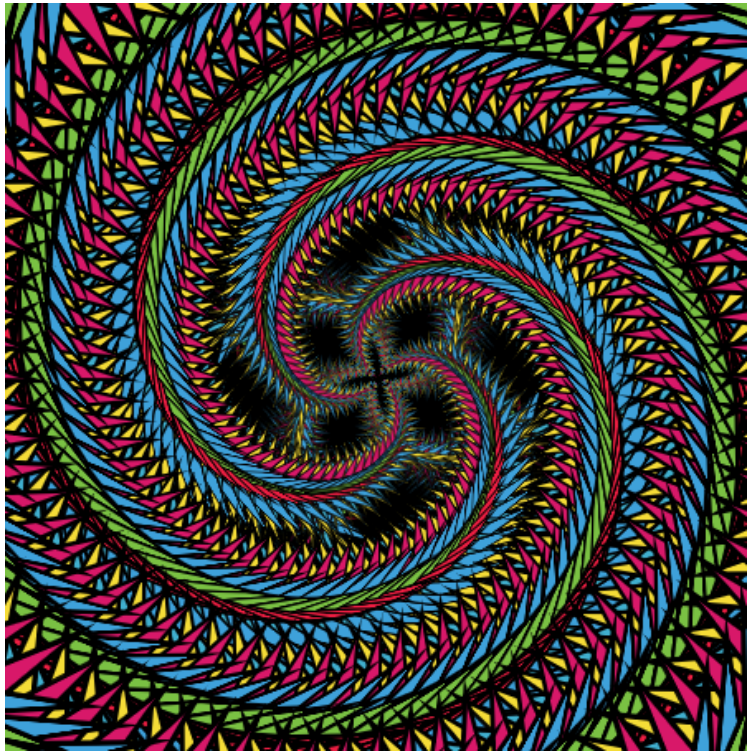
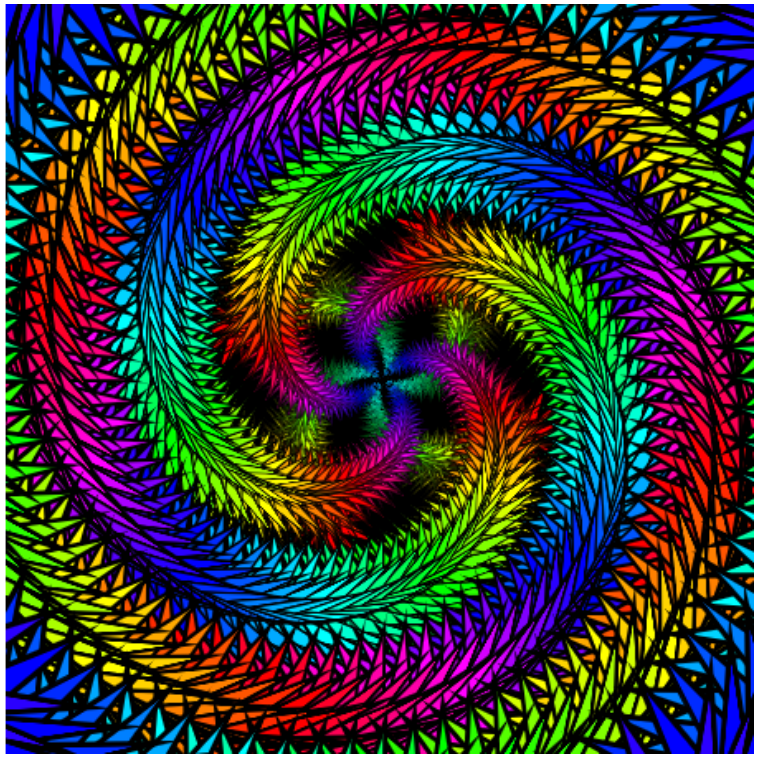


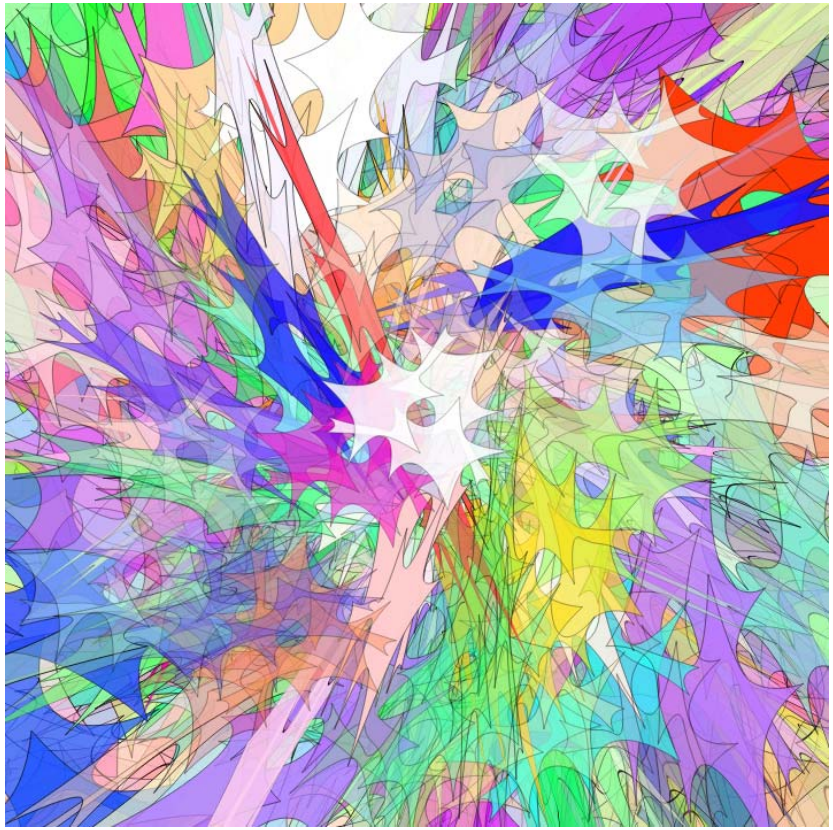
# MathArt Math

Developed by
Paul G Phillips
And
Shandley K Phillips

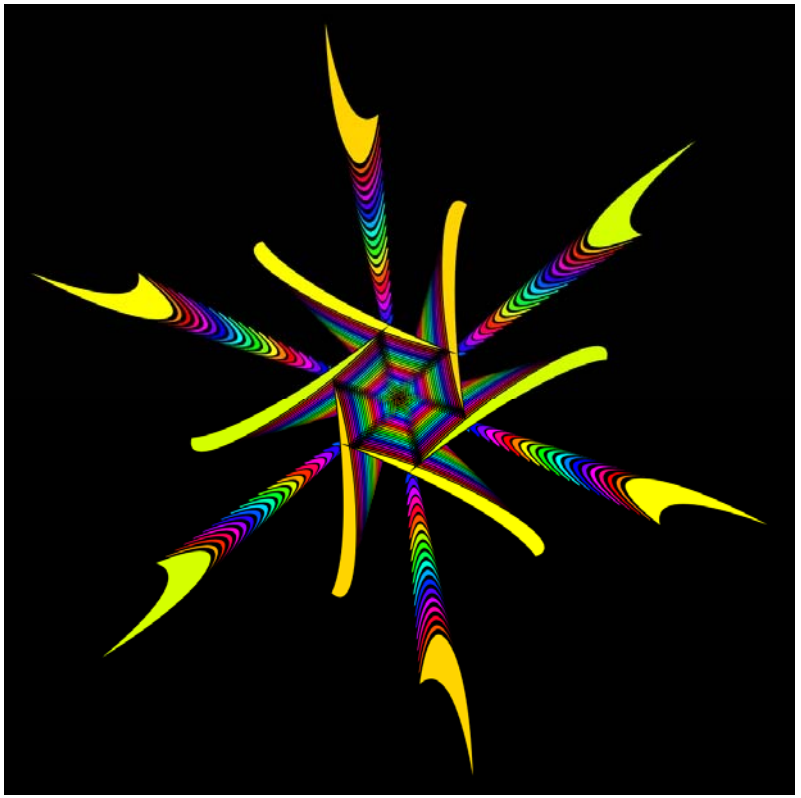


# Why Program?

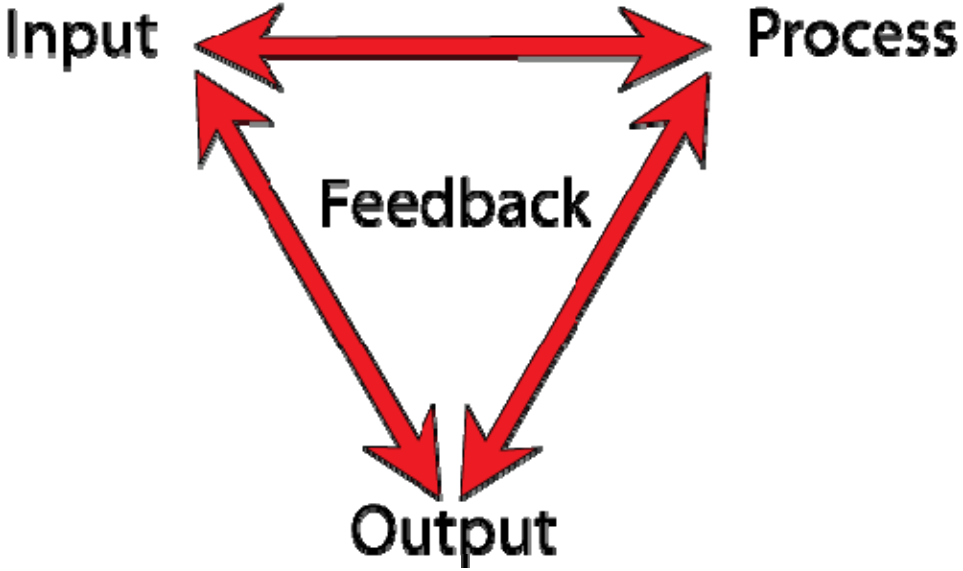
- Because I'm lazy.
- Because I do not want to do a task that is repeated over and over.
- Because what I want to do doesn't have a tool.
- Because I want to build a tool to do it my way.

