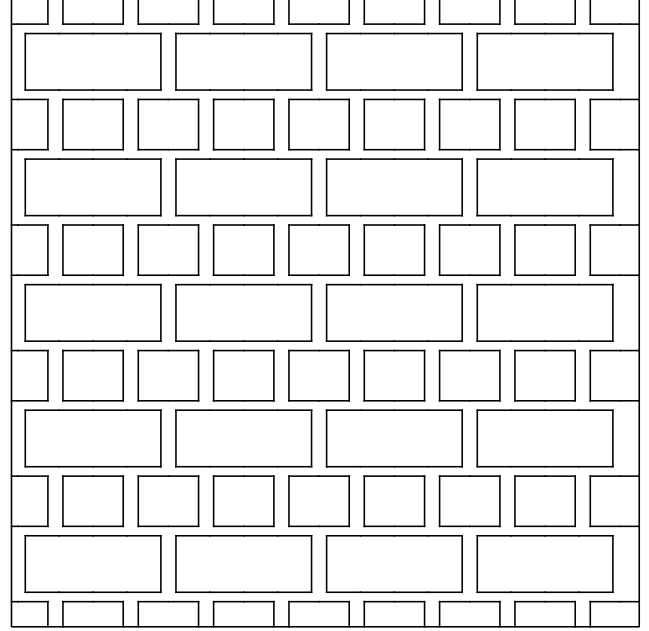


**Teaser: example c**

1 sec

```

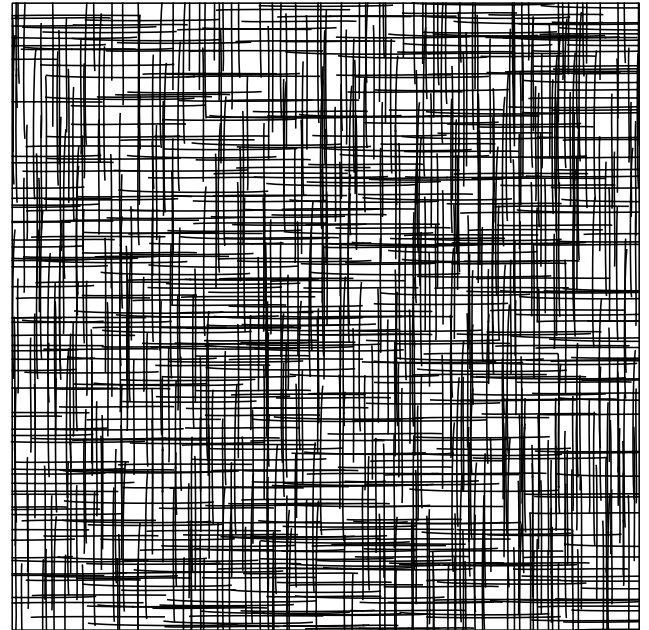
1 def teaser_c():
2     size = 2000
3
4     # Grid partition, including edge labels
5     lines1 = StripesProperties(0,200)
6     lines2 = StripesProperties(pi/2,120)
7     SetEdgeLabels(lines1,"h1","h2")
8     SetEdgeLabels(lines2,"v1","v2","v3","v4")
9     grid_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)
10
11    # Mapper: remove edges containing specific label sequences
12    def remove_edge(edge):
13        if HasLabel(edge,"h1") or HasLabel(edge,"h2"):
14            return ToCurve(edge)
15        if HasLabel(IncidentEdges(TargetVertex(edge)),"h1"):
16            if HasLabel(edge,"v1"):
17                return ToCurve(edge)
18            else:
19                return Nothing()
20        else:
21            if HasLabel(edge,"v2") or HasLabel(edge,"v4"):
22                return ToCurve(edge)
23            else:
24                return Nothing()
25
26    # Mapper: rescale each face
27    def shrink_face(face):
28        if HasLabel(IncidentEdges(face),"v1"):
29            return Scale(Contour(face),0.9)
30        else:
31            return Scale(Contour(face),0.8)
32
33    # Mapping operator
34    wall_tex = MapToEdges(remove_edge,grid_tex)
35    shrink_tex = MapToFaces(shrink_face,wall_tex)
36
37    # Final texture
38    ExportSVG(shrink_tex,size)
    
```


**Teaser: example d**

3 min, 40 sec

```

1 def teaser_d():
2     # Load several curves
3     curves = [ImportSVG("data/bighatch1.svg"),
4               ImportSVG("data/bighatch2.svg"),
5               ImportSVG("data/bighatch3.svg"),
6               ImportSVG("data/bighatch4.svg")]
7     size = 2000
8
9     # Uniform partition (with around 1000 faces)
10    props = IrregularProperties(1000/(size*size))
11    init_tex = UniformPartition(props,KEEP_OUTSIDE)
12
13    # Mapper: place a vertical or horizontal curve on each face
14    def face_to_hatches(face):
15        curve = curves[floor(Random(face,0,len(curves),0))]
16        if Random(face,0,1,1)>0.5:
17            curve = Rotate(curve,pi/2)
18        return MatchPoint(curve,BBoxCenter(curve),Centroid(face))
19
20    # Mapping operator
21    hatch_tex = MapToFaces(face_to_hatches,init_tex)
22
23    # Final texture
24    ExportSVG(hatch_tex,size)
    
```



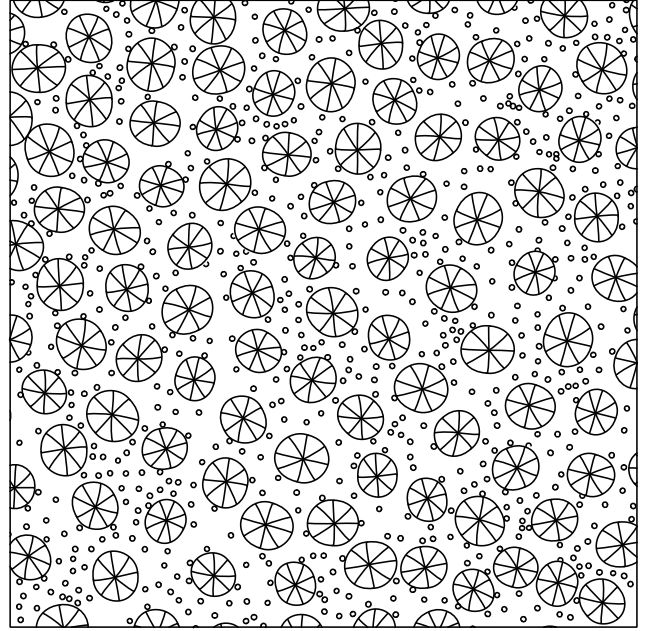
## Teaser: example e

20 sec

```

1 def teaser_e():
2     size = 2000
3     wheel = ImportSVG("data/wheel1.svg")
4     stipple = ImportSVG("data/stipple1.svg")
5
6     # Uniform partition (around 100 faces)
7     props1 = IrregularProperties(100/(size*size))
8     tex1 = UniformPartition(props1, KEEP_OUTSIDE)
9
10    # Random partition (around 1200 faces)
11    props2 = IrregularProperties(1200/(size*size))
12    tex2 = RandomPartition(props2, KEEP_OUTSIDE)
13
14    # Mapper: place a wheel in each face
15    def face_to_wheel(face):
16        w = Scale(Rotate(wheel, Random(face, 0, 2*pi, 0)), Random(face,
17            0.8, 1, 1))
18        return MatchPoint(w, BBoxCenter(w), Centroid(face))
19
20    # Mapper: place a stipple in each face
21    def face_to_stipples(face):
22        s = Scale(Rotate(stipple, Random(face, 0, 2*pi, 0)), Random(face,
23            0.9, 1, 1))
24        return MatchPoint(s, BBoxCenter(s), Centroid(face))
25
26    # Mapping operators
27    tex_wheel = MapToFaces(face_to_wheel, tex1)
28    tex_stipples = MapToFaces(face_to_stipples, tex2)
29
30    # Combining operators
31    texture = Union(tex_wheel, Outside(tex_stipples, tex_wheel,
32        KEEP_INSIDE))
33
34    # Final texture
35    ExportSVG(texture, size)

```



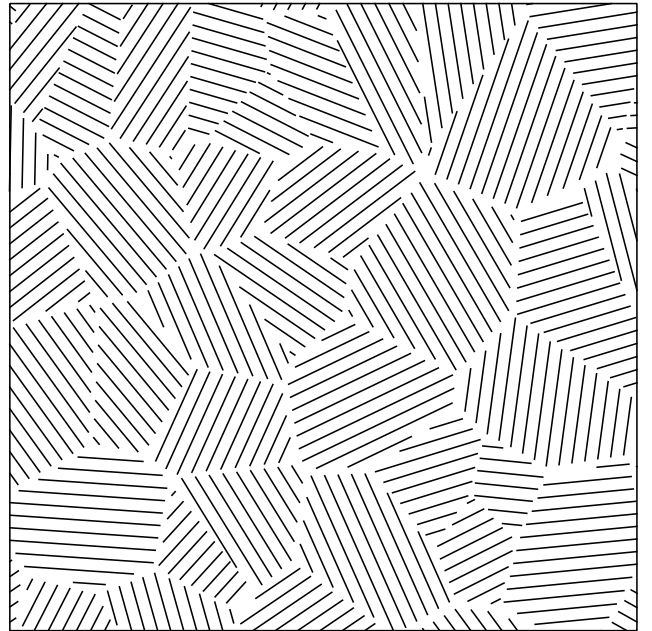
## Teaser: example f

3 sec

```

1 def teaser_f():
2     # Random partition (around 30 faces)
3     size = 2000
4     props = IrregularProperties(30/(size*size))
5     init_tex = RandomPartition(props, KEEP_OUTSIDE)
6
7     # Mapper: rescale each face
8     def scale_map(face):
9         return Scale(Contour(face), 0.95)
10
11    # Mapper: produces stripes on each face
12    def hatch_map(face):
13        angle = Random(face, 0, 2*pi, 1)
14        lines = StripesProperties(angle, 40)
15        return StripesPartition(lines)(face)
16
17    # Mapper: remove boundary edges
18    def border_map(edge):
19        if IsBoundary(edge):
20            return Nothing()
21        return ToCurve(edge)
22
23    # Mapping operators
24    tex1 = MapToFaces(scale_map, init_tex)
25    tex2 = MapToFaces(hatch_map, tex1)
26    tex3 = MapToEdges(border_map, tex2)
27
28    # Final texture
29    ExportSVG(tex3, size)

```

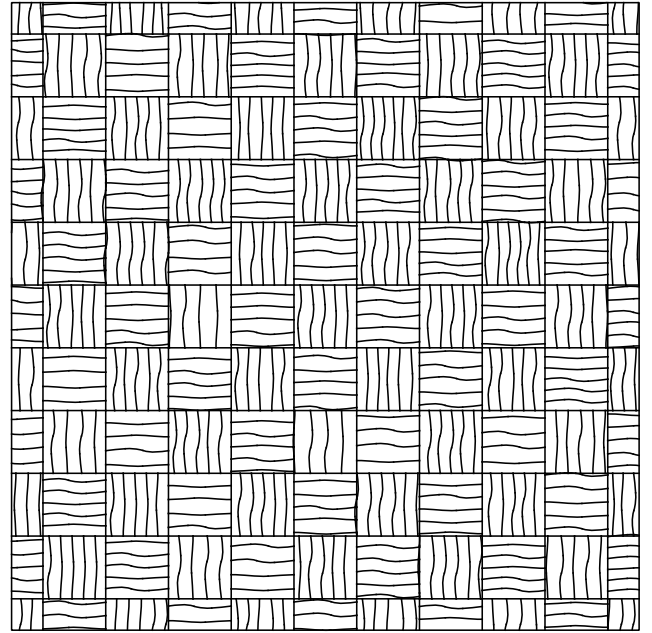


Teaser: example g

6 sec

```

1 def teaser_g():
2     curves = [ImportSVG("data/line1.svg"),
3                 ImportSVG("data/line2.svg")]
4
5     # Grid partition, including face labels
6     size = 2000
7     lines1 = StripesProperties(0,200)
8     lines2 = StripesProperties(pi/2,200)
9     SetFaceLabels(lines1,"h1","h2")
10    SetFaceLabels(lines2,"v1","v2")
11    grid1_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)
12
13    # Mapper: replace each edge by a random curve
14    def edge_to_curve(edge):
15        if IsBoundary(edge):
16            return ToCurve(edge)
17        curve = curves[floor(Random(edge,0,len(curves),0))]
18        src_c = PointLabeled(curve,"start")
19        dst_c = PointLabeled(curve,"end")
20        src_v = Location(SourceVertex(edge))
21        dst_v = Location(TargetVertex(edge))
22        if Random(edge,0,1,1)<0.5:
23            return MatchPoints(curve,src_c,dst_c,src_v,dst_v)
24        else:
25            return MatchPoints(curve,dst_c,src_c,src_v,dst_v)
26
27    # Mapper: creates stripes in each face
28    def face_to_stripes(face):
29        width = BBoxWidth(face)/Random(face,4,6,0)
30        theta = 0
31        if((HasLabel(face,"h1") and HasLabel(face,"v1")) or
32            (HasLabel(face,"h2") and HasLabel(face,"v2"))):
33            theta = pi/2
34        lines = StripesProperties(theta,width)
35        stripes = StripesPartition(lines)
36        return MapToEdges(edge_to_curve,stripes)(face)
37
38    # Mapping operator
39    tiled_tex = MapToFaces(face_to_stripes,grid1_tex)
40
41    # Final texture
42    ExportSVG(tiled_tex,size)
    
```



