UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

MOSAICOS DE IMAGENS PLANARES

EDUARDO COELHO

BLUMENAU
2009

2009/1-05

EDUARDO COELHO

MOSAICOS DE IMAGENS PLANARES

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação – Bacharelado.

Prof. Paulo César Rodacki Gomes, Dr. – Orientador

BLUMENAU
2009

2009/1-05

MOSAICOS DE IMAGENS PLANARES

Por

EDUARDO COELHO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo César Rodacki Gomes, Dr. – Orientador, FURB
Membro:	Prof. Mauro Marcelo Mattos, Dr. – FURB
Membro:	Prof. Dalton Solano dos Reis, Ms. – FURB

BLUMENAU
2009

Dedico este trabalho aos meus pais, Amauri e Sorlene e à minha irmã Fernanda, pelo amor, carinho e apoio em minhas decisões.

"*Live as if you were to die tomorrow. Learn as if you were to live forever.*"

Mahatma Gandhi

AGRADECIMENTOS

À minha família, pelo amor e força que me deram durante a realização deste trabalho.
Aos meus amigos, pela amizade sempre constante e pelos momentos de diversão e apoio.
Ao meu orientador, Paulo César Rodacki Gomes, pela amizade, pelos ensinamentos e pela confiança depositada neste trabalho.
Ao George Ruberti Piva, pelo companheirismo e ajuda fornecida ao longo da graduação.
Ao professor Dalton Solano dos Reis, por ter dedicado parte de seu tempo realizando a revisão do conteúdo deste trabalho.
Ao pesquisador Antonio Escaño Scuri, pela disponibilidade e atenção esclarecendo dúvidas através de *e-mails*.
A todos que contribuiram, indiretamente, para a realização deste trabalho.

RESUMO

Este trabalho apresenta as técnicas envolvidas no desenvolvimento de um software que realiza a construção de mosaicos de imagens planares. A partir de múltiplas imagens de entrada e de um conjunto de pontos de correspondência fornecidos pelo usuário, o software é capaz de construir mosaicos de imagens, permitindo a representação de cenas com maior área de cobertura do que a suportada pelas câmeras fotográficas tradicionais. A construção do mosaico é feita de forma interativa e incremental, permitindo ao usuário a correção de erros de registro de imagens e a visualização dos resultados a medida que o mosaico cresce, não havendo a necessidade de métodos de calibração de câmera. Para a estimação da homografia que representa o alinhamento local entre os pares de imagens, utiliza-se a transformação projetiva planar bidimensional. A distorção das imagens a serem inseridas no mosaico é realizada através do mapeamento inverso em conjunto com a técnica de interpolação bilinear. Foi atribuída uma função de peso aos *pixels* das imagens de entrada de forma a suavizar as emendas existentes na região de transição entre as imagens que compõem o mosaico.

Palavras-chave: Mosaico de imagens. Registro de imagens. Mapeamento projetivo. Interpolação bilinear.

ABSTRACT

This work presents the techniques used in the development of a software that creates planar image mosaics. Given multiple input images and a set of corresponding points chosen by the user, the software is capable to construct image mosaics, allowing the representation of scenes whose field of view are wider than the supported by traditional photographic cameras. The mosaic construction is performed in an interactive and incremental way, so that the user can fix registration errors and visualize the results as the mosaic grows, discarding the need of camera calibration methods. For the estimation of the homography that represents the local alignment among pairs of images, a bidimensional planar projective transformation was applied. The image warping is done through the inverse mapping combined with the bilinear interpolation technique. It was imposed a weighting function for the input images' pixels in order to blend the mosaic's images in a seamless manner.

Key-words: Image mosaicing. Image registration. Projective mapping. Bilinear interpolation.

LISTA DE ILUSTRAÇÕES

Figura 1 – Uma ilustração de registro de imagens 21

Figura 2 – Etapas do processo de registro de imagens: (a) extração de feições; (b) casamento das feições; (c) estimação da função de transformação e (d) mosaico de imagens construído após a transformação e reamostragem das imagens utilizando uma técnica de interpolação 22

Figura 3 – Exemplo de casamento de feições pelo método direto: janela W na imagem de referência submetida à comparação com as demais janelas na imagem de ajuste 25

Figura 4 – Exemplos de funções de mapeamento: (a) similaridade; (b) afim; (c) projetiva e (d) arbitrária 27

Figura 5 – Uma homografia planar H é induzida entre duas imagens de uma cena planar obtidas a partir de diferentes pontos de vista. O ponto de cena X é projetado nos pontos x e x' nas duas imagens, o quais são relacionados por $x' = Hx$ 29

Figura 6 – A medida que a câmera é rotacionada, os pontos de intersecção dos raios com o planos das imagens são relacionados por uma homografia planar. Os pontos de imagem x e x' correspondem ao mesmo ponto de cena X. Os pontos de imagem são relacionados por $x' = Hx$ . . . 29

Figura 7 – Projeção central mapeando um ponto x situado em um plano π para outro ponto x' situado em outro plano π' 30

Figura 8 – Remoção da distorção perspectiva: (a) imagem original com distorção perspectiva – as linhas das janelas convergindo para um ponto finito e (b) visão frontal ortogonal do edifício, gerada através da aplicação da transformação projetiva inversa 31

Figura 9 – (a) imagem de uma cena vista sob projeção perspectiva e (b) imagem da janela retificada após a aplicação de uma transformação projetiva . 31

Figura 10 – Transformações geométricas planares do espaço projetivo 32

Figura 11 – Imagem submetida a transformações afins: o paralelismo, a razão de comprimentos entre linhas paralelas e a razão entre as áreas das imagens são preservados 34

Figura 12 – Projeção perspectiva de uma cena: linhas paralelas convergindo para pontos de fuga. No caso das linhas horizontais e verticais os pontos de fuga estão localizados no infinito 35
Figura 13 – Um quadrilátero submetido a transformações projetivas: (a) quadrilátero original, (b) quadrilátero demonstrando a não preservação da razão de comprimentos entre linhas e (c) quadrilátero enfatizando a presença de pontos de fuga 36
Quadro 1 – Grupo de transformações planares do espaço projetivo 36
Figura 14 – Transformação e reamostragem da imagem de entrada através das funções X e Y 39
Figura 15 – Mapeamento direto causando "buracos" e sobreposição de *pixels* na imagem de saída 40
Figura 16 – Imagem transformada através do mapeamento direto: ocorrência de buracos e sobreposição de *pixels* na imagem de saída 41
Figura 17 – Exemplo de área de representação de um *pixel* nas imagems de entrada e de saída 41
Figura 18 – Aplicação do mapeamento inverso em uma imagem: todos os *pixels* da imagem de saída preenchidos sem ocorrência de buracos e/ou sobreposição 42
Figura 19 – (a) transformação de coordenada inteiras (x, y) para coordenadas não inteiras (x', y') e (b) interpolação bilinear de $I(x + p, y + q)$ com base nos *pixels* vizinhos 43
Figura 20 – (a) imagem original, reamostrada utilizando três técnicas de interpolação diferentes: (b) vizinho mais próximo; (c) bilinear e (d) bicúbica 44
Figura 21 – (a) 8 imagens de uma seqüencia de 25 capturadas por uma câmera fotográfica e (b) todas as imagens combinadas através da técnica de mosaico de imagens 45
Figura 22 – Etapas da construção de mosaicos de imagens 46
Figura 23 – Mosaico de imagens sendo renderizado em uma superfície de projeção planar 47
Figura 24 – Mosaico de imagens sendo renderizado em uma superfície de projeção cilíndrica 48
Figura 25 – (a) mosaico de imagens criado através de uma projeção planar e (b) mosaico de imagens criado através de uma projeção cilíndrica 49

Figura 26 – Mosaico panorâmico 360° composto a partir de 178 imagens e renderizado utilizando uma superfície de projeção cilíndrica 50
Quadro 2 – Algoritmo para a renderização de cada *pixel* pertencente à imagem final de um mosaico 51
Figura 27 – Seis cores primárias sendo mescladas através de diferentes técnicas: (a) imagem do centro mais próximo; (b) média simples e (c) média ponderada (suavização) 52
Figura 28 – Mosaico panorâmico construído a partir de uma sequência de 146 imagens 54
Figura 29 – Mosaico de imagens utilizando: (a) mesclagem através da média ponderada e (b) mesclagem baseada na análise de múltiplas resoluções e com definição da linha de corte automática. Em (c) e (d) são apresentados os detalhes dos mosaicos dentro das janelas marcadas em (a) e (b), respectivamente 55
Figura 30 – Diagrama de casos de uso 58
Figura 31 – Relacionamento entre as camadas de controle, visualização e modelo do domínio 60
Figura 32 – Diagrama de classes da camada de modelo do domínio 61
Quadro 3 – Principais métodos da classe `Mosaic` 62
Figura 33 – Diagrama de classes da camada de visualização 63
Figura 34 – Diagrama de classes da camada de controle 65
Figura 35 – Diagrama de sequência para o caso de uso `Requisitar criação do mosaico` 66
Figura 36 – Diagrama de sequência para o método `createMosaic` 66
Quadro 4 – Método `invert` da classe `Matrix`, parte 1 69
Quadro 5 – Método `invert` da classe `Matrix`, parte 2 70
Quadro 6 – Método `solveLinearSystem` da classe `Matrix` 72
Quadro 7 – Método `estimate` da classe `Homography` 73
Quadro 8 – Método `estimateInverse` da classe `Homography` 75
Quadro 9 – Método `transformPoint` da classe `Homography` 76
Quadro 10 – Método `createMosaic` da classe `Mosaic` 77
Quadro 11 – Pseudo-código para o método `calculateMosaicedImageBoundingBox` da classe `Mosaic` 78

Quadro 12 – Método `warpAndMergeImages` da classe `Mosaic`, parte 1 78
Quadro 13 – Método `warpAndMergeImages` da classe `Mosaic`, parte 2 79
Quadro 14 – Pseudo-código para o método `getAt` da classe `model::IMImage` 81
Quadro 15 – Método `drawCorrespondencePoint` da classe `IMDialog` 82
Figura 37 – Desenho de pares de PCs correspondentes selecionados (em amarelo) e não selecionados (em vermelho, verde e azul) 83
Figura 38 – Interface gráfica do software e seus principais componentes 84
Figura 39 – Abertura de uma imagem de entrada 85
Quadro 16 – Operações da área de edição do mosaico 85
Figura 40 – Imagens de ajuste e de referência definidas e PCs correspondentes posicionados 86
Figura 41 – Processo de criação do mosaico de imagens 86
Figura 42 – Mosaico criado apresentado na área de edição 87
Figura 43 – Imagem do mosaico final criado selecionada 87
Figura 44 – (a) mosaico de imagens criado utilizando a técnica de interpolação bilinear e (b) mosaico de imagens criado sem utilizar nenhuma técnica de interpolação, é possível perceber o *aliasing* linhas diagonais 89
Figura 45 – Mosaico de imagens representando uma cena cujo ângulo de abertura é superior a 90°. A má escolha da imagem de referência favoreceu o aparecimento de distorções na extremidade esquerda do mosaico . . . 90
Figura 46 – (a) Mosaico de imagens utilizando a média ponderada para suavização da região de transição entre as imagens e (b) mosaico de imagens sem suavização da região de transição entre as imagens, as emendas são visíveis 90
Figura 47 – (a) mosaico com mesclagem das imagens ineficiente; (b) objetos em movimento causando o efeito "fantasma" e (c) diferença de luminosidade acentuada entre imagens de entrada prejudicou a qualidade do resultado final 91
Figura 48 – Mosaico de imagens representando uma cena planar, as imagens são obtidas com movimentações de câmera arbitrárias. É possível perceber a emenda entre as imagens, visto que a técnica de mesclagem foi suprimida 92

Figura 49 – Mosaico de imagens composto por cinco fotografias obtidas por uma câmera que é rotacionada em torno de seu centro óptico 93
Quadro 17 – Características dos mosaicos de imagens criados através do software desenvolvido 94

LISTA DE SIGLAS

2D – Duas dimensões

3D – Três dimensões

PCs – Pontos de Controle

DOFs – *Degrees Of Freedom*

RF – Requisito Funcional

RNF – Requisito Não-Funcional

MVC – *Model-View-Controller*

RGB – *Red-Green-Blue*

RHS – *Right-Hand Side*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO **17**

1.1 OBJETIVO 18

1.2 ESTRUTURA DO TRABALHO 19

2 FUNDAMENTAÇÃO TEÓRICA **20**

2.1 REGISTRO DE IMAGENS 20

2.1.1 Modos de aquisição de imagens 21

2.1.2 Etapas do registro de imagens 22

2.2 MAPEAMENTO PROJETIVO 28

2.2.1 Transformações projetivas planares 30

2.2.2 Coordenadas homogêneas 32

2.2.3 Hierarquia de transformações projetivas planares 32

2.2.4 Estimação da função de mapeamento projetivo 37

2.3 TRANSFORMAÇÃO E REAMOSTRAGEM DE IMAGENS 39

2.3.1 Mapeamento direto 39

2.3.2 Mapeamento inverso 41

2.3.3 Função de interpolação 43

2.4 MOSAICO DE IMAGENS 44

2.4.1 Superfícies de projeção 47

2.4.2 Mesclagem 51

2.5 TRABALHOS CORRELATOS 52

2.5.1 An image mosaicing module for wide area surveillance 52

2.5.2 Mosaico de imagens baseado em múltiplas resoluções 54

2.6 CONSIDERAÇÕES FINAIS 55

3 DESENVOLVIMENTO DO SOFTWARE **57**

3.1 REQUISITOS PRINCIPAIS 57

3.2 ESPECIFICAÇÃO 57
3.2.1 Visão externa do sistema 58
3.2.2 Visão estrutural estática do sistema 59
3.2.3 Visão estrutural dinâmica do sistema 65
3.3 IMPLEMENTAÇÃO 66
3.3.1 Ferramentas e bibliotecas utilizadas 66
3.3.2 Técnicas e código fonte implementados 67
3.4 OPERACIONALIDADE 83
3.4.1 Interface gráfica do software e principais componentes 84
3.4.2 Criação de um mosaico de imagens 85
3.5 RESULTADOS E DISCUSSÃO 88

4 CONCLUSÕES **95**
4.1 TRABALHOS FUTUROS 96

REFERÊNCIAS BIBLIOGRÁFICAS **98**

1 INTRODUÇÃO

A necessidade de combinar várias imagens de uma cena em uma única imagem mais abrangente existe desde o início da fotografia, uma vez que o campo de visão de uma câmera é menor que o campo de visão humano. Enquanto tesouras e cola eram as ferramentas utilizadas na composição de mosaicos de imagens geradas com fotografia analógica, métodos mais sofisticados foram desenvolvidos com o surgimento da imagem digital (PELEG; HERMAN, 1997, p. 261).

Paralelamente à evolução dos computadores, a fotografia digital tem evoluído e tornado-se mais acessível, possibilitando a coleta de fotografias com maior rapidez, facilidade e a custos reduzidos. Tal fato acabou propiciando a utilização de imagens fotográficas para a resolução de muitos problemas do cotidiano, nas mais diversas áreas do conhecimento e com área de abrangência ilimitada, visto que a obtenção de imagens pode ser feita utilizando-se de qualquer tecnologia óptica (desde um microscópio, passando por câmeras digitais comuns até imagens de satélites).

Em muitos casos, a área de cobertura de uma única imagem não é suficiente para tratar o problema em questão, fazendo-se necessário dispor de alguma técnica capaz de utilizar as informações contidas em múltiplas imagens de entrada para produzir outra imagem mais ampla e de melhor qualidade, de forma a melhor representar uma cena de interesse. A este processo de junção de múltiplas imagens dá-se o nome de mosaico de imagens.

Segundo Ye (2005, p. 15), mosaico de imagens é o processo de efetivamente aumentar o campo de visão de uma câmera através da combinação de várias visões parciais de uma cena em uma única visão mais abrangente. Capel (2001, p. 2) afirma que mosaico de imagens consiste no alinhamento de múltiplas imagens em uma composição mais ampla que representa porções de uma cena 3D.

A criação de mosaicos de imagens amplos e de alta resolução é uma área de pesquisa ativa nos campos de fotogrametria, processamento de imagens, visão computacional e computação gráfica. Em visão computacional, mosaicos de imagens fazem parte de uma vertente de estudo recente, denominada de estudo das representações das cenas visuais. Em computação gráfica, mosaicos de imagens têm um papel importante na área de renderização baseada em imagens, cujo objetivo é a renderização rápida de visões fotorealistas a partir de coleções de imagens reais (ou pré-renderizadas) (SHUM; SZELISKI, 1997, p. 1).

Mosaicos de imagens têm sido utilizados em diversas aplicações, tais como criação de mosaicos de alta resolução para mapas digitais e fotos de satélites, compressão, sumarização e estabilização de vídeo, análise de imagens médicas, sensoriamento remoto, aplicações nas áreas de fotogrametria e fotografia, chegando a atender as necessidades de usuários finais (SHUM; SZELISKI, 1997, p. 1).

Além do mais, a criação de mosaicos de imagens tem causado impacto no mercado da fotografia digital, com a emergência de produtos que permitem que grandes quantidades de imagens ou até mesmo *streams* de vídeos obtidos por câmeras convencionais sejam utilizados para a geração de mosaicos, tais como o sistema Videobrush (PELEG; HERMAN, 1997). O mosaico de imagens também forma a base tecnológica de sistemas de realidade virtual, tais como Quicktime VR (CHEN, 1995).

Através de técnicas de registro e de mesclagem de imagens, tais tarefas tornaram-se viáveis de serem realizadas sem a necessidade da utilização de *hardwares* exóticos e de alto custo, tais como câmeras panorâmicas ou com lentes de ângulo de visão amplo (*fisheye*), apresentando resultados mais eficientes e com menores distorções.

Com base nesse contexto, este trabalho visa investigar como um conjunto de imagens – as quais representam visões parciais de uma cena e que se sobrepõem – e as informações nelas contidas podem ser combinadas para produzir uma única imagem mais ampla e de melhor qualidade, denominada mosaico de imagens.

Para tal, foi desenvolvido um software que permite a criação de mosaicos de imagens panorâmicos planares de forma interativa e incremental, permitindo ao usuário a correção e a visualização dos resultados à medida que o mosaico cresce.

Um ponto chave neste trabalho é que assume-se que são utilizadas câmeras fotográficas tradicionais não calibradas. Nenhum conhecimento *a priori* dos parâmetros de câmera, características fotométricas ou ópticas são requeridos. Os métodos estudados e os resultados obtidos são apresentados através imagens do mundo real, de forma a validar a eficácia da solução.

1.1 OBJETIVO

O objetivo deste trabalho é desenvolver um software que permita a construção de mosaicos de imagens, de forma a aumentar o campo de visão de uma câmera fotográfica através da combinação de várias visões parciais de uma cena em uma única visão mais abrangente.

Os objetivos específicos do trabalho são:

a) permitir a entrada das imagens obtidas por câmeras fotográficas convencionais

que representam a cena a ser reconstruída;

b) calcular a homografia que relaciona cada par de imagens através de quatro ou mais pontos de correspondência fornecidos pelo usuário;

c) combinar as imagens de entrada de forma a representar a cena em questão em uma única imagem, minimizando os efeitos de transição entre as imagens;

d) apresentar o mosaico de imagens construído, permitindo seu crescimento de forma incremental.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em quatro capítulos. O capítulo 2 apresenta o embasamento teórico necessário para construção de mosaicos de imagens. Primeiramente são explicados os conceitos fundamentais e as etapas envolvidas no processo de registro de imagens. Em seguida é feita uma explanação sobre mapeamento projetivo e técnicas de transformação e reamostragem de imagens. Na sequência, são descritas as etapas e as diferentes abordagens existentes para a construção de mosaicos de imagens. Por fim, são apresentados e discutidos alguns trabalhos correlatos ao presente trabalho.

O capítulo 3 descreve o desenvolvimento do software, apresentando os seus principais requisitos, sua especificação – através de diagramas da *Unified Modeling Language* (UML) – e detalhes referentes à sua implementação. Também são comentadas questões relacionadas à operacionalidade do software e realizada uma discussão sobre os resultados obtidos.

Finalmente, o capítulo 4 apresenta as conclusões deste trabalho e perspectivas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os tópicos relevantes ao processo de construção de mosaicos de imagens. A seção 2.1 aborda os conceitos básicos de registro de imagens e as etapas envolvidas ao longo do processo. A seção 2.2 explica o processo de mapeamento projetivo, demonstrando o método de estimação da homografia que relaciona pares de imagens que compõem um mosaico. Na seção 2.3 são discutidas as técnicas existentes para transformação e reamostragem de imagens. A seção 2.4 apresenta o conceito de mosaico de imagens, os tipos mais comuns existentes, as etapas envolvidas no processo de construção e as abordagens mais comuns para cada uma destas etapas. Por fim, na seção 2.5 são apresentados alguns trabalhos correlatos ao trabalho proposto.

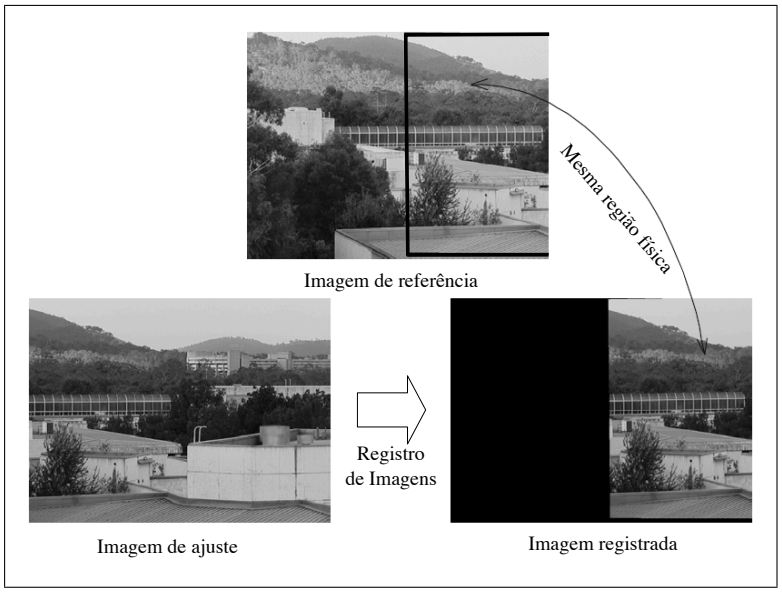
2.1 REGISTRO DE IMAGENS

O registro de imagens é uma etapa essencial nas aplicações de fusão[1] de imagens, as quais a informação final obtida é resultado da combinação de várias imagens de entrada (CAPEL, 2001, p. 1). É uma técnica utilizada para sobrepor duas ou mais imagens de uma mesma cena que podem ser obtidas em diferentes tempos, sob diferentes pontos de vista e/ou por diferentes sensores (ZITOVA; FLUSSER, 2003, p. 977).

Ye (2005, p. 6) define registro de imagens como o processo de combinar espacialmente duas imagens, denominadas imagem de referência e imagem de ajuste, de forma que pontos de coordenadas correspondentes nas duas imagens correspondam à mesma região física da cena sendo exibida (fig. 1).

A evolução dos dispositivos de aquisição de imagens propiciou um aumento tanto no número, quanto na diversidade das imagens obtidas, o que incentivou a utilização das mesmas para a resolução de problemas, nas mais variadas áreas do conhecimento. Técnicas de registro de imagens têm sido amplamente aplicadas nas áreas de visão computacional, sensoriamento remoto, análise de imagens médicas, processamento de vídeo, entre outras. Uma compreensiva pesquisa sobre possíveis áreas de aplicação das técnicas de registro de imagens pode ser encontrada em Brown (1992).

[1] Mosaico de imagens e super-resolução são exemplos de aplicações de fusão de imagens.



Fonte: adaptado de Ye (2005, p. 9).

Figura 1 – Uma ilustração de registro de imagens

2.1.1 Modos de aquisição de imagens

Embora a aplicações que utilizam das técnicas de registro de imagens pertençam as mais variadas áreas do conhecimento, Ye (2005, p. 8) dividiu estas em três grupos principais tomando como base o modo de aquisição de imagem, podendo ser: em diferentes tempos, sob diferentes pontos de vista ou a partir de diferentes sensores[2].

Imagens de uma cena obtidas em diferentes tempos podem ser utilizadas para a verificação e a avaliação de possíveis mudanças ocorridas nesta cena ao longo das consecutivas aquisições. Exemplos de aplicações para este grupo são: monitoramento e planejamento de terrenos (sensoriamento remoto); detecção de movimento e detecção de eventos em sistemas de segurança (visão computacional); monitoramento da evolução de tumores (análise de imagens médicas); estimação de movimento e super-resolução (processamento de vídeo).

Imagens de uma cena obtidas sob diferentes pontos de vista permitem que seja construída uma representação 2D mais ampla ou uma representação 3D da cena em questão. Exemplos de aplicações deste grupo são a criação de mosaico de imagens (fotogrametria e sensoriamento remoto) e a obtenção de formas de objetos (visão computacional).

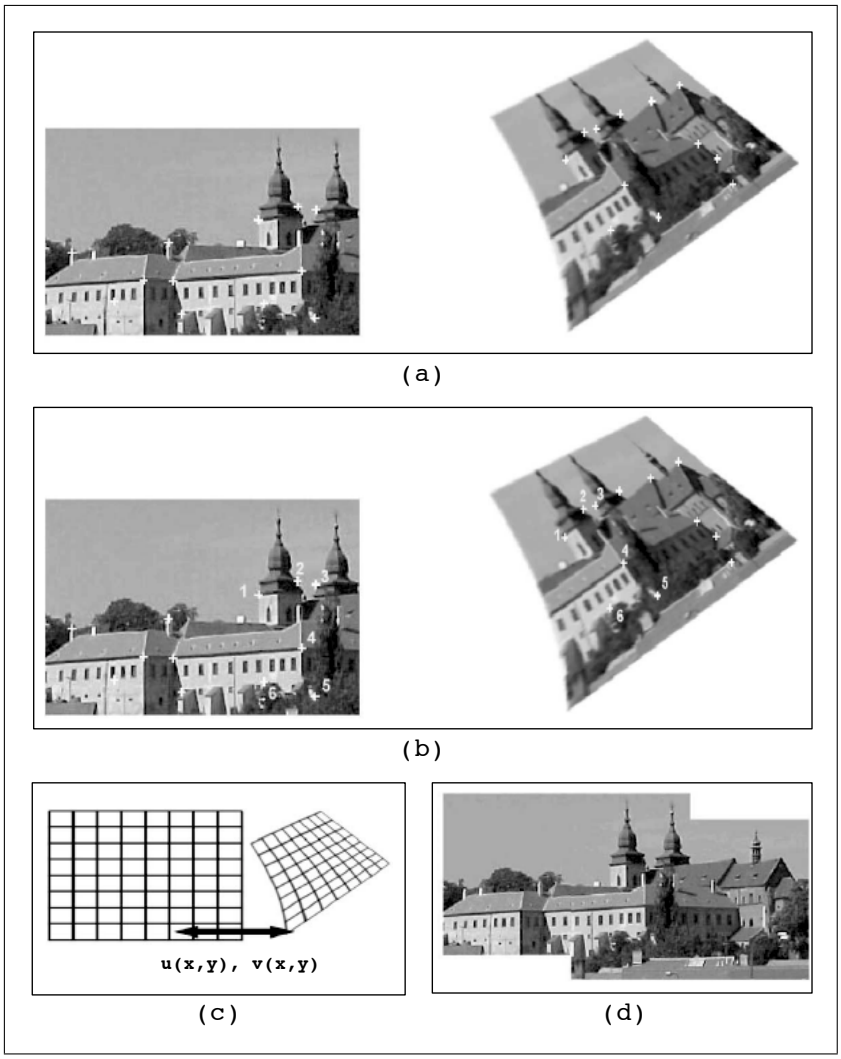
Por fim, aplicações podem utilizar imagens obtidas a partir de diferentes sensores quando visam integrar as informações que estão contidas nestas diferentes fontes de dados. Isto permite que se construa uma representação mais complexa e detalhada da cena em

[2]Os termos encontrados na literatura para as técnicas relacionadas à aquisição de imagens em diferentes tempos, sob diferentes pontos de vista e a partir de diferentes sensores, são, respectivamente, *multitemporal analysis*, *multiview analysis* e *multimodal analysis* (ZITOVA; FLUSSER, 2003, p. 978).

questão. Aplicações classificadas neste grupo são: fusão multisensorial de imagens (sensoriamento remoto e análise de imagens médicas); monitoração de atividades (sistemas de vigilância multisensoriais) e fusão de imagens (navegação veicular).

2.1.2 Etapas do registro de imagens

É notável que, devido a grande diversidade de imagens que podem ser registradas e aos vários tipos de degradações possivelmente nelas presente, é impossível definir um método universal aplicável a todas as tarefas de registro de imagens. No entanto, Zitova e Flusser (2003, p. 978) afirmam que o processo de registro de imagens consiste basicamente em quatro etapas (fig. 2): extração de feições; casamento das feições; estimação da função de transformação e, a última, transformação e reamostragem (*resampling*) das imagens.



Fonte: adaptado de Zitova e Flusser (2003, p. 979).

Figura 2 – Etapas do processo de registro de imagens: (a) extração de feições; (b) casamento das feições; (c) estimação da função de transformação e (d) mosaico de imagens construído após a transformação e reamostragem das imagens utilizando uma técnica de interpolação

Uma explicação mais detalhada de cada uma das etapas envolvidas no registro de imagens é apresentada nas seções seguintes.

2.1.2.1 Extração de feições

A etapa de extração de feições consiste em identificar um conjunto de feições (objetos) relevantes nas duas imagens, tais como contornos de regiões, bordas, cantos, interseções de retas, entre outros. Esta etapa pode ser realizada manualmente ou em alguns casos automaticamente.

Após identificadas, as feições são representadas sob forma de pontos, denominados na literatura por Pontos de Controle (PCs), os quais são utilizados nas etapas posteriores do registro de imagens (ZITOVA; FLUSSER, 2003, p. 978).

A escolha do tipo de feição apropriada a ser extraída é dependente da aplicação. De modo geral, devem ser objetos bem destacados, fáceis de serem encontrados e que estejam espalhados ao longo das imagens. Um algoritmo de extração de feições deve ser capaz de extrair o número mínimo de feições a serem requeridas na etapa seguinte do registro de imagens, mesmo nos casos onde as imagens de ajuste e de referência não possuem uma área de sobreposição muito significativa ou em situações onde objetos da cena aparecem oclusos em uma das imagens. O algoritmo também deve ser capaz de extrair as feições independentemente das características geométricas e/ou condições radiométricas das imagens, assim como ser tolerante à presença de ruídos e mudanças ocorridas na cena representada pela imagem (ZITOVA; FLUSSER, 2003, p. 978). Quanto aos métodos automáticos de extração de feições, estes estão classificados em métodos diretos[3] e métodos baseados em feições.

Os métodos diretos enfatizam a segunda etapa do registro de imagens (casamento de feições) ao invés da etapa de extração. Desta forma, a primeira etapa do registro de imagens é omitida e nenhuma feição é extraída. São comumente utilizados em aplicações de análise de imagens médicas, visto que estas aplicações geralmente lidam com imagens não tão ricas em detalhes e não possuem feições muito destacadas.

