UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”
PROGRAMA DE PÓS-GRADUAÇÃO EM TELEVISÃO DIGITAL

Sérgio Carlos Portari Júnior

UM SISTEMA PARA EXTRAÇÃO AUTOMÁTICA DE KEYFRAMES A PARTIR DE FLUXOS DE VÍDEO DIRECIONADO À RECONSTRUÇÃO TRIDIMENSIONAL DE CENÁRIOS VIRTUAIS

BAURU, SP
2013

Sérgio Carlos Portari Júnior

UM SISTEMA PARA EXTRAÇÃO AUTOMÁTICA DE KEYFRAMES A PARTIR DE FLUXOS DE VÍDEO DIRECIONADO À RECONSTRUÇÃO TRIDIMENSIONAL DE CENÁRIOS VIRTUAIS

Exame para conclusão de Mestrado apresentado ao Programa de Pós-Graduação em Televisão Digital (PPGTVD), da Faculdade de Arquitetura, Artes e Comunicação (FAAC), da Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), para obtenção do título de Mestre em Televisão Digital: Informação e Conhecimento sob orientação do Prof. Dr. Antonio Carlos Sementille.

BAURU, SP
2013

Sérgio Carlos Portari Júnior

UM SISTEMA PARA EXTRAÇÃO AUTOMÁTICA DE KEYFRAMES A PARTIR DE FLUXOS DE VÍDEO DIRECIONADO À RECONSTRUÇÃO TRIDIMENSIONAL DE CENÁRIOS VIRTUAIS

Área de Concentração: Tecnologia e Televisão Digital
Linha de Pesquisa: 3 - Inovação tecnológica para televisão digital

Banca Examinadora:

Presidente/Orientador: Prof. Dr. Antonio Carlos Sementille
Instituição: Universidade Estadual Paulista Júlio de Mesquita Filho - UNESP - Bauru

Prof. 1: Prof. Dr. Wilson Massashiro Yonezawa
Instituição: Universidade Estadual Paulista Júlio de Mesquita Filho - UNESP - Bauru

Prof. 2: Prof. Dr. Ildeberto Aparecido Rodello
Instituição: Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto – FEARP/USP – Ribeirão Preto

Resultado: _____

Bauru, _____/_____/_____

Dedico esta dissertação aos meus pais Lilia e Sérgio Portari que não estão mais presentes neste mundo carnal, a minha família, Luciani e Sérgio Neto, pela paciência e compreensão nos momentos em que não estive presente nestes dias, a meu irmão Rodrigo Portari, que me trouxe para este curso, ao meu orientador Dr. Antonio Carlos Sementille e a Deus por permitir a realização de um sonho, que era estudar na UNESP.

AGRADECIMENTOS

Agradeço ao professor Dr. Antonio Carlos Sementille, que não apenas preocupou-se em orientar um aluno, mas em formar um pesquisador e pela paciência de receber-me em sua sala toda semana, cobrando cada vez mais de mim para o sucesso deste trabalho.

Agradeço ao companheiro e amigo Humberto Cecconi, que um ano antes de ingressarmos no programa, já viajávamos para Uberlândia buscando um programa de pós-graduação e que tornou possível a conclusão desta etapa.

Aos professores do PPGTVD, pelo visível prazer em ensinar, não só a mim, mas a todos da turma, pelas dicas e contribuições preciosas aos nossos trabalhos.

Aos colegas de turma, uma das turmas mais unidas da qual eu já fiz parte, que sempre esteve unida durante esses dois últimos anos.

Aos colegas de trabalho e docentes na UEMG, no campus de Frutal, gostaria de externar minha grande satisfação em saber que sempre, quando precisei de algo, pude contar com todo o apoio necessário para que eu pudesse cursar o programa.

A todos do UNESCO-HidroEx, que não mediram esforços para que eu estivesse na equipe, o que permitiu a realização deste curso.

Aos amigos, que direta ou indiretamente participaram deste trabalho, como o Secretário de Estado de Ciência, Tecnologia e Ensino Superior de Minas Gerais, Narcio Rodrigues, Deputado Estadual Zé Maia, Prof. Octávio Elísio, Alexandre Saad, Sheila Paiva, Manoel Musa, Dr. Ronaldo Wilson Santos, Alexandre Cardoso, Rodrigo Portari, Aureliane Oliveira, Estela Portari, Adriano Reis, Leonardo Barcelos, Josney Freitas, Marcelo Murari, Ivan Filho, Jennifer Anne Rosa, o pessoal do Hotel Cidade Bauru, Fernanda Pazini e os companheiros da Eduvale de Avaré, Gustavo Molin e outros que não foram citados por limitação de espaço.

Ainda a Wanda e a Mandy, que por muitos dias e noites estiveram sempre ao meu lado prontas para me ajudarem a relaxar antes de voltar a concentrar no trabalho. Minhas fiéis companheiras caninas.

A Luciani Portari e Sérgio Carlos Portari Neto, por tudo.

PORTARI JR, SÉRGIO C. **Um sistema para extração automática de keyframes a partir de fluxos de vídeo direcionado à reconstrução tridimensional de cenários virtuais**. 82f. Trabalho de Conclusão (Mestrado em TV Digital: Informação e Conhecimento) – FAAC – UNESP, sob orientação do prof. Dr. Antonio Carlos Sementille, Bauru, 2013.

RESUMO

Utilizar um cenário virtual em TV Digital tornou-se comum com o avanço das tecnologias de *hardware* e *software*. Mas para se obter um cenário virtual que convença o telespectador pode-se utilizar reconstruções tridimensionais foto-realísticas como uma possível alternativa. Este trabalho apresenta uma proposta de um método para o pré-processamento de um vídeo, capturado no mundo real, onde são extraídos *frames* adequados à reconstrução 3D (*keyframes*) pelo método SFM (*Structure From Motion*). Desta forma o processamento para a reconstrução 3D utiliza apenas os *frames* considerados essenciais, diminuindo as redundâncias e falhas. Com a utilização deste método conseguiu-se reduzir o tempo de processamento durante a reconstrução 3D. Neste trabalho também comparou-se o método proposto com os métodos tradicionais, onde não existe uma prévia seleção de *keyframes*, utilizando-se diferentes ferramentas de reconstrução baseadas no método SFM.

Palavras-chaves: TV digital, Structure from motion - SFM, Reconstrução 3D, Visão computacional, Estúdio virtual.

PORTARI JR, SÉRGIO C. **Um sistema para extração automática de keyframes a partir de fluxos de vídeo direcionado à reconstrução tridimensional de cenários virtuais**. 79f. Trabalho de Conclusão (Mestrado em TV Digital: Informação e Conhecimento) – FAAC – UNESP, sob orientação do prof. Dr. Antonio Carlos Sementille, Bauru, 2013.

ABSTRACT

Using a virtual scenario on Digital TV became common with the advance of hardware and software technologies. However, to obtain a virtual scenario that persuades the viewer, photo-realistic three-dimensional reconstructions can be used as a possible alternative. This work proposes a method for the pre-processing of a video, captured in the real world, where frames which are appropriate for 3D reconstruction (keyframes) by the method SFM (Structure From Motion) are extracted. Thus the processing for 3D reconstruction use only the frames considered essential, reducing redundancies and gaps. Using this method it was possible to reduce the processing time for the 3D reconstruction. In this work, the proposed method was also compared to traditional methods, where there is no prior selection of keyframes, using different tools of reconstruction based on the SFM method.

Keywords: Digital television, Structure from motion - SFM, 3D reconstruction, Computational vision, Virtual studio.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

2D	Duas dimensões
3D	Três dimensões
AVI	Audio Video Interleave
CMVS	Clustering views for multi-view stereo
CPD	Central de Processamento de Dados
FFT	Fast fourier transform
FTC	Função de transferência de contrastes
HD	High Definition
LUT	Look-up tables
MPEG	Moving picture experts group
NC	Nível de contrastes
PMVS	Patch based multi view stereo
RANSAC	Random sample consensus
RL	Realce linear
SFM	Structure from motion
SIFT	Scale-invariant feature transform
SURF	Speeded up rpbust feature

SUMÁRIO

1. INTRODUÇÃO

A crescente evolução dos *hardwares* gráficos na Computação, desde a década de 1990, possibilitou desenvolvimento de técnicas mais apuradas de produção para TV, antes baseada apenas em uma substituição de cor chave do fundo por uma imagem ou vídeo, em duas dimensões. Como visto em Van DenBergh e Lalioti (1999), esta técnica é conhecida como *chroma-key*.

Utilizando a técnica é possível produzir uma composição de imagens, possibilitando desde a colocação do ator em qualquer lugar do mundo até inserção de objetos que não existem no cenário onde a filmagem é realizada.

O cenário, então, não é apenas "mais um" dos elementos presente em uma produção. Em alguns casos, o cenário pode ser o responsável pelo sucesso ou fracasso dessa produção. Para Millerson e Owens (2009) "O que procuramos em um cenário [...] é um substituto imaginário da realidade, tendo em mente que os diretores produzem ilusões. Entretanto, por mais básico que sejam os materiais no cenário, eles devem aparecer ao público como mais reais possíveis."

Os cenários podem ser muito complexos e exigirem grandes espaços e construções para parecerem reais. Além disso, requerem um grande espaço de armazenamento quando não estão sendo utilizados. Estes fatores podem elevar o custo da produção podendo, inclusive, inviabilizar algumas produções onde são cruciais para seu desenvolvimento.

Para conseguir amenizar estes problemas, novas técnicas envolvendo Computação Gráfica estão sendo criadas. Uma delas é a utilização de estúdios virtuais na criação de cenários virtuais. Segundo Grau et al. (2002), o termo "estúdio virtual" foi criado para distinguir estes novos sistemas que permitem a utilização, por meio de Computação Gráfica, de objetos tridimensionais virtuais, dos tradicionais sistemas de *chroma-key*.

O estúdio virtual é uma alternativa crescente entre as produtoras e redes de televisão, pois não precisa de grandes investimentos como construções em alvenaria ou madeira, não precisam ser armazenados em locais com muitos metros cúbicos de dimensão, podem ser facilmente criados ou alterados a qualquer momento, pois tratam-se de um *software* de computador.

Segundo Millerson e Owen (2009), embora no início, o custo da criação do estúdio virtual seja bastante significativo, devido à aquisição de todo equipamento de hardware, câmeras, iluminação e *software*, a economia que se tem no decorrer de sua utilização faz mudar rapidamente esta ideia, pois muitas vezes os próprios recursos se pagarão.

O espaço necessário e o tempo para a construção do estúdio virtual também é menor, comparando-se com um estúdio tradicional.

A utilização de estúdios virtuais possibilita, com recursos de computação, a criação de cenários virtuais tridimensionais (3D). Estes cenários permitem a inserção de objetos também em três dimensões, assim como todo o cenário em si, abrindo as portas para criação de um mundo totalmente controlável em 3D.

No entanto, a criação destes objetos tridimensionais não é uma tarefa trivial. A modelagem destes objetos de forma manual não produzem, normalmente, resultados satisfatórios quanto ao aspecto realismo.

Neste sentido, muitas pesquisas, atualmente, abordam o termo reconstrução 3D baseada em informações obtidas da captura de características topológicas e das texturas dos objetos existentes no mundo real.

Existem diversos métodos criados com este enfoque, como em Bethencourt e Jaulin (2013), Apolinário et al. (2006) e Santos et al. (2011), dentre os quais destaca-se o *Structure From Motion* (SFM), como o apresentado em Pollefeys (2002).

Este método permite a obtenção de um ambiente virtual 3D (composto por um ou mais objetos 3D) reconstruído a partir de um ambiente real capturado sob diversos pontos de vista. Isto é realizado por meio da análise de uma sequência de imagens em 2D (geralmente fotos), observando-se algumas característica semelhantes entre elas. A Figura 1 ilustra o *pipeline* tradicional de reconstrução baseado em SFM, no qual se destacam duas etapas principais: a etapa de captura e seleção de imagens e a etapa da reconstrução tridimensional propriamente dita.

Na etapa de captura e seleção, as imagens são normalmente escolhidas de forma manual pelo usuário, e nem sempre serão úteis à etapa de reconstrução.

Caso o conjunto de entrada contenha imagens que não possuam as características esperadas pelo método de reconstrução, as mesmas devem ser descartadas. Se as imagens válidas restantes no conjunto de entrada não forem suficientes para a reconstrução, o usuário deverá efetuar uma nova captura de imagens do objeto original.

A reconstrução com o método SFM é uma alternativa que substitui a forma tradicional de modelagem manual, que em alguns casos pode demorar a ser concluída e possuir resultados finais não muito satisfatórios. A modelagem manual pode não ser tão realista quanto à modelagem feita pelo SFM.

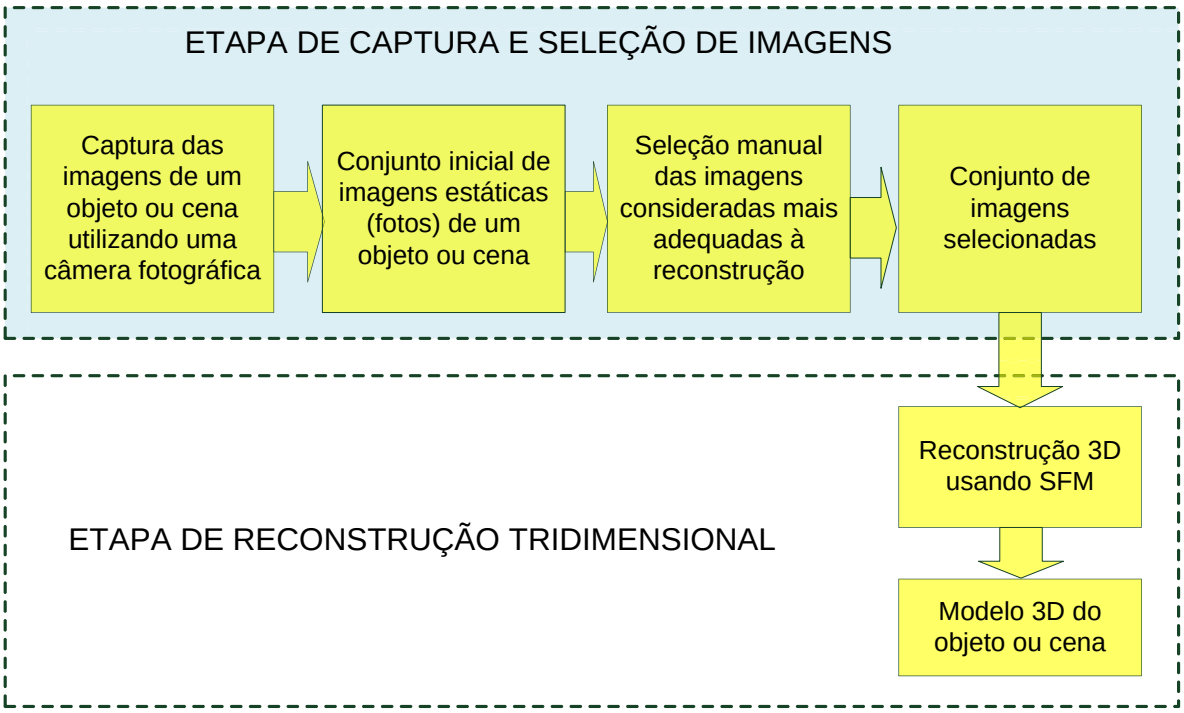


Figura 1 – *Pipeline* tradicional para a de reconstrução tridimensional por SFM.

Porém como o método requerer imagens, que contenham certas características específicas, pode ser difícil conseguir um conjunto de imagens satisfatórias na primeira tentativa de captura.

Para amenizar este problema pode-se utilizar um vídeo para realizar a captura das imagens. Segundo Paula Filho (2011) um vídeo é uma sequência de imagens, capturadas e reproduzidas a uma velocidade constante que transmitem noção de movimento. Cada imagem é denominada de *frame*[1].

Ainda, segundo o autor, capturar um fluxo constante de imagens (ou seja, um vídeo) tem-se uma grande quantidade de imagens que podem ser selecionadas, porém a possibilidade de elas serem muito semelhantes é grande. Se as imagens não forem corretamente escolhidas, a reconstrução não será bem sucedida.

Portanto, utilizar um procedimento de captura e seleção manual de imagens pode não ser produtivo.

Utilizando-se um fluxo de vídeo, tem-se um maior número de imagens em sequência, aumentando as possibilidades de captura de nuances importantes sobre o objeto ou cena a ser reconstruída. Isto porque, normalmente, o movimento de câmera é mais linear, sem

[1] *Frame:* em inglês significa quadro. É cada um dos quadros ou imagens fixas de um produto audiovisual. Aumont e Marie (2003).

mudanças abruptas de ponto de vista, características essenciais para uma reconstrução realizada com o método SFM.

1.1. OBJETO

O objeto principal de estudo deste trabalho é o levantamento das características relevantes à seleção automática de *frames*, a partir de um fluxo de vídeo, para a geração de um conjunto de imagens de entrada, visando o método de reconstrução SFM.

1.2. OBJETIVOS

O objetivo principal do trabalho é, a partir do levantamento das características consideradas essenciais à reconstrução baseada em SFM, desenvolver um protótipo em *software* que atue como filtro para seleção automática de *frames*. Portanto, considerando o *pipeline* de reconstrução baseado em SFM mostrado na Figura 2, o enfoque do trabalho localiza-se sem sua fase inicial, ou seja, a etapa de captura e seleção de imagens.

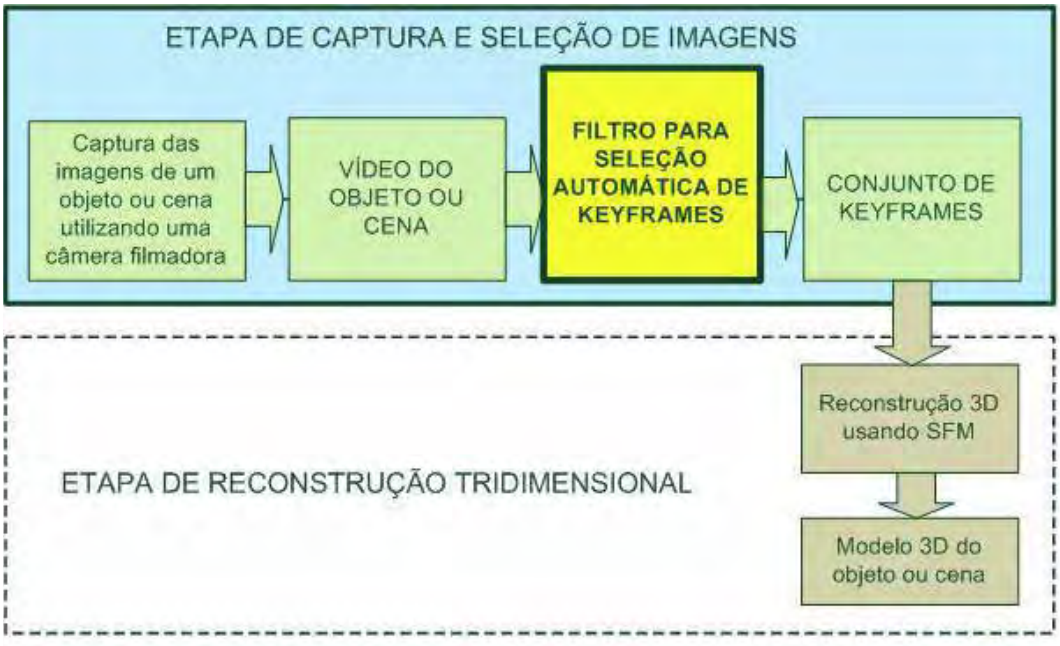


Figura 2 – *Pipeline* proposto com ênfase no filtro para seleção automática de *keyframes*.

São objetivos secundários deste trabalho:

- Testar o protótipo desenvolvido, comparando seus resultados com a solução manual.
- Testar se os resultados obtidos, após a reconstrução utilizando-se o protótipo, são aplicáveis à montagem de cenários virtuais.

1.3. ORGANIZAÇÃO DA DISSERTAÇÃO

Esta dissertação foi estruturada da forma que segue.

No capítulo 2 foi realizado um estudo sobre o método *Structure From Motion* – SFM – onde são apresentadas as principais etapas da reconstrução 3D com a utilização deste método, levantados os fatores e características que são procurados nas imagens e com maior ênfase para definição do *pipeline* de desenvolvimento do método.

No capítulo 3 é apresentado um estudo sobre o pré-processamento de imagens para reconstrução por SFM, abordando as principais características em imagens digitais, tais como iluminação, borramento e filtros de realce essenciais para este trabalho.

O capítulo 4 apresenta o sistema desenvolvido, começando com a descrição do método utilizado. Também é descrita a implementação do protótipo, com a ênfase nos principais módulos desenvolvidos.

No capítulo 5 são mostrados os testes e avaliação dos resultados para validação do sistema.

Por fim, no capítulo 6 são apresentadas as conclusões e indicação dos trabalhos futuros para o aperfeiçoamento do sistema implementado.

2. STRUCTURE FROM MOTION - SFM

Os humanos são dotados de dois dispositivos (os olhos) que adquirem dois ângulos de uma mesma imagem simultaneamente e isto permite estimar a profundidade relativa dos pontos capturados. Isto levou os pesquisadores a desenvolver estudos sobre técnicas binoculares de reconstrução, como visto em Clark (1992), que utilizam um método chamado *shape from stereo*.

Um princípio semelhante, denominado *multi-view stereo matching*, pode ser utilizado para estimar a profundidade de um objeto a partir da observação deste objeto em várias imagens com pontos de vistas diferentes. Os passos básicos são mostrados em Pollefeys et al. (2004) e Hartley e Zisserman (2004).

Outra forma possível de obter esta profundidade é a denominada *Structure From Motion*, como mostrada em Pollefeys (2002). Assim, a partir de imagens sequenciais adquiridas movimentando a câmera em torno do objeto ou ambiente original, uma representação 3D em escala é gerada e, em seguida, texturizada com uma imagem da cena real para aumentar o realismo da reconstrução.

Serão apresentados os principais tópicos para a realização de uma reconstrução em 3D utilizando o SFM.

2.1. EXTRAÇÃO DOS PONTOS CARACTERÍSTICOS

Dada as características correspondentes em duas imagens, a posição 3D dos pontos que geram esses resultados e as posições das câmeras que geram as imagens podem ser computadas (Hartley & Zisserman, 2004). Portanto, o primeiro passo importante no processo de reconstrução 3D é a extração dos pontos característicos (*features*).

Uma escolha bem criteriosa, que contenha informações sobre alterações na geometria podem diminuir o espaço a ser analisado em busca do emparelhamento estéreo para conseguir os dados suficientes na reconstrução 3D do objeto analisado (Ozanian, 1995).

Existem muitas técnicas para detectar características em imagens. Eles incluem aqueles baseados em características globais e as características locais.

Características globais como os baseados na área, perímetro, ou simetria de um objeto. Mas as medidas de área e perímetro são sensíveis a zoom e orientação. Com isso, torna-se interessante o uso das técnicas de extração de características locais, pois elas podem ser mais adequadas ao *zoom* da imagem, orientação do objeto e outras formas de transformações na imagem.

Outros métodos de extração de recursos, incluindo a extração de borramentos, modelo correspondente e de extração do contorno, podem também ser encontrados na literatura.

Os pontos, denominados pontos característicos, são aqueles que oferecerem certa discrepância entre os seus valores de intensidades, comparados com os seus pontos vizinhos.

Um dos métodos mais utilizados na detecção dos pontos característicos é o método de Harris (Harris e Stephens, 1988), que realiza uma auto correlação entre os pontos de uma imagem.

Este método baseia-se na matriz de covariância do gradiente (M) definida pela equação 2.1:

$$M=\begin{bmatrix} \Delta^2.x & \Delta x.\Delta y \\ \Delta x.\Delta y & \Delta^2.y \end{bmatrix} \quad (2.1)$$

Na equação, $\Delta x \; e \; \Delta y$ são os gradientes das imagens, nas direções dos eixos x e y. Se os valores da matriz M, $\alpha \; e \; \beta$, forem elevados e semelhantes em amplitude, significa que foi detectado um vértice da imagem, visto na Figura 3.

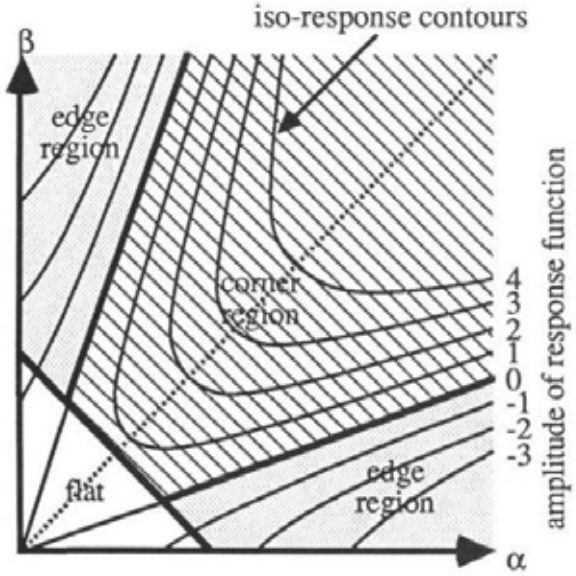


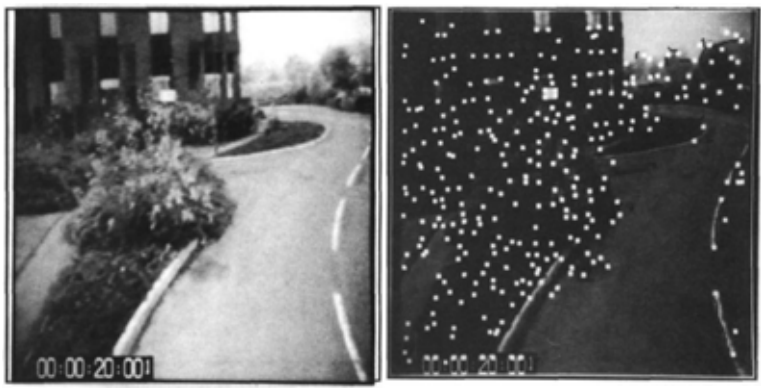
Figura 3 – Divisão do espaço dos valores próprios da matriz M para a detecção de pontos característicos (*features*) ou fronteiras de estruturas em imagens. Fonte: Harris e Stephens, 1988.

Para determinar os pontos característicos, o autor definiu equação 2.2:

$$R = \det(M)-k(tr(M))^2 \quad (2.2)$$

Na equação, $\det(M) = \alpha.\beta = (\Delta x)^2 - (\Delta x.\Delta y)^2$ e $tr(M)= \alpha + \beta = (\Delta x)^2 + (\Delta y)^2$. O valor da constante k deve ser estimado, sugerido pelo autor em 0,04 inicialmente e

sendo incrementado ou diminuído de acordo com o limiar desejado. Veja o exemplo na Figura 4.



(a) (b)

Figura 4 – Aplicação do detector Harris.
Em (a) a imagem original, em (b) os pontos branco são cantos e as áreas em preto são bordas.
Fonte: Harris e Stephens, 1988

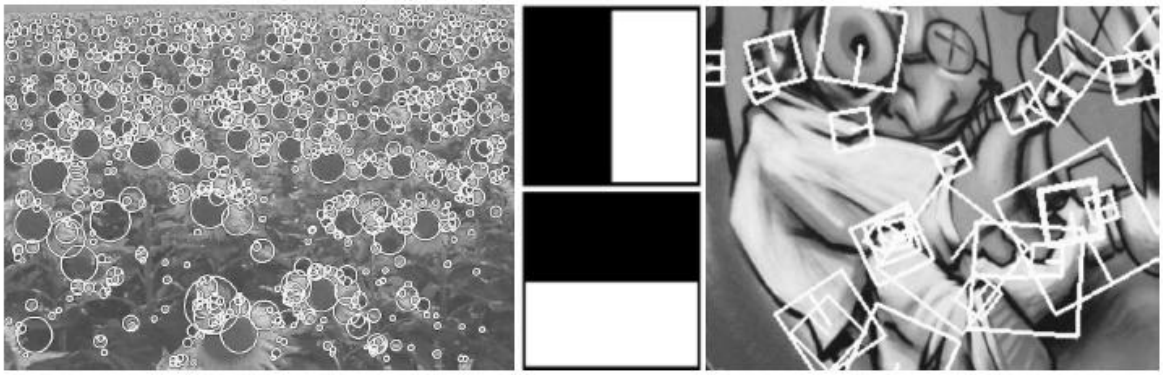
Uma vez detectados os pontos característicos, passamos para a segunda etapa da reconstrução SFM, o emparelhamento dos pontos característicos entre as imagens.

Outro método de detecção de pontos característicos é o *Scale-invariant feature transform* (SIFT), descrito por Lowe (2004). Este método é invariante quanto à escala, rotação e é robusto em mudanças bruscas na iluminação.

São quatro os principais estágios do método:

- **Escala de espaço – Detecção de extremidades** - Esta etapa pesquisa (sobre todas as escalas e locais da imagem) por pontos invariantes à escala e orientação.
- **Localização de pontos chaves** - os pontos detectados na etapa anterior são classificados como pontos-chaves com base em sua estabilidade. A estabilidade pode ser determinada por ajustes no modelo de localização e dimensão.
- **Atribuição de orientação** - uma orientação é atribuída a cada ponto chave com base na direção do gradiente de domínio. Ao fazer isso, o recurso torna-se invariante para a orientação, escala, e de transformação local.
- **Descritor de ponto chave** - gradientes são medidos dentro de uma pequena janela em torno do ponto central e resumido em um vetor descritor de modo que as alterações na iluminação e distorção da forma local podem ser toleradas.

Outro método robusto de detecção de pontos característicos é o *Speeded Up Robust Features* – SURF, como visto em Bay et al. (2006). O descritor do SURF é semelhante ao SIFT, mas sua complexidade é menor. Veja o exemplo na Figura 5.



(a) (b) (c)

Figura 5 – Detecção de pontos característicos (*features*) com SURF.
Em (a), características reproduzíveis. Em (b) as ondas Haar, utilizadas na detecção. Em (c), pontos chaves, com detecção de orientação. Fonte: Bay et al. (2006)

Para realizar a rotação invariante com o SURF, características com orientações reproduzíveis são detectadas. Para calcular a orientação, as respostas das ondas de Haar nas direções vertical e horizontal são combinadas para conseguir um vetor gradiente. Uma região quadrada, alinhada com a orientação selecionada é utilizada para extração do descritor.

Tabela 1 - Desempenho do SURF e SIFT

	U-SURF	SURF	SURF-128	SIFT
Tempo (ms)	255	354	391	1036

Fonte: Bay et al. (2006)

Na Tabela 1, uma comparação de desempenho dos métodos SIFT e SURF mostram que o SURF é mais eficiente em qualquer de suas variações do que o SIFT.

2.2. CORRESPONDÊNCIA DAS CARACTERÍSTICAS

Os métodos SIFT e SURF retornam correspondências de orientação, escala, iluminação e informações de localização. O próximo passo então é encontrar correspondências dessas características em um par de imagens.

A técnica mais popular foi proposta por Lowe (2004), combinando as características detectadas com os quadros vizinhos mais próximos.

> Para a correspondência de recurso, cada ponto-chave é compensado com seu vizinho mais próximo, minimizando a distância euclidiana entre os principais vetores de pontos descritores. (Lowe, 2004)

Os resultados não são tipicamente perfeitos, e alguns erros podem ser classificados como correspondências devido à ambiguidade de padrões que podem acontecer.

Para descartar estas entradas inválidas, a técnica só aceita entradas em que a razão entre a distância entre o primeiro e o segundo resultado estiverem acima de um valor do limiar determinado. Um exemplo é mostrado na Figura 6.

(a) (b) (c)

Figura 6 – Detecção de correspondências.
Em (a) objetos que se deseja a correspondência. Em (b) a imagem original. Em (c) aplicação do algoritmo de Lowe para detecção das correspondências de características. Fonte: Lowe (2004)

2.3. MODELO MATEMÁTICO DA CÂMERA

Para que se possa entender como realmente funciona o SFM é preciso conhecer alguns conceitos de visão computacional. Primeiramente é necessário definir um modelo matemático da câmera, que irá descrever o comportamento da projeção do mundo real em si mesma.

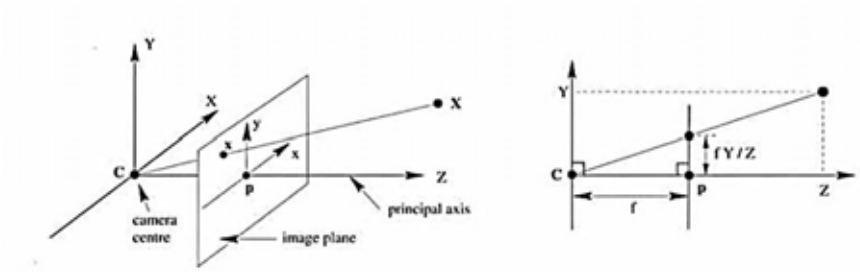


Figura 7 - Modelo geométrico da câmera *Pinhole*.
Fonte: Hartley e Zisserman (2004)

O modelo mais simples e mais utilizado é o denominado *pinhole*, detalhado em Hartley e Zisserman (2004). O modelo geométrico da câmera é mostrado na Figura 7.

A câmera possui um centro de coordenadas próprio. C é o centro da câmera e P o ponto principal. O centro da câmera está colocado na origem do sistema de coordenadas. Note que o plano da imagem está colocado em frente ao centro da câmera.

Como a origem coincide com o centro de projeção, o vetor normal do centro de projeção é a direção do eixo principal Z. Os eixos X e Y são, portanto, paralelos às bordas da imagem. A rotação e a translação deste sistema próprio de coordenadas podem ser representadas por uma matriz de dimensão 3x4, que é chamada de "parâmetros extrínsecos" da câmera.

Os chamados "parâmetros intrínsecos" da câmera serão formados pelas coordenadas da projeção do centro da câmera no plano focal e pela medida da distância focal. Estes parâmetros são representados em uma matriz 3x3 e são utilizados para transformar as coordenadas da câmera para as coordenadas da imagem. Os parâmetros intrínsecos da câmera são extraídos durante o processo de calibração.

Marques (2007) definiu calibração:

> A calibração de câmera consiste no processo de determinar dados geométricos digitais e ópticos da câmera, admitindo que sejam conhecidos um conjunto de pares de pontos bidimensionais em uma imagem e seus respectivos pontos tridimensionais. Com este intuito, fica evidente a necessidade de definir relações entre as coordenadas de pontos 3D com as coordenadas 2D de imagens dos mesmos. (Marques, 2007)

2.4. GEOMETRIA EPIPOLAR

Outro conceito primordial para a reconstrução SFM é a geometria epipolar (Hartley e Zisserman, 2004). Ela consiste no relacionamento de duas câmeras, onde é visualizado um conjunto de pontos 3D. Para cada ponto 3D visualizado pode-se ressaltar um plano epipolar.

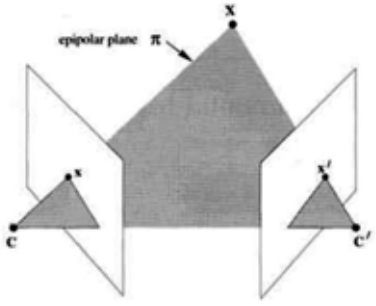


Figura 8 - O plano epipolar.
Fonte: Hartley e Zisserman (2004)

Este plano epipolar é definido por três pontos, sendo os centros das câmeras e o ponto em questão, visto na Figura 8.

Os pontos C e C’ representam os centros das câmeras, X é o ponto no espaço 3D e π o plano epipolar. O plano π corta os planos das imagens em dois pontos. Considera-se um dos pontos como sendo o epipolo, que será a projeção do centro da segunda câmera no plano da imagem.

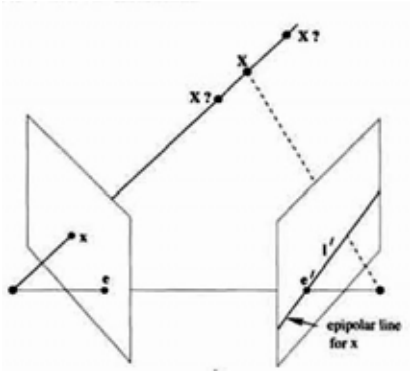


Figura 9 - A linha epipolar.
Fonte: Hartley e Zisserman (2004)

O outro ponto será a projeção do ponto 3D (X) no plano da imagem. Estes dois pontos, quando projetados, formarão uma linha denominada linha epipolar, visto na Figura 9.

2.5. A MATRIZ FUNDAMENTAL

A linha epipolar está representada por l’. Sendo assim, para cada ponto X existirá uma linha epipolar em cada plano de imagem. Dessa forma, pode-se descrever uma matriz F (matriz fundamental) vista na equação 2.3:

$$l.F.l' = 0 \quad (2.3)$$

A partir de oito correspondências de pontos é possível determinar uma matriz fundamental que relacionaram todas as possíveis coincidências da imagem, ignorando a existência das falsas coincidências.

Existem também outros métodos que funcionam com menos pontos, apresentados por Hartley e Zisserman (2004), por exemplo.

A grande chave é a escolha da matriz fundamental correta. Esta escolha será feita por um rastreador 2D. Existem métodos robustos, como o RANSAC[2], para solucionar este problema.

Este método achará a hipótese, no caso da matriz fundamental, que melhor adaptar-se ao conjunto de correspondências, atribuindo um erro à hipótese. A hipótese vencedora é aquela que ao mesmo tempo tenha o menor erro associado e o maior número de correspondências.

2.6. POSES DAS CÂMERAS

Após conseguir a matriz fundamental, pode-se indicar a posição das câmeras envolvidas no processo. Dessa forma, os pontos 3D são encontrados de acordo com a resolução da fórmula 2.4:

$$K.P.X = x \quad (2.4)$$

Na fórmula, K representa uma matriz com os parâmetros intrínsecos da câmera, como descrito em 2.3, P é a pose da câmera, X representa o ponto 3D em questão e o x representará o ponto em coordenadas na imagem 2D. Desta forma pode-se calcular a projeção do ponto x (2D) que foi capturado do espaço 3D pela câmera e representá-lo em uma imagem.

Assim, com foi determinado os pontos 3D no espaço 2D e as poses das câmeras conclui-se a etapa de reconstrução do SFM.

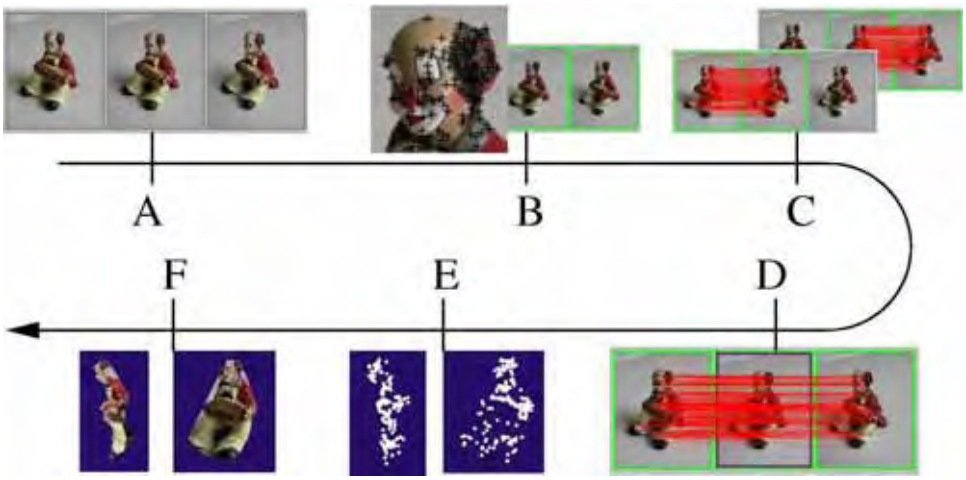


Figura 10 - *Pipeline* da reconstrução por SFM.
Fonte: Häming e Peters (2010)

[2] RANSAC é uma abreviatura de "RANdom SAmple Consensus". É um método iterativo para estimar os parâmetros de um modelo matemático.

Na Figura 10 pode ser observado o *pipeline* da reconstrução por SFM. Em A, as imagens de entrada. Em B, a detecção dos pontos. Em C, a correspondência dos pontos. Em D, a filtragem. Em E, a visualização da nuvem de pontos. Em F, a visualização do modelo reconstruído e já com sua textura.

2.7. APLICAÇÕES EXISTENTES

Atualmente existem diversas aplicações que realizam a construção por SFM. Nesta seção serão apresentadas as ferramentas que estão em evidência no meio acadêmico e comercial, de acordo com Kersten et al.. (2012) e Bjørnstad e Fedreheim (2012).

2.7.1. SFM TOOLKIT 3/PHOTOSYNTH TOOLKIT 7/VISUAL SFM

Esses três *softwares* são toolkits compostos de vários pequenos aplicativos. O SFM Toolkit e o Photosynth Toolkit foram escritos por Henri Astre, que atualmente trabalha na Microsoft. São ferramentas automatizadas e de fácil utilização e podem ser utilizados em Windows ou Linux, por serem escritos em C++.

Já o Visual SFM (WU, 2011) foi escrito por Changchang Wu, da Universidade de Washington, em Seattle e possui uma base muito semelhante aos outros dois *softwares*.

Todos eles seguem os passos padrões do SFM (detecção e correspondência dos pontos, orientação, geração esparsa e finalmente geração densa) e utilizam o SIFT na reconstrução.

2.7.2. BUNDLER

Escrito por Noé Snavely, ele gera uma nuvem de pontos dispersa com dados de orientação interior e exterior. Escrito em C++ atualmente está na versão 4. Possui filtros de correção para distância focal e distorção radial. Utiliza o SIFT para detectar e realizar a correspondência dos pontos de ligação.

Mas ele apenas gera uma nuvem de pontos, incapaz de gerar a reconstrução densa. Seu foco está em obter os parâmetros de orientação e localização dos pontos que estão a frente ou atrás de outros pontos.

O Bundler tem uma restrição em questão da dimensão da imagem, que não pode ter mais de 2048 *pixels*[3] de largura. Este problema é devido ao fato da utilização de uma versão demo do algoritmo SIFT utilizado.

[3] *Pixel*: abreviação de *Picture Element*, é a unidade básica da imagem digital. Gomes (2001)

2.7.3. PMVS2/CMVS

PMVS são as iniciais de "*Patch-based Multi View Stero*", ou visão estéreo baseada em remendos e encontra-se em sua versão 2.

Foi desenvolvida por Yasutaka Furukawa na Universidade de Washington e Jean Ponce - Ecole Normale Supérieure.

Ele gera um ponto de nuvem densa e limpa os pontos que podem ser considerados como ruído. Os ruídos ainda estarão na nuvem de pontos, mas não serão utilizados na reconstrução.

Pixels, por exemplo, que pertençam a um gramado atrás de um edifício seriam capazes de atrapalhar na detecção das extremidades do telhado e as superfícies adjacentes poderiam conter várias camadas de pontos, o que ocasionaria num maior tempo de processamento para realizar uma reconstrução.

O *software* armazena em um arquivo os valores de entrada necessários para como executar a reconstrução. As definições controlam tudo, desde o tamanho das manchas a ser utilizados em correspondência até a possibilidade de utilizar a detecção de borda e em que medida.

CMVS é a abreviação de *Clustering Views for Multi-View Stereo*, ou divisão das visões para multi-visão estéreo. Ele permite dividir o processamento da reconstrução para evitar o travamento do sistema devido, por exemplo, a uma grande quantidade de imagens de entrada em uma reconstrução 3D.

O *Software* divide as imagens em grupos menores de imagem, permitindo então que cada grupo seja processado em locais diferentes (mais de um processador, por exemplo), de modo que não se perca as conexões entre esses grupos, que serão reconstruídos posteriormente com o algoritmo PMVS2.

2.7.4. MICROSOFT PHOTOSYNTH

Este *Toolkit* é diferente. Ele é utilizado para detecção e adequação dos pontos de ligação e processos de orientação. Não existe documentação que indique os algoritmos são utilizados no Photosynth.

O Photosynth é uma aplicação web que funciona através da criação de uma conta com uma capacidade de 20GB de armazenamento de imagens. Em conjunto com um pequeno aplicativo utilizado para realizar o *upload* de imagens para a conta de usuário, onde o processamento de imagens na aplicação *web* será executado.

O Photosynth gera parâmetros de orientação interna e externa, bem como que se podem ver os resultados em um visualizador muito conveniente. Existe o recurso de sobrepor imagens umas sobre as outras é particularmente interessante, uma vez que, em certa medida vai obter um efeito 3D, embora as imagens sejam em 2D.

Pode-se também visualizar os dados como uma nuvem de pontos dispersos. Esta nuvem de pontos pode ser baixada e usada em outras aplicações, na forma de um arquivo de camadas com os parâmetros da câmara de orientação e coordenadas de pontos.

2.7.5. AUTODESK 123DCATCH (BETA)

Em 2010, a Autodesk introduziu o projeto chamado *PhotoFly*, consistente em um serviço baseado na *web* que permite que usuários do mundo inteiro possam gerar seus modelos 3D com o envio de, pelo menos, 5 imagens de um objeto.

O *PhotoFly* foi baseado em um programa francês, o smart3Dcapture, da Acute3D, cujo algoritmo está descrito em Courchay et al. (2010). O *PhotoPhly* utiliza algoritmos de visão computacional, associados com fotogrametria e utiliza o conceito de computação nas nuvens para oferecer um alto desempenho e eficiência nas conversões de imagens 2D para objetos 3D.

Assim como no Photosynth, é necessária a instalação de um *software* na máquina do cliente para que as fotos sejam transferidas para o servidor.

Dependendo da complexidade do modelo 3D a ser gerado, ele pode ser recebido pelo *software* em alguns minutos ou pode ser cadastrada uma conta de correio eletrônico, onde será enviado um *e-mail* com um *link* para *download* do arquivo quando o processamento estiver finalizado.

Em novembro de 2011, a Autodesk trocou o nome do projeto de *PhotoFly* para 123D Catch (Beta) (AUTODESK, 2012) depois que a companhia accute3D apresentou o *software* smart3Dcapture ao público, em outubro de 2011.

2.8. COMPARAÇÃO ENTRE OS PRINCIPAIS *SOFTWARES*

Em Kerten e Lindstaedt (2012) foram feitas algumas comparações de desempenho dos *software*s Bundler, VisualSFM, Microsoft Photosynth e 123D Catch (Beta) (AUTODESK, 2012) em diversas reconstruções. Os resultados são mostrados na Tabela 2.

Como exemplo, em uma das reconstruções (do antigo prédio dos bombeiros, descrito como *Fire House*) foi utilizado um computador com processador Intel Core i7 Q740

com 1,73Ghz, com memória interna de 16GB rodando na plataforma Windows 7 Enterprise (64 bits), com placa gráfica NVIDIA GeForce GT 445M, processando 66 fotos da Nikon D90 com lente 18mm.

Tabela 2 - Dados dos resultados obtidos em reconstruções por SFM

Objeto	Câmera/Lente(mm)	*Pixels*	Nº de fotos	*Software*	Pontos	Triângulos
Town House	Nikon D90/20	4288x2848	19	PhotoSynth	20.237	-
				123D Catch	272.350	515.442
				Bundler	1.016.874	895.986
Fire House	Nikon D90/18	4288x2848	66	123D Catch	176.919	352.091
				Bundler	1.541.924	-
				Visual SFM	1.167.906	-
Zwinger	Nikon D90/28	4288x2848	15	PhotoSynth	18.553	-
				123D Catch	155.697	285.669
				Bundler	917.965	-
Lion	Nikon D90/20	4288x2848	39	123D Catch	344.679	686.285
				Bundler	1.373.712	2.669.244
Moai Poike	Nikon D70/35	3008x2000	27	123D Catch	85.092	169.131
				Bundler	629.644	-

Fonte: Adaptado de Kerten e Lindstaedt (2012)

Para noção de comparação de tempo de processamento dos *softwares* analisados, o Bundler precisou de 24 horas para gerar 1.541.924 pontos, enquanto o VisualSFM precisou apenas de 42 minutos para obter aproximadamente 374.000 pontos, o que corresponde a um fator de aproximadamente 33 vezes mais rapidez e 24% menos pontos obtidos, com diferenças maiores percebidas nos locais com sombras.

(a) (b) (c)

Figura 11 - Reconstrução da *Fire House*.
Em (a) a reconstrução obtida pelo 123D Catch. Em (b) reconstrução do VisualSFM e em (c) reconstrução com Bundler/PMVS2. Fonte: Kerten e Lindstaedt (2012)

Já o mesmo teste com o Autodesk 123D Catch (AUTODESK, 2012) necessitou de 16 minutos de processamento no serviço via *Web*. O resultado ficou visualmente melhor do que os outros dois, mas as bordas são suavizadas e existem alguns pontos virtuais (geometricamente incorretos) nas antenas e em algumas áreas no topo do telhado.

Outros detalhes das demais reconstruções podem ser obtidos no trabalho completo dos autores.

2.9. CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os principais conceitos para a realização da reconstrução por SFM.

Foram levantados, também, alguns *softwares* que demonstram o estado da arte nas áreas acadêmicas, comerciais e baseadas em serviços pela web que utilizam o método para realizar reconstruções.

Testes foram mostrados, cujos resultados estão na Tabela 2, comparando reconstruções com um mesmo conjunto de imagens e seus dados podem ser comparados (número de pontos ou número de polígonos gerados, por exemplo).

Com base nos resultados dos testes, encontrados na literatura levantada, é possível concluir que a qualidade do resultado depende muito da qualidade das imagens, mostrando que o processo de obtenção pode refletir em uma reconstrução sem erros.

3. PRÉ-PROCESSAMENTO PARA RECONSTRUÇÃO POR SFM

Para realizar uma reconstrução pelo método SFM sem falhas e com um uma texturização adequada deve-se observar certas características que facilitarão o processamento e a detecção dos elementos chaves descritos no capítulo 2 deste trabalho.

3.1. PROPRIEDADES DAS IMAGENS DIGITAIS

Antes de tratar diretamente sobre os assuntos pré-processamento de imagens, é necessário conhecer alguns conceitos sobre imagem para facilitar o entendimento.

Gomes (2001) apresentou um resumo de conceitos essenciais que definem uma imagem digital:

> Um *pixel* é a abreviação de *picture element*, é a unidade básica de uma imagem digital. Resolução espacial, ou simplesmente, resolução, consiste no tamanho, na imagem real, que um *pixel* da imagem digital representa, ... A profundidade, quantização ou resolução espectral, consiste no número máximