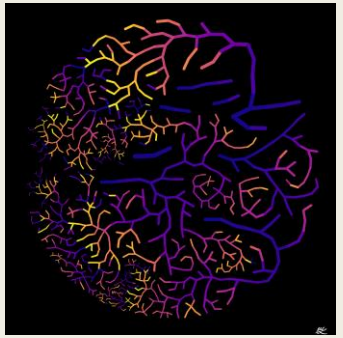
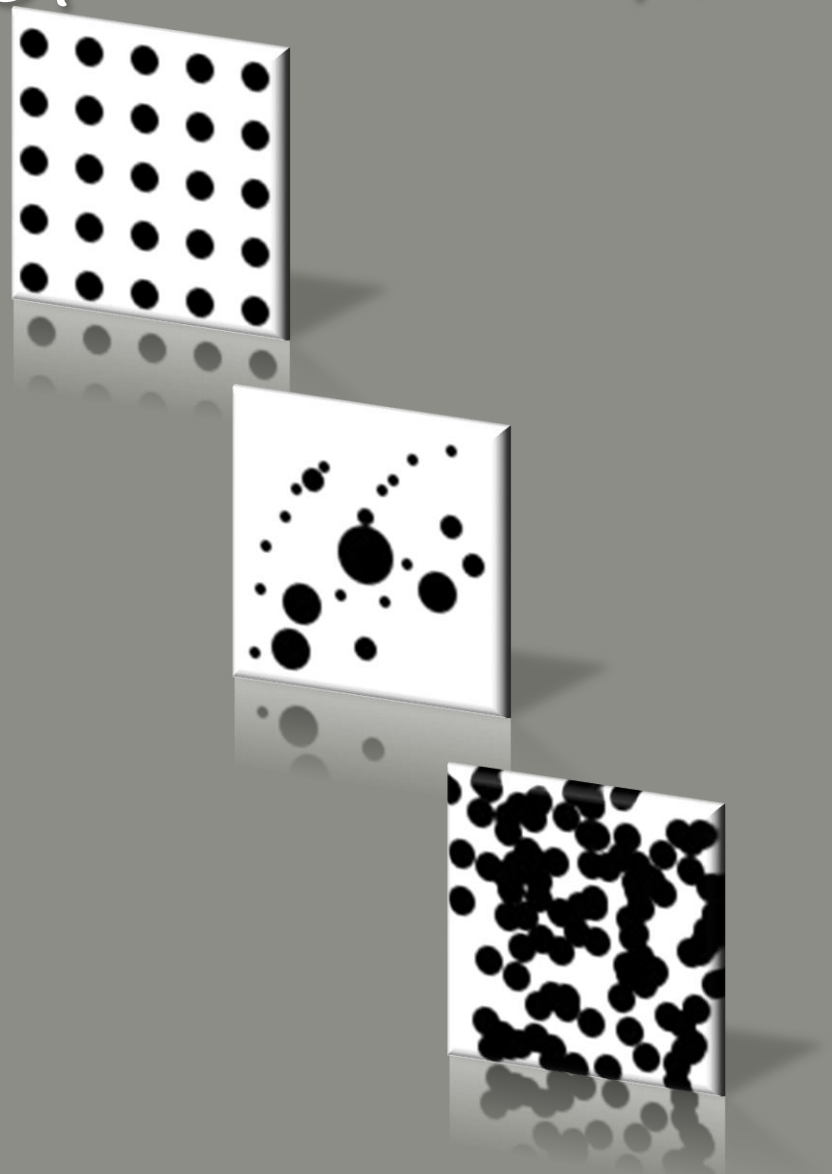
XML FOR ART