Métodos baseados em feições consistem em identificar as feições presentes nas imagens de ajuste e de referência. São exemplos de feições:

a) regiões significantes: florestas, lagos e campos – tem sua representação em forma de PCs através de centros de gravidade. Estes tipos de feições são obtidas através de métodos de segmentação de imagens (PAL; PAL, 1993);
b) linhas: estradas, rios, limites de terrenos – geralmente expressadas sob forma

[3]Métodos diretos são também conhecidos na literatura como métodos baseados em área.

de PCs por pares de fins de segmentos de reta ou por pontos intermediários, são detectadas utilizando-se de métodos de detecção de bordas tais como detector de Canny (CANNY, 1986) ou detectores baseados no método gaussiano (MARR; HILDRETH, 1980);

c) pontos: intersecções de linhas, cantos, pontos em curvas acentuadas. Métodos para detecção de pontos podem ser encontrados em Rohr (2001).

Diferentemente de métodos diretos, métodos baseados em feições não lidam diretamente com os valores dos (*pixels*) da imagem. Esta propriedade faz desta abordagem a mais adequada para situações onde mudanças de luminosidade são esperadas ou quando as imagens são obtidas a partir de diferentes sensores. Este método é comumente utilizado em aplicações de sensoriamente remoto e visão computacional, as quais geralmente lidam com imagens que possuem objetos facilmente detectáveis. Recentemente, aplicações híbridas utilizando métodos diretos em conjunto com métodos baseados em feições também tem sido desenvolvidas (HELLIER; BARILLOT, 2003).

2.1.2.2 Casamento das feições

Após extraído um conjunto de feições nas imagens de ajuste e de referência, pode-se dar início à segunda etapa do registro de imagens.

> O processo de casamento estabelece a correspondência entre as feições. Cada feição na imagem de ajuste é casada com a correspondente feição na imagem de referência. As feições casadas são identificadas por coordenadas espaciais que identificam as suas posições na imagem. (FONSECA, 1999 apud BAGLI, 2007, p. 35).

É possível efetuar o casamento das feições extraídas nas imagens de ajuste e de referência tomando como base: a distribuição espacial das feições em ambas as imagens; os valores dos *pixels* nas regiões de vizinhança das feições e descrições simbólicas atribuídas às feições. Alguns métodos, ao tentar estabelecer o casamento entre as feições, efetuam simultaneamente a estimação dos parâmetros da função de transformação entre as imagens e, desta forma, combinam a segunda e a terceira etapa do registro de imagens (ZITOVA; FLUSSER, 2003, p. 981).

Problemas encontrados na etapa de extração de feições tais como a extração incorreta de feições, bem como a existência de degradações nas imagens podem dificultar a etapa de casamento de feições. Caso esta etapa seja feita de forma automática, deve-se escolher qual o algoritmo a ser utilizado, o que nem sempre é uma tarefa trivial. O algoritmo deve ser capaz de fazer uma boa distinção entre os vários tipos de feições extraídas nas imagens e ao mesmo tempo ser suficientemente estável de forma que não seja

influenciado por variações inesperadas das feições extraídas ou por ruídos existentes nas imagens (ZITOVA; FLUSSER, 2003, p. 979). Feições extraídas em uma imagem sem uma correspondente na outra imagem não devem, a princípio, afetar o resultado do algoritmo.

Os métodos para etapa de casamento de feições podem ser classificados em métodos diretos e métodos baseados em feições.

Métodos diretos combinam a etapa de extração e casamento de feições em uma só e utilizam as imagens sem que ter como objetivo detectar objetos de destaque. Nesta abordagem, janelas de tamanhos pré-definidos ou até mesmo imagens inteiras são utilizadas para a estimação das correspondências entre as imagens (ZITOVA; FLUSSER, 2003, p. 981). A fig. 3 exemplifica a utilização de uma janela na imagem de referência sendo comparada com as demais na imagem de ajuste.

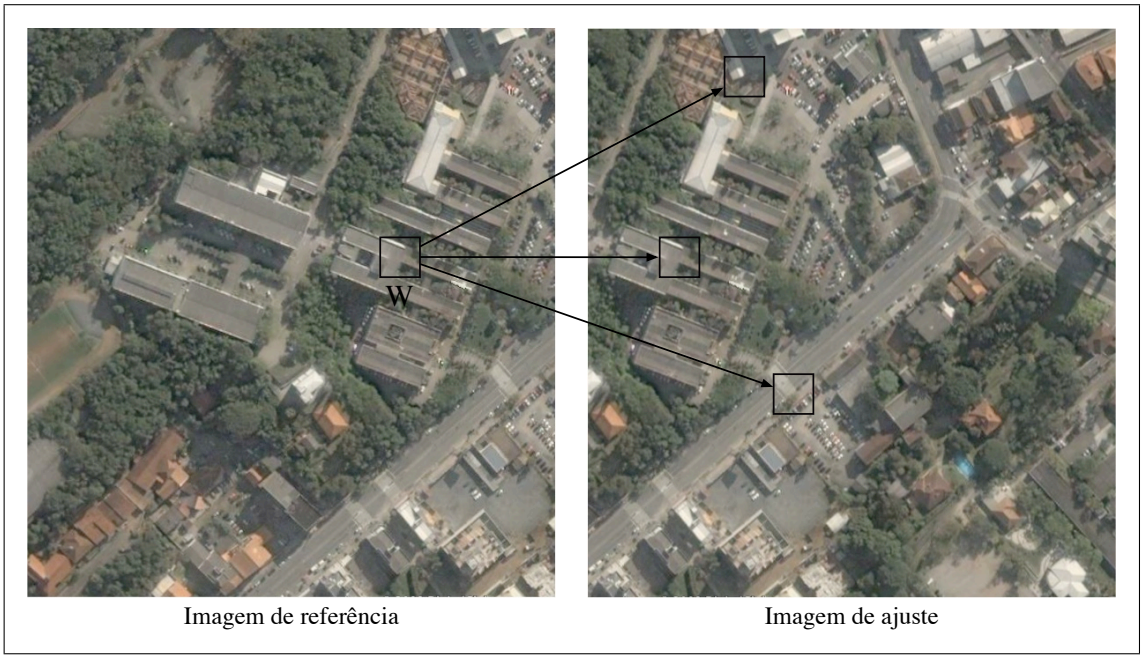


Figura 3 – Exemplo de casamento de feições pelo método direto: janela W na imagem de referência submetida à comparação com as demais janelas na imagem de ajuste

A maneira mais simples de comparação das janelas entre as imagens de ajuste e de referência é verificar todas as combinações possíveis (força-bruta). No entanto, técnicas hierárquicas baseadas em pirâmides de imagens têm sido utilizadas, muitas delas em conjunto com transformações de Fourier para tornar o processo mais rápido (SZELISKI, 2006, p. 15).

A comparação é realizada, normalmente, utilizando métodos de correlação[4] (SZELISKI, 2006, p. 35), onde tenta-se estimar a correspondência entre regiões das imagens de

[4]Consiste em medir o coeficiente de similaridade existente entre pares de janelas presente nas imagens de referência e de ajuste. Os pares que obterem os maiores coeficientes são definidos como correspondentes (ZITOVA; FLUSSER, 2003, p. 982).

referência e de ajuste através da minimização de uma função de erro, a qual é baseada nas diferenças de intensidade dos *pixels* na área de sobreposição das imagens (HEIKKILÄ; PIETIKÄINEN, 2005, p. 11). Este método apresenta como vantagem grande precisão no registro de imagens devido ao fato de que todos os dados (*pixels*) disponíveis na imagem são utilizados. Esta abordagem apresenta bons resultados quando as imagens de ajuste e de referência possuem uma grande área de sobreposição e tem sido submetidas a pequenas translações ou rotações (MALLICK, 2002, p. 1).

No entanto, métodos diretos possuem algumas desvantagens. Não são muito robustos contra variações de iluminação e a presença de objetos em movimento na cena geralmente resulta em sérios problemas, visto que todos os *pixels* da imagem são utilizados ao longo do processo (HEIKKILÄ; PIETIKÄINEN, 2005, p. 11). A utilização de janelas (quadradas) também apresenta uma desvantagem por sua natureza, visto que deformações ocorridas nas imagens podem inabilitar a utilização da mesma (uma forma retangular em uma figura pode ser mapeada para outra forma geométrica em outra). Alguns autores têm proposto outras formas de janelas (circulares), porém ainda apresentando deficiências (ZITOVA; FLUSSER, 2003, p. 982).

Métodos baseados em feições, ao invés de utilizar todos os dados presentes nas imagens, tentam estabelecer a correspondência entre feições extraídas na etapa anterior. Diferentemente dos métodos diretos, esta abordagem apresenta bons resultados mesmo na ocorrência de variações de iluminação, rotações e operações de *zoom* nas imagens de entrada (HEIKKILÄ; PIETIKÄINEN, 2005, p. 11). Além do mais, feições discrepantes podem ser removidas em casos de cenas contendo objetos em movimento. Esta abordagem é aplicada quando as imagens possuem pequena área de sobreposição e geralmente tende a ser bastante precisa, porém exige de mais recursos computacionais do que métodos diretos (MALLICK, 2002, p. 1). Ye (2005, p. 13) comenta a existência de métodos híbridos, os quais combinam tanto os métodos direto e baseado em feições para efetuar o casamento de feições.

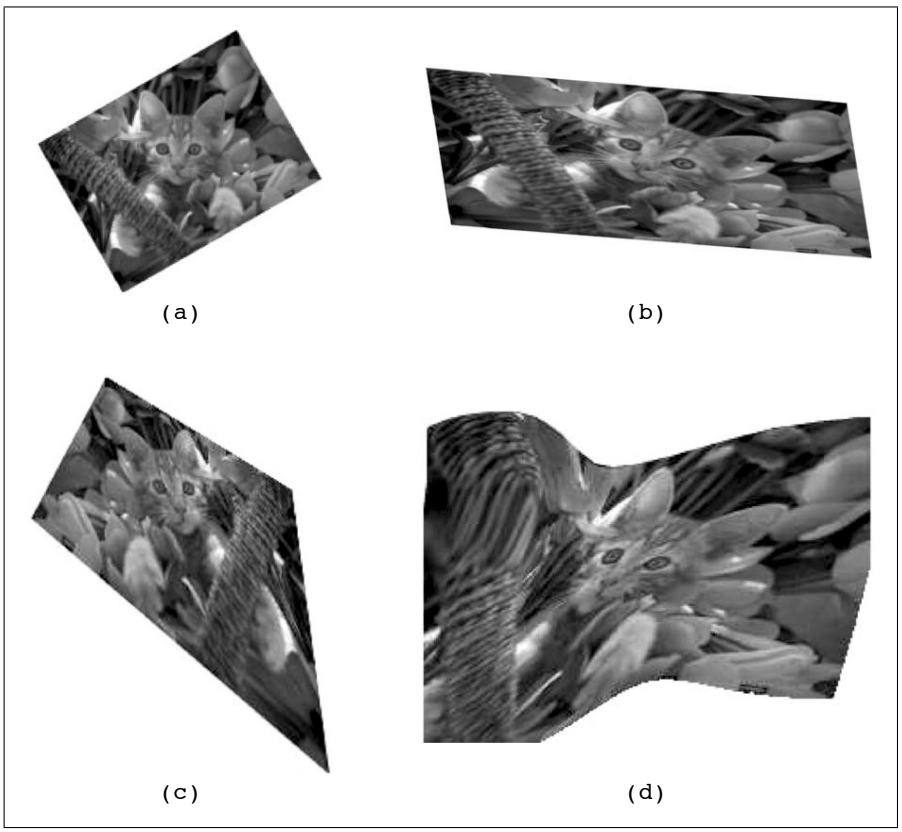
2.1.2.3 Estimação da função de transformação

Com as respectivas feições extraídas e casadas, pode-se aplicar a terceira etapa do registro de imagens, que consiste em escolher o tipo de função de transformação entre as imagens a ser utilizada e estimar os seus parâmetros. Em outras palavras, consiste em estabelecer um relacionamento matemático capaz mapear as coordenadas dos *pixels* de uma imagem para a outra (SZELISKI, 2006, p. 2). Após a aplicação da função de transformação, os PCs contidos na imagem de ajuste devem ter valores de coordenadas os mais

próximos possíveis aos da imagem de referência ocorrendo, desta forma, a sobreposição das imagens (ZITOVA; FLUSSER, 2003, p. 989).

Ye (2005, p. 8) afirma que o uso de funções de mapeamento[5] que descrevam de forma adequada a transformação entre as imagens é fator essencial para um registro de imagens preciso. Para a escolha do tipo de função de mapeamento deve-se tomar como base a deformação ocorrida entre as imagens, o método de aquisição e a precisão requerida pelo registro (ZITOVA; FLUSSER, 2003, p. 989).

Embora muitos tipos de distorções possam estar presentes em cada imagem, deve-se selecionar um tipo de transformação capaz de remover apenas as distorções espaciais (ocorridas devido aos diferentes modos de aquisição) e não as diferenças contidas nas características da cena em questão (BROWN, 1992, p. 336). A fig. 4 demonstra uma imagem submetida a diferentes funções de mapeamento.



Fonte: adaptado de Zitova e Flusser (2003, p. 989).

Figura 4 – Exemplos de funções de mapeamento: (a) similaridade; (b) afim; (c) projetiva e (d) arbitrária

Um estudo mais detalhado sobre funções de mapeamento, em especial a função de mapeamento projetivo é apresentado na seção 2.2.

[5]Refere-se à aplicação de uma função de transformação em todos os *pixels* de uma imagem.

2.1.2.4 Transformação e reamostragem das imagens

Esta etapa do registro de imagens consiste em utilizar as funções de transformação estimadas na etapa anterior para transformar (deformar) a imagem de ajuste de forma que esta seja registrada com a imagem de referência (ZITOVA; FLUSSER, 2003, p. 992).

Conforme House (2008, p. 1), este processo de deformação é também conhecido na literatura por *image warping* e consiste na aplicação de uma função de transformação capaz de alterar as propriedades geométricas de uma imagem. Heckbert (1989, p. 10) define esta técnica de deformação como o processo de reamostragem de uma imagem de referência que produz como saída outra imagem – denominada imagem de destino – de acordo com um mapeamento geométrico 2D.

Segundo Zitova e Flusser (2003, p. 992), a transformação e reamostragem de uma imagem pode ser realizada utilizando a função de transformação direta ou a função de transformação inversa. Uma vez que a primeira abordagem é mais complicada de ser implementada e pode produzir muitos artefatos[6] na imagem transformada, a transformação inversa tem sido utilizada mais freqüentemente nas aplicações de registro de imagem.

Um estudo mais detalhado sobre a transformação e reamostragem de imagens utilizando as funções de transformação direta e inversa, bem como as técnicas de interpolação necessárias para melhorar a qualidade visual das imagens resultantes é apresentado na seção 2.3.

2.2 MAPEAMENTO PROJETIVO

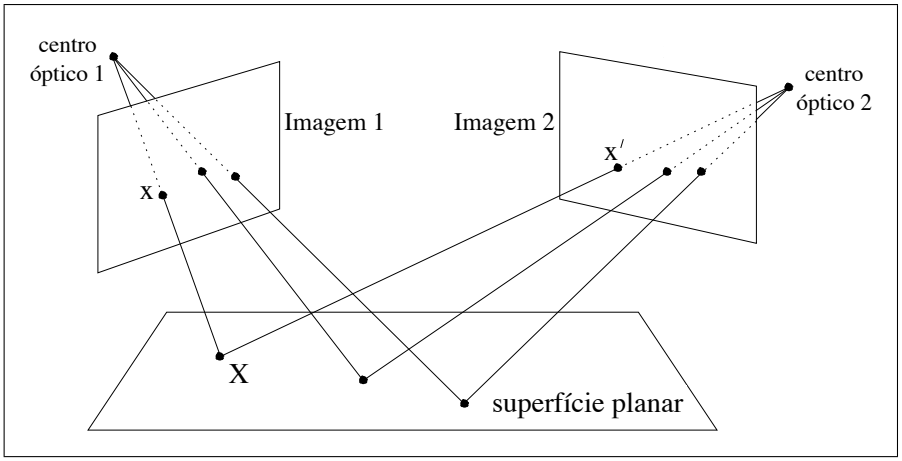
Nos casos onde os parâmetros de calibração de câmera são desconhecidos ou quando as distorções não-lineares são ou desprezadas ou compensadas dentro da imagem, a função de mapeamento entre as imagens pode ser estabelecida através de uma transformação projetiva planar (MCLAUCHLAN; JAENICKE, 2002, p. 3).

Uma homografia – também conhecida como projetividade ou colineação – é uma transformação projetiva planar linear e inversível capaz de mapear pontos (representados em sua forma homogênea) entre planos do espaço projetivo (HARTLEY; ZISSERMAN, 2004, p. 32). Estes planos podem ser representados pelos planos das imagens de ajuste e de referência, desde que sejam respeitadas algumas restrições.

Conforme Capel (2001, p. 18), existem duas importantes situações onde o mapeamento entre duas imagens é completamente capturado através de uma homografia planar:

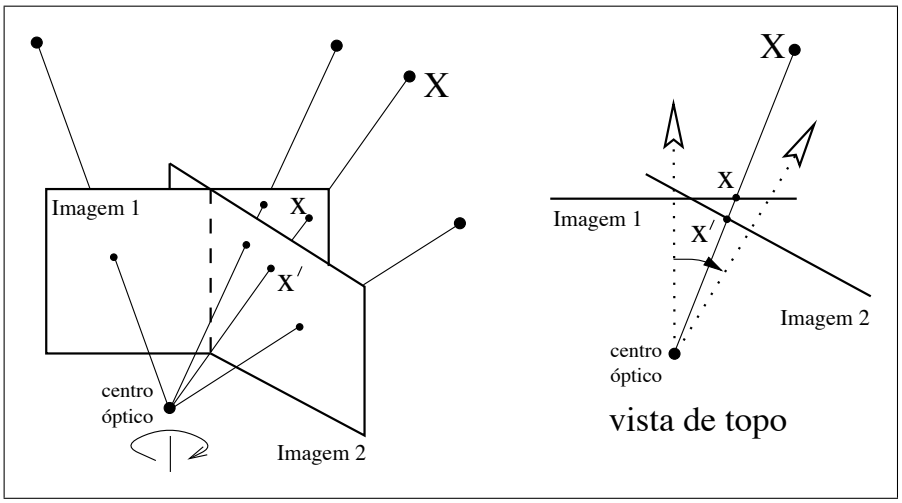
[6]Um artefato é qualquer característica que aparece em uma imagem representativa que não está presente na imagem original e é, geralmente, resultado de uma operação executada incorretamente em uma imagem.

a) imagens de uma região planar são obtidas por uma câmera submetida a movimentações arbirtrárias (fig. 5);
b) imagens de uma cena 3D são obtidas por uma câmera submetida a rotações em torno de seu centro óptico ou submetida a operações de *zoom* (variando a distância focal) (fig. 6).



Fonte: adaptado de Capel (2001, p. 23).

Figura 5 – Uma homografia planar H é induzida entre duas imagens de uma cena planar obtidas a partir de diferentes pontos de vista. O ponto de cena X é projetado nos pontos x e x' nas duas imagens, o quais são relacionados por $x' = Hx$



Fonte: adaptado de Capel (2001, p. 23).

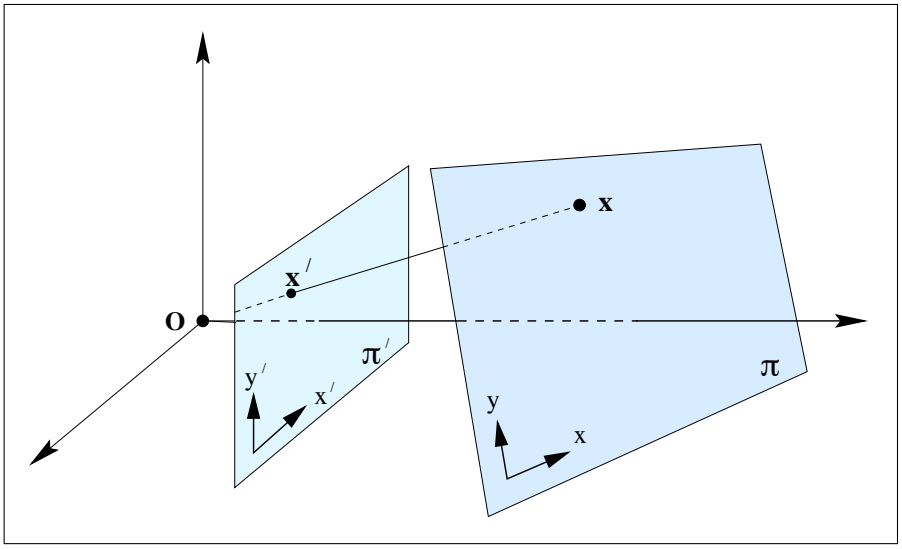
Figura 6 – A medida que a câmera é rotacionada, os pontos de intersecção dos raios com o planos das imagens são relacionados por uma homografia planar. Os pontos de imagem x e x' correspondem ao mesmo ponto de cena X. Os pontos de imagem são relacionados por $x' = Hx$

Uma terceira situação onde uma homografia pode ser apropriada é quando são obtidas imagens de uma cena muito distante – possivelmente por uma câmera em movimento livre – que é o caso de fotografias aéreas ou fotografias de satélite. O fato de a distância entre a câmera e a cena ser muito maior que a movimentação da câmera entre a obtenção

das imagens faz com que os efeitos de paralaxe[7] causados pela natureza tridimensional da cena se tornem relativamente pequenos (CAPEL, 2001, p. 18).

2.2.1 Transformações projetivas planares

Quando aplicada em imagens, uma transformação projetiva planar é capaz de projetar qualquer imagem em outra imagem projetivamente equivalente, deixando todas as suas propriedades projetivas invariantes (HARTLEY; ZISSERMAN, 2004, p. 33). Conforme demonstrado na fig. 7, a projeção de um raio ao longo de um ponto comum (centro de projeção O) define o mapeamento de um ponto x situado em um plano π para outro ponto x' situado em outro plano π'. O mesmo é valido caso um plano seja projetado ao longo do centro de projeção O, resultando no mapeamento de duas linhas retas em cada plano e, portanto, demonstrando que um mapeamento projetivo é capaz de preservar linhas retas (HARTLEY; ZISSERMAN, 2004, p. 34).



Fonte: Hartley e Zisserman (2004, p. 34).

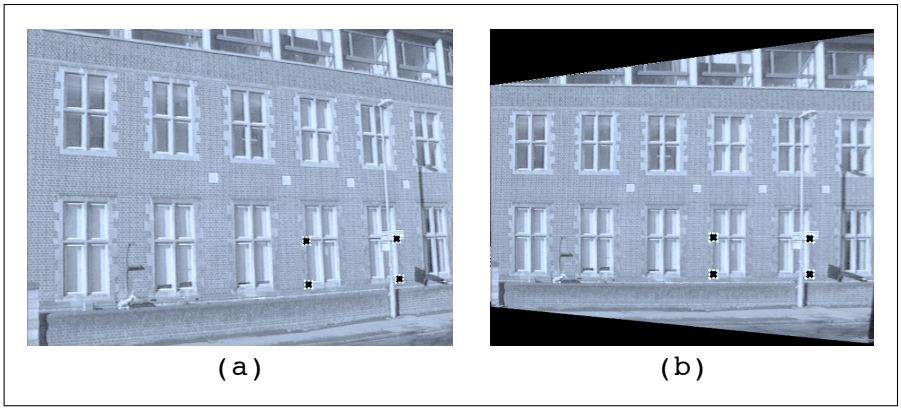
Figura 7 – Projeção central mapeando um ponto x situado em um plano π para outro ponto x' situado em outro plano π'

Segundo Hartley e Zisserman (2004, p. 34), as formas de um objeto são distorcidas em uma projeção perspectiva, como pode ser visto na fig. 8 (a) onde as janelas do edifício não aparecem retangulares na imagem, embora sejam no mundo real. De forma geral, neste tipo de projeção linhas paralelas em uma cena não são paralelas em uma imagem e sim, convergem para um ponto finito, podendo-se dizer que a imagem em questão é uma distorção projetiva da original.

No entanto, é possível desfazer a distorção projetiva calculando-se a transformação

[7] Paralaxe é a medida da aparente mudança da posição de um objeto em relação a um plano mais distante, quando visto a partir de ângulos diferentes.

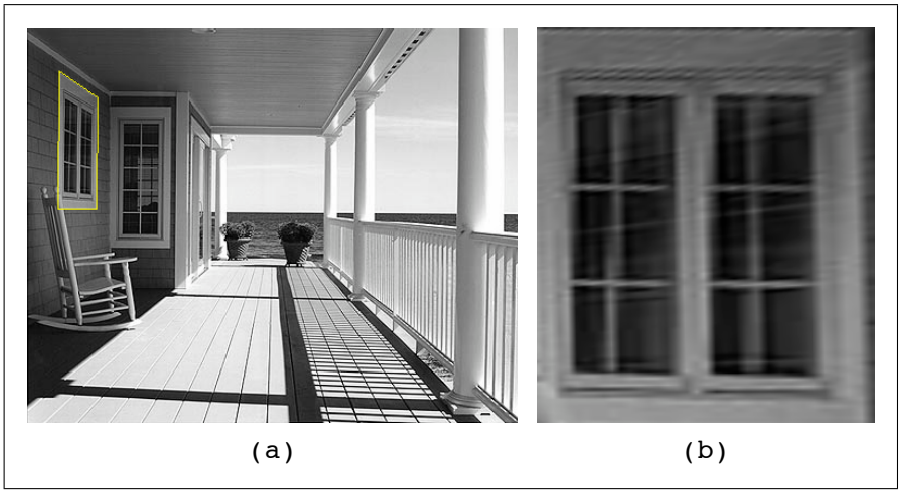
projetiva inversa e aplicando-a na imagem. O resultado será uma nova imagem na qual os objetos situados no plano (representado pelos quatro PCs) serão exibidos em sua forma geométrica correta (fig. 8 (b)).



Fonte: adaptado de Hartley e Zisserman (2004, p. 35).

Figura 8 – Remoção da distorção perspectiva: (a) imagem original com distorção perspectiva – as linhas das janelas convergindo para um ponto finito e (b) visão frontal ortogonal do edifício, gerada através da aplicação da transformação projetiva inversa

Outro exemplo é demonstrado na fig. 9, onde os cantos de uma janela presente na imagem (fig. 9 (a)) são mapeados para um polígono retangular (fig. 9 (b)), de forma a obter uma visão retificada da janela. Uma vez estabelecidos quatro PCs, sendo que três destes não sejam colineares, é possível calcular a transformação projetiva e aplicá-la na região de interesse da imagem.



Fonte: adaptado de Estrada, Jepson e Fleet (2005, p. 12).

Figura 9 – (a) imagem de uma cena vista sob projeção perspectiva e (b) imagem da janela retificada após a aplicação de uma transformação projetiva

Para a aplicação de transformações projetivas, uma consistente notação para representar os pontos em uma imagem é através de coordenadas homogêneas, cujos conceitos fundamentais são apresentados na seção seguinte, com base em Heckbert (1989, p. 13).

2.2.2 Coordenadas homogêneas

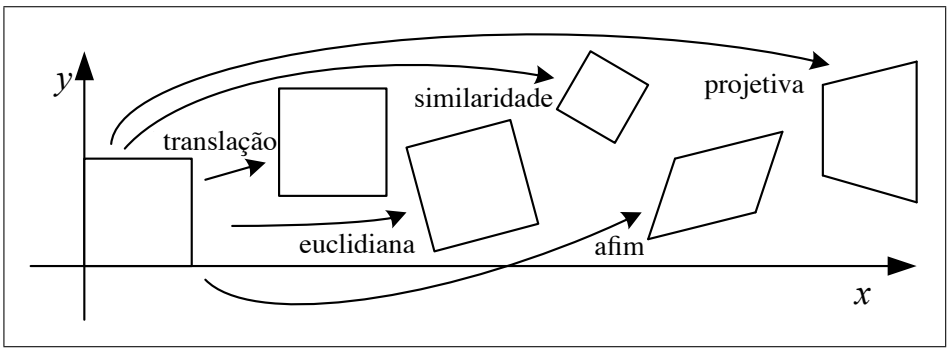
Na geometria euclidiana pontos do plano real $\mathbb{R}^2$ são representados através de vetores na forma (x, y). A geometria projetiva, por sua vez, lida com o plano projetivo $\mathbb{P}^2$ – um super-conjunto do plano real – cujas coordenadas homogêneas são representadas através de (x', y', w).

Na geometria projetiva, um ponto 2D real (x, y) é representado pelo vetor homogêneo $p = (x', y', w) = (xw, yw, w)$, onde w é um número arbitrário diferente de zero. Vetores da forma (xw, yw, w) para $w \neq 0$ constituem a classe de equivalência das representações homogêneas para um ponto real (x, y). Para obter novamente as coordenadas reais de um ponto a partir de um vetor homogêneo, deve-se simplesmente dividi-lo pelo componente homogêneo. Ex: O vetor homogêneo $p = (xw, yw, w) = (x', y', w)$ representa o ponto verdadeiro $(x, y) = (x'/w, y'/w)$. Esta divisão representa a projeção do ponto no plano $w = 1$ e cancela o efeito da multiplicação escalar por w.

Ao representar pontos através da notação homogênea, pode-se utilizar qualquer valor w diferente de zero, no entanto, é comum, por motivos de conveniência, escolher $w = 1$ de forma que coordenadas reais possam ser obtidas novamente sem divisão. O espaço projetivo também inclui os pontos no infinito[8]: vetores da forma $(x', y', 0)$, excluindo $(0, 0, 0)$.

2.2.3 Hierarquia de transformações projetivas planares

Segundo Hartley e Zisserman (2004, p. 37–43), Farias (2006, p. 13–15) e Szeliski (2006, p. 3–5), as transformações geométricas planares do espaço projetivo podem ser divididas em quatro classes: euclidiana, similaridade, afim e projetiva. Estas classes de transformações são exemplificadas na fig. 10, onde a transformação euclidiana aparece subdividida em uma translação seguida de uma rotação.



Fonte: adaptado de Szeliski (2006, p. 4).

Figura 10 – Transformações geométricas planares do espaço projetivo

[8] Também conhecidos como pontos ideais.

2.2.3.1 Transformação euclidiana

Uma transformação euclidiana, também conhecida como isometria (iso = mesmo, metria = medida), modela as translações e rotações de um objeto sem que ocorra nenhum tipo de deformação (transformação de corpos rígidos) e pode ser escrita como:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{1}$$

ou, mais concisamente:

$$\mathbf{x}' = \mathbf{H_E x} = \begin{bmatrix} \mathbf{R} & t \\ 0^t & 1 \end{bmatrix} \mathbf{x} \tag{2}$$

Uma transformação euclidiana possui três graus de liberdade (*Degrees Of Freedom* (DOFs)), sendo um de rotação e dois de translação. Suas invariantes[9] mais características são distância, ângulo e área. Este tipo de transformação pode ser calculada a partir de dois PCs correspondentes.

