UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Frederico Ribeiro Resende

Processamento Digital de Imagens e Visão Computacional no ambiente Mosaicode

São João del-Rei

2019

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Frederico Ribeiro Resende

Processamento Digital de Imagens e Visão Computacional no ambiente Mosaicode

Monografia apresentada como requisito da disciplina de Projeto Orientado em Computação II 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

Frederico Ribeiro Resende

Processamento Digital de Imagens e Visão Computacional no ambiente Mosaicode

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

Flávio Luiz Schiavoni
Orientador

Daniel Luiz Alves Madeira
Convidado 1

Marcos Antonio de Matos Laia
Convidado 2

São João del-Rei
2019

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Primeiramente, agradeço aos meus pais Ana Maria e Agnaldo pelo amor, incentivo incondicional e suporte oferecido ao longo desses quatro anos. Sem eles, nada seria possível. Agradeço ao meu tio Renato e minhas irmãs por estarem presentes nos momentos mais difíceis e pelo apoio constante. Agradeço de forma especial à minha namorada por todo o tempo dedicado a mim e por me ajudar a superar todas as barreiras enfrentadas.

Agradeço também toda minha família e amigos, em especial para o Diego, Antônio Carlos e Gustavo, que sempre estiveram presente desde a infância até o presente momento, e que torceram muito por mim durante esta jornada. Agradeço também todas as amizades conquistadas durante a graduação, principalmente aos colegas de laboratório pelos constantes auxílios durante toda a caminhada.

Agradeço de forma extremamente especial ao meu orientador Flávio Luiz Schiavoni, por todo o suporte, paciência, ensinamentos, pelos dois anos de pesquisa juntos e por todas as oportunidades proporcionadas a mim. Agradeço também aos professores participantes da banca examinadora, Daniel Luiz Alves Madeira e Marcos Antonio de Matos Laia, por aceitarem o convite e estarem presentes neste momento tão importante para mim.

Agradeço à Universidade Federal de São João del-Rei, ao seu corpo docente, direção e administração por todo o suporte oferecido neste tempo. Em especial, agradeço ao departamento de ciência da computação por todo o auxílio e pelos momentos proporcionados, onde pude ser feliz por quatro anos conjunto dos melhores amigos e professores.

Agradeço de forma geral, todos aqueles que fizeram parte, direta ou indiretamente, de minha formação e do meu crescimento como profissional. A todos, o meu muito obrigado.

"Os que se encantam com a prática sem a ciência são como os timoneiros
que entram no navio sem timão nem bússola,
nunca tendo certeza do seu destino."
(Leonardo da Vinci)

Resumo

O Processamento Digital de Imagens e a Visão Computacional são duas áreas da computação que têm mostrado grandes avanços nas últimas décadas, marcando presença nos mais variados processos vigentes no cotidiano atual. Ambas são motivos de estudo por alunos que desejam idealizar aplicações que utilizem grandes níveis de tecnologia avançada. Porém, estudos apontam que a desistência do ingresso em tais áreas por parte dos alunos tem se tornado um fato comum, muitas vezes causada pelo entrave da codificação. Abordando ocorrências similares, muitos artistas têm tentado usufruir dos conceitos dessas mesmas áreas para construírem aplicações de Arte Digital, entretanto, o mesmo impasse calhado aos alunos se repete. Neste contexto, apresentaremos o ambiente de programação visual Mosaicode, uma ferramenta que busca facilitar a construção de aplicações por alunos e artistas, utilizando conceitos visuais para simplificar a criação. Desta forma, este trabalho tratará de desenvolver um conjunto de funcionalidades de Visão Computacional e Processamento Digital de Imagens que auxilie aos devidos alunos e artistas à programarem.

Palavras-chaves: processamento de imagens. visão computacional. programação visual.

Abstract

Digital Image Processing and Computer Vision are two areas of computer science that have shown great advances in the last decades, building their presence through the most varied processes in current day-to-day life. Both areas give students a reason to acomplish ideas of applications that use a higher level of advanced technology. However, studies have pointed out that the number of students signing up to these areas has decreased, mostly because coding is a holdback. Approaching similar occurrences, many artists have tried to use the concepts of these same areas to build applications of Digital Art, although, the the students get to same barrier. In this context, we will introduce the Mosaicode visual programming environment, a tool that seeks to easy the building of applications by both students and artists, using visual concepts to simplify creation. Thus, this work will develop a set of functionalities of Computer Vision and Digital Image Processing that should help the students and artists to code.

Key-words: image processing. computer vision. visual programming.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

LIDPV	Linguagem de Programação Visual
LINDE	Linguagem de Domínio Específico
PDI	Processamento Digital de Imagens
VC	Visão Computacional

Sumário

1 Introdução

O Processamento Digital de Imagens (PDI) e a Visão Computacional (VC) são duas áreas da computação que atuam de forma inter-relacionada, principalmente por possuírem um mesmo habitat e estarem alinhadas em um processo de processamento gráfico e análise bidimensional. O PDI pode ser definido como o domínio responsável por abranger técnicas e métodos para realizar tratamentos e manipulações em dados dispostos em duas dimensões, onde muitas dessas técnicas envolvem paradigmas de processamento de sinais (1). Já a VC é a área que trata de emular a visão humana perante ao computador, de forma a extrair e inferir informações sobre dados multidimensionais (2).

Nota-se após estas definições, que ambas as áreas descritas estão diretamente relacionadas, e compõem um sistema maior, a Computação Gráfica, que engloba ainda outros campos como a Síntese de Imagens (3). Continuando a tratar especificamente sobre o PDI e a VC, é possível perfilá-las de forma que funcionem sequencialmente, aplicando algoritmos e operações que utilizem o processamento obtido anteriormente como fruto da próxima iteração. Nesta situação, podemos elicitar principalmente, o emprego do PDI aplicando algoritmos para tratar e preparar a imagem que será utilizada para extração de informações pela VC (4, 5, 6).

É possível enumerar variadas aplicabilidades destas subáreas, tanto singularmente, quanto trabalhando em conjunto, como no exemplo citado há pouco. Dentre estas possibilidades, podemos listar aplicações para reconhecimento gestuais e de objetos, detecção de padrões, sistemas de realidade aumentada e várias outras funcionalidades que podem ser construídas utilizando PDI e VC em sincronia. Além disto, ambas possuem aplicações em campos interdisciplinares, possibilitando que elas sejam delegadas também a âmbitos não computacionais, como para a vertente da Arte Digital (7).

A Arte Digital pode ser definida como a criação artística contemporânea que faz uso de meios tecnológicos para a elaboração de aplicações voltadas a este intuito (8). Nos tempos atuais, cada vez mais a tecnologia e a computação, com suas devidas áreas e particularidades, têm obtido seu espaço no mundo de forma maiúscula, incluindo vários contextos e ambientes distintos. Tratando-se especificamente do domínio artístico, é notável que artistas busquem por tecnologias que os auxiliem e permitam a construção de aplicações gráficas com vários artifícios inovadores; aliás, muitos deles buscam utilizar técnicas oriundas de PDI e VC para comporem suas peças digitais, onde os próprios não necessariamente se preocupam com a estética computacional e organizacional das aplicações construídas. Contudo, ferramentas que desempenhem estas funções específicas são escassas neste meio.

É importante ressaltar que, em geral, artistas nem sempre possuem conhecimento específico de tecnologias computacionais para utilizar na criação de aplicações artísticas. Logo, ferramentas e ambientes de desenvolvimento que proporcionem estas criações de modo facilitado são úteis e auxiliam na construção de algoritmos que gerem aplicações para os próprios artistas.

Com este cenário, abordaremos o ambiente de programação visual Mosaicode (9, 10), uma Linguagem de Programação Visual (LIDPV) que possui o intuito de gerar código para domínios bem definidos, utilizando Linguagens Específicas de Domínio (LINDEs). O enfoque do Mosaicode é permitir que pessoas como estudantes e artistas que conheçam teoricamente um determinado domínio, mas que não possuam conhecimento específico de uma LINDE relativa a este, possam programar e criar programas com maior facilidade, sem a necessidade de uma codificação textual.

Abordando especificamente o meio educacional, apresentaremos o Mosaicode como uma ferramenta que pode ser utilizada não só por artistas para comporem suas obras, mas também por alunos recém-chegados a cursos da área de computação, de forma que possa ser possível adquirir conhecimentos em lógica de programação e obviamente, na LINDE utilizada. Além disto, o foco se dará em estudantes específicos das áreas de PDI e VC, onde a ferramenta utilizará técnicas e conceitos sobre tais de uma forma descomplicada como uma tentativa de atrair os alunos para o meio de um modo mais simples e cativante, eliminando o entrave causado pela dificuldade imposta na abstração de um *framework* ou pela programação codificada.

Retornando ao ponto de vista artístico, a ferramenta possui o intuito de facilitar a criação e a exportação de aplicações de forma análoga à descrita anteriormente para alunos de computação. Porém, neste caso, o foco não se trata da aprendizagem de lógica e da arquitetura computacional por trás do funcionamento da aplicação propriamente dita, mas sim no poderio dos recursos e na complexidade exigida para a composição de programas. Subentende-se que a praticidade e a simplicidade são fatores cruciais para possibilitar que a ferramenta seja amplamente utilizada por esses artistas.

Considerando todos os pontos aqui listados, este trabalho abordará uma discussão que relacione as áreas de PDI e VC com vertentes como a Arte Digital e o domínio educacional, onde é importante considerar a ferramenta Mosaicode como um objeto intermediador para realizar as devidas ligações entre esses campos.

1.1 Objetivos

Este trabalho tem como objetivo discutir a interação entre o PDI e a VC como um todo, de forma a interagir com outros conceitos como geração de código, Arte Digital e com o âmbito educacional. Desta maneira, define-se a criação de um pacote específico

de domínio para o ambiente Mosaicode utilizando uma LINDE que englobe os campos abordados.

Um dos objetivos secundários definidos será observar a relação obtida da interligação destes domínios e em analisar o poderio computacional oferecido pelo conjunto de plugins, considerando o processamento demandado pelas aplicações, a ortogonalidade das técnicas elaboradas e a metodologia utilizada para a geração de código. Outro objetivo definido em segundo plano é aplicar uma revisão sistemática de literatura, possibilitando posicionar esta pesquisa dentro de um contexto histórico e indicando trabalhos similares que ainda não foram encontrados anteriormente.

1.2 Justificativa

Em cursos da área de computação, a programação é essencial e amplamente difundida para o desenvolvimento de aplicações e para a criação de funcionalidades didáticas, sendo abordada desta última forma como uma porta de entrada para estes mesmos cursos. No entanto, ela também atua repelindo novos alunos que se assustam com seu funcionamento, ou então, que não se interessam pela mesma (11).

Dados estes fatos e observando o crescimento gradativo do índice de evasão dos cursos da área de computação, alguns métodos e ferramentas têm sido implementados nas universidades a fim de diminuir esta estatística (12). Neste contexto, um dos métodos que podem ser utilizados em relação à dificuldade imposta pela programação é a utilização de LIDPVs e meios interativos visuais para facilitar o entendimento e a programação propriamente dita (13).

Desta forma, podemos propiciar a implantação do Mosaicode como um ambiente de aprendizagem para estudantes novatos na área de programação, onde é possível abstrair conceitos básicos de programação e de fluxo de código. Assim, cria-se a possibilidade de viabilizar o aumento do interesse dos alunos pela programação utilizando o retorno gráfico para obter possivelmente, uma maior aprovação e, consequentemente, reduzir o índice de evasão nos cursos em questão.

Além do mais, esta seria apenas uma das vantagens oriundas à aplicação do Mosaicode e do pacote a ser criado especificamente como uma ferramenta de ensino de PDI e VC. Propriamente a estas áreas, o ambiente pode ser difundido para alunos destas disciplinas em uma fase inicial, de forma a facilitar a aprendizagem dos conceitos básicos e fundamentais, como foi realizado em (14). Nesta situação, o ambiente foi utilizado na disciplina de Introdução à Computação Musical do curso de Ciência da Computação, na Universidade Federal de São João del-Rei, onde os trabalhos práticos foram demandados por meio da ferramenta.

O Mosaicode em si é um gerador de código com a possibilidade da execução embutida da aplicação construída. Além de auxiliar na praticidade para criação das aplicações, usuários com conhecimento do *framework* utilizado podem utilizar a ferramenta essencialmente para geração de código, a fim de economizar tempo e trabalho utilizando tal código como base para o seu desenvolvimento, ou até mesmo, como aplicação final.

Abordando outra vertente, a Arte Digital, como já citado anteriormente, não possui um vasto poderio de ferramentas prontas para uso para geração de aplicações especializadas para o contexto. Em base neste fundamento, esperamos que a construção dos plugins para o Mosaicode vá suprir um vago espaço deixado à artistas que desejam criar seus próprios programas utilizando as mais variadas tecnologias possíveis relativas à PDI e VC.

Utilizando o ambiente Mosaicode, visa-se facilitar a criação destas aplicações, dado que as LIDPVs apresentam recentemente um constante crescimento e têm sido cada vez mais buscadas, principalmente por agilizar o processo de construção e descomplicar a aprendizagem para aqueles que estão iniciando os estudos. Além disso, ele auxilia na elaboração de programas para quem não possui conhecimento específico em programação textual e molda-se como base para geração de programas complexos e extensos.

1.3 Trabalhos Relacionados

Nesta subseção, abordaremos algumas ferramentas relacionadas ao Mosaicode e à programação visual com mídias como imagens, enfatizando aquelas que trabalhem com o processamento e a extração de dados.

O Pure Data é um ambiente de programação visual open-source para criação de som e música em tempo real desenvolvido por Miller Puckette (15). A ferramenta que também pode funcionar como hospedeira para o ambiente GEM de processamento de gráfico (16). Como o próprio nome já declara, o Pure Data trabalha com "dados puros", de modo a permitir que dados de diferentes fontes possam ser tratados de forma interligada, facilitando o acoplamento de aplicações de áudio, vídeo e outras fontes possíveis.

Outra ferramenta de programação visual é o EyesWeb, um ambiente de programação open-source para som, imagem e vídeo, que se assemelha em partes ao Pure Data, porém é focado na captura e análise de gestos e movimento corporais (17, 18). Uma de suas principais funcionalidades e que desperta olhares é que o EyesWeb suporta vários dispositivos de entrada como câmeras, sensores e interfaces de videogames.

O Labview (Laboratory Virtual Instrument Engineering Workbench) é um *software* multiplataforma de programação visual para ensino e prototipação desenvolvido pela National Instruments que teve sua primeira versão em 1986 (19). O programa utiliza re-

presentações gráficas para simular componentes de *hardware* e aparelhos de teste, controle e medição, simplificando o projeto de aplicações de sistemas distribuídos. Porém, o mesmo possui foco na área de engenharia de sistemas a fim de que seja utilizado em medições e testes de aplicações; vale ressaltar a sua praticidade e interface de fácil de manejo pelo usuário.

Abordando a vertente de ensino através de LIDPVs, temos o EarSketch, um ambiente de programação visual para ensino de programação de linguagens como JavaScript e Python com mixagem e composição musical (20). Além do EarSketch, temos o Processing, uma linguagem de programação textual open-source que utiliza a didática visual através de artes gráficas para ensinar lógica de programação à iniciantes (21).

1.4 Organização do Texto

Após a preparação e colocação dos fatos que nos remetem a este projeto, o Capítulo 2 irá abordar a revisão sistemática e os resultados alcançados pela mesma. Será explicitada toda a metodologia do processo, o embasamento necessário para a execução, e como se dá a conexão dos trabalhos encontrados com a nossa pesquisa.

Já o Capítulo 3 irá explanar o que é o ambiente Mosaicode, como se dá seu funcionamento, arquitetura, dentre outros pontos importantes. Além disso, este capítulo irá retratar a metodologia para a construção de uma extensão (conjunto de funcionalidades específicas), que se trata de um componente vital do projeto em si.

Seguindo adiante, o Capítulo 4 define a metodologia elaborada para a construção da extensão e das funcionalidades elicitadas. No Capítulo 5, será abordada a composição resultante da metodologia disposta no capítulo anterior, trazendo as várias vertentes do conjunto de funcionalidades desenvolvido.

Já no Capítulo 6, serão retratados os resultados obtidos perante a extensão e a atuação real da mesma. Além disso, será demonstrado em partes o funcionamento do conjunto de plugins, exibindo algumas composições possíveis utilizando a ferramenta Mosaicode. Por fim, o Capítulo 7 descreve as devidas conclusões obtidas diante a toda pesquisa e ao processo de mesclar variadas áreas por meio de um só componente geral.

2 Revisão Sistemática

Uma revisão sistemática é um meio de estudo para pesquisar, coletar, avaliar e interpretar todo tipo de dado disponível sobre determinado tema ou assunto. Ela consiste em abstrair toda pesquisa relacionada ao tópico em questão a fim de que o presente trabalho possa ser situado de uma forma geral no próprio ambiente, considerando toda produção acadêmica já existente em relação ao meio de atuação do mesmo. A intenção de uma revisão sistemática normalmente é reunir uma conjuntura de estudos sobre determinado assunto de modo a compará-los, identificando ou não resultados de pesquisas que sejam conflitantes, onde irão auxiliar a guiar a presente pesquisa (22).

Uma revisão sistemática segue necessariamente um protocolo bem definido, que guiará a busca pelos estudos existentes que melhor se encaixem com o corrente projeto. Após a obtenção desses estudos, os atos seguinte se dão pela rígida e confiável extração, sumarização e interpretação dos dados relevantes, criando a base do ambiente para a atual pesquisa.

Esta revisão sistemática foi conduzida e definida de acordo com as diretrizes em (23) e com o processo realizado em (24), onde fez-se necessário a segmentação em três passos: planejamento, condução e análise dos resultados, como exibido na Figura 1. As etapas serão apresentadas a fundo nos itens a seguir.

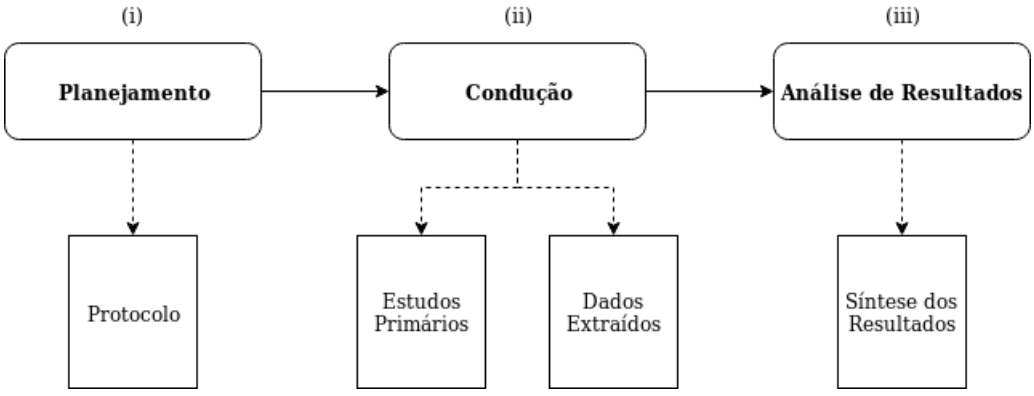


Figura 1 – Fases da revisão sistemática.

A. *Planejamento da Revisão*

O objetivo principal desta revisão é validar o trabalho atual, além de contextualizar e explorar o estado da arte. Em nosso caso, buscamos encontrar ferramentas ou ambientes de programação visual para geração de código de VC e PDI. Para tal, um protocolo muito bem definido deve ser seguido inicialmente, este contendo *(a)* questões de revisão; *(b)* estratégia de busca; *(c)* os critérios de inclusão e exclusão;

e *(d)* estratégia de seleção de dados e síntese dos resultados. As questões de revisão devem sintetizar o objetivo geral da pesquisa, e considerando nosso propósito de geração de código para ambas as áreas definidas, independentemente do contexto, foram formuladas as seguintes questões:

1) ***Existem estudos ou ferramentas, em andamento ou concluídos, sobre geração de código de Visão Computacional e/ou Processamento Digital de Imagens?***
2) ***Existem estudos ou ferramentas, em andamento ou concluídos, sobre criação de Arte Digital com Visão Computacional e/ou Processamento Digital de Imagens?***
3) ***Existem estudos ou ferramentas, em andamento ou concluídos, sobre ensino de programação de Visão Computacional e/ou Processamento Digital de Imagens com LIDPVs?***

Com base nestas perguntas poderemos elicitar toda ferramenta que contenha o mesmo intuito que o nosso, ou parte dele. Desta forma, será possível encontrar estudos primários que busquem funcionalidades específicas conforme determinado nas questões, e assim, determinar o estado da arte em ambas as virtudes requeridas.

Neste contexto, elencamos palavras-chave que correspondam aos resultados esperados das questões de revisão, palavras estas que integram uma *string* de busca (composição de vários termos). Para a formação dessa *string*, devemos definir as palavras que estão relacionadas com as questões de revisão elaboradas e utilizar também sinônimos de termos relevantes como geração de código e PDI ou VC. Após serem escolhidas, serão concatenadas através dos operadores lógicos **"OR"** unindo todos os sinônimos ou termos similares, e **"AND"** para conectar todas as expressões formadas, a fim de obter a *string* de busca, esta que é disposta abaixo:

```
("computer vision" OR "computational vision" OR "image
    processing") AND ("code generation" OR "code generator" OR
    "digital art" OR "learning") AND ("vpl" OR "visual programming
    language" OR visual programming environment OR "visual
    programming" OR "dsl" OR "domain specific language")
```

A *string* de busca foi aplicada nas seguintes bases de dados:

- *ACM (www.acm.org);*
- *IEEE Xplore (www.ieeexplore.ieee.org);*
- *ScienceDirect (www.sciencedirect.com);*
- *Scopus (www.scopus.com);*

- *Web of Science (www.webofknowledge.com).*

Além destas, foram executadas buscas manuais em eventos e congressos específicos, analisando títulos relevantes e posteriormente, o resumo. Após estas etapas, com os resultados gerados por ambas as buscas, foram então definidos os critérios de seleção para inclusão e remoção de trabalhos. Deste modo, todas as pesquisas retornadas pela busca inicial transpassaram por uma análise com base nos critérios definidos abaixo:

Critérios de inclusão:

- Estudos primários que apresentem evidências ou uso de geração de código para VC e/ou PDI ou arte digital;
- Estudos primários que apresentem evidências ou uso de LIDPVs para VC e/ou PDI ou arte digital;
- Estudos primários que apresentem evidências ou uso de LIDPVs para o ensino de conceitos de VC e/ou PDI.

Critérios de exclusão:

- Estudos primários repetidos;
- Estudos primários que não estão escritos em inglês ou português;
- Estudos primários que não são *full paper* ou *short paper* (posters, tutoriais, sessões técnicas);
- Estudos primários que não foram realizados entre 1990 e 2019;
- Estudos primários que não estejam disponíveis no formato digital.

B. *Condução da Revisão*

Nesta etapa, foram analisados todos os estudos primários obtidos pelas buscas realizadas nas bases definidas na fase anterior, com o objetivo de reduzir consideravelmente a quantidade de pesquisas, restando apenas aqueles que se encaixem nos critérios definidos. Para tal, a Figura 2 exibe quantitativamente os resultados obtidos em cada uma das etapas de filtragem.

É válido relatar que a quantidade de estudos primários obtidos com as buscas foram relevantes, porém muitos deles devem ser cortados, já que não se adéquam ao caso de pesquisa em questão. Para tal ação, os critérios foram aplicados levando a tona apenas o título e resumo dos trabalhos, já que este se torna um processo bem menos trabalhoso e que determina o corte de uma grande parte dos trabalhos. Foram 423 trabalhos obtidos, onde 179 foram repetidos entre as bases, restando 244 para a primeira etapa de filtragem. Através do título e resumo, 203 foram excluídos. Dos

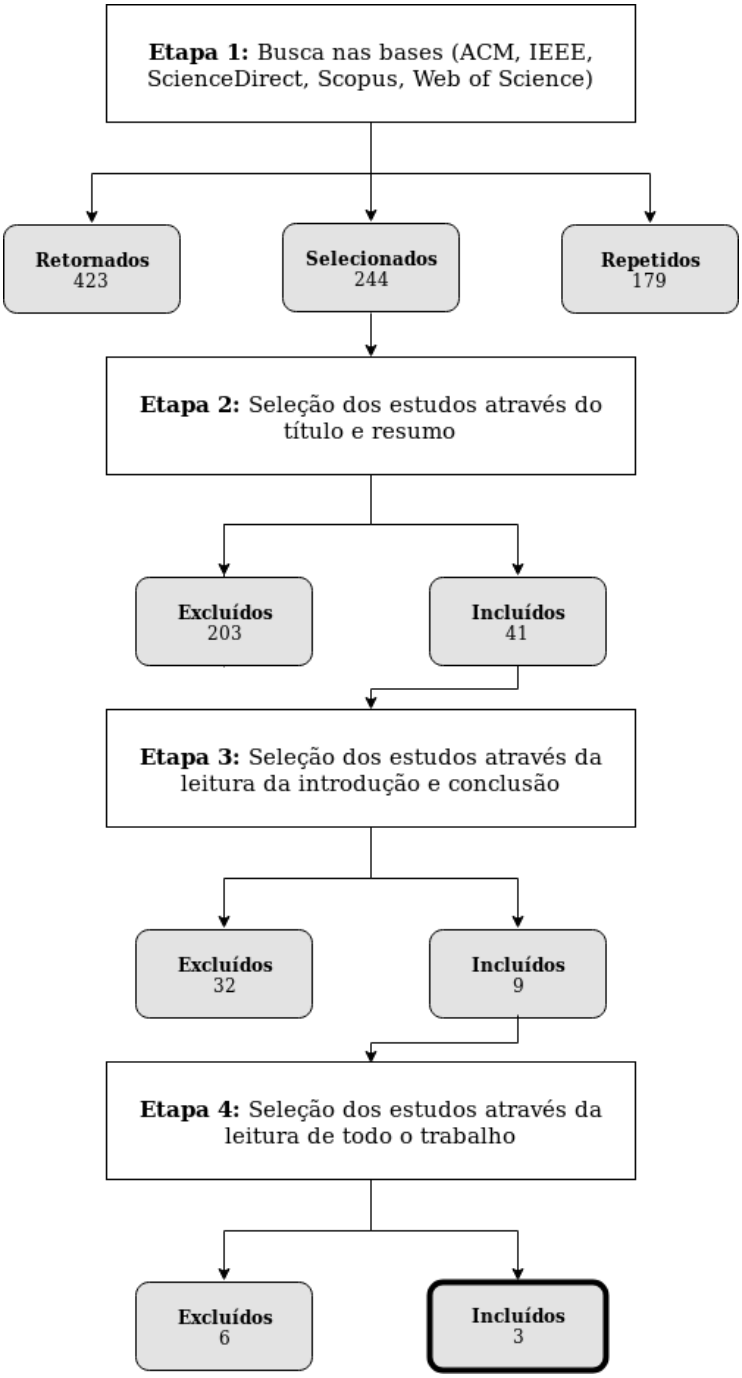


Figura 2 – Condução e seleção dos estudos primários.

