CENTRO UNIVERSITÁRIO SERRA DOS ORGÃOS - UNIFESO

Matheus da Silva Correa Motizuki

photoGraph: Modelagem de Fluxos de Edição Fotográfica Através de Estruturas de Grafos

Teresópolis

2025

CENTRO UNIVERSITÁRIO SERRA DOS ORGÃOS - UNIFESO

Matheus da Silva Correa Motizuki

photoGraph: Modelagem de Fluxos de Edição Fotográfica Através de Estruturas de Grafos

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Serra dos Órgãos como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Victor de Almeida Thomaz

Centro Universitário Serra dos Órgãos - UNIFESO

Bacharelado em Ciência da Computação

Teresópolis

2025

Matheus da Silva Correa Motizuki

photoGraph: Modelagem de Fluxos de Edição Fotográfica Através de Estruturas de Grafos

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Serra dos Órgãos como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Serra dos Órgãos como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação.

Victor de Almeida Thomaz
Orientador

Professor
Nome Membro 1

Doutor
Nome membro 2

Teresópolis
2025

Dedico este trabalho ao meu falecido pai,
que sempre me apoiou.

Agradecimentos

Agradeço a Deus pela força e discernimento que me permitiram concluir esta etapa acadêmica. Agradeço a minha namorada por me acompanhar nessa jornada, sempre ao meu lado. Agradeço aos meus pais, que me apoiaram durante esta jornada acadêmica. Ao professor Victor de Almeida Thomaz, minha gratidão pela orientação e compartilhamento de conhecimentos essenciais para este trabalho.

“Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

Ferramentas de edição de imagem existem a muito tempo e, desde a sua concepção, esses softwares sempre apresentaram dificuldades de utilização para usuários sem conhecimento técnico. A proposta deste trabalho é desenvolver um aplicativo de edição de imagens simplificado e mais direto, voltado especificamente para o usuário final não técnico. Como diferencial, o projeto explora o uso de tecnologias visuais, empregando programação visual baseada em grafos direcionados para tornar o processo de edição mais intuitivo e acessível.

Palavras chave: Programação visual, Edição de imagens, Grafo direcionado.

Abstract

Image editing tools have existed for a long time and, since their inception, these software solutions have always presented usability challenges for non-technical users. The aim of this work is to develop a simplified and more straightforward image editing application, specifically designed for end users without technical expertise. As a key differentiator, the project explores the use of visual technologies, employing visual programming based on directed graphs to make the editing process more intuitive and accessible.

Key-words: Visual Programming, Image Editor, Directed Graph.

Lista de ilustrações

Lista de abreviaturas e siglas

PFD	Programação de fluxo de dados
DFP	Dataflow programming
CSS	Cascading Style Sheets

Sumário

1 Introdução

Os aplicativos de edição de imagens têm uma longa trajetória. Um dos marcos iniciais que viria a se tornar o Adobe Photoshop, criado pelos irmãos John e Thomas Knoll em 1987. Após um período de testes e pequenas distribuições sob o nome de Barneyscan XP, em 1989, o programa chamou a atenção da Adobe Systems, que fez um contrato de licenciamento com os irmãos Knoll. Assim, em 1990, foi lançado oficialmente o Adobe Photoshop 1.0, inicialmente para o sistema Macintosh (**??**).

Com o avanço das tecnologias de hardware e software, surgiram novos programas de edição de imagem, estabelecendo padrões que hoje são referências na indústria do design gráfico. Ferramentas como o próprio Photoshop, que desde a sua concepção vem se adaptando aos padrões de cada época, e o Gimp, (GNU Image Manipulation Program) tornaram-se amplamente reconhecidos (**??**).

Ao longo do tempo, os softwares de edição de imagem buscaram tornar suas interfaces mais intuitivas e simplificadas, a fim de atingir um público mais amplo e não técnico. Essa tendência de design, proporcionou a criação de diversos aplicativos, dentro e fora da área de edição de imagens, mais simples e com um menor requerimento de expertise em uma área específica.

Um conceito que se destacou durante esse processo foi o desenvolvimento de interfaces visuais, ou programação visual. Isso ocorre porque, de maneira geral, a compreensão de informações apresentadas graficamente tendem a ser mais intuitivas. Essa abordagem reforça a importância de interfaces gráficas bem planejadas, permitindo com que os usuários apliquem seus conhecimentos de forma mais prática, otimizando o fluxo de trabalho e a produtividade.

