



UNIVERSIDADE FEDERAL DO PARÁ  
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS  
FACULDADE DE COMPUTAÇÃO  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**Giordanna De Gregoriis**

## **Utilização da técnica de preenchimento de espaço em texturas**

Belém – Pará

2017

Giordanna De Gregoriis

## **Utilização da técnica de preenchimento de espaço em texturas**

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Faculdade de Computação. Universidade Federal do Pará.

Orientadora: Prof<sup>a</sup>. Dr<sup>a</sup>. Cristina Lúcia Dias Vaz  
Coorientador: Prof. Dr. Dionne Cavalcante Monteiro

Belém – Pará  
2017

---

Giordanna De Gregoriis

Utilização da técnica de preenchimento de espaço em texturas/ Giordanna De Gregoriis. – Belém – Pará, 2017-  
60 p. : il. (algumas color.) ; 30 cm.

Orientadora: Profª. Drª. Cristina Lúcia Dias Vaz

Coorientador: Prof. Dr. Dionne Cavalcante Monteiro

Trabalho de Conclusão de Curso – Universidade Federal do Pará – UFPA

Instituto de Ciências Exatas e Naturais – ICEN

Faculdade de Computação – Facomp, 2017.

1. Preenchimento de espaço. 2. Texturas. 3. Função zeta de Riemann. 4. Arte digital. I. Profª. Drª. Cristina Lúcia Dias Vaz. II. Universidade Federal do Pará. III. Faculdade de Computação. IV. Utilização da técnica de preenchimento de espaço em texturas

CDU - - : - - - : - - - . -

---

Giordanna De Gregoriis

## **Utilização da técnica de preenchimento de espaço em texturas**

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Faculdade de Computação. Universidade Federal do Pará.

---

**Prof<sup>a</sup>. Dr<sup>a</sup>. Cristina Lúcia Dias Vaz**  
Orientadora

---

**Prof. Dr. Dionne Cavalcante Monteiro**  
Coorientador

---

**Prof. Dr. Bianchi Serique Meiguins**  
Avaliador

---

**Prof<sup>a</sup>. Dr<sup>a</sup>. Marianne Kogut  
Eliasquevici**  
Avaliadora

Belém – Pará, 17 de Abril de 2017.

**Conceito:** Excelente.

Belém – Pará  
2017

*“Todas as vitórias ocultam uma abdicação”*

*Simone de Beauvoir*

*Dedico este trabalho a minha família, namorada, amigos e principalmente ao meu pai, que enfrentou ao longo de dois anos e meio uma longa batalha contra o câncer.*

# AGRADECIMENTOS

Diversas pessoas me ajudaram ao longo do período em que permaneci na universidade. Muitos novos laços surgiram, foi dado apoio mútuo, e também muitos foram desfeitos. Independentemente disto, cada um deixou em mim uma contribuição que me ajudou a ser a pessoa que sou agora.

Devo admitir que um dos meus maiores medos ao entrar na vida universitária era de permanecer isolada, pois um dos requisitos fundamentais de uma pessoa adulta é saber viver em sociedade, onde existe a questão do trabalho em equipe, compartilhar recursos, saber ser mediador em momentos de dificuldade, entre outras questões. Ao passar dos anos este medo foi superado, pois consegui —do meu jeito estranho e desorientado— me aproximar de pessoas que me ajudaram muito na formação tanto da minha vida acadêmica quanto da minha pessoa. A minha família também foi importante para que eu conseguisse chegar a este ponto, principalmente meus pais, que presenciaram os meus momentos onde estive mais fragilizada e me ajudaram na medida do possível.

Gostaria de agradecer aos meus amigos de curso —com quem eu permaneci mais tempo ao lado por todos esses anos de bacharelado— pela força e apoio que me deram para permanecer avançando nos estudos, e também pelos momentos de lazer, que serviram como uma válvula de escape para os estresses do dia-a-dia. Agradeço especialmente ao Renan Filip e Filipe Damasceno, pelas poucas mas significativas vezes em que fizemos trabalhos em grupo. A vossa dedicação me serviu de inspiração para melhorar meu desempenho ao longo dos últimos semestres. Também gostaria de agradecer ao Caio Pinheiro e Yvan Brito, por terem sido minhas duplas de trabalho diversas vezes e em diversas matérias. É triste pensar que a turma de 2013 só decidiu se reunir com mais frequência no último semestre, visto que é um momento em que muitos têm pouca disponibilidade de tempo. Porém, agradeço de coração todos os momentos em que nós passamos juntos, todos contribuíram para o meu crescimento moral, acadêmico e pessoal.

Aos professores com quem tive aula ao longo desses anos, tenho muito a agradecer pela paciência e dedicação ao ensino. Graças a vós, consegui amadurecer como adulta e a buscar absorver e entender o máximo que pudesse dos assuntos. Agradeço também à professora Cristina Vaz e ao professor Dionne Monteiro, que me receberam de braços abertos ao estágio na Assessoria de Educação a Distância e me orientaram neste trabalho.

Por fim, agradeço a todos os meus amigos do centro espírita Yvon Costa, que por seis anos nos reunimos semanalmente para estudar o Espiritismo, realizar atividades de assistência ao centro e conviver de forma sadia. Mesmo com a vida ocupada, ainda arranjávamos tempo para nos reunirmos e sairmos de vez em quando.

# RESUMO

A textura é um dos principais atributos visuais para a descrição de padrões encontrados na natureza. Diversos métodos de análise de textura têm sido usados como uma poderosa ferramenta para aplicações reais que envolvem análise de imagens e visão computacional. Entretanto, os métodos existentes não conseguem discriminá-los com sucesso a complexidade dos padrões de textura. Tais métodos desconsideram a possibilidade de se descrever estruturas de imagens por meio de medidas como a dimensão fractal. Medidas baseadas em fractalidade permitem uma interpretação geométrica não-inteira que têm aplicações em áreas como Matemática, Física, Biologia, entre outras. Nos últimos anos, a análise fractal surgiu como uma abordagem promissora para captar autossimilaridades em texturas. Com base na análise fractal, foram propostas muitas abordagens bem sucedidas para a classificação de textura. A ideia básica destes métodos é usar dimensão fractal para resumir a distribuição espacial de padrões de imagem. Neste trabalho foi analisado e implementado o método de textura proposto por Shier e Bourke que usa, de modo empírico, ideias e alguns procedimentos da Geometria Fractal. Além da aplicação na geração de textura para computação gráfica em geral, foi investigado o uso do algoritmo em duas situações especiais: preenchimento de espaço e textura artística. Este algoritmo que tem como objetivo preencher uma determinada região utilizando estampas cuja distribuição obedece uma Lei de Potência governada pela função zeta de Hurwitz, deste modo as estampas são inseridas *ad infinitum*. Como resultado final têm-se imagens com texturas autossimilares semelhantes ao fractal de Apolônio (quando usa-se círculos). Como produto final da monografia foi desenvolvida uma ferramenta, chamada *Mosaico Fractal*, capaz de preencher uma região plana com figuras geométricas e gerar texturas artísticas. Uma característica importante deste trabalho são as interações entre Computação, Matemática e Arte, permitindo uma visão interdisciplinar do tema.

**Palavras-chaves:** Preenchimento de espaço, Texturas, Função zeta de Riemann, Arte digital.

# ABSTRACT

Texture is one of the main visual attributes for describing patterns found in nature. Several methods of texture analysis have been used as a powerful tool for real applications involving image analysis and computational vision. However, existing methods can not successfully discriminate the complexity of texture patterns. Such methods disregard the possibility of describing image structures through measures such as the fractal dimension. Measures based on fractality allow a non-integer geometric interpretation that has applications in areas such as Mathematics, Physics, Biology, etc. In recent years, fractal analysis has emerged as a promising approach to capturing self-similarities in textures. Based on the fractal analysis, many successful approaches to texture classification were proposed. The basic idea of these methods is to use fractal dimension to summarize the spatial distribution of image patterns. In this paper the algorithm proposed by Shier and Bourke which uses, in an empirical way, ideas and some procedures of Fractal Geometry, was analyzed and implemented. Besides the application in the generation of textures for computer graphics in general, it has been investigated the use of the algorithm in two special situations: space filling and artistic texture. This algorithm that aims to fill a specific region using shapes whose distribution obeys a Power Law ruled by the Hurwitz zeta function, in this way these shapes are inserted *ad infinitum*. As a final result images with self-similar textures that resembles the Apollonian fractal (when using circles) are displayed. As a final product of the monograph it was developed a tool, called *Mosaico Fractal*, which is able to fill a two-dimensional region with geometric figures and generate artistic textures. An important feature of this paper is the interactions between Computer Science, Mathematics and Art, allowing an interdisciplinary view of the subject.

**Key-words:** Space filling, Textures, Riemann zeta function, Digital art.

# LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplos de texturas bidimensionais.	15
Figura 2 – Exemplos de texturas tridimensionais.	15
Figura 3 – 5.000 círculos sem interseções.	23
Figura 4 – (a): $A_i$ decresce muito rápido. (b): $A_i$ decresce muito lentamente. (c): $A_i$ decresce adequadamente para preencher a região.	24
Figura 5 – O efeito de diferentes valores de $c$ e $N$ na $i$ -ésima iteração.	25
Figura 6 – Probabilidade de parada para preenchimento de uma região retangular com círculos para limites inclusivos e periódicos.	28
Figura 7 – Região não retangular, preenchida com quadrados. Limites inclusivos.	30
Figura 8 – Fractal de Apolônio.	31
Figura 9 – Geração do fractal de Apolônio.	32
Figura 10 – Variações do fractal de Apolônio.	32
Figura 11 – Diagrama de atividades do fluxo principal.	37
Figura 12 – Diagrama de casos de uso.	38
Figura 13 – Diagrama de classes.	39
Figura 14 – Diagrama de pacotes.	40
Figura 15 – Três tipos de módulos da arquitetura da <i>toolkit</i> .	42
Figura 16 – Ícones criados para o programa.	45
Figura 17 – Algumas estampas criadas para o programa.	45
Figura 18 – Tela inicial do programa <i>Mosaico Fractal</i> com algumas opções selecionadas.	46
Figura 19 – Exemplos de execuções do programa utilizando formas simples.	48
Figura 20 – Utilização para geração de imagem com continuidade ( <i>seamless</i> ).	48
Figura 21 – Exemplos de execuções do programa utilizando rotação de estampas.	49
Figura 22 – Exemplos de execuções do programa utilizando formas irregulares e múltiplas estampas.	50
Figura 23 – Exemplo de execuções do programa utilizando uma forma complexa, com limites periódicos.	50
Figura 24 – Exemplo de execuções do programa utilizando múltiplas formas.	51
Figura 25 – Exemplos de execuções do programa utilizando formas de anéis anulares.	52
Figura 26 – Exemplo de execução do programa utilizando uma forma extremamente irregular para a região de preenchimento e estampas de ângulos agudos.	53
Figura 27 – Fractal de Apolônio gerado com uma versão anterior do <i>Mosaico fractal</i> .	58

# LISTA DE QUADROS

Quadro 1 – Requisitos funcionais da ferramenta. . . . .	34
Quadro 2 – Requisitos não-funcionais da ferramenta. . . . .	34
Quadro 3 – Principais casos de uso da ferramenta. . . . .	35
Quadro 4 – Requisitos atendidos. . . . .	54

# LISTA DE ABREVIATURAS E SIGLAS

PNG	<i>Portable Network Graphics</i> , em português “gráficos de rede portáteis”, formato de imagem que utiliza compressão sem perdas de dados ( <i>lossless</i> )
XML	<i>eXtensible Markup Language</i> , em português “linguagem de marcação extensiva”, recomendação da W3C ( <i>World Wide Web Consortium</i> ) para gerar linguagens de marcação específicas
SVG	<i>Scalable Vector Graphics</i> , em português “gráficos vetoriais escaláveis”, linguagem XML para descrever gráficos bidimensionais de forma vetorial através de fórmulas matemáticas
IDE	<i>Integrated Development Environment</i> , em português “ambiente de desenvolvimento integrado”, programa que reúne ferramentas de apoio ao desenvolvimento de <i>software</i>
UML	<i>Unified Modeling Language</i> , em português “linguagem de modelagem unificada”, linguagem de modelagem que permite representar um sistema de forma padronizada
GVT	<i>Graphic Vector Toolkit</i> , em português “conjunto de ferramentas para gráficos vetorizados”, módulo de nível mais baixo da biblioteca Batik
DOM	<i>Document Object Model</i> , em português “modelo de objeto de documento”, representação na memória de um documento XML
API	<i>Application Programming Interface</i> , em português “interface de programação de aplicações”, conjunto de padrões estabelecidos por um <i>software</i> para a utilização de suas funções por outros aplicativos

# LISTA DE SÍMBOLOS

$\mathbb{C}$	Conjunto dos números complexos
$\mathbb{N}$	Conjunto dos números naturais
$\in$	Pertença a conjunto
$\mapsto$	Notação utilizada para denotar uma função sem nome
$\Sigma$	Letra grega maiúscula sigma, representando um somatório
$\infty$	Infinito
$\zeta$	Letra grega minúscula zeta, representando uma função zeta

# SUMÁRIO

<b>1</b>	<b>Introdução . . . . .</b>	<b>14</b>
1.1	Considerações iniciais . . . . .	14
1.2	Motivação, materiais e métodos . . . . .	16
1.3	Organização do trabalho . . . . .	18
<b>2</b>	<b>Algoritmo de preenchimento de espaço . . . . .</b>	<b>19</b>
2.1	Fundamentação . . . . .	19
2.1.1	Sequências e séries em $\mathbb{C}$ . . . . .	19
2.1.2	A função zeta de Riemann . . . . .	20
2.1.3	Elementos da teoria de probabilidade . . . . .	21
2.2	Principal ideia do método . . . . .	22
2.3	Regras de construção . . . . .	23
2.4	Algoritmo . . . . .	26
2.5	Critério de parada . . . . .	27
2.6	Formas irregulares . . . . .	29
<b>3</b>	<b>Modelagem da ferramenta . . . . .</b>	<b>31</b>
3.1	Ferramenta <i>Mosaico Fractal</i> . . . . .	31
3.2	Requisitos . . . . .	33
3.3	Diagramas . . . . .	36
<b>4</b>	<b>Ferramenta desenvolvida . . . . .</b>	<b>41</b>
4.1	Programação . . . . .	41
4.1.1	Batik . . . . .	42
4.2	Recursos utilizados . . . . .	44
4.3	Resultados . . . . .	46
4.4	Requisitos atendidos . . . . .	53
<b>5</b>	<b>Conclusões . . . . .</b>	<b>55</b>
5.1	Pontos positivos . . . . .	55
5.2	Dificuldades encontradas . . . . .	56
5.3	Trabalhos futuros . . . . .	57
<b>Referências . . . . .</b>		<b>59</b>

# 1 INTRODUÇÃO

Este capítulo tem por objetivo apresentar uma visão geral do trabalho. Será feito um breve resumo das principais ideias, dos objetivos, dos métodos e técnicas usados. Além disso, será descrito como o trabalho foi organizado.

## 1.1 Considerações iniciais

A textura é um dos principais atributos visuais para a descrição de padrões encontrados na natureza. Diversos métodos de análise de textura têm sido usados como uma poderosa ferramenta para aplicações reais que envolvem análise de imagens e visão computacional. Entretanto, os métodos existentes não conseguem discriminar com sucesso a complexidade dos padrões de textura. Tais métodos desconsideram a possibilidade de se descrever estruturas de imagens por meio de medidas como a dimensão fractal. Medidas baseadas em fractalidade permitem uma interpretação geométrica não-inteira que têm aplicações em áreas como Matemática, Física, Biologia, entre outras.

A Geometria Fractal é um ramo da Geometria que estuda objetos que podem ser obtidos de forma recursiva e que apresentam determinadas características especiais tais como autossimilaridade, estrutura fina em qualquer escala, dimensão fracionada, entre outras. Tais objetos são chamados fractais e foram popularizada por Mandelbrot (1977) que inspirou outros pesquisadores a encontrarem tais padrões em diversas áreas. Mandelbrot investigou padrões que aparecem com frequência na natureza e os modelou usando objetos da geometria fractal. Para ele as nuvens não eram esferas, as montanhas não eram cones, e os relâmpagos não percorrem uma linha reta. Seu entendimento da complexidade da natureza não era apenas algo aleatório, mas exigia a convicção de que o interessante na trajetória do raio, por exemplo, não era sua direção, mas sim a distribuição dos seus ziguezagues. Em particular, a dimensão fractal tornou-se um modo de medir o grau de aspereza, ou de fragmentação, ou de irregularidade de um objeto e pode ser usada para medir um certo grau de textura.

Nos últimos anos, a análise fractal surgiu como uma abordagem promissora para captar as autossemelhanças em texturas. Com base na análise fractal, foram propostas muitas abordagens bem sucedidas (QUAN YONG XU; LUO, 2014) para a classificação de textura. A ideia básica destes métodos é usar dimensão fractal para resumir a distribuição espacial de padrões de imagem.