41 restantes, 9 foram selecionados pela leitura da introdução e conclusão. Após isto, todos os estudos primários foram lidos e sintetizados de forma a selecionar apenas aqueles que correspondessem aos critérios de inclusão, onde apenas três se encaixaram.

C. *Análise de Resultados*

Após determinar os estudos primários que conduzem e assemelham-se com a nossa pesquisa em uma série de aspectos, três trabalhos foram elencados para leitura e

extração de informações, pelo fato de se associarem com maior magnitude com o corrente trabalho. Dentre eles, (25) traz o desenvolvimento de uma LINDE para processamento de imagens, o PolyMage. Em síntese, o usuário possui vários algoritmos e funcionalidades dispostas em que o mesmo poderá usar para construir uma aplicação sobre o domínio do PDI. O programa elaborado é assemelhado a um grafo de estágios, onde há uma fonte e uma saída, e dentre elas, uma sequência de operações é realizada. Trata-se, em termos curtos, de um processo inverso daquele proporcionado por uma LIDPV, onde o código construído pode ser visto como uma composição gráfica.

Outro estudo primário considerado ao final foi o ambiente de programação visual TiViPE (26). Trata-se de um espaço de desenvolvimento onde o usuário cria uma rede de conexões com componentes interativos. Cada componente é representado por uma unidade computacional, e o ambiente disponibiliza inúmeros elementos para criação, permitindo a construção de aplicações de subáreas distintas, sem um foco necessariamente localizado. Além deste fator, o TiViPE é também um gerador de código para C++.

Além destes estudos, também foi elencada a ferramenta VPL, desenvolvida por (27) no início da década de 1990, que visava utilizar a interface gráfica como meio facilitador para programação, voltado para PDI. Assim como nos outros trabalhos citados, o usuário trata de construir e conectar uma sequência de componentes, estes que formarão uma aplicação que utilizará uma imagem como meio de entrada e de fluxo. Possui uma série de funções e algoritmos de PDI já encapsulados, porém, trata-se apenas de um ambiente de compilação que trabalha com as imagens inseridas, e que não contém um gerador de código inserido.

Associando estes trabalhos ao Mosaicode e à pesquisa aqui proposta, podemos verificar várias similaridades entre ambos. O TiViPE por exemplo, tem uma estrutura de composição bastante parecida a aquela utilizada no Mosaicode, onde o usuário tem um conjunto de funcionalidades dispostas para construir sua aplicação. Além disso, as operações oferecidas são sobre o domínio de PDI, onde há uma imagem fonte, uma sequência de operações, e por fim, uma saída. Entretanto, a forma de inserir os blocos se assemelham ao modo como o Pure Data funciona, inserindo caixas nulas e digitando o nome do componente, fato que obriga ao usuário conhecer quais as funcionalidades a ferramenta o proporciona; o Mosaicode, em contramão, oferece uma sub-janela com todos plugins acessíveis.

Já o PolyMage, trata a elaboração dos programas em um modelo inverso ao utilizado pelo Mosaicode. Ao contrário de criar aplicações gráficas e gerar o código através das mesmas, ele permite ao usuário codificar textualmente para construir suas aplicações. Ela demonstra um fluxo de dados correspondente a um programa,

onde é possível visualizar se o repasse ocorreu da forma como planejada.

Por último, o funcionamento da ferramenta VPL torna-se, em teoria, similar aos pilares de nosso projeto. Ela implementa a ideia de programação visual assim como o Mosaicode, e disponibiliza variados conceitos e algoritmos de PDI para que o usuário consiga construir as aplicações deste meio. Porém, as aplicações somente podem ser construídas e executadas adentro ao ambiente, se diferenciando do Mosaicode, que permite ao usuário exportar o código gerado ou até mesmo, editá-lo em tempo real. No mais, é válido ressaltar que os trabalhos encontrados foram realizados há, em média, mais de uma década, onde estes utilizaram tecnologias e embasamentos que hoje estão em defasados ou sequer são utilizados. Além disso, nenhum deles aborda sobre o domínio de VC, sendo este o fato que nos difere fortemente dos demais.

3 O Ambiente Mosaicode

O Mosaicode é um ambiente de programação visual para linguagens de domínio específico (LINDEs), com a finalidade da geração de código. O usuário tem a sua disposição um determinado conjunto de blocos que podem ser conectados entre si, gerando um diagrama. Um diagrama forma e gera código para uma aplicação específica, que pode também ser executada internamente ao ambiente.

Devemos constar primeiramente que Mosaicode é uma ferramenta extensível, e que em princípio, não possui qualquer funcionalidade inicial. Após instalar a ferramenta, o usuário deve também instalar o que denominamos de extensões. Uma extensão é uma disposição de blocos e conexões para o Mosaicode, onde é definido um domínio e um *framework* (LINDE) para geração do código. Desta forma, podemos dizer que o objetivo de criar um pacote de funcionalidades específicas para o Mosaicode seria a mesma proposição de criar uma extensão para o ambiente.

Uma extensão do Mosaicode é composta de blocos, portas e um padrão de código para geração da aplicação. Os blocos podem ser definidos como unidades de funcionamento mínimo da aplicação, sendo que cada bloco possui uma funcionalidade específica do domínio trabalhado, no caso deste trabalho, PDI ou VC. Os blocos são agrupados na ferramenta por categorias, onde cada uma delas representa um paradigma de funcionamento; desta maneira, blocos que tenham um mesma norma de funcionamento pertencerão a uma mesma categoria.

Usualmente, a maioria das funcionalidades aplicada aos blocos possuem parâmetros que alteram seu modo de funcionamento ou execução. Assim, cada bloco possui associado a si, quando necessário, um conjunto de propriedades, que manipulam os seus parâmetros. Consequentemente, o usuário possui uma maior amplitude de possibilidades e variedades no momento da construção de aplicações.

Outra unidade essencial a uma extensão são as portas. As portas são definidas como objetos de interligação entre os blocos. Logo, cada bloco em específico deve conter um conjunto não nulo de portas para se conectar com outros blocos. Cada porta possui um tipo específico de dado para transmitir, como números inteiros, *strings*, ponto flutuante, imagens, sons, dentre outros, onde isto depende indubitavelmente da extensão e do domínio em questão. Desta forma, uma porta só se conectará a uma porta de um outro bloco caso ambas sejam do mesmo tipo.

Além dos blocos e das portas, uma extensão também deverá possuir um padrão de código, responsável por unir e gerar toda a aplicação. Trata-se de um arquivo padronizado, que possui o que é comum a todo programa que utiliza a LINDE determinada. Além disto,

o padrão irá acoplar as partes dinâmicas, onde ingressarão os blocos com seus devidos códigos, além das conexões realizadas entre eles (código referente às portas). Na Figura 3 é exibida uma visão geral da ferramenta, onde é empregada uma extensão para síntese musical utilizando JavaScript / WebAudio.

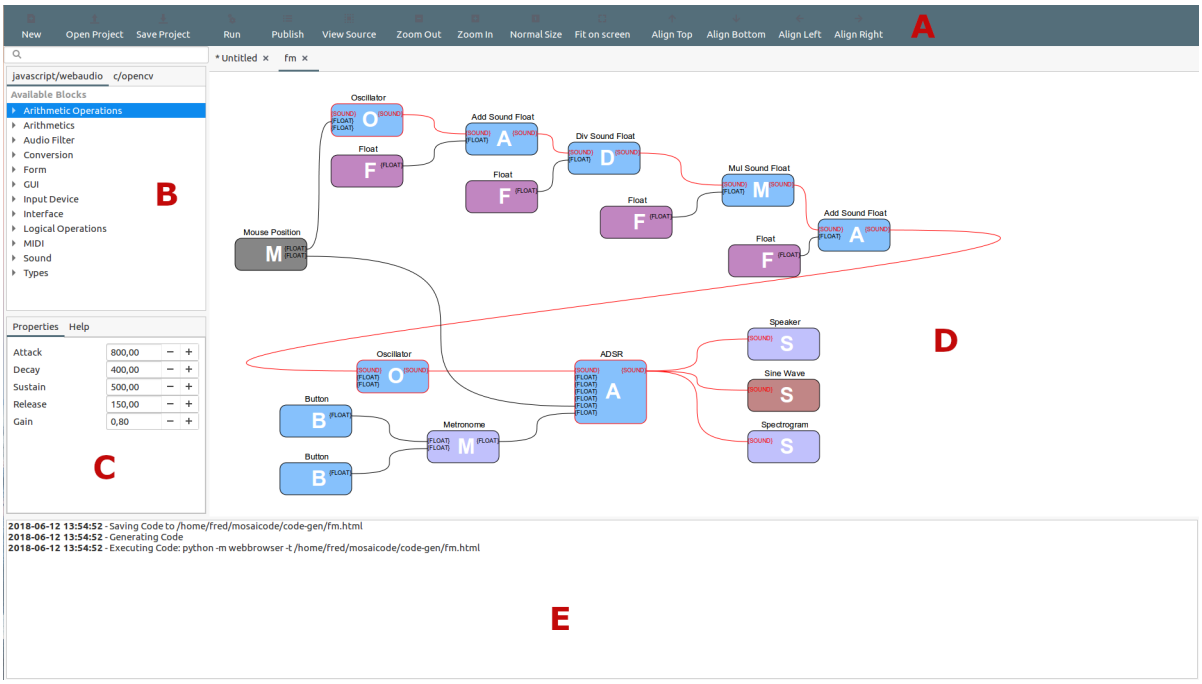


Figura 3 – Visão geral da área de trabalho do ambiente Mosaicode.

Observa-se na Figura 3, na área demarcada por “A”, uma barra de ferramentas que conta com operações usuais, como abrir e salvar arquivos, além de funcionalidades como visualizar o código gerado e executar a aplicação. Em “B”, é exibido as extensões instaladas na máquina e os blocos disponíveis a cada uma delas. Nesta região da ferramenta, encontram-se as categorias que a mesma possui, onde os blocos podem ser arrastados e utilizados.

Em “C”, exibe-se uma sub-janela referente às propriedades estáticas do bloco selecionado. Neste local, o usuário pode alterar os parâmetros da funcionalidade do bloco. Além disto, há a possibilidade dos parâmetros serem alterados por meio das propriedades dinâmicas, que são controladas através das portas. Por exemplo, um bloco que utiliza um número inteiro para controlar determinado parâmetro, pode ter seu valor alterado por meio da sub-janela dos parâmetros (propriedades estáticas), ou então, através de um bloco que emita um valor inteiro, conectando-o diretamente em sua entrada (propriedades dinâmicas).

Já “D” apresenta a área de trabalho da ferramenta. Ali, os blocos são inseridos e conectados através de suas portas, formando os chamados diagramas. Um diagrama é utilizado para gerar o código-fonte e a aplicação final construída. Por último, na área demarcada por “E”, é exibido um histórico geral das operações realizadas pelo usuário,

incluindo operações como como gerar o código, executar e publicar a aplicação em rede.

Abordando especificamente a forma como o Mosaicode funciona, constata-se que ela é semelhante à arquitetura de Fluxo de Dados (*Pipes and Filters*). Este padrão diz respeito a um perfil de execução em que os dados de entrada são transformados nos dados de saída por meio de uma sequência de componentes que alterem e repassem estes dados (28). No caso em questão, podemos dizer que os dados são inseridos como entrada, e transpassarão através de uma série de blocos (*filters*) interligados entre si por suas conexões (*pipes*).

Como já colocado, cada bloco diz respeito a uma função específica; por conseguinte, os filtros da arquitetura são exatamente os blocos do Mosaicode, que condizem a um algoritmo ou operação, onde no padrão em questão são como um conjunto de refinamentos, um após ao outro, até que o diagrama se encerre. Podemos assemelhar tal arquitetura ao domínio de PDI e VC, de forma que cada iteração operacional possa ser representada como uma etapa dentre uma sequência de procedimentos realizados em uma imagem. Na Figura 4a é exibido um esquema representando o funcionamento padrão de uma aplicação que utiliza a arquitetura *Pipes and Filters*, onde podemos realizar uma comparação com a Figura 4b, que demonstra um exemplo de diagrama criado no ambiente Mosaicode.

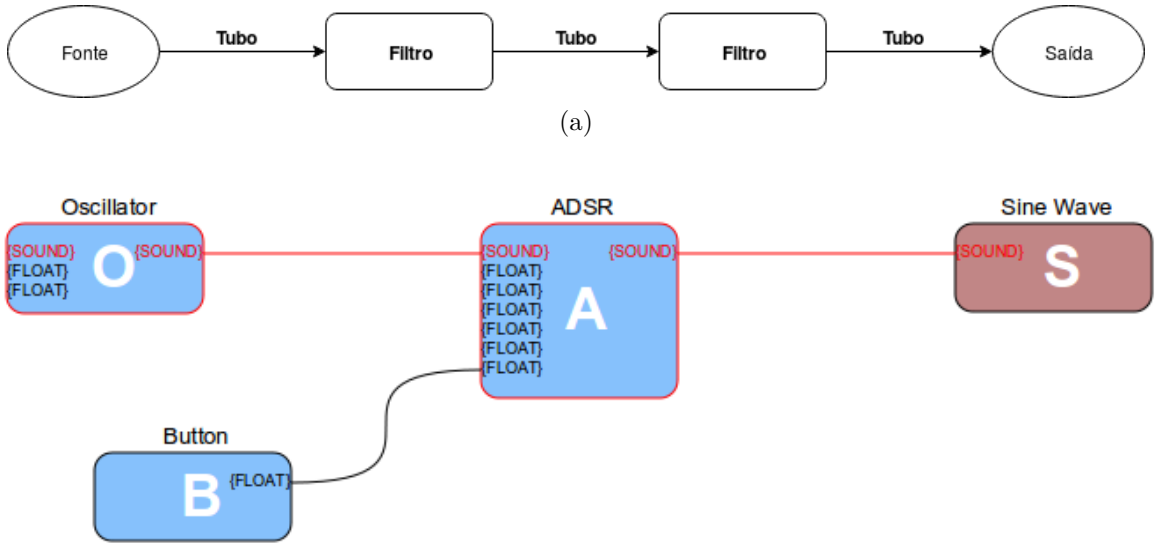


Figura 4 – Em (a) é demonstrado um esquema básico de funcionamento da arquitetura *Pipes and Filters*. Já em (b) é exibido um diagrama construído no Mosaicode utilizando a extensão JavaScript/WebAudio.

Segundo a estrutura disposta na Figura 4a, existe uma fonte e uma saída, que são encadeadas por uma sequência de filtros, conectados por tubos que repassam a informação oriunda da fonte. Logo, é possível visualizar de forma clara a semelhança entre alguns fatores desta com a segunda figura. Neste caso em específico, podemos considerar os blocos *Oscillator* e *Button* como as fontes do diagrama e o bloco *Sine Wave* como a

única saída. O filtro existente neste cenário é interpretado pelo bloco *ADSR*, e o fluxo de informação é transpassado pelas conexões, onde existem dois tipos listados nesta situação: fluxo de som (linha vermelha) e de ponto flutuante (linha preta). Aplicando estes conceitos à metodologia de funcionamento de PDI e VC conjuntamente, podemos idealizar os blocos como filtros, fontes ou saídas; no caso de PDI, poderiam atuar como funcionalidades de refinamento ou tratamento, ou como um algoritmo de detecção específica, em vista de VC.

4 Metodologia

Considerando toda a ambientação disposta nas seções anteriores, o rudimento se dá pelos estudos das áreas envolvidas na ambientação dos blocos que deverão ser criados. Para tal, serão abordados todos os campos básicos e avançados do PDI e também da VC. O conhecimento de ambas as áreas está relacionado ao passo posterior, onde serão elicitados todas as funções e algoritmos que são encapsuláveis e que poderão ser distribuídos como plugins para o ambiente Mosaicode.

Após discutido estes conceitos e também o funcionamento do ambiente Mosaicode, como na última seção, é possível compreender a relação que pode ser estabelecida entre a ferramenta e as áreas de PDI e VC, criando uma nova extensão que venha a abranger estes domínios em conjunção. Desta forma, iremos definir um processo para criação de uma extensão que compreenda ambos os domínios utilizando uma LINDE em particular. Após uma breve análise, definiu-se o uso da biblioteca OpenCV, dado que a mesma possui ampla utilização nos dias atuais (abrigando uma forte comunidade de suporte) e fornece uma vasta disposição de algoritmos e funcionalidades de ambas as áreas (29). Além destes fatores, é válido relatar que a geração de código se dará na linguagem C++, elevando a execução e o desempenho das aplicações geradas, principalmente se compararmos a outras linguagens que também fornecem suporte à biblioteca, como Python e Java.

Retornando ao último passo citado, será realizado o levantamento de funcionalidades e algoritmos fundamentais de PDI, além de operações e tipos básicos de dados essenciais necessários. Posteriormente, para cada funcionalidade listada deve-se reproduzir o cenário de atuação da mesma e identificar as devidas interações possíveis com outros componentes e funções.

Em sequência, estas funcionalidades devem ser idealizadas como blocos. Por este motivo, é extremamente importante a identificação de todo o cenário de atuação de uma funcionalidade, já que a mesma é vista como uma fração de toda uma aplicação final, ou melhor definindo dentro do contexto, como um bloco de um diagrama. Além disso, é indispensável analisar a viabilidade de se implantar a maior quantidade possível de blocos, principalmente se tratando de operações básicas.

Como já premeditado, a etapa seguinte à idealização dos blocos se dará pela criação e incorporação dos mesmos ao ambiente Mosaicode. É importante ressaltar novamente que funções e algoritmos básicos deverão ser primeiramente inseridos, para criar uma estrutura fundamental na construção avançada que virá ao final. Ao fim desta etapa recursiva, obteremos a extensão OpenCV em sua primeira versão, contendo apenas recursos de PDI.

Após esta consolidação inicial da extensão, será necessário a idealização de exemplos de aplicações com os blocos construídos; contextualizando, podem ser definidos como diagramas tutoriais. Estes exemplos auxiliarão e irão exibir como se dá o funcionamento da extensão para novos usuários. Logo após estas etapas, a extensão já poderá ser previamente empacotada e distribuída nos repositórios do Mosaicode como uma primeira versão oficial de distribuição.

O processo de criação da extensão pode ser subdividido em duas fases, onde a primeira é concluída justamente com a distribuição da primeira versão da extensão. A partir deste ponto, inicia-se a segunda fase, responsável por aprimorar a extensão e incluir blocos que constituam funções de VC, complementando o processo de sincronia entre ambas as áreas relatado durante todo o trabalho.

A modelagem do processo de levantamento de requisitos, idealização e encapsulamento dos blocos será novamente utilizada, porém agora, para as funcionalidades de VC. Em síntese, iremos buscar algoritmos que realizem diversos tipos de identificação, seja de pessoas, faces, objetos, formas, dentre outros padrões possíveis. Uma outra vertente de adição de blocos na extensão é voltada para o ambiente artístico digital, buscando construir blocos que retornem efeitos visuais relevantes para artistas que venham à utilizar a ferramenta, ou então, de forma que os blocos de PDI e VC em união possam proporcionar as devidas funcionalidades a estes artistas.

Finalizadas as etapas de criação de novos blocos, é preciso também idealizar exemplos de diagramas que utilizem estes novos blocos, para serem dispostos no Mosaicode com os exemplos mais básicos anteriormente criados. Por fim, se dará finaliza a implementação da extensão OpenCV com uma disposição de variados blocos com diferentes funcionalidades distribuídos em categorias que alinhem unidades com a mesma perspectiva de processamento. Além da distribuição da nova versão da extensão nos repositórios da ferramenta, também é válido aludir a necessidade de uma documentação que facilite o uso dos blocos aos usuários do Mosaicode, e até mesmo que demonstre a criação de um bloco para futuros desenvolvedores da ferramenta.

Em síntese, a Figura 5 demonstra como se dará toda a metodologia para criação e inserção dos blocos que foi descrito. Cada uma das fases mencionadas representam, na verdade, um ciclo, onde a completude de um deles caracteriza a integração de um novo bloco à extensão. Após a Fase 1 ser concluída, obteremos a extensão OpenCV em sua primeira versão, contendo apenas funcionalidades básicas de PDI como mencionado anteriormente. Já ao fim da Fase 2, a criação do pacote de blocos estará finalizada, restando apenas seu empacotamento e distribuição.

Outros componentes extremamente importantes constituem uma extensão genérica para o Mosaicode, entretanto, estes serão considerados a partir do próximo capítulo.

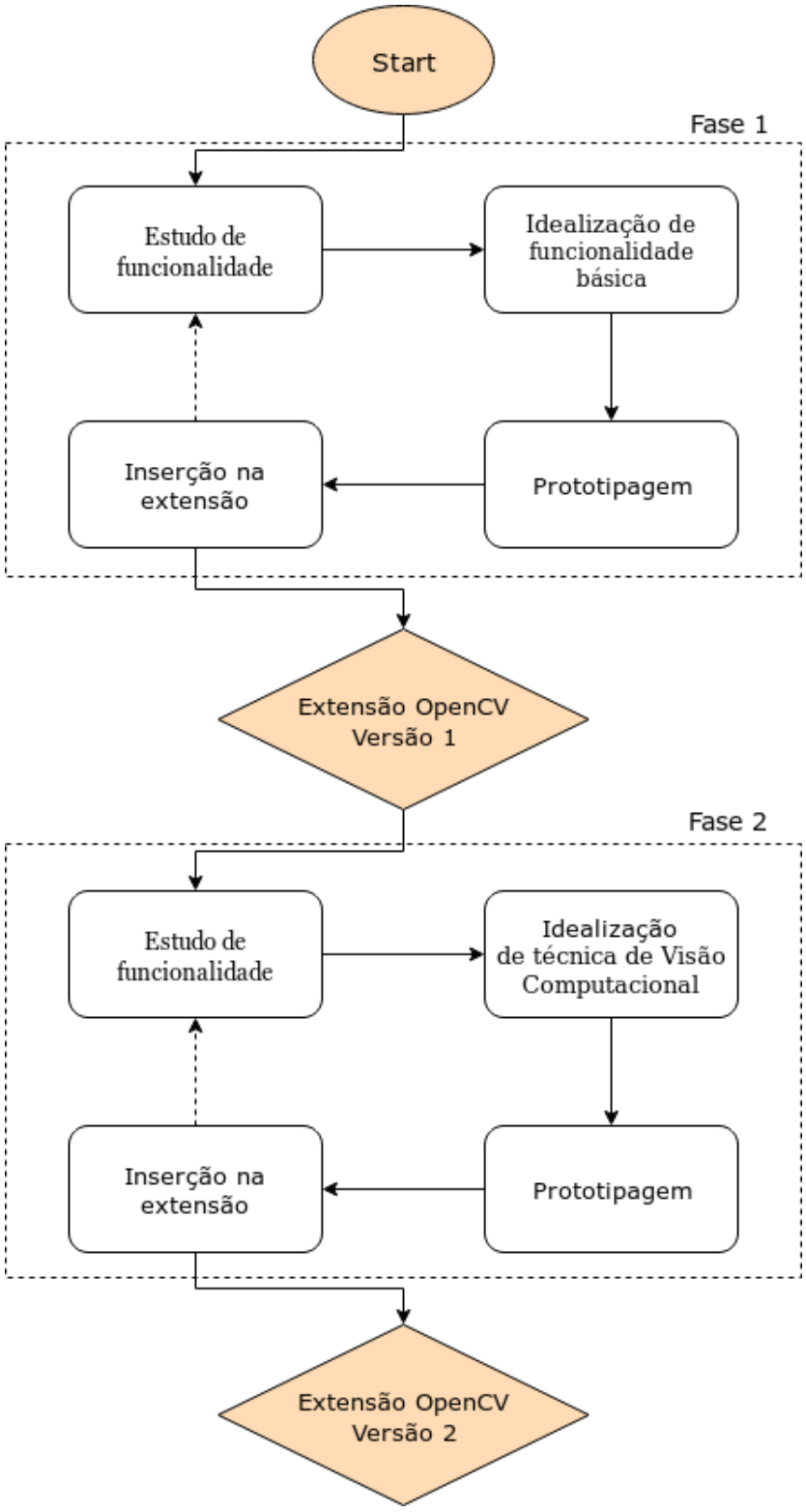


Figura 5 – Fluxograma guia para o desenvolvimento dos plugins da extensão.

5 Desenvolvimento

Como explicado anteriormente, uma extensão é composta por blocos, conexões e um padrão de código gerador. Nesta seção iremos abordar separadamente cada um desses itens e como se dá suas interações.

5.1 Padrão de código

Quando se é tratado sobre o tópico de geração de código, devemos primeiramente analisar todo o ambiente em que isto irá ocorrer, além de se determinar todas as bases necessárias para tal procedimento. Como já citado anteriormente, para a geração de código foi-se escolhida a biblioteca OpenCV na versão 3 utilizando a linguagem C++. Logo, com a definição do *framework* utilizado é possível mapear as situações possíveis e os cenários de funcionamento da mesma.

Se tratando especificamente de um gerador de código, é possível definir a construção do mesmo como uma união de similaridades com encaixes para variações, como descrito em (30). Estritamente, um código qualquer de uma mesma biblioteca em uma mesma linguagem, possui um certo padrão que pode ser identificado e utilizado para realizar a geração. É possível encontrar estes padrões, por exemplo, em um código em C++ que utiliza o OpenCV, e é baseado nesta premissa que o padrão de código da extensão será criado. Desta maneira, o primeiro passo nesta etapa se dá na identificação dos padrões e das variações contidas em um código qualquer dentro deste cenário. Para isto, basta observar um determinado conjunto de códigos que utilizam diferentes funcionalidades, atentando-se principalmente ao comportamento estrutural do OpenCV. Após este estudo, foi-se determinada a estrutura base para a geração de código, demonstrada na Figura 6.

Como definido, o padrão de código deve conter as partições fixas e variáveis, onde na Figura 6, as fixas são representadas pelos blocos esbranquiçados, e as variáveis pelas caixas acinzentadas. As partes fixas serão inseridas automaticamente quando uma aplicação for gerada, pois elas estarão contidas em qualquer programa genérico que utiliza o OpenCV. Já as partes variáveis serão inseridas nos espaços demarcados pelas áreas cinzas, onde estas correspondem aos códigos dos blocos e conexões contidos no diagrama em questão.

Mais à frente, a construção dos blocos será melhor definida; neste momento, é interessante relatar apenas que um bloco deverá conter várias frações de código, onde cada uma delas corresponde a uma das caixas acinzentadas dispostas na Figura 6. Devemos

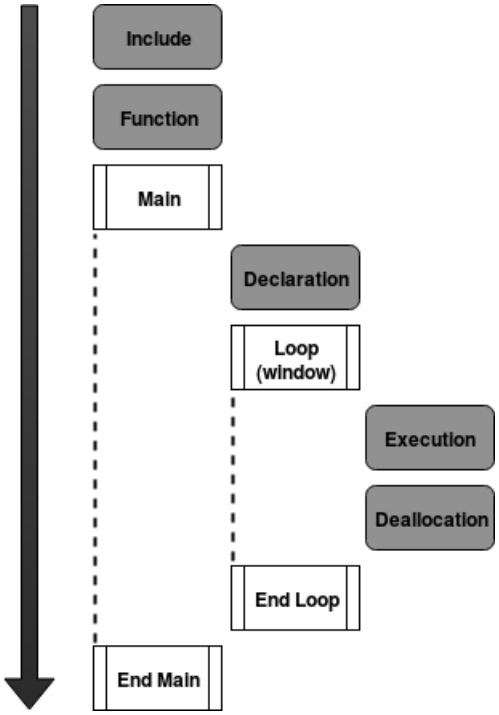


Figura 6 – Estrutura organizacional representativa ao padrão de código idealizado.

então, definir explicitamente o que elas necessariamente representam.