A Programação de Fluxo de Dados (PFD), ou *Dataflow Programming (DFP)*, é um paradigma de programação que modela a lógica de um programa como um grafo dirigido, de forma semelhante aos Diagramas de Fluxo de Dados (DFD). Nesse modelo, o programa é representado como um conjunto de nós, também conhecido como vértices (ou nós) interligados por conexões direcionadas, chamadas de arcos (ou bordas), que indicam o percurso dos dados entre as operações. Cada nó representa uma função ou operação específica (**????**).

A programação de fluxo de dados permite, por natureza, um alto grau de paralelismo. Diferentemente da arquitetura tradicional de von Neumann, onde as instruções são executadas em ordem e o controle de fluxo é centralizado, na PFD a execução ocorre de maneira assíncrona e descentralizada. Cada *node* (nó) é executado assim que todos os dados necessários para sua operação estejam disponíveis. Esse modelo elimina a necessidade de um controle centralizado de fluxo, permitindo que múltiplas operações sejam realizadas simultaneamente, com melhor aproveitamento dos recucos computacionais disponíveis (**??**).

O alto paralelismo natural da PFD, faz com que, na área de edição de imagem, o fluxo de edição se torne descentralizado e não linear. Em programas tradicionais de edição de imagem, o usuário segue uma ordem linear de operações, fazendo com que cada mudança dependa da anterior. Com a implementação de um modelo Dataflow em um programa de edição de imagem, esse foco muda, e se torna descentralizado, permitindo que o usuário edite uma imagem de diversas maneiras, sem depender de uma sequência rígida.

Programas desenvolvidos sob este paradigma são estruturados por meio de grupos de nodes (ou blocos), interligados por conexões que representam o fluxo de dados. Cada node possui portas de entrada e/ou saída, que pode exercer diferentes funções dentro do sistema, demonstrado na figura 1 De forma geral, esses nós são classificados em três categorias principais, sendo elas: blocos de entrada, blocos de processamento e blocos de saída. As informações percorrem o programa seguindo essas conexões, introduzidos pelos blocos de entrada, sendo transformados conforme passam pelos blocos processadores, até atingirem os destinos finais nos blocos de saída, que coletam os resultados (**??**).