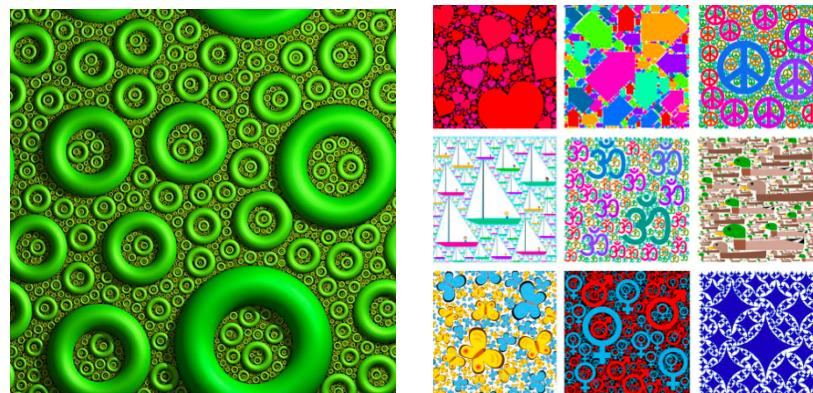
Neste trabalho foi analisado e implementado o método de textura proposto por Shier e Bourke (2013) que usa, de modo empírico, ideias e alguns procedimentos da Geo-

metria Fractal. O método proposto pelos autores é baseado na intuição e em experimentos e simulações computacionais, de modo que os autores não apresentam provas formais e teóricas sobre o método.

Além da aplicação na geração de textura para computação gráfica em geral, os autores investigam o uso do algoritmo em duas situações especiais: preenchimento de espaço e textura artística.

A Figura 1 ilustra uma região quadrada preenchida com toros e exemplos de texturas artísticas.

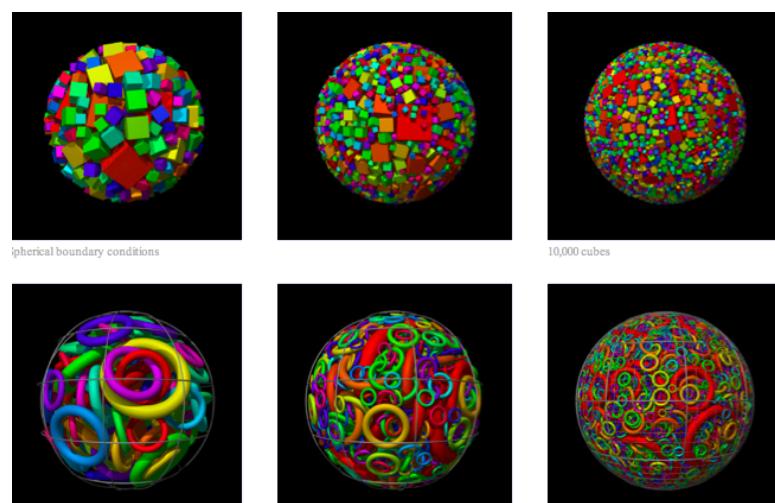
Figura 1 – Exemplos de texturas bidimensionais.



Fonte – Shier e Bourke (2013), Dunham e Shier (2014)

Na Figura 2 tem-se regiões espaciais preenchidas com as figuras geométricas cubos e toros.

Figura 2 – Exemplos de texturas tridimensionais.



Fonte – Shier e Bourke (2013)

## 1.2 Motivação, materiais e métodos

A principal motivação para realização deste trabalho foi a oportunidade de produzir uma ferramenta para a geração de padrões com a capacidade de edição das variáveis do algoritmo que permitam a criação de novos padrões, além de um ferramenta de acesso livre e código aberto. Portanto, o objetivo principal do trabalho é estudar e implementar o algoritmo proposto por Bourke (2013) e ter como produto final uma ferramenta capaz de preencher um espaço plano com figuras geométricas primitivas e gerar texturas artísticas.

Dentre os materiais utilizados para o desenvolvimento da ferramenta descrita no trabalho, tem-se:

- **O sistema de controle de versão Git:** é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte, que permitiu analisar cada etapa superada pelo projeto, e realizar operações para desfazer modificações que não obtiveram sucesso no trabalho, assim como bifurcar o projeto para o desenvolvimento de cada nova etapa com cautela, para depois juntar e/ou sobrepor ao projeto principal;
- **O serviço de hospedagem de projetos controlados Bitbucket:** é um serviço de hospedagem de projetos controlados que oferece uma versão gratuita para hospedar o projeto desenvolvido em um repositório. Este serviço foi utilizado pois oferece suporte a utilização do sistema de controle de versões Git, além de permitir que o repositório utilizado seja privado;
- **O ambiente de desenvolvimento NetBeans:** é uma IDE gratuita e de código aberto, que permite desenvolver grandes projetos em Java, JavaScript, HTML5, PHP, C/C++, entre outros. Também possui suporte ao controle de versões Git, o qual foi de suma importância para o desenvolvimento da ferramenta. O seu módulo para desenvolvimento de interface de usuário *Swing* foi muito importante para a construção da identidade da ferramenta;
- **A ferramenta de modelagem Astah:** é uma ferramenta de modelagem UML, a qual foi utilizada a versão Astah Professional disponibilizada para estudantes porém possui a versão gratuita Astah Community. Permite adicionar métodos no diagrama de sequência e a alteração se refletir no diagrama de classes. Foi utilizada para a geração de diagramas para desenvolver o projeto;
- **A linguagem de programação Java:** esta linguagem foi escolhida por estar de acordo com as habilidades da desenvolvedora, e também por ser orientada a objetos, sendo própria para o uso de esquemas de engenharia de *software*;

- **O paradigma de programação orientada a objetos:** este paradigma, fortemente implicado na linguagem Java, permite o encapsulamento de métodos e classes, herança de classes, sobrecarga de métodos, e outras características que devem ser bem empregadas quando se utiliza engenharia de *software* para desenvolver a modelagem da ferramenta.
- **O editor de imagens vetoriais Adobe Illustrator:** é uma ferramenta de geração e edição de imagens vetoriais de cunho comercial, utilizada para gerar estampas que foram necessárias para testar o funcionamento da ferramenta.
- **O conjunto de ferramentas Batik:** este conjunto de ferramentas baseado em Java serve para permitir a utilização imagens no formato SVG na ferramenta desenvolvida, assim permitindo que usuários utilizem suas próprias imagens vetorizadas no programa.

Para implementação do algoritmo proposto pelos autores realizou-se uma programação capaz de receber informações do usuário através de várias opções, entre elas destacam-se as seguintes:

- Inserir e utilizar formas personalizadas criadas pelo usuário, para serem usadas tanto como estampa quanto para definir uma região;
- Definir se as estampas serão posicionadas com rotações variadas, definidas aleatoriamente, ou se serão posicionadas sem nenhuma mudança de ângulo. A rotação será definida em graus, variando de 0° a 360°;
- Escolher determinadas cores ou texturas que serão utilizadas para o preenchimento das estampas;
- Definir se irá utilizar as bordas da imagem como limites para o posicionamento das estampas, ou se irá utilizar as bordas para replicar as estampas de forma que aja uma sensação de continuidade, tal propriedade encontrada similarmente em azulejos.

Este trabalho é caracterizado pela sua interdisciplinaridade, ou seja, há uma convergência de duas ou mais áreas de conhecimento. No caso, as áreas que houveram uma intersecção foram Computação, Matemática e Arte. Esta característica permitiu uma transferência de métodos de uma área (no caso, Matemática) para outra (Computação), sendo possível então gerar como resultado algo novo e inovador para uma outra área (Arte). Isto permite uma expansão das fronteiras do conhecimento, tanto para quem deseja contribuir para o desenvolvimento de mais funcionalidades para a ferramenta quanto para os usuários. Espera-se que estes resultados sirvam de incentivo em investigações futuras de como a interdisciplinaridade pode ajudar na criatividade, inovação e qualidade

para as pessoas de áreas de conhecimento diferentes, despertando habilidades diferentes, e removendo limitações que comumente ocorrem quando alguém se restringe à uma única área.

O *software* descrito neste trabalho permite gerar imagens artísticas com uma mistura de geometria e aleatoriedade. Isto permite então que artistas possam desenvolver suas próprias figuras artísticas com seus próprios esquemas de formas e valores. Por esta ferramenta gerar resultados lúdicos, pode-se considerar o seu uso também como recurso para realização de atividades de ensino-aprendizagem com crianças para estímulo da criatividade.

### 1.3 Organização do trabalho

Nesta monografia foi implementado o algoritmo para geração de texturas e preenchimento de espaço proposto por Bourke (2013) e foi produzida uma ferramenta capaz de preencher um espaço plano com figuras geométricas primitivas e gerar texturas artísticas. No Capítulo 2 descreve-se detalhadamente o algoritmo apresentando as principais ideias e procedimentos e a fundamentação matemática necessária.

No Capítulo 3 apresentare-se os processos e técnicas usados na programação. Para isto, descreve-se a metodologia de desenvolvimento utilizada, diagramas gerados para uma melhor organização e outras informações relevantes. No Capítulo 4 apresenta-se a ferramenta produzida, as tecnologias utilizadas, suas funcionalidades, aparência e execução, além de alguns experimentos computacionais. Por fim, no Capítulo 5 é feita algumas considerações destacando-se pontos positivos e as dificuldades encontradas e apontando direções para trabalhos futuros.

## 2 ALGORITMO DE PREENCHIMENTO DE ESPAÇO

Este capítulo tem como objetivo apresentar, de forma clara, as principais ideias e o funcionamento do algoritmo descrito no artigo de Bourke (2013), cujo título é “*A space filling algorithm for generating procedural geometry and texture*”, publicado no periódico *GSTF Journal on Computing* em 2013 pelo fórum *Global Science and Technology Forum*. Inicia-se com um breve resumo sobre convergência de série numérica no conjunto dos números complexos e alguns conceitos da teoria de probabilidade, para auxiliar no entendimento da utilização da função zeta de Hurwitz no algoritmo.

### 2.1 Fundamentação

Nesta seção é apresentado um breve resumo sobre sequências e séries no conjunto dos números complexos e alguns conceitos da teoria de probabilidade. Para mais detalhes consulte (CHURCHILL, 1980), (DANTAS, 2008), (PAULINO, 2012).

#### 2.1.1 Sequências e séries em $\mathbb{C}$

Uma sequência de números complexos é uma função cujo domínio é o conjunto dos números naturais  $\mathbb{N}$  e o contra-domínio é o conjunto dos números complexos  $\mathbb{C}$ ,  $z : \mathbb{C} \mapsto \mathbb{N}$ . O  $n$ -ésimo termo da sequência é denotado  $z(n)$ , alternativamente por  $\{z_n, n \in \mathbb{N}\}$  ou por  $\{z_n\}$ .

Uma sequência  $\{z_n\}$  converge se ela possuir um limite, assim deve existir  $z \in \mathbb{C}$  tal que  $z = \lim_{n \rightarrow \infty} z_n$ . Caso contrário, considera-se que a sequência diverge. Também é usada a notação  $z_n \rightarrow z$  para representar a convergência.

Geometricamente, o fato de  $z_n \rightarrow z$  significa que por menor que seja o disco centrado em  $z_o$ , sempre será possível encontrar  $n_o \in \mathbb{N}$  de modo que  $z_n$  pertença a este disco para todo  $n \geq n_o$ . Em geral, quanto menor o disco, maior será  $n_o$ .

Seja  $\{z_n\}$  uma sequência em  $\mathbb{C}$ . Considera-se que a série  $\sum_{n=0}^{\infty} z_n$  converge se a sequência das somas parciais

$$s_n = \sum_{k=0}^n z_k = z_0 + \dots + z_n$$

for convergente e usa-se a notação  $\sum_{n=0}^{\infty} z_n$ . Ou seja,

$$\sum_{n=0}^{\infty} z_n = \lim_{n \rightarrow \infty} s_n.$$

**Exemplo 2.1**  $\sum_{n=0}^{\infty} \frac{i}{n}$  não converge, pois  $\sum_{n=0}^{\infty} \operatorname{Im}\left(\frac{i}{n}\right) = \sum_{n=0}^{\infty} \frac{1}{n}$  diverge (série harmônica).

### 2.1.2 A função zeta de Riemann

A função zeta foi investigada por L. Euler no estudo da distribuição dos números primos. Euler deu uma prova analítica do teorema de Euclides sobre a existência de infinitos números primos baseada numa de suas propriedades. Generalizando os métodos de Euler, Dirichlet provou, em 1837, seu famoso teorema sobre primos em progressão aritmética. Porém, todos estes trabalhos consideravam as funções no domínio dos números reais. Foi Riemann quem enxergou profundamente a relação entre a função zeta e a distribuição dos números primos ao estudá-la como função de uma variável complexa. Suas ideias foram tão brilhantes que a partir de então a função zeta ganhou o seu nome (OLIVEIRA, 2013).

A função zeta de Riemann é definida para um argumento complexo  $s$  no semiplano  $\operatorname{Re}(s) > 1$  como:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} \quad (2.1)$$

A série (2.1) é uma série importante na Teoria dos números e possui aplicações na Física, na Probabilidade e na Estatística Aplicada. É uma série uniformemente convergente em qualquer disco fechado no semiplano  $\operatorname{Re}(s) > 1$ .

Existem várias funções do tipo zeta, neste trabalho tem-se um interesse na função zeta de Hurwitz, definida para um argumento complexo  $c$  e um argumento real  $N$  como:

$$\zeta(c, N) = \sum_{k=0}^{\infty} \frac{1}{(k + N)^c} \quad (2.2)$$

A série (2.2) é absolutamente convergente para  $N > 0$  e  $\operatorname{Re}(c) > 1$  (WEISSSTEIN, 2002).

### 2.1.3 Elementos da teoria de probabilidade

A teoria da probabilidade é conhecida há bastante tempo e a grande maioria dos conceitos que são conhecidos hoje tiveram grande contribuição de pesquisadores como Bernoulli, De Moivre, Laplace, Gauss e Poisson (DANTAS, 2008).

Seja  $X$  uma variável discreta, se esta variável for medida muitas vezes durante repetidas observações, pode-se determinar um valor específico de  $X$ , ( $X_i$ ), cuja tendência é de que ao longo do tempo, conforme se aumenta o número de observações também aumente o número de vezes em que a variável assuma o valor  $X_i$ . Depois de um número suficientemente grande de medidas, a frequência relativa com que  $X_i$  ocorre vai se estabilizando e tendendo a um valor bem determinado, ao qual chama-se de probabilidade da ocorrência de  $X_i$ . Indica-se essa probabilidade por  $P(X = X_i)$ , ou de maneira simplificada  $P(X_i)$ . Se considerar o conjunto formado pelos diversos valores  $X_i$  que a variável  $X$  pode assumir, e suas respectivas probabilidades, têm-se uma distribuição de probabilidades (DANTAS, 2008).

Uma distribuição de probabilidade é definida como um modelo matemático que relaciona o valor da variável com a ocorrência daquele valor na população. A média de uma distribuição de probabilidade é uma medida da tendência central da distribuição. Na literatura, existem vários tipos de distribuições: normal, exponencial, lognormal, as Leis de Potência, entre outras. Neste trabalho tem-se um interesse na distribuição de Lei de Potência.

A maioria das variáveis inerentes aos mais diversos processos apresentam um comportamento de agrupamento em torno de um dado valor central. Embora este seja um comportamento comum para a maioria dos casos, existem algumas variáveis que não possuem esse comportamento, apresentando grande variabilidade, às vezes de muitas ordens de magnitude, como o tamanho das cidades (PAULINO, 2012). Dentre as distribuições que não seguem este comportamento central, as Leis de Potência tem se destacado devido a algumas de suas propriedades matemáticas que podem conduzir a diversos resultados práticos.

Muitas análises empíricas sugerem que o comportamento das Leis de Potência na distribuição de valores é muito frequente na natureza. Esse comportamento pode ser encontrado na distribuição dos tamanhos dos incêndios florestais, da intensidade de terremotos, do número de mortos em ataques terroristas até da riqueza de pessoas e nações (PAULINO, 2012).

Se  $k$  for uma variável inteira, um modo alternativo de declarar que ela segue uma Lei de Potência é se a probabilidade  $p_k$  dos valores de  $k$  obedece:

$$p_k = C k^{-\alpha},$$

para alguma constante  $\alpha$ . Para  $k \neq 0$ , a constante  $C$  é então definida pela condição de

normalização:

$$1 = \sum_{k=1}^{\infty} p_k = C \sum_{k=1}^{\infty} k^{-\alpha} = C\zeta(\alpha), \quad (2.3)$$

com  $\zeta$  a função zeta de Riemann.

Rearranjando a equação (2.3) e identificando o comportamento da Lei de Potência na cauda da distribuição para valores de  $k \geq k_{min}$ , a expressão equivalente é (PAULINO, 2012):

$$p_k = \frac{k^{-\alpha}}{\zeta(\alpha, k_{min})}, \quad (2.4)$$

com  $\zeta(\alpha, k_{min}) = \sum_{k=k_{min}}^{\infty} k^{-\alpha}$  uma generalização de  $\zeta$ .

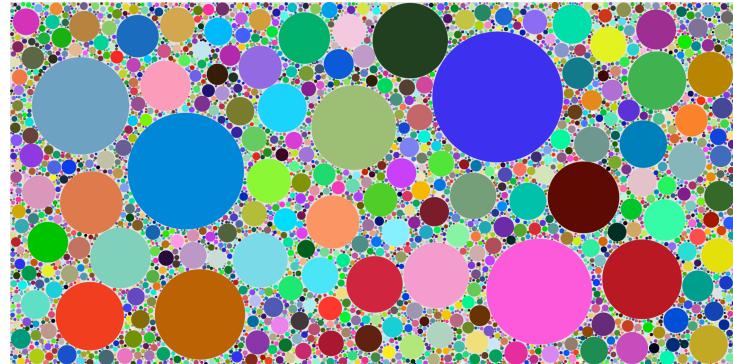
No trabalho de Bourke (2013), o autor escolhe a função zeta de Hurwitz para distribuir as áreas das figuras geométricas num espaço bidimensional com a finalidade de gerar estampas e texturas. Apesar de não justificar os motivos de tal escolha, acredita-se que se trata de uma escolha motivada pelas propriedades da distribuição de leis de potência, que, como foi mencionado acima, tem um comportamento muito similar aos padrões frequentes na natureza.