Partindo inicialmente do segmento “Include”, este se diz referente ao fragmento dos blocos onde estarão constadas as referências a todos os arquivos e módulos do OpenCV para importação, estes necessários para a execução. Desta forma, o código que será gerado a partir de um diagrama qualquer conterá a união de todas as frações com o marcador “Include” de cada bloco participante do próprio diagrama. Este procedimento se repete a todos segmentos variáveis (“Include”, “Function”, “Declaration”, “Execution” e “Deallocation”).

Dando sequência ao particionamento de código do padrão definido, a porção “Function” diz respeito às funções específicas para a execução de um determinado bloco, como por exemplo, para realizar conversões internas de tipos e cores. Após esta entrada, são relatadas algumas definições constantes no código, além da função base “Main” iniciada neste ponto.

Após isto, a sequência se dá com a seção “Declaration”, que como o próprio nome já referencia, realiza a declaração de todos os tipos utilizados internamente ao programa gerado. Logo depois, inicia-se uma estrutura de repetição interna que mantém a execução até que o usuário encerre-a pressionando uma tecla. Com esta estrutura “Loop”, é possível realizar por exemplo, capturas de vídeo em tempo real, ou manter uma imagem fixa na tela controlada diretamente pelo usuário.

Adentro à repetição, constam-se duas frações muito importantes. Primeiramente, a “Execution” é responsável por reunir o código de execução de cada bloco independente.

Essencialmente, esta fração contém as funções e todas as operações realizadas com os dados recebidos e repassados bloco por bloco.

Por último, a fração "Deallocation" realiza todo o processo de liberação dos tipos e variáveis utilizados, como imagens e objetos de classificação. Após este elemento, a estrutura de repetição e a função principal são, respectivamente, encerradas, e o programa é então finalizado.

Conclui-se após a definição deste processo que o padrão de código construído será o elemento responsável por unir os fragmentos variáveis encontrados nos blocos e nas portas, atuando como unidade base para a geração de código.

5.2 Portas

Podemos definir as portas como os meios de entrada e saída dos blocos, utilizadas para interação e repasse de fluxo entre os mesmos. Porém, as portas devem possuir uma tipagem estática, de forma que não haja a possibilidade de uma mesma conexão incluir tipos distintos de dados, facilitando a construção e a abstração de diagramas no ponto de vista estrutural. Assim, fez-se necessário realizar um breve estudo de viabilidade para determinar quais os tipos de dados deveriam ser criados para trafegar através das portas, e com isto, estes foram os definidos:

- **Image** (imagem);
- **Int** (números inteiros);
- **Double** (números decimais);
- **Points** (lista de pontos);
- **Rects** (lista de retângulos).

Primeiramente, definido como fluxo principal da extensão, a porta "Image" transpassa todo e qualquer gênero de imagem, variando-se em tipos de cores, canais ou até mesmo em estruturas. Há também portas para tipos de dados numéricos, seja para números inteiros ou para decimais. E por fim, há tipos mais complexos, envolvendo lista de objetos específicos da biblioteca utilizada como pontos ou retângulos. O primeiro pode ser definido também como um tipo para definição de contornos ou formas, já que uma agregação de pontos é comumente utilizada para tal ato; já o segundo será útil a blocos que contenham algoritmos de detecção e extração de dados.

5.3 Blocos

A este ponto é possível abordar de forma clara como se dará o funcionamento e a participação dos blocos no processo da geração de código. Eles representam as unidades mínimas de processamento dentro de um contexto (neste caso, PDI e VC), onde uma combinação dos mesmos gera uma aplicação. Define-se estas unidades de processamento como funções, algoritmos, gerações de tipos de dados, dentre várias outras possibilidades de operações.

Qualquer programa genérico que atue utilizando PDI ou VC, seja em conjunção ou não, exercitam suas operações de forma sequencial, onde o dado é repassado de um procedimento para o outro. Com o OpenCV, as aplicações são criadas desta forma, e deste modo, utilizando tal conceito de sequencialidade, iremos definir os blocos como artefatos operacionais que possuam no mínimo uma interação com um outro artefato qualquer.

Para explicitar melhor esta ideia, utilizaremos como base, a sequência de operações realizada na Figura 7. Ela descreve de forma resumida, uma sequência de passos que realiza a detecção de faces em uma dada imagem de entrada. Trata-se de um processo onde temos inicialmente, a leitura de uma imagem, que é posteriormente convertida a uma escala de tons de cinza, e por fim, é submetida a um algoritmo de classificação para detecção de faces.

Figura 7 – Sequência de operações realizadas para detecção de faces em uma imagem.

O que é necessário enfatizar é que estas operações podem ser subdivididas, como demonstrado na imagem, e não apenas vistas como uma única aplicação para detecção de faces. À visão do Mosaicode, cada uma das etapas realizadas podem ser interpretadas como blocos contidos em um diagrama, o que a conjunção simbolizaria toda a aplicação em si. Logo, a construção dos blocos estaria diretamente ligada às funções e algoritmos possíveis envolvendo PDI e VC, de forma que a junção destes possa formar uma determinada aplicação.

Após estas definições, vale ressaltar que o processo de criação dos blocos deveria seguir a metodologia descrita na Figura 5, onde inicialmente, as funcionalidades devem ser descritas e idealizadas, arquitetando seu funcionamento e sua interação com os demais

artefatos no Mosaicode. Após esta etapa é que a funcionalidade deve ser encapsulada e anexada à extensão; é válido ressaltar que este processo repete-se para cada bloco criado.

Se tratando especificamente à geração de código e ao padrão definido na seção anterior, os blocos atuarão como os encaixes variáveis, fornecendo as funcionalidades determinadas pelo usuário. Os códigos definidos em cada um dos blocos serão anexados ao padrão que contém todas as seções fixas predefinidas, onde restarão apenas as ligações entre esses blocos.

Como os blocos são definidos como as frações variáveis do código a ser gerado, eles serão formados apenas pelas seções acinzentadas destacadas na Figura 6. Ainda assim, elas são opcionais e não necessariamente devem estar presentes em todos os blocos, variando de acordo com a demanda necessária para cada um deles. Além disso, um bloco deve conter a definição de suas portas de entrada e saída, e também das propriedades estáticas disponíveis.

Por conseguinte, com esta metodologia básica para criação, os protótipos foram idealizados e posteriormente encapsulados. Adentro à extensão, eles foram subdivididos em categorias que representam um mesmo paradigma de processamento, listados na Tabela 1. A seguir, discutiremos individualmente estas categorias e seus artefatos construídos.

Categorias	**Blocos**
Arithmetic and Logical Operations	And, Division, Multiplication, Not, Or, Side by Side, Subtraction, Sum, Up to Bottom, Xor
Basic Data Type	New Double, New Int, New Point, New Rectangle
Basic Shapes	Circle, Contours, Ellipse, Line, Rectangle
Experimental	Add Border, Crop Image, Get Size, Invert Image, Resize Image, Rotate Image, Text
Feature Detection	Blurred Face, Color Detection, Eyes Detection, Face Detection, Facial Landmark, Human Detection, Match Object, Shape Detection, Smile Detecion
Filters and Conversions	Adaptive Threshold, Brightness, Color Conversion, Compose Channels, Contrast, Decompose Channels, Histogram, Histogram Equalization, Smooth, Threshold
General	Fill Image, Save Image, Select, Show Image
Gradients, Edges and Corners	Canny, Harris Corner Detector, Laplace, Shi Tomasi Corner Detector, Sobel
Image Source	Image File, Live Mode, New Image, Video File
Math Functions	Exp, Log, Pow
Morphological Operations	Closing, Dilate, Erosion, Opening

Tabela 1 – Categorias e seus respectivos blocos na extensão OpenCV.

5.3.1 Arithmetic and Logical Operations

Esta categoria denomina-se propriamente a operações aritméticas básicas e lógicas, onde possuem sempre duas imagens como entrada e retornam uma outra imagem como resultado do procedimento realizado. As operações aqui definidas executam de forma bit-a-bit, onde as imagens inseridas nestes blocos dever possuir a mesma resolução. Caso as entradas não sejam equivalentes, há uma função interna a estes plugins para igualar a segunda imagem de entrada à resolução da primeira, permitindo a operação *pixel* por *pixel* sem falhas.

Dentre as funcionalidades mais utilizadas nesta categoria encontram-se a soma, subtração, e as operações lógicas de conjunção, disjunção e negação de imagens. Além disso, há também blocos que realizam a união de imagens lado a lado ou uma acima da outra, úteis para realizar comparações de imagens distintas em uma única janela.

5.3.2 Basic Data Type

Esta subdivisão retrata a criação de tipos de dados básicos como números inteiros, reais, pontos e retângulos. Está estritamente ligada aos tipos de portas criados, já que existe um bloco específico para geração de cada um destes tipos, com exceção do tipo *Image* que se localiza em outra categoria.

É notável que como se tratam de funcionalidades de criação de dados para fluxo, seus blocos possuam apenas portas de saída, pois não há demanda possível para que existam portas de entradas.

5.3.3 Basic Shapes

Já esta seção é destinada à síntese de formas básicas geométricas como círculos, retângulos, elipses, linhas e contornos. Esta geração pode ocorrer de duas maneiras, onde a mais comum delas trata de inserir uma imagem de entrada e definir as propriedades necessárias à síntese de cada bloco, como tamanho, posição e cor; todos os blocos desta categoria possuem apenas uma porta de saída, onde esta refere-se à imagem de entrada em conjunção com a forma devidamente sintetizada.

O outro modo possível de realizar a criação das formas é por meio da interação do próprio bloco com outros que tenham como saída um vetor de retângulos ou contornos. Neste caso, o usuário precisa apenas inserir a imagem e o vetor em questão, e a síntese será criada sem a necessidade de definição parâmetros.

Para exemplificarmos, utilizaremos a funcionalidade definida na Figura 7, onde iremos considerar que queremos projetar em uma outra imagem, o desenho de um círculo para cada um dos rostos detectados, exatamente na posição em que foram encontrados

pelo algoritmo. Para tal, basta o usuário exportar o vetor de retângulos (áreas das faces) na saída do bloco de detecção para o bloco de síntese, onde o este irá utilizar dados como a posição do retângulo para criar os devidos círculos.

5.3.4 Experimental

Esta divisão abrange blocos com aplicabilidades consideradas alternativas e que não necessariamente representam conceitos de PDI e VC. Dentre os blocos aqui alocados, existem aqueles com funcionalidades para criação de bordas, rotação, inversão e recorte de imagens, inserção de textos, e outros variados.

5.3.5 Feature Detection

Esta é uma das categorias com as maiores quantidades de blocos da extensão, onde a mesma consiste em funcionalidades para detecção de padrões, abrangendo uma vasta variação de algoritmos de VC, porém ainda utilizando técnicas de PDI. Para definir melhor, explicitaremos alguns dos principais blocos aqui idealizados.

Iniciando-se pelos mais simples, há um bloco voltado para detecção de cores específicas já padronizadas pela ferramenta. Neste caso, foi previamente definido um intervalo de valores dentro do sistema de cores HSV, de forma a elicitar na imagem todos os *pixels* correspondentes a este intervalo. Há também um bloco para detecção de formas geométricas, em que se é calculado o número de faces, correlato a um modelo geométrico. Ambos possuem como saída uma imagem com as detecções elicitadas e um vetor com os contornos (combinação de pontos) detectados.

Exploraremos outra vertente, agora adentrando ao lado da VC com o auxílio do Aprendizado de Máquina, um campo da computação que possibilita o reconhecimento de padrões e a aprendizagem dos computadores sem serem explicitamente programados (31). Utilizando conceitos destas subáreas, empregamos os classificadores Haar Cascade, que facilitam o treinamento de bases para detecções específicas e que já são disponibilizados de forma simplificada pelo OpenCV (32) (33).

Os blocos Face *Detection, Smile Detection* e *Eye Detection* têm exatamente o mesmo princípio de funcionamento. Uma imagem de entrada é inserida e convertida para uma escala em tons de cinza, onde posteriormente um classificador Haar Cascade, que foi previamente treinado com uma base de dados específica, é utilizado para reconhecer padrões localizados na própria imagem. A distinção entre cada um dos blocos citados se dá justamente na base de treinamento utilizada, já que cada uma delas diz respeito ao padrão no qual se deseja encontrar, armazenando os devidos dados correspondentes à tal. Ambos os blocos possuem como saída duas alternativas: uma imagem com os padrões encontrados ou um vetor de retângulos com o posicionamento dos devidos artefatos identificados.

Além das unidades construídas já citadas, foram idealizados também blocos para detecções de pessoas, identificação de objetos com repetições em diferentes imagens, suavização de faces, dentre outros que possuem uma base de funcionamento bastante similar aos anteriores.

5.3.6 Filters and Conversions

Nesta categoria estão listados algumas conversões de sistemas de cores e os filtros passa-baixa (por opção de implementação, uma categoria específica a filtros de passa-alta e de aguçamento foi criada). Primeiramente, foi-se construído plugins que envolvam operações essenciais, como ajuste de brilho e contraste, e para conversão de cores, envolvendo os principais sistemas como RGB, HSV, Gray, HLS, dentre outros. Além destes citados, há também dois plugins que trabalham com a composição e decomposição de canais dos mais variados sistemas de cores, mesclando ou separando as devidas camadas. Outra funcionalidade fundamental implantada é a equalização do histograma gerado a partir de uma imagem, atéatuando com sistemas multicanais.