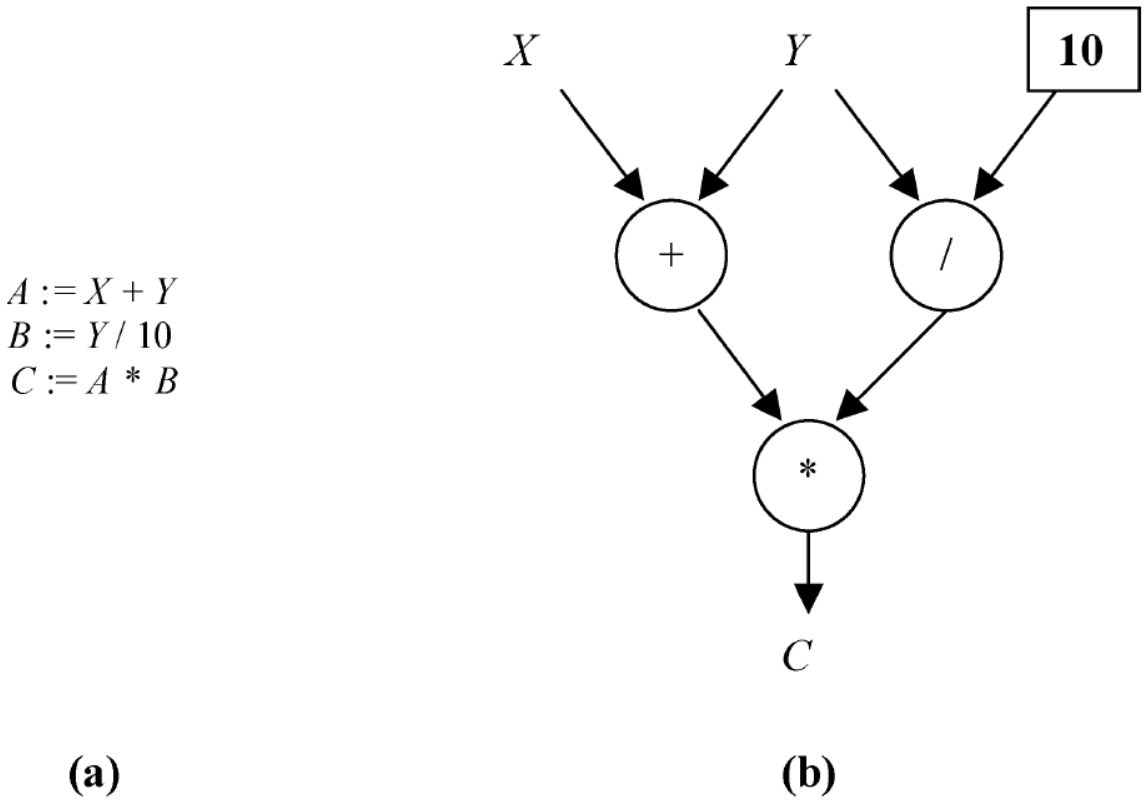

Figura 1 – Um simples programa (a) e o seu equivalente em no formato Dataflow (b)
(**??**)

1.1 Justificativa

Este trabalho propõe a criação de um aplicativo de edição de imagens baseado em grafos para a modelagem do fluxo da edição, tornando possível não apenas compreender e documentar o processo, mas também oferecer ao usuário final, que não possui o conhecimento técnico necessário para operar um aplicativo de edição mais avançado, meios mais intuitivos de modificar e experimentar sequências de transformações. Dessa forma, obtém-se uma solução de edição descentralizada, em que a criatividade é o único limite para a edição. Além disso, a adoção dessa metodologia pode servir como base para o desenvolvimento de aplicações que automatizem ou recomendem etapas de edição.

1.2 Motivação

A criação deste aplicativo origina-se principalmente na dificuldade de utilização de softwares profissionais, que, apesar de sua popularização, ainda apresentam uma barreira de entrada significativa para usuários não técnicos. Esses softwares tendem a não ser amigáveis para novos interessados na área, exigindo muitas vezes um verdadeiro "curso" para que se aprenda a utilizá-los, se tornando mais um requerimento do que apenas um complemento para usuários já familiarizados que querem elevar o nível de conhecimento.

1.3 Objetivos

Na presente seção serão expostos os objetivos do trabalho, tanto gerais como específicos. Objetivo geral consiste em um texto, de natureza extensa, com um detalhamento menos aprofundado, sobre as finalidades a serem atingidas com este projeto.

Objetivo específico é composto por tópicos, de média a curta extensão, de maior detalhamento, especificando finalidades a serem abrangidas pelo projeto.

1.3.1 Objetivos Gerais

O objetivo principal deste projeto é a criação de um aplicativo de edição de foto baseado em nodes, com a sua principal finalidade sendo a implementação do diagrama de fluxo de dados, de uma forma visual, de forma que o usuário possa editar fotos e/ou imagens de uma forma disruptiva e não linear.

Diferindo da tradicional forma de edição fotográfica sendo, uma forma linear de editar as imagens, com instruções uma após a outra, assim limitando o usuário a seguir um padrão ao invés de dar a liberdade para que o usuário possa, da forma que preferir, editar a imagem.

1.3.2 Objetivos Específicos

- Implementar uma UI interativa e de fácil compreensão, que facilite a manipulação e configuração de componentes, sem a necessidade de grande um conhecimento técnico.
- Implementar o paradigma de programação de fluxo de dados, permitindo que usuários construam pipelines de processamento de forma visual e modular.
- Implementar criação de Shaders para edições de imagens e/ou criação de imagens dinâmicas.
- Implementar *nodes* funcionais capazes de realizar mudanças nas imagens, atendendo os parâmetros e requisitos definidos pelo usuário, independentemente da posição ou ordem no grafo.
- Implementar nodes capazes de criar Shaders, tanto em versões de código, feitos pelo usuário, quanto nodes de código já previamente feitos pelo programa.

1.4 Organização dos Capítulos

Na presente seção será apresentada a organização deste trabalho, divido em seções da seguinte forma:

- Introdução
- Fundamentação Teórica
- Trabalhos Relacionados
- Desenvolvimento
- Conclusão

2 Fundamentação Teórica

Na presente seção serão abordados tópicos específicos sobre programação e processamento de imagens que são fundamentais para a compreensão do presente trabalho. Inicialmente, será apresentado, de forma breve, os principais elementos que compõem a base do sistema desenvolvido, incluindo a linguagem escolhida para o desenvolvimento do aplicativo, bibliotecas, aspectos de UI/UX e sua importância para este projeto, bem como o paradigma de desenvolvimento adotado, sendo considerado um dos diferenciais do projeto.

Posteriormente, será abrangido o tema de processamento de imagens, em maior profundidade,e suas aplicações neste trabalho, com ênfase nos conceitos práticos que embasam sua aplicação no projeto. Por fim, será discutido o processo de criação e utilização dos *Shaders*, explicando seu funcionamento e importância.

2.1 Python

Python, uma linguagem de programação alto nível e propósito geral, amplamente reconhecida por sua simplicidade sintática, legibilidade e ampla comunidade de usuários. Sua praticidade e versatilidade tornam-no uma escolha popular em diversas áreas da computação, incluindo, mas não se limitando a, ciência de dados, automação desenvolvimento web e processamento de imagens (**??**).

Nesse contexto, o Python se destaca por oferecer um ecossistema robusto de bibliotecas especializadas, como OpenCV, NumPy, Scikit-image, Pillow, entre outras, que permitem desde simples operações de manipulação de imagem até o desenvolvimento de algorítimos avançados de análise, restauração e aperfeiçoamento visual. Essa ampla gama de recursos torna a linguagem uma ferramenta altamente eficaz para projetos que exigem processamento e tratamento de imagens digitais, como é o caso deste trabalho (**????????**).

2.2 Bibliotecas

Bibliotecas, no sentido tradicional, são locais onde inúmeros livros são armazenados para consulta, permitindo que os leitores adquiram conhecimento em diversas áreas e sobre os mais variados temas. As bibliotecas de programação seguem um conceito semelhante sendo elas, coleções de funções, rotinas ou módulos reutilizáveis, escritas em determinada linguagem de programação (**??**). E assim como livros, aonde releituras ou adaptações são constantemente criadas, as bibliotecas também passam por esse mesmo processo, dependendo da licença de uma biblioteca, ela pode ser reescrita em outra linguagem, ou adaptada como um binding ou

wrapper, ou uma nova versão pode ser refeita, por uma pessoa diferente ou pelo criador original da mesma (**??**).

Nesta seção serão abordadas as bibliotecas utilizadas neste projeto, tanto de processamento de imagens como de interface de usuário.

2.2.1 Dear PyGui

O Dear PyGui (DPG) é uma biblioteca para Python rápida e poderosa para a criação de interfaces gráficas, originalmente desenvolvida em C++ sob o nome de Dear ImGui (*Immediate-Mode Graphical User Interface*). Devido a crescente popularidade da linguagem Python, a biblioteca foi adaptada para esse ambiente, mantendo os princípios de interface em modo imediato e oferecendo uma API simplificada e eficiente (**??**).

O Dear PyGui (DPG) segue o paradigma de GUI em modo imediato ("Immediate mode GUI") que, ao invés de manter uma árvore de componentes entre interações, como ocorre no modo retido (retained mode), o framework desenha a interface a cada frame com base no estado atual da aplicação (**??????**).

No modo retido (*retained mode*), frameworks que utilizam esse estilo armazenam os elementos da interface, como botões, janelas e caixas de texto, em uma árvore ou hierarquia de componentes. A biblioteca gerencia automaticamente os estados dos elementos e atualiza apenas aqueles que sofrem alterações, ou seja, elementos cujo estado, cor, tamanho ou outras propriedades foram modificados pelo usuário ou por outro componente.

Um exemplo de tecnologia amplamente conhecida por utilizar esse método é o CSS (*Cascading Style Sheets*), que mantém o estado de cada elemento em uma estrutura hierárquica e realiza a atualização somente do elemento que foi interagido ou afetado por alterações (**??**).

Entretanto, um dos problemas associados a esse modelo de interface é justamente o gerenciamento de estados. Em aplicações com muitos componentes, esse controle pode se tornar difícil de manter, gerando error que passam despercebidos. Por exemplo, ao deletar um componente e, inadvertidamente, tentar modificar o estado do mesmo, cuja referência ainda permaneça em algum ponto da hierarquia, isso acabaria causando em um erro, pois, o elemento modificado não estará mais presente na hierarquia.

No modo imediato, característico da biblioteca Dearpygui, exige que os elementos sejam desenhados a cada ciclo de atualização, ou a cada frame. Isso oferece alto controle e facilita a criação de interfaces altamente dinâmicas e responsivas, especialmente em tempo real (**??**). Computadores mais fracos, em relação a CPU, acabam sofrendo com aplicações que utilizam esse modo de interface gráfica.

2.2.2 PILLOW

A biblioteca Pillow é uma continuação aprimorada e mantida até os dias atuais, originalmente desenvolvida sob o nome de PIL (Python Imaging Library) feita por Fredrik Lundh (**??**), amplamente utilizada para processamento de imagens em Python. O pillow surgiu como uma fork do PIL, garantindo uma compatibilidade retroativa e adicionando suporte a recursos modernos, além da manutenção continua da comunidade *open source*.

