UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

André Lucas Nascimento Gomes

Extensão de Síntese de Imagens no Mosaicode para Arte Digital

São João del-Rei

2019

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

André Lucas Nascimento Gomes

Extensão de Síntese de Imagens no Mosaicode para Arte Digital

Monografia apresentada como requisito da disciplina de Projeto Orientado em Computação do Curso de Bacharelado em Ciência da Computação da UFSJ.

Orientador: Flávio Luiz Schiavoni

Universidade Federal de São João del-Rei – UFSJ

Bacharelado em Ciência da Computação

São João del-Rei

2019

André Lucas Nascimento Gomes

Extensão de Síntese de Imagens no Mosaicode para Arte Digital

Monografia apresentada como requisito da disciplina de Projeto Orientado em Computação do Curso de Bacharelado em Ciência da Computação da UFSJ.

Flávio Luiz Schiavoni
Orientador

Charles Figueredo de Barros
Convidado 1

Diego Roberto Colombo Dias
Convidado 2

Ivana de Vasconcellos Latosinski
Convidado 3

São João del-Rei
2019

Este trabalho é dedicado a minha família, que me suportaram durante esta caminhada, gerando um ambiente propicio para que me focasse em atingir meus objetivos.

Agradecimentos

Primeiramente agradeço meus pais, pelo todo tipo de apoio e por acreditarem na minha capacidade e manter o investimento no meu desenvolvimento. Esta força me auxiliou bastante o meu crescimento, em que me fez dedicar sem me preocupar com problemas alheios.

Agradeço também aos meus colegas de classe, em que juntos enfrentamos dificuldades, aprendemos com os erros e evoluímos. Que estas amizades construídas aqui possam se estender para a vida toda.

Ao ALICE e aos integrantes do Laboratório 2.00, do qual fiz parte por dois anos, que acelerou meu processo de desenvolvimento, sendo um ambiente onde aprendi me aprimorei, tanto como programador quanto como pessoa.

Aos professores que fizeram parte da minha jornada, em especial ao orientador Flávio Luiz Schiavoni, pelos ensinamentos, paciência e por me guiarem durante o meu período na Universidade.

Aos meus amigos, mesmo aqueles que residem em regiões distantes, que compartilharam comigo minhas felicidades, angustias, momentos de luta e momentos de glória.

A UFSJ e a cidade de São João del-Rei, que me proveram uma excelente infraestrutura, auxílio educacional e financeiro e boas experiencias, durante este percurso.

E finalmente a Deus, por me iluminar durante esta caminhada.

"O artista capta a mensagem dos desafios culturais décadas antes que seus impactos e transformações ocorram. O artista então constrói uma arca de noé para que enfrentemos os desafios que estão por vir."

(Marshall McLuhan)

Resumo

O processo de síntese de imagens pode ser realizado a partir de várias ferramentas. Porém, para construir aplicações na área das artes digitais, o artista pode esbarrar em dificuldades em relação a lógicas de programação, ou a ferramenta utilizada pode possuir uma curva de aprendizado elevada, para que configurações específicas sejam realizadas. Neste cenário surge o Mosaicode, um ambiente de programação visual com o foco no domínio da arte digital, onde as criações podem ser exportadas em forma de diagrama ou de código. Levando em consideração esta perspectiva, este trabalho tem como objetivo a descrição da extensão de Síntese de Imagens para o Mosaicode, a qual abstrai conceitos da computação gráfica e de aparatos matemáticos, além de gerar um conjunto de funcionalidades que auxiliariam tanto um artista a criar sua obra, quanto a um programador para aprender sobre este domínio e prototipar suas aplicações.

Palavras-chaves: arte digital. geração de código. computação gráfica. programação visual.

Abstract

The process of image synthesis can be performed from various tools. However, to build applications in the digital arts area, the artist may run into difficulties in relation to programming logics, or the tool used may have a high learning curve for specific settings to be made. In this scenario comes Mosaicode, a visual programming environment focused on the digital art domain, where creations can be exported in diagram or code form. Considering this perspective, this work aims at the description of the Image Synthesis extension for Mosaicode, which abstracts concepts of computer graphics and mathematical apparatus, and generate a set of features that would help both an artist to create your a programmer to learn about this domain and to prototype its applications.

Key-words: digital art. code generation. computer graphics. visual programming.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ALICE	Arts Lab in Interfaces, Computers, and Everything Else;
API	Application Programming Interface;
ARB	Architecture Review Board;
ASIO	Audio Stream Input/Output;
GEM	The Graphics Environment for Multimedia;
GIMP	GNU Image Manipulation Program;
IMD	Instrumento Musical Digital;
MIDI	Musical Instrument Digital Interface;
OpenCV	Open Source Computer Vision Library;
OpenGL	Open Graphics Library;
OSC	Open Sound Control;
STEAM	Science Technology Engineering Agriculture Mathematics;
UFSJ	Universidade Federal de São João del-Rei;
VST	Virtual Studio Technology;
WebGL	Web Graphic Library;

Sumário

1 Introdução

1.1 Contextualização

O surgimento da era tecnológica, a utilização da internet e de potentes *softwares* e *hardwares* permitiu que artistas criassem um grande número de obras de artes baseadas nestas tecnologias (COLSON, 2007). Nestas circunstâncias, surge a arte digital, também conhecida como arte multimídia, arte interativa ou nova mídia arte. Independentemente de sua nominação, este formato de arte se relaciona com diversas áreas de conhecimento, como informática, eletrônica e robótica, gerando artefatos que se baseiam em diferentes mídias como vídeo, som e imagem (TRIBE; JANA; GROSENICK, 2006).

A criação de aplicações voltadas para a arte digital pode envolver diversos componentes e formar um ambiente computacional complexo. Estes componentes podem ser divididos de várias formas, como sua relação com os sentidos do espectador que a obra pretende envolver. Imagens para visão, áudio para audição, atuadores para tato ou a mistura de mais elementos, como vídeo para visão e audição são exemplos de suportes digitais que podem ser utilizados para cativar e envolver o usuário com uma determinada obra de arte.

Outra possibilidade de categorização é a partir do processo de desenvolvimento da obra, ou seja, em que parte da a criação deste componente é utilizado um computador. Como opções desta categoria são: captura, processamento e síntese de dados. Estas classificações também podem ser utilizadas para a confecção e análise de Instrumentos Musicais Digitais (IMD), que será abordado com mais detalhes na seção 2.2. Estes instrumentos podem ser uma forma de encapsular e analisar aplicações voltadas para arte digital interativa.

Neste trabalho, iremos abordar o processo de síntese de imagem voltadas para a Arte Digital. A criação de imagens sintéticas passa por um modelo de objeto representado num plano, onde este modelo é renderizado a partir de valores de vértices e pontos, num ambiente tridimensional. Este objeto pode ser manipulado através de transformações, como rotações, translações, extrusões, escalonamento, entre outros (TAVARES et al., 1995).

A criação de arte digital depende da existência de aplicações para este fim que normalmente são construídas por artistas. Usualmente, estes artistas possuem formação em áreas tradicionais da arte e podem não possuir nenhum contato com algoritmos e lógicas de programação (SCHIAVONI et al., 2018). Também é possível que tais ambientes sejam criados por programadores, que não possuem necessariamente conhecimentos específicos

no campo das artes e que podem acabar por inserir uma infinidade de funcionalidades em um domínio especifico, e exigir do usuário do sistema um alto nível de aprendizagem, resultando em uma grande dificuldade de estes usuários utilizarem estas ferramentas. Para simplificar este processo foram desenvolvidas diversas bibliotecas para linguagens de propósito geral bem como *softwares* ou linguagens de domínio específico (SCHIAVONI; GONçALVES, 2017a). OpenGL, Java3D, WebGL e Vulkan são exemplos de bibliotecas para linguagens de propósito geral conhecidas para criação de imagens sintéticas.

O OpenGL (WOO et al., 1999) é uma biblioteca para a construção de aplicações gráficas e modelagem iniciada em 1992. Ela é mantida pelo consórcio ARB, conselho formado por empresas como Silicon Graphics Inc. (SGI), NVIDIA e Intel, criado para inserir funcionalidades e modificar versões da biblioteca (COHEN; MANSSOUR, 2006). Java 3D é uma API desenvolvida pela Sun Microsystems Inc. para desenvolvimento de aplicações e modelagens tridimensionais na linguagem Java (BICHO et al., 2002). WebGL é uma poderosa API criada especificamente para desenvolvimento de aplicações 3D para Web, escrita em Javascript, fazendo com que seja compatível com navegadores (PARISI, 2012). Vulkan (SINGH, 2016) é considerado o sucessor do OpenGL, criado pela AMD e inspirado na API Mantle, esta biblioteca foi feita para a linguagem C e possui como vantagem a utilização dos recentes avanços dos processadores para gerar ambientes gráficos de alta performance em várias plataformas.

Exemplos de *softwares* para a criação de imagens no contexto da arte digital são o Pure Data (PUCKETTE et al., 1997), desenvolvido inicialmente para computação musical que pode ser estendido para outros domínios e Eyesweb (CAMURRI et al., 2000), que é um *software* para geração de aplicações para captura de movimentos e gestos. Existem também ferramentas de desenho auxiliado por computadores, como o AutoCAD[1], de edição gráfica de imagens tipo raster, Photoshop[2] e GIMP[3] para o processamento de imagens e Ilustrator[4] e o Inkscape[5] para edição vetorial.

Entre os softwares de programação para o domínio da Arte Digital e criação de imagens temos o **Mosaicode** [6]. Desenvolvido e mantido pelo **ALICE** - *Arts Lab in Interfaces, Computers, and Everything Else* - do Departamento de Computação da UFSJ. Esta ferramenta é um ambiente de programação Visual de código aberto voltado para a criação de arte digital cuja programação é feita por meio de blocos e conexões. Entre suas vantagens estão a prototipação rápida de aplicações para este domínio e a geração, visualização e execução de código criado. Além disto, este ambiente conta com a possibilidade de ser estendido em diversos domínios. O Mosaicode uniu a praticidade de utilizar

[1] Site da ferramenta: <https://www.autodesk.com.br/products/autocad/overview>
[2] Site da ferramenta: <https://www.adobe.com/photoshop>
[3] Site da ferramenta: <https://www.gimp.org/>
[4] Site da ferramenta: <https://www.adobe.com/illustrator>
[5] Site da ferramenta: <https://inkscape.org/pt-br/>
[6] Site da ferramenta: <https://mosaicode.github.io/>

uma ferramenta que fornece uma interface para a programação em uma linguagem de domínio específico com a dinamicidade de usufruir das opções de parametrização de uma biblioteca/API para uma linguagem de propósito geral. A ferramenta possui como objetivo simplificar a criação artística e também auxiliar o ensino de técnicas para esta área. Uma descrição mais detalhada da ferramenta será apresentada na subseção 2.3, onde cada elemento deste ambiente será explicitado.

Existem ambientes de programação visual com a concepção semelhante ao Mosaicode. Estas ferramentas possuem o mesmo objetivo, auxiliar na criação de aplicações no domínio das artes digitais, trabalhando com síntese de imagens. Estas ferramentas serão apresentadas na Seção 2.4.

1.2 Objetivos

O Objetivo deste trabalho é construir uma extensão para o ambiente de programação Mosaicode que permita facilitar a prototipação de síntese de imagens e que possa auxiliar no ensino das técnicas envolvidas nesta síntese, facilitando o ensino desta área e a criação da arte digital. Para isto, será utilizado como base para a construção desta extensão a biblioteca OpenGL e a linguagem de programação C. Este trabalho tem ainda, como objetivos específicos:

- Geração de Exemplos de síntese de imagens em OpenGL:
  - partindo das funcionalidades da biblioteca;
  - partindo de conceitos matemáticos que podem ser utilizados para criações em arte digital;
- Criação de blocos:
  - referentes aos exemplos criados;
  - que auxiliam a criação e manipulação das funcionalidades implementadas nos exemplos.

1.3 Cenário e Justificativa

Este trabalho possui como base para sua justificativa a criação do **Grupo de Estudos STEAM** (Ciência, Tecnologia, Engenharia, Arte e Matemática - do inglês - Science, Technology, Engineering, Arts and Mathematics)(CATTERALL, 2017), em agosto de 2018 na UFSJ. Neste grupo, foram levantadas diversas questões sobre a utilização de matemática para desenvolvimento de Arte Digital em diversas mídias utilizando métodos como Síntese e Processamento de Som e Imagem, Fotografia e Arquitetura.

Para o caso específico da síntese de imagem, a criação de uma extensão para o ambiente Mosaicode que atende a este requisito aliaria os desejos do grupo de estudos de facilitar a criação de arte digital por pessoas interessadas nesta área com menos esforço e pouco conhecimento de programação, ampliando o leque de funcionalidades deste ambiente de programação. O potencial de criação da extensão, aliado com outras já existentes, tornaria o Mosaicode um ambiente mais completo para o desenvolvimento de arte digital, encapsulado em uma única ferramenta.

Este trabalho também se relaciona com o campo da arte digital buscando prover uma opção tecnológica, a partir de requisições dos artistas, de soluções de síntese de imagens sob demanda. O conhecimento adquirido como consequência destas aplicações geradas são traduzidos, se possível, em blocos da ferramenta Mosaicode, com a intenção de tornar a construção deste programa mais acessível ao público-alvo.

1.4 Metodologia

O primeiro passo para o desenvolvimento da extensão de Síntese de Imagens para o ambiente Mosaicode foi a escolha de uma biblioteca que atenda as necessidades levantadas para esta tarefa. Na seção 3.1 será apresentada a biblioteca escolhida e também os motivos desta escolha. Após isto, criou-se protótipos de aplicações utilizando esta biblioteca com o objetivo de exercitar conceitos dela e identificar padrões de escrita de códigos nestes exemplos.

O terceiro passo foi identificar um padrão de código nestas aplicações. Um padrão de código é um modelo de programa que possui as características básicas que todo código escrito com esta biblioteca costuma possuir. Também foi necessário definir neste modelo as seções de código que são diferentes em cada exemplo. Outra definição fundamental para construção da extensão foi definir o script de compilação e execução do algoritmo feito utilizando este padrão. Por fim, blocos foram construídos a partir dos conceitos estudados na etapa de prototipação de exemplos.

A evolução desta extensão necessitou um desenvolvimento em 4 etapas, como mostrado no fluxograma da figura 1. A primeira é o estudo de modelos matemáticos, propostos pelo grupo de estudos de Arte Digital, citado na seção 1.3, que possam ser relacionadas com as Artes. Inicialmente, apareceram sugestões de utilização de Simetrias, Sequência de Fibonacci e geometria não-euclidiana, como Diagrama de Voronoi (CHEW; III, 1985) e Fractais (REDIES; HASENSTEIN; DENZLER, 2007) para síntese gráfica de aplicações 3D, provenientes de sugestões levantadas durante os encontros do Grupo de Estudos de Arte Digital. Uma introdução destes conceitos matemáticos bem como os exemplos utilizados para prototipação serão apresentados na seção 2.5.