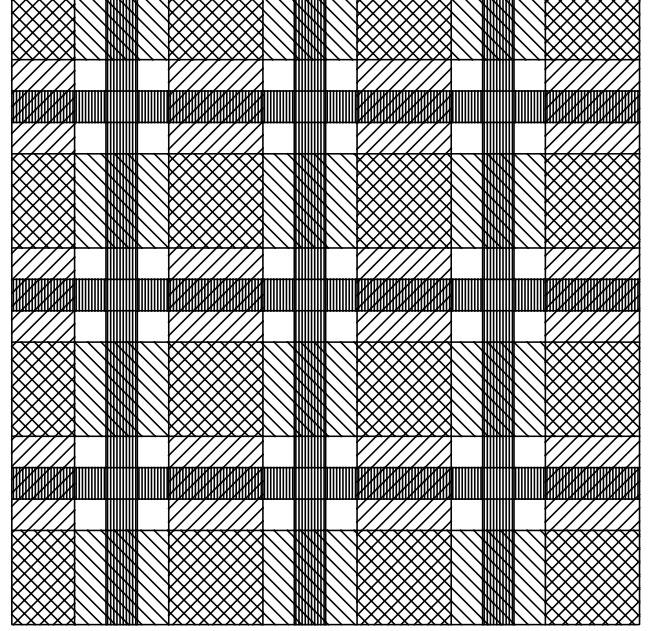
## Teaser: example h

26 sec

```

1 def teaser_h():
2     size = 2000
3
4     # Stripe partition
5     lines1 = StripesProperties(0.0,300)
6     SetFaceTags(lines1,"h1","h2")
7     stripes1 = StripesPartition(lines1)
8
9     # Stripe partition
10    lines2 = StripesProperties(pi/2,300)
11    SetFaceLabels(lines2,"v1","v2")
12    stripes2 = StripesPartition(lines2)
13
14    # Grid partition with labels
15    lines3 = StripesProperties(0.0,100,100,100,300)
16    SetFaceLabels(lines3,"h1","h2","h3","h4")
17    lines4 = StripesProperties(pi/2,100,100,100,300)
18    SetFaceLabels(lines4,"v1","v2","v3","v4")
19    grid = GridPartition(lines3,lines4,KEEP_OUTSIDE)
20
21    # Create a stripe partition with specified angle and width
22    def hatch(angle,width):
23        lines = StripesProperties(angle,width)
24        return StripesPartition(lines)
25
26    # Mapper: create -pi/4 stripes in each face
27    def hatch_map_stripes1(f):
28        angle = -pi/4
29        if HasLabel(f,"h1"):
30            return hatch(angle,30)(f)
31        else:
32            return Nothing()
33
34    # Mapper: create pi/4 stripes in each face
35    def hatch_map_stripes2(f):
36        angle = pi/4
37        if HasLabel(f,"v1"):
38            return hatch(angle,30)(f)
39        else:
40            return Nothing()
41
42    # Mapper: create pi/2 stripes in specific faces
43    def hatch_map_grid(f):
44        angle = pi/2
45        if HasLabel(f,"h1"):
46            return hatch(angle,10)(f)
47        elif HasLabel(f,"v1"):
48            return hatch(angle,10)(f)
49        else:
50            return Nothing()
51
52    # Mapping operators
53    tex1 = MapToFaces(hatch_map_stripes1, stripes1)
54    tex2 = MapToFaces(hatch_map_stripes2, stripes2)
55    tex4 = MapToFaces(hatch_map_grid, grid)
56
57    # Merging operators
58    tex3 = Union(tex1, tex2)
59    tex5 = Union(tex3, tex4)
60
61    ExportSVG(tex5, size)

```

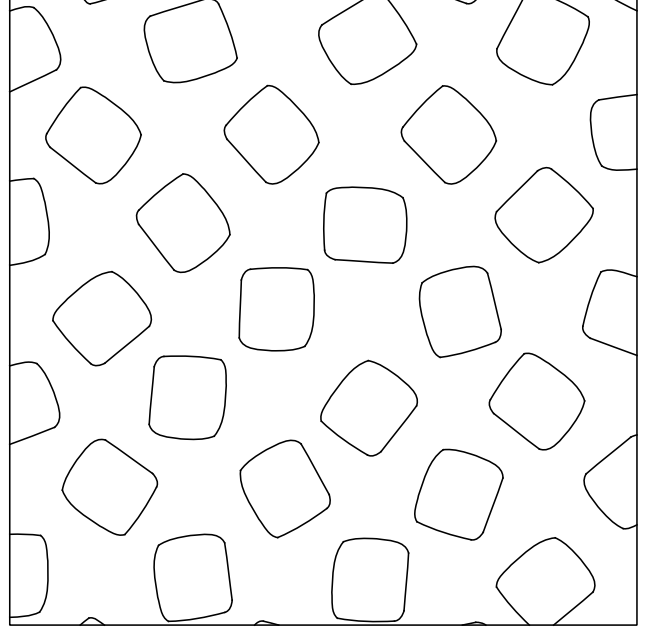


**Combining a: texture 1**

1 sec

```

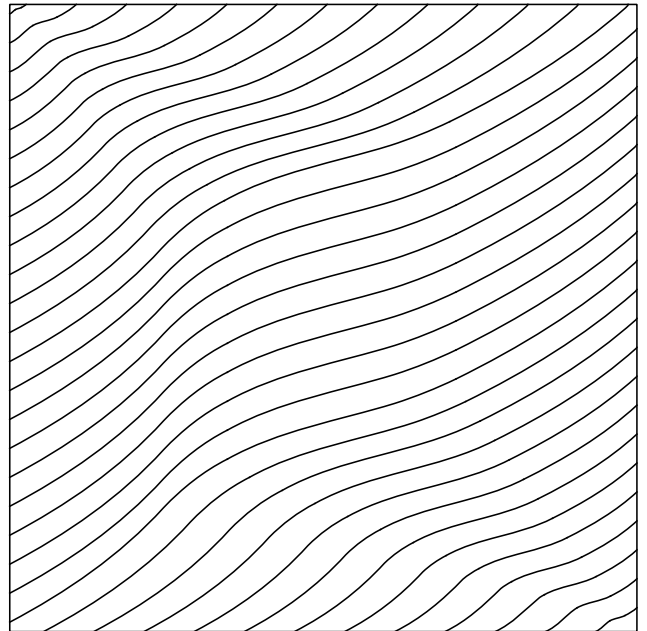
1 def squares(theta, width):
2     # Grid partition with given width and orientation
3     lines1 = StripesProperties(theta, width)
4     lines2 = StripesProperties(theta+pi/2.0, width)
5     grid_tex = GridPartition(lines1, lines2, KEEP_OUTSIDE)
6     square = ImportSVG("data/square.svg")
7
8     # Mapper: place a rounded square in each face
9     def face_to_square(face):
10         return Rotate(MatchFace(square, face), Random(face, 0.0, 2.0*
11             pi, 1))
12
13     # Return the texture generated via the mapping operator
14     return MapToFaces(face_to_square, grid_tex)
15
16 def combine_a():
17     tex1 = squares(pi/4, 400)
18     ExportSVG(tex1, 2000)
    
```


**Combining b: texture 2**

1 sec

```

1 def curves(theta, width):
2     # Stripes partition with given width and orientation
3     props = StripesProperties(theta, width)
4     stripes = StripesPartition(props)
5     line = ImportSVG("data/line6.svg")
6
7     # Mapper: replace each edge by a curve
8     def line_to_curve(edge):
9         if IsBoundary(edge):
10             return Nothing()
11         src_c = PointLabeled(line, "start")
12         dst_c = PointLabeled(line, "end")
13         src_v = Location(SourceVertex(edge))
14         dst_v = Location(TargetVertex(edge))
15         return MatchPoints(line, src_c, dst_c, src_v, dst_v)
16
17     # Return the texture generated via the mapping operator
18     return MapToEdges(line_to_curve, stripes)
19
20 def combine_b():
21     tex2 = curves(pi/6, 80)
22     ExportSVG(tex2, 2000)
    
```



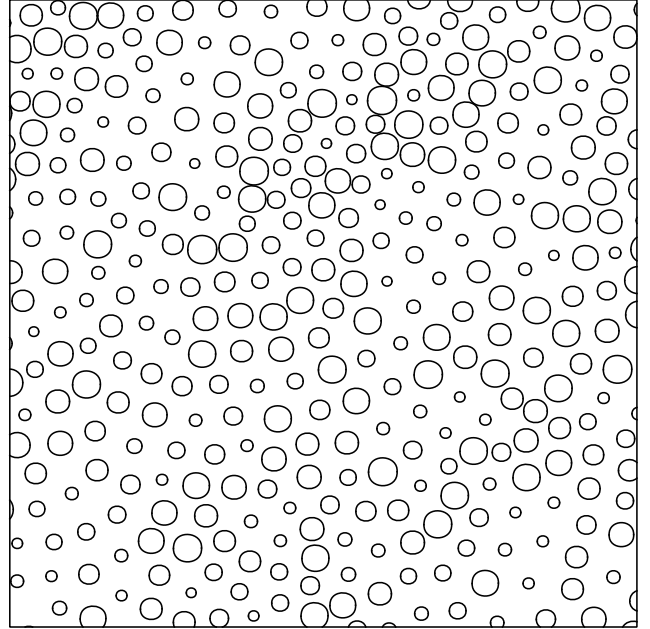
**Combining c: texture 3**

13 sec

```

1 def circles(density):
2     # Uniform partition with given density
3     props = IrregularProperties(density)
4     part = UniformPartition(props, KEEP_OUTSIDE)
5     circle = ImportSVG("data/circle.svg")
6
7     # Mapper: place a circle in each face
8     def face_to_circle(face):
9         src_p = BBoxCenter(circle)
10        dst_p = Centroid(face)
11        return Scale(MatchPoint(circle, src_p, dst_p), Random(face
12                        ,0.05,0.15,0))
13
14    # Mapping operator
15    return MapToFaces(face_to_circle, part)
16
17 def combine_c():
18     tex3 = circles(8.5e-5)
19     ExportSVG(tex3, 2000)

```

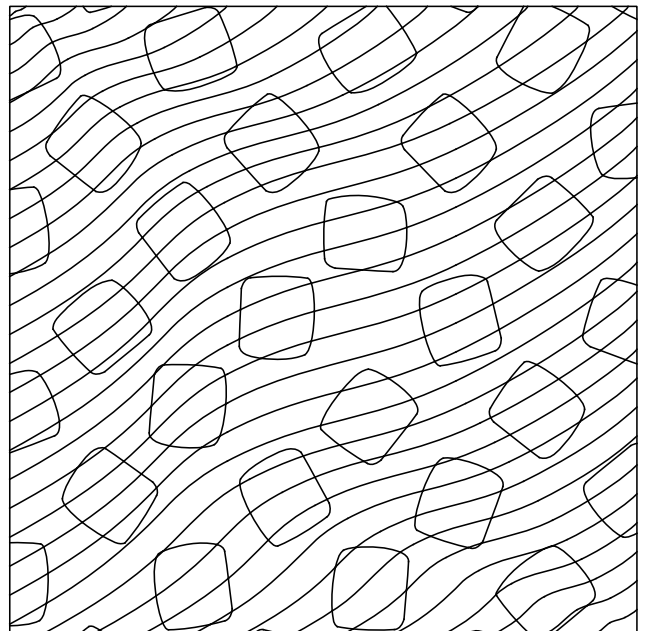
**Combining d: merging tex1 and tex2**

4 sec

```

1 def combine_d():
2     tex1 = squares(pi/4, 400)
3     tex2 = curves(pi/6, 80)
4     texf = Union(tex1, tex2)
5     ExportSVG(texf, 2000)