2.2.3.2 Transformação de similaridade

A este tipo de transformação pode-se aplicar um fator de escala isotrópico s, que permite alterar a proporção de um objeto. Sua representação matricial é dada por:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3}$$

ou, de forma mais concisa:

$$\mathbf{x}' = \mathbf{H_S x} = \begin{bmatrix} s\mathbf{R} & t \\ 0^t & 1 \end{bmatrix} \mathbf{x} \tag{4}$$

Após esta transformação, perde-se a noção da distância euclidiana por causa da mudança de escala, mas a medida dos ângulos permanece invariante, assim como a relação de paralelismo e a razão entre distâncias. Esta transformação possui quatro DOFs (três da transformação euclidiana mais um do fator de escala) e pode ser calculada a partir de dois PCs correspondentes.

[9]Elementos e/ou quantidades que são preservados após a aplicação de uma transformação (HARTLEY; ZISSERMAN, 2004, p. 38).

2.2.3.3 Transformação afim

Uma transformação afim pode ser expressa na forma matricial como:

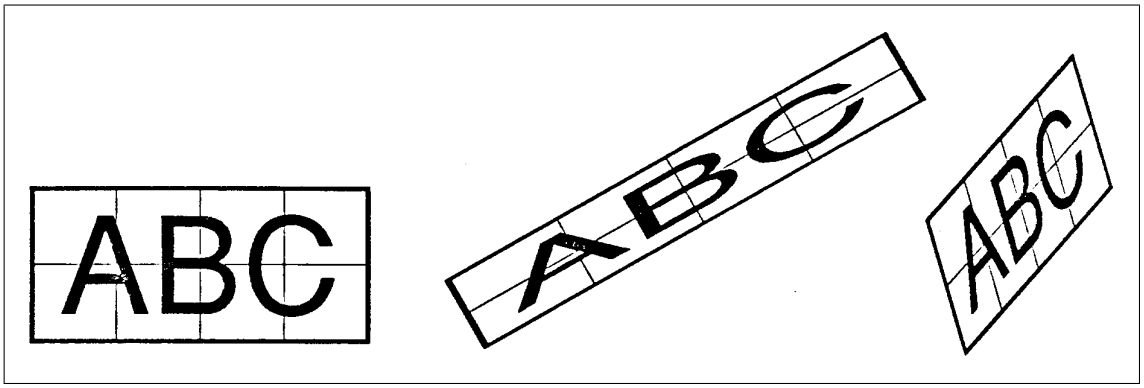
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{5}$$

ou então, mais concisamente:

$$\mathbf{x}' = \mathbf{H_A x} = \begin{bmatrix} \mathbf{A} & t \\ 0^t & 1 \end{bmatrix} \mathbf{x} \tag{6}$$

Neste caso, a submatriz A tem menos restrições que R na equação (2). Basta que A seja invertível, enquanto a matriz de rotação R tem que ser ortogonal. Esta característica faz com que H_A modele uma quantidade maior de operações, como o cisalhamento. Consequentemente, a quantidade de invariantes geométricas diminui, sendo estas: paralelismo; razão de comprimentos entre linhas paralelas e razão entre áreas. A fig. 11 demonstra uma imagem sendo submetida a diferentes transformações afins.

Este tipo de transformação possui seis DOFs (correspondendo aos seis elementos livres da matriz) e pode ser calculada a partir de três PCs correspondentes.



Fonte: Heckbert (1989, p. 14).

Figura 11 – Imagem submetida a transformações afins: o paralelismo, a razão de comprimentos entre linhas paralelas e a razão entre as áreas das imagens são preservados

2.2.3.4 Transformação projetiva

Sendo a classe de transformação mais genérica que pode-se obter, a transformação projetiva, também conhecida como transformação perspectiva ou homogênea, opera em

coordenadas homogêneas e pode ser expressa como:

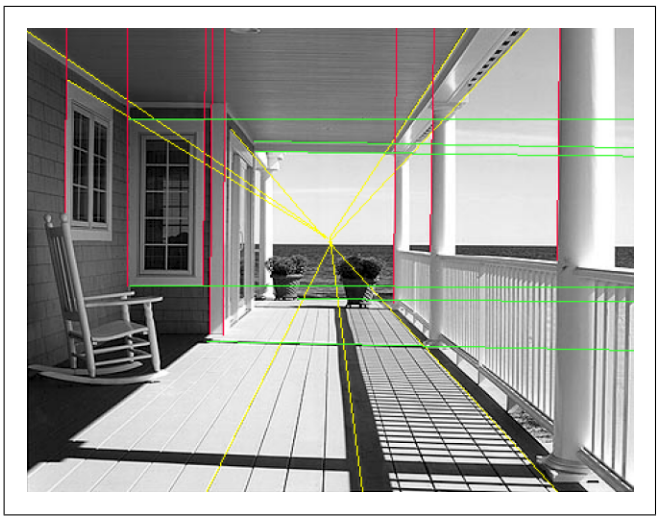
$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & v \end{bmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix} \tag{7}$$

ou então, mais concisamente por:

$$\mathbf{x}' = \mathbf{H_P x} = \begin{bmatrix} \mathbf{A} & t \\ \mathbf{v}^t & v \end{bmatrix} \mathbf{x} \tag{8}$$

A matriz H_P pode ser multiplicada por um fator de escala arbitrário $k \neq 0$ sem que seja alterada a transformação projetiva. Consequentemente, diz-se que H_P é uma matriz homogênea onde, semelhantemente a representação homogênea de um ponto, apenas a razão de seus elementos é significante. Existem oito razões independentes entre os nove elementos da matriz e, portanto, este tipo de transformação possui oito DOFs.

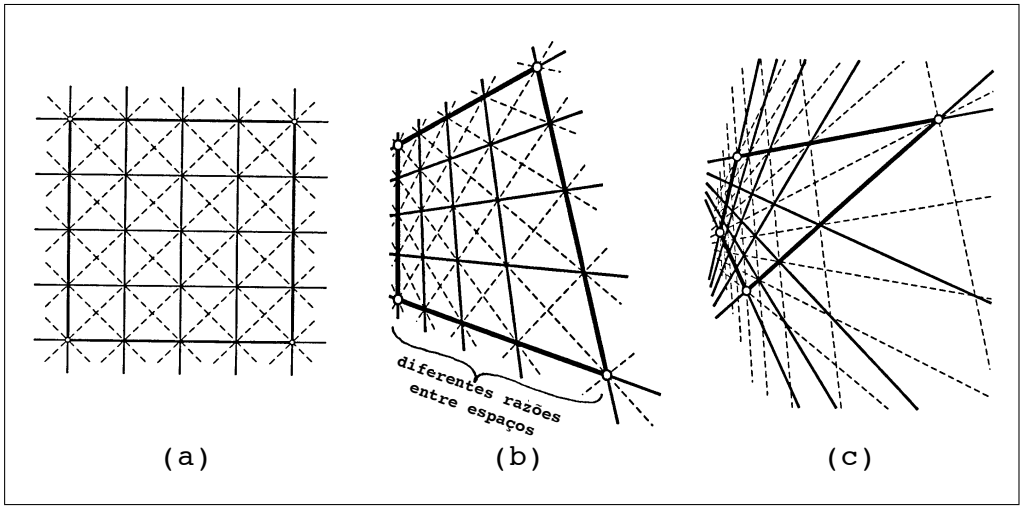
Semelhantemente a uma transformação afim, uma transformação projetiva preserva linhas retas em todas as orientações. A maior diferença entre estas duas transformações é que na transformação projetiva o vetor v não precisa ser necessariamente nulo. Esta característica faz com que a esta transformação tenha uma ação não-linear sobre os pontos quando representados de forma não-homogênea. Como consequência, pontos ideais podem ser mapeados como pontos finitos, tornando possível a representação de pontos de fuga (fig. 12), o que é fundamental para modelar uma projeção perspectiva. Por isso, os pontos resultantes da transformação devem ser normalizados de forma a obter seus valores na forma não-homogênea novamente.



Fonte: Estrada, Jepson e Fleet (2005, p. 13).

Figura 12 – Projeção perspectiva de uma cena: linhas paralelas convergindo para pontos de fuga. No caso das linhas horizontais e verticais os pontos de fuga estão localizados no infinito

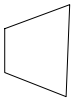
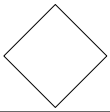
Uma transformação projetiva pode ser calculada a partir de quatro PCs correspondentes e seu invariante geométrico mais importante é o *cross ratio* ou razão cruzada (FARIAS, 2006, p. 11). A fig. 13 demonstra um quadrilátero submetido a transformações projetivas.



Fonte: adaptado de Heckbert (1989, p. 16–18).

Figura 13 – Um quadrilátero submetido a transformações projetivas: (a) quadrilátero original, (b) quadrilátero demonstrando a não preservação da razão de comprimentos entre linhas e (c) quadrilátero enfatizando a presença de pontos de fuga

O quadro 1 sumariza o grupo de transformações planares do espaço projetivo. Transformações mais inferiores no quadro são especializações das transformações mais superiores, herdando as propriedades invariantes das mesmas.

Grupo	Matriz	DOFs	Distorção	Invariantes
Projetivo	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & v \end{bmatrix}$	8		Razão cruzada, concorrência, colinearidade
Afim	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	6		Paralelismo, razão de comprimentos entre linhas paralelas, razão entre áreas
Similaridade	$\begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	4		Razão entre distâncias, ângulos
Euclidiano	$\begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$	3		Distância, área

Quadro 1 – Grupo de transformações planares do espaço projetivo

Estas classes de transformações constituem um grupo – denominado grupo projetivo linear – uma vez que tanto a transformação inversa quanto a transformação resultante

da composição de duas transformações de determinada classe resultam em outra transformação da mesma classe (*closed under composition*).

2.2.4 Estimação da função de mapeamento projetivo

Sendo os PCs nas imagens de ajuste e de referência representados em sua forma homogênea por $p_A = (u, v, q)^t$ e $p_R = (x, y, w)^t$, respectivamente, pode-se reescrever a equação (7), que mapeia PCs da imagem de ajuste para a imagem de referência como:

$$p_R = \mathbf{H_{AR}} p_A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} p_A \tag{9}$$

Conforme Heckbert (1989, p. 17), pode-se assumir sem perda de generalidade que $i = 1$. Desta forma, o mapeamento projetivo de um ponto p_A para um ponto p_R é dado por:

$$x = \frac{au + bv + c}{gu + hv + 1} \tag{10}$$

$$y = \frac{du + ev + f}{gu + hv + 1} \tag{11}$$

onde $a, b, \ldots, h$ são os parâmetros da transformação a serem encontrados. Sendo os PCs correspondentes a serem mapeados (u_k, v_k) e (x_k, y_k) para $k = 0, 1, \ldots, n$, aplicados à equação (10) e à equação (11), tem se:

$$x_k = \frac{au_k + bv_k + c}{gu_k + hv_k + 1} \Rightarrow u_k a + v_k b + c - u_k x_k g - v_k x_k h = x_k \tag{12}$$

$$y_k = \frac{du_k + ev_k + f}{gu_k + hv_k + 1} \Rightarrow u_k d + v_k e + f - u_k y_k g - v_k y_k h = y_k \tag{13}$$

Estas equações podem ser reescritas na forma de um sistema linear de ordem $2n \times 8$:

$$\begin{bmatrix} u_0 & v_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -v_0x_0 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -v_1x_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_n & v_n & 1 & 0 & 0 & 0 & -u_nx_n & -v_nx_n \\ 0 & 0 & 0 & u_0 & v_0 & 1 & -u_0y_0 & -v_0y_0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1y_1 & -v_1y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & u_n & v_n & 1 & -u_ny_n & -v_ny_n \end{bmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \\ y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{14}$$

Nota-se que devem existir $n \geq 4$ PCs correspondentes para que seja possível resolver o sistema linear e encontrar os parâmetros da transformação. Caso $n = 4$, o sistema linear pode ser resolvido diretamente, através do método de Gauss[10]. Na prática, o número de PCs correspondentes fornecidos é maior que quatro, resultando em um sistema linear com mais equações do que incógnitas, fazendo-se necessário utilizar o método dos mínimos quadrados (SZENBERG, 2001, p. 83).

Conforme Press et al. (1992, p. 34), se um sistema linear $Ax = b$ possui mais equações do que incógnitas, é denominado sobredeterminado e pode apresentar nenhuma ou várias soluções para o vetor x (vetor das incógnitas). Desta forma, pode-se utilizar a estimativa dos mínimos quadrados para encontrar uma solução que melhor satisfaça todas as equações simultaneamente.

Uma vez que transformações projetivas formam um grupo, a inversa de uma transformação projetiva é também uma transformação projetiva. Após estimados os parâmetros da matriz de transformação H_{AR}, é possível que seja calculada a matriz de transformação inversa H_{RA} para que esta seja utilizada para a transformação e reamostragem das imagens. Conforme Heckbert (1989, p. 18–19) pode-se tanto utilizar a matriz inversa quanto a matriz adjunta[11] de H_{AR} para obter-se a transformação inversa.

Em álgebra homogênea a matriz adjunta pode ser utilizada ao invés da matriz inversa sempre que uma transformação inversa se faz necessária, uma vez que a matriz adjunta sempre existe, enquanto a matriz inversa pode não existir caso a matriz de transformação seja singular.

A transformação inversa que mapeia PCs da imagem de referência para a imagem de ajuste pode ser escrita, tomando como base os parâmetros $a, b, \ldots, i$ da transformação H_{AR}, através de:

$$p_A = \mathbf{H_{RA}} p_R = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} p_R = \begin{bmatrix} ei - fh & fg - di & dh - eg \\ ch - bi & ai - cg & bg - ah \\ bf - ce & cd - af & ae - bd \end{bmatrix} p_R \qquad (15)$$

[10] Outras abordagens para a estimação da função de mapeamento projetivo envolvendo apenas quadriláteros podem ser encontradas em Heckbert (1989, p. 19–21).

[11] A matriz adjunta é a matriz transposta da matriz dos cofatores, ou seja, $M^{-1} = adj(M)/det(M)$.

2.3 TRANSFORMAÇÃO E REAMOSTRAGEM DE IMAGENS

Conforme House (2008, p. 1–2), o processo de transformação e reamostragem pode ser entendido como um mapeamento que transfere os *pixels* de uma imagem (imagem de entrada) para outra imagem (imagem de saída).

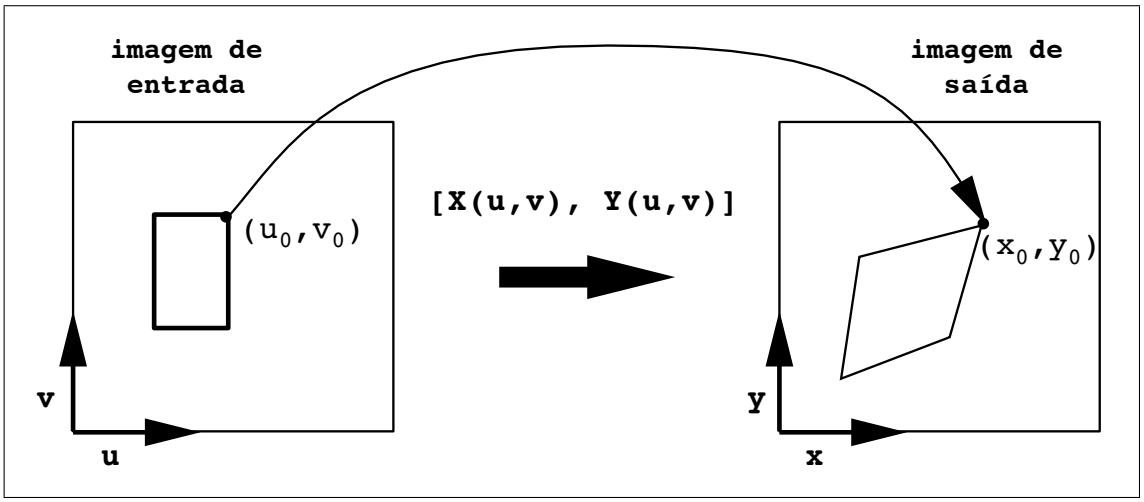
Com relação à ordem de processamento dos *pixels*, existem duas estratégias para efetuar o mapeamento entre as imagens: o mapeamento direto e o mapeamento inverso (GOMES; VELHO, 2003, p. 506).

2.3.1 Mapeamento direto

Segundo House (2008, p. 1–3), nesta abordagem cada *pixel* (u, v) da imagem de entrada tem seu valor transferido para o *pixel* $[X(u, v), Y(u, v)]$ da imagem de saída, onde as funções X e Y determinam as coordenadas (x, y) que correspondem às coordenadas (u, v). Esta concepção é ilustrada na fig. 14 e pode ser expressa matematicamente como:

$$x = X(u, v) \tag{16}$$

$$y = Y(u, v) \tag{17}$$



Fonte: adaptado de House (2008, p. 2).

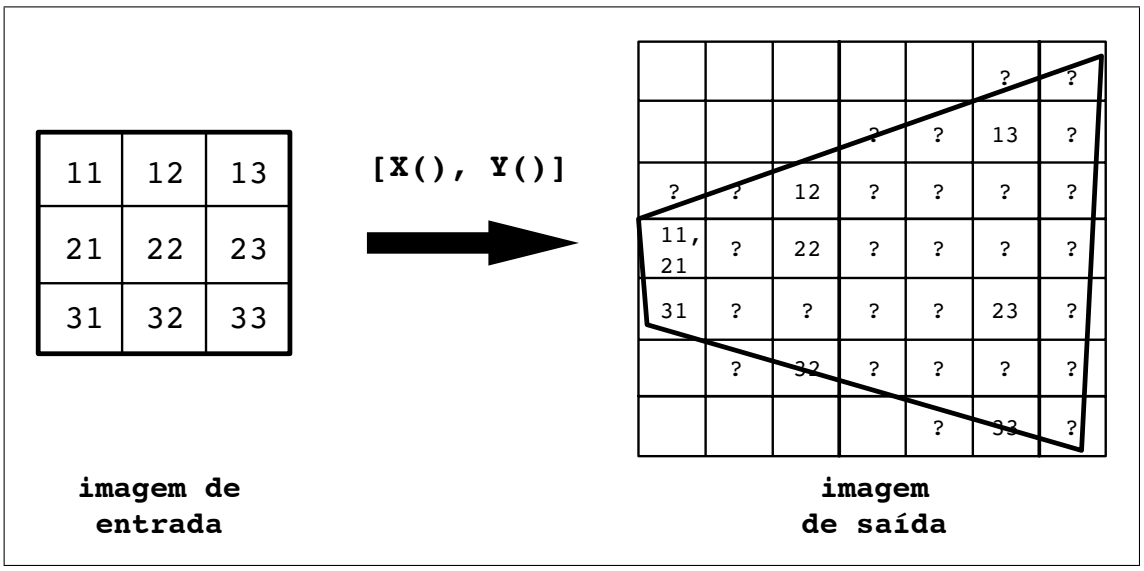
Figura 14 – Transformação e reamostragem da imagem de entrada através das funções X e Y

Dado que as funções X e Y podem mapear coordenadas inteiras para coordenadas não inteiras e que imagens digitais têm valores de *pixels* definidos apenas no domínio discreto, os valores de coordenadas resultantes da transformação devem ser arredondados. Este processo pode causar o aparecimento de artefatos na imagem de saída (HOUSE, 2008, p. 2).

Caso a função de transformação seja expansiva, isto é, produz uma imagem de saída

com escala maior do que a imagem de entrada, haverá uma diminuição da freqüência de amostragem e *pixels* na imagem de saída poderão ficar sem valores definidos, acarretando no aparecimento de "buracos" na imagem. Por outro lado, caso a função represente uma contração (aumento da freqüência de amostragem) vários *pixels* da imagem de entrada serão mapeados para um mesmo *pixel* na imagem de saída, ocorrendo a sobreposição dos mesmos (GOMES; VELHO, 2003, p. 503).

Dependendo do tipo de transformação aplicada à imagem, estes efeitos podem acontecer de forma simultânea, conforme esquematizado na fig. 15. A fig. 16 demonstra uma imagem submetida a transformação e reamostragem pelo mapeamento direto.



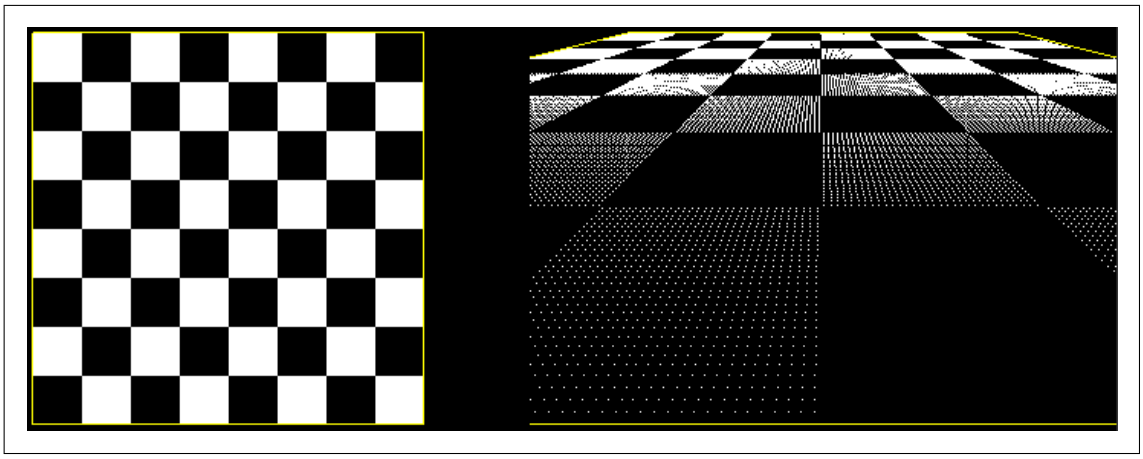
Fonte: adaptado de House (2008, p. 3).

Figura 15 – Mapeamento direto causando "buracos" e sobreposição de *pixels* na imagem de saída

Conforme House (2008, p. 3), tais efeitos ocorrem porque os *pixels* da imagem de entrada representam uma área bem definida dentro desta imagem e podem ser projetados de forma a representar uma área diferente na imagem de saída (fig. 17). Após a projeção, um *pixel* da imagem de entrada poderá cobrir apenas uma parte de um *pixel* na imagem saída, ou então cobrir vários *pixels* de forma completa ou parcial.

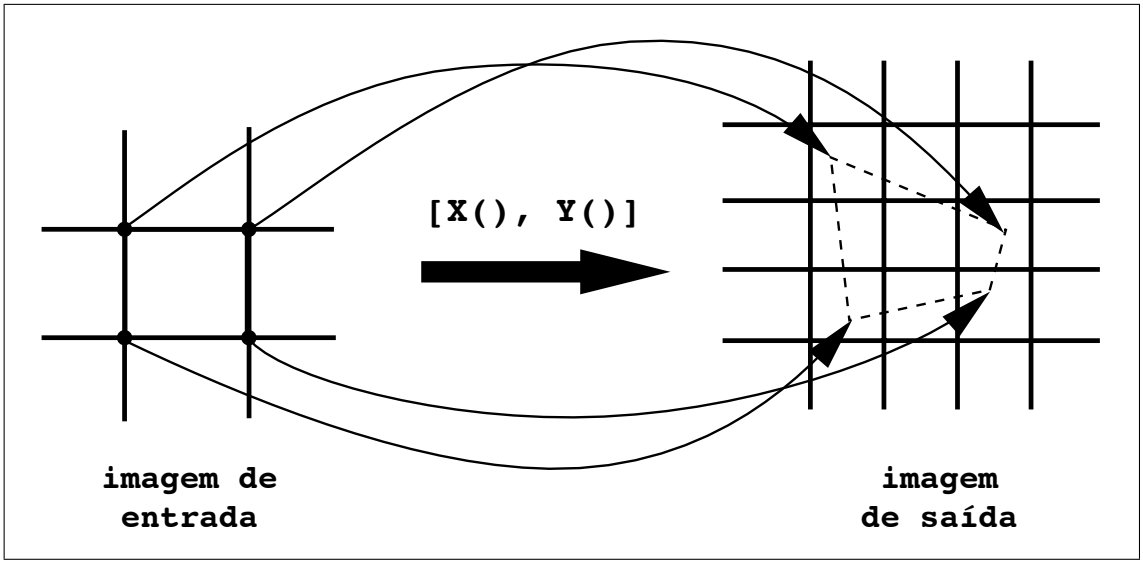
Uma maneira para contornar estes problemas seria percorrer todos os *pixels* da imagem de entrada e salvar a seu percentual de influência em cada *pixel* da imagem saída, tornando possível preenchê-la complemente sem a ocorrência de buracos ou sobreposição de *pixels*.

No entanto House (2008, p. 3) afirma que este cálculo poderia tornar-se bastante complexo, especialmente nos casos onde a função transformação é não-linear (linhas retas são mapeadas para linhas curvas).



Fonte: Vieira e Fernandes (2007, p. 4).

Figura 16 – Imagem transformada através do mapeamento direto: ocorrência de buracos e sobreposição de *pixels* na imagem de saída



Fonte: adaptado de House (2008, p. 4).

Figura 17 – Exemplo de área de representação de um *pixel* nas imagems de entrada e de saída

Por estes motivos, a abordagem do mapeamento inverso tem sido utilizada pela maioria das aplicações de registro de imagens, pois apresenta uma solução bem mais simples para estes problemas (ZITOVA; FLUSSER, 2003, p. 992).

2.3.2 Mapeamento inverso

Os efeitos indesejados apresentados pelo mapeamento direto podem ser removidos simplesmente invertendo o problema em questão, fazendo a transformação da imagem de entrada na imagem de saída através da função transformação inversa.

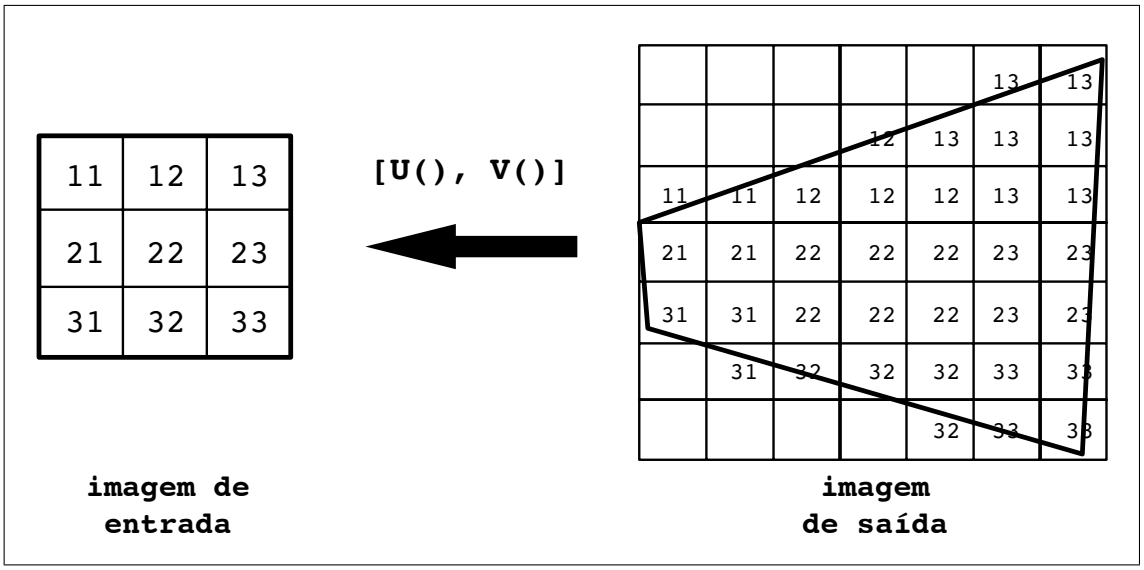
Ao invés de transferir o valor de cada *pixel* da imagem de entrada para a imagem de saída, percorre-se cada *pixel* da imagem de saída e determina-se qual é seu *pixel* correspondente na imagem de entrada. Assim então, as funções X e Y podem ser invertidas

e denominadas por $U(x, y)$ e $V(x, y)$ onde:

$$u = U(x, y) \tag{18}$$

$$v = V(x, y) \tag{19}$$

Desta forma, dada uma coordenada (x, y) na imagem de saída, sua correspondente na imagem de entrada é dada por $(u, v) = [U(x, y), V(x, y)]$. Assim como no mapeamento direto, os valores de coordenadas resultantes das funções U e V devem ser arredondados, porém, uma vez que esta operação acontece na imagem de entrada, não aparecerão buracos nem *pixels* sobrepostos na imagem de saída (fig. 18).



Fonte: adaptado de House (2008, p. 5).

Figura 18 – Aplicação do mapeamento inverso em uma imagem: todos os *pixels* da imagem de saída preenchidos sem ocorrência de buracos e/ou sobreposição

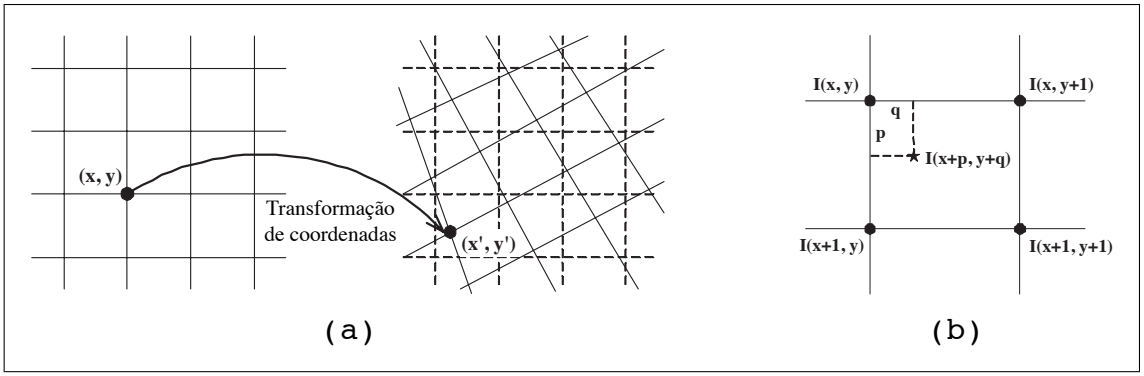
O mapeamento pelo método inverso é mais simples e rápido de ser calculado e por isso tem sido amplamente utilizado em várias aplicações nas área de computação gráfica e processamento de imagens, tais como mapeamento de textura e *morphing*[12] (HOUSE, 2008, p. 4).

[12]Técnica que provoca a mudança das formas de uma imagem através de efeitos de distorção e decomposição de cores (CONCI; AZEVEDO; LETA, 2008, p. 122).

2.3.3 Função de interpolação

Embora a transformação e reamostragem de imagens através mapeamento inverso resolva os problemas apresentados pelo mapeamento direto, deve-se ainda aplicar uma função de interpolação nos *pixels* da imagem de entrada para que estes sejam transferidos para a imagem de saída de forma a não favorecer o aparecimento de *aliasing*, o que pode degradar a qualidade visual da mesma (HOUSE, 2008; GOMES; VELHO, 2003).

