

# < generate . font >

research notes on generative letterforms

## <Contents>

Disclaimer	01	18	Spore Abstract
Amalgam Abstract	02	19	Spore.Code
Process Visualization	03	20	
Amalgam.Code	04	21	
	05	22	
Amalgam.Std: First Run	06	23	
	07	24	
	08	25	Spore.Results
	09	26	
Amalgam.Serif	10	27	
	11	28	
	12	29	
	13	30	
Amalgam.Sans	14	31	
	15	32	
	16	33	
	17		

Disclaimer: The following pages contain the documentation of experiments in generating typographic forms through the use of code.

Code was written for, and executed by the Processing programming language. Works documented in this volume have been licensed under a GNU General Public License. The code is free to modify and distribute under the condition that all derivative works will carry the same license.

Posit Labs provides no warranty to any of its open-source materials, nor will its use imply Posit Labs' association to the end-product.

Results have been validated by Joshua Beckwith:  
[ 2.08.2010 - 3.27.2010 ]

## <Amalgam>

Abstract: The goal of this exploration is to better understand how people remember letterforms. The idea is that everybody is daily exposed to a wide variety of typographic forms. They might remember bits and pieces of these forms, but most likely never a whole typeface unless they happen to be a typographer.

The Amalgam code takes a number of typefaces and looks for commonalities between the letterforms. It tries to assemble those bits and pieces into something tangible.

First run: Sixteen non-script.

Second run: Eight sans-serifs.

Third run: Nine serifs.

Process visualization:



01: Load typefaces, center position, opacity 16%.

02: Find darkest regions.

03: Fill darkest regions with black, delete the rest.

```
Create font variables: PFont font01;
                      PFont font02;
                      etc... PFont font03;

Create letter variable: String letter = "A";

Setup environment: void setup(){

Set background to white: background(255);
Set window size in pixels: size(400, 400);

Load fonts: font01 = loadFont("TexturaModern-200.vlw");
etc... font02 = loadFont("fontName-fontSize.vlw");

Run program once: noLoop();
                  }
```

```
Execute process: void draw(){

Text color (black,alpha): fill(0, 50);
Center text origin: textAlign(CENTER, CENTER);

Set typeface: textFont(font01);
Place text centered: text(letter, width/2, height/2);
etc... textFont(font02);
text(letter, width/2, height/2);

Apply filter to sort levels: filter(THRESHOLD, 0.75);

Save result: save(letter + "_amglm.png");
Close program: exit();
               }
```

<Amalgam.Std>

First run results:

# Amalgam Standard

DEVELOPED 2/12/10

A B C D E F G H I J K L M  
N O P Q R S T U V W X Y Z  
a b c d e f g h i j k l m n  
o p q r s t u v w x y z  
0 1 2 3 4 5 6 7 8 9

.....  
↑  
06

</Std.details>

NGASM

<Amalgam.Serif>

Second run results:

Amalgam  
Serif

DEVELOPED 2/18/10

A B C D E F G H I J K L M  
N O P Q R S T U V W X Y Z  
a b c d e f g h i j k l m  
n o p q r s t u v w x y z  
0 1 2 3 4 5 6 7 8 9



</Serif.details>



<Amalgam.Sans>

Third run results:

Amalgam Sans

DEVELOPED 2.23.01

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m

n o p q r s t u v w x y z

0 1 2 3 4 5 6 7 8 9

</Sans.details>

Ww

5

@g

T

## <Spore>

Abstract: Spore uses a physics engine to generate a particle system that grows the typeface. The particles initial position is determined by information retrieved from the GNU FreeFont typeface, FreeSans.

Three sets of particles are generated - one with fixed position and two free-floating. Only one of the free-floating particle systems is visible. The others serve only to affect the behavior of the visible particles.

Particle size diminishes over time, and after about frame four-hundred and fifty, the particles disappear.

## <Spore.code>

```
get all methods in these libraries: import geomerative.*;
import traer.physics.*;

characters to display: String letter = ".font";

initialize font variables: RFont f;
RShape grp;
RPoint[] points;

initialize variable for alpha fade: int fade = 100;

init particle system: ParticleSystem physics;
init particle arrays: Particle[] p;
Particle[] p2;
Particle[] pFix;
```

```

    setup environment: void setup(){
        set bg to white: background(255);
    window size(x, y, renderer): size(2000, 1000, P3D);

    create particle system: physics = new ParticleSystem();

    init geomerative: RG.init(this);
    (text, typeface, pointsize, align): grp = RG.getText(letter, "FreeSans.ttf", 200, RIGHT);

    distance b/w points: RG.setPolygonizer(RG.UNIFORMLENGTH);
    RG.setPolygonizerLength(1);
    points = grp.getPoints();

    particle array length = # of points: p = new Particle[points.length];
    p2 = new Particle[points.length];
    pFix = new Particle[points.length];

```

```

    for every point, do (this): for(int i=0; i<points.length; i++){
        position points: points[i].x = points[i].x + width/1.3;
        points[i].y = points[i].y + height/1.5;

        (mass, x, y, z): p[i] = physics.makeParticle(1.0, points[i].x-
        (random(-10,10)), points[i].y-(random(-10, 10)), 0);

        p2[i] = physics.makeParticle(1.0, points[i].x-
        (random(-10,10)), points[i].y-(random(-10, 10)), 0);

        pFix[i] = physics.makeParticle(1.0, points[i].x, points[i].y, 0);
    pFix particles don't move: pFix[i].makeFixed();

    (p,p, strength, minDist): physics.makeAttraction(p[i], pFix[i], 100, 10);
    physics.makeAttraction(p[i], p2[i], 50, 10);
    physics.makeAttraction(pFix[i], p2[i], -90, 10);
    (p,p, str, damping, dist) physics.makeSpring(p[i], p2[i], 2, 1.2, 6.5);
    }
}

```

```

draw function loops: void draw(){
wall collision handler:   handleCollisions();
reset bg until frame 12: if(frameCount<12){
                          background(255);
                          }
begin fading particles at 400: if(frameCount> 400){
    fade = fade-1:         fade--;
                          }

    fill (black, alpha):   fill(0,fade);
    stroke(white, alpha):  stroke(255,fade/2);

    for each point, do (this): for(int j=0; j<points.length; j++){
        draw circle at p2:  ellipse(p2[j].position().x(), p2[j].position().y(),
    diminish size based on time: 550/frameCount, 550/frameCount);
    }

    progress physics sim: physics.tick();
}

```

```

void handleCollisions(){
    for(int i = 0; i<points.length; i++){
        if ( p[i].position().x() < 0 || p[i].position().x() > width )
            p[i].velocity().set( -0.9*p[i].velocity().x(), p[i].velocity().y(), 0 );
        if ( p[i].position().y() < 0 || p[i].position().y() > height )
            p[i].velocity().set( p[i].velocity().x(), -0.9*p[i].velocity().y(), 0 );
        p[i].position().set( constrain( p[i].position().x(), 0, width ), constrain( p[i].
            position().y(), 0, height ), 0 );

        if ( p2[i].position().x() < 0 || p2[i].position().x() > width )
            p2[i].velocity().set( -0.9*p2[i].velocity().x(), p2[i].velocity().y(), 0 );
        if ( p2[i].position().y() < 0 || p2[i].position().y() > height )
            p2[i].velocity().set( p2[i].velocity().x(), -0.9*p2[i].velocity().y(), 0 );
        p2[i].position().set( constrain( p2[i].position().x(), 0, width ), constrain( p2[i].
            position().y(), 0, height ), 0 );
    }
}

```

monorato

.font