A Segunda etapa foi a criação de protótipos, implementando códigos relacionados

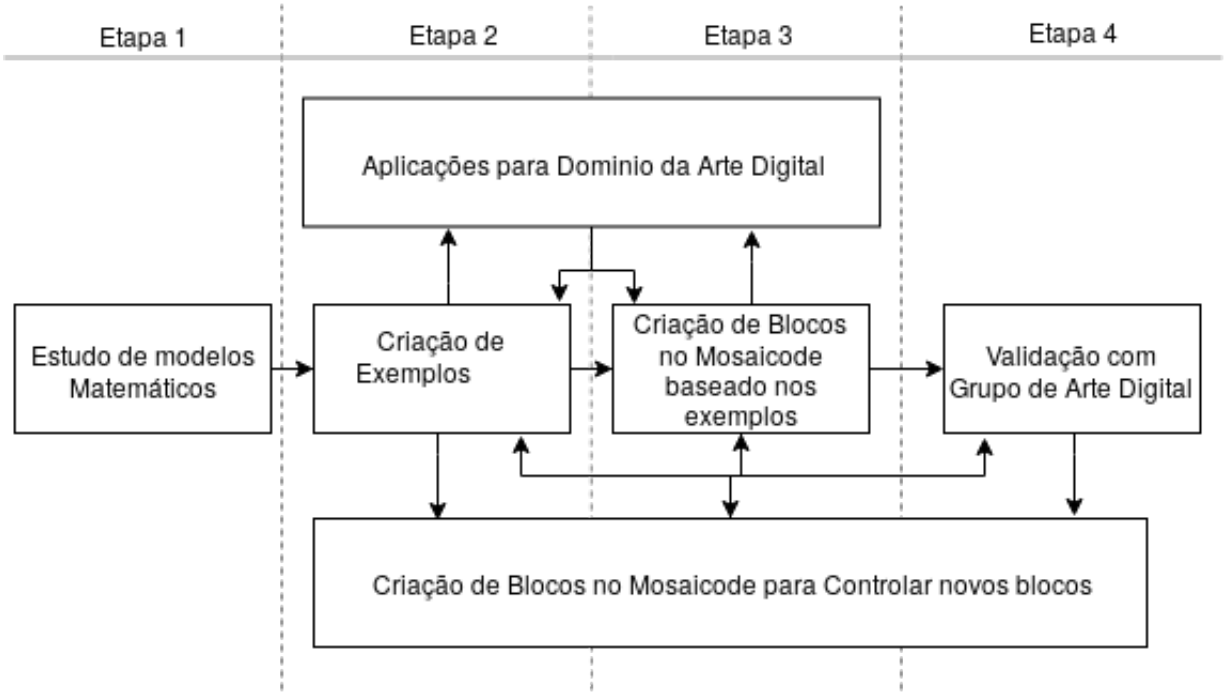


Figura 1 – Fluxograma do processo de criação de um bloco

aos modelos citados, utilizando a biblioteca escolhida, visando aplicar os conceitos matemáticos para síntese de imagens. Estes exemplos criados foram construídos de modo que adequassem à estrutura exigida pela extensão. Em paralelo a isso, para auxiliar no desenvolvimento dos protótipos, foram criadas aplicações artísticas, visando a experimentação destes conceitos.

No próximo passo, foi necessária a integração desses modelos no ambiente Mosaicode, criando as classes referentes aos blocos, explorando diversas maneiras de utilizar as ferramentas matemáticas de modo que se tornem uma ação atômica, visando simplificar a prototipação no ambiente.

Juntas, as três últimas etapas levaram à implementação de blocos que auxiliam na manipulação dos valores e dos dados para a criação em arte digital. Dentre eles, blocos para dispositivos de entrada e saída, como teclado e mouse, e que realizam alteração de valores das conexões existentes.

1.5 Organização do Trabalho

Este trabalho está dividido desta forma: no Capítulo 2 iremos apresentar definições importantes para o entendimento do trabalho, como contextualização sobre o que é arte digital e instrumentos digitais, mostrar as funcionalidades do Mosaicode, ambiente de programação que utilizaremos, e a biblioteca que utilizaremos para criação da extensão de síntese de imagens, e conceitos matemáticos que utilizamos na extensão.

No Capítulo 3, abordaremos o desenvolvimento do trabalho, detalhando escolhas de implementação dos protótipos feitos. O Capítulo 4 traz os artefatos produzidos pela pesquisa, desde a extensão até aplicações e códigos criados durante o processo de desenvolvimento. Para finalizar, o Capítulo 5 detalha as conclusões retiradas do trabalho e detalha possíveis problemas futuros que podem ser resolvidos com o progresso da pesquisa.

2 Referencial Teórico

Este capitulo apresenta os conceitos e definições importantes que guiaram este trabalho cuja explicação é necessária para o entendimento do trabalho como um todo. Iniciaremos contextualizando o leitor sobre arte digital e algumas das suas diversas categorias. Após isto, será debatida a definição de instrumento digital e onde a extensão criada se encaixa com a pesquisa deste trabalho. Depois, falaremos sobre o Mosaicode e suas funcionalidades. No final, apresentaremos um material teórico sobre os elementos matemáticos utilizados para criação dos blocos mais complexos.

2.1 Arte Digital

A arte digital pode ser caracterizada como a mistura de ciência, arte e tecnologia, utilizada para produção de conteúdo utilizando conceitos como imersão, interatividade e virtualização em projetos interdisciplinares, aproveitando de interfaces digitais (GASPARETTO, 2016), que usufrui destes recursos para desenvolvimento de peças, instalações, apresentações e aplicações. Este tipo de arte permite um processo criativo em que o artista pode contar com a colaboração de uma equipe técnica diversificada, desenvolvendo conhecimento e tecnologia em distintas áreas, gerando como um resultado final um conteúdo de vários formatos de mídia (MARCOS, 2012). Como resultado deste processo, é possível alcançar sistemas híbridos em que haja comunicação entre a obra, o ambiente em que a obra esteja sendo exibida junto ao meio digital em que ela esteja sendo gerada e o iterador, que pode alterar ou até se sentir parte deste sistema com suas ações (GASPARETTO et al., 2012).

Em sistemas de arte digital interativa, a possibilidade de interagir com a obra passa a não ser apenas atividade do artista, que produziu e na maioria dos casos interpreta a apresentação, mas também uma tarefa do espectador, que passa a participar ativamente auxiliando no processo de criação, trazendo novas maneiras tanto de desenvolver quanto de experimentar a obra, diferente de alguns outros campos da arte. Este processo utiliza possibilidades que só são viáveis utilizando o computador como ferramenta para expressar conteúdo, trazendo novas realidades e sensações ao público envolvido (GASPARETTO et al., 2012). **Imersão**, **Interatividade** e **Virtualização** são características chaves para este paradigma.

Imersão é a ideia de estar envolvido com algum ambiente do qual não está inserido no contexto atual (NETTO; MACHADO; OLIVEIRA, 2002), sendo assim objetivo de muitas áreas de pesquisa, como literatura, cinema, jogos e arte digital (OLIVEIRA, 2013). O envolvimento do espectador está relacionado aos sentidos da visão, audição e motor,

que estimulam a ilusão de experimentar outras sensações (BORBA, 2014).

A interatividade possui uma importância no estudo da comunicação via computador, com aplicação na educação a distância, engenharia de software e interação humano-computador. Há diversos conceitos de interação, como interação social na sociologia, que estuda as sociedades e suas relações ou na física, onde toda interação da matéria ocorre a partir da gravidade, eletromagnetismo e a força nuclear (PRIMO, 1999). Um sistema interativo pode prover uma resposta autônoma, criativa e não prevista. Ele deve prover ao seu espectador ou usuário total autonomia, sendo assim, tornando todos os envolvidos com a obra um agente ativo (PRIMO, 1998).

A virtualização nos remete ao conceito de virtual. A definição de virtual não é o contrário de real, mas a de criar elementos que alteram a realidade, o que difere de criações imaginárias, pois possuem um diálogo com a realidade (JUNGBLUT, 2004). A construção do virtual passa a se tornar real na medida que o usuário tem a sensação da criação sintética. Virtual está mais para oposto de atual (LÉVY, 2003), sendo que o atual é o que já está construído, determinado, fixo. Então, a definição de virtual refere-se ao desprendimento do imediato, do conceito do atual (CARISSIMI, 2008).

Com um significado amplo, a arte digital abrange diversas formas, mídias e formatos, como instalações, *web art*, realidade virtual, aumentada e mista. Além disso, ela é tema de discussões em diversas áreas como inteligência artificial, jogos e ambientes narrativos e hacktivismo. As categorias feitas com as formas e temas podem se tornar problemáticas, pois elas se sobrepõem e se misturam, criando novos tipos de conteúdos ainda inseridos no contexto da arte digital (PAUL, 2003).

Instalações em arte digital podem ser provenientes de instalações de vídeos que incorporam em suas projeções captura de imagens em tempo real, ou sistemas com a intenção de se tornar ambientes imersivos e interativos harmonizados em um âmbito virtual (PAUL, 2003). *Web art* é uma definição onde os criadores de conteúdo não utilizam a web apenas como meio de visualização e distribuição, mas para criar arte utilizando ambientes e elementos disponíveis na web. Ela é uma arte fluida, que busca modificar a maneira em que o usuário navega na página, fazendo que cada pessoa veja uma obra diferente de acordo com sua experiência na página (WEINTRAUB, 1997). Um exemplo de página feita exclusivamente para *Web Art* está representada na Figura 2. Diferente de outras interfaces computacionais, que são limitadas ao espaço bidimensional, a realidade virtual, aumentada, mista e outras variações exploram um ambiente tridimensional, onde provêm ao usuário uma ação multissensorial, manipulando sentidos como audição, visão e tato (KIRNER; KIRNER, 2011).

Com o crescimento da utilização no cotidiano de agentes inteligentes para comunicação e manejamento de informação, e a expansão do uso de sensores para IoT (Internet das Coisas), a inteligência artificial é saudada como artifício para facilitar nossas vidas,

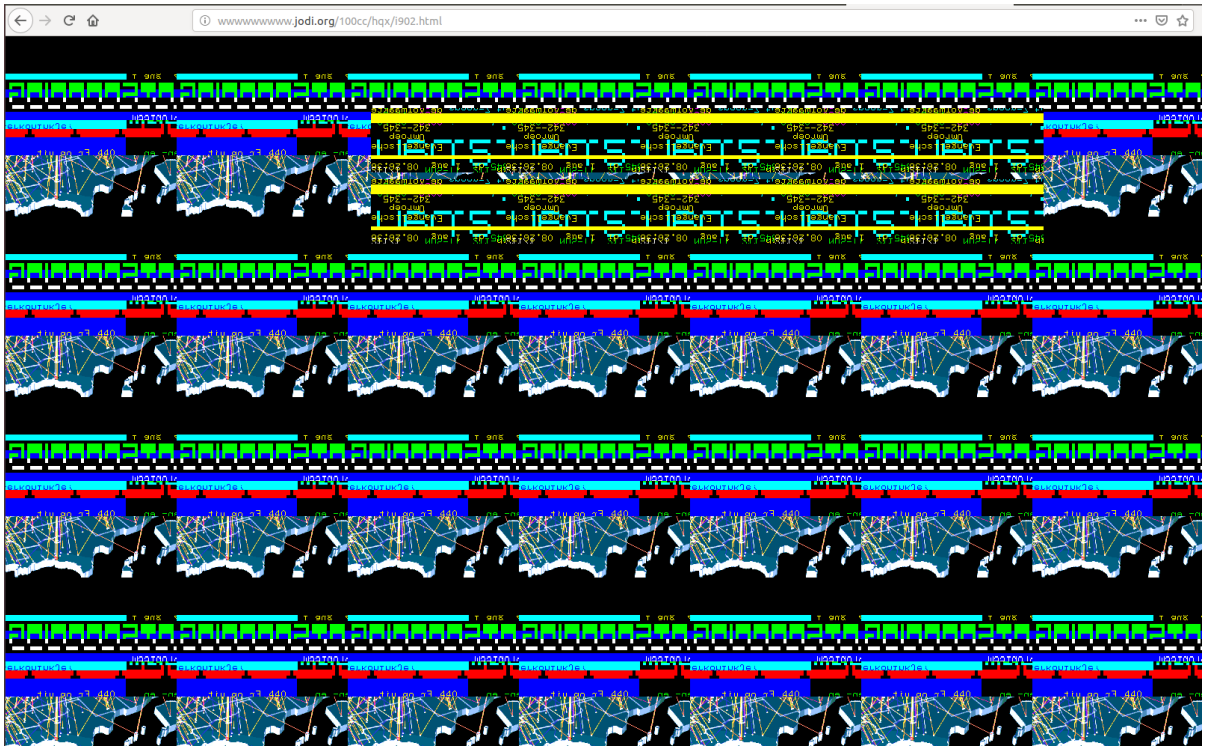


Figura 2 – Página web redirecionada através do site jodi.org. A cada acesso, a página encaminha para o navegador uma aplicação de *Web Art*. Página: <wwwwwwwww.jodi.org/100cc/hqx/i902.html>. Data de Acesso: 12:56, 15/04/2019

bem como artifícios que retiram nossa privacidade, dependendo do ponto de vista (PAUL, 2003). Utilizando estas ferramentas, é possível criar no contexto artístico ambientes dos quais o usuário interaja com a máquina que, por meio de seus sensores, tomam decisões criando obras mutáveis a cada vez que apresentadas ou utilizadas. Nos jogos, são explorados paradigmas e problemas semelhantes à arte digital, como a criação de ambientes virtuais 3D, simulação e exploração de narrativas e sistemas multi jogadores (PAUL, 2003).

Hacktivismo é uma palavra utilizada para empregar ativismo político por via do *hacking*. É a utilização de atividades *hackers* em prol ou resistência de uma causa (BARROS, 2013), diferente do ciberterrorismo que, por exemplo, que tem como características buscar violência ou ameaça de violência para geração de medo, realizando atos criminosos e imorais de natureza política e religiosa, através de meio digitais (NOVAIS, 2012). O hacktivismo pode ter uma ligação muito forte com a arte digital que tem dificuldade de ser comercializada e que quebra o paradigma de que a arte é feita apenas por gênios, dando a possibilidade a qualquer um com acesso a computador a criar obras, dando a oportunidade a alguns artistas utilizarem ideais do ativismo político junto com habilidades de *hacking* para contestar a realidade de ambientes como redes sociais e internet, causando estranheza e desconforto ao espectador (ARISAWA, 2011). Na figura 3 mostra uma intervenção artística feita utilizando arte digital.