Conforme Ye (2005, p. 11–13), as funções de transformação podem produzir como resultado valores de coordenadas não inteiros, posições na imagem de entrada onde não existem valores de intensidade definidos (fig. 19 (a)). Desta forma, faz-se necessário realizar uma aproximação ou arredondamento baseado em algum tipo de média entre *pixels* vizinhos, utilizando-se de uma função de interpolação.



Fonte: adaptado de Ye (2005, p. 11).

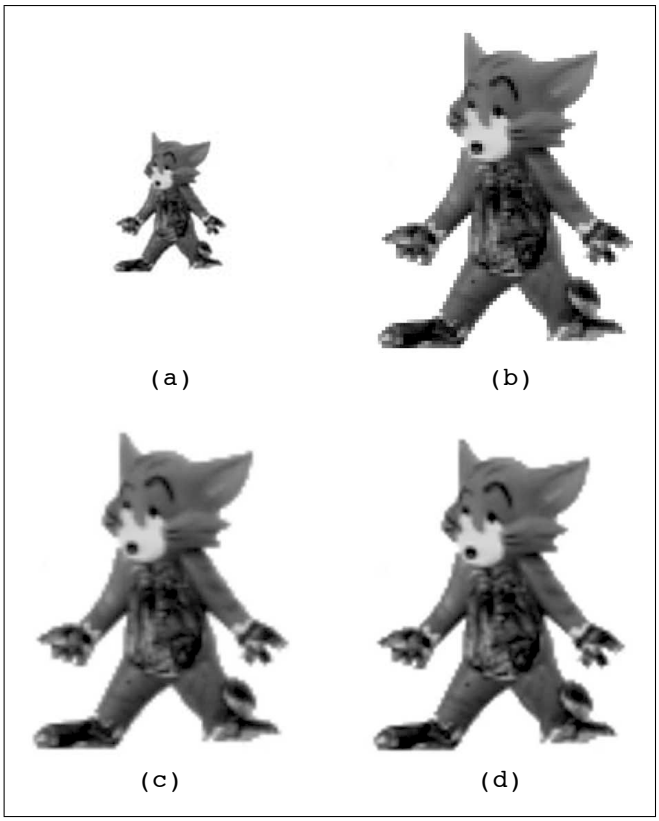
Figura 19 – (a) transformação de coordenada inteiras (x, y) para coordenadas não inteiras (x', y') e (b) interpolação bilinear de $I(x + p, y + q)$ com base nos *pixels* vizinhos

Segundo Zitova e Flusser (2003, p. 992), a técnica de interpolação ótima é feita através do interpolador *sinc*, porém, na prática é difícil de implementá-la devido a sua extensão infinita. Outras técnicas para interpolação de valores de *pixels* têm sido estudadas visando reduzir a complexidade computacional.

A técnica mais simples é a do vizinho mais próximo – também conhecida como interpolação de ordem zero – que simplesmente considera o valor do *pixel* localizado na coordenada inteira mais próxima. Esta técnica geralmente produz artefatos na imagem, tais como a distorção de linhas retas (YE, 2005, p. 12). Melhores resultados podem ser obtidos utilizando-se de técnicas de interpolação mais sofisticadas, tais como interpolação bilinear, bicúbica, *B-spline* quadrática, cúbica ou de ordens mais altas (ZITOVA; FLUSSER, 2003, p. 993). A fig. 20 demonstra uma imagem sendo reamostrada através das técnicas de interpolação de vizinho mais próximo, bilinear e bicúbica.

Dentre as técnicas citadas, a interpolação bilinear é superada por outras técnicas

de ordem mais alta em relação à precisão e aos resultados visuais obtidos. Mesmo assim, é a técnica mais comumente utilizada, pois oferece a melhor relação entre desempenho e complexidade computacional (YE, 2005; HEIKKILÄ; PIETIKÄINEN, 2005; ZITOVA; FLUSSER, 2003).



Fonte: adaptado de Zitova e Flusser (2003, p. 993).

Figura 20 – (a) imagem original, reamostrada utilizando três técnicas de interpolação diferentes: (b) vizinho mais próximo; (c) bilinear e (d) bicúbica

A técnica de interpolação bilinear determina o valor de intensidade baseada na distância ponderada média dos valores dos quatro *pixels* mais próximos ao redor de um ponto central. A fig. 19 (b) exibe quatro *pixels* vizinhos em volta de um *pixel* na posição $(x + p, y + q)$, onde $0 \leq p \leq 1$ e $0 \leq q \leq 1$. O valor de intensidade bilinearmente interpolado $I(x + p, y + q)$ pode ser expressado como:

$$\begin{aligned} I(x+p, y+q) =& (1-p)(1-q)I(x,y) + p(1-q)I(x+1,y) + \\ & (1-p)qI(x,y+1) + pqI(x+1,y+1) \end{aligned} \tag{20}$$

2.4 MOSAICO DE IMAGENS

Sendo uma das técnicas mais importantes de fusão de imagens, o mosaico de imagens consiste no alinhamento de múltiplas imagens em uma composição mais ampla que representa porções de uma cena 3D (CAPEL, 2001, p. 68). Ye (2005, p. 15) define mosaico

de imagens como o processo de efetivamente aumentar o campo de visão de uma câmera através da combinação de várias visões parciais de uma cena em uma única visão mais abrangente.

Imagens geometricamente combinadas e reamostradas, de forma que ruídos nelas presentes sejam reduzidos e que dêem a sensação de que todo o conjunto representa uma única cena consistem em um mosaico de imagens. Esta técnica pode ser utilizada para compor dezenas ou centenas de imagens permitindo a criação de cenas panorâmicas com ângulos de visão bastante amplos, como pode ser visto na fig. 21.

Fonte: adaptado de Capel (2001, p. 5).

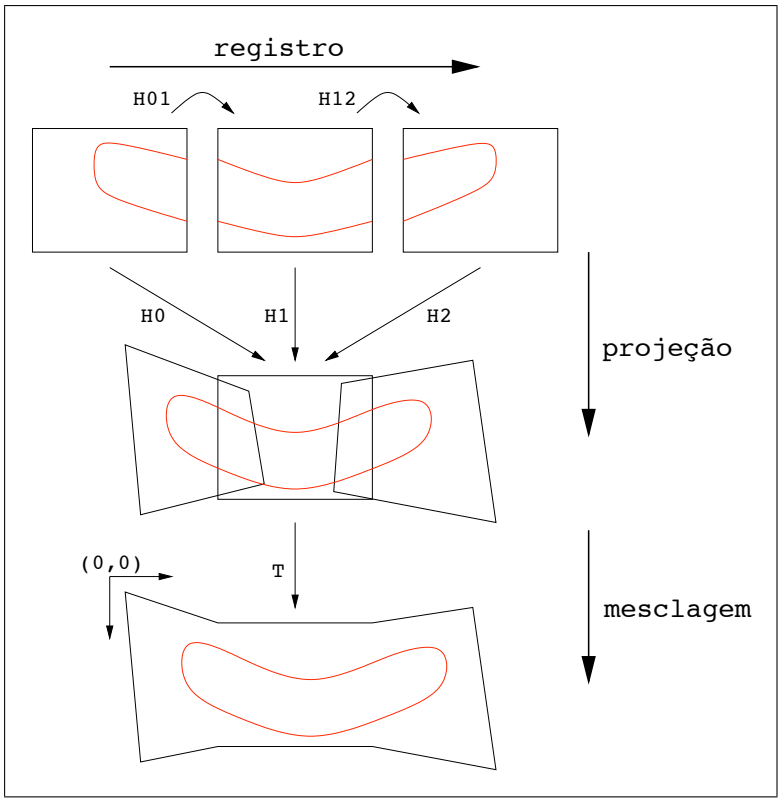
Figura 21 – (a) 8 imagens de uma seqüencia de 25 capturadas por uma câmera fotográfica e (b) todas as imagens combinadas através da técnica de mosaico de imagens

De acordo com Capel (2001, p. 68), tomando como base um conjunto de imagens que podem ser registradas através de uma homografia planar, existem três fatores importantes a serem considerados na construção de um mosaico de imagens: a estimação de homografias que representam de forma consistente o relacionamento entre as várias visões da cena em questão; a escolha de uma superfície de projeção na qual as imagens serão compostas e a escolha de uma técnica para efetuar a mesclagem das imagens nas regiões de sobreposição. Em outras palavras, o processo de construção de mosaico de imagens envolve basicamente três etapas (fig. 22): o registro; a projeção e a mesclagem[13] das imagens.

O registro de imagens, conforme explicado de forma mais detalhada na seção 2.1, no contexto de mosaico de imagens consiste em registrar cada imagem da cena em questão em um sistema de coordenadas comum. A princípio qualquer sistema de coordenadas pode ser

[13]Esta etapa é também conhecida na literatura *image blending* ou *image compositing*.

escolhido, no entanto, para termos de conveniência, escolhe-se o sistema de coordenadas da imagem que está melhor centralizada na cena – denominada imagem de referência – na qual as demais imagens do mosaico serão inseridas. Desta forma, uma transformação T entre o sistema de coordenadas da imagem de referência e o sistema de coordenadas global do mosaico é simplesmente a identidade (CAPEL, 2001, p. 69).



Fonte: adaptado de Capel (2001, p. 71).

Figura 22 – Etapas da construção de mosaicos de imagens

Após registradas as imagens de entrada, tem-se a etapa de projeção que consiste em definir uma transformação capaz de fazer o mapeamento destas imagens para uma superfície de projeção na qual o mosaico será representado. Dependendo da superfície de projeção adotada, este mapeamento pode ser feito simplesmente através de uma transformação de similaridade, em outros casos, mapeamentos mais complexos podem ser necessários, tais como de rotações de câmeras sintéticas ou transformações cilíndricas polares (CAPEL, 2001; YE, 2005).

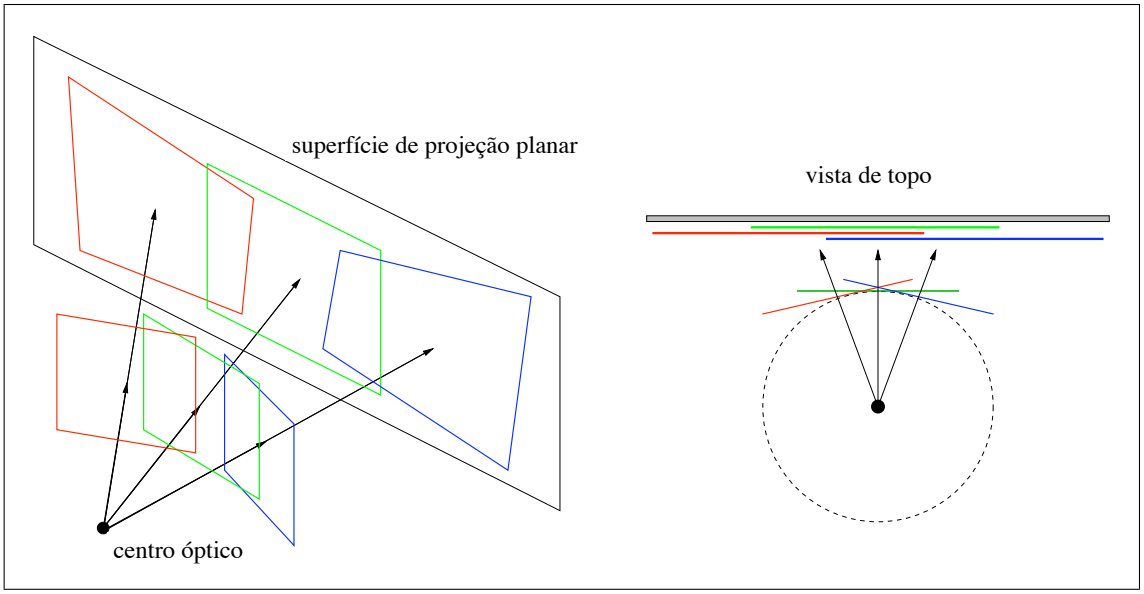
Por fim, a etapa de mesclagem tem como objetivo combinar as imagens que compõem o mosaico nas suas regiões de sobreposição de forma que a emenda de ligação entre estas imagens se torne imperceptível e, desta forma, dando impressão de que a toda cena em questão é representada por uma única imagem (CAPEL, 2001, p. 70).

2.4.1 Superfícies de projeção

Para que seja possível construir um mosaico de imagens, deve-se especificar uma transformação que mapeia pontos das imagens registradas para pontos da superfície de projeção. Esta transformação é denominada transformação de renderização e é determinada com base na superfície de projeção adotada. Os tipos de projeção mais utilizados são a planar e a cilíndrica, embora imagens possam ser mapeadas para superfícies de projeção mais complexas (CAPEL, 2001, p. 73).

2.4.1.1 Projeção planar

A superfície mais simples e mais comumente utilizada para a projeção das imagens que compõem um mosaico é um plano, conforme demonstrado na fig. 23. Neste tipo de projeção linhas retas são preservadas a transformação de renderização é determinada por um homografia.



Fonte: adaptado de Capel (2001, p. 74).

Figura 23 – Mosaico de imagens sendo renderizado em uma superfície de projeção planar

Mosaicos de imagens renderizados em uma superfície de projeção planar possuem a forma clássica de "gravata", uma vez que os efeitos da distorção projetiva aumentam em direção à periferia do mosaico. A fig. 25 (a) demonstra um mosaico de imagens renderizado neste tipo de superfície de projeção.

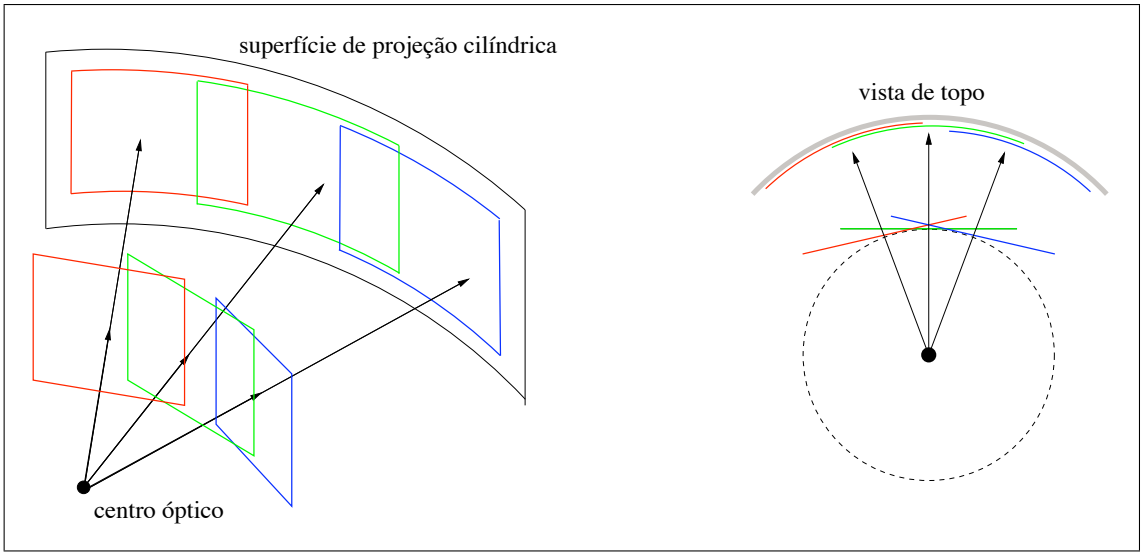
Caso a seqüencia de imagens obtidas represente uma cena cujo ângulo de cobertura seja superior a 90°, a distorção projetiva fará com que o mosaico passe a ter tamanho infinito e, desta forma, inviabilize o uso da superfície de projeção planar.

Capel (2001, p. 73) afirma que superfície de projeção planar pode ser entendida

como uma simulação de uma câmera que possui um sensor de captura extremamente amplo[14].

2.4.1.2 Projeção cilíndrica

Uma superfície de projeção cilíndrica pode ser utilizada para a projeção de uma sequência de imagens obtida por uma câmera que é rotacionada em torno de seu centro óptico (fig. 24). Neste caso, o ângulo de cobertura pode ser bastante amplo, chegando a 360°, como pode ser visto na fig. 26.



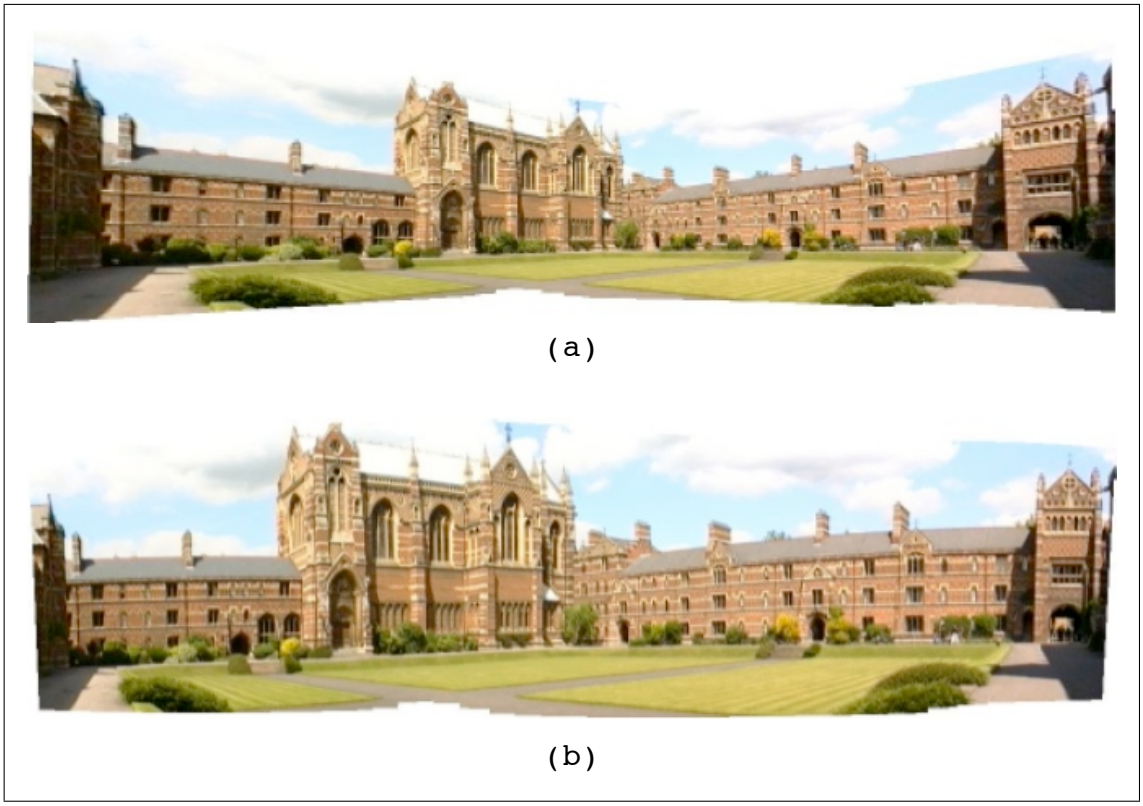
Fonte: adaptado de Capel (2001, p. 75).

Figura 24 – Mosaico de imagens sendo renderizado em uma superfície de projeção cilíndrica

Conforme Capel (2001) e Oliveira (2002), na projeção cilíndrica a câmera deve ser rotacionada em torno de um único eixo, no qual as imagens de entrada são planos tangentes à superfície de projeção.

Um mosaico de imagens renderizado em uma superfície de projeção cilíndrica não sofre a mesma distorção projetiva vista nos mosaicos planares. No entanto, linhas retas são mapeadas para sinusóides e a distorção projetiva é perdida, como pode ser visto na fig. 25 (b).

[14]Na realidade seria bastante difícil construir este tipo de câmera, pois imagens obtidas por um sensor bastante amplo estariam sujeitas aos efeitos de sombreamento (*vignetting*) em suas regiões periféricas (CAPEL, 2001, p. 73).



Fonte: adaptado de Capel (2001, p. 78).

Figura 25 – (a) mosaico de imagens criado através de uma projeção planar e (b) mosaico de imagens criado através de uma projeção cilíndrica

Fonte: Capel (2001, p. 112).

Figura 26 – Mosaico panorâmico 360° composto a partir de 178 imagens e renderizado utilizando uma superfície de projeção cilíndrica

2.4.2 Mesclagem

Cada *pixel* pertencente a imagem final do mosaico corresponde a um ponto na cena observada. Este ponto de cena pode ser projetado em várias imagens de entrada que se sobrepõem. Desta forma, a atribuição do valor de um *pixel* na imagem final pode ser feita com base em informações obtidas a partir de várias imagens de entrada, fazendo-se necessária a utilização de alguma técnica de mesclagem para decidir qual será o valor resultante do *pixel* (CAPEL, 2001, p. 76).

Segundo Capel (2001, p. 76–79), um algoritmo para a renderização de cada *pixel* pertencente à imagem final do mosaico pode ser descrito através do quadro 2.

1. Utilizando a transformação de renderização e as homografias previamente calculadas, transforma-se as coordenadas de cada *pixel* do mosaico final para as coordenadas de cada imagem de entrada (transformação inversa);
2. Para cada imagem no qual o *pixel* transformado está situado dentro das suas extremidades, obtém-se os valores de intensidade utilizando de uma técnica de interpolação;
3. Combina-se o conjunto de valores de intensidade obtidos na etapa anterior em um único valor utilizando-se de uma técnica de mesclagem.

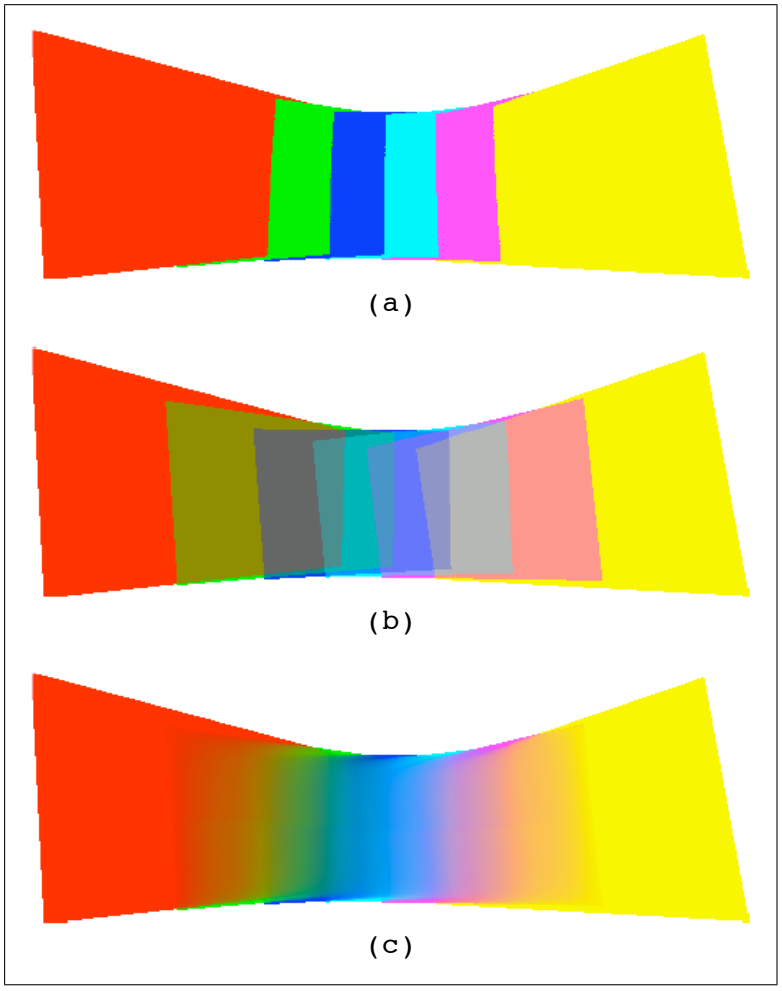
Quadro 2 – Algoritmo para a renderização de cada *pixel* pertencente à imagem final de um mosaico

As técnicas de mesclagem mais comuns encontradas na literatura são:

a) imagem do centro mais próximo: para cada imagem de entrada utilizada, calcula-se a distância do *pixel* em relação ao centro da imagem. O *pixel* que estiver mais próximo do centro tem seu valor transferido para a imagem final (fig. 27 (a));
b) média simples: o valor do *pixel* resultante é obtido simplesmente efetuando a média dos valores dos *pixels* extraídos nas imagens de entrada (fig. 27 (b));
c) média ponderada: uma função de peso é atribuída aos *pixels* das imagens de entrada, variando do valor máximo no centro das imagens para zero nas extremidades. Com base nestes valores define-se o valor do *pixel* resultante, de forma a suavizar a transição entre as imagens (fig. 27 (c)).

Dependendo das propriedades da câmera que obteve as imagens, bem como a natureza da cena observada, as imagens de entrada do mosaico podem apresentar padrões não comportados. Em muitos casos, as imagens podem apresentar efeitos de distorção radial, apresentarem diferenças radiométricas muito acentuadas ou então serem obtidas em cenas quem apresentam objetos em movimento. Nestes casos, técnicas mais especializadas

devem ser aplicadas de forma que o registro das imagens ocorra de forma correta e que as imagens, depois de mescladas, não contenham borrões ou objetos "fantasmas". Técnicas existentes para contornar estes tipos de problemas são comentadas em Capel (2001).



Fonte: adaptado de Capel (2001, p. 80).

Figura 27 – Seis cores primárias sendo mescladas através de diferentes técnicas: (a) imagem do centro mais próximo; (b) média simples e (c) média ponderada (suavização)

2.5 TRABALHOS CORRELATOS

Existem alguns trabalhos que desempenham papel semelhante ao proposto neste trabalho. Dentre eles, são descritos de forma geral: An Image Mosaicing Module for Wide Area Surveillance (HEIKKILÄ; PIETIKÄINEN, 2005) e Mosaico de Imagens Baseado em Múltiplas Resoluções (BAGLI, 2007).

2.5.1 An image mosaicing module for wide area surveillance

Heikkilä e Pietikäinen (2005) apresentam um sistema totalmente automático para construção de mosaicos a partir de imagens obtidas por câmeras de vigilância. O método proposto no trabalho é dividido em seis estágios: aquisição das imagens, extração de

feições, casamento das feições, estimação das transformações geométricas, *warping* e *blending* das imagens.

A aquisição das imagens é realizada por uma câmera que gira em torno de seu próprio centro óptico, de forma que as imagens possam ser relacionadas através de uma homografia planar.

Na etapa de extração de feições, PCs são obtidos através do processamento de cada imagem que é adicionada ao mosaico, utilizando um detector de feições. Segundo Heikkilä e Pietikäinen (2005, p. 12), este é o processo mais crítico do sistema e deve ser robusto em relação a ruídos, variações de iluminação, diferenças de escala e distorções perspectivas presentes nas imagens.

A terceira etapa estabelece a correspondência entre as feições detectadas na nova imagem obtida e as presentes no sub-mosaico.

Em seguida, utilizando-se do algoritmo RANdom SAmple Consensus (RANSAC), PCs correspondentes inválidos (causados por objetos em movimento ou por imperfeições do algoritmo utilizado na etapa anterior) são removidos e é estimada a transformação geométrica que relaciona os pares de imagens, a qual é refinada posteriormente através do algoritmo de Levenberg-Marquardt.

Na etapa de *warping* cada nova imagem de entrada é adicionada ao sub-mosaico utilizando-se do mapeamento inverso em conjunto com o método de interpolação bilinear.

Por fim, após registradas as imagens, a etapa de *blending* elimina as descontinuidades nas áreas de sobreposição das imagens ocasionadas devido a diferenças de iluminação, suavizando a transição entre estas imagens de forma que emendas não fiquem visíveis. Os valores dos *pixels* resultantes são determinados através da média ponderada dos valores dos *pixels* das imagens de entrada. Heikkilä e Pietikäinen (2005, p. 14) afirmam que a etapa de *blending* também serve para evitar o acúmulo de erros ao longo do processo, uma vez que o mosaico cresce incrementalmente. O resultado de um teste realizado é demonstrado na fig. 28.

Fonte: Heikkilä e Pietikäinen (2005, p. 17).
Figura 28 – Mosaico panorâmico construído a partir de uma sequência de 146 imagens

2.5.2 Mosaico de imagens baseado em múltiplas resoluções

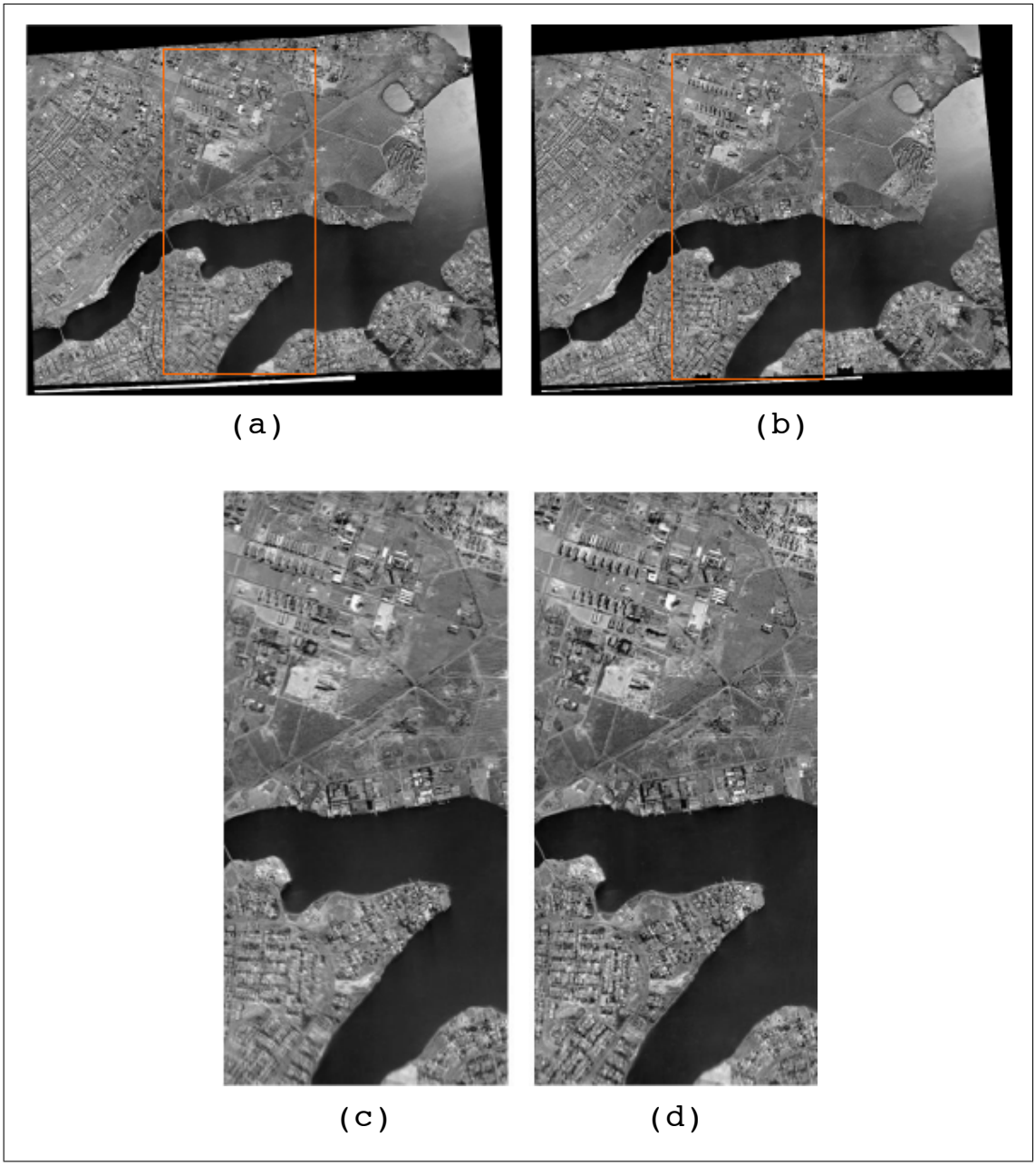
O trabalho proposto por Bagli (2007) apresenta uma metodologia de construção de mosaico de imagens aplicada ao sensoriamento remoto. O trabalho tem como foco a etapa de mesclagem de imagens, visando a construção de mosaico de imagens nos quais a emenda entre as imagens seja a mais imperceptível possível.