## 2.2 Principal ideia do método

A ideia principal apresentada no artigo de Bourke (2013) é gerar texturas ou figuras geométricas “preenchendo o espaço” (ou *space filling*) com figuras geométricas primárias. De um modo bem geral, as figuras são inseridas aleatória e iterativamente em posições vazias (não ocupadas) de tal modo que não acorram interseções.

Deste modo, deseja-se preencher aleatoriamente uma região previamente definida com um número “infinito” de figuras geométricas primárias (triângulos, quadrados, círculos) que diminuem progressivamente. Um exemplo gerado por este algoritmo é mostrado na Figura 3:

Figura 3 – 5.000 círculos sem interseções.



Fonte – Shier (2011)

## 2.3 Regras de construção

Considere uma região de bidimensional  $R$  com área  $A_0$ , deseja-se preenchê-la com formas geométricas primárias similares  $F_i$  cujas área são  $A_i$ . Deseja-se calcular as áreas  $A_i$  de tal modo que quando  $i$  cresce,  $A_i$  decresce. Observe que a sequência das áreas é dada por:

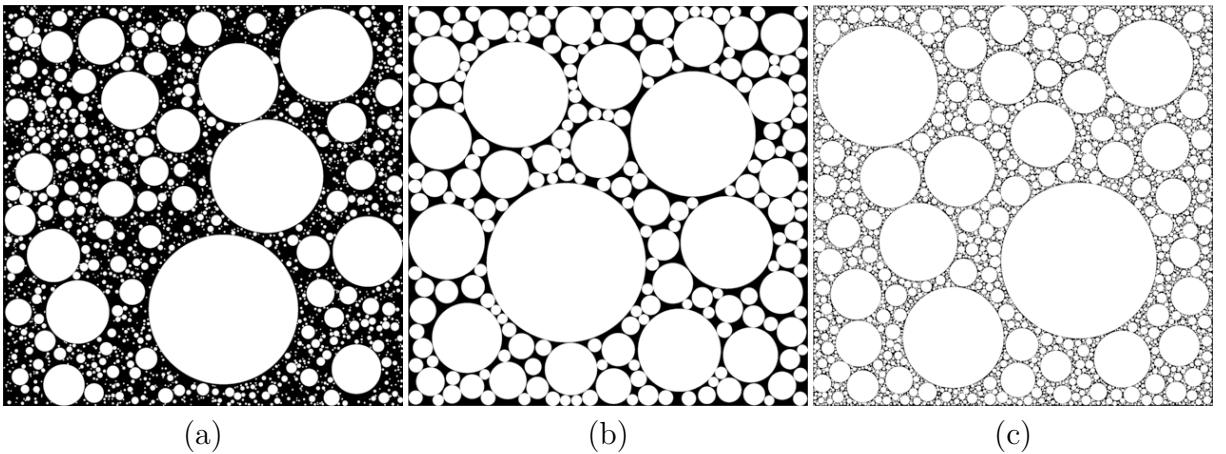
$$A_0, A_0 A_1, A_0 A_2, \dots, A_0 A_n, \dots$$

Logo, a área total , representada por  $A$ , é dada pela série

$$A = A_0 \sum_{n=1}^{\infty} A_n. \quad (2.5)$$

Para garantir o preenchimento da a região  $R$  deve-se garantir a convergência da série (2.5). Além disso, precisa-se controlar a “rapidez” desta convergência, pois se  $A_i$  decresce muito rapidamente, a região não será preenchida, por outro lado, se não decresce com uma rapidez adequada o processo falhará, pois não existirá espaço suficiente para próxima figura geométrica. Estas ideias são ilustradas na Figura 4:

Figura 4 – (a):  $A_i$  decresce muito rápido. (b):  $A_i$  decresce muito lentamente. (c):  $A_i$  decresce adequadamente para preencher a região.



Fonte – Bourke e Shier (2013)

Portanto, deve-se escolher uma distribuição dos valores  $A_i$  cujo comportamento simule as ideias descritas acima. A escolha de uma distribuição adequada é importante, pois, como mencionado acima, a rapidez com que os valores  $A_i$  decrescem influenciará no processo de preenchimento da região. Porém, destaca-se que, pode-se usar os valores pequenos de  $A_i$  para simular distribuições que ocorrem na natureza, tais como bolhas de sabão ou vitórias-rélias em um lago.

Como foi descrito na Seção 2.1.3, a distribuição mais adequada é a distribuição de Leis de Potência, em particular, a lei de potência dada pela função zeta de Riemann. Deste modo, (2.5) torna-se

$$A = A_0 \sum_{i=1}^{\infty} \frac{1}{i^c} = A_0 \zeta(c). \quad (2.6)$$

Sabe-se que a série (2.6) converge para  $c > 1$ .

Para preencher a região  $R$  com área  $A_0$  deve-se escolher um valor para o parâmetro  $c$ . Os valores indicados pelo autor foram  $1 < c < 2$ , pois neste intervalo é otimizada a redução progressiva das áreas das figuras geométricas, que será lenta o suficiente para que haja espaço vazio (não ocupado), de modo que outra figura possa ser inserida. A disponibilidade de espaço deve ficar cada vez mais reduzida quanto maior for o parâmetro  $c$ .

Para melhorar o desempenho do processo (veja Seção 2.1.3), usa-se uma variante da função zeta de Riemann: a função zeta de Hurwitz,

$$\zeta(c, N) = \sum_{i=0}^{\infty} \frac{1}{(i + N)^c} \quad (2.7)$$

que converge para  $c > 1$  e  $N > 0$ .

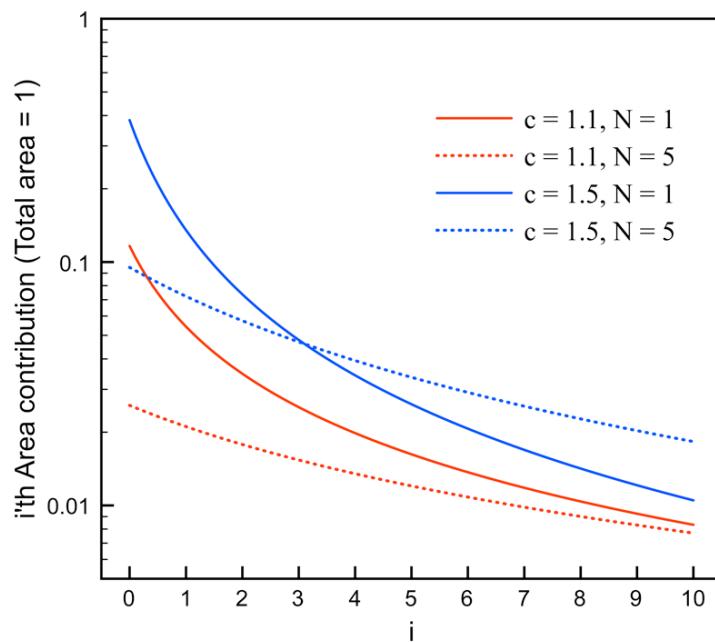
Logo, (2.5) torna-se

$$A = \sum_{i=0}^{\infty} A_i = \sum_{i=0}^{\infty} \frac{A_0}{\zeta(c, N)(i + N)^c}. \quad (2.8)$$

Neste caso, para preencher a região precisa-se escolher os parâmetros  $c$  e  $N$ . Existem várias opções de escolha destes parâmetros. O parâmetro  $N$  não precisa ser necessariamente um número inteiro, mas os valores inteiros são utilizados nos exemplos e na discussão apresentada. A influência geral dos parâmetros  $c$  e  $N$  pode ser ilustrada na Figura 5.

Observe que, conforme  $c$  aumenta, as figuras iniciais têm uma área maior, mas à medida que  $i$  aumenta a área das figuras decresce mais rapidamente. Similarmente, conforme  $N$  diminui, as figuras iniciais têm maior área, mas decrescem mais rapidamente à medida que  $i$  aumenta.

Figura 5 – O efeito de diferentes valores de  $c$  e  $N$  na  $i$ -ésima iteração.



Fonte – Bourke (2013)

Em resumo, inicia-se posicionando aleatoriamente a figura geométrica primária  $F_1$  em algum local dentro da região  $R$ , respeitando seus limites. Em seguida, busca-se posições aleatórias do eixo  $x$  e  $y$  da região  $R$  para inserir a próxima figura  $F_2$ , a qual possui uma área menor que a de  $F_1$ , em seguida verifica-se se  $F_2$  intercepta  $F_1$ . Se não há interseção então considera-se que a inserção foi feita com sucesso.

Após realizar o mesmo processo inicial de encontrar uma posição aleatória para uma figura e reduzir seu tamanho segundo a regra de distribuição adotada, insere-se a nova figura  $F_3$ , em seguida verifica-se se  $F_3$  intersecta as figuras já posicionadas na região, ou seja, se intercepta  $F_1$  e  $F_2$ . Se não há interseção então considera-se que a inserção foi feita com sucesso. Assim sucessivamente, para inserir a figura  $F_i$ , verifica-se se  $F_i$  intersecta com as figuras  $F_1, F_2, \dots, F_{i-1}$  já inseridas. Se não há interseção então considera-se que a inserção foi feita com sucesso.

## 2.4 Algoritmo

Inicialmente, deve-se ressaltar que é desejado preencher uma região de tal modo que não tenha-se figuras ou pedaços de figuras fora da região  $R$ . Na implementação do algoritmo é mostrada que esta exigência pode ser ignorada. Para isto, ao inserir figuras ou pedaços de figuras fora da região  $R$  é possível posicionar as figuras ou pedaços de figuras excedentes em lados opostos da região, o que dará uma visualização (falsa) de continuidade (*seamless*).

No que segue, descreve-se os passos do algoritmo:

**Passo 1:** Considere  $i = 1$ . Insira uma figura com área  $A_1$  em uma posição aleatória dentro da região  $R$ , com área  $A_0$ , a ser preenchida, de modo que não se sobreponha aos limites (bordas) da região. Incremente  $i$ . Isso é referido como o posicionamento inicial;

**Passo 2:** Escolha uma posição aleatória para uma nova figura com área  $A_i$ , novamente, deve estar inteiramente dentro do limite da região a ser preenchida. Esta é uma tentativa. Teste se a forma  $A_i$  intersecta qualquer figura previamente inserida;

**Passo 3:** Se a figura a ser inserida intercepta alguma já inserida, repita o passo 2. Caso contrário, registre a posição e as dimensões da figura na base de dados (das figuras inseridas), incremente  $i$  e repita o passo 2. Pare quando um número desejado  $n$  de figuras inseridas for atingido ou quando um preenchimento percentual da área escolhido for atingido.

O resultado é uma coleção de figuras geométricas primárias posicionadas aleatoriamente, preenchendo a região  $R$ , cujas áreas seguem uma distribuição de leis de potência descrevendo padrões autossimilares que simulam as características fractais.

Observe que o algoritmo é iterativo somente no passo 3, o que simplifica bastante o processo. Para variar a distribuição das figuras e, logo, o padrão final, deve-se variar os parâmetros  $c$  e  $N$ . Além disso, o algoritmo independe das figuras geométricas escolhidas.

A implementação deste algoritmo de preenchimento de espaço requer três coisas:

- a) Uma relação entre as dimensões lineares determinadas pela lei de potência (escala) e a área da figura.
- b) Um teste de interseção entre duas figuras, ou seja, “A figura1 (na posição, escala e orientação atual) intercepta a figura 2 (em sua posição, escala e orientação existentes)?”
- c) Um teste de interseção de uma figura em uma determinada posição, escala e orientação com o limite da região a ser preenchida.

Acredita-se que o algoritmo pode ser utilizado com qualquer figura ou combinação de múltiplas figuras, desde que a série das áreas seja dada por (2.6).

Por outro lado, nossas simulações indicam que quanto maior for o valor de  $c$ , maior será a área das primeiras figuras e mais rapidamente as áreas diminuirão, conforme mais figuras são inseridas. Existe um valor máximo de  $c$  para cada figura e para a região a ser preenchida. Por esta razão, a quantidade de experimentos necessários para a inserção das figuras depende muito fortemente de  $c$ . O parâmetro  $N$  pode ser usado para ajustar o tamanho da maior figura. Quando  $N$  é grande, as primeiras figuras são menores e suas áreas caem mais lentamente com o número de inserções. Com relação a um valor mínimo para o parâmetro  $c$ , têm-se que a única exigência é que  $c$  seja maior que 1.

Geralmente para simulações com figuras primárias, tais como círculos ou quadrados, o valor máximo para  $c$  é maior do que seu valor máximo para figuras mais irregulares. Das figuras geométricas bidimensionais tratadas por Bourke (2013), os quadrados têm o maior valor máximo de  $c$ . Os quadrados requerem menos testes por inserção do que qualquer outra figura bidimensional investigada. O maior valor de  $c$  foi aproximadamente 1,57 para quadrados e 1,2 para cubos.

A inserção de uma determinada figura é aleatória, mas esta é uma aleatoriedade relativa, pois é influenciada por todas as inserções anteriores. Em qualquer passo do algoritmo, a inserção depende de toda a história anterior do processo.

Apesar de tratar do caso bidimensional, o algoritmo pode ser estendido para o caso tridimensional. Este caso não será investigado neste trabalho.

## 2.5 Critério de parada

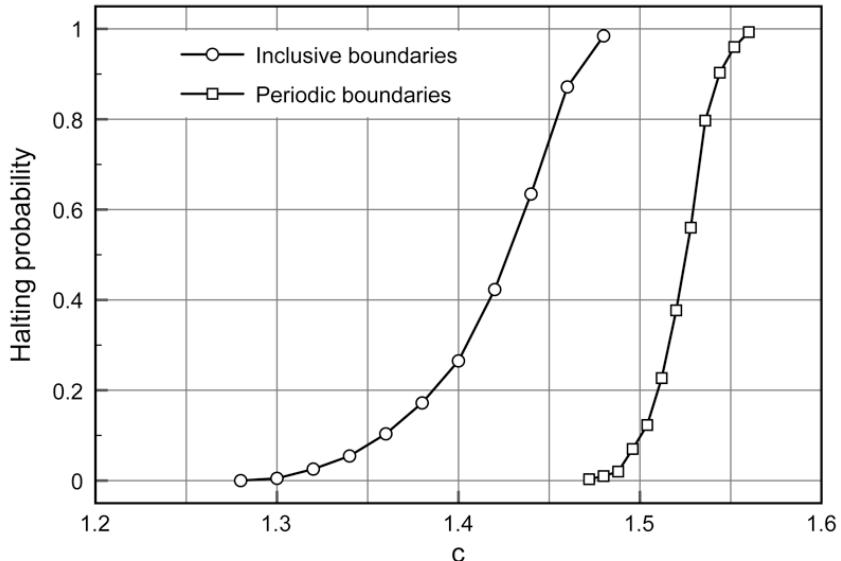
Considera-se como critério de parada do algoritmo o estágio que não existe espaço suficiente para inserir a próxima figura, de acordo com os procedimentos estabelecidos. Baseado em um grande número de experimentos computacionais, Bourke (2013) afirma que, para  $1 < c < c_{max}$ , o algoritmo funciona sem critério de parada.

Para valores grande de  $c$  (maiores de um valor crítico), o algoritmo para automaticamente. Uma das razões para este comportamento é a geometria das primeiras figuras. Suponha, por exemplo, que deseja-se preencher uma região retangular com quadrados. Neste caso, o procedimento ótimo para inserir os dois primeiros quadrados é posicionar o primeiro quadrado num canto da região e o segundo quadrado no canto diagonalmente oposto. Deste modo, otimiza-se o espaço para inserção dos quadrados subsequentes.

Se os lados dos primeiros quadrados  $Q_1$  e  $Q_2$  são  $l_1$  e  $l_2$ , respectivamente, é evidente que se  $l_1 + l_2$  exceder o lado da região retangular não será possível inserir nem  $Q_1$  ou  $Q_2$  por busca aleatória e o algoritmo vai parar.

A Figura 6 mostra as probabilidades de parada (*halting*) para cada valor de  $c$ , com os testes realizados por Bourke (2013). Apesar do algoritmo primeiramente ter sido aplicado para preenchimento do interior de uma região (chamados limites inclusivos), o mesmo pode ser estendido para permitir que as figuras ultrapassem as fronteiras da região, criando cópias de si mesmo e sendo posicionadas nos lados opostos da região. Neste caso, são chamados de limites periódicos.

Figura 6 – Probabilidade de parada para preenchimento de uma região retangular com círculos para limites inclusivos e períodicos.



Fonte – Bourke (2013)

O argumento sobre o ajuste das duas primeiras figuras no exemplo dos quadrados é mais complicado quando considera-se limites periódicos. Aqui cada figura que ultrapassa a fronteira tem um ou mais “parceiros” localizados em  $x \pm L_x$  e/ou  $y \pm L_y$ , com  $L_x$  e  $L_y$ , respectivamente, largura e comprimento da região retangular.

Uma pergunta surge naturalmente: por que o valor máximo de  $c$  é menor para

uma figura irregular?