Figura 3 – Intervenção pública de Alfredo Jaar nomeada ***A Logo for America***, exposta em Piccadilly Circus, London em 2016. Fonte: <http://digicult.it/hacktivism/alfredo-jaar-a-logo-for-america/>. Data de Acesso: 13:37, 15/04/2019

2.2 Instrumento Digital

A criação de um instrumento acústico pode possuir um processo criativo que se dá através de várias tentativas e erros. Diferente disto, um instrumento digital se inicia com o desejo de chegar em determinado resultado (ROCHA et al., 2018). A estrutura de um instrumento digital pode ser representada por um conjunto de três módulos: um dispositivo de interação / interface do usuário, um aparato que sintetiza um resultado e um mapeamento que interliga e controla os dois anteriores.

Interação ou Interface: O equipamento de interação do usuário captura os movimentos do seu utilizador / usuário. É interessante que este dispositivo possua uma interface amigável para o instrumentista poder ser expressivo em sua ação artística. Teclado e *mouse* são aparatos convencionais que podem realizar esta tarefa mas que possuem pouca expressividade. Outros exemplos são a captura de gestos via câmera, dispositivos MIDI, *joysticks*, sensores disponíveis em *smartphones* como acelerômetro, giroscópio e *touchscreen*, entre outros.

Síntese: O objetivo deste módulo seria sintetizar algum artefato artístico que dê um *feedback* às ações dos usuários (ROCHA et al., 2018). Tipos de retorno possíveis seriam

retornos visuais, sonoros e tácteis, além de uma mistura dos três.

Mapeamento: Este módulo define como o dispositivo de Interação se interligará e controlará o de Síntese. Esta ligação definirá como o instrumento se comportará, pois ele que configurará o sentido das ações do movimento de usuário para gerar o artefato.

Este trabalho se encaixa tanto na parte de Síntese, pois ele gera código para uma biblioteca de Síntese de Imagens, como na interação do usuário, dado que a ferramenta fornece módulo de Interface com o usuário, quanto no Mapeamento, pois a programação visual facilita ao artista realizar esta interligação e controle entre interface e síntese.

Para auxiliar na tarefa de interação do artista com a aplicação, foram criados alguns blocos de Interação, como blocos de mouse e teclado, embora não seja o foco da extensão, como seria numa extensão de Captura e Processamento de Imagens. Com estes elementos, é possível pensar nesta extensão como um módulo de criação capaz de desenvolver um instrumento completo, apesar das limitações de opções no módulo de Interação.

A construção da aplicação como um instrumento digital poderia facilitar a distribuição e na utilização de várias pessoas. Estes dispositivos fornecem uma interface mais amigável, além de ser mais simples de controlar e retornar um feedback previsto.

2.3 Mosaicode

O Mosaicode (SCHIAVONI et al., 2018; SCHIAVONI; GONçALVES, 2017a; GOMES; SCHIAVONI, 2018; GOMES; RESENDE; SCHIAVONI, 2018; SCHIAVONI; GONçALVES; GOMES, 2017; GONcALVES, 2017; SCHIAVONI; GONçALVES, 2017c; SCHIAVONI; GONçALVES, 2017b) é um ambiente de programação visual que permite a criação de aplicações por meio da conexão entre **blocos**. A ferramenta fornece diversas funcionalidades para criação artística em diferentes linguagens de programação. Neste ambiente, o código na linguagem em que os blocos foram escritos ficam disponíveis para eventuais modificações, tornando o ambiente interessante para prototipações de programas, que podem ser otimizados fora do ambiente.

No Mosaicode, os blocos são unidades minímas de código que executam uma determinada funcionalidade no sistema. Esta funcionalidade é uma atividade atômica específica e sua estrutura é formada por linhas de código referentes às aplicações desenvolvidas para um domínio específico normalmente utilizando comandos de uma biblioteca.

Um bloco pode ser configurado e seu estado pode ser modificado de duas maneiras: estática e dinâmica. As propriedades estáticas são campos personalizados para cada bloco, que são especificados no momento de criação do bloco, pelo desenvolvedor. Eles alteram o valor destes campos em tempo de programação, ou seja, no momento em que o código

é gerado. As propriedades dinâmicas são conhecidas também como portas, que podem ser de entrada ou de saída, e recebem ou enviam dados de um tipo selecionado pelo desenvolvedor. Elas possuem um tipo de dado que pode ser transferido e modificam a aplicação em tempo de execução.

Os blocos podem ser interligados por **conexões**, que são ligações de código que transferem dados de um determinado tipo entre portas de um bloco de saída para um bloco de entrada. De acordo com o tipo da porta, elas podem ou não conter linhas de códigos referentes ao deslocamento de dados. Elas também definem a hierarquia em que os blocos são inseridos para geração de código fonte final, sendo o primeiro bloco do fluxo de ligações posicionado antes do segundo bloco no fluxo, no código gerado.

Um conjunto de blocos e conexões formam um diagrama. Sua representação é apresentada na tela do Mosaicode. Com ele, é gerado um artefato, onde é possível salvar estado do diagrama em um arquivo com uma extensão própria (**.mscd**), que armazena as informações dos blocos, como posição no canvas, valores das propriedades, conexões e outros dados.

A geração de código de um diagrama é feita por meio de um **padrão de código**. Um padrão de código é um modelo de um algoritmo genérico, que define quais trechos de código serão mantidos em qualquer programa gerado pela ferramenta, e seções que determinam em que lugares serão inseridos os códigos provenientes do diagrama, conhecidas também como **rótulos**. Estes rótulos podem possuir dois tipos: *code* ou *single code*. Trechos de código escritos em com o rótulo *code* são inseridos no código final gerado a cada vez que um mesmo bloco é colocado no diagrama. Já rótulos *single code*, inserem trechos de código uma única vez, mesmo que forem inseridos vários blocos iguais no diagrama. Por exemplo, ao criar um bloco, deve ser inserido o rótulo junto ao trecho de código no algoritmo, então quando este bloco for inserido no diagrama, este código será colocado onde o rótulo indicar no diagrama. Um exemplo de padrão de código é mostrado na figura 4.

Blocos, Conexões e Padrão de código juntos formam uma extensão, que é um componente que representa um domínio, e é geralmente escrita utilizando uma linguagem de programação específica, utilizando bibliotecas e API's. As extensões podem ser instaladas no Mosaicode para adicionar um novo domínio que pode ser trabalhado na ferramenta. O ambiente de programação sozinho não faz tarefa nenhuma, é preciso ao menos uma extensão para que a ferramenta funcione.

Além da extensão de Síntese de Imagens, que é o foco deste trabalho, atualmente o Mosaicode possui extensões de WebArt em Javascript, de Captura e Processamento de Imagens em C++ utilizando OpenCV (PULLI et al., 2012) e Captura, Síntese e Proces-

```
#ifdef _CH_
#pragma package <opencv>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/opencv.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/objdetect.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/features2d/features2d.hpp"
<<single_code - include>>
using namespace cv;
using namespace std;

#define FRAMERATE 1000.0 / 25.0
<<single_code - function>>
int main(int argc, char ** argv){
   char key = ' ';
   <<code - declaration>>
   while((key = (char)waitKey(FRAMERATE)) != 27){
      <<code - execution, connection>>
      <<code - deallocation>>
   }

   destroyAllWindows();
   return 0;
}
```

Figura 4 – Pseudocódigo do padrão de código da Extensão de Síntese de Imagens em C++/OpenGL

samento de áudio em C++/PortAudio[1].

Na Figura 5 é exibido o ambiente de programação Mosaicode e sua Extensão de Webart em Javascript, com um exemplo de diagrama. No campo **1** são localizadas as extensões, suas categorias e seus respectivos blocos. No campo **2** é mostrada uma representação visual do diagrama da aplicação, com seus blocos, portas e conexões. O

[1] Site do Projeto: http://portaudio.com/

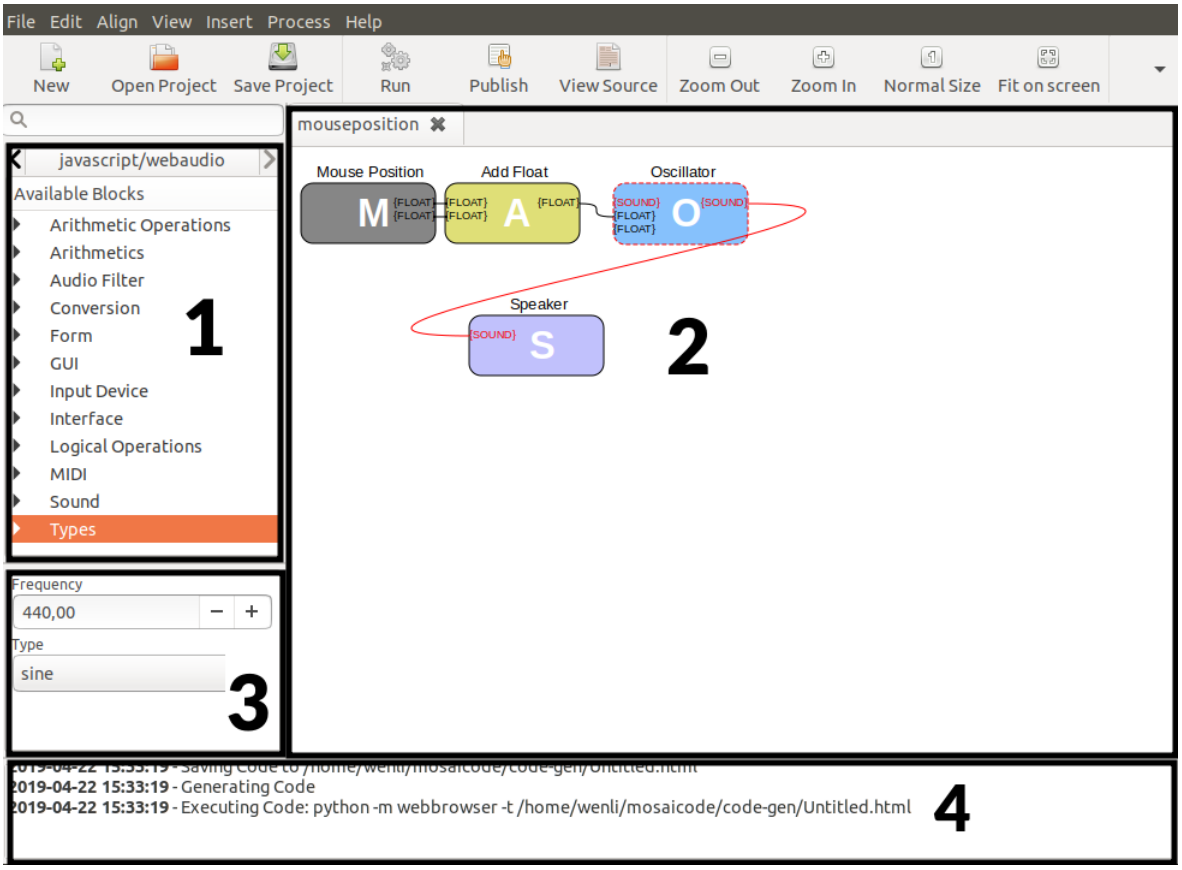


Figura 5 – Ambiente de Programação Mosaicode com a Extensão de Webart em Javascript

campo **3** representa as propriedades do bloco *Oscillator*, que está selecionado no canvas. O campo **4** mostra o terminal, indicando as ações realizadas no ambiente, bem como a linha de código para execução do código gerado.

Na barra de tarefas, os três primeiros botões são referentes a criação, abertura e salvar um diagrama feito na tela. À direita, o botão **Run** de execução da aplicação, o botão **Publish** para disponibilizar a aplicação num servidor, e o botão **View Source** para mostrar o código gerado a partir do diagrama. Após isto, a barra possui botões de controle de zoom e alinhamento da tela.

2.4 Trabalhos Relacionados

Nesta seção falaremos sobre os trabalhos relacionados ao Mosaicode. Estes trabalhos são ferramentas que atuam na área de arte digital e possuem plugins ou componentes que realizam síntese de imagens.

2.4.1 GEM/Pure Data

Pure Data (PUCKETTE et al., 1997) é um ambiente de computação musical com programação visual lançado em 1996 por Miller Puckette para corrigir problemas do

Max, programa criado pelo mesmo desenvolvedor e antecessor do Max/MSP, sob a licença BSD. Além da síntese e processamento de áudio, ele realiza comunicação com sensores de interface, dispositivos de entrada e MIDI, e permite facilmente trabalhar em redes locais e remotas.

O Pure Data é OpenSource, com uma comunidade ativa e é extensível, a partir de objetos (externals) ou módulos (patches), e funciona em diversas plataformas, como Windows, GNU/Linux e MacOS.

A GEM (DANKS, 1997) é uma plataforma gráfica independente desenvolvida em 1996 pelo Silicon Graphics, Inc. (SGI), escrita em OpenGL, originalmente criada para Max/MSP e utilizada no Pd. Esta biblioteca executa manipulação de polígonos, além de processamento em tempo real de imagens e vídeos, usando a licença GNU. A programação visual do Pure Data junto com as funcionalidades do GEM possibilita que não-programadores gerem gráficos e áudios complexos simultaneamente com uma maior facilidade.

2.4.2 EyesWeb

O EyesWeb (CAMURRI et al., 2000) foi desenvolvido na universidade de Genova em 1997 sob a licença da própria universidade e é um ambiente de programação visual para análise em tempo real de gestos e movimentos corporais. A partir desses movimentos, pode se gerar áudio, imagens ou controlar alguns atuadores. O objetivo do projeto é desenvolver modelos de interação utilizando uma linguagem corporal para se expressar de modo visual ou musical.

Possui versões para Windows, Linux e Dispositivos móveis, além de permitir interligações com diversos dispositivos de entrada e saída, e suportar diversos formatos e protocolos como OSC, MIDI, FreeFrame e VST plugins, ASIO.

2.4.3 Isadora

Com objetivo de atender artistas e *designers*, Isadora[2] é um ambiente de desenvolvimento que facilita a iteração entre vídeos e mídias iterativas com performers. Este ambiente combina um servidor de mídia, programação visual e um mecanismo de processamento de áudio e imagem em tempo real. Com comunidade ativa e uma plataforma que permite a utilização da criatividade para improvisação e experimentação, diversas apresentações foram feitas utilizando a ferramenta como apoio técnico.

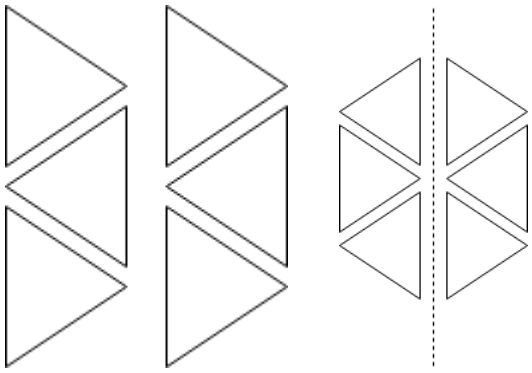
[2] Site do Projeto: https://troikatronix.com

2.5 A Intersecção entre a Matemática e a Arte

Nesta seção, iremos abordar as definições matemáticas utilizadas nos blocos construídos para a ferramenta. Utilizamos três principais conceitos para nortear este trabalho: a concepção do que é simetria, a ideia de fractais e geometria não-euclidiana e a noção de o que é o diagrama de Voronoi.