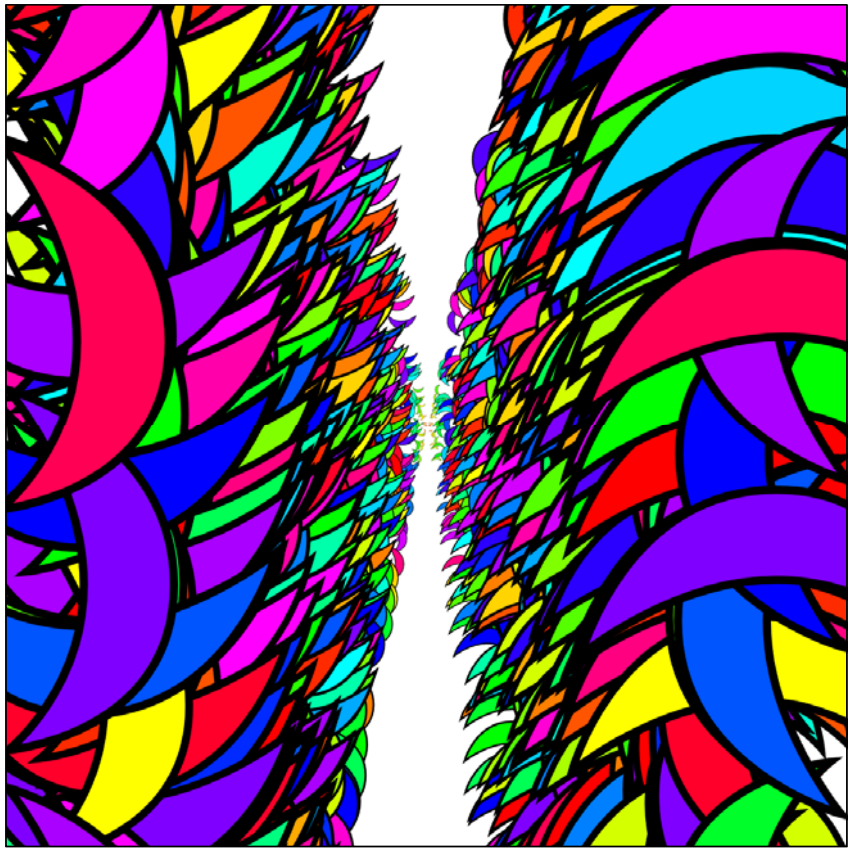


# Basic Programming Concept

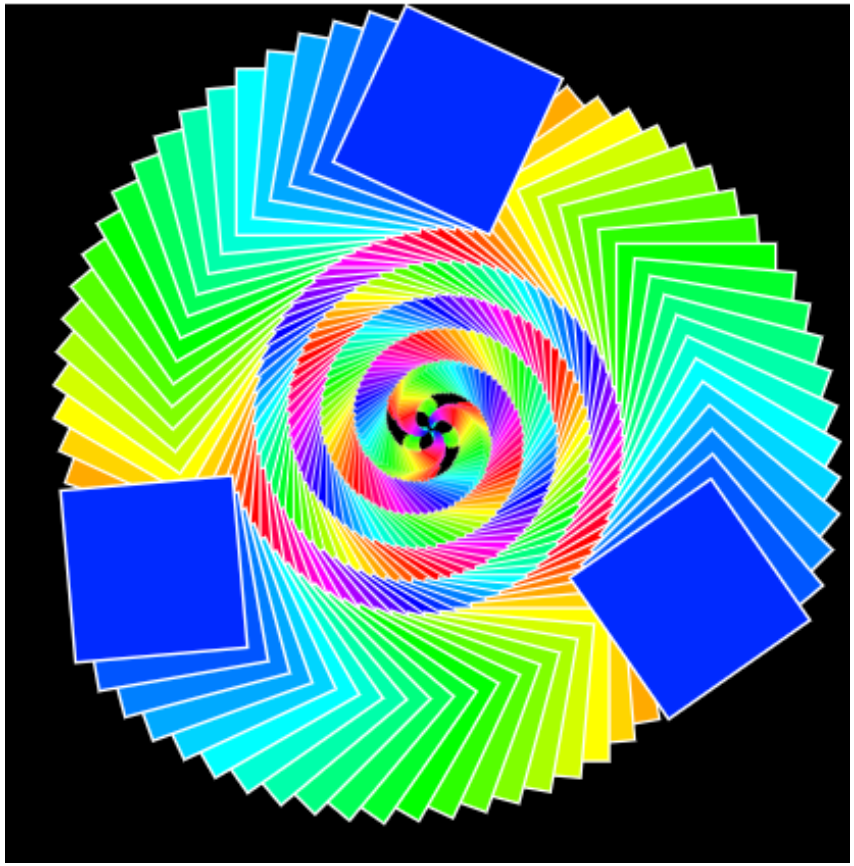
# True Programming Process



# Why I Created Math Art.

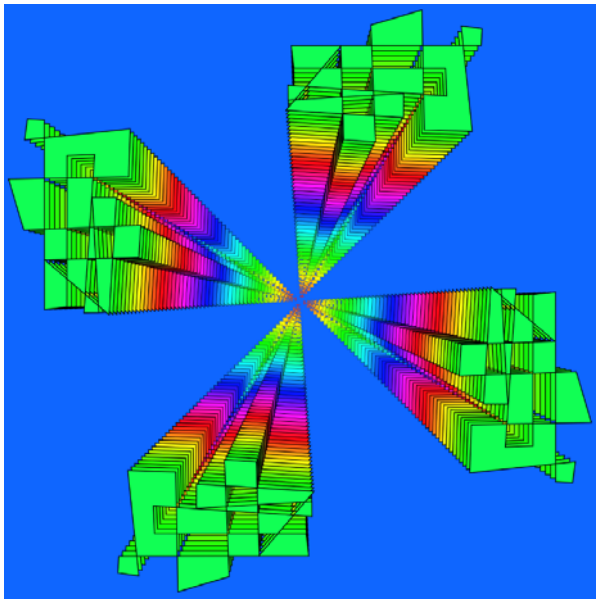
- I wanted to set a number of layers.
- I wanted to set a shape.
- I wanted to push a button and have each of the visual parameters assigned to the shape be different on each layer.
- I wanted to vary:
  - Rotate
  - Scale
  - Skew X
  - Skew Y
  - Horizontal
  - Vertical
  - Hue