A biblioteca oferece uma ampla variedade de funcionalidades, como a abertura e a possibilidade de salvar imagens em diversos formatos como por exemplo, JPEG, PNG, BMP GIF e TIFF, além de suas outras funcionalidades como redimensionamento de imagens, corte, filtros alteração de cores entre outras (**??**).

2.2.3 PyOpenGL

O PyOpenGL é um conjunto de bindings (ligações) em Python para a biblioteca gráfica OpenGL, permitindo que programas escritos em Python façam o uso direto dessa API de gráficos 2D e 3D. Por meio dela, é possível acessar praticamente todas as funcionalidades da API OpenGL, inclusive extensões modernas como shaders em GLSL. O PyOpenGL é multiplataforma, e é amplamente utilizado para o desenvolvimento de visualizações científicas, simulações e jogos (**??**).

2.3 Shaders

Shaders são pequenos programas que são executados diretamente na GPU (Graphical Process Unit), responsáveis por controlar o processamento de vértices e pixels durante o pipeline gráfico. Escritos geralmente em linguagens como GLSL (OpenGL Shading Language), permitem criar efeitos visuais complexos, como iluminação dinâmica, mapeamento de texturas, sombras e reflexos. Os tipos mais comuns de shaders são dois, *vertex shaders*, que manipulam posições e propriedades dos vértices, e os *fragment shaders* que determina a cor final de cada fragmento (pixel) renderizado (**??**).

2.4 UI/UX

A área de *User Interface/User Experience* representa um aspecto essencial no desenvolvimento de qualquer software sendo, site ou aplicativo. Uma interface mal projetada pode comprometer significativamente a usabilidade da aplicação, deixando-a confusa e pouco intuitiva.

No contexto deste trabalho, a atenção à UI/UX foi uma das prioridades no processo de desenvolvimento. A proposta do aplicativo de ser de fácil utilização e rápido aprendizado,

busca se distanciar de soluções complexas, constantemente acompanhadas com funcionalidades ocultas ou mal estruturadas.

2.5 Dataflow

"Dataflow programming (DFP) é um paradigma de programação, onde a execução do programa é conceitualizada como dados fluindo por uma série de operações ou transformações"(**??**). Diferente dos paradigmas tradicionais imperativos, onde o controle do fluxo é determinando pela sequência explícita de instruções, na PFD o foco está no fluxo dos dados e na maneira como eles propagam pelos nós (ou blocos funcionais) do sistema. Nesse modelo, cada operação ou nó executa seu processamento assim que todos os dados necessários para sua execução estão disponíveis, independentemente de uma ordem de execução global. Isso torna a PFD naturalmente concorrente e paralelizável, uma vez que operações independentes podem ser executadas simultaneamente sempre que seus dados estiverem disponíveis (**??**).

Entre as principais características da PFD, destaca-se a execução orientada a dados. A execução de cada nó é acionada pela chegada de dados completos, dispensando o uso de estruturas tradicionais de controle de fluxo como por exemplo loops ou comandos condicionais. Ademais, como os nós são independentes, podem ser mapeados facilmente para diferentes threads ou até mesmo para processadores distintos, explorando ao máximo arquiteturas multicore e distribuídas (**??**). O modelo PFD contribui para a tolerância a latências, pois em sistemas distribuídos ou heterogêneos diferentes nós podem prosseguir com o processando dados independentemente do tempo de processamento de outros nós. (**??**)

O Dataflow possui múltiplas implementações. Uma das implementações clássicas e mais simples é o modelo baseado em tokens, no qual os dados fluem pelos grafos na forma de tokens que transitam por filas do tipo FIFO (first in, first out) (**????**). Quando os dados necessários estão disponíveis nas entradas de um nó, ele é denominado de um nó *fireable*, ou seja, o node está pronto para ser executado. Um nó *fireable* será executado após um determinado intervalo de tempo desde o recebimento de seus dados. Passando esse período, ele é ativado, seus tokens de saída avançam para o próximo nó, o qual poderá então se tornar *fireable*. O nó anterior, por sua vez, perde a denominação de *fireable*, retorna ao seu estado padrão e aguarda até que o processo possa se repetir.