A case study

Mary Holstege @mathling



Generative Art



Abstract

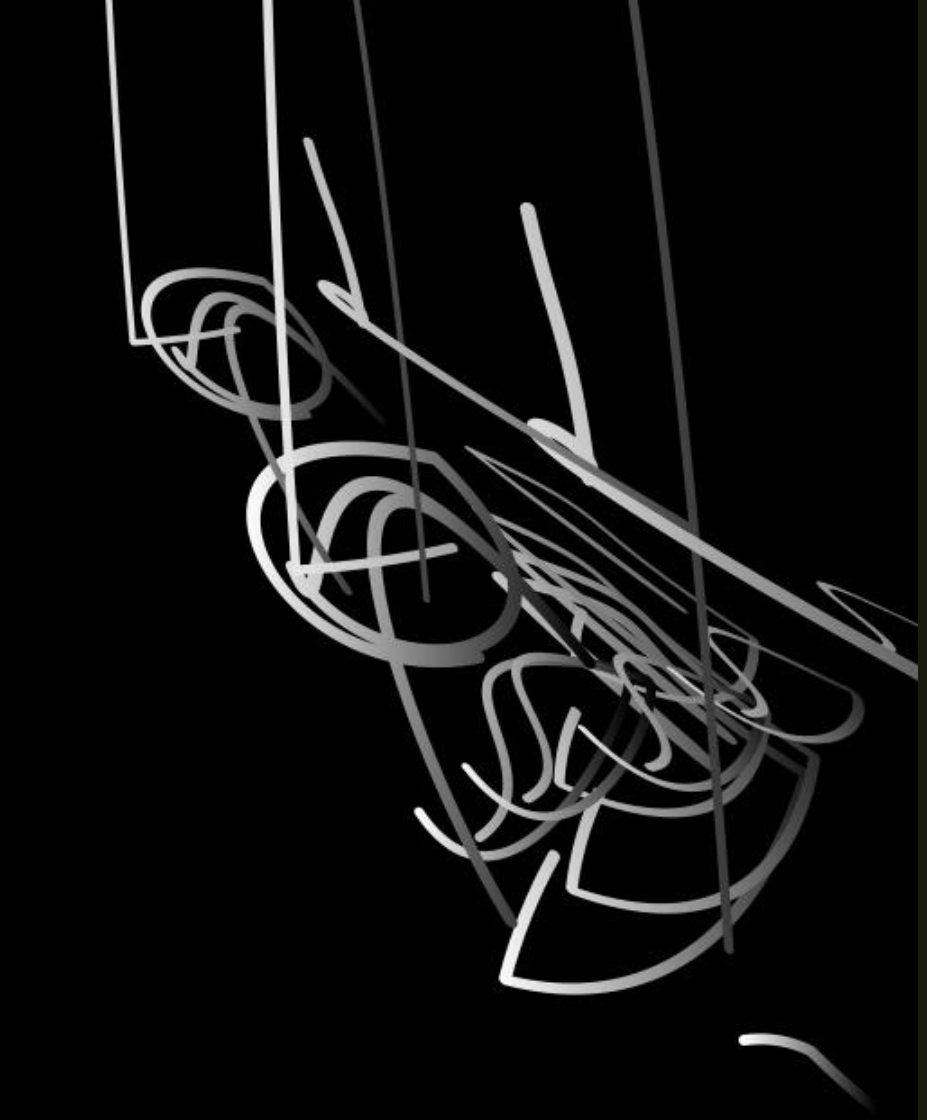
Generated

created by a system or rules

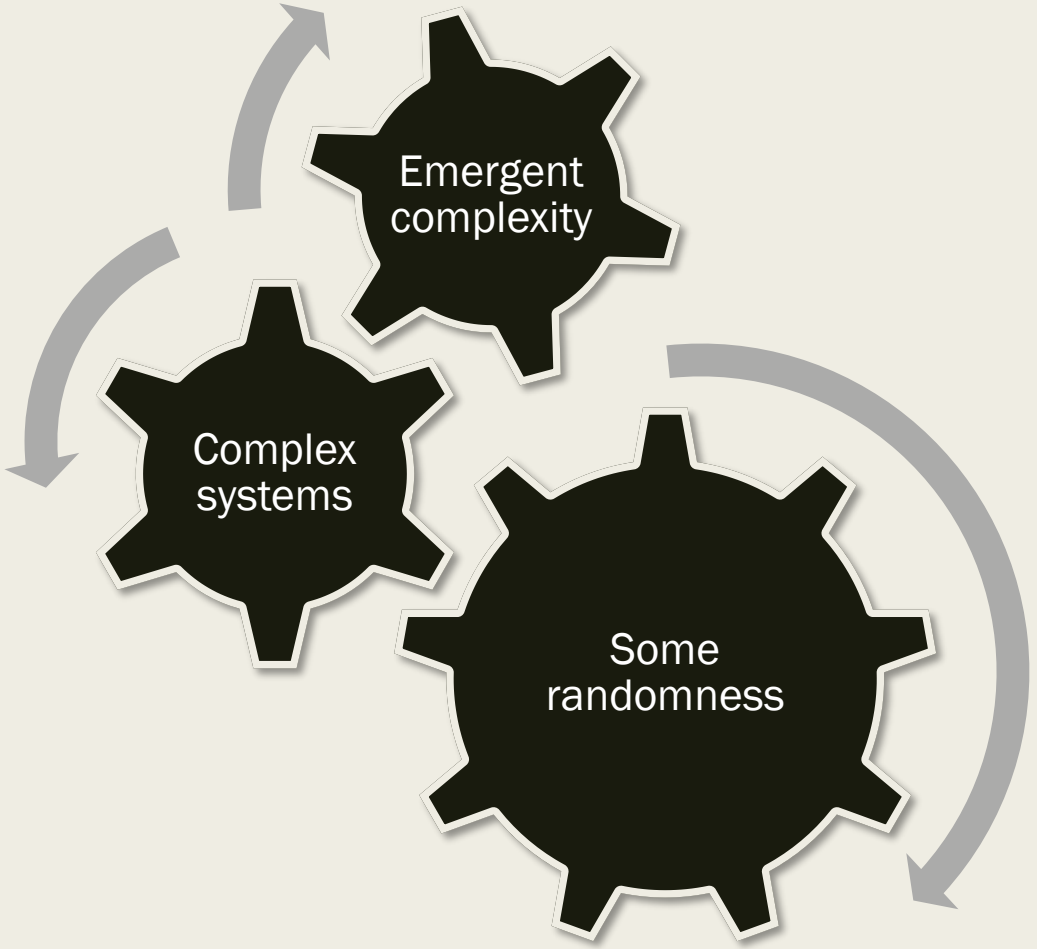
Artistic complexity

Too orderly: boring

Too random: confusing, therefore boring



Artistic Complexity



Story of a Project

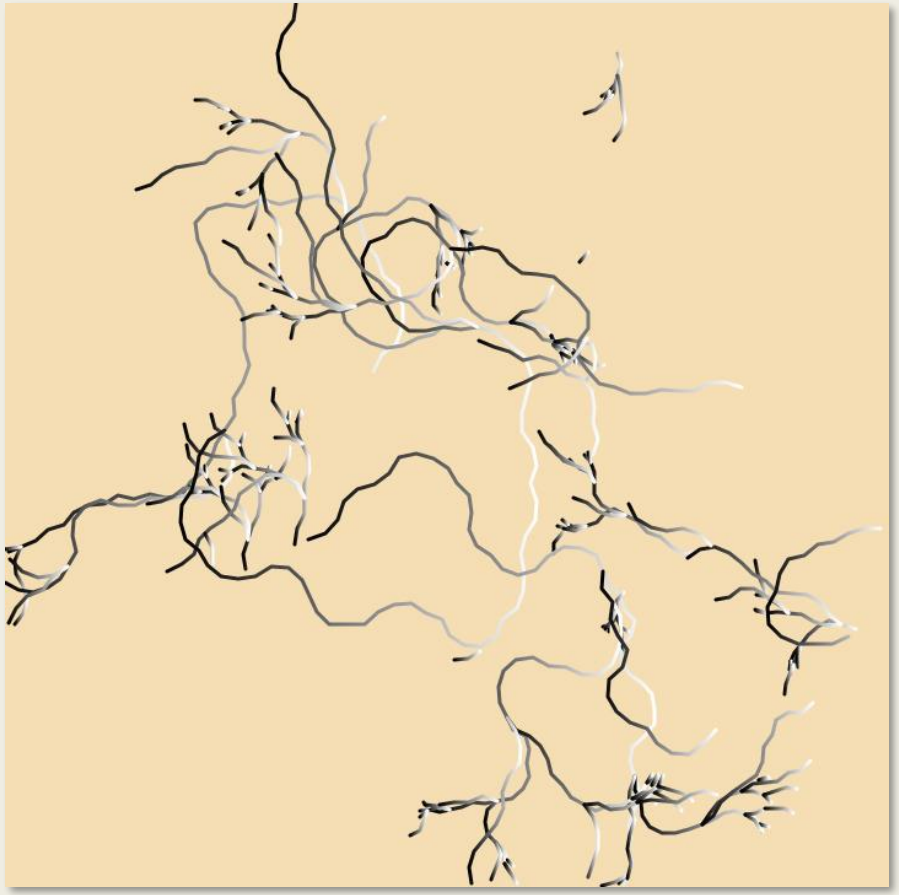
Goal: cracks and sand

Stochastic L-system to create branchings

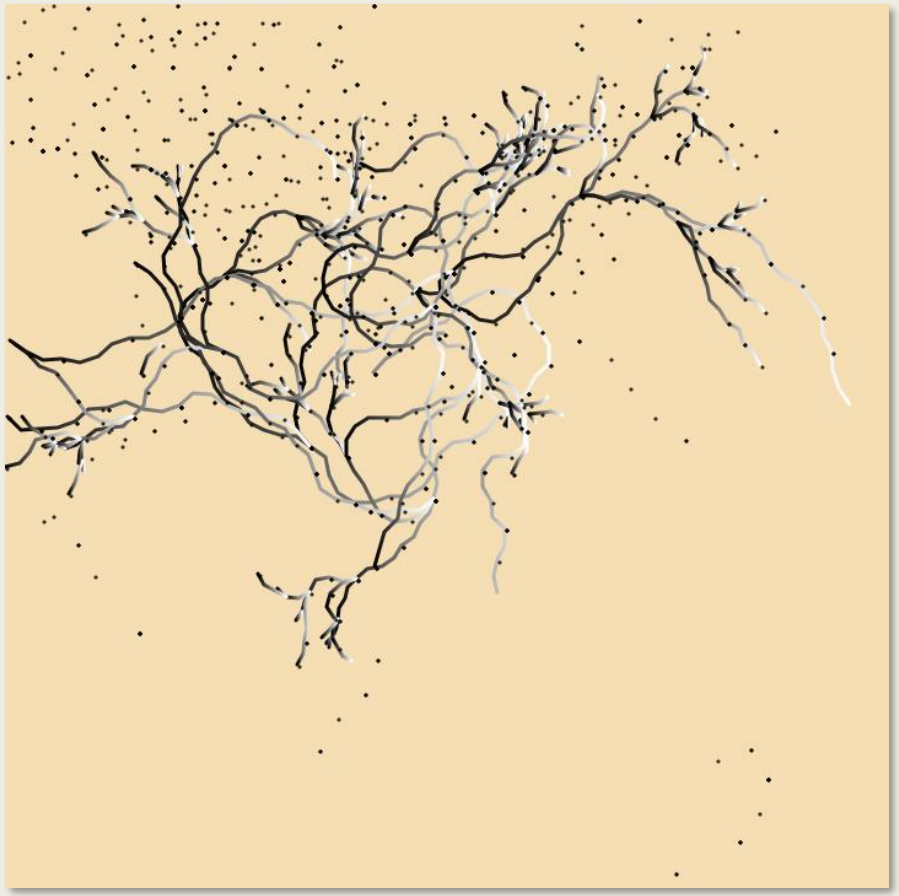
First try

Only render cracks for now

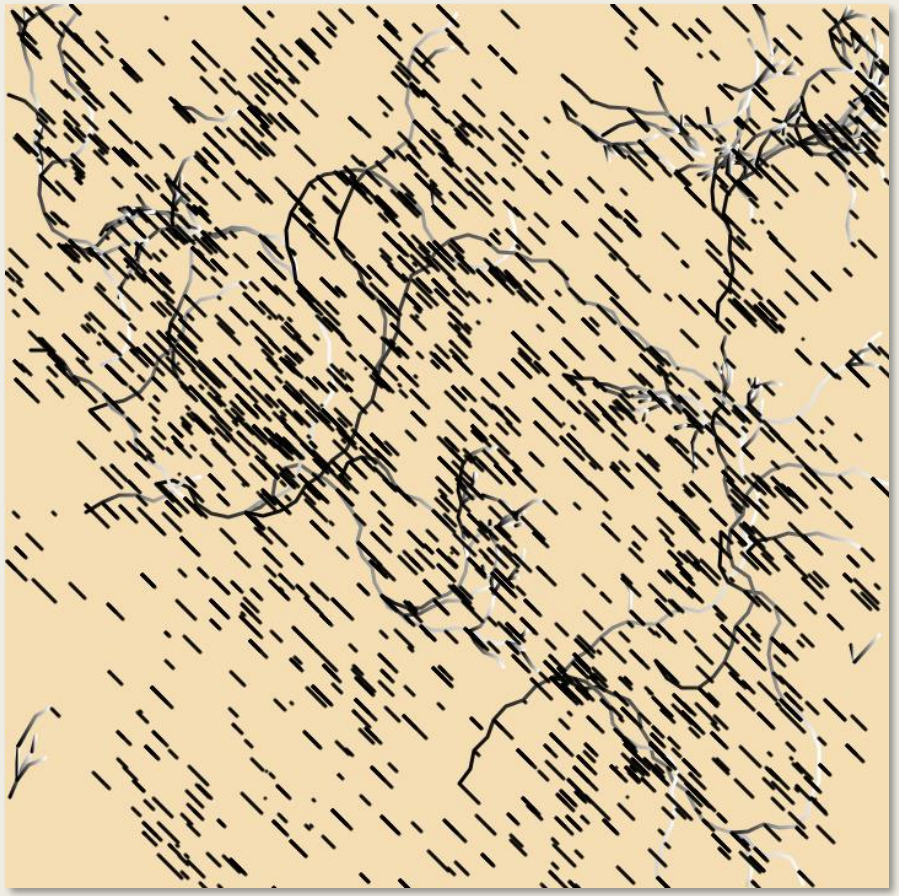
Creepy vines?