# Scalable Vector Graphics (SVG)

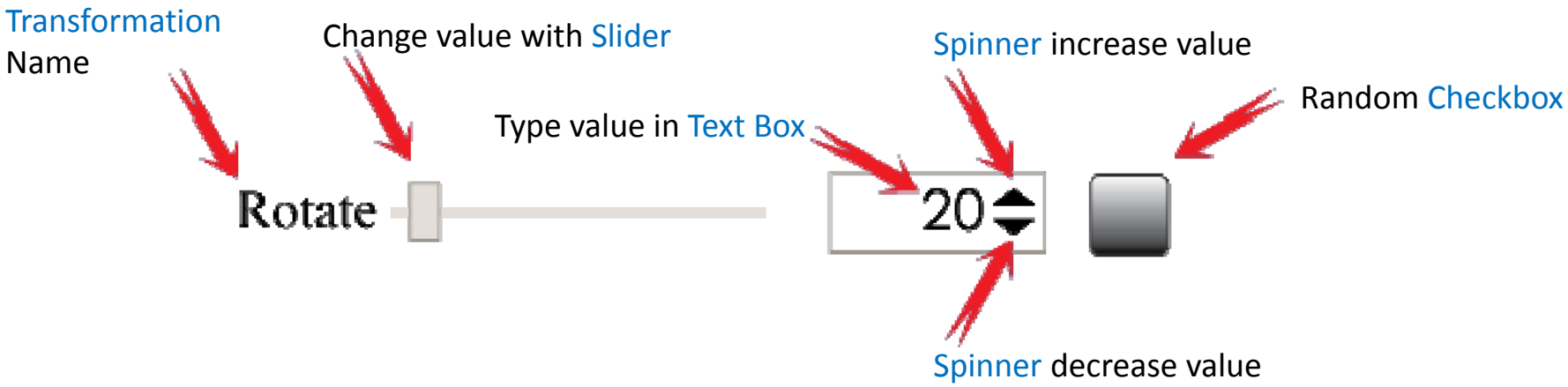
Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. This programming language will do the task I wanted.

- Object primitives code
  - <circle>
  - <rect>
  - <ellipse>
  - <line>
  - <polygon>
  - <polyline>
  - <path>
  - <text>



# MathArt Input

- Put your choices into the application.



# Special Inputs

- Paste your input into the application.

## ArtData

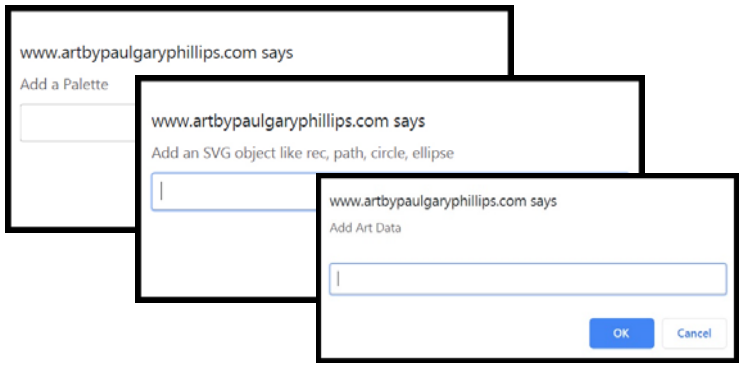
3_200_20_168_5_0.23_0_0_0_0_10_100_100_1_0_5_0_#stick_false_false_false_false_false_false_false_false_false_false_false_false

## SVG Objects

<polygon points="168.364,-95.454 110.546,16.559 170.042,127.691 45.644,107.316 -41.663,198.242 -60.727,73.636 -174.181,18.7 -61.566,-37.936 -44.378,-162.813 44.287,-73.211" fill="#5c45bc" stroke="black" stroke-width="5" />

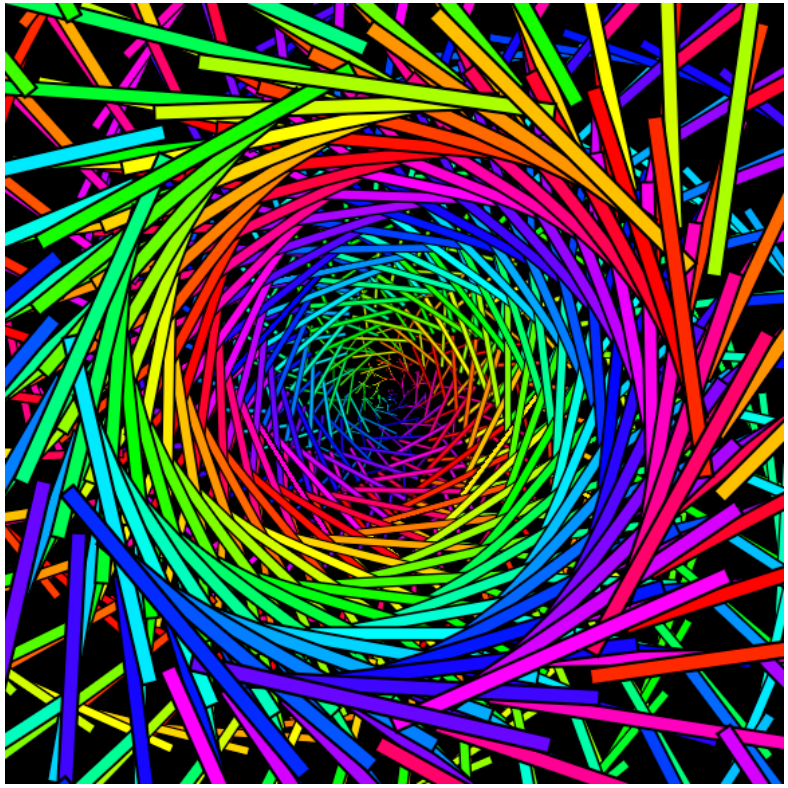
## Palettes

356,94,97,1 356,4,99,1 241,92,97,1



# Programming Requirements

1. Calculate the Math for each transformation, color, stroke.
2. Convert the math results into SVG programming code as text.
3. Construct JavaScript Functions to concatenate command strings.
4. Display the command strings on the screen as Computer Graphics