De forma geral, programas de fluxo de dados são representados por grafos direcionados, nos quais os nós (ou vértices) representam operações ou transformações sobre os dados, enquanto arestas representam canais por onde os dados circulam entre operações. Esse grafo pode ser estático ou dinâmico. Em grafos estáticos, a estrutura do fluxo é conhecida em tempo de compilação, o que facilita otimização e analises. Entretanto, certos programas não podem ser representados por esse tipo de grafo, pois laços (loops), por exemplo, só podem ser modelados caso o número de iterações já seja conhecido em tempo de compilação. Já os grafos dinâmicos,

oferecem uma maior flexibilidade, permitindo que nós com múltiplos arcos sejam diferenciados por cores, em que cada cor é ativada exclusivamente para seu arco específico (**??**).

Em comparação com a tradicional programação imperativa, a PFD não depende de uma sequencia explícita de instruções, mas sim das dependências entre dados. Já em relação à programação funcional, embora ambos incentivem programas sem efeitos colaterais e baseados em composição, a PFD enfatiza o movimento e a dependência de dados entre operações, enquanto a programação funcional foca principalmente em funções puras e transformações sobre coleções (**??**).

A adoção desse paradigma traz diversas vantagens. O paralelismo surge de forma implícita, eliminando a necessidade de escrever explicitamente código de sincronização, pois o próprio modelo organiza as dependências de dados. Além disso, muitas implementação de PFD garantem determinismo, assegurando que, dado o mesmo conjunto de entradas, o programa sempre produzirá o mesmo resultado, independentemente da ordem em que as operações internas são realizadas.

2.6 Teoria dos grafos

A teoria dos grafos é um ramo da matemática discreta que estuda as relações entre objetos representados por meio de estruturas chamadas grafos. Um grafo é definido, formalmente, como um par ordenado $G = (V, A)$, onde V é um conjunto de vértices (ou nós), e A é o conjunto de arestas que conectam pares de vértices. Dependendo do problema ou da aplicação, grafos podem ser direcionados ou não direcionados (**??**).

Nos grafos não direcionados, as conexões entre vértices (ou nós) não possuem direção ou sentido específico; já nos grafos direcionados, cada aresta possui sentido definido, sendo por isso também chamado de arco.

A origem formal do estudo dos grafos é atribuída ao matemático Leonhard Euler, que em 1736 discutiu o famoso enigma das pontes de Königsberg, estabelecendo assim os fundamentos do que hoje é conhecido como teoria dos grafos. Desde então, o campo expandiu-se substancialmente, encontrando aplicações em diversas áreas, como ciência da computação, engenharia elétrica, biologia, logística e redes sociais (**??**).

3 Trabalhos Relacionados

Na presente seção serão apresentados softwares de edição de imagem baseados em nodes (ou grafos). Serão analisados os principais aspectos desses programas, destacando como cada um implementa a modelagem de fluxo de edição, quais recursos são oferecidos aos usuários e a praticidade de uso. O objetivo é identificar pontos fortes e limitações, de modo a fornecer uma base comparativa para a proposta desenvolvida neste trabalho.

3.1 Gimel Studio

O Gimel Studio é um software de código aberto destinado à edição de imagens baseado em nodes. Desenvolvido na linguagem de programação Dart e utilizando o framework Flutter para a construção da interface gráfica, o Gimel Studio apresenta similaridades visuais com editores tradicionais, como o Adobe Photoshop.

Atualmente o Gimel Studio se encontra em estágio inicial de desenvolvimento. O repositório oficial fornece instruções detalhadas para a execução do projeto localmente. Até o presente momento, existe somente um lançamento de uma versão alpha, feita em Janeiro de 2022. Desde então, o desenvolvimento não apresenta atualizações significativas.

