

ANDERSON KOYAMA VIEIRA

## **Design Generativo - Estudo exploratório sobre o uso de programação no design**

Trabalho de Conclusão de  
Curso apresentado à Faculdade  
de Arquitetura e Urbanismo da  
Universidade de São Paulo como  
exigência parcial para obtenção do  
grau de Bacharel em Design.

Área de Concentração:  
Design e arquitetura

Orientador: Prof. Dr. Ricardo Nakamura

São Paulo  
2014



*A plausible result of sustained spontaneity*

- John Borowicz



## **RESUMO**

Atualmente, o uso de novas tecnologias influencia diretamente o processo de criação do designer. A programação é uma destas ferramentas e vem sendo utilizada, por exemplo, em aplicações do design que tem sido chamadas de Design Paramétrico e Design Generativo. Assim sendo, o objetivo desse trabalho é realizar um estudo exploratório sobre a influência da programação no processo de design e em sua linguagem visual. Para atingir esse objetivo, foi levantado um histórico desta prática, traçado um panorama de projetos da atualidade e foram realizadas experimentações práticas que exploraram as possibilidades e potencialidades deste modo de fazer design na área visual. Estas explorações revelaram pontos importantes do processo de design que envolve a programação. Pontos esses que discutimos neste trabalho e que nos permitiram compreender melhor como se dá esta prática.

## **PALAVRAS-CHAVE**

estudo exploratório, design paramétrico, design generativo, programação, Processing, conversa reflexiva, emergência, *open source*.



## **ABSTRACT**

Today, the design process is directly influenced by the use of new technologies. Programming is one of these tools and it has been used, for example, in design applications called parametric design and generative design. Thus, the goal of this work is to perform an exploratory study about the influence of programming in the design process and in the visual language. To achieve this goal, we have summed a history of this practice, traced a scenario of present projects and produced practical experimentations that explored the possibilities and potentialities of this way of designing in the visual field. This exploration revealed important points of the programming related design process. Points in which we have discussed in this work that allowed us to better understand how this practice occurs.

## **KEYWORDS**

exploratory study, parametric design, generative design, programming, Processing, reflective conversation, emergency, *open source*.



# SUMÁRIO

<b>1. INTRODUÇÃO</b>	13
1.1 Termos e definições .....	13
1.1.1 Design paramétrico .....	13
1.1.2 Design generativo .....	16
1.1.3 Escolha de um termo genérico .....	17
1.2 Panorama atual.....	18
1.2.1 Artes .....	18
1.2.2 Arquitetura .....	20
1.2.3 Design gráfico .....	22
1.2.4 Design de produto .....	24
1.3 Histórico da parametrização.....	26
1.3.1 A parametrização na matemática.....	26
1.3.2 A parametrização analógica.....	26
1.3.3 O Sketchpad .....	27
1.3.4 Era digital.....	28
1.4 Tecnologia.....	28
1.4.1 Cultura open source .....	29
1.4.2 Ambientes colaborativos .....	29
1.5 Metodologia .....	30
<b>2. ESTUDO DE CASO [PROCESS COMPENDIUM]</b>	31
2.1 Sobre o autor .....	31
2.2 Sobre o projeto .....	31
2.3 Elemento = forma + comportamento(s) .....	32
2.4 Sobre Processo .....	34
<b>3. PROGRAMAÇÃO NO PROCESSO REFLEXIVO DO DESIGN</b>	37
3.1 Algoritmo.....	37
3.2 Programação.....	38
3.3 Conversa reflexiva do processo de design.....	39
<b>4. EXPERIMENTOS 2.0</b>	41
4.1 DEFINIÇÕES INICIAIS.....	41
4.1.1 Sistema de partículas.....	42
4.1.2 Agentes autônomos.....	43

4.2 _dla .....	44
4.3 _pong .....	46
4.4 _retro .....	48
4.5 _frame .....	50
4.6 _perlin .....	52
4.7 _spline .....	54
4.8 _metaball .....	56
4.9 _seek .....	58
4.10 _outline .....	60
4.11 _stipple .....	62
4.12 _grid .....	64
4.13 _tentacle .....	66
4.14 _pattern .....	68
4.15 _shadow .....	70
4.16 _orbit .....	72
4.17 _swarm .....	74
4.18 _explode .....	76
4.19 _thread .....	78
4.20 _swirl .....	80
4.21 _drop .....	82
<b>5. DISCUSSÃO</b>	<b>85</b>
5.1 Compreensão do código .....	85
5.2 Construção temporal .....	85
5.3 Emergência no processo de design generativo .....	86
5.4 Criação não intencional a partir de “bugs” .....	87
5.5 Hardcoding: solução improvisada .....	88
5.6 Importância da ferramenta open source .....	88
5.7 Referências .....	89
5.8 Evolução .....	89

5.9 Controle .....	90
5.10 Métodos .....	90
5.11 Codificação .....	90
5.12 Linguagem.....	91
<b>6. EXPERIMENTOS 3.0</b>	<b>93</b>
6.1 The source of the new is the random - Gregory Bateson.....	94
6.2 Intention is the key and the process is the product - Genesis P-Orridge .....	96
6.3 A set is a Many that allows itself to be thought of as a One - Georg Cantor.....	98
<b>7. CONCLUSÃO</b>	<b>101</b>
<b>8. REFERÊNCIAS BIBLIOGRÁFICAS</b>	<b>103</b>
<b>9. APÊNDICE: EXPERIMENTOS 1.0</b>	<b>105</b>
9.1 Fundamentos.....	105
9.1.1 Lógica.....	105
9.1.3 Parametrização .....	106
9.1.2 Randomização .....	106
9.2 Abordagens .....	106
9.2.1 Repetição .....	108
9.2.2 Transformação.....	110
9.2.3 Visualização .....	112
9.2.4 Simulação .....	114



# 1. INTRODUÇÃO

O uso de novas tecnologias tem influenciado o processo de criação em design e também tem possibilitado a concepção de resultados inéditos. Por exemplo, a programação facilita o processo ao passo que o designer cria suas próprias ferramentas e passa a utilizá-las na concepção de novos produtos.

Mais do que as ferramentas computacionais já concretizadas no design, como os aplicativos de modelagem e de desenho digital, a programação permite que o designer utilize a capacidade procedural do computador para gerar novas ferramentas de design, como o que temos visto no design paramétrico e no design generativo.

Esses enfoques ainda são carentes de estudos mais aprofundados, principalmente no Brasil. Um dos motivos desta dificuldade se deve à falta de familiaridade com a programação, que ainda é comum entre designers.

Este trabalho teve a duração de um ano e foi dividido em dois semestres que compreenderam o TCC1 e TCC2. Seu objetivo é realizar um estudo exploratório sobre o uso do design paramétrico e generativo na atualidade. Pretendemos abordar como os conceitos de algoritmo, código e parametrização podem ser usados no processo criativo, além de analisar a linguagem visual gerada a partir deste procedimento. Em termos metodológicos pretendemos realizar experimentações que explorem as possibilidades e potencialidades deste modo de fazer design, a fim de melhor compreendê-lo.

Neste primeiro capítulo são definidos termos relevantes à área estudada, é apresentado um panorama do que está sendo produzido atualmente, é apresentado um histórico do desenho paramétrico e, por fim, são explicados aspectos tecnológicos importantes para esse cenário.

## 1.1 TERMOS E DEFINIÇÕES

A primeira etapa para entender melhor essa área de estudo do design foi definir dois termos importantes: design paramétrico e design generativo. Esses dois termos representam o cenário do design a ser estudado nesse trabalho, que tratam da programação como ferramenta no processo de design.

### 1.1.1 Design paramétrico

Para definir design paramétrico, antes é necessário definir o que é parâmetro. Do livro “*Form and Code*” de Casey Reas e do estúdio LUST, parâmetro é “um valor que tem efeito sobre o resultado de um processo. Ele pode ser direto como a quantidade de açúcar em uma receita, ou complexo como o limiar de ativação de um neurônio no cérebro. No contexto da arquitetura e do design, parâmetros descrevem, codificam e quantificam as opções e restrições existentes dentro de um sistema. Uma restrição comum é o orçamento disponível para um projeto, enquanto uma opção de configuração é o controle de cor, tamanho, densidade ou material.”<sup>1</sup> (REAS; LUST 2010, tradução nossa)

<sup>1</sup> “a value that has an effect on the output of a process. This could be something as straightforward as the amount of sugar in a recipe, or as complex as the activation threshold of a neuron in the brain. In the context of architecture and design, parameters describe, encode, and quantify the options and constraints at play in a system. A common constraint

De acordo com o professor Javier Monedero, “Design paramétrico é, em certo sentido, um termo bastante restrito; ele implica o uso de parâmetros para definir uma forma”<sup>2</sup> (MONEDERO 2000, tradução nossa). A visão de Branko Kolarevic complementa a definição anterior: “No design paramétrico, são os parâmetros de um design particular que são declarados e não a sua forma”<sup>3</sup> (KOLAREVIC 2000, tradução nossa). O site Parametric Camp ainda diz que “o fundamento do design paramétrico é a geração de geometria a partir da definição de uma família de parâmetros iniciais e o delineamento de relações formais que eles mantém entre si.”<sup>4</sup>

A partir das definições fornecidas, é possível concluir que design paramétrico é a área do design<sup>5</sup> em que o produto projetado é concebido através da definição de um conjunto de parâmetros vinculados a atributos desse produto. Atualmente, sistemas paramétricos estão relacionados ao uso de ferramentas digitais, tanto 2D quanto 3D, capazes de modelar formas diversas de acordo com os parâmetros fornecidos pelo designer.

Para exemplificar esse pensamento, a Figura 1 mostra um pente de cabelo e alguns de seus atributos (espessura e largura da alça, altura dos dentes, etc). Se esse pente fosse projetado em um sistema paramétrico, cada atributo seria controlado por um parâmetro, sendo que a alteração desses parâmetros modificaria o produto final, gerando inúmeras possibilidades, como mostrado na Figura 2.

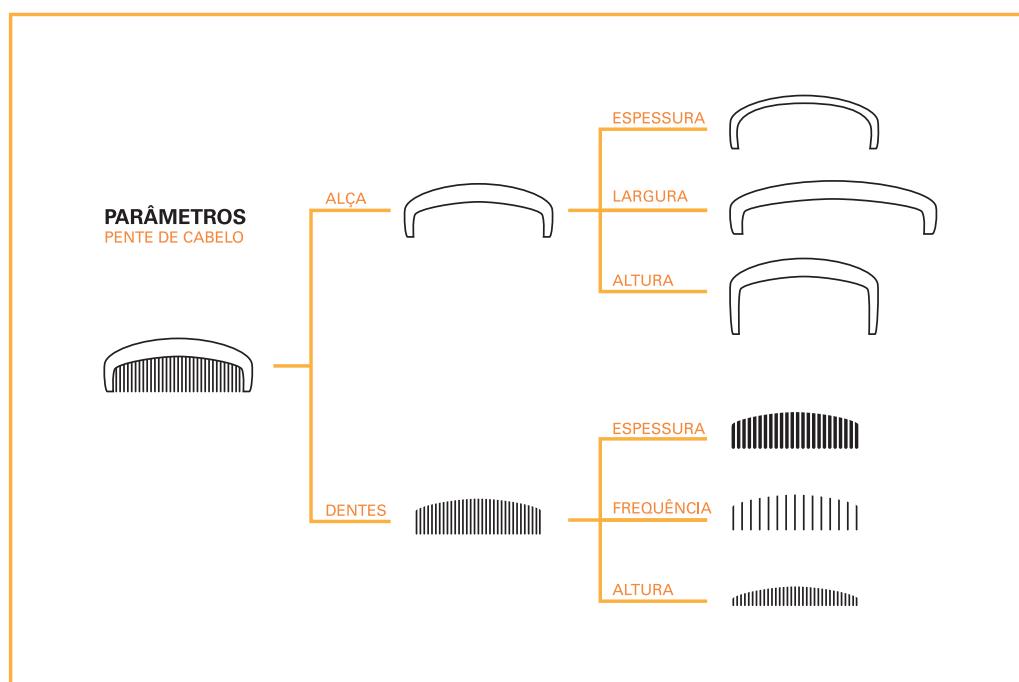


Figura 1: Pente de cabelo e alguns de seus atributos

might be the budget available for a project, while a configuration option might control color, size, density, or material.”

2 “Parametric design is, in a sense, a rather restricted term; it implies the use of parameters to define a form”

3 “In parametric design, it is the parameters of a particular design that are declared, not its shape”

4 Fonte: <http://www.parametriccamp.com/en/what-is-parametric-design>

5 Seguimos aqui uma definição bastante abrangente do termo “design”, como a proposta por Buchanan (vide citação), que vai além do design gráfico e desenho industrial, e engloba diversas atividades relacionadas à prática projetual, como arquitetura, engenharia e a arte.

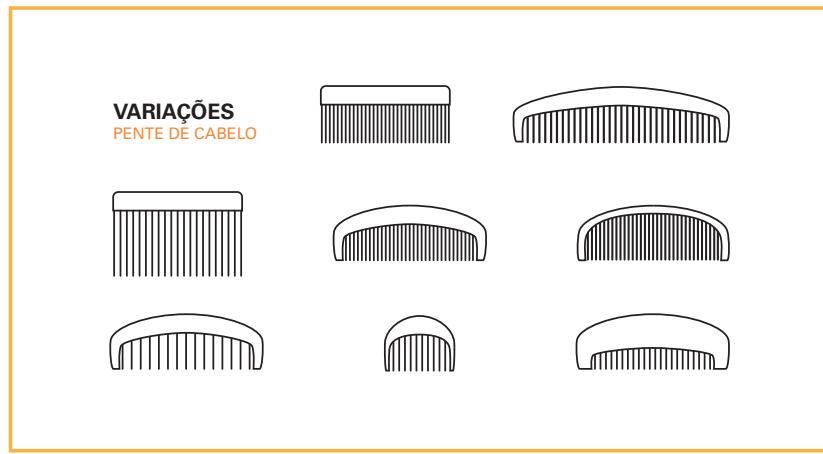


Figura 2: Algumas possibilidades de produtos possíveis a partir da alteração de seus parâmetros

A figura 3 mostra um exemplo de design gráfico que utiliza o conceito de parametrização. Esse poster chamado de *Morisawa 8* faz parte de um conjunto de posters desenvolvidos pelo artista John Maeda. Essa obra foi gerada a partir de um programa de computador que repetia o logotipo da empresa Morisawa diversas vezes a partir de três principais parâmetros: largura do layout, altura do layout e fator de progressão aritmética. A largura da composição define o tamanho do primeiro logotipo; a altura define o máximo de linhas a serem preenchidas; o fator de progressão define quantos logotipos serão adicionados a cada nova linha.

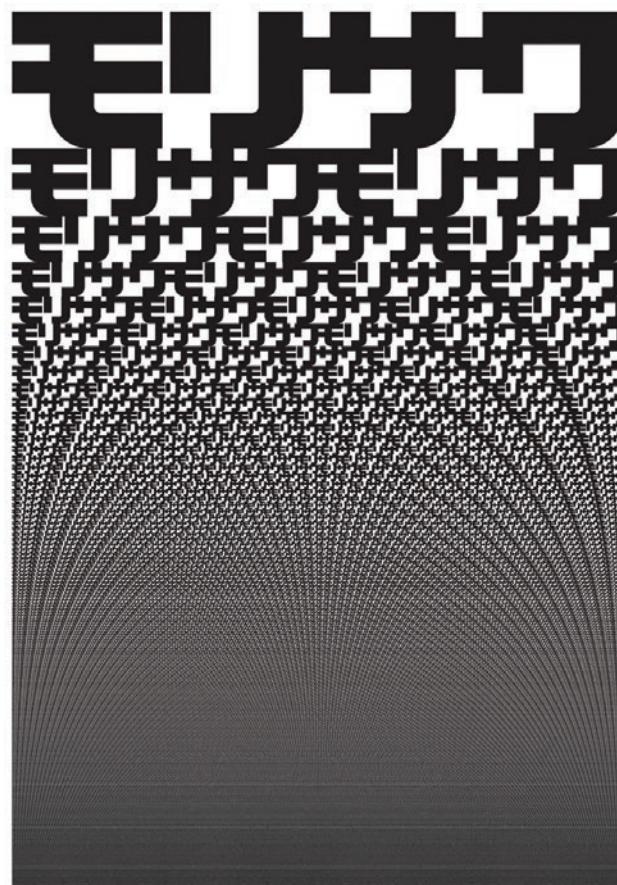


Figura 3: Poster paramétrico *Morisawa 8* por John Maeda

### 1.1.2 Design generativo

Kristina Shea em seu artigo “*Towards integrated performance-driven generative design tools*”, diz que sistemas de design generativo “são focados em criar novos processos de design que produzem projetos espaciais inovadores e ao mesmo tempo construíveis através da exploração das capacidades atuais de computação e de fabricação”.<sup>6</sup> (SHEA 2005, tradução nossa)

Sivam Krish acrescenta que o design generativo é “um processo de exploração restrito parametricamente e orientado pelo designer, que opera em sistemas paramétricos CAD para auxiliar o projeto como um processo emergente”.<sup>7</sup> (KRISH 2010, tradução nossa)

O professor Celestino Soddu em 1992 diz que o design generativo é “um processo morfogênico que usa algoritmos estruturados como sistemas não lineares para resultados únicos infinitos e sem repetição realizados por uma ideia-código, como na natureza.”<sup>8</sup>

A partir das definições anteriores, percebemos que o design paramétrico engloba o design generativo conceitualmente, visto que este se utiliza de parâmetros para definir resultados. O fator que diferencia os dois conceitos é a abordagem ou a intenção do designer. Como o design generativo tem uma proposta exploratória que busca por possibilidades inéditas e emergentes a fim de solucionar um problema, esse enfoque normalmente é utilizado em estágios iniciais do processo de design, já que o intuito nesses estágios é o de propor novas ideias e achar diferentes possibilidades.

O caráter exploratório é facilmente identificado no trabalho *Path* feito por Casey Reas em 2001. O mesmo programa de computador é capaz de gerar inúmeros resultados possíveis, como vemos na figura 4. É possível presumir que os resultados obtidos sejam randômicos, porém a composição é gerada a partir de restrições paramétricas que delimitam a irregularidade das linhas, como dito anteriormente.

Essa abordagem exploratória está intimamente ligada com o objetivo deste trabalho, que busca estudar, compreender e investigar as possibilidades de criação que a programação oferece como ferramenta disponível ao designer.

6 “Integrated performance-driven generative design systems are aimed at creating new design processes that produce spatially novel yet efficient and buildable designs through exploitation of current computing and manufacturing capabilities”

7 “a designer driven, parametrically constrained design exploration process, operating on top of history based parametric CAD systems structured to support design as an emergent process”

8 “a morphogenetic process using algorithms structured as not-linear systems for endless unique and un-repeatable results performed by an idea-code, as in Nature” Fonte: <http://www.soddu.it>

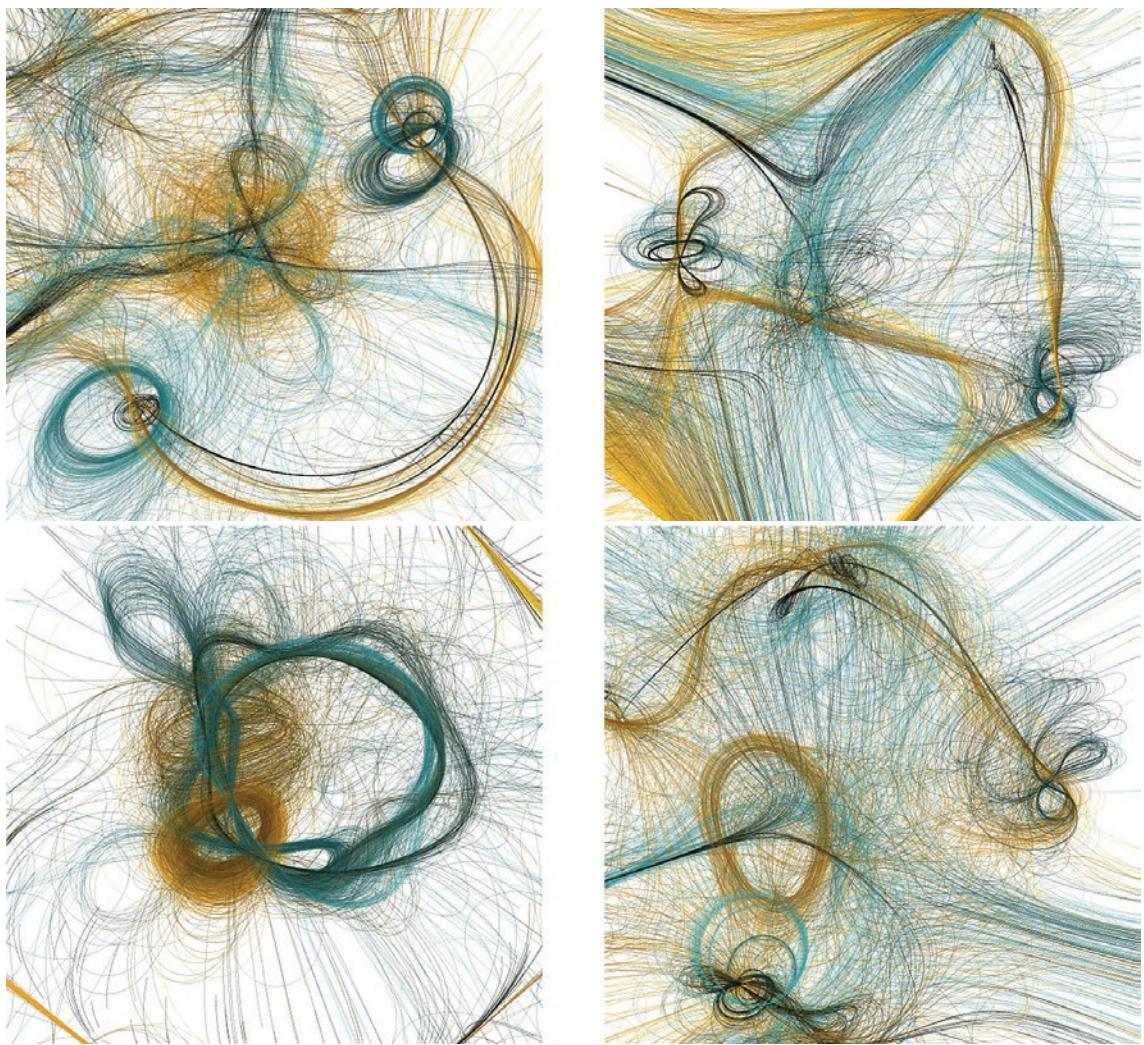


Figura 4: Path por Casey Reas. Formas distintas são geradas a cada execução do programa.

### 1.1.3 Escolha de um termo genérico

Para fins práticos e com o intuito de utilizar um termo genérico que se refira ao uso da programação no processo de design em geral, utilizaremos “design paramétrico”. Caso um assunto esteja relacionado apenas ao design generativo, esse termo será utilizado ao invés de design paramétrico.

Não há a intenção aqui de criar um novo enfoque ou um novo conceito de design, visto que já existe atualmente um excesso de termos, que acabam definindo enfoques muito parecidos entre si.

Além das áreas definidas anteriormente, também é possível encontrar outros termos que delimitam áreas semelhantes que se utilizam da programação como ferramenta, como: design computacional, design algorítmico, design procedural, design condicional e design associativo.

## 1.2 PANORAMA ATUAL

Após a definição de alguns termos relevantes ao tema desse trabalho, um panorama foi gerado a fim de se observar o que está sendo produzido sob o enfoque do design paramétrico. Ao observar este cenário, é possível identificar aspectos característicos em relação à linguagem visual, à tecnologia e ao tipo de aplicação. O panorama é dividido em quatro campos de aplicação: artes, arquitetura, design gráfico e design de produto; em cada área são escolhidos dois trabalhos com abordagens distintas entre si.

### 1.2.1 Artes

#### Substrate - Jared Tarbell 2004

“Linhas como cristais crescem em um substrato computacional. Uma simples regra de crescimento perpendicular cria estrutura intrincadas que aparentam cidades. A regra simples, os resultados complexos, o enorme potencial para modificação; isso tem que ser o meu algoritmo auto-descoberto favorito. Permitindo que as fendas dentro do substrato se rotacionem, vemos maior irregularidade, incluindo estruturas isoladas que se fundem de modos complexos.”<sup>9</sup>

*Substrate* é a obra dinâmica de Jared Tarbell. A peça é criada utilizando o programa Processing, porém o resultado final aparenta ter sido produzido a partir de ferramentas tradicionais devido ao autor simular a aparência da aquarela em seu código. Essa peça é uma exemplo da programação sendo usada para criar ilustrações digitais.



Figura 5: O programa Substrate gera resultados caóticos e sublimes. Fonte: Código-fonte disponível em: [http://www.complexification.net/gallery/machines/substrate/applet1/substrate\\_1.pde](http://www.complexification.net/gallery/machines/substrate/applet1/substrate_1.pde)

<sup>9</sup> Texto originalmente em inglês escrito por Tarbell, extraído de: [www.complexification.net/gallery/machines/substrate/applet1/substrate\\_1.pde](http://www.complexification.net/gallery/machines/substrate/applet1/substrate_1.pde)

### **Being not truthful works against me - Ralph Ammer e Stefan Sagmeister 2006**

Essa obra é uma instalação interativa que consiste na projeção de uma teia virtual com a frase “Being not truthful works against me” em uma tela branca. Um sensor capta o movimento da sombra de um observador, rasgando a teia por onde ela passa, causando assim a destruição da obra e, posteriormente, a sua recomposição quando inativa.

Essa peça é um exemplo de como a programação é usada como ferramenta para se criar experiências interativas, gerando artes visuais dinâmicas que se relacionam com o expectador.

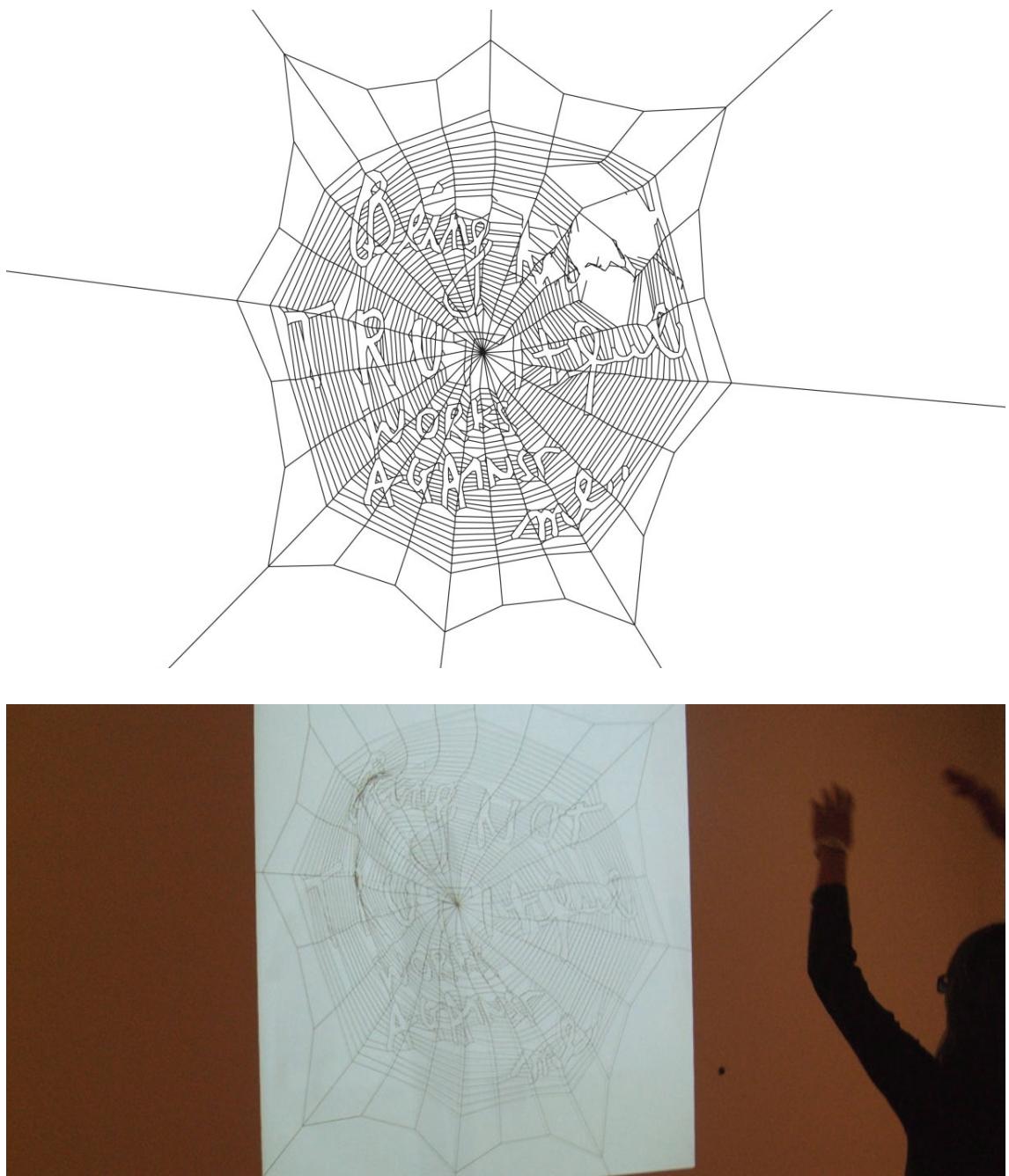


Figura 6: Obra interativa que interage com os expectadores. Fonte: <http://rashomonija.blogspot.com.br/2012/06/stefanova-lista.html> e [http://chicago.about.com/od/artsulture/ss/ModernWingTop10\\_5.htm](http://chicago.about.com/od/artsulture/ss/ModernWingTop10_5.htm)

## 1.2.2 Arquitetura

### Shellstar Pavilion - Andrew Kudless 2012

Shellstar é um pavilhão temporário feito para o Detour, festival sobre arte e design em Hong Kong. O projeto é estruturado a partir do cálculo de curvas catenárias e foi modelado utilizando como base o programa Rhinoceros e o plugin Grasshopper.

“Projetado completamente dentro de um ambiente de modelagem paramétrica, o design foi rapidamente desenvolvido e iterado em seis semanas que incluíram projeto, fabricação e montagem. O processo de design pode ser dividido em três processos que foram possíveis a partir de técnicas avançadas de modelagem digital:”<sup>10</sup>

**descoberta da forma:** foi possível criar a estrutura catenária através da programação.

**otimização de superfície:** 1500 células foram organizadas via código para compor a estrutura.

**planejamento de fabricação:** Cada célula foi desdobrada e etiquetada para fácil identificação no processo de montagem.



Figura 9: Podemos ver a estrutura instalada, uma simulação digital, as linhas de corte de uma das células e um detalhe da junção das peças. Fonte: <http://matsysdesign.com/2013/02/27/shellstar-pavilion/>

10 Fonte: [www.matsysdesign.com/2013/02/27/shellstar-pavilion](http://matsysdesign.com/2013/02/27/shellstar-pavilion/)

## **BMW Welt - Coop Himmelb(l)au 2007**

BMW Welt é um edifício multi-funcional da empresa BMW localizado em Munique, Alemanha. Ele é utilizado para exposições de produtos, conferências, eventos e serve como museu e centro de distribuição.

