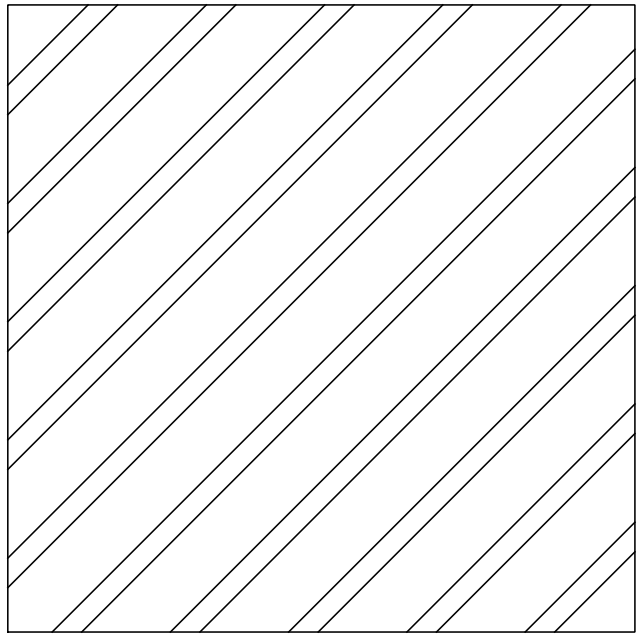
Programmable 2D Arrangements for Element Texture Design

HUGO LOI
Inria-LJK (UGA, CNRS)
and
THOMAS HURTUT
École Polytechnique de Montréal
and
ROMAIN VERGNE and JOELLE THOLLOT
Inria-LJK (UGA, CNRS)

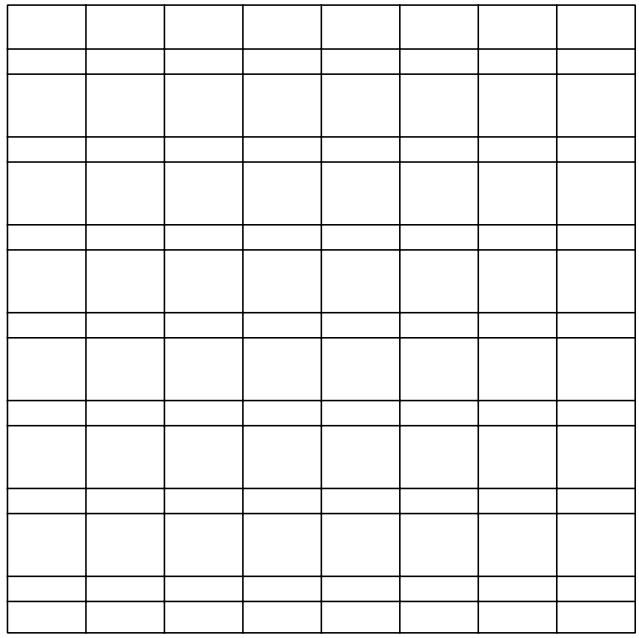
Stripes <1 sec

```
def stripes():
    # Texture size
    size   = 2000

    # Stripes angle=pi/4, widths=200 and 67
    lines = StripesProperties(pi/4,size/10,size/30)

    # Partition
    tex    = StripesPartition(lines)

    # Export result
    ExportSVG(tex,size)
```



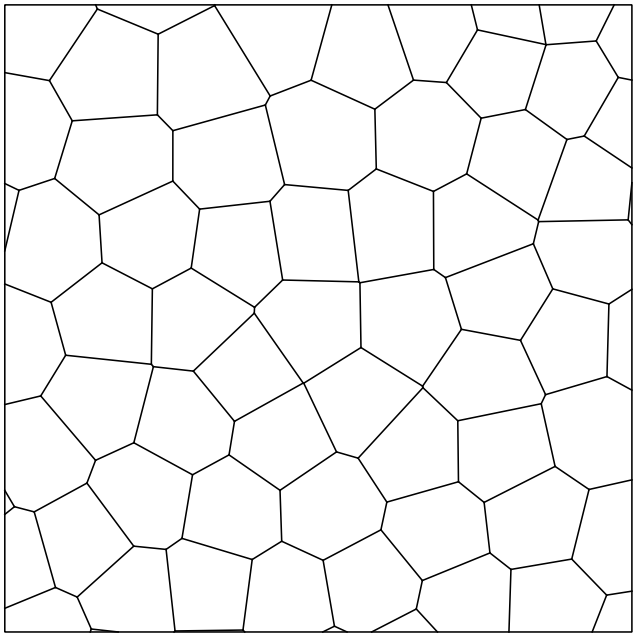
Grid <1 sec

```
def grid():
    # Texture size
    size    = 2000

    # Horizontal stripes angle=0, widths=200 and 80
    lines1 = StripesProperties(0,size/10,size/25)

    # Vertical stripes angle=pi/2, width=250
    lines2 = StripesProperties(pi/2,size/8)

    # Partition
    tex     = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Export result
    ExportSVG(tex,size)
```



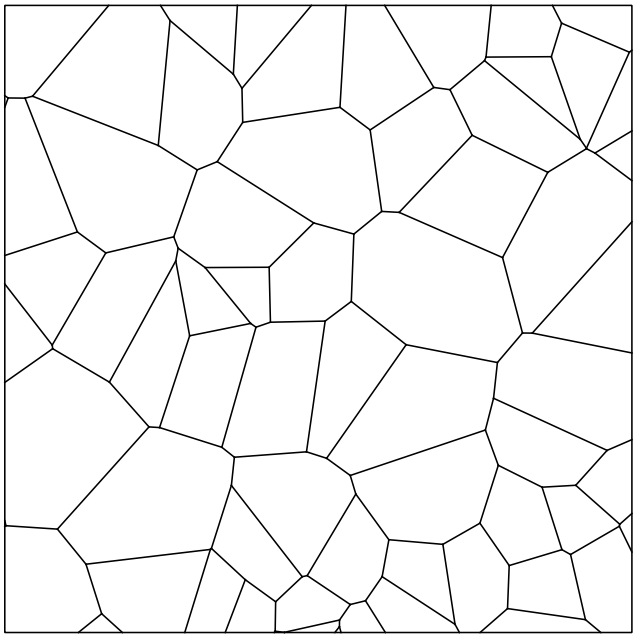
Uniform 2 sec

```
def uniform():
    # Texture size
    size = 2000

    # Density ~= 50 faces
    prop = IrregularProperties(50/(size*size))

    # Partition
    tex  = UniformPartition(prop,KEEP_OUTSIDE)

    # Export result
    ExportSVG(tex,size)
```



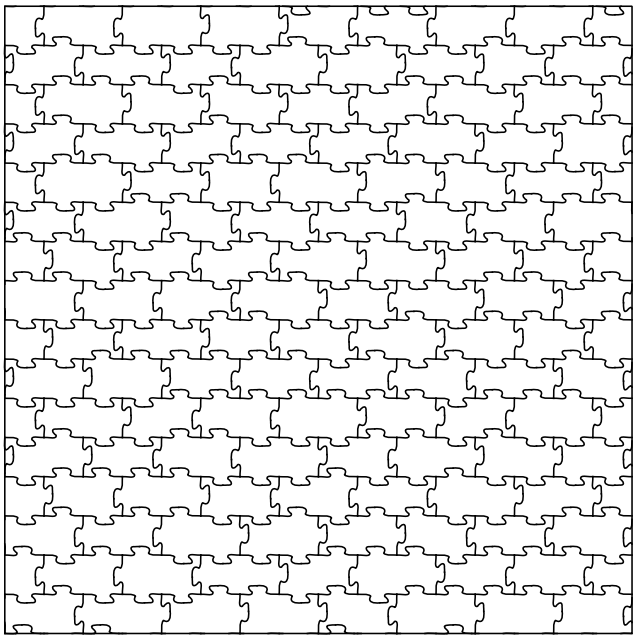
Random <1 sec

```
def random():
    # Texture size
    size = 2000

    # Density ~= 50 faces
    prop = IrregularProperties(50/(size*size))

    # Partition
    tex  = RandomPartition(prop,KEEP_OUTSIDE)

    # Export result
    ExportSVG(tex,size)
```



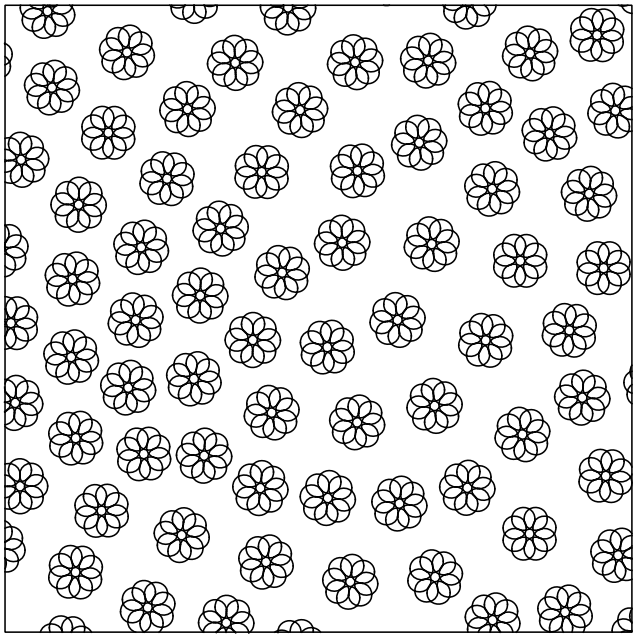
Puzzle 18 sec

```
def puzzle():
    size    = 2000
    zig     = ImportSVG("data/zig2.svg")

    # Grid partition, including edge labels
    lines1 = StripesProperties(0.0,size/16)
    lines2 = StripesProperties(pi/2.0,size/16)
    SetEdgeLabels(lines1,"h1","h2")
    SetEdgeLabels(lines2,"v1","v2")
    grid_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapper: remove edges
    def grid_to_wall(edge):
        if((HasLabel(edge,"v1") and HasLabel(IncidentEdges(
            TargetVertex(edge)),"h1")) or
           (HasLabel(edge,"v2") and HasLabel(IncidentEdges(
               TargetVertex(edge)),"h2"))):
            return Nothing()
        return ToCurve(edge)

    # Mapper: replace each edge by a curved line
    def edge_to_curve(edge):
        src_c = PointLabeled(zig,"start")
        dst_c = PointLabeled(zig,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        if Random(edge,0,1,0)<0.5:
            return MatchPoints(zig,src_c,dst_c,src_v,dst_v)
        else:
            return MatchPoints(zig,src_c,dst_c,dst_v,src_v)

    # Mapping operators
    wall_tex   = MapToEdges(grid_to_wall,grid_tex)
    puzzle_tex = MapToEdges(edge_to_curve,wall_tex)

    # Final texture
    ExportSVG(puzzle_tex,size)
```



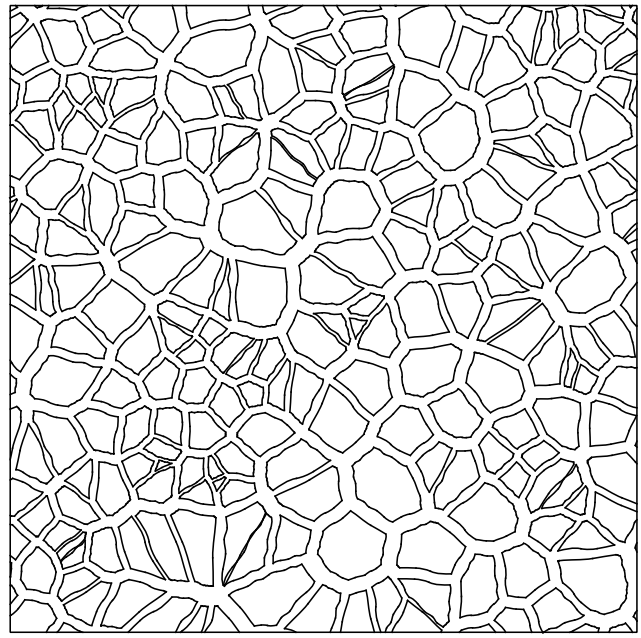
Rosettes 36 sec

```
def rosette():
    size     = 2000
    line     = ImportSVG("data/line.svg")
    circle   = ImportSVG("data/circle.svg")

    # Uniform partition (around 70 faces)
    props    = IrregularProperties(70/(size*size))
    part_tex = UniformPartition(props,KEEP_OUTSIDE)

    # Mapper: create a star in each face
    def face_to_star(face):
        rot   = Random(face,0.0,2.0*pi,0)
        elem1 = MatchPoint(line,BBoxCenter(line),Centroid(face))
        elem2 = Append(Rotate(elem1,rot),Rotate(elem1,rot+pi/2))
        elem3 = Append(elem2,Rotate(elem2,pi/4))
        return elem3

    # Mapper: place a circle on each vertex
    def vertex_to_circle(vertex):
        if(len(IncidentEdges(vertex))>1):
            return Nothing()
        return MatchPoint(circle,BBoxCenter(circle),Location(
            vertex))

    # Mapping operators
    stars_tex    = MapToFaces(face_to_star,part_tex)
    flowers_tex  = MapToVertices(vertex_to_circle,stars_tex)

    # Final texture
    ExportSVG(flowers_tex,size)
```



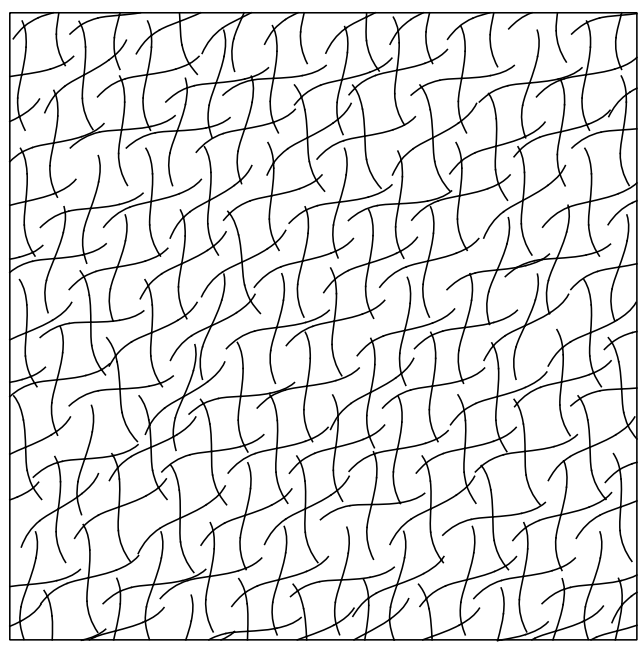
Cracks 35 sec

```
def cracks():
    size     = 2000
    lines    = [ImportSVG("data/line1.svg"),
                ImportSVG("data/line2.svg"),
                ImportSVG("data/line3.svg")]

    # Random partition (around 200 faces)
    props    = IrregularProperties(200/(size*size))
    part_tex = RandomPartition(props,KEEP_OUTSIDE)

    # Mapper: rescale each face
    def rescale_face(face):
        return Scale(Contour(face),0.8)

    # Mapper: replace each edge by a random curved line
    def edge_to_curve(edge):
        line  = lines[floor(Random(edge,0,len(lines),0))]
        src_c = PointLabeled(line,"start")
        dst_c = PointLabeled(line,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(line,src_c,dst_c,src_v,dst_v)

    # Mapping operators
    reduced_tex = MapToFaces(rescale_face,part_tex)
    cracks_tex  = MapToEdges(edge_to_curve,reduced_tex)

    # Final texture
    ExportSVG(cracks_tex,size)
```



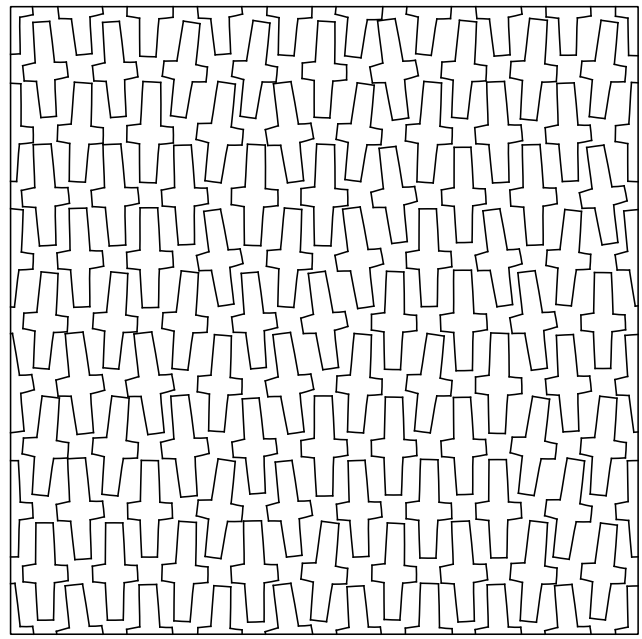
By example limitations: bimodal hatching 7 sec

```
def examplar_based_a():
    size    = 2000
    hatch   = ImportSVG("data/hatch.svg")

    # Grid partition, including face labels
    lines1 = StripesProperties(0,size/10)
    lines2 = StripesProperties(pi/2,size/20)
    SetFaceLabels(lines1,"h1","h2")
    SetFaceLabels(lines2,"v1","v2")
    grid_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapper: place 2 hatches in one face on two
    def face_to_hatches(face):
        if ((HasLabel(face,"v1") and HasLabel(face,"h1")) or
            (HasLabel(face,"v2") and HasLabel(face,"h2"))):
            elem1 = Scale(hatch,Random(face,2.6,2.7,1))
            elem2 = Rotate(elem1,pi/7+Random(face,-pi/12,pi/12,2))
            elem3 = Rotate(elem1,pi/2+Random(face,-pi/12,pi/12,3))
            elem4 = Append(elem2,elem3)
            w     = BBoxWidth(face)/7
            h     = BBoxHeight(face)/7
            pos   = Centroid(face)+Point(Random(face,-w,w,4),
                    Random(face,-h,h,5))
            return MatchPoint(elem4,BBoxCenter(elem4),pos)
        return Nothing()

    # Mapping operator
    hatch_tex = MapToFaces(face_to_hatches,grid_tex)

    # Final texture
    ExportSVG(hatch_tex,size)
```



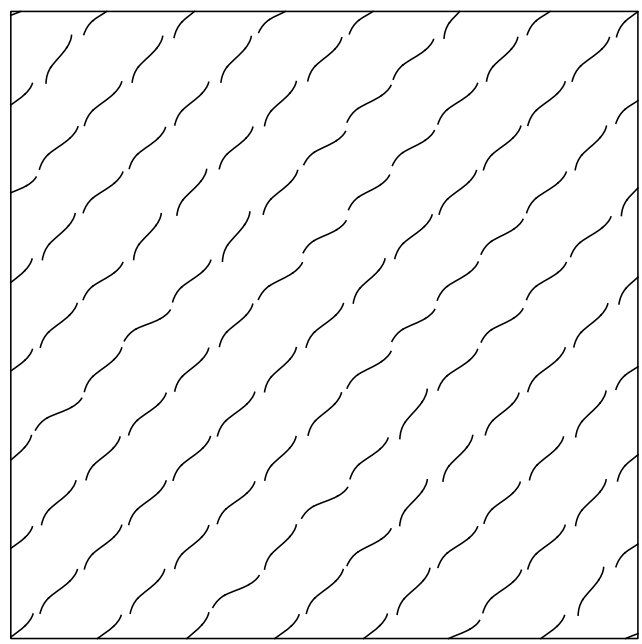
By example limitations: regular crosses 6 sec

```
def examplar_based_b():
    size    = 2000
    cross   = ImportSVG("data/cross.svg")

    # Grid partition, including face labels
    lines1 = StripesProperties(0,size/10)
    lines2 = StripesProperties(pi/2,size/18)
    SetFaceLabels(lines1,"h1","h2")
    SetFaceLabels(lines2,"v1","v2")
    grid_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapper: place a cross in one face on two
    def face_to_hatches(face):
        if ((HasLabel(face,"v1") and HasLabel(face,"h1")) or
            (HasLabel(face,"v2") and HasLabel(face,"h2"))):
            elem1 = Scale(cross,Random(face,1.35,1.45,2))
            elem2 = Rotate(elem1,Random(face,-pi/20,pi/20,4))
            return MatchPoint(elem2,BBoxCenter(elem2),BBoxCenter(
                face))
        return Nothing()

    # Mapping operator
    hatch_tex = MapToFaces(face_to_hatches,grid_tex)

    # Final texture
    ExportSVG(hatch_tex,size)
```



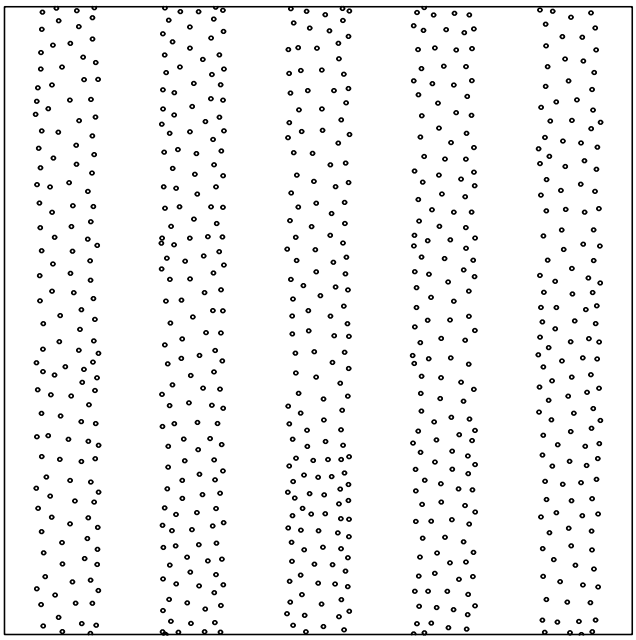
By example limitations: regular curves 4 sec

```
def examplar_based_c():
    size      = 2000
    hatch     = ImportSVG("data/hatch.svg")

    # Grid partition
    lines1    = StripesProperties(pi/4,size/10)
    lines2    = StripesProperties(pi/2+pi/4,size/10)
    grid_tex  = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapper: place a curved line in each face
    def face_to_hatch(face):
        src_p = BBoxCenter(hatch)
        dst_p = Centroid(face)
        elem1 = Scale(hatch,Random(face,1.3,1.4,0))
        elem2 = Rotate(elem1,pi/4+Random(face,-pi/12,pi/12,1))
        return MatchPoint(elem2,src_p,dst_p)

    # Mapping operator
    hatch_tex = MapToFaces(face_to_hatch,grid_tex)

    # Final texture
    ExportSVG(hatch_tex,size)
```



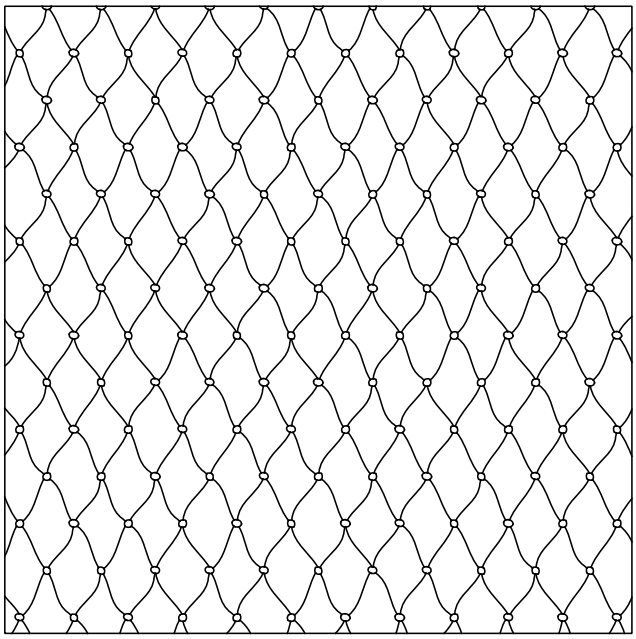
By example limitations: clusters 42 sec

```
def examplar_based_d():
    size   = 2000
    circle = ImportSVG("data/circle.svg")

    # Stripes partition, including face labels
    lines = StripesProperties(pi/2,size/10)
    SetFaceLabels(lines,"v1","v2")
    stripes_tex = StripesPartition(lines)

    # Mapper: place a circle in each face
    def face_to_circle(face):
        return Scale(MatchPoint(circle,BBoxCenter(circle),
            Centroid(face)),0.02)

    # Mapper: creates a dot pattern in one face on two
    def face_to_circles(face):
        if HasLabel(face,"v2"):
            props = IrregularProperties(100/(BBoxWidth(face)*
                BBoxHeight(face)))
            part  = UniformPartition(props,CROP_ADD_BOUNDARY)
            circ  = MapToFaces(face_to_circle,part)
            return circ(face)
        return Nothing()

    # Mapping operator
    texture = MapToFaces(face_to_circles,stripes_tex)

    # Final texture
    ExportSVG(texture,size)
```



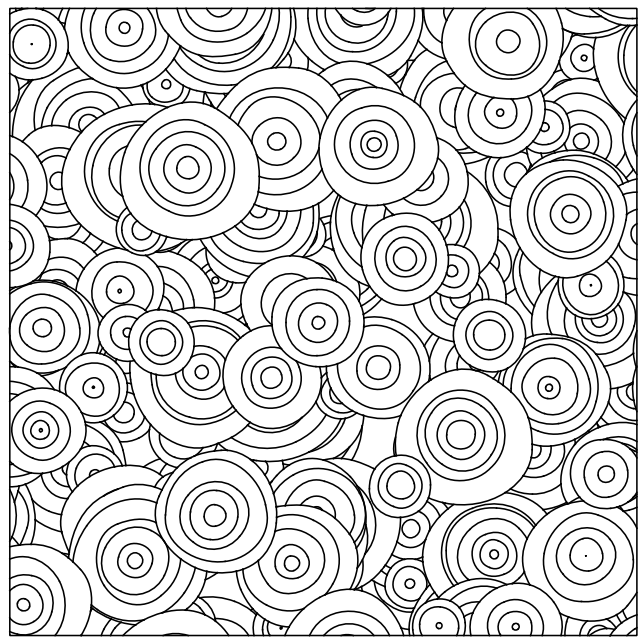
Teaser: example a 25 sec

```
def teaser_a():
    # Load multiple curve and blob shapes
    curves = [ImportSVG("data/curve1.svg"),
              ImportSVG("data/curve2.svg"),
              ImportSVG("data/curve3.svg"),
              ImportSVG("data/curve4.svg")]
    blobs  = [ImportSVG("data/blob1.svg"),
              ImportSVG("data/blob2.svg"),
              ImportSVG("data/blob3.svg"),
              ImportSVG("data/blob4.svg")]
    size = 2000

    # Mapper: replace each edge by a random curve
    def edge_to_curve(edge):
        curve = curves[floor(Random(edge,0,len(curves),0))]
        src_c = PointLabeled(curve,"bottom")
        dst_c = PointLabeled(curve,"top")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(curve,src_c,dst_c,src_v,dst_v)

    # Mapper: replace each vertex by a random blob
    def vertex_to_blob(vertex):
        blob = blobs[floor(Random(vertex,0,len(blobs),0))]
        return MatchPoint(blob,BBoxCenter(blob),Location(vertex))

    # Grid Partition
    lines1    = StripesProperties(pi/3,150)
    lines2    = StripesProperties(-pi/3,150)
    grid_tex  = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapping operators
    curve_tex = MapToEdges(edge_to_curve,grid_tex)
    blob_tex  = MapToVertices(vertex_to_blob,grid_tex)

    # Combining
    final_tex = Union(blob_tex,Outside(curve_tex,blob_tex,CROP))

    # Final texture
    ExportSVG(final_tex,size)
```



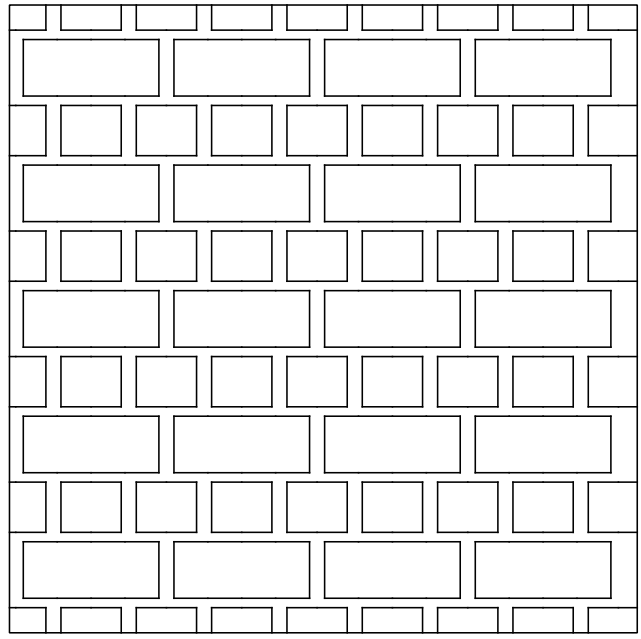
Teaser: example b 2 min, 9 sec

```
# create a texture with non-overlapping concentric circles
def circles_in_circles(density):
    cir = [ImportSVG("data/circle1.svg"),
           ImportSVG("data/circle2.svg"),
           ImportSVG("data/circle3.svg")]

    # Uniform partition
    props  = IrregularProperties(density)
    part   = UniformPartition(props,KEEP_OUTSIDE)

    # Mapper: place circles in each face
    def face_to_circles(face):
        all_circ   = Nothing()
        cur_scale  = Random(face,0.3,0.8,0);

        # Perturb shape and scale for each circle
        i = 0
        while cur_scale>0.0:
            circle     = cir[floor(Random(face,0,len(circles),i))]
            cur_circ   = Scale(MatchFace(circle,face),cur_scale)
            cur_circ   = Rotate(cur_circ,Random(face,0,2*pi,i+10))
            all_circ   = Append(all_circ,cur_circ)
            cur_scale  = cur_scale-Random(face,0.09,0.2,i+20)
            i = i+1

        return all_circ

    return MapToFaces(face_to_circles,part)

def teaser_b():
    size = 2000

    # Create a first layer of circles
    tex = circles_in_circles(15/(size*size))

    # Combine layers
    for i in range(0,9):
        tmp = circles_in_circles(15/(size*size))
        tex = Union(tmp,Outside(tex,tmp,CROP))

    ExportSVG(tex,size)
```



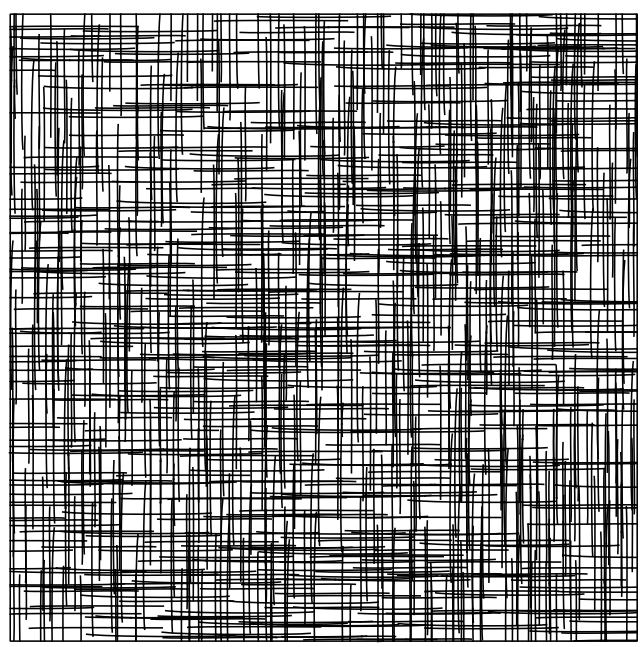
Teaser: example c 1 sec

```
def teaser_c():
    size    = 2000

    # Grid partition, including edge labels
    lines1 = StripesProperties(0,200)
    lines2 = StripesProperties(pi/2,120)
    SetEdgeLabels(lines1,"h1","h2")
    SetEdgeLabels(lines2,"v1","v2","v3","v4")
    grid_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapper: remove edges containing specific label sequences
    def remove_edge(edge):
        if HasLabel(edge,"h1") or HasLabel(edge,"h2"):
            return ToCurve(edge)
        if HasLabel(IncidentEdges(TargetVertex(edge)),"h1"):
            if HasLabel(edge,"v1"):
                return ToCurve(edge)
            else:
                return Nothing()
        else:
            if HasLabel(edge,"v2") or HasLabel(edge,"v4"):
                return ToCurve(edge)
            else:
                return Nothing()

    # Mapper: rescale each face
    def shrink_face(face):
        if HasLabel(IncidentEdges(face),"v1"):
            return Scale(Contour(face),0.9)
        else:
            return Scale(Contour(face),0.8)

    # Mapping operator
    wall_tex    = MapToEdges(remove_edge,grid_tex)
    shrink_tex = MapToFaces(shrink_face,wall_tex)

    # Final texture
    ExportSVG(shrink_tex,size)
```



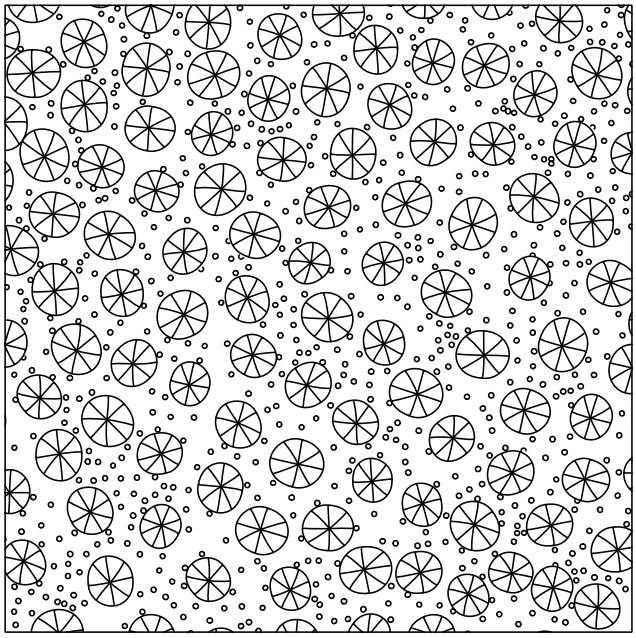
Teaser: example d 3 min, 40 sec

```
def teaser_d():
    # Load several curves
    curves    = [ImportSVG("data/bighatch1.svg"),
                 ImportSVG("data/bighatch2.svg"),
                 ImportSVG("data/bighatch3.svg"),
                 ImportSVG("data/bighatch4.svg")]
    size      = 2000

    # Uniform partition (with around 1000 faces)
    props     = IrregularProperties(1000/(size*size))
    init_tex  = UniformPartition(props,KEEP_OUTSIDE)

    # Mapper: place a vertical or horizontal curve on each face
    def face_to_hatches(face):
        curve = curves[floor(Random(face,0,len(curves),0))]
        if Random(face,0,1,1)>0.5:
            curve = Rotate(curve,pi/2)
        return MatchPoint(curve,BBoxCenter(curve),Centroid(face))

    # Mapping operator
    hatch_tex = MapToFaces(face_to_hatches,init_tex)

    # Final texture
    ExportSVG(hatch_tex,size)
```



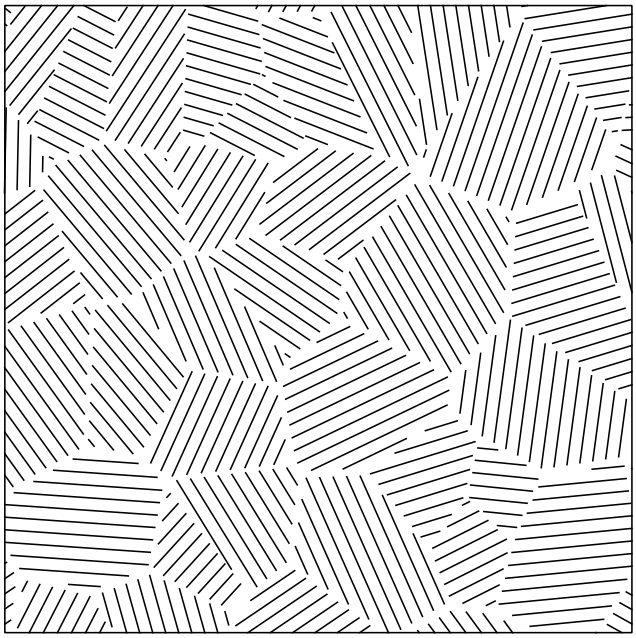
Teaser: example e 20 sec

```
def teaser_e():
    size     = 2000
    wheel    = ImportSVG("data/wheel1.svg")
    stipple  = ImportSVG("data/stipple1.svg")

    # Uniform partition (around 100 faces)
    props1   = IrregularProperties(100/(size*size))
    tex1     = UniformPartition(props1,KEEP_OUTSIDE)

    # Random partition (around 1200 faces)
    props2   = IrregularProperties(1200/(size*size))
    tex2     = RandomPartition(props2,KEEP_OUTSIDE)

    # Mapper: place a wheel in each face
    def face_to_wheel(face):
        w = Scale(Rotate(wheel,Random(face,0,2*pi,0)),Random(face
            ,0.8,1,1))
        return MatchPoint(w,BBoxCenter(w),Centroid(face))

    # Mapper: place a stipple in each face
    def face_to_stipples(face):
        s = Scale(Rotate(stipple,Random(face,0,2*pi,0)),Random(
            face,0.9,1,1))
        return MatchPoint(s,BBoxCenter(s),Centroid(face))

    # Mapping operators
    tex_wheel     = MapToFaces(face_to_wheel,tex1)
    tex_stipples  = MapToFaces(face_to_stipples,tex2)

    # Combining operators
    texture = Union(tex_wheel,Outside(tex_stipples,tex_wheel,
        KEEP_INSIDE))

    # Final texture
    ExportSVG(texture,size)
```



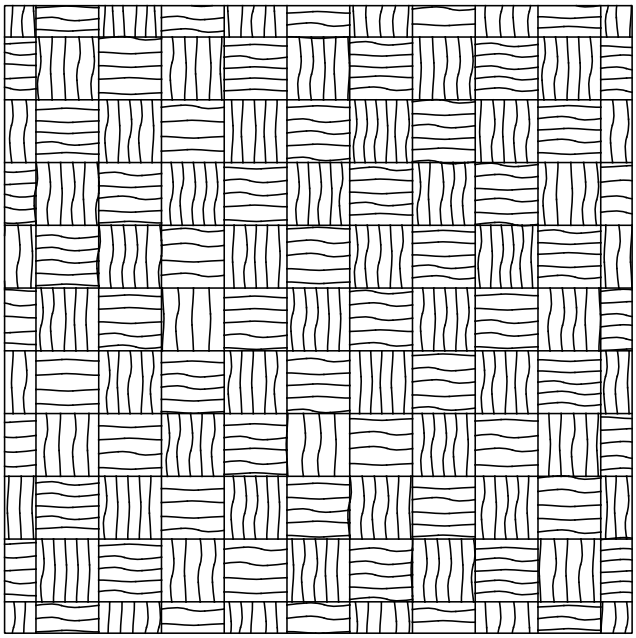
Teaser: example f 3 sec

```
def teaser_f():
    # Random partition (around 30 faces)
    size     = 2000
    props    = IrregularProperties(30/(size*size))
    init_tex = RandomPartition(props,KEEP_OUTSIDE)

    # Mapper: rescale each face
    def scale_map(face):
        return Scale(Contour(face),0.95)

    # Mapper: produces stripes on each face
    def hatch_map(face):
        angle = Random(face,0,2*pi,1)
        lines = StripesProperties(angle,40)
        return StripesPartition(lines)(face)

    # Mapper: remove boundary edges
    def border_map(edge):
        if IsBoundary(edge):
            return Nothing()
        return ToCurve(edge)

    # Mapping operators
    tex1 = MapToFaces(scale_map,init_tex)
    tex2 = MapToFaces(hatch_map,tex1)
    tex3 = MapToEdges(border_map,tex2)

    # Final texture
    ExportSVG(tex3,size)
```



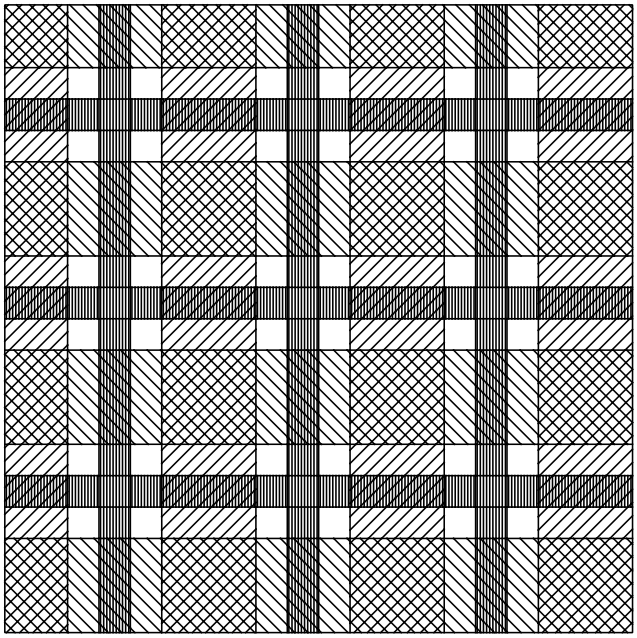
Teaser: example g 6 sec

```
def teaser_g():
    curves = [ImportSVG("data/line1.svg"),
              ImportSVG("data/line2.svg")]

    # Grid partition, including face labels
    size   = 2000
    lines1 = StripesProperties(0,200)
    lines2 = StripesProperties(pi/2,200)
    SetFaceLabels(lines1,"h1","h2")
    SetFaceLabels(lines2,"v1","v2")
    gridl_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapper: replace each edge by a random curve
    def edge_to_curve(edge):
        if IsBoundary(edge):
            return ToCurve(edge)
        curve = curves[floor(Random(edge,0,len(curves),0))]
        src_c = PointLabeled(curve,"start")
        dst_c = PointLabeled(curve,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        if Random(edge,0,1,1)<0.5:
            return MatchPoints(curve,src_c,dst_c,src_v,dst_v)
        else:
            return MatchPoints(curve,dst_c,src_c,src_v,dst_v)

    # Mapper: creates stripes in each face
    def face_to_stripes(face):
        width = BBoxWidth(face)/Random(face,4,6,0)
        theta = 0
        if((HasLabel(face,"h1") and HasLabel(face,"v1")) or
           (HasLabel(face,"h2") and HasLabel(face,"v2"))):
            theta = pi/2
        lines  = StripesProperties(theta,width)
        stripes = StripesPartition(lines)
        return MapToEdges(edge_to_curve,stripes)(face)

    # Mapping operator
    tiled_tex = MapToFaces(face_to_stripes,gridl_tex)

    # Final texture
    ExportSVG(tiled_tex,size)
```



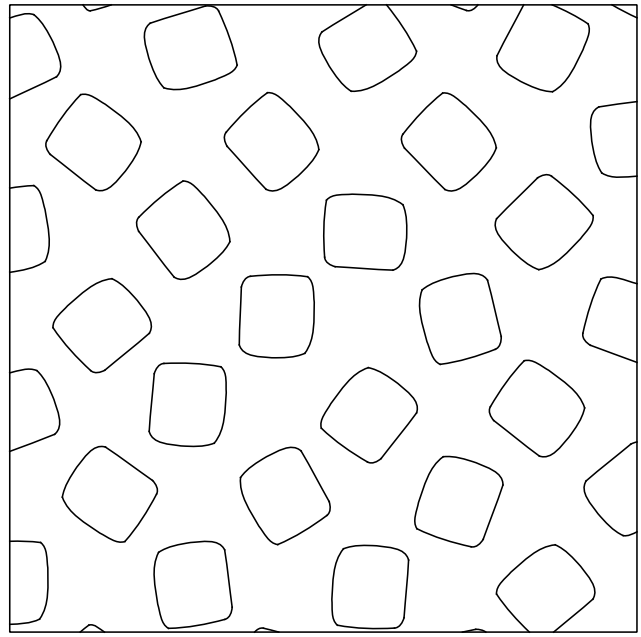
Teaser: example h 26 sec

```
def teaser_h():
    size    = 2000

    # Stripe partition
    lines1 = StripesProperties(0.0,300)
    SetFaceTags(lines1,"h1","h2")
    stripes1 =  StripesPartition(lines1)

    # Stripe partition
    lines2 = StripesProperties(pi/2,300)
    SetFaceLabels(lines2,"v1","v2")
    stripes2 =  StripesPartition(lines2)

    # Grid partition with labels
    lines3 = StripesProperties(0.0,100,100,100,300)
    SetFaceLabels(lines3,"h1","h2","h3","h4")
    lines4 = StripesProperties(pi/2,100,100,100,300)
    SetFaceLabels(lines4,"v1","v2","v3","v4")
    grid = GridPartition(lines3,lines4,KEEP_OUTSIDE)

    # Create a stripe partition with specified angle and width
    def hatch(angle,width):
        lines = StripesProperties(angle,width)
        return StripesPartition(lines)

    # Mapper: create -pi/4 stripes in each face
    def hatch_map_stripes1(f):
        angle = -pi/4
        if HasLabel(f,"h1"):
            return hatch(angle,30)(f)
        else:
            return Nothing()

    # Mapper: create pi/4 stripes in each face
    def hatch_map_stripes2(f):
        angle = pi/4
        if HasLabel(f,"v1"):
            return hatch(angle,30)(f)
        else:
            return Nothing()

    # Mapper: create pi/2 stripes in specific faces
    def hatch_map_grid(f):
        angle = pi/2
        if HasLabel(f,"h1"):
            return hatch(angle,10)(f)
        elif HasLabel(f,"v1"):
            return hatch(angle,10)(f)
        else:
            return Nothing()

    # Mapping operators
    tex1 = MapToFaces(hatch_map_stripes1, stripes1)
    tex2 = MapToFaces(hatch_map_stripes2, stripes2)
    tex4 = MapToFaces(hatch_map_grid,grid)

    # Merging operators
    tex3 = Union(tex1,tex2)
    tex5 = Union(tex3,tex4)

    ExportSVG(tex5,size)
```



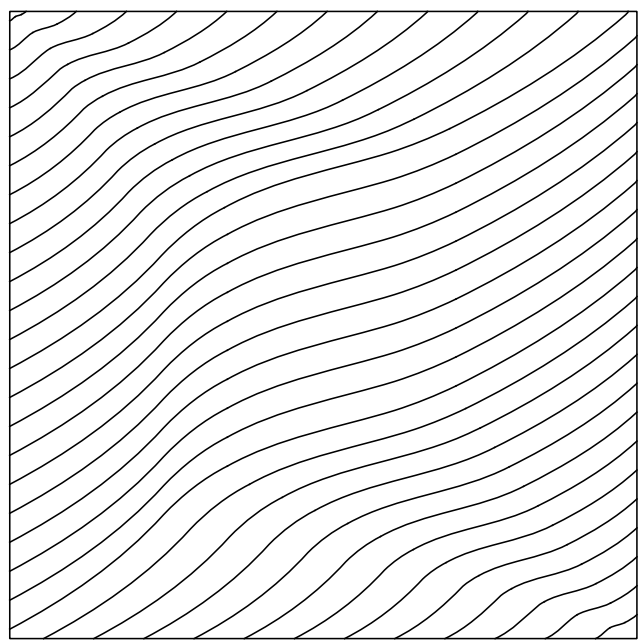
Combining a: texture 1 1 sec

```
def squares(theta,width):
    # Grid partition with given width and orientation
    lines1   = StripesProperties(theta,width)
    lines2   = StripesProperties(theta+pi/2.0,width)
    grid_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)
    square   = ImportSVG("data/square.svg")

    # Mapper: place a rounded square in each face
    def face_to_square(face):
        return Rotate(MatchFace(square,face),Random(face,0.0,2.0*
            pi,1))

    # Return the texture genrated via the mapping operator
    return MapToFaces(face_to_square,grid_tex)

def combine_a():
    tex1 = squares(pi/4,400)
    ExportSVG(tex1,2000)
```



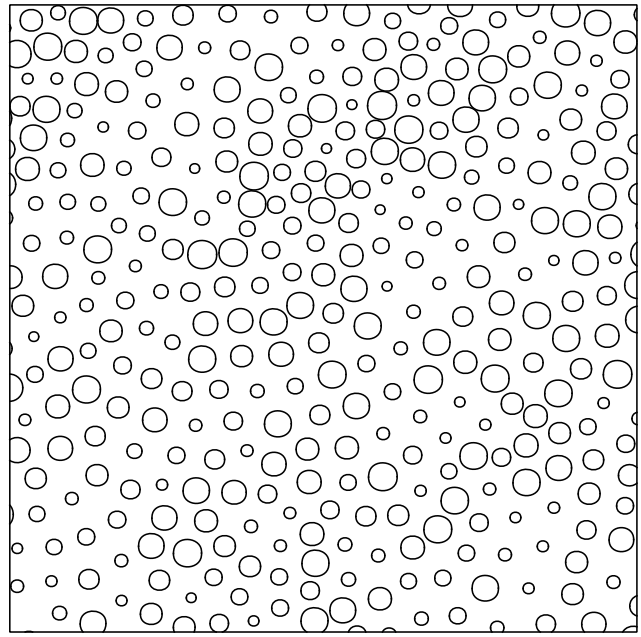
Combining b: texture 2 1 sec

```
def curves(theta,width):
    # Stripes partition with given width and orientation
    props   = StripesProperties(theta,width)
    stripes = StripesPartition(props)
    line    = ImportSVG("data/line6.svg")

    # Mapper: replace each edge by a curve
    def line_to_curve(edge):
        if IsBoundary(edge):
            return Nothing()
        src_c = PointLabeled(line,"start")
        dst_c = PointLabeled(line,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(line,src_c,dst_c,src_v,dst_v)

    # Return the texture genrated via the mapping operator
    return MapToEdges(line_to_curve,stripes)

def combine_b():
    tex2 = curves(pi/6,80)
    ExportSVG(tex2,2000)
```



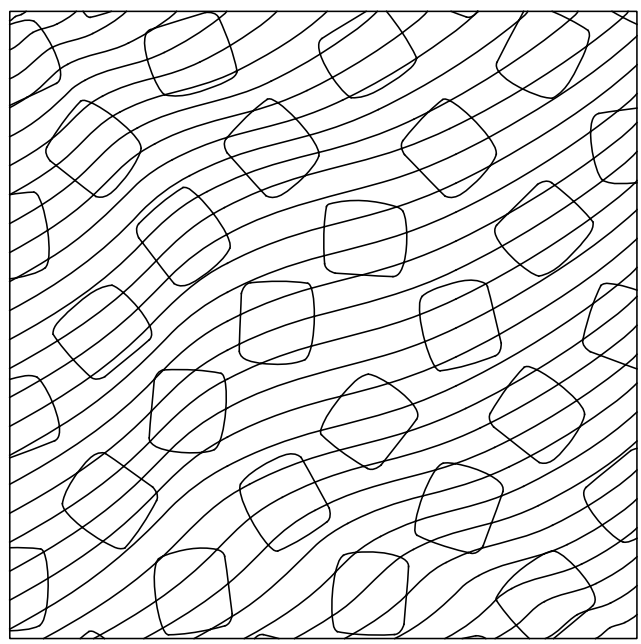
Combining c: texture 3 13 sec

```
def circles(density):
    # Uniform partition with given density
    props  = IrregularProperties(density)
    part   = UniformPartition(props,KEEP_OUTSIDE)
    circle = ImportSVG("data/circle.svg")

    # Mapper: place a circle in each face
    def face_to_circle(face):
        src_p = BBoxCenter(circle)
        dst_p = Centroid(face)
        return Scale(MatchPoint(circle,src_p,dst_p),Random(face
            ,0.05,0.15,0))

    # Mapping operator
    return MapToFaces(face_to_circle,part)

def combine_c():
    tex3 = circles(8.5e-5)
    ExportSVG(tex3,2000)
```



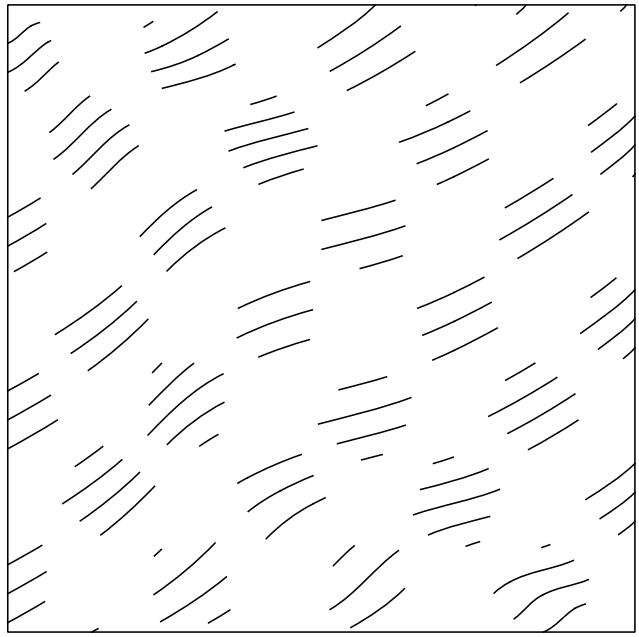
Combining d: merging tex1 and tex2 4 sec

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



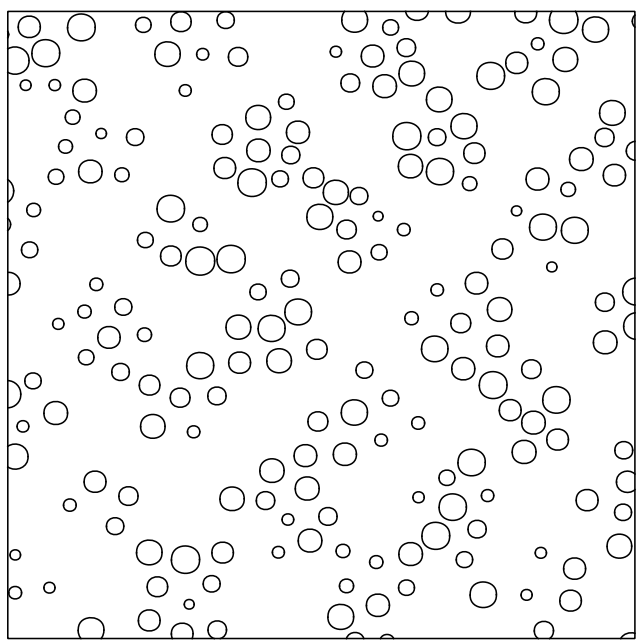
Combining e: tex2 inside tex1 (crop) 3 sec

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



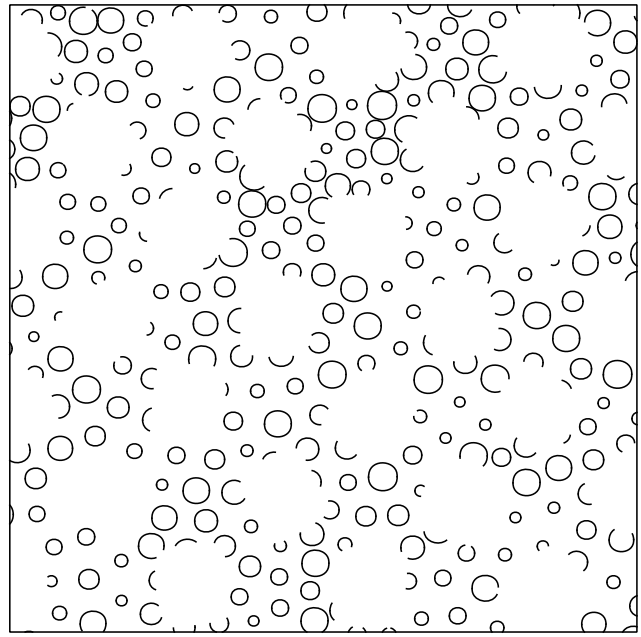
Combining f: tex3 inside tex1 (keep outside) 15 sec

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



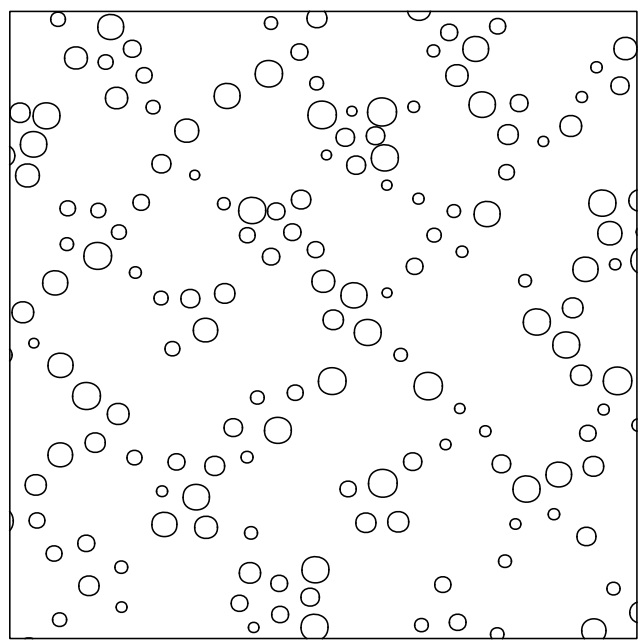
Combining g: tex3 outside tex1 (crop) 15 sec

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



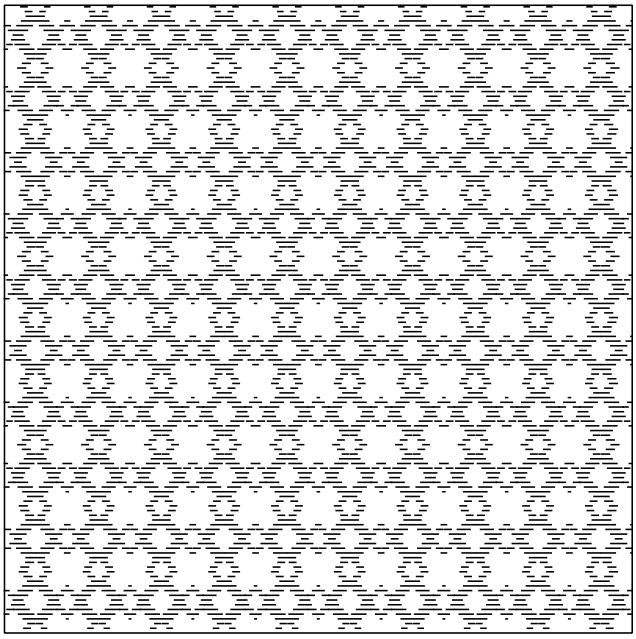
Combining h: tex3 outside tex1 (keep inside) 15 sec

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



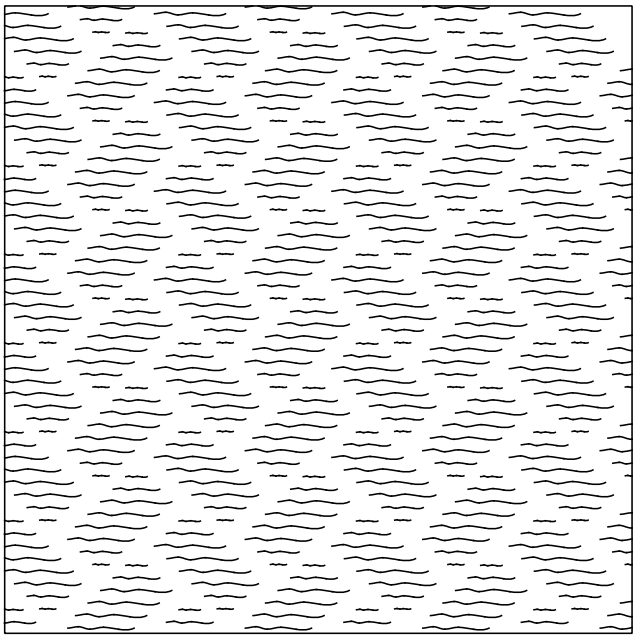
Regular result (a) 30 sec

```
def complex_regular01():
    size    = 2000

    # Grid partition
    lines1 = StripesProperties(0.0,200)
    lines2 = StripesProperties(pi/2.0,200)
    grid   = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Stripe partition
    lines = StripesProperties(0,15)
    hatch = StripesPartition(lines)

    # Mapper: rotate faces with a pi/4 angle
    def rotate_map(f):
        return Rotate(Contour(f), pi/4)

    # Mapper: scale each face with a factor 0.8
    def scale_map(f):
        return Scale(Contour(f), 0.8)

    # Mapping operators (rotate/scale)
    tex1 = MapToFaces(rotate_map, grid)
    tex2 = MapToFaces(scale_map, tex1)

    # Combining operator
    tex3 = Outside(hatch,tex2,CROP)

    ExportSVG(tex3,size)
```



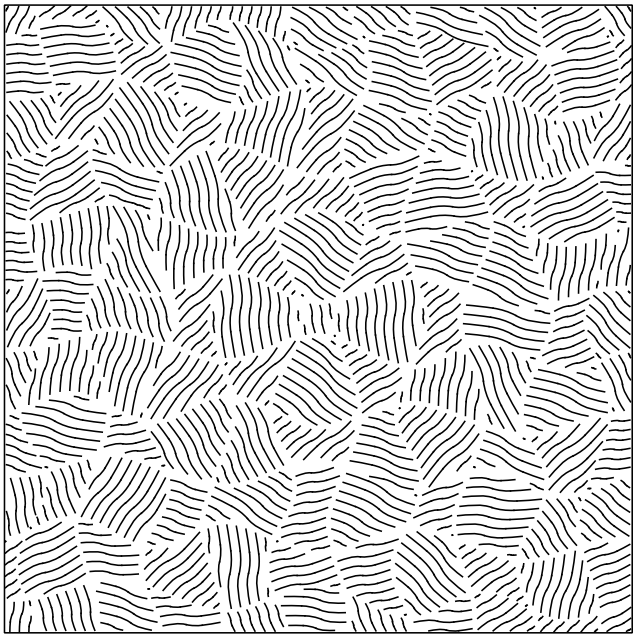
Regular result (b) 39 sec

```
def complex_regular02():
    size = 2000
    line = ImportSVG("data/line3.svg")

    # Grid partition and labels
    lines1 = StripesProperties(pi/4.0,200)
    lines2 = StripesProperties(-pi/4.0,200)
    SetEdgeTags(lines1,"h1","h2","h3","h4")
    SetEdgeTags(lines2,"v1","v2","v3","v4")
    grid = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapper: keep or remove edge
    def grid_to_V_mapper(edge):
        if HasTag(edge,"v1") and HasTag(IncidentEdges(
             TargetVertex(edge)),"h1"):
            return Nothing()
        if HasTag(edge,"v2") and HasTag(IncidentEdges(
             TargetVertex(edge)),"h2"):
            return Nothing()
        if HasTag(edge,"v3") and HasTag(IncidentEdges(
             TargetVertex(edge)),"h3"):
            return Nothing()
        if HasTag(edge,"v4") and HasTag(IncidentEdges(
             TargetVertex(edge)),"h4"):
            return Nothing()
        if HasTag(edge,"h1") and HasTag(IncidentEdges(
             TargetVertex(edge)),"v3"):
            return Nothing()
        if HasTag(edge,"h2") and HasTag(IncidentEdges(
             TargetVertex(edge)),"v4"):
            return Nothing()
        if HasTag(edge,"h3") and HasTag(IncidentEdges(
             TargetVertex(edge)),"v1"):
            return Nothing()
        if HasTag(edge,"h4") and HasTag(IncidentEdges(
             TargetVertex(edge)),"v2"):
            return Nothing()
        return ToCurve(edge)

    # Mapper: rescale face
    def scale_map(f):
        return Scale(Contour(f), 0.8)

    # Mapper: replace each edge by a curve
    def map_curve_to_edge(edge):
       start = PointTagged(line,"start")
       end   = PointTagged(line,"end")
       if IsBoundary(edge):
          return Nothing()
       else:
          return MatchPoints(line,start,end,Location(
               SourceVertex(edge)),Location(TargetVertex(edge)))

    # Create a stripe partition
    def hatch(angle):
       lines = StripesProperties(angle,40)
       return StripesPartition(lines)

    # Mapper: create stripes in each face
    def hatch_map(f):
       angle = 0.0
       return hatch(angle)(f)

    # Mapping operator: select grid edges
    tex1 = MapToEdges(grid_to_V_mapper,grid)

    # Mapping operator: rescale faces
    tex2 = MapToFaces(scale_map, tex1)

    # Mapping operator: place hatches faces
    tex3 = MapToFaces(hatch_map, tex2)

    # Mapping operator: replace hatches by smooth curves
    tex4 = MapToEdges(map_curve_to_edge, tex3)

    ExportSVG(tex4,size)
```



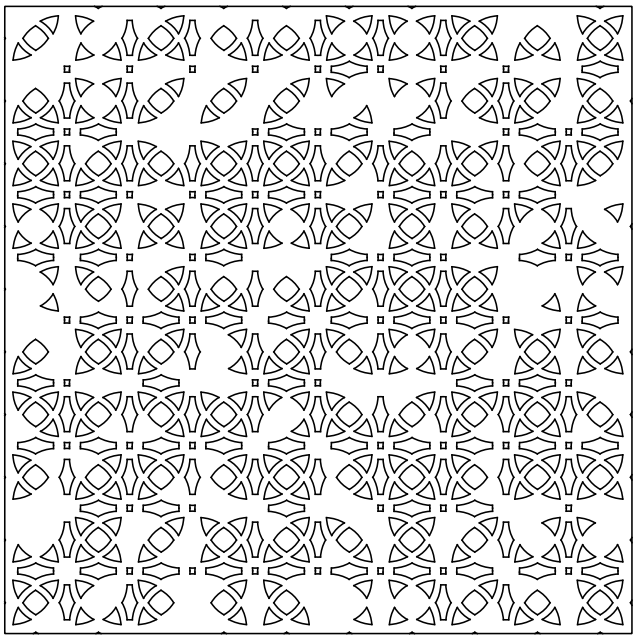
Regular result (c) 30 sec

```
def complex_regular03():
    size   = 2000
    line = ImportSVG("data/line4.svg")
    line2 = ImportSVG("data/line2.svg")

    # Stripe partitions and their labels
    lines1 = StripesProperties(0,200)
    lines2 = StripesProperties(pi/2,200)
    SetEdgeLabels(lines1,"h1","h2")
    SetEdgeLabels(lines2,"h1","h2")
    stripes1 = StripesPartition(lines1)
    stripes2 = StripesPartition(lines2)

    # Mapper: replace each edge by a curve
    def map_curve_to_edge(edge):
        start = PointTagged(line,"start")
        end   = PointTagged(line,"end")
        if IsBoundary(edge):
            return ToCurve(edge)
        elif  HasLabel(edge,"h1"):
            return MatchPoints(line,start,end,Location(
                SourceVertex(edge)),Location(TargetVertex(edge)))
        else:
            return MatchPoints(line,end,start,Location(
                SourceVertex(edge)),Location(TargetVertex(edge)))

    # Mapper: replace each edge by a curve
    def map_curve_to_edge2(edge):
        start = PointTagged(line2,"start")
        end   = PointTagged(line2,"end")
        if IsBoundary(edge):
            return Nothing()
        else:
            return MatchPoints(line2,start,end,Location(
                SourceVertex(edge)),Location(TargetVertex(edge)))

    # Mapper: create stripes in each face with random
        orientations
    def hatch_map(f):
        angle = Random(f,0,2*pi,0)
        lines = StripesProperties(angle,30)
        hatch = StripesPartition(lines)
        return hatch(f)

    # Mapper: rescale each face
    def scale_map(f):
        return Scale(Contour(f), 0.9)

    # Mapping / combining operators
    tex1 = MapToEdges(map_curve_to_edge,stripes1)
    tex2 = MapToEdges(map_curve_to_edge,stripes2)
    tex3 = Union(tex1,tex2)
    tex4 = MapToFaces(scale_map,tex3)
    tex5 = MapToFaces(hatch_map,tex4)
    tex6 = MapToEdges(map_curve_to_edge2,tex5)

    ExportSVG(tex6,size)
```



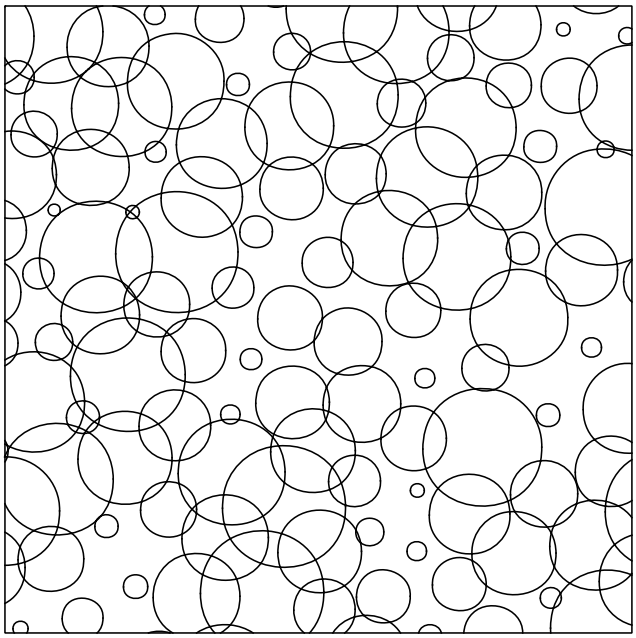
Regular result (d) 25 sec

```
def complex_regular04():
    size   = 2000
    circle = ImportSVG("data/circle.svg")

    # Grid partition
    lines1 = StripesProperties(pi/2.0,200)
    lines2 = StripesProperties(pi,200)
    partition1 = GridPartition(lines1,lines2,KEEP_OUTSIDE)

    # Mapper: place a circle in each face
    def circle_map(f):
        return Scale(MatchFace(circle,f), 2.5)

    # Mapper: rescale face
    def scale_map(f):
        return Scale(Contour(f), 0.65)

    # Mapper: keep or remove face
    def delete_map(f):
        r = Random(f,0,1,2)
        if r<0.3:
            return Nothing()
        else:
            return Contour(f)

    # Mapping operators
    tex1 = MapToFaces(circle_map,partition1)
    tex2 = MapToFaces(scale_map,tex1)
    tex3 = MapToFaces(delete_map,tex2)

    ExportSVG(tex3,size)
```



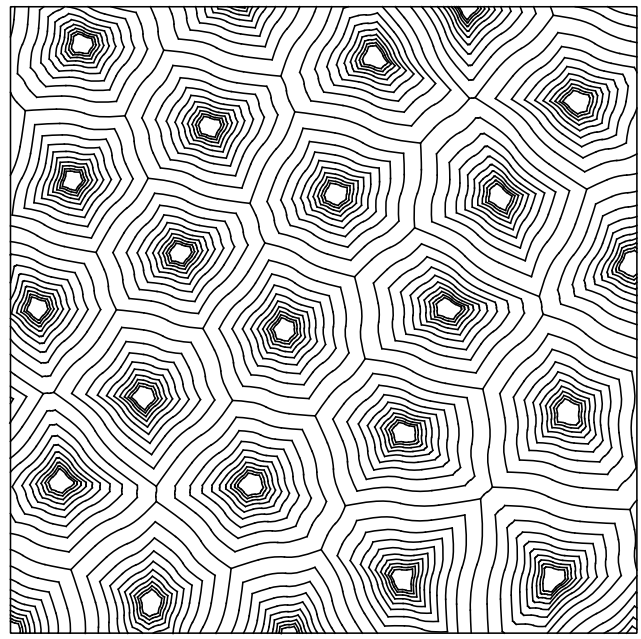
Non regular result (a) left 6 sec

```
def complex_non_regular01():
    size   = 2000
    circle = ImportSVG("data/circle.svg")

    # Uniform partition
    density = 100/(size*size)
    props = IrregularProperties(density)
    partition= UniformPartition(props, KEEP_OUTSIDE)

    # Mapper: place a circle in each face
    def circle_map(f):
        return Scale(MatchFace(circle,f), Random(f,0.2,2.0,2))

    # Mapping operator
    tex1 = MapToFaces(circle_map,partition)

    ExportSVG(tex1,size)
```



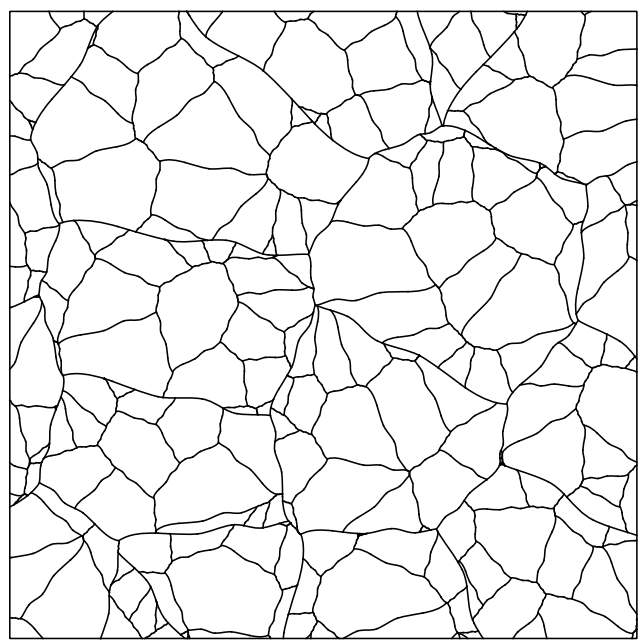
Non regular result (a) right 29 sec

```
def complex_non_regular02():
    size    = 2000
    line = ImportSVG("data/line2.svg")

    # Uniform partition
    density = 20/(2000*2000)
    props = IrregularProperties(density)
    partition= UniformPartition(props, KEEP_OUTSIDE)

    # Mapper: replace each edge by a curve
    def map_curve_to_edge(edge):
        start = PointTagged(line,"start")
        end    = PointTagged(line,"end")
        return MatchPoints(line,start,end,Location(SourceVertex(
            edge)),Location(TargetVertex(edge)))

    # Mapper: rescale a face
    def scale_map(f):
        return Scale(Contour(f), 0.8)

    # Mapping operators (iteratively rescale faces)
    l1 = MapToEdges(map_curve_to_edge,partition)
    l2 = MapToFaces(scale_map,l1)
    l3 = MapToFaces(scale_map,l2)
    l4 = MapToFaces(scale_map,l3)
    l5 = MapToFaces(scale_map,l4)
    l6 = MapToFaces(scale_map,l5)
    l7 = MapToFaces(scale_map,l6)
    l8 = MapToFaces(scale_map,l7)
    l9 = MapToFaces(scale_map,l8)
    l10 = MapToFaces(scale_map,l9)

    # Combining operators: merge all arrangements
    tex = Union(Union(Union(Union(Union(Union(Union(Union(
        l1,l2),l3),l4),l5),l6),l7),l8),l9),l10)

    ExportSVG(tex,size)
```



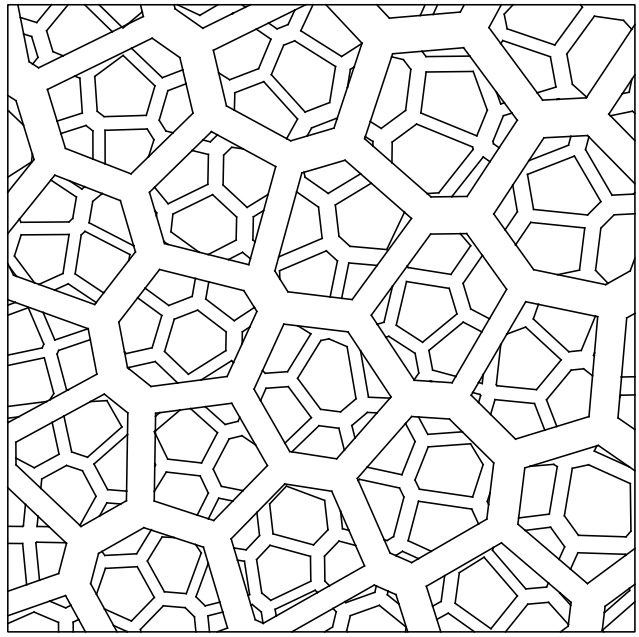
Non regular result (b) 12 sec

```
def complex_non_regular03():
    size    = 2000
    line = ImportSVG("data/line3.svg")

    # Random partition
    density = 10/(size*size)
    props = IrregularProperties(density)
    partition = RandomPartition(props, KEEP_OUTSIDE)

    # Mapper: replace internal edges by curves
    def map_curve_to_edge(edge):
        start = PointTagged(line,"start")
        end    = PointTagged(line,"end")
        if IsBoundary(edge):
            return ToCurve(edge)
        else:
            return MatchPoints(line,end,start,Location(
                SourceVertex(edge)),Location(TargetVertex(edge)))

    # Mapper: create a random partition in each face
    def partition_map(f):
        density2 = 100/(size*size)
        props2 = IrregularProperties(density2)
        partition2 = RandomPartition(props2, CROP_ADD_BOUNDARY)
        return partition2(f)

    # Mapping operators
    tex1 = MapToEdges(map_curve_to_edge,partition)
    tex2 = MapToFaces(partition_map, tex1)
    tex3 = MapToEdges(map_curve_to_edge,tex2)

    ExportSVG(tex3,size)
```



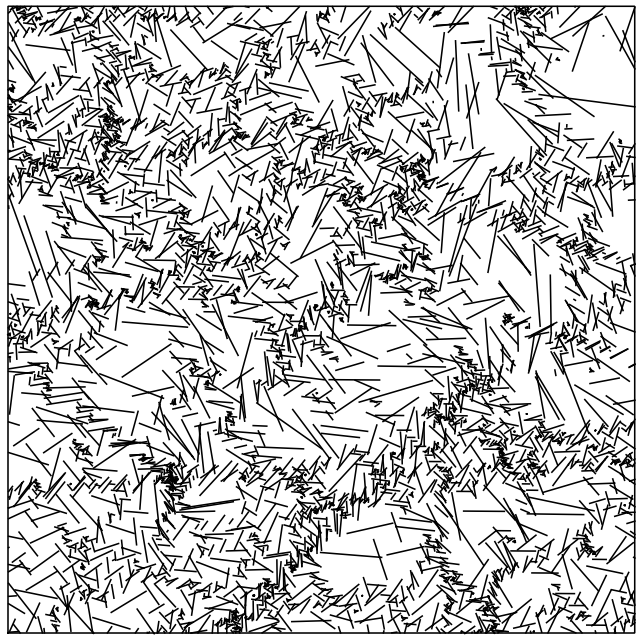
Non regular result (c) 30 sec

```
def complex_non_regular04():
    size    = 2000

    # Uniform partition
    density1 = 20/(size*size)
    props1 = IrregularProperties(density1)
    partition1 = UniformPartition(props1, KEEP_OUTSIDE)

    # Uniform partition
    density2 = 90/(size*size)
    props2 = IrregularProperties(density2)
    partition2 = UniformPartition(props2, KEEP_OUTSIDE)

    # Mapper: rescale faces
    def scale_map(f):
        return Scale(Contour(f), 0.8)

    # Mapper: create a uniform partition in each face
    def partition2_map(f):
       return MapToFaces(scale_map, partition2)(f)

    # Mapping operators
    tex1 = MapToFaces(scale_map, partition1)
    tex2 = MapToFaces(scale_map, partition2)

    # Combining operators
    tex3 = Inside(tex2, tex1, CROP)
    tex4 = Union(tex1, tex3)

    ExportSVG(tex4, size)
```



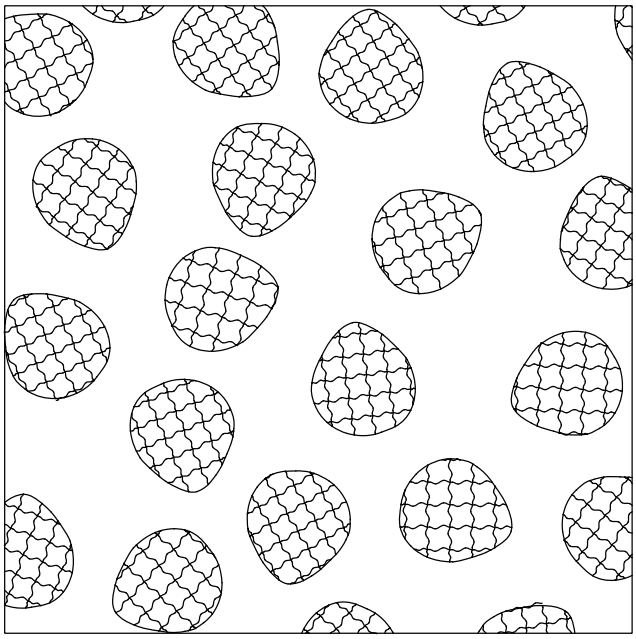
Non regular result (d) 1 min, 28 sec

```
def complex_non_regular05():
    size    = 2000

    # Random partition
    density = 30/(2000*2000)
    props = IrregularProperties(density)
    partition= RandomPartition(props, KEEP_OUTSIDE)

    # Mapper: rotate edge
    def rotate_e(e):
       return Rotate(ToCurve(e), pi/4)

    # Mapper: rescale edge
    def scale_map_5(e):
       return Scale(ToCurve(e), 5.0)

    # Mapper: rescale edge
    def scale_map_10(e):
       return Scale(ToCurve(e), 10.0)

    # Mapping operators
    tex1 = MapToEdges(scale_map_5, partition)
    tex2 = MapToEdges(scale_map_10, partition)
    tex3 = MapToEdges(rotate_e, tex2)

    ExportSVG(tex3, size)
```



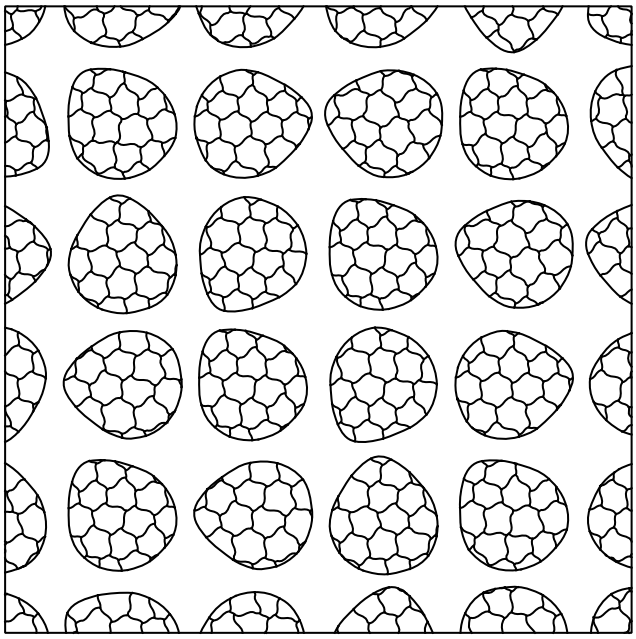
Script edition (a) original script 25 sec

```
def overview_orig():
    size = 2500
    blob = ImportSVG("data/blob.svg")
    zig  = ImportSVG("data/zig.svg")

    # Mapper: place a blob in each face
    def map_blob_to(face):
        new_blob = Rotate(blob,Random(face,0,2*pi,0))
        return MatchPoint(new_blob,BBoxCenter(new_blob),Centroid(
            face))

    # Mapper: replace each edge by a curved line
    def map_curve_to(edge):
        if IsBoundary(edge):
            return ToCurve(edge)
        src_c = PointLabeled(zig,"start")
        dst_c = PointLabeled(zig,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(zig,src_c,dst_c,src_v,dst_v)

    def create_blob_tex():
        # Uniform partition
        props    = IrregularProperties(11/(2000*2000))
        init_tex = UniformPartition(props,KEEP_OUTSIDE)

        # Mapping operators
        return MapToFaces(map_blob_to,init_tex)

    # Mapper: generate a texture in each face
    def create_zig_tex(face):

        # Grid partition with randomized orientations
        theta    = Random(face,0,2*pi,1)
        width    = BBoxWidth(face)/5
        lines1   = StripesProperties(theta,width)
        lines2   = StripesProperties(theta+pi/2,width)
        init_tex = GridPartition(lines1,lines2,CROP_ADD_BOUNDARY)

        # Mapping operator
        texture2 = MapToEdges(map_curve_to,init_tex)
        return texture2(face)

    init_tex  = create_blob_tex()
    final_tex = MapToFaces(create_zig_tex,init_tex)

    # Export final texture
    ExportSVG(final_tex,size)
```



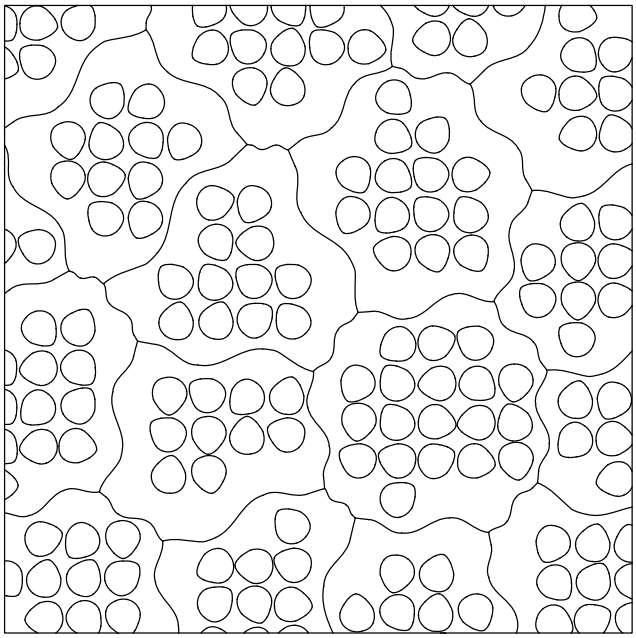
Script edition (b) switch partitions 52 sec

```
def overview_var1():
    size = 2500
    blob = ImportSVG("data/blob.svg")
    zig  = ImportSVG("data/zig.svg")

    # Mapper: place a blob in each face
    def map_blob_to(face):
        new_blob = Rotate(blob,Random(face,0,2*pi,0))
        return MatchPoint(new_blob,BBoxCenter(new_blob),Centroid(
            face))

    # Mapper: replace each edge by a curved line
    def map_curve_to(edge):
        if IsBoundary(edge):
            return ToCurve(edge)
        src_c = PointLabeled(zig,"start")
        dst_c = PointLabeled(zig,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(zig,src_c,dst_c,src_v,dst_v)

    def create_blob_tex():
        # Grid partition
        theta    = 0
        width    = size/4
        lines1   = StripesProperties(theta,width)
        lines2   = StripesProperties(theta+pi/2,width)
        init_tex = GridPartition(lines1,lines2,KEEP_OUTSIDE)

        # Mapping operators
        return MapToFaces(map_blob_to,init_tex)

    # Mapper: generate a texture in each face
    def create_zig_tex(face):
        # Uniform partition
        props    = IrregularProperties(11/(BBoxWidth(face)*
            BBoxHeight(face)))
        init_tex = UniformPartition(props,CROP_ADD_BOUNDARY)

        # Mapping operator
        texture2 = MapToEdges(map_curve_to,init_tex)
        return texture2(face)

    init_tex  = create_blob_tex()
    final_tex = MapToFaces(create_zig_tex,init_tex)

    # Export final texture
    ExportSVG(final_tex,size)
```



Script edition (c) switch arrangements 12 sec

```
def overview_var2():
    size = 2500
    blob = ImportSVG("data/blob.svg")
    zig  = ImportSVG("data/zig.svg")

    # Mapper: place a blob in each face
    def map_blob_to(face):
        new_blob = Rotate(blob,Random(face,0,2*pi,0))
        return MatchPoint(new_blob,BBoxCenter(new_blob),Centroid(
            face))

    # Mapper: replace each edge by a curved line
    def map_curve_to(edge):
        if IsBoundary(edge):
            return ToCurve(edge)
        src_c = PointLabeled(zig,"start")
        dst_c = PointLabeled(zig,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(zig,src_c,dst_c,src_v,dst_v)

    def create_blob_tex(face):
        # Grid partition
        theta    = 0 #Random(face,0,2*pi,0)
        width    = size/16
        lines1   = StripesProperties(theta,width)
        lines2   = StripesProperties(theta+pi/2,width)
        init_tex = GridPartition(lines1,lines2,KEEP_INSIDE)

        # Mapping operators
        return MapToFaces(map_blob_to,init_tex)(face)

    # Mapper: generate a texture in each face
    def create_zig_tex():
        # Uniform partition
        props    = IrregularProperties(11/(size*size))
        init_tex = UniformPartition(props,KEEP_OUTSIDE)

        # Mapping operator
        return MapToEdges(map_curve_to,init_tex)

    init_tex  = create_zig_tex()
    final_tex = Union(init_tex,MapToFaces(create_blob_tex,
        init_tex))

    # Export final texture
    ExportSVG(final_tex,size)
```



Script edition (d) vary mappings 56 sec

```
def overview_var3():
    size   = 2500
    flower = ImportSVG("data/flower.svg")
    leaf   = ImportSVG("data/leaf.svg")
    zig    = ImportSVG("data/zig7.svg")

    # Mapper: place a blob in each face
    def map_blob_to(face):
        new_flower = Scale(Rotate(flower,Random(face,0,2*pi,0)),
             Random(face,0.7,1,1))
        new_leaf = Scale(Rotate(leaf,Random(face,0,2*pi,2)),
             Random(face,0.7,1,3))
        if Random(face,0,1,4)>0.5:
            return MatchPoint(new_flower,BBoxCenter(new_flower),
                 Centroid(face))
        else:
            return MatchPoint(new_leaf,BBoxCenter(new_leaf),
                 Centroid(face))

    # Mapper: replace each edge by a curved line
    def map_curve_to(edge):
        if IsBoundary(edge):
            return ToCurve(edge)
        src_c = PointLabeled(zig,"start")
        dst_c = PointLabeled(zig,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(zig,src_c,dst_c,src_v,dst_v)

    def create_blob_tex(face):
        # Grid partition
        theta  = 0
        width  = size/16
        lines1 = StripesProperties(theta,width)
        lines2 = StripesProperties(theta+pi/2,width)
        init_tex = GridPartition(lines1,lines2,KEEP_INSIDE)

        # Mapping operators
        return MapToFaces(map_blob_to,init_tex)(face)

    # Mapper: generate a texture in each face
    def create_zig_tex():
        # Uniform partition
        props    = IrregularProperties(11/(size*size))
        init_tex = UniformPartition(props,KEEP_OUTSIDE)

        # Mapping operator
        return MapToEdges(map_curve_to,init_tex)

    init_tex  = create_zig_tex()
    final_tex = Union(init_tex,MapToFaces(create_blob_tex,
         init_tex))

    # Export final texture
    ExportSVG(final_tex,size)
```

Parquetry 20 sec

```
def parquetry():
    # Label filtering: transforms a grid partition into
    # parquetry slats with space in between
    def parquetry_structure(e):
        faces = Faces()
        faces.push_back(LeftFace(e))
        faces.push_back(RightFace(e))
        if not LeftFace(e).is_bounded_face()\
        or not RightFace(e).is_bounded_face():
            return ToCurve(e)
        if HasTag(faces, "v1") and HasTag(faces, "v2") and
            HasTag(faces, "h1")\
        or HasTag(faces, "v2") and HasTag(faces, "v3") and
            HasTag(faces, "h1")\
        or HasTag(faces, "v3") and HasTag(faces, "v4") and
            HasTag(faces, "h1")\
        or HasTag(faces, "v4") and HasTag(faces, "v1") and
            HasTag(faces, "h1")\
        or HasTag(faces, "v1") and HasTag(faces, "v2") and
            HasTag(faces, "h3")\
        or HasTag(faces, "v2") and HasTag(faces, "v3") and
            HasTag(faces, "h3")\
        or HasTag(faces, "v3") and HasTag(faces, "v4") and
            HasTag(faces, "h3")\
        or HasTag(faces, "v4") and HasTag(faces, "v1") and
            HasTag(faces, "h3")\
        or HasTag(faces, "h1") and HasTag(faces, "h2") and
            HasTag(faces, "v1")\
        or HasTag(faces, "h2") and HasTag(faces, "h3") and
            HasTag(faces, "v1")\
        or HasTag(faces, "h3") and HasTag(faces, "h4") and
            HasTag(faces, "v3")\
        or HasTag(faces, "h4") and HasTag(faces, "h1") and
            HasTag(faces, "v3")\
        or HasTag(faces, "v2") and HasTag(faces, "v3") and
            HasTag(faces, "h2")\
        or HasTag(faces, "v3") and HasTag(faces, "v4") and
            HasTag(faces, "h2")\
        or HasTag(faces, "v4") and HasTag(faces, "v1") and
            HasTag(faces, "h4")\
        or HasTag(faces, "v1") and HasTag(faces, "v2") and
            HasTag(faces, "h4"):
            return Nothing()
        return ToCurve(e)

    # Grid partition
    my_scale = 50
    interstice_size = 0.2*my_scale
    lines1   = StripesProperties(0.0,1*my_scale,interstice_size
        ,1*my_scale,interstice_size)
    SetFaceLabels(lines1, "h1", "h2", "h3", "h4")
    lines2   = StripesProperties(pi/6+pi/32+pi/100,8*my_scale,
        interstice_size,8*my_scale, interstice_size)
    SetFaceLabels(lines2, "v1", "v2", "v3", "v4")
    part = GridPartition(lines1,lines2,KEEP_OUTSIDE)
    # Refinement
    struct = MapToEdges(parquetry_structure,part)
    # Export
    frame = ImportSVG("data/new_carpet_outline.svg")
    ExportSVGinDomain(struct, frame)
```

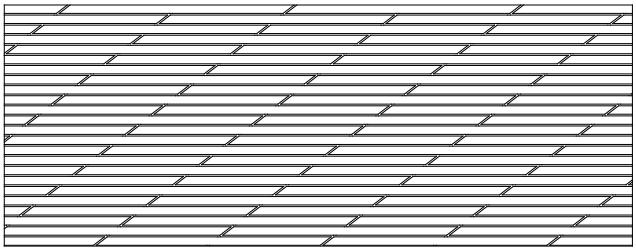
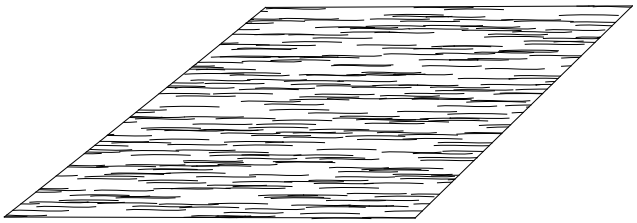


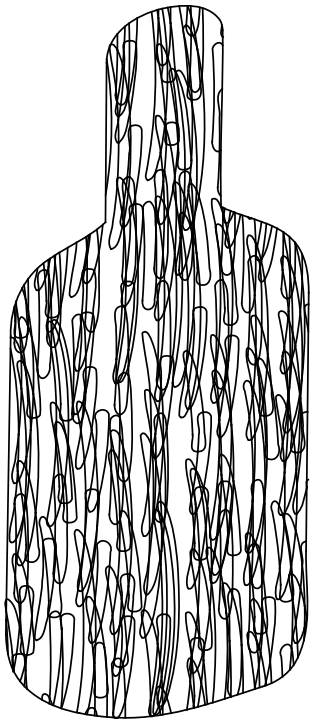
Table 45 sec

```
def table():
    # Import elements
    hatch1 = ImportSVG("data/smoothhatch1.svg")
    hatch2 = ImportSVG("data/smoothhatch2.svg")
    hatch3 = ImportSVG("data/smoothhatch3.svg")
    hatches = [hatch1, hatch2, hatch3]
    # Parameter for modifying hatch orientation
    side_angle = 0

    # Mapper: place a hatch on a vertex
    def place_hatch(v):
        randx = Random(v, -20, 20, 1)
        randy = Random(v, -80, 80, 2)
        loc = Location(v) + Point(randx, randy)
        elem0 = hatches[floor(Random(0,len(hatches)))]
        elem  = Rotate(elem0, side_angle)
        return MatchPoint(Scale(elem, 1.0),BBoxCenter(elem),loc)

    # Grid partition
    linesA   = StripesProperties(side_angle + pi/3.0,60)
    linesB= StripesProperties(side_angle + 2.0*pi/3.0, 150)
    part = GridPartition(linesA, linesB, KEEP_OUTSIDE)
    # Apply the mapper
    texture = MapToVertices(place_hatch, part)
    # Fill the outline
    frame = ImportSVG("data/outline_table_topside.svg")
    ExportSVGinDomain(texture,frame)
```



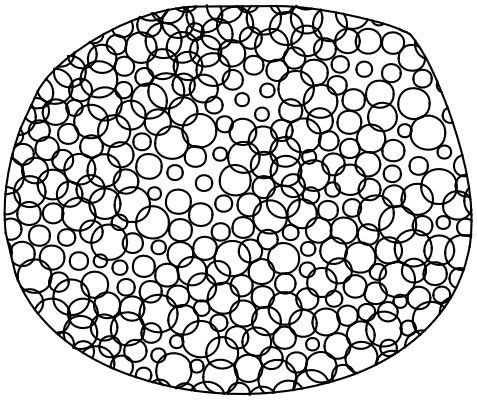
Bottle 11 sec

```
    frame = ImportSVG("data/outline_bottle.svg")
    bubble1 = ImportSVG("data/bubble1.svg")
    bubble2 = ImportSVG("data/bubble2.svg")
    bubble3 = ImportSVG("data/bubble3.svg")
    bubble4 = ImportSVG("data/bubble4.svg")
    bubbles = [bubble1, bubble2, bubble3, bubble4]
    side_angle = pi / 2.0

    # Mapper: place a bubble on a vertex
    def place_bubble(v):
        randx = Random(v, -20, 20, 1)
        randy = Random(v, -40, 40, 2)
        loc = Location(v) + Point(randx, randy)
        elem0 = bubbles[floor(Random(0,len(bubbles)))]
        elem  = Rotate(elem0, side_angle)
        return MatchPoint(Scale(elem, 1.0),BBoxCenter(elem),loc)

    # Grid partition
    linesA   = StripesProperties(side_angle + pi/3.0,60)
    linesB= StripesProperties(side_angle +2* pi/3.0, 60)
    part = GridPartition(linesA, linesB, KEEP_OUTSIDE)
    # Apply the mapper
    texture = MapToVertices(place_bubble, part)
    # Fill the outline
    ExportSVGinDomain(texture, frame)
```



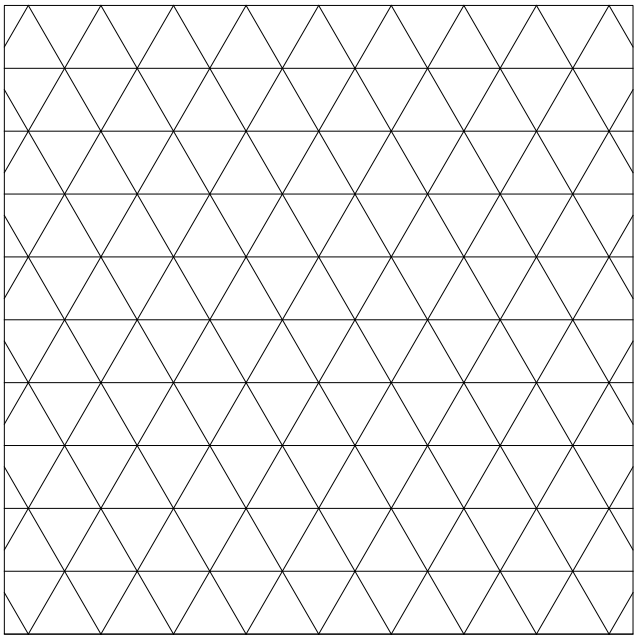
Teapot 22 sec

```
def teapot():
    density = 3.0e-4
    circle = ImportSVG("data/circle.svg")

    # Mapper: place a circle in each face
    def face_to_circle(face):
        src_p = BBoxCenter(circle)
        dst_p = Centroid(face)
        return Scale(MatchPoint(circle,src_p,dst_p),Random(face
            ,0.05,0.15,0))

    # Uniform partition with given density
    props  = IrregularProperties(density)
    part   = UniformPartition(props,KEEP_OUTSIDE)
    # Apply the mapper
    tex3 = MapToFaces(face_to_circle,part)
    # Fill the outline
    frame = ImportSVG("data/outline_teapot.svg")
    ExportSVGinDomain(tex3, frame)
```



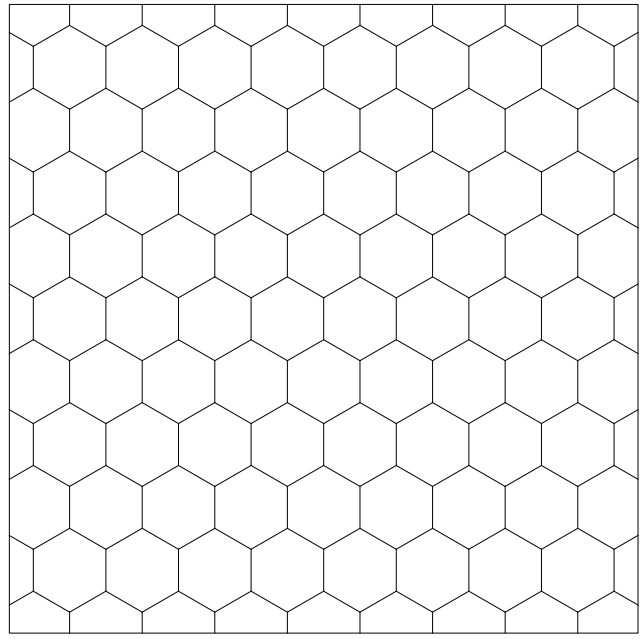
Triangles 4 sec

```
# Mapper: splits in two a unique face from a grid partition
def split(face):
    are_incident = lambda e,f: f == LeftFace(e) or f ==
        RightFace(e)
    v1 = 0
    v2 = 0
    for v in IncidentVertices(face):
        score_source = 0
        score_target = 0
        for e in IncidentEdges(v):
            if v == SourceVertex(e) and are_incident(e, face):
                score_source += 1
            if v == TargetVertex(e) and are_incident(e, face):
                score_target += 1
        if score_source == 2:
            v1 = v
        if score_target == 2:
            v2 = v
    if v1 == 0
        return Nothing()
    seg = ImportSVG("segment.svg")
    start_seg = PointLabeled(seg, "start")
    end_seg = PointLabeled(seg, "end")
    return MatchPoints(seg, start_seg, end_seg, Location(v1),
        Location(v2))

def example_triangles():
    # Grid partition
    props1 = StripesProperties(pi/3.0, 200)
    props2 = StripesProperties(-pi/3.0, 200)
    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
    # Apply mapper and merge the result with the original grid
    triangles = Union(grid, MapToFaces(split, grid))
    # Export final texture
    ExportSVG(triangles, 2000)
```



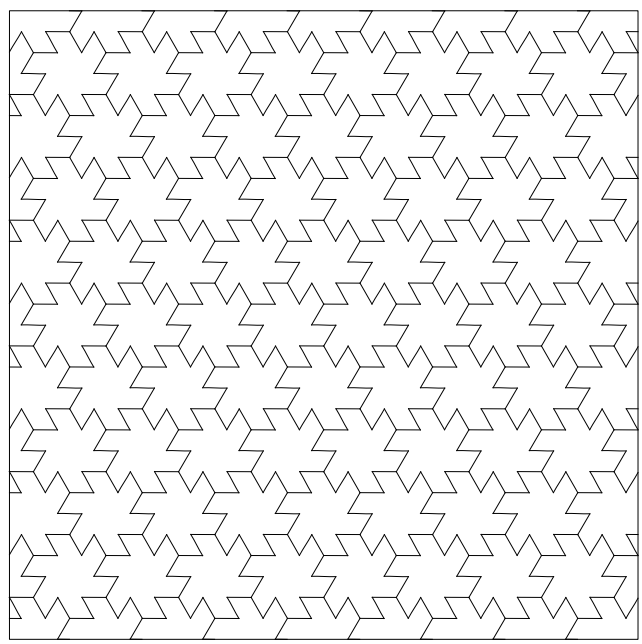
Bee hive 5 sec

```
# Mapper: returns a segment linking the centroids of the faces incident to e
def dual(e):
    if e.is_outside():
        return Nothing()
    seg = ImportSVG("segment.svg")
    start_seg = PointLabeled(seg, "start")
    end_seg = PointLabeled(seg, "end")
    return MatchPoints(seg, start_seg, end_seg, Centroid(LeftFace(e)), Centroid(RightFace(e)))
def example_hexa():
    # Grid partition
    props1 = StripesProperties(pi/3.0, 200)
    props2 = StripesProperties(-pi/3.0, 200)
    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
    # Split faces (from example ``Triangles'')
    triangles = Union(grid, MapToFaces(split, grid))
    # Take the duals of all edges
    hexa = MapToEdges(dual, triangles)
    # Export final tetxure
    ExportSVG(hexa, 2000)
```



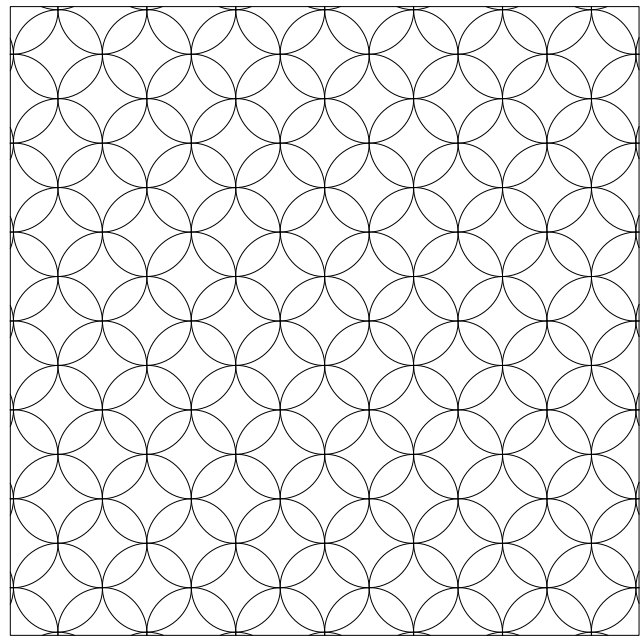
Wallpaper group P6 48 sec

```
def example_p6():
    # Import SVG element
    line = ImportSVG("data/p6_elt.svg")
    # Mapper: replaces an edge by the imported SVG element
    def line_to_curve(edge):
        src_c = PointLabeled(line,"start")
        dst_c = PointLabeled(line,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(line,src_c,dst_c,src_v,dst_v)
    props1 = StripesProperties(pi/3.0, 200)
    props2 = StripesProperties(-pi/3.0, 200)
    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
    # Split faces (from example ``Triangles'')
    triangles = Union(grid, MapToFaces(split, grid))
    # Take duals of all edges (from example ``Bee hive'')
    hexa = MapToEdges(dual, triangles)
    # Replace edges by Z-shaped curves
    p6 = MapToEdges(line_to_curve, hexa)
    # Export final texture
    ExportSVG(p6, 2000)
```



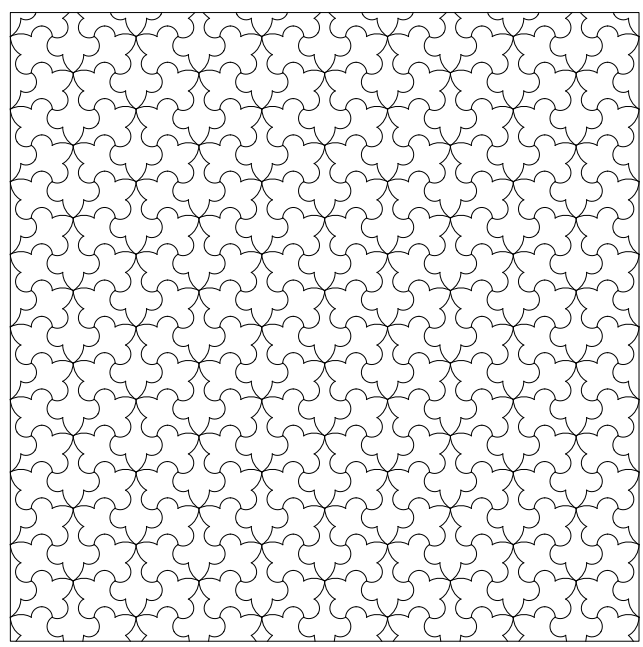
Wallpaper group P4M 24 sec

```
def example_p4m():
    # Import SVG element
    line = ImportSVG("data/p4m_elt.svg")
    # Mapper: replaces an edge by the imported SVG element
    def line_to_curve(edge):
        src_c = PointLabeled(line,"start")
        dst_c = PointLabeled(line,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(line,src_c,dst_c,src_v,dst_v)
    # Mapper: replaces an edge by a rotation of the imported
        SVG element
    def line_to_curve_inv(edge):
        dst_c = PointLabeled(line,"start")
        src_c = PointLabeled(line,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(line,src_c,dst_c,src_v,dst_v)
    # Grid partition
    props1 = StripesProperties(pi/4.0, 200)
    props2 = StripesProperties(-pi/4.0, 200)
    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
    # Take the results of both mappers
    p4m = Union(MapToEdges(line_to_curve, grid), MapToEdges(
        line_to_curve_inv, grid))
    # Export final texture
    ExportSVG(p4m, 2000)
```



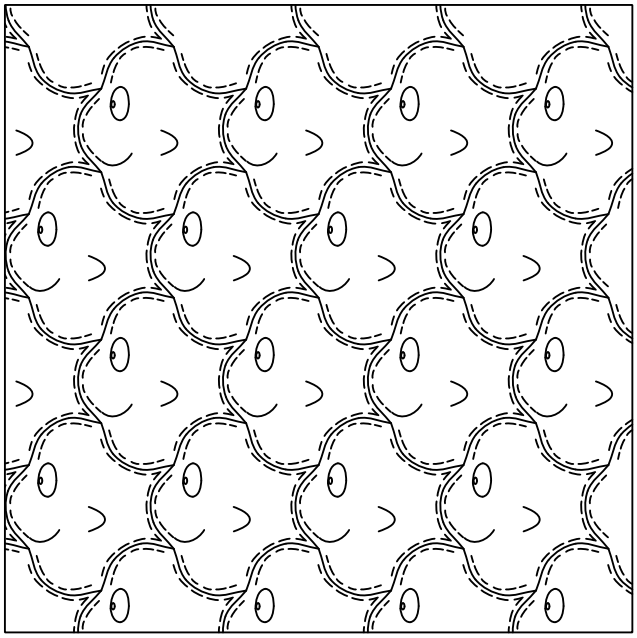
Wallpaper group P31M 50 sec

```
def example_p31m():
    line0 = ImportSVG("data/p31m_elt.svg")
    line0_sym = ImportSVG("data/p31m_elt_sym.svg")
    # Mapper: subdivides a triangle face in three pseudo-
        triangles
    def subdivide(face):
        out = Nothing()
        line = line0
        nb_left = 0
        for e in IncidentEdges(face):
            if face == LeftFace(e):
                nb_left = nb_left + 1
        if nb_left > 1:
            line = line0_sym
        src_c = PointLabeled(line,"start")
        dst_c = PointLabeled(line,"end")
        dst_v = Centroid(face)
        for v in IncidentVertices(face):
            src_v = Location(v)
            curve = MatchPoints(line,src_c,dst_c,src_v,dst_v)
            out = Append(out, curve)
        return out
    # Grid partition
    props1 = StripesProperties(pi/6.0, 200)
    props2 = StripesProperties(pi-pi/6.0, 200)
    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
    # Split faces (from example ``Triangles'')
    triangles = Union(grid, MapToFaces(split, grid))
    # Subdivide triangle faces
    p31m = MapToFaces(subdivide, triangles)
    # Export final texture
    ExportSVG(p31m, 2000)
```



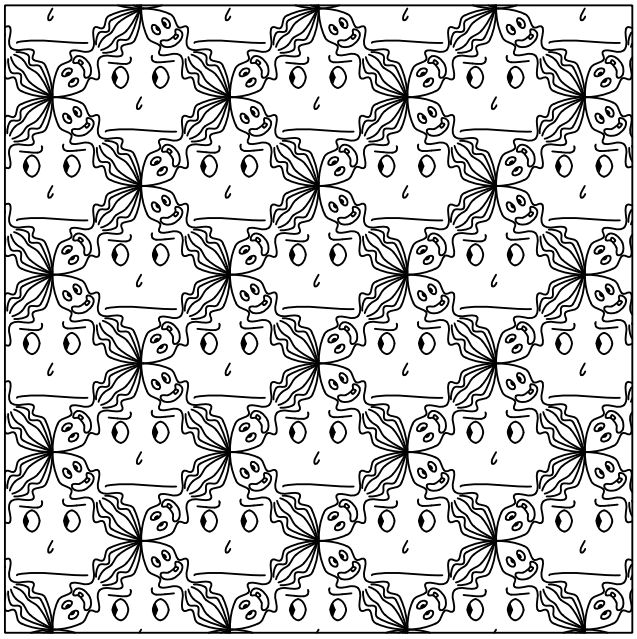
Escher P6 38 sec

```
def example_p6_escher():
    # Imported SVG elements
    line = ImportSVG("data/p6_elt_complex3.svg")
    center_elt = ImportSVG("data/p6_elt_complex3_center.svg")
    # Mapper: replaces an edge by a piece of a fish outline
    def line_to_curve(edge):
        src_c = PointLabeled(line,"start")
        dst_c = PointLabeled(line,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(line,src_c,dst_c,src_v,dst_v)
    # Mapper: places a fish interior inside a face
    def face_to_center_elt(f):
        centers_center = PointLabeled(center_elt,"end") + Point(20,0)
        return MatchPoint(center_elt, centers_center, Centroid(f))
    # Grid partition
    props1 = StripesProperties(pi/3.0, 200)
    props2 = StripesProperties(-pi/3.0, 200)
    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
    # Split faces (from example ``Triangles'')
    triangles = Union(grid, MapToFaces(split, grid))
    # Take duals of all edges (from example ``Bee hive'')
    hexa = MapToEdges(dual, triangles)
    # Replace edges by fish outlines
    p6 = MapToEdges(line_to_curve, hexa)
    # Place fish interiors
    p6_center = MapToFaces(face_to_center_elt, hexa)
    # Export final texture
    final = Union(p6, p6_center)
    ExportSVG(final, 1000)
```



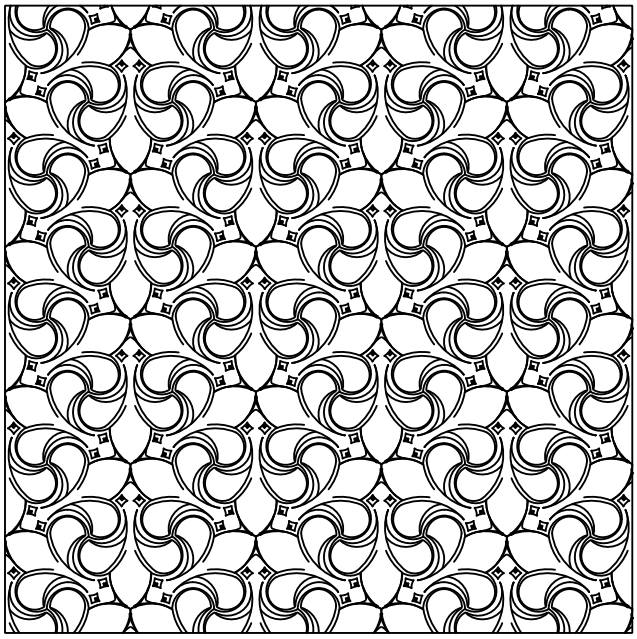
Escher P4M 43 sec

```
def example_p4m_escher():
    # Imported SVG elements
    octopus = ImportSVG("data/p4m_elt_complex1.svg")
    guy = ImportSVG("data/p4m_elt_complex2.svg")
    # Mapper: places an octopus on an edge
    def line_to_octopus(edge):
        src_c = PointLabeled(octopus,"start")
        dst_c = PointLabeled(octopus,"end")
        src_v = Location(SourceVertex(edge))
        dst_v = Location(TargetVertex(edge))
        return MatchPoints(octopus,src_c,dst_c,src_v,dst_v)
    # Mapper: places a guy on a face's center
    def face_to_guy(face):
        center_guy = PointLabeled(guy,"start")
        c = Centroid(face)
        return MatchPoint(guy,center_guy, c)
    # Grid partition
    props1 = StripesProperties(pi/4.0, 200)
    props2 = StripesProperties(-pi/4.0, 200)
    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
    # Merge octopuses and guys
    p4m = Union(MapToEdges(line_to_octopus, grid), MapToFaces(face_to_guy, grid))
    # Export final texture
    ExportSVG(p4m, 1000)
```



Escher P31M 35 sec

```
def example_p31m_escher():
    # Imported SVG elements
    line0 = ImportSVG("data/p31m_elt_complex.svg")
    line0_sym = ImportSVG("data/p31m_elt_complex_sym.svg")
    # Mapper: subdivides each triangle face into three pseudo-
        triangles
    def subdivide(face):
        out = Nothing()
        line = line0
        nb_left = 0
        for e in IncidentEdges(face):
            if face == LeftFace(e):
                nb_left = nb_left + 1
        if nb_left > 1:
            line = line0_sym
        src_c = PointLabeled(line,"start")
        dst_c = PointLabeled(line,"end")
        dst_v = Centroid(face)
        for v in IncidentVertices(face):
            src_v = Location(v)
            curve = MatchPoints(line,src_c,dst_c,src_v,dst_v)
            out = Append(out, curve)
        return out
    # Grid partition
    props1 = StripesProperties(pi/6.0, 200)
    props2 = StripesProperties(pi-pi/6.0, 200)
    grid = GridPartition(props1, props2, KEEP_OUTSIDE)
    # Split faces (from example ``Triangles'')
    triangles = Union(grid, MapToFaces(split, grid))
    # Subdivide resulting triangle faces
    p31m = MapToFaces(subdivide, triangles)
    # Export final texture
    ExportSVG(p31m, 1000)
```



Microprocessor 54 sec

```
def processor():
    # Global parameters
    size = 2000
    step = 50

    # ———————— Level 1 ————————
    densities = [0.002, 0.00175, 0.00125, 0.00225, 0.001]
    # Paths and density indexes
    elts_densities = [["net1.svg",0],["net2.svg",0],["net3.svg",1],["net4.svg",0],["net5.svg",1],["net6.svg",1],["net7.svg",2],["net8.svg",1],["net9.svg",0],["net10.svg",3],["net11.svg",3],["net12.svg",4]]

    # Higher order function: generates a mapper for 'elt'
    def mapper(elt):
        def new_face_mapper(face):
            pt = Centroid(face)
            # Clamp position for regular silicium look
            clamp = lambda x: step * floor(x/step)
            pt = Point(clamp(pt.x()), clamp(pt.y()))
            return MatchPoint(elt, PointLabeled(elt, "start"), pt)
        return new_face_mapper

    # dst: distributes randomly elt 'e' with density 'd'
    dst = lambda e, d: MapToFaces(mapper(e), RandomPartition(IrregularProperties(d), KEEP_OUTSIDE))

    # Distribute all elements
    level1 = lambda f: Nothing()
    for e_d in elts_densities:
        d = densities[e_d[1]]
        level1 = Merge(level1, dst(ImportSVG(e_d[0]), d))

    # ———————— Level 2 ————————
    ships = ["ship1.svg","ship2.svg","ship3.svg","ship4.svg","ship5.svg","ship6.svg"]

    # Mapper: generates and places a ship on the face
    def face_to_ship(face):
        ship = ImportSVG(ships[floor(Random(face,0,6,0))])
        ship = Scale(ship, floor(Random(face,0,4,0)))
        pt = Centroid(face)
        # Clamp position for regular silicium look
        clamp = lambda x: step * floor(x/step)
        pt = Point(clamp(pt.x()), clamp(pt.y()))
        res = MatchPoint(ship, BBoxCenter(ship), pt)
        return res

    props = IrregularProperties(8/(2000*2000))
    part_ships = UniformPartition(props, KEEP_OUTSIDE)
    level2 = MapToFaces(face_to_ship, part_ships)

    # ———————— Final texture ————————
    c = ImportSVG("data/connector.svg")
    j = ImportSVG("data/joint.svg")

    # Mapper: generates connectors and soldered joints
    def make_cj(v):
        # Connectors at contacts between level1 and level2
        if len(IncidentEdges(v)) == 3 and HasLabel(IncidentFaces(v),"ship"):
            return MatchPoint(c, BBoxCenter(c), Location(v))
        # Generate joints at junction ends
        elif len(IncidentEdges(v)) == 1:
            return MatchPoint(j, BBoxCenter(j), Location(v))
        return Nothing()

    lvls_1_2 = Outside(level1,level2,CROP_ADD_BOUNDARY)
    cjs = MapToVertices(make_cj, lvls_1_2)
    final_tex = Outside(lvls_1_2, cjs, CROP_ADD_BOUNDARY)
    ExportSVG(final_tex, size)
```