PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA PROPOSTA NÃO LINEAR PARA O FLUXO DE EDIÇÃO DE IMAGENS

ALEXANDRE KAZUO SEKI

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. João Batista Souza de Oliveira

Porto Alegre
2012

S463p Seki, Alexandre Kazuo
Uma proposta não linear para fluxo de edição de imagens / Alexandre Kazuo Seki. – Porto Alegre, 2012.
72 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. João Batista Souza de Oliveira.

1. Informática. 2. Processamento de Imagens – Técnicas Digitais. I. Oliveira, João Batista Souza de. II. Título.

CDD 006.61

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***Uma Proposta de Representação Não-Linear para o Fluxo de Edição de Imagens***", apresentada por Alexandre Kazuo Seki como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Interativos e Visualização, aprovada em 15/03/2012 pela Comissão Examinadora:

Prof. Dr. João Batista Souza de Oliveira - Orientador — PPGCC/PUCRS

Profa. Dra. Milene Selbach Silveira - — PPGCC/PUCRS

Profa. Dra. Carla Maria Dal Sasso Freitas - — UFRGS

Homologada em 10/07/2012, conforme Ata No. 014/2012 pela Comissão Coordenadora.

Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

"Creativity is allowing yourself to make mistakes.
Art is knowing which ones to keep."
(Scott Adams)

AGRADECIMENTOS

À Deus em primeira instância pelo dom da vida.

À minha família por todo o apoio.

Ao meu orientador, por todos os ensinamentos tanto técnicos quanto pessoais que certamente levarei para o resto da vida.

Aos meus amigos que sempre me apoiaram e acreditaram no meu potencial.

Aos demais colegas, professores e funcionários da PUCRS que de alguma forma auxiliaram no processo.

UMA PROPOSTA NÃO LINEAR PARA O FLUXO DE EDIÇÃO DE IMAGENS

RESUMO

Este trabalho disserta sobre modelos utilizados para representar e armazenar o fluxo de operações para edição de imagens. As ferramentas de edição de imagens, geralmente, utilizam uma pilha para armazenar este fluxo, e neste trabalho são apresentados alguns problemas encontrados neste modelo, como por exemplo: quando o usuário desfaz algumas operações e aplica uma nova, as operações desfeitas são perdidas. Outro problema detectado é que o histórico é mantido por sessão, ou seja, ao salvar uma imagem e abrir uma edição dela em outro momento o histórico estará vazio, perdendo-se as transformações que já foram executadas e não é mais possível retroceder. Este trabalho apresenta uma forma não-linear para armazenar e visualizar estes fluxos, baseando-se em uma árvore. Com uma árvore é possível ter vários caminhos que são diversas edições em uma única forma de visualizar. Salvando esta árvore pode-se manter o histórico para uma futura edição. Outras vantagens e detalhes do modelo proposto são descritos ao longo do trabalho. Por fim é introduzido o protótipo desenvolvido para avaliar o modelo, em seguida são apresentados resultados de avaliações com usuários utilizando o protótipo.

Palavras Chave: Visualização, desfazer e refazer, programação visual.

A NONLINEAR APPROACH FOR THE REPRESENTATION OF AN IMAGE EDITTING FLOW

ABSTRACT

This work explores the current model used to store the flow of operations done in the image editing process. Tools for editing normally use a stack to store this flow, and this work presents problems of this model: when the user undoes some operations and applies a new one the undone operations are lost. Other problem is that the history is kept on a session basis, and if the user saves an image after editing and opens it at another time the history will be empty, thus losing the changes that have already been executed and is no longer possible to undo. This work presents a nonlinear way for storing and viewing these flows, based on trees. With one tree users can have multiple paths in a unique way to visualize. Saving this tree keeps the history for future editing. Other advantages and details the proposed model is described throughout the paper. Finally we introduce the prototype developed to evaluate the model, after we present the results of evaluations with users using the prototype.

Keywords: Vizualization, undo and redo algorithms, visual programming.

LISTA DE FIGURAS

LISTA DE SIGLAS

CAD – Computer-Aided Design

JPEG – Joint Photographic Experts Group

PNG – Portable Network Graphics

SVG – Scalable Vector Graphics

XML – eXtensible Markup Language

PHP – PHP: Hypertext Preprocessor

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

MB – Megabyte

KB – Kilobyte

SUMÁRIO

1. INTRODUÇÃO

A edição de imagens consiste em aplicar operações que alteram uma imagem original com o intuito de aprimorá-la para uma situação desejada. Estas transformações são executadas de forma sequencial, gerando um fluxo linear. As principais ferramentas de edição de imagens possuem uma opção para visualizar este fluxo durante o seu processo de edição.

Normalmente estas representações são chamadas de "Históricos", pois possibilitam a visualização das operações executadas durante a edição. Estes históricos permitem retroceder a um ponto da edição, para desfazer e refazer operações. Esta funcionalidade permite ao usuário explorar a ferramenta e aprender com seus erros sem a necessidade de perder resultados já obtidos, no entanto, estes históricos possuem algumas limitações que são discutidas neste capítulo. Em seguida uma breve introdução da proposta deste trabalho para aprimorar este histórico. Por fim, há uma seção indicando a organização do restante do texto.

1.1 Históricos

Atualmente existem diversos programas para a edição de imagens e dentre os mais conhecidos estão *Adobe Photoshop* [17], *GIMP* [7], *PaintShop Photo X3* [5] e *Picasa* [8]. Estes programas são capazes de editar imagens oferecendo ferramentas para cortar, colar, pintar, etc. e também podem transformar a imagem [1] utilizando algoritmos como a conversão para preto e branco ou tons de cinza. Os programas aplicam estas transformações sobre a imagem, gerando outra imagem resultante que substituirá a imagem atual na janela principal. Por exemplo, um programa abre a Figura 1.1a em uma janela, que quando convertida para escala de cinza será comutada pelo resultado apresentado pela Figura 1.1b e, por fim, quando aplicado um filtro de detecção de bordas, a Figura 1.1b é substituída pela Figura 1.1c na janela principal. Isto gera um fluxo linear de edição que possui algumas deficiências, como, por exemplo, o fato de as imagens intermediárias ficarem escondidas e somente o resultado final ser apresentado. Ou seja, para comparar o resultado atual e o intermediário é necessário salvar uma imagem e abri-la posteriormente ou então abrir duas janelas de edição.

Um dos principais problemas do fluxo linear de edição é a impossibilidade, de a partir de uma imagem, gerar duas ou mais imagens resultantes como mostra a Figura 1.2, onde a partir da Figura 1.2a espera-se gerar as Figuras 1.2b e 1.2c. Um fluxo linear de transformações obriga o programa a ter dois fluxos diferentes, um para a Figura 1.2b e outro para a Figura 1.2c.

Os programas citados armazenam as transformações aplicadas em um histórico de transformações (uma pilha), e cada nova transformação é inserida no topo desta pilha. Este histórico também é conhecido como histórico de desfazer (*undo history*), visto que apresenta as transformações executadas sobre a imagem auxiliando o usuário a visualizá-las e permitindo desfazê-las.

[1] Neste trabalho será usado o termo "transformações" para descrever estas operações executadas sobre a imagem.

Figura 1.1 – Exemplo de fluxo linear de transformações.

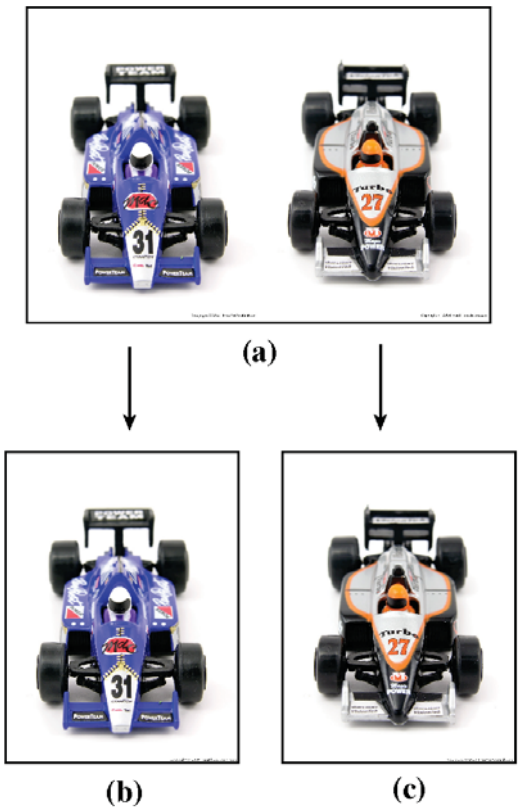


Figura 1.2 – Exemplo de geração de múltiplas imagens a partir de uma entrada.

Um inconveniente desta técnica é que quando o usuário necessitar desfazer uma transformação executada a uma distância *n* do topo da pilha, este deverá desempilhar *n* transformações para poder desfazer a que deseja. E quando aplicar uma nova transformação, as transformações que foram desempilhadas serão perdidas, logo o usuário deverá aplicá-las novamente de forma manual.

Outro inconveniente é que geralmente os programas mantêm o histórico apenas durante uma sessão de trabalho e quando esta é encerrada o histórico é descartado, ou seja, o resultado final das transformações pode ser salvo, todavia as entradas da pilha serão perdidas porque não existe persistência dos passos executados durante uma edição. Para salvar as imagens intermediárias

o usuário deve salvar o resultado de cada transformação, gerando trabalho extra e aumentando significativamente o número de arquivos que devem ser salvos.

Por exemplo, na Figura 1.3 é apresentado um fluxo linear de transformações onde a Figura 1.3a é a original, a Figura 1.3b é o resultado de um recorte, a Figura 1.3c é a consequência de uma rotação e as Figuras 1.3d-g são os produtos gerados com a troca da cor de fundo. Em um dos programas descritos anteriormente seria possível salvar apenas a Figura 1.3g em um arquivo, pois as demais seriam perdidas quando o usuário encerrarsse a sessão. Para armazenar os resultados gerados nas Figuras 1.3d, 1.3e e 1.3f o usuário deveria salvar 3 arquivos diferentes.

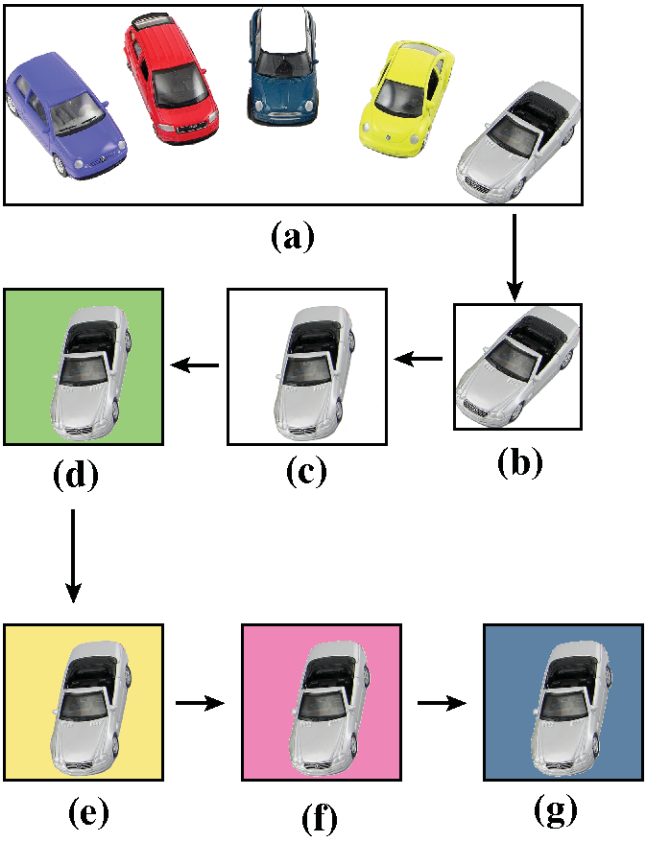


Figura 1.3 – Exemplo de fluxo linear de transformações.

Outro motivo para salvar o histórico é que assim seria possível replicar as transformações executadas durante uma edição sobre outra entrada, habilitando assim a automatização de uma edição em um conjunto de imagens, por exemplo, converter todas as imagens de uma pasta para tons de cinza.

1.2 Proposta

O objetivo do presente trabalho é investigar técnicas para aprimorar estes históricos, visando solucionar os problemas citados anteriormente. Neste trabalho será proposto um modelo de representação não-linear que faz uso de árvores para representar o histórico. Com uma árvore

é viável criar vários ramos a partir de um único nodo, gerando assim fluxos não-lineares. A possibilidade de gerar estes fluxos permite a implementação de funcionalidades como a cópia de um ramo de uma árvore para ser inserido abaixo de outro nodo, repetindo uma sequência de transformações. Isto auxilia em ocasiões nas quais o usuário necessita replicar as mesmas transformações sobre outro nodo da árvore.

O modelo é dividido em uma interface gráfica para o usuário interagir com a ferramenta, uma visualização da árvore e uma *engine* que processa as transformações executadas pelo usuário. Para a avaliação deste modelo foram implementadas duas versões de protótipo, uma com o histórico linear e outra com a implementação da árvore. Estes protótipos foram testados com usuários e os resultados são descritos no Capítulo 5.

1.3 Organização do texto

No Capítulo 2 serão abordados trabalhos utilizados como base para o desenvolvimento de um modelo com o intuito de solucionar os problemas descritos. Em seguida, no Capítulo 3, é feita a descrição deste modelo, contemplando as suas funcionalidades. Após, no Capítulo 4, é detalhada a implementação do protótipo desenvolvido para a avaliação do modelo proposto. No Capítulo 5 são descritos os testes executados para a avaliação do protótipo e a avaliação dos usuários sobre o novo modelo. Por fim, são apresentadas as conclusões e trabalhos futuros no Capítulo 6.

2. TRABALHOS RELACIONADOS

Primeiramente são descritos trabalhos que estudam mecanismos para desfazer e refazer ações ou comandos como os trabalhos de *Vitter* [26] e *Berlage* [2], utilizados para auxiliar o usuário a explorar as ferramentas sem a preocupação de não conseguir retornar a um determinado resultado. Em seguida é mostrado como funcionam as duas principais ferramentas de edição de imagens: *GIMP* [7] e *Adobe Photoshop* [17], descrevendo as técnicas utilizadas para visualizar e interagir com o fluxo de edição de imagens. Após são apresentados trabalhos da área de programação visual, que visam buscar formas de programar utilizando elementos gráficos, com o intuito de facilitar o trabalho do usuário.

2.1 Histórico de desfazer

O histórico de desfazer não é uma funcionalidade utilizada apenas em editores de imagens, programas como editores gráficos [2, 14, 28], navegadores *Web* [12], programas para desenho industrial (CAD) [25], ferramentas para visualização de informação [9, **?**], editores de texto e outros também utilizam estes mecanismos. Os primeiros trabalhos encontrados na literatura sobre mecanismos para desfazer e refazer são da década de 80. A seguir serão apresentados avanços nestas técnicas e serão discutidos problemas e soluções para os mesmos.

Um dos primeiros trabalhos sobre o histórico de desfazer é o artigo de *Vitter* [26] que apresenta um framework para desfazer e refazer comandos executados no ambiente de programação *COPE* [1]. No artigo são introduzidos os problemas detectados no modelo que utiliza um fluxo linear para armazenar o histórico. Por exemplo, quando o usuário retrocede até um determinado comando e insere comandos novos, os comandos executados posteriormente são perdidos e sobrepostos pelos novos. Para solucionar este problema foi proposta uma árvore para armazenar os comandos, permitindo ao usuário retroceder e gerar outros caminhos ao invés de sobrescrever o caminho atual. O artigo identificou que pode existir uma ambiguidade quando o usuário quiser refazer um comando e existirem dois caminhos. Por exemplo, na Figura 2.1 o usuário executou os comandos A_1 até A_n então voltou até A_1, executou B_1 e B_2 e retrocedeu duas vezes, logo o usuário estará no comando A_1. Se solicitar para refazer um comando, será um comando ambíguo pois pode refazer tanto B_1 ou A_2, para solucionar este problema o programa irá perguntar se quer voltar para B_1 ou A_2.

Além dos comandos de desfazer e refazer é introduzido o comando de pular (*skip*) que não executa um comando marcado pelo usuário. Isso auxilia na experimentação sem a necessidade de perder um ou mais comandos.

Zhou [28] afirma que as funcionalidades de desfazer e refazer são importantes para aprimorar a usabilidade e aumentar a capacidade de aprendizado do usuário, pois aumenta o controle e a liberdade do usuário. Em seu trabalho é introduzido um modelo de histórico para uma ferramenta de edição gráfica de forma não-linear, que não tem uma pilha para armazenar as edições e, sim,

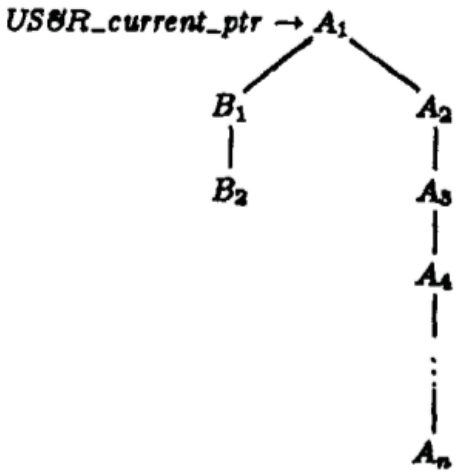


Figura 2.1 – Árvore criada por *Vitter* [26] para armazenar comandos.

um histórico para cada objeto, que armazena todas as edições aplicadas sobre ele. Por exemplo, o objeto A possui um histórico contendo:

- *Move(A, (x,y))* = (move o objeto A para a coordenada definida por *(x,y)*);
- *Color(A, pink)* = (pinta o objeto A de rosa);
- *Color(A, yellow)* = (pinta o objeto A de amarelo);

O modelo também propõe o uso de um histórico global, que proporciona desfazer operações realizadas sobre mais de um elemento. Neste caso, o sistema busca o histórico de todos os objetos envolvidos nesta edição e retrocede em cada um deles, como por exemplo:

- *Select(A, B)* = (seleciona os objetos A e B));
- *Color((A, B), red)* = (pinta os objetos A e B de vermelho);

Caso retrocedida a operação que pinta os objetos de vermelho no histórico global, o sistema irá procurar em todos os históricos dos objetos envolvidos e irá desfazer a operação, no caso nos históricos de A e B.

O trabalho de *Kaasten* [12] disserta sobre navegadores *Web*, discutindo funcionalidades como voltar/avançar, o histórico de páginas e os *bookmarks*. Os autores afirmam que existem problemas no modelo atual: por exemplo a funcionalidade de voltar/avançar é baseada em uma pilha, ou seja, é um fluxo linear que pode confundir o usuário, por exemplo, o usuário está na *página A* e clica em um *link B*, em seguida aciona a funcionalidade de voltar e clica no *link C*, então o usuário clica no botão voltar buscando a página do *link B*, porém este já foi removido da pilha quando clicou no *link C* confundindo o usuário. Outro problema detectado é que esta operação é mantida durante a sessão, ou seja, quando a janela ou aba for fechada as entradas da função voltar e avançar são descartadas. Sobre o histórico dos navegadores foi identificado que a representação visual não é

muito eficiente, pois normalmente os históricos apresentam o *link* ou o título da página, o que não facilita para o usuário identificar a página que quer ver. Em seu trabalho os autores propuseram um modelo que guarda uma imagem da página e apresenta em tamanho reduzido juntamente com o título da página.