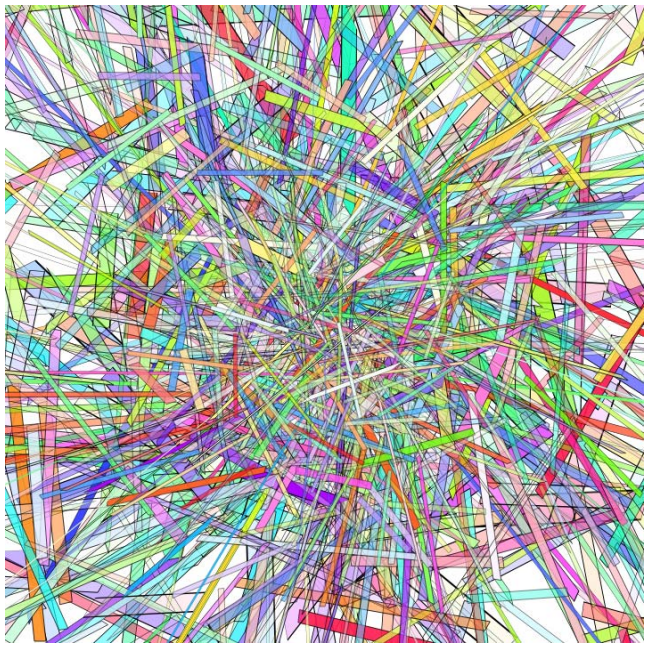
# MathArt Process

- Take SVG primitives (1-4) positioned around the Stage area center(0,0).
- Transform each layer using the values dynamically sent by the sliders to JavaScript Functions.
- Apply Rotate, Scale, SkewX, SkewY, MoveH, MoveV transformations by writing functions in JavaScript to convert the values into proper SVG code using the <use> command.

# Output Stage

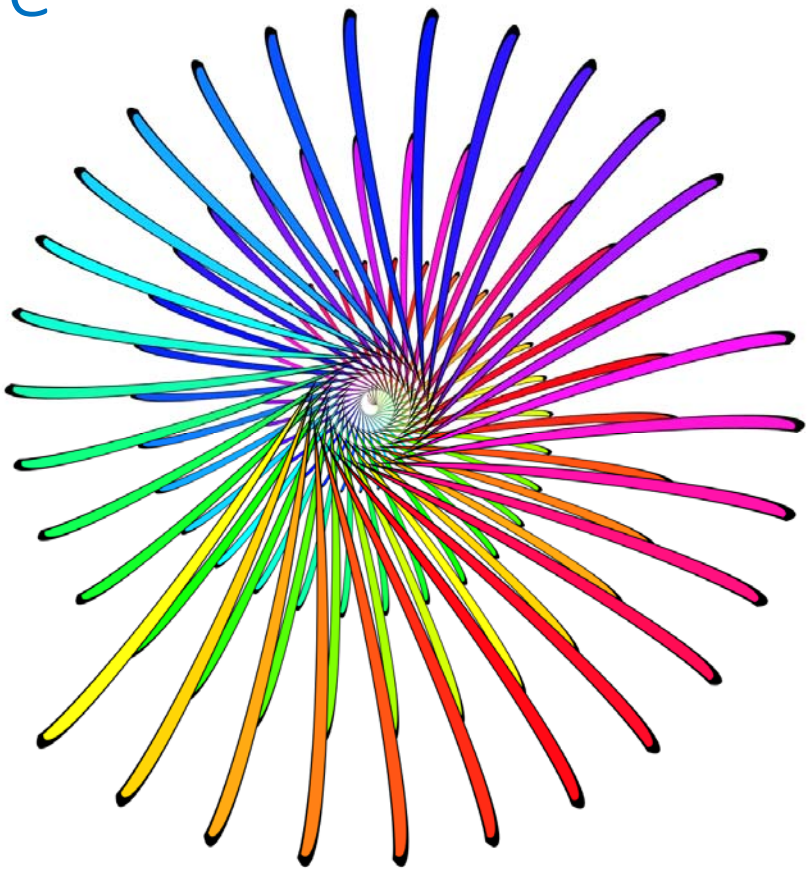
- Display the SVG commands as Computer Graphics on the screen inside an area I called the Stage In MathArt. But the HTML5 code defines a box containing the SVG area with an ID of Playspace. The Geometry coordinate system is defined in this command.

```
<div id="box"
<svg id="playspace" version="1.1"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"

x="-1000px" y="-1000px"
width="2000px" height="2000px"

viewBox="-1000 -1000 2000 2000">
```



# SVG Transformation Code

- <use
id=“layerNumber"
transform="
rotate(centerx,centery,angle)
scale(x,y)
skewX(angle)
skewY(angle)
translate(xpixels,ypixelx)"
xlink:href="#PrimitiveShapes">
- </use>



# Calculate the Math

The Input devices on the right create values.
The program assigns the values to variables.

Some variables are relative, applying the same number added to the previous parameter.