2.5.1 Simetria

Simetria é uma operação em que é definido um padrão invariante, replicado diversas vezes a partir de uma fórmula. Ela pode ser definida ou por representações numéricas, tentando descrever eventos da Natureza, ou por uma representação não é avaliada através do rigor científico, buscando se definir através de aspectos visuais (ROHDE, 1997). As transformações simétricas podem ser feitas de diversos tipos: translação, rotação, inversão, dilatação e a combinação das simetrias anteriores.



(a) Translação (b) Reflexão

Figura 6 – Operações de Simetria

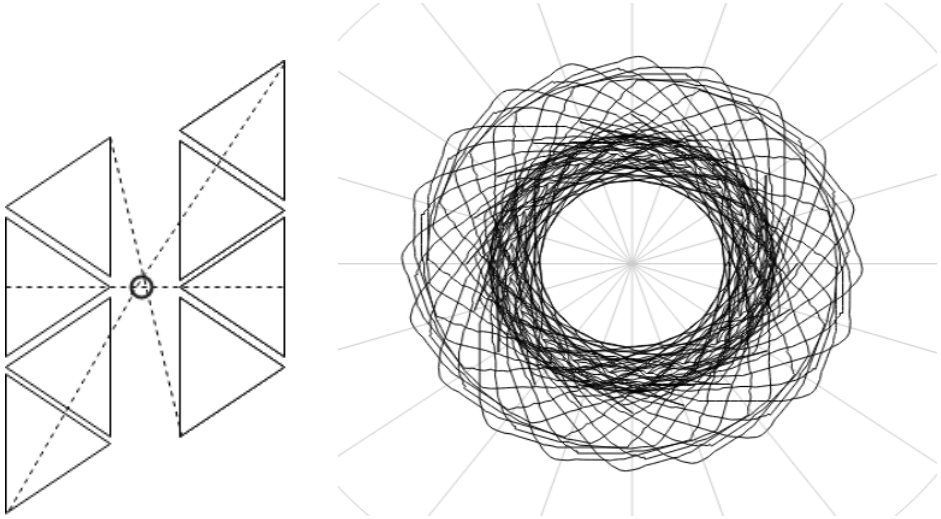
Translação: Esta simetria é resumida no deslocamento de uma imagem, por uma direção, causando repetições, como representada na Figura 6a. É um tipo de simetria simples, facilmente localizado no ambiente, como pegadas na areia formam uma simetria de translação (RIPPLINGER, 2006).

Reflexão: Simetria ocasionada ao espelhar uma imagem. Ao passar uma linha imaginária no ponto de reflexão, e juntar uma imagem sobre a outra, as duas devem ser idênticas, como mostrado na figura 6b.

Rotação: Conhecido também como simetria cíclica e simetria rotatória. É a simetria ocasionada ao rotacionar a cópia da imagem por algum valor entre 0º e 360º, por 'n' vezes. Um exemplo famoso de simetria de rotação é a mandala, como exibido na figura 7b.

Inversão: A simetria de inversão é ocasionada a partir de uma linha ou um ponto imaginário, onde tudo de um lado é o inverso do outro lado. Um exemplo está mostrado na figura 7a.

Dilatação: Nesta simetria, a imagem se mantém, porém a forma dilata ou se contrai sem alterar as proporções iniciais.



(a) Inversão (b) Mandala é um exemplo de simetria de rotação

Figura 7 – Operações de Simetria

2.5.2 Fractais

A geometria fractal surgiu para descrever formas na natureza em que elementos da geometria euclidiana, como esferas, cubos, planos e curvas, não conseguem explicar. Apresentada por Mandelbrot em 1975 (MANDELBROT, 1975), a teoria fractal possui três elementos importantes para serem explicitados: a dimensão, a auto-semelhança e a complexidade infinita (ASSIS et al., 2008).

- **Dimensão:** A matemática que utilizamos no dia-a-dia, conhecida como matemática euclidiana, descreve a dimensão por um número natural. Neste conceito de dimensão, podemos retratar a primeira dimensão com uma reta, a segunda dimensão por um plano, e a terceira dimensão por um espaço. Porém, fractais não são representados na geometria euclidiana. Para fractais, surge o conceito de dimensão de Hausdorff-Besicovitch. Nesta definição de dimensão, ela pode assumir valores fracionários. Tendo em vista estas ideias, a definição de dimensão fractal está atrelada a um valor fracionário para a dimensão, descrevendo que conforme a alteração do comprimento entre dois pontos aumenta, a medida em que sua escala diminui (CRUZ, 2018).

Dimensão de Homotetia: para auxiliar o entendimento de dimensão fractal, utilizaremos o conceito da dimensão de Homotetia. Homotetia seria uma operação em que um objeto se multiplica por um valor constante a distância de qualquer valor fixo, deslocando a partir de duas figuras em relação a um ponto inicial (BOGOMOLNY, 2005). Sendo assim, para o cálculo da dimensão D (CRUZ, 2018), é seguido esta fórmula:

$$D = \frac{\log N}{\log \frac{L}{U}}$$

Onde:

- **D:** Dimensão na qual a Homotetia está inserida;
- **N:** Número de partes da qual o objeto foi dividido;
- **L:** Comprimento do objeto inicial antes da divisão;
- **U:** Comprimento de uma parte da divisão do objeto;

Este valor é calculado a partir com o desejo de se ampliar um objeto a partir de uma escala k = L/U, obtendo um número de N de objetos. Seja um quadrado, para duplicar a escala do quadrado original, o resultado é um quadrado onde esteja dividido em quatro partes. Para triplicar, o quadrado final terá em sua divisão nove vezes partes, como mostrado na figura 8.

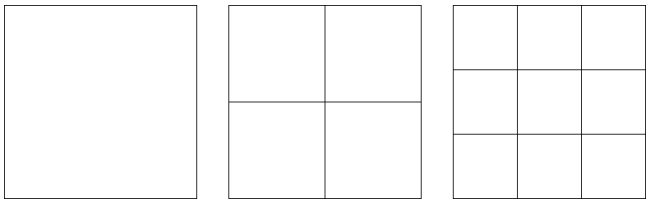


Figura 8 – Exemplo para Dimensão de Homotetia

A dimensão de homotetia obtém resultados fracionários quando são aplicados sob fractais com auto-semelhança perfeita, onde é possivel realizar este calculo. Se são aplicados sob formas da geometria euclidiana, o resultado da dimensão será em inteiro.

- **Auto-semelhança:** Esta definição está atrelada a quando um objeto, mesmo ao aumentar ou diminui sua escala, manter sua forma e seu aspecto. Isto é possível quando uma figura possui uma regra de iteração, ou seja, uma repetição recursiva de um ato (NUNES, 2006).

- **Complexidade Infinita:** A criação de um fractal possui uma complexidade infinita pois não teria um final para a recursão de geração do objeto, pois seria um número

infinito de repetições de um mesmo procedimento, para que ele seja auto-semelhante sob qualquer circunstância (ASSIS et al., 2008).

Abaixo segue como exemplos alguns fractais, utilizados neste trabalho:

Conjunto de Cantor: Conhecido também como "Poeira de Cantor", este conjunto foi descrito pelo matemático George Cantor. A partir de um segmento de reta, o mesmo é subdividido em três partes de mesmo tamanho, e destes segmentos, o localizado no centro é retirado. Após isto, os passos citados são repetidos nos segmentos de retas remanescentes, como mostrado na figura 9.

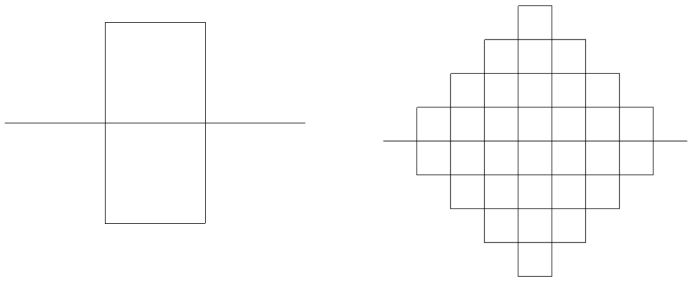
Figura 9 – Conjunto de Cantor, após seis iterações

Seguindo a fórmula para o cálculo da Dimensão Fractal, onde U são um terço do comprimento inicial, L é o comprimento inicial, e o N é dois, referente à quantidade de objetos criados a partir de uma iteração, a dimensão é aproximadamente 0,63.

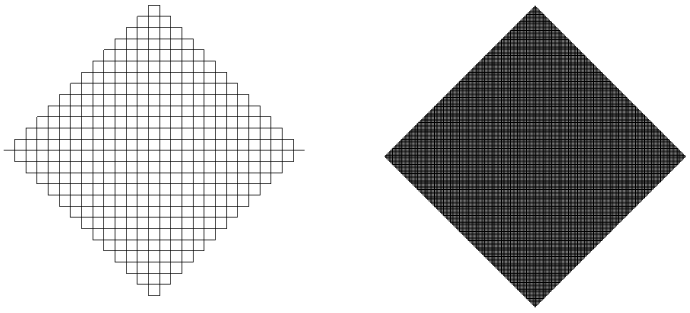
Curva de Peano: Foi exposta por Gieseppe Peano, matemático professor da Universidade de Turim, entre o fim do século XIX até as primeiras décadas do século XX. A curva apresentada por ele é construída a partir de um segmento de reta, que é separado em três partes iguais. Sob o segmento localizado no centro da reta, são construídos dois quadrados, com o lado de mesmo tamanho do segmento, como mostrado da Figura 10.

Para cada parte separada da reta inicial, somada a cada lado dos quadrados gerados, formam um novo segmento de reta, do qual será efetuada uma nova iteração, conforme descrito anteriormente. Utilizando o cálculo da Dimensão Fractal, cada segmento gera 9 segmentos com um terço do tamanho da inicial, fazendo com que a dimensão seja 2. Isto quer dizer que se o número de iterações tende ao infinito, a área será totalmente preenchida, formando um losango (CRUZ, 2018).

Curva de Koch: Criada por Helge von Koch, onde sua construção começa a partir de um segmento de reta dividido em três partes, assim como ocorrido nos fractais anteriores. É removido o segmento central, e em seu lugar, são adicionados outros dois, que formariam um triângulo equilátero caso não fosse retirada a parte central. Para cada iteração, em



(a) 1ª (b) 2º

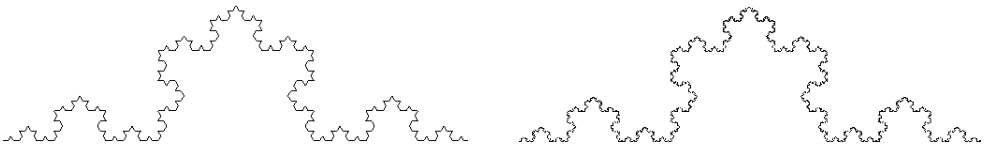


(c) 3º (d) 5º

Figura 10 – Iterações para construção da Curva de Peano

todas as quatro retas resultantes são efetuados os mesmos passos. Várias iterações da Curva de Koch são exibidas na figura 11.

(a) 1ª (b) 2º



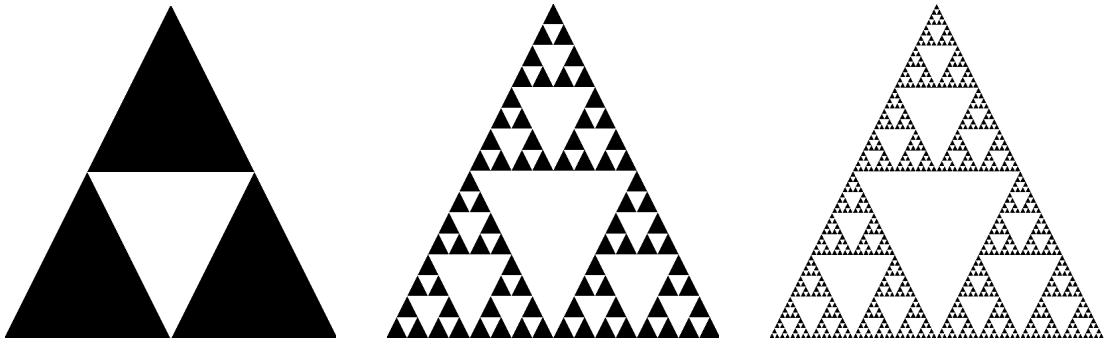
(c) 4º (d) 8º

Figura 11 – Iterações para construção da Curva de Koch

Para cada segmento de reta em que é iterado a Curva de Koch, são gerados quatro novos segmentos com um terço do tamanho do original. Sabendo destes detalhes, a dimensão fractal deste objeto é aproximadamente 1,26.

Triângulo de Sierpinski: È um fractal descrito por Waclaw Sierpinski onde são efetuadas operações a partir de triângulos. Inicia-se com um triângulo de lado unitário, que

é dividido em quatro, com metade do valor original no lado, separados de acordo com a figura 12a. O triângulo centralizado é retirado, e as próximas iterações são feitas com os passos anteriores nos objetos restantes.

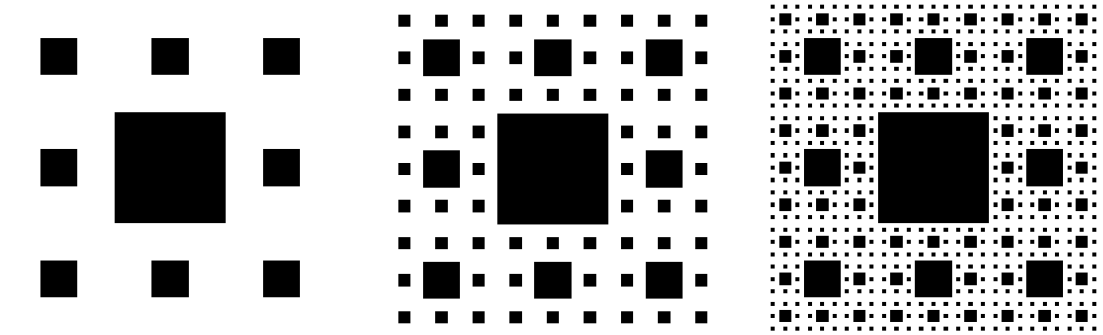


(a) 1ª - Triangulo de Sierpinski (b) 4º - Triangulo de Sierpinski (c) 6º - Triangulo de Sierpinski

Figura 12 – Iterações para construção do Triangulo de Sierpinski

Cada triângulo que sofre uma iteração para este fractal gera três triângulos com metade do comprimento do original, sendo assim, a dimensão fractal dele é próxima de 1,58.