A etapa de registro de imagens é realizada automaticamente utilizando o sistema desenvolvido por Fedorov (2002). Durante esta etapa, após as feições terem sido extraídas e casadas, é estimada a função de transformação entre as imagens que, no caso de imagens de sensoriamento remoto, é comumente uma transformação afim.

Assumindo que a etapa de registro de imagens tenha sido feita de forma satisfatória, é realizada a mesclagem das imagens. Bagli (2007, p. 29) afirma que em muitos casos, mesmo utilizando-se a técnica de média ponderada para a mesclagem das imagens, a área de transição entre as imagens pode continuar perceptível devido à vários fatores, tais como diferenças de textura e de resolução espacial entre as imagens, processo de registro inadequado, áreas de transição e sobreposição pequenas ou linha de corte inadequada.

O método de mesclagem desenvolvido por Bagli (2007) combina técnicas de geração automática da linha de corte[15] entre as imagens – tais como a técnica *graphcut* – e a análise em múltiplas resoluções para suavizar a área de transição entre as imagens. A fig. 29 apresenta os resultados obtidos.

[15]Local onde as imagens são combinadas ou "costuradas".



Fonte: Bagli (2007, p. 78).

Figura 29 – Mosaico de imagens utilizando: (a) mesclagem através da média ponderada e (b) mesclagem baseada na análise de múltiplas resoluções e com definição da linha de corte automática. Em (c) e (d) são apresentados os detalhes dos mosaicos dentro das janelas marcadas em (a) e (b), respectivamente

2.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os aspectos teóricos envolvidos na construção de mosaicos de imagens. Foram apresentados o conceito de registro de imagens e as etapas envolvidas nesta técnica. Com relação à etapa de estimação da função de transformação, mostrou-se na seção de mapeamento projetivo como uma homografia planar pode ser induzida dado um par de imagens obtidas por uma câmera não calibrada. Quanto à etapa de

transformação e reamostragem das imagens, foram apontadas as diferentes abordagens existentes para a realização da mesma, bem como problemas que podem ocorrer ao longo do processo.

Mostrou-se também o conceito de mosaico de imagens, técnica que utiliza do registro de imagens para o aumento efetivo do campo de visão de uma câmera. Foram comentados os tipos mais comuns de mosaico de imagens existentes, as etapas envolvidas no processo de construção e as abordagens mais comuns para cada uma destas etapas.

Por fim, foram apresentados alguns trabalhos correlatos ao trabalho proposto, evidenciando as suas principais características.

3 DESENVOLVIMENTO DO SOFTWARE

Este capítulo detalha as etapas do desenvolvimento do software. São apresentados os requisitos, a especificação e a implementação do mesmo, mencionando as técnicas e ferramentas utilizadas. Também são comentadas questões referentes à operacionalidade do software e os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS

O software proposto possui uma interface gráfica que tem por objetivo permitir ao usuário fornecer as informações necessárias para a construção do mosaico.

Desta forma, foram elicitados, analisados e validados os seguintes requisitos:

a) permitir ao usuário fornecer como entrada imagens que representam a cena a ser reconstruída (Requisito Funcional (RF) 01);
b) permitir ao usuário definir as imagens de referência e de ajuste (RF 02);
c) solicitar ao usuário quatro ou mais PCs correspondentes entre as imagens (RF 03);
d) construir o mosaico de imagens com base nos dados informados (RF 04);
e) exibir o mosaico construído (RF 05);
f) permitir ao usuário salvar a imagem do mosaico construído (RF 06);
g) ser implementado na linguagem de programação C++ (Requisito Não-Funcional (RNF) 01);
h) utilizar a biblioteca IUP para a construção da interface gráfica (RNF 02);
i) utilizar a biblioteca IM para a manipulação das imagens (RNF 03);
j) utilizar a biblioteca CD para a renderização das imagens e dos elementos gráficos (RNF 04).

3.2 ESPECIFICAÇÃO

A especificação do software em questão foi desenvolvida seguindo a análise orientada a objetos, utilizando a notação UML (OMG, 2005). Sua a representação lógica é apresentada através dos diagramas de casos de uso, classes e sequência, os quais foram confeccionados utilizando a ferramenta OmniGraffle (THE OMNI GROUP, 2009).

3.2.1 Visão externa do sistema

Esta seção visa fornecer uma perspectiva do software desenvolvido a partir de um ponto de vista externo. Para tal, será utilizando o diagrama de casos de uso. No software em questão, o usuário faz o papel de iniciador, isto é, é a entidade externa que dá início à sequência de interações dele com o sistema. As principais interações do usuário com o sistema são apresentadas no diagrama de casos de uso da fig. 30.

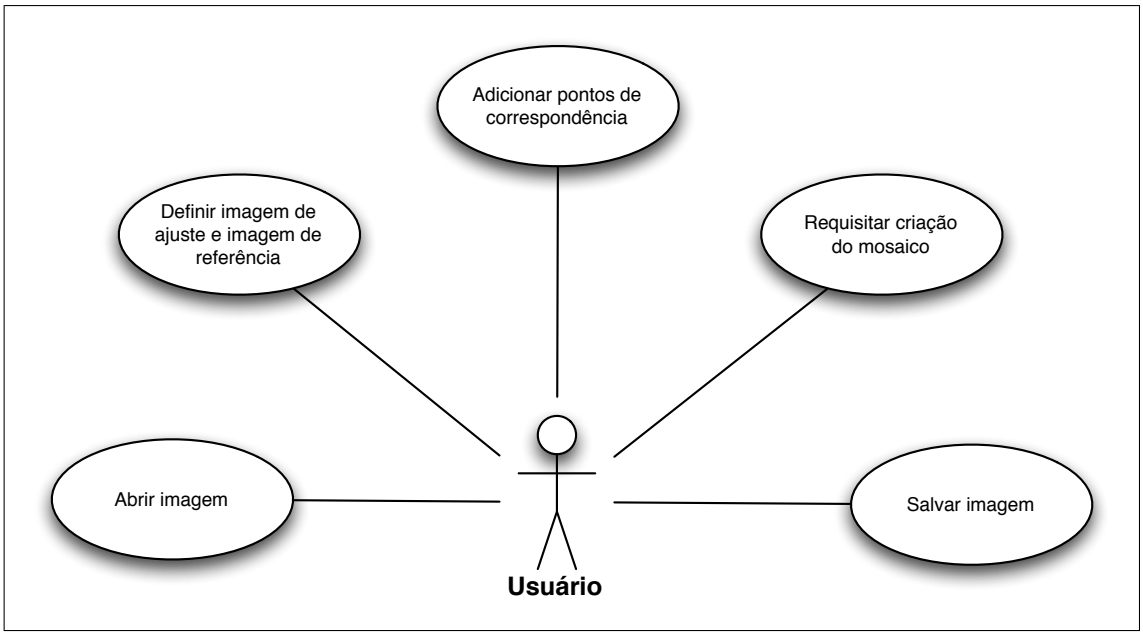


Figura 30 – Diagrama de casos de uso

A primeira interação que o usuário pode ter com o sistema é especificada no caso de uso `Abrir imagem`, o qual permite ao usuário fornecer como entrada as imagens que irão compor o mosaico.

O segundo caso de uso (`Definir imagem de ajuste e imagem de referência`) tem como objetivo permitir ao usuário definir qual será a imagem de ajuste (imagem que será transformada) e qual será a imagem de referência (imagem que permanecerá intacta dentro do mosaico). Uma pré-condição para este caso de uso é que ambas as imagens já tenham sido carregadas.

O caso de uso `Adicionar pontos de correspondência` tem por finalidade permitir que o usuário identifique em ambas as imagens PCs correspondentes, os quais serão utilizados para estimar a homografia que relaciona o par de imagens. Devem ser adicionados pelo menos quatro pares PCs correspondentes entre as imagens. A pré-condição deste caso de uso é que ambas as imagens de ajuste e de referência tenham sido devidamente definidas.

O quarto caso de uso (`Requisitar criação do mosaico`) possui como pré-

condição que tenham sido identificados pelo menos quatro pares de PCs correspondentes nas imagens. Satisfeita esta condição, o usuário pode requisitar a criação do mosaico. Um cenário de exceção para este caso de uso ocorre quando três dos PCs identificados forem colineares em uma das imagens, desta forma, a homografia não pode ser calculada e uma mensagem de advertência é apresentada.

O quinto e último caso de uso (`Salvar imagem`) permite ao usuário salvar em disco a imagem que estiver selecionada (comumente, o mosaico construído). Este caso de uso pode ser executado a qualquer momento, desde que exista alguma imagem carregada e selecionada.

3.2.2 Visão estrutural estática do sistema

Esta seção apresenta o software desenvolvido visto sob seu aspecto estrutural estático. É apresentada a arquitetura do sistema e, através de diagramas de classes, descreve-se como as classes que o compõem estão estruturadas e relacionadas.

O software foi desenvolvido utilizando o padrão de arquitetura *Model-View-Controller* (MVC) em conjunto com o padrão de projeto *Observer*.

O padrão MVC define como componentes presentes em diferentes camadas do software devem interagir. Como o próprio nome diz, são definidas três camadas: *Model*, *View* e *Controller*. A *Model* consiste na camada lógica do programa, contendo as classes referentes ao domínio do problema. A camada *View* consiste na interface com o usuário, é a tela de apresentação, efetivamente. Já a camada *Controller* é a encarregada de fazer a ligação entre as duas camadas anteriores, realizando as modificações que mostrarem-se necessárias (GAMMA et al., 1994, p. 4)[16].

O padrão *Observer*, também conhecido como *Dependents* ou *Publish-Subscribe*, define uma dependência de um-para-muitos entre objetos de modo que quando um objeto muda de estado, todos os seus dependentes sejam notificados e atualizados automaticamente (GAMMA et al., 1994, p. 293). Em outras palavras, permite que objetos interessados sejam avisados da mudança de estado ou outros eventos que venham a ocorrer dentro de um objeto de interesse.

A utilização destes padrões em conjunto permite um fraco acoplamento entre classes presentes em diferentes camadas. A fig. 31 apresenta a arquitetura do software de forma esquemática, onde linhas sólidas representam uma associação direta e as linhas tracejadas

[16]Variações deste padrão de projeto foram surgindo com o passar dos tempos para adaptarem-se a determinadas aplicações, onde algumas destas, embora desenvolvidas seguindo o padrão MVC, acabam unindo das camadas *View* e *Controller*.

representam uma associação indireta.

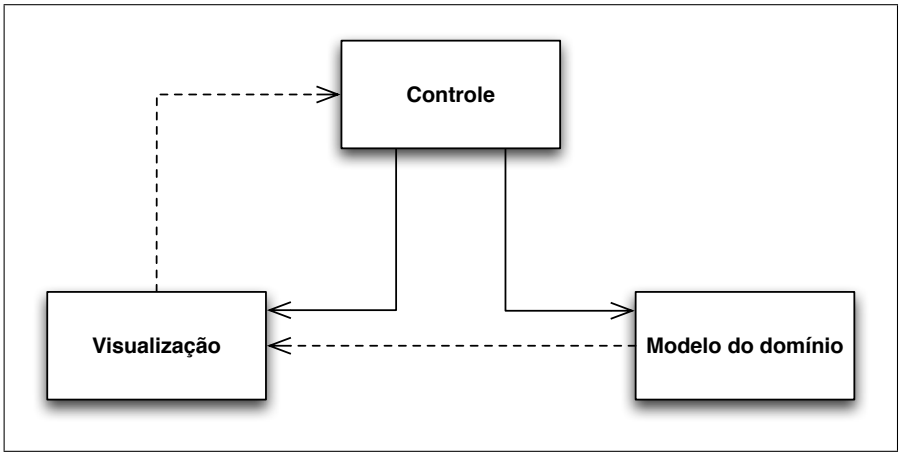


Figura 31 – Relacionamento entre as camadas de controle, visualização e modelo do domínio

As classes pertencentes às camadas de modelo do domínio, visualização e de controle foram descritas em diferentes espaços de nomes (*namespaces*), denominados `model`, `view` e `controller`, respectivamente. Uma explicação mais detalhada sobre cada uma destas camadas é apresentada na sequência.

3.2.2.1 Camada de modelo do domínio

Esta camada do sistema tem por objetivo descrever as entidades referentes ao domínio do problema, o qual refere-se às técnicas envolvidas na construção de mosaicos de imagens. O diagrama de classes da fig. 32 apresenta, de maneira macro, as classes e as estruturas de dados auxiliares que compõem esta camada, bem como o relacionamento entre estas.

Primeiramente, são definidas as estruturas de dados básicas para a manipulação de imagens, realização de cálculos matemáticos e definição dos modos de criação do mosaico. Estas estruturas estão destacadas em negrito no diagrama de classes da fig. 32.

Na sequência, tem-se a classe `Vector`, a qual tem por objetivo representar um vetor n-dimensional na aplicação. Além dos elementos do vetor, esta classe possui funções básicas para manipulação vetorial, a exemplo da multiplicação por um escalar. Objetos desta classe são comumente utilizados para representar pontos de coordenadas em sua notação homogênea.

A classe `Matrix`, por sua vez, representa uma matriz de ordem $n \times m$ e possui, além de métodos que implementam as funções elementares de manipulação de matrizes, um método estático para a resolução de sistemas lineares sobredeterminados, denominado `solveLinearSystem`.

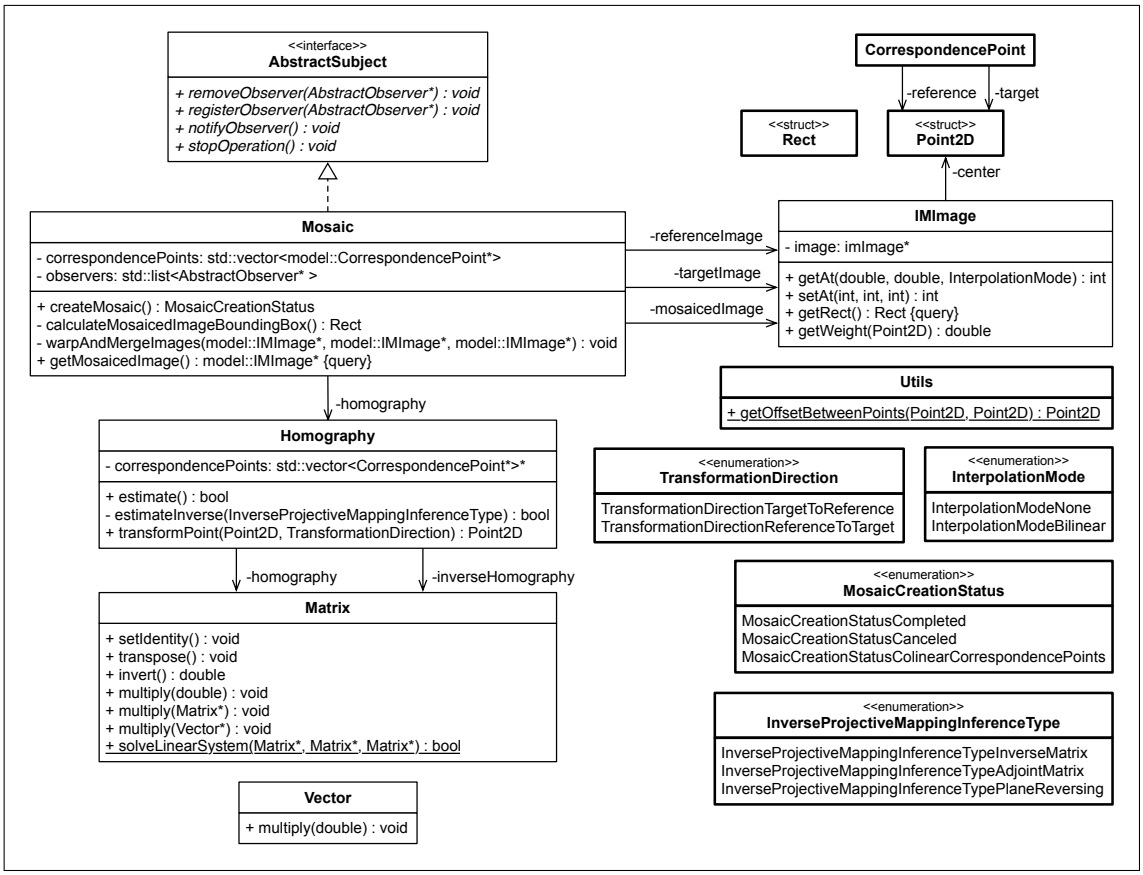


Figura 32 – Diagrama de classes da camada de modelo do domínio

Dado um conjunto de pontos de correspondência (referenciados no atributo `correspondencePoints`), a classe `Homography` tem como função utilizá-los para a estimação de homografias (direta e inversa), as quais são instâncias da classe `Matrix`. A classe `Homography` fornece outra funcionalidade essencial para a construção de mosaico de imagens onde, através do método `transformPoint`, pontos de coordenadas podem ser transformados tomando como base as homografias previamente estimadas.

Para a representação das imagens que compõem o mosaico, foi definida a classe `IMImage`, sendo que o sufixo `IM` é um acrônimo para *Image Mosaicing*. Esta classe encapsula a imagem propriamente dita (atributo `image`), sua posição relativa dentro do mosaico, bem como as suas dimensões. Dentre os principais métodos desta classe, tem-se:

a) `getAt`: retorna o componente *Red-Green-Blue* (RGB) em uma determinada coordenada de imagem (não necessariamente inteira) utilizando ou não de alguma função de interpolação;
b) `setAt`: define o componente RGB de determinado *pixel* da imagem;
c) `getRect`: retorna a área ocupada pela imagem dentro do mosaico de imagens;
d) `getWeight`: retorna o peso de determinada coordenada na imagem. Os valores possíveis estão no intervalo $[0 \ldots 1]$, com valor 1 para centro da imagem,

decaindo linearmente até 0 nas extremidades.

Por fim, tem-se a classe `Mosaic`, a qual representa o mosaico de imagens propriamente dito, objeto de estudo da aplicação. Esta classe armazena, primeiramente, as informações necessárias para a criação de um mosaico, sendo estas:

a) imagem de referência (`referenceImage`);

b) imagem de ajuste (`targetImage`);

c) conjunto de pontos de correspondência (`correspondencePoints`).

Tendo uma referência a um objeto do tipo `Homography`, é possível, com base nos pontos de correspondência, utilizar as homografias estimadas por este objeto para efetuar a transformação das imagens de entrada e compor o mosaico final (armazenado no atributo `mosaicedImage`).

Dentre os principais métodos referentes à criação de mosaico de imagens fornecidos por esta classe, destacam-se os apresentados no quadro 3.

Método	Descrição
`createMosaic`	Inicia a criação do mosaico de imagens, invocando os demais métodos privados da classe conforme necessário.
`calculateMosaicedImageBoundingBox`	Calcula a *bounding box* do mosaico final, de forma a alocar memória suficiente para a respresentação do mesmo.
`warpAndMergeImages`	Primeiramente, efetua a transformação da imagem de ajuste com base na homografia inversa estimada a partir dos pontos de correspondência. Em seguida, efetua a mesclagem das imagens de ajuste e de referência de forma a compor o mosaico final.
`getMosaicedImage`	Retorna o mosaico de imagens criado, uma instância de `model::IMImage`.

Quadro 3 – Principais métodos da classe `Mosaic`

Visto que as operações executadas nesta classe podem ser interrompidas pela camada de visualização do software, ou então devem ter seu estado notificado para a mesma, a classe `Mosaic` implementa a interface `AbstractSubject`. Esta interface define os métodos que devem ser implementados por um objeto que tem seu comportamento observado por outros objetos de interesse, conforme definido pelo padrão de projeto *Observer* (GAMMA et al., 1994, p. 293).

3.2.2.2 Camada de visualização

Esta camada tem por finalidade disponibilizar ao usuário uma interface gráfica que permita ao mesmo criar interativamente um mosaico de imagens, exibir os resultados obtidos e permitir que correções sejam realizadas. O diagrama de classes da camada de visualização em sua forma resumida pode ser analisado através da fig. 33.

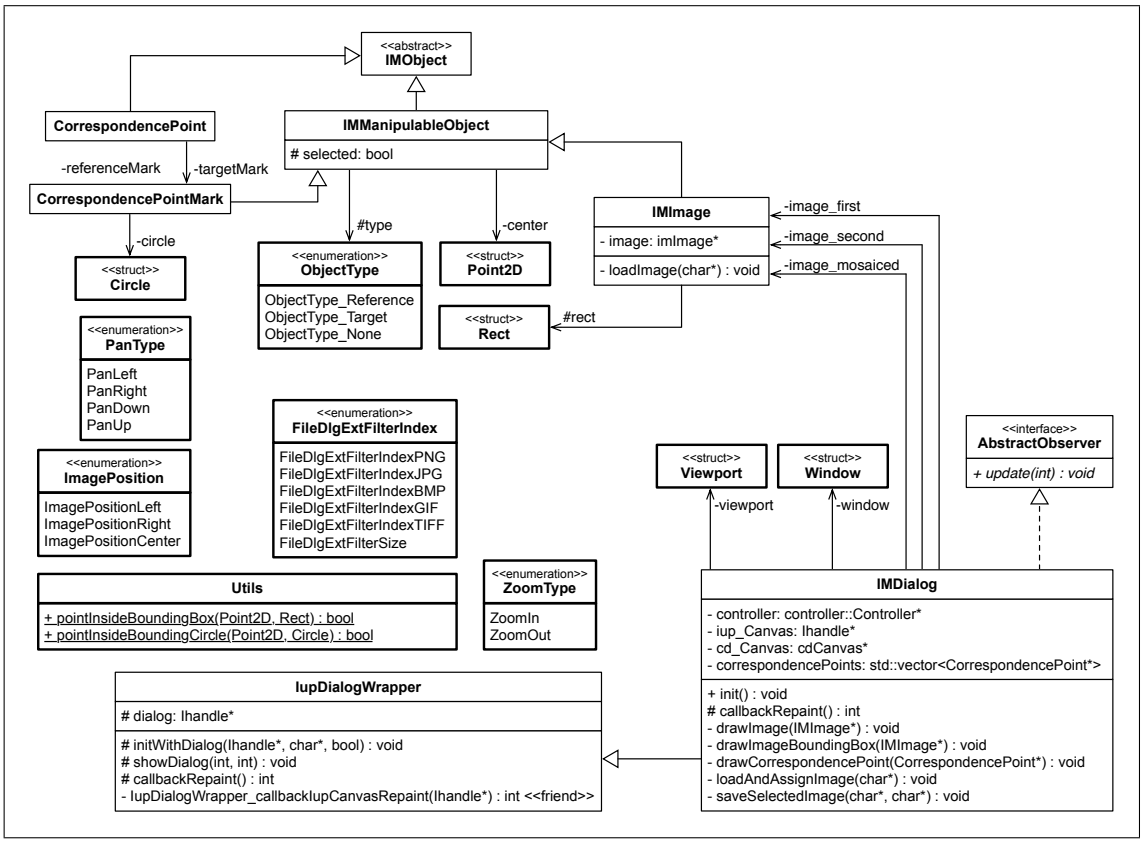


Figura 33 – Diagrama de classes da camada de visualização

Analogamente à camada de modelo do domínio, primeiramente são definidas as estruturas de dados básicas (destacadas em negrito no diagrama de classes da fig. 33) responsáveis pelas definições do *canvas*[17], *layout* e de operações básicas de manipulação de dados gráficos e de arquivos.

A classe abstrata `IMObject` serve como base para as demais classes da camada de visualização e representa os objetos que podem ser renderizados dentro de um *canvas*.

A classe `IMManipulableObject`, por sua vez, representa os objetos que podem ser renderizados e manipulados dentro de um *canvas*. Para isso, possui atributos que representam a posição, o estado e o tipo do objeto em questão.

Dentro do *namespace* `view`, a classe `CorrespondencePoint` consiste em representar graficamente pares de PCs, os quais são ligados por uma linha reta. Desta forma, esta

[17] Área designada na computação para desenho de formas gráficas 2D ou 3D.

classe agrega dois objetos do tipo `CorrespondencePointMark`.

A classe `CorrespondencePointMark`, a qual herda de `IMManipulableObject`, consiste na representação gráfica de um ponto de controle. Para que uma instância desta classe possa ser visualizada e manipulada de forma prática, definiu-se sua representação em forma de um círculo.

Sendo outra especialização de `IMManipulableObject`, a classe `IMImage`, definida no *namespace* `view`, representa tanto as imagens que são carregadas pelo usuário quanto o mosaico de imagens resultante. Esta classe contém a imagem propriamente dita (atributo `image`) – a qual pode ser carregada do disco através do método `loadImage` – bem como a sua posição relativa do *canvas* (atributo `rect`).

De forma a atender ao RNF 01 e ao RNF 02, foi definida uma *wrapper class* – denominada `IupDialogWrapper` – capaz de mapear as *callbacks* da biblioteca IUP (escrita em linguagem C) para métodos de classes escritas em C++ que representam o diálogo do software, neste caso, a classe `IMDialog`. Um exemplo desta funcionalidade pode ser observado no diagrama de classes da fig. 33, onde foram definidos na *wrapper class* uma função *friend* (`IupDialogWrapper_callbackIupCanvasRepaint`) e um método virtual (`callbackRepaint`), o qual deve ser implementado na sub-classe `IMDialog` para que esta passe a utilizar as *callbacks* conforme necessário. As demais *callbacks* foram suprimidas visando manter uma representação compacta do diagrama de classes.

Por fim, a classe `IMDialog` é a responsável por agrupar os elementos de interface e tratar os eventos que venham a ser disparados pelo usuário durante a utilização da ferramenta. Esta classe agrega três objetos `IMImage` visto que podem existir, no máximo, três imagens carregadas (e consequentemente renderizadas no *canvas*) ao mesmo tempo. Isto se deve ao fato de o mosaico ser construído a partir de pares de imagens, de forma incremental.

Através de uma referência à um objeto `controller`, a classe `IMDialog` é capaz de requisitar o processo de criação de um mosaico de imagens, bem como a sua interrupção, atendendo aos padrões MVC e *Observer*. Uma vez que a classe `IMDialog` implementa a interface `AbstractObserver`, torna-se possível acompanhar o progresso da operação de criação de um mosaico e apresentá-lo ao usuário sem que haja forte acoplamento entre classes de diferentes camadas.

3.2.2.3 Camada de controle

Conforme explicado na seção 3.2.2, esta camada tem por finalidade estabelecer a ligação entre as camadas de visualização e de modelo do domínio. O diagrama de classes

que representa a camada de controle pode ser visto na fig. 34 e consiste em uma única classe denominada `Controller`.

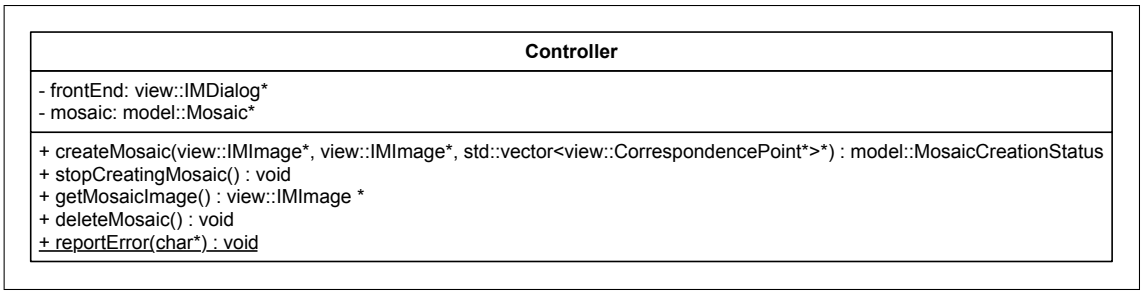


Figura 34 – Diagrama de classes da camada de controle

A classe `Controller` é a responsável por dar início à execução do software, instanciando e inicializando o diálogo da aplicação (`frontEnd`), bem como encerrar a execução liberando a memória alocada por este objeto. Esta classe é também responsável pela conversão entre os tipos de dados de diferentes *namespaces*, compatibilizando-os.

Através do método `createMosaic`, a camada de visualização do software pode requisitar a criação de um mosaico de imagens (o qual é retornado através do método `getMosaicImage`), passando as informações fornecidas pelo usuário. O método `stopCreatingMosaic` permite que seja interrompida a criação de um mosaico de imagens e o método `deleteMosaic` permite liberar a memória alocada para este objeto. Por fim, um método estático – denominado `reportError` – é definido de forma que classes de outras camadas do software possam reportar mensagens de erro para a saída de erro padrão (*stderr*).

3.2.3 Visão estrutural dinâmica do sistema

Nesta seção será apresentada uma visão dinâmica do sistema utilizando-se de diagramas de interações, mais especificamente diagramas de sequência, de forma que seja possível entender o fluxo de execução e as mensagens trocadas entre os objetos durante a execução das principais operações realizadas pelo software.

A fig. 35 apresenta o diagrama de sequência referente ao caso de uso `Requisitar criação do mosaico`, abordando as principais operações executadas pelo sistema quando a criação de um mosaico de imagens é requisitada pelo usuário.

O método `createMosaic` definido na classe `Mosaic` realiza uma das operações mais cruciais do software que é a criação de um mosaico de imagens, efetivamente. O processo de execução deste método é explicado de forma mais detalhada no diagrama de sequência da fig. 36.