Reset and break line

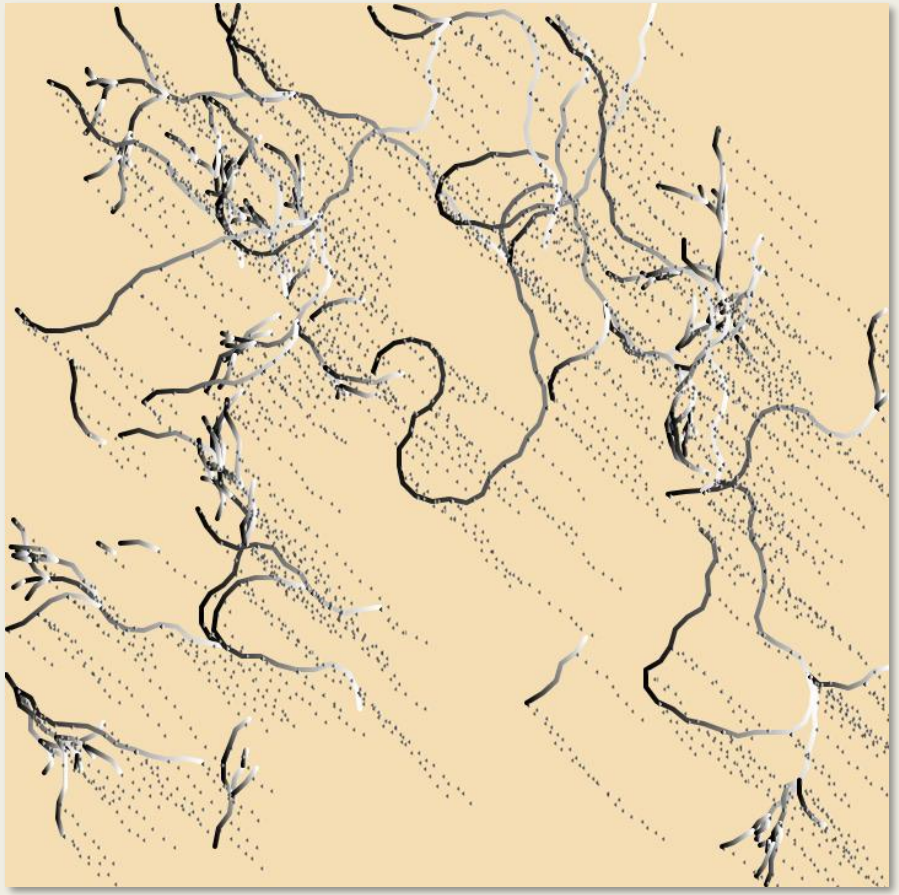


Add sand

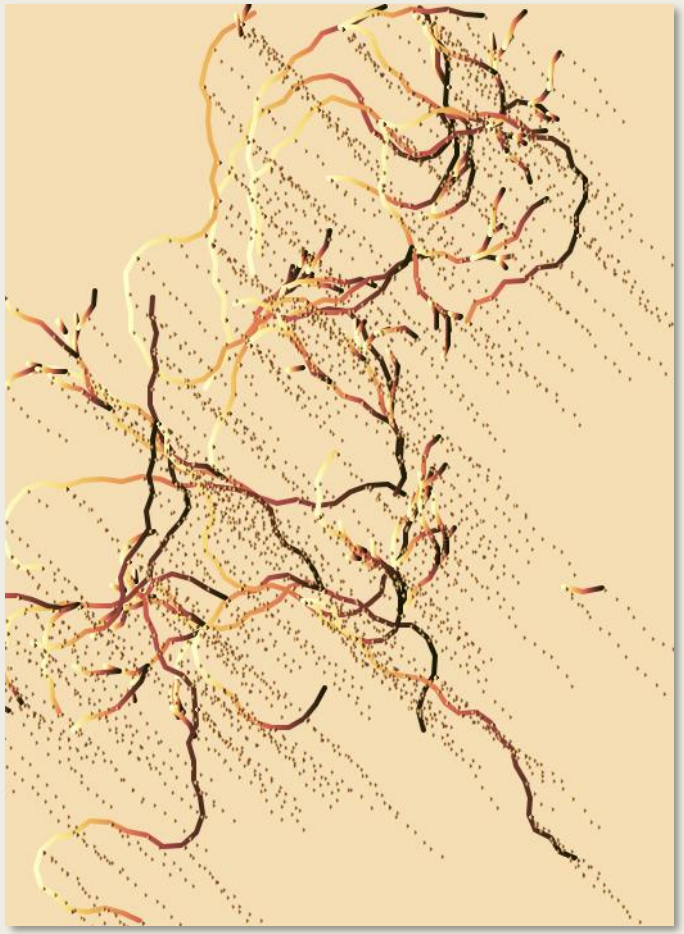


Drifts?

Maybe for some other project

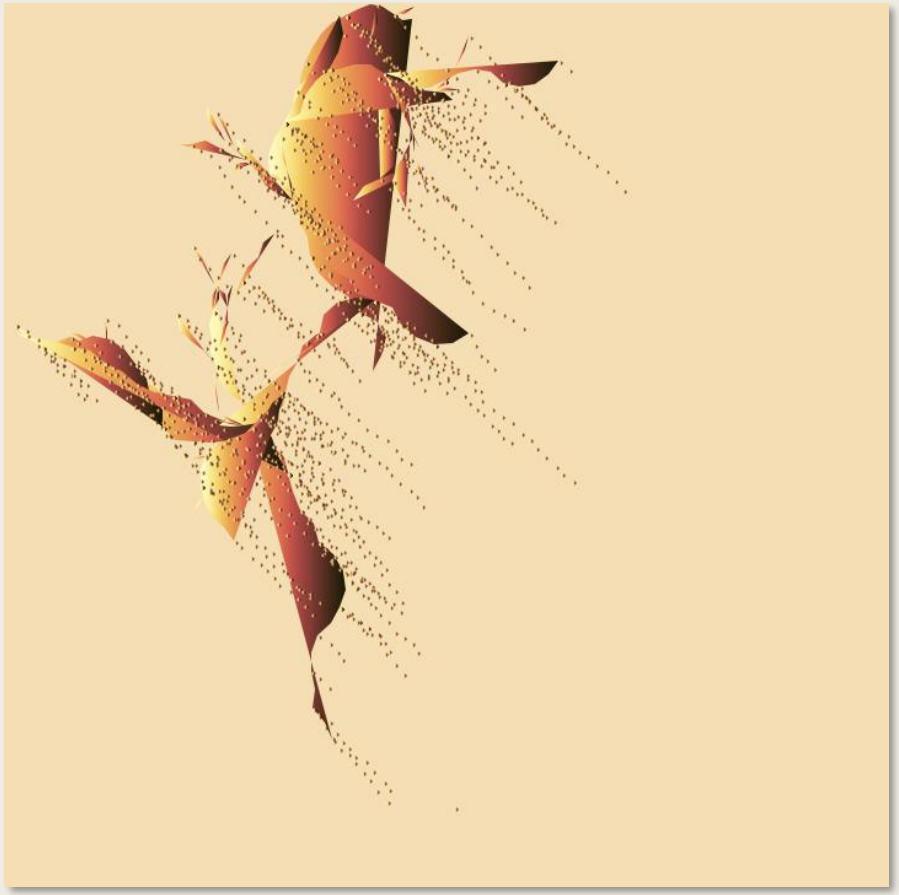


Jittering dot lines



Styling

Sizes, colours, fills,...

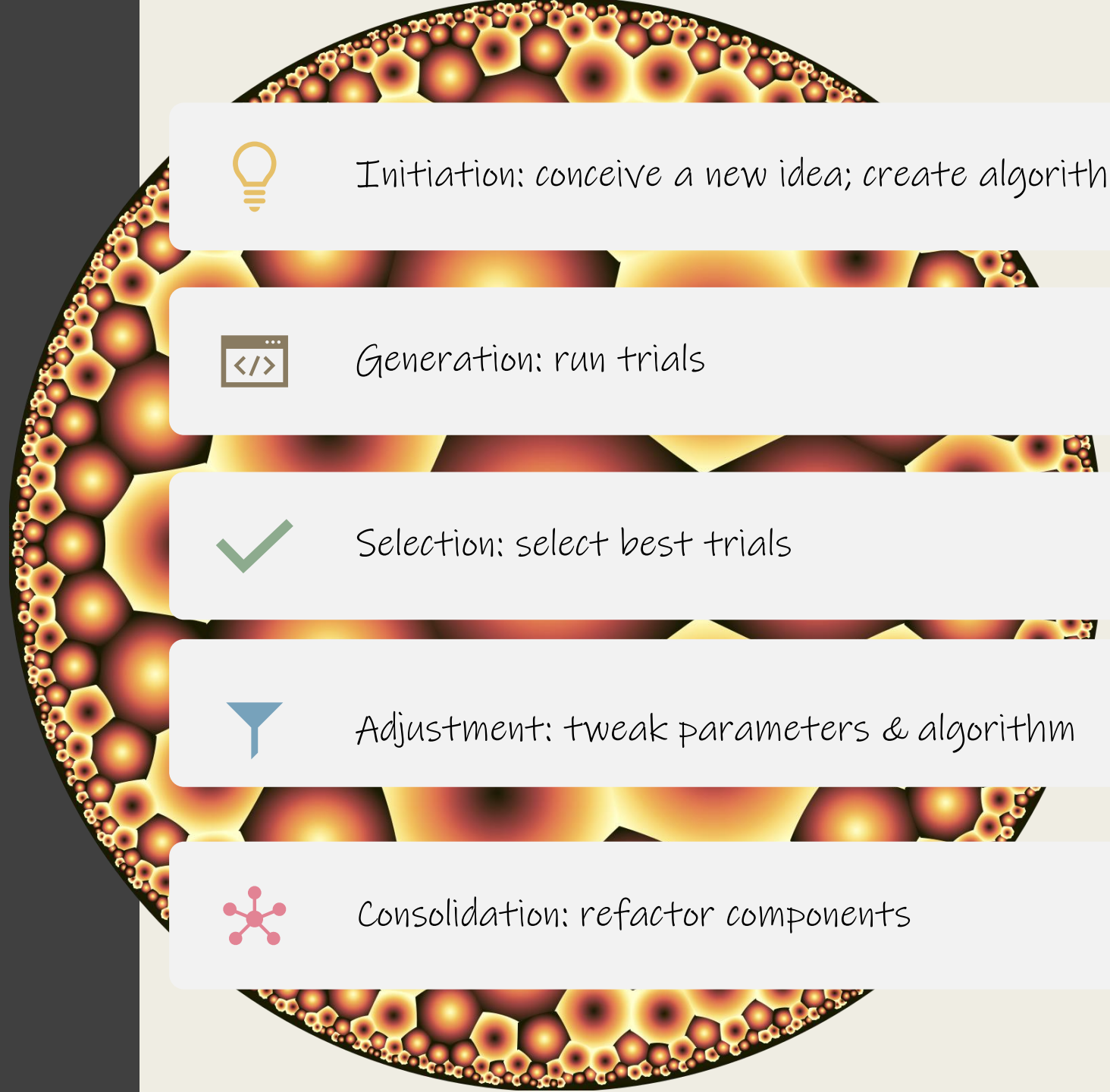


Consolidation

Component for jittered lines

Encapsulate crack system

Cycle of Experimentation



- Initiation: conceive a new idea; create algorithm
- Generation: run trials
- Selection: select best trials
- Adjustment: tweak parameters & algorithm
- Consolidation: refactor components

- View in everyday tools (browser & Emacs)
- View source works
- Hand editing works
- Full XML toolchain
- Powerful capabilities
- Annoyingly low level in places

SVG is Low-Level

```
<svg:svg width="720px" height="720px" viewBox="0 0 720 720"...>
<svg:defs>…</svg:defs>
<svg:g class="background">
 <svg:g transform="translate(6,0)">
  <svg:g transform="translate(0,0)"><svg:g class="BlueSwoosh" stroke-width="4">
   <svg:path d="m 0 5 L -5 20 m 0 5 L 5 20"/>
  </svg:g></svg:g>
  <svg:g transform="translate(0,30)"><svg:g class="BlueSwoosh" stroke-width="4">
   <svg:path d="m 0 5 L -5 13 m 0 5 L 5 13"/>
  </svg:g></svg:g>
  <svg:g transform="translate(0,53)"><svg:g class="BlueSwoosh" stroke-width="4">
   <svg:path d="m 0 5 L -5 15 m 0 5 L 5 15"/>
  </svg:g></svg:g>
  <svg:g transform="translate(0,78)"><svg:g class="BlueSwoosh" stroke-width="4">
```