Tapete de Sierpinski: Também exposto por Waclaw Sierpinski, realiza iterações a partir de um quadrado, onde a cada repetição de passos, são criados outros oito quadrados em volta, com um terço de lado, em comparação com o quadrado da iteração passada, como mostrado nas figuras 13a e 13c. Com as informações passadas anteriormente, é possível calcular a dimensão fractal deste objeto, que é aproximadamente 1,89.



(a) 2º - Tapete de Sierpinski (b) 3º - Tapete de Sierpinski (c) 4º - Tapete de Sierpinski

Figura 13 – Iterações para construção do Tapete de Sierpinski

Esponja de Menger: Conhecida também como Esponja de Sierpinski-Menger, é uma adaptação do Tapete de Sierpinski para o ambiente tridimensional. Nele, um cubo original é transformado em vinte e sete cubos de um terço do lado do original. Destes são removidos o central e todos os outros que possuem conexão com as faces deste que foi retirado, totalizando no final vinte cubos restantes. A dimensão deste objeto é próxima a 2,72.

2.5.3 Diagrama de Voronoi

Seja β um plano e S um conjunto de pontos pertencentes ao plano. Segundo (AURENHAMMER, 1991), o Diagrama de Voronoi é um conjunto de regiões, onde cada uma possui uma dominância de um ponto **P** pertencente a **S**, que são pintadas de cores diferentes, como na figura 14. Esta dominância é descoberta calculando a distância euclidiana de cada ponto do plano para cada ponto de **S**. Todos os pontos que possuem a menor distância para o mesmo ponto **P** forma uma região.

Figura 14 – Exemplo para Diagrama de Voronoi

3 Desenvolvimento

Neste capítulo, serão apresentadas as escolhas feitas durante o processo de desenvolvimento do projeto, como a opção de biblioteca utilizada como base para a criação da extensão e os protótipos de aplicações criados com o intuito de aprimorar o entendimento desta API.

3.1 Biblioteca para criação da Extensão: OpenGL

Entre as bibliotecas que permitem a manipulação e criação de imagens sintéticas, escolhemos o OpenGL (WOO et al., 1999) para a criação da extensão de Síntese de Imagens proposta neste trabalho. Esta biblioteca foi desenvolvida com o objetivo de geração de aplicações e rotinas gráficas para modelagem 2D e 3D para diversas plataformas, com a missão de ser rápida e portável (COHEN; MANSSOUR, 2006). Utilizamos esta biblioteca por ser bem conhecida e popular, com uma comunidade extensa e ativa, com bastante aplicações desenvolvidas e desenvolvedores experientes. Outro motivo é que ela pode ser executada desde computadores pessoais até máquinas com alto processamento (COHEN; MANSSOUR, 2006), tornando-a acessível para o maior número possível de usuários. Por fim, outra razão para utiliza-la é que seja a biblioteca padrão empregada nas aulas de Computação Gráfica no curso de Ciência de Computação da UFSJ.

3.2 Exemplos em OpenGL

O início da pesquisa em síntese de imagens envolveu a criação de diversos protótipos de aplicações utilizando a biblioteca escolhida. Os protótipos foram criados visando que, a partir deles, fosse possível a geração de blocos utilizando os conceitos empregados em cada protótipo. Eles também podem ser considerados artefatos da pesquisa, sendo resultado de estudos realizados referentes ao capítulo 2.

Os primeiros exemplos gerados foram funcionalidades básicas da biblioteca: programas que abrem e modificam propriedades das janelas, como nome, tamanho e localização; aplicações que modificam o plano de fundo da imagem ou método para que a tela inteira seja preenchida pelo canvas são exemplos destas funcionalidades.

Protótipos criados com o objetivo de geração de modelos 3D e 2D foram os próximos conceitos a serem estudados. Inicialmente, foram construídas formas que utilizam uma estrutura de *glBegin + glVertex + glEnd* para composição, onde *glBegin* marca onde no código iniciará a construção do modelo, além de definir como será a estrutura de sua

forma, passando um valor como parâmetro, dentre as opções apresentadas na tabela 1; *glVertex* determinando em quais coordenadas será fixado o vértice; e *glEnd*, indicando onde finalizará a construção do modelo. Na figura 15, é exibido como funcionaria uma função onde utiliza-se desta estrutura para geração de um modelo 2D.

Tabela 1 – Possíveis parâmetros para *glBegin*.

Parâmetros	Função
GL_POINTS	Pontos espaçados no canvas;
GL_LINES	Linhas construídas a cada dois vértices;
GL_LINE_STRIP	Linhas interligadas;
GL_LINE_LOOP	Linhas conectadas, ponto final se liga ao ponto inicial;
GL_POLYGON	Gera um polígono;
GL_QUADS	Quadriláteros a cada 4 vértices
GL_TRIANGLES	Triângulos a cada 3 vértices
GL_TRIANGULE_STRIP	A partir do terceiro vértice, monta um triangulo a cada dois últimos pontos;
GL_TRIANGULE_FAN	A partir do terceiro vértice, monta um triangulo junto a dois pontos próximos;

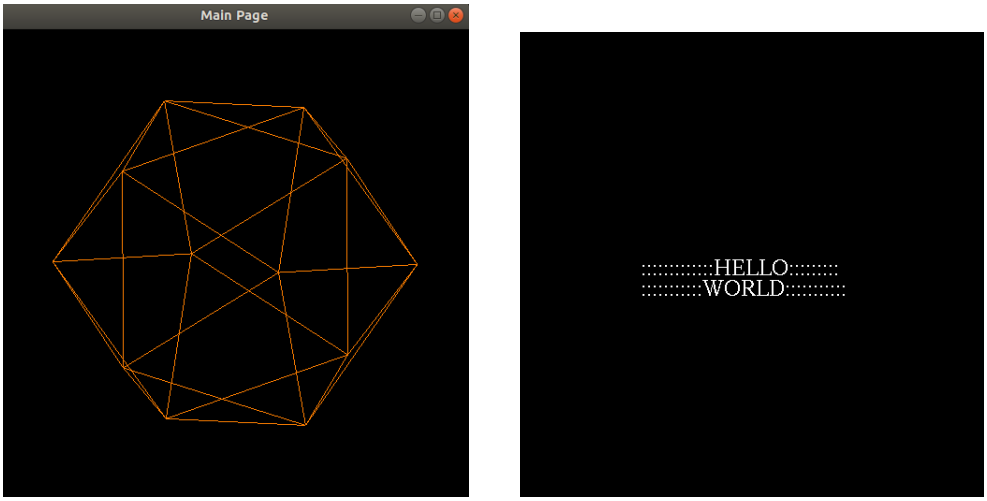
```
void mosaicgraph_draw_elipse(float radius, float * rgb){
  glColor3f(rgb[0],rgb[1],rgb[2]);
  glBegin(GL_POLYGON);
    for (int i=0; i < 360; i++){
      float degInRad = i*3.14159/180;
      glVertex2f(cos(degInRad)*(radius),sin(degInRad)*(radius));
    }
  glEnd();
}
```

Figura 15 – Função para criação de um Círculo usando glBegin, glVertex e glEnd

Protótipos de modelos 3D nativos da biblioteca foram desenvolvidos em sequência. Estes protótipos criam objetos preestabelecidos com a possibilidade de alteração da forma com passagem de parâmetros. Dos objetos possíveis, foram criados exemplos com esfera, cubo, cone, torus, dodecahedron, octahedron, tetrahedron, teapot[1] e icosahedron, este último mostrado na figura 16a.

A seguir, foram desenvolvidos códigos para o estudo de como são exploradas as transformações geométricas de translação, rotação e transformação escalar, em paralelo à utilização da biblioteca de interfaces de entrada como mouse e teclado, controlando estas transformações, que foram efetuadas sob um modelo 3D qualquer. Além disto, foram

[1] O teapot é uma representação de um bule de chá.



(a) Icosahedron nativo do OpenGL

(b) Screenshot retirado do código que renderiza uma String tamanho 24 em Times New Roman na tela

Figura 16 – Protótipos Construídos

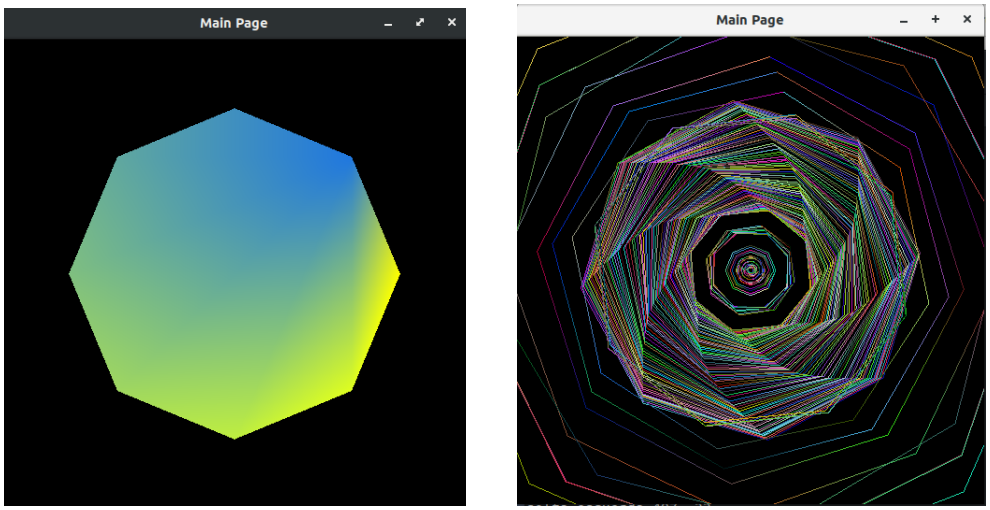
criados protótipos que utilizam as funções *glPushMatrix* e *glPopMatrix* para empilhar e desempilhar respectivamente o estado do canvas, dando a possibilidade de uma manipulação dos objetos renderizados com maior precisão. Outro protótipo feito foi um exemplo de mudança do ponto de vista em que o canvas está apontado utilizando a função *glOrtho*, que em seus parâmetros definem a região onde a janela irá captar.

Com isto, outros tipos de implementações foram exploradas, com o foco em incrementar ou auxiliar as funcionalidades disponíveis na biblioteca, como um protótipo que tira screenshot do canvas, armazenando os pixels em um buffer e após isto, salva o conteúdo em um arquivo, sob o formato .pmm ou uma aplicação gerada em que uma janela recebe uma string e a frase é renderizada na tela, na posição especificada pelo código. Na figura 16b são exibidos os dois exemplos aplicados em um único programa.

Para iniciar o estudo sobre simetria utilizando códigos, traduzindo os conhecimentos adquiridos na subseção 2.5.1 em algoritmos, a ferramenta pincel foi a primeira funcionalidade a ser implementada. Ela possui três tipos implementados: **linha reta**, **curvas de Bézier** e **modo livre**. No modo **linha reta**, gera uma linha a partir de onde foi pressionado na tela até onde ele solta o botão do mouse. No **modo Bézier**, ao clicar gera um ponto inicial e outro para um ponto final. No terceiro clique, gera um ponto intermediário, onde gera-se uma linha do início até o novo ponto e do novo ponto ao ponto final. A partir destas retas, ele gera uma curva de Bézier quadrática (SIMONI et al., 2005). Por fim, no **modo livre**, onde o mouse estiver pressionado, será desenhado no Canvas.

Para elaboração de outro exemplo utilizando simetria foi necessário o estudo de coordenadas polares. As coordenadas polares são uma representação de um ponto em um

plano que utiliza a distância do ponto da origem do plano e o ângulo formado pelo eixo X para localizá-lo. Foi feito de exemplo um gerador de polígono de N lados, utilizando o raio como distância dos vértices e diferenciando apenas o ângulo. Se N for 8, por exemplo, igual representado na figura 17a, haverá uma diferença de 45º em cada vértice, criando um polígono com 8 vértices.



(a) Octágono feito com Coordenadas Polares

(b) Mandala feita com OpenGL, ligando pontos que estão sendo repetidos

Figura 17 – Protótipos Construídos

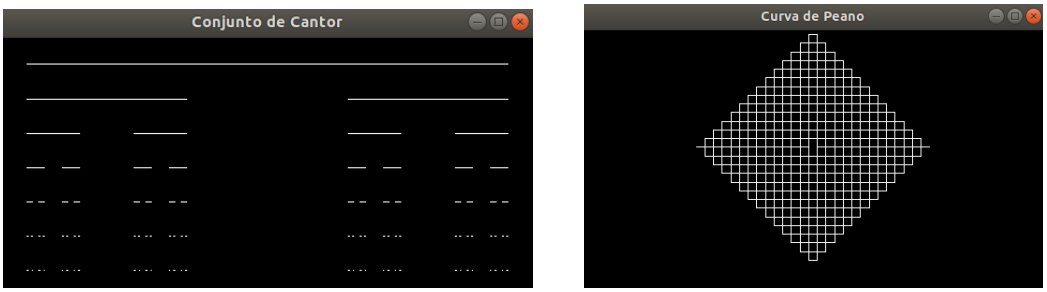
Com as coordenadas polares, tornou-se viável a criação de mandalas. Elas utilizaram as coordenadas polares junto com a ferramenta pincel, citada anteriormente para renderização na janela. Independente de onde se desenha no canvas, eles replicam para todas as partes, em fatias de um número preestabelecido no código. Foram criadas de duas versões diferentes da mandala: em uma, os pontos são apenas replicados, alterando apenas o ângulo e mantendo a distância do centro. Em outra, a cada ponto renderizado, são ligados aos pontos gerados pela replicação, formando um polígono a cada pixel desenhado, como exemplificado na figura 17b.

Após isto, foram gerados exemplos para os fractais citados na subseção 2.5.2. Para o conjunto de Cantor, foi criada uma função que recebe como parâmetro um inteiro, para definir a quantidade de iterações que sofrerá o objeto, além das coordenadas em float da posição em x, determinando a largura do objeto inicial, e também são passados parâmetros da região do canvas onde pode ser inserido o conjunto, delimitando uma altura máxima e mínima, em y.

A linha resultante formada pelos dois pontos de x junto à altura máxima é renderizada. São calculados os dois pontos que formam os segmentos de reta para uma nova iteração. Com isto, é medido qual a próxima altura máxima, subtraindo a atual pela mínima, dividindo pelo número de iterações passado na assinatura da função, pegando o

resultado e retirando este valor na altura máxima.

Se o número de repetições for maior que um, é subtraído um dele, e a função é chamada recursivamente com este novo valor, a nova altura máxima e a mínima antiga. No primeiro método recursivo, os valores x passados na assinatura e a coordenada medida localizados mais à esquerda são passados como largura, já no segundo, são passados os valores mais à direita. Caso o número de iterações seja menor ou igual a um, é finalizada a função, retornando para onde foi chamada. A figura 18a mostra um Conjunto de Cantor implementado pela função acima, com o número de iterações igual a 7.