O edifício é um exemplo de modelagem paramétrica feito por um método que permitiu a repetição automatizada e precisa de painéis de vidro em uma forma fluida e retorcida com base em uma estrutura de aço chamada pela empresa de Cone Duplo (do inglês, *Double Cone*). “Preso em um turbilhão de vidro e aço, o tornado torce para cima e termina em um teto que toma a forma de uma nuvem flutuante e voadora. Gerado pelas dinâmicas torcidas dos dois estratos de suporte, esse tornado funciona como principal apoio para o telhado.”<sup>11</sup>



Figura 11: Detalhe do Cone Duplo, por fora e por dentro. Fonte: <http://www.bmwblog.com/2011/07/31/bmwblog-special-feature-bmw-welt-in-the-belly-of-the-beast>

11 Fonte: [www.bmw-welt.com/en/location/welt/architecture.html](http://www.bmw-welt.com/en/location/welt/architecture.html)

### 1.2.3 Design gráfico

#### Identidade da Casa da Música - Stefan Sagmeister 2007

A Casa da Música é um conservatório localizado na cidade de Porto, Portugal. A identidade foi feita baseada na arquitetura de seu edifício, projetado por Rem Koolhaas. Ao todo foram criadas seis marcas que correspondem a perspectivas diferentes do edifício.

Além disso, um aplicativo foi desenvolvido no qual se carrega uma fotografia e dela são retiradas cores de pontos pré-determinados a fim de colorir o logo do conservatório, criando assim um aspecto mutável. “A característica do logo muda para refletir o gênero de música a ser apresentado, para que a imagem da marca mantenha harmonia entre os vários tipos de ofertas musicais do conservatório”<sup>12</sup> (BOHNACKER 2012, tradução nossa).



Figura 15: Na parte superior vemos as diferentes perspectivas do logo; na parte inferior vemos a interface do programa que gera cores a partir de uma imagem. Fonte: <http://www.sagmeisterwalsh.com/work/project/casa-da-musica-identity/>

12 “The logo's character changes to reflect the genre of music being performed, so that the brand image maintains harmony among the concert hall's various types of musical offerings”

## In Climbing Income Ladder, Location Matters - Amanda Cox 2013

Amanda Cox é editora gráfica do jornal New York Times há 8 anos e desenvolve visualizações desde simples infográficos impressos até ferramentas on-line de dados interativos. Seu trabalho, junto com o de seus colegas, já ganhou diversos prêmios e atualmente ela é uma das profissionais mais importantes da área de visualização de dados.

A área de visualização de dados é uma área extremamente funcional que mescla design, programação e dados estatísticos para se extrair de maneira mais eficiente as informações a serem analizadas.

A matéria do New York Times “In Climbing Income Ladder, Location Matters”<sup>20</sup> exibe uma visualização que aborda a probabilidade, dividida por estado, de uma criança que nasceu com uma renda entre as 20% mais baixas dos EUA subir para um renda entre as 20% mais altas. Uma outra visualização mostra as chances de se manter entre as 20% rendas mais altas do país, vindo tanto de famílias pobres quanto de famílias ricas.

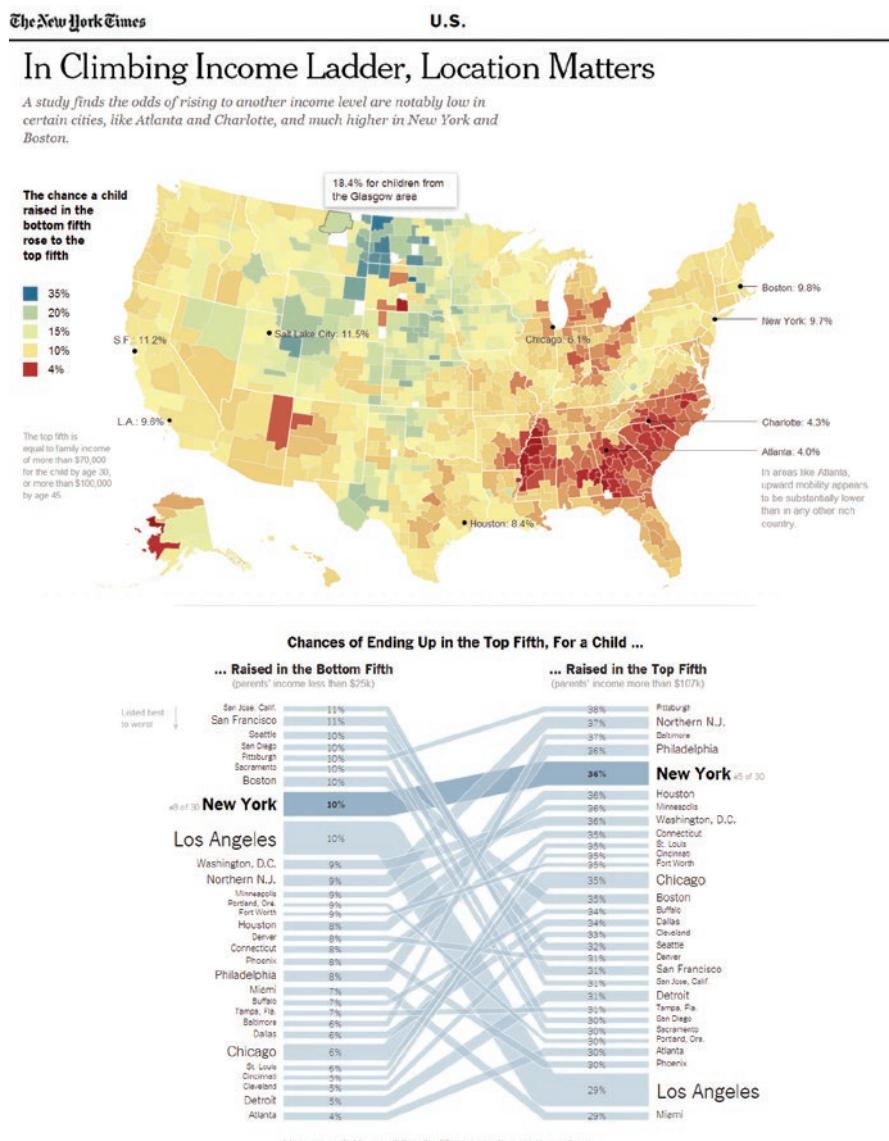


Figura 16: Nessa imagem vemos as duas visualizações presentes no artigo do New York Times. Fonte: [http://www.nytimes.com/2013/07/22/business/in-climbing-income-ladder-location-matters.html?\\_r=0](http://www.nytimes.com/2013/07/22/business/in-climbing-income-ladder-location-matters.html?_r=0)

#### 1.2.4 Design de produto

##### Luminária *Hypae* - Nervous System 2011

A empresa Nervous System é conhecida por seus produtos experimentais orgânicos que utilizam algoritmos que simulam comportamentos naturais. A luminária de mesa *Hypae* é impressa em nylon e fazer parte de uma coleção de produtos com linguagem orgânica, que abrange jóias e luminárias.

“Inspirado pelas estruturas das veias que carregam fluidos em organismos, desde as folhas de plantas até nosso próprio sistema circulatório, nós criamos uma simulação que usa princípios de crescimento físico para construir estruturas esculturais orgânicas. Começando a partir de uma semente e uma superfície, produzimos uma rede hierárquica onde nódulos são constantemente ramificados e fundidos. A estrutura densamente interconectada é ao mesmo tempo resistente e delicada.”<sup>13</sup>

Nesse contexto a programação é utilizada para criar produtos diferenciados, únicos com uma linguagem que dificilmente poderia ser produzida em larga escala utilizando ferramentas convencionais.



Figura 18: Luminária *Hiphae* e suas referências generativas. Fonte: <http://www.flickr.com/photos/jrosenk/sets/72157626136298521/> e <http://n-e-r-v-o-u-s.com/blog/?p=1701>

13 Fonte: [www.n-e-r-v-o-u-s.com/shop/generativeProduct.php?code=99](http://www.n-e-r-v-o-u-s.com/shop/generativeProduct.php?code=99)

## Layer Chair - Jens Dyvik 2011

Jens Dyvik é um designer de produto que trabalha dentro da perspectiva colaborativa e atualmente concede palestras em eventos da comunidade Fablab. O objetivo do projeto *Layer Chair* é fabricar cadeiras paramétricas através da modelagem digital utilizando o plugin Grasshopper e da manufatura utilizando uma máquina de corte CNC.

A modelagem paramétrica pelo Grasshopper “torna fácil o ajuste do perfil do assento, da largura, do comprimento, da espessura do material, entre outros. A definição do Grasshopper (assim chamado o arquivo do *plugin*) usa duas curvas do programa Rhino como dados de entrada.” O programa então “gera linhas de corte e furos de alinhamento” a fim de deixar o arquivo pronto para corte.<sup>14</sup>

Os projetos paramétricos como a Layer Chair entram dentro do contexto DIY (Do It Yourself) uma vez que ele é customizável e o ferramental necessário para a produção desses objetos é acessível.



Figura 21: Interface do Grasshopper, cadeira sendo cortada em uma máquina de corte CNC e a cadeira montada.  
Fonte: <http://www.dyvikdesign.com/site/portfolio-jens/the-layer-chair-amsterdam-edition.html>

14 Fonte: <http://www.dyvikdesign.com/site/portfolio-jens/the-layer-chair-amsterdam-edition.html>

## 1.3 HISTÓRICO DA PARAMETRIZAÇÃO

Feito um panorama geral da produção atual, consideramos importante fazer uma revisão da evolução dessa área ao longo do tempo e conhecer um pouco sobre suas origens. Neste tópico é apresentado de maneira breve alguns eventos históricos baseado no artigo “*A History of Parametric*”<sup>15</sup> do acadêmico Daniel Davis.

### 1.3.1 A parametrização na matemática

“Um dos primeiros exemplos que pude encontrar de parâmetros sendo usados para descrever modelos tridimensionais” foi o trabalho de “James Dana chamado ‘On the Drawing of Figures of Crystals’ de 1837” (DAVIS 2013) . Nele, Dana descreve os passos para se elaborar desenhos de cristais a partir de equações paramétricas.

O conceito de “parâmetro” foi originado na matemática para se referir a uma equação paramétrica, que pode ser definida como “um conjunto de equações que expressam um conjunto de quantidades como funções explícitas de um número de variáveis independentes, conhecidos como ‘parâmetros’”<sup>16</sup> (WEISSTEIN 2003, p.2150, tradução nossa).

Um exemplo de uma equação paramétrica é a fórmula que define uma curva catenária<sup>17</sup>, como vemos na figura 22, cujos parâmetros  $a$  e  $t$  definem os resultados para as coordenadas  $x$  e  $y$ .

$$\begin{aligned}x(a,t) &= t \\y(a,t) &= a \cosh\left(\frac{t}{a}\right)\end{aligned}$$

Figura 22: Fórmula da catenária. Fonte: <http://www.danieldavis.com/a-history-of-parametric>

### 1.3.2 A parametrização analógica

Antoni Gaudí (1852-1926) foi um dos precursores da “parametrização analógica” (DAVIS 2013), no qual podemos perceber características desse tipo ferramenta na construção de elementos presentes nos seus projetos arquitetônicos.

O projeto para a Igreja de *Colònia Güell* de Gaudi é um exemplo de projeto paramétrico, que infelizmente nunca foi concluído. Ele construiu um modelo em escala 1:10 composto por cordas amarradas ao teto com diversos pesos atados a elas, como vemos na figura 23. As cordas construíram estruturas baseadas na curva catenária, formando arcos e abóbadas; os pesos representavam os pontos de apoio, como colunas e interseções de parede.<sup>18</sup>

15 <http://danieldavis.com/a-history-of-parametric>

16 “set of equations that express a set of quantities as explicit functions of a number of independent variables, known as ‘parameters’”

17 A catenária descreve uma família de curvas planas semelhantes às que seriam geradas por uma corda suspensa pelas suas extremidades e sujeitas à ação da gravidade. Fonte: <http://pt.wikipedia.org/wiki/Catenária>

18 [http://en.wikipedia.org/wiki/Antoni\\_Gaudi](http://en.wikipedia.org/wiki/Antoni_Gaudi)



Figura 23: Modelo invertido para o projeto da Igreja de *Colònia Güell*. Fonte: [http://en.wikipedia.org/wiki/Antoni\\_Gaudí](http://en.wikipedia.org/wiki/Antoni_Gaudí)

### 1.3.3 O Sketchpad

Depois de Gaudí, um outro personagem histórico importante para o desenho paramétrico foi Ivan Sutherland que, com o objetivo de criar um sistema digital capaz de fazer “um homem e um computador conversar”, desenvolveu o projeto *Sketchpad*<sup>19</sup>, o primeiro programa interativo de desenho assistido por computador (do inglês, computer-aided design, CAD) em 1963.

O *Sketchpad* utilizava uma caneta de luz no qual o “designer podia desenhar linhas e arcos que se relacionavam entre si com, o que Sutherland chamou, restrições atômicas”, que eram baseadas no mesmo princípio das equações paramétricas, sendo que o designer podia modificar os parâmetros livremente para obter formas variadas e relações diversas como paralelismo, ortogonalidade e coincidência. (DAVIS, 2013)

19 <http://en.wikipedia.org/wiki/Sketchpad>

### 1.3.4 Era digital

O que aconteceu nas décadas seguintes foi um desenvolvimento acelerado de novos programas. Em 1988 foi lançado o PRO/ENGINEER, o primeiro programa comercial tridimensional baseado no desenho paramétrico. Em 1994, o Catia V4 apareceu, incorporando elementos paramétricos do PRO/ENGINEER. Em 2000 o programa Revit foi lançado no mercado apresentando um novo conceito de modelagem paramétrica chamada de Modelagem das Informações de Construção (do inglês, Building Information Modelling, BIM). Em 2010, o AutoCAD 2010 tardiamente adicionou funções paramétricas, dezoito versões depois de seu primeiro lançamento.

Em paralelo a essa geração de *software*, interfaces de *script* também foram desenvolvidas. A necessidade de se criar tais programas foi pensada para evitar que aplicações e funções específicas tivessem que ser adicionadas aos programas principais. Cada vez mais, funções customizadas foram sendo exigidas para atender projetos heterogêneos, por isso, esses programas de *script* ganharam espaço, como o Generative Components, lançado em 2003, o Grasshopper (figura 24), em 2007, além de muitos outros, como o Processing lançado em 2001 voltado para a área visual.

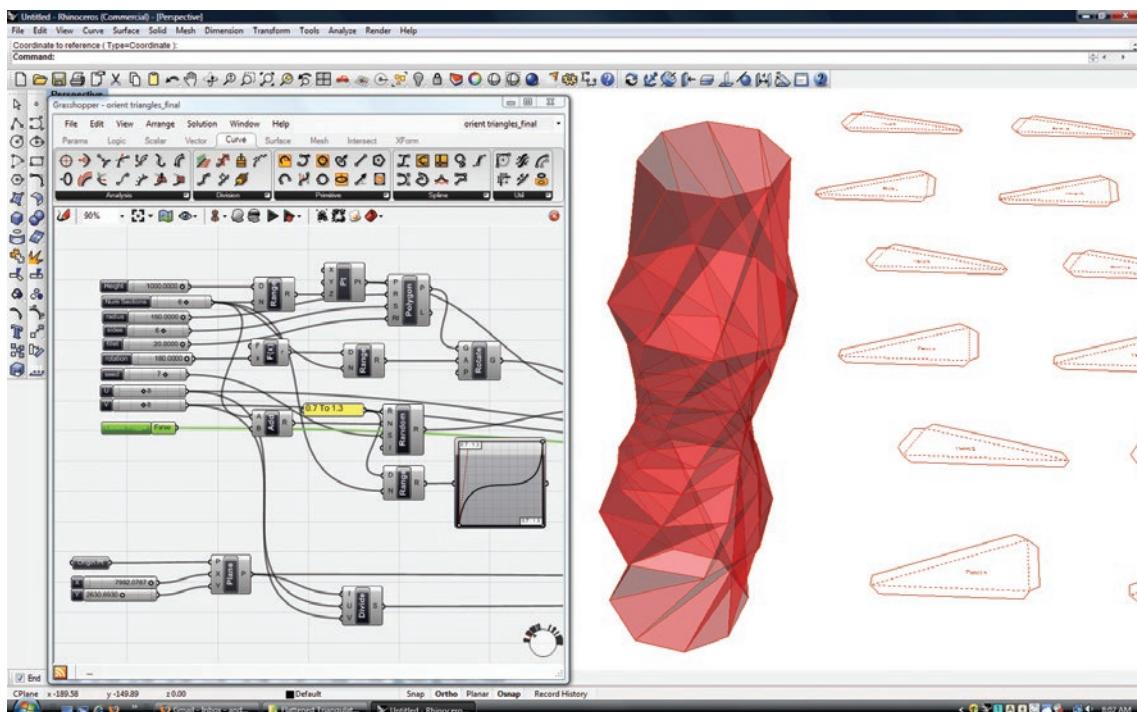


Figura 24: Interface do *plugin* de script Grasshopper em primeiro plano e do programa Rhinoceros em segundo plano.  
Fonte: <http://mlab.cca.edu/2009/08/fall-2009-workshop-intermediate-grasshopper-modeling-in-rhino/>

## 1.4 TECNOLOGIA

A partir do histórico podemos observar que nos últimos anos ocorreu um desenvolvimento sucessivo de *software* especializado em parametrização. Isso mostra que a importância do desenho paramétrico como ferramenta de projeto vem aumentando ao longo do tempo.

Os projetos realizados pelo enfoque do design paramétrico, mesmo que possíveis antes da existência do computador, só foram viabilizados em termos práticos devido aos

avanços tecnológicos relacionadas a computação, que possibilitou a criação de novas ferramentas disponíveis ao designer.

Hoje em dia, ferramentas digitais são criadas para os designers e pelos designers. O princípio de código aberto (do inglês, *open source*) introduz uma nova forma de desenvolvimento de projeto, que exploraremos a seguir. Além disso, com a disseminação da internet, ambientes colaborativos são criados, facilitando a divulgação, discussão e troca de conhecimento.

Dentro do design paramétrico existe uma alta rotatividade das ferramentas digitais utilizadas, com linguagens de programação, plugins, bibliotecas e programas que são constantemente atualizados. No entanto alguns princípios básicos se mantêm inalterados. Por este motivo focaremos nossa análise nos princípios tecnológicos, e não nos detalhes das ferramentas e nem nas linguagens de programação disponíveis hoje.

#### **1.4.1 Cultura open source**

De acordo com a Wikipédia, o termo código aberto “refere-se a um programa de computador no qual seu código-fonte está disponível para o público em geral para uso e/ou modificação do seu projeto original. O código-fonte aberto é tipicamente criado em um esforço colaborativo no qual programadores melhoram o código e compartilham as mudanças com a comunidade.”<sup>20</sup>

A “cultura *open source*” é um termo mais abrangente que engloba outras áreas além da computação, é “a prática criativa de apropriação e livre compartilhamento de conteúdos que foram encontrados ou criados”.

Dentro do cenário da cultura *open source* existe o conceito de *open design* que, segundo a Wikipédia, é “o desenvolvimento de produtos físicos, máquinas e sistemas através do uso de informações de projeto compartilhadas publicamente”, no qual seus “objetivos e filosofias são idênticas ao do movimento *open source*, mas são implementadas no desenvolvimento de produtos físicos ao invés de *software*. *Open design* é uma forma de cocriação, no qual o produto final é projetado por usuários e não por investidores externos, como empresas privadas.”<sup>21</sup>

O assunto da cultura *open source* é demasiadamente vasto para o escopo deste trabalho, porém, um ponto importante a ser dito é que várias ferramentas utilizadas por designers são desenvolvidas baseadas nesse conceito, inclusive o Processing<sup>22</sup>, ferramenta de programação que será melhor explicada no capítulo 4.

#### **1.4.2 Ambientes colaborativos**

Muitas ferramentas *open source* estão vinculadas a ambientes colaborativos, onde acontece o compartilhamento de conhecimento entre os usuários desses sistemas. O *OpenProcessing*<sup>23</sup>, por exemplo, é a comunidade on-line que serve como plataforma para se compartilhar e discutir programas criados no Processing.

20 [n.wikipedia.org/wiki/Open\\_source](https://pt.wikipedia.org/wiki/Open_source)

21 [en.wikipedia.org/wiki/Open\\_design](https://en.wikipedia.org/wiki/Open_design)

22 [www.processing.org](https://www.processing.org)

23 [www.openprocessing.org](https://www.openprocessing.org)

Existem diversas comunidades on-line com o mesmo perfil colaborativo, sendo que muitas delas não são necessariamente vinculadas a área da programação. A comunidade *Thingiverse*<sup>24</sup>, por exemplo, está relacionada ao compartilhamento de arquivos de objetos a serem produzidos usando a tecnologia de impressão 3D ou de corte CNC, baseada no conceito de *open design*. Outro exemplo é a comunidade *Instructables*<sup>25</sup>, que está relacionada ao compartilhamento de projetos de praticamente qualquer coisa, desde produtos eletrônicos utilizando o arduino até a fabricação artesanal de canoas e receitas culinárias.

As comunidades colaborativas não precisam ser necessariamente on-line. Um exemplo disso é o projeto Fablab<sup>26</sup>, conceito criado por Neil Gershenfeld, diretor do *Center for Bits and Atoms* do MIT. O Fablab é um laboratório de fabricação onde são feitas oficinas, cursos e palestrar voltados a prática e ensino da fabricação digital com o intuito de democratizar o conhecimento nessa área.

## 1.5 METODOLOGIA

Esse trabalho é apresentado em sete capítulos principais. O primeiro já tratou de introduzir o tema, explicar a proposta de trabalho, definir termos importantes, ilustrar e descrever um panorama do que está sendo produzido atualmente, expor um histórico relacionado à área estudado e por último apresentar aspectos tecnológicos relevantes a área de estudo.

O segundo capítulo apresenta um estudo de caso a fim de proporcionar uma visão global sobre um método de projeto envolvendo programação e a fim constatar a existência de uma conversa reflexiva no processo de design, conceito proposto por Donald Schön e esclarecido no terceiro capítulo.

O capítulo três trata de mostrar como a programação é incorporada pelo processo de design. Para isso é explicado o conceito de algoritmo, o conceito da própria programação e o conceito de processo reflexivo.

O capítulo quatro traz resultados de um estudo empírico e exploratório realizado durante o TCC2 no qual experimentos gráficos foram realizadas a fim de obter conclusões sobre o processo de design de forma prática. Para isso foram definidos alguns limitantes com a finalidade de se ter um melhor controle sobre o processo de criação.

No capítulo cinco são discutidos os pontos que surgiram a partir da reflexão e da comparação entre a exploração prática e os conceitos teóricos abordados nos capítulos anteriores.

O capítulo seis mostra uma terceira fase de experimentação no qual nós propomos criar uma aplicação dos conhecimentos adquiridos que une a tecnologia apresentada no capítulo quatro com a área da programação visual.

O capítulo sete conclui o trabalho, avaliando tudo o que foi realizado.

24 [www.thingiverse.com](http://www.thingiverse.com)

25 [www.instructables.com](http://www.instructables.com)

26 [www.fablabbrasil.org](http://www.fablabbrasil.org)

## 2. ESTUDO DE CASO [PROCESS COMPENDIUM]

Decidimos apresentar um estudo de caso a fim de proporcionar um exemplo sobre um processo de design envolvendo programação. Para isso, foi escolhido um projeto em que o processo metodológico estivesse explícito. O autor do estudo destrincha sua metodologia de protejo de maneira didática, já que o foco do trabalho é averiguar como o processo influencia o resultado final.

Antes de expor o método, primeiro é feita uma introdução sobre o autor acompanhada de uma explicação geral sobre o projeto. Em seguida, a metodologia é dividida em dois capítulos, um que aborda o conceito de “Elemento”; e o outro de “Processo”.

### 2.1 SOBRE O AUTOR

Casey Reas possui um mestrado pelo MIT e atualmente leciona na Universidade da Califórnia, LA. Ele, junto com Ben Fry, desenvolveram a linguagem e o ambiente de programação Processing, que atualmente é a principal ferramenta de trabalho de Reas. Além de lecionar, Reas é artista digital e utiliza a programação tanto como ferramenta quanto como objeto de estudo.

### 2.2 SOBRE O PROJETO

O projeto *Process Compendium*, como o nome já diz, é um compêndio de processos que descrevem os programas desenvolvidos por Reas ao longo de sete anos, entre 2004 e 2010. Ao todo, foram realizados quinze trabalhos de maneira sequencial. Cada trabalho é chamado de *Process*. O *Process 6*, por exemplo, é o sexto trabalho do compêndio, o *Process 7* é o sétimo e assim por diante. O conjunto começa com o *Process 4<sup>1</sup>* e segue até o *Process 18*.

Segundo a descrição do autor, “o fenômeno de emergência é o núcleo da exploração e cada arte se constrói sobre os trabalhos anteriores e também informa o que vem em seguida” (REAS 2012). Cada programa é processado em função do tempo, portanto eles são dinâmicos, como uma animação.

O próprio nome do projeto revela que o processo de design é o fator que é valorizado e que está em evidência em suas obras, visto que o trabalho trata da influência do método criativo no produto realizado. O autor apresenta um perfil pedagógico ao querer descrever como os artefatos<sup>2</sup> finais foram alcançados.

Além do conteúdo apresentado neste capítulo, dentro do projeto foram gerados diversos artefatos como impressões, esculturas e performances visuais, todos expostos em museus e galerias.

1 Os três primeiros processos são chamados de Structures 1, 2 e 3 e fazem parte de um projeto anterior.

2 Termo usado pelo autor para se referir aos produtos gerados pelos programas desenvolvidos por ele.

## 2.3 ELEMENTO = FORMA + COMPORTAMENTO(S)

Casey Reas começa seu trabalho definindo um agente estruturado, chamado por ele de “Elemento”, que interage com o ambiente e é responsável pela criação das composições gráficas. Segundo Reas, “um Elemento é uma máquina simples que é composta por uma Forma e por um ou mais Comportamentos”<sup>3</sup> (REAS 2012). A Figura 25<sup>4</sup> descreve a estrutura do Elemento 1, que é constituído por uma Forma, o círculo, e quatro Comportamentos (do inglês, Behaviors).

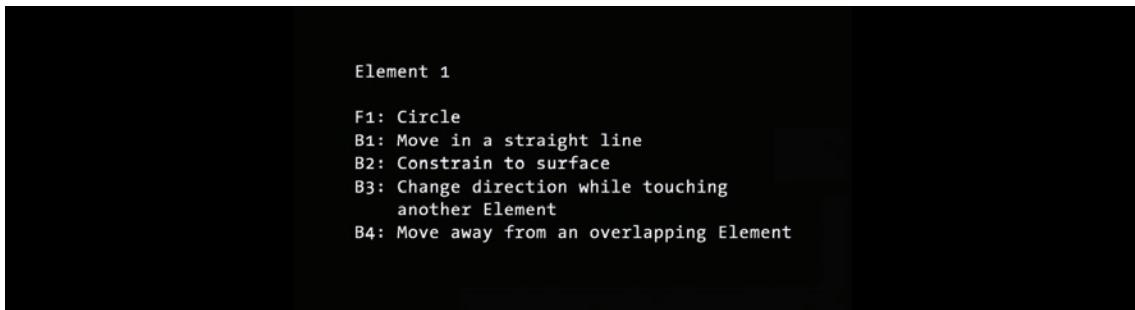


Figura 25: Descrição do Elemento 1

O diagrama 26 mostra um quadro daquilo que seria o Elemento 1 em ação. Nele são desenhados as Formas, no caso círculos, e seus respectivos Comportamentos, representados por setas. O Comportamento 1 é representado pelas setas que vão do centro até as bordas de cada círculo e atuam como a direção do movimento. O Comportamento 2 evita que os círculos saiam para fora do quadro. O Comportamento 3 são as setas curvas que mostram a mudança de direção de um círculo ao tocar outro. O Comportamento 4 são as pequenas setas que aparecem quando duas ou mais Formas são sobrepostas, representando o afastamento das formas em sentido oposto ao choque.

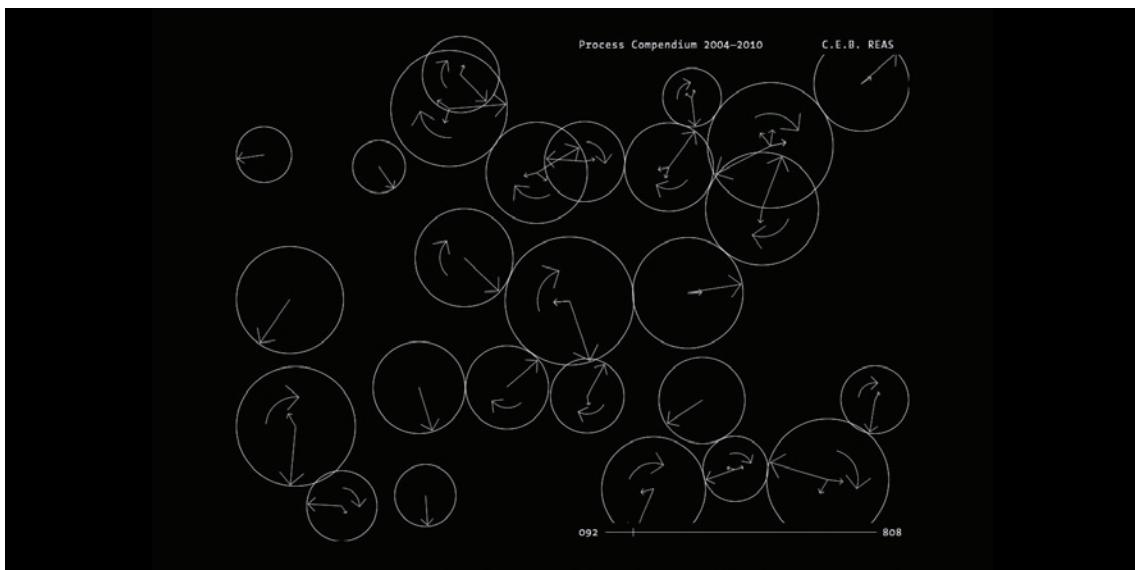


Figura 26: Diagrama do Elemento 1 em ação

3 “An Element is a simple machine that is comprised of a Form and one or more Behaviors.”

4 Todas as imagens do projeto apresentadas aqui foram retiradas do vídeo “Process Compendium (Introduction)”. Disponível em: <http://vimeo.com/22955812>

O Elemento 2 é uma versão simplificada do Elemento 1, onde círculos se movem em linha reta mas sem nenhum tipo de choque ou mudança de direção. Ao sair por uma das bordas tela, o Elemento aparece na borda oposta com a mesma direção e sentido.

O Elemento 3, descrito na figura 27, parte agora de uma outra Forma, a linha. Os Comportamentos 2 e 4 foram retirados e um quinto Comportamento foi adicionado.

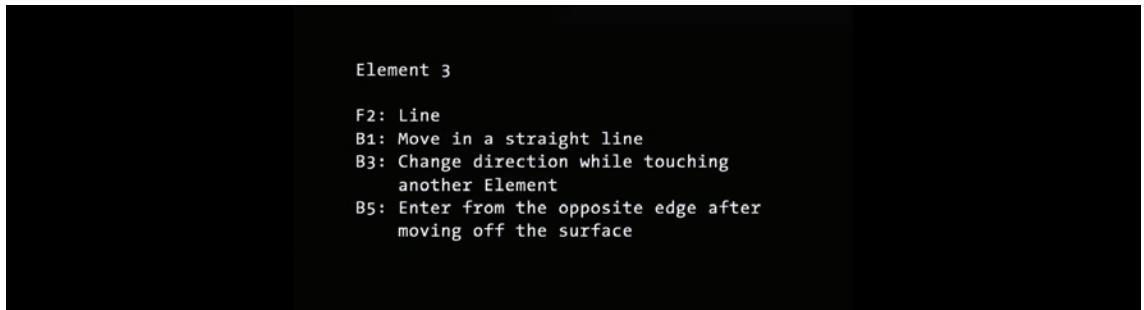


Figura 27: Descrição do Elemento 3

No diagrama 28, que representa a ação do Elemento 3, a seta linear representa tanto a Forma quando o Comportamento 1. O Comportamento 3 é representado pela mesma seta curva e o Comportamento 5 faz com que, ao sair por uma das bordas tela, o Elemento aparece na borda oposta com a mesma direção e sentido.