```

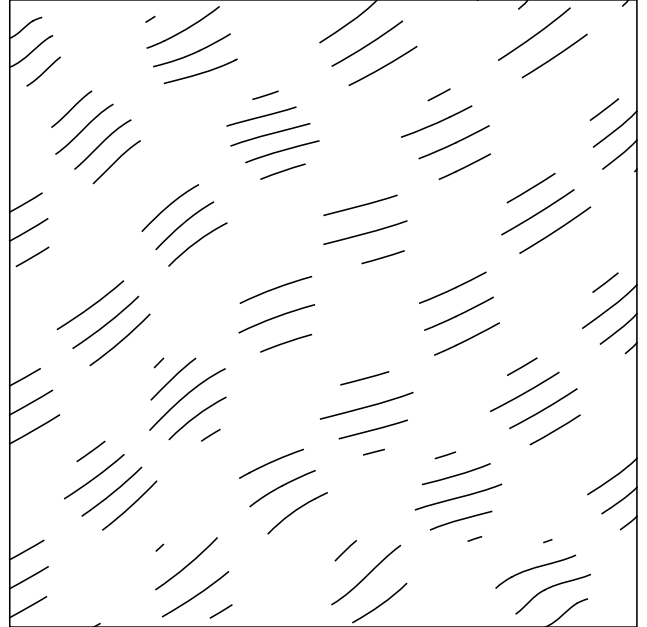


**Combining e: tex2 inside tex1 (crop)**

3 sec

```

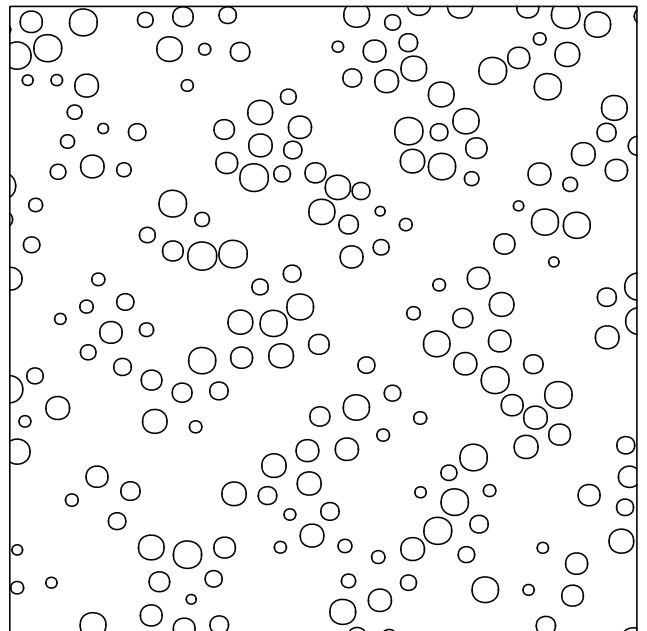
1 def combine_e():
2     tex1 = squares(pi/4,400)
3     tex2 = curves(pi/6,80)
4     texf = Inside(tex2,tex1,CROP)
5     ExportSVG(texf,2000)
    
```


**Combining f: tex3 inside tex1 (keep outside)**

15 sec

```

1 def combine_f():
2     tex1 = squares(pi/4,400)
3     tex3 = circles(8.5e-5)
4     texf = Inside(tex3,tex1,KEEP_OUTSIDE)
5     ExportSVG(texf,2000)
    
```





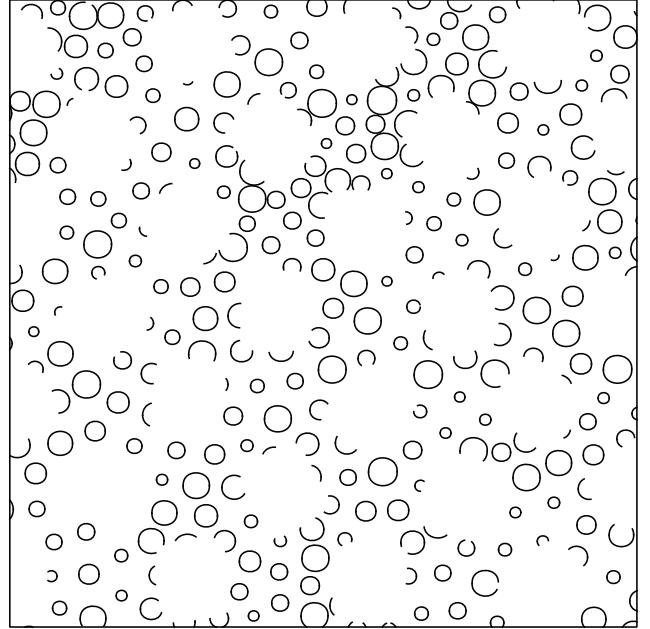
**Combining g: tex3 outside tex1 (crop)**

15 sec

```

1 def combine_g():
2   tex1 = squares(pi/4,400)
3   tex3 = circles(8.5e-5)
4   texf = Outside(tex3, tex1, CROP)
5   ExportSVG(texf, 2000)

```

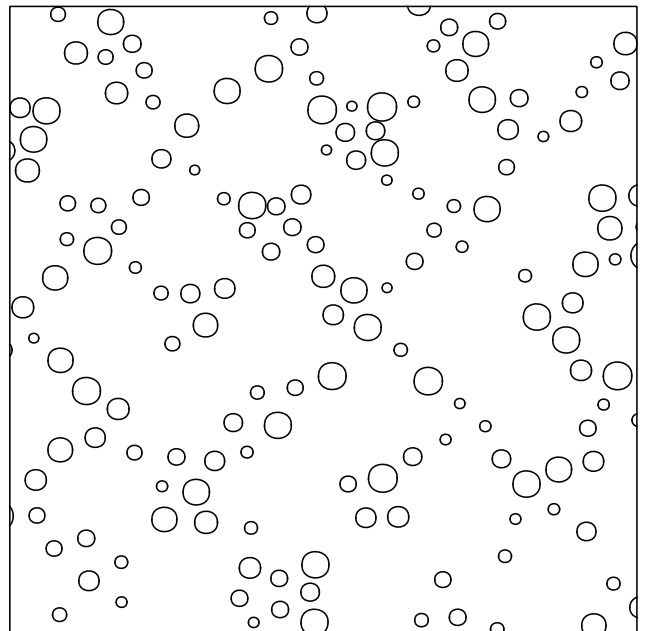
**Combining h: tex3 outside tex1 (keep inside)**

15 sec

```

1 def combine_h():
2   tex1 = squares(pi/4,400)
3   tex3 = circles(8.5e-5)
4   texf = Outside(tex3, tex1, KEEP_INSIDE)
5   ExportSVG(texf, 2000)

```



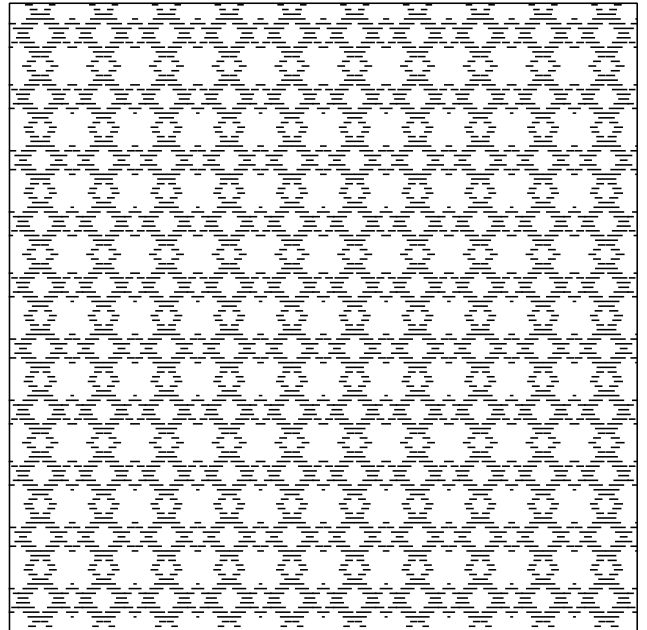


**Regular result (a)**

30 sec

```

1 def complex_regular01():
2     size = 2000
3
4     # Grid partition
5     lines1 = StripesProperties(0.0,200)
6     lines2 = StripesProperties(pi/2.0,200)
7     grid = GridPartition(lines1, lines2, KEEP_OUTSIDE)
8
9     # Stripe partition
10    lines = StripesProperties(0,15)
11    hatch = StripesPartition(lines)
12
13    # Mapper: rotate faces with a pi/4 angle
14    def rotate_map(f):
15        return Rotate(Contour(f), pi/4)
16
17    # Mapper: scale each face with a factor 0.8
18    def scale_map(f):
19        return Scale(Contour(f), 0.8)
20
21    # Mapping operators (rotate/scale)
22    tex1 = MapToFaces(rotate_map, grid)
23    tex2 = MapToFaces(scale_map, tex1)
24
25    # Combining operator
26    tex3 = Outside(hatch, tex2, CROP)
27
28    ExportSVG(tex3, size)
    
```



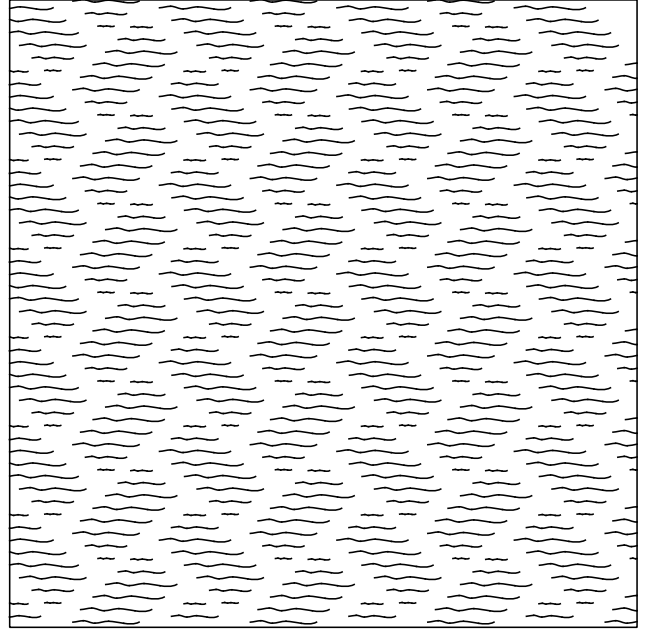
## Regular result (b)

39 sec

```

1 def complex_regular02():
2     size = 2000
3     line = ImportSVG("data/line3.svg")
4
5     # Grid partition and labels
6     lines1 = StripesProperties(pi/4.0,200)
7     lines2 = StripesProperties(-pi/4.0,200)
8     SetEdgeTags(lines1,"h1","h2","h3","h4")
9     SetEdgeTags(lines2,"v1","v2","v3","v4")
10    grid = GridPartition(lines1,lines2,KEEP_OUTSIDE)
11
12    # Mapper: keep or remove edge
13    def grid_to_V_mapper(edge):
14        if HasTag(edge,"v1") and HasTag(IncidentEdges(
15            TargetVertex(edge)), "h1"):
16            return Nothing()
17        if HasTag(edge,"v2") and HasTag(IncidentEdges(
18            TargetVertex(edge)), "h2"):
19            return Nothing()
20        if HasTag(edge,"v3") and HasTag(IncidentEdges(
21            TargetVertex(edge)), "h3"):
22            return Nothing()
23        if HasTag(edge,"v4") and HasTag(IncidentEdges(
24            TargetVertex(edge)), "h4"):
25            return Nothing()
26        if HasTag(edge,"h1") and HasTag(IncidentEdges(
27            TargetVertex(edge)), "v3"):
28            return Nothing()
29        if HasTag(edge,"h2") and HasTag(IncidentEdges(
30            TargetVertex(edge)), "v4"):
31            return Nothing()
32        if HasTag(edge,"h3") and HasTag(IncidentEdges(
33            TargetVertex(edge)), "v1"):
34            return Nothing()
35        if HasTag(edge,"h4") and HasTag(IncidentEdges(
36            TargetVertex(edge)), "v2"):
37            return Nothing()
38        return ToCurve(edge)
39
40    # Mapper: rescale face
41    def scale_map(f):
42        return Scale(Contour(f), 0.8)
43
44    # Mapper: replace each edge by a curve
45    def map_curve_to_edge(edge):
46        start = PointTagged(line,"start")
47        end = PointTagged(line,"end")
48        if IsBoundary(edge):
49            return Nothing()
50        else:
51            return MatchPoints(line,start,end,Location(
52                SourceVertex(edge)),Location(TargetVertex(edge)))
53
54    # Create a stripe partition
55    def hatch(angle):
56        lines = StripesProperties(angle,40)
57        return StripesPartition(lines)
58
59    # Mapper: create stripes in each face
60    def hatch_map(f):
61        angle = 0.0
62        return hatch(angle)(f)
63
64    # Mapping operator: select grid edges
65    tex1 = MapToEdges(grid_to_V_mapper,grid)
66
67    # Mapping operator: rescale faces
68    tex2 = MapToFaces(scale_map, tex1)
69
70    # Mapping operator: place hatches faces
71    tex3 = MapToFaces(hatch_map, tex2)
72
73    # Mapping operator: replace hatches by smooth curves
74    tex4 = MapToEdges(map_curve_to_edge, tex3)
75
76    ExportSVG(tex4,size)