XQuery for algorithms

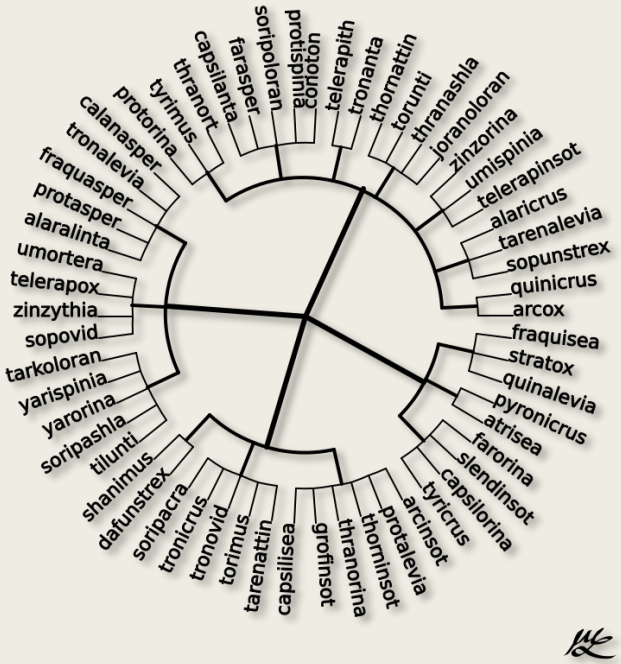
XSLT to transform to SVG

Powerful capabilities

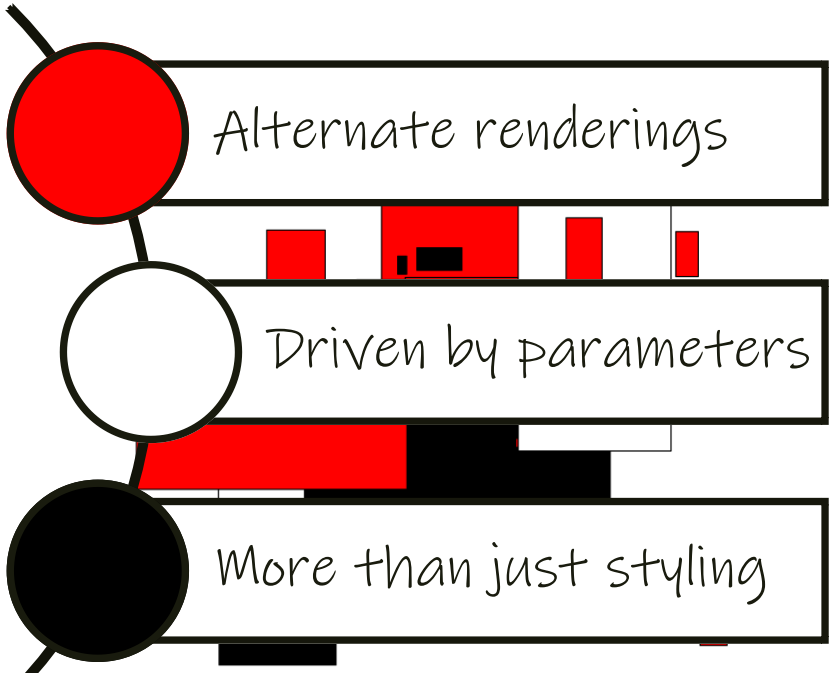
Limited libraries

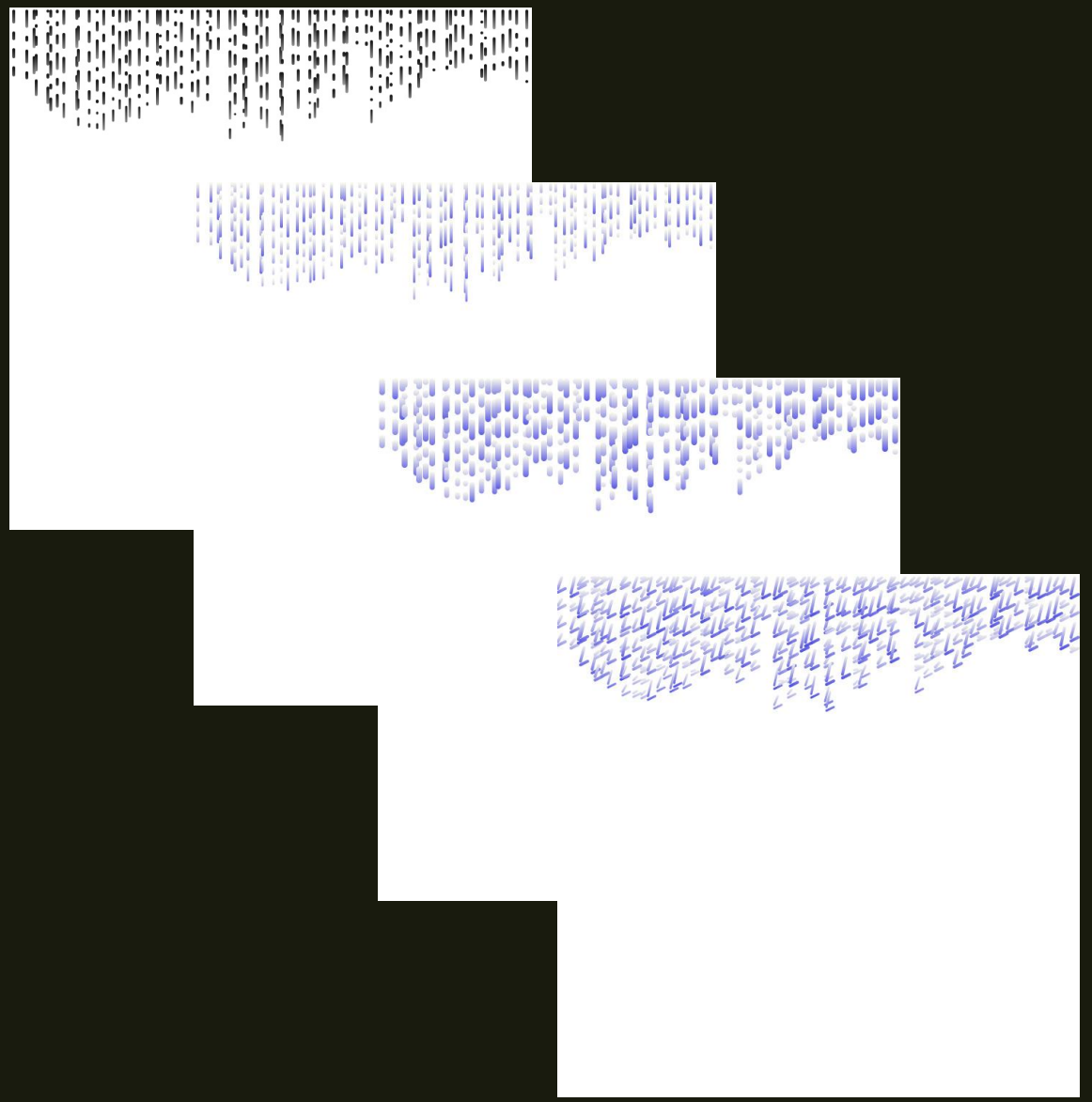
Domain-specific Language

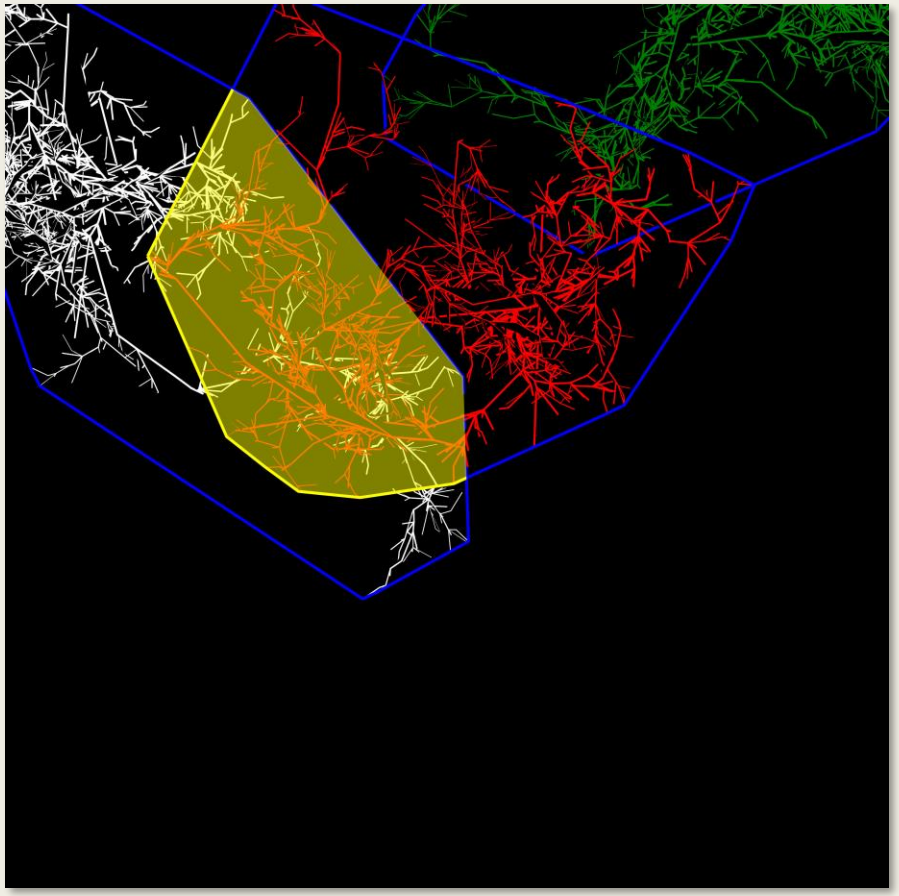
```
<art:canvas width="720" height="720" ...
 <art:thread x="6">
  <art:drop y="0" length="20"/>
  <art:drop y="25" length="13"/>
  <art:drop y="43" length="15"/>
  <art:drop y="63" length="15"/>
 </art:thread>
 <art:thread x="24">
  <art:drop y="0" length="26"/>
  <art:drop y="31" length="9"/>
  <art:drop y="45" length="20"/>
  <art:drop y="70" length="12"/>
 </art:thread>
 <art:thread x="34">
  <art:drop y="0" length="15"/>
  <art:drop y="20" length="29"/>
  <art:drop y="54" length="26"/>
 </art:thread>
```