Para responder, considere a figura irregular  $J$  gerada do seguinte modo: seja  $Q_0$  um quadrado com lado  $l_0$ , retire de  $Q_0$  um quadrado  $Q_1$  comprimento  $l_1 = l_0/2$ . A figura resultante é a figura  $J$ .

Suponha que deseja-se preencher o interior de uma região quadrangular de lado  $s$  (limites inclusivos) com figuras irregulares  $J$ . Considere as três primeiras iterações que são o resultado de inserir as figuras  $J_1$ ,  $J_2$  e  $J_3$ .

Pode-se mostrar que, para  $c = 1,2313$  obtém-se que  $l_0 = \frac{2}{3}s$  e, logo, os lados  $s_1$  e  $s_2$  das duas primeiras figuras serão maiores do que o lado da região ( $s_1 + s_2 > s$ ). No entanto, para este valor de  $c$  é possível que  $J_1$  caiba dentro de  $J_2$  (possível, mas improvável em uma busca aleatória).

Agora, para  $c = 1,2936$  obtém-se que  $s_1 + s_3 > s$ , de modo que as figuras  $J_1$  e  $J_3$  não se encaixem e o algoritmo vai parar quando  $c > 1,2936$ . Este valor máximo de  $c$  é muito inferior ao valor correspondente  $c = 1,5224$  para um quadrado cheio.

Este exemplo ilustra que, para uma figura irregular, o valor máximo é de  $c$  pode ser menor do que para uma figura geométrica primitiva (BOURKE, 2013).

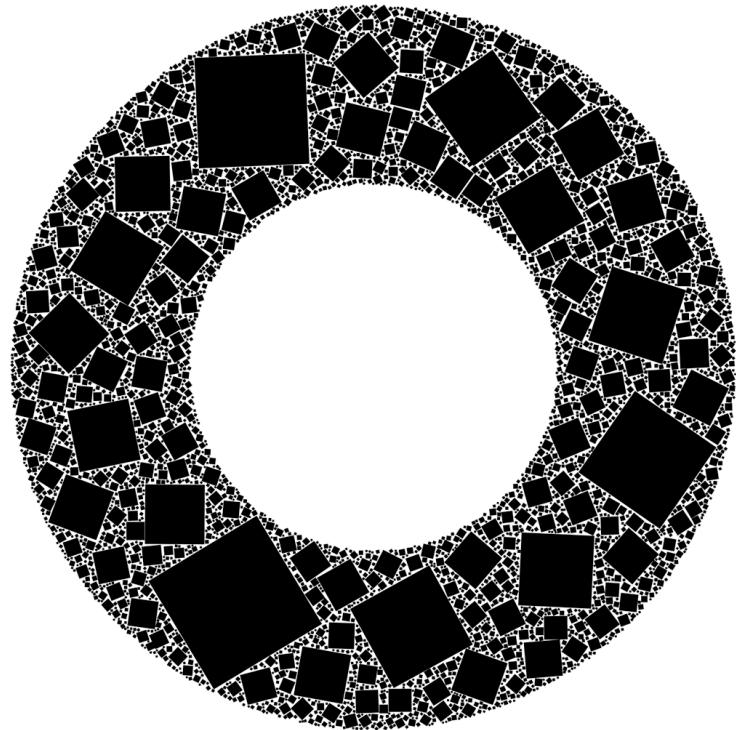
## 2.6 Formas irregulares

A definição do algoritmo (dada na Seção 2.4) depende somente da área da figura. Isto sugere que é possível usar uma mistura de formas contanto que a relação da área fosse mantida. O autor Bourke (2013) sugere que a autossimilaridade das figuras não é uma condição primordial do algoritmo.

Além disso, a descrição do algoritmo não se restringe ao preenchimento apenas regiões retangulares. Tudo o que é necessário é o cálculo da área da região a ser preenchida e um teste de interseção entre as figuras de preenchimento e as dimensões da região.

A Figura 7 ilustra o caso de quadrados aleatoriamente orientados preenchendo um anel. Neste exemplo não há possibilidade de utilizar limites periódicos, pelo menos não no sentido de um ladrilho retangular regular.

Figura 7 – Região não retangular, preenchida com quadrados. Limites inclusivos.



Fonte – Bourke (2013)

# 3 MODELAGEM DA FERRAMENTA

Neste capítulo é detalhada a modelagem da ferramenta desenvolvida no trabalho, chamada *Mosaico Fractal*.

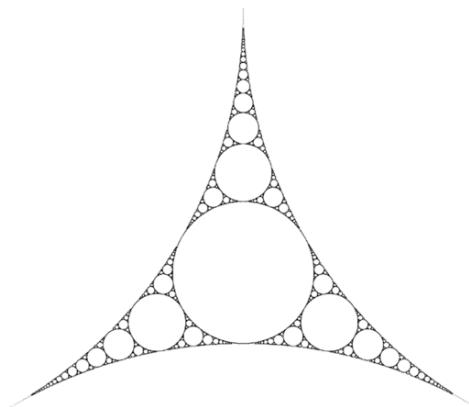
Como o autor do artigo não disponibilizou nenhum material sobre a implementação do algoritmo, nem uma ferramenta computacional para preenchimento de espaço, tem-se que criar uma ferramenta livre e de código aberto, para se usada de forma colaborativa e aplicada em texturas e texturas artísticas, é uma contribuição importante.

Os detalhes da modelagem abrangem a metodologia de desenvolvimento utilizada, diagramas gerados para uma melhor organização e outras informações relevantes.

## 3.1 Ferramenta *Mosaico Fractal*

O *Mosaico Fractal* é uma ferramenta de preenchimento de espaço que gera com produto final uma imagem no formato similar a um mosaico com figuras autossimilares semelhantes ao fractal de Apolônio (quando usa-se círculos). A Figura 8 ilustra o fractal de Apolônio citado pelos autores dos artigos estudados:

Figura 8 – Fractal de Apolônio.



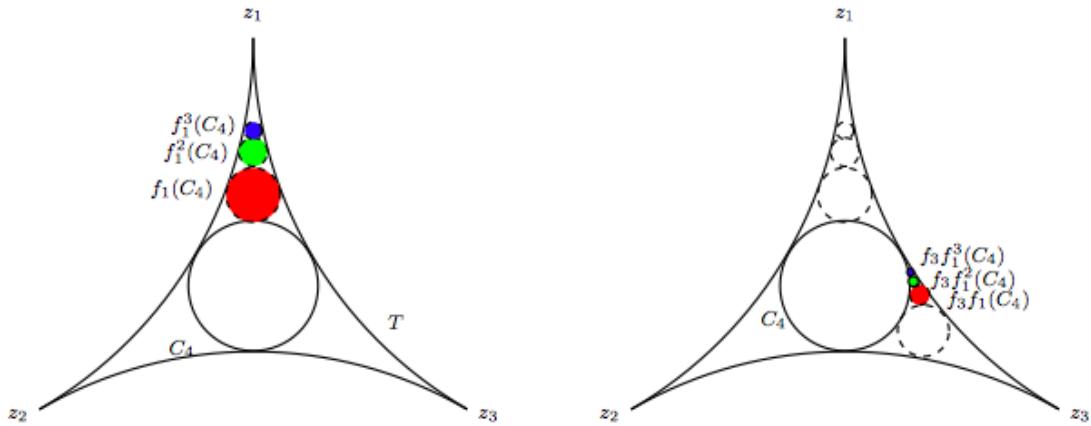
Fonte – Bourke e Shier (2013)

O problema de Apolônio é um dos grandes problemas da História da Geometria. Apolônio de Perga foi um grande geômetra grego, contemporâneo de Arquimedes, que viveu, aproximadamente, entre 287 a.C. e 212 a.C. Ele propôs um problema que atualmente é conhecido pelo seu nome: encontrar um círculo tangente a três outros círculos, que degeneram-se em retas (círculo de raio infinito) ou pontos (círculo de raio zero). Desde en-

tão, diversos matemáticos têm se empenhado na busca de soluções para este interessante problema.

No contexto da Geometria Fractal, o fractal de Apolônio é um fractal gerado por um sistema de funções iteradas, definidas num triângulo curvilíneo do plano complexo, que transformam um círculo inicial dado numa sequência de círculos, como mostra a Figura 9:

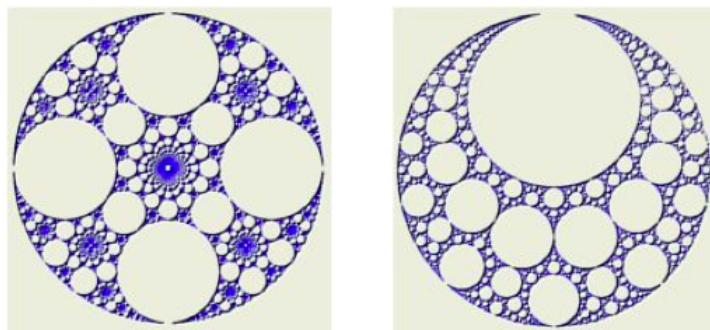
Figura 9 – Geração do fractal de Apolônio.



Fonte – Pollicott (2014)

Existem muitas variações do fractal de Apolônio, a Figura 10 ilustra duas delas:

Figura 10 – Variações do fractal de Apolônio.



Fonte – LLC (2017)

A ferramenta *Mosaico Fractal* permite ao usuário algumas opções antes da execução tais como cores, formatos, etc. e pode ser armazenada no computador do usuário.

A modelagem desta ferramenta foi baseada nos resultados obtidos nos artigos estudados (BOURKE, 2013; DUNHAM; SHIER, 2014), tendo-se como prioridade a facilidade

de utilização de modo que seja de uso agradável a usuários comuns, caracterizado como *user-friendly*.

Sem nenhuma informação sobre a implementação do algoritmo, surgiu a necessidade de criação de diagramas baseados em conceitos básicos de orientação a objetos e também foi preciso pesquisar requisitos com base nas possíveis necessidades do usuário.

## 3.2 Requisitos

A coleta de requisitos foi baseada das opções listadas pelos autores Dunham e Shier (2014, p. 85). Boa parte destes requisitos estão relacionados a interação do usuário, além das opções de distribuição das figuras na região, porém estes requisitos também estão ligados ao fato do algoritmo apresentado ser muito geral e está sujeito a um grande número de possíveis variações, na sua maioria inexploradas.

Apesar de muitos requisitos terem sido listados pelos próprios autores, foram acrescentados outros para a implementação da ferramenta do trabalho, requisitos relacionados com a disponibilização da ferramenta como *software* livre e de código aberto, tendo assim requisitos específicos relacionados a estas características.

Também foi feita uma implementação de programação com utilização de *threads*, assim permitindo realizar o teste de intersecção em duas ou mais tarefas que podem ser executadas concurrentemente. Essa implementação foi feita com objetivo de agilizar a execução do programa.

Os requisitos funcionais identificam as principais funções pertinentes para a implementação da ferramenta, dizendo o que o programa deve fazer. Já os requisitos não-funcionais descrevem como ele fará as atividades descritas. Os requisitos funcionais estão descritos no Quadro 1 a seguir:

Quadro 1 – Requisitos funcionais da ferramenta.

Nº do Requisito	Descrição
RF 01	O programa deve ser capaz de gerar uma imagem com as entradas de dados definidas pelo usuário.
RF 02	O programa deve ser capaz de receber entradas de arquivos vetoriais.
RF 03	O programa deve permitir ao usuário de escolher as entradas: forma da estampa, forma da região de preenchimento e preenchimento das estampas.
RF 04	O programa deve permitir ao usuário de salvar a imagem.
RF 05	O programa deve permitir ao usuário modificar o valor dos parâmetros numéricos utilizados na síntese de imagem: parâmetros $c$ e $N$ , número de estampas, número de iterações e tempo de execução.
RF 06	O programa deve permitir ao usuário modificar o valor dos parâmetros de condição utilizados na síntese de imagem: limites inclusivos ou periódicos, rotação de estampas e tipo de preenchimento.
RF 07	O programa deve ter uma documentação clara sobre suas classes e métodos.
RF 08	O programa deve ter seu código-fonte disponibilizado em um repositório e permitir contribuições.

Fonte – Elaborada pela autora

Os requisitos funcionais listados têm como objetivo principal permitir ao usuário uma certa liberdade criativa no preenchimento de espaço com autossimilaridade, dando-lhe a oportunidade de gerar texturas mais artísticas.

Por outro lado, os requisitos não-funcionais coletados servem para assegurar alguns aspectos de qualidade de desempenho e delimitar características do *software*, e estão descritos no Quadro 2:

Quadro 2 – Requisitos não-funcionais da ferramenta.

Nº do Requisito	Descrição
RN 01	O programa deve executar a tarefa dentro de um tempo aceitável delimitado pelo usuário.
RN 02	O programa deve indicar claramente ao usuário quais foram as opções por ele escolhidas até o momento antes de executar o processamento.
RN 03	O programa deve permitir a opção de salvar a imagem rasterizada e vetorizada.
RN 04	O programa deve permitir ao usuário de escolher um grande número de cores para o preenchimento de estampas.
RN 05	O programa deve permitir ao usuário realizar o preenchimento de espaço com múltiplas formas diferentes em uma mesma execução.

Fonte – Elaborada pela autora

Como pode ser observado, os requisitos não-funcionais servem para garantir alguns aspectos de qualidade do programa, como, por exemplo, não permitir que o usuário es-

pere indefinidamente pelo resultado da execução, caso os valores escolhidos por ele sejam inapropriados para a figura escolhida (no caso, serem grandes demais para uma figura complexa, que necessita de coeficientes pequenos). Assim, o programa deve fazer uma escolha, no caso citado, modifica um dos coeficientes utilizados ou estabelece um tempo de execução limite, que pode ser definido pelo usuário.

Os casos de uso, detalhados no Quadro 3, servem para descrever como será realizada as ações para satisfazer determinados requisitos da ferramenta, que pode ser ilustrado também através de diagramas. O quadro demonstra alguns casos de uso ligados à função principal do programa de preenchimento de espaço.

Quadro 3 – Principais casos de uso da ferramenta.

Nº do Caso de Uso	Descrição	Requisitos Relacionados
UC 01	Gerar uma imagem.	RF 01
UC 02	Inserir formas vetoriais em arquivos vetoriais.	RF 02
UC 03	Inserir texturas vetoriais em arquivos vetoriais.	RF 02
UC 04	Escolher a forma da estampa desejada.	RF 03
UC 05	Escolher a forma da região de preenchimento desejada.	RF 03
UC 06	Salvar a imagem gerada.	RF 04
UC 07	Definir se as estampas serão posicionadas com rotações aleatórias ou não.	RF 06
UC 08	Escolher as cores que serão utilizadas para o preenchimento das estampas.	RF 03
UC 09	Escolher a cor que será utilizada para o fundo da tela.	RF 03
UC 10	Escolher as texturas que serão utilizadas para o preenchimento das estampas.	RF 03
UC 11	Definir se irá utilizar limites inclusivos ou periódicos.	RF 06
UC 12	Escolher valor dos parâmetros numéricos.	RF 05

Fonte – Elaborada pela autora

O caso de uso UC 01 possui como precondições: UC 04, UC 05, UC 07, UC 08, UC 09, UC 11 e UC 12. Estas precondições são resolvidas com escolhas marcadas por padrão na ferramenta. O caso de uso UC 06 é considerado uma pós-condição opcional para UC 01, onde o usuário é informado da opção de salvar a imagem se desejado. Já os casos de uso UC 02, UC 03, UC 10 são opcionais. O fluxo de evento principal do programa se descreve da seguinte forma:

1. O programa *Mosaico Fractal* exibe a tela com as opções marcadas por padrão: limites inclusivos, não rotacionar estampas, região de preenchimento sem forma específica, estampa de forma circular, cor preta para as estampas, cor branca para

a região de preenchimento, coeficiente  $c$  igual a 1,53, valor de  $N$  igual a 3, limite de quantidade de formas igual a 1.000, limite de iterações igual a 1.000 e limite de tempo de execução igual a 1 minuto;

2. (opcional) O usuário escolhe as opções desejadas relacionadas: limites periódicos, rotacionar aleatoriamente estampas e/ou forma da região de preenchimento;
3. (opcional) O usuário adiciona as suas formas e/ou texturas personalizadas;
4. (opcional) O usuário modifica os valores: coeficiente  $c$ ,  $N$ , limite de quantidade de formas e/ou limite de iterações;
5. (opcional) O usuário escolhe as cores/texturas desejadas para as estampas e/ou para a região de preenchimento;
6. (opcional) O usuário escolhe o tempo limite da execução do preenchimento de espaço;
7. O usuário aciona o comando *Iniciar*;
8. O programa preenche a região com as formas escolhidas pelo usuário dentro de um determinado tempo;
9. O programa finaliza o preenchimento com uma mensagem *pop-up* perguntando se o usuário deseja salvar a imagem;
10. (opcional) O usuário salva a imagem rasterizada em PNG e/ou vetorizada em SVG.