(a) Conjunto de Cantor (b) Curva de Peano

Figura 18 – Implementações utilizando Fractais

A curva de Peano é definida a partir de duas funções, que realizam a mesma operação, alterando apenas a orientação na qual está o segmento de reta original: horizontal ou vertical. Na sua assinatura é recebido o número de iterações, e as coordenadas dos pontos que formam a linha. Dentro da função, são calculados os pontos intermediários para separação do segmento de reta em três. O traço centralizado oriundo destas três linhas geradas cria dois quadrados, um para cada lado do segmento. Com isto, estes quadriláteros mais a linha feita pelos pontos originais são renderizados.

Se o número de iterações for menor que um, para cada segmento feito a partir da linha original, e para cada lado do quadrado gerado, fora o segmento que sobrescreve o traço oriundo da reta primária, é chamada à função recursiva referente a orientação na qual o segmento está, passando na assinatura o número de iteração subtraindo um. Na figura 18b, está sendo exibido este algoritmo, onde o número de iterações está fixado como 3.

Na curva de Koch, a função construída referente ao fractal começa a partir de um segmento de reta. Na sua assinatura, são passados por referência as coordenadas do segmento mais o número de iterações que serão realizados no objeto. Primeiro, são definidas as coordenadas intermediárias, pontos em que separam o segmento de reta original em três partes. Os pontos intermediários juntos com os pontos iniciais formam dois segmentos de retas oriundas de uma iteração, porém é necessário encontrar outro ponto, perpendicular ao centro da linha inicial, que tenha o mesmo tamanho dos segmentos gerados nesta

repetição. Para localizar esta coordenada, foi encontrado o vetor unitário do traço original. Sabendo que se o produto escalar de um vetor com outro der zero, os vetores são perpendiculares, é possível colocar o x do vetor desejado em função do y.

$$V_{\perp x} = -\frac{V_{\perp y} * U_y}{U_x}$$

Onde:

$\vec{V_\perp}$:Vetor perpendicular ao segmento de reta que sofrerá a iteração da curva de Koch;
$\vec{U}$: Vetor paralelo ao segmento de reta inicial;

Utilizando o teorema de Pitágoras é possível descobrir o módulo do vetor do qual desejamos encontrar. Sabendo que ele é perpendicular a linha original, há a possibilidade formar um triângulo retângulo entre ele, a distância do centro do traço até um ponto intermediário, que é o tamanho da reta original dividido por seis, e a distância deste ponto intermediário ao ponto desejado, que é um terço da linha primária. Sabendo disto, o valor do módulo é:

$$|\vec{V_\perp}| = \frac{|\vec{U}| * \sqrt{3}}{6}$$

Com seu módulo, é possível encontrar o outro valor para o par de coordenadas, pois:

$$|\vec{V_\perp}| = \sqrt{V_{\perp x}^2 + V_{\perp y}^2}$$

$$\frac{|\vec{U}| * \sqrt{3}}{6} = \sqrt{(-\frac{V_{\perp y} * U_y}{U_x})^2 + V_{\perp y}^2}$$

$$\frac{|\vec{U}| * \sqrt{3}}{6} = \sqrt{(V_{\perp y})^2 * (U_y)^2 + (U_x)^2 * (V_{\perp y})^2}$$

$$\frac{|\vec{U}| * \sqrt{3}}{6} = (V_{\perp y}) * \sqrt{[(U_y)^2 + (U_x)^2]}$$

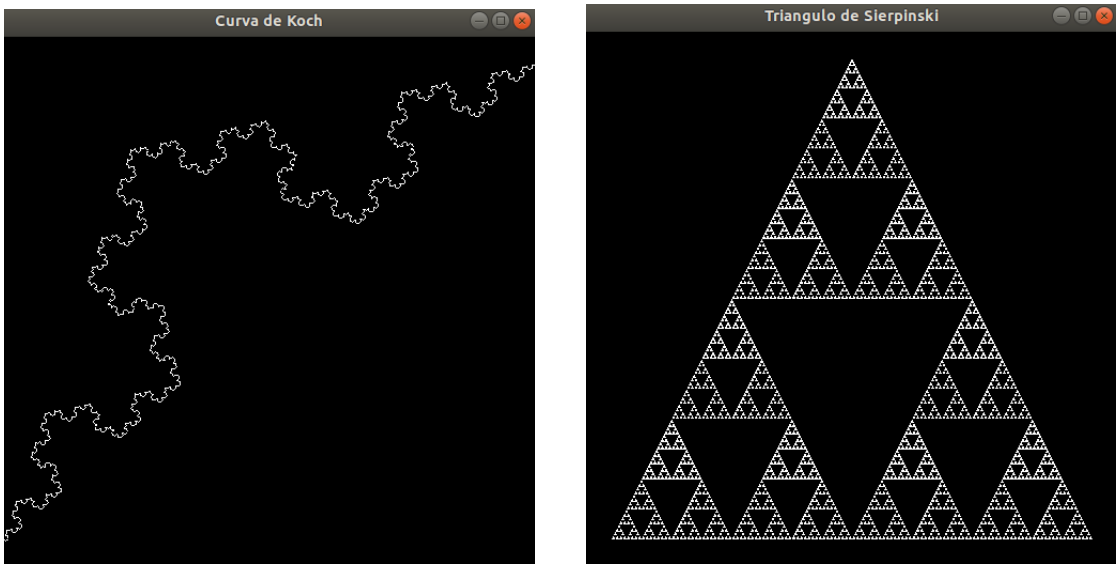
Sabendo que o vetor da reta original usado é unitário:

$$(V_{\perp y}) = \frac{|\vec{U}| * \sqrt{3}}{6}$$

Com isto, é possível descobrir o x e o y do vetor perpendicular, e, multiplicado pelo módulo encontrado dele e somado pelo ponto central da linha original, é encontrado o ponto perpendicular desejado.

Se o número de iterações for maior que um, é chamada a função recursivamente quatro vezes, sendo duas com os segmentos feitos dos pontos intermediários com os pontos do traço inicial, e dos segmentos feitos dos pontos intermediários ao ponto perpendicular

encontrado pelo calculo executado na função. Na figura 19a, há um exemplo deste código implementado, onde o traço inicial foi construído nas coordenadas (-1.0,-1.0) e (1.0,1.0), e a curva sofre 5 iterações.



(a) Curva de Koch (b) Triangulo de Sierpinski

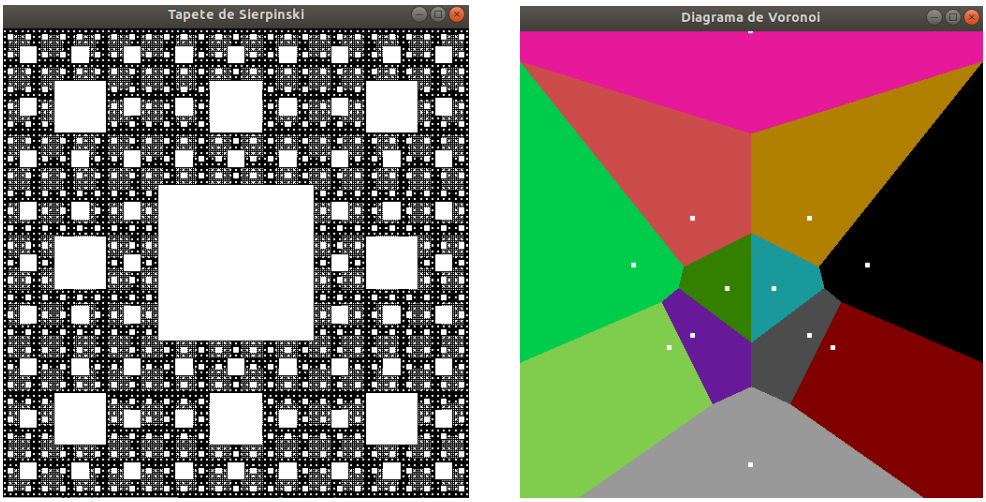
Figura 19 – Implementações utilizando Fractais

O Triângulo de Sierpinski é feito a partir de um triângulo como base. Na assinatura de função são passados o número de iterações que o objeto sofrerá, e com as coordenadas dos três pontos que formaram o triângulo inicial. Se o número de iterações for maior que um, são encontrados os três pontos médios entre os segmentos de retas feitos entre os três pontos iniciais, e chamada a função recursiva três vezes, passando em sua assinatura o número de iterações menos um, um ponto inicial e os dois pontos médios que o cercam. Caso contrário, é renderizado o triângulo com os pontos iniciais, e retornada a função. Na figura 19b está apresentado uma aplicação referente a este algoritmo, onde sofre 8 vezes a iteração.

O Tapete de Sierpinski é feito inicializado com um quadrado. Na assinatura da função, são passados o número de iteração e as quatro coordenadas necessárias para definição de um quadrado. Primeiramente, é calculado o lado do quadrado, e o mesmo é renderizado com um terço do seu tamanho. Se o número de iterações for maior que 1, são chamadas oito funções recursivas, passando como referência o número de iterações menos um e as coordenadas de regiões que cercam o quadrado renderizado e possuem o mesmo comprimento. Caso contrário, a função retorna. Um tapete de Sierpinski que sofreu 6 iterações utilizando este código está sendo exibido na figura 20a.

Para geração da Esponja de Menger, que possui como base um cubo, passamos na assinatura o número de iterações, o tamanho desejado do cubo e o ponto central dele. Se o número de iterações for maior que um, são realizadas vinte chamadas recursivas da

função, com o número de iterações menos um e o tamanho é dividido por três. Para os valores dos pontos para as vinte funções, são adicionados novos valores onde ao menos duas coordenadas são alteradas, sendo que as modificações possíveis feitas nas coordenadas são a adição ou a subtração da coordenada inicial pelo tamanho dividido por três. Caso o número de iterações for maior que um, é feita uma translação para o ponto indicado na assinatura e renderizado um cubo com o tamanho indicado, e função retorna.



(a) Tapete de Sierpinski (b) Diagrama de Voronoi

Figura 20 – Protótipos construídos com conceitos da seção 2.5.

Por fim, geramos também um algoritmo para geração de diagramas de Voronoi. Na função que cria este diagrama possui como parâmetro uma lista de pontos. São iniciados dois laços aninhados que cobrem os valores dos pixels do canvas. Um outro laço é iniciado, buscando qual ponto da lista que possui uma menor distância euclidiana do pixel. Para cada ponto da lista, é vinculada uma cor, então se um pixel possuir menor distância para um ponto da lista, ele receberá a cor deste ponto e será renderizado.

4 Resultados

Neste capítulo, abordaremos alguns resultados obtidos no decorrer deste trabalho. Primeiro será apresentada a extensão resultante da pesquisa, junto a escolhas de implementação para todas as etapas importantes para criação de uma extensão: o padrão de código, tipos de conexões e os blocos gerados para ela até o momento.

Após isto, será apresentada uma aplicação desenvolvida para uma apresentação de arte digital, que surgiu como resultado paralelo da pesquisa desenvolvida. Será feita uma explicação do que é a peça e como se encaixa a síntese de imagens nela. Por fim, um resumo das publicações científicas que se relacionam com este trabalho será mostrado.

4.1 Extensão de Síntese de Imagens em OpenGL/C++

A extensão para Sínstese de Imagens[1] (GOMES; SCHIAVONI, 2018) é escrita em C++ utilizando a biblioteca OpenGL (WOO et al., 1999).

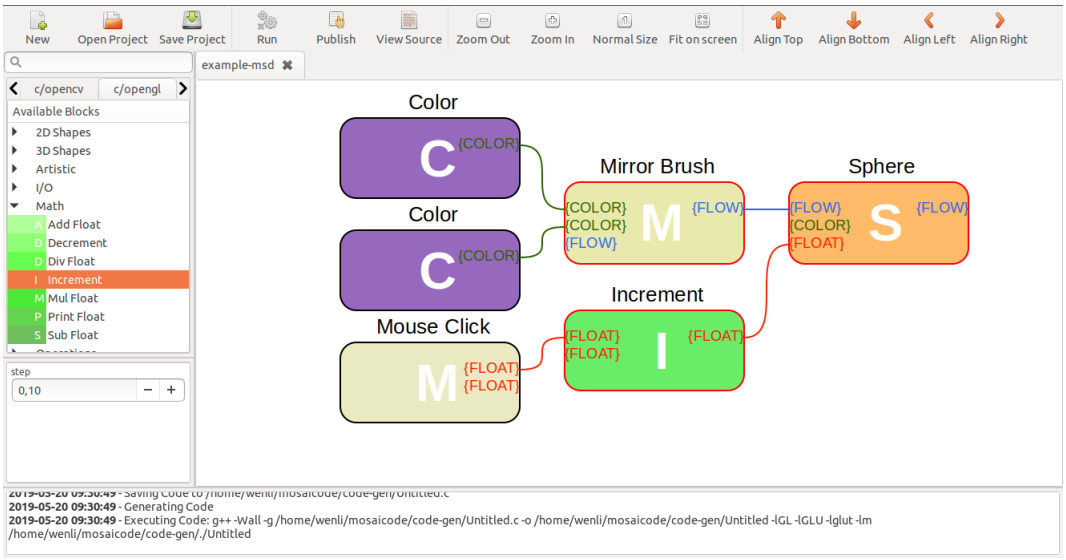


Figura 21 – Ambiente de Programação Mosaicode com a Extensão de Síntese de Imagens

4.1.1 Padrão de Código

O padrão de código da extensão possui os rótulos: **global**, **function**, **call**, **idle**, **keyboard**, **declaration** e **execution**, representado na figura 22. Ele inicia com chamadas de bibliotecas, e inclui o trecho de código referente ao rótulo **global**, onde são inseridas declaração de variáveis globais ou bibliotecas específicas para determinado tipo de bloco.

[1] Repositório no Github:<http://github.com/Mosaicode/mosaicode-c-opengl>

Após isto, é colocada uma *struct* onde são armazenados os dados da janela e uma função que inicializa os valores desta estrutura. Funções específicas de cada bloco são colocadas em seguida, local marcado pelo rótulo **function**.

```
// Bibliotecas do C/C++ e do OpenGL
<<code - global>>
typedef struct mosaicgraph_window{
        ...
}mosaicgraph_window_t;
mosaicgraph_window_t * mosaicgraph_create_window(float width, float height,float x, float
y) { ... }
int mosaicgraph_draw_window(mosaicgraph_window_t * window) { ... }
<<single_code - function>>
void display(){
        ...
        <<code - call>>
        ...
}
void keyPressed(unsigned char key, int x, int y) {
        ...
        <<code - keyboard>>
}

    void idle(){
            <<code - idle>>
            display();
    }
    int main (int argc, char** argv){
            ...
            mosaicgraph_window_t * window = mosaicgraph_create_window(500,500,0,0);
            <<code - declaration>>
            strcpy(window->title, "Main Page");
            mosaicgraph_draw_window(window);
            <<single_code - execution, connection>>
            glutDisplayFunc(display);
            glutIdleFunc(&idle);
            glutKeyboardFunc(&keyPressed);
            glutMainLoop();
            return 0;
    }
```

Figura 22 – Padrão de código para Extensão de Síntese de Imagens do Mosaicode.