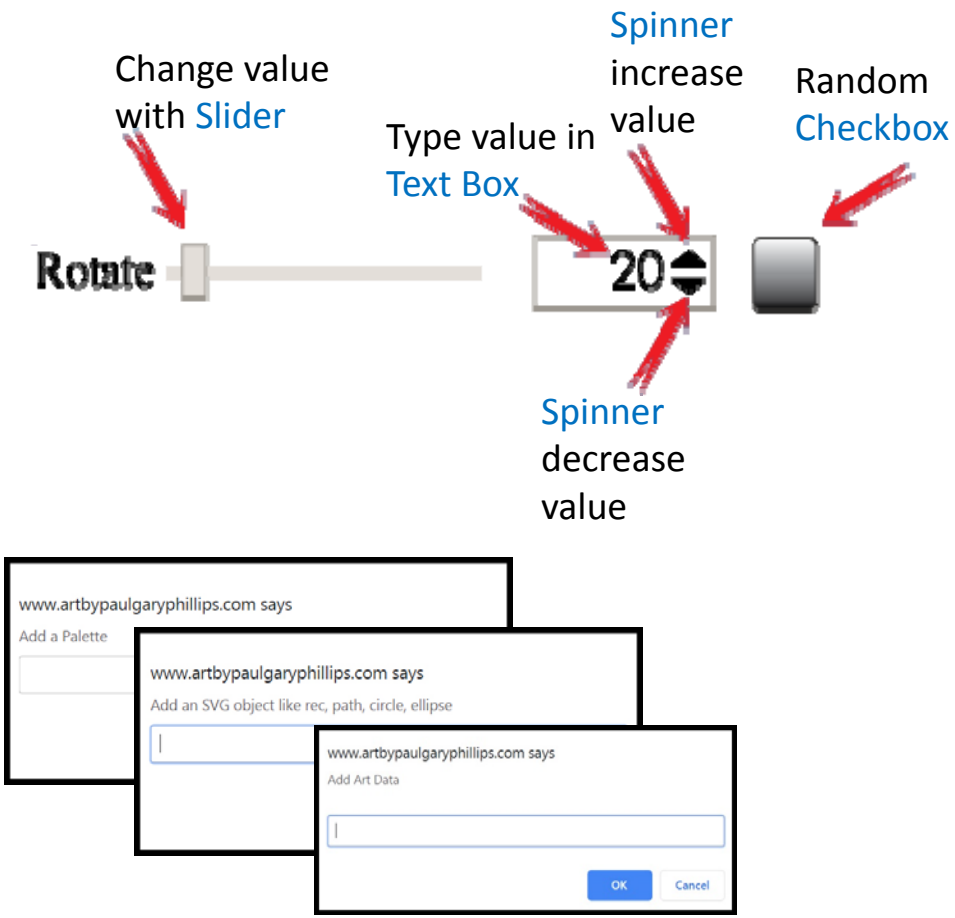
Rotate, Scale, Skew X, Skew Y, Horizontal, Vertical, Hue

Some variables are set for this work of art and the parameter does not change with layers.

Stage Size, Spread, Zoom, Layers, Sat, Val, Alpha, Stroke-Color, Stroke-Width

Some variables show on stage but are not captured with save or download.

Background



# Convert the Math Results to Text.

```
this.rotateString = "rotate(" + r + ",0,0)";
this.scaleString = "scale(" + s + ", " + s + ")";
this.skewXString = "skewX(" + sx + ")";
this.skewYString = "skewY(" + sy + ")";
this.translateString = "translate(" + th + "," + tv + ")";
```

```
this.transformString =
  this.rotateString +" " +
  this.scaleString +" " +
  this.skewXString +" " +
  this.skewYString +" " +
  this.translateString;
return this.transformString;
```

EXAMPLE:

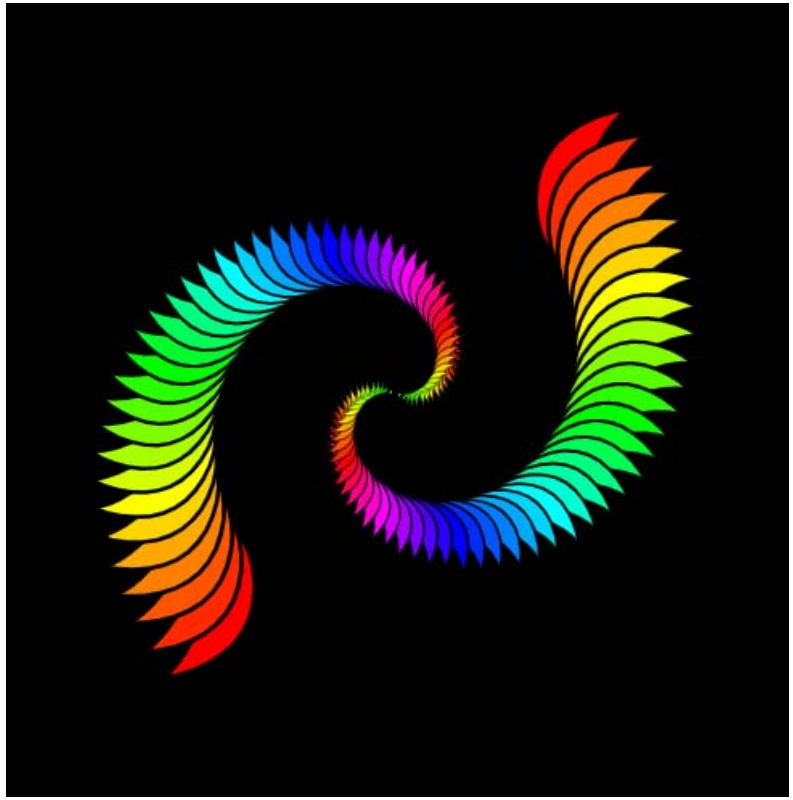
```
<use xlink:href="#bob"
transform="rotate(r,0,0) scale(s,s) skewX(sx) skewY(sy) translate(th,tv)"
stroke="rgba(rs,gs,bs,as)" stroke-width="w" fill="rgba(r,g,b,a)">
</use>
```

# Display The Command Strings On The Stage

Angular 5 uses HTML Code to program loops for dynamic variables.

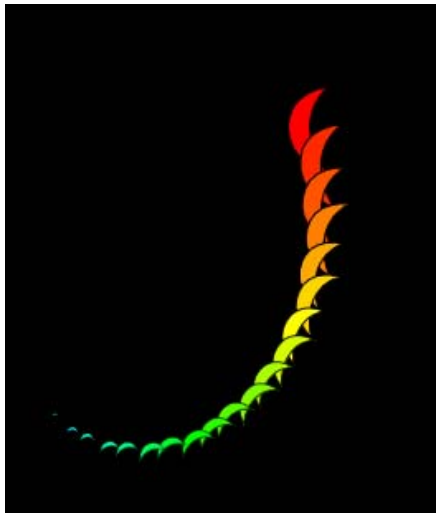
It looks like this:

```
<use *ngFor="let i of Arr(form.value.cycles).fill(1); let r = index"
id="tom{{ r }}“ xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:href="#bob" [attr.transform]="outputTransform('rotate',
form.value.rotateValue * r, form.value.randomRotateValue, 'scale',
form.value.scale * (1 - r / zoomsize), form.value.randomScaleValue,
'skewX', form.value.skewXValue, form.value.randomSkewXValue,
'skewY', form.value.skewYValue, form.value.randomSkewYValue,
'translate', form.value.translateHValue, form.value.randomHValue,
form.value.translateVValue, form.value.randomVValue, r)"
[attr.stroke]="strokeWithWhite2rgba(form.value.stroke)" [attr.stroke-
width]="form.value.strokeWidth| strokeWidthRandom:
form.value.randomStrokeWidthValue" [attr.fill]="fillWithWhite2rgba(r
* form.value.hue, form.value.sat, form.value.val,
form.value.alpha)"></use>
```