### 3.3 Diagramas

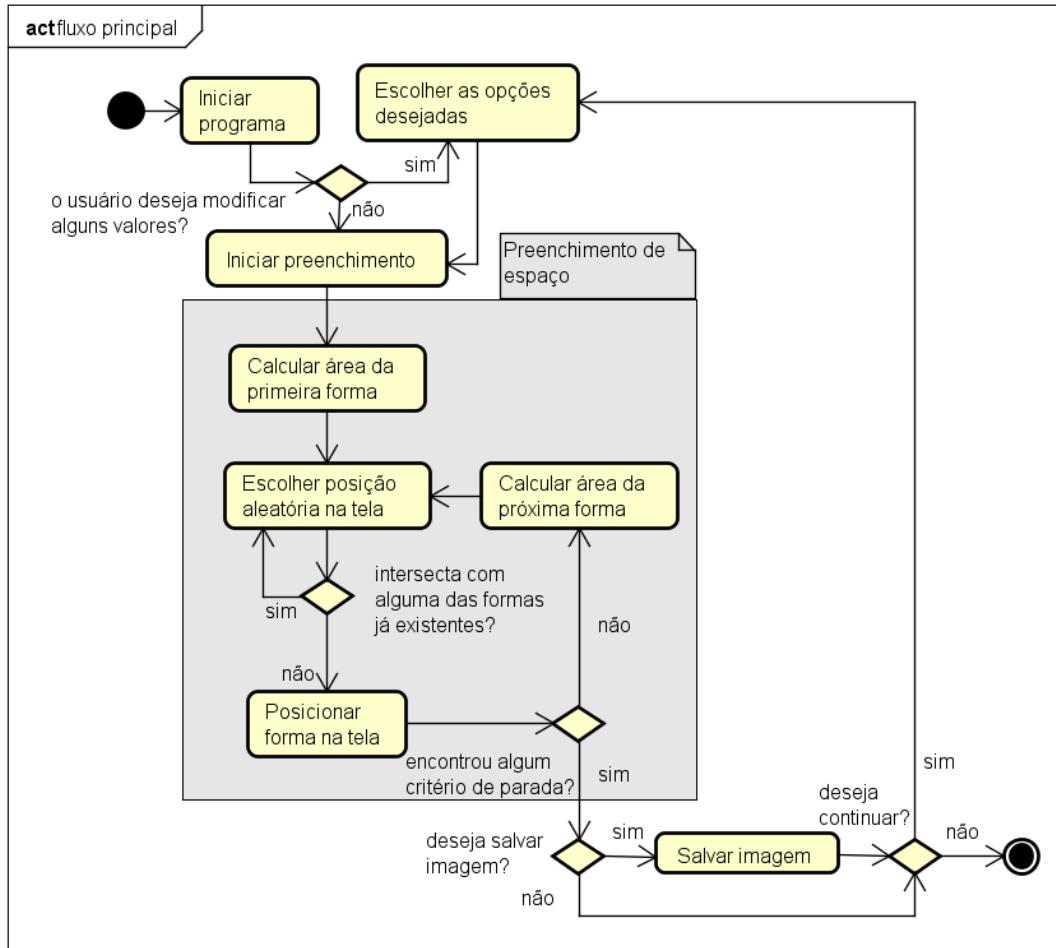
A partir dos requisitos listados, foram gerados outros documentos para apoiar o desenvolvimento da ferramenta. Em primeira instância foi criado um diagrama de fluxo simplificado descrevendo a execução da atividade principal do programa. Todos os diagramas foram criados utilizando a ferramenta de modelagem UML Astah Professional (Change Vision, 2017). Foram gerados 4 diagramas diferentes, sendo estes:

1. Diagrama de atividades: fluxograma utilizado para determinar as etapas que o programa pode ou deve seguir, e quais as condições que podem influenciar na tomada dos caminhos escolhidos;
2. Diagrama de caso de uso: usado para ilustrar as ações que o usuário pode realizar, juntamente com as dependências ou opções;
3. Diagrama de classe: ilustrando a comunicação entre as classes criadas;

4. Diagrama de pacotes: mostra as dependências entre os pacotes criados ou utilizados.

A Figura 11 tem como objetivo definir o fluxo principal da ferramenta.

Figura 11 – Diagrama de atividades do fluxo principal.

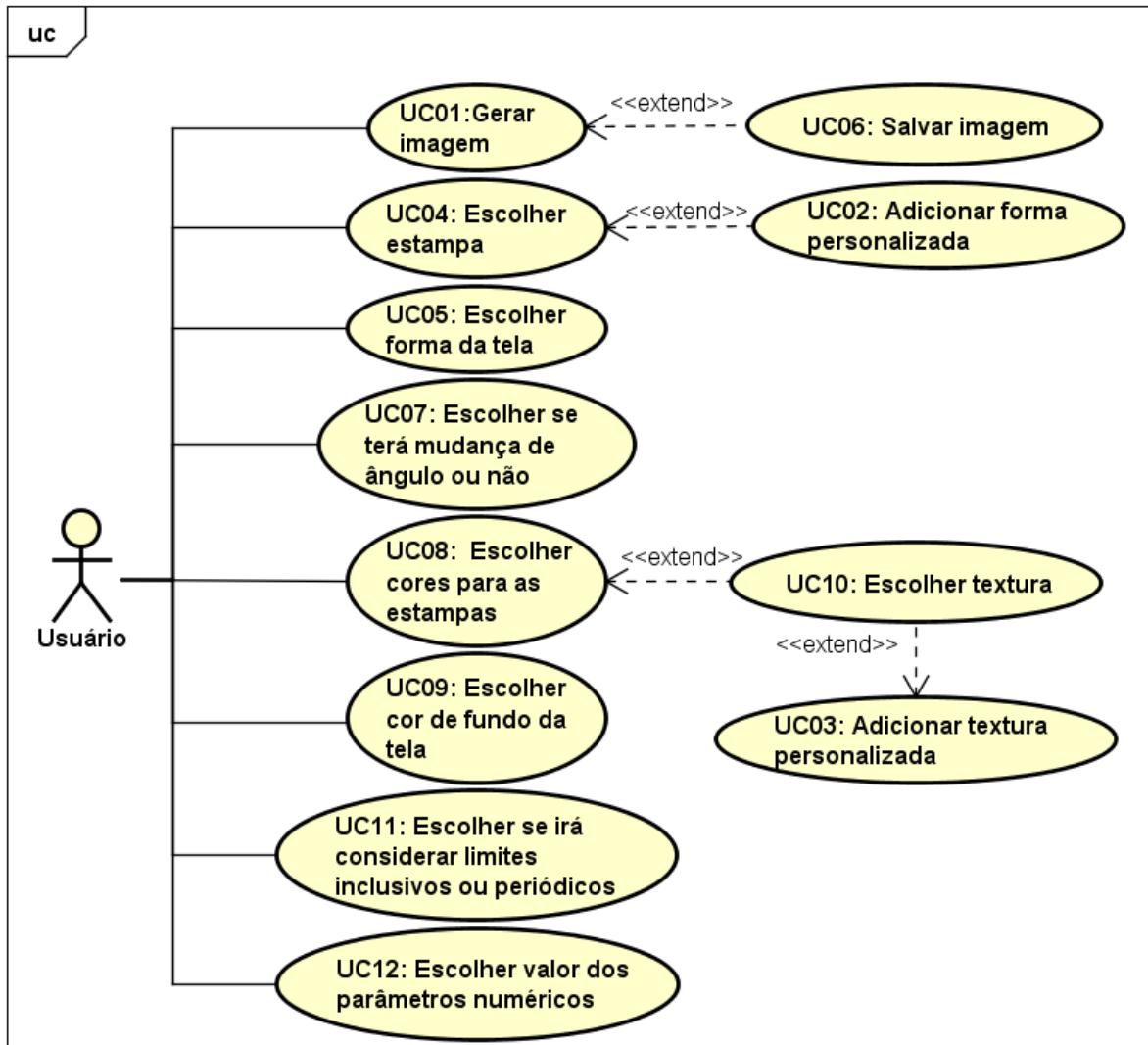


Fonte – Elaborada pela autora

O fluxo principal já foi descrito no fim da Seção 3.2 por passos, sendo o diagrama utilizado para ilustrar de forma que a modelagem seja apresentável na documentação da ferramenta e assim de fácil entendimento a quem desejar contribuir com o desenvolvimento de versões melhoradas da ferramenta.

A Figura 12 trata do diagrama de caso de uso que também já foi detalhado na Seção 3.2, onde são ilustradas as opções que o usuário possui na interação com a ferramenta, e também indicando quais as funções que tratam de determinados UC do Quadro 3.

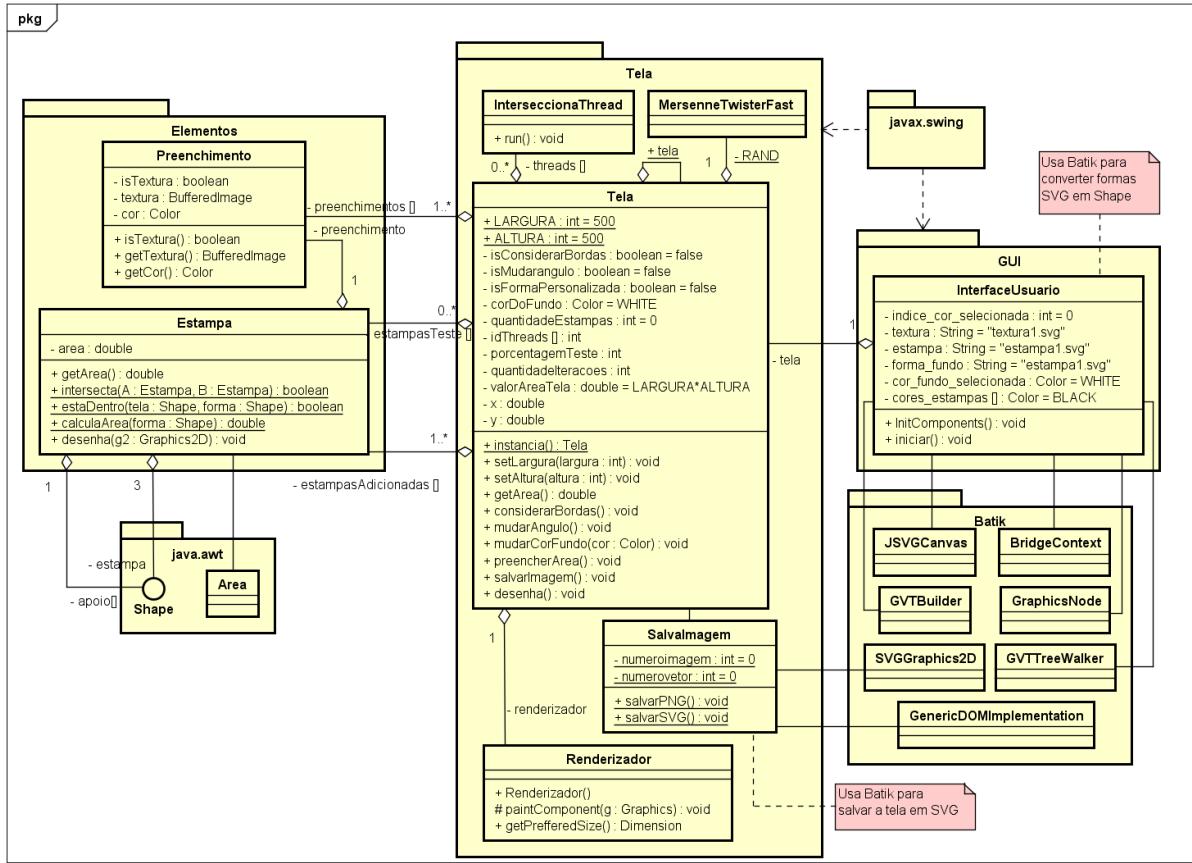
Figura 12 – Diagrama de casos de uso.



Fonte – Elaborada pela autora

O paradigma utilizado para o desenvolvimento da ferramenta foi o de orientação a objetos. No diagrama de classes simplificado mostrado na Figura 13 estão algumas das classes envolvidas na programação da ferramenta e suas funções principais, além de atributos relacionados.

Figura 13 – Diagrama de classes.



Fonte – Elaborada pela autora

Dentre as classes principais têm-se a classe *Tela*, que é usada para encapsular as funções principais do programa na geração da tela e preenchimento com estampas, de acordo com os valores definidos pelo usuário através da classe *InterfaceUsuario*, e utiliza a classe *MersenneTwisterFast* para gerar números aleatórios com mais velocidade, visto que a classe *Random* do Java é 1/3 mais lenta que a anterior (MATSUMOTO; NISHIMURA, 1998). A classe *Tela* utiliza alguns elementos da biblioteca *Swing* da linguagem Java (Oracle, 2017), para visualizar os gráficos gerados.

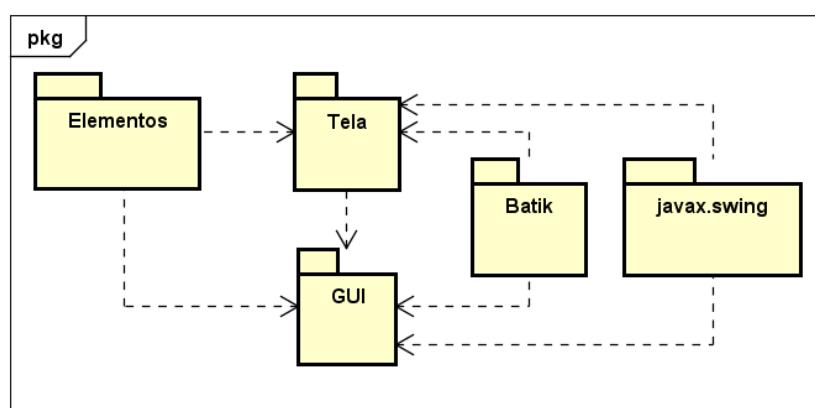
Já a classe *Estampa* é usada para encapsular as funções relacionadas a estampas que são manipuladas na tela. Ela cuida de verificar intersecção com outra estampa, e se a mesma se encontra totalmente dentro da tela personalizada. Esta classe utiliza a interface *Shape* para representar a forma, e também utiliza a classe *Area* para operações de intersecção. A forma como estes elementos são utilizados será melhor explicada na Seção 4.1, que trata com mais detalhes sobre a programação da ferramenta. Por fim, a classe *Preenchimento* cuida de valores relacionados ao elemento usado para preencher uma estampa. Um preenchimento pode ser uma simples cor ou uma textura, a qual é uma imagem rasterizada.

Existem classes que cuidam de tarefas secundárias, como a classe *Renderizador*, que serve para renderizar corretamente os elementos gráficos da tela. Ela possui como objetivo impedir que elementos sejam renderizados abaixo da barra de título da janela que é aberta quando se deseja visualizar a tela, além de permitir que a região possua as devidas dimensões de largura e altura. Já a classe *SalvaImagem* cuida de rasterizar a tela gerada pelo programa para PNG, ou salvando como um documento de imagem vetorizada SVG. Esta classe possui dependências com as classes *SVGGraphics2D* e *GenericDOMImplementation*, da toolkit Batik (Apache, 2017), para converter os elementos gráficos de *Java2D* para SVG, e disso salvar o documento.

A classe *InterfaceUsuario* utiliza-se de diversos elementos *Swing* para gerar uma interface interativa de usuário, juntamente com a classe *JSVGCanvas* da toolkit Batik, onde é utilizada para mostrar ao usuário uma pré-visualização das estampas e/ou texturas escolhidas. As classes *BridgeContext*, *GraphicsNode*, *GVTBuilder* e *GVTTreeWalker* são utilizadas para extrair em baixo nível a implementação da interface *Shape* do documento SVG inserido pelo usuário para a questão de estampas. Estas classes serão explicadas com mais detalhe na Seção 4.1.1.

A Figura 14 mostra a relação entre os pacotes criados e utilizados pela ferramenta. O pacote *Elementos* fornece a estrutura necessária para a representação das estampas, utilizada nos pacotes *Tela* e *GUI*, os quais utilizam os pacotes externos Batik e *Swing*, importantes para a ferramenta. Neste caso foi ignorado a visão dos diversos pacotes utilizados como *java.awt*, *java.util*, e *java.io*, por tratarem de operações muito básicas para o contexto da ferramenta e serem pacotes utilizados por quase todos os demais, o que acabaria atrapalhando a visualização do esquema mais importante.

Figura 14 – Diagrama de pacotes.



Fonte – Elaborada pela autora

# 4 FERRAMENTA DESENVOLVIDA

Este capítulo tem como objetivo apresentar a ferramenta desenvolvida. Nele serão feitas abordagens em um nível mais baixo de programação descrevendo-se as técnicas utilizadas.

A ferramenta *Mosaico Fractal* se encontra disponibilizada ao público no serviço de hospedagem de projetos Bitbucket, onde usuários podem, além de baixar a ferramenta compilada, possuir acesso ao código-fonte, documentação, diagramas, histórico de *commits*, etc. Os usuários também podem realizar a operação de bifurcação (*fork*) no projeto, podendo desenvolver de forma independente uma nova e possível funcionalidade (*feature*) (Atlassian, 2017).

## 4.1 Programação

A IDE NetBeans foi utilizada para criar a ferramenta por ser a IDE oficial para Java 8.1, assim possui um constante aprimoramento do editor e dos recursos avançados, com um analisador de código que aprimora a qualidade do código desenvolvido. Também possui suporte ao controle de versões Git, mostrando previamente as alterações feitas após cada *commit*. A IDE também permitiu gerar a documentação da ferramenta, o que atribui uma melhor compreensão de seus métodos e classes para os que desejam contribuir para o seu desenvolvimento no futuro.

A linguagem Java foi escolhida por estar de acordo com as habilidades da desenvolvedora, e também por ser orientada a objetos, cujo paradigma permite o encapsulamento de métodos e classes, herança de classes, sobrecarga de métodos, entre outras características (Oracle, 2017).