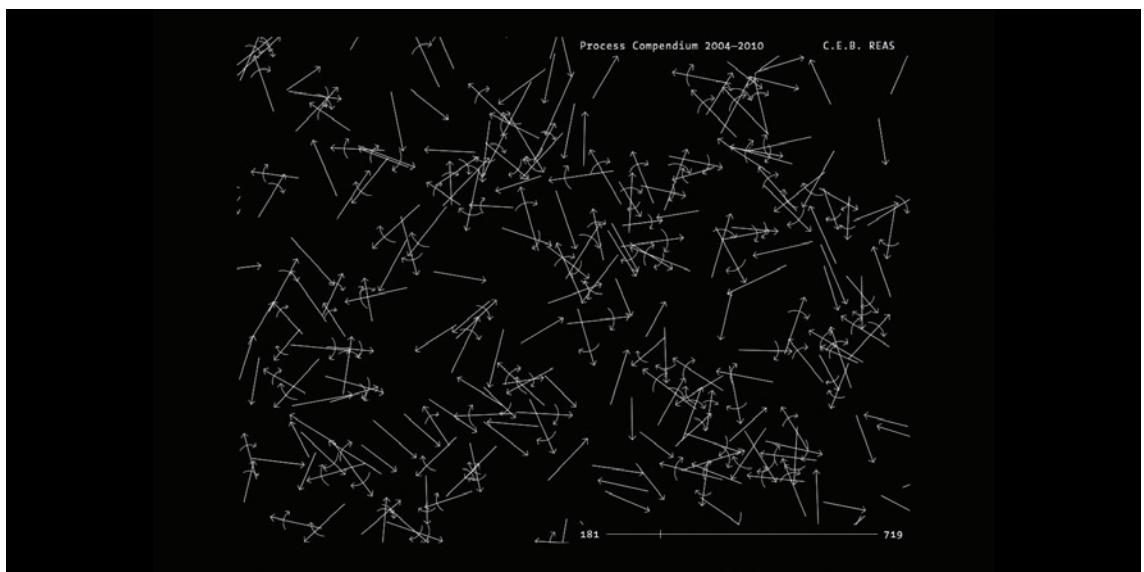


Figura 28: Diagrama do Elemento 3 em ação

Ao todo, Reas criou cinco elementos diferentes, cada um com Formas e Comportamentos específicos. Ele reuniu todos os componentes em um registro que ele chamou de “biblioteca”, apresentado na figura 29. Nela é possível ver quais Formas e quais Comportamentos estão presentes em cada Elemento.

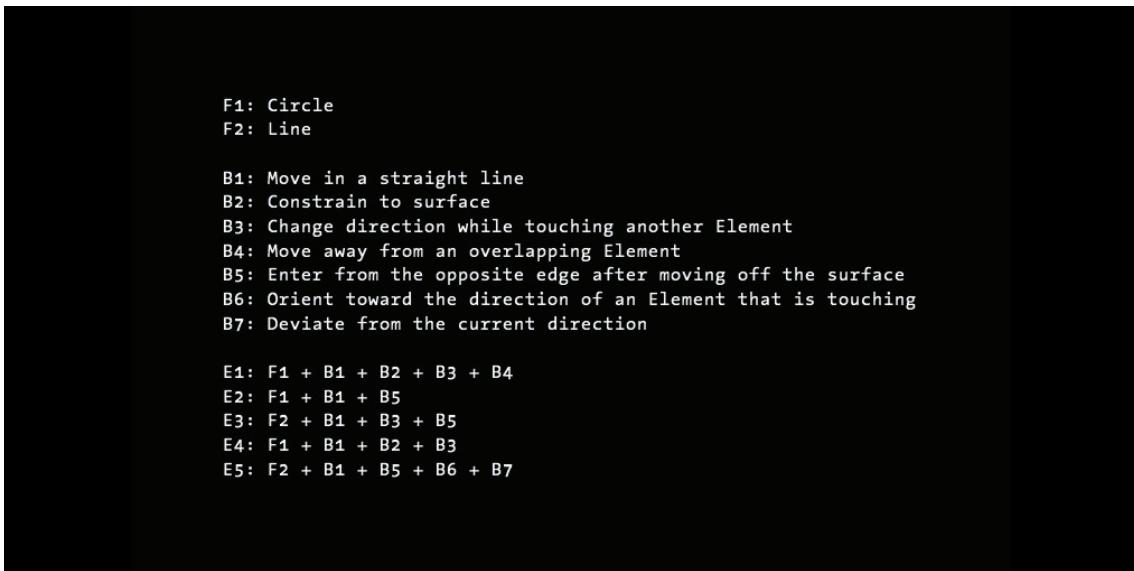


Figura 29: Biblioteca de Formas, Comportamentos e Elementos

A ação dos Elementos por si só não representa a aparência da composição final. Eles são apenas agentes animados responsáveis em gerar efeitos gráficos somente quando um Processo for definido. Até agora só foi possível observar como os Elementos interagem entre si.

## 2.4 SOBRE PROCESSO

Seguindo com a metodologia de Reas, após definir os Elementos, apresentamos aqui a definição de Processo dada pelo autor, que é o grande foco do projeto. Segundo ele:

“Cada Processo é um texto curto que define um espaço a se explorar através de múltiplas interpretações. Um interpretação de um Processo em *software* é uma máquina de desenho cinético que apresenta início, mas sem fim definido. Ele leva um passo de cada vez, sendo que a cada passo cada Elemento se modifica de acordo com seus Comportamentos. As formas visuais correspondentes emergem na medida em que um Elemento se modifica; cada ajuste é incrementado às formas previamente desenhadas.”<sup>5</sup> (REAS 2012, tradução nossa)

Os frutos desse projeto podem ser classificados em dois tipo, o Processo em si e os artefatos criados a partir das interpretações desse Processo feitas pelo *software*. Sobre Processo é necessário dizer que “O mais importante elemento do Processo é o texto. O texto é o Processo descrito em inglês, escrito com a intenção de traduzir seu conteúdo para o *software*”. Nesse contexto, “a interpretação do *software* é secundária para o texto. O texto deixa muitas decisões abertas para o programador, decisões que devem ser feitas usando o julgamento pessoal. O ato de traduzir o Processo do Inglês para uma linguagem de máquina interpreta o texto”<sup>6</sup>. A figura 30 mostra o Processo 4.

5 “Each Process is a short text that defines a space to explore through multiple interpretations. A Process interpretation in software is a kinetic drawing machine with a beginning but no defined end. It proceeds one step at a time, and at each discrete step, every Element modifies itself according to its behaviors. The corresponding visual forms emerge as the Elements change; each adjustment adds to the previously drawn shapes.”

6 “The most important element of Process is the text. The text is the Process described in English, written with the intent to translate its content into software” | “The software interpretation is secondary to the text. The text leaves many decisions open to the programmer, decisions that must be made using personal judgment. The act of translating the Process from English into a machine language interprets the text.”

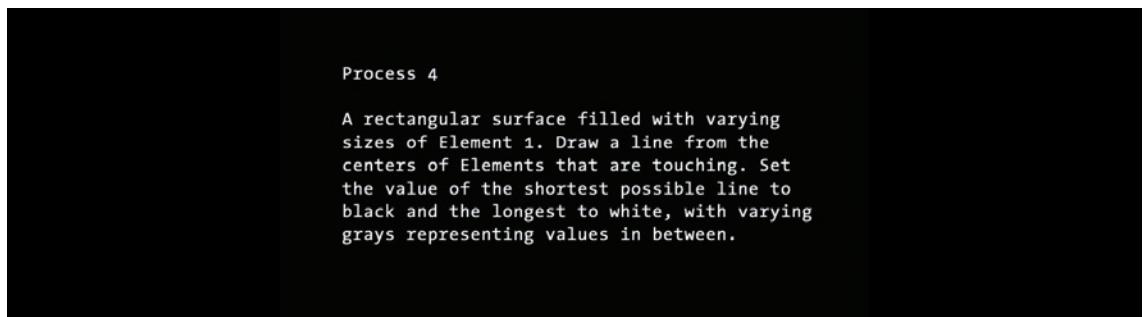


Figura 30: Processo 4

A figura 31 mostra um quadro daquilo que seria uma interpretação do Processo 4 feita pelo *software* em ação. Formas se movem de acordo com os Comportamentos definidos nos Elementos e a composição é construída a partir da interação entre esses agentes, sem um fim definido.

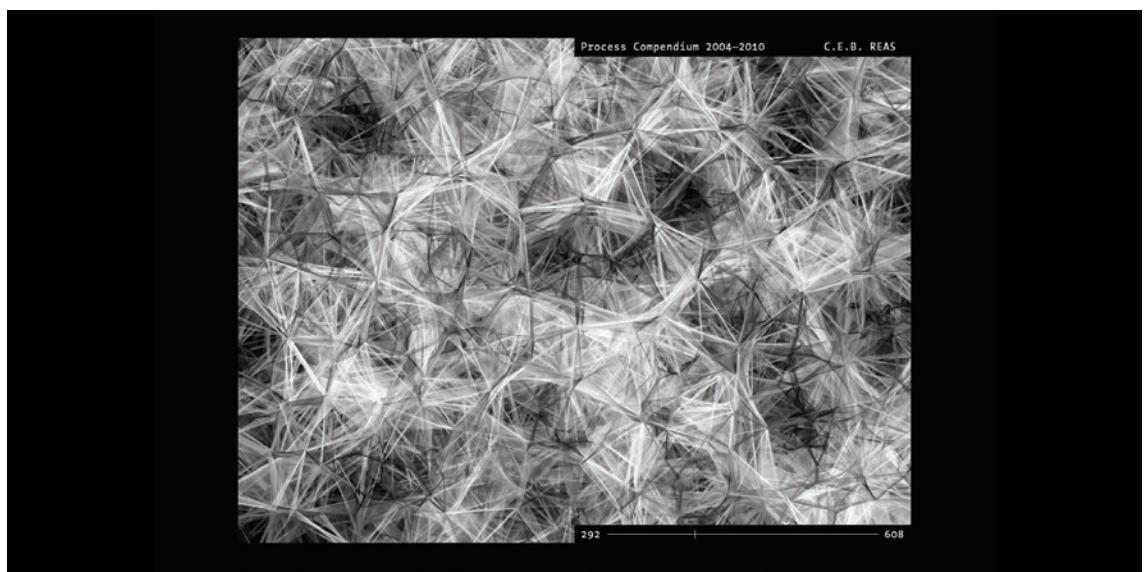


Figura 31: Interpretação do Processo 4

A figura 32 mostra o Processo 18, o último gerado por Casey Reas no projeto.

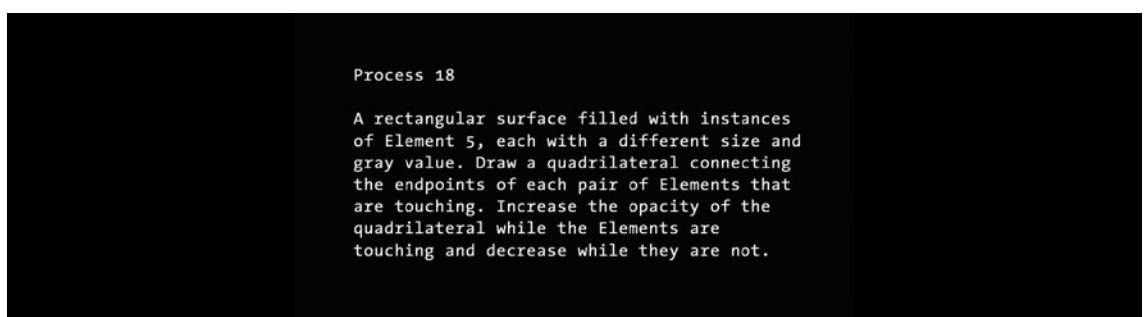


Figura 32: Processo 18

Como dito anteriormente, um Processo pode ter mais de uma interpretação, gerando diferentes possibilidades de resultado. A figura 33 mostra uma das infinitas interpretações possíveis pelo *software* e a figura 34 mostra uma outra variação, feita especificamente para ser impressa.

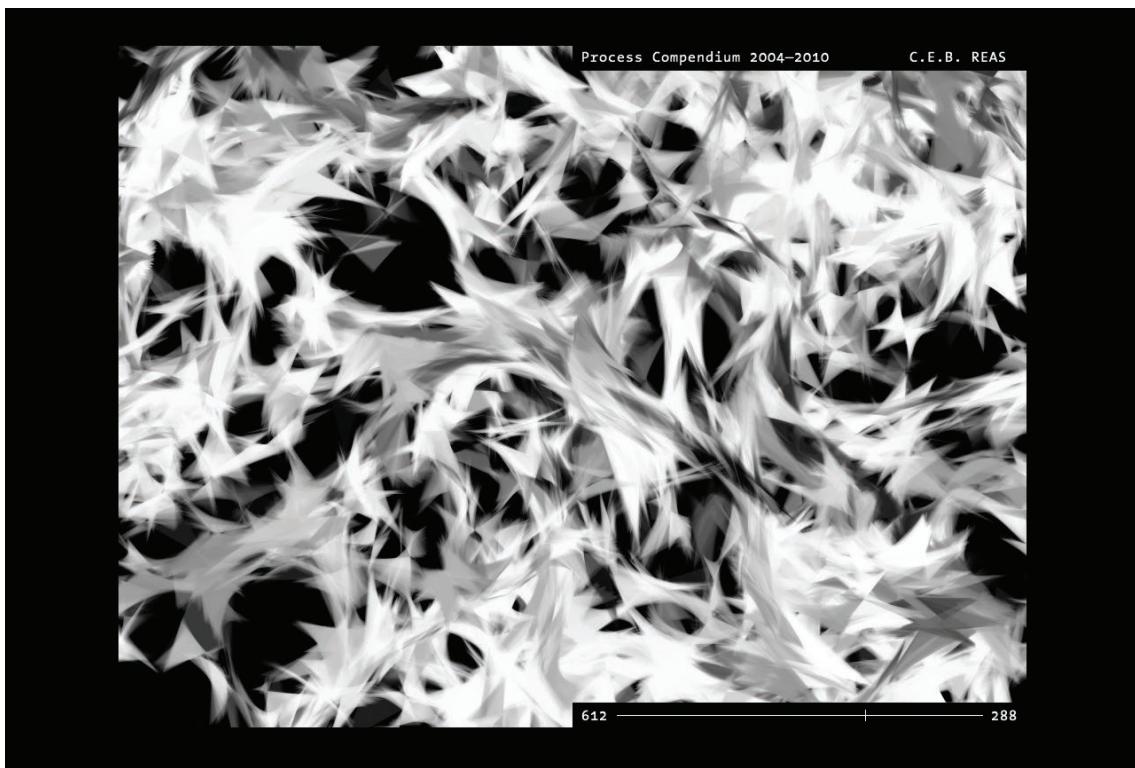


Figura 33: Possível interpretação do Processo 18

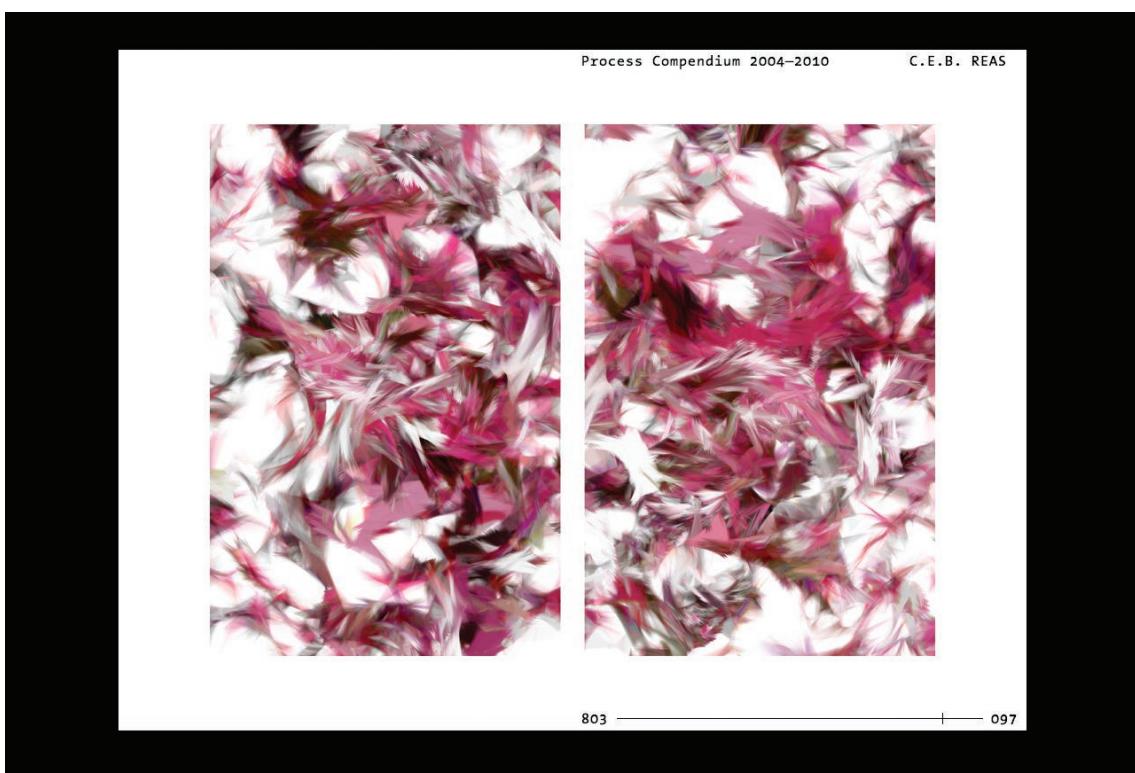


Figura 34: Outra possível interpretação do Processo 18

É possível concluir que o método usado por Reas para esse projeto generativo segue os seguintes passos: definição de agentes, definição de processos e interpretação do *software*. A estrutura metodológica é simples e permite a múltipla combinação entre seus componentes para se gerar resultados inéditos.

### 3. PROGRAMAÇÃO NO PROCESSO REFLEXIVO DO DESIGN

A partir da apresentação do estudo de caso, em que é demonstrado uma metodologia de projeto, é possível comprovar alguns conceitos presentes no processo de design que envolve a programação. Apesar desse processo ser explícito no estudo de caso, existe uma estrutura de pensamento por trás desse processo que não é demonstrado no projeto, mas que é intrínseco ao campo de design, que é chamado de conversa reflexiva.

Para explicar o processo de design e a conversa reflexiva no design paramétrico, primeiro é preciso entender a importância do algoritmo e quais são os elementos principais na programação que são relevantes ao design.

#### 3.1 ALGORITMO

Segundo a Wikipédia, “Um algoritmo é uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais pode ser executada mecanicamente num período de tempo finito e com uma quantidade de esforço finita”<sup>1</sup>. Uma definição mais simples é encontrada no dicionário *Oxford*, que estabelece algoritmo como ‘um processo ou conjunto de regras a ser seguido no cálculo ou em outras operações de resolução de problema, especialmente por um computador’.<sup>2</sup>

O algoritmo não é usado somente no contexto da computação. Ele define qualquer sequência de instruções, como uma receita de bolo ou um manual de montagem. Algumas obras de arte conceituais como os trabalhos de Yoko Ono (figura 35) e de Sol Lewitt (figura 36), podem ser classificadas como algoritmos por serem um conjunto de instruções. Nesse tipo de arte, o que está em foco é o método, assim como o projeto de Casey Reas.

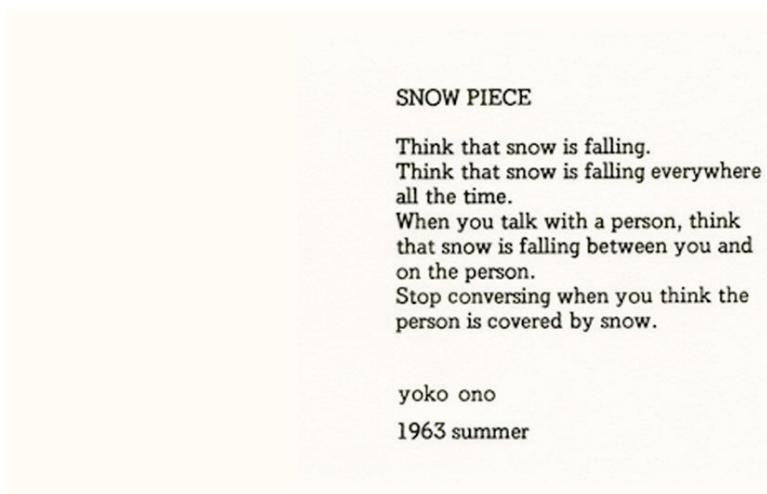


Figura 35: *Snow Piece* por Yoko Ono tirado do livro *Grapefruit*, 1964. Fonte: [likeafielmouse.com/post/41386650242/yoko-ono-grapefruit-1964](http://likeafielmouse.com/post/41386650242/yoko-ono-grapefruit-1964)

1 Fonte: [pt.wikipedia.org/wiki/Algoritmo](http://pt.wikipedia.org/wiki/Algoritmo)

2 Fonte: [www.oxforddictionaries.com/definition/english/algorithm?q=algorithm](http://www.oxforddictionaries.com/definition/english/algorithm?q=algorithm)

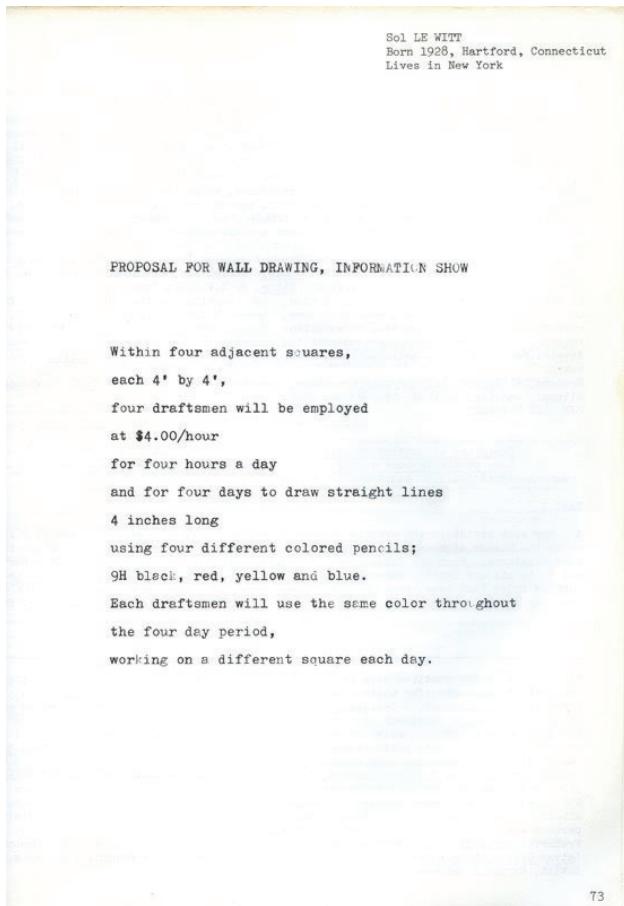


Figura 36: *Proposal for Wall Drawing, Information Show* por Sol Lewitt, 1970. Fonte: journal.mattniebuhr.com/2010/05/29/proposal-for-wall-drawing-information-show-sol-le-witt

Pode-se dizer então que o foco do projeto de Reas é a produção de algoritmos, visto que os Processos são instruções finitas e definidas a serem interpretadas e codificadas pelo programador (as duas palavras são expressões sinônimas, inclusive). O algoritmo é parte fundamental da programação e, portanto, do processo de design paramétrico.

No contexto da programação, o processo de transformar e segmentar uma ideia em instruções não é um processo intuitivo ao ser humano e por isso exige um alto grau de abstração. Essa abstração é um procedimento que precisa ser praticado com frequência para se tornar um hábito na atividade do designer.

### 3.2 PROGRAMAÇÃO

A programação está relacionada a codificação de um algoritmo utilizando uma linguagem de programação. Essa linguagem, como no caso do Processing, funciona com base em um modelo estrutural lógico que é inteligível para o computador. Esse modelo<sup>3</sup> possui quatro propriedades que a linguagem de programação deve seguir. São elas:

- 1. Ordem sequencial:** as linhas de código são lidas pelo computador e são executadas de maneira sequencial e segmentada.
- 2. Condicionais:** condições são elaboradas para direcionar e restringir as ações realizadas pelo programa.

3 O modelo mencionado refere-se ao paradigma de programação imperativa. Fonte: [http://pt.wikipedia.org/wiki/Programma%C3%A7%C3%A3o\\_imperativa](http://pt.wikipedia.org/wiki/Programma%C3%A7%C3%A3o_imperativa)

**3. Iterações:** instruções são repetidas até se obter o resultado desejado.

**4. Variáveis:** dados variáveis são implementados para proporcionar cálculos dinâmicos dentro do código a fim de causar algum efeito durante sua execução.

No estudo de caso em nenhum momento foi falado sobre linguagem de programação ou foi exibido qualquer linha de código. O autor do estudo decidiu omitir a parte técnica e definiu o termo genérico ‘interpretação do *software*’ para englobar a codificação dos Processos feita pelo programador, no caso o próprio Reas, a partir de seu julgamento e interpretação.

### 3.3 CONVERSA REFLEXIVA DO PROCESSO DE DESIGN

Donald Schön defende em seu artigo “*Designing as Reflective Conversation with the Materials of a Design Situation*” que o processo de design é uma conversa reflexiva com o problema a ser resolvido. “Não há caminho direto entre a intenção do designer e o resultado. Na medida em que você trabalha um problema, você está constantemente no processo de desenvolver um caminho para ele, formando novas avaliações e entendimentos na medida em que realiza ações”<sup>4</sup> (SCHÖN 1992, tradução nossa).

Continuando esse raciocínio, “designers estão em negociação com uma situação de design. Eles respondem às demandas e possibilidades de uma situação de design que, por sua vez, ajudam a criar”<sup>5</sup> (SCHÖN 1992, tradução nossa).

Do pensamento proposto por Schön é possível concluir que a resolução de um problema através do design acontece de maneira dinâmica, no qual o designer avalia um determinado problema, realiza uma ação e volta a analisar o problema em um processo repetido continuadamente. O problema, ao ser modificado, gera novos questionamentos, que consequentemente geram novas resoluções.

No contexto do design paramétrico essa conversa reflexiva permanece. “O design executado pela programação é parte da conversa reflexiva, então durante a sua iteração ele é analisado pelo designer, que abstrai padrões dele e usa esses padrões para formar a estrutura do seu design, usando ela para modificar o código. O código é então usado para executar uma nova versão do design”<sup>6</sup> (CARLI; BARROS 2012, tradução nossa).

Para se entender melhor essa conversa, uma representação genérica do processo de design envolvendo programação é apresentando na figura 37. O designer abstrai uma ideia em um algoritmo e o formaliza em um código utilizando uma linguagem de programação. O computador interpreta o código e gera um resultado. Esse resultado é julgado pelo designer, que pode alterar tanto o algoritmo como o código-fonte. O ciclo acontece até o resultado satisfazer a ideia, que pode ou não ser a mesma ideia inicial, já que durante a conversa reflexiva o resultado pode alterar a resolução do problema.

4 “There is no direct path between the designer’s intention and the outcome. As you work a problem, you are continually in the process of developing a path into it, forming new appreciations and understandings as you make new moves.”

5 “Designers, I shall argue, are in transaction with a design situation; they respond to the demands and possibilities of a design situation, which, in turn, they help to create.”

6 “The design executed by the programming is part of the reflective conversation, so during its loop it’s analyzed by the designer, who abstract patterns from it and use these patterns to form the structure of his design, which he uses to modify the code. The code is then used to execute a new version of the design”

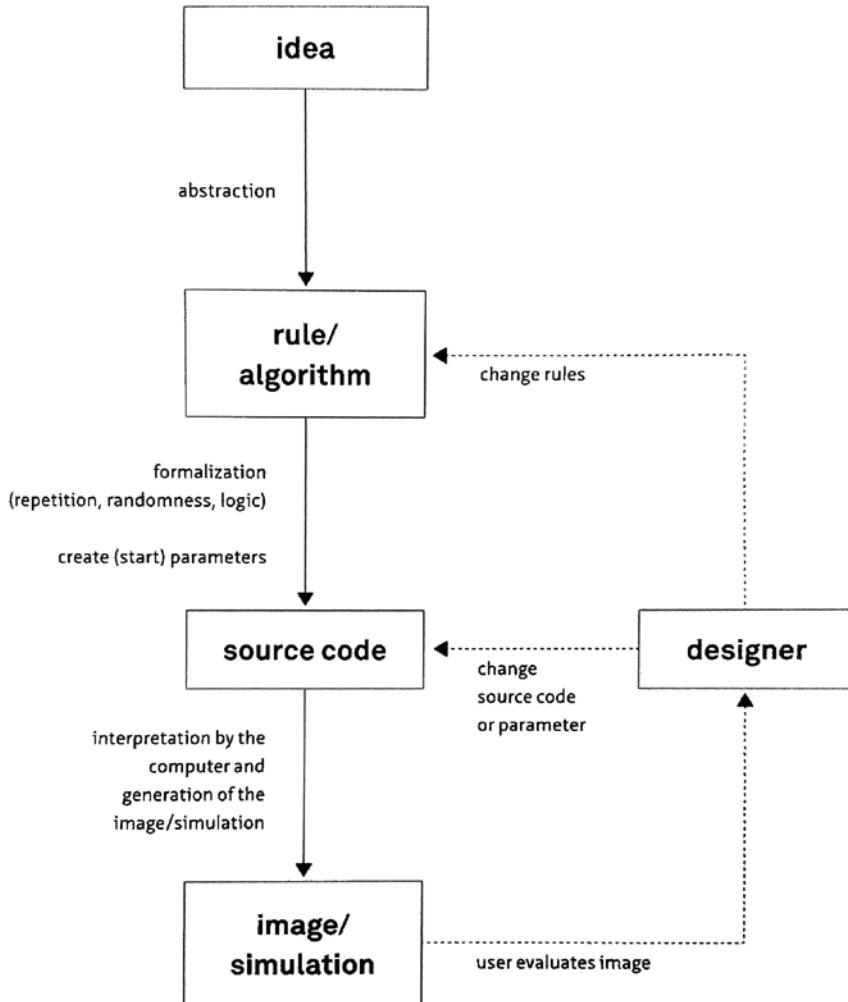


Figura 37: Representação do processo de design que inclui a programação. Fonte: BOHNACKER, 2012 p.461

De acordo com a representação, o resultado “precisa primeiro ser descrito completamente utilizando um conjunto de regras - uma camada sistemática e intermediária”. No entanto, a introdução da programação no processo de design “gera dois desafios: abstrair uma vaga ideia”, ou seja, criar um algoritmo, “e colocar uma ideia no computador de um jeito formalizado” (BOHNACKER, 2012), ou seja, codificar procedimentos utilizando linguagem de programação.

## 4. EXPERIMENTOS 2.0

Como parte essencial do trabalho, decidimos realizar uma série de experimentos para investigar e analisar a influência e a participação da programação dentro do processo de design e também para explorar as possibilidades da programação como ferramenta. Por fim uma discussão foi realizada a fim de elaborar uma visão crítica sobre o processo criativo como um todo, desde os estudos iniciais até a avaliação do resultado obtido.

Durante o TCC1, uma série de experimentos iniciais foram realizados para se ter um primeiro contato com a utilização da programação no processo de design. Visando o melhor fluxo de leitura e a melhor compreensão desse trabalho como um todo, esses experimentos iniciais junto com os conceitos utilizados para fundamentá-los foram colocados como apêndice neste relatório, *capítulo 9 Apêndice - Experimentos 1.0*.