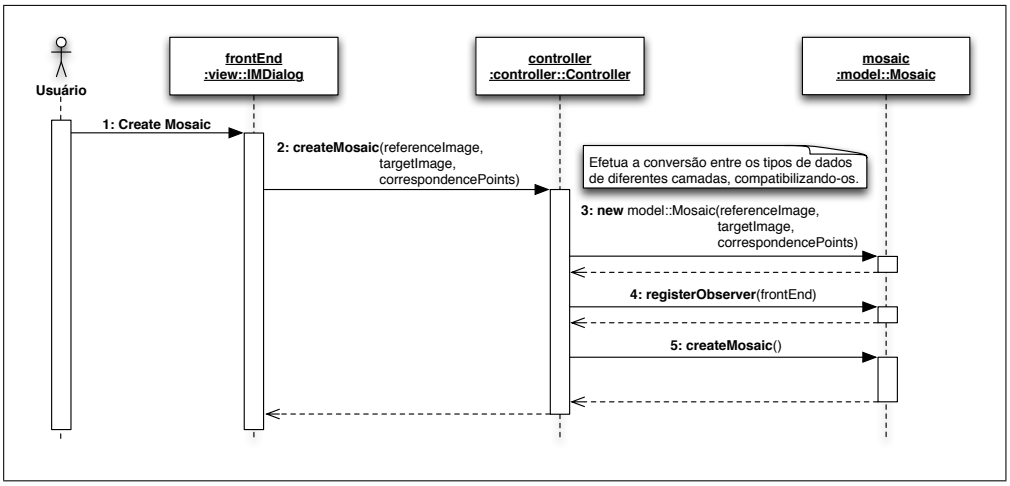


Figura 35 – Diagrama de sequência para o caso de uso `Requisitar criação do mosaico`

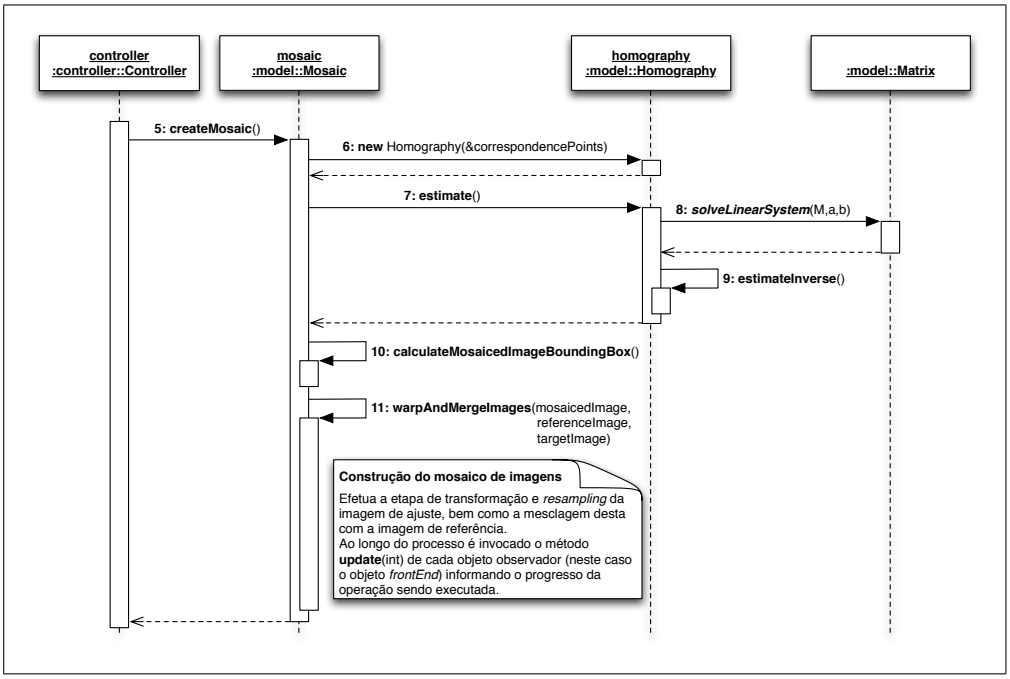


Figura 36 – Diagrama de sequência para o método `createMosaic`

3.3 IMPLEMENTAÇÃO

Esta seção tem por objetivo apresentar as ferramentas e bibliotecas utilizadas para a implementação do software proposto, bem como trechos de código fonte e questões referentes à operacionalidade do mesmo.

3.3.1 Ferramentas e bibliotecas utilizadas

A implementação software foi realizada utilizando a linguagem de programação C++, em conjunto com as bibliotecas: IUP (TECGRAF, 2009c) - para a construção da interface gráfica; IM (TECGRAF, 2009b) - para a manipulação de imagens e CD (TECGRAF, 2009a) para a renderização de imagens e de elementos gráficos.

O ambiente de desenvolvimento escolhido foi o Eclipse Ganymede (ECLIPSE, 2009b) em conjunto com o *plugin* Eclipse C/C++ Development Tooling (ECLIPSE, 2009a) e o compilador Minimalist GNU for Windows (MINGW, 2009). Para a documentação do código fonte foi utilizada a ferramenta Doxygen (HEESCH, 2009).

3.3.2 Técnicas e código fonte implementados

Nas seções seguintes são apresentadas as técnicas utilizadas para a resolução das principais operações realizadas pelo software durante a construção de mosaicos de imagens. Pseudocódigos e trechos de código fonte referentes às implementações de tais técnicas são demonstrados e brevemente explicados.

Primeiramente são explicadas as funções mais fundamentais para construção de mosaicos de imagens, que consistem nas operações matriciais, resolução de sistemas lineares, estimação de homografias (direta e inversa) e transformação de pontos.

Na sequência, apresenta-se o processo de construção de mosaico de imagens, efetivamente, explanando-se as principais etapas e métodos envolvidos, tais como: cálculo do tamanho do mosaico final; transformação, reamostragem e mesclagem das imagens de entrada e função de interpolação.

Também é apresentado um exemplo de método referente à camada de visualização do software que é responsável por desenhar um par de PCs correspondentes utilizando as bibliotecas gráficas citadas.

3.3.2.1 Inversão matricial

Para efetuar a operação de inversão matricial foi utilizado o método numérico direto de eliminação gaussiana com pivotamento parcial. Na solução implementada, o método é dividido em três fases: construção da matriz aumentada; eliminação progressiva e retro-substituição.

Para a construção da matriz aumentada, adiciona-se à direita da matriz a ser invertida uma outra matriz, denominada matriz da mão direita (*Right-Hand Side* (RHS) *matrix*), a qual é inicializada como uma matriz identidade.

Na segunda etapa do método é realizada a eliminação progressiva (*forward elimination*), onde é efetuado o pivotamento parcial aplicando-se as transformações elementares de linha na matriz aumentada, visando transformar a matriz RHS em uma matriz triangular.

Por fim, é executada a fase de retro-substituição (*back substitution*) onde, com a

matriz original alterada e a matriz RHS triangular, constrói-se a matriz inversa resolvendo-se as equações por elas relacionadas, percorrendo a matriz aumentada a partir da última até a primeira linha.

Para a escolha de tais métodos, foram levados em conta as seguintes questões:

a) o método de eliminação gaussiana, ao transformar a matriz a ser invertida em uma matriz triangular efetua menos pivotamentos que se esta fosse transformada em uma matriz diagonal (método de Gauss-Jordan) e, portanto, é menos suscetível a erros;

b) ao utilizar uma estratégia de pivotamento, é possível, através da reordenação das linhas da matriz, realizar a inversão desta matriz mesmo que exista algum elemento nulo em sua diagonal. Além do mais, esta estratégia reduz a propagação de erros durante a fase de eliminação;

c) a estratégia de pivotamento parcial exige menos esforço computacional que outras estratégias, a exemplo do pivotamento completo, sem acarretar em uma perda significativa na exatidão da solução.

Um embasamento teórico mais aprofundado sobre a inversão de matrizes utilizando tais técnicas pode ser encontrado em (PRESS et al., 1992).

No software desenvolvido, a operação de inversão de matrizes é realizada no método `invert` da classe `Matrix` e pode ser aplicada em matrizes n-dimensionais quadradas. O método `invert` retorna o determinante da matriz (0.0 caso a matriz seja singular) e altera os valores da matriz original somente se esta for invertível. Seu código fonte[18] é demonstrado no quadro 4 e no quadro 5.

[18] Os trechos de código fonte apresentados possuem comentários na língua inglesa visto que estes foram extraídos a partir do código fonte original do software.

```
 1 /**
 2  * Inverts inline this matrix using the Gaussian elimination algorithm with:
 3  * partial pivoting, forward elimination and back substitution.
 4  * If this matrix is invertible, its data is changed,
 5  * otherwise it is preserved.
 6  * @return The matrix determinant (<code>0.0</code> in the case of a
 7  * singular matrix).
 8  */
 9 double Matrix::invert() {

11     // Check whether the matrix is a square matrix, if not,
12     // report an error and return 0.0 ...

14     int N = this->rows; // Get the matrix order.

16     // Make a copy of the original matrix, in order not to leave it in a messy
17     // state in the case of a singular matrix detection during the process.
18     Matrix* original = new Matrix(this);

20     Matrix* inverseMatrix = new Matrix(N, N);    // Create the inverse matrix.
21     Matrix* rightHandMatrix = new Matrix(N, N); // Create the right-hand side (RHS)
22     rightHandMatrix->setIdentity();              // matrix and set it as identity.

24     // Create the Nx1 matrix which holds each row's scale factor.
25     Matrix* scaleFactor = new Matrix(N, 1);

27     // Create the rows indexer, for bookkeeping the rows swapping while pivoting.
28     int rowIndex[N];

30     // The row, column and k, working variables, respectively.
31     int row, col, k;
32     int determinantSign = 1; // Initializes the signal of the determinant (positive).

34     // For each row, set the scale factor (the largest number of the row).
35     for (row = 0; row < N; row++) {
36         rowIndex[row] = row;    // Use the iteration loop to initialize the row index.
37         double scaleMax = 0.0;
38         for (col = 0; col < N; col++)
39             scaleMax = (scaleMax > fabs(original->at(row, col)) ?
40                         scaleMax : fabs(original->at(row, col)));

42         scaleFactor->set(row, 0, scaleMax);
43     }

45     // Loop over rows k = [0, ..., N-1-1]
46     for (k = 0; k < N - 1; k++) {
47         double maxRatio = 0.0;
48         int pivotRow = k;

50         // Find the pivot row (partial pivoting).
51         // The pivot row is the row whose element at the current column (k)
52         // divided by the largest element of its row has the largest ratio.
53         for (row = k; row < N; row++) {
54             double ratio = fabs(original->at(rowIndex[row], k) /
55                                 scaleFactor->at(rowIndex[row], 0));
56             if (ratio > maxRatio) {
57                 maxRatio = ratio;
58                 pivotRow = row;
59             }
60         }

62 // Define the minimum error accepted for a non-singular matrix.
63 #define kMinimumError 1.0e-12

65         if (fabs(maxRatio) < kMinimumError)
66             // Report an error and return 0.0 ...
```

Quadro 4 – Método `invert` da classe `Matrix`, parte 1

```
67          // Swap the row which contains the pivot, if it isn't the current row (k).
68          int pivotRowIndex = rowIndex[k];
69          if (pivotRow != k) {
70              pivotRowIndex = rowIndex[pivotRow];
71              rowIndex[pivotRow] = rowIndex[k];
72              rowIndex[k] = pivotRowIndex;

74              determinantSign *= -1; // Flip sign of determinant.
75          }

77          // Perform forward elimination in all the augmented matrix.
78          for (row = k + 1; row < N; row++) {
79              double coefficient = original->at(rowIndex[row], k) /
80                                                  original->at(pivotRowIndex, k);

82              // Do the pivoting in the original matrix.
83              for (col = k + 1; col < N; col++)
84                  original->set(rowIndex[row], col, (original->at(rowIndex[row], col)
85                                  - coefficient * original->at(pivotRowIndex, col)));
86              original->set(rowIndex[row], k, coefficient);

88              // Do the pivoting in the RHS matrix.
89              for (col = 0; col < N; col++)
90                  rightHandMatrix->set(rowIndex[row], col,
91                      rightHandMatrix->at(rowIndex[row], col) - coefficient *
92                                          rightHandMatrix->at(pivotRowIndex, col));
93          }
94      }

96      // Compute determinant as product of diagonal elements.
97      double determinant = determinantSign;
98      for (row = 0; row < N; row++)
99          determinant *= original->at(rowIndex[row], row);

101     // Verify, again, if this matrix is not singular.
102     if (fabs(original->at(rowIndex[N - 1], N - 1)) < kMinimumError)
103         // Report an error and return 0.0 ...

105     // Perform back-substitution.
106     for (k = 0; k < N; k++) {
107         inverseMatrix->set(N - 1, k, rightHandMatrix->at(rowIndex[N - 1], k) /
108                                      original->at(rowIndex[N - 1], N - 1));

110         for (row = N - 2; row >= 0; row--) {
111             double sum = rightHandMatrix->at(rowIndex[row], k);
112             for (col = row + 1; col < N; col++)
113                 sum -= original->at(rowIndex[row], col) * inverseMatrix->at(col, k);

115             inverseMatrix->set(row, k, sum / original->at(rowIndex[row], row));
116         }
117     }

119     // Transfer the inverse matrix data to this matrix.
120     for (row = 0; row < this->rows; row++)
121         for (col = 0; col < this->columns; col++)
122             this->data[col * this->rows + row] = inverseMatrix->at(row, col);

124     // delete working matrices ...

126     return determinant;
127 }
```

Quadro 5 – Método `invert` da classe `Matrix`, parte 2

Analisando o código fonte do quadro 4 percebe-se que, primeiramente, é feita a consistência para garantir que a matriz a ser invertida é uma matriz quadrada e em seguida são declaradas as estruturas auxiliares para realizar a inversão desta matriz.

De forma a efetuar o pivotamento parcial, na etapa inicial do algoritmo é encon-

trado o maior elemento de cada uma das linhas da matriz `original` (uma cópia da matriz a ser invertida), o qual é armazenado no seu respectivo índice em `scaleFactor` (linhas 35–43).

Em seguida, inicia-se então a iteração pelas linhas $\{0, \ldots, n-2\}$ da matriz aumentada (linhas 53–60) onde, primeiramente, escolhe-se a linha pivô (`pivotRow`) e verifica-se o pivô escolhido não é menor que `kMinimumError` (linhas 63–66), o que classificaria a matriz como singular.

Após determinado o elemento pivô, se necessário, a matriz aumentada tem suas linhas reordenadas (linhas 68–75, no quadro 5) para que o elemento pivô permaneça na diagonal da matriz. Consequentemente, o sinal do determinante é invertido. A ordem relativa das linhas é armazenada na estrutura auxiliar `rowIndex`, de forma que a estrutura de dados interna da matriz não tenha que ser alterada para que seja efetuada a troca de linhas.

Na sequência, é realizada a etapa de eliminação progressiva, efetuando o pivotamento em toda a matriz aumentada e alterando os valores dos elementos da matriz que estão localizados abaixo e à direita do elemento pivô (linhas 78–93).

Após efetuada a eliminação progressiva, calcula-se o valor do determinante da matriz e inicia-se a etapa retro-substituição, utilizando-se dos valores da matriz `original` e da matriz RHS (`rightHandMatrix`) para preencher a matriz inversa (`inverseMatrix`), que é depois atribuída para a matriz cuja inversão foi requisitada. Por fim, o determinante da matriz original é retornado.

3.3.2.2 Resolução de sistemas lineares

Conforme explicado na seção 2.2.4, os sistemas lineares a serem resolvidos durante o processo da estimação da função de transformação possuem, na maioria dos casos, mais equações do que incógnitas e devem ser resolvidos utilizando-se da estimativa dos mínimos quadrados. Em outras palavras, dado um sistema linear $Ax = b$, deve-se encontrar um vetor x tal que Ax esteja o mais próximo de b.

Uma maneira de encontrar o vetor x baseado na estimativa dos mínimos quadrados é através da pseudo-inversa da matriz A, definida por $A^{+}b$. Segundo Datta (1995), a pseudo-inversa (também conhecida como inversa generalizada de Moore-Penrose) da matriz A é definida pela equação:

$$A^{+} = (A^{t}A)^{-1}A^{t} \tag{21}$$

Portanto, para o sistema linear $Ax = b$, a solução única dos mínimos quadrados é dada por:

$$x = (A^t A)^{-1} A^t b = A^+ b \tag{22}$$

O método `solveLinearSystem` da classe `Matrix` implementa esta solução e tem o código fonte apresentado no quadro 6.

```
 1 /**
 2  * Solve the overconstrained linear system Ax = b using the Least Squares Fit
 3  * with the pseudoinverse approach, where: x = (A^T A)^-1 A^T b.
 4  * @param <code>A</code> The MxN matrix with the coefficients.
 5  * @param <code>x</code> The 1xN matrix with the unknowns.
 6  * @param <code>b</code> The 1xM matrix with the constant terms.
 7  * @return <code>true</code> if the overconstrained linear system was sucessfully
 8  * solved, otherwise returns <code>false</code>.
 9  */
10 bool Matrix::solveLinearSystem(Matrix* A, Matrix* x, Matrix* b) {
11   // Verify if the arguments' dimensions represents an overconstrained
12   // linear system, otherwise returns false.

14   // Calculate the A transpose.
15   Matrix* At = new Matrix(A);
16   At->transpose();

18   // Calculate the At.A (left multiply matrix A transposed by the matrix A).
19   Matrix* AtA = new Matrix(At);
20   AtA->multiply(A);

22   // Verify if the At.A is an invertible matrix, if not, the system cannot be solved.
23   if (!AtA->invert()) return false;

25   // Multiply the At.A inverted by At.
26   Matrix *AtAiAt = new Matrix(AtA);
27   AtAiAt->multiply(At);

29   // Multiply the (At.A)i.At by b.
30   Matrix *AtAiAtb = new Matrix (AtAiAt);
31   AtAiAtb->multiply(b);

33   // Store the result in x (unknowns matrix).
34   for (int index = 0; index < AtAiAtb->getRows(); index++)
35     x->set(index,0,AtAiAtb->at(index,0));

37   // Free allocated memory... and return.
38   return true;
39 }
```

Quadro 6 – Método `solveLinearSystem` da classe `Matrix`

Este método recebe como parâmetro três instâncias da classe `Matrix` que representam o sistema linear a ser resolvido e consistem, respectivamente, na matriz $n \times m$ dos coeficientes (`A`), na matriz coluna das incógnitas (`x`) e na matriz coluna dos termos independentes (`b`).

Primeiramente, verifica-se se os parâmetros informados representam um sistema linear sobredeterminado. Se sim, são efetuadas as operações definidas na equação (22) e é obtida a solução do sistema linear, que é armazenada na matriz `x`.

3.3.2.3 Estimação das homografias

Tomando como base o sistema linear $Ax = b$ (ou então $Ma = b$) demonstrado na equação (14), o cálculo das homografias que relacionam pares de imagens é realizado no método `estimate` da classe `Homography`, cujo código fonte é apresentado no quadro 7.

```
/**
 * Estimates the forward and inverse homographies represented by the set
 * of correspondence points.
 * @return <code>true</code> if both inverse and forward homographies
 * were successfully estimated, otherwise returns <code>false</code>.
 */
bool Homography::estimate() {
    int numberOfEquations = this->correspondencePoints->size() * 2;
    Matrix *M = new Matrix(numberOfEquations, 8);
    Matrix *a = new Matrix(8, 1);
    Matrix *b = new Matrix(numberOfEquations, 1);

    for (int index = 0; index < numberOfEquations / 2; index++) {
        // Retrieve the n-th correspondence point.
        CorrespondencePoint *corrPoint = this->correspondencePoints->at(index);
        Point2D aTargetPoint = corrPoint->getTarget();
        Point2D aReferencePoint = corrPoint->getReference();

        M->set(index * 2, 0, aTargetPoint.x);
        M->set(index * 2, 1, aTargetPoint.y);
        M->set(index * 2, 2, 1);
        M->set(index * 2, 3, 0);
        M->set(index * 2, 4, 0);
        M->set(index * 2, 5, 0);
        M->set(index * 2, 6, -(aTargetPoint.x * aReferencePoint.x));
        M->set(index * 2, 7, -(aTargetPoint.y * aReferencePoint.x));
        b->set(index * 2, 0, aReferencePoint.x);
        M->set(index * 2 + 1, 0, 0);
        M->set(index * 2 + 1, 1, 0);
        M->set(index * 2 + 1, 2, 0);
        M->set(index * 2 + 1, 3, aTargetPoint.x);
        M->set(index * 2 + 1, 4, aTargetPoint.y);
        M->set(index * 2 + 1, 5, 1);
        M->set(index * 2 + 1, 6, -(aTargetPoint.x * aReferencePoint.y));
        M->set(index * 2 + 1, 7, -(aTargetPoint.y * aReferencePoint.y));
        b->set(index * 2 + 1, 0, aReferencePoint.y);
    }

    if (!Matrix::solveLinearSystem(M, a, b)) {
        // Report error ...
        return false;
    }

    // Populate the homography matrix attribute.
    this->homography = new Matrix(3, 3);
    this->homography->set(0, 0, a->at(0, 0));
    this->homography->set(0, 1, a->at(1, 0));
    this->homography->set(0, 2, a->at(2, 0));
    this->homography->set(1, 0, a->at(3, 0));
    this->homography->set(1, 1, a->at(4, 0));
    this->homography->set(1, 2, a->at(5, 0));
    this->homography->set(2, 0, a->at(6, 0));
    this->homography->set(2, 1, a->at(7, 0));
    this->homography->set(2, 2, 1);

    // Free allocated memory ...
    // Try to estimate the inverse homography using the adjoint matrix approach.
    return this->estimateInverse(InverseProjectiveMappingInferenceTypeAdjointMatrix);
}
```

Quadro 7 – Método `estimate` da classe `Homography`

Primeiramente são declaradas as matrizes dos coeficientes (`M`), das incógnitas (`a`) e dos termos independentes (`b`) que constituem o sistema linear. Na sequência, tomando como base os valores dos PCs correspondentes (atributo `correspondencePoints`), o sistema linear é preenchido e é então resolvido através do método `solveLinearSystem`. Caso três PCs identificados em uma mesma imagem forem colineares o sistema não pode ser resolvido e, portanto, nenhuma homografia pode ser estimada.

Por fim, com a matriz `a` devidamente preenchida, seus valores são transferidos para o atributo `homography` e pode-se então estimar a homografia inversa, invocando o método `estimateInverse` da mesma classe.

O método `estimateInverse` (quadro 8) permite efetuar a estimação da homografia inversa utilizando-se de três abordagens diferentes definidas pelo parâmetro `type`, as quais podem ser:

a) `InverseProjectiveMappingInferenceTypeInverseMatrix`: obtém a matriz de homografia inversa através da inversão da matriz de homografia direta. Esta abordagem não pode ser aplicada em todos os casos visto que podem existir situações onde a matriz de homografia direta é singular;

b) `InverseProjectiveMappingInferenceTypeAdjointMatrix`: sendo a abordagem *default* implementada, permite que seja obtida a matriz de homografia inversa com base na matriz adjunta da matriz de homografia direta, conforme demonstrado na equação (15);

c) `InverseProjectiveMappingInferenceTypePlaneReversing`: obtém a matriz de homografia inversa de forma idêntica à estimação da homografia direta, porém, invertendo a ordem dos PCs correspondentes. Esta abordagem requer a resolução de um sistema linear e por isso não foi estabelecida como *default*.

```
 1 /**
 2  * Estimates the current homography's inverse using the specified inferring type.
 3  * @param <code>type</code> The type of the inverse projective mapping inferring
 4  * to be used.
 5  * @return <code>true</code> if the current homography's inverse was successfully
 6  * estimated, otherwise returns <code>false</code>.
 7  */
 8 bool Homography::estimateInverse(InverseProjectiveMappingInferenceType type) {

10     if (type == InverseProjectiveMappingInferenceTypeInverseMatrix) {
11         this->inverseHomography = new Matrix(this->homography);
12         if (!this->inverseHomography->invert()) {
13             // Report error ...
14             return false;
15         }
16         return true;
17     }

19     if (type == InverseProjectiveMappingInferenceTypeAdjointMatrix) {
20         this->inverseHomography = new Matrix(3, 3);

22         // Retrieve the forward homography matrix elements.
23         double A = homography->at(0, 0);
24         double B = homography->at(0, 1);
25         double C = homography->at(0, 2);
26         double D = homography->at(1, 0);
27         double E = homography->at(1, 1);
28         double F = homography->at(1, 2);
29         double G = homography->at(2, 0);
30         double H = homography->at(2, 1);
31         double I = homography->at(2, 2);

33         // Calculate the inverse homography matrix elements.
34         double A_h = E*I - F*H;
35         double B_h = C*H - B*I;
36         double C_h = B*F - C*E;
37         double D_h = F*G - D*I;
38         double E_h = A*I - C*G;
39         double F_h = C*D - A*F;
40         double G_h = D*H - E*G;
41         double H_h = B*G - A*H;
42         double I_h = A*E - B*D;

44         // Populate the inverse homography matrix attribute.
45         this->inverseHomography->set(0, 0, A_h);
46         this->inverseHomography->set(0, 1, B_h);
47         this->inverseHomography->set(0, 2, C_h);
48         this->inverseHomography->set(1, 0, D_h);
49         this->inverseHomography->set(1, 1, E_h);
50         this->inverseHomography->set(1, 2, F_h);
51         this->inverseHomography->set(2, 0, G_h);
52         this->inverseHomography->set(2, 1, H_h);
53         this->inverseHomography->set(2, 2, I_h);

55         return true;
56     }

58     if (type == InverseProjectiveMappingInferenceTypePlaneReversing) {
59         // The same as the forward homography estimation but reversing
60         // the plane-to-plane mapping.
61     }

63     return false;
64 }
```

Quadro 8 – Método `estimateInverse` da classe `Homography`

3.3.2.4 Transformação de pontos

Uma vez estimadas as homografias direta e inversa, pode ser realizada a operação de transformação de pontos (representados em coordenadas homogêneas) através do método `transformPoint` (quadro 9).

```
 1 /**
 2  * Applies a projective transformation to the specified point accordingly to the
 3  * specified direction.
 4  * @param <code>point</code> The point in which the transformation will the applied.
 5  * @param <code>direction</code> The direction of the transformation to be applied.
 6  * @return The new point after being transformed.
 7  */
 8 Point2D Homography::transformPoint(Point2D point,
 9     TransformationDirection direction) {

11     Vector *pointVector = new Vector(3);
12     pointVector->set(0, point.x);
13     pointVector->set(1, point.y);
14     pointVector->set(2, 1);

16     switch (direction) {
17     case TransformationDirectionTargetToReference:
18         this->homography->multiply(pointVector);
19         break;

21     case TransformationDirectionReferenceToTarget:
22         this->inverseHomography->multiply(pointVector);
23         break;
24     }

26     Point2D transformedPoint;
27     transformedPoint.x = pointVector->at(0) / pointVector->at(2);
28     transformedPoint.y = pointVector->at(1) / pointVector->at(2);

30     // Free allocated memory ...

32     return transformedPoint;
33 }
```

Quadro 9 – Método `transformPoint` da classe `Homography`

Este método recebe um ponto de imagem (`Point2D`) e encapsula-o em um 3-vetor (`Vector`) juntamente com a sua coordenada homogênea, cujo valor inicial é 1. Em seguida, o vetor é multiplicado por uma matriz de homografia, a qual é determinada com base na direção da transformação, podendo ser:

a) `TransformationDirectionTargetToReference`: transformação de um ponto da imagem de ajuste para a imagem de referência (homografia direta);
b) `TransformationDirectionReferenceToTarget`: transformação de um ponto da imagem de referência para a imagem de ajuste (homografia inversa).

Ao final da operação, obtém-se novamente o ponto em sua forma não-homogênea normalizando-o (dividindo suas coordenadas 2D pela coordenada homogênea), de forma que possa ser utilizado para referenciar um ponto de imagem novamente.

3.3.2.5 Criação do mosaico de imagens

O método `createMosaic` (quadro 10) da classe `Mosaic` é o responsável por gerenciar o processo de criação do mosaico de imagens, requisitando as operações mais fundamentais conforme necessário.

Os possíveis valores de retorno do método são auto-explicativos e têm por objetivo informar à camada de visualização do software o *status* da operação realizada. Caso o usuário cancele esta operação, todo o processo é interrompido e `MosaicCreationStatusCanceled` é retornado.

```
1 /**
2  * Creates the mosaic with the given reference and target images, which are
3  * related by the set of correspondence points.
4  * @return The status of the current attempt of mosaic creation.
5  */
6 MosaicCreationStatus Mosaic::createMosaic() {

8     // Estimate the homography (and it's inverse).
9     this->homography = new Homography(&correspondencePoints);
10    if (!this->homography->estimate())
11        return MosaicCreationStatusColinearCorrespondencePoints;

13    // Calculate the mosaiced image bounds.
14    Rect mosaicedImageBounds = this->calculateMosaicedImageBoundingBox();

16    // Calculate the mosaic image dimensions.
17    int imageWidth = ROUND(mosaicedImageBounds.xMax - mosaicedImageBounds.xMin + 1);
18    int imageHeight = ROUND(mosaicedImageBounds.yMax - mosaicedImageBounds.yMin + 1);

20    // Allocate data for the final mosaic image.
21    imImage *rawMosaicImage =
22        imImageCreate( imageWidth, imageHeight,
23                       this->targetImage->getImage()->color_space,
24                       this->targetImage->getImage()->data_type);
25    IMImage *mosaicImage = new IMImage (rawMosaicImage, mosaicedImageBounds);

27    // Warp the target image and blend it with the reference image.
28    this->warpAndMergeImages(mosaicImage, this->referenceImage, this->targetImage);
29    this->mosaicedImage = mosaicImage;

31    if (this->mosaicCreationActive == false)
32        return MosaicCreationStatusCanceled;

34    return MosaicCreationStatusCompleted;
35 }
```

Quadro 10 – Método `createMosaic` da classe `Mosaic`

Como pode ser visto no quadro 10, primeiramente tenta-se estimar as homografias (direta e inversa) com base nos PCs correspondentes. Caso estas tenham sido estimadas com sucesso, obtém-se o tamanho final do mosaico de imagens através da chamada do método `calculateMosaicedImageBoundingBox`, cujo pseudocódigo é apresentado no quadro 11.

```
1 Posicione a imagem de referência e a imagem de ajuste no centro do
  mosaico (transformação de renderização para uma superfície de projeção
  planar);
2 dimensaoDoMosaicoFinal ← área ocupada pela imagem de referência;
3 dimensaoDaImagemDeAjuste ← área ocupada pela imagem de ajuste;
4 para cada ponto p das extremidades de dimensaoDaImagemDeAjuste
  faça
5 |  transforme p utilizando a homografia direta;
6 se dimensaoDaImagemDeAjuste > dimensaoDoMosaicoFinal então
7 |  dimensaoDoMosaicoFinal ← dimensaoDaImagemDeAjuste;
8 retorne dimensaoDoMosaicoFinal;
```

Quadro 11 – Pseudo-código para o método `calculateMosaicedImageBoundingBox` da classe `Mosaic`

Voltando ao método `createMosaic`, após alocada memória para a imagem do mosaico final, é invocado o método `warpAndMergeImages`, o qual é baseado no algoritmo apresentado no quadro 2. Este método que efetua a transformação e reamostragem da imagem de ajuste, bem como a mesclagem desta com a imagem de referência, compondo o mosaico de imagens, efetivamente.

3.3.2.6 Transformação, reamostragem e mesclagem de imagens

O método `warpAndMergeImages` realiza a operação mais importante da criação do mosaico de imagens que é a de transformar a imagem de ajuste e mesclá-la com a imagem de referência, de forma não que sejam visíveis as emendas na área de transição entre estas imagens.