Ainda relacionado a itens da linguagem Java, a interface *java.awt.Shape* foi de grande importância para o desenvolvimento do programa, pois as suas implementações devem obrigatoriamente sobreescriver o método *contains()*, o qual é utilizado no método desenvolvido que verifica se as coordenadas escolhidas aleatoriamente *x* e *y* estão dentro da região de preenchimento. Há um outro método da interface *Shape* que foi utilizada no início, chamado *intersects()*, porém o método não possui uma boa precisão por retornar verdadeiro (*true*) quando há uma alta probabilidade de que uma forma *java.awt.geom.Rectangle2D* e o *Shape* se cruzarem (sendo utilizada para verificar se havia intersecção entre duas formas posicionadas na região de preenchimento), mas os cálculos para determinar com precisão este cruzamento são proibitivamente complexos computacionalmente. Isso significa que para algumas formas este método pode retornar verdadeiro

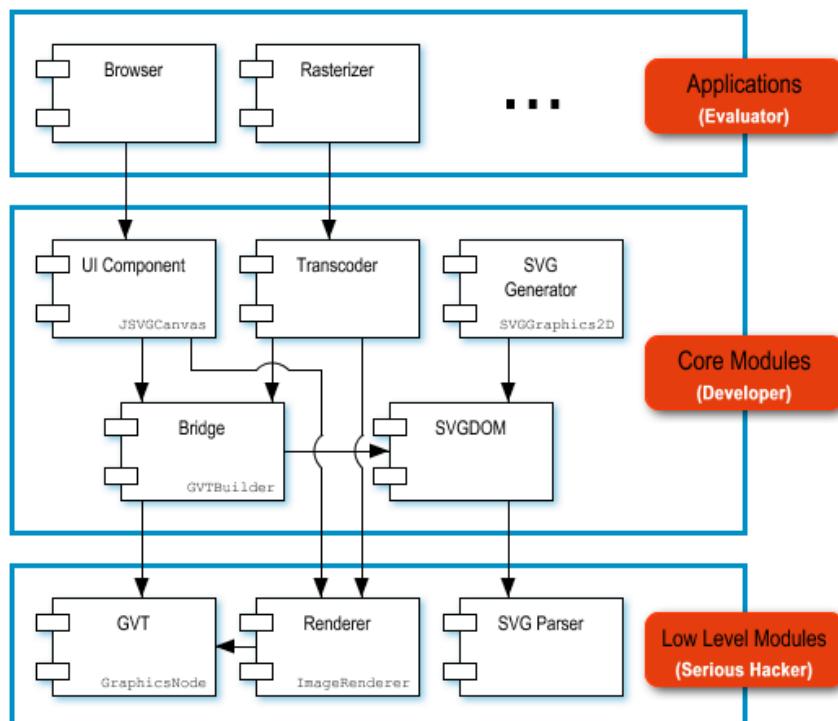
mesmo que o *Rectangle2D* não intersecte com *Shape*, assim não sendo de bom uso para este trabalho.

Pesquisando-se uma forma de resolver este problema, a classe *java.awt.geom.Area* foi extremamente importante para a resolução deste, que ao ser instanciada recebendo como entrada uma forma que implementa *Shape* pode-se utilizar o seu método *intersects()*, que realiza cálculos geométricos bem mais precisos que o método de mesmo nome de *Shape*. Com este método é possível verificar se duas formas que implementam *Shape* se intersectam ou não, pois caso se não se toquem o método retornará vazio (*null*), e caso contrário retornará a intersecção entre as formas (Oracle, 2017).

#### 4.1.1 Batik

Com o método de verificação de intersecção entre as formas e o método para assegurar que a forma esteja dentro da região de preenchimento prontos, o que faltava no momento era uma forma de extrair dos arquivos vetoriais SVG criados pelo usuário uma forma que implemente *Shape* para ser utilizada. Para isso foi utilizado o Batik, por oferecer uma biblioteca que permite tanto visualizar os arquivos SVG na interface de usuário quanto extrair implementações de *Shape* dos arquivos vetoriais (Apache, 2017). A Figura 15 mostra uma visão geral da arquitetura da *toolkit* e os pacotes oferecidos.

Figura 15 – Três tipos de módulos da arquitetura da *toolkit*.



Fonte – Apache (2017)

Foram utilizadas classes de módulos pertencentes à 2<sup>a</sup> e à 3<sup>a</sup> camada da arquitetura, sendo os módulos descritos a seguir os utilizados para o desenvolvimento desta ferramenta:

- **UI Component:** oferece componentes de interface de usuário para visualizar conteúdos em SVG. Foi utilizada a classe *JSVGCanvas* para mostrar as formas selecionadas pelo usuário como pré-visualização;
- **SVG Generator:** é o módulo que contém a classe *SVGGraphics2D*, utilizada para converter facilmente elementos gráficos do Java para o formato SVG;
- **SVGDOM:** é uma implementação da API de SVG DOM e permite a manipulação de documentos SVG. Foi utilizada a classe *GenericDOMImplementation* para construir o documento SVG e inserir as formas geradas pela *SVGGraphics2D*;
- **Bridge:** é responsável por criar e manter um objeto apropriado correspondente a um elemento SVG, convertendo um documento SVG na representação interna que o Batik usa para gráficos (GVT). É utilizada a classe *BridgeContext*, cujo objeto instanciado está relacionado a um determinado documento SVG, e encapsula as ligações dinâmicas entre os elementos DOM e os nós GVT. Também é utilizada no programa para extrair o objeto instanciado da classe *GraphicsNode* de um elemento de um documento SVG. É um módulo raramente usado diretamente;
- **GVT:** representa uma visão da árvore DOM que é mais adequada para fins de renderização e manipulação de eventos. Este módulo descreve a árvore DOM em termos de uma árvore de objetos gráficos em Java. A classe *GraphicsNode* é utilizada para extrair a forma implementada em *Shape*.

Na Listagem de código 4.1 é mostrado como era feita inicialmente a extração da forma de um documento SVG através das classes fornecidas pela biblioteca Batik:

Listagem 4.1 – Código antigo de extração de formas Java2D de um arquivo SVG

```
BridgeContext contexto = canvas.getUpdateManager().getBridgeContext();
SVGDocument documento = canvas.getSVGDocument();
Element elemento = documento.getElementById("XMLID_1_");
GraphicsNode nodo = contexto.getGraphicsNode(elemento);
Shape forma = ((ShapeNode)nodo).getShape();
```

Fonte – Elaborada pela autora

O objeto *canvas* é do tipo *JSVGCanvas*, do módulo *UI Component*. Este método implica uma diversidade de limitações para o programa. Primeiro, só é capaz de extrair um elemento de um documento inteiro, o que limita o uso de estampas para o preenchimento de espaço para apenas 1. Segundo, este mesmo elemento necessita ter um ID

muito específico, “*XMLID\_1\_*”, que é o ID padrão criado pelo Adobe Illustrator para a primeira camada gerada de um arquivo vetorial. Caso não fosse encontrado um elemento com tal ID, o programa alerta ao usuário que o arquivo vetorial inserido deve possuir este determinado padrão para ser utilizado. Ao exigir que o usuário comum modifique o arquivo internamente, acaba inherentemente deixando o programa não mais *user-friendly*. Para contornar isto, foi utilizado mais uma outra forma de extração de formas. A Listagem de código 4.2 mostra a nova maneira de extrair mais formas de um único documento SVG.

Listagem 4.2 – Novo código de extração de formas Java2D de um arquivo SVG

```
BridgeContext contexto = canvas.getUpdateManager().getBridgeContext();
SVGDocument documento = canvas.getSVGDocument();
contexto.setDynamicState(BridgeContext.DYNAMIC);
GVTBuilder gvt = new GVTBuilder();
GraphicsNode nodo = gvt.build(contexto, documento);
GVTTreewalker arvore = new GVTTreewalker(nodo);
ArrayList<Shape> formas = new ArrayList<>();
while ( (nodo = arvore.nextGraphicsNode()) != null) {
    formas.add(nodo.getOutline());
}
```

Fonte – Elaborada pela autora

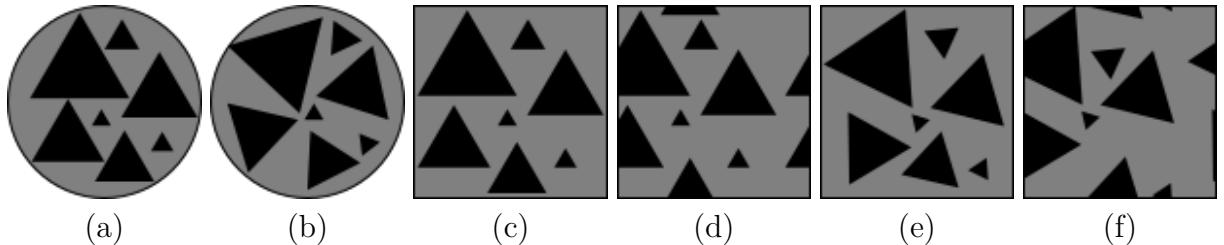
Um objeto *GVTBuilder* é instanciado, servindo para criar um nó de *GraphicsNode*, o qual é utilizado para gerar a árvore de elementos do documento SVG *GVTTreewalker*. Esta árvore é iterada coletando cada nó de *GraphicsNode* e adicionando o seu contorno em uma lista de formas utilizando o método *getOutline()*. Com isso, o usuário pode utilizar um arquivo vetORIZADO SVG contendo diversas formas dentro de um mesmo espaço, sendo inserido alternadamente e preenchendo o espaço de maneiras diferentes do que era possível anteriormente. Existem agora menos limitações, porém o que remanesce é o fato do usuário não poder combinar as estampas selecionadas dentro do programa, apenas as que estiverem em um único arquivo. Porém, isso já abre mais oportunidades de criatividade para o usuário.

## 4.2 Recursos utilizados

Foi utilizado o editor de imagens vetoriais Adobe Illustrator para gerar as estampas que foram necessárias para testar o funcionamento da ferramenta, além de ter sido utilizado para gerar alguns ícones necessários para uma melhor experiência de usuário com a ferramenta desenvolvida, indicando as principais opções selecionadas. Outras ferramentas podem ser utilizadas para gerar gráficos SVG e serem utilizados no programa, desde que siga as restrições descritas no parágrafo anterior. A Figura 16 mostra os ícones

gerados para ilustrar as diversas escolhas do usuário para a distribuição das formas na região de preenchimento.

Figura 16 – Ícones criados para o programa.

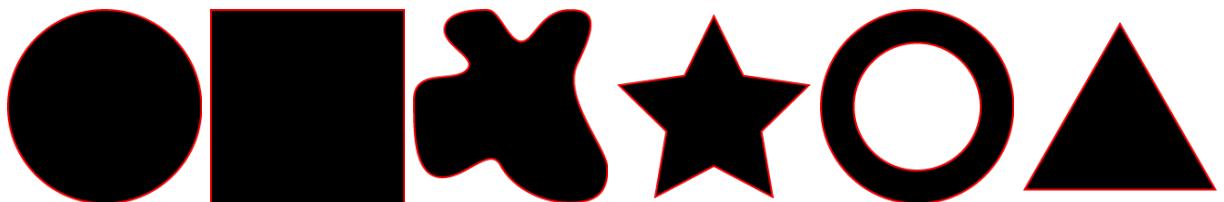


Fonte – Elaborada pela autora

Os primeiros dois ícones da esquerda para a direita (a) e (b) indicam que a região de preenchimento terá uma forma personalizada definida pelo usuário, ao contrário de uma região quadrada comum. Pode-se verificar que, quando utiliza a opção de região personalizada, acaba tornando inviável a opção de tornar os limites periódicos, pois uma região de preenchimento não-quadrada não geraria uma sensação de continuidade caso os seus limites fossem periódicos. Os próximos dois ícones (c) e (d) representam, respectivamente, inserção de formas em uma região quadrada sem rotações aleatórias das formas com limites inclusivos e periódicos. Por fim, os dois últimos ícones (e) e (f) indicam a mesma configuração, com a única diferença de que as formas agora possuem rotações aleatórias.

A Figura 17 mostra algumas das estampas criadas para realizar diversos testes com o programa, verificando se formas irregulares, com cantos agudos, ocas, etc, funcionam no programa com o resultado esperado. Além de servirem como estampas, as formas podem também ser usadas para definir a região de preenchimento da tela. No caso do usuário selecionar a opção de utilizar uma forma personalizada para a tela, os limites serão necessariamente sempre inclusivos.

Figura 17 – Algumas estampas criadas para o programa.



Fonte – Elaborada pela autora

## 4.3 Resultados

Com os principais métodos desenvolvidos e dependências definidas para resolver os problemas mais pertinentes, aliados aos requisitos e diagramas criados para facilitar a visão do desenvolvimento do programa, o resultado do trabalho é um *software* capaz de gerar imagens gráficas com o intuito de preencher o espaço de forma criativa e elegante, podendo salvar a imagem de forma rasterizada (em PNG) ou de forma vetorizada (em SVG). O usuário tem o poder de inserir suas próprias formas geradas para aumentar ainda mais o poder de criação do programa, e também pode definir até no máximo 50 cores para serem utilizadas no preenchimento das estampas. A Figura 18 mostra a tela inicial do programa, com algumas opções selecionadas para mostrar melhor o funcionamento da ferramenta e a janela de cores visível.

Figura 18 – Tela inicial do programa *Mosaico Fractal* com algumas opções selecionadas.



Fonte – Elaborada pela autora

Foram utilizados *radio buttons* para fornecer a visão da seleção de suas opções. O campo relacionado ao coeficiente  $c$  permite a entrada de valores entre 1,01 e 2, que deve ser selecionado com cautela para evitar o caso de parada por conta do coeficiente estar alto demais para uma determinada forma, como já foi visto na Seção 2.5. O campo relacionado ao valor  $N$  também está ligado a esta questão, e os valores disponibilizados pela interface estão entre 1 e 100. Caso aconteça uma condição de parada, haverá inserção de apenas uma ou pouquíssimas formas, pois quando o número de iterações realizadas em um teste de inserção de uma forma atingir o limite determinado, o programa deve parar a sua execução.

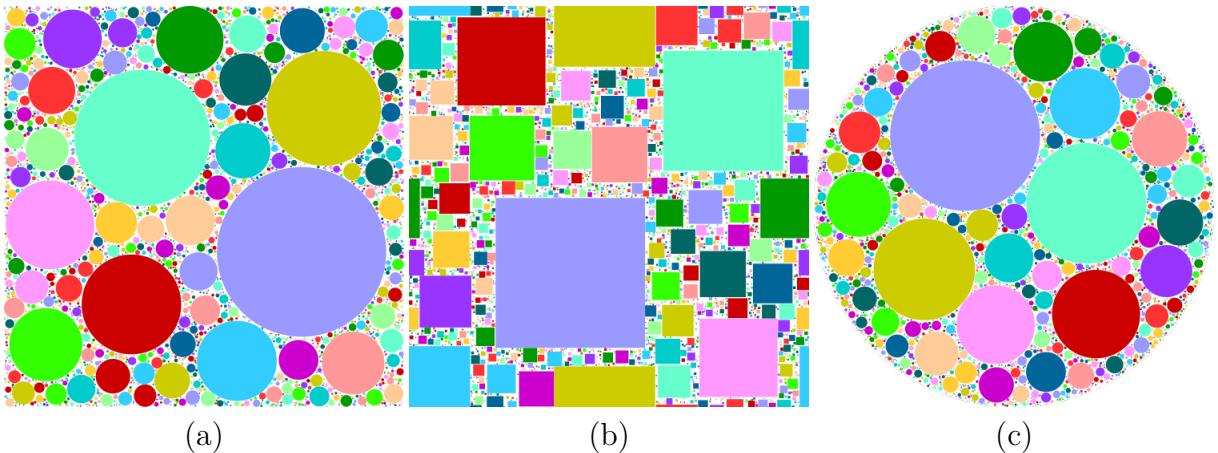
O campo sobre a quantidade de estampas permite que o usuário insira de 10 a 90.000 formas, enquanto que o campo delimitador da quantidade de iterações possui limites entre 100 e 400.000. Quanto mais simples for a forma da estampa mais rápida serão a execução do programa, demorando por exemplo em torno de 6 segundos para preencher uma região quadrada com 1.000 formas circulares, preenchendo 83% de sua área. Por outro lado, quanto mais complexa for a forma a ser utilizada, mais complexo será os cálculos geométricos realizados para determinar se há intersecção entre a forma a ser inserida com as demais já posicionadas, levando muito mais tempo de execução. Para contornar isso, o usuário pode determinar o tempo limite de execução, que pode ser configurado para até 60 minutos.

Um dos problemas que se tornaram pertinentes no trabalho era a busca de uma função inversa onde, dado a área desejada da forma, retorna-se a escala necessária para que transformar a estampa para tal área. Este problema foi encontrado quando incluiu-se formas irregulares inseridas pelo usuário, onde a única maneira de calcular a área da estampa era através de uma “integral”. Basicamente, rasteriza-se a figura para uma instanciação da classe *java.awt.image.BufferedImage*, e depois conta-se os *pixels* preenchidos, retornando o total contado. Mesmo após diversas tentativas de contornar o problema, ora acontecia o caso de esgotamento de espaço, ora a área não se tornava preenchida visualmente, obtendo resultados insatisfatórios de preenchimento da região quando utilizavam-se o valor do coeficiente  $c$  igual a 1,48 e de  $N$  como 2.