Tratando-se dos filtros abordados, foi-se idealizado um bloco que reúne variados filtros de passa-baixa (suavização), permitindo ao usuário optar por uma das alternativas dispostas nas propriedades do próprio bloco. Dentre estas opções estão os filtros de média, mediana e gaussiano, onde somente uma das opções é executada de acordo com a escolha do usuário.

Outro bloco disponibilizado nesta categoria executa o processo de limiarização, uma técnica de segmentação de imagens que decompõe a entrada em dois grupos opostos ao limiar, de forma a distinguir grandes variações. O bloco possibilita a abordagem dos diferentes tipos básicos da operação, como os modos binário, truncado, e até mesmo utilizando a binarização de Otsu, método normalmente aplicado para encontrar um melhor limite para uma determinada imagem a se aplicar a limiarização (34). Há também outro artefato, onde este implementa a funcionalidade do limiar adaptativo, método que emprega limiares locais de forma a tornar a própria operação mais adequada a, por exemplo, imagens que possuem um alto desnivelamento de luminosidade.

5.3.7 General

Corresponde a uma das categorias com funcionalidades básicas e usuais. Dentre as possíveis, existem operações para geração de janelas para visualizações de imagens, seleção de fluxo, gravação e preenchimento de imagens. Geralmente, os blocos aqui dispostos são utilizados apenas para operações de fim de diagramas.

5.3.8 Gradients, Edges and Corners

Nesta seção, o foco se dá na criação de artefatos que contenham algoritmos para detecção de bordas e vértices, podendo inferir informações dos dados encontrados. Dentre os blocos aqui construídos, estão algumas operações clássicas do PDI como as de Laplace e Sobel, que claramente poderiam ser enquadrados na categoria "Filters and Conversions", pois trata-se de filtros passa-alta. Porém, definiu-se como uma decisão de implementação, a criação desta categoria em específico para reunir algoritmos que focassem na identificação de bordas e cantos, integrando os filtros citados acima.

Além destes, foi-se implementado o algoritmo desenvolvido por John Canny, que baseia-se na aplicação do filtro gaussiano, seguido do cálculo do gradiente e de uma limiarização, onde o resultado final elicita os vértices e as linhas (bordas) mais intensas em uma imagem (35).

Buscando algoritmos mais rebuscados e recentes, reproduzimos primeiramente o detector de Harris (36). Trata-se de um algoritmo muito utilizado para a detecção de cantos que superou comprovadamente o detector de Moravec, que também tratava a inferência de vértices em uma imagem. Após este, foi-se produzido um bloco para realizar a detecção de vértices elaborada por Shi e Tomasi (37). Ela consiste em definir os cantos mais fortes e consistentes da imagem, organizando-os em uma lista decrescente de acordo com o índice de potência recebido por cada um dos vértices identificados.

5.3.9 Image Source

Assim como a categoria General, esta é utilizada para operações usuais para geração de imagens e *frames*. Entre as possibilidades, o usuário pode optar por gerar a imagem através de um arquivo local, por meio de uma câmera (modo em tempo real), por um arquivo de vídeo ou criando uma imagem sintética preenchida por uma cor definida em RGB.

5.3.10 Math Functions

Outra decisão de implementação tomada na construção dos blocos diz respeito a esta categoria, que contém operações matemáticas extras em relação à primeira categoria listada. A principal diferença em relação à seção de *Arithmetic and Logical Operations*, é que os blocos pertencentes a esta executam operações unárias, e portanto, possuem apenas uma imagem como entrada. Neste caso, entre as funções possíveis estão a potenciação, exponenciação e o logaritmo de imagens.

5.3.11 Morphological Operations

Por último, esta seção reúne os blocos que exercitam operações morfológicas nas imagens como a dilatação, erosão, fechamento e abertura. Vale ressaltar que todos os blocos desta categoria possuem a mesma quantidade de portas de entrada e saída, além de que todos disponibilizam os mesmos parâmetros para definição do tamanho da máscara aplicada na operação em questão.

5.4 Visão Geral

Por fim, a extensão teve sua implementação finalizada com um total de 11 categorias de blocos, totalizando 65 funcionalidades distintas distribuídas por estas mesmas categorias. Elas possibilitam ao usuário a construção de uma vasta gama de aplicações, incluindo os mais variados conceitos e algoritmos de ambas as áreas, como filtros, gerações de imagens, operações matemáticas, dentre outros. Além disso, foram cinco tipos de portas implantados, cobrindo uma vasta variedade de possibilidades de transmissão dos dados.

Como explicitado anteriormente, o padrão de código idealizado é responsável por fixar as seções invariáveis e incluir os trechos oriundos dos blocos utilizados pelo usuário no diagrama. Além disso, haverá uma linha de código incumbida de transpassar os dados, conforme as devidas conexões realizadas na montagem da aplicação.

Além destes itens, a extensão leva consigo um conjunto de arquivos para execução interna ao Mosaicode, como por exemplo, algumas imagens predefinidas que o usuário poderá utilizar em seus diagramas. Também é inclusa uma coleção de arquivos *XML* e *YAML*, que constituem as bases de dados para classificação nos algoritmos de detecções explanados na seção anterior. Outro elemento distribuído conjuntamente à extensão é o pacote exemplos, onde estes são designados como diagramas que elucidam o funcionamento dos blocos, conexões e das várias possibilidades que a ferramenta proporciona ao usuário.

Por fim, a extensão em sua versão final foi empacotado e distribuído através dos repositórios da ferramenta no Github[1] e via repositórios Python[2] (Pypi), como já havia sido estipulado.

[1] Link do repositório: https://github.com/Mosaicode/mosaicode-c-opencv
[2] Link do repositório: https://pypi.org/project/mosaicode-lib-c-opencv

6 Resultados

Neste capítulo, iremos abordar o funcionamento da extensão já vinculada ao ambiente Mosaicode, além de discutir algumas aplicações possíveis com a disposição de blocos criados. Após a segunda etapa de desenvolvimento demonstrada no Capítulo 4, totalizaram-se 65 blocos envolvendo diversas funcionalidades factíveis, incluindo algoritmos, operações, gerações de tipos, dentre várias outras já listadas anteriormente.

Além dos blocos, a extensão é composta por cinco naturezas de conexões que interligam estes mesmos blocos. Manuseando estes artefatos, diversos diagramas foram idealizados e entregues conjuntamente à extensão, abordados como pequenos tutoriais para exemplificação de uso dos blocos e também das portas. A seguir, explanaremos alguns destes diagramas que tratam dos mais variados paradigmas de funcionamento referentes aos blocos construídos.

Primeiramente, abordando uma composição trivial, elucidaremos o diagrama disposto na Figura 8. O mesmo é integrado por quatro unidades distintas com diferentes funcionalidades, onde é definido uma fonte e uma saída genérica. Para facilitar a compreensão, um diagrama normalmente possui seu fluxo estruturado da esquerda para a direita. Partindo desta definição, a fonte é interpretada pelo bloco *Image File*, responsável por coletar uma imagem local e repassá-la adiante; esta mesma imagem é repassada à entrada de dois outros blocos. Sequentemente, o artefato *Erosion* aplica a operação morfológica de erosão no *frame* de entrada, e os dados são transpassados a uma das entradas disponíveis no bloco *Subtraction*. Neste último, é então realizada uma operação bit-a-bit de subtração da imagem inicial (primeira porta de entrada do bloco) pela imagem gerada na saída do bloco *Erosion*. O resultado obtido é então repassado à unidade *Show Image*, que irá criar uma janela e exibir a imagem processada pelo diagrama.

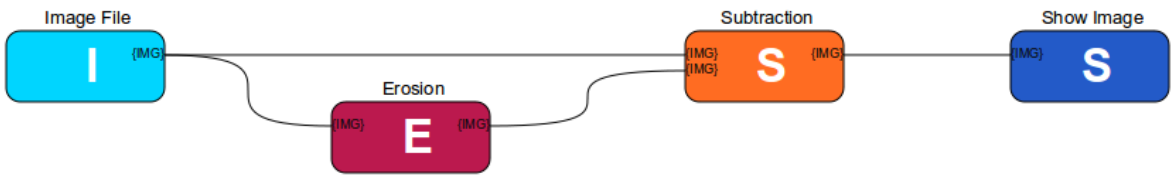


Figura 8 – Diagrama realizando uma subtração de imagens criado através da extensão OpenCV.

Em síntese, a aplicação acima simplesmente realiza uma subtração da imagem original pela própria imagem erodida, exibindo o resultado em sequência. Vale ressaltar que a erosão é uma operação morfológica utilizada geralmente para diminuir partículas, aumentar vácuos ou para separar componentes conectados em uma imagem. A subtra-

ção da imagem original pela própria imagem erodida retorna como resultado apenas os contornos mais fortes encontrados na imagem (reforçados pela erosão), onde o restante se mantém em preto. Podemos visualizar este feito nas Figura 9a e Figura 9b, que apresentam, respectivamente, as imagens de entrada e saída do diagrama em questão.

(a)

(b)

Figura 9 – Em (a), é exibida a imagem de entrada do diagrama. Já em (b), é apresentado o resultado obtido do diagrama composto na Figura 8. Fonte: <https://pt.freeimages.com>.

Dando sequência aos diagramas criados como exemplos de utilização da extensão, a Figura 10 retrata um esquema distinto do apresentado anteriormente. Este por sua vez, possui três ramificações notáveis, onde é possível observar uma saída (*Save Image*) destinada a cada um dos ramos criados. Outra forma de se analisar a estrutura é particionando o diagrama em outros três, onde cada um deles é visto como uma subaplicação única. Especificamente neste programa, foram utilizados blocos das mais diversas categorias, com funcionalidades de diferentes paradigmas.

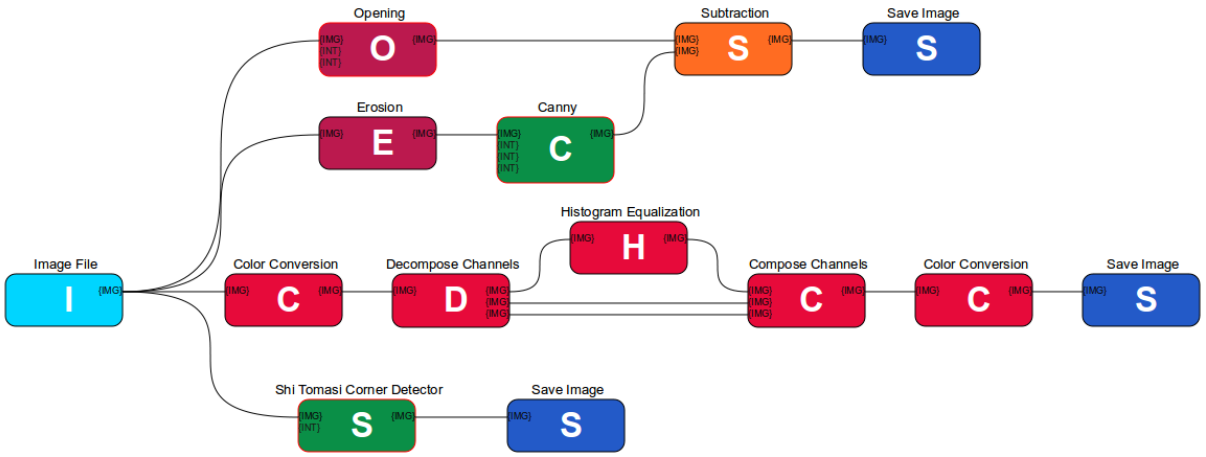


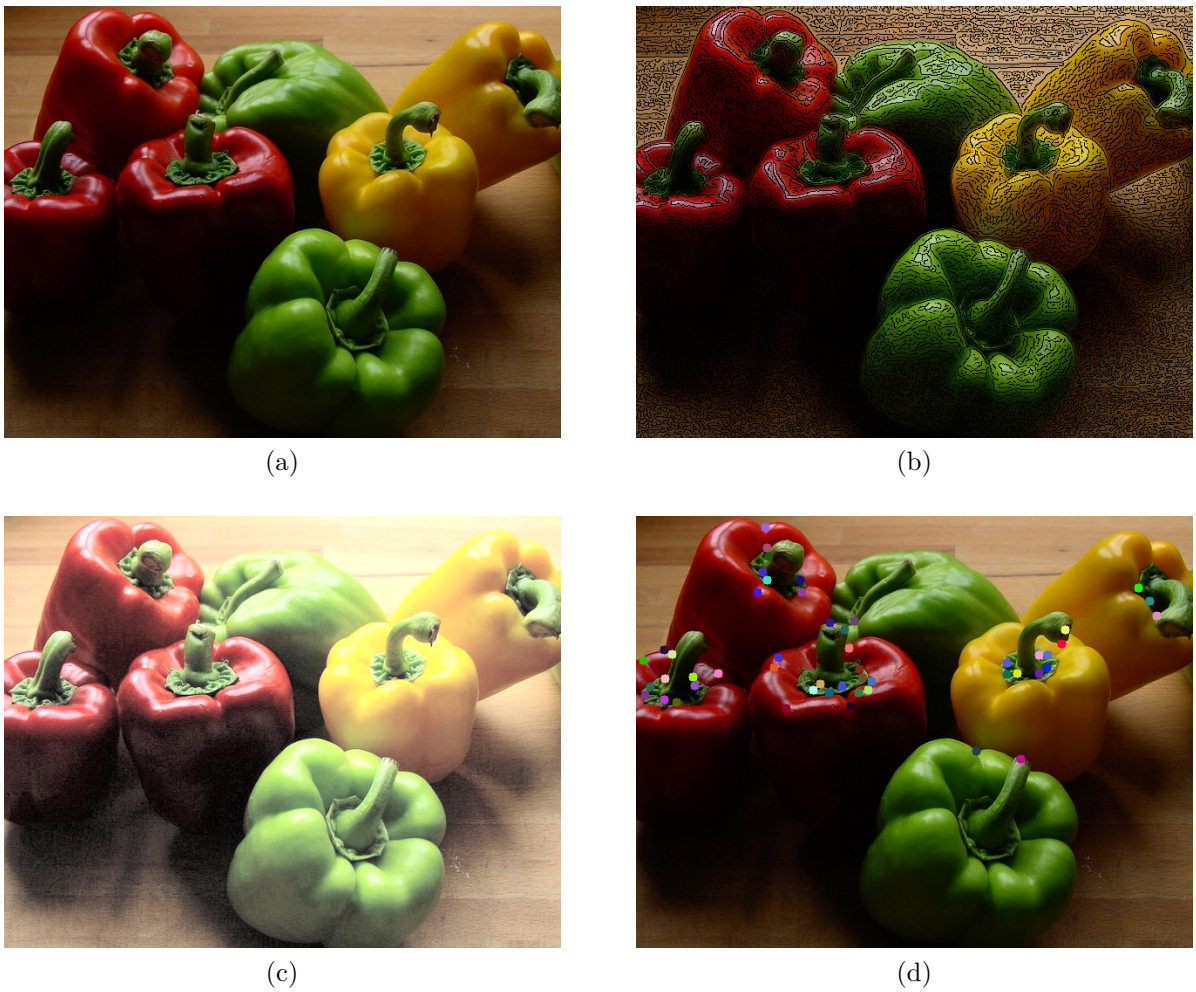
Figura 10 – Outro diagrama criado utilizando a extensão OpenCV, este por sua vez, apresenta várias ramificações, que resultam em saídas distintas.