```

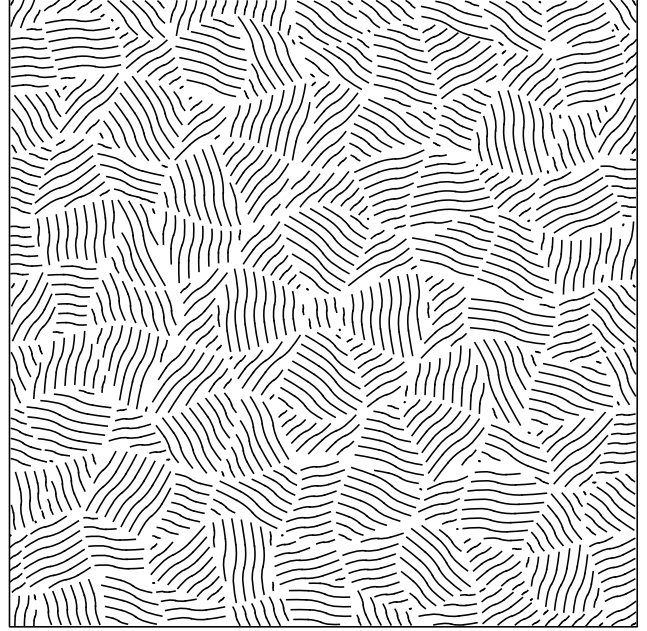


Regular result (c)

30 sec

```

1 def complex_regular03():
2     size = 2000
3     line = ImportSVG("data/line4.svg")
4     line2 = ImportSVG("data/line2.svg")
5
6     # Stripe partitions and their labels
7     lines1 = StripesProperties(0,200)
8     lines2 = StripesProperties(pi/2,200)
9     SetEdgeLabels(lines1,"h1","h2")
10    SetEdgeLabels(lines2,"h1","h2")
11    stripes1 = StripesPartition(lines1)
12    stripes2 = StripesPartition(lines2)
13
14    # Mapper: replace each edge by a curve
15    def map_curve_to_edge(edge):
16        start = PointTagged(line,"start")
17        end = PointTagged(line,"end")
18        if IsBoundary(edge):
19            return ToCurve(edge)
20        elif HasLabel(edge,"h1"):
21            return MatchPoints(line,start,end,Location(
22                SourceVertex(edge),Location(TargetVertex(edge)))
23        else:
24            return MatchPoints(line,end,start,Location(
25                SourceVertex(edge),Location(TargetVertex(edge)))
26
27    # Mapper: replace each edge by a curve
28    def map_curve_to_edge2(edge):
29        start = PointTagged(line2,"start")
30        end = PointTagged(line2,"end")
31        if IsBoundary(edge):
32            return Nothing()
33        else:
34            return MatchPoints(line2,start,end,Location(
35                SourceVertex(edge),Location(TargetVertex(edge)))
36
37    # Mapper: create stripes in each face with random
38    # orientations
39    def hatch_map(f):
40        angle = Random(f,0,2*pi,0)
41        lines = StripesProperties(angle,30)
42        hatch = StripesPartition(lines)
43        return hatch(f)
44
45    # Mapper: rescale each face
46    def scale_map(f):
47        return Scale(Contour(f), 0.9)
48
49    # Mapping / combining operators
50    tex1 = MapToEdges(map_curve_to_edge, stripes1)
51    tex2 = MapToEdges(map_curve_to_edge, stripes2)
52    tex3 = Union(tex1,tex2)
53    tex4 = MapToFaces(scale_map, tex3)
54    tex5 = MapToFaces(hatch_map, tex4)
55    tex6 = MapToEdges(map_curve_to_edge2, tex5)
56
57    ExportSVG(tex6,size)
    
```



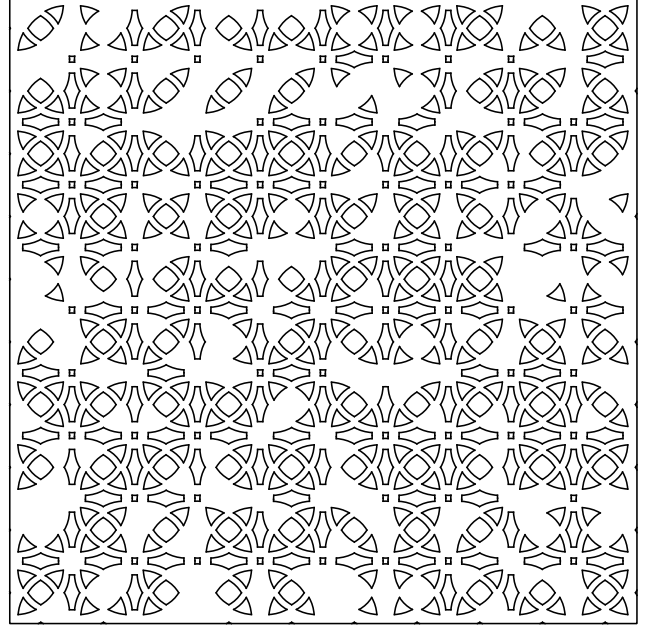
**Regular result (d)**

25 sec

```

1 def complex_regular04():
2     size = 2000
3     circle = ImportSVG("data/circle.svg")
4
5     # Grid partition
6     lines1 = StripesProperties(pi/2.0,200)
7     lines2 = StripesProperties(pi,200)
8     partition1 = GridPartition(lines1,lines2,KEEP_OUTSIDE)
9
10    # Mapper: place a circle in each face
11    def circle_map(f):
12        return Scale(MatchFace(circle,f), 2.5)
13
14    # Mapper: rescale face
15    def scale_map(f):
16        return Scale(Contour(f), 0.65)
17
18    # Mapper: keep or remove face
19    def delete_map(f):
20        r = Random(f,0,1,2)
21        if r<0.3:
22            return Nothing()
23        else:
24            return Contour(f)
25
26    # Mapping operators
27    tex1 = MapToFaces(circle_map,partition1)
28    tex2 = MapToFaces(scale_map,tex1)
29    tex3 = MapToFaces(delete_map,tex2)
30
31    ExportSVG(tex3,size)

```

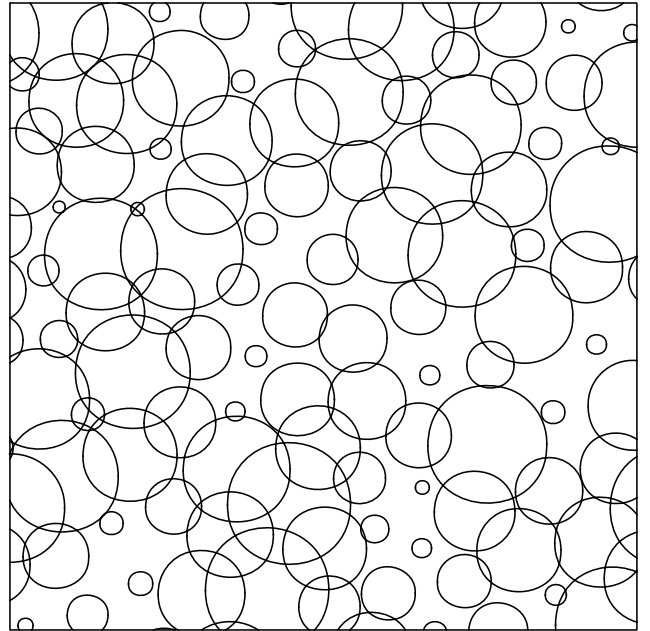
**Non regular result (a) left**

6 sec

```

1 def complex_non_regular01():
2     size = 2000
3     circle = ImportSVG("data/circle.svg")
4
5     # Uniform partition
6     density = 100/(size*size)
7     props = IrregularProperties(density)
8     partition = UniformPartition(props, KEEP_OUTSIDE)
9
10    # Mapper: place a circle in each face
11    def circle_map(f):
12        return Scale(MatchFace(circle,f), Random(f,0.2,2.0,2))
13
14    # Mapping operator
15    tex1 = MapToFaces(circle_map,partition)
16
17    ExportSVG(tex1,size)

```

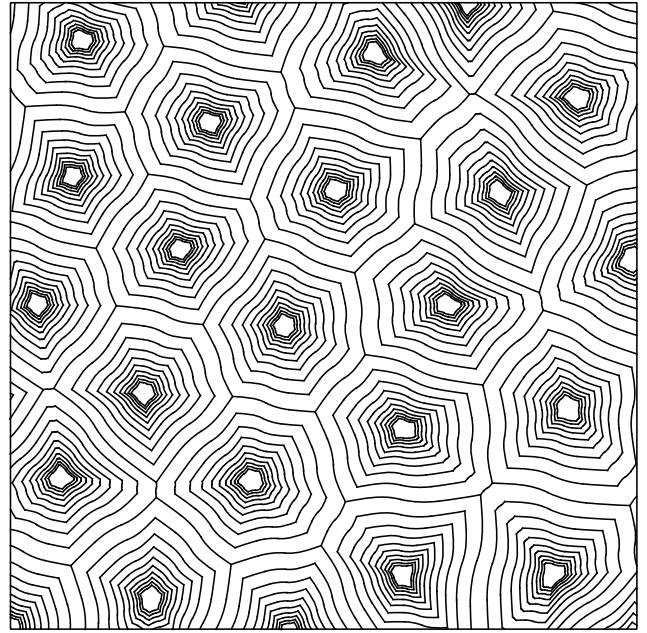


## Non regular result (a) right

29 sec

```

1 def complex_non_regular02():
2     size = 2000
3     line = ImportSVG("data/line2.svg")
4
5     # Uniform partition
6     density = 20/(2000*2000)
7     props = IrregularProperties(density)
8     partition = UniformPartition(props, KEEP_OUTSIDE)
9
10    # Mapper: replace each edge by a curve
11    def map_curve_to_edge(edge):
12        start = PointTagged(line, "start")
13        end = PointTagged(line, "end")
14        return MatchPoints(line, start, end, Location(SourceVertex(
15            edge)), Location(TargetVertex(edge)))
16
17    # Mapper: rescale a face
18    def scale_map(f):
19        return Scale(Contour(f), 0.8)
20
21    # Mapping operators (iteratively rescale faces)
22    l1 = MapToEdges(map_curve_to_edge, partition)
23    l2 = MapToFaces(scale_map, l1)
24    l3 = MapToFaces(scale_map, l2)
25    l4 = MapToFaces(scale_map, l3)
26    l5 = MapToFaces(scale_map, l4)
27    l6 = MapToFaces(scale_map, l5)
28    l7 = MapToFaces(scale_map, l6)
29    l8 = MapToFaces(scale_map, l7)
30    l9 = MapToFaces(scale_map, l8)
31    l10 = MapToFaces(scale_map, l9)
32
33    # Combining operators: merge all arrangements
34    tex = Union(Union(Union(Union(Union(Union(Union(Union(
35        l1, l2), l3), l4), l5), l6), l7), l8), l9), l10)
36
37    ExportSVG(tex, size)
    
```

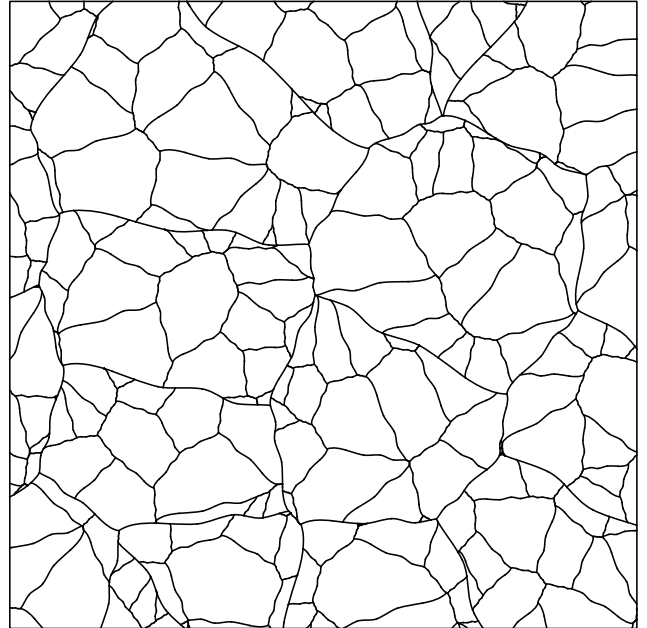


## Non regular result (b)

12 sec

```

1 def complex_non_regular03():
2     size = 2000
3     line = ImportSVG("data/line3.svg")
4
5     # Random partition
6     density = 10/(size*size)
7     props = IrregularProperties(density)
8     partition = RandomPartition(props, KEEP_OUTSIDE)
9
10    # Mapper: replace internal edges by curves
11    def map_curve_to_edge(edge):
12        start = PointTagged(line, "start")
13        end = PointTagged(line, "end")
14        if IsBoundary(edge):
15            return ToCurve(edge)
16        else:
17            return MatchPoints(line, end, start, Location(
18                SourceVertex(edge)), Location(TargetVertex(edge)))
19
20    # Mapper: create a random partition in each face
21    def partition_map(f):
22        density2 = 100/(size*size)
23        props2 = IrregularProperties(density2)
24        partition2 = RandomPartition(props2, CROP_ADD_BOUNDARY)
25        return partition2(f)
26
27    # Mapping operators
28    tex1 = MapToEdges(map_curve_to_edge, partition)
29    tex2 = MapToFaces(partition_map, tex1)
30    tex3 = MapToEdges(map_curve_to_edge, tex2)
31
32    ExportSVG(tex3, size)
    
```