Manteve-se configurado então que o valor retornado da função inversa, que corresponde a porcentagem da área da região que a estampa atual deve preencher, será o valor escalar que multiplica os valores matriciais das formas. Desta forma o decaimento para  $c = 1,48$  e  $N = 2$  se torna bem mais acentuado, porém este problema pode ser contornado alternando tais valores e assim preenchendo melhor a região. Os valores encontrados que preenchem razoavelmente bem a região e sem o fenômeno de parada, para formas simples como círculos e quadrados, foram com  $c = 1,546$  e  $N = 3$ . Para estes valores, preenche-se até 86% da área com círculos e 90% com quadrados, nos dois casos utilizando 1.000 estampas.

A Figura 19 mostra alguns exemplos gerados utilizando formas simples, alternando com as diversas opções disponíveis, e com os valores de  $c = 1,53$  e  $N = 3$ . Na primeira Figura (a) tem-se a geração de círculos que preenchem uma região quadrada, com limites inclusivos. Já a segunda (b) tem-se quadrados preenchendo o espaço, com limites periódicos. A última (c) já inclui a questão de preencher uma região de forma personalizada, no caso um círculo. Nos três casos teve-se inserção de 2.000 estampas, e a porcentagem de preenchimento de espaço, na ordem, foi de 83%, 88% e 83%.

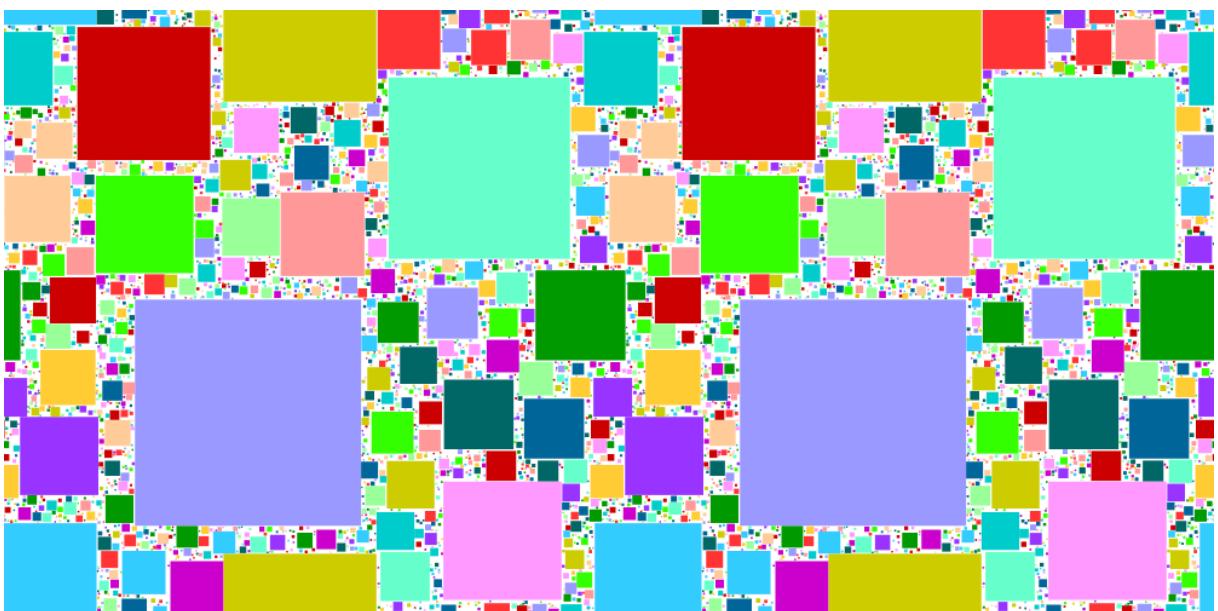
Figura 19 – Exemplos de execuções do programa utilizando formas simples.



Fonte – Elaborada pela autora

A segunda Figura (b) possui limites periódicos, isto é, as formas que ultrapassam os limites da região de preenchimento recebem “parceiros”, que são simplesmente replicações da mesma forma, porém em lados opostos da posição da estampa. Esta técnica pode ser utilizada para geração de texturas ou até mesmo de *tiles* em jogos, que quase sempre seguem estes princípios. A Figura 20 mostra a segunda figura do conjunto acima, porém replicada duas vezes.

Figura 20 – Utilização para geração de imagem com continuidade (*seamless*).

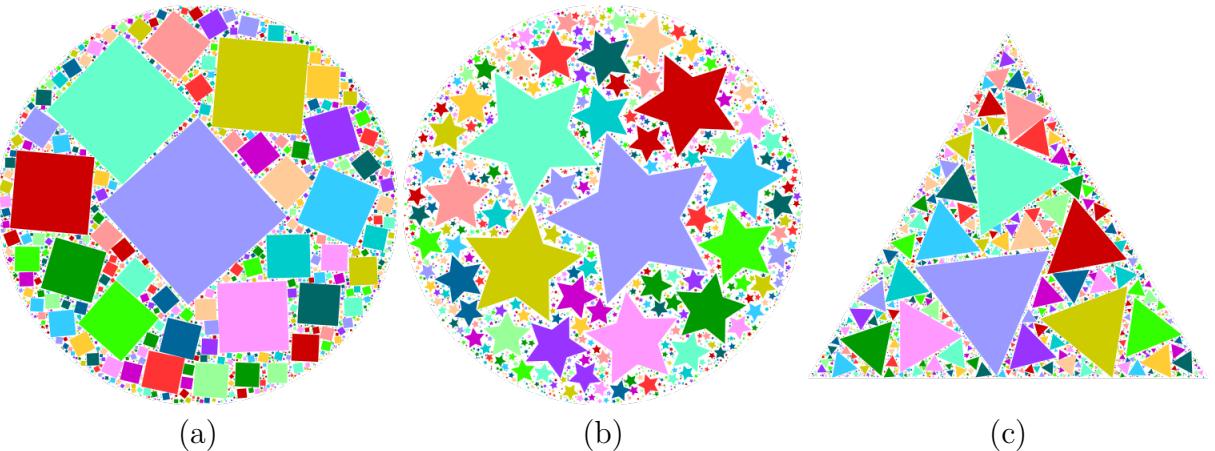


Fonte – Elaborada pela autora

Além de permitir a geração de resultados artísticos contendo replicação de estampas que ultrapassaram os limites da região, a ferramenta permite também a rotação

aleatória das estampas a serem inseridas, como pode ser visto na Figura 21. Pode-se observar também que formas com arestas agudas funcionam também para o preenchimento tanto de regiões simples, como de regiões que também possuem tais arestas agudas. Foi utilizados os valores  $c$ , na ordem, 1, 53, 1, 45 e 1, 5, e com  $N = 3$  para todas as formas. O preenchimento resultante foi respectivamente: 83%, 65% e 73%, e todas obtiveram inserção de 2.000 estampas.

Figura 21 – Exemplos de execuções do programa utilizando rotação de estampas.

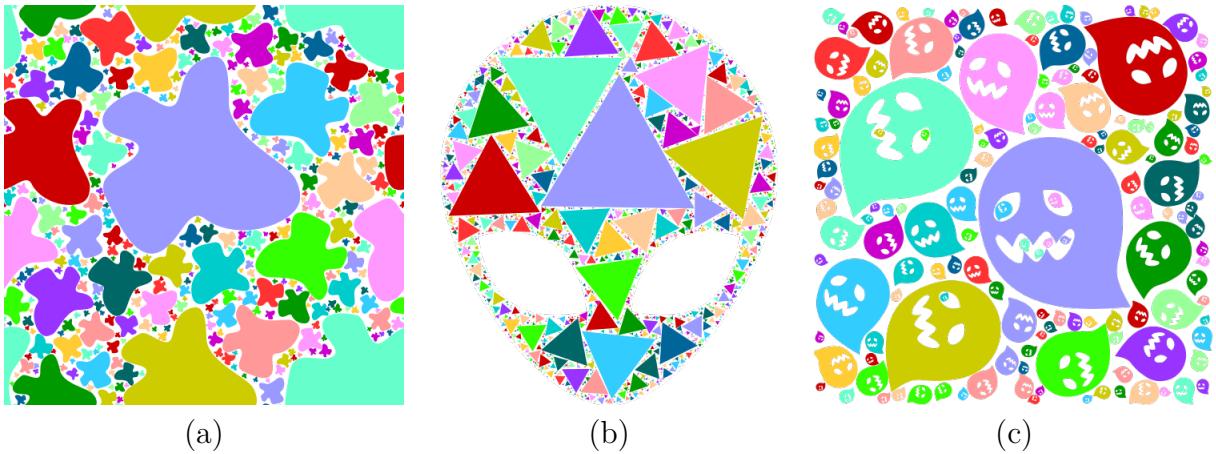


Fonte – Elaborada pela autora

A sensação de continuidade também é disponível para formas irregulares, assim podendo conter curvas, partes ocas, entre outras demais diferenças. O único requisito necessário é a aplicação do teste de intersecção entre formas e a região. Este teste deve levar em consideração que, no caso de uma região, as suas partes ocas não devem ser preenchidas, e já para estampas com partes vazias deve permitir tal inserção caso uma estampa tenha o tamanho suficientemente pequeno e sua posição correta para ser inserida dentro das partes ocas.

Exemplos podem ser visualizados na Figura 22. Os valores escolhidos para cada figura foram  $c = 1,53$  para (a), obtendo 81% de preenchimento,  $c = 1,49$  para (b), permitindo preencher 73% da tela, e  $c = 1,48$  para (c), com apenas 65% da região preenchida. Em todos os casos, foi utilizado  $N = 3$ . Vale notar que nem todas alcançaram o preenchimento com 2.000 formas, dentro do tempo limite estabelecido de 2 minutos. A primeira figura inseriu 387 estampas, a segunda sendo a única a inserir 2.000 estampas, e por último —com menor quantidade— 155.

Figura 22 – Exemplos de execuções do programa utilizando formas irregulares e múltiplas estampas.



Fonte – Elaborada pela autora

Em uma outra execução foi utilizada a forma da estampa da terceira figura do conjunto acima para preencher uma região quadrada. Neste procedimento foi utilizado limites periódicos, desejando observar o percentual de preenchimento de espaço e a quantidade de estampas que uma forma complexa permitiria inserir. Para  $c = 1, 4$  e  $N = 3$ , foram inseridas 333 estampas dentro de 2 minutos, com um preenchimento de 50% do espaço. A Figura 23 mostra o resultado obtido replicado duas vezes.

Figura 23 – Exemplos de execuções do programa utilizando uma forma complexa, com limites periódicos.

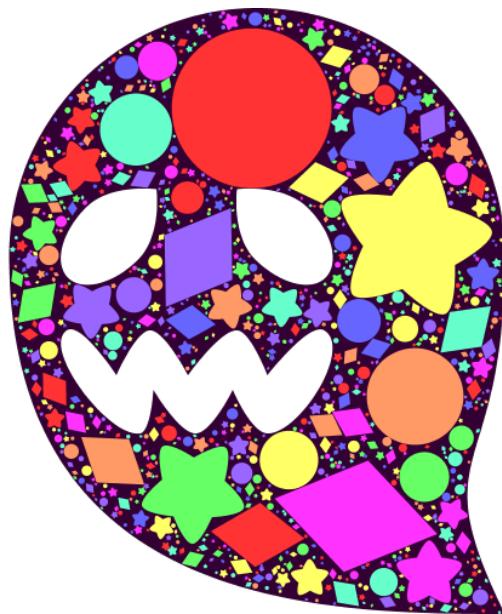


Fonte – Elaborada pela autora

Ainda se tratando da mesma forma irregular com partes ocas utilizadas nas figuras anteriores, foi feita também uma execução utilizando múltiplas formas em uma única região de preenchimento, procedimento tal possível graças a nova maneira de extrair formas, detalhada na Listagem 4.2. Esta execução se encontra na Figura 24.

Foram utilizadas 3 tipos de formas diferentes: círculo, losango, e estrela de pontas arredondadas. A inserção foi feita utilizando rotação de estampas, e foram definidos os parâmetros  $c = 1,45$  e  $N = 3$ , sendo assim inseridas 878 estampas dentro do limite de execução de 2 minutos, e o preenchimento resultante foi de 63%.

Figura 24 – Exemplo de execuções do programa utilizando múltiplas formas.



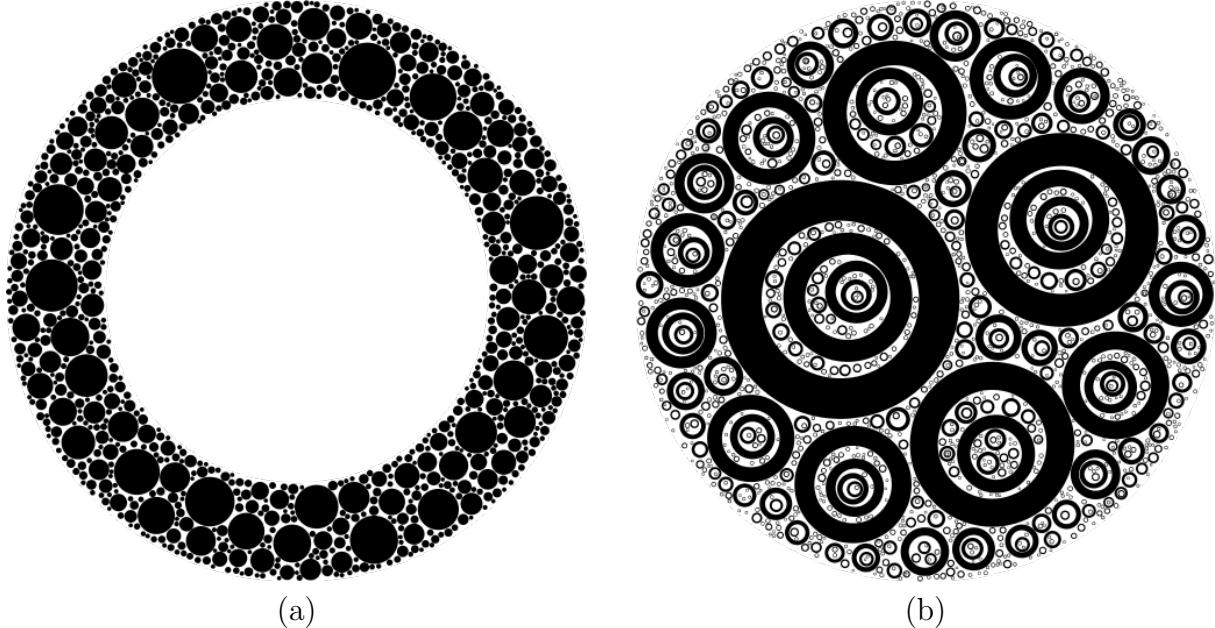
Fonte – Elaborada pela autora

Outro exemplo de uso de formas ocas tanto para região da tela quanto para as estampas pode ser observado na Figura 25, onde uma forma de anel anular é utilizado. No caso da utilização da forma para a região, foi necessário definir o valor de  $N = 20$ , por cair na questão de não possuir muitas regiões espaçosas para inserir formas cuja área da estampa inicial seja muito grande e o decaimento das áreas das próximas formas seja muito rápido. O coeficiente  $c$  neste caso foi 1,35, preenchendo 72% da região com 953 estampas. Nesta condição, a parada foi causada não pelo tempo de execução, mas sim pelo número de vezes que o teste de inserção foi feito, ultrapassando 2.000, resultando no tempo de execução igual a 57 segundos.

No caso contrário, onde a região de preenchimento era de círculos simples e as estampas utilizadas eram os anéis, o valor de  $c$  utilizado foi maior (1,45), e o de  $N$  menor (3). Foi possível preencher 63% da tela com 1.551 formas em 2 minutos. Pode-se observar

um grande número de estampas aninhadas dentro das primeiras formas inseridas. A Figura 25 mostra os resultados descritos.

Figura 25 – Exemplos de execuções do programa utilizando formas de anéis anulares.



Fonte – Elaborada pela autora

Por fim, foi testada uma imagem vetorial que, além de ser complexa no quesito de possuir diversas curvas e ângulos agudos, possui elementos totalmente desconectados entre si. Trata-se de um dragão, onde os objetos vetoriais (*paths*) que representam suas escamas e garras estão totalmente desconectadas do objeto principal, além de boa parte das asas, uma de suas patas e um de seus chifres. Claramente, a execução do algoritmo é extremamente demorada para preencher uma região dessas, sendo crucial o uso de limitadores de tempo. Outra questão importante se trata de que esta forma também não possui muitas regiões espaçosas para inserir formas em uma configuração de coeficiente  $c$  alto e valor  $N$  baixo, o que recai numa solução similar a da situação da Figura 25 acima.

Para isso, o valor inicial de  $N$  foi configurado para 20, permitindo um decaimento mais demorado, podendo preencher com maior uniformidade a região. Já o coeficiente  $c$  utilizado teve que ser pequeno, neste caso 1,25. Com isso, teve-se um preenchimento de 33% da área, inserindo 1.374 estampas em um período de 2 minutos. O resultado da execução se encontra na Figura 26, utilizando losangos como estampas e permitindo rotações aleatórias. Apesar do percentual de preenchimento baixo, o resultado obtido se assemelha a um mosaico geométrico, além de se apresentar esteticamente agradável. Porém, o resultado não é considerado satisfatório.

Figura 26 – Exemplo de execução do programa utilizando uma forma extremamente irregular para a região de preenchimento e estampas de ângulos agudos.



Fonte – Elaborada pela autora

#### 4.4 Requisitos atendidos

Após o desenvolvimento da ferramenta ter sido finalizado, foi realizada a verificação de quais os requisitos foram cumpridos dentro da descrição de cada um. Dentro do tempo determinado deste trabalho, foi possível cumprir os diversos requisitos funcionais listados no Quadro 1 e os requisitos não-funcionais descritos no Quadro 2.