XSLT


Alternate renderings
Driven by parameters
More than just styling

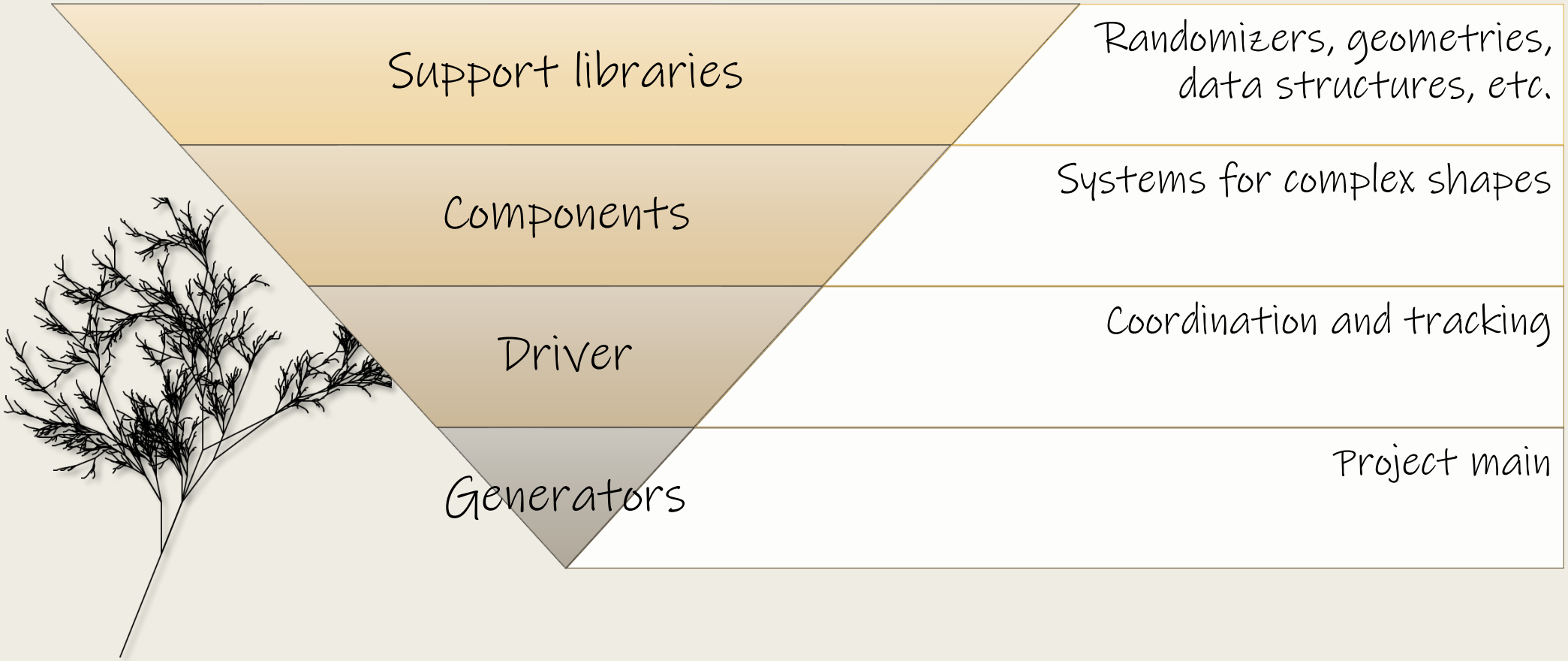




Debugging

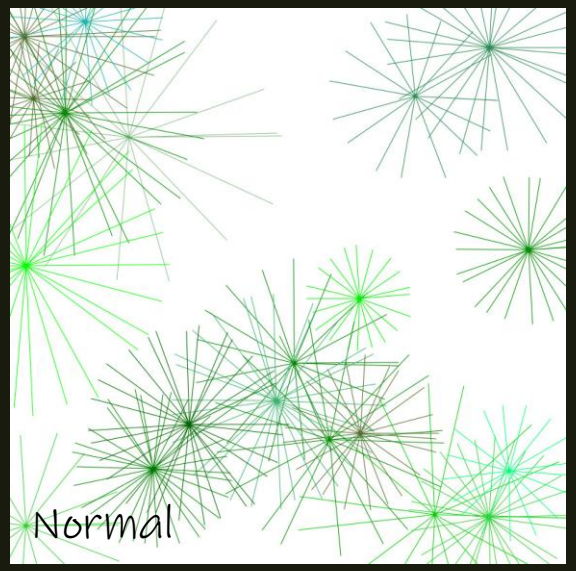
One kind of alternative rendering

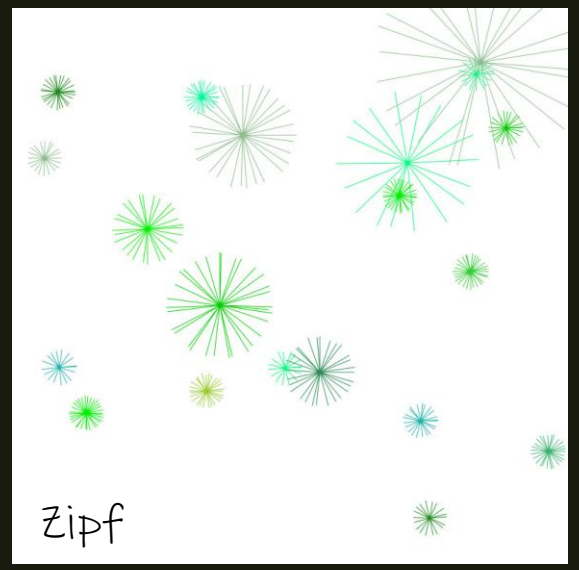
XQuery Stack

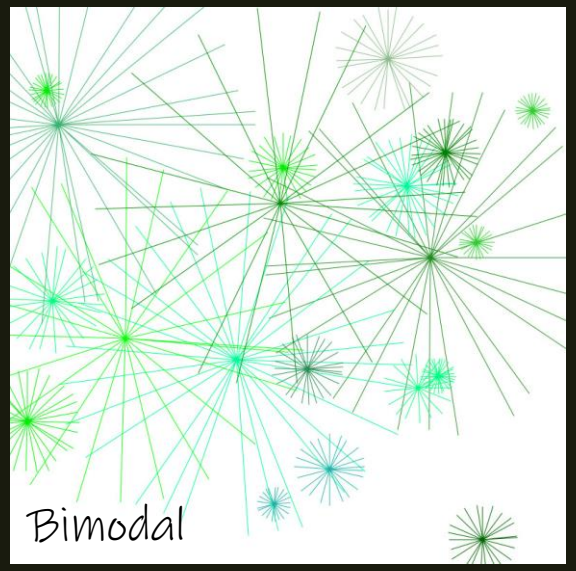


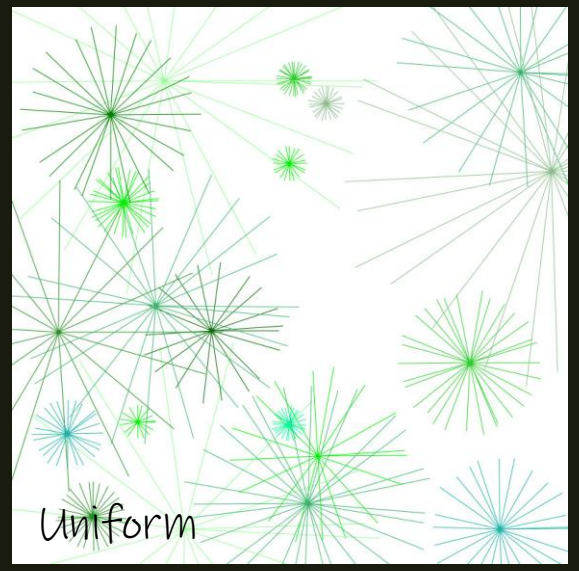
Randomizers

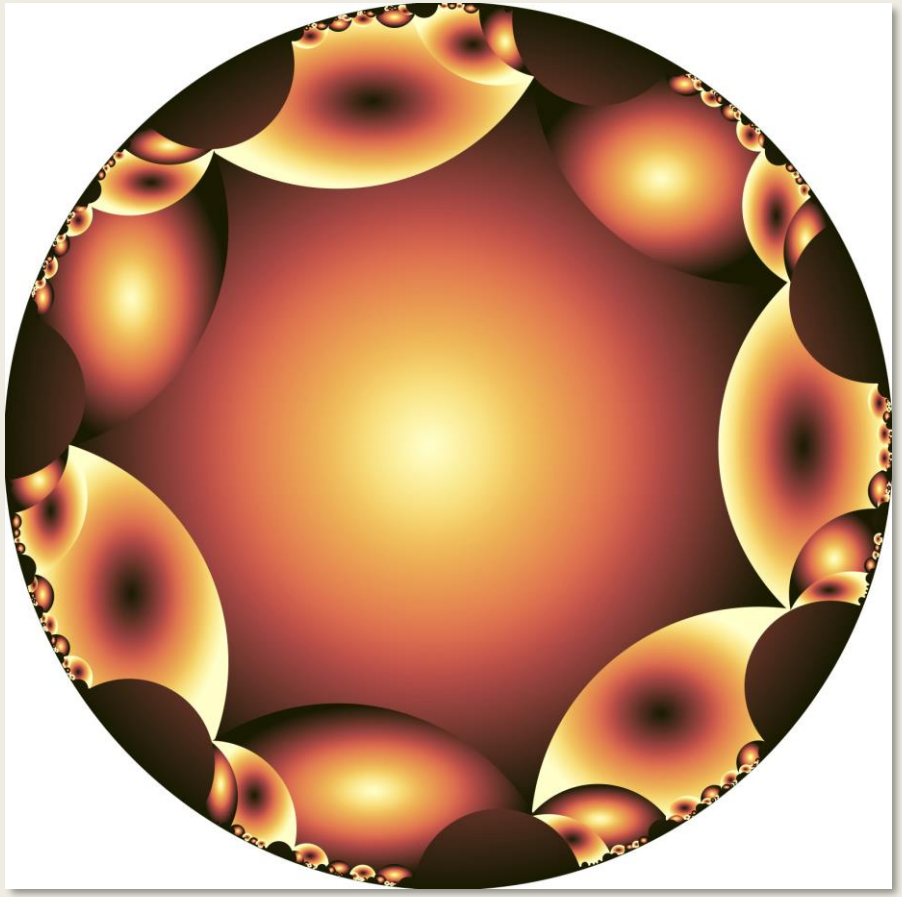
- Different distribution = different aesthetic effect
- API on top of xdmp:random