Neste capítulo serão apresentados os experimentos realizados durante o TCC2. Ao todo foram gerados vinte programas diferentes, cada um deles criando peças gráficas distintas. Os experimentos apresentados estão em ordem de idealização e possuem uma nomenclatura do tipo “\_nome” para criar uma padronização e para facilitar quando formos fazer referência a eles. Todos os programas executáveis estão presentes no CD anexado ao relatório e recomendamos que eles sejam visualizados para compreender melhor aquilo que está relatado e discutido no relatório. As imagens apresentadas aqui são frames retirados desses programas em execução.

Os programas a seguir foram feitos utilizando a linguagem de programação Processing. Processing é uma linguagem de programação de código aberto, um ambiente de desenvolvimento integrado (IDE) e uma comunidade on-line desenvolvida por Casey Reas e Ben Fry, antigos membros do Grupo de Computação do ‘MIT Media Lab’. Essa linguagem é focada na pesquisa e produção de arte eletrônica por pessoas que são desde leigos no universo da programação até peritos no assunto.

Para realizar os experimentos, definimos alguns critérios e limitantes que serviram de diretrizes para guiar seu desenvolvimento e para que eles não saíssem do escopo desse trabalho.

### 4.1 DEFINIÇÕES INICIAIS

A programação é uma área extensa e permite que técnicas variadas possam ser utilizadas pelo programador para se atingir um único resultado. Dentro do contexto deste trabalho, que tem caráter exploratório e experimental, o excesso de técnicas e a vasta abertura para a criação de programas gráficos podem tornar a atividade desconexa, incoerente e desassociada.

Pensando nisso, foi necessário criar um ponto de partida para a definição de um escopo com a finalidade de criar uma integração, mesmo que sutil, entre os experimentos e de guiar o desenvolvimento dos experimentos dentro do enfoque amplo que é o design paramétrico.

Já que o trabalho lida com uma mídia virtual que permite a criação de animações e de peças gráficas em movimento, pensamos em tirar o melhor proveito desse meio, por isso definimos que os experimentos seriam **dinâmicos**. Decidimos não incorporar a interatividade com o usuário para concentrarmos no estudo da linguagem e do processo de criação.

Definimos também que os programas seriam desenvolvidos utilizando técnicas de controle de **sistema de partículas** e de **agentes autônomos**. Essas duas técnicas foram escolhidas devido ao nosso interesse em trabalhar com a manipulação de grandes quantidades de objetos, operação que dificilmente é tratada fora do enfoque do design paramétrico.

Durante o desenvolvimento dos experimentos, não nos prendemos fielmente a utilização dessas técnicas, visto que elas foram apenas o ponto de partida para a exploração a ser realizada e não tiveram o papel de frear as ideias que não estavam diretamente relacionadas com essas técnicas. Essa conduta de não limitar outras ideias segue o raciocínio da conversa reflexiva na qual o designer parte de uma ideia inicial para desenvolver um produto. Mas tanto essa ideia quanto o produto e o designer se modificam e evoluem ao longo do tempo, não precisando necessariamente a proposta inicialmente configurada.

Nos experimentos apresentados, muitas vezes as características dessas duas entidades se mesclam, não sendo possível identificá-los apenas como um ou outro. Assim, para fins práticos, os termos ‘partícula’ e ‘agente autônomo’ serão referidos genericamente como ‘objeto’, conceito comum em programação orientada a objetos que se refere a uma instância de uma classe, que por sua vez é um conjunto de objetos de características semelhantes.

Por último, definimos que os programas tivessem a presença de um elemento gráfico que interagisse graficamente com os objetos. Esse elemento gráfico foi definido como sendo uma letra, ou número, ou qualquer outro **elemento tipográfico**, com a finalidade de criar uma composição menos abstrata e de criar uma relação semântica e sintática entre os experimentos, além de ser um etapa inicial daquilo que se transformou os experimentos 3.0 do capítulo 6.

Resumindo, os experimentos são dinâmicos, baseados nas técnicas de sistemas de partículas e de agentes autônomos que interagem graficamente com um elemento tipográfico. A seguir, explicamos de maneira sucinta o que são sistema de partículas e agente autônomo e depois apresentamos individualmente cada um dos experimentos.

#### 4.1.1 Sistema de partículas

No contexto da computação gráfica, um sistema de partículas pode ser definido como “uma coleção de inúmeras partículas minúsculas que juntas representam um objeto indistinto. Ao longo de um período, partículas são geradas dentro do sistema, movem e mudam dentro do sistema e morrem do sistema”.<sup>1</sup> (REEVES,WILLIAM apud SHIFFMAN 2012, tradução nossa)

<sup>1</sup> “A particle system is a collection of many many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system”

Nesse contexto, o sistema de partículas é tipicamente utilizado para simular efeitos especiais de dois tipos, animados e estáticos<sup>2</sup>. Explicando de maneira breve, nos sistemas animados cada frame da animação contém uma partícula que ocupa um único ponto no espaço, podendo simular explosões, fumaça, neblina, poeira, neve, faíscas, fluídios, etc. Já nos sistemas estáticos, a trajetória das partículas são renderizadas simultaneamente, servindo para simular pêlos, fios de cabelo, grama e materiais similares. A comparação entre as duas simulações podem ser observadas na figura 38. Ambos os tipos de simulação estão presentes no experimentos apresentados adiante.

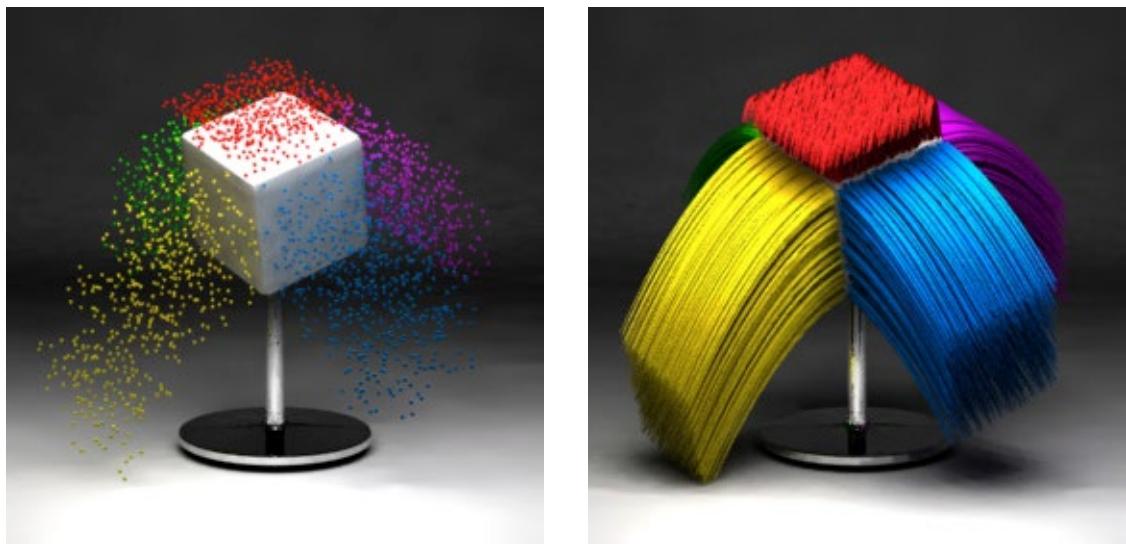


Figura 38: Sistema de partículas animado e sistema de partículas estático. Fonte: [http://en.wikipedia.org/wiki/Particle\\_system](http://en.wikipedia.org/wiki/Particle_system)

#### 4.1.2 Agentes autônomos

Também no contexto da computação gráfica, de acordo com Daniel Shiffman, “o termo agentes autônomos geralmente se refere a uma entidade que faz suas próprias escolhas sobre como agir em seu ambiente sem a influência de um líder ou de um plano. Para nós, ‘agir’ significa se mover”.<sup>3</sup> (SHIFFMAN 2012, tradução nossa)

Esses agentes possuem uma habilidade limitada de perceber o ambiente, controlado pelo código, e processa a informação do ambiente para calcular uma ação, como girar, acelerar, mudar de cor, aumentar de tamanho, etc.

Em simulações realistas, é possível estudar diversos comportamentos como: trânsito de carros, movimento de grupos de animais e de insetos, fluxo de pessoas, etc. Como estamos lidando com estudos visuais, os sistemas utilizados são mais simplificados já que não precisam ser criteriosos em relação às leis físicas e as regras de comportamento coletivo, por exemplo.

2 Fonte: [http://en.wikipedia.org/wiki/Particle\\_system](http://en.wikipedia.org/wiki/Particle_system)

3 “The term autonomous agent generally refers to an entity that makes its own choices about how to act in its environment without any influence from a leader or global plan. For us, “acting” will mean moving”

## **4.2 \_DLA**

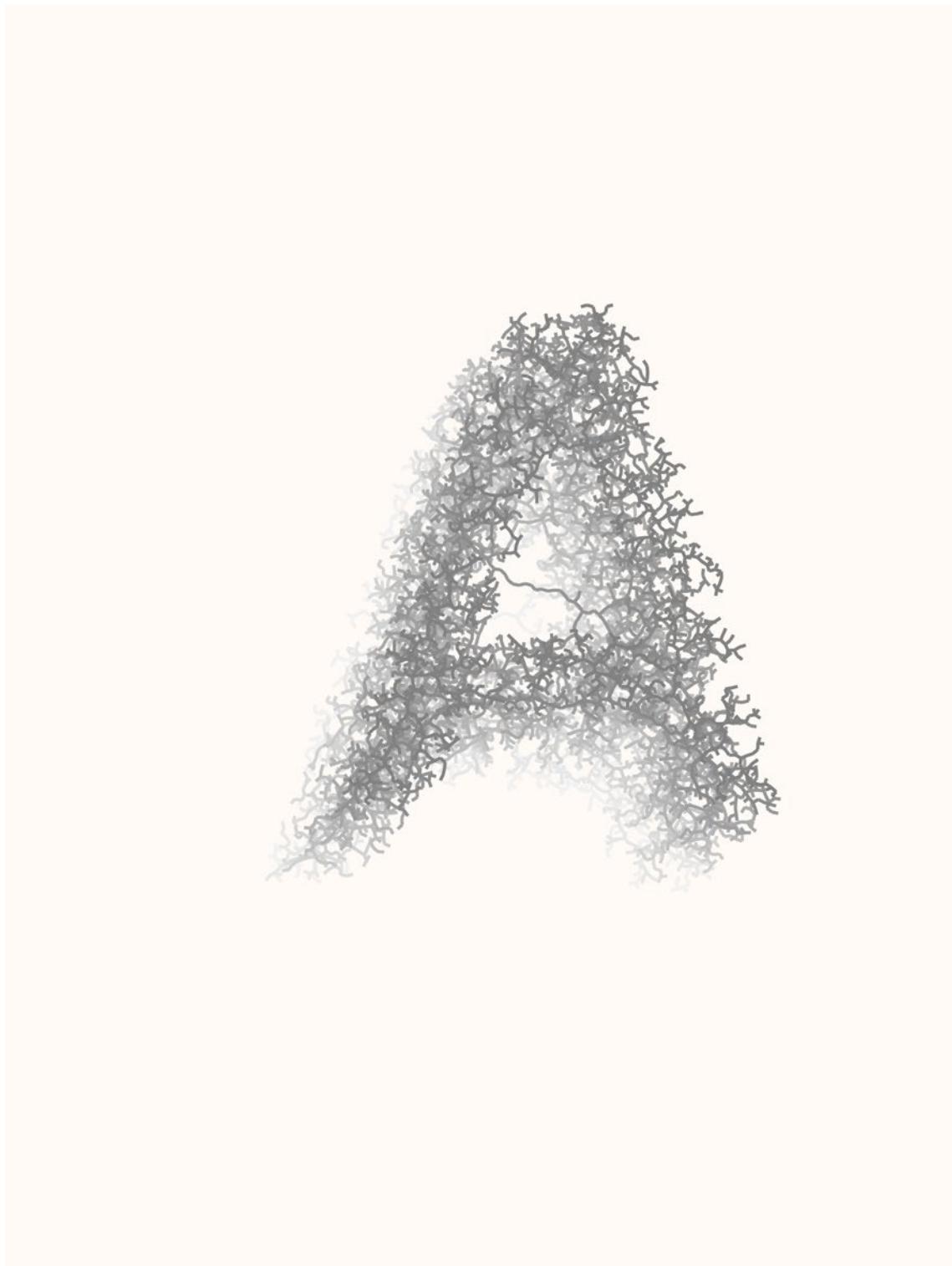
O crescimento do objeto inicia em três pontos localizados randomicamente. A medida que partículas se agregam a esses pontos, uma estrutura ramificada de 12.000 segmentos é gerada. Quando todas as partículas são agregadas, uma nova estrutura começa a emergir com segmentos mais escuros e com um certo deslocamento da primeira estrutura para criar um efeito de profundidade. Uma restrição é criada para que o crescimento fique limitado ao elemento tipográfico, criando assim a letra 'A'.

O resultado do experimento é uma estrutura de aparência orgânica que se assemelha a um fungo ou uma alga. A sobreposição de camadas foi implementada com a finalidade de criar um efeito de profundidade e também de gerar uma textura intricada, como se o grupo de todas as camadas fossem um único organismo.

O experimento \_dla foi o primeiro experimento feito durante o TCC2. Nele foi utilizado como base o código de um dos experimentos realizados no TCC1 que simula o princípio do DLA<sup>4</sup>. Como esse foi o primeiro experimento nessa segunda fase, o maior desafio foi conseguir fazer com que as interações ficassem limitadas apenas ao elemento gráfico. Feito isso, o código do experimento do TCC1 foi reciclado para atender aos novos parâmetros.



4 Conceito apresentado no apêndice, capítulo 9.2.4 Simulação



### 4.3 \_PONG

Um conjunto de objetos circulares de tamanhos variados é gerado e cada um deles se locomove em uma direção randômica. Quando a posição deles coincide com as paredes internas do elemento gráfico, aparentemente invisível, eles seguem para direções contrárias como se tivessem sido rebatidos por essas paredes. Esses objetos são exibidos de tal forma que somente a silhueta do conjunto de objetos sobrepostos fique aparente, criando bolhas protuberantes e aglomeradas.

O programa `_pong` foi o primeiro a ter que lidar a interação entre objetos e as paredes do elemento gráfico. Para isso foi preciso fazer um teste para averiguar se cada objeto estava dentro ou fora do elemento, caso estiver na iminência de sair do elemento, o objeto muda de direção, caso contrário se movimenta em linha reta.

Esse teste, introduzido no `_dla` e expandido para todos os outros programas, consiste em desenhar o elemento tipográfico e guardar a cor e a posição de cada pixel da tela em uma variável. Toda vez que o programa faz o teste (figura 39), a posição de um objeto é comparada com a posição da variável, se a cor do pixel na posição desse objeto for preta, quer dizer que ele está dentro do elemento tipográfico, se for branca ele está do lado de fora.

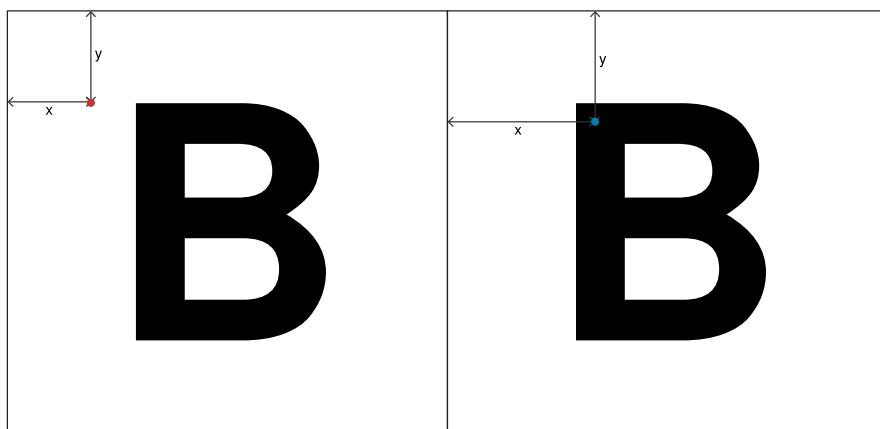
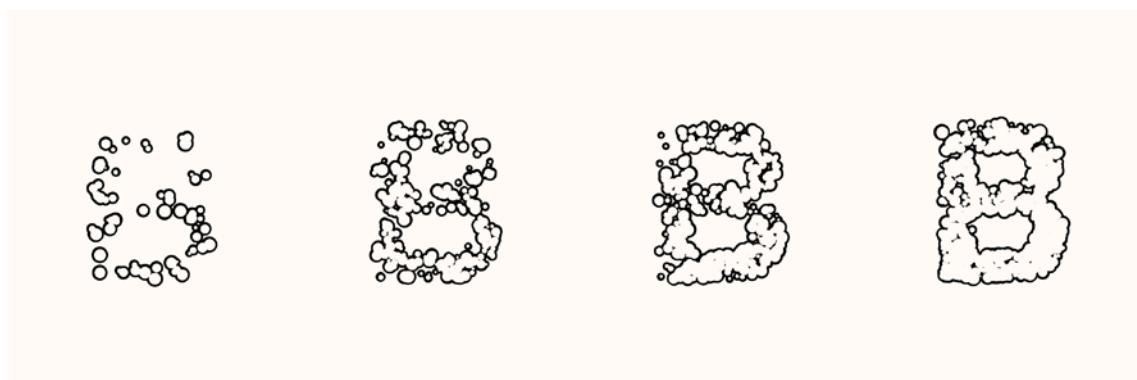
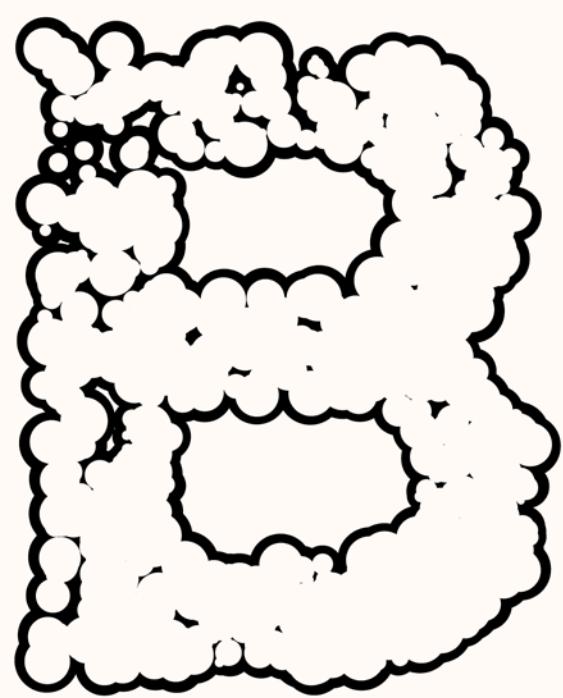


Figura 39: Teste de localização. Se o objeto estiver em um fundo branco (esquerda), quer dizer que eles estão fora do elemento. Se ele estiver em um fundo preto (direita), quer dizer que ele está dentro.





#### **4.4 \_RETRO**

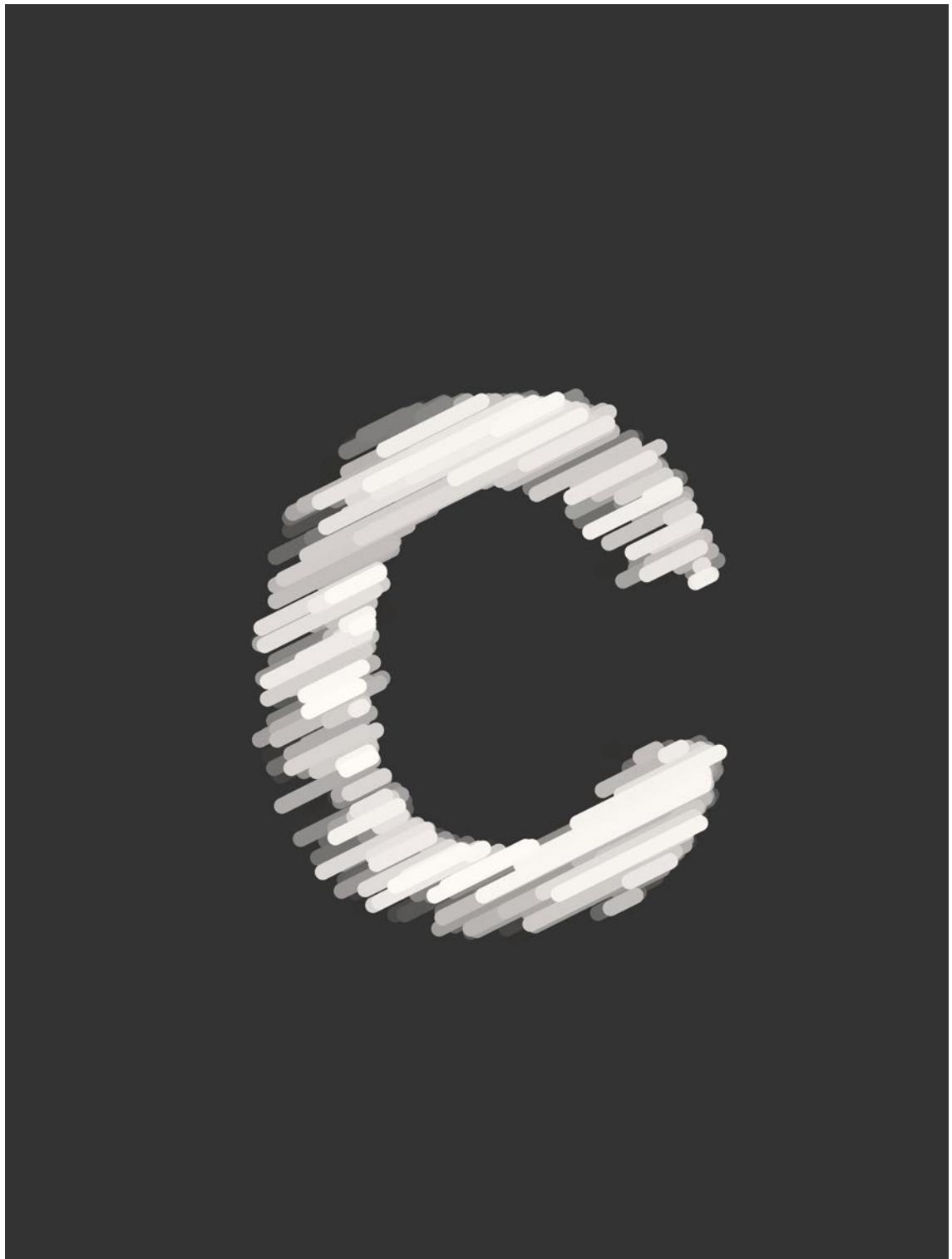
Objetos nascem em posições randômicas dentro do elemento gráfico e crescem em uma determinada direção até chegar em alguma borda do elemento e lá permanecem. Novos objetos vão sendo gerados por trás dos primeiros, dando destaque a eles. O resultado é um conjunto de linhas inclinadas sobrepostas umas as outras que causam um efeito de profundidade. Esse experimento foi inspirado nas esculturas tubulares do coreano Kang Duck-Bong, que constrói formas diversas a partir da junção de tubos plásticos.

\_retro é um exemplo de experimento que não trata exatamente de partículas nem de agente autônomos. O objeto criado possui um comportamento simples e agrupa características do \_pong e do \_dla, como o de detectar as bordas do elemento gráfico e o de gerar posições randômicas dentro dele para iniciar o movimentos dos objetos.



Figura 40: Escultura de Kang Duck-Bong. Fonte: <http://www.juxtapoz.com/current/pipe-sculptures-by-kang-duck-bong>



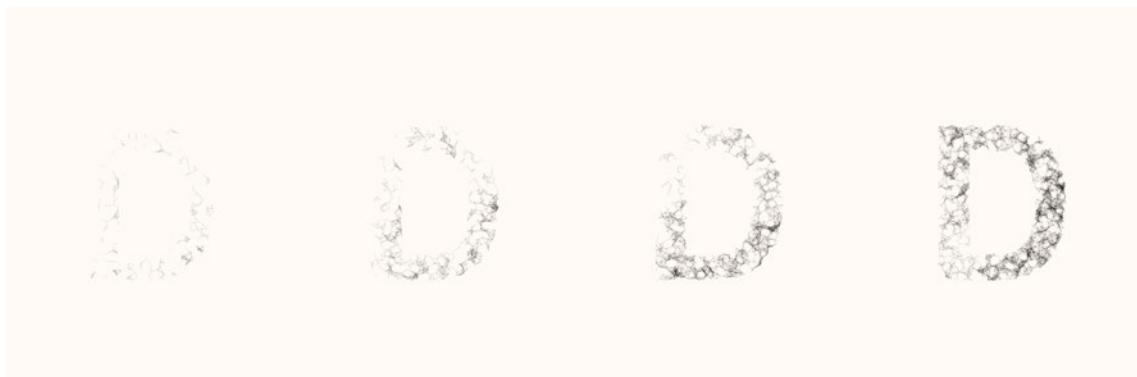


## **4.5 \_ FRAME**

Assim como no programa \_pong, objetos rebatem nas paredes internas do elemento gráfico, mas agora eles são representados por pequenos pontos que, ao se aproximarem de outros objetos, uma linha temporária entre eles é criada. Caso eles se distanciem, essa linha desaparece. O resultado é uma letra formada por uma estrutura efêmera de linhas que se constroem e se desconstroem indeterminadamente.

Essa aparência estruturada por pontos e linhas não é novidade, mas esse experimento teve sua importância ao ser um exercício de engenharia reversa no qual um linguagem ou aparência é recriada através de sua análise visual. Além disso foi um exercício de abstração de uma ideia para a criação do código.

Para criar as linha de conexão, é preciso verificar a posição de cada um dos 1300 objetos e compará-la com a de todos os outros objetos restante. Como esse teste precisa ser feito a cada frame, isso exige um grande número de cálculo até mesmo para o computador, por isso o número máximo de partículas se torna bastante limitado.



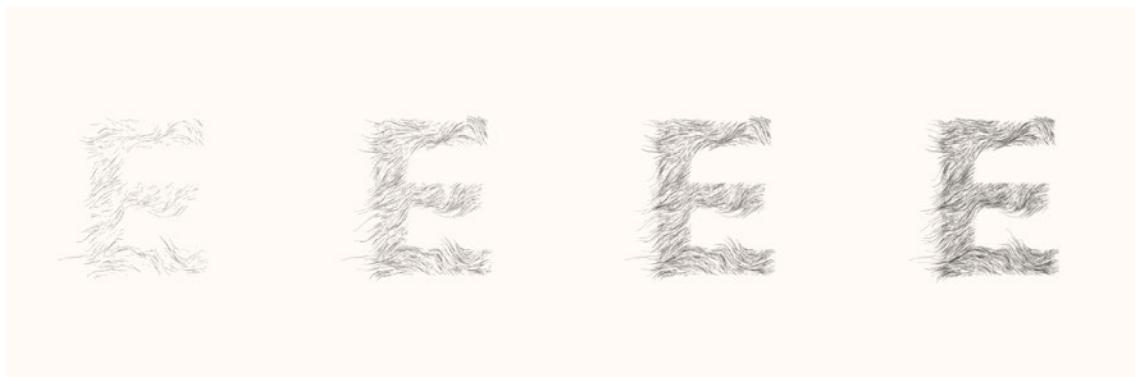


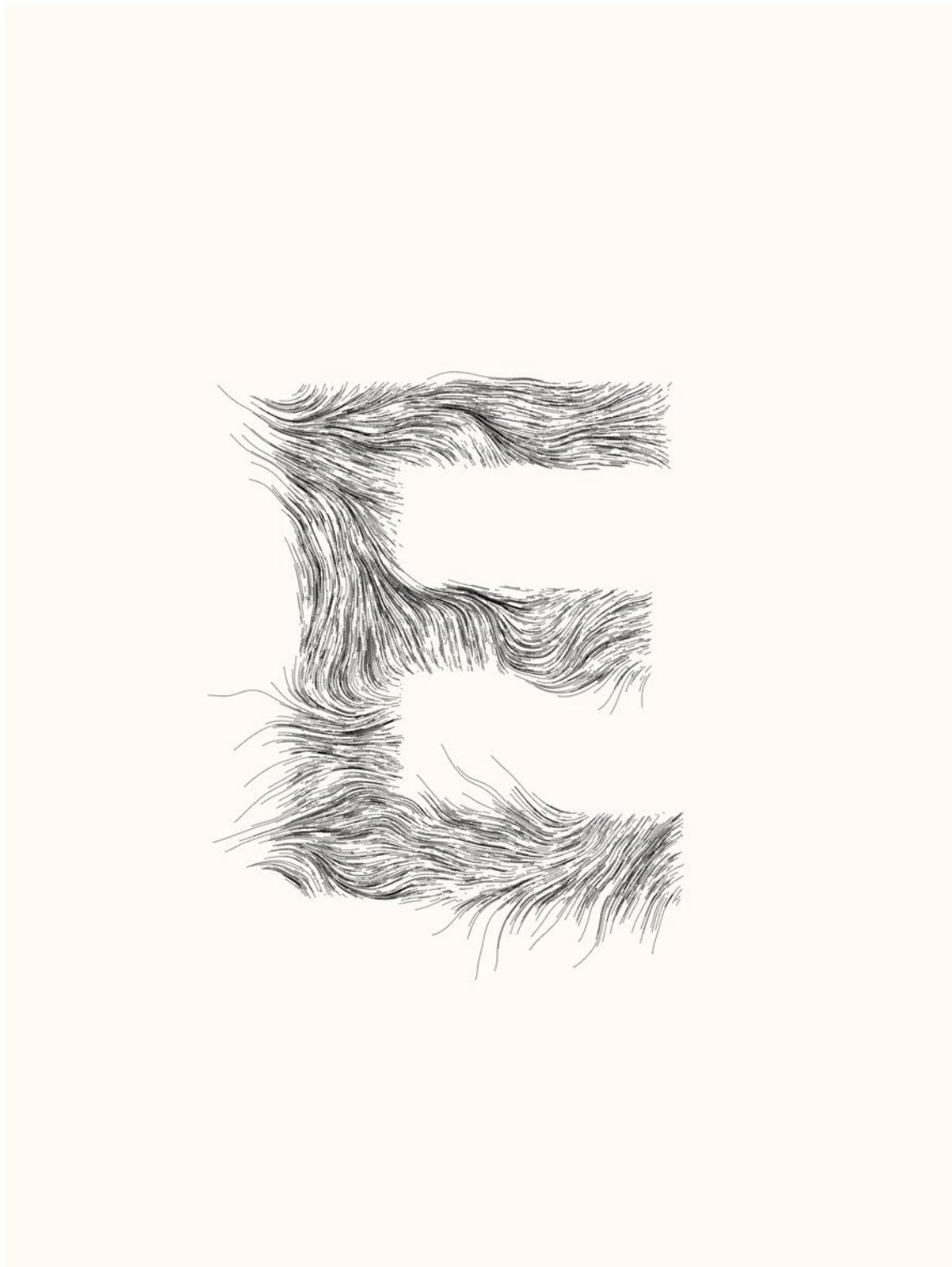
#### **4.6 \_ PERLIN**

Centenas de objetos nascem dentro do elemento gráfico e se locomovem sem rumo por uma trajetória sinuosa e morrem depois de um período curto de tempo. O resultado é um elemento tipográfico com uma textura composta por um conjunto de filamentos que se assemelham a pêlos.

A trajetória criada pelos objetos segue uma função pseudo-randômica conhecida como Perlin Noise que cria variações suaves e serve tipicamente para simular efeitos orgânicos. Como as trajetórias de todos os objetos estão baseadas nessa função, os movimentos desses objetos estão associados, criando caminhos semelhantes.

Cada objeto tem um tempo de vida curto e param de se mover em até poucos segundos, restando apenas sua trajetória. O elemento gráfico só foi utilizado para limitar o local onde os objetos são gerados, sendo que eles podem ultrapassar livremente os limites do elemento.