# Spread

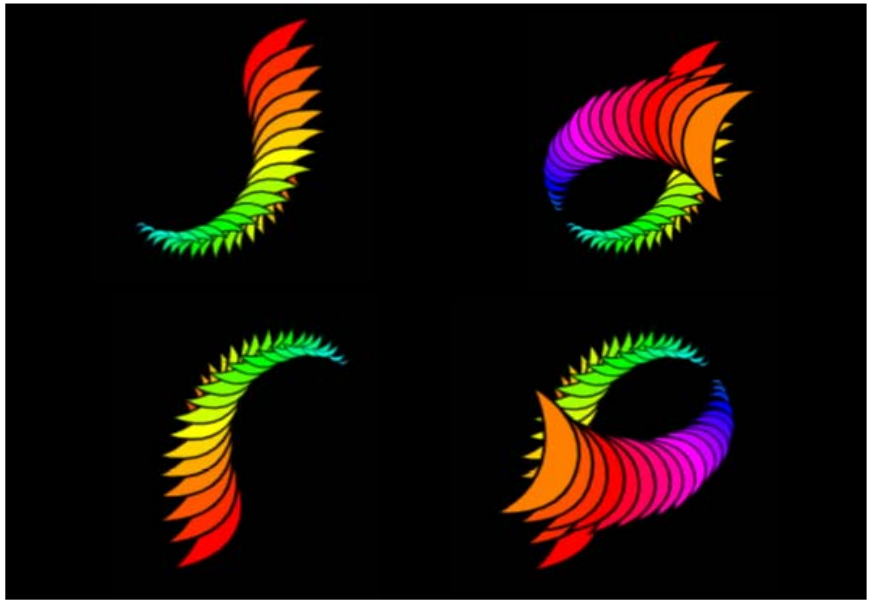
- Spread is needed if you want more or less overlap of the multiple shapes for your primitive. I need this to make coloring books.



# Calculating the Scale Factor

## Zoom: smaller on top or behind

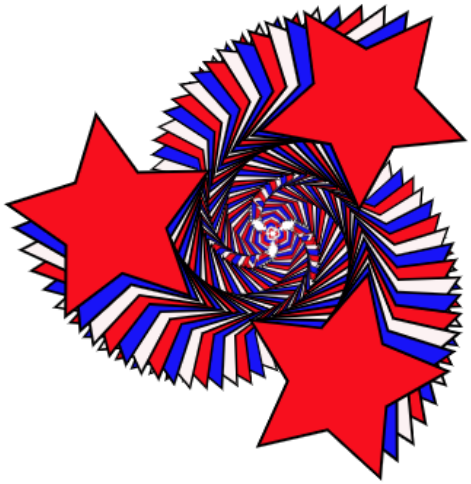
S = scale-slider-value * (1-layers/zoom)



| s=slider*(1-layer/zoom) | | | | | | |
|---|---|---|---|---|---|---|
| l=layer/zoom | | | | | | |
| | | | | | | |
| layer | 20 | 20 | | | s | l |
| | s | l | | 20 | 0.00 | 0.00 |
| 1 | 0.48 | 0.95 | | 21 | -0.03 | -0.05 |
| 2 | 0.45 | 0.90 | | 22 | -0.05 | -0.10 |
| 3 | 0.43 | 0.85 | | 23 | -0.08 | -0.15 |
| 4 | 0.40 | 0.80 | | 24 | -0.10 | -0.20 |
| 5 | 0.38 | 0.75 | | 25 | -0.13 | -0.25 |
| 6 | 0.35 | 0.70 | | 26 | -0.15 | -0.30 |
| 7 | 0.33 | 0.65 | | 27 | -0.18 | -0.35 |
| 8 | 0.30 | 0.60 | | 28 | -0.20 | -0.40 |
| 9 | 0.28 | 0.55 | | 29 | -0.23 | -0.45 |
| 10 | 0.25 | 0.50 | | 30 | -0.25 | -0.50 |
| 11 | 0.23 | 0.45 | | 31 | -0.28 | -0.55 |
| 12 | 0.20 | 0.40 | | 32 | -0.30 | -0.60 |
| 13 | 0.18 | 0.35 | | 33 | -0.33 | -0.65 |
| 14 | 0.15 | 0.30 | | 34 | -0.35 | -0.70 |
| 15 | 0.13 | 0.25 | | 35 | -0.38 | -0.75 |
| 16 | 0.10 | 0.20 | | 36 | -0.40 | -0.80 |
| 17 | 0.08 | 0.15 | | 37 | -0.43 | -0.85 |
| 18 | 0.05 | 0.10 | | 38 | -0.45 | -0.90 |
| 19 | 0.03 | 0.05 | | 39 | -0.48 | -0.95 |
| 20 | 0.00 | 0.00 | | 40 | -0.50 | -1.00 |