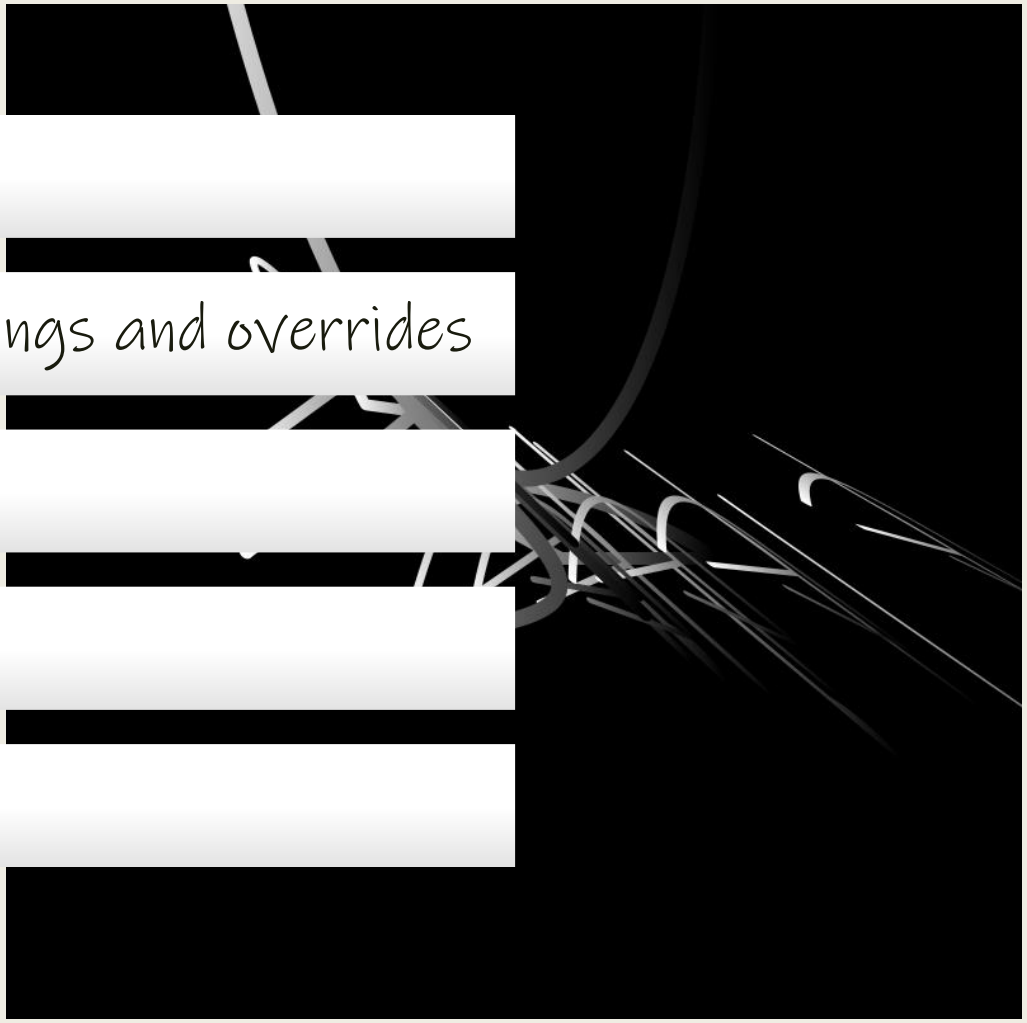
Geometry

Regions, affine transformations, tangent angles, convex hulls, point at angle, region overlap,...

Components

L-systems, iterated functions, shape families,...

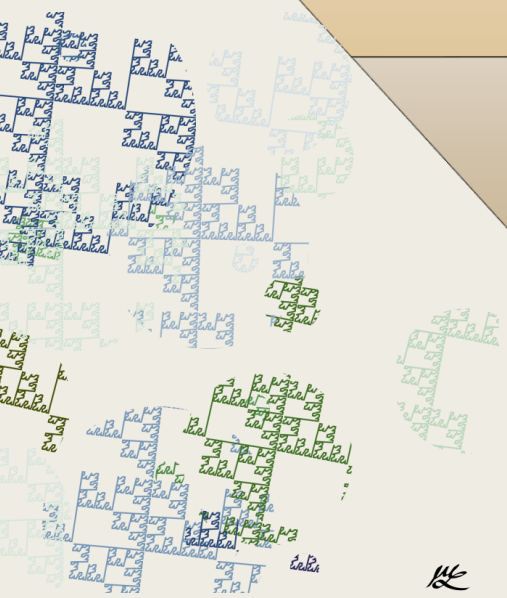
Anatomy of a Generator



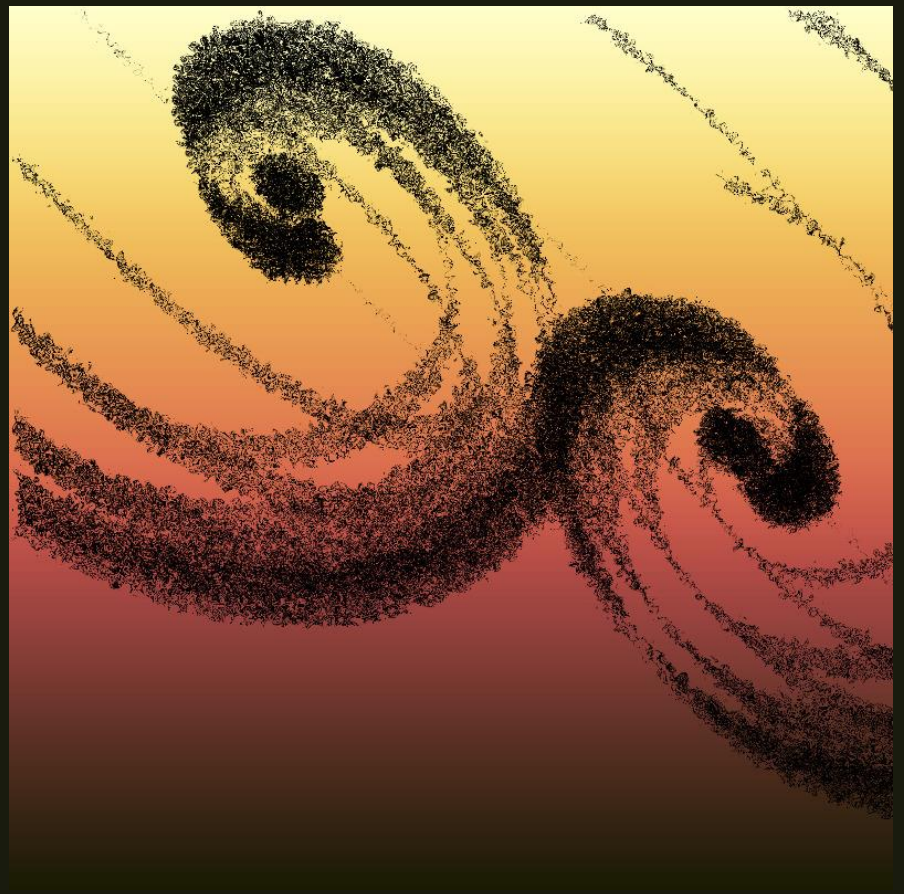
- Default parameters
- External variables for settings and overrides
- Metadata function
- Content function
- Call driver

```
declare function this:content()
{
  let $array := path:array($SLOT, $CANVAS)
  let $num-lines := path:lines-per-canvas($SLOT, $CANVAS)
  let $num-slots := path:slots-per-line($SLOT, $CANVAS)
  for $slot in 1 to $num-slots return (
    <art:thread x="{geom:x($array[$slot])}" draw="line"/>,
    for $line in 1 to rand:randomize($COINS)
    let $point := $array[($line - 1) * $num-slots + $slot]
    let $center := geom:translate($point, 0, rand:randomize($JITTER))
    let $scale :=
      for $s in rand:randomize(4, (), $SCALE)
      order by $s descending
      return $s
    let $colours := rand:randomize(2, (), $COLOUR-INDEX)
    return (
      <art:coin x="{geom:x($center)}" y="{geom:y($center)}"
                top="{$colours[1]}" base="{$colours[2]}"
                sxtop="{$scale[4]}" sytop="{$scale[3]}"
                sxbase="{$scale[2]}" sybase="{$scale[1]}"/>,
      <art:dot x="{geom:x($center)}" y="{geom:y($center)}"/>
    )
  )
};
```

XSLT Stack

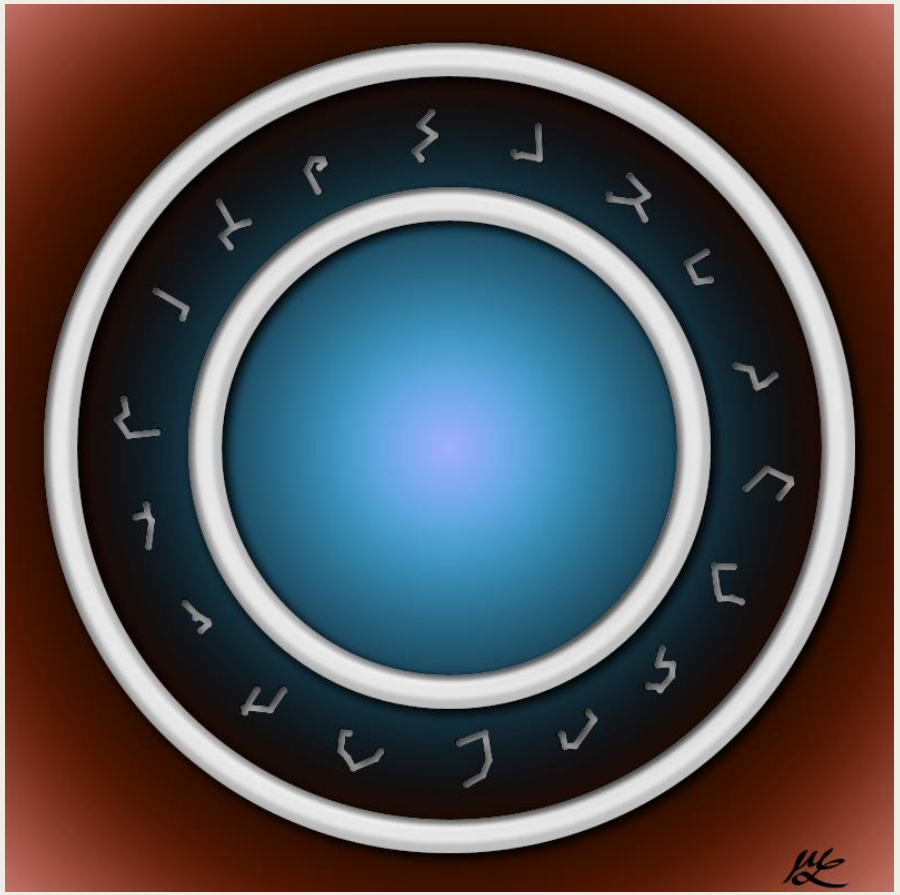


Support libraries
Named stylings
Core rendering
Standard parameters & templates
Components
Complex shapes
Renderers
Customization layer