Este método deve também informar o progresso da operação sendo realizada à camada de visualização do software (utilizando-se do padrão de projeto *Observer*). O código fonte do método `warpAndMergeImages` é apresentado de forma resumida no quadro 12 e no quadro 13.

```
 1 /**
 2  * Warps the target image taking into account the estimated homographies and
 3  * merges both the target image and the reference image into the final mosaic.
 4  * @param <code>finalMosaicImage</code> The final mosaic image, result of the
 5  * warping/merging operations in the target and reference images.
 6  * @param <code>referenceImage</code> The reference image to be added to the
 7  * final mosaic image, it remains unchanged into the final mosaic image.
 8  * @param <code>targetImage</code> The target image to be added to the final
 9  * mosaic image, it is warped and projected into the final mosaic image.
10  */
11 void Mosaic::warpAndMergeImages (model::IMImage *finalMosaicImage,
12                                  model::IMImage *referenceImage,
13                                  model::IMImage *targetImage) {
```

Quadro 12 – Método `warpAndMergeImages` da classe `Mosaic`, parte 1

```
14      // Calculate the total number of interations in order to estimate the
15      // mosaicing operation progress ...
16
17      // Perform a scanline iteration through the mosaic image bounds from the
18      // bottom-left corner
19      for (int x = ROUND(boundingBox.xMin); x <= ROUND(boundingBox.xMax); x++) {
20          for (int y = ROUND(boundingBox.yMin); y <= ROUND(boundingBox.yMax); y++) {
21              // Notify the user interface about the mosaicing operation progress ...
22
23              Point2D mosaicImagePoint = { x, y };
24              Point2D transformedPoint;
25              // Retrieve the pixel value located at the reference image.
26              referenceImageRgbValue =
27                  referenceImage->getAt(  mosaicImagePoint.x,mosaicImagePoint.y,
28                                          InterpolationModeNone);
29
30              // Transform the point using the inverse homography.
31              transformedPoint = this->homography->transformPoint(mosaicImagePoint,
32                                          TransformationDirectionReferenceToTarget);
33
34              // Retrieve the interpolated pixel value located at the target image.
35              warpedTargetImageRgbValue = targetImage->getAt(transformedPoint.x,
36                                  transformedPoint.y, InterpolationModeBilinear);
37              // ...
38              // Verify if the transformed pixel coordinate lies inside the
39              // input images boundary.
40              if (referenceImageRgbValue != kPixelOutOfBounds &&
41                  warpedTargetImageRgbValue != kPixelOutOfBounds) {
42
43                  // Get the weight associated with each input pixel
44                  referencePointWeight = referenceImage->getWeight(mosaicImagePoint);
45                  warpedTargetPointWeight = targetImage->getWeight(transformedPoint);
46
47                  // Compute the averaged weight for each input pixel
48                  // (feathered blending).
49                  mosaicPointWeight = referencePointWeight + warpedTargetPointWeight;
50                  referencePointWeight /= mosaicPointWeight;
51                  warpedTargetPointWeight /= mosaicPointWeight;
52
53                  referenceRGB_R = IM_GET_RGB_R(referenceImageRgbValue);
54                  referenceRGB_G = IM_GET_RGB_G(referenceImageRgbValue);
55                  referenceRGB_B = IM_GET_RGB_B(referenceImageRgbValue);
56                  warpedTargetRGB_R = IM_GET_RGB_R(warpedTargetImageRgbValue);
57                  warpedTargetRGB_G = IM_GET_RGB_G(warpedTargetImageRgbValue);
58                  warpedTargetRGB_B = IM_GET_RGB_B(warpedTargetImageRgbValue);
59
60                  // Compute the weighted average at each output pixel and write it out
61                  outputRGB_R = (unsigned char) (referenceRGB_R*referencePointWeight +
62                                          warpedTargetRGB_R*warpedTargetPointWeight);
63                  outputRGB_G = (unsigned char) (referenceRGB_G*referencePointWeight +
64                                          warpedTargetRGB_G*warpedTargetPointWeight);
65                  outputRGB_B = (unsigned char) (referenceRGB_B*referencePointWeight +
66                                          warpedTargetRGB_B*warpedTargetPointWeight);
67
68                  outputImageRgbValue = 0x0;
69                  // ------------------------------------------------- Macro expansion
70                  IM_SET_RGB_R(outputImageRgbValue,outputRGB_R);// ((rgb) |= (r) << 16)
71                  IM_SET_RGB_G(outputImageRgbValue,outputRGB_G);// ((rgb) |= (g) << 8)
72                  IM_SET_RGB_B(outputImageRgbValue,outputRGB_B);// ((rgb) |= (b))
73                  outputImage->setAt(x,y,outputImageRgbValue);
74              } else {
75                  if (referenceImageRgbValue != kPixelOutOfBounds) {
76                      outputImage->setAt(x,y,referenceImageRgbValue);
77                  }
78                  if (warpedTargetImageRgbValue != kPixelOutOfBounds) {
79                      outputImage->setAt(x,y,warpedTargetImageRgbValue);
80                  }
81              }
82          }
83      }
84  }
```

Quadro 13 – Método `warpAndMergeImages` da classe `Mosaic`, parte 2

O *loop* externo que engloba o método consiste em percorrer cada ponto de coordenada (*pixel*) da imagem final do mosaico, denominado `mosaicImagePoint`, a partir de sua extremidade inferior-esquerda, até que toda sua área (`boundingBox`) seja percorrida. Estes pontos estão representados no sistema de coordenadas do mosaico, cuja origem encontra-se no centro do mesmo.

Cada ponto é arredondado através da macro `ROUND`[19], a qual utiliza a técnica de *Asymmetric Arithmetic Rounding* (*Round-Half-Up*). Isto se deve ao fato de o sistema de coordenadas do mosaico permitir pontos de coordenadas com valores negativos.

Na sequência é obtido o valor do componente RGB (através do método `getAt`) referente ao ponto `mosaicImagePoint` nas imagens de entrada . No caso da imagem de referência, que é simplesmente inserida dentro do mosaico final (sem distorções), este valor é obtido sem a necessidade de uma função de interpolação.

Para a imagem de ajuste, o ponto é transformado (mapeamento inverso) e o valor do componente RGB é obtido utilizando-se de uma técnica de interpolação (apresentada na seção 3.3.2.7).

Caso o ponto `mosaicImagePoint` corresponda à um ponto dentro de somente uma das imagens de entrada, simplesmente atribui-se o valor do componente RGB da imagem em questão para o *pixel* da imagem final.

No entanto, caso o ponto `mosaicImagePoint` corresponder à pontos localizados dentro das duas imagens de entrada, é efetuada a média ponderada dos valores de seus respectivos componentes RGB[20] e é então atribuído para o *pixel* da imagem final (linhas 44–73).

3.3.2.7 Função de interpolação

Conforme explicado na seção 2.3.3, a transformação de pontos de coordenadas entre as imagens pode produzir pontos com valores de coordenadas não-inteiros, fazendo-se necessária a utilização de uma função de interpolação para obter-se um valor aproximado para o *pixel* em questão.

A técnica implementada foi a interpolação bilinear, conforme especificada pela equação (20), porém otimizada de forma que sejam efetuadas apenas três ao invés de oito multiplicações.

Desta forma, o valor de intensidade $I(x+p, y+q)$ bilinearmente interpolado é dado

[19] A macro `ROUND` é dada por: `#define ROUND(x) (int) floor(x + 0.5)`.

[20] A média é calculada de forma independentemente em cada um dos canais do componente RGB.

por:

$$\begin{aligned} I(p,y) &= I(x,y) + p * (I(x+1,y) - I(x,y)) \\ I(p,y+1) &= I(x,y+1) + p * (I(x+1,y+1) - I(x,y+1)) \\ I(x+p,y+q) &= I(p,y) + q * (I(p,y+1) - I(p,y)) \end{aligned} \tag{23}$$

O método `getAt` da classe `IMImage` (*namespace* `model`) é o responsável por retornar o valor RGB referente à um ponto de coordenada relativo (no sistema de coordenadas do mosaico de imagens). Este ponto é representado pelos parâmetros `x` e `y` e deve, primeiramente, ser convertido para o sistema de coordenadas da imagem (0,0,`width`−1,`height`−1).

Outro parâmetro informado ao método `getAt` é o método de interpolação a ser executado (`InterpolationModeNone` ou `InterpolationModeBilinear`). O pseudocódigo para o método `getAt` é apresentado no quadro 14.

Entrada:
`x` (double): Coordenada x no sistema de coordenadas do mosaico
`y` (double): Coordenada y no sistema de coordenadas do mosaico
`interpolationMode` (InterpolationMode): Modo de interpolação a ser utilizado para calcular o valor RGB na posição especificada

```
1 Transforme as coordenadas x e y para o sistemas de coordenadas da
  imagem;
2 se as coordenadas x e y transformadas estiverem fora dos limites da
  imagem então
3     retorne kPixelOutOfBounds;
4 selecione interpolationMode faça
5     caso InterpolationModeNone
6         retorne o valor RGB referente à posição x e y truncadas na
          imagem;
7     caso InterpolationModeBilinear
8         retorne o valor RGB referente à posição I(x+p, y+q) conforme
          a equação (23);
```

Quadro 14 – Pseudo-código para o método `getAt` da classe `model::IMImage`

3.3.2.8 Desenho dos pontos de correspondência

Com relação à camada de visualização do software, é demonstrado no quadro 15 o método `drawCorrespondencePoint`, o qual utiliza das funções das bibliotecas CD e IM para desenhar um par de PCs correspondentes.

```
 1 /**
 2  * Draws a CorrespondencePoint object on the cdCanvas with a line connecting both
 3  * the reference and target CorrespondencePointMark objects.
 4  * @param <code>correspondencePoint</code> The correspondence point object
 5  * to be drawn.
 6  */
 7 void IMDialog::drawCorrespondencePoint(CorrespondencePoint *correspondencePoint) {
 8     bool isSelected = correspondencePoint->getTargetMark()->isSelected();
 9     model::Point2D targetCenter =
10         correspondencePoint->getTargetMark()->getCenter();

12     model::Point2D referenceCenter =
13         correspondencePoint->getReferenceMark()->getCenter();
14     cdCanvasMarkSize(this->cd_Canvas, kCorrespondencePointMarkSize);

16     // Get the offset between the two correspondence point marks
17     // (in world coordinates).
18     model::Point2D targetReferenceOffset =
19         model::Utils::getOffsetBetweenPoints(targetCenter, referenceCenter);

21     // Calculate the length of the line between both target and reference
22     // correspondence point marks.
23     double lineLength = sqrt(   targetReferenceOffset.x*targetReferenceOffset.x +
24                                 targetReferenceOffset.y*targetReferenceOffset.y);

26     // Calculate the correspondence point mark size (the diameter).
27     double worldXOrigin, worldXCorrPointSize;
28     wdCanvasCanvas2World(this->cd_Canvas, 0, 0, &worldXOrigin, NULL);
29     wdCanvasCanvas2World(this->cd_Canvas, kCorrespondencePointMarkSize, 0,
30                                           &worldXCorrPointSize, NULL);
31     double markSizeMM = worldXCorrPointSize - worldXOrigin;

33     // Calculates the parametric line factor to be seek in order no to draw the line
34     // inside the circle (1.0 = no line drawn, 0.0 = full line drawn).
35     double parametricLineFactor = 1.0 - (markSizeMM / 2.0) / lineLength;

37     if (isSelected) {
38        // Draw the correspondence points with the CD_YELLOW color.
39     } else {
40        // Draw the correspondence points using the predefined colors.
41        cdForeground(CD_GREEN);
42        wdCanvasLine(this->cd_Canvas,
43            referenceCenter.x + targetReferenceOffset.x * parametricLineFactor,
44            referenceCenter.y + targetReferenceOffset.y * parametricLineFactor,
45               targetCenter.x - targetReferenceOffset.x * parametricLineFactor,
46               targetCenter.y - targetReferenceOffset.y * parametricLineFactor);

48        cdCanvasMarkType(this->cd_Canvas, CD_HOLLOW_CIRCLE);

50        cdForeground(CD_RED);
51        wdCanvasMark(this->cd_Canvas, targetCenter.x, targetCenter.y);
52        cdForeground(CD_BLUE);
53        wdCanvasMark(this->cd_Canvas, referenceCenter.x, referenceCenter.y);

55        wdCanvasPixel(this->cd_Canvas, targetCenter.x, targetCenter.y, CD_RED);
56        wdCanvasPixel(this->cd_Canvas, referenceCenter.x, referenceCenter.y, CD_BLUE);
57     }
58 }
```

Quadro 15 – Método `drawCorrespondencePoint` da classe `IMDialog`

Este método primeiramente obtém a posição dos PCs correspondentes (`targetCenter` e `referenceCenter`), calcula a distância euclidiana entre os mesmos (`lineLength`) e o diâmetro do círculo que representa cada ponto de controle (`markSizeMM`).

Em seguida, através da equação paramétrica da reta, calcula-se qual será a extensão

da reta que liga os pares de PCs correspondentes, de forma que esta não sobreponha o círculo representativo de cada um.

Por fim, o par de PCs correspondentes é desenhado no *canvas* (`cd_Canvas`) com as cores vermelho, verde e azul, ou então, na cor amarela caso o par de PCs esteja selecionado, conforme pode ser visto na fig. 37.

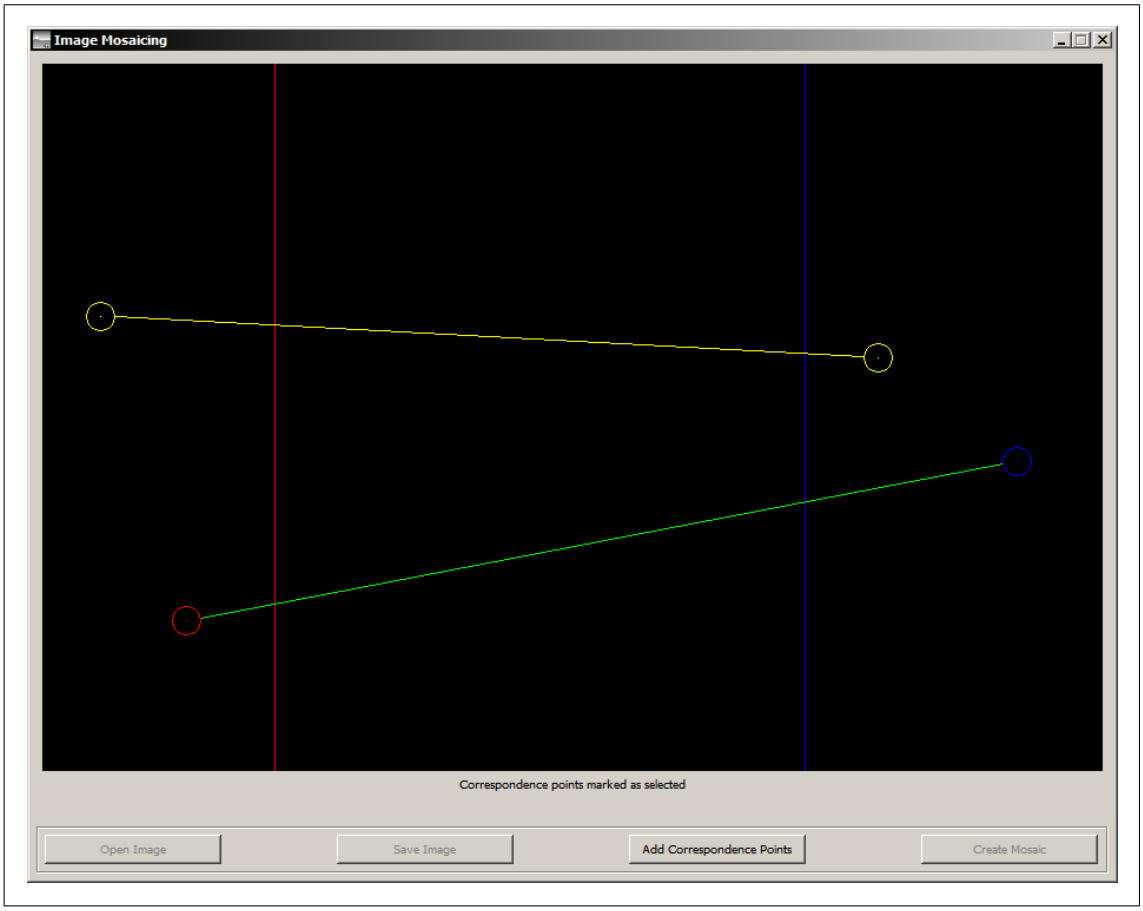


Figura 37 – Desenho de pares de PCs correspondentes selecionados (em amarelo) e não selecionados (em vermelho, verde e azul)

3.4 OPERACIONALIDADE

Esta seção tem por objetivo mostrar, em nível de usuário, a operacionalidade do software desenvolvido. Para isto, primeiramente é apresentada a sua interface gráfica e explicados seus principais componentes.

Em seguida, é apresentado um estudo de caso que exemplifica a construção de um mosaico composto por três imagens, de forma que sejam demonstradas as principais funcionalidades do software.

3.4.1 Interface gráfica do software e principais componentes

Ao executar o software, sua interface gráfica é carregada e apresentada ao usuário, permitindo ao mesmo efetuar a criação de mosaicos de imagens de forma interativa. Esta interface é apresentada de forma esquemática na fig. 38, onde são explicitados os principais componentes nela existentes.

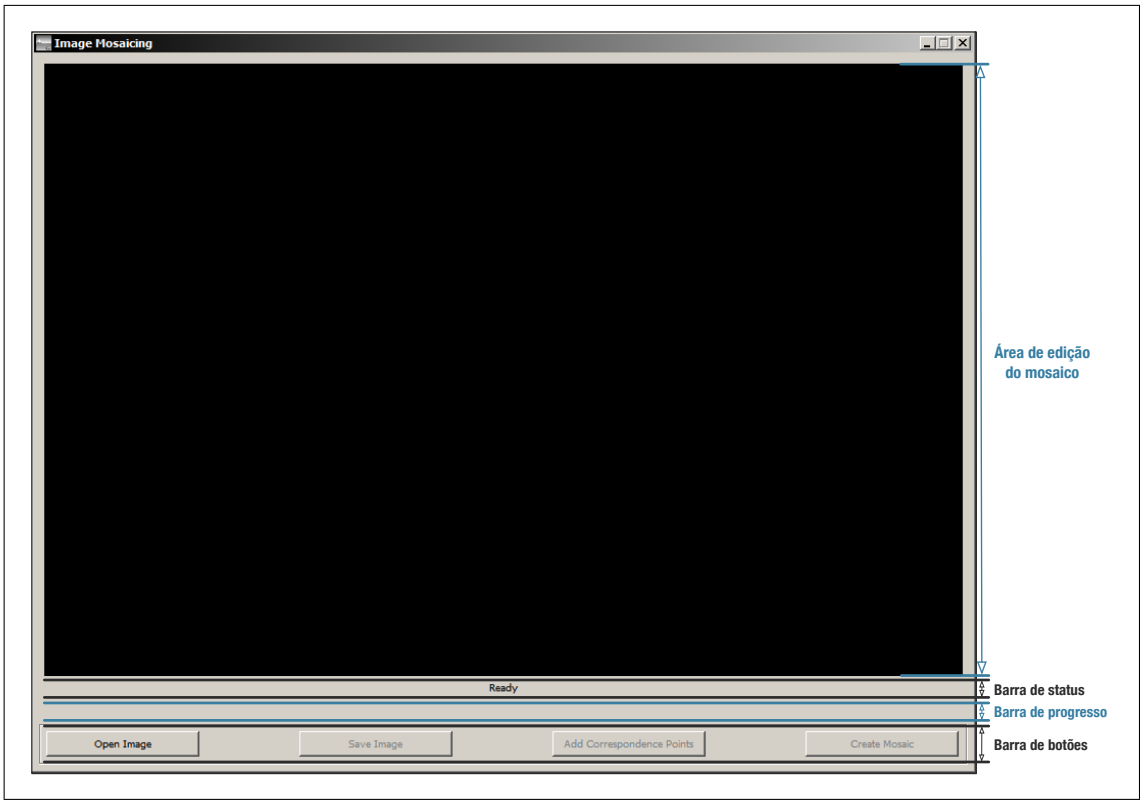


Figura 38 – Interface gráfica do software e seus principais componentes

A área de edição do mosaico tem por objetivo permitir ao usuário visualizar as imagens que serão utilizadas para a construção do mosaico e realizar as operações necessárias para a construção do mesmo, as quais são: definição das imagens de ajuste e de referência; seleção/remoção de imagens e posicionamento dos PCs correspondentes. É nesta área também, que o mosaico construído é apresentado.

Imediatamente abaixo da área de edição, encontra-se a barra de *status*, que tem por objetivo indicar as operações sendo executadas tanto pelo software quando pelo usuário. Logo abaixo encontra-se a barra de progresso (inicialmente não visível), cuja função é exibir ao usuário o progresso da operação de criação do mosaico.

Por fim, na parte inferior da interface, estão localizados os botões que permitem, respectivamente: abrir uma imagem; salvar uma imagem (selecionada); adicionar um par de PCs correspondentes entre as imagens e requisitar a criação do mosaico de imagens.

3.4.2 Criação de um mosaico de imagens

Como pode ser visto na fig. 38, inicialmente a única operação disponível ao usuário é a de abrir uma imagem de entrada (fig. 39), através do botão `Open Image`, para que na sequência, seja possível iniciar a edição do mosaico.

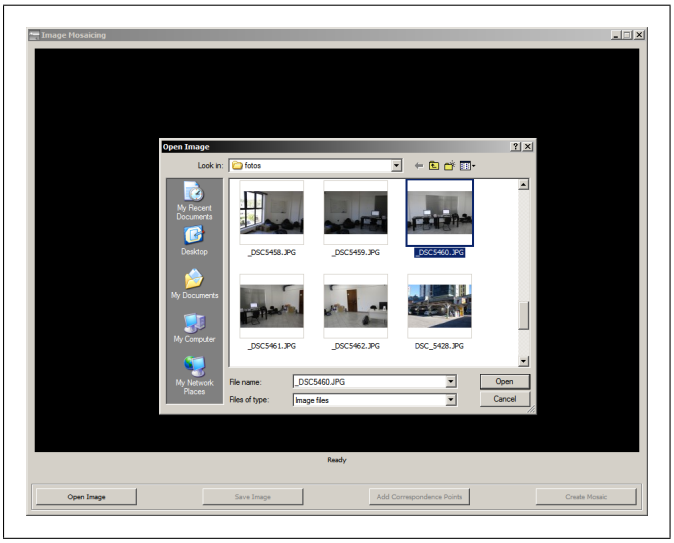


Figura 39 – Abertura de uma imagem de entrada

Após aberta uma imagem, esta é exibida na área de edição do mosaico. Visto que mosaico de imagens é construído a partir de pares de imagens, podem ser carregadas, no máximo, duas imagens simultaneamente.

As operações na área de edição do mosaico são realizadas utilizando-se o *mouse* em conjunto algumas teclas de controle, conforme explicado de forma mais detalhada no quadro 16.

Comando			
Tecla	Botão	Clique	**Operação**
ctrl	*mouse1*	duplo	Definir imagem de ajuste
ctrl	*mouse3*	duplo	Definir imagem de referência
-	*mouse2*	simples	Selecionar imagem/PCs
ctrl	*mouse1*	simples	Mover imagem/PCs
shift	*mouse1*	simples	*Pan*
-	*mouse wheel up*	-	*Zoom In* (fator 0.2)
-	*mouse wheel down*	-	*Zoom Out* (fator 0.2)
delete	-	-	Deletar imagem/PCs (selecionados)

Quadro 16 – Operações da área de edição do mosaico

Através das operações citadas, o usuário pode interativamente definir as imagens de ajuste (destacada em vermelho) e de referência (destacada em azul) e então posicionar

PCs correspondentes conforme desejado, como pode ser visto na fig. 40.

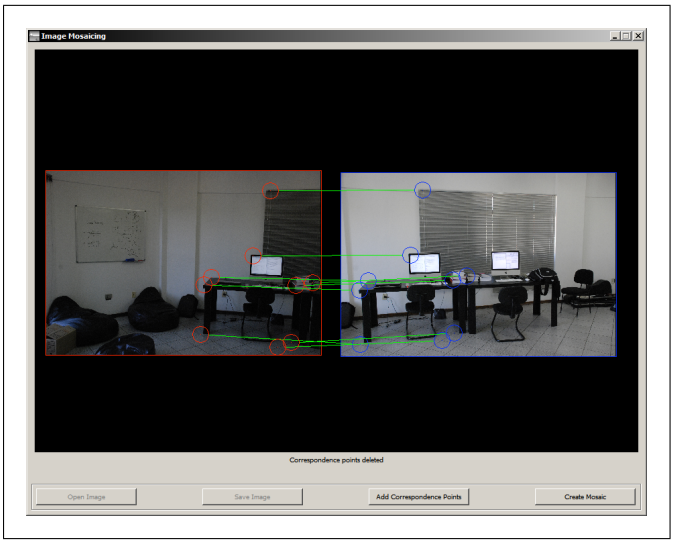


Figura 40 – Imagens de ajuste e de referência definidas e PCs correspondentes posicionados

Após adicionar um par de PCs correspondentes (através do botão `Add Correspondence Points`), utilizando a operação de *Zoom In*, o usuário pode definir estes pontos nas imagens com precisão de *subpixels*, permitindo a estimação mais acurada das homografias que relacionam o par de imagens em questão.

Analisando a fig. 40, verifica-se que após definidos quatro ou mais PCs correspondentes, é possível requisitar a criação do mosaico (botão `Create Mosaic`), cujo processo é demonstrado na fig. 41.

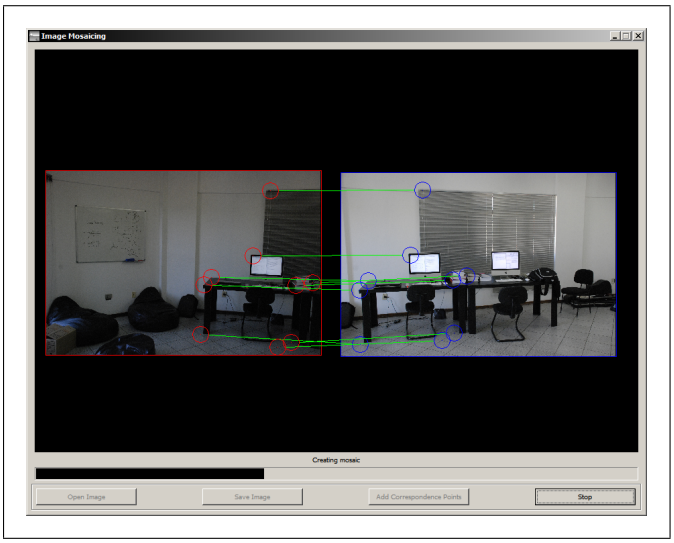


Figura 41 – Processo de criação do mosaico de imagens

O usuário pode cancelar a operação de criação do mosaico sendo realizada através do botão `Stop`. Depois de criado, o mosaico de imagens é apresentado na área de edição. Esta situação é demonstrada na fig. 42.

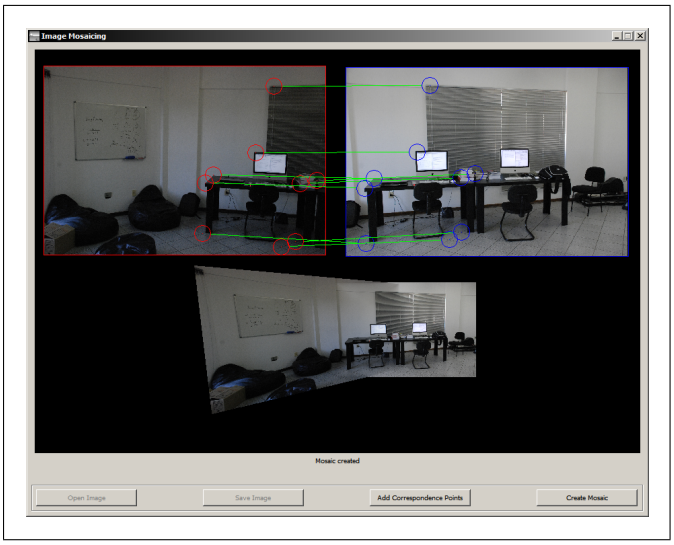


Figura 42 – Mosaico criado apresentado na área de edição

Visto que o processo de criação de mosaicos de imagens é incremental e é realizado a partir da composição de pares de imagens, o usuário pode prosseguir com a construção do mosaico apagando as imagens de entrada previamente fornecidas e abrindo outra imagem, repetindo o processo realizado anteriormente. Este ciclo deve ser realizado até que o mosaico final desejado seja construído (fig. 43) e, uma vez que é possível deletar as imagens contidas dentro da área de edição, reparos podem ser feitos quando necessário.

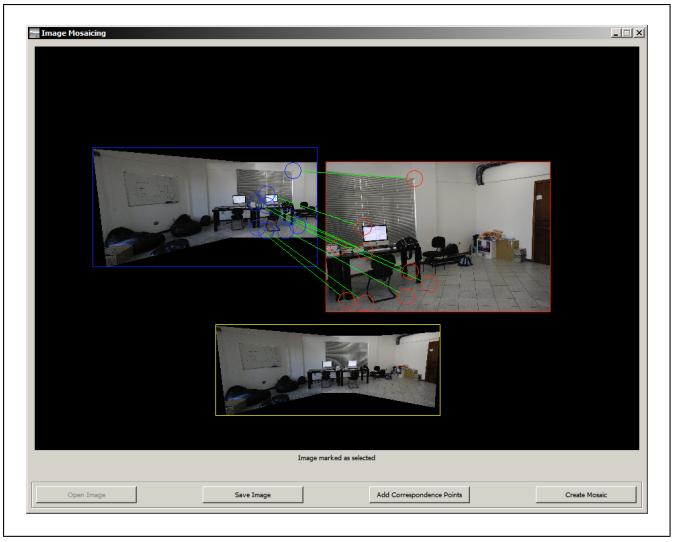


Figura 43 – Imagem do mosaico final criado selecionada

Por fim, o usuário pode selecionar a imagem desejada (destacada em amarelo na fig. 43) e, através do botão `Save Image`, salvar esta em disco, nos formatos de imagem mais populares, tais como: PNG, JPG, JPEG, BMP, GIF e TIFF.

3.5 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou um estudo referente ao processo de criação de mosaicos de imagens, mais especificamente mosaicos planares criados a partir de imagens obtidas por câmeras fotográficas não calibradas.

Primeiramente foi discutida a problemática do registro de imagens, etapa essencial para a construção de mosaicos de imagens. Nesta etapa, inicialmente devem ser feitas as operações de detecção e casamento de feições nas imagens de entrada. No software desenvolvido, estas operações são realizadas manualmente com o auxílio do usuário, diferentemente dos trabalhos correlatos, os quais utilizaram de bibliotecas previamente desenvolvidas para efetuar o processo de registro de imagens automaticamente.

O registro de imagens automático apresenta como vantagem o fato de o usuário não precisar intervir no processo, porém, conforme apresentado nas seções 2.1.2.1 e 2.1.2.2, é dependente do contexto da cena (tipo de cena e condições de luminosidade) que irá ser composta na forma de um mosaico. Desta forma, em determinadas situações, o registro de imagens pode não ser realizado de forma precisa, comprometendo o resultado final do mosaico.

Heikkilä e Pietikäinen (2005) têm como foco a montagem de mosaicos de imagens a partir vídeos obtidos por câmeras de vigilância. Neste tipo de aplicação, o número de imagens que irão compor o mosaico é geralmente grande e, portanto, exige-se o uso de algum algoritmo que efetue as operações iniciais do registro de imagens automaticamente.

O trabalho proposto por Bagli (2007) visa a implementação de diferentes técnicas de mesclagem de imagens (etapas posteriores do processo de construção de mosaicos de imagens) e, desta forma, Bagli (2007) também optou por efetuar o registro de imagens automaticamente, utilizando bibliotecas já existentes.

Quanto à escolha do modelo matemático para modelar as transformações entre as imagens, embora algumas aplicações efetuem o mapeamento entre as imagens através de transformações afins (tipicamente aplicações de mosaicagem de fotos aéreas), o uso de transformações projetivas mostrou-se o mais adequado para o trabalho em questão, visto que este objetiva a construção de mosaicos de imagens de cenas arbitrárias, obtidas por câmeras fotográficas convencionais.

Para a transformação e reamostragem das imagens, a implementação da técnica de interpolação bilinear apresentou bons resultados e a um custo computacional viável, apresentando uma boa relação custo/benefício, conforme descrito na literatura. A fig. 44 exemplifica a criação de um mosaico de imagens com e sem a utilização desta técnica, ressaltando as diferenças obtidas no mosaico final.

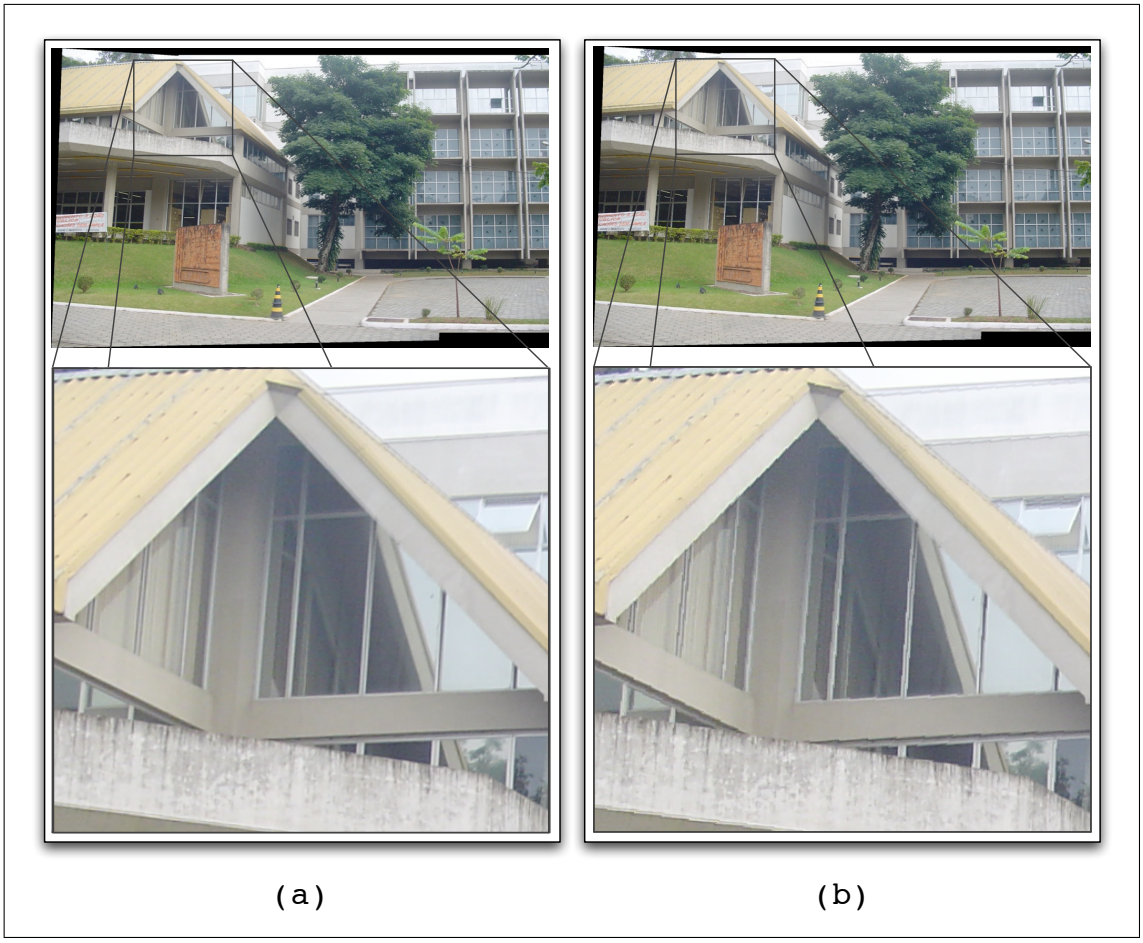


Figura 44 – (a) mosaico de imagens criado utilizando a técnica de interpolação bilinear e (b) mosaico de imagens criado sem utilizar nenhuma técnica de interpolação, é possível perceber o *aliasing* linhas diagonais

Com relação à escolha da superfície de projeção do mosaico, optou-se por utilizar o plano (superfície de projeção planar) visto que este apresenta como resultados mosaicos de imagens mais realistas. Isto se deve ao fato de as distorções perspectivas presentes na cena reconstruída serem preservadas.

No entanto, a superfície de projeção planar apresenta algumas limitações. Como pode ser visto na fig. 45, nos casos onde a cena a ser reconstruída apresenta um ângulo de abertura superior a 90°, o mosaico de imagens começa a apresentar distorções severas à medida que a imagens de entrada são projetadas em suas extremidades. Para estes casos, sugere-se ou a escolha de outra superfície de projeção ou então a utilização de técnicas de câmeras virtuais, conforme apresentado em Capel (2001).

A fig. 45 também demonstra que a escolha da imagem de referência (neste caso, situada na extremidade direita do mosaico) é fator importante para que se obtenha um bom resultado final. Neste caso, a escolha de uma imagem situada mais próxima à região central do mosaico resultaria em um mosaico final com menos distorções.

Figura 45 – Mosaico de imagens representando uma cena cujo ângulo de abertura é superior a 90°. A má escolha da imagem de referência favoreceu o aparecimento de distorções na extremidade esquerda do mosaico

Para a mesclagem das imagens em suas regiões de sobreposição, a utilização da média ponderada – com peso máximo no centro da imagem, decaindo em direção as suas extremidades – mostrou-se bastante satisfatória para imagens obtidas sob condições controladas (sem grandes variações de luminosidade e obtidas por câmeras fotográficas utilizando o mesmo tempo de exposição). A fig. 46 (a) apresenta um mosaico onde emenda entre as imagens que o compõe foi suavizada, demonstrando a eficácia da técnica implementada.

Figura 46 – (a) Mosaico de imagens utilizando a média ponderada para suavização da região de transição entre as imagens e (b) mosaico de imagens sem suavização da região de transição entre as imagens, as emendas são visíveis

A fig. 46 (b) apresenta o mesmo mosaico de imagens, porém com a técnica de mesclagem implementada suprimida. Nota-se que esta técnica também ajuda a suavizar os erros de registro de imagens.

Um caso onde a técnica de mesclagem implementada não obteve bons resultados é apresentado na fig. 47, onde o mosaico final não deu a impressão de que a cena é formada por uma única imagem. Neste mosaico, as imagens de entrada foram editadas de forma a ter diferenças de luminosidade bastante acentuadas. Também é possível notar a presença de objetos em movimento durante a aquisição das imagens de entrada, o que acabou causando na aparição de objetos "fantasmas" no mosaico resultante. Embora não seja muito comum a obtenção de imagens nestas condições, técnicas, tais como a de registro fotométrico, apresentam soluções para tais problemas e são apontadas em Capel (2001).

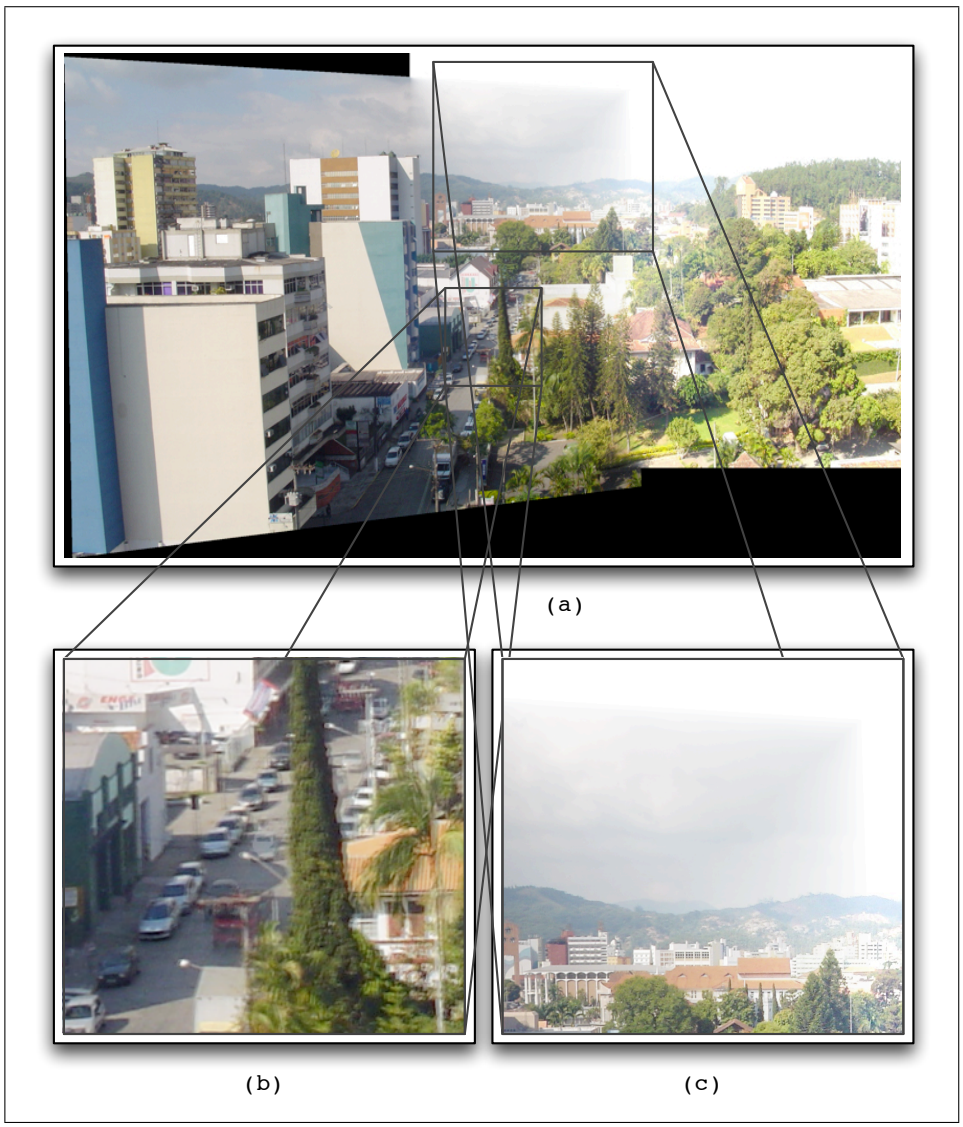


Figura 47 – (a) mosaico com mesclagem das imagens ineficiente; (b) objetos em movimento causando o efeito "fantasma" e (c) diferença de luminosidade acentuada entre imagens de entrada prejudicou a qualidade do resultado final

Outros mosaicos de imagens podem ser vistos nas figuras 48 e 49. O primeiro apresenta uma situação onde a cena a ser reconstruída é uma superfície planar e, portanto, as fotos foram obtidas com movimentações de câmera arbitrárias. Este mosaico foi composto sem utilizar de uma técnica de mesclagem, desta forma, a área de transição entre as imagens de entrada é perceptível. O segundo mosaico apresenta, como nos demais mosaicos apresentados nesta seção, uma situação onde as imagens que compõem a cena são obtidas por uma câmera que é rotacionada em torno de seu centro óptico.

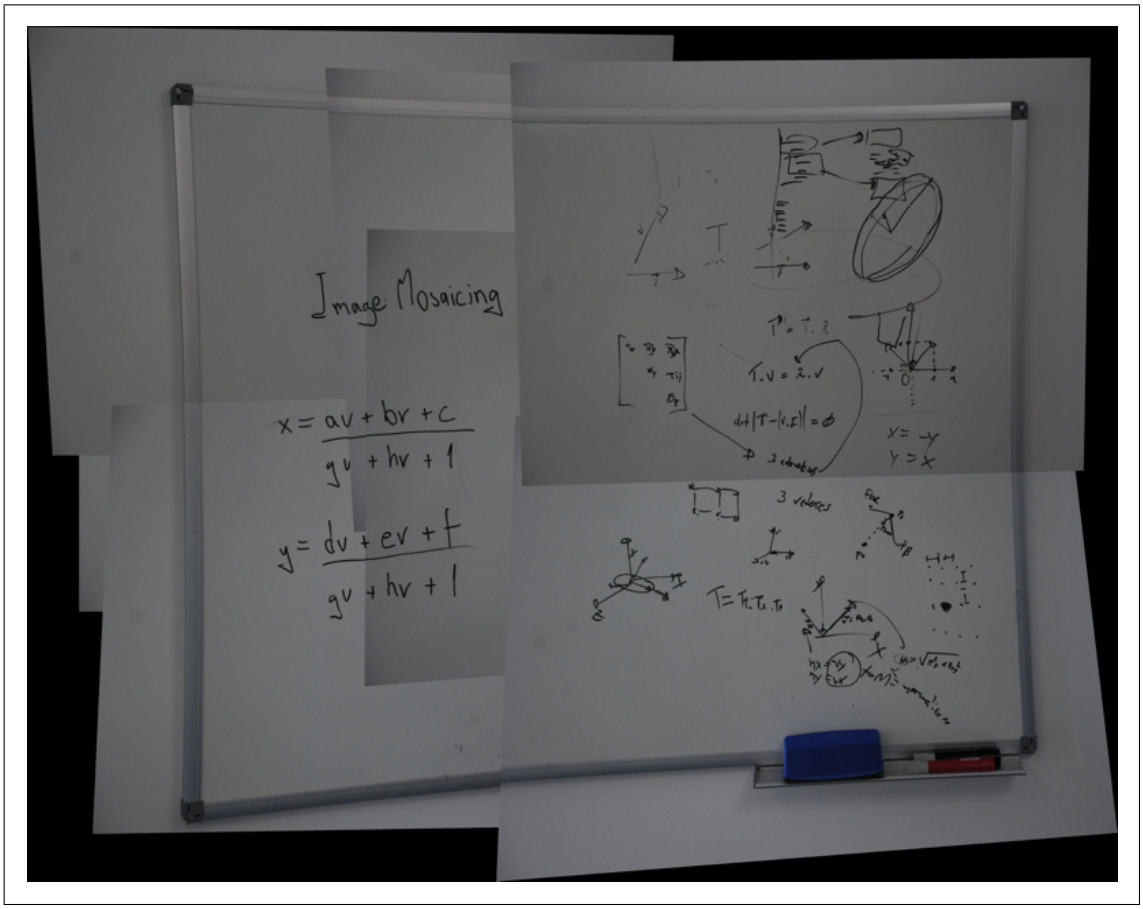


Figura 48 – Mosaico de imagens representando uma cena planar, as imagens são obtidas com movimentações de câmera arbitrárias. É possível perceber a emenda entre as imagens, visto que a técnica de mesclagem foi suprimida

Com relação às tecnologias utilizadas para a implementação do software, optou-se por utilizar a linguagem C++ por esta ser amplamente utilizada pela comunidade científica, devido ao ótimo desempenho oferecido para aplicações que exigem de grande processamento, o que é típico nas áreas de computação gráfica e visão computacional.

As bibliotecas utilizadas para a construção da interface gráfica software forneceram todos os recursos necessários para garantir a usabilidade do mesmo, além de apresentarem boa documentação. A decisão de implementar todo o processo de registro e construção de mosaicos de imagens, sem o auxílio de bibliotecas de visão computacional e processamento

de imagens já existentes, permitiu um estudo mais aprofundado do processo de construção de mosaicos de imagens e justificou as técnicas a abordagens escolhidas para tal.

Figura 49 – Mosaico de imagens composto por cinco fotografias obtidas por uma câmera que é rotacionada em torno de seu centro óptico

Embora o tempo necessário para a criação de mosaicos de imagens seja dependente tanto da resolução das imagens de entrada, quanto da característica da cena em questão, constatou-se, através dos testes realizados, que o software desenvolvido permite efetuar a criação de mosaicos de imagens em tempo satisfatório sem necessitar de grandes recursos computacionais.

O quadro 17 apresenta as características referentes aos mosaicos de imagens apresentados nesta seção, sendo estas: número de imagens de entrada; resolução das imagens de entrada; resolução da imagem final resultante e tempo total de criação do mosaico (em milissegundos)[21]. Os testes foram executados em um computador com processador Intel Pentium M 2.0 GHz e com 1 GB de memória RAM. As fotos foram obtidas com uma câmera NIKON D40X e uma câmera SONY DSC-P52.

[21] O tempo total de criação do mosaico é a soma dos tempos de composição de cada par de imagens, até que o mosaico final seja obtido.

Figura	Número de imagens de entrada	Resolução das imagens de entrada	Resolução do mosaico	Tempo de criação (ms)
fig. 44 (a)	2	1632×1224	2193×1321	11.777
fig. 44 (b)	2	1632×1224	2193×1321	10.725
fig. 45	5	1632×1224	11404×3712	236.421
fig. 46 (a)	2	1632×1224	2172×1297	11.456
fig. 46 (b)	2	1632×1224	2172×1297	10.185
fig. 47	2	1632×1224	2780×1489	14.901
fig. 48	7	3872×2592	6813×5340	629.165
fig. 49	5	1632×1224	3206×2271	76.970

Quadro 17 – Características dos mosaicos de imagens criados através do software desenvolvido

4 CONCLUSÕES

Este trabalho investigou como utilizar a informação contida em múltiplas imagens, as quais representam visões parciais de uma cena e que se sobrepõem, para gerar uma única imagem mais abrangente, denominada mosaico. Para tal, foi desenvolvido um software que permite a criação de mosaicos de imagens de forma interativa e incremental.

Um ponto chave do trabalho desenvolvido é permitir que sejam criados mosaicos de imagens de cenas arbitrárias, sem a necessidade de se utilizar câmeras fotográficas calibradas, bem como dispor de grandes recursos computacionais.

Para efetuar o registro das imagens, a escolha da função de transformação projetiva, em conjunto com a técnica de interpolação bilinear, mostrou-se a mais adequada para o contexto do trabalho, apresentando bons resultados e a um custo computacional reduzido. Tal fato pode ser comprovado através dos resultados obtidos e justifica a utilização de tais técnicas pela maioria dos trabalhos relacionados a mosaicos de imagens presentes na literatura, os quais encontram-se no estado da arte dentro desta área de estudo.

Uma limitação da técnica de registro de imagens adotada é a necessidade da intervenção do usuário durante as etapas de detecção e casamento de feições nas imagens de entrada. Embora isto seja necessário em determinadas situações, técnicas de registro de imagens automático têm sido estudadas recentemente pela comunidade científica e poderiam ser agregadas ao software desenvolvido, facilitando ainda mais a construção dos mosaicos de imagens, porém limitando o escopo das cenas a serem representadas. A implementação de um método automático de registro de imagens, conforme apresentado nas seções 2.1.2.1 e 2.1.2.2, envolve a utilização de diversas técnicas de processamento de imagens e, desta forma, necessita de um estudo específico nesta área.

Após registradas as imagens, a mesclagem das mesmas através da média ponderada propiciou uma suavização nas regiões de transição entre as imagens e, na maioria dos casos, conseguiu dar a impressão de que toda a cena é composta por uma única imagem, atendendo ao propósito de um mosaico de imagens. Nos casos onde as diferenças radiométricas entre as imagens de entrada são muito acentuadas, constatou-se a necessidade da implementação de técnicas de mesclagem mais avançadas, conforme apresentado em Bagli (2007) e Capel (2001).

Com relação às técnicas e tecnologias utilizadas para o desenvolvimento do software, a linguagem de programação C++, em conjunto com as bibliotecas IUP (TECGRAF,

2009c), CD (TECGRAF, 2009a) e IM (TECGRAF, 2009b) atenderam as expectativas do trabalho, permitindo que todos os requisitos elicitados fossem atendidos e que as operações executadas pelo software fossem realizadas em tempo satisfatório. A utilização do padrão de arquitetura MVC e o padrão de projeto *Observer* (GAMMA et al., 1994) possibilitou a construção de uma arquitetura de software separada em camadas com fraco acoplamento. Isto permite que novas funcionalidades venham a ser agregadas ao software em trabalhos futuros.

As técnicas envolvidas na construção de mosaicos de imagens formam a base tecnológica de inúmeras aplicações nas áreas de computação gráfica e visão computacional. O ato de aumentar o campo de visão de uma câmera fotográfica não é trivial e exige a aplicação de algoritmos de registro e de mesclagem de imagens. Estes algoritmos vêm sendo constantemente estudados pela comunidade científica e, consequentemente, podem ser incorporados ao trabalho desenvolvido, de forma que seja possível obter-se resultados ainda mais realistas.

4.1 TRABALHOS FUTUROS

A grande área de aplicabilidade de mosaicos de imagens, bem como o número de técnicas envolvidas em sua construção, abre perspectivas para futuros trabalhos. Tais trabalhos poderiam agregar novas funcionalidades ou aperfeiçoar as já existentes no presente trabalho. Uma relação de possíveis extensões para o trabalho desenvolvido é apresentada a seguir:

a) automatizar as etapas iniciais do registro de imagens através da implementação de métodos de extração e casamento de feições nas imagens de entrada. Conforme explicado nas seções 2.1.2.1 e 2.1.2.2, a escolha do método a ser implementado deverá levar em conta as características das cenas que serão representadas na forma de um mosaico;

b) implementar diferentes técnicas de interpolação nos *pixels* das imagens de entrada durante a etapa de transformação e reamostragem de imagens, a fim de estabelecer uma correlação entre os resultados obtidos e o custo computacional envolvido. A escolha da técnica de interpolação a ser utilizada é uma tarefa importante quando as imagens de entrada começam a ter dimensões muito grandes ou quando as distorções aplicadas nas imagens de entrada são muito acentuadas;

c) implementar a composição de mosaicos de imagens em uma superfície de projeção cilíndrica. Esta funcionalidade poderia ser implementada reaproveitando as demais funções já desenvolvidas no software e permitiria a construção de mosaicos com ângulos de abertura mais amplos. A escolha da superfície de

projeção a ser adotada poderia ficar a critério do usuário, o qual teria a liberdade de escolhê-la com base nas características do mosaico que o mesmo deseja obter;

d) aprimorar a técnica de mesclagem de imagens, tornando o software menos sensível às diferenças de luminosidade existentes nas imagens de entrada. Exemplos de tais técnicas são a de registro fotométrico ou de definição da linha de corte, como apresentado em Capel (2001) e Bagli (2007), respectivamente. Outras técnicas de mesclagem de imagens mais dependentes do contexto da cena a ser representada em forma de mosaico são apresentadas em Szeliski (2006).

REFERÊNCIAS BIBLIOGRÁFICAS

BAGLI, V. V. **Mosaico de imagens baseado em múltiplas resoluções**. 2007. 111 f. Dissertação (Mestrado em Computação Aplicada) — Curso de Pós-Graduação em Computação Aplicada, Instituto Nacional de Pesquisas Espaciais, São José dos Campos.

BROWN, L. G. A survey of image registration techniques. **ACM Computing Surveys**, New York, v. 24, n. 4, p. 325–376, dez. 1992.

CANNY, J. A computational approach to edge detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Washington, v. 8, n. 6, p. 679–698, nov. 1986.

CAPEL, D. P. **Image mosaicing and super-resolution**. 2001. 263 f. PhD Thesis (Doctor of Philosophy) — Department of Engineering Science, University of Oxford, Oxford.

CHEN, S. E. Quicktime VR: an image-based approach to virtual environment navigation. In: SPECIAL INTEREST GROUP ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES (SIGGRAPH), 22., 1995, New York. **Proceedings...** New York: Association for Computing Machinery, 1995. p. 29–38.

CONCI, A.; AZEVEDO, E.; LETA, F. R. **Computação gráfica**: teoria e prática. Rio de Janeiro: Elsevier, 2008.

DATTA, B. D. **Numerical linear algebra and applications**. Pacific Grove: Brooks/Cole Publishing, 1995.

ECLIPSE. **Eclipse CDT - C/C++ Development Tooling**. Version 5.0. [S.l.], 2009a. Disponível em: <http://www.eclipse.org/cdt/>. Acesso em: 8 jan. 2009.

_____. **Eclipse Ganymede**. Version 3.4.1. [S.l.], 2009b. Disponível em: <http://www-.eclipse.org/ganymede/>. Acesso em: 8 jan. 2009.

ESTRADA, F. J.; JEPSON, A. D.; FLEET, D. **Planar homographies**. [S.l.], 2005. Disponível em: <http://www.cs.utoronto.ca/~strider/vis-notes/tutHomography04.pdf>. Acesso em: 18 abr. 2009.

FARIAS, P. C. M. de A. **Desenvolvimento de uma metodologia de visualização tridimensional aplicada a estereoradiografia**. 2006. 105 f. Tese (Doutorado em Engenharia Nuclear) — Programa de Pós-Graduação em Engenharia Nuclear, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

FEDOROV, D. **Sistema semi-automático de registro e mosaico de imagens**. 2002. 150 f. Dissertação (Mestrado em Computação Aplicada) — Curso de Pós-Graduação em Computação Aplicada, Instituto Nacional de Pesquisas Espaciais, São José dos Campos.

GAMMA, E. et al. **Design patterns**: elements of reusable object-oriented software. Reading: Addison-Wesley, 1994.

GOMES, J.; VELHO, L. **Fundamentos da computação gráfica**. Rio de Janeiro: IMPA, 2003.

HARTLEY, R.; ZISSERMAN, A. **Multiple view geometry in computer vision**. 2nd. ed. Cambridge: Cambridge University Press, 2004.

HECKBERT, P. S. **Fundamentals of texture mapping and image warping**. 1989. 87 f. Dissertation (Master's Thesis in Computer Science and Engineering) — Electrical Engineering and Computer Sciences Department, University of California, Berkeley.

HEESCH, D. V. **Doxygen**. Version 1.5.8. [S.l.], 2009. Disponível em: <http://www-.doxygen.org/>. Acesso em: 9 jan. 2009.

HEIKKILÄ, M.; PIETIKÄINEN, M. An image mosaicing module for wide-area surveillance. In: INTERNATIONAL WORKSHOP ON VIDEO SURVEILLANCE AND SENSOR NETWORKS, 3., 2005, Hilton. **Proceedings...** New York: Association for Computing Machinery, 2005. p. 11–18.

HELLIER, P.; BARILLOT, C. Coupling dense and landmark-based approaches for nonrigid registration. **IEEE Transactions on Medical Imaging**, [S.l.], v. 22, n. 2, p. 217–227, fev. 2003.

HOUSE, D. H. **Image warps**. [S.l.], 2008. Disponível em: <http://www.cs-.clemson.edu/~dhouse/courses/405/notes/image-warps.pdf>. Acesso em: 15 maio 2009.

MALLICK, S. P. **Feature based image mosaicing**. Technical report of the Department of Electrical and Computer Engineering, University of California. San Diego, dez. 2002. 6 p.

MARR, D.; HILDRETH, E. Theory of edge detection. **Proceedings of the Royal Society of London. Series B**, London, v. 207, n. 1167, p. 187–217, fev. 1980.

MCLAUCHLAN, P. F.; JAENICKE, A. Image mosaicing using sequential bundle adjustment. **Image and Vision Computing**, [S.l.], v. 20, n. 9–10, p. 751–759, ago. 2002.

MINGW. **Minimalist GNU for Windows**. Version 5.1.4. [S.l.], 2009. Disponível em: <http://www.mingw.org/>. Acesso em: 9 jan. 2009.

OLIVEIRA, M. M. Image-based modeling and rendering techniques: a survey. **Revista de Informática Teórica e Aplicada**, [S.l.], v. 9, n. 2, p. 37–66, out. 2002.

OMG. **UML - Unified Modeling Language specification**. Version 2.0. [S.l.], 2005. Disponível em: <http://www.uml.org/>. Acesso em: 12 fev. 2009.

PAL, N. R.; PAL, S. K. A review on image segmentation techniques. **Pattern Recognition Letters**, [S.l.], v. 26, n. 9, p. 1277–1294, mar. 1993.

PELEG, S.; HERMAN, J. Panoramic mosaics with videobrush. In: DARPA IMAGE UNDERSTANDING WORKSHOP, 25., 1997, New Orleans. **Proceedings...** New Orleans: Morgan Kaufmann, 1997. p. 261–264.

PRESS, W. H. et al. **Numerical recipes in C**: the art of scientific computing. Cambridge: Cambridge University Press, 1992.

ROHR, K. **Landmark-based image analysis**: using geometric and intensity models. Dordrecht: Springer, 2001.

SHUM, H.; SZELISKI, R. **Panoramic image mosaics**. Microsoft Research technical report MST–TR–97–23. Redmond, 1997. 50 p.

SZELISKI, R. Image alignment and stitching: a tutorial. **Foundations and Trends in Computer Graphics and Computer Vision**, Hanover, v. 2, n. 1, p. 1–104, dez. 2006.

SZENBERG, F. **Acompanhamento de cenas com calibração automática de câmeras**. 2001. 150 f. Tese (Doutorado em Ciências em Informática) — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

TECGRAF. **CD - Canvas Draw, a 2D graphics library**. Version 5.1.1. Rio de Janeiro, 2009a. Disponível em: <http://www.tecgraf.puc-rio.br/cd/>. Acesso em: 5 jan. 2009.

_____. **IM - image representation, storage, capture and processing**. Version 3.4. Rio de Janeiro, 2009b. Disponível em: <http://www.tecgraf.puc-rio.br/im/>. Acesso em: 5 jan. 2009.

_____. **IUP - Portable User Interface**. Version 3.0. Rio de Janeiro, 2009c. Disponível em: <http://www.tecgraf.puc-rio.br/iup/>. Acesso em: 5 jan. 2009.

THE OMNI GROUP. **OmniGraffle**. [S.l.], 2009. Disponível em: <http://www-.omnigroup.com/applications/OmniGraffle/>. Acesso em: 16 jan. 2009.

VIEIRA, P.; FERNANDES, R. **Mapeamento projetivo e filtragem EWA**. [Rio de Janeiro], 2007. Disponível em: <http://www.inf.puc-rio.br/˜pvieira/fcg/T2/T2.pdf>. Acesso em: 8 maio 2009.

YE, G. **Image registration and super-resolution mosaicing**. 2005. 223 f. PhD Thesis (Doctor of Philosophy) — School of Information Technology and Electrical Engineering, University of New South Wales, Sydney.

ZITOVA, B.; FLUSSER, J. Image registration methods: a survey. **Image and Vision Computing**, [S.l.], v. 21, n. 11, p. 977–1000, out. 2003.