Berlage [2] introduz a ideia de desfazer de forma seletiva, ou seja, o sistema apresenta uma lista de comandos que podem ser desfeitos e o usuário seleciona o desejado. Isto foi apresentado em uma época em que os históricos eram lineares e baseados em pilha e quando o usuário quisesse retroceder até um determinado estado, deveria desempilhar até o desejado. Desta forma o usuário apenas escolhe qual comando deseja retroceder. Um exemplo da janela criada para selecionar o comando para ser desfeito é apresentado na Figura 2.2.

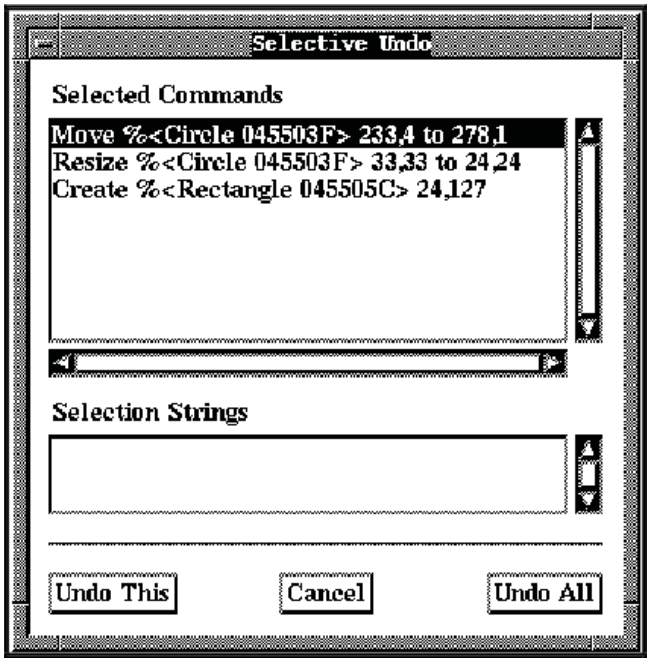


Figura 2.2 – Janela proposta por *Berlage* [2] para escolher um comando para ser desfeito.

No artigo de *Meng* [15] é introduzida uma forma de visualizar o histórico para desfazer e refazer de forma seletiva em um editor gráfico. Na época do artigo as funcionalidades de desfazer e refazer eram textuais, ou seja, o histórico mostrava as entradas com o nome da operação que a gerou. Os autores propuseram que para cada edição fosse armazenada uma foto da edição, e em seu histórico eram apresentadas as funções e as fotos. O estudo descreveu formas de ver estas entradas do histórico de forma a preservar a ordem cronológica destas entradas, alterando o tamanho da entrada: quanto menor significa que é mais antiga e quanto maior, mais recente. Um exemplo da visualização proposta pelo trabalho é mostrado na Figura 2.3

Meng afirma que mostrar uma foto do resultado nas entradas do histórico facilita para usuário identificar o estado ao qual retroceder. Cada uma destas entradas deve ser selecionável de forma direta, ao invés de retroceder uma a uma até a entrada desejada.

No trabalho de *Scheidegger* [**?**] são apresentadas algumas funcionalidades do *VisTrails* que é um sistema para visualizações científicas. Este sistema facilita a alteração dos valores de entradas,

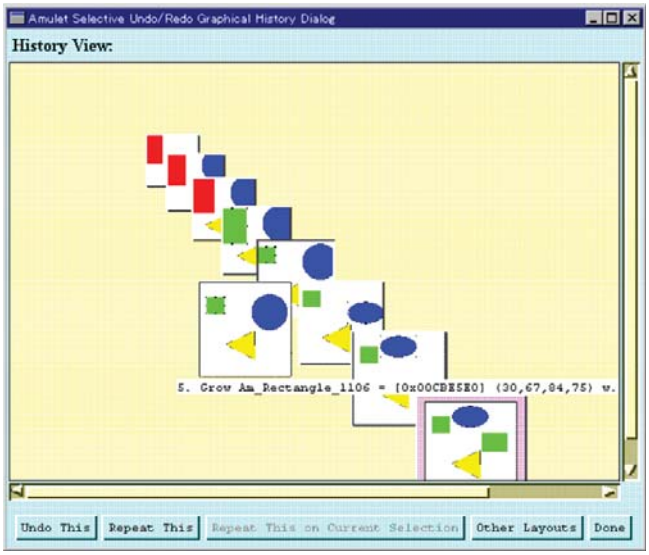


Figura 2.3 – Exemplo da visualização do histórico proposto por *Meng et al* [15]

nas funções e nos parâmetros utilizados para gerar uma nova visualização. O usuário pode fazer estas alterações de forma interativa e comparar os resultados de forma simples.

Cada *workflow* é um conjunto de passos realizados para criar uma visualização. O *VisTrails* pode interligar *workflows* para gerar resultados mais refinados. Neste caso os *workflows* são representados em forma de árvore que permite forma intuitiva retroceder a uma visualização, comparar os resultados de diferentes *workflows* e facilita na compreenção dos passos já realizados.

Estes trabalhos serviram de base para o desenvolvimento da abordagem proposta, fornecendo ideias para a representação não-linear do fluxo de edição de imagens. Como por exemplo a forma de retroceder de forma direta apresentada por *Meng* e a possibilidade de manter os estados que já foram desfeitos como no trabalho de *Vitter*.

2.2 Programas atuais

Nesta seção serão abordados os programas mais populares de edição de imagens, descrevendo seus mecanismos para visualizar o fluxo de edição de imagens e as técnicas utilizadas para desfazer e refazer. O primeiro programa de edição de imagens a ser apresentado é o *GIMP* [7] seguido do *Adobe Photoshop* [17].

2.2.1 GIMP

O programa *GIMP* [7] (*GNU Image Manipulation Program*) é um dos principais editores de imagens utilizadas pelas comunidades de software livre. Este programa possui uma interface dividida em janela principal com um menu no topo juntamente com a imagem a ser editada e janelas configuráveis. A janela mais utilizada é a que contém as principais ferramentas usadas para

a edição de imagens, no entanto existem outras, como por exemplo a janela que visualiza as camadas de uma imagem, a janela de cores e a janela do *Histórico de Desfazer*.

Esse *Histórico de Desfazer* apresenta entradas constituídas de uma imagem pequena do resultado e o nome da transformação que o gerou. Estas entradas são empilhadas formando um fluxo linear, ou seja, a cada nova transformação é adicionada uma nova entrada na pilha. A Figura 2.4 mostra o *Histórico de Desfazer* do *GIMP*, onde foram aplicadas transformações de troca de fundo utilizando a ferramenta *Preenchimento*. Na figura, os resultados das transformações podem ser vistos no histórico pois as transformações resultaram em diferenças significativas entre as imagens. No entanto, na Figura 2.5 é apresentado um exemplo que dificulta a visualização destas entradas no histórico, onde a Figura 2.5a mostra o histórico, a Figura 2.5b a imagem original seguida da Figura 2.5c que mostra o resultado após algumas transformações. Nesta figura (Figura 2.5) percebe-se que é difícil visualizar as diferenças apresentadas no histórico, pois os resultados são imperceptíveis quando apresentados em imagens pequenas.

Figura 2.4 – Histórico representando a troca de fundo de uma imagem do programa *GIMP*.

Para visualizar o resultado de uma transformação existem duas opções: a primeira é desfazer até a entrada desejada utilizando a funcionalidade de desfazer pelo menu ou pelo atalho **CTRL+Z**; e a segunda é clicar com o mouse sobre a entrada desejada na pilha. Contudo não é possível visualizar o resultado de duas transformações ao mesmo tempo, ou seja, se uma imagem é apresentada na janela principal, quando o usuário retroceder ou clicar em uma entrada do histórico o resultado desta será apresentado na janela principal. E quando o usuário aplicar uma nova transformação sobre este estado, todas as aplicadas posteriormente serão sobrescritas pela nova e serão perdidas.

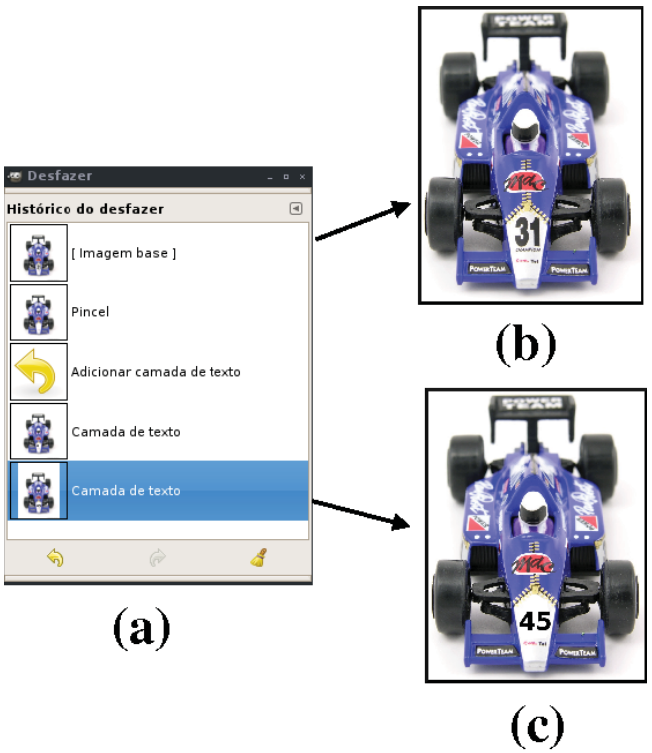


Figura 2.5 – Histórico de desfazer do *GIMP* apresentando a imagem original e a resultante.

Durante a sessão, quando a imagem é fechada as entradas nesta pilha são descartadas, ou seja, se a imagem for reaberta o histórico estará vazio. Um dos inconvenientes desta técnica é a perda do histórico pois se o usuário quiser continuar uma edição em outro momento e tentar retroceder alguma transformação ele será impossibilitado, obrigando-o a aplicar novamente as transformações executadas anteriormente. Acredita-se que o programa não armazena este histórico pois necessitaria salvar a imagem original e todas as transformações aplicadas sobre a imagem, por consequência, os arquivos seriam cada vez maiores, dependendo do número de transformações executadas. E seria necessária uma operação explícita "Limpar Histórico" para diminuir o tamanho dos arquivos.

2.3 Adobe Photoshop

O *Adobe Photoshop* [17] é um dos programas mais populares para a edição de imagens e é um produto comercial oferecido pela empresa *Adobe*. Possui uma janela principal, com o menu principal no topo, uma janela móvel contendo as ferramentas e outras janelas que podem ser configuradas. Entre elas existe a janela *History*, que mostra o histórico de transformações aplicadas sobre a imagem sendo editada.

Este histórico é linear e usa uma pilha de transformações como o *GIMP* [7] e cada entrada desta pilha é representada pelo ícone da ferramenta e ao lado o nome desta. Na Figura 2.6 pode se observar um exemplo, no qual a imagem sofre retoques com as ferramentas *Brush Tool* (Pincel) e *Pencil* (Caneta). O uso do ícone da operação gera mais problemas que o uso de imagens resultantes em tamanho reduzido, pois não se pode ter noção do resultado obtido pela transformação.

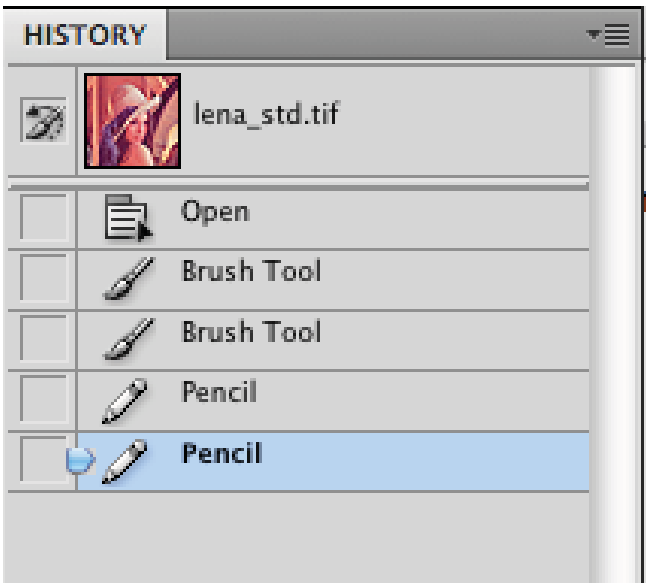


Figura 2.6 – Histórico de transformações do programa *Adobe Photoshop*

Para visualizar o resultado de uma transformação, o usuário deve clicar na transformação desejada apresentada neste histórico e o resultado da transformação é mostrado na janela principal da ferramenta. Porém este histórico tem um tamanho máximo de entradas que irá depender de quanta memória o usuário disponibilizar para o programa, e por *default* tem-se 20 entradas no histórico. Isto pode causar problemas caso um usuário queira desfazer mais transformações que o programa permita.

Uma vantagem deste programa é a funcionalidade *Snapshot* que consiste em "tirar uma foto" do estado atual, ou seja, seleciona-se um resultado e armazena-se este resultado. O usuário pode fazer transformações posteriores ou retroceder, mas o resultado desta transformação estará salvo durante a sessão. Por exemplo na Figura 2.7 são apresentadas transformações de troca de fundo e para armazenar temporariamente estes resultados é utilizada a função *Snapshot*. Os resultados são representados por um ícone com o resultado e ao lado o nome da *Snapshot*, no exemplo são *Snapshot 1*, *Snapshot 2*, *Snapshot 3* e *Snapshot 4*.

Outra funcionalidade adicional do *Photoshop* é a possibilidade de clonagem, criando um novo documento a partir de uma entrada do histórico, ou seja, é criado um novo documento abrindo uma nova janela ou aba com a imagem resultante do estado selecionado e o histórico mostra a operação de duplicação e a edição original que foi clonada continua aberta. Um exemplo desta funcionalidade pode ser visto na Figura 2.8 onde a primeira janela mostra a imagem original (Figura 2.8*a*), em seguida o resultado de algumas transformações (Figura 2.8*b*) e quando iniciar uma nova edição a partir deste resultado é aberta uma nova aba como mostra a Figura 2.8*c* e pode-se visualizar o histórico com a operação de duplicação.

O histórico do *Photoshop* também armazena as entradas apenas durante uma sessão, ou seja, todas as entradas são descartadas quando a sessão é encerrada; o mecanismo é muito semelhante ao visto na Seção 2.2.1 e, como no histórico, as entradas criadas pela função *Snapshot* também são descartadas.

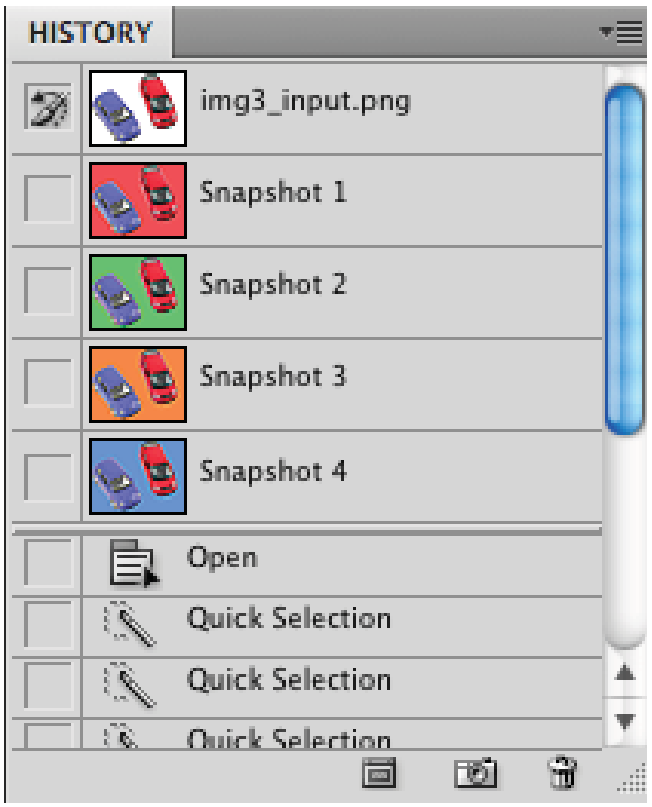


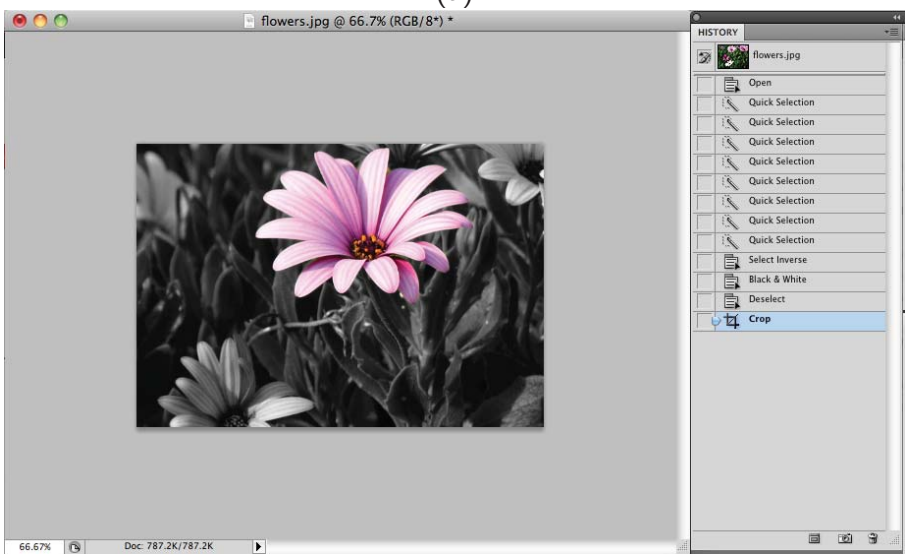
Figura 2.7 – Função *Snapshot* do programa *Adobe Photoshop* [17]

Porém o *Adobe Photoshop* [17] contém uma alternativa para o fluxo linear de transformações, que por *default* é desligada, mas que possibilita ao usuário usar uma pilha de transformações sem uma ordem linear. Ou seja, quando o usuário retroceder até um ponto e executar uma nova transformação e aquelas que foram executadas posteriormente não serão perdidas, porém a nova entrada do histórico será empilhada ao final da última transformação da pilha. Este mecanismo auxilia na persistência do histórico, porém é uma pilha sem uma ordem correta.

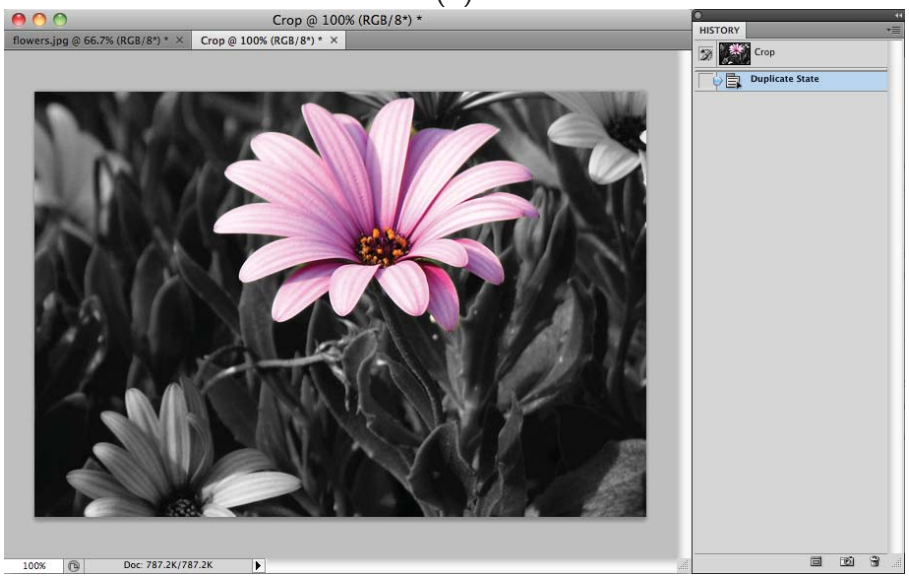
Por exemplo, na Figura 2.9 [1] o usuário executou as transformações A, B, C, D em sequência, mas decidiu retornar ao resultado gerado pela transformação B. Para retornar ao resultado desta transformação o usuário pode selecionar a entrada no Histórico com um clique do mouse sobre esta transformação. Feito isso o usuário decidiu executar as transformações E e F, porém decidiu retornar ao resultado gerado por D e executar uma nova transformação G. O resultado gerado por G ignora as transformações E e F, no entanto, quando o usuário clicar sobre a entrada no Histórico sobre uma delas o resultado gerado ainda será apresentado. Uma das vantagens desta técnica é a possibilidade de criar dois resultados intermediários a partir de uma entrada sem perder as transformações aplicadas posteriormente, ou seja, utilizando uma técnica linear as transformações C e D da Figura 2.9 seriam perdidas quando o usuário executasse a transformação E, porém com esta técnica as transformações não são perdidas. Contudo quando o projeto for salvo o histórico é descartado e todas as entradas são perdidas.

[1] O grafo é meramente ilustrativo, o programa Photoshop não cria visualizações deste tipo.

(a)



(b)



(c)

Figura 2.8 – Exemplo de clonagem a partir de uma entrada do histórico do *Adobe Photoshop*.

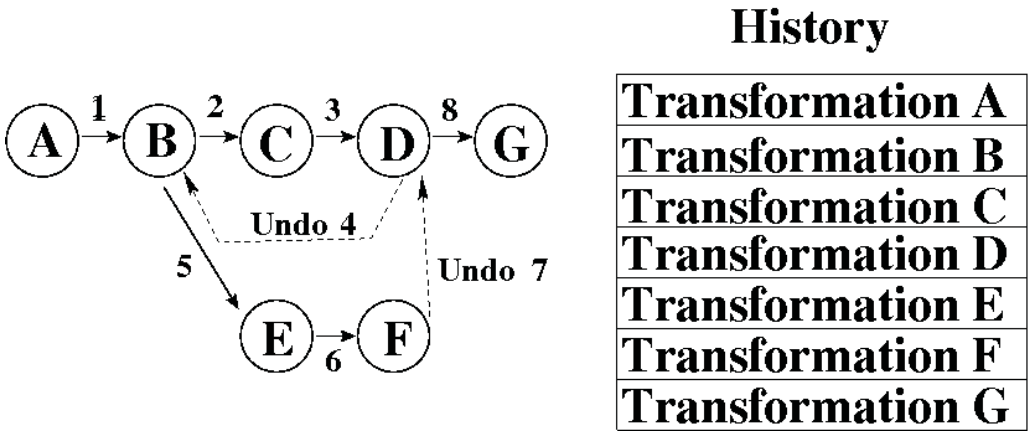


Figura 2.9 – Histórico não linear do *Adobe Photoshop*.

2.4 Programação Visual

Uma das ideias para melhorar a visualização das transformações executadas sobre uma imagem surge com a programação visual: de acordo com diversos autores [13, 18, 22, 23, 24, 27] programação visual é o uso de expressões visuais como imagens ou ícones no processo de programação, com o objetivo de facilitar a visualização do programa, tornando o ensino mais intuitivo para usuários não familiarizados com a lógica de programação e permitindo que usuários mais experientes se preocupem mais com a lógica do programa do que na implementação do código.

Normalmente estes programas utilizam ícones e ligam estes ícones formando um fluxo de operações. Ícones são posicionados manualmente pelo usuário e formam um programa que pode ser não-linear, pois existem operações que recebem mais de um parâmetro de entrada e operações que geram mais de uma saída como por exemplo as ferramentas *Cantata* [27], JLS [18] e SCIL-VP [13], que serão descritas a seguir.

2.4.1 Cantata

Um exemplo de programação visual é o ambiente *Cantata* [27] desenvolvido sobre o sistema *Khoros*, que é um conjunto de funções utilizadas para processamento de imagens, manipulação de dados, processamento de matrizes e visualização de dados. *Cantata* é uma interface gráfica que serve como camada superior para facilitar o uso das funções do *Khoros*. O ambiente também utiliza ícones para representar as funções do programa e linhas para ligar estas funções, como se pode observar na Figura 2.10. De acordo com *Young et al* [27] o ambiente visual aumenta a produtividade pois permite ao usuário desenvolver soluções de maneira mais natural.

Esta interface é utilizada para auxiliar no desenvolvimento de programas que usam sequências de operações para o processamento de imagens. Buscou-se saber mais sobre este ambiente, no entanto, o *link* referenciado no artigo de *Young et al* [27] estava indisponível.

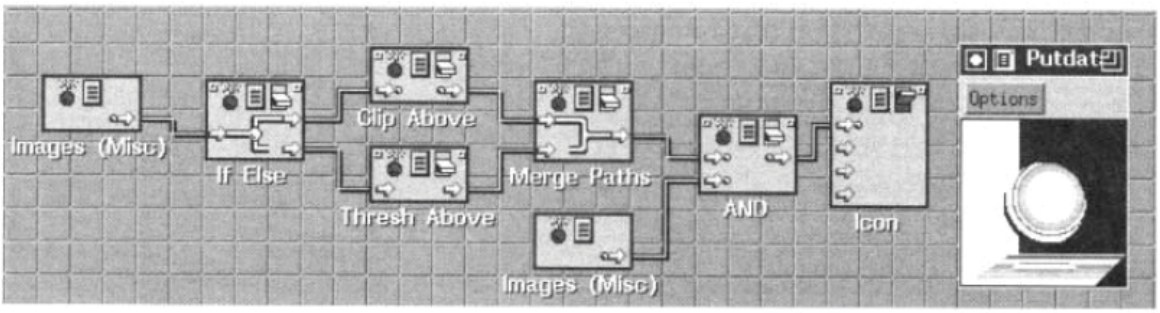


Figura 2.10 – Fluxo de funções criado no ambiente *Cantata*.

2.4.2 JLS

A ferramenta *JLS* foi descrita no artigo *A Pedagogically Targeted Logic Design and Simulation Tool* [18] e consiste em uma ferramenta voltada para o ensino de circuitos lógicos. Os livros que ensinam circuitos lógicos utiliza-se de diagramas lógicos para ilustrar estes circuitos, então uma ferramenta de simulação deve ter uma interface gráfica para que os estudantes possam experimentar o que viram nos livros. Para os autores deste artigo especificações textuais de circuitos são muito complexas e pouco intuitivas para estudantes que nunca viram circuitos lógicos. Eles citam a conhecida frase "*Uma imagem vale mais que mil palavras!*".

Na Figura 2.11 se pode observar a interface da *JLS*, onde existe um menu principal no topo, uma caixa contendo os componentes do circuito, uma área de trabalho ao centro e abaixo um simulador de sinal. Como pode-se observar, o circuito é desenvolvido na área central da ferramenta, onde o usuário seleciona um componente e insere-o na área de trabalho, depois o usuário deve ligar este componente a outros e simular os resultados obtidos.

Operações como *AND* e *OR* possuem dois parâmetros de entrada e um de saída, enquanto operações como *IF* e *ELSE* possuem duas entradas e duas saídas. Isto nos dá uma ideia de fluxo não linear: por exemplo a partir do resultado de um *AND* pode se gerar um fluxo que seja à entrada de um *IF* e *ELSE* gerando dois novos fluxos.

2.4.3 SCIL-VP

No artigo de *Koelma et al* [13] é apresentado um ambiente de programação visual chamado SCIL-VP. A partir de um arquivo de configuração é construída em tempo de execução uma palheta que contém ícones que representam funções. Este arquivo de configuração contém as informações sobre o nome da função, subgrupo de funções à qual pertencem, parâmetros de entrada e saída, o ícone que representa esta função, os parâmetros fixos da função, especificação dos valores permitidos para os parâmetros e uma breve descrição dos parâmetros. O exemplo apresentado no trabalho consiste de funções para a manipulação de imagens e possui funções para abrir, salvar e aplicar transformações sobre uma imagem.

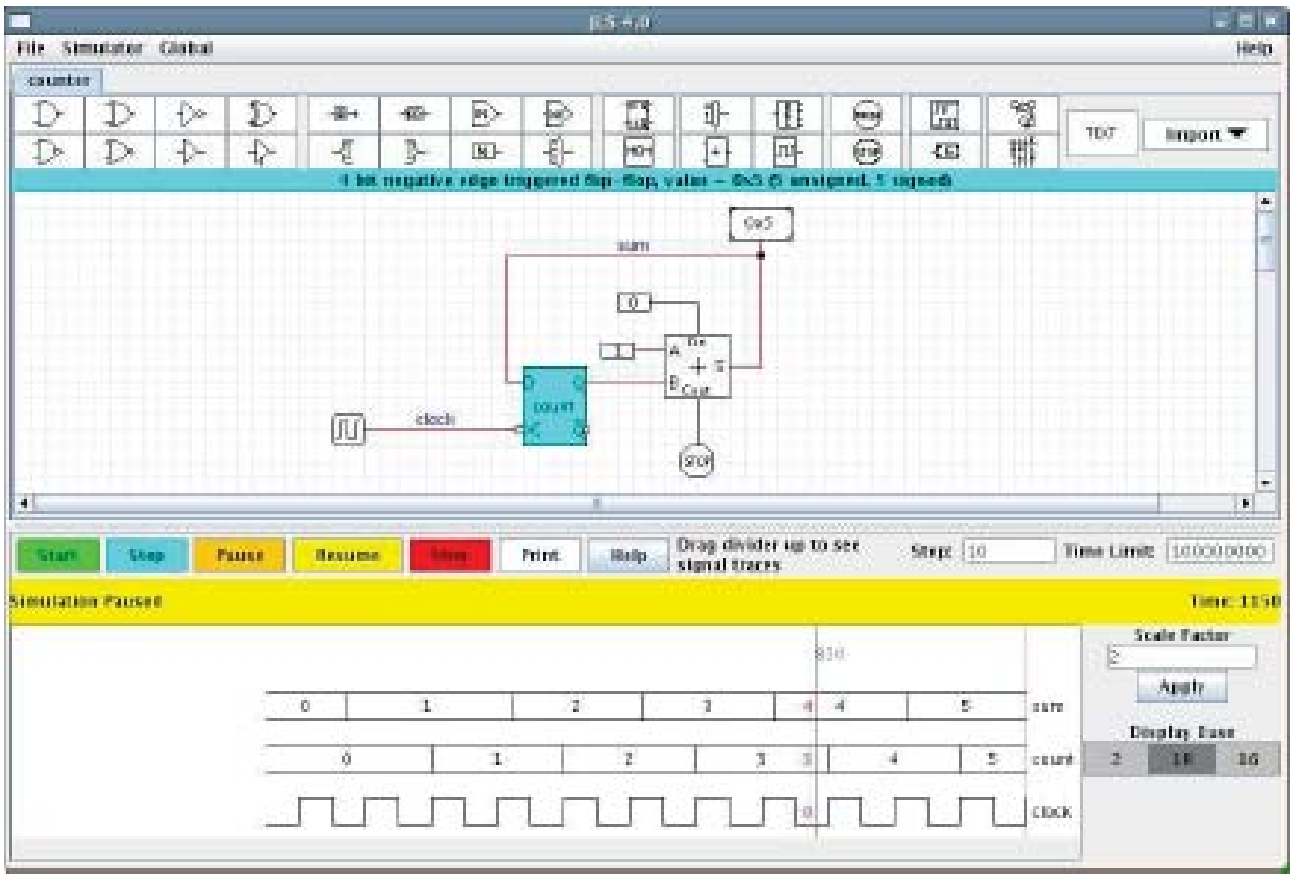


Figura 2.11 – Interface gráfica da ferramenta *JLS* [18].

O ambiente possui uma área de trabalho com um menu no topo, e uma palheta contendo as principais funções do sistema, como apresentado na Figura 2.12. Esta área de trabalho é onde os ícones que representam funcionalidades devem ser adicionados, e possui barras de rolagem tanto na vertical quanto na horizontal para que programas grandes possam ser visualizados.

As funções são representadas por ícones especificados no arquivo de configuração, e caso um ícone não seja especificado o programa irá criar um novo com o nome da função. Estes ícones recebem parâmetros como entrada e podem retornar saídas. Para os parâmetros de entrada foram criados pinos à esquerda do ícone, para representar a saída das funções foram criados pinos à direita. Caso a função tenha parâmetros fixos como por exemplo um limiar, existirá um quadrado acima do ícone. A Figura 2.13 mostra o ícone que representa a função *erosion3x3*, que possui dois parâmetros de entrada, uma saída e possui parâmetros fixos pois existe um quadrado acima do ícone.

Os ícones devem ser interligados pelo usuário, conectando os pinos de saída de um ícone com os pinos de entrada de outro. Estes pinos são tipados, ou seja, o dado de entrada ou saída possui um tipo como, por exemplo inteiro, imagem, bit. A ligação entre os pinos é utilizada para manter a consistência destes dados, ou seja, um pino de entrada que receba como entrada um número não pode ser ligado com um pino de saída que contenha uma imagem por exemplo.

Para visualizar o resultado de uma transformação existe a função *Monitor* que mostra o resultado de uma sequência de transformações em um ícone. Na Figura 2.15 existem dois monitores, o primeiro mostra a imagem original a partir da função *readfile*, que lê o arquivo de entrada, e o

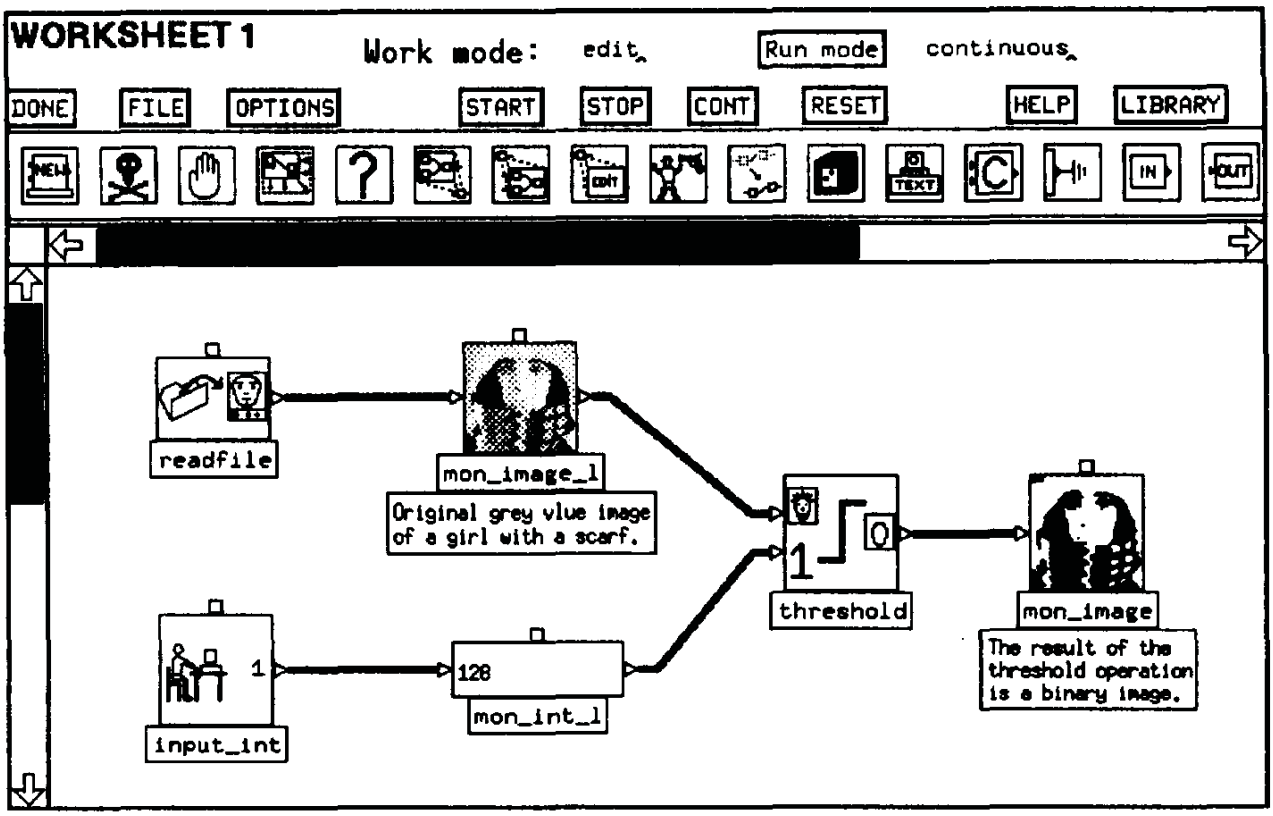


Figura 2.12 – Área de trabalho do ambiente de programação visual do *SCIL-VP* [13].

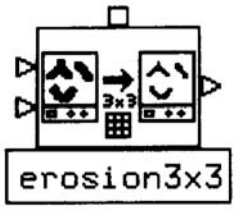


Figura 2.13 – Exemplo de ícone proposto por *Koelma et al* [13]

segundo mostra um fluxo que vem da função *readfile*, passa para a função *isodata_threshold* que nada mais é que um filtro e o resultado desta função é enviado para o segundo monitor.

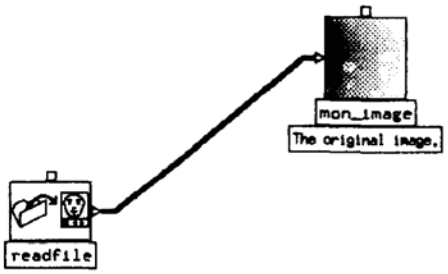


Figura 2.14 – Exemplo de ligação por linha proposto por *Koelma et al* [13]

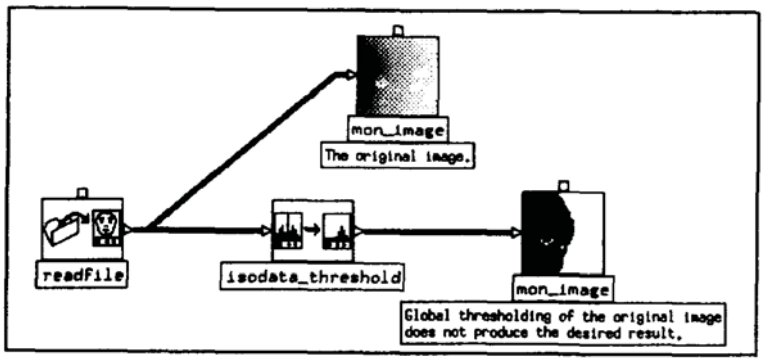


Figura 2.15 – Exemplo de monitores proposto por *Koelma et al* [13]

3. UMA PROPOSTA NÃO-LINEAR PARA O FLUXO DE EDIÇÃO DE IMAGENS

O presente trabalho apresenta um modelo para tornar o fluxo linear de imagens em não-linear. É proposto o uso de uma estrutura de dados para armazenar este fluxo não-linear de entradas do histórico, e na próxima seção são comentadas as estruturas estudadas e a escolhida. Em seguida é abordada a representação gráfica da estrutura, introduzindo elementos da interface gráfica do sistema, juntamente com funcionalidades adicionais que o modelo proporciona descrevendo suas vantagens. Na Seção 3.3 é discutida uma ferramenta para aplicar as transformações solicitadas pelo usuário. Por fim, é abordado o uso de um mecanismo para manter a persistência de dados.

Foi desenvolvido um protótipo semelhante aos programas atuais, onde é apresentada a imagem a ser editada em uma janela com uma barra de ferramentas. O usuário seleciona a ferramenta e aplica a transformação sobre a imagem, como está acostumado. No entanto, o histórico baseado em pilha foi descartado e substituído por uma árvore para permitir fluxos não-lineares. A cada nova transformação é inserido um novo nodo na árvore e cada nodo é uma entrada do histórico. Com o uso de uma árvore é possível ter fluxos não-lineares, pois um nodo pode gerar mais de um nodo filho, possibilitando assim a geração de múltiplos resultados a partir de uma imagem.

3.1 Estrutura de dados

A maioria dos programas de edição de imagens utiliza uma pilha de transformações já que o fluxo mais comum é o linear, no entanto, um fluxo não-linear é difícil de ser implementado usando pilhas. Por exemplo, caso uma transformação resulte em duas imagens, o programa deve criar uma pilha para cada imagem resultante, obrigando-o a administrar estas pilhas.

Além disso o armazenamento em uma pilha causa um problema ao retroceder para alterar uma transformação executada anteriormente. O usuário deve desfazer até a transformação desejada, alterá-la e consequentemente perder as transformações seguintes que foram desfeitas. Uma alternativa para eliminar este problema é o uso de uma lista ao invés de uma pilha. Considerando que cada nodo armazene o resultado de uma transformação, pode-se remover qualquer nodo desta lista sem perder o resto da lista, o que não era possível com o uso de uma pilha. No entanto, esta estrutura continua tendo um fluxo linear de transformações, pois a partir de um nodo não podem ser gerados dois nodos, como no caso de uma imagem resultar em duas novas. A Figura 3.1 apresenta dois tipos de fluxos que não podem ser reproduzidos utilizando listas, a Figura 3.1a apresenta um fluxo onde o nodo *b* se divide em dois novos nodos *c* e *e*, como a operação de recorte sobre duas regiões de imagem, o que não é permitido em uma implementação de lista que só pode ter um único fluxo. O inverso ocorre na Figura 3.1b onde dois fluxos são unidos, o que também não é permitido.

Outra alternativa para armazenar as transformações é utilizar uma árvore, onde um nodo é uma entrada do histórico e armazena a transformação e a imagem resultante em tamanho reduzido.

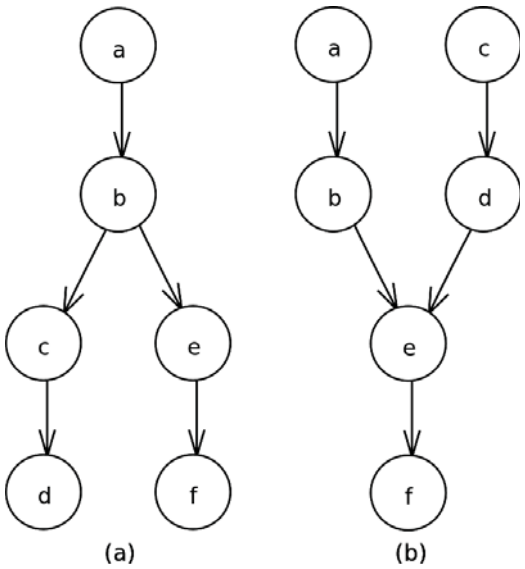


Figura 3.1 – Exemplos de estruturas que não podem ser reproduzidas com listas.

Pela definição de árvore um nodo pai pode ter um ou mais filhos e com esta modelagem uma entrada poderia gerar uma ou mais entradas resultantes, gerando um fluxo não-linear. No entanto, um nodo filho não pode ter mais que um único nodo pai, o que pode não ocorrer no contexto de edição de imagens: por exemplo, pode ser usada uma imagem e uma máscara para gerar outra imagem, logo seria necessário um nodo filho com dois nodos pais, uma para a imagem e um para a máscara. Um exemplo que não pode ser atingido utilizando a estrutura de árvore é apresentado na Figura 3.2, onde o nodo *d* possui dois pais, os nodos *b* e *e*, o que não é permitido pela definição de árvore.

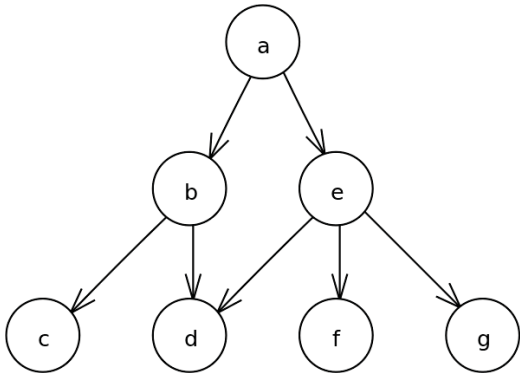


Figura 3.2 – Exemplo de fluxo de transformações que não pode representado com uma árvore.

A utilização de um grafo foi cogitada para armazenar este fluxo, porém o grafo teria que ter características específicas que tornariam o trabalho mais complexo: o primeiro problema detectado seria que o grafo deveria ser dirigido para que o usuário não confunda quem é a entrada e a saída de uma transformação, como mostra a Figura 3.3a, onde o usuário pode confundir qual imagem é o nodo pai e qual é o filho. Outro problema é que este grafo tem que ser acíclico, pois uma imagem não pode ser a entrada e a saída da mesma função, por exemplo, na Figura 3.3b onde o resultado da transformação serve de entrada para a própria transformação.

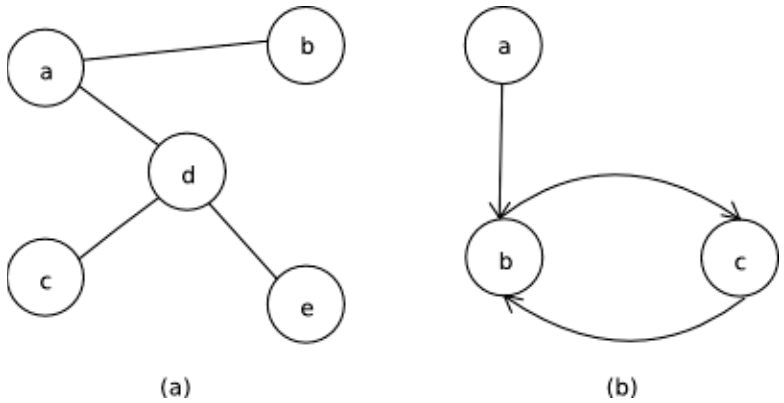


Figura 3.3 – Grafo não dirigido à esquerda e grafo com ciclos à direita.

Optou-se por utilizar uma árvore pois o trabalho visa identificar se a representação de fluxos não-lineares auxilia na edição de imagens. Com o uso de uma árvore é possível visualizar e interagir com fluxos não-lineares, mesmo sem ser permitido gerar o fluxo representado na Figura 3.1b, porém acredita-se que seja suficiente para identificar melhorias no modelo atual.

3.2 Representação Gráfica

O modelo proposto deve ser similar ao utilizado pelas ferramentas de edição de imagens atuais, que possuem uma janela principal contendo um menu com as funções básicas como abrir imagem, salvar imagem, sair, etc., uma área de trabalho onde a imagem é disposta e uma barra de ferramentas. Em outra janela deve ser apresentado o histórico da edição, contendo as entradas que representam as transformações executadas.

O histórico é a representação gráfica da estrutura apresentada na seção anterior, posicionada na janela secundária do sistema. Baseando-se na programação visual é possível definir elementos gráficos para representar estas entradas. O nodo de uma árvore é um retângulo contendo uma imagem resultante em tamanho reduzido e o nome da transformação que a gerou. Estes retângulos podem ser ligados indicando as transformações em sequência. Na Figura 3.4 pode-se observar um exemplo de entrada no histórico, onde é ampliada a transformação de um filtro para inversão horizontal (*flop*) em uma imagem colorida.

A raiz da árvore deve ser a imagem original e todas as entradas devem ter um caminho da raiz até a entrada, pois este caminho indica as transformações executadas desde a imagem original até a resultante. Este caminho é o fluxo linear de uma edição e o conjunto de caminhos forma a árvore gerando um fluxo não-linear.

Quando uma nova transformação é executada o sistema deve inserir a nova entrada como filha desta e esta filha deve se tornar a corrente. Na representação a entrada corrente deve ser sinalizada de forma que seja possível distingui-la das outras, indicando em qual imagem será aplicada a próxima transformação.

Figura 3.4 – Exemplo de uma entrada do histórico representando uma operação de conversão horizontal (*flop*).

De acordo com *Meng* [15] é aconselhável que o usuário possa retroceder de forma arbitrária e direta, ou seja, o usuário visualiza todas as entradas do histórico e seleciona para qual retroceder. Com o intuito de permitir esta funcionalidade as entradas devem ser selecionáveis através de um duplo clique do mouse sobre a entrada desejada. Quando uma nova transformação for executada, um novo ramo da árvore deve ser criado, ao invés de sobrescrever as posteriores, mantendo assim todas as entradas no histórico.

Analisando a árvore durante o processo de desenvolvimento do protótipo observou-se que seria interessante possibilitar ao usuário copiar ou mover sub-árvores, ou seja, copiar ou mover um ramo inteiro da árvore de uma entrada para outra. Ao faze isto, todas as transformações a partir de uma entrada até as folhas são executadas no resultado de outra entrada. Por exemplo, na Figura 4.14 quando o usuário copiar a entrada *c* para a entrada *f*, o sistema deve executar as transformações *c*, *d* e *e* utilizando a imagem resultante da entrada *f* ao invés da entrada *b*, gerando as entradas *g*, *h* e *i*, marcadas com uma linha pontilhadas. O mesmo ocorre ao mover um galho, porém as transformações são removidas da entrada origem e inseridas abaixo da entrada destino.

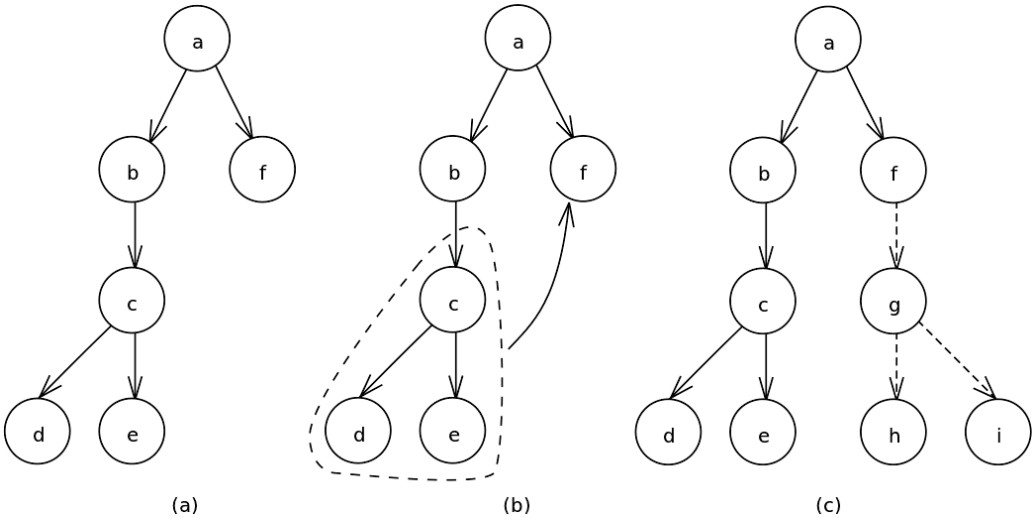


Figura 3.5 – Exemplo de uma operação de cópia.

Na representação desta árvore deve existir um método para ampliar as entradas, pois um dos problemas detectados nos modelos atuais é a dificuldade em se comparar resultados intermediários. Para solucionar este problema o sistema deve possibilitar as funcionalidades *zoom in* e *zoom out*. Artigos como [13, 27, 3] afirmam que o uso de *Drag and Drop* é essencial para a programação visual, pois facilita a manipulação dos ícones. Seguindo esta linha de raciocínio, foi decidido que as entradas devem ser móveis, ou seja, cada entrada pode ser movida para que seja possível deixar duas entradas distantes mais próximas para facilitar a comparação entre os resultados.

A janela secundária possui um tamanho grande e com barra de rolagem, pois o trabalho de *Burnett* [3] afirma que um dos principais problemas da programação visual é a área de trabalho, pois programas grandes necessitam de uma área de trabalho grande. Além disso, quando é utilizada uma pilha para representar o histórico todas as entradas possuem a mesma largura, fixando assim a largura do histórico, o que pode não acontecer com uma árvore, pois um nodo pode gerar um ou mais nodos filhos, logo a largura do histórico irá depender do número de nodos filhos inseridos na árvore.

3.3 *Engine* de Transformação

O trabalho de *Sim* [24] apresenta um ambiente de programação visual para criar fluxos para o processamento de imagem. No modelo apresentado por *Sim* a interface gráfica é executada no computador do cliente, mas o processamento da imagem é executado em sistemas distribuídos de forma transparente para o usuário. Com base neste trabalho, foi definido que o modelo não deve se preocupar em aplicar as transformações, e sim em como interagir e representar para o usuário, ou seja, deve ser definida uma *engine* externa para aplicar as transformações. Neste caso a interface gráfica solicita à *engine* para aplicar as transformações, e quando necessário, a interface solicita parâmetros da transformação para o usuário.

Antes de a interface gráfica enviar parâmetros para a *engine* é necessário que o protótipo valide estes valores, pois a *engine* é responsável apenas por aplicar a transformação. Um exemplo é a situação de um recorte (*crop*) pois esta transformação necessita de uma região sobre a imagem para ser recortada e o usuário pode ter selecionado uma região fora da imagem, gerando um erro na *engine*.

Na Seção 4.2 é apresentada a *engine* escolhida para realizar as transformações.

3.4 Persistência dos Dados

A persistência dos dados do histórico é uma das deficiências dos programas atuais pois neles o histórico é descartado após o encerramento da sessão e se o usuário quiser este histórico de volta será impossibilitado. Com a perda deste histórico tem-se a perda das imagens intermediárias,

pois elas são armazenadas no histórico e se o usuário não as salvou estas serão perdidas. Outro problema detectado é que o usuário fica impossibilitado de retroceder pois o histórico fica vazio.

Uma vantagem de manter este histórico é a possibilidade de visualizar as transformações que já foram executadas, para que o usuário possa identificar os passos dados para chegar a uma determinada imagem. Por estas razões, acredita-se que deva existir um mecanismo para salvar as transformações e os parâmetros utilizados para replicar a edição.

O programa *GIMP* [7] possibilita o uso de *scripts* contendo comandos para aplicar transformações sobre uma imagem. No entanto, estes *scripts* devem ser escritos em uma linguagem própria do programa. Este mecanismo poderia ser utilizado para manter a persistência dos dados, no entanto o usuário necessita conhecer esta linguagem específica. Baseando-se na abordagem do *GIMP* optou-se por armazenar o histórico em um arquivo texto que deve ser interpretado pelo protótipo para restaurar uma edição.

Uma das vantagens de armazenar o histórico em um arquivo texto é o espaço em disco gasto, pois uma edição pode gerar várias imagens e mantendo apenas o histórico e a imagem original é possível criar novamente as imagens resultantes, logo não é necessário armazenar todos os resultados gerados e quando for solicitado é gerado novamente. No fim é uma troca de espaço em disco por tempo de processamento.

4. PROTÓTIPO DESENVOLVIDO

Este capítulo faz uma descrição geral do protótipo desenvolvido para a avaliação do modelo proposto, além de uma descrição sobre sua arquitetura, abordando a ferramenta utilizada para aplicar as transformações e os métodos utilizados para manter a persistência do histórico. Por fim, são apresentadas as operações que podem ser executadas neste protótipo, tanto na edição da imagem quanto na interação com o histórico.

4.1 Descrição Geral

A principal tarefa do protótipo é servir como plataforma para avaliar como os usuários se sentem ao interagir com a representação do histórico utilizando uma árvore ao invés de uma pilha de transformações e se os usuários se sentem confortáveis visualizando o histórico organizado em forma de árvore. Como objetivo secundário foi avaliada a utilidade de alguns mecanismos implementados, como *copy* e *move*. O protótipo também serviu para avaliar a forma de automatizar a edição de um conjunto de imagens a partir de uma edição. Nesta seção serão discutidos os principais elementos criados para a implementação do protótipo, incluindo o algoritmo utilizado para organizar visualmente os elementos da árvore.

A primeira parte do protótipo a ser implementada foi a estrutura de dados para armazenar a árvore criada durante o processo de edição de imagens. Pela definição de árvore, todos os nodos, exceto o nodo raiz, têm de ter um único nodo pai e cada nodo pode ter nodos filhos. Foram definidos alguns atributos que um nodo deve ter:

- Nodo pai: é uma referência para o nodo pai, sendo que para a raiz este será nulo;
- Id: um inteiro para facilitar na busca por um nodo;
- Transformação: o nome da transformação que será aplicada;
- Parâmetros da transformação: atributo não obrigatório, pois algumas transformações não necessitam de parâmetros;
- Nome da imagem resultante: armazena o nome da imagem criada;
- Nome da imagem de entrada: é o nome da imagem resultante do nodo pai;
- Lista de filhos: uma lista contendo todos os filhos deste nodo;

Com estes nodos foi implementada uma árvore para armazenar os dados gerados durante uma edição. A árvore possui um nodo raiz que é criado pela operação de abrir uma imagem (*open*) e os demais são transformações aplicadas sobre a imagem. Um nodo transmite seu resultado como entrada para cada nodo filho e este gera uma nova imagem resultante.

Cada nodo é representado por um ícone como no trabalho de *Koelma* [13]. No trabalho de *Meng* [15] os ícones tinham tamanhos variados que indicavam uma ordem cronológica das edições, no entanto todos os ícones deste trabalho tem tamanho igual, pois a ordem cronológica é mantida pelo caminho da imagem original até um nodo folha. Estes ícones mudam de acordo com o *zoom* aplicado sobre o histórico. O ícone é dividido em duas partes, no topo a operação e abaixo o resultado em tamanho reduzido como mostra a Figura 4.1. Por padrão o ícone possui um espaço de 80 *pixels* para a largura e 96 *pixels* para a altura, sendo que 80x16 *pixels* são para o texto indicando a operação e 80x80 *pixels* para o resultado em miniatura, como mostra a Figura 4.1. Para que o resultado seja inserido na região é preciso redimensionar a imagem sem distorcer, mantendo a razão de aspecto, pois a região tem tamanho fixo de 80x80 e os resultados podem ter tamanhos variados e por consequência razões de aspecto diferentes. Por isso é calculado o tamanho da imagem maximizando a altura ou a largura na região. Para isto é necessário calcular a razão de aspecto da imagem e da região reservada, com as fórmulas:

```
razaoRegiao = larguraRegiao / alturaRegiao e
razaoImagem = larguraImagem / alturaImagem,
```

A razão de aspecto é maior que um se a largura for maior que a altura e menor que um se a altura for maior que a da região. Em seguida são comparadas as duas razões de aspecto, gerando as seguintes alternativas:

- $razaoImagem == razaoRegiao$: a largura e altura são maximizadas pois a razão de aspecto é a mesma;
- $razaoImagem > razaoRegiao$: isto significa que a imagem é mais larga que a região, então maximiza-se a largura e a altura da imagem é recalculada da seguinte forma: $novaAltura = larguraRegiao / razaoImagem$;
- $razaoImagem < razaoRegiao$: isto significa que a imagem é mais alta que a região, então maximiza-se a altura e a largura é recalculada com: $novaLargura = alturaRegiao * razaoImagem$.

Utilizando estes cálculos é possível maximizar a imagem do ícone sem distorcer.

Para dispor as entradas do histórico foi investigada a possibilidade de utilizar o pacote *Graphviz* [6] que contém várias ferramentas que geram imagens de um grafo com os nodos organizados. Este programa gera diversos formatos de imagens, como JPEG, PNG e SVG. Este programa poderia ser utilizado para gerar a organização da árvore na janela secundária, no entanto, a saída é uma imagem estática, isto tornaria o histórico estático e não permitiria ao usuário retroceder para uma determinada entrada. Outra desvantagem de utilizar um imagem para representar o histórico é que não é possível mover os nodos e esta é uma funcionalidade necessária para comparar dois nodos distantes.

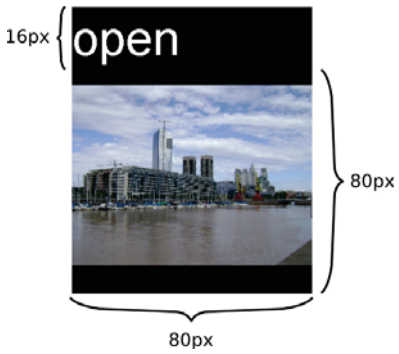


Figura 4.1 – Representação do ícone, na parte superior a operação e abaixo o resultado gerado.

Para tornar esta imagem dinâmica poderia ser criada uma imagem em formato *SVG*, usar as posições dos elementos e reproduzi-las, tornando o trabalho complexo e demorado pois o sistema tem que encontrar os elementos do *SVG* e remapeá-los na janela secundária. Por estes motivos descartou-se o uso do pacote *Graphviz*.

Para a organização da árvore foi desenvolvido um algoritmo que dispõe os ícones de forma que estes não se sobreponham e que seja possível identificar facilmente qual é o nodo pai de um nodo. Considerando estas regras foi desenvolvida uma função que recebe como parâmetros um nodo e duas coordenadas (P_x e P_y) que indicam a posição onde o nodo é desenhado, estas coordenadas informam a posição horizontal (P_x) e vertical (P_y). Esta função retorna um valor inteiro que indica a última posição na horizontal que foi inserido um nodo, para que não exista sobreposição. A árvore é inicializada no canto superior esquerdo, dando uma margem de 10 *pixels*. Para a primeira chamada é passado como parâmetro o nodo raiz e 10 *pixels* da margem para P_x e P_y. Em seguida é aberta uma recursão para a mesma função para os filhos deste nodo modificando estas coordenadas. Todos os filhos de um nodo devem estar abaixo deste nodo, para indicar a paternidade e manter uma hierarquia, então P_y é incrementado para os nodos filhos da seguinte forma:

$$P_y = P_y + h + d_y,$$

onde h é a altura da entrada e d_y é uma distância entre as duas entradas, como é mostrado na Figura 4.2a. A função faz um caminhamento em profundidade, inserindo os nodos cada vez mais abaixo até encontrar um nodo folha. No caso de nodos irmãos é necessário incrementar P_x para evitar a sobreposição das entradas, movendo o próximo irmão para direita pois a altura é a mesma para os dois. O valor de retorno da função é a última posição indicada pela coordenada P_x que é calculada por:

$$P_x = P_x + w + d_x,$$

onde w é a largura do ícone e d_x é a distância entre dois nodos na horizontal como mostra a Figura 4.2b.

O pseudo código do algoritmo é apresentado no Algoritmo 4.3. A função **createTree** recebe a raiz e as coordenadas P_x e P_y inicializando com valores iguais a 10. Esta função insere

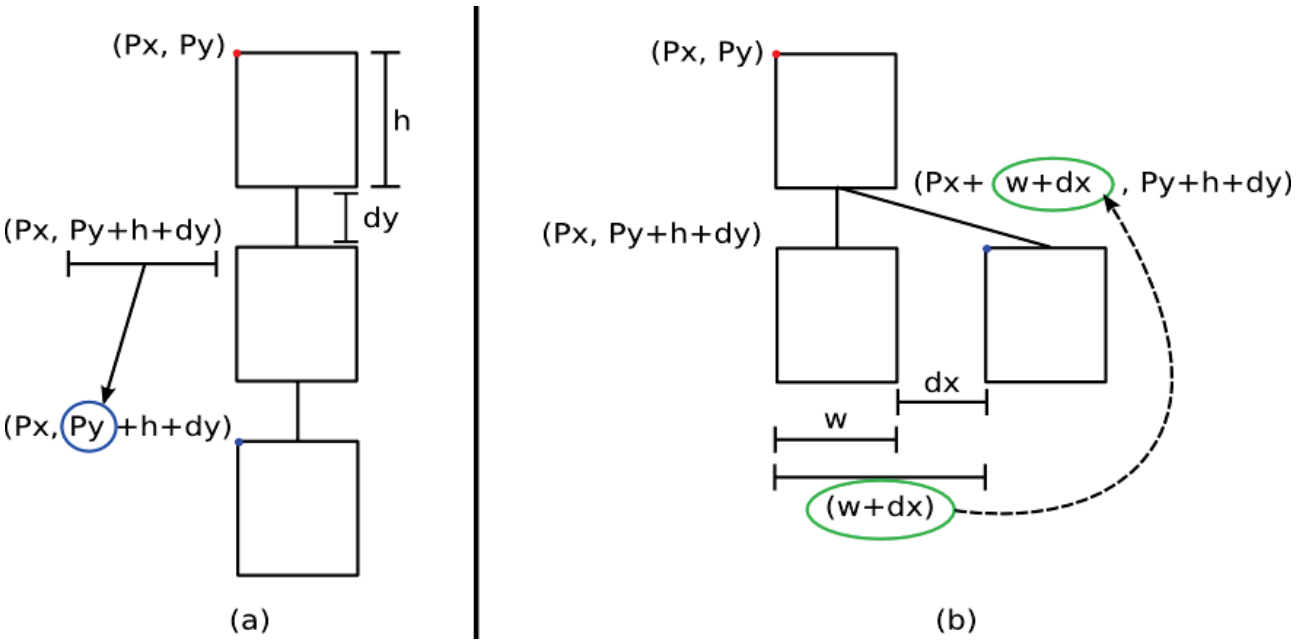


Figura 4.2 – (a) Posicionamento vertical (b) Posicionamento horizontal

o ícone na posição indicada por P_x e P_y através da função **insert** e abre a recursão para os nodos filhos deslocando os eixos na vertical e na horizontal.

1: **function createTree**(P_x, P_y, *nodo*)
2: **insert**(P_x, P_y, *nodo*)
3: **for** *filho* **in children**(*nodo*) **do**
4: $P_x \leftarrow$ **createTree**(P_x, $P_y + h + d_y$, *filho*);
5: **if** *filho* $\neq$ *ultimofilho* **then**
6: $P_x \leftarrow P_x + w + d_x$;
7: **end if**
8: **end for**
9: **return** P_x
10: **createTree**(10, 10, *nodoRaiz*)

Figura 4.3 – Algoritmo que posiciona os ícones.

Para a interface gráfica foram usadas duas janelas para o usuário interagir com o sistema. A primeira é a janela principal onde se encontram as ferramentas para editar, uma área onde será disposta a imagem editada e um menu principal, com opções como abrir imagem, salvar imagem, etc. A janela secundária foi desenvolvida para mostrar e interagir com a árvore gerada durante a edição de uma imagem e possui um menu com opções de *zoom in* e *zoom out* e uma área onde é organizada a árvore, como mostra a Figura 4.4.

Para manter a persistência dos dados foi desenvolvido um formato *XML* para armazenar a árvore gerada. O formato *XML* foi escolhido por permitir o uso de *tags* de forma recursiva. Este mecanismo foi utilizado para identificar os nodos filhos de um nodo, de forma que a *tag* mais externa é o nodo pai e as internas são os nodos filhos e estes podem ter nodos filhos de forma recursiva. Um exemplo é apresentado na Figura 4.5, onde acima é mostrado o *XML* e abaixo

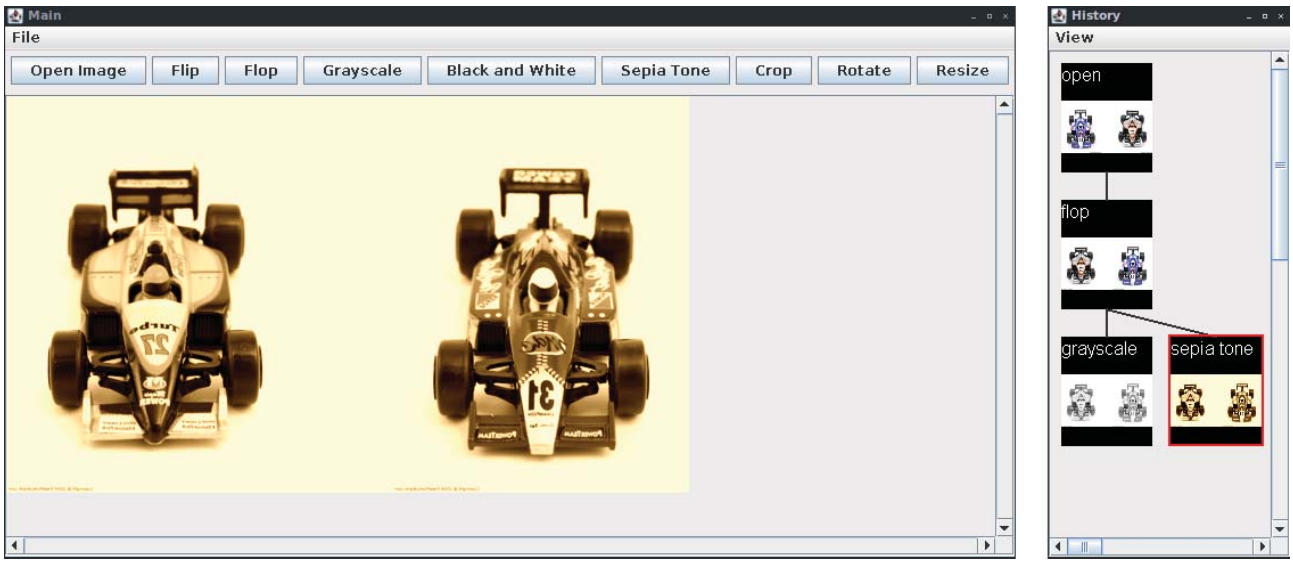


Figura 4.4 – Janela principal à esquerda e a secundária à direita.

a árvore correspondente. No *XML* a *tag* mais externa é a função de abrir uma imagem (*open*). Dentro desta são apresentadas as funções de inversão horizontal (*flop*) e conversão para tons de cinza (*grayscale*). A *tag flop* contém as operações de *sépia* (*sepia tone*) com parâmetros de 70% e 80%.

O arquivo *XML* não armazena as posições dos nodos no histórico: o programa lê o arquivo e monta a árvore. Isto diminui o número de informações que devem ser armazenadas no arquivo e o algoritmo que posiciona as entradas é determinístico, logo as entradas serão inseridas nas mesmas posições.

4.2 Arquitetura do Protótipo

A arquitetura do protótipo foi dividida em 3 camadas: a primeira é a interface gráfica que irá servir de meio para o usuário interagir com o protótipo, apresentando o fluxo não-linear gerado pela edição; a segunda camada é a estrutura para armazenar este fluxo e por fim, a camada que irá processar as edições solicitadas pelo usuário. Estas camadas devem se comunicar: o usuário aplica uma transformação, a interface recebe a requisição e solicita para a camada da *engine* para processar e retornar o resultado, então esta nova entrada é armazenada na estrutura e o resultado é enviado para a interface gráfica. Uma representação da comunicação entre as camadas é apresentada na Figura 4.6.

Para o desenvolvimento das janelas da interface gráfica foi realizado um primeiro experimento com uma interface gráfica para a *Web*, programada em *PHP*, *JQuery*, *Javascript*, *CSS* e *HTML*. Porém o protótipo era lento pois a cada entrada do histórico era necessário carregar uma nova imagem. Portanto, optou-se por buscar uma nova plataforma para desenvolver o protótipo, considerando que esta nova não fosse necessariamente voltada para a *Web*, pois não é um dos focos deste trabalho. Foi desenvolvida uma interface utilizando apenas a linguagem *JAVA* pois esta é

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<history>
    <entry id="1" input="img_1047.jpg"
           operation="open" output="img_1047-1.jpg" parameters=" " path="/home/seki" region="">
        <entry id="2" input="img_1047-1.jpg"
               operation="flop" output="img_1047-2.jpg" parameters=" " path="/home/seki" region="">
            <entry id="4" input="img_1047-2.jpg"
                   operation="sepia tone" output="img_1047-4.jpg" parameters="70% " path="/home/seki" region=""/>
            <entry id="5" input="img_1047-2.jpg"
                   operation="sepia tone" output="img_1047-5.jpg" parameters="80% " path="/home/seki" region=""/>
        </entry>
        <entry id="3" input="img_1047-1.jpg"
               operation="grayscale" output="img_1047-3.jpg" parameters=" " path="/home/seki" region=""/>
    </entry>
</history>
```

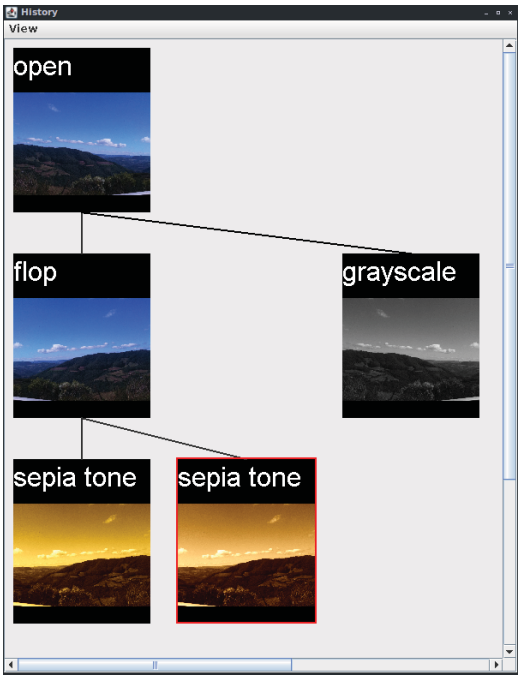


Figura 4.5 – Árvore armazenada em formato XML.

capaz de criar elementos gráficos como os propostos neste trabalho. Na Figura 4.4 é apresentada a janela principal do protótipo na esquerda, com o menu superior, a barra de ferramentas e a área de trabalho onde a imagem editada é apresentada.

A janela principal apresentada à esquerda da Figura 4.4 é responsável por permitir que o usuário edite a imagem e a cada nova transformação executada na janela principal é adicionada uma nova entrada na árvore disposta na janela secundária.

À direita da Figura 4.4 é apresentada a janela secundária com uma área onde serão dispostas as entradas do histórico. No menu superior estão as funcionalidades de *zoom in* e *zoom out* como é possível identificar na Figura 4.7. A área onde as entradas do histórico serão inseridas tem um tamanho fixo e possui barras de rolagem.

O histórico mantém um fluxo constituído de transformações aplicadas sobre a imagem original para gerar um resultado. Na Figura 4.8 é apresentado um exemplo deste fluxo onde a operação de abrir uma imagem (*open*) recebe como entrada a imagem original *IMG.png* e a partir desta é criada uma cópia gerando *IMG-1.png*, em seguida esta cópia é utilizada como entrada da conversão para tons de cinza (*grayscale*) gerando como saída *IMG-2.png*. Esta serve de entrada

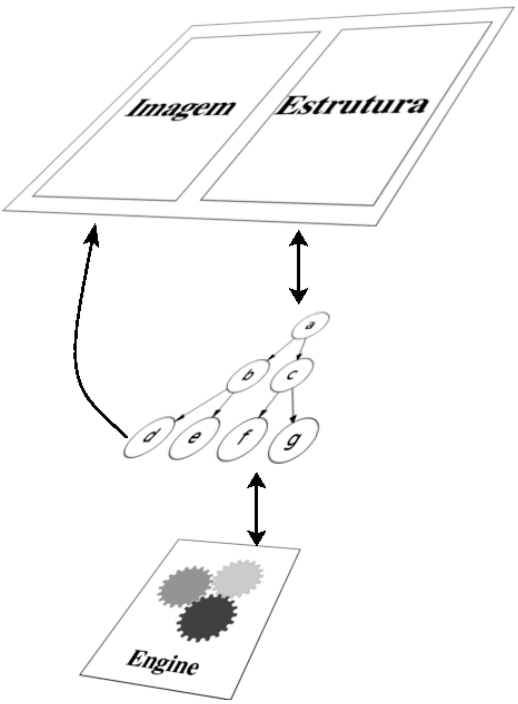


Figura 4.6 – Representação das camadas do protótipo.

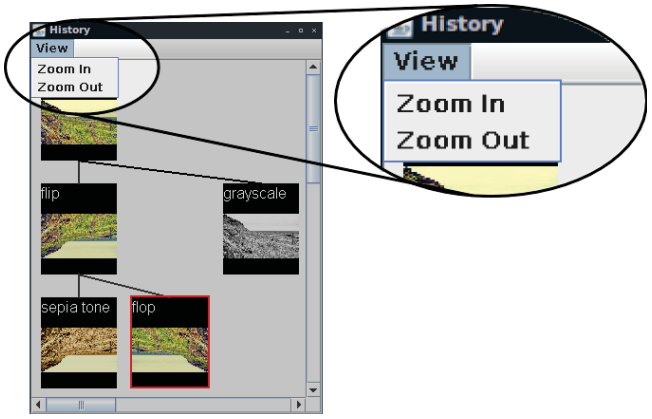


Figura 4.7 – Menu contendo as funcionalidades de *zoom in* e *zoom out*.

para a operação de inversão horizontal (*flop*) que resulta na imagem *IMG-3.png*. Foi implementado um padrão para a nomenclatura das imagens, que consiste no nome da imagem original seguido de um "-" e um número que é o identificador (*id*) da transformação.

Os elementos são renderizados através da biblioteca *Java 2D* que permite desenhar linhas, retângulos, círculos, imagens, entre outros. Para dispor as imagens para o usuário foi desenvolvida uma classe que estende a classe *JPanel* e sobrescreve o método *paintComponent* para renderizar as entradas do histórico dentro do *JPanel*, organizando os nodos utilizando o algoritmo descrito anteriormente.

O protótipo foi construído de maneira que a interface gráfica recebesse as ações do usuário, enviasse estas informações para a *engine* e esta processasse e retornasse o resultado para a interface gráfica. Para a *engine* foi utilizado o pacote *ImageMagick* [11] que contém programas para editar imagens, normalmente executados por linha de comando em um terminal. O programa escolhido

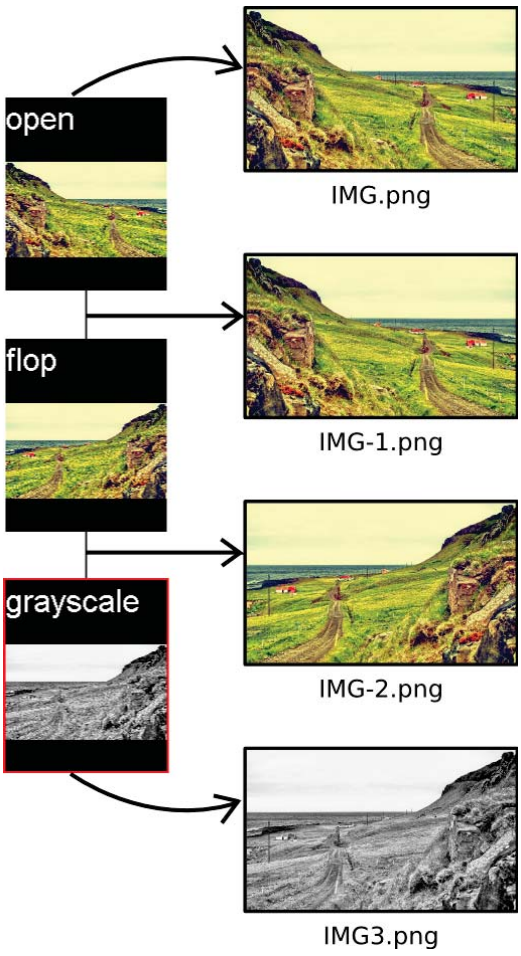


Figura 4.8 – Descrição do fluxo de edição do modelo proposto.

para executar as transformações foi o *convert*, pois recebe como entrada uma imagem e gera uma nova imagem aplicando a transformação desejada com seus parâmetros. Este programa foi escolhido pois permite a execução de várias transformações para editar uma imagem e facilita a geração de *scripts* para automatização, pois cada transformação gera um comando que pode ser executado externamente utilizando uma chamada de sistema.

Foi desenvolvido um método no qual o protótipo faz uma chamada de sistema passando a imagem corrente, a transformação com seus parâmetros e o nome da imagem resultante. Após a conclusão desta chamada a interface gráfica atualiza a imagem corrente com o resultado da transformação e uma nova entrada no histórico é inserida na janela secundária. Em seguida são apresentados os comandos utilizados durante a edição apresentada na Figura 4.8:

```
convert IMG.png IMG-1.png
convert IMG-1.png -flop IMG-2.png
convert IMG-2.png -colorspace Gray IMG-3.png
```

Considerando que uma edição é formada por comandos executados pela *engine*, pode-se alterar a imagem de entrada para aplicar toda a edição sobre outra imagem, ou seja, alterando a primeira imagem a edição será aplicada sobre esta outra imagem. Isto foi utilizado para facilitar a

automatização de uma sequência de edições sobre um conjunto de imagens. De acordo com *Cook* [4] o usuário deve ser capaz de automatizar suas tarefas de forma simples e fácil, e o autor acredita que deve existir uma forma intuitiva que não necessite que o usuário saiba programar para automatizar suas tarefas. Considerando o trabalho de *Cook* foi desenvolvida uma funcionalidade para aplicar todas as transformações de uma edição sobre um conjunto de imagens, por exemplo, converter todas as imagens de uma pasta para tons de cinza. Esta funcionalidade abre uma edição e quando o usuário selecionar uma pasta, todas as imagens desta sofrerão as transformações executadas nesta edição. Um exemplo desta automatização pode ser o redimensionamento de uma pasta contendo diversas imagens para um tamanho reduzido com apenas um clique.

Uma edição pode gerar vários resultados distintos e preservando o histórico e a imagem original é possível restaurar estes resultados. Uma vantagem de armazenar o histórico é a economia de espaço em disco, pois para gerar todos os resultados de uma edição é necessário armazenar apenas o histórico e a imagem original e quando solicitado o usuário gera os resultados. Por exemplo na Figura 4.11 é apresentada uma edição convertendo a imagem original para tons de cinza e duas variações de sépia. Considerando que a imagem tem cerca de 1MB e cada resultado tenha 1MB, salvando as 4 imagens tem-se um gasto de 4MB. Ao armazenar apenas o *XML* e a imagem original é gasto aproximadamente 1MB, pois um arquivo *XML* contendo esta edição tem aproximadamente 10 KB.

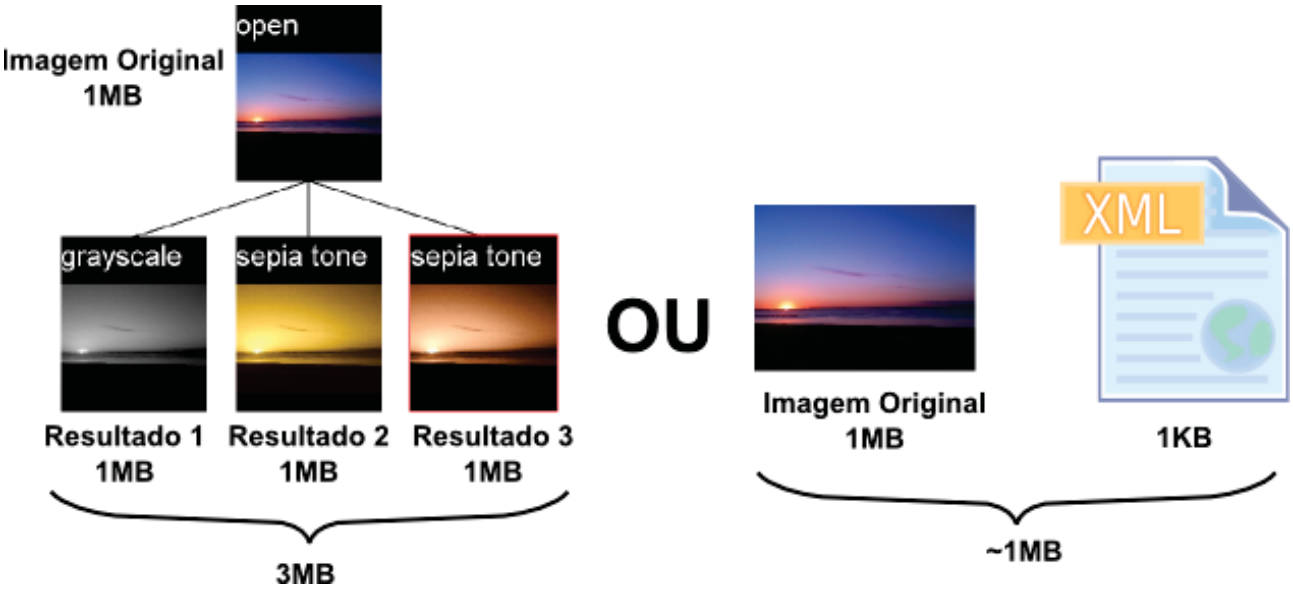
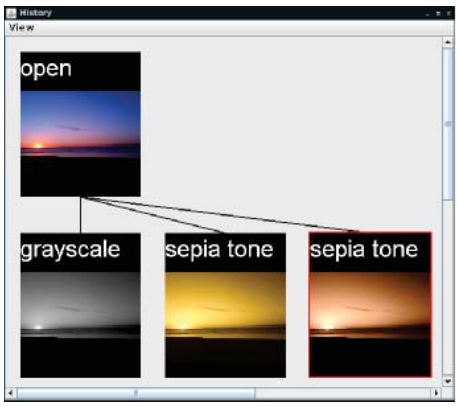


Figura 4.9 – Exemplo de economia de espaço em disco armazenando apenas arquivo *XML* e a imagem original.

Neste trabalho foi apresentada uma forma de armazenar esta edição em um arquivo *XML*, porém este formato é específico e compreendido apenas pelo protótipo. Para promover versatilidade foi criado uma forma de armazenar e replicar a edição sem a necessidade do protótipo. Optou-se por armazenar em um *script* que possa ser interpretado pelo *shell* [16, 19]. O protótipo cria este *script* contendo os comandos executados durante uma edição possibilitando salvar e replicar a edição.

Uma vantagem de um *script* para armazenar a edição é a possibilidade de executar a edição sobre outra imagem, pois a entrada do *script* é a imagem original. Um exemplo é apresentado na Figura 4.10 onde à esquerda é mostrada a árvore da edição e à direita o *script* correspondente.




```
#!/bin/bash
if [ ! -e "$1" ] ; then
        echo "$1: No such file.";
        exit 0;
fi
file=${1%.*};
convert $1 ${file}-1.png;
convert $file-1.png -colorspace Gray ${file}-2.png;
convert $file-1.png -sepia-tone 70% ${file}-3.png;
convert $file-1.png -sepia-tone 90% ${file}-4.png;
```

Figura 4.10 – À esquerda a árvore gerada em uma edição e à direita o *script* correspondente.

Para rodar este *script* basta executá-lo passando o caminho da imagem. Por exemplo, o *script* da Figura 4.10 foi salvo com o nome "edicao1" e é chamado da seguinte forma:

$./edicao1 < file >$

Considerando que o *script* pode ser executado sobre qualquer imagem, pode se armazenar apenas as imagens originais e o *script*, economizando espaço em disco como o armazenamento em formato *XML*. É possível aplicar esta edição sobre uma pasta utilizando o comando *for* do *shell*. Por exemplo, é possível aplicar o *script* "edicao1" sobre todas as imagens da pasta *Pictures* da seguinte forma:

$for\ i\ in\ Pictures/* \ ;\ do\ ./edicao1\ "\$i";\ done$

Seguindo o mesmo exemplo descrito anteriormente é possível economizar espaço em disco armazenando apenas o *script* e as imagens originais. Por exemplo na Figura 4.11 é aplicada uma edição sobre uma pasta contendo 100 imagens, são gastos aproximadamente 400MB para armazenar as imagens originais e os resultados da edição. Por outro lado, ao armazenar o *script*, que tem em média 1KB e as imagens originais, o espaço gasto será de 100MB, ganhando assim uma troca de espaço em disco por tempo de processamento pois quando o usuário quiser visualizar as imagens resultantes deverá aplicar a edição sobre a pasta.

Na próxima seção serão abordados os comandos disponíveis no protótipo para editar a imagem e as funcionalidades adicionadas no histórico com o para facilitar a execução de tarefas do usuário.

4.3 Operações Disponíveis no Protótipo

Nesta seção serão abordadas as operações disponíveis para o usuário interagir com o protótipo. São descritas as operações para editar a imagem, funcionalidades para salvar as imagens

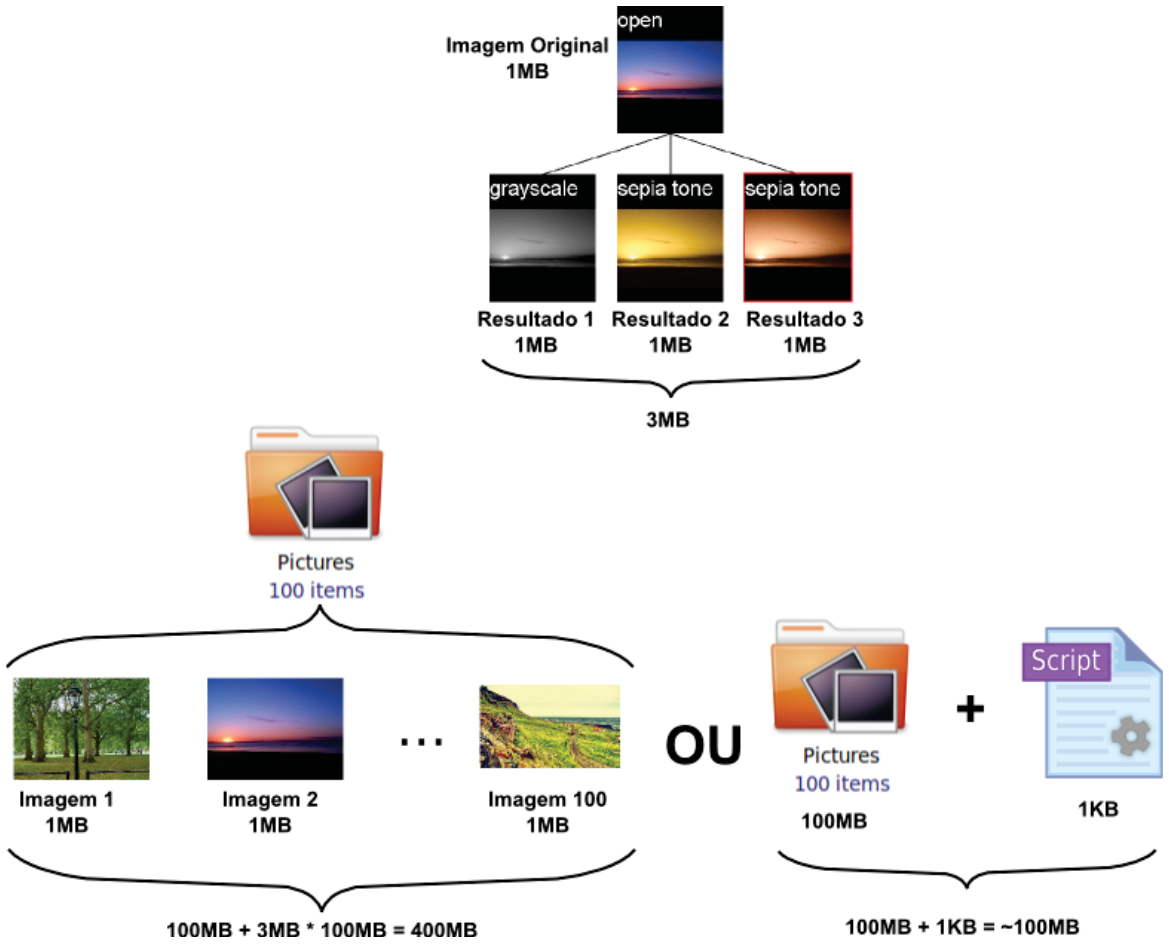


Figura 4.11 – Exemplo de economia de espaço em disco armazenando um *script* para editar todas as imagens de uma pasta.

e os projetos, e outras. Em seguida são abordadas as operações da janela secundária criadas para a interação com a árvore gerada.

Neste trabalho foi considerado que um projeto é uma edição na qual o usuário define as transformações que são executadas sobre uma imagem e seus parâmetros. No menu da janela principal estão as operações:

- Abrir imagem (*Open Image*): apresenta uma janela para escolher uma imagem para editar, como mostra a Figura 4.12;
- Salvar imagem (*Save Image*): mostra uma janela para identificar onde a imagem vai ser salva e o usuário informa o nome da nova imagem;
- Abrir projeto (*Open Project*): mostra a janela para escolher um projeto para ser aberto;
- Salvar projeto (*Save Project*): abre uma janela parecida com a de Salvar Imagem se o projeto não for salvo nenhuma vez, caso contrário somente salva;
- Salvar projeto como (*Save Project As*): abre uma janela semelhante a Salvar Imagem;

- Aplicar em uma pasta (*Apply in Folder*): aplica a edição corrente sobre uma pasta selecionada com uma janela semelhante a Abrir Imagem;
- Criar *script* (*Create Script*): cria um *script* que executa esta edição sobre uma imagem, o protótipo abre uma janela solicitando o nome e o local para salvar.

Para escolher um arquivo ou pasta para abrir ou salvar é apresentada para o usuário a janela de escolher como mostra a Figura 4.12, onde são apresentadas as pastas para selecionar um arquivo ou pasta.

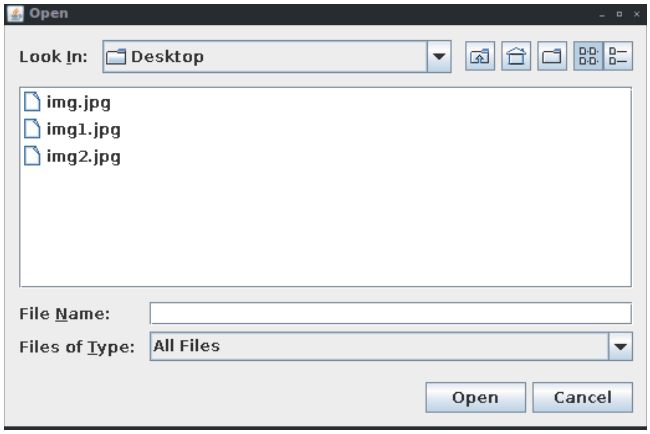


Figura 4.12 – Exemplo de janela apresentada ao usuário para escolher uma imagem para ser editada.

Na janela principal estão as operações selecionadas para o usuário editar a imagem, posicionadas em uma barra de ferramentas. A primeira é a função que abre uma imagem para ser editada, o que também pode ser executado através do menu principal selecionando a opção de abrir uma imagem (*open image*). A segunda e a terceira funções (*flip* e *flop*) são semelhantes: elas produzem uma inversão na vertical e na horizontal, respectivamente. As próximas 3 operações influenciam na cor da imagem, a primeira (*grayscale*) converte a imagem para tons de cinza, em seguida a função que converte para preto e branco (*black and white*) e por fim o filtro de sépia (*sepia tone*) que converte as cores da imagem para parecer uma foto antiga. A operação de sépia necessita de um parâmetro que indique a densidade desejada, em porcentagem, e o protótipo exibe uma janela solicitando ao usuário que insira uma porcentagem como apresentado na Figura 4.13.

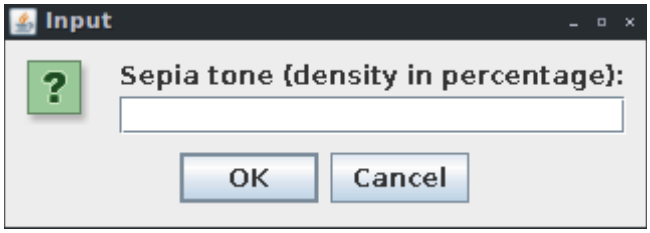


Figura 4.13 – Diálogo para solicitar porcentagem da densidade para o filtro de sépia tone.

Outra funcionalidade implementada no protótipo é a operação de recorte. Para realizar um recorte é necessário que o usuário defina uma área da imagem com o mouse e clique sobre o

botão de recorte (*crop*). Neste caso o protótipo deve verificar se uma região foi selecionada, caso contrário o usuário deve ser informado de que deve selecionar uma região. O protótipo verifica se a região está sobre a imagem, se estiver é executado o recorte, caso contrário é solicitado ao usuário que selecione uma região sobre a imagem.

A operação de rotação serve para rotacionar a imagem, onde o usuário informa o ângulo de rotação. É permitido rotacionar tanto no sentido horário (ângulo positivo) quanto no anti-horário (ângulo negativo). Por fim, tem-se a operação para redimensionar a imagem, onde o usuário informa a escala em porcentagem.

Estas operações foram selecionadas com o intuito de ter uma variedade distinta de operações. Alguns métodos modificam a imagem, suas cores ou dimensões. Outros possuem parâmetros, aumentando a diversidade de interação.

A funcionalidade *Drag and Drop* foi implementada na segunda janela, para que o usuário possa posicionar as entradas como desejar, com o intuito de compará-las. Esta funcionalidade pode ser utilizada em conjunto com as operações de *zoom in* e *zoom out* disponíveis no menu desta janela. O *zoom* permite ao usuário ampliar o resultado das transformações e compará-las na janela do histórico, sem a necessidade de abrir uma nova janela. Com a opção de *Drag and Drop* é possível comparar imagens que estejam distantes umas das outras, trazendo-as para perto.

Outras funcionalidades foram adicionadas para facilitar o reuso ou movimentação da sub-árvore do fluxo de edição. A primeira é a operação de cópia, que consiste em copiar toda uma sub-árvore para outra entrada. Para executar esta funcionalidade é necessário mover um nodo qualquer para cima de outro, ativando um menu com opções, em seguida escolhe-se a opção *copy*. O nodo arrastado é considerado a origem e o nodo no qual o usuário solta é o nodo destino. Todas as transformações aplicadas a partir do nodo origem são aplicadas sobre o nodo destino. Ou seja, quando o usuário arrastar um nodo origem sobre um nodo destino, todas as transformações aplicadas sobre a origem, inclusive esta transformação, serão aplicadas sobre a imagem resultante do nodo destino. Por exemplo, na Figura 4.14 o usuário aplicou uma transformação de *flop* em seguida uma conversão para tons de cinza, retornou para o resultado anterior e executou um filtro de sépia com densidade de 80% gerando o fluxo da Figura 4.14a. Porém o usuário percebe que gostaria de recortar a imagem antes de aplicar estas transformações, então aplica o recorte na região desejada, em seguida clica sobre a transformação *flop*, arrasta-a até o recorte e solta. O protótipo então solicita ao usuário qual operação deve executar e o usuário seleciona o *copy* como mostra a Figura 4.14b e as transformações são aplicadas sobre o recorte, como mostra a Figura 4.14c.

O protótipo é responsável por antes de executar uma operação de cópia avaliar se esta não provocará uma situação de *loops* infinitos, caso esta seja detectada, o protótipo não irá executar esta operação.

No caso de um fluxo linear utilizando uma pilha para armazenar e representar estas transformações seria impossível executar a operação de cópia realizada no exemplo da Figura 4.14, pois o usuário teria que retroceder desempilhando as operações até a imagem original e aplicar o recorte e todas as transformações que foram desfeitas seriam perdidas e deveriam ser executadas novamente.

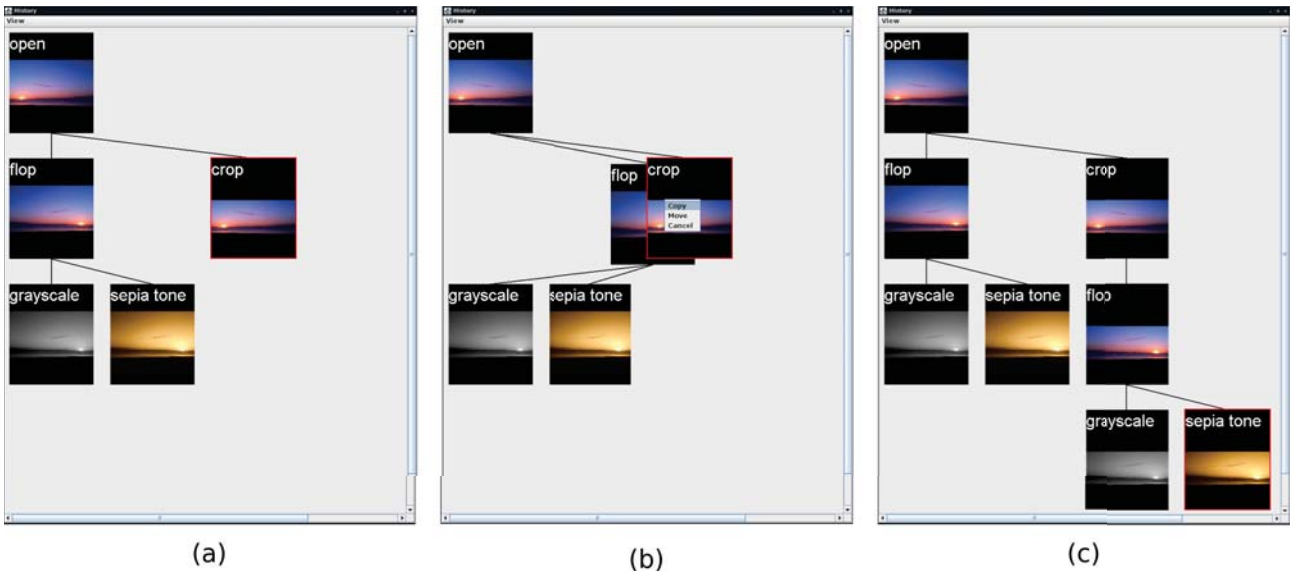


Figura 4.14 – Exemplo da execução da operação de cópia de uma sub-árvore.

Outra operação suportada pelo histórico é a remoção de sub-árvores. O usuário clica sobre o nodo raiz da sub-árvore e seleciona a opção de remover (*delete*), então o protótipo remove esta sub-árvore.

5. AVALIAÇÃO DO MODELO

Com o objetivo de avaliar o modelo proposto foram realizados testes com usuários sobre o protótipo desenvolvido no Capítulo 4. A aplicação dos testes apontou vantagens e desvantagens da representação com uma árvore em comparação com o modelo tradicional que utiliza uma pilha. Neste capítulo será apresentada uma descrição da avaliação.

5.1 Protótipos avaliados

Como visto no Capítulo 4 foi implementado um protótipo capaz de produzir fluxos não lineares, utilizando uma árvore para representar o fluxo gerado durante uma edição onde os nodos da árvore representam as entradas do histórico e permitem funcionalidades como visto na Seção 4.3. Este protótipo foi usado para realizar os testes com usuários e avaliar o modelo proposto, identificando vantagens e desvantagens do mesmo.

Para facilitar a adaptação do usuário com o protótipo foi desenvolvida uma variação utilizando uma pilha de transformações, semelhante às ferramentas atuais com uma janela principal onde a imagem é editada e o histórico na segunda janela utiliza uma pilha. As entradas são semelhantes às do histórico do programa *GIMP* [7], como é mostrado na Figura 5.1. Nesta variação é possível retroceder para um estado com um duplo clique do mouse sobre a entrada desejada. Este histórico é baseado em pilha, logo, quando o usuário retroceder as operações executadas posteriormente serão perdidas como nos programas atuais.

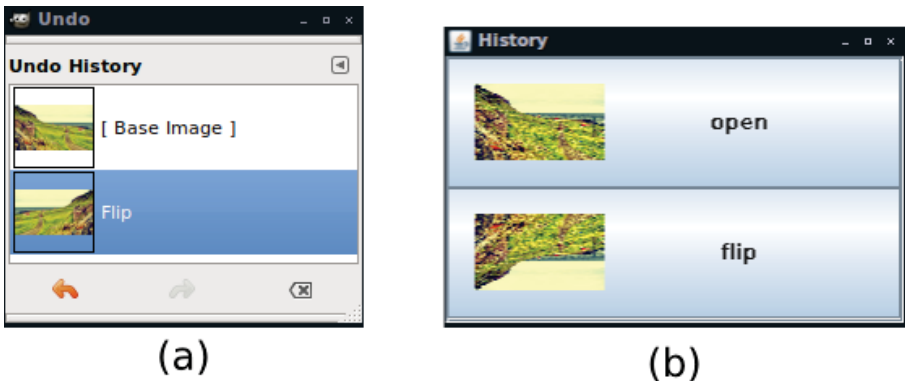


Figura 5.1 – Exemplo de históricos lineares, à direita do programa *GIMP* e à esquerda o *Protótipo A*.

O protótipo baseado em pilha não suporta as operações de copiar (*copy*) e mover (*move*) sub-árvores, pois a pilha oferece apenas um fluxo linear.

No texto foi usada a nomenclatura de *Protótipo A* para a implementação que utiliza uma pilha de transformações e *Protótipo B* para o fluxo de edição de imagens com uma árvore.

5.2 Metodologia de avaliação

Um dos problemas encontrados para a avaliação deste modelo foi a dificuldade em encontrar nos trabalhos relacionados [15, 13, 10] algum tipo de avaliação com usuários. Na época destes trabalhos não era comum fazer testes com os usuários e as ferramentas eram avaliadas levando em consideração o julgamento dos autores. O trabalho de *Meng* [15] por exemplo identifica que a seleção de um estado para retroceder é mais eficiente quando o usuário visualiza os resultados de forma ampla e sugere dois modelos para aprimorar o método para retroceder, no entanto, o estudo indica como trabalhos futuros a avaliação destes modelos com usuários.

No artigo de *Seals* é apresentado um ambiente de programação visual voltada para professores iniciantes na programação. O ambiente foi desenvolvido para criar simulações científicas para auxiliar na aprendizagem da matéria. Para a avaliação deste ambiente foram realizados testes comparando o ambiente desenvolvido com outro semelhante já existente no mercado. O estudo contou com o auxílio de 19 professores que realizaram os testes. A primeira etapa consistia em preencher um questionário pré-teste para identificar o perfil do usuário. Em seguida o usuário recebia instrução para realizar o "*think aloud*" durante a sessão de teste, que consiste em uma técnica para o usuário realizar a tarefa e dizer o que está pensando [20]. Os usuários tinham que criar 3 simulações nos dois programas. Após a realização das tarefas os usuários respondiam um questionário pós-teste avaliando as reações perante os dois ambientes. Por fim o usuário era entrevistado para garantir que o avaliador coletou todas as informações relevantes ao trabalho. O critério principal foi o tempo gasto em média para os dois ambientes, foi destacado que o novo ambiente reduzia o tempo gasto e que 64% dos usuários ficaram mais satisfeitos com o ambiente proposto.

Para avaliação do modelo proposto foram realizados testes com usuários utilizando uma metodologia semelhante ao apresentado no artigo de *Seals* [23]. Porém a avaliação foi realizada utilizando os dois protótipos desenvolvidos, ao contrário de *Seals* que comparou com outro ambiente já existente. Os testes foram realizados da seguinte forma:

- Termo de consentimento: introduz o teste, informa que os dados serão usados para fins de pesquisa e se o usuário aceitar ele assina o termo autorizando o uso das informações adquiridas;
- Pré-teste: para identificar o perfil do usuário;
- Tutorial: para apresentar como o protótipo funciona e suas principais funcionalidades;
- Tarefas: para o usuário interagir e avaliar o protótipo;
- Pós-teste: para o avaliador discutir com o usuário a experiência com o protótipo.

Estas etapas serão discutidas nas subseções a seguir.

5.2.1 Termo de consentimento

Este termo é obrigatório para o uso das informações coletadas com os usuários. O termo serve para informar o usuário sobre o teste de forma clara e objetiva, dissertando sobre os objetivos do teste, como será realizado e os fins do teste. No Apêndice **??** é apresentado o termo de consentimento utilizado neste trabalho.

O termo de consentimento deve ser assinado em caso de aceitação do usuário e deve ser assinado pelo avaliador, possibilitando assim o uso das informações coletadas durante o teste de forma legal.

5.2.2 Pré-teste

O questionário pré-teste tem 6 questões, das quais 3 exploravam a experiência do usuário:

- Com quais ferramentas de edição de imagens você já trabalhou?
  - Adobe Photoshop
  - Corel Paint Shop
  - GIMP
  - Picasa
  - Outros:
- Há quanto tempo utiliza este tipo de ferramenta?
  - Menos de 6 meses
  - Mais de 6 meses
  - Mais de 1 ano
  - Mais de 2 anos
- Com que frequência utiliza este tipo de ferramenta?
  - Uma vez por semana
  - Duas ou mais vezes por semana
  - Uma vez por mês
  - Somente quando descarrego fotos da máquina
  - Somente em ocasiões especiais

Estas questões serviram para avaliar o nível de experiência dos usuários utilizando ferramentas de edição de imagens. Em seguida foram apresentadas questões específicas sobre as funcionalidades discutidas neste trabalho. A primeira verifica se o usuário já utilizou algum mecanismo para automatizar a edição de um conjunto de imagens:

- Você já automatizou a edição de um conjunto de imagens (por exemplo: aplicar a remoção de olhos vermelhos ou a conversão para preto e branco para todas as imagens de uma pasta?) Em caso afirmativo, foi fácil?

Em seguida foi questionado se o usuário já teve que gerar duas imagens a partir de uma:

- Você já precisou gerar duas imagens a partir de uma única imagem? Se sim, encontrou alguma dificuldade? Qual ou quais?

Por fim foi questionado se o usuário já teve que comparar ou visualizar resultados intermediários:

- Você já precisou comparar resultados intermediários de uma edição (ex.: comparar fundos diferentes para uma imagem)? Se sim, encontrou alguma dificuldade? Qual ou quais?

Estas questões auxiliaram para identificar o perfil do usuário e a experiência em tarefas menos triviais como a automatização de uma edição sobre um conjunto de imagens, Os resultados deste questionário serão discutidos na Seção 5.4.

5.2.3 Tutorial

O tutorial foi realizado após o pré-teste. Primeiro era apresentado o *Protótipo A* que representa o fluxo linear utilizando uma pilha de transformações. No tutorial é apresentado como são aplicadas as transformações sobre uma imagem. Foi explicado como o *Protótipo A* representa o histórico, que é possível retroceder clicando sobre a entrada desejada e quando uma nova transformação for aplicada após retroceder as posteriores serão perdidas. Por fim foi mostrado como se salva uma edição, como salva uma imagem e como se aplicam as transformações a um conjunto de imagens.

Em seguida foi apresentado o *Protótipo B* mostrando como funciona a edição, como retroceder, como funciona a função de *zoom*, a cópia de um ramo da árvore e como mover um ramo. Por fim eram recapituladas as funções básicas de salvar.

O tutorial serviu para dar uma noção de como os protótipos funcionam, com o objetivo de não deixar o usuário desconfortável sem ter ideia de como iniciar as tarefas. Outro motivo pelo qual foi realizado um tutorial antes das tarefas é que o objetivo dos testes não era identificar problemas com a interface, e sim, apresentar e avaliar o modelo de visualização do fluxo de edição de imagens utilizando uma árvore.

5.2.4 Tarefas

O objetivo das tarefas é identificar se o modelo apresentado é um avanço sobre o modelo atual utilizado para representar o fluxo de edição. Todas as tarefas devem ser possíveis de executar nos dois modelos.

O modelo proposto possui funcionalidades adicionais como por exemplo a função de copiar e de mover, que podem ser utilizadas durante a execução das tarefas, mas deve ser possível executar a tarefa sem a necessidade destas.

Para avaliar o modelo proposto foram definidas 3 tarefas que foram executadas no *Protótipo A* e em seguida utilizando o *Protótipo B*.

A primeira tarefa tem o objetivo de adaptar o usuário com o protótipo, com a forma como se abre uma imagem para editar, como se aplicam transformações, como inserir parâmetros nas funções e como se salva uma imagem. A descrição da tarefa é apresenta a seguir:

Tarefa 1:

> Para experimentar a ferramenta, primeiro converta a imagem "ferrari.png", localizada na área de trabalho, para tons de cinza (Grayscale), depois aplique uma rotação (rotate) de 15 graus na imagem e salve-a na área de trabalho com o nome de "nova_ferrari.png".

A segunda tarefa foi criada para testar o fluxo não-linear: o usuário deve produzir duas imagens a partir de uma como é descrito a seguir:

Tarefa 2:

> Para usar as transformações, abra a imagem "dois_carros.png" e recorte (crop) apenas o carro da direita. Em seguida aplique no recorte uma inversão na vertical (flip). Após execute a transformação de sépia tone (sepia tone) de 70% e salve o resultado com a nomenclatura "direita_70.png". Retroceda para o resultado da inversão horizontal e execute novamente a transformação de sépia tone agora com 80% e salve com a nomenclatura "direita_80.png". Após aplicar estas transformações no carro da direita, replique-as no segundo carrinho salvando as duas imagens como "esquerda_70.png" e "esquerda_80.png".

Nesta tarefa o modelo linear é capaz de gerar os resultados solicitados, com um trabalho a mais por não ter a opção de copiar um ramo da árvore. Utilizando o *Protótipo B* é possível identificar os fluxos não lineares gerados durante a edição. No *Protótipo A* o usuário gera a imagem "*direita_70.png*" e deve retroceder um estado para gerar "*direita_80.png*" e quando a gera o usuário verifica que a imagem anterior é perdida no histórico, o mesmo acontece quando retrocede até a imagem original para gerar as imagens "*esquerda_70.png*" e "*esquerda_80.png*".

A tarefa 3 foi criada para apresentar a funcionalidade de aplicar uma edição sobre um conjunto de imagens. A tarefa foi descrita da seguinte forma:

Tarefa 3:

> Aprofundando as transformações, redimensione a imagem "ferrari.png" em 70% do seu tamanho original e converta-a para preto e branco. Feito isso, aplique estas configurações sobre a pasta "todas" da área de trabalho.

Estas foram as 3 tarefas realizadas para avaliar o modelo proposto. Na próxima seção serão abordadas as sessões de teste.

5.2.5 Pós-teste

Após a aplicação das tarefas foram realizadas perguntas para a avaliação do modelo proposto. As primeiras perguntas eram destinadas a identificar dificuldades na realização das tarefas e se o usuário havia realizado todas as tarefas ou se teve alguma dificuldade. Em seguida eram feitas perguntas específicas sobre o modelo proposto, como:

- O que você achou da função de copiar e mover?
- Você acha possível comparar resultados com os ícones criados para cada entrada do histórico, considerando o uso de *zoom*?
- Você acha útil a funcionalidade de aplicar uma edição sobre um conjunto de imagens?

Em seguida era questionado qual o modelo o usuário gostou mais e pedia-se uma justificativa. Por fim, o usuário era questionado se havia alguma sugestão para a melhoria do modelo.

5.3 Sessões de teste

As sessões de teste foram realizadas com 9 estudantes da Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul, incluindo alunos da graduação e da pós-graduação. Os testes foram realizados individualmente, apenas o usuário e o avaliador em uma sala utilizando um notebook e um mouse. Os testes levaram em média entre 20 e 30 minutos.

O avaliador entregava o termo de consentimento para o usuário e explicava que o trabalho seria realizado para avaliar o protótipo criado, sem fins lucrativos e mantendo a privacidade dos dados coletados. Se o usuário concordasse era solicitado que assinasse o termo. Em seguida era realizado o pré-teste de forma a identificar o perfil do usuário.

Optou-se por realizar as 3 tarefas sobre o *Protótipo A* em seguida as 3 sobre o *Protótipo B*. O avaliador entregava uma tarefa de cada vez e esperava o usuário indicar a conclusão ou a desistência da tarefa. O avaliador interagia quando solicitado pelo usuário para tirar alguma dúvida.

Após a conclusão das tarefas com os dois protótipos era realizado o pós-teste para identificar a reação dos usuários ao modelo proposto. Na próxima seção são discutidos os resultados dos testes.

5.4 Resultados obtidos

Esperava-se ter dois grupos de usuários, o primeiro com usuários experientes e o segundo com mais inexperientes. Porém os resultados indicaram que todos os usuários se autodenominaram experientes, por este motivo foi limitado em apenas um grupo.

Uma das características identificadas nos usuários é que 6 usuários não utilizaram a ferramenta de cópia para a tarefa 2, que consistia em aplicar as mesmas transformações em outra região da imagem. Um dos 6 usuários afirmou que a edição era curta (apenas 3 transformações) então não achou necessário o uso da funcionalidade. No entanto se fosse algo mais complexo com mais transformações o usuário acredita que iria utilizar esta operação. Aqueles que utilizaram afirmaram que é útil e melhora o modelo atual. Um dos usuários disse:

"Fiz apenas uma vez e depois copiei e ele fez tudo para mim"

Em relação à visualização, todos os usuários preferiram a representação em árvore. Dentre os 9, 4 afirmaram que fica mais fácil de ver utilizando uma árvore. Outros afirmaram que o uso de uma árvore é bom por permitir caminhos alternativos, ou seja, sem a perda de uma edição como ocorria no modelo tradicional. Um dos usuários afirmou que o modelo permite ramificações e que isto dá liberdade ao usuário de tentar sem a preocupação de não conseguir retroceder.

Quando questionados sobre o uso do *zoom* e se era possível comparar resultados, todos afirmaram que sim, dependendo do nível de detalhes e do nível máximo de *zoom*. Um dos usuários indicou que todas as transformações seriam possíveis de serem comparadas exceto a operação de redimensionar (*resize*) pois as entradas possuem um tamanho fixo.

Apenas 2 usuários já haviam utilizado a funcionalidade de aplicar uma edição sobre um conjunto de imagens. Os demais, após conhecerem a acreditavam que seria muito útil. Um dos usuários disse que seria útil para aplicar uma edição para a remoção de olhos vermelhos em um conjunto de imagens.

5.5 Análise dos resultados

Analisando os resultados foi identificado que todos os usuários que avaliaram o protótipo aceitaram a nova forma de visualizar o fluxo de edição de imagens, principalmente com o incremento de funcionalidades como copiar e mover, o que era esperado pois o novo modelo aumenta o atual sem perder características do modelo já existente.

Outro fator identificado é que o uso de uma árvore permite aos usuários explorarem mais sem a preocupação de perder algum resultado, já que todas as alterações são mantidas na árvore. Em relação à automatização de uma edição, muitos não conheciam esta funcionalidade, mas após conhecerem identificaram vários usos para a mesma.

6. CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho sugere o uso de uma árvore para representar o fluxo de edição de imagens, tornando o fluxo em não-linear. Foram apresentadas as deficiências do modelo linear, apresentando exemplos e em seguida foi introduzido o modelo proposto utilizando uma árvore, descrevendo as vantagens do uso da mesma. Por fim foram abordadas as avaliações realizadas sobre o modelo, descrevendo os testes com os usuários. Neste capítulo serão discutidas as contribuições deste trabalho, além de possíveis trabalhos futuros.

6.1 Modelo baseado em árvore

Como visto no Capítulo 1 o modelo baseado em pilha tem algumas inconveniências para o usuário, como por exemplo a perda de transformações aplicadas quando o usuário retrocede e aplica uma nova transformação. Isto motivou o estudo de uma alternativa para representar e armazenar este histórico. Para solucionar alguns dos problemas encontrados, foi proposta a estrutura de árvore. Com uma árvore é possível criar ramificações, que permitem ao usuário desfazer e refazer quaisquer transformações aplicadas anteriormente, sem a perda de transformações.

O uso de uma árvore possibilitou a implementação das operações de copiar e mover. Como visto na avaliação do modelo elas auxiliam no reuso de transformações, diminuindo o trabalho do usuário.

Outro vantagem do modelo proposto é a persistência do histórico, pois normalmente quando se edita uma imagem todas as transformações aplicadas durante a edição são mantidas no histórico, no entanto quando este é fechado e reaberto as entradas do histórico são perdidas. Neste modelo o histórico é salvo em um arquivo XML e guarda a árvore com todos os seus nodos, o que facilita quando o usuário quiser retroceder uma edição posteriormente. Isto auxilia o usuário a ver as transformações que já foram realizadas para chegar a um resultado e possibilita desfazer e refazer.

O uso de *zoom* sobre a área onde é apresentada a árvore auxilia na comparação de resultados, no entanto, podem haver transformações que possam ser imperceptíveis, como a operação de redimensionar (*resize*).

O modelo apresentado possibilita a edição de um conjunto de imagens, como por exemplo, a conversão de todas as imagens de uma pasta para tons de cinza, diminuindo assim o trabalho do usuário. Isto também pode ser realizado utilizando um modelo linear, no entanto, só é possível gerar um resultado final. No modelo baseado em árvore pode-se gerar vários resultados, por exemplo a execução da edição apresentada na Figura 6.1 onde existem 3 resultados finais.

Com estas vantagens é possível identificar que o uso de uma árvore auxilia na visualização e na edição de uma imagem. A proposta incrementa o modelo atual, com o uso de uma árvore foi possível implementar funcionalidades que facilitam a edição para os usuários.

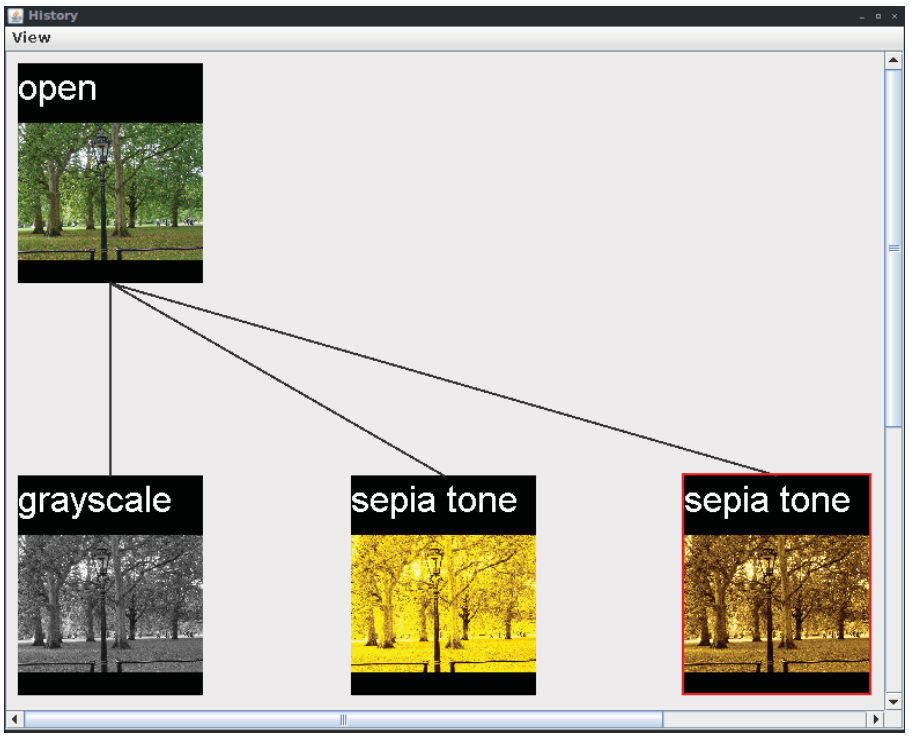


Figura 6.1 – Exemplo que gera 3 saídas diferentes.

6.2 Trabalhos Futuros

Nesta seção são discutidas funcionalidades e técnicas que podem ser implementadas para aprimorar o protótipo desenvolvido, além descrever problemas identificados e propor possíveis soluções.

O protótipo foi desenvolvido para identificar a eficácia do uso de uma árvore ao invés de uma pilha para representar o histórico de desfazer. Após a avaliação pode-se identificar que os usuários foram favoráveis ao uso de uma árvore. Como trabalhos futuros pode-se estudar o uso de um grafo ao invés de uma árvore, para realizar operações como a geração de uma imagem através da união de outras duas como mostra a Figura 3.2.

Um dos problemas do uso de uma árvore ou um grafo é seu tamanho, pois se a edição for extensa, o número de nodos pode ser significativamente grande. Neste caso pode-se implementar técnicas para reduzir o número de nodos visíveis, como por exemplo a técnica *Fisheye* [21], que dá um *zoom* sobre o nodo corrente e os demais ficam menores.

Em relação à funcionalidade de executar uma edição sobre uma pasta contendo imagens, foi identificado que seria possível criar uma interface para selecionar os resultados que devem ser gerados, pois atualmente a ferramenta gera todos os passos executados durante a edição. No entanto o usuário pode optar por gerar apenas o resultado final, ou então apenas alguns resultados intermediários. Uma alternativa seria uma interface que mostrasse todos os passos da edição e o usuário selecionasse as que desejam.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Archer, Jr., J. E.; Conway, R. W. "Cope: A cooperative programming environment", Relatório Técnico, Ithaca, NY, USA, 1981.

[2] Berlage, T. "A selective undo mechanism for graphical user interfaces based on command objects", *ACM Trans. Comput.-Hum. Interact.*, vol. 1, September 1994, pp. 269–294.

[3] Burnett, M. M. "Visual Programming". Wiley Publishing, 1999, vol. 3, pp. 275 – 283.

[4] Cook, W. R. "Applescript". In: HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages, 2007, pp. 1–21.

[5] Corel. ""PaintShop Photo Pro X3 - Edição de fotos"". Capturado em: http://www.corel.com/servlet/Satellite/br/pt/Product/1184951547051, Outubro de 2010.

[6] Ellson, J.; Gansner, E.; Koutsofios, E.; North, S.; Woodhull, G. "Graphviz and dynagraph – static and dynamic graph drawing tools". In: *Graph Drawing Software*, Junger, M.; Mutzel, P. (Editores), Springer-Verlag, 2003, pp. 127–148.

[7] GIMP. ""GIMP - The GNU Image Manipulation Program"". Capturado em: http://www.gimp.org, Outubro de 2010.

[8] Google. ""Picasa 3: Download gratuito do Google"". Capturado em: http://picasa.google.com/, Outubro de 2010.

[9] Heer, J.; Mackinlay, J.; Stolte, C.; Agrawala, M. "Graphical histories for visualization: Supporting analysis, communication, and evaluation", *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, vol. 14, 2008, pp. 1189–1196.

[10] Hirakawa, M.; Ichikawa, T. "Advances in visual programming", 1992, pp. 538 –543.

[11] ImageMagick. ""ImageMagick: Convert, Edit, and Compose Images"". Capturado em: http://www.imagemagick.org/script/index.php/, Outubro de 2010.

[12] Kaasten, S.; Greenberg, S. "Integrating back, history and bookmarks in web browsers". In: CHI '01 extended abstracts on Human factors in computing systems, 2001, pp. 379–380.

[13] Koelma, D.; van Balen, R.; Smeulders, A. "Scil-vp: a multi-purpose visual programming environment". In: In Proc.eedings of the 1992 ACM/SIGAPP Symposium on Applied Computing, 1992, pp. 1188–1198.

[14] Kurlander, D.; Feiner, S. "Editable graphical histories". In: Visual Languages, 1988., IEEE Workshop on, 1988, pp. 127 –134.

[15] Meng, C.; Yasue, M.; Imamiya, A.; Mao, X. "Visualizing histories for selective undo and redo", *Asia-Pacific Computer and Human Interaction*, vol. 0, 1998, pp. 459.

[16] Newham, C.; Rosenblatt, B. "Learning the Bash Shell". Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1998, 2nd ed..

[17] Photoshop, A. ""Photo editing software and photo editing programs"". Capturado em: http://www.adobe.com/products/photoshop/family/?promoid=BPDEK, Outubro de 2010.

[18] Poplawski, D. A. "A pedagogically targeted logic design and simulation tool". In: WCAE '07: Proceedings of the 2007 workshop on Computer architecture education, 2007, pp. 1–7.

[19] Robbins, A.; Beebe, N. H. F. "Classic Shell Scripting". O'Reilly Media, Inc., 2005, 1st ed..

[20] Rogers, Y.; Sharp, H.; Preece, J. "Interaction Design: Beyond Human-Computer Interaction". John Wiley and Sons Ltd, 2002.

[21] Sarkar, M.; Brown, M. H. "Graphical fisheye views of graphs". In: Proceedings of the SIGCHI conference on Human factors in computing systems, 1992, pp. 83–91.

[22] Schmucker, K. J. "Rapid prototyping using visual programming tools". In: CHI '96: Conference companion on Human factors in computing systems, 1996, pp. 359–360.

[23] Seals, C. "Visual programming for novice programmer teachers". In: TAPIA '05: Proceedings of the 2005 conference on Diversity in computing, 2005, pp. 26–27.

[24] Sim, Y.-S.; Lim, C.-S.; Moon, Y.-S.; Park, S.-H. "Design and implementation of the visual programming environment for the distributed image processing", 1996, pp. 149 –152 vol.2.

[25] Toriya, H.; Satoh, T.; Ueda, K.; Chiyokura, H. "Undo and redo operations for solid modeling", *Computer Graphics and Applications, IEEE*, vol. 6–4, April 1986, pp. 35 –42.

[26] Vitter, J. "Us&r: A new framework for redoing", *Software, IEEE*, vol. 1–4, October 1984, pp. 39 –52.

[27] Young, M.; Argiro, D.; Kubica, S. "Cantata: visual programming environment for the khoros system", *SIGGRAPH Comput. Graph.*, vol. 29–2, 1995, pp. 22–24.

[28] Zhou, C.; Imamiya, A. "Object-based nonlinear undo model". In: Proceedings of the 21st International Computer Software and Applications Conference, 1997, pp. 50–55.

ANEXO A – Termo de Consentimento

O termo de consentimento caracteriza a concordância do usuário em realizar o teste. Na próxima página é apresentado o termo utilizado para a realização deste trabalho.

Programa de Pós-graduação em Ciência da Computação
Faculdade de Informática/PUCRS
Avenida Ipiranga, 6681 – Prédio 32 - 90619-900 – Porto Alegre – RS

Termo de Consentimento Livre e Esclarecido

O aluno Alexandre Seki, do Programa de Pós-Graduação em Ciência da Computação da Faculdade de Informática da PUCRS, agradece a todos os participantes de testes realizados sob sua responsabilidade, a inestimável contribuição que prestam para o avanço de sua pesquisa.

O objetivo do teste ora em realização é investigar questões relacionadas ao uso de novas funcionalidades em um protótipo para edição de imagens. Para isto, os participantes dos testes são convidados a realizar diferentes tarefas utilizando este protótipo, enquanto são observados pelo aluno. Esta observação será registrada por meio de um software de captura, que armazena tudo o que acontece na tela do computador, e, também, através de gravação em áudio que ocorre durante e após a realização das atividades. Estas informações servirão não apenas para verificar a qualidade do sistema em questão, mas, também, para auxiliar o aluno a compreender melhor os métodos sob estudo.

Lembramos que o objetivo deste estudo **não é** avaliar o participante, **mas, sim,** avaliar o sistema computacional que o participante estará usando durante os testes. O uso que se faz dos registros efetuados durante o teste é **estritamente** limitado a atividades acadêmicas, garantindo-se para tanto que:

1. O anonimato dos participantes será preservado em todo e qualquer documento divulgado em foros científicos (tais como conferências, periódicos, livros e assemelhados) ou pedagógicos (tais como apostilas de cursos, *slides* de apresentações, e assemelhados).
2. Todo participante que se sentir constrangido ou incomodado durante uma situação de teste pode interrompê-lo e estará fazendo um favor ao aluno se registrar por escrito as razões ou sensações que o levaram a esta atitude. O aluno fica obrigada a descartar o teste para fins da avaliação a que se destinaria.
3. Os participantes que forem menores de idade terão, obrigatoriamente, que apresentar o consentimento de seu responsável para participação no estudo, o qual será declarado ciente do estudo a ser realizado através de sua assinatura no presente Termo de Compromisso.
4. Todo participante tem direito de expressar por escrito, na data do teste, qualquer restrição ou condição adicional que lhe pareça aplicar-se aos itens acima enumerados (1, 2 e 3). O aluno se compromete a observá-las com rigor e entende que, na ausência de tal manifestação, o participante concorda que rejam o comportamento ético do aluno somente as condições impressas no presente documento.
5. O aluno tem direito de utilizar os dados dos testes, mantidas as condições acima mencionadas, para quaisquer fins acadêmicos, pedagógicos e/ou de desenvolvimento contemplados por seus membros.

[a ser preenchido pelo observador]
Sistema:_____ Data: __ / __ / ____

Condições especiais (caso não haja condições especiais, escreva "nenhuma"):
_____
_____
_____
_____
_____
_____

☐ continua no verso

Por favor, indique sua posição em relação aos termos acima:

☐ Estou de pleno acordo com os termos acima.

☐ Em anexo registro condições adicionais para este teste.

_____
Assinatura do participante

_____
Assinatura do responsável
(caso o participante seja menor de idade)

_____
Assinatura do observador

Nome do Participante: _____

Nome do Responsável (se o participante for menor de idade): _____

Nome do Observador Responsável pela Aplicação do Teste: _____