Styling

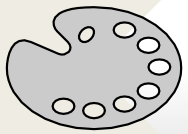
Colours, gradients, effects



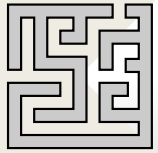
Components

Special rendering rules

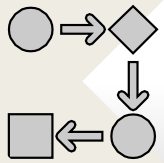
Anatomy of a Renderer



Import default rendering layer



Import complex components



Overrides

Custom parameters & templates

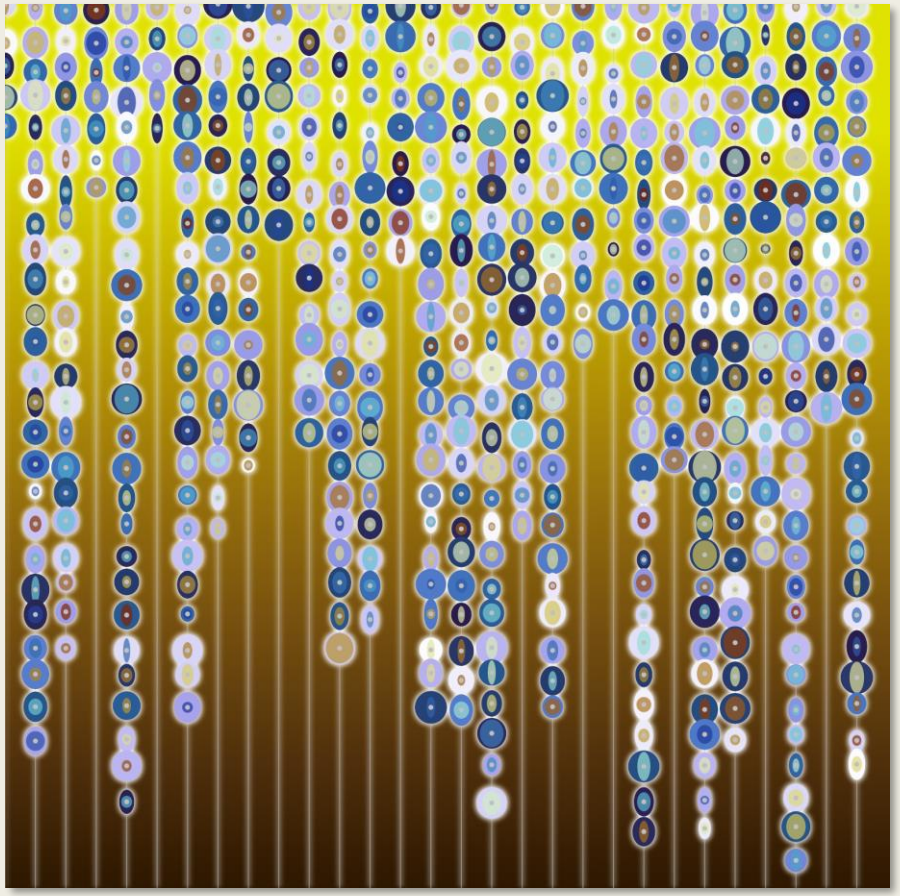
```
<xsl:import href="render.xsl"/>

<xsl:param name="body-filter">white-glow</xsl:param>

<!-- Project-specific parameters -->
<xsl:param name="base-coin">devon</xsl:param>

...

<xsl:template match="art:coin">
  <xsl:variable name="top" select="art:fill(concat('stop-',@top,'-',$top-coin))"/>
  <xsl:variable name="base" select="art:fill(concat('stop-',@base,'-',$base-coin))"/>
  <svg:ellipse cx="{@x}" cy="{@y}"
               rx="{$coin-size*@sxbase}" ry="{$coin-size*@sybase}"
               fill="{$base}" fill-opacity="1"/>
  <svg:ellipse cx="{@x}" cy="{@y}"
               rx="{$coin-size*$coin-ratio*@sxtop}" ry="{$coin-size*$coin-ratio*@sytop}"
               fill="{$top}" fill-opacity="0.7"/>
</xsl:template>
```



Glowing coins

Tracking

When was it made?

What is it?

What randomizers?

What rules?

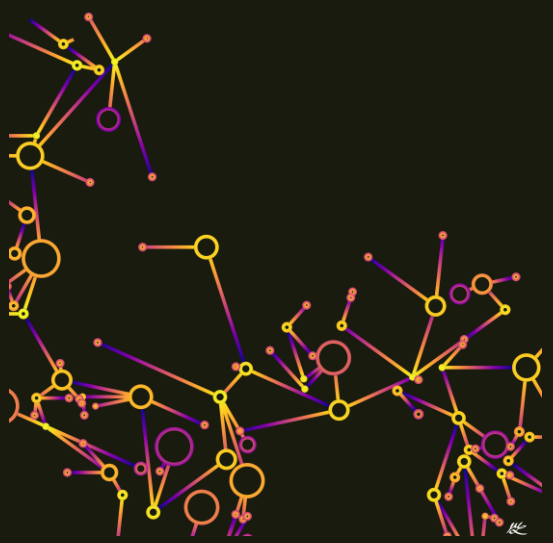
What parameters?

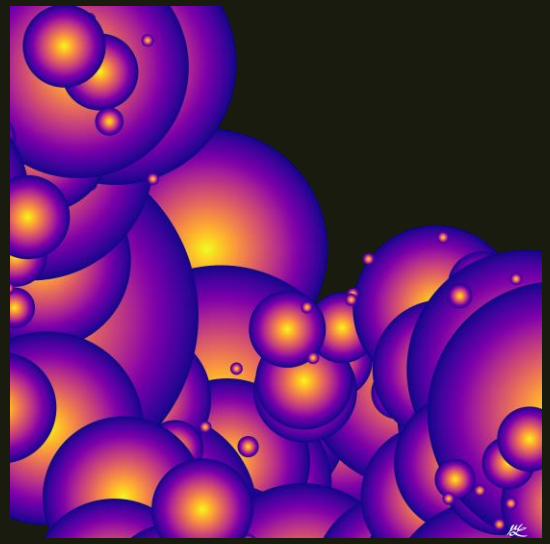
What styling?

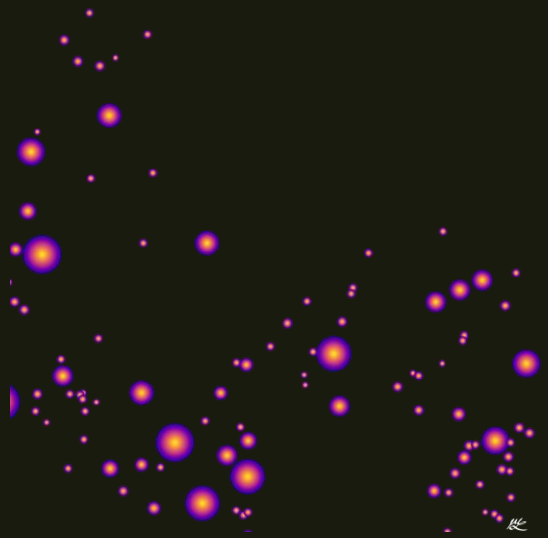
Embedded metadata

- Metadata function creates metadata
- Driver augment and emits
- Customization layer creates metadata
- Rendering layer augments and copies
- Metadata embedded in SVG

```
<art:metadata>
  <art:description>coins: hanging twisted circles</art:description>
  <art:created>2020-05-18T10:23:41.812141-07:00</art:created>
  <art:parameter name="coin-size">25</art:parameter>
  <art:parameter name="resolution">medium</art:parameter>
  <art:algorithm name="coins" distribution="uniform" min="5" max="29" cast="integer"/>
  <art:algorithm name="jitter" distribution="uniform" min="0" max="16" cast="integer"/>
  <art:algorithm name="colour-index" distribution="uniform" min="1" max="512" cast="integer"/>
  <art:algorithm name="scale" distribution="uniform" min="0.25" max="1.2"/>
  <art:processed>2020-05-18T10:23:50.091207-07:00</art:processed>
  <art:description>ochres: ochres gradient background</art:description>
  <art:stylesheet-parameters>
    <art:parameter name="coin-size">25</art:parameter>
    <art:parameter name="coin-ratio">25</art:parameter>
    <art:parameter name="base-coin">devon</art:parameter>
    <art:parameter name="top-coin">roma</art:parameter>
    <art:parameter name="palettes">basic-grey simpleIce-invert</art:parameter>
    <art:parameter name="effects">white-glow</art:parameter>
    <art:parameter name="stroke-class">basic-grey</art:parameter>
    <art:parameter name="default-stroke-width">3</art:parameter>
    <art:parameter name="max-stroke-width">4</art:parameter>
    <art:parameter name="background-fill">ochres-invert</art:parameter>
    <art:parameter name="body-filter">white-glow</art:parameter>
    <art:parameter name="dot-shape">circle</art:parameter>
    <art:parameter name="dot-fill">grey</art:parameter>
    <art:parameter name="dot-size">3</art:parameter>
  </art:stylesheet-parameters>
</art:metadata>
```