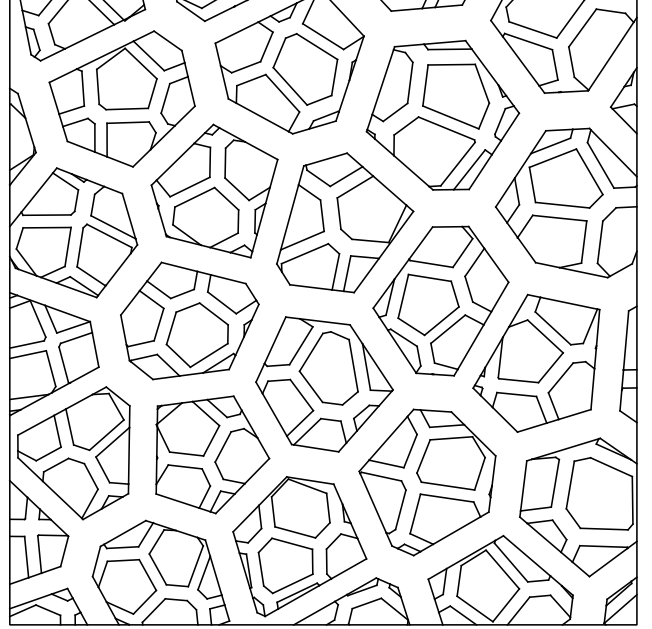
## Non regular result (c)

30 sec

```

1 def complex_non_regular04():
2     size = 2000
3
4     # Uniform partition
5     density1 = 20/(size*size)
6     props1 = IrregularProperties(density1)
7     partition1 = UniformPartition(props1, KEEP_OUTSIDE)
8
9     # Uniform partition
10    density2 = 90/(size*size)
11    props2 = IrregularProperties(density2)
12    partition2 = UniformPartition(props2, KEEP_OUTSIDE)
13
14    # Mapper: rescale faces
15    def scale_map(f):
16        return Scale(Contour(f), 0.8)
17
18    # Mapper: create a uniform partition in each face
19    def partition2_map(f):
20        return MapToFaces(scale_map, partition2)(f)
21
22    # Mapping operators
23    tex1 = MapToFaces(scale_map, partition1)
24    tex2 = MapToFaces(scale_map, partition2)
25
26    # Combining operators
27    tex3 = Inside(tex2, tex1, CROP)
28    tex4 = Union(tex1, tex3)
29
30    ExportSVG(tex4, size)

```



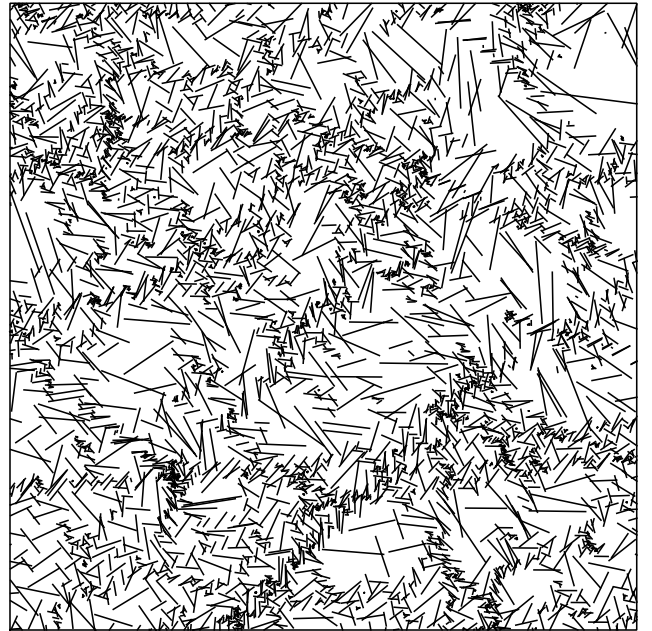
## Non regular result (d)

1 min, 28 sec

```

1 def complex_non_regular05():
2     size = 2000
3
4     # Random partition
5     density = 30/(2000*2000)
6     props = IrregularProperties(density)
7     partition = RandomPartition(props, KEEP_OUTSIDE)
8
9     # Mapper: rotate edge
10    def rotate_e(e):
11        return Rotate(ToCurve(e), pi/4)
12
13    # Mapper: rescale edge
14    def scale_map_5(e):
15        return Scale(ToCurve(e), 5.0)
16
17    # Mapper: rescale edge
18    def scale_map_10(e):
19        return Scale(ToCurve(e), 10.0)
20
21    # Mapping operators
22    tex1 = MapToEdges(scale_map_5, partition)
23    tex2 = MapToEdges(scale_map_10, partition)
24    tex3 = MapToEdges(rotate_e, tex2)
25
26    ExportSVG(tex3, size)

```



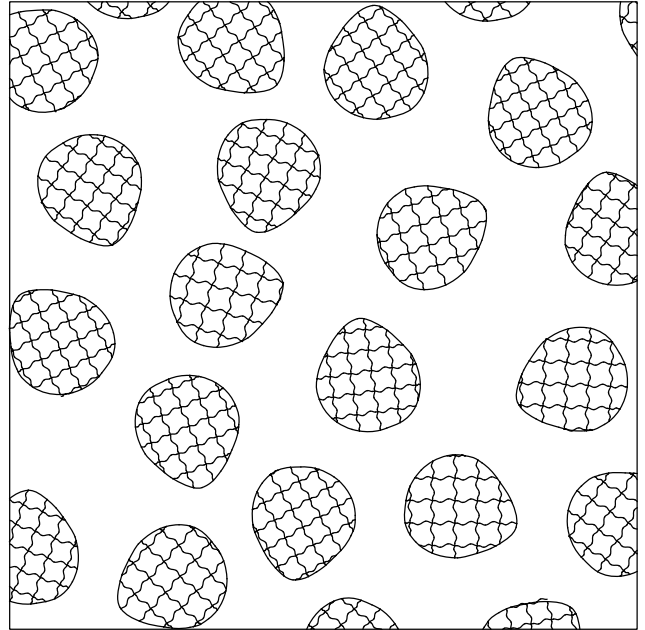


## Script edition (a) original script

25 sec

```

1 def overview_orig():
2     size = 2500
3     blob = ImportSVG("data/blob.svg")
4     zig = ImportSVG("data/zig.svg")
5
6     # Mapper: place a blob in each face
7     def map_blob_to(face):
8         new_blob = Rotate(blob, Random(face, 0, 2*pi, 0))
9         return MatchPoint(new_blob, BBoxCenter(new_blob), Centroid(
10             face))
11
12     # Mapper: replace each edge by a curved line
13     def map_curve_to(edge):
14         if IsBoundary(edge):
15             return ToCurve(edge)
16         src_c = PointLabeled(zig, "start")
17         dst_c = PointLabeled(zig, "end")
18         src_v = Location(SourceVertex(edge))
19         dst_v = Location(TargetVertex(edge))
20         return MatchPoints(zig, src_c, dst_c, src_v, dst_v)
21
22     def create_blob_tex():
23         # Uniform partition
24         props = IrregularProperties(11/(2000*2000))
25         init_tex = UniformPartition(props, KEEP_OUTSIDE)
26
27         # Mapping operators
28         return MapToFaces(map_blob_to, init_tex)
29
30     # Mapper: generate a texture in each face
31     def create_zig_tex(face):
32         # Grid partition with randomized orientations
33         theta = Random(face, 0, 2*pi, 1)
34         width = BBoxWidth(face)/5
35         lines1 = StripesProperties(theta, width)
36         lines2 = StripesProperties(theta+pi/2, width)
37         init_tex = GridPartition(lines1, lines2, CROP_ADD_BOUNDARY)
38
39         # Mapping operator
40         texture2 = MapToEdges(map_curve_to, init_tex)
41         return texture2(face)
42
43     init_tex = create_blob_tex()
44     final_tex = MapToFaces(create_zig_tex, init_tex)
45
46     # Export final texture
47     ExportSVG(final_tex, size)
    
```



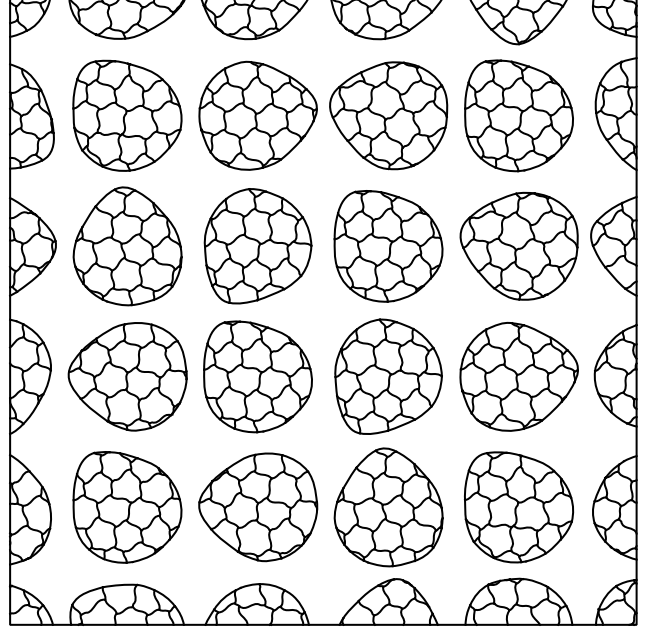
## Script edition (b) switch partitions

52 sec

```

1 def overview_var1():
2     size = 2500
3     blob = ImportSVG("data/blob.svg")
4     zig = ImportSVG("data/zig.svg")
5
6     # Mapper: place a blob in each face
7     def map_blob_to(face):
8         new_blob = Rotate(blob, Random(face, 0, 2*pi, 0))
9         return MatchPoint(new_blob, BBoxCenter(new_blob), Centroid(
10             face))
11
12     # Mapper: replace each edge by a curved line
13     def map_curve_to(edge):
14         if IsBoundary(edge):
15             return ToCurve(edge)
16         src_c = PointLabeled(zig, "start")
17         dst_c = PointLabeled(zig, "end")
18         src_v = Location(SourceVertex(edge))
19         dst_v = Location(TargetVertex(edge))
20         return MatchPoints(zig, src_c, dst_c, src_v, dst_v)
21
22     def create_blob_tex():
23         # Grid partition
24         theta = 0
25         width = size/4
26         lines1 = StripesProperties(theta, width)
27         lines2 = StripesProperties(theta+pi/2, width)
28         init_tex = GridPartition(lines1, lines2, KEEP_OUTSIDE)
29
30         # Mapping operators
31         return MapToFaces(map_blob_to, init_tex)
32
33     # Mapper: generate a texture in each face
34     def create_zig_tex(face):
35         # Uniform partition
36         props = IrregularProperties(11/(BBoxWidth(face)*
37             BBoxHeight(face)))
38         init_tex = UniformPartition(props, CROP_ADD_BOUNDARY)
39
40         # Mapping operator
41         texture2 = MapToEdges(map_curve_to, init_tex)
42         return texture2(face)
43
44     init_tex = create_blob_tex()
45     final_tex = MapToFaces(create_zig_tex, init_tex)
46
47     # Export final texture
48     ExportSVG(final_tex, size)