# MathArt Color Controls

## *Color Concepts*

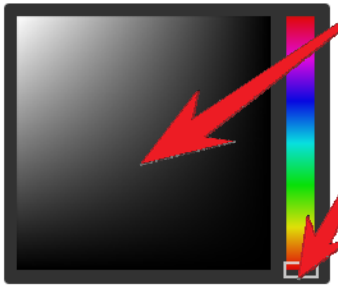


***Color Palette***

Make/Use Palette ☑

Click on Slider Color
Click on Shade in Box

Click ADD COLOR



Shade Box

Slider Color

Add Color activates Color choices and activates pasted-in Palettes

Clears Color Palette

Selected Color: { "r": 100, "g": 136, "b": 150, "a": 1 }

ADD COLOR
NEW PALETTE
SHOW/HIDE PALETTE
PASTE PALETTE
SAVE PALETTE
Random Palette Colors ☐

SHOW/HIDE PALETTE

356,94,97,1 356,4,99,1 241,92,97,1

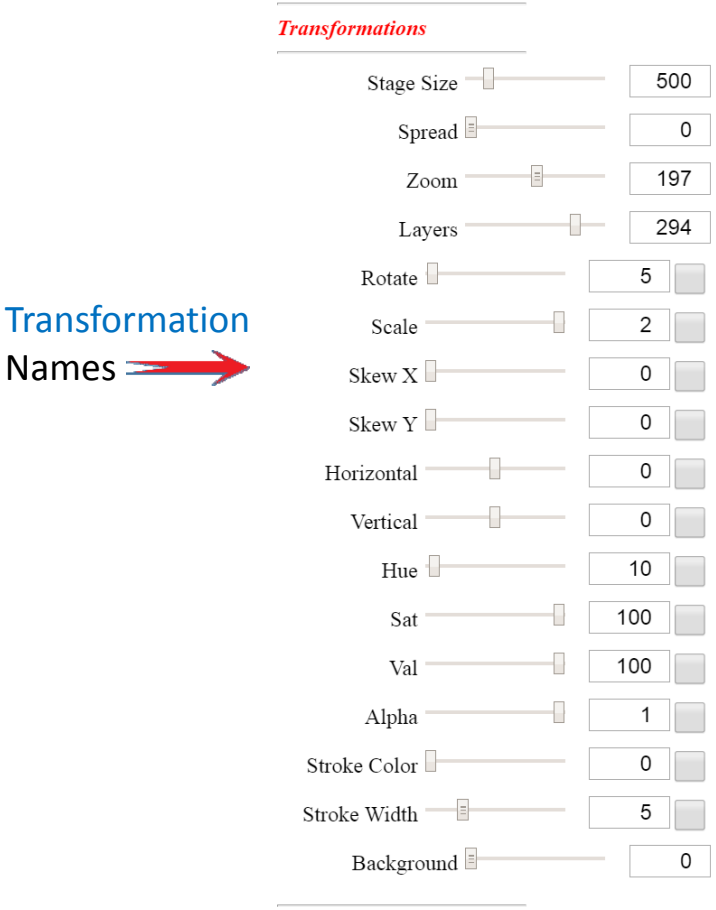
PASTE PALETTE

Opens the Paste Palette Input Box

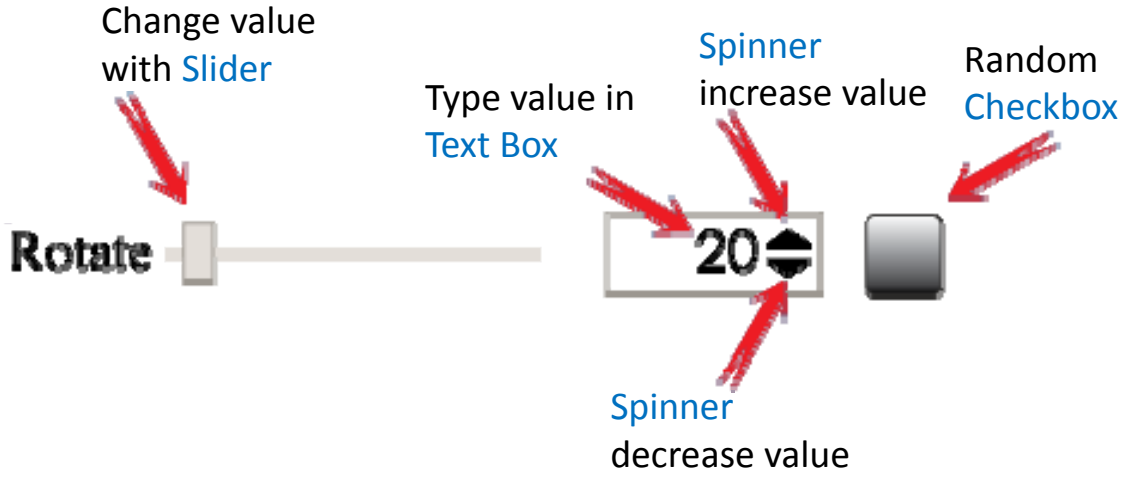
Downloads palette to file named “yourname_palette (#).txt”

Random Order for Palette Colors

# Review
# The MathArt Interface's Dynamic Controls




***Angular 5 a Google Frontend***



# MathArt Knowledge Foundation

The following area of knowledge were used or referred to during the development of this product.

Art

- Pattern Recognition
- Vector Graphics
- Raster Graphics

Web Page Construction

- Application Design
- HTML 5 Coding
- Angular 5 Programming
- Typescript Programming
- JavaScript Programming
- SVG Programming

Computer Color

- HSVA Color Systems
- RGBA Color Systems
- Color Palettes data structure

SVG Coding for Transformations

- Scale
- Rotate
- Shear or Skew
- Move Horizontally
- Move Vertically
- Stroke Coding
- Fill Coding

Math Calculations

- Algebra
- Geometry
- Trigonometry

Computer Operations

# REMEMBER

You can learn to code in any language.

But before you do,

you have to be able to

DO THE MATH