Uma função que representa o que será exibido no *canvas*, chamada *display*, é declarada depois, sendo que dentro dela há um rótulo **call**, onde os procedimentos dos

blocos são chamados de acordo com a hierarquia definida pelas conexões. Neste rótulo, são chamadas funções criadas em **function**. Logo em seguida é declarada uma função *idle*, onde são inseridas alterações a cada estado de exibição do *canvas*, colocadas no local pelo rótulo **idle**, e chamada a função *display*. Após isto, é declarada uma função para manipulação de eventos de teclado, chamada *keyPressed*, onde é definido que a tecla ESC fecha a janela criada para a aplicação, e possui o rótulo **keyboard**, onde são inseridos no código teclas que manipulam algum evento do software.

Na função principal, são feitas inicializações padrão do OpenGL. O rótulo **declaration** é incluído, para declaração de variáveis ou definição de valores de parâmetro da janela. A função de criação e da janela é gerada e outro trecho de código, com o rótulo **execution**, fazendo operações de transferência de dados entre conexões e colocando adicionando comportamentos com funções da biblioteca antes do chamada de função do *display*, que é feita logo em seguida. Após isto, o algoritmo entra em *loop*, sendo que na primeira iteração é chamada a função *display* e na segunda, a função *idle*.

4.1.2 Conexões

A ferramenta dispõe também três tipos de conexões, sendo elas do tipo **Flow**, **Float** e **Color**. As do tipo **Flow** , definem a ordem em que os blocos serão colocados durante a geração de código, sendo que seu conteúdo não possui nenhuma linha de código. O objeto que possuir a porta de saída da conexão será lido antes do bloco que possui a porta de entrada desta conexão.

As conexões **Float** e **Color** transferem valores do tipo que nomeia a conexão de um bloco ao outro. As conexões tipo Float seguem o padrão de projeto Observer, onde as entradas se inscrevem na saída que, em tempo de execução, preenchem as entradas com o valor requisitado. Já em **Color**, é copiado o valor da memória da porta de saída para a porta de entrada.

4.1.3 Blocos

Os blocos são divididos em oito categorias, e a distribuição de blocos através delas está disponibilizada na tabela 2. Os blocos foram construídos a partir dos protótipos explicitados na seção 3. As escolhas de implementação estão na seção referente aos exemplos. Nesta seção, serão mostradas decisões sobre propriedades e conexões dos blocos oriundos destes protótipos.

Em **2D Shapes e 3D Shapes** trazem formas geométricas nativas do OpenGL. Para os blocos modelados em três, *Retangle, Square e Triangle* são as formas já disponíveis pela biblioteca. *Circle e Elipse* foram outras formas criadas da ferramenta de criação de polígonos disponível. *Cone, Cube, Dodecahedron, Icosahedron, Octahedron, Sphere,*

Tabela 2 – Blocos presentes na Extensão de Síntese de Imagens do Mosaicode

Categorias	**Blocos**
2D Shapes	Circle, Elipse, Rectangle, Square, Triangle
3D Shapes	Cone, Cube, Dodecahedron, Icosahedron, Octahedron, Sphere, Teapot, Tetrahedron, Torus
Artistic	Cantor Set, Koch Curve, Mandala, Menger Sponge, Peano Curve, Sierpinski Carpet, Sierpinski Triangle, Voronoi Diagram
I/O	Brush, Keyboard, Mirror Brush, Mouse Click
Math	Add Float, Decrement, Div Float, Increment, Mul Float, Print Float, Sub Float
Operations	Pop, Push, Rotate, Rotate Loop, Scale, Scale Loop, Translate, Translate Loop
Types	Color, Float
Window	Window Properties

Teapot, Tetrahedron e Torus eram comandos padrões do OpenGL que foram transformados em blocos para 3D Shapes.

Artistic

A categoria **Artistic** possui blocos referentes aos fractais citados na seção 2.5. Os primeiros blocos desta categoria estão relacionados com a geometria Fractal, onde todos os blocos possuem uma configuração de portas semelhantes. Além das portas de entrada e saída do tipo Flow, para definir o que será desenhado primeiro no canvas, eles possuem uma porta para alterar a cor do objeto e outra para modificar a quantidade de interações do fractal que serão feitas. Este último parâmetro pode ser configurado também a partir das propriedades do bloco, sendo que todo bloco que constrói um fractal possui um campo de recursividade para ser alterado.

O primeiro bloco a ser citado é o *Cantor Set*, onde são configuráveis o tamanho inicial para a reta que será dividida, bem como a região do canvas esta operação será feita. Já o bloco *Koch Curve*, possui como parâmetros os pontos iniciais para a formação de um segmento de reta, que é o passo zero para geração deste fractal. O bloco *Peano Curve* possui propriedades definindo duas coordenadas em x, que definem onde será construída a figura, e uma coordenada em y que define onde se iniciará a reta geradora do fractal.

Para o *Sierpinski Triangle*, é possível a configuração dos vértices onde serão criados o fractal, passando pelas propriedades a informação das coordenadas em x e y. Diferente do triângulo, no *Sierpinski Carpet* são recebidas duas coordenadas de altura e de largura, onde ao junta-las, formam o retângulo em que o primeiro passo deste fractal se inicia. Para a *Menger Sponge*, são definidos como propriedades valores para a posição do centro do cubo inicial do fractal, e o tamanho máximo onde ele será construído.

Além dos blocos com Fractais, existem outros dois blocos: *Voronoi Diagram* e *Mandala*. No primeiro, é feito um diagrama com seis pontos onde suas coordenadas são especificadas nas propriedades do bloco. Já o segundo, é possível definir teclas para limpeza de tudo que foi desenhado a partir da mandala, e remoção do último desenho feito, quando a pessoa pressiona e solta o mouse. É possivel definir também, através de propriedade, quantas vezes o desenho pode ser replicado e pode ser escolhida o cor da mandala, por parâmetro.

I/O

A categoria **I/O** contém os blocos: *Brush*, *Keyboard*, *Mirror Brush* e *Mouse Click*. Eles foram criados pois interagem com interfaces comuns em qualquer computador, além de que a biblioteca GLUT do OpenGL fornece funções que facilitam a comunicação com estes dispositivos. Em *Mouse Click*, o bloco pega a coordenada do clique do mouse em X e em Y. Ele retorna valores em inteiro, apesar da saída ser em float, de acordo com o tamanho especificado da janela. Em *keyboard*, tem uma propriedade para indicar o caractere que será utilizado. Ao pressionar o caractere, a saída se alterna entre zero e um. Sua função é alterada diretamente no rótulo *keyboard* , que insere código na função que chama o teclado. Sendo assim, fica estabelecido no padrão de código que a tecla *ESC* é utilizada para fechar a janela.

Os blocos *Brush* e *Mirror Brush* funcionam de maneira parecida: São pincéis em que utilizam-se do mouse, sendo que ao clicar o botão esquerdo e pressiona-lo, cria-se um rastro que é desenhado na tela. Nos dois blocos, possuem nas propriedades teclas configuráveis para limpar todo desenho feito com o pincel, ou remover apenas o último rastro pressionando o mouse. Já entre as diferenças, em Mirror Brush é desenhado um rastro simétrico, que pode ser horizontal, vertical, ou na diagonal, enquanto em Brush, só é realizado o desenho sob o rastro do mouse. A definição de como será feito o espelhamento é escolhida pela propriedade **type**.

Math

A categoria **Math** foi criada devido a uma necessidade de manipulação de valores transportados pela porta float. Antes desta categoria ser criada, a única utilidade das portas float eram definir um valor fixo para o tamanho dos objetos 2D e 3D. Agora, são ampliadas as possibilidades, sendo possível receber valores como posição do mouse como entrada, por exemplo. A categoria possui sete novos blocos, sendo que quatro deles realizam operações entre dois valores de float de entrada, dando um resultado na saída. As operações são adição, subtração, multiplicação e divisão. Há ainda um bloco quem recebe um valor em float e retorna-o no terminal.

Além destes, há blocos para incremento e decremento. Eles são uteis, pois realizam

alterações de seus valores em float em tempo de execução. Eles possuem duas entradas, uma de valor inicial, e outra como gatilho, para realizar adições ou subtrações toda vez que um valor for setado. Se o valor inicial se alterar em tempo de execução, a saída do bloco volta ao valor inicial. O valor de cada incremento é definido na propriedade *step*.

Operations

Dentre os blocos da categoria **Operations**, possui blocos que realizam transformações geométricas nos objetos desenhados na tela, como *Scale, Rotate e Translate* para uma mudança transformação escalar, rotação e deslocamento de um objeto, respectivamente, sendo que estas operações são realizadas apenas uma vez durante a execução do programa. Para que realizem de forma contínua, determinando um limite máximo e mínimo para deslocamento e transformação escalar, e rotacionando completamente um modelo, além de determinar a velocidade em que esta operação ocorrerá, existem os blocos *Scale Loop, Rotate Loop e Translate Loop*. Além disto, possui as operações hierárquicas, como empilhar e desempilhar o estado do *canvas* numa pilha com os blocos *Push e Pop*.

Types

Na categoria **Types**, possui blocos para inicializar variáveis do tipo requisitado, como *Float*, que é passado o valor deste tipo como propriedade, e em *Color*, onde pode ser escolhida a cor a partir de uma palheta, onde este valor é transformado em string e convertido para RGB, formatado para ser utilizado na biblioteca OpenGL. Em **Window**, há parâmetros para modificar o estado da janela, tudo dentro do bloco *Window Properties*. Podem ser alterados o tamanho da tela e a posição inicial dela, em pixels. Também pode ser inserido um título novo, e alterar como os polígonos vão ser exibidos (com os vértices ligados por linhas, exibidos apenas os vértices, ou todo o polígono ser preenchido). Além disto, pode ser alterado o plano de fundo para uma cor sólida.

4.1.4 Visão geral da extensão

A extensão de Síntese de Imagens para o Mosaicode possui 44 blocos divididos em oito categorias: **2D Shapes, 3D Shapes, Artistic, I/O, Math, Operations, Types e Window** e 3 tipos de conexões e portas, onde inserem trecho de algoritmos a partir de um padrão de código, que possui 7 seções onde estes códigos pode ser adicionados. Para auxiliar no desenvolvimento do usuário iniciante na utilização desta extensão do Mosaicode, foram criados alguns exemplos, trazendo conceitos básicos de utilização dos diagramas e dicas de como realizar algumas operações fazendo uma mescla de blocos.

Figura 23 – Apresentação do Chaos das 5 em Belo Horizonte

4.2 O Chaos das 5

O Chaos das 5 é uma apresentação colaborativa entre o laboratório ALICE[2] (Arts Lab in Interfaces, Computers, and Everything Else) em seu projeto Orchidea junto com o laboratório Ecolab com sua companhia Moverè[3]. A peça tem como base referências do livro **"Alice in Wonderland and through the looking glass"** (CARROLL; ROUNTREE, 1946), e traz ao espectador três camadas, musical, visual e gestual, de uma experiência imersiva a um mundo de maravilhas que se altera do sintético ao natural.

O presente trabalho se encaixa na camada visual deste espetáculo, que é composto a partir de projeções providas por dois computadores, onde dois membros do espetáculo executam uma performance, criando um ambiente que mistura imagens artificiais e reais, passando por conceitos como living code, ascii art e glitch art. Os programas utilizados na peça, que estes dois performers utilizam em seus computadores foram construídos pelo ALICE, sob demanda para produção desta peça.

Dentre os softwares utilizados, o da primeira cena foi construído a partir de diagramas construídos com o Mosaicode, onde todas as projeções foram sintetizadas por computador, sendo uma aplicação criada para projetar objetos em OpenGL, onde comandos são dados a partir do teclado e mouse. Nela é possível criar formas 2D (Círculos, Quadrados e Triângulos) e 3D (Esferas, Cubos, Cones, Torus e Teapot). É possível seleci-

[2] Site do Laboratório:<http://alice.dcomp.ufsj.edu.br/>
[3] Site do Projeto: <https://ufsj.edu.br/teatro/movere.php>

onar as cores dos objetos ou deixar esta escolha aleatória. Para renderizá-lo, é necessário clicar, definindo seu centro, e arrastar o mouse a algum ponto. A distância do ponto em que foi clicado o mouse ao ponto que foi solto o botão define o raio do objeto criado.

Figura 24 – Apresentação do Chaos das 5 em São Paulo mostrando ao fundo as imagens sintéticas criadas ao longo deste trabalho

Neste ambiente, é possível realizar algumas transformações: aumentar a escala das formas; rotacionar nos eixos X, Y e Z e centralizar todos os objetos em um ponto. Também são funções da aplicação, uma tecla que retorna os objetos no seu centro de origem, uma tecla que para realizar todas transformações realizadas no ambiente e uma tecla que limpa a janela, desalocando espaço dos objetos construídos.

4.3 Produções Acadêmicas

Além desta monografia, o presente trabalho acadêmico culminou nas seguintes publicações:

O Ambiente de Programação Visual Mosaicode - 9º Sessão de Ferramentas do CBSoft/2018 (SCHIAVONI et al., 2018): Esta publicação aborda uma visão geral do ambiente de programação, pegando desde o histórico do surgimento da ferramenta, passando pela mudança de arquitetura, motivação da criação da ferramenta, ferramentas relacionadas e funcionalidades disponíveis, até as extensões que até a publicação deste trabalho estavam em desenvolvimento.

Desenvolvimento de Extensões de Processamento e Síntese de Imagens para a ferramenta Mosaicode - Workshop of Undergraduated Works in 31st Conference on Graphics, Patterns and Images (SIBGRAPI)/2018 (GOMES; RESENDE; SCHIAVONI, 2018): Neste trabalho, apresentamos os passos que um desenvolvedor necessita realizar para a criação de uma extensão no Mosaicode. Há uma descrição sucinta sobre cada componente de uma extensão, e de como um programador deve escrever o código para criação de cada um destes elementos. Como exemplo para criação das extensões, apresentamos as extensões para Síntese de Imagens e para Processamento de Imagens, inserindo diagramas como exemplo.

Desenvolvimento de blocos em OpenGL para a ferramenta Mosaicode - XXV SIC/2018 (GOMES; SCHIAVONI, 2018): Esta produção feita em forma de artigo para o relatório de iniciação científica descreve a extensão de Síntese de Imagens dando ênfase nos detalhes de implementação. Este artigo é uma produção anterior ao desenvolvimento deste projeto orientado a computação, com menos informações, contendo apenas blocos básicos de computação gráfica.

Semana Acadêmica Integrada dos Cursos de Letras - SILE/2019: Foi realizada uma descrição objetiva sobre o funcionamento das aplicações relacionadas a camada visual da apresentação O Chaos das 5, dando detalhes sobre o que é projetado nas telas improvisadas durante o espetáculo, e ações relacionadas aos performers que atuam as cenas.