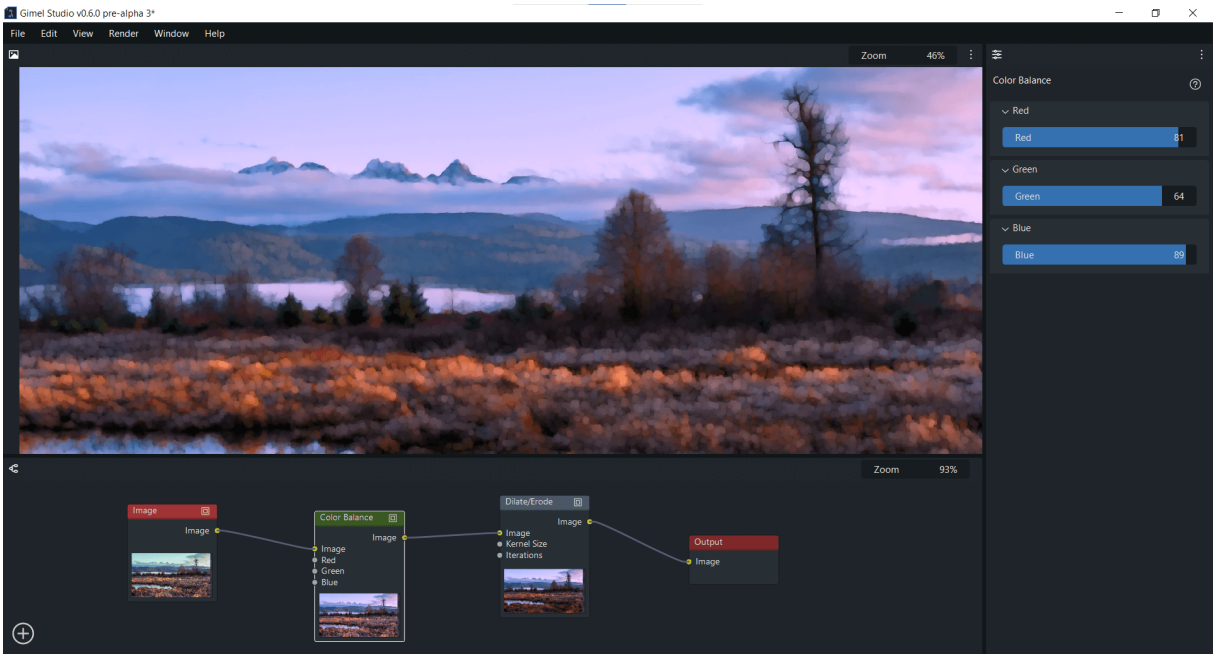

Figura 2 – Imagem do Gimel Studio
(??)

3.2 Cresliant

O Cresliant é software de código aberto destinado à edição de imagens baseado em nodes. Desenvolvido na linguagem de programação Python, o Cresliant destaca-se por oferecer uma interface gráfia intuitiva e acessível ao usuário final, frequentemente referida como *User-Friendly*. Entre tecnologias utilizadas em sua criação, destaca-se o uso do *Poetry*, para o gerenciamento eficiente de dependências e ambientes virtuais, bem como a biblioteca PILLOW (fork da Python Imaging Library - PIL), que faz o papel crucial no processamento e manipulação de imagens do software.

Apesar de o repositório do Cresliant, no Github, não apresentar atualizações recentes, sendo o último *commit* registrado há aproximadamente dois anos, e a versão mais recente datada de dezembro de 2023, o software permanece funcional e estável, cumprindo de maneira eficaz o propósito a que se destina.

O Cresliant oferece, por padrão, poucos nodes/módulos de edição. No entanto, possibilita que o usuário crie novos nodes/módulos e os adicione como submódulos/*addons* externos, sem a necessidade de modificações diretas no código-fonte. Essa abordagem promove uma certa liberdade ao usuário final, mas ao mesmo tempo limita a funcionalidade apenas àqueles que possuem conhecimento técnico suficiente para realizar tal integração de novas funcionalidades.