```



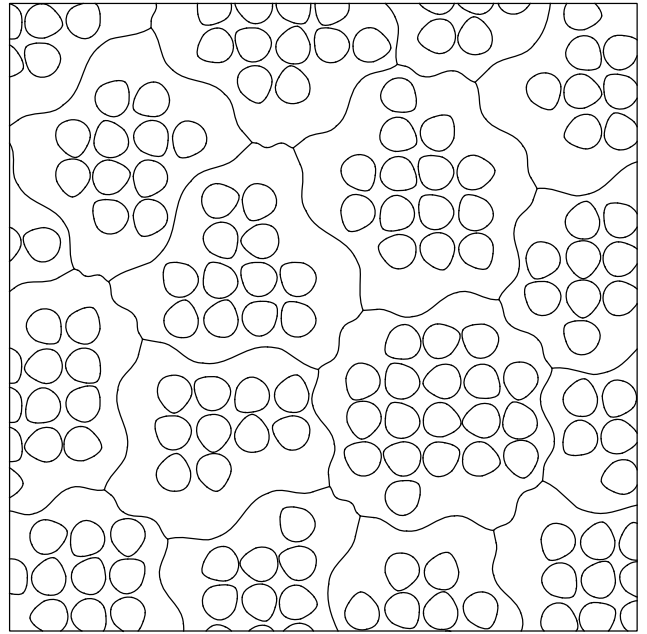


## Script edition (c) switch arrangements

12 sec

```

1 def overview_var2():
2     size = 2500
3     blob = ImportSVG("data/blob.svg")
4     zig = ImportSVG("data/zig.svg")
5
6     # Mapper: place a blob in each face
7     def map_blob_to(face):
8         new_blob = Rotate(blob, Random(face, 0, 2*pi, 0))
9         return MatchPoint(new_blob, BBoxCenter(new_blob), Centroid(
            face))
10
11    # Mapper: replace each edge by a curved line
12    def map_curve_to(edge):
13        if IsBoundary(edge):
14            return ToCurve(edge)
15        src_c = PointLabeled(zig, "start")
16        dst_c = PointLabeled(zig, "end")
17        src_v = Location(SourceVertex(edge))
18        dst_v = Location(TargetVertex(edge))
19        return MatchPoints(zig, src_c, dst_c, src_v, dst_v)
20
21    def create_blob_tex(face):
22        # Grid partition
23        theta = 0 #Random(face, 0, 2*pi, 0)
24        width = size/16
25        lines1 = StripesProperties(theta, width)
26        lines2 = StripesProperties(theta+pi/2, width)
27        init_tex = GridPartition(lines1, lines2, KEEP_INSIDE)
28
29        # Mapping operators
30        return MapToFaces(map_blob_to, init_tex)(face)
31
32    # Mapper: generate a texture in each face
33    def create_zig_tex():
34        # Uniform partition
35        props = IrregularProperties(11/(size*size))
36        init_tex = UniformPartition(props, KEEP_OUTSIDE)
37
38        # Mapping operator
39        return MapToEdges(map_curve_to, init_tex)
40
41    init_tex = create_zig_tex()
42    final_tex = Union(init_tex, MapToFaces(create_blob_tex,
        init_tex))
43
44    # Export final texture
45    ExportSVG(final_tex, size)
    
```



## Script edition (d) vary mappings

56 sec

```

1 def overview_var3():
2     size = 2500
3     flower = ImportSVG("data/flower.svg")
4     leaf = ImportSVG("data/leaf.svg")
5     zig = ImportSVG("data/zig7.svg")
6
7     # Mapper: place a blob in each face
8     def map_blob_to(face):
9         new_flower = Scale(Rotate(flower, Random(face, 0, 2*pi, 0)),
10                             Random(face, 0.7, 1, 1))
11         new_leaf = Scale(Rotate(leaf, Random(face, 0, 2*pi, 2)),
12                           Random(face, 0.7, 1, 3))
13         if Random(face, 0, 1, 4) > 0.5:
14             return MatchPoint(new_flower, BBoxCenter(new_flower),
15                               Centroid(face))
16         else:
17             return MatchPoint(new_leaf, BBoxCenter(new_leaf),
18                               Centroid(face))
19
20     # Mapper: replace each edge by a curved line
21     def map_curve_to(edge):
22         if IsBoundary(edge):
23             return ToCurve(edge)
24         src_c = PointLabeled(zig, "start")
25         dst_c = PointLabeled(zig, "end")
26         src_v = Location(SourceVertex(edge))
27         dst_v = Location(TargetVertex(edge))
28         return MatchPoints(zig, src_c, dst_c, src_v, dst_v)
29
30     def create_blob_tex(face):
31         # Grid partition
32         theta = 0
33         width = size/16
34         lines1 = StripesProperties(theta, width)
35         lines2 = StripesProperties(theta+pi/2, width)
36         init_tex = GridPartition(lines1, lines2, KEEP_INSIDE)
37
38         # Mapping operators
39         return MapToFaces(map_blob_to, init_tex)(face)
40
41     # Mapper: generate a texture in each face
42     def create_zig_tex():
43         # Uniform partition
44         props = IrregularProperties(11/(size*size))
45         init_tex = UniformPartition(props, KEEP_OUTSIDE)
46
47         # Mapping operator
48         return MapToEdges(map_curve_to, init_tex)
49
50     init_tex = create_zig_tex()
51     final_tex = Union(init_tex, MapToFaces(create_blob_tex,
52                                             init_tex))
53
54     # Export final texture
55     ExportSVG(final_tex, size)

```

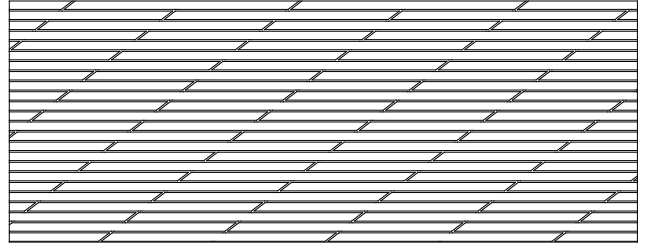


## Parquetry

20 sec

```

1 def parquetry():
2     # Label filtering: transforms a grid partition into
3     # parquetry slats with space in between
4     def parquetry_structure(e):
5         faces = Faces()
6         faces.push_back(LeftFace(e))
7         faces.push_back(RightFace(e))
8         if not LeftFace(e).is_bounded_face() \
9         or not RightFace(e).is_bounded_face():
10            return ToCurve(e)
11         if HasTag(faces, "v1") and HasTag(faces, "v2") and
12            HasTag(faces, "h1") \
13         or HasTag(faces, "v2") and HasTag(faces, "v3") and
14            HasTag(faces, "h1") \
15         or HasTag(faces, "v3") and HasTag(faces, "v4") and
16            HasTag(faces, "h1") \
17         or HasTag(faces, "v4") and HasTag(faces, "v1") and
18            HasTag(faces, "h1") \
19         or HasTag(faces, "v1") and HasTag(faces, "v2") and
20            HasTag(faces, "h3") \
21         or HasTag(faces, "v2") and HasTag(faces, "v3") and
22            HasTag(faces, "h3") \
23         or HasTag(faces, "v3") and HasTag(faces, "v4") and
24            HasTag(faces, "h3") \
25         or HasTag(faces, "h1") and HasTag(faces, "h2") and
26            HasTag(faces, "v1") \
27         or HasTag(faces, "h2") and HasTag(faces, "h3") and
28            HasTag(faces, "v1") \
29         or HasTag(faces, "h3") and HasTag(faces, "h4") and
30            HasTag(faces, "v3") \
31         or HasTag(faces, "h4") and HasTag(faces, "h1") and
32            HasTag(faces, "v3") \
33         or HasTag(faces, "v2") and HasTag(faces, "v3") and
34            HasTag(faces, "h2") \
35         or HasTag(faces, "v3") and HasTag(faces, "v4") and
36            HasTag(faces, "h2") \
37         or HasTag(faces, "v4") and HasTag(faces, "v1") and
38            HasTag(faces, "h4") \
39         or HasTag(faces, "v1") and HasTag(faces, "v2") and
40            HasTag(faces, "h4"):
41            return Nothing()
42            return ToCurve(e)
43
44     # Grid partition
45     my_scale = 50
46     interstice_size = 0.2*my_scale
47     lines1 = StripesProperties(0.0,1*my_scale,interstice_size
48                               ,1*my_scale,interstice_size)
49     SetFaceLabels(lines1, "h1", "h2", "h3", "h4")
50     lines2 = StripesProperties(pi/6+pi/32+pi/100,8*my_scale,
51                               interstice_size,interstice_size)
52     SetFaceLabels(lines2, "v1", "v2", "v3", "v4")
53     part = GridPartition(lines1,lines2,KEEP_OUTSIDE)
54     # Refinement
55     struct = MapToEdges(parquetry_structure,part)
56     # Export
57     frame = ImportSVG("data/new_carpet_outline.svg")
58     ExportSVGinDomain(struct, frame)
    
```



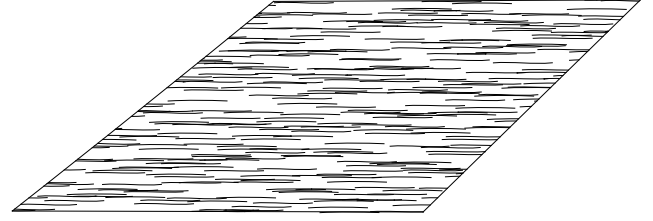
Table

45 sec

```

1 def table():
2     # Import elements
3     hatch1 = ImportSVG("data/smoothhatch1.svg")
4     hatch2 = ImportSVG("data/smoothhatch2.svg")
5     hatch3 = ImportSVG("data/smoothhatch3.svg")
6     hatches = [hatch1, hatch2, hatch3]
7     # Parameter for modifying hatch orientation
8     side_angle = 0
9
10    # Mapper: place a hatch on a vertex
11    def place_hatch(v):
12        randx = Random(v, -20, 20, 1)
13        randy = Random(v, -80, 80, 2)
14        loc = Location(v) + Point(randx, randy)
15        elem0 = hatches[floor(Random(0, len(hatches)))]
16        elem = Rotate(elem0, side_angle)
17        return MatchPoint(Scale(elem, 1.0), BBoxCenter(elem), loc)
18
19    # Grid partition
20    linesA = StripesProperties(side_angle + pi/3.0, 60)
21    linesB = StripesProperties(side_angle + 2.0*pi/3.0, 150)
22    part = GridPartition(linesA, linesB, KEEP_OUTSIDE)
23    # Apply the mapper
24    texture = MapToVertices(place_hatch, part)
25    # Fill the outline
26    frame = ImportSVG("data/outline_table_topside.svg")
27    ExportSVGinDomain(texture, frame)

```



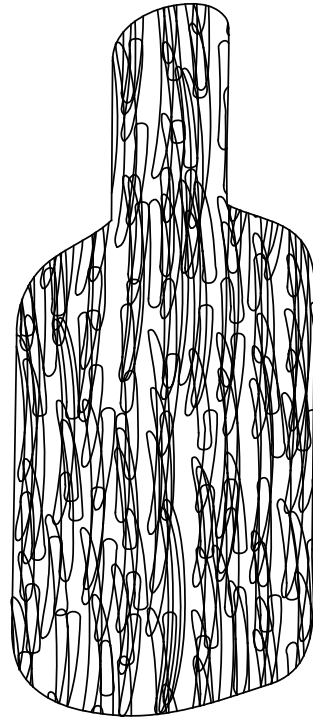
Bottle

11 sec

```

1 frame = ImportSVG("data/outline_bottle.svg")
2 bubble1 = ImportSVG("data/bubble1.svg")
3 bubble2 = ImportSVG("data/bubble2.svg")
4 bubble3 = ImportSVG("data/bubble3.svg")
5 bubble4 = ImportSVG("data/bubble4.svg")
6 bubbles = [bubble1, bubble2, bubble3, bubble4]
7 side_angle = pi / 2.0
8
9 # Mapper: place a bubble on a vertex
10 def place_bubble(v):
11     randx = Random(v, -20, 20, 1)
12     randy = Random(v, -40, 40, 2)
13     loc = Location(v) + Point(randx, randy)
14     elem0 = bubbles[floor(Random(0, len(bubbles)))]
15     elem = Rotate(elem0, side_angle)
16     return MatchPoint(Scale(elem, 1.0), BBoxCenter(elem), loc)
17
18 # Grid partition
19 linesA = StripesProperties(side_angle + pi/3.0, 60)
20 linesB = StripesProperties(side_angle + 2*pi/3.0, 60)
21 part = GridPartition(linesA, linesB, KEEP_OUTSIDE)
22 # Apply the mapper
23 texture = MapToVertices(place_bubble, part)
24 # Fill the outline
25 ExportSVGinDomain(texture, frame)