Abordando especificamente o diagrama em questão, é perceptível que apesar das ramificações existentes, há apenas uma fonte de imagem para todas elas. Iniciando a análise no sentido vertical, de cima para baixo, veremos uma subaplicação que assemelha-se ao diagrama disposto na Figura 8. O que ocorre é uma subtração de imagens, onde a primeira entrada da própria operação é a imagem fonte após uma aplicação da operação morfológica de abertura, e a segunda entrada corresponde à mesma imagem fonte, após transpassar, respectivamente, por uma erosão e por uma aplicação do filtro de Canny, algoritmo que realiza a detecção de contornos em uma imagem. O resultado obtido é demonstrado na Figura 11b, onde os traços pretos criados na imagem correspondem aos contornos encontrados pelo bloco *Canny*, já que estes foram utilizados como os subtraendos em *Subtraction*.

Seguindo nos ramos do diagrama, a segunda subaplicação realiza uma equalização de histograma na imagem de entrada. Inicialmente, por questões de otimização, a imagem é convertida do sistema de cores RGB para o sistema YCbCr, e posteriormente, será devolvida ao formato RGB, mantendo uma parcela mínima de perda das cores durante o processo de conversão (38). Após a conversão inicial, a imagem é decomposta em seus devidos canais (Y, Cb e Cr), onde posteriormente, a equalização do histograma é realizada apenas no canal Y, representante do fator *luma* no sistema, responsável por armazenar o brilho da imagem. Após isto, o bloco *Compose Channels* realiza a união dos canais, e o

resultado é então convertido novamente ao sistema RGB. A imagem obtida é exibida na Figura 11c.

Por fim, o último ramo do diagrama trata-se de uma aplicação simplificada, se comparada às outras duas anteriores. A imagem de entrada é repassada ao bloco *Shi Tomasi Corner Detector*, que realiza a detecção dos K vértices mais fortes encontrados pelo algoritmo. Nesta situação, o número de vértices definidos para serem apresentados foi 30, onde eles podem ser visualizados na Figura 11d, similares a pequenos círculos preenchidos e coloridos. É válido ressalvar que a saída é comum à cada uma das ramificações explanadas, onde a imagem não é exibida, mas sim salva em diretório local.



(a) (b)

(c) (d)

Figura 11 – Em (a), é exibida a fonte de entrada do diagrama. Já (b), (c) e (d), respectivamente, representam as saídas geradas pelas três diferentes subaplicações exemplificadas no texto, na ordem em que foram expostas. Fonte: <https://pt.freeimages.com>.

Prosseguindo a outro exemplo embarcado conjuntamente à extensão, a Figura 12 apresenta um diagrama que realiza a síntese de círculos e retângulos a partir de detecções realizadas em uma imagem de entrada.

Inicialmente, a fonte de imagem é gerada através do bloco *Live Mode*, que cria

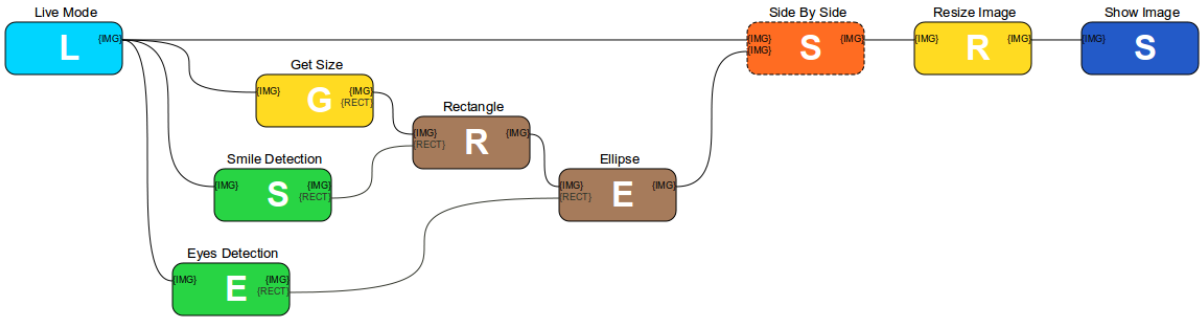


Figura 12 – Diagrama criado utilizando a extensão OpenCV, utilizado para controlar a síntese de círculos e retângulos com os olhos e sorrisos.

uma transmissão recorrente de *frames* gravados a partir de uma câmera conectada ao computador. Esta mesma imagem é repassada a 4 blocos, onde 2 deles realizam detecções de olhos e sorrisos, gerando como saída uma união de retângulos com os parâmetros devidamente detectados.

Outro repasse realizado pela fonte é para o bloco *Get Size*, onde este coleta as dimensões da imagem inserida e recria uma nova, com a mesma proporção. Voltando às detecções, após identificar as regiões de interesse com os algoritmos de reconhecimento, o repasse é realizado aos blocos de síntese de elipses e retângulos, onde ambos são compostos na mesma imagem. Por fim, o bloco *Side By Side* integra o *frame* fonte e a imagem sintetizada lado a lado, expondo o resultado demonstrado na Figura 13, onde as elipses amarelas correspondem aos olhos detectados, e os retângulos azuis aos sorrisos.

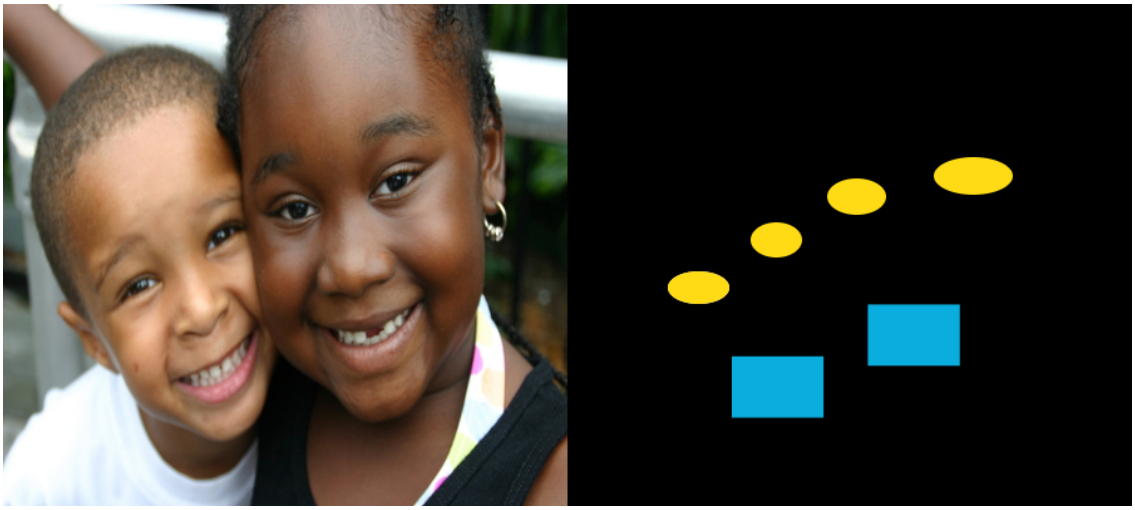


Figura 13 – Saída exibida na execução do diagrama disposto na Figura 12, onde a imagem no lado esquerdo corresponde à uma fotografia representativa para simular a entrada, e no lado direito é exibida síntese realizada. Fonte: <https://pt.freeimages.com>.

É válido relatar que há ainda outros exemplos que foram encapsulados com a extensão, porém, os aqui relatados abordam todas as vertentes que a ferramenta oferta ao usuário. Além do mais, os 65 blocos embutidos, conjuntamente, possibilitam ao usuário

uma alta ortogonalidade, permitindo criar uma vasta gama de aplicações com conceitos de ambos os domínios utilizados. Por tal motivo, foi-se exibido aqui apenas os três exemplos anteriores, em vista de que os mesmos demonstram de forma eficiente a capacidade proporcionada pela extensão.

6.1 O Chaos das 5

Nesta subseção, abordaremos brevemente o primeiro uso real da extensão OpenCV, no espetáculo O Chaos das 5. Trata-se de uma apresentação performática que envolve computação, artes cênicas, música, dentre outras áreas, arquitetada pelos laboratórios ALICE (Arts Lab in Interfaces, Computers, and Else) e Ecolab (Laboratório de Ecopoéticas), pertencente ao Grupo de pesquisa Transdisciplinar (Gtrans), ambos da Universidade Federal de São João del-Rei.

O espetáculo trata de abordar a performance em um ambiente imersivo, no qual o público participa e interage diretamente com a execução da peça, onde é possibilitado que o espectador utilize um celular para conectar-se a um servidor local disponibilizado. Após realizada a conexão, o espectador pode interagir executando amostras de sons pré-definidas que possuem uma ligação direta com a execução geral da peça.

Em síntese, a apresentação ocorre em três segmentos distintos: uma cena inicial representando um mundo sintético, outra abordando o domínio real, e uma última demonstrando um ambiente surrealista. A Figura 14 retrata a captura de uma fotografia durante uma apresentação recente da peça.

Figura 14 – Fotografia capturada durante uma execução da performance no 5º Seminário de Artes Digitais em Belo Horizonte. Fonte: Thiago de Andrade Morandi.

Explanando as cenas construídas, o primeiro segmento utiliza conceitos sintéticos para as projeções realizadas, assim como para o ambiente musical executado pelos músicos; não somente, o público pode interagir e participar deste momento com os sons também sintéticos elaborados, como já mencionado. As projeções exibidas nesta cena utilizam conceitos e aplicações criadas com Síntese de Imagens, mais uma das subáreas pertencentes à Computação Gráfica (assim como PDI e VC).

Já na segunda cena, o ambiente dispensa as projeções e sons sintéticos, exibindo vídeos reais nos projetores, e contando com uma aclimatação sonora produzida por guitarras. Por último, o terceiro segmento aborda um mundo sinestésico e surrealista, utilizando a extensão OpenCV para realizar as projeções.

Foram construídos com o auxílio da ferramenta, os mais variados efeitos que retratam um ambiente surreal, utilizando como entrada câmeras que captam os artistas e projetam imagens de reversão à segunda cena do espetáculo. Nos efeitos elaborados foram utilizadas diversas técnicas de PDI e VC, como conversões de cores, detecções de contornos e operações matemáticas que produzem uma sensação clara de surrealismo. Vale ressaltar que o Mosaicode possibilita a permutação dos blocos em um diagrama, o que neste caso, facilitou a criação de inúmeras aplicações utilizadas na peça, que inicialmente aparentam ser distintas, mas que apenas cambiaram a ordem das operações e geraram um resultado adverso ao anterior. Na Figura 15 é exibida uma fotografia do espetáculo que retrata melhor a projeção das imagens.

Figura 15 – Outra fotografia capturada durante uma execução da performance no 5º Seminário de Artes Digitais em Belo Horizonte. Fonte: Thiago de Andrade Morandi.

7 Conclusão

A programação visual é um conceito que tem crescido de uma forma exponencial, sendo muito utilizada desde em *softwares* para geração de interfaces gráficas até em scripts para banco de dados. Através dessa atual demanda é que o ambiente Mosaicode surge unindo diferentes campos da computação com o propósito de facilitar a geração de Arte Digital e da utilização de LINDEs em modo gráfico para o âmbito educacional.

Tratando-se especificamente da extensão OpenCV, o desafio de elaborar e construir um conjunto de funcionalidades para um *software* de programação visual torna-se a própria motivação de fato, ao ponto em que consideramos que a ferramenta e sua devida extensão em desenvolvimento possa ser utilizada futuramente por alunos e pessoas que desejam aprender PDI ou VC, assim como artistas que pretendem criar suas próprias aplicações visuais digitais, porém possuem suas respectivas dificuldades no momento de colocar suas ideias em prática.

Foram dois anos de pesquisa dedicados exclusivamente à idealização e ao desenvolvimento da extensão, assim como de algumas funcionalidades particulares do ambiente Mosaicode. Durante este período, o trabalho realizado resultou em algumas contribuições científicas. Primeiramente, em setembro de 2018, ocorreu a publicação do artigo "O ambiente de programação visual Mosaicode" (10), nos Anais da 9ª Sessão de Ferramentas do CBSoft (Congresso Brasileiro de Software), onde este retrata o que é a ferramenta, seu funcionamento e as extensões que o ambiente possui.