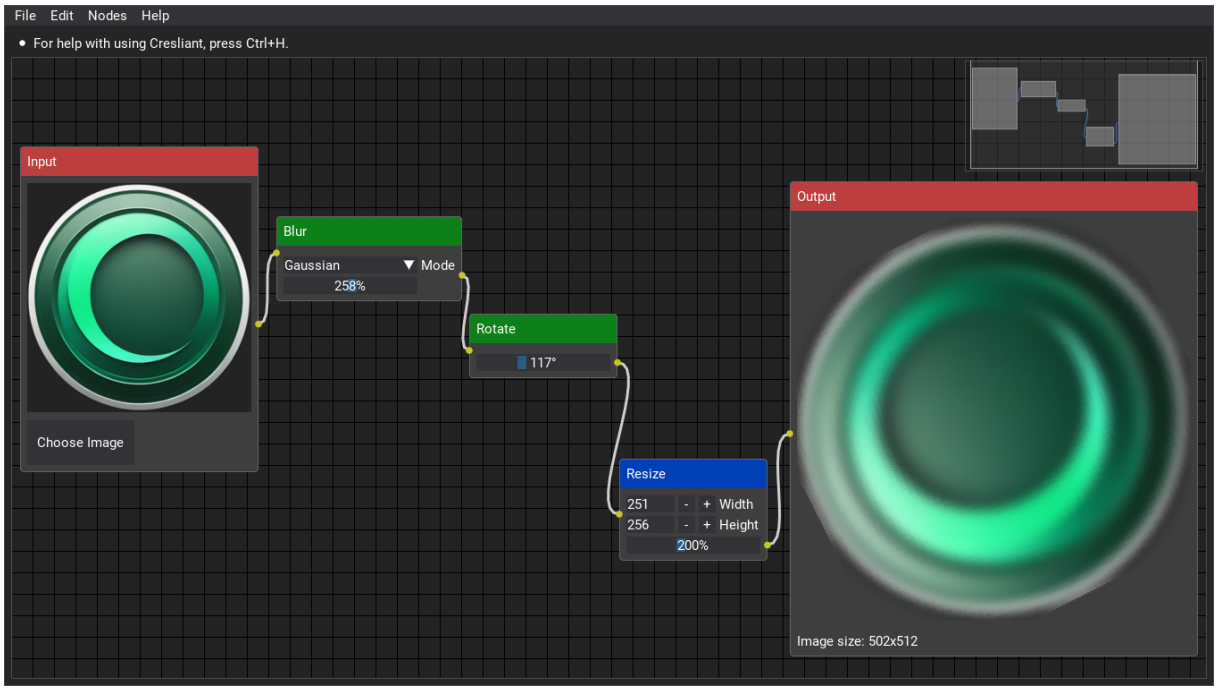

Figura 3 – Imagem do Cresliant
(**??**)

3.3 PixiEditor

O PixiEditor é software de código aberto destinado à criação de gráficos procedurais por meio de nodes. Desenvolvido na linguagem de programação Csharp, o PixiEditor destaca-se por sua robustez e ampla gama de funcionalidades. Além do editor de gráficos procedurais, o software oferece recursos adicionais, como criação de animações 2D, edição tradicional de imagens 2D, entre outras ferramentas, sendo ainda compatível com múltiplas plataformas.

Atualmente o PixiEstudio encontra-se com o desenvolvimento ativo, com sua ultima versão lançada no dia 10 de setembro de 2025.

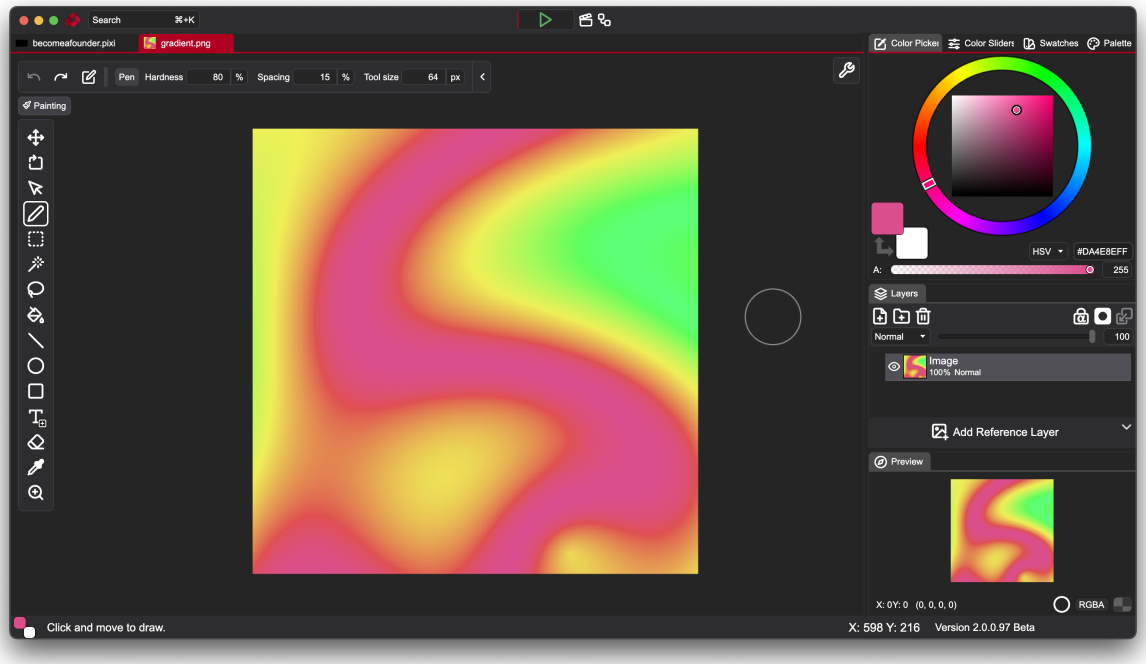

Figura 4 – Imagem do Pixi Editor
(**??**)

4 Metodologia

5 Conclusão