```

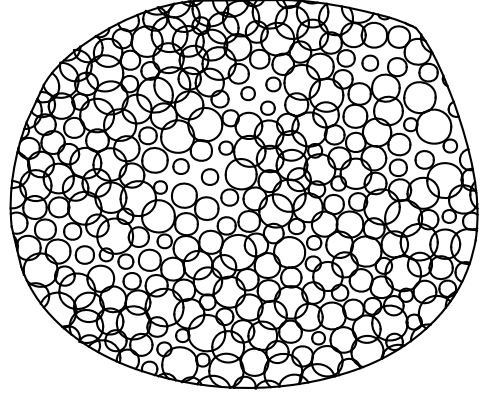


**Teapot**

22 sec

```

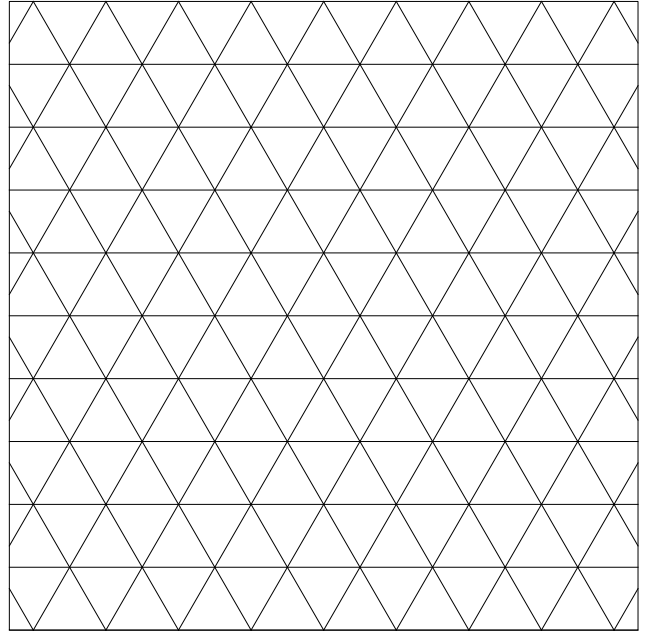
1 def teapot():
2     density = 3.0e-4
3     circle = ImportSVG("data/circle.svg")
4
5     # Mapper: place a circle in each face
6     def face_to_circle(face):
7         src_p = BBoxCenter(circle)
8         dst_p = Centroid(face)
9         return Scale(MatchPoint(circle, src_p, dst_p), Random(face
10                        ,0.05,0.15,0))
11
12     # Uniform partition with given density
13     props = IrregularProperties(density)
14     part = UniformPartition(props, KEEP_OUTSIDE)
15     # Apply the mapper
16     tex3 = MapToFaces(face_to_circle, part)
17     # Fill the outline
18     frame = ImportSVG("data/outline_teapot.svg")
19     ExportSVGinDomain(tex3, frame)
    
```


**Triangles**

4 sec

```

1 # Mapper: splits in two a unique face from a grid partition
2 def split(face):
3     are_incident = lambda e,f: f == LeftFace(e) or f ==
4         RightFace(e)
5     v1 = 0
6     v2 = 0
7     for v in IncidentVertices(face):
8         score_source = 0
9         score_target = 0
10        for e in IncidentEdges(v):
11            if v == SourceVertex(e) and are_incident(e, face):
12                score_source += 1
13            if v == TargetVertex(e) and are_incident(e, face):
14                score_target += 1
15        if score_source == 2:
16            v1 = v
17        if score_target == 2:
18            v2 = v
19    if v1 == 0:
20        return Nothing()
21    seg = ImportSVG("segment.svg")
22    start_seg = PointLabeled(seg, "start")
23    end_seg = PointLabeled(seg, "end")
24    return MatchPoints(seg, start_seg, end_seg, Location(v1),
25                       Location(v2))
26
27 def example_triangles():
28     # Grid partition
29     props1 = StripesProperties(pi/3.0, 200)
30     props2 = StripesProperties(-pi/3.0, 200)
31     grid = GridPartition(props1, props2, KEEP_OUTSIDE)
32     # Apply mapper and merge the result with the original grid
33     triangles = Union(grid, MapToFaces(split, grid))
34     # Export final texture
35     ExportSVG(triangles, 2000)
    
```



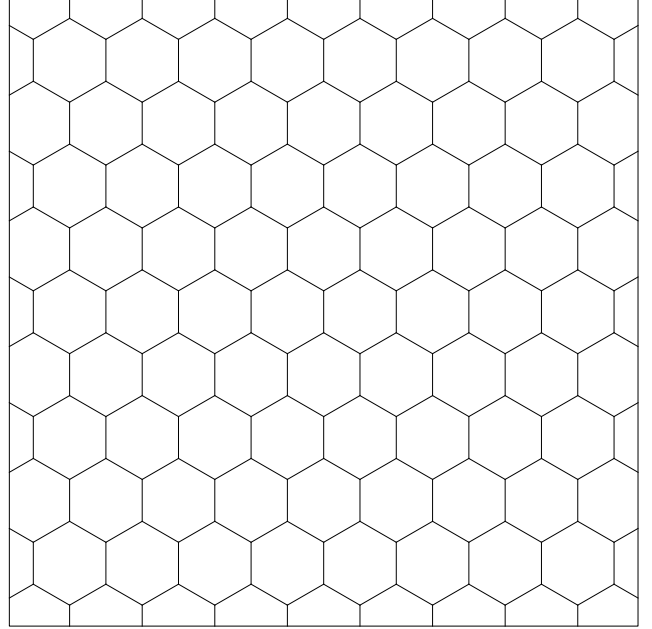
**Bee hive**

5 sec

```

1  # Mapper: returns a segment linking the centroids of the faces
   incident to e
2  def dual(e):
3      if e.is_outside():
4          return Nothing()
5      seg = ImportSVG("segment.svg")
6      start_seg = PointLabeled(seg, "start")
7      end_seg = PointLabeled(seg, "end")
8      return MatchPoints(seg, start_seg, end_seg, Centroid(
9          LeftFace(e)), Centroid(RightFace(e)))
10 def example_hexa():
11     # Grid partition
12     props1 = StripesProperties(pi/3.0, 200)
13     props2 = StripesProperties(-pi/3.0, 200)
14     grid = GridPartition(props1, props2, KEEP_OUTSIDE)
15     # Split faces (from example "Triangles")
16     triangles = Union(grid, MapToFaces(split, grid))
17     # Take the duals of all edges
18     hexa = MapToEdges(dual, triangles)
19     # Export final texture
20     ExportSVG(hexa, 2000)

```

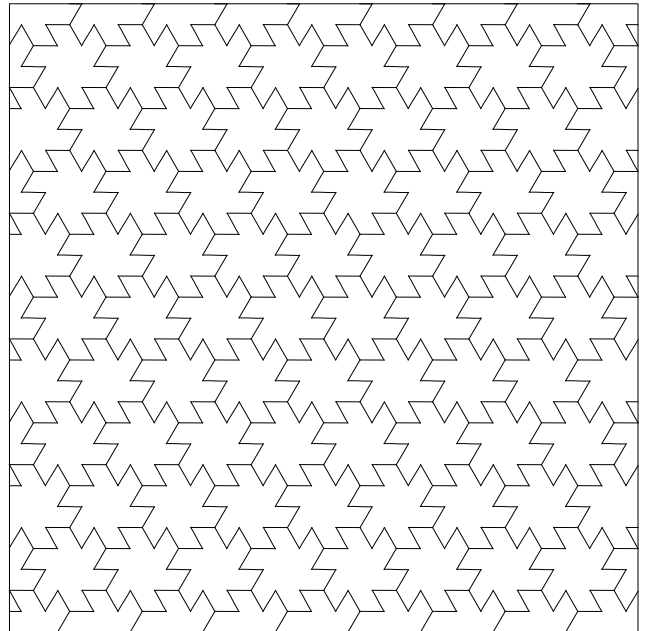
**Wallpaper group P6**

48 sec

```

1  def example_p6():
2      # Import SVG element
3      line = ImportSVG("data/p6_elt.svg")
4      # Mapper: replaces an edge by the imported SVG element
5      def line_to_curve(edge):
6          src_c = PointLabeled(line, "start")
7          dst_c = PointLabeled(line, "end")
8          src_v = Location(SourceVertex(edge))
9          dst_v = Location(TargetVertex(edge))
10         return MatchPoints(line, src_c, dst_c, src_v, dst_v)
11     props1 = StripesProperties(pi/3.0, 200)
12     props2 = StripesProperties(-pi/3.0, 200)
13     grid = GridPartition(props1, props2, KEEP_OUTSIDE)
14     # Split faces (from example "Triangles")
15     triangles = Union(grid, MapToFaces(split, grid))
16     # Take duals of all edges (from example "Bee hive")
17     hexa = MapToEdges(dual, triangles)
18     # Replace edges by Z-shaped curves
19     p6 = MapToEdges(line_to_curve, hexa)
20     # Export final texture
21     ExportSVG(p6, 2000)

```

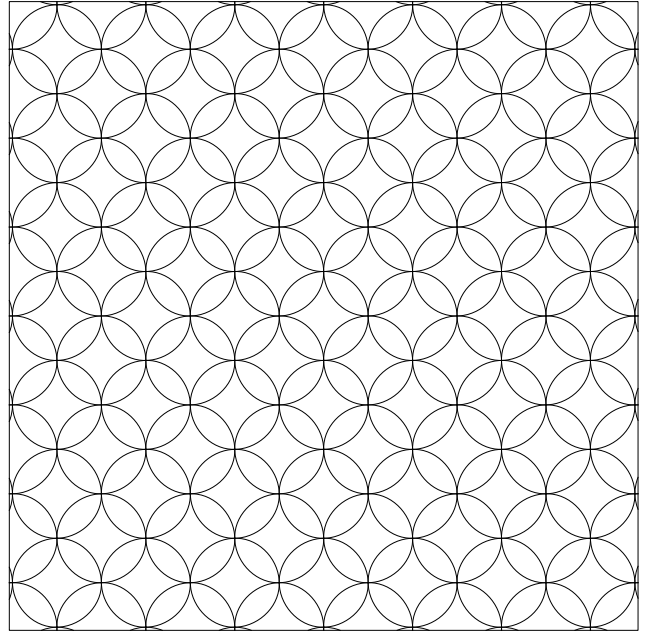


**Wallpaper group P4M**

24 sec

```

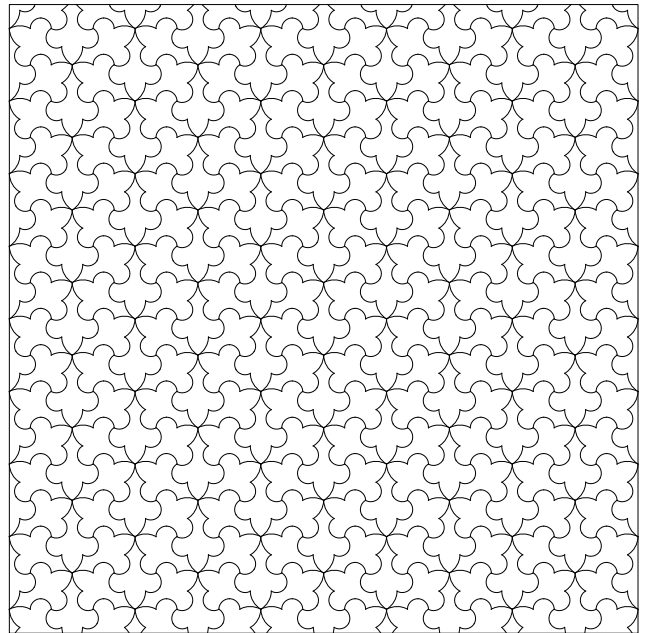
1 def example_p4m():
2     # Import SVG element
3     line = ImportSVG("data/p4m_elt.svg")
4     # Mapper: replaces an edge by the imported SVG element
5     def line_to_curve(edge):
6         src_c = PointLabeled(line, "start")
7         dst_c = PointLabeled(line, "end")
8         src_v = Location(SourceVertex(edge))
9         dst_v = Location(TargetVertex(edge))
10        return MatchPoints(line, src_c, dst_c, src_v, dst_v)
11    # Mapper: replaces an edge by a rotation of the imported
12    # SVG element
13    def line_to_curve_inv(edge):
14        dst_c = PointLabeled(line, "start")
15        src_c = PointLabeled(line, "end")
16        src_v = Location(SourceVertex(edge))
17        dst_v = Location(TargetVertex(edge))
18        return MatchPoints(line, src_c, dst_c, src_v, dst_v)
19    # Grid partition
20    props1 = StripesProperties(pi/4.0, 200)
21    props2 = StripesProperties(-pi/4.0, 200)
22    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
23    # Take the results of both mappers
24    p4m = Union(MapToEdges(line_to_curve, grid), MapToEdges(
25        line_to_curve_inv, grid))
26    # Export final texture
27    ExportSVG(p4m, 2000)
    
```


**Wallpaper group P31M**

50 sec

```

1 def example_p31m():
2     line0 = ImportSVG("data/p31m_elt.svg")
3     line0_sym = ImportSVG("data/p31m_elt_sym.svg")
4     # Mapper: subdivides a triangle face in three pseudo-
5     # triangles
6     def subdivide(face):
7         out = Nothing()
8         line = line0
9         nb_left = 0
10        for e in IncidentEdges(face):
11            if face == LeftFace(e):
12                nb_left = nb_left + 1
13        if nb_left > 1:
14            line = line0_sym
15            src_c = PointLabeled(line, "start")
16            dst_c = PointLabeled(line, "end")
17            dst_v = Centroid(face)
18            for v in IncidentVertices(face):
19                src_v = Location(v)
20                curve = MatchPoints(line, src_c, dst_c, src_v, dst_v)
21                out = Append(out, curve)
22        return out
23    # Grid partition
24    props1 = StripesProperties(pi/6.0, 200)
25    props2 = StripesProperties(pi-pi/6.0, 200)
26    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
27    # Split faces (from example "Triangles")
28    triangles = Union(grid, MapToFaces(split, grid))
29    # Subdivide triangle faces
30    p31m = MapToFaces(subdivide, triangles)
31    # Export final texture
32    ExportSVG(p31m, 2000)
    
```





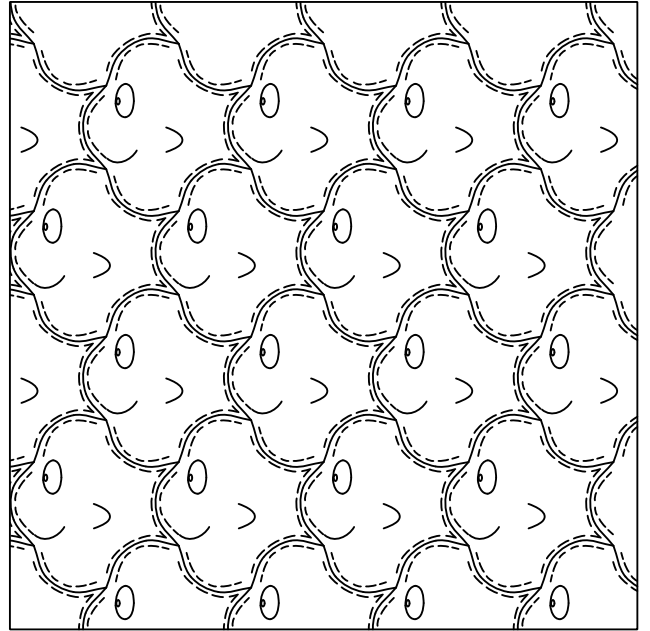
## Escher P6

38 sec

```

1 def example_p6_escher():
2     # Imported SVG elements
3     line = ImportSVG("data/p6_elt_complex3.svg")
4     center_elt = ImportSVG("data/p6_elt_complex3_center.svg")
5     # Mapper: replaces an edge by a piece of a fish outline
6     def line_to_curve(edge):
7         src_c = PointLabeled(line, "start")
8         dst_c = PointLabeled(line, "end")
9         src_v = Location(SourceVertex(edge))
10        dst_v = Location(TargetVertex(edge))
11        return MatchPoints(line, src_c, dst_c, src_v, dst_v)
12    # Mapper: places a fish interior inside a face
13    def face_to_center_elt(f):
14        centers_center = PointLabeled(center_elt, "end") + Point
15        (20, 0)
16        return MatchPoint(center_elt, centers_center, Centroid(
17        f))
18    # Grid partition
19    props1 = StripesProperties(pi/3.0, 200)
20    props2 = StripesProperties(-pi/3.0, 200)
21    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
22    # Split faces (from example "Triangles")
23    triangles = Union(grid, MapToFaces(split, grid))
24    # Take duals of all edges (from example "Bee hive")
25    hexa = MapToEdges(dual, triangles)
26    # Replace edges by fish outlines
27    p6 = MapToEdges(line_to_curve, hexa)
28    # Place fish interiors
29    p6_center = MapToFaces(face_to_center_elt, hexa)
30    # Export final texture
31    final = Union(p6, p6_center)
32    ExportSVG(final, 1000)

```



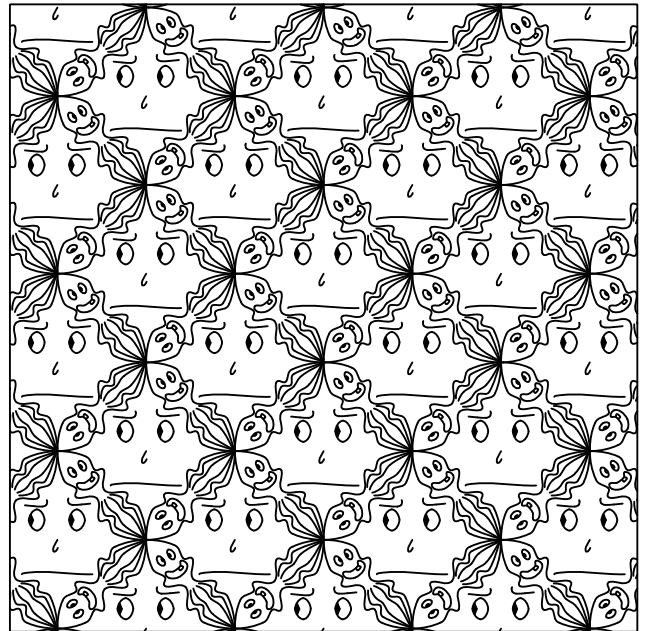
## Escher P4M

43 sec

```

1 def example_p4m_escher():
2     # Imported SVG elements
3     octopus = ImportSVG("data/p4m_elt_complex1.svg")
4     guy = ImportSVG("data/p4m_elt_complex2.svg")
5     # Mapper: places an octopus on an edge
6     def line_to_octopus(edge):
7         src_c = PointLabeled(octopus, "start")
8         dst_c = PointLabeled(octopus, "end")
9         src_v = Location(SourceVertex(edge))
10        dst_v = Location(TargetVertex(edge))
11        return MatchPoints(octopus, src_c, dst_c, src_v, dst_v)
12    # Mapper: places a guy on a face's center
13    def face_to_guy(f):
14        center_guy = PointLabeled(guy, "start")
15        c = Centroid(f)
16        return MatchPoint(guy, center_guy, c)
17    # Grid partition
18    props1 = StripesProperties(pi/4.0, 200)
19    props2 = StripesProperties(-pi/4.0, 200)
20    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
21    # Merge octopuses and guys
22    p4m = Union(MapToEdges(line_to_octopus, grid), MapToFaces(
23    face_to_guy, grid))
24    # Export final texture
25    ExportSVG(p4m, 1000)

```



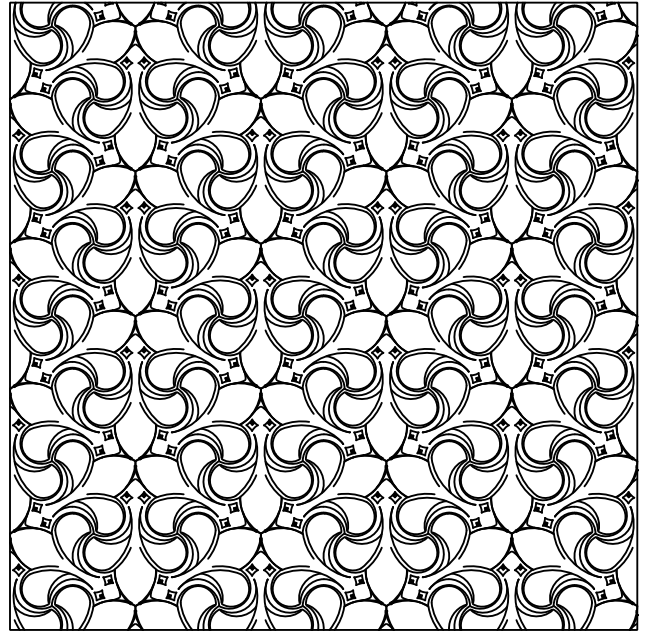


**Escher P31M**

35 sec

```

1 def example_p31m_escher():
2     # Imported SVG elements
3     line0 = ImportSVG("data/p31m_elt_complex.svg")
4     line0_sym = ImportSVG("data/p31m_elt_complex_sym.svg")
5     # Mapper: subdivides each triangle face into three pseudo-
6     # triangles
7     def subdivide(face):
8         out = Nothing()
9         line = line0
10        nb_left = 0
11        for e in IncidentEdges(face):
12            if face == LeftFace(e):
13                nb_left = nb_left + 1
14            if nb_left > 1:
15                line = line0_sym
16            src_c = PointLabeled(line, "start")
17            dst_c = PointLabeled(line, "end")
18            dst_v = Centroid(face)
19            for v in IncidentVertices(face):
20                src_v = Location(v)
21                curve = MatchPoints(line, src_c, dst_c, src_v, dst_v)
22                out = Append(out, curve)
23        return out
24    # Grid partition
25    props1 = StripesProperties(pi/6.0, 200)
26    props2 = StripesProperties(pi-pi/6.0, 200)
27    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
28    # Split faces (from example "Triangles")
29    triangles = Union(grid, MapToFaces(split, grid))
30    # Subdivide resulting triangle faces
31    p31m = MapToFaces(subdivide, triangles)
32    # Export final texture
33    ExportSVG(p31m, 1000)
    
```



```

1 def processor():
2     # Global parameters
3     size = 2000
4     step = 50
5
6     # ----- Level 1 -----
7     densities = [0.002, 0.00175, 0.00125, 0.00225, 0.001]
8     # Paths and density indexes
9     elts_densities = [{"net1.svg", 0}, {"net2.svg", 0}, {"net3.svg",
10         1}, {"net4.svg", 0}, {"net5.svg", 1}, {"net6.svg", 1}, {"net7.svg", 2}, {"net8.svg", 1}, {"net9.svg", 0}, {"net10.svg",
11         3}, {"net11.svg", 3}, {"net12.svg", 4}]
12
13     # Higher order function: generates a mapper for 'elt'
14     def mapper(elt):
15         def new_face_mapper(face):
16             pt = Centroid(face)
17             # Clamp position for regular silicium look
18             clamp = lambda x: step * floor(x/step)
19             pt = Point(clamp(pt.x()), clamp(pt.y()))
20             return MatchPoint(elt, PointLabeled(elt, "start"),
21                 pt)
22         return new_face_mapper
23
24     # dst: distributes randomly elt 'e' with density 'd'
25     dst = lambda e, d: MapToFaces(mapper(e), RandomPartition(
26         IrregularProperties(d), KEEP_OUTSIDE))
27
28     # Distribute all elements
29     levell = lambda f: Nothing()
30     for e_d in elts_densities:
31         d = densities[e_d[1]]
32         levell = Merge(levell, dst(ImportSVG(e_d[0]), d))
33
34     # ----- Level 2 -----
35     ships = ["ship1.svg", "ship2.svg", "ship3.svg", "ship4.svg", "ship5.svg", "ship6.svg"]
36
37     # Mapper: generates and places a ship on the face
38     def face_to_ship(face):
39         ship = ImportSVG(ships[floor(Random(face, 0, 6, 0))])
40         ship = Scale(ship, floor(Random(face, 0, 4, 0)))
41         pt = Centroid(face)
42         # Clamp position for regular silicium look
43         clamp = lambda x: step * floor(x/step)
44         pt = Point(clamp(pt.x()), clamp(pt.y()))
45         res = MatchPoint(ship, BBoxCenter(ship), pt)
46         return res
47
48     props = IrregularProperties(8/(2000*2000))
49     part_ships = UniformPartition(props, KEEP_OUTSIDE)
50     level2 = MapToFaces(face_to_ship, part_ships)
51
52     # ----- Final texture -----
53     c = ImportSVG("data/connector.svg")
54     j = ImportSVG("data/joint.svg")
55
56     # Mapper: generates connectors and soldered joints
57     def make_cj(v):
58         # Connectors at contacts between levell and level2
59         if len(IncidentEdges(v)) == 3 and HasLabel(
60             IncidentFaces(v), "ship"):
61             return MatchPoint(c, BBoxCenter(c), Location(v))
62         # Generate joints at junction ends
63         elif len(IncidentEdges(v)) == 1:
64             return MatchPoint(j, BBoxCenter(j), Location(v))
65         return Nothing()
66
67     lvl1_2 = Outside(levell, level2, CROP_ADD_BOUNDARY)
68     cjs = MapToVertices(make_cj, lvl1_2)
69     final_tex = Outside(lvl1_2, cjs, CROP_ADD_BOUNDARY)
70     ExportSVG(final_tex, size)

```