Por fim, citaremos as apresentações acadêmicas do **Chaos das 5**: a primeira apresentação ocorreu na **Mostra Vestígios**, realizada no Solar da Baronesa em São João del-Rei, no dia 23 de Novembro de 2018. Após isto, houveram até o momento as apresentações no **Sons do Silício**, onde foi apresentado no Espaço das Artes da USP, na cidade de São Paulo em 08 de abril de 2019; no **5º Seminário de Artes Digitais (SAD 2019)** ocorrido no Circuito Liberdade em Belo Horizonte na data de 26 de abril de 2019; e na **Quinta Cultural**, realizada no Campus Tancredo Neves da UFSJ em São João del-Rei, no dia 23 de maio de 2019.

5 Conclusão

Neste trabalho, apresentamos uma proposta de extensão da ferramenta **Mosaicode** [1] de síntese de imagem com a biblioteca OpenGL de C/C++, criando blocos para o domínio da Arte Digital. Nele, apresentamos a motivação para esta extensão e quais blocos criamos.

O desenvolvimento deste trabalho foi desafiante devido à mesclagem de áreas que não se comunicam usualmente, porém, quando unidas, geram trabalhos fantásticos. A minha experiência como programador em um grupo de pesquisa focado em arte contribuiu bastante em alguns momentos para criação de aplicações com maior praticidade, porém, a barreira criativa trouxe algumas dificuldades na hora da elaboração das aplicações.

Com isto, a função do artista programador seria como provedor de um instrumento para que outros possam gerar arte a partir de sua aplicação, ou a geração de produtos finais, aproveitando das técnicas de programação para prover ambientes imersivos e interativos, com o objetivo de criar novas formas de se fazer arte.

A inserção de aparatos matemáticos para criação artística auxiliou na elaboração de fórmulas e padrões alternativos para construção e de ideias criativas de como utilizar as aplicações geradas neste trabalho. Neste ponto, a participação no grupo de estudos STEAM foi essencial para o desenvolvimento deste projeto.

Esta multidisciplinaridade ocorrida durante o decorrer desta pesquisa em conjunto com diversas pessoas de perfil diferente trouxe uma contribuição na minha formação enquanto cientista da computação, no quesito de se tornar um membro híbrido capaz de dentro de um vasto campo de habilidades escolher a melhor opção para determinado problema, e conseguir desenvolver uma solução interessante usualmente e esteticamente.

Como extensão deste trabalho, há diversas possibilidades a serem exploradas. A primeira delas seria expandir os blocos, sendo eles a partir de protótipos já construídos, como um *screenshot* ou renderização de *strings* no *canvas*, ou com ênfase em conceitos da computação gráfica, como inserção de blocos de iluminação e alteração do modo projeção do canvas, modificando de um modo perspectiva para uma matriz ortogonal, por exemplo. Estes blocos podem eventualmente cobrir toda a ementa da disciplina de computação gráfica e, com isto, se tornar uma ferramenta útil para o ensino desta matéria. Um teste de usabilidade com alunos recém aprovados nesta disciplina seria interessante, para descobrir pontos em que a ferramenta pode melhorar como ferramenta de síntese de imagens.

Outra perspectiva de desenvolvimento na pesquisa, seria dar continuidade na cria-

[1] Site da ferramenta: <https://mosaicode.github.io/>

ção de blocos com um suporte de conceitos matemáticos providos pelo grupo de estudos. Definições já discutidas no grupo que não foram examinadas, como séries de Fibonacci, proporção áurea, polinomiografia, glitch art e utilização de criptografia para criação de arte. Um teste de usabilidade com membros deste grupo de estudo seria relevante para atestar a possibilidade de utilização desta ferramenta como desenvolvedora de aplicações e instrumentos no domínio da arte digital. Outra possibilidade com artistas digitais seria o desenvolvimento de outra apresentação ou uma instalação a partir de diagramas feitos no Mosaicode, ou sendo eles feitos diretamente no ambiente de programação ou prototipando seções e incorporando os diagramas em uma aplicação ou instrumento.

Por fim, uma abordagem a ser discutida para o crescimento da ferramenta como um ambiente de programação de visual no contexto da arte digital seria a interligação das extensões já criadas. Ao interligar a extensão de Síntese de Imagens com a extensão de Visão Computacional e Processamento de Imagens (GOMES; RESENDE; SCHIAVONI, 2018) se tornaria uma ferramenta de criação de instrumentos digitais para imagens com possibilidades complexas de criação. Ao encaixá-las com as extensões de áudio, torna-se viável a construção de ambientes virtuais com grande nível de imersão e interação em tempo real. Outras extensões que futuramente podem ser inseridas no Mosaicode, como uma extensão de interfaces gráficas e uma extensão de objetos de controle podem incrementar o número de possibilidades de manipulação da síntese.

Referências

ARISAWA, N. Arte e guerrilha na internet. Universidade Estadual Paulista (UNESP), 2011. Citado na página 20.

ASSIS, T. A. d. et al. Geometria fractal: propriedades e características de fractais ideais. SciELO Brasil, 2008. Citado 2 vezes nas páginas 28 e 30.

AURENHAMMER, F. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, ACM, v. 23, n. 3, p. 345–405, 1991. Citado na página 33.

BARROS, L. S. d. As manifestações de junho no brasil e o hacktivismo: uma análise das referências ao anonymous nos portais folha. com e g1. 2013. Citado na página 20.

BICHO, A. d. L. et al. Programação gráfica 3d com opengl, open inventor e java 3d. 2002. Citado na página 13.

BOGOMOLNY, A. *Homothety - an Affine Transform*. 2005. Disponivel em: <https://www.cut-the-knot.org/Curriculum/Geometry/Homothety.shtml>. Citado na página 29.

BORBA, E. Z. *Imersão visual e corporal: paradigmas da percepção em simuladores*. 2014. Citado na página 19.

CAMURRI, A. et al. Eyesweb: Toward gesture and affect recognition in interactive dance and music systems. *Computer Music Journal*, MIT Press, v. 24, n. 1, p. 57–69, 2000. Citado 2 vezes nas páginas 13 e 26.

CARISSIMI, A. Virtualização: da teoria a soluções. *Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC*, v. 2008, p. 173–207, 2008. Citado na página 19.

CARROLL, L.; ROUNTREE, H. Alice in wonderland and through the looking glass. Grosset & Dunlap Kingsport, TN, 1946. Citado na página 48.

CATTERALL, L. G. A brief history of stem and steam from an inadvertent insider. *The STEAM Journal*, v. 3, n. 1, p. 5, 2017. Citado na página 14.

CHEW, L. P.; III, R. L. S. D. Voronoi diagrams based on convex distance functions. In: ACM. *Proceedings of the first annual symposium on Computational geometry*. [S.l.], 1985. p. 235–244. Citado na página 15.

COHEN, M.; MANSSOUR, I. H. *OpenGL: uma abordagem prática e objetiva*. [S.l.]: Novatec editora, 2006. Citado 2 vezes nas páginas 13 e 34.

COLSON, R. *The fundamentals of digital art*. [S.l.]: Bloomsbury Publishing, 2007. Citado na página 12.

CRUZ, R. d. C. M. d. *Geometria Fractal: conjunto de Cantor, dimensão e medida de Hausdorff e aplicações*. Tese (Doutorado) — Universidade de São Paulo, 2018. Citado 3 vezes nas páginas 28, 29 e 30.

DANKS, M. Real-time image and video processing in gem. In: *ICMC*. [S.l.: s.n.], 1997. Citado na página 26.

GASPARETTO, D. A. Arte digital no brasil e as (re) configurações no sistema da arte. 2016. Citado na página 18.

GASPARETTO, D. A. et al. Arte digital e circuito expositivo: um curto em torno do file. Universidade Federal de Santa Maria, 2012. Citado na página 18.

GOMES, A. L. N.; RESENDE, F. R.; SCHIAVONI, F. L. Desenvolvimento de extensões de processamento e síntese de imagens para a ferramenta Mosaicode. In: *Proceedings of the CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, 31 (SIBGRAPI)*. Foz do Iguaçu - PR - Brazil: [s.n.], 2018. p. 1–4. Citado 3 vezes nas páginas 22, 50 e 52.

GOMES, A. L. N.; SCHIAVONI, F. L. Desenvolvimento de blocos em opengl para a ferramenta mosaicode. In: *Anais do XVI Congresso de Produção Científica da Universidade Federal de São João Del Rei*. São João del-Rei - MG - Brazil: [s.n.], 2018. v. 1, n. 1, p. 1–11. Citado 3 vezes nas páginas 22, 42 e 50.

GONcALVES, L. L. *Sound Design com o Mosaicode*. Dissertação (Monografia (Bacharelado em Ciência da Computação)) — Universidade Federal de SÃ£o JoÃ£o del-Rei, 2017. Citado na página 22.

JUNGBLUT, A. L. A heterogenia do mundo on-line: algumas reflexões sobre virtualização, comunicação mediada por computador e ciberespaço. *Horizontes Antropológicos*, SciELO Brasil, v. 10, n. 21, p. 97–121, 2004. Citado na página 19.

KIRNER, C.; KIRNER, T. G. Evolução e tendências da realidade virtual e da realidade aumentada. *Realidade Virtual e Aumentada: Aplicações e Tendências. Cap*, v. 1, p. 10–25, 2011. Citado na página 19.

LÉVY, P. *Que é o Virtual?, O*. [S.l.]: Editora 34, 2003. Citado na página 19.

MANDELBROT, B. B. Stochastic models for the earth's relief, the shape and the fractal dimension of the coastlines, and the number-area rule for islands. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 72, n. 10, p. 3825–3828, 1975. Citado na página 28.

MARCOS, A. Instanciando mecanismos de a/r/tografia no processo de criação em arte digital/computacional. *inVISIBILIDADES-Revista Iberoamericana de Pesquisa em Educação, Cultura e Artes*, APECV–Associação de Professores de Expressão e Comunicação Visual, v. 3, p. 138–145, 2012. Citado na página 18.

NETTO, A. V.; MACHADO, L. d. S.; OLIVEIRA, M. d. Realidade virtual-definições, dispositivos e aplicações. *Revista Eletrônica de Iniciação Científica-REIC. Ano II*, v. 2, p. 34, 2002. Citado na página 18.

NOVAIS, R. A. Media e (ciber) terrorismo. *NAÇÃO E DEFESA, Lisboa: Instituto de Defesa Nacional, n*, p. 89–103, 2012. Citado na página 20.

NUNES, R. S. R. Geometria fractal e aplicações. *Departamento de Matemática Pura, Faculdade de Ciências da Universidade de Porto*, 2006. Citado na página 29.

OLIVEIRA, T. Imersão em jogos pervasivos. *Rumores*, v. 7, n. 14, p. 315–334, dez. 2013. Disponível em: <<http://www.journals.usp.br/Rumores/article/view/69445>>. Citado na página 18.

PARISI, T. *WebGL: up and running*. [S.l.]: "O'Reilly Media, Inc.", 2012. Citado na página 13.

PAUL, C. *Digital art*. [S.l.]: Thames & Hudson London, 2003. Citado 2 vezes nas páginas 19 e 20.

PRIMO, A. F. T. Interação mútua e interação reativa: uma proposta de estudo. In: *Congresso Brasileiro de Ciências da Comunicação (21.: 1998: Recife).[Anais..] Recife: Intercom, 1998*. [S.l.: s.n.], 1998. Citado na página 19.

PRIMO, A. F. T. Explorando o conceito de interatividade: definições e taxonomias. *Informática na educação: teoria & prática. Vol. 2, n. 2 (out. 1999), p. 65-80*, 1999. Citado na página 19.

PUCKETTE, M. S. et al. Pure data. In: *ICMC*. [S.l.: s.n.], 1997. Citado 2 vezes nas páginas 13 e 25.

PULLI, K. et al. Real-time computer vision with opencv. *Communications of the ACM*, ACM, v. 55, n. 6, p. 61–69, 2012. Citado na página 23.

REDIES, C.; HASENSTEIN, J.; DENZLER, J. Fractal-like image statistics in visual art: similarity to natural scenes. *Spatial Vision*, Brill, v. 21, n. 1, p. 137–148, 2007. Citado na página 15.

RIPPLINGER, H. M. G. A simetria nas práticas escolares. 2006. Citado na página 27.

ROCHA, G. L. et al. Desenvolvimento de instrumentos digitais a partir de dispositivos ubíquos. *Proceedings of 8th Workshop on Ubiquitous Music (UbiMus)*, v. 1, p. 98–110, 2018. Citado na página 21.

ROHDE, G. M. *Simetria: rigor e imaginação*. [S.l.]: EDIPUCRS, 1997. Citado na página 27.

SCHIAVONI, F. L. et al. O ambiente de programação visual mosaicode. *Anais da 9ª Sessão de Ferramentas do CBSoft*, v. 1, p. 25–30, 2018. Citado 3 vezes nas páginas 12, 22 e 49.

SCHIAVONI, F. L.; GONçALVES, L. L. From virtual reality to digital arts with mosaicode. In: *2017 19th Symposium on Virtual and Augmented Reality (SVR)*. Curitiba - PR - Brazil: [s.n.], 2017. p. 200–206. ISBN 978-1-5386-3588-9. Citado 2 vezes nas páginas 13 e 22.

SCHIAVONI, F. L.; GONçALVES, L. L. Programação musical para a web com o mosaicode. In: *Anais do XXVII Congresso da Associação Nacional de Pesquisa e Pós-Graduação em Música*. Campinas - SP - Brazil: [s.n.], 2017. p. 1–6. Citado na página 22.

SCHIAVONI, F. L.; GONçALVES, L. L. Teste de usabilidade do sistema mosaicode. In: *Anais do IV Workshop de Iniciação Científica em Sistemas de Informação (WICSI)*. Lavras - MG - Brazil: [s.n.], 2017. p. 5–8. Citado na página 22.

SCHIAVONI, F. L.; GONçALVES, L. L.; GOMES, A. L. N. Web audio application development with mosaicode. In: *Proceedings of the 16th Brazilian Symposium on Computer Music*. SÃ£o Paulo - SP - Brazil: [s.n.], 2017. p. 107–114. Citado na página 22.

SIMONI, R. et al. Teoria local das curvas. 2005. Citado na página 36.

SINGH, P. *Learning Vulkan*. [S.l.]: Packt Publishing Ltd, 2016. Citado na página 13.

TAVARES, M. et al. Os processos criativos com os meios eletrônicos. [sn], 1995. Citado na página 12.

TRIBE, M.; JANA, R.; GROSENICK, U. *New media art*. [S.l.]: Taschen Los Angeles, 2006. Citado na página 12.

WEINTRAUB, A. Art on the web, the web as art. *Communications of the ACM*, Association for Computing Machinery, Inc., v. 40, n. 10, p. 97–103, 1997. Citado na página 19.

WOO, M. et al. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1999. Citado 3 vezes nas páginas 13, 34 e 42.