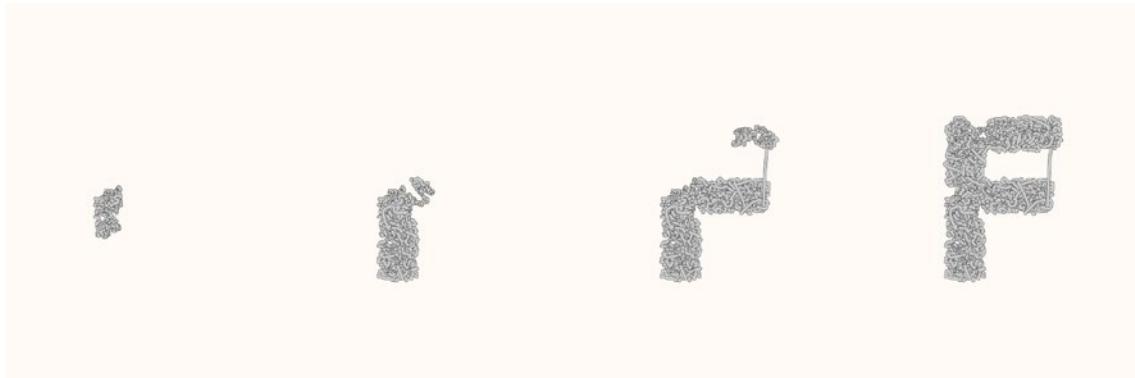


## 4.7 \_ SPLINE

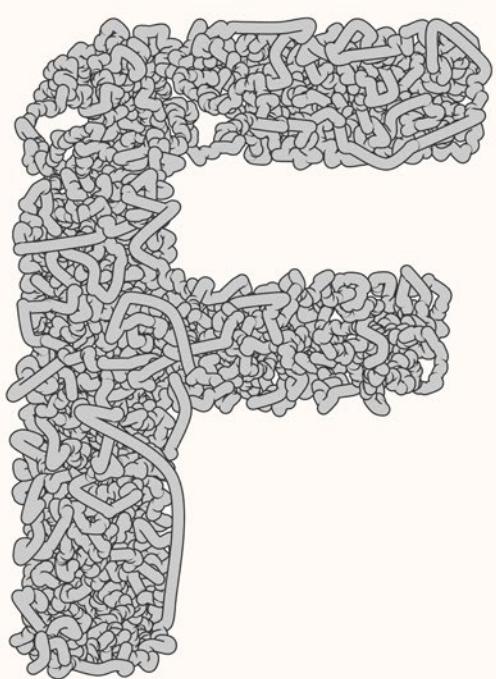
Milhares de pontos invisíveis são criados dentro do elemento gráfico. Um objeto representado por uma linha persegue cada um desses pontos e sempre procura o mais próximo. Ao alcançar esses pontos, o objeto cresce de tamanho e vai em busca de outros ponto até consumir todos os pontos criados. O produto final é um órgão esguio emaranhado em si mesmo, orgânico e irregular.

O programa tem um funcionamento simples, no qual inúmeros pontos são gerados e a partir da escolha randômica de um deles, uma linha curva do tipo spline<sup>1</sup> começa a ser traçada. Esses pontos são gerados da mesma forma que os programas \_dla e \_retro. A diferença é que esses pontos servem como âncoras para ligar os pontos da spline, como no passatempo “ligue os pontos”.

O desafio desse programa foi criar o efeito de sobreposição da linha sobre si mesma para criar a impressão de algo emaranhado. Como é complicado fazer com que uma mesma linha perceba que está sobrepondo a si mesma, foi mais fácil dividí-la em pequenos segmentos de spline. Assim, os segmentos são desenhados uns por cima dos outros a cada iteração do programa, criando o efeito de sobreposição.



<sup>1</sup> Spline é uma linha definida por pontos de controle, sendo que esse pontos criam relações de tangenciamento para criar uma linha curvilínea.



#### 4.8 \_ METABALL

Similar ao \_pong, um conjunto de objetos rebatem nas paredes internas no elemento gráfico, mas com um diferencial no tipo de tratamento gráfico dado a composição. Esse programa reproduz o efeito da modelagem denominada ‘metaballs’ que criam formas orgânicas ameboides. O resultado é um elemento fluido que aparenta ter certa viscosidade.

Para criar esse efeito, objetos representados por esferas em degrade foram gerados um filtro visual conhecido como “*threshold*” foi aplicado para “achatar” o degradê e fundir esses objetos, um exemplo desse processo pode ser visto na figura 41.

O Processing já possui uma função de manipulação de imagem chamado “*threshold*”, mas ela utiliza um número excessivo de processamento do computador ocasionando um programa extremamente lento. Para contornar essa situação, tivemos que criar nosso próprio algoritmo para gerar esse filtro de maneira otimizada.

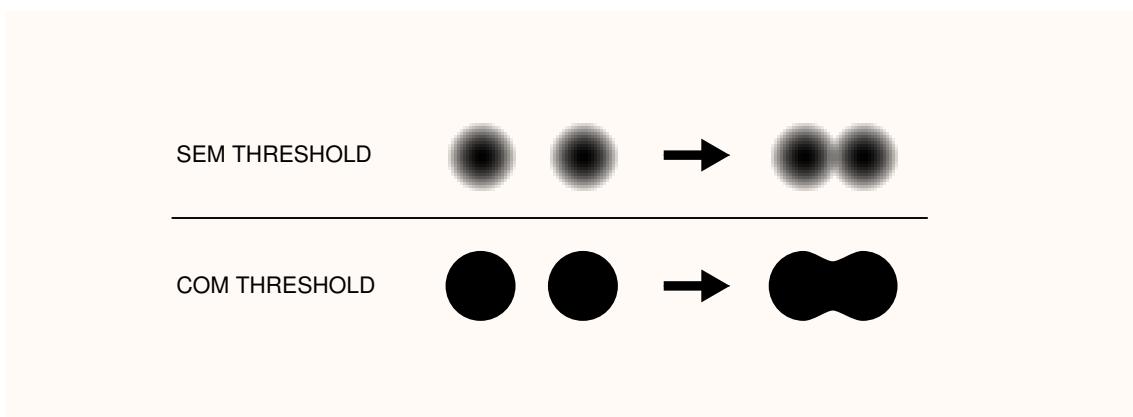


Figura 41: Criação do filtro “*threshold*”.



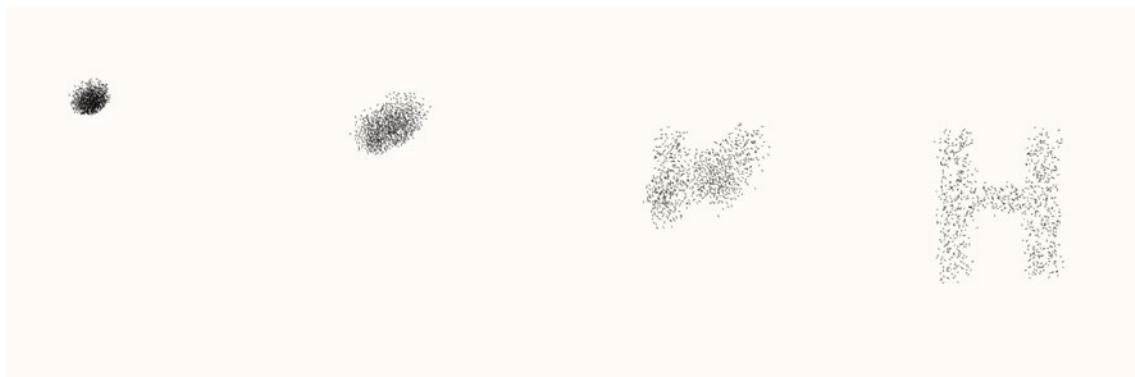


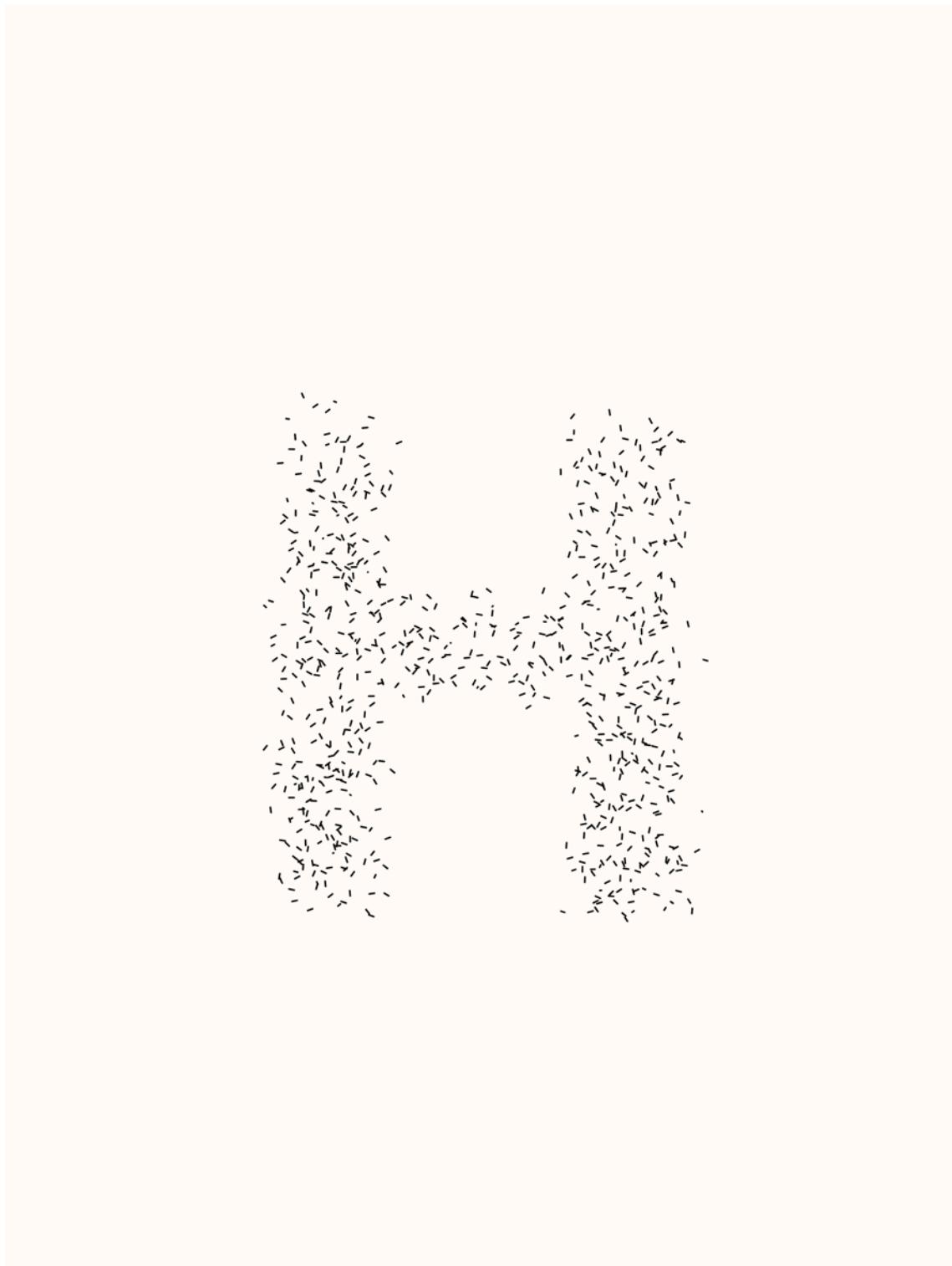
#### **4.9 \_ SEEK**

Centenas de agente são gerados com o objetivo de chegar em um local específico do elemento gráfico. Esses agentes não sabem controlar sua velocidade e sempre passam do ponto de destino, orbitando esse ponto. Esses agentes não gostam de estar próximos de seus semelhantes e tentam se afastar quando uma distância mínima é atingida. O resultado disso é um conjunto de diversos pontos que se aglomeraram no elemento tipográfico, como um enxame de abelha.

Esse programa introduz uma nova característica que são comportamentos de relações entre objetos. Agora eles interagem entre si, repelindo uns aos outros. Além disso, cada um deles tem um objetivo, um alvo para seguir. Devido a esses dois comportamentos dos agentes, o de se chegar a um ponto e o de ser repelido pelos outros agentes, um movimento irregular e desordenado é criado.

Os agentes utilizados foram baseados nos agentes criados por Daniel Shiffman, presentes em seu livro “The Nature of Code”. O código base dos agentes de Shiffman é versátil e permite que novos comportamentos sejam adicionados aos agentes de maneira fácil. Esse livro foi de grande importância no aprendizado de recursos de programação para a criação de efeitos visuais, principalmente na utilização de agentes autônomos.



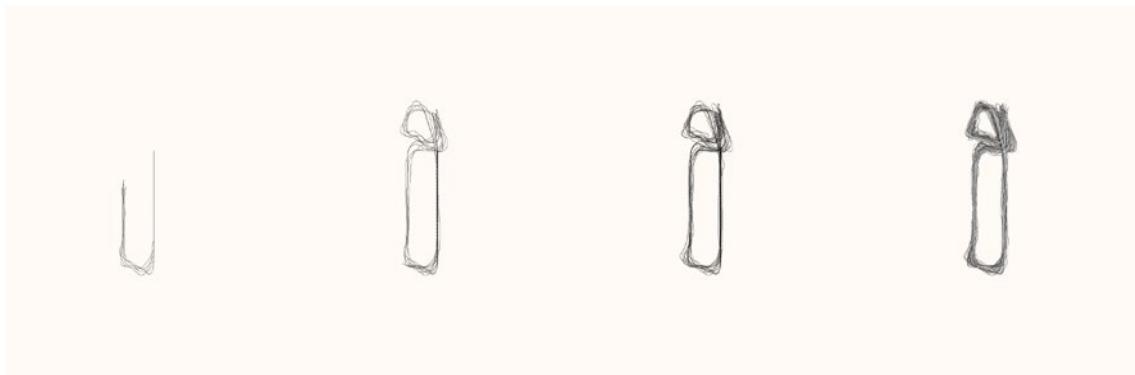


#### **4.10 \_OUTLINE**

Um conjunto de agentes persegue um alvo invisível que percorre a silhueta do elemento tipográfico. Essa perseguição é incessante já que os objetos nunca alcançam seus alvos. O que é possível de ver na tela são os rastros deixados por esse agentes que criam anéis irregulares em volta da letra “i”.

Esse programa se apropria do mesmo tipo de objeto do programa anterior \_seek, um agente cujo comportamento é se afastar de outros objetos e tem um único objetivo de alcançar seu alvo. Porém nesse caso, o alvo é móvel e fica sempre a frente do objeto.

Para conseguir gerar pontos ao longo do elemento tipográfico, uma biblioteca foi implementada para acrescentar novas funções ao Processing que potencializam o uso dessa ferramenta. A vantagem de se usar um programa open source é que terceiros podem criar suas próprias bibliotecas para implementar funções específicas ao programa, o que torna ele mais completo e versátil.



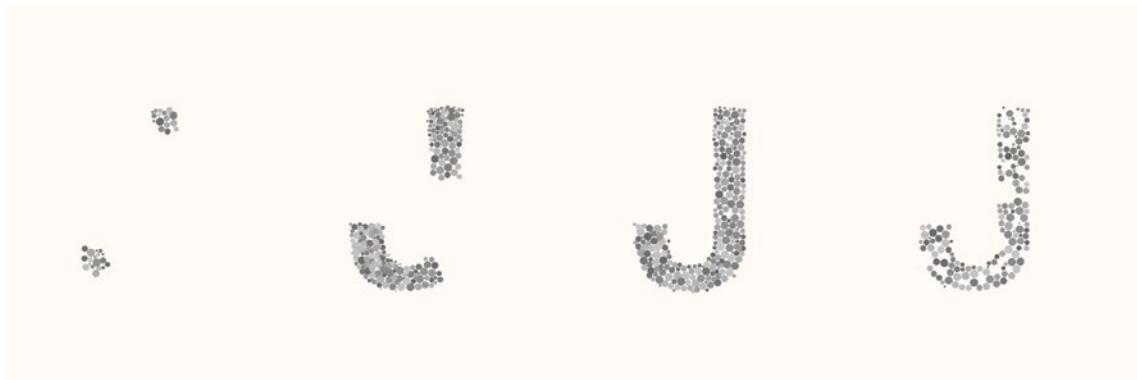


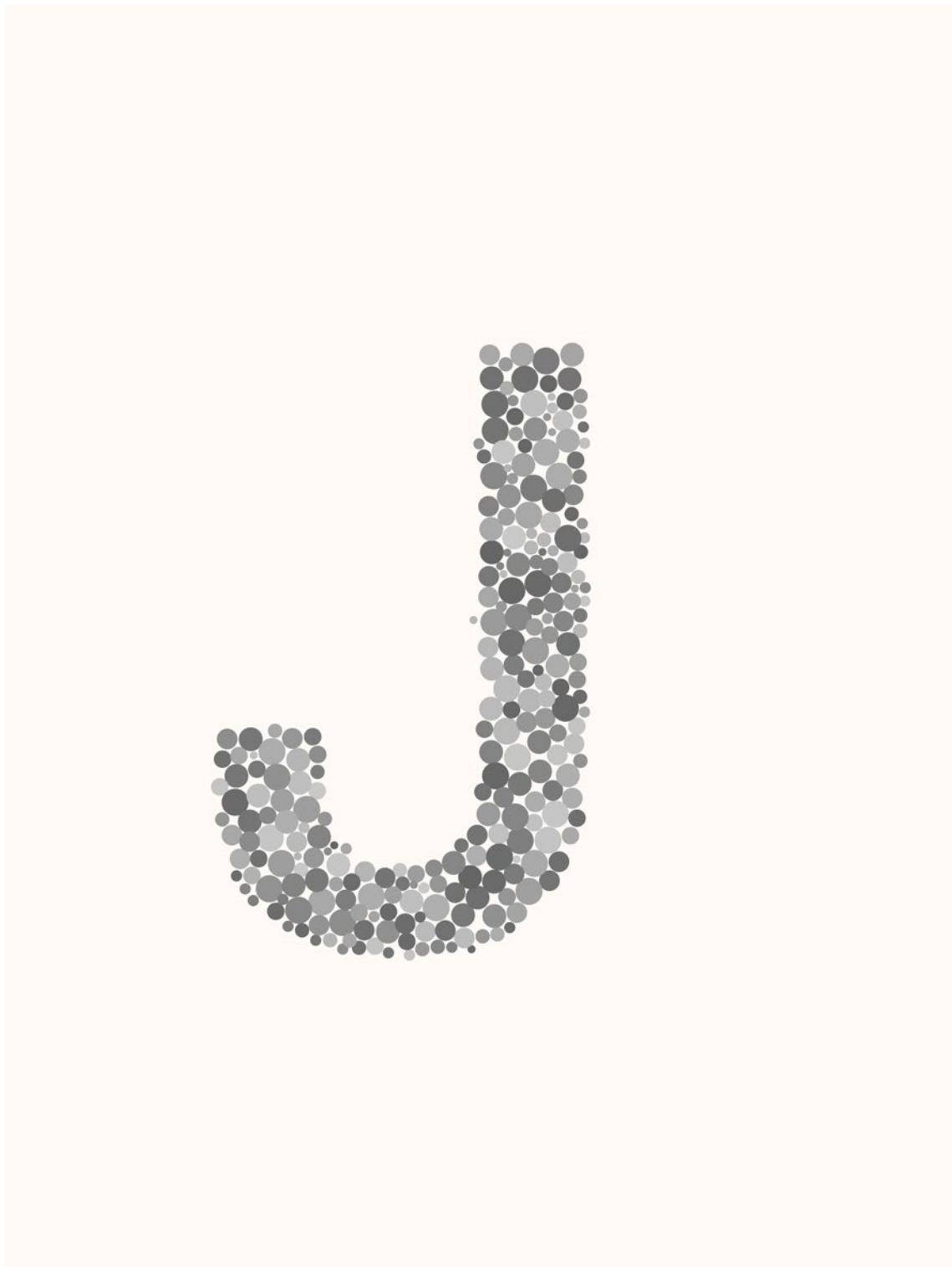
#### **4.11 \_STIPPLE**

Círculos de diversos tamanhos surgem dentro do elemento gráfico e são empurrados para os lugares vazios, sendo que um objeto pressiona os outros até todos caberem dentro do elemento. Quando um certo número de círculos são criados, objetos param de surgir e os que estão na tela começam a estourar, um a um, até não sobrar mais nenhum.

Esse programa agrupa características dos dois últimos programas. Os objetos tendem a se manter afastados de maneira similar ao `_seek` e a mesma biblioteca do `_outline` é utilizada para fazer como que os objetos fiquem contidos dentro do elemento tipográfico.

Apesar de ter sido encontrado um caminho para a resolução do problema logo no início de seu desenvolvimento, demoramos para resolver um bug que fazia com que os círculos vazassem para fora do elemento gráfico. Depois de muito tempo percebemos que uma variável estava com um valor errado, causando todo o problema.





#### **4.12 \_GRID**

Uma trama de linhas horizontais e verticais inicialmente estável começa a se movimentar como um tecido ao vento. Um movimento bidimensional que ocasiona um efeito visual tridimensional.

Um programa visto no site openprocessing.org foi utilizado como inspiração para a realização deste experimento (figura 42). O programa consiste de uma cortina interativa no qual é possível puxar, arrastar qualquer ponto do tecido, caso a cortina for puxada demais, ela se rasga, podendo ser inteiramente destruída.

No programa `_grid`, a trama de linhas é na verdade uma malha de pontos que se conectam por pequenos segmentos de linhas. Como o movimento de todos esses pontos são regidos pela função randômica matricial Perlin Noise, esse movimento é contínuo e fluido. O programa está longe de ser um simulador de tecidos já que para isso um objeto bem mais complexo deve ser utilizado e diversos parâmetros e forças precisariam ser aplicados e ele, como: gravidade, massa, maleabilidade, coeficiente de atrito, etc. O que o programa faz é apenas disfarçar essa aparência para criar um efeito gráfico interessante.

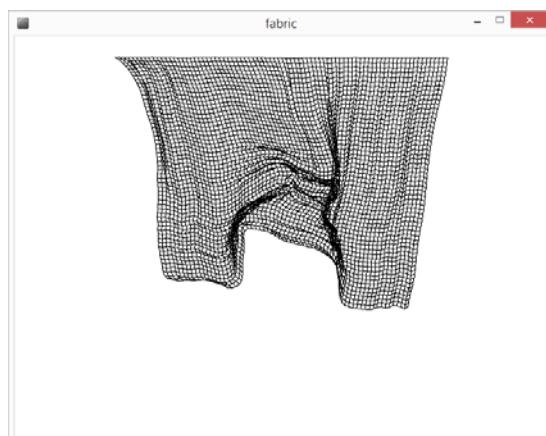
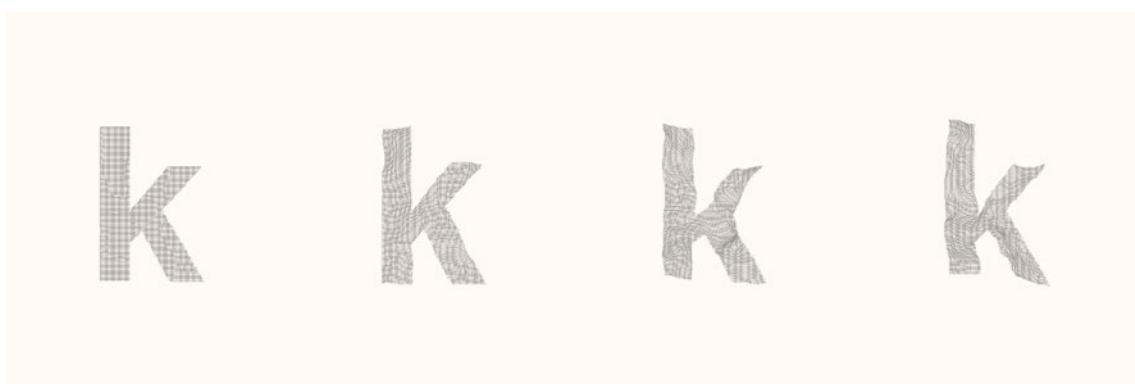
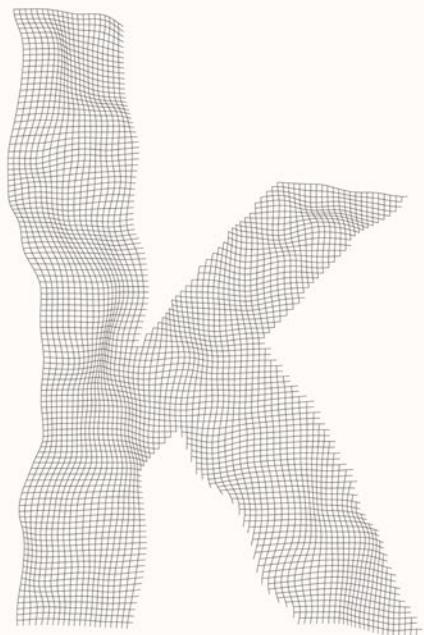


Figura 42: Programa que serviu de referência para a criação do `_grid`. O programa pode ser acessado em: <http://openprocessing.org/sketch/20140>



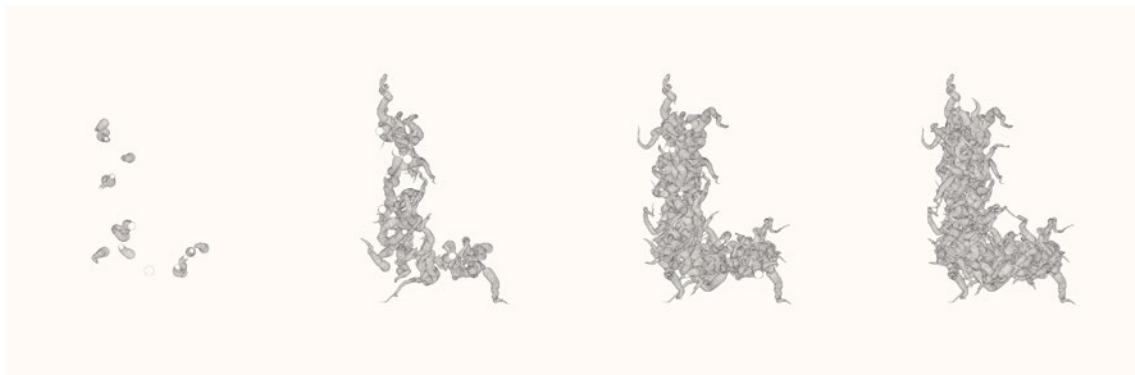


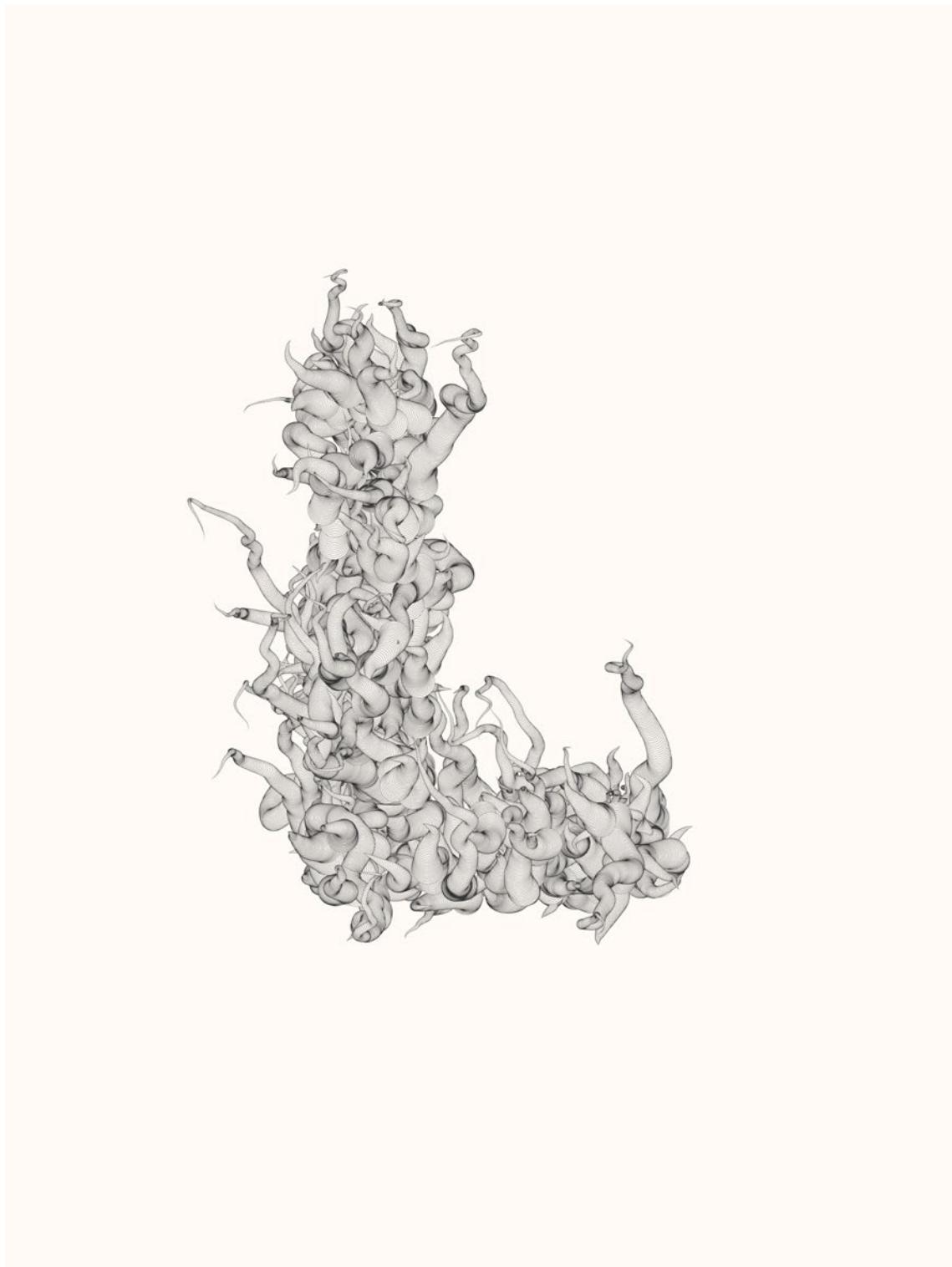
#### **4.13 \_TENTACLE**

Esse experimento apresenta um aglomerado de galhos que crescem em tempo real e seguem rumos não definidos, o que cria uma aparência orgânica, caótica e adquire textura com o passar do tempo para enfim formar o elemento gráfico.

O que o programa \_tentacles faz é criar objetos que são representados por suscetivos círculos criados de maneira sobreposta. Esses círculos seguem um caminho randômico baseado na função Perlin Noise e diminuem de tamanho ao longo do tempo, morrendo ao ficar imperceptível. Como o programa segue a linha de sistemas estáticos, a trajetória dos objetos fica aparente, criando assim uma forma que se assemelha a um galho, raiz ou tentáculo.

Como os objetos seguem rumos randômicos, é necessário limitar a quantidade de objetos criados para que eles mantenham um nível mínimo de legibilidade do elemento tipográfico e não se torne uma forma abstrata.



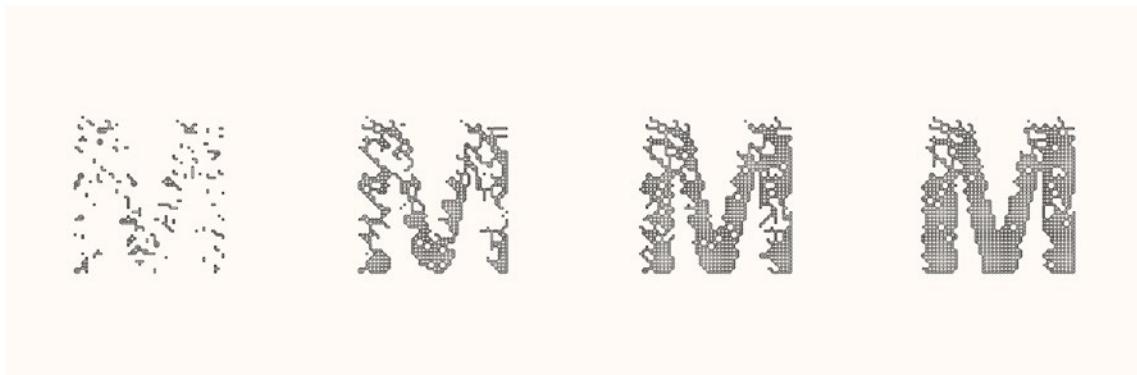


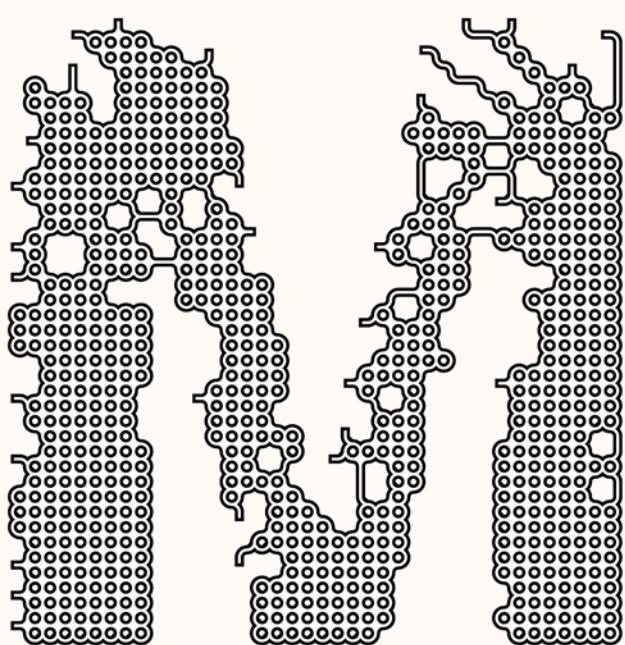
#### **4.14 \_PATTERN**

Formas geométricas mutantes são criadas para formar o elemento gráfico. Elas são mutantes porque se atualizam de acordo com os espaços vizinhos. No final, um padrão estático de formas geométricas é gerado a partir do movimento de objetos não aparentes.

O que o programa `_pattern` faz é gerar objetos com um comportamento igual ao programa `_pong`. A tela é dividida em um grid quadrado e quando um objeto se encontra em uma das células desse grid, uma forma geométrica é criada. As formas geométricas se adaptam para se ligarem a outras células. Essas formas são arquivos vetoriais que são utilizados como input no programa. Esse programa é interessante por misturar e integrar tanto a ferramenta da programação quanto uma ferramenta de manipulação de imagens vetoriais.

Esse programa usou como referência um código presente no livro “Generative Design” do H. Bohnacker. Nele é ensinado como utilizar os arquivos vetoriais para se criar a malha de células que se adaptam de acordo com os espaços vizinhos.



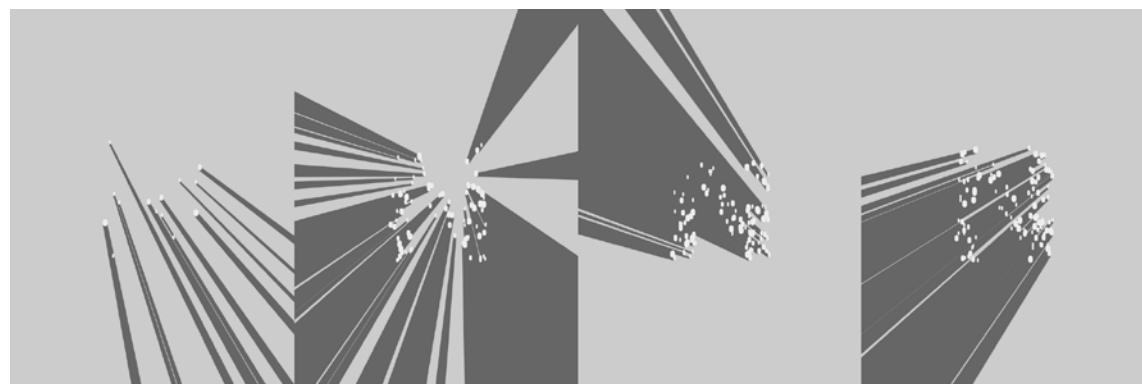


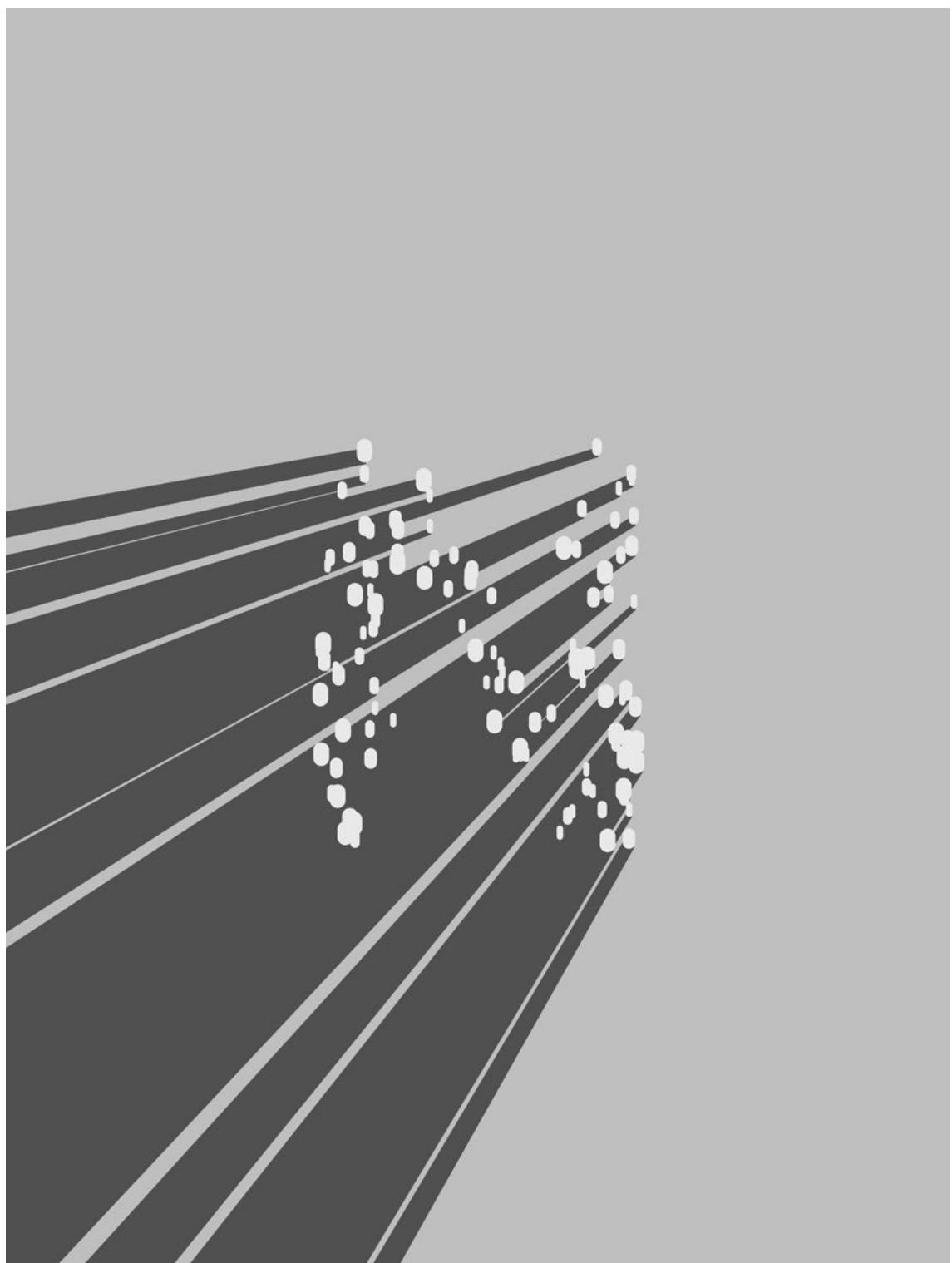
#### **4.15 \_SHADOW**

Corpos se movem pelo elemento tipográfico e suas sombras são projetadas na tela por uma fonte de luz oculta que os rodeiam. Diferente dos outros experimentos, há uma interação dentro e fora do elemento gráfico. Isso cria uma relação visual diferente no qual toda a tela é aproveitada.

Novamente objetos seguem o comportamento do \_pong. Agora através de um cálculo vetorial, uma sombra é gerada de acordo com a posição do objeto em relação ao foco de luz invisível. Esses cálculos exigem muito processamento do computador por isso a quantidade de objetos que pode ser criada é bastante limitada.

Apesar de serem claros, os vetores requerem certa abstração espacial para posicioná-los de maneira correta já que eles funcionam a partir de coordenadas cartesianas. Assim, é preciso prever qual a direção, sentido e magnitude que eles vão adquirir de acordo com o movimento dos objetos.





#### **4.16 \_ORBIT**

Um conjunto de corpos celestes são criados para formar o elemento tipográfico. Cada um deles orbita uma região diferente e é composto por uma calda que segue sua trajetória. O resultado é uma peça gráfica que explora a tridimensionalidade pelo movimento circular.

O movimento dos corpos são regidos por uma função trigonométrica simples que resulta em uma órbita elíptica. Cada objeto tem uma luminosidade diferente para distinguir e identificar melhor cada um eles.

As ilustrações de Brendan Monroe (figura 43) foram utilizadas como referência para realizar esse experimento. Elas são compostas por linhas coloridas paralelas ou concêntricas que causam a impressão de movimento.



Figura 43: Ilustração de Brendan Monroe. Fonte: <http://www.juxtapoz.com/current/brendan-monroe-melting-into-the-floor-richard-heller-santa-monica>



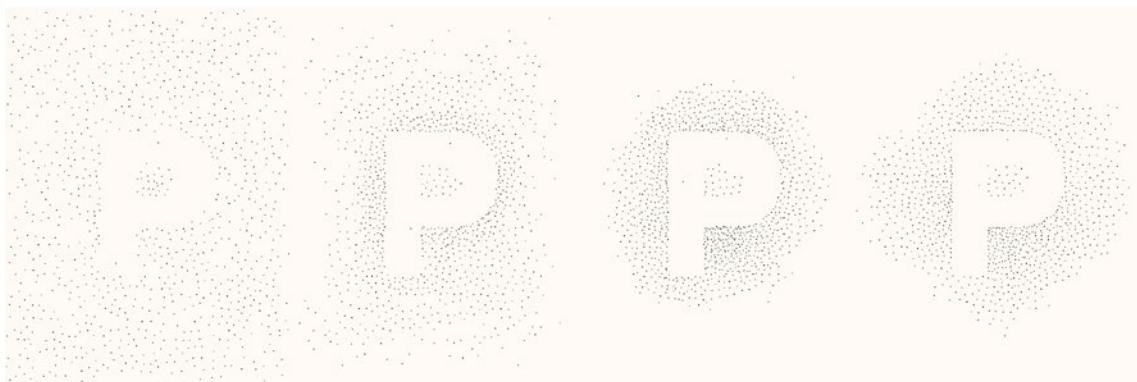


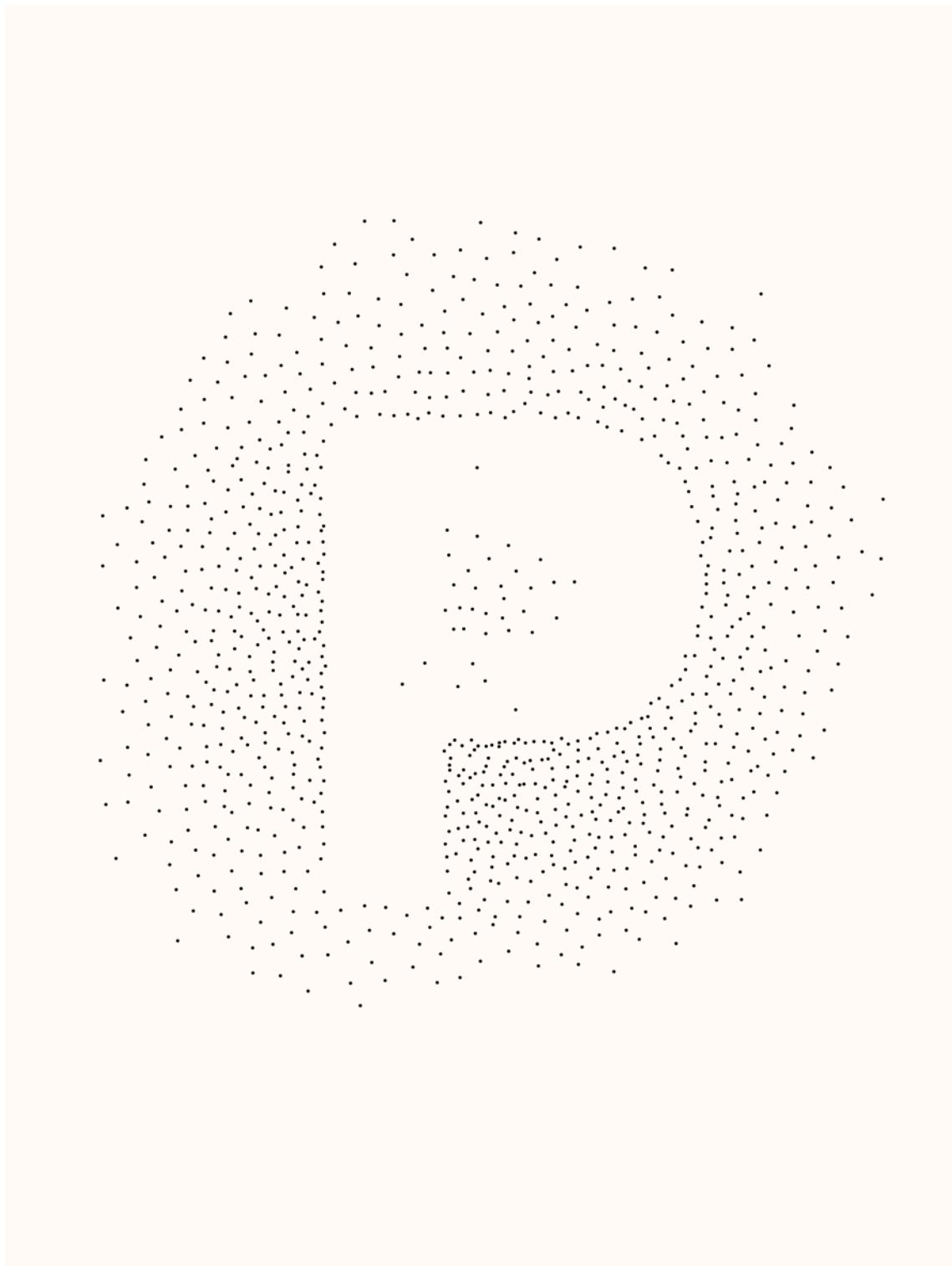
#### **4.17 \_SWARM**

Múltiplos agentes são gerados em locais aleatórios e vão em busca do centro da tela, mas em seu caminho há outros agente e também há o elemento tipográfico para atrapalhar. O conjunto de agentes se aglomeram em volta elemento, delimitando sua silhueta.

Esse programa é diferente dos outros devido a interação dos objetos com o elemento gráfico acontecer externamente ao elemento, deixando aparente somente sua silhueta.

Nesse programa novamente aparecem objetos com o comportamento de busca pelo alvo e de afastamento dos outros objetos. Agora eles interagem externamente ao elemento que por sua vez é uma barreira que está no caminho deles. A biblioteca é novamente utilizada para criar essa barreira que impede os agentes de avançarem. Por causa do conflito entre os dois comportamentos, ocorre um movimento pendular de aglomeração e dispersão dos agentes.

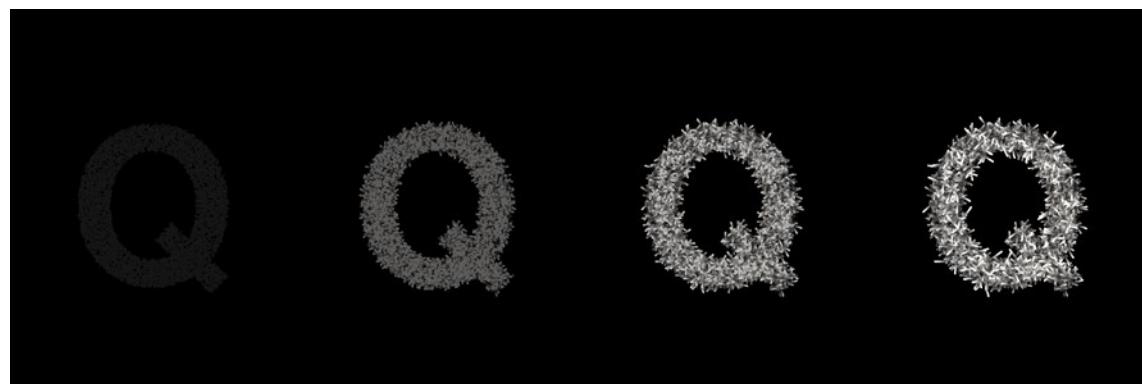


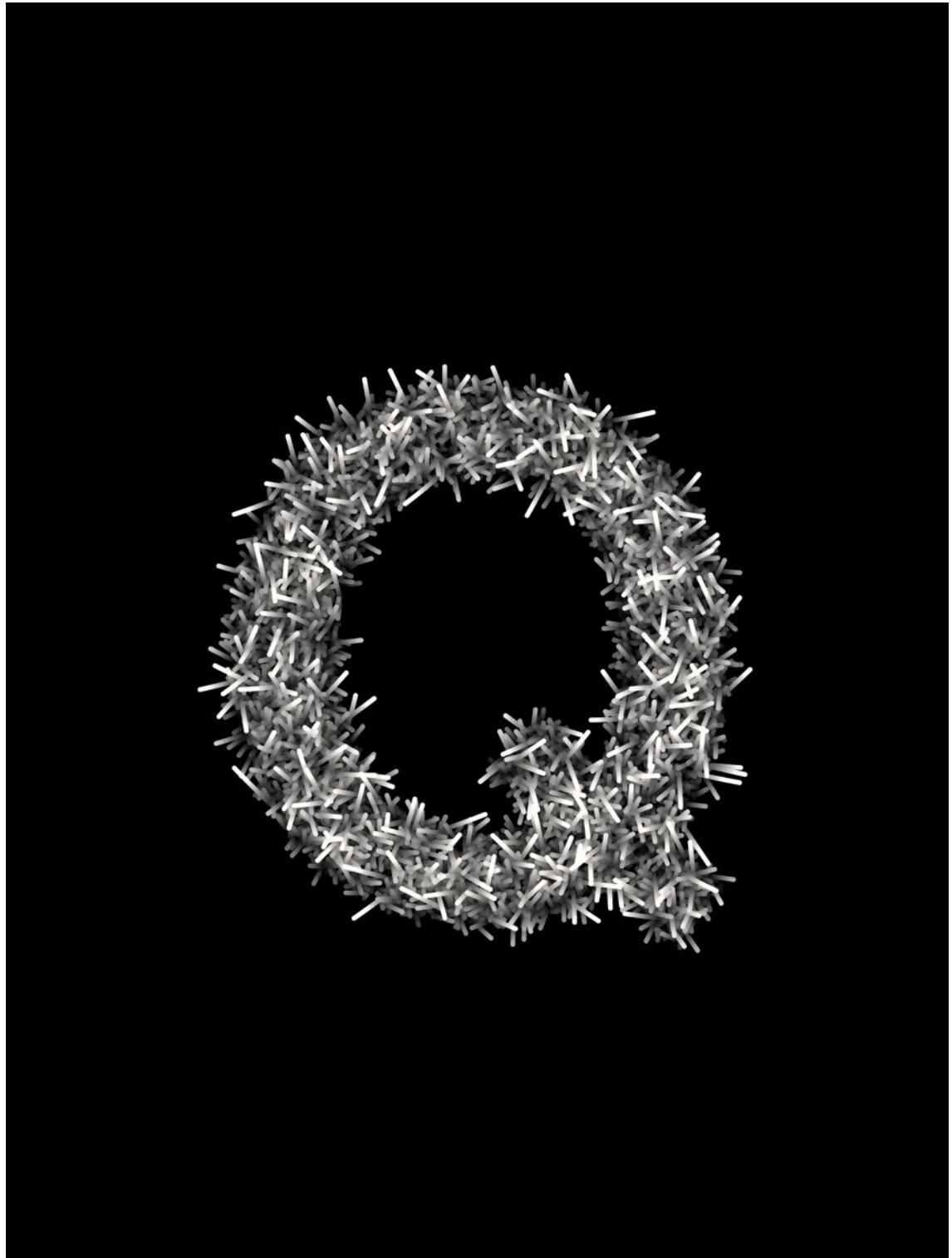


#### **4.18 \_EXPLODE**

Objetos emergem da tela, ficando cada vez mais claros e saturados. Eles seguem rotas retilíneas porém em direções randômicas. Cada objeto tem velocidades diferentes e tempos de vida diferentes. O efeito de transição de cor cria a impressão de profundidade fazendo com que o elemento se destaque do fundo por contraste.

Os objetos têm apenas um comportamento que faz com que eles se desloquem com uma velocidade constante em uma direção reta. A diferença de velocidades e de tempos de vida serve para que a composição fique mais heterogênea com diferenças de luminosidade. Esse experimento é baseado no programa \_tentacles, no qual objetos nascem dentro do elemento tipográfico, movimentam-se para caminho diversos e morrem depois de certo tempo.



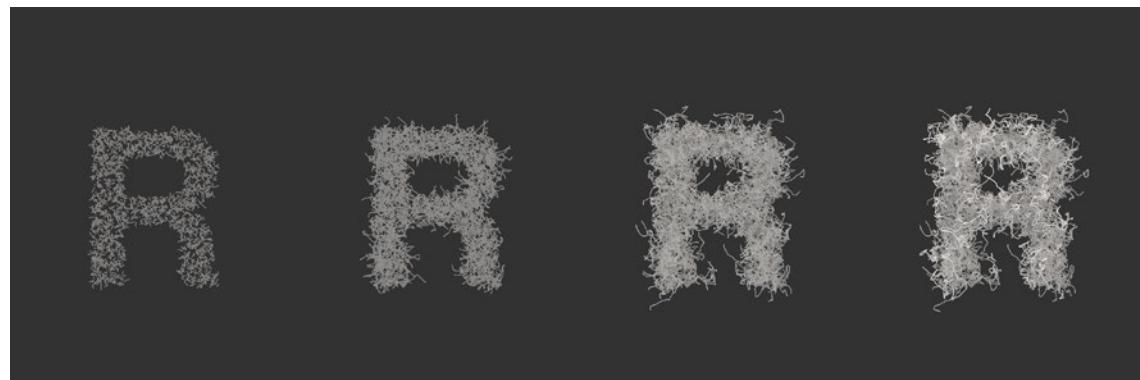


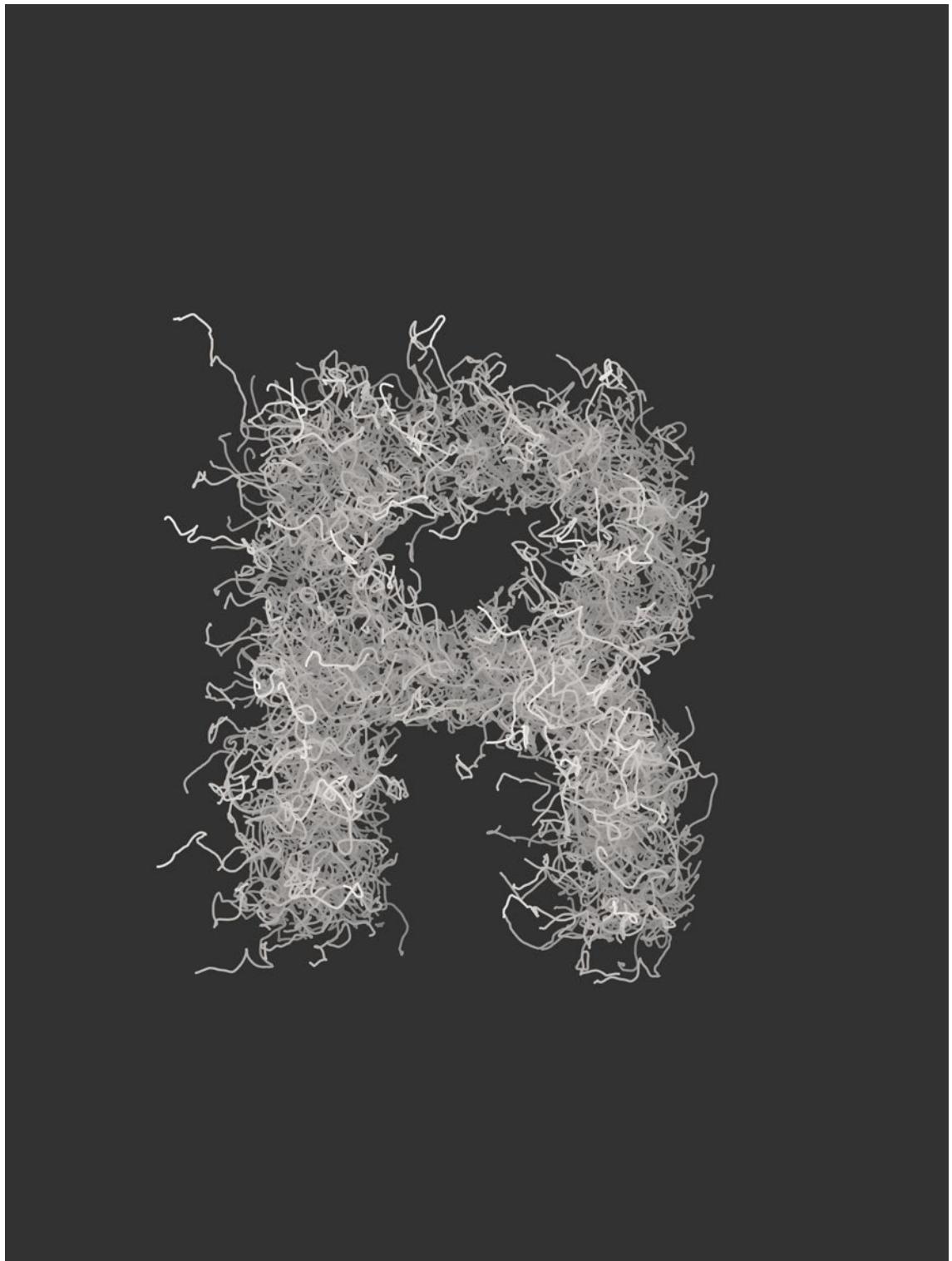
#### **4.19 \_THREAD**

Objetos nascem e se movimentam a deriva, sem rumo pela tela. A composição cria uma trama confusa de fios emaranhados que se dispersam ao longo do tempo.

Esse programa é similar ao anterior `_explode` e possui os mesmos tipos de objetos, com a diferença de ter objetos que se movem em trajetória randômica baseada na função Perlin Noise. Somente essa diferença é responsável por criar uma composição bastante distinta da composição criada pelo `_explode`.

Esse programa foi uma evolução do programa `_tentacles` que ocorreu de maneira não intencional quando se tentava aprimorar o programa. O erro fez com que o tamanho do círculo ficasse constante, fazendo com que o movimento do objeto nunca tivesse um fim. Como o efeito pareceu interessante, resolvemos criar este experimento.

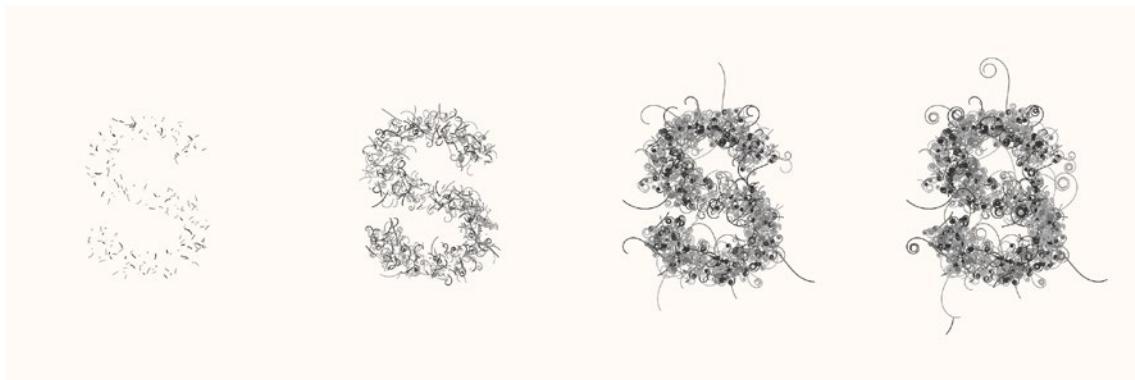


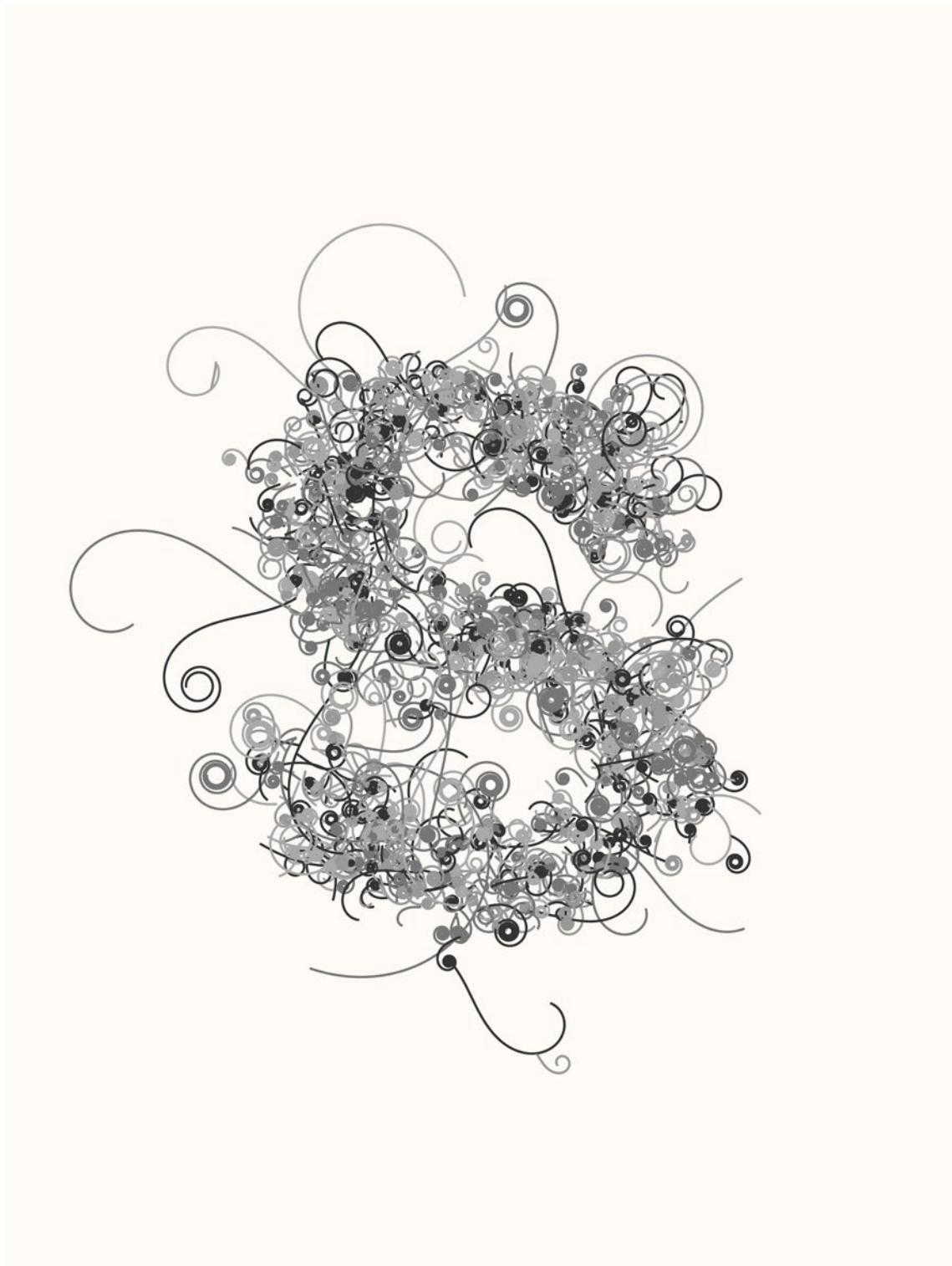


#### **4.20 \_SWIRL**

Trajetórias espirais são formadas pelo movimento dos objetos gerados na tela. As espirais criam um resultado gráfico floral e orgânico que remete ao estilo Art Nouveau.

Os programas \_tentacles, \_explode, \_thread e \_swirl seguem a mesma linha de raciocínio. Este último traz um novo tipo de movimento que criam trajetórias espirais de vários tamanhos que podem ir para várias direções diferentes. A trajetória espiral acontece ao incluir no objeto o comportamento de girar de acordo com um ângulo que se incrementa ao longo do tempo. Assim, a cada frame o objeto rotaciona com um ângulo maior, tendendo para um único ponto.



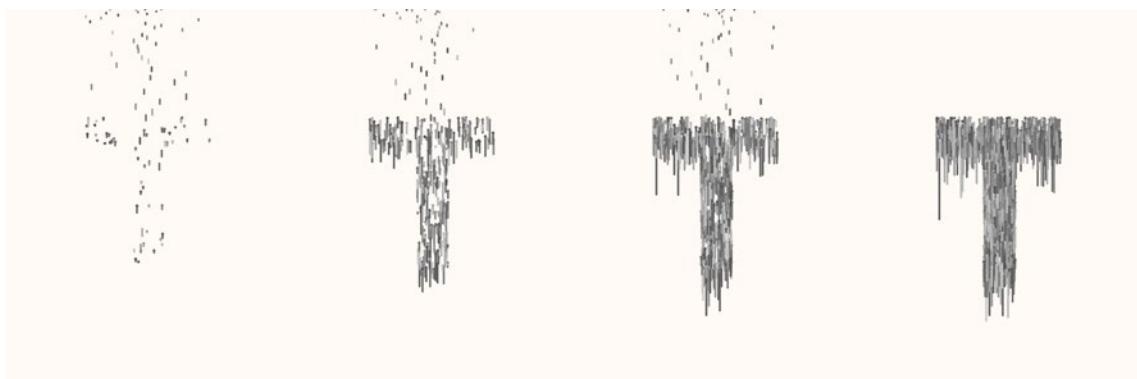


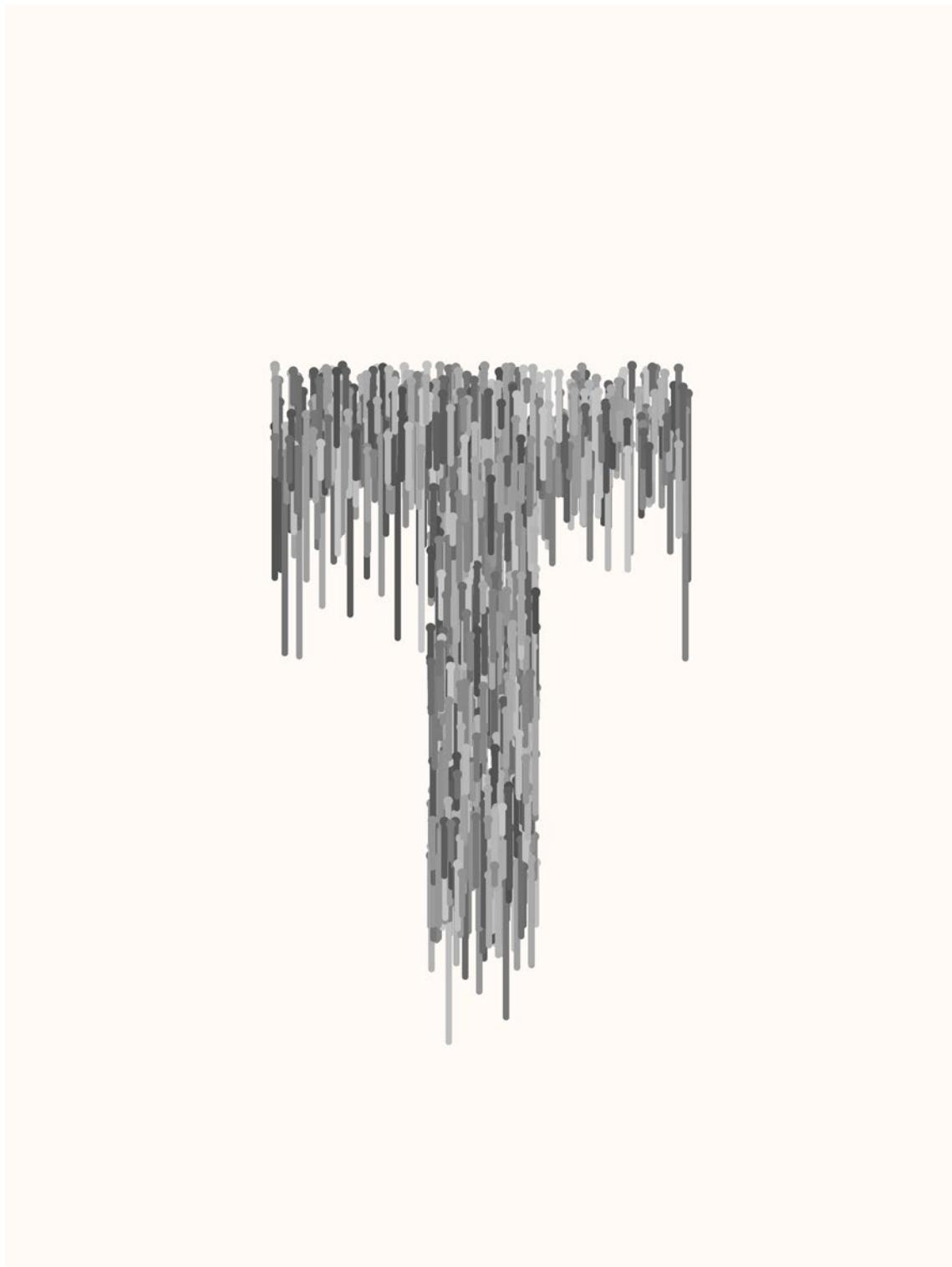
#### **4.21 \_DROP**

Uma chuva de tinta cai sobre o elemento gráfico e as gotas derramadas escorrem verticalmente pela tela evidenciado a forma do elemento.

O último programa da série cria uma composição de linhas verticais que crescem e param com tamanhos diferentes para formar, de maneira disforme, o elemento tipográfico.

Semelhante ao \_retro, linha retas crescem em um determinado sentido, mas ao invés de parar nas bordas do elemento gráfico, elas param de acordo com uma função randômica em posições variadas.







## **5. DISCUSSÃO**

A partir da correspondência entre os conhecimentos teóricos apresentados até o momento e o estudo prático efetuado no capítulo anterior, é importante desempenhar uma atividade de reflexão sobre o processo de design. Essa reflexão provoca o pensamento crítico sobre os resultados elaborados até então.

Refletir sobre a metodologia permite ter uma melhor compreensão sobre a cadeia de ações e decisões que acontecem durante o decorrer do processo de design, que normalmente são desempenhadas de maneira automática através de um conhecimento tácito. Tentar descrever esse conhecimento instiga o designer a ter um pensamento mais analítico sobre o seu modo de projetar.

A presença de uma parte prática dentro do trabalho se mostrou de grande valor, visto que foi possível obter uma melhor compreensão sobre o processo criativo. Na medida em que as experimentações eram desenvolvidas, um registro desse processo foi formado para que pudesse ser posteriormente analisado. Os pontos evidenciados por esse registro são detalhados a seguir.

### **5.1 COMPREENSÃO DO CÓDIGO**

Todos os experimentos começaram com estudos iniciais sobre como codificar os recursos a serem utilizados em cada programa. Esses estudos foram úteis para explorar funções nunca antes utilizadas por nós e testar diferentes abordagens e algoritmos para se chegar a um resultado satisfatório.

Do mesmo modo que o designer recorre a busca na internet por algum comando que satisfaça sua necessidade ao não saber executar alguma ação em um *software*, parte do processo de criação de alguns experimentos desse trabalho foi a busca por códigos-fonte e bibliotecas que cumprissem uma função necessária para a experimentação.

Essa prática revela que a programação é mais uma das várias ferramentas disponíveis ao designer e que como tal deve ser utilizada de maneira criteriosa. Ou seja, ao invés do designer aprender todos os comandos de um *software* para depois aplicá-los, parece fazer mais sentido aprendê-los à medida que ele necessite executar as funções relativas a esses comandos.

### **5.2 CONSTRUÇÃO TEMPORAL**

Os programas foram elaborados de modo que a construção da composição fosse feita gradativamente em função do tempo, como vemos na figura 44. Isso proporcionou duas percepções sobre o trabalho:

Uma delas é que os experimentos produzidos se tornaram dinâmicos, algo característico do meio digital que é difícil de se acontecer em uma arte impressa, por exemplo. Essa dinamicidade foi explorada em duas abordagens diferentes, uma foi durante o processo de construção das peças que duram um certo espaço de tempo e depois permanecem

estáticas. A outra abordagem faz com que as peças passem por construções constantes, perpetuando indefinidamente, como no projeto Process Compendium.

Outra percepção obtida foi a presença de um feedback em tempo real daquilo que se estava sendo programando. Isso facilitou a descoberta de que erros de código, já que um trecho de código sempre estaria atribuído a um tempo específico da construção da composição.

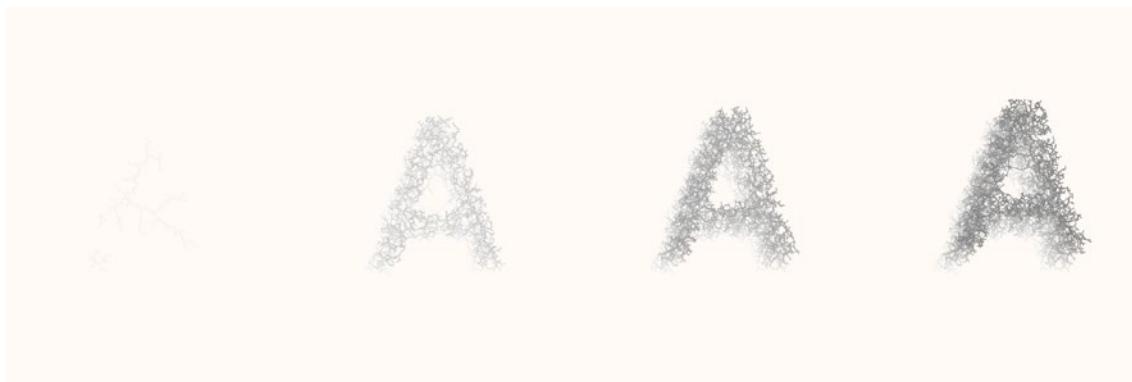


Figura 44: Construção gradativa das composições

### 5.3 EMERGÊNCIA NO PROCESSO DE DESIGN GENERATIVO

O termo “emergência” apareceu escrito frequentemente durante esse trabalho, inclusive na definição de design generativo apresentado na introdução. Porém esse termo ainda não foi definido. É possível classificar “emergência como um fator-chave no estágio inicial da exploração do desenho” no qual a “direção da exploração do desenho é dependente e é dirigida pelos resultados de explorações passadas - que é a característica chave da emergência”<sup>1</sup> (KRISH 2011, tradução nossa). Segundo Bohnacker:

“No contexto do design generativo, processos são emergentes quando seus resultados não são predeterminados e quando a interação de todos os elementos acarreta algo a mais do que é evidente a partir de suas propriedades individuais. Um exemplo comum de emergência é o comportamento aglomerado de pássaros: regras simples resultam em comportamentos imprevisíveis altamente complexos.”<sup>2</sup> (BOHNACKER 2012, p.463, tradução nossa)

Das definições acima, podemos concluir que o evento da emergência é uma consequência da conversa reflexiva, visto que há um processo de retroalimentação no qual um resultado evolui a partir de resultados passados. Também se conclui que o evento de emergência ocorre na composição gráfica do próprio resultado, visto que a ferramenta da programação oferece a capacidade de criar resultados infinitos a partir de um único programa.

1 “emergence as the key driver of early stage design exploration. In creative design processes, the direction of design exploration is dependent on and is directed by the result of previous explorations—which is the key characteristic of emergence.”

2 “In the context of generative design, processes are emergent when their results are not predetermined and when the interaction of all the elements leads to more than is obvious from their individual properties. A common example of emergence is the swarming behaviour of birds: simple rules result in highly complex, unpredictable behaviour.”

Consideramos que os experimentos apresentados no capítulo anterior podem ser considerados emergentes. Como o intuito era explorar as possibilidades da programação no processo de criação, nenhum resultado particular era esperado, sendo que ele evoluiu de acordo com explorações anteriores, caracterizando um processo emergente.

Juntando as características do processo emergente com os atributos do design generativo apresentados na introdução, podemos dizer também que os experimentos fazem parte desse enfoque exploratório do design.

#### **5.4 CRIAÇÃO NÃO INTENCIONAL A PARTIR DE “BUGS”**

Erros acidentais de código são comuns de acontecer e podem alterar o resultado, criando algo não desejado ou pretendido. Isso se deve ao fato do ato de se programar ser um processo de abstração de informação, como já foi mencionado anteriormente. Algumas vezes esses erros não intencionais podem gerar um efeito ou um produto interessante visualmente.

A ideia de que algo não intencional pode influenciar o processo de design complementa aquilo que foi dito sobre o processo reflexivo de que o designer ao se deparar com sua criação, intencional ou não, avalia o resultado e decide qual será sua próxima ação. Assim sendo ele tem a possibilidade de incorporar o efeito dentro do processo que estava em andamento, pode se desfazer dele, ou ainda pode se apropriar do “bug” e partir para outra direção e elaborar algo completamente diferente, criando uma ramificação no projeto.

Essa ramificação foi observada no desenvolvimento do programa noise.v1 do TCC1, por exemplo. Ao tentar fazer com que a randomização das linhas fossem dirigidas pela movimentação do mouse, um efeito visual completamente diferente foi obtido. O novo programa foi chamado de skyline.v1 e foi arquivado para ser utilizado em um experimento futuro.

Outro exemplo é o programa\_thread que foi criado a partir de uma alteração do programa \_tentacles. O resultado pareceu interessante e achamos que deveria ser transformado em um experimento diferente.

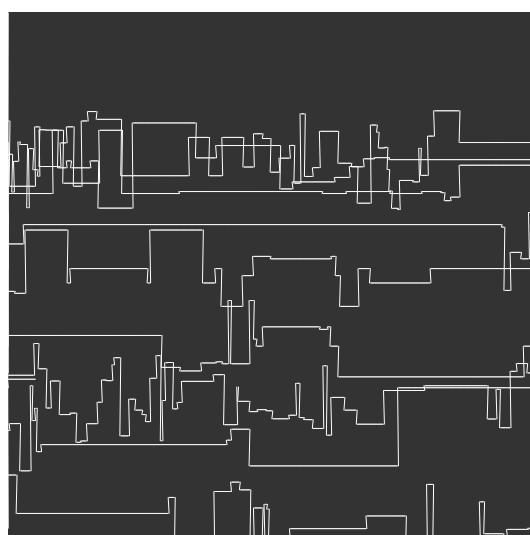


Figura 41: Versão alternativa do noise.v1, chamada de skyline.v1

## 5.5 HARDCODING: SOLUÇÃO IMPROVISADA

Ocasionalmente durante o processo de programação, há certos momentos em que o programador decide escrever um código menos elaborado ao invés um código paramétrico e mais elegante. Há casos em que isso traz prejuízos, como o gasto desnecessário de processamento ou a limitação de uso do programa, mas há casos em que essa decisão não influencia em nada o programa elaborado.

Uma das razões para se utilizar essa prática é a falta de conhecimento do programador em relação à linguagem programação, o que faz ele recorrer a comandos mais básicos e diretos ao invés de comandos mais rebuscados que ofereceriam mais automatização de uso ao programador.

Outra razão é a falta de necessidade de se ter funções paramétrica com dados dinâmicos. Portanto o esforço de se criar um código mais elegante e complicado se torna uma perda de tempo ao programador. Essa percepção se aplica a este trabalho já que estamos tratando de uma fase exploratória de projeto, sendo que os programas podem ser considerados como esboços ou protótipos que não precisam necessariamente ter um acabamento refinado visto que eles servem sobretudo para o estudo e investigação do enfoque em questão.

Por exemplo, para se criar um conjunto de cinco círculos é mais direto repetir várias linhas de código que desenhe essas formas do que se criar um loop, que necessita do uso de variáveis e de condições para desenhar essa forma de maneira iterativa.

Nos experimentos 2.0, por exemplo, a posição do elemento tipográfico teve que ser ajustada diretamente no código para que ele ficasse no centro da tela. Isso se deve pelo fato de cada programa ter interações diferentes com o fundo, o que afeta sua posição na tela. Além disso, cada letra ocupa um espaço diferente, dificultando a parametrização de suas posições.

Essa prática pode ser chamada de *hardcoding*<sup>3</sup>. Esse termo é abrangente e polêmico, já que não existe uma definição oficial. De acordo com o artigo da Wikipédia, *hardcoding* se refere a prática de incorporar um *input* ou de configuração diretamente no código-fonte ou formatar um dado de maneira fixa, que por outro lado, esse dado poderia ser obtido de uma fonte externa ou gerado a partir do próprio programa.

## 5.6 IMPORTÂNCIA DA FERRAMENTA OPEN SOURCE

O Processing é um *software* que funciona dentro de um sistema colaborativo no qual seus usuários podem compartilhar códigos-fonte em um ambiente virtual, como já foi explicado na introdução. Durante o desenvolvimento dos experimentos foi possível perceber a grande importância e as vantagens de se trabalhar dentro desse sistema.

A participação na comunidade virtual *OpenProcessing* representou um papel significativo dentro do processo de design. Como dito anteriormente, assim que uma função específica era necessária, mas não se tinha o conhecimento de seu código, era frequente a prática de recorrer à comunidade para averiguar se outros usuários já tinham produzido programas com funções semelhantes.

<sup>3</sup> [en.wikipedia.org/wiki/Hard\\_coding](https://en.wikipedia.org/wiki/Hard_coding)

Muitas vezes os usuários não só disponibilizam os códigos-fonte, como também escreviam comentários que ajudavam a entender a função de cada passagem de código. Esses usuários ainda ficam à disposição para discutir sobre o programa desenvolvido, uma vez que a comunidade também serve como rede social.

Dentro dessa comunidade, os códigos ficam à disposição para ser modificados por outros usuários. Foi comum observar usuários referenciando programas de outros usuários, dizendo que foram úteis para suas próprias produções. Também foi comum observar usuários postando dúvidas e respostas, assim como postagens de melhorias para programas de terceiros.

A partir da verificação prática, foi possível dizer que sistemas colaborativos alavancam a produção de conhecimento de maneira horizontal e mais democrática. Esses sistemas, como o *OpenProcessing*, incentivam o aprendizado, o treinamento técnico, assim como o compartilhamento de conteúdo expressivo para a área de estudo.

## 5.7 REFERÊNCIAS

Várias referências foram coletadas desde o início desse trabalho e serviram para embasar tanto a formação do panorama do design paramétrico quanto a criação das experimentações. Analisando essas referências foi possível perceber que havia uma divisão em três categorias: referências visuais feitas com a programação, referências visuais feitas com ferramentas tradicionais<sup>4</sup> e por último referências de códigos-fonte, como foi explicado no tópico anterior.

As referências visuais por código serviram para conhecer melhor qual é o potencial criativo da programação como ferramenta e para saber o que está sendo produzido atualmente. As referências visuais tradicionais, como as esculturas de Kang Duck-Bong, colaboraram com a apresentação de linguagens variadas que puderam ser usadas como inspiração para a atividade empírica.

## 5.8 EVOLUÇÃO

Analizando a sequência de experimentos, foi possível observar que houve uma evolução em relação a habilidade de programar. Esse avanço chegou no auge com o programa `_orbit`, que consideramos ser o mais complexo dos experimentos por ter exigido maior nível de abstração da ideia conceitual para se chegar a um código que a representasse. Finalizado o último experimento `_drop`, percebemos que a abstração entre ideia e código pareceu se tornar menos complicada e com isso o uso da ferramenta se integrou melhor no fluxo de trabalho e que isso se deveu ao exercício e a prática regular com esse tipo de ferramenta.

Um outro tipo de evolução que percebemos foi em relação aos programas em si. Percebemos que um programa foi possível de ser feito devido ao código criado nos programas anteriores a ele. O `_metaball` foi feito com base no `_pong`; o `_swirl` foi feito com base no `_tentacles`; o `_stipple` com base no `_outline`; e todos com base no `_dla`, que introduziu o teste de contato com o elemento gráfico. Acreditamos que o conhecimento

<sup>4</sup> Neste caso, ferramentas tradicionais são aquelas que não utilizam programação e abrangem tanto as ferramentas manuais como pincel, grafite, caneta, tinta guache, etc, quanto as ferramentas digitais como Photoshop, Illustrator, Cinema 4D, entre outros.

adquirido ao criar um programa criou a oportunidade para se ter novas possibilidades de ideias para se criarem outros experimentos.

## 5.9 CONTROLE

Um ponto percebido ao se trabalhar com design generativo foi a de que se tem pouco controle sobre a peça final. O domínio do designer em relação a peça final está limitado ao controle dos parâmetros e restrições das ações dos objetos presentes na tela. Uma vez que o programa começa a rodar, as decisões ficam nas mãos do computador e da casualidade. Essa característica está relacionada com o evento de emergência explicado no tópico 5.3 no qual o comportamento dos objetos é de certa maneira imprevisível.

Essa característica desse tipo de enfoque permite que peças gráficas adquiram formas diversificadas, umas mais desejáveis do que outras. Isso faz com que o programa precise ser rodado mais de uma vez para se chegar a uma peça pretendida.

## 5.10 MÉTODOS

Ao se trabalhar com uma ferramenta na área de projeto, é possível atingir um mesmo objetivo utilizando métodos diferentes, logo isso também acontece com a programação. No processo de tentar codificar uma ideia, diferentes sequências de passos podem ser pensados para se chegar a uma forma, sendo que sequências podem ser mais eficazes do que outras. Como nosso contato com a programação ainda não é muito consistente, os algoritmos utilizados para desenvolver os experimentos talvez não tenham sido os melhores e nem os mais eficientes.

Para exemplificar, o programa `_orbit` atual se baseia em uma função trigonométrica para criar a trajetória elíptica, porém antes disso ele utilizava outro método. O programa criava uma elipse, segmentava ela em diversos pontos equidistantes e fazia com que o objeto percorresse cada um desses pontos sequencialmente, criado assim a trajetória. Pensando ainda em um outro método possível, ao invés de apenas simular um efeito tridimensional no plano 2D, o programa poderia criar os objetos em um ambiente 3D de fato, no qual cada uma das orbitas ficariam situadas ao longo do eixo Z.

## 5.11 CODIFICAÇÃO

O nível de concentração exigido ao se trabalhar com programação é muito alto. É preciso prestar atenção em cada vírgula, em cada parênteses e em cada uma das variáveis porque qualquer erro pode fazer a diferença no resultado final.

Não foram raros os momentos em que o desenvolvimento dos programas ficou estagnado devido a um problema desconhecido, sendo que mais tarde foi descoberto que era algum erro relacionado a alguma variável que possuía um valor equivocado, como no caso do programa `_stipple`, por exemplo.

Organização é essencial para facilitar o entendimento e a escrita de um código. É fácil se perder na sequência de instruções, por isso é importante segmentar o código em pequenas funções a fim de deixar o programa mais intuitivo.

## 5.12 LINGUAGEM

Ao analisar a linguagem dos experimentos, percebemos que alguns recursos gráficos foram utilizados de maneira recorrente.

Entre esses recursos estão a **percepção visual de profundidade e tridimensionalidade**. Nos programas como \_dla, \_retro, \_orbit, \_grid, \_tentacles, entre outros, foram utilizadas técnicas de sobreposição, mudança de luminosidade e angulações para criar um efeito gráfico que destacasse os objetos do plano bidimensional.

Diferente de técnicas propriamente tridimensionais, os recursos utilizados exigem menos processamento do computador, são mais práticos e criam um visual diferenciado que se assemelha à colagem, no qual os vários objetos parecem ter sido colados uns sobre os outros. Esse efeito se deve à ausência da iluminação típica da representação 3D e também pela falta de perspectiva, criando uma aparência “achatada”.

Outro recurso bastante utilizado nos experimentos foi a **aparência orgânica e natural**. Podemos perceber esse recurso nos programas \_perlin, \_spline, \_tentacles, \_thread e \_swirl que incorporaram a função randômica Perlin Noise em seu código. Além disso o programa \_dla também cria uma peça gráfica orgânica ao simular um efeito natural.

Acreditamos que o efeito orgânico nos pareça interessante devido ao fato dessa linguagem natural poder ser sintetizada em um algoritmo e recriada a partir de cálculos matemáticos. É curioso perceber e atestar de maneira visual que a lógica matemática pode gerar formas caóticas e naturais, diferente das formas retilíneas e duras que a priori são as mais previsíveis. Como disse Matt Pearson, “o artista generativo vem do mundo da lógica para olhar o mundo natural em busca de inspiração”<sup>5</sup> (PEARSON, 2011, tradução nossa).

O **controle de múltiplos objetos** também pode ser considerado como um recurso gráfico, já que é através das interações praticadas por eles que a peça gráfica é gerada. Ao observar os experimentos, temos a percepção visual do sistema como um todo, e não como algo fragmentado no qual podemos identificar cada um dos componentes simultaneamente. Involuntariamente tendemos a agrupar as formas que vemos, e é devido a esse agrupamento que podemos identificar os elementos tipográficos nos experimentos.

Um outro tipo de recurso foi a **implementação de inconstâncias e ruídos gráficos** para que a estética baseada na regularidade intrínseca do computador fosse quebrada. No programa \_spline, a linha que percorre o elemento não o completa por inteiro, dando a impressão de que ele possui uma falha de construção. No \_orbit, algumas trajetórias são propositalmente deslocadas do centro original. No \_explode, os objetos possuem tempos de vida e velocidades diferentes, criando distâncias diferentes uma vez que eles não param simultaneamente.

A partir da constatação de que o ruído gráfico é interessante visualmente e de que isso pode ser observado nos experimentos realizados, é possível concluir que o uso eficaz da linguagem de programação no processo de design acontece quando abordado de maneira diferenciada, atípica e que tende a fugir da linguagem previsível do computador.

5 “The generative artist comes from the world of logic to look toward the natural world for inspiration”



## 6. EXPERIMENTOS 3.0

Até o momento o trabalho consistiu em apresentar o enfoque do design que utiliza a programação como ferramenta, gerar experimentos iniciais que serviram para gerar as primeiras reflexões sobre o assunto, gerar uma segunda fase de experimentos um pouco mais refinados que serviram para continuar a investigação sobre as possibilidades da ferramenta, discutir os pontos que foram revelados ao longo da fase de exploração.

Pensando sobre qual seria o próximo passo, nos pareceu natural integrar o uso da tecnologia explorada ao longo do trabalho com a área da programação visual. Neste capítulo exibimos portanto uma aplicação do nosso trabalho, que ganhou ao longo do tempo um caráter técnico e conceitual. Não consideramos os experimentos desse capítulo como sendo a síntese do trabalho, mas sim como uma possível aplicação, visto que a partir dos experimentos do capítulo 6, outras propostas poderiam ter sido geradas ao invés das aplicações apresentada aqui.

Os experimentos deste capítulo são divididos em três grandes programas. Cada um deles atua com a finalidade de formar uma frase diferente. A frase é apresentada de maneira segmentada no qual cada palavra, ou conjunto de palavras, é exibida em uma cena diferente. Cada cena é formada por um dos experimentos utilizados nos experimentos 2.0, sendo que o conjunto dessas cenas forma a frase por inteira.

Essas frases são citações de alguns personagens da história que de alguma forma possui uma relação semântica com o trabalho. As citações foram retiradas de um blog chamado “*but does it float*”<sup>1</sup> que apresenta conteúdo relacionado à arte, arquitetura e design. Cada post desse blog tem como título uma citação que possui alguma relação conceitual com o assunto tratado no post. Apesar desse blog ter postagens que fazem referência ao design generativo, curiosamente nenhum dos títulos escolhidos para este trabalho estavam relacionados a essas postagens.

Como no trabalho estudamos as possibilidades do processo de criação utilizando uma ferramenta particular, decidimos escolher frases que manifestassem e refletissem as práticas e os conceitos relacionados ao processo estudado.

As imagens presentes nesse capítulo são frames de cada uma das cenas dos programas e foram colocados justapostos para se ter uma ideia geral do experimento. Lembramos que os programas executáveis e os vídeos dos três experimentos estão presentes no CD anexado ao relatório.

<sup>1</sup> [www.butdoesitfloat.com](http://www.butdoesitfloat.com)

## **6.1 THE SOURCE OF THE NEW IS THE RANDOM - GREGORY BATESON<sup>2</sup>**

Todos os experimentos realizados utilizaram algum tipo de função randômica que guiou e direcionou as interações dos programas para certos rumos que, mesmo de caráter aleatório, formaram uma linguagem distinta e exclusiva.

Nesse contexto, o resultado gráfico adquiriu caráter único, singular, peculiar, original e novo a partir da formulação randômica, como podemos interpretar da frase “A origem do novo é o randômico”<sup>3</sup>. Assim, a ideia da frase se relaciona de maneira direta com os eventos emergentes que norteiam os experimentos.

Também podemos relacionar a frase com a ideia de imprevisibilidade que ocorre na conversa reflexiva. Ao se deparar com a peça criada, novas ideias podem ou não surgir. Essas ideias vêm de associações cognitivas com os conhecimentos únicos adquiridos pelo designer ao refletir sobre a peça. Porém, o acesso a esse conhecimento é de certo modo imprevisível, já que não temos controle sobre quais associações com a peça podem ser feitas.

A frase foi elaborada por Gregory Bateson (1904 - 1980) que foi antropólogo, sociólogo, biólogo e cientista da comunicação nascido na Inglaterra. Ele focou seu estudo principalmente na área da comunicação sob o ponto de vista epistemológico.

2 Fonte: [www.butdoesitfloat.com/The-source-of-the-new-is-the-random](http://www.butdoesitfloat.com/The-source-of-the-new-is-the-random)

3 Tradução nossa

# THE SOURCE

THE SOURCE

THE RANDOM

## **6.2 INTENTION IS THE KEY AND THE PROCESS IS THE PRODUCT - GENESIS P-ORRIDGE<sup>4</sup>**

Retomando o conceito de conversa reflexiva e a representação do processo de design de Bohnacker, a ideia é o ponto de partida no qual inicia o processo de criação. Ela é o objetivo inicial o qual pretendemos atingir através da abstração para se criar um programa.

Se a ideia é o objetivo, o resultado do desenvolvimento dessa ideia é o código de programação. O código é o conjunto de passos que o computador executa para formar a peça gráfica. Assim como na arte de Sol LeWitt, o código denota diretamente a ideia de processo e é o produto concretizado. As peças gráficas geradas por esse código são meras representações, interpretações, encenações criadas pelo computador.

A frase “Intenção é a chave e o processo é o produto”<sup>5</sup> foi criada por Genesis P-Orridge (1950), uma compositora, musicista, poeta e artista performática inglesa.

4 Fonte: [www.butdoesitfloat.com/Intention-is-the-key-and-the-process-is-the-product](http://www.butdoesitfloat.com/Intention-is-the-key-and-the-process-is-the-product)

5 Tradução nossa

# Intention

the process

AND

the process

IS

the product

### **6.3 A SET IS A MANY THAT ALLOWS ITSELF TO BE THOUGHT OF AS A ONE - GEORG CANTOR<sup>6</sup>**

A frase “Um conjunto é um Muitos que permite ser pensado como Um”<sup>7</sup> foi expressa por Georg Cantor (1845 - 1918), matemático e filósofo russo que ficou conhecido por ter criado a Teoria dos Conjuntos.

Os experimentos 2.0 utilizaram técnicas de controle de sistemas de partículas e de agente autônomos. Esses objetos podem ser considerados como sendo pequenas partes particulares e individuais que juntas formam um todo, um conjunto. Assim como dissemos no capítulo 5.11, percebemos o “Muitos” não como indivíduos isolados, mas como um único grupo, uma única forma, um grande “Um”.

Expandindo essa ideia, podemos falar que os experimentos realizados não são vistos de maneira isolada, mas sim dentro do conjunto de todos os experimentos. Pensando dessa forma, não só a peça final, mas o processo de criação se inclui dentro desse conjunto, junto com as reflexões, discussões e decisões. Agregando esses atributos, algo maior e mais complexo é criado, que pode ser representado por esse trabalho como um todo.

Seguindo esta linha de raciocínio, pode-se dizer que nos experimentos 3.0, na programação e neste trabalho em geral, podemos encontrar características do pensamento reducionista na investigação exploratória, que divide os problemas, os conceitos e as ideias intrincadas em partes menores para facilitar a compreensão e resolução destes.

6 Fonte: [www.butdoesitfloat.com/A-set-is-a-Many-that-allows-itself-to-be-thought-of-as-a-One](http://www.butdoesitfloat.com/A-set-is-a-Many-that-allows-itself-to-be-thought-of-as-a-One)

7 Tradução nossa





## **7. CONCLUSÃO**

Este capítulo apresenta conclusões obtidas durante o trabalho que, por sua vez, tinha o objetivo de realizar um estudo exploratório sobre o uso do design paramétrico e generativo na atualidade e de estudar a influência da programação no processo de design e em sua linguagem visual.

A exploração prática através da criação de experimentos foi fundamental para a validação e compreensão dos conceitos apresentados no trabalho. A reflexão sobre essa exploração empírica permitiu que dados ocultos pelo conhecimento tácito fossem revelados e analisados, gerando pontos que foram argumentados no capítulo “Discussão”. Essa exploração também permitiu o exercício da prática de projeto com enfoque no design paramétrico.

Como explicado no capítulo três, a programação influencia o processo de design se integrando ao ciclo da conversa reflexiva desse processo. Esta conversa reflexiva foi observada durante a realização dos experimentos, visto que soluções foram criadas a partir de resultados anteriores em um processo emergente e reflexivo.

A ferramenta de programação Processing, utilizada neste trabalho, evidenciou a importância da cultura open source, visto que pudemos participar de um ambiente colaborativo que emprega essa filosofia. Essa participação auxiliou o processo de design ao permitir que códigos disponíveis on-line fossem estudados com a finalidade de serem implementados no desenvolvimento dos experimentos realizados nesse trabalho.

O uso de referências foi de grande importância ao complementar e potencializar os experimentos práticos. O panorama forneceu uma visão horizontal sobre o cenário do design paramétrico atual ao passo que o histórico mostrou uma proposta vertical ao mostrar fatos contextuais.

Como a exploração do trabalho foca no uso dos processos de design na busca de novas possibilidades computacionais de projeto, julgamos apropriado categorizar este trabalho sob o ponto de vista do design generativo, enfoque que intitula este trabalho.

Do trabalho apresentado, foi possível perceber a pontencialidade da ferramenta da programação. Sua versatilidade vai de encontro as necessidades atuais nas diversas áreas de projeto que cada vez mais exigem instrumentos mais especializados e customizáveis.

A ferramenta por si só não é criativa. A exploração da linguagem acontece ao integrar a capacidade das ferramentas utilizadas com o interesse, a intenção, a prática e os conhecimentos do designer, que são adquiridos e transformados ao longo do tempo. A partir da exploração investigativa da programação, um conjunto particular de experimentos foi gerado neste trabalho. Esse conjunto reflete as decisões de projeto tomadas aos longo do processo, reflete nosso planejamento, reflete o nosso gosto visual e também reflete nossa competência com a ferramenta.



## 8. REFERÊNCIAS BIBLIOGRÁFICAS

- BARRIOS HERNANDEZ, C. R. **Thinking parametric design: introducing parametric Gaudi.** Design Studies, maio. 2006. v. 27, n. 3, p. 309–324.
- BOHNACKER, H. **Generative design: visualize, program, and create with processing.** New York: Princeton Architectural Press, 2012.
- BUCHANAN, Richard, 1992. **Wicked Problems in Design Thinking.** In: Design Issues. 1992. Vol. VIII, no. 2.
- CARLI, L; BARROS, G. **O papel da programação de computador na conversa reflexiva do processo de design gráfico de visualização da informação.** 4º Congresso Sul Americano de Design de Interação. 2012
- DAVIS, D. **A History of Parametric.** [S.I.], 2013. Disponível em: <<http://www.danieldavis.com/a-history-of-parametric/>>.
- KOLAREVIC, B. **Digital Morphogenesis and Computational Architectures**, in Proceedings of the 4th Conference of Congreso Iberoamericano de Grafica Digital, SIGRADI 2000 - Construindo o Espaço Digital, Rio de Janeiro, Setembro 2000.
- KRISH, S. **A practical generative design method**, jan. 2011. v. 43, n. 1, p. 88–100.
- MENGES, A.; AHLQUIST, S. **Computational design thinking.** Chichester, UK: John Wiley & Sons, 2011.
- MONEDERO, J. **Parametric design: a review and some experiences.** Automation in Construction, jul. 2000. v. 9, n. 4, p. 369–377. . Acesso em: 13 nov. 2013.
- PEARSON, M. **Generative Art: A Practical Guide Using Processing.** New York : Manning, 2011
- NADIN, M. **Computational Design: Design in the Age of a Knowledge Society.** in form/diskurs Journal of Design and Design Theory, 2, 1/1996. Frankfurt/Main: Verlag Form, pp. 40-60. Disponível em: <<http://www.nadin.ws/archives/125>>
- REAS, C.; LUST. **Form+code in design, art, and architecture.** 1st ed ed. New York: Princeton Architectural Press, 2010.
- REAS, C. **Process Compendium text.** [S.I.], 2010. Disponível em: <[http://www.reas.com/compendium\\_text/](http://www.reas.com/compendium_text/)>.
- REAS, C. **Process Compendium (Introduction).** [S.I.] 2012. Disponível em: <<http://vimeo.com/22955812>>.
- SCHON, D. A. **Designing as reflective conversation with the materials of a design situation.** Research in Engineering Design, v. 3, n. 3, p. 131–147, 1992.
- SCHUMACHER, P. **Parametricism as Style - Parametricist Manifesto.** [S.I.], 2008. Disponível em: <<http://www.patrikschumacher.com/Texts/Parametricism%20as%20Style.htm>>.
- SHEA, K.; AISCH, R.; GOURTOVAIA, M. **Towards integrated performance-driven generative design tools.** Automation in Construction, mar. 2005. v. 14, n. 2, p. 253–264.
- SHIFFMAN, D. **The nature of the code.** [S.I.] Shiffman, 2012
- WEISSTEIN, E. W. **CRC concise encyclopedia of mathematics.** 2nd ed. Boca Raton: Chapman & Hall/CRC, 2003.



## **9. APÊNDICE: EXPERIMENTOS 1.0**

Como parte essencial do trabalho, decidimos realizar uma série de experimentos ainda no TCC1 para investigar e analisar a influência e a participação da programação dentro do processo de design e também para explorar as possibilidades da programação.

Para isso, definimos conceitos que serviram de fundamento para a atividade prática e em seguida segmentamos os experimentos em quatro diferentes abordagens para facilitar o desenvolvimento dos artefatos. Por fim uma discussão foi realizada a fim de elaborar uma visão crítica sobre o processo criativo como um todo, desde os estudos iniciais até a avaliação do resultado obtido.

### **9.1 FUNDAMENTOS**

De acordo com Bohnacker em seu livro “Generative Design”, existem três conceitos básicos na linguagem de programação que funcionam como “blocos elementares de construção” (BOHNACKER, 2012) que conduzem o processo criativo do design paramétrico. São eles: a lógica, a randomização e a repetição.

Como crítica a esse conjunto de fundamentos, é proposta a substituição da “repetição” por um outro conceito, visto que ela é uma função intrínseca da lógica de programação, o que torna de certo modo redundante ter “repetição” e “lógica” como fundamentos. Uma vez que a “repetição” é de grande importância, não só como estrutura fundamental, mas como recurso gráfico, ela é apresentada como uma abordagem de experimentação.

Como proposta alternativa, o conceito de “repetição” é substituído pelo conceito de “parametrização” nesse trabalho. Esta última se relaciona diretamente com o design paramétrico e se mostra como um bloco elementar condutor dentro do processo de design.

Os fundamentos influenciam o processo de design devido ao fato da programação ser a ferramenta para a criação de conteúdo no design paramétrico. Por isso, a gama de possibilidades daquilo que pode ser produzido é delimitado de acordo com a potencialidade e a capacidade da ferramenta utilizada.

#### **9.1.1 Lógica**

Lógica se refere à linguagem de programação que “no sentido de estruturas de controle, é usada para guiar o processo gerativo. Condições podem ser estabelecidas com o qual o programa flui e pode ser divergido para diferentes direções.”<sup>1</sup> (BOHNACKER 2012, tradução nossa). Esse conceito é complementar aos fundamentos da lógica de programação apresentados no capítulo anterior, que servem de alicerce à prática a ser desenvolvida.

<sup>1</sup> “in the sense of control structures, is used to guide the generative process. Conditions can be established with which the program flow can be diverted into different directions.”

### **9.1.3 Parametrização**

“Identificar e descrever os elementos variáveis em um processo - seja a seção de um código ou as regras de um poema dadaísta - é chamado de parametrização. Esse processo constituído por múltiplas etapas exige que o designer decida tanto o que pode ser alterado quanto a gama de possíveis valores para cada parâmetro.”<sup>2</sup> (REAS; LUST 2010).

Em programação, parâmetros são dados variáveis<sup>3</sup> que geram um efeito. A utilização de variáveis para influenciar aspectos no design proporciona a automatização dos processos pelo computador e o controle do que e de quanto há variações em uma composição, como foi dito na definição de ‘parâmetro’ na introdução desse trabalho.

### **9.1.2 Randomização**

“Randomização é usada para criar variações para quebrar a regularidade estrita do computador. Randomização pura raramente produz resultados interessantes de composição. Estes são criados quando a randomização é limitada e aplicada em doses moderadas.”<sup>4</sup> (BOHNACKER 2012).

A randomização está mais presente no design generativo do que no design paramétrico devido ao primeiro trabalhar com processos não lineares e/ou morfogenéticos que necessitam de elementos randômicos para funcionar.

## **9.2 ABORDAGENS**

No livro “Form + Code” de Casey Reas e do estúdio LUST, os trabalhos apresentados estão divididos em uma taxonomia proposta pelos autores para distinguir diversos conceitos relevantes a produção digital feita pelo uso de código. As obras estão divididas em: repetição, transformação, parametrização, visualização e simulação.<sup>5</sup>

Assim como no livro, a decisão de dividir os experimentos por seções, aqui chamadas de abordagens, foi feita apenas por questões didáticas a fim de facilitar a compreensão dos recursos utilizados para gerar um experimento, já que na prática essa divisão não existe. Nesse trabalho foi determinado que a taxonomia utilizada seria baseada na taxonomia do livro, mas com a proposta de retirada da seção ‘parametrização’. Sobre essa seção os autores escrevem que:

“Pensar sobre parâmetros cria uma ponte entre repetição e transformação, assim como visualização e simulação. Enquanto transformação descreve um efeito do parâmetro na forma, a repetição oferece caminhos para explorar um campo de projetos possíveis para variações favoráveis. Ambas as visualização e simulação requerem o uso de parâmetros para definir o sistema e eles descrevem como dados ou outros *inputs* vão influenciar o

2 “Identifying and describing the variable elements in a process - be it a section of code or the rules of a Dadaist poem - is called parametrization. This multistep process requires that the designer decide both what can change and the range of possible values for each parameter”

3 O conceito de variáveis já foi explicado no tópico “Programação”.

4 “Randomness is used to create variations to break up the strict regularity of the computer. True randomness rarely produces compositionally interesting results. These are created instead when randomness is limited and applied in measured doses.”

5 As seções em inglês são chamadas de: repeat, transform, parameterize, visualize, simulate.

comportamento daquele sistema.”<sup>6</sup> (REAS; LUST 2010, tradução nossa)

A partir dessa definição, concluímos que a parametrização é um conceito fundamental para as outras seções, como já foi colocado no tópico ‘Fundamentos’, assim sendo, ela não é utilizada diretamente como experimentação.

Para cada programa criado, atribuímos um nome que será utilizado para identificar e diferenciar a abordagem de cada um deles. São eles: **noise.v2** para repetição, **jimi.v2** para transformação, **book.v2** para visualização e **dla.v3** para simulação. O número presente em cada nome representa a versão do programa. O nome do programa **dla.v3**, por exemplo, mostra que houveram duas versões anteriores para que o resultado atual pudesse ser obtido, **dla.v1** e **dla.v2**.

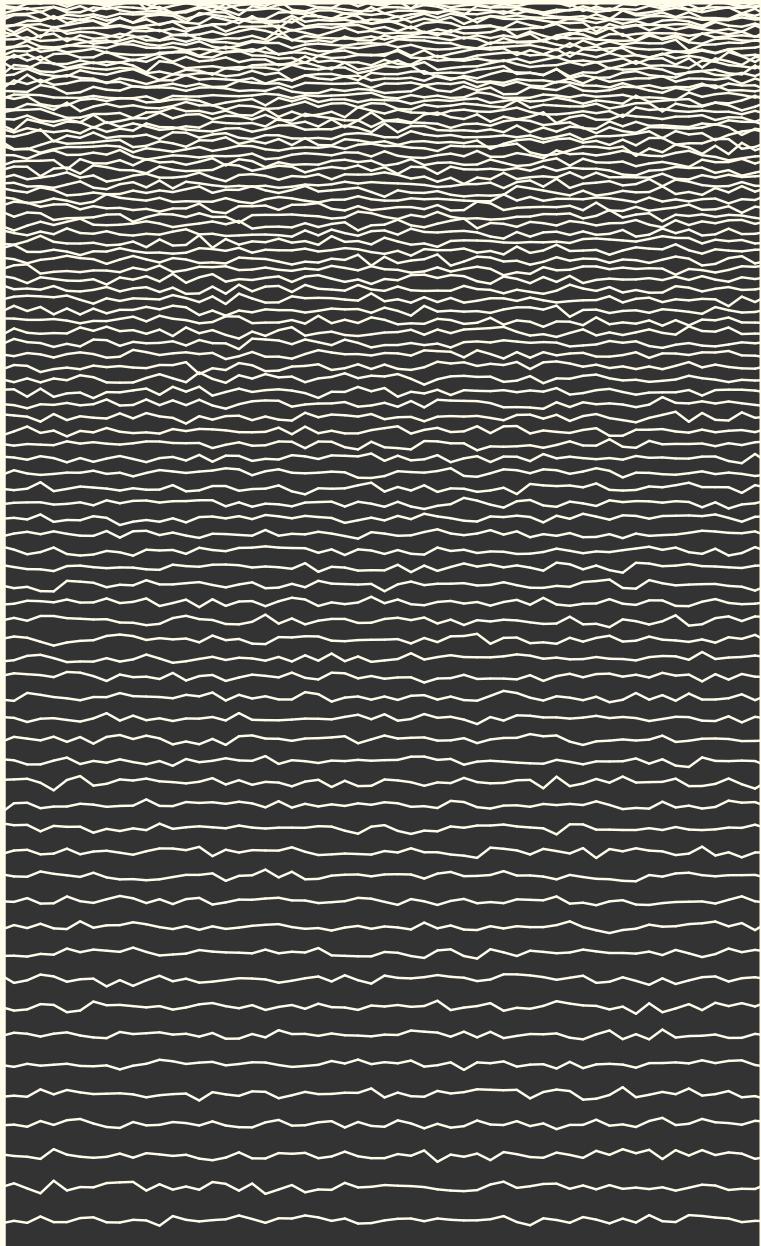
6 “Thinking about parameters provides a bridge between repetition and transformation, as well as visualization and simulation. While transformation describes a parameter’s effect on form, repetition offers a way to explore a field of possible designs for favorable variations. Both visualizations and simulation require the use of parameters to define the system, and they describe how data or other inputs will influence the behaviors of that system”

### 9.2.1 Repetição

Como já dito anteriormente, a repetição é um recurso básico quando se trata de linguagem de programação, tanto na parte estrutural do código representado pelas iterações quanto na repetição de formas em uma composição.

No experimento **noise.v2** linhas horizontais são repetidas ao longo da vertical do layout em uma progressão exponencial. Cada uma dessas linhas é composta por um conjunto de linhas menores que possuem angulações variadas de acordo com uma função randômica conhecida como Perlin Noise, apresentada no panorama desse trabalho. O final de uma linha sempre coincide com o começo de outra, o que causa o efeito de ser uma linha contínua que segue de um lado para o outro do layout, porém inconstante e tortuosa. A capa de disco de vinil da banda Joy Division intitulado Unknown Pleasures<sup>7</sup> de 1979 foi utilizada como referência visual para a criação da peça.

<sup>7</sup> Para visualizar a capa e obter mais informações sobre ela, visitar: [www.ideafixa.com/a-historia-por-tras-da-capa-do-joy-division](http://www.ideafixa.com/a-historia-por-tras-da-capa-do-joy-division)



### **9.2.2 Transformação**

Dois recurso frequentemente utilizados na manipulação de formas são a transformação geométrica e a transcodificação. O primeiro envolve o ato de se alterar uma imagem ou forma utilizando mecanismos como: movimentação, rotação, escalonamento, esticamento, espelhamento, torção ou distorção. O segundo se refere a conversão de algum dado ou informação de um formato em outro distinto.

O experimento **jimi.v2** trata desses dois tipos de transformação. Nele, uma matriz de triângulos é criada movendo cada elemento e rotacionando alternadamente em 180° cada um deles. Um arquivo de imagem é carregado pelo programa e a cor de cada pixel dessa imagem é convertida para uma outra cor, dentro da escala de cinza, que por sua vez, é usada para colorir cada triângulo, localizado em um ponto análogo a um ponto da matriz em que o cor foi retirada. Alguns triângulos são escolhidos randomicamente e são pintados com uma cor aleatória dessaturada a fim de criar um ruído estético na composição. A imagem usada neste experimento é um retrato do músico Jimi Hendrix.



### 9.2.3 Visualização

Mapas, gráficos, tabelas, diagramas e infográficos são alguns exemplos de como os dados precisam ser organizados, ordenados e transformados em uma linguagem mais visual para que a comunicação com o receptor seja feita de maneira mais rápida, objetiva e direta, visto que certas informações ficam encobertas antes de serem passadas por esse ‘tratamento visual’.

A visualização de dados e de informação é uma ferramenta que conecta o design paramétrico à estatística, visto que a programação pode lidar com uma quantidade extremamente alta de dados estatísticos de maneira automatizada e mais rápida do que ferramentas tradicionais.

A experimentação **book.v2** utiliza um arquivo de texto como *input* com a finalidade de ordenar as palavras desse texto em ordem de frequência, neste caso o texto é o romance Dom Casmurro do escritor Machado de Assis. As palavras aparecem desde aquela mais frequente no texto e seguem em ordem decrescente. O tamanho das palavras diminui a cada palavra exibidas na tela. Na parte inferior do layout há um gráfico de barras que mostra as cinco palavras mais frequentes, exibindo o número de ocorrências de cada uma. O total de palavras no texto a parece na última barra em comparação com a quantidade de repetições da palavra mais frequente.

QUE DE NÃO ME SE UM É OS  
DA MAS DO AS ERA PARA COM EU  
LHE EM UMA COMO AO POR MINHA NO  
CAPITU MAIS NA OU À NEM ELE MÃE FOI  
DOS DIAS QUANDO TUDO TAMBÉM CAPÍTULO ELA  
DISSE SÓ CASA MEU SER JÁ OLHOS MIM JOSÉ  
ASSIM MUITO HÁ DEPOIS VEZ AGORA DAS AINDA  
OUTRA VOCÊ SEM NADA TINHA NOS LO TÃO TEMPO  
MESMO PADRE DIA ESCOBAR OUTRO ERAM ATÉ ENTÃO IA  
DIZER COISA SÃO BEM ESTA FEZ ISTO ESTAVA ESTE AQUI  
AOS ANTES VIDA SEMINÁRIO PODE MELHOR FOSSE DELA  
LOGO LA VER IR RUA MAL PALAVRAS IDÉIA OUTROS TAL  
POUCO ÀS FILHO PAI PODIA ANOS TODOS ALGUMA VERDADE  
DEUS ESTÁ TANTO GRANDE VI HOMEM PELA TEM ALGUNS COMIGO SIM  
CÉU PRIMA QUIS FALAR CABEÇA MÃO PORQUE DOIS MEUS ISSO LÁ  
SEUS POIS TODAS NOSSA GLÓRIA MAMÃE BENTINHO ALGUM MESMA  
FIM SUA FUI COSME TIO SEI JUSTINA TER PORTA OUTRAS TARDE  
QUEM DUAS FORA PÉ PALAVRA EZEQUIEL PRIMEIRO SENHOR SEMPRE  
ENTRE DEU DAR FAZER SEU MENOS OUVI PELO CREIO TODO GESTO  
ELES SI NÓS CORAÇÃO VAI NUNCA QUERIA SABER AMIGO RESTO  
NOITE VOZ PÁDUA DELE DEPRESSA SALA ONDE PESSOA PRIMEIRA  
ACHEI NATURALMENTE TODA BOCA NAS D GENTE VEIO PERGUNTOU  
MÃOS VEZES ALGUMAS SENHORA NOSSO ALMA MUNDO PRECISO SANCHA AQUELA  
HAVIA DENTRO VELHO SOBRE MEIO SERIA TRÊS RAZÃO PESSOAS QUASE CARA  
ALI LEMBRA PARTE BOM CASO DIZIA MORTE ORA TALVEZ PARECE FILHA SUAS  
VOU GOSTO FIQUEI ENTRAR DESDE CERTO DIGO VAMOS SAIR UNS SEJA CABRAL  
FALAVA MULHER LONGE ESTAR LIVRO TENHO MUITA QUER MODO ANO NINGUÉM  
DAVA PAPAI MINUTOS VINHA PAPEL TIVE MISSA CÁ TRAZIA AGREGADO MINHAS  
ACABOU GRAÇA VÉ COSTUME VONTADE PONTO BOA VOCAÇÃO SIMPLES HORA  
LIVROS FIZ ESTOU COISAS PROMESSA MESES SABIA PEDIR DEVIA AR DEI FORAM  
NOVA CONTRÁRIOLADO MANHÃ AMIGA QUALQUERJANELA MOÇA APENAS TAIS TU RESPONDEU  
REALMENTEFICAR DIZENDO FICOU APESAR HORAS FAZIA ESSE SENTI PROTNOTÁRIO VÀ  
MAIOR HAVER NOSSAS CADA MAR DIGA SAIU CONTAR LOS CAMA SABE FORÇA NENHUMA  
POSTO MEDO FICA FALOU MATA CAVALOS GRANDES IDADE MEMÓRIA DESTE PRINCÍPIO  
EUROPA LATIM DELES POSSO SANTA ENTROU AMIGOS AMANHÃ PEQUENO PARTICULAR QUANTO  
PENSEI FAMÍLIA IGREJA QUERO CONTINUOU AQUELE CONTAVA PERTO NENHUM LEITOR  
VÉSPERA CHEGUEI DELAS IAM CLARO MANDUCA SEGUINTE ESSA SENTIDO CABELOS  
NOME INSTANTES CORREDOR QUARTO VIVIA SEGUNDO TERRA AMOR DÀ FLOR HOJE PEDIU  
VIA PORÉM AFINAL JANTAR DAQUI ESCRIVER DAQUELA MUITOS DOCE HISTÓRIA AÍ FAZ  
LÁGRIMAS PUDE IDÉIAS NATURAL EFEITO RETRATO VERSO ANDAR OCASIÃO POSSÍVEL  
GESTOS SENTIA ACHAVA VIR CAPAZ LAS FAVOR ACABAR CONTEI PASSO DESEJO ROSTO  
ALGUÉM OUVIR NESTE PARECIA MORREU CIMA MARIDO OPINIÃO ESTAS PRÓPRIO DEIXEI

SE [850]

ME [1012]

NÃO [1531]

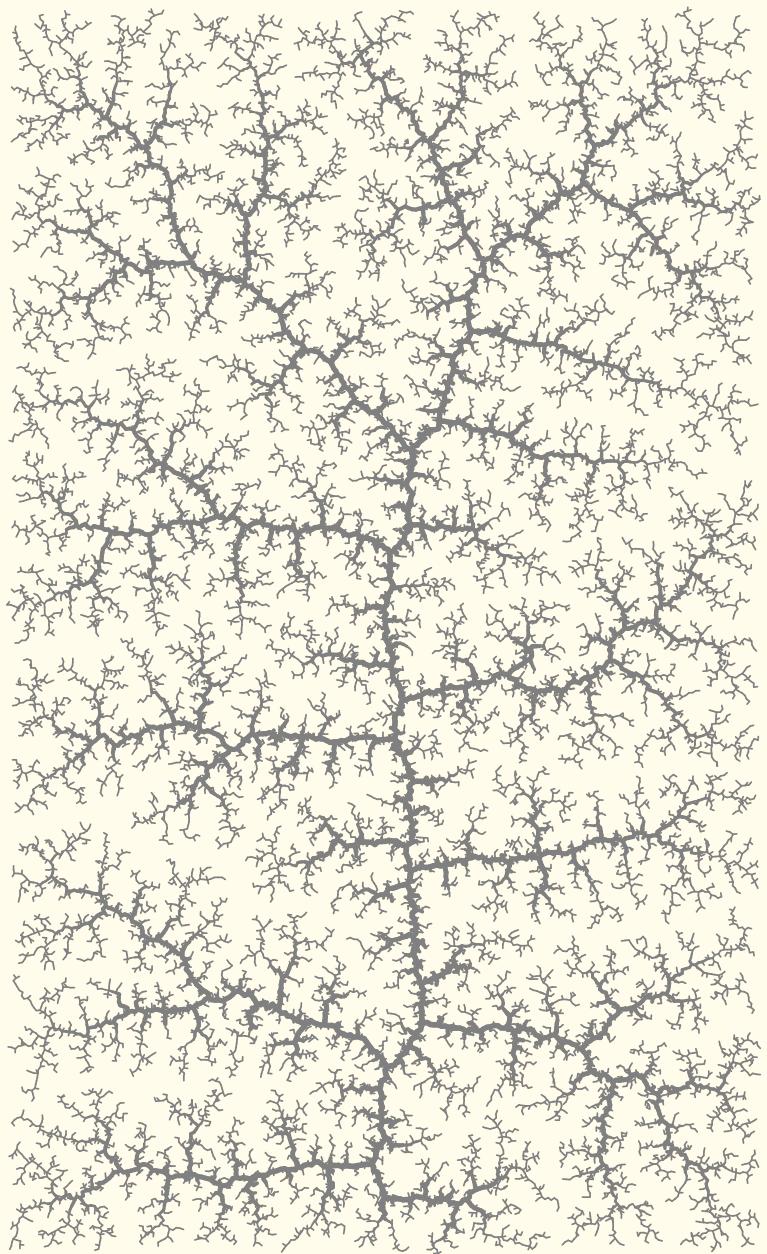
DE [1970]

QUE [2684/66846]

#### **9.2.4 Simulação**

Agregação por difusão limitada (DLA) descreve o comportamento do crescimento natural de algumas plantas, cristais, raios (descarga elétrica) e minerais. Esse comportamento é baseado na criação de uma “estrutura estável resultada da convergência de múltiplos agentes baseados em uma regra simples” (BOHNACKER 2012). Hoje em dia é possível encontrar algoritmos que simulam esse crescimento permitindo seu estudo e sua aplicação nas área de design e de artes visuais.

A experimentação **dla.v3** utiliza um algoritmo simples que simula o princípio do DLA. O crescimento inicia em um único ponto de origem localizado na parte inferior central do layout. A medida que agentes independentes e randômicos se agregam a esse ponto uma estrutura ramificada de 25.000 segmentos é gerada. A espessura dos segmentos reduz a medida que a estrutura se estende na composição. Uma restrição é criada para que o crescimento fique limitado a um espaço retangular, criando assim uma borda por princípios da gestalt.





## INTRUÇÕES PARA USO DO CD

O CD em anexo contém os programas relativos aos experimentos 1.0, 2.0 e 3.0 junto com vídeos desses respectivos programas em execução. Para o melhor uso deste CD recomendamos ler as instruções abaixo<sup>1</sup>:

- Os programas estão divididos em pastas por **sistema operacional**. Certifique-se de abrir a pasta do seu respectivo sistema. Copie as pastas para o computador antes de rodar os programas.

- É necessário ter o plugin **Java** instalado no computador para executar os programas. Caso ainda não tenha instalado, o arquivo de instalação está disponível na pasta “Java”. Se preferir fazer o download diretamente do site, acesse:

<[www.java.com/pt\\_BR/download/manual.jsp](http://www.java.com/pt_BR/download/manual.jsp)>

- Recomendamos fechar outros programas abertos a fim de liberar memória e processamento do computador. Se mesmo assim a lentidão persistir, talvez o processador não seja robusto o suficiente para manter uma taxa de frames satisfatória. Uma outra razão para o problema é que, como os programas foram desenvolvidos na plataforma Windows, é provável que em outras plataformas os aplicativos rodem de maneira indesejada.

- O Processing é um programa independente e ainda contém alguns bugs que podem causar problemas na execução dos programas. Se por acaso os programas não rodarem mesmo após seguir estas recomendações, pedimos que visualizem os vídeos para entender como são os programas em execução.

- Os programas possuem pequenas interações com o usuário. Abaixo estão as teclas utilizadas em cada um dos experimentos.

**EXPERIMENTOS 1.0:** Aperte ‘Z’ para repetir o programa.

**EXPERIMENTOS 2.0:** Pressione qualquer **letra** ou **número** no teclado para alterar o elemento gráfico na tela. Letras em caixa alta também funcionam, assim como outros símbolos (exemplo: ?!\$%@).

**EXPERIMENTOS 3.0:** Aperte ‘Z’ para avançar de cena. Aperte ‘X’ para repetir a cena atual.

**TODOS:** Para fechar os programas aperte a tecla “**ESC**” a qualquer momento. Os programas também podem ser finalizados clicando no botão **Stop** no canto inferior esquerdo da tela.

- Os vídeos podem ser acessados online e estão disponíveis no Youtube. Acesse:

<[www.youtube.com/user/slardoc/videos](http://www.youtube.com/user/slardoc/videos)>

- O conteúdo completo do CD também está disponível online. O link está disponível na descrição dos vídeos do YouTube.

<sup>1</sup> Essas instruções também estão disponíveis no CD no arquivo “LEIAME.pdf”.