O Quadro 4 é a lista de verificação (*checklist*) gerada para informar todos os requisitos coletados e seus status, identificando se foram superados ou se não foram alcançados.

Quadro 4 – Requisitos atendidos.

Nº do Requisito	Descrição	Status
RF 01	O programa deve ser capaz de gerar uma imagem com as entradas de dados definidas pelo usuário.	Cumprido
RF 02	O programa deve ser capaz de receber entradas de arquivos vetoriais.	Cumprido
RF 03	O programa deve permitir ao usuário de escolher as entradas: forma da estampa, forma da região de preenchimento e preenchimento das estampas.	Cumprido
RF 04	O programa deve permitir ao usuário de salvar a imagem.	Cumprido
RF 05	O programa deve permitir ao usuário modificar o valor dos parâmetros numéricos utilizados na síntese de imagem: parâmetros $c$ e $N$ , número de estampas, número de iterações e tempo de execução.	Cumprido
RF 06	O programa deve permitir ao usuário modificar o valor dos parâmetros de condição utilizados na síntese de imagem: limites inclusivos ou periódicos, rotação de estampas e tipo de preenchimento.	Cumprido
RF 07	O programa deve ter uma documentação clara sobre suas classes e métodos.	Cumprido
RF 08	O programa deve ter seu código-fonte disponibilizado em um repositório e permitir contribuições.	Cumprido
RN 01	O programa deve executar a tarefa dentro de um tempo aceitável delimitado pelo usuário.	Cumprido
RN 02	O programa deve indicar claramente ao usuário quais foram as opções por ele escolhidas até o momento antes de executar o processamento.	Cumprido
RN 03	O programa deve permitir a opção de salvar a imagem rasterizada e vetorizada.	Cumprido
RN 04	O programa deve permitir ao usuário de escolher um grande número de cores para o preenchimento de estampas.	Cumprido
RN 05	O programa deve permitir ao usuário realizar o preenchimento de espaço com múltiplas formas diferentes em uma mesma execução.	Cumprido

Fonte – Elaborada pela autora

Os requisitos RF 01, RF 03, RF 05, RF 06, RN 01 e RN 04 foram atendidos através de uma programação que aceita entradas do usuário através de uma interface gráfica e lança eventos para tratar das funções utilizadas. Já os requisitos RF 02, RF 04 e RN 05 foram cumpridos utilizando a *toolkit* Batik para extrair as formas dos arquivos vetoriais e para salvar as imagens em SVG. Por fim, RF 07 e RF 08 foram resolvidos inserindo comentários seguindo o padrão definido pelo Java para gerar a documentação, e hospedando o repositório em um site que permita a contribuição de usuários. Para ter acesso à documentação e ao código-fonte, consulte (Atlassian, 2017).

# 5 CONCLUSÕES

Neste trabalho investigou-se e implementou-se o algoritmo proposto no artigo de Bourke (2013) e o produto final desta investigação foi o desenvolvimento de uma ferramenta livre de código aberto.

Destaca-se que a literatura estudada (BOURKE, 2013; DUNHAM; SHIER, 2014) não disponibilizou a implementação do referido algoritmo, assim como também não foi possível encontrar nenhuma documentação ou programação da ferramenta final disponível na *internet*. Deste modo, a modelagem do *software*, a coleta de requisitos, a criação de diversos diagramas, a escolha das bibliotecas adequadas (para superar algumas necessidades), a codificação das funcionalidades, entre outros, são contribuições desta monografia.

A ferramenta desenvolvida chama-se *Mosaico Fractal* e tem como base um algoritmo de preenchimento de espaço que usa figuras geométricas primárias cujas áreas obedecem uma distribuição de probabilidade dada por Lei de Potência através da função zeta de Hurwitz. Como produto final têm-se imagens com texturas autossimilares semelhantes ao fractal de Apolônio (quando usa-se círculos).

## 5.1 Pontos positivos

Como principais características destacam-se: a execução do programa para gerar uma grande variedade de imagens; o acesso ao código-fonte para que o usuário investigar a funcionalidade e permissão de copiar, modificar e distribuir o *software* para benefício da comunidade. Isto possibilita aos usuários desenvolvam, colaborem e aprimorem a ferramenta, além de permitir que o conhecimento adquirido durante o desenvolvimento desta ferramenta seja acessível, lembrando que a programação não foi disponibilizada. Destaca-se também que a ferramenta tem um potencial lúdico e artístico e pode ser explorada em atividades de ensino-aprendizagem para estimular a criatividade ou produções de arte digital (DUNHAM; SHIER, 2014).

Foi introduzido então uma ferramenta com muitos dos recursos que são desejáveis ao criar texturas bidimensionais para multirresoluções processuais e iterativas. Esta habilidade é ideal para criar texturas com detalhe suficiente para satisfazer as restrições de desempenho dentro de um motor de jogo, dependendo da distância dos jogadores e da direção de visualização. O algoritmo implementado permite a geração de textura em uma escala de dimensões fractais e permite também criar texturas com continuidade dentro das regiões retangulares (BOURKE; SHIER, 2013).

O *software* desenvolvido está disponível como *open-source* e livre, sob a licença

GPL. O repositório encontra-se disponível no Bitbucket e para acessá-lo consulte (Atlas-sian, 2017).

No contexto da interdisciplinaridade, pode-se citar a Revista Brasileira de Computação SBC (2016, p. 9): “No âmbito universitário necessitamos construir, definir e avaliar práticas interdisciplinares que forneçam aos nossos alunos experiências de aprendizagem que qualifiquem sua formação profissional”.

Este trabalho buscou interações entre Computação, Matemática e Arte:

- a) Computação, com a modelagem e a programação de uma nova ferramenta que aplica na prática os conhecimentos teóricos estudados;
- b) Matemática, com o levantamento de estudo, fundamentação e envolvimento com a séries numéricas e introdução a probabilidade, ao tratar da função zeta de Hurwitz;
- c) Arte, com a utilização de estampas gráficas e criação de figuras artísticas.

A motivação para realizar um trabalho interdisciplinar vem da busca por uma formação profissional mais diversificada e inovadora. Foi uma experiência bastante positiva e interessante. Espera-se que esta monografia incentive a realização de mais trabalhos interdisciplinares e tenha contribuído com a criatividade e a inovação dos envolvidos.

## 5.2 Dificuldades encontradas

Um aspecto negativo do *software* é uma execução consideravelmente demorada para figuras geométricas complexas e/ou para uma região de formato complexo. Estas formas complexas também limitam o espectro de valores para as variáveis  $c$  e  $N$ , como foi visto na Seção 2.5. Na tentativa de reduzir o tempo de execução foi implementado o uso de *threads*, mas não obteve-se sucesso. Suspeita-se que o número de *threads* utilizado pela biblioteca *Swing* na interface de usuário possa interferir na avaliação entre uma execução *monothread* e uma com duas *threads*.

O aspecto considerado mais prejudicial é a impossibilidade de redimensionar, com precisão, as figuras irregulares retiradas dos documentos SVG inseridos pelo usuário para que tenham uma determinada área. Isto acontece por que não foi possível obter uma função inversa capaz de receber como entrada a área desejada e gerar como saída a figura com esta área.

Nas versões iniciais do programa, utilizou-se apenas figuras primárias tais como círculos e quadrados, e nestes casos era possível obter uma função que, dada uma determinada área, retornava-se o raio do círculo ou a diagonal do quadrado. A Listagem de código 5.1 mostra o procedimento para círculos:

Listagem 5.1 – Função geradora de raio de círculo baseado na área.

```
public static double raioGerado(double areaTela, double porcentagem) {
    return Math.sqrt(areaTela * porcentagem/Math.PI);
}
```

Fonte – Elaborada pela autora

A fim de contornar este problema, foi utilizado o valor da porcentagem da área da tela a ser preenchida como um escalar para transformar a matriz de pontos das figuras, redimensionando-a. Isto ocasiona um decaimento muito rápido das áreas das figuras. Assim, para este programa, os melhores valores para um preenchimento otimizado com figuras primárias foram  $c = 1,546$  e  $N = 3$  e não  $c = 1,48$  e  $N = 2$ .

Na implementação inicial, os testes de interseção das figuras eram mais simples, o que gerava um custo computacional bem mais barato comparado aos cálculos realizados pelo método *intersects()* da classe *Area* da linguagem Java (mais detalhes na Seção 4.1).

Todas estas diferentes propriedades causam impacto no tempo de execução do programa atual.

Um outro problema encontrado foi a baixa qualidade das texturas, visto que foi necessário rasterizar a textura para uma imagem com uma dimensão de  $100 \times 100$  pixels (o tamanho das janelas de pré-visualização das estampas e texturas), para depois aplicar nas figuras. Mesmo que este problema seja contornado, o fato de ser necessário rasterizar a textura que é vetorial faz-se perder a oportunidade de salvar a imagem resultante final para SVG, visto que documentos SVG são arquivos de texto que descrevem fórmulas matemáticas para desenhar os gráficos, não permitindo assim inserir em texto uma imagem rasterizada.

### 5.3 Trabalhos futuros

Ainda há muito a ser explorado com o *software* desenvolvido, visto que diversas outras funções podem ser programadas e assim aperfeiçoar a sua potencialidades computacionais, lúdicas e artísticas. Por exemplo, é possível estender as estampas de uma única figura para um conjunto de figuras agrupadas para serem manipuladas como uma única figura. Para isso deve-se realizar transformações como rotação e escala na região. Um caso particular que não foi citado pelos autores dos artigos tratados é a possibilidade de parar a execução de um preenchimento, modificar os parâmetros escolhidos, e reiniciar a partir daquele determinado ponto. Outra direção que pode ser explorada é o caso tridimensional, ou seja, preenchimento de volumes.

Com a aplicação para preenchimento de volumes em regiões tridimensionais, pode-se estender este trabalho para reconstrução de imagens tomográficas, preenchendo o vo-

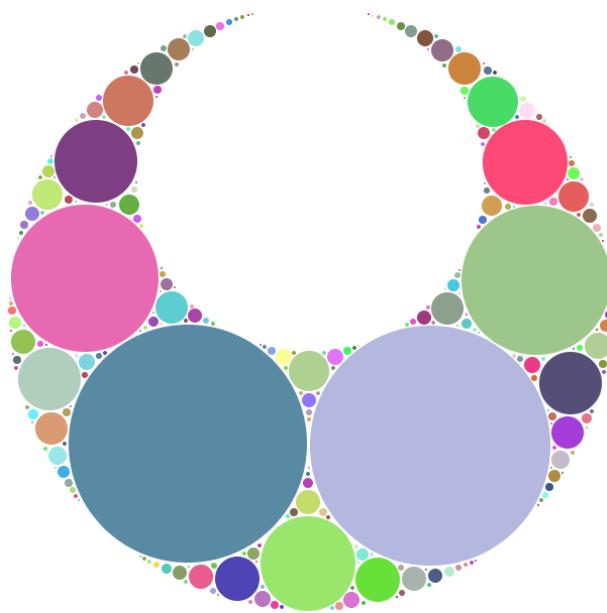
lume de regiões reconstruídas com técnicas de triangulação, obtendo assim modelos reais através de modelos virguais. Tais modelos podem ser manufaturados através de processos que podem ser úteis em muitas aplicações médicas como a fabricação de próteses, diagnósticos, planejamento de tratamento ou para guiar procedimentos cirúrgicos (SOUZA et al., 2001).

É necessário realizar um estudo mais aprofundado a respeito do espectro de valores disponíveis dos parâmetros  $c$  e  $N$ , e traçar gráficos verificando a porcentagem de cobertura de preenchimento de espaço para determinadas áreas. Apesar dos testes realizados pelos autores dos artigos estudados mostrarem uma relação entre estes parâmetros com os tamanhos das áreas das formas ao longo da execução, é necessário uma visualização da relação entre os parâmetros escolhidos, a forma da estampa e a porcentagem de preenchimento.

Durante a realização deste trabalho foi estudado artigos relacionados com fractal de Apolônio e suas variações, pois tinha-se como proposta inicial também implementar algoritmos para preenchimento espaço baseado em empacotamento em espirais (*spiral packing*) (BROWNE; WAMELEN, 2006), porém não foi finalizado a programação e não foi possível inserir a funcionalidade na ferramenta desenvolvida. Como trabalhos futuros pode-se ampliar as funcionalidades da ferramenta *Mosaico Fractal* para incluir preenchimento de espaço por espirais,

Com resultado preliminar deste estudo foi implementada uma geração simples e recursiva do fractal de Apolônio. A Figura 27 ilustra esta implementação:

Figura 27 – Fractal de Apolônio gerado com uma versão anterior do *Mosaico fractal*.



Fonte – Elaborada pela autora

# REFERÊNCIAS

- Apache. *Apache(tm) Batik SVG Toolkit - a Java-based toolkit for applications or applets that want to use images in the Scalable Vector Graphics (SVG)*. 2017. Disponível em: <<https://xmlgraphics.apache.org/batik/>>. Acesso em: 28 março 2017.
- APOSTOL, T. M. Zeta and related functions. NIST, 2010.
- Atlassian. *giordanna / Mosaico Fractal – Bitbucket*. 2017. Disponível em: <<https://bitbucket.org/giordanna/mosaico-fractal/>> Acesso em: 03 abril 2017.
- BOURKE, P. *Random space filling tiling of the plane*. 2011. Disponível em: <[http://paulbourke.net/texture\\_colour/randomtile](http://paulbourke.net/texture_colour/randomtile)>. Acesso em: 20 março 2017.
- BOURKE, P. A space filling algorithm for generating procedural geometry and texture. *GSTF Journal on Computing (JoC)*, Global Science and Technology Forum, v. 3, n. 2, p. 75, 2013.
- BOURKE, P.; SHIER, J. Space filling: A new algorithm for procedural creation of game assets. In: *Proceedings of the 5th Annual International Conference on Computer Games Multimedia & Allied Technology*. [S.l.: s.n.], 2013.
- BROWNE, C.; WAMELEN, P. van. Spiral packing. *Computers & Graphics*, Elsevier, v. 30, n. 5, p. 834–842, 2006.
- Change Vision. *Astah - Software Design Tools for Agile teams with UML, ER Diagram, Flowchart, Mindmap and More / Astah.net*. 2017. Disponível em: <<https://astah.net/>> Acesso em: 24 março 2017.
- CHURCHILL, R. V. *Variáveis complexas e suas aplicações*. [S.l.]: McGraw-Hill do Brasil, 1980.
- DANTAS, C. *Probabilidade: um curso introdutório*. [S.l.]: Editora da Universidade de São Paulo, 2008.
- DUNHAM, D.; SHIER, J. The art of random fractals. *Bridges Seoul,(eds. Gary Greenfield, George Hart, and Reza Sarhangi)*, Seoul, Korea, Citeseer, p. 79–86, 2014.
- DUNHAM, D.; SHIER, J. Fractal wallpaper patterns. *Bridges*, p. 183–190, 2015.
- LLC. *Fractal Science Kit Examples*. 2017. Disponível em: <<http://www.fractalsciencekit.com/tutorial/examples/examples.htm>>. Acesso em: 08 abril 2017.
- MANDELBROT, B. B. *Fractals*. [S.l.]: Wiley Online Library, 1977.
- MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, ACM, v. 8, n. 1, p. 3–30, 1998.
- OLIVEIRA, W. D. Zeros da função zeta de riemann e o teorema dos números primos. Universidade Estadual Paulista (UNESP), 2013.

- Oracle. *Software Java / Oracle Brasil*. 2017. Disponível em: <<https://www.oracle.com/br/java/index.html>> Acesso em: 27 março 2017.
- PAULINO, W. Determinação da distribuição de lei de potência aos tempos de reparo de uma linha férrea. *Faculdade de Engenharia de Bauru*, UNESP, 2012.
- POLLICOTT, M. *Apollonian Circle Packings*. 2014. Disponível em: <<http://homepages.warwick.ac.uk/~masdbl/apollo-29Dec2014.pdf>>. Acesso em: 08 abril 2017.
- QUAN YONG XU, Y. S. Y.; LUO, Y. Lacunarity analysis on image patterns for texture classification. *Computer Vison Foundation-CVF*, Citeseer, p. 1–8, 2014.
- SBC. *Computação e Interdisciplinaridade*. 2016. Disponível em: <[http://www.sbc.org.br/images/flippingbook/computacaobrasil/computa\\_31/Comp\\_Brasil\\_02\\_2016.pdf](http://www.sbc.org.br/images/flippingbook/computacaobrasil/computa_31/Comp_Brasil_02_2016.pdf)>. Acesso em: 08 abril 2017.
- SHIER, J. Filling space with random fractal non-overlapping simple shapes. *Hyperseeing summer*, p. 131–140, 2011.
- SHIER, J.; BOURKE, P. An algorithm for random fractal filling of space. In: WILEY ONLINE LIBRARY. *Computer Graphics Forum*. [S.l.], 2013. v. 32, n. 8, p. 89–97.
- SOUZA, M. de; RICETTI, F.; CENTENO, T.; PEDRINI, H.; ERTHAL, J.; MEHL, A. Reconstrução de imagens tomográficas aplicada à fabricação de próteses por prototipagem rápida usando técnicas de triangulação. *Proc Cong Latinoam Ingen Bioméd*, p. 1–5, 2001.
- WEISSTEIN, E. W. Hurwitz zeta function. Wolfram Research, Inc., 2002.
- ZANI, S. L. Funções de uma variável complexa. 2011.