Dando prosseguimento, o artigo "Utilização do ambiente Mosaicode como ferramenta de apoio para o ensino de Computação Musical" (14), foi publicado nos Anais do VIII Workshop on Ubiquitous Music (UBIMUS), também em setembro de 2018. Esta pesquisa discute a aplicação do Mosaicode e da extensão de síntese musical como uma ferramenta didática para auxiliar no ensino de Computação Musical, possibilitando a criação de técnicas e sínteses através da programação visual. Este ponto também se aplica as demais extensões que o ambiente possui atualmente, inclusive ao pacote OpenCV.

Outra publicação realizada durante o desenvolvimento da extensão foi do artigo "Desenvolvimento de extensões de processamento e síntese de imagens para a ferramenta Mosaicode" (39), exposto nos Anais da 31ª Conference on Graphics, Patterns and Images (SIBGRAPI). Este artigo discute a criação de extensões para o ambiente, onde é trabalhada uma metologia de desenvolvimento genérica, abordando diferentes LINDEs possíveis para serem embutidas no ambiente.

Já em relação à aplicação artística da extensão, um resumo foi publicado na 2ª Semana Acadêmica Integrada dos Cursos de Letras, com o título "A interatividade da

audiência no espetáculo O Chaos das 5", que discute o modo de participação do público perante à apresentação. Ainda abordando este cenário, a peça foi apresentada em quatro oportunidades distintas, incluindo performances no Espaço das Artes da Universidade de São Paulo (mostra Sons de Silício), no Circuito Liberdade em Belo Horizonte (SAD 2019), no Centro Cultural da UFSJ (mostra Vestígio) e na Quinta Cultural, realizada no campus Tancredo Neves (UFSJ).

A união dos diferentes conceitos abordados nesta pesquisa tornou o desafio de criar a extensão ainda maior. Como aluno do curso de ciência da computação e como pesquisador, buscar aliar estas áreas de forma que a proposta da ferramenta de alcançar diferentes públicos tenha sido cumprida, beneficia ainda mais os resultados já obtidos com o funcionamento da extensão e de seu uso real.

Abordando especificamente a interação entre computação e seus conceitos contextualizados neste plano, e Arte Digital, não há sequer receio em dizer que a proposta de inserir PDI e VC no meio artístico foi no mínimo curiosa, e completamente inédita para mim (assim como seria para a grande maioria dos alunos de um curso de ciência da computação). Porém, após a concretização da proposta, os resultados foram inovadores e satisfatórios, principalmente, se constatarmos que os protótipos criados para a apresentação O Chaos das 5 foram elaborados apenas com o uso da extensão.

Já abordando a vertente educacional da ferramenta, é válido relatar que todos os blocos construídos foram idealizados da maneira mais simples possível, de forma que não haja um impasse para novos usuários ao utilizar a ferramenta, seja pela complexidade dos códigos gerados ou na construção dos diagramas. Um dos maiores desafios impostos à criação desta extensão, era justamente possibilitar que a ferramenta pudesse auxiliar novos estudantes da área, entretanto, a validação desta aplicação se demandará a trabalhos futuros, e a novos desenvolvedores que venham a aprimorar a extensão. Logo, planeja-se que a extensão OpenCV possa ser aplicada em sala de aula para alunos de disciplinas como PDI, e também de VC, buscando facilitar o compreendimento dos conceitos iniciais abordados.

Para trabalhos futuros, além da possível aplicação em disciplinas de PDI e VC, a extensão pode ser difundida também em grupos de Arte Digital, de forma que os próprios artistas possam avaliar e contribuir com o aprimoramento da ferramenta como um todo.

Além disso, outra pesquisa já idealizada é a união das extensões do Mosaicode. Como já citado, além da extensão OpenCV, o ambiente possui um conjunto de artefatos nos domínios de Computação Musical, Síntese de Imagens, interfaces gráficas, dentre outras que estão em desenvolvimento. Para as extensões que utilizam a mesma linguagem para geração de código, é possível unir os conceitos de forma a ampliar o domínio das aplicações possíveis. Por exemplo, uma das propostas já idealizadas é a conjunção entre esta extensão e o conjunto de blocos OpenGL, que realizam síntese de imagens.

Desta maneira, seria possibilitado, em hipótese, controlar a síntese de estruturas gráficas complexas utilizando a detecção de faces como entrada para tal, como abordado anteriormente na Figura 13. Outra perspectiva possível seria englobar o conjunto OpenCV com a extensão de síntese musical, utilizando parâmetros inferidos pelas detecções para compor e reproduzir variados sons. Ambos os exemplos proporcionariam a entrada da ferramenta no âmbito da Realidade Virtual e Ampliada, já que seria possível, por exemplo, idealizar instrumentos musicais controlados por detecções específicas.

Por fim, outra funcionalidade a ser empenhada perante à extensão OpenCV é a implementação e execução de testes de software, de forma a validar o funcionamento da ferramenta em conjunção ao pacote de blocos criados.

No sentido geral, este projeto contribuiu de forma maiúscula ao desenvolvimento do Mosaicode, possibilitando a abordagem de novas áreas da computação no mesmo. Os conceitos aqui adquiridos foram imprescindíveis para o meu desenvolvimento como um profissional da computação. Além disso, os momentos e vivências inovadores experimentados foram de extrema importância para o meu crescimento, e estão constados como experiências únicas em minha carreira acadêmica.

Referências

1 QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. *RITA*, v. 13, n. 2, p. 11–42, 2006. Citado na página 12.

2 RIOS, L. R. S. Visão computacional. *Departamento de Ci^encia da computa c ao—Universidade Federal da Bahia, Salvador, BA*, 2011. Citado na página 12.

3 FISHER, R. B.; BRECKON, T. P.; DAWSON-HOWE, K.; FITZGIBBON, A.; ROBERTSON, C.; TRUCCO, E.; WILLIAMS, C. K. *Dictionary of computer vision and image processing*. [S.l.]: John Wiley & Sons, 2013. Citado na página 12.

4 PARKER, J. R. *Algorithms for image processing and computer vision*. [S.l.]: John Wiley & Sons, 2010. Citado na página 12.

5 NIXON, M. S.; AGUADO, A. S. *Feature extraction & image processing for computer vision*. [S.l.]: Academic Press, 2012. Citado na página 12.

6 LOW, A. A. *Introductory computer vision and image processing*. [S.l.]: McGraw-Hill, Inc., 1991. Citado na página 12.

7 YAMAMOTO, F. S.; SILVA, A. F. da; ZANUTTO, J.; ZAMPIROLLI, F. A. Interdisciplinaridade no ensino de ciência da computaçao. In: *Anais do XXV Congresso da SBC, Unisinos, São Leopoldo, RS*. [S.l.: s.n.], 2005. Citado na página 12.

8 GIANNETTI, C. Estética digital. *Barcelona: Asociación de cultura contemporánea*, 2002. Citado na página 12.

9 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 na página 13.

10 SCHIAVONI, F. L.; SANDY, J. M. d. S.; CARDOSO, T. T. S.; GOMES, A. L. N.; RESENDE, F. R. O ambiente de programação visual mosaicode. In: *Anais da 9ª Sessão de Ferramentas do CBSoft*. [S.l.]: Sociedade Brasileira de Computação, 2018. v. 1, p. 25–35. Citado 2 vezes nas páginas 13 e 47.

11 HOED, R. M. Análise da evasão em cursos superiores: o caso da evasão em cursos superiores da área de computação. *Brasília, DF: Universidade de Brasília*, 2016. Citado na página 14.

12 PRIETCH, S. S.; PAZETO, T. A. Estudo sobre a evasão em um curso de licenciatura em informática e considerações para melhorias. *WEIBASE, Maceió/AL*, 2010. Citado na página 14.

13 SORVA, J. *Visual program simulation in introductory programming education*. [S.l.]: Aalto University, 2012. Citado na página 14.

14 SCHIAVONI, F. L.; CARDOSO, T. T. S.; GOMES, A. L. N.; RESENDE, F. R.; SANDY, J. M. S. Utilização do Ambiente Mosaicode como ferramenta de apoio para o ensino de Computação Musical. In: *Proceedings of the VIII Workshop on Ubiquitous*

Music (UBIMUS). São João del-Rei - MG - Brazil: [s.n.], 2018. v. 8, p. 25–32. Citado 2 vezes nas páginas 14 e 47.

15 PUCKETTE, M. S. Pure data. In: *ICMC*. [S.l.: s.n.], 1997. Citado na página 15.

16 DANKS, M. Real-time image and video processing in gem. In: *ICMC*. [S.l.: s.n.], 1997. Citado na página 15.

17 CAMURRI, A.; HASHIMOTO, S.; RICCHETTI, M.; RICCI, A.; SUZUKI, K.; TROCCA, R.; VOLPE, G. 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 na página 15.

18 CAMURRI, A.; MAZZARINO, B.; VOLPE, G. Analysis of expressive gesture: The eyesweb expressive gesture processing library. In: SPRINGER. *International Gesture Workshop*. [S.l.], 2003. p. 460–467. Citado na página 15.

19 TRAVIS, J.; KRING, J. *LabVIEW for everyone: graphical programming made easy and fun*. [S.l.]: Prentice-Hall, 2007. Citado na página 15.

20 FREEMAN, J.; MAGERKO, B.; MCKLIN, T.; REILLY, M.; PERMAR, J.; SUMMERS, C.; FRUCHTER, E. Engaging underrepresented groups in high school introductory computing through computational remixing with earsketch. In: ACM. *Proceedings of the 45th ACM technical symposium on Computer science education*. [S.l.], 2014. p. 85–90. Citado na página 16.

21 REAS, C.; FRY, B. Processing: programming for the media arts. *AI & SOCIETY*, Springer, v. 20, n. 4, p. 526–538, 2006. Citado na página 16.

22 RF, S. Estudos de revisão sistemática: um guia para síntese criteriosa da evidência científica. *Revista brasileira de fisioterapia*, SciELO Brasil, v. 11, n. 1, p. 83–89, 2007. Citado na página 17.

23 KITCHENHAM, B.; BRERETON, O. P.; BUDGEN, D.; TURNER, M.; BAILEY, J.; LINKMAN, S. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, Elsevier, v. 51, n. 1, p. 7–15, 2009. Citado na página 17.

24 SANTOS, A. C. C. dos; DELAMARO, M. E.; NUNES, F. L. The relationship between requirements engineering and virtual reality systems: A systematic literature review. In: IEEE. *Virtual and Augmented Reality (SVR), 2013 XV Symposium on*. [S.l.], 2013. p. 53–62. Citado na página 17.

25 MULLAPUDI, R. T.; VASISTA, V.; BONDHUGULA, U. Polymage: Automatic optimization for image processing pipelines. In: ACM. *ACM SIGARCH Computer Architecture News*. [S.l.], 2015. v. 43, n. 1, p. 429–443. Citado na página 21.

26 LOURENS, T. Tivipe-tino's visual programming environment. In: IEEE. *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*. [S.l.], 2004. p. 10–15. Citado na página 21.

27 LAU-KEE, D.; BILLYARD, A.; FAICHNEY, R.; KOZATO, Y.; OTTO, P.; SMITH, M.; WILKINSON, I. Vpl: an active, declarative visual programming system. In: IEEE. *Visual Languages, 1991., Proceedings. 1991 IEEE Workshop on*. [S.l.], 1991. p. 40–46. Citado na página 21.

28 MEUNIER, R. The pipes and filters architecture. In: ACM PRESS/ADDISON-WESLEY PUBLISHING CO. *Pattern languages of program design*. [S.l.], 1995. p. 427–440. Citado na página 25.

29 PULLI, K.; BAKSHEEV, A.; KORNYAKOV, K.; ERUHIMOV, V. Real-time computer vision with opencv. *Communications of the ACM*, ACM, v. 55, n. 6, p. 61–69, 2012. Citado na página 27.

30 ARANHA, E.; BORBA, P. Testes e geração de código de sistemas web. *16th SBES*, p. 114–129, 2002. Citado na página 30.

31 BISHOP, C. M. *Pattern recognition and machine learning*. [S.l.]: springer, 2006. Citado na página 36.

32 WILSON, P. I.; FERNANDEZ, J. Facial feature detection using haar classifiers. *Journal of Computing Sciences in Colleges*, Consortium for Computing Sciences in Colleges, v. 21, n. 4, p. 127–133, 2006. Citado na página 36.

33 LIENHART, R.; MAYDT, J. An extended set of haar-like features for rapid object detection. In: IEEE. *Proceedings. International Conference on Image Processing*. [S.l.], 2002. v. 1, p. I–I. Citado na página 36.

34 LIU, D.; YU, J. Otsu method and k-means. In: IEEE. *2009 Ninth International Conference on Hybrid Intelligent Systems*. [S.l.], 2009. v. 1, p. 344–349. Citado na página 37.

35 CANNY, J. F. *Finding edges and lines in images*. [S.l.], 1983. Citado na página 38.

36 HARRIS, C. G.; STEPHENS, M. A combined corner and edge detector. In: CITESEER. *Alvey vision conference*. [S.l.], 1988. v. 15, n. 50, p. 10–5244. Citado na página 38.

37 SHI, J.; TOMASI, C. *Good features to track*. [S.l.], 1993. Citado na página 38.

38 BUZULOIU, V. V.; CIUC, M.; RANGAYYAN, R. M.; VERTAN, C. Adaptive-neighborhood histogram equalization of color images. *Journal of Electronic Imaging*, International Society for Optics and Photonics, v. 10, n. 2, p. 445–460, 2001. Citado na página 42.

39 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 na página 47.