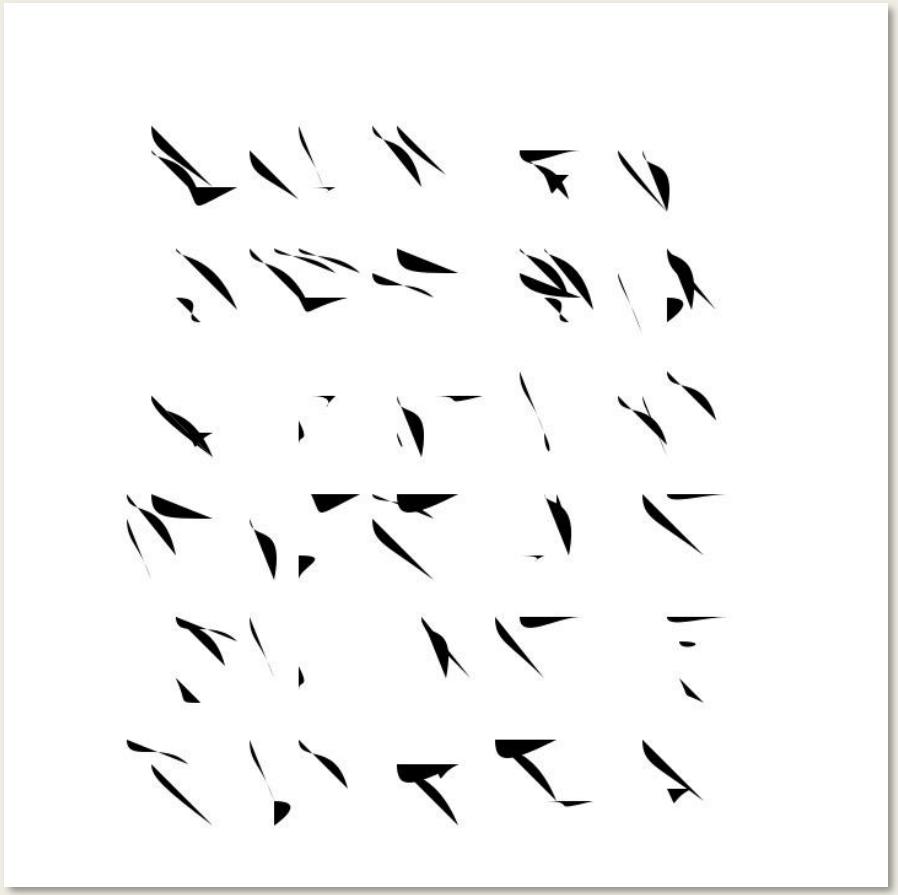






Theme and Variations

Recombining components: glyphs, paths, slots, background

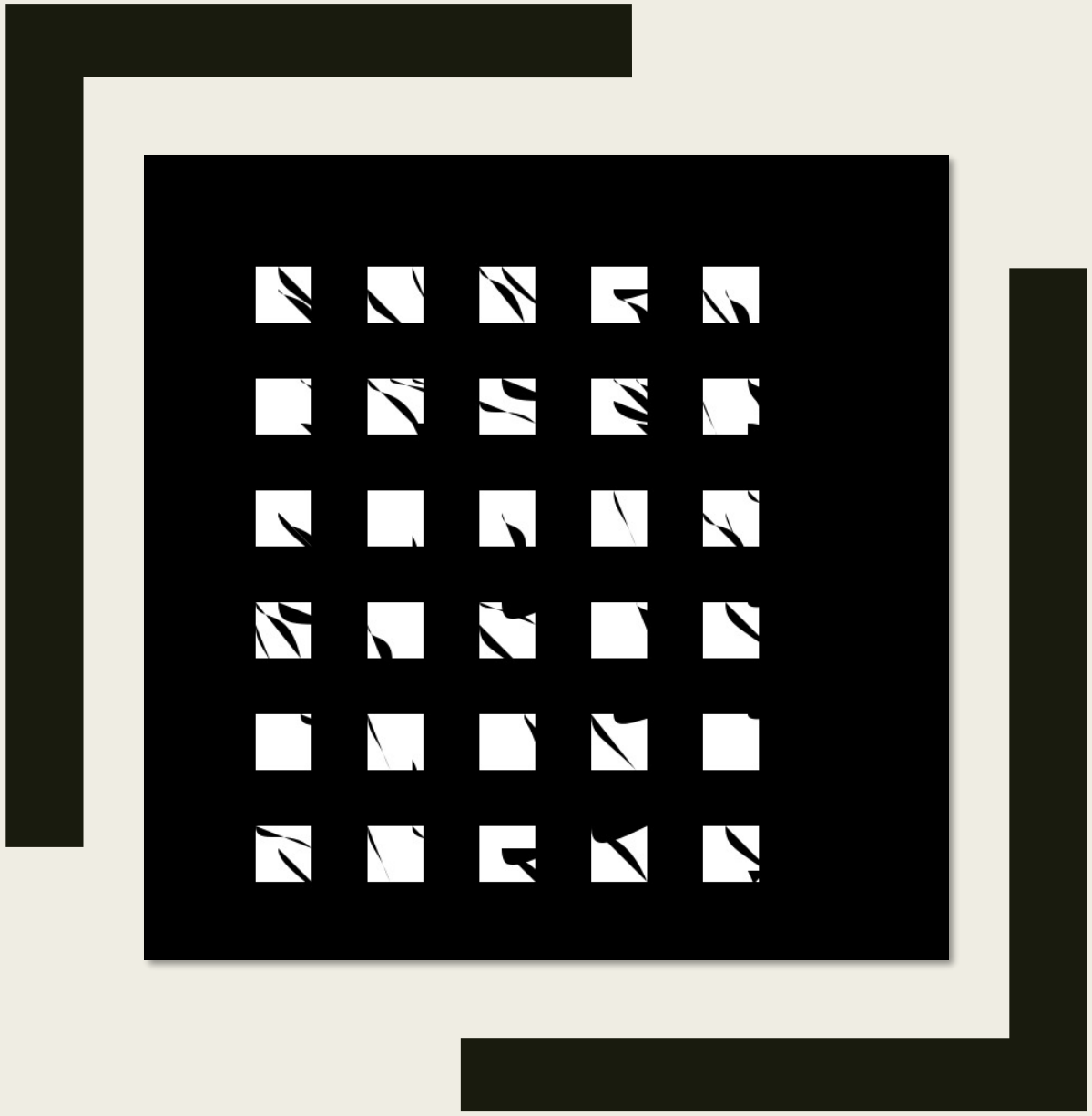


Array of glyphs

Glyph shape

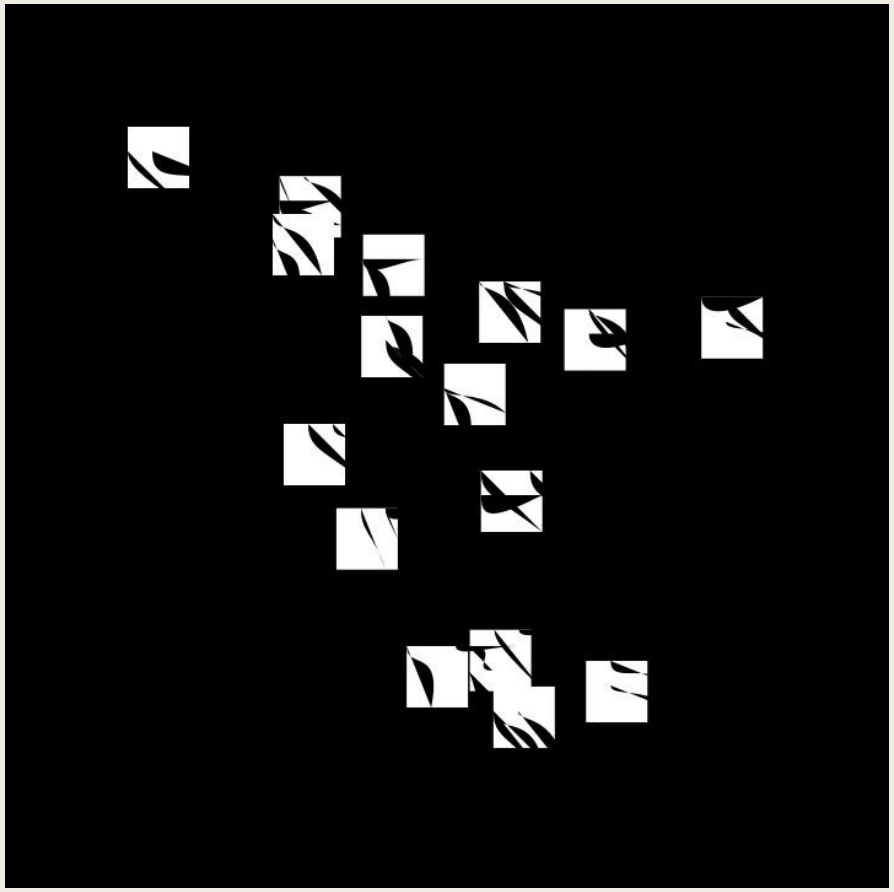
Array path

Invisible slot



Latticing

Latticed slot rendering



Meandering lattice

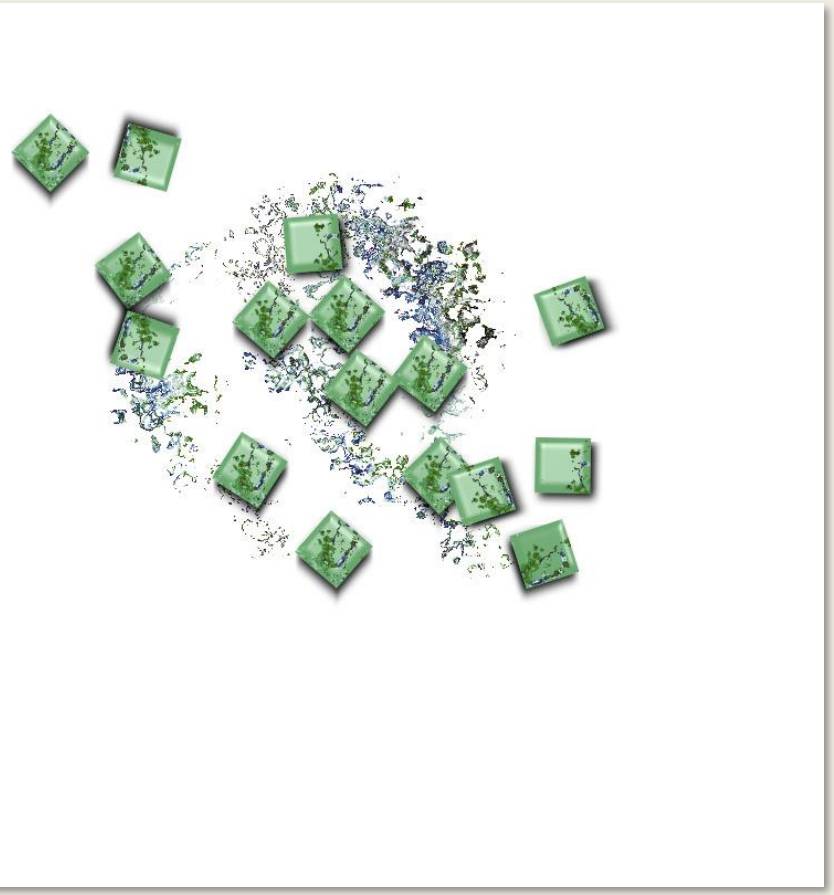
Meander path

Circle

Circle path

Slot scaling and rotation

Gradient background



Textured

Drop shadow on slot

Splotch on glyph

Splotch background spiral

- Yes, you can use the XML stack to make art
- It takes work
- Build up hierarchical palette of named things
- Domain specific language
- Automatic metadata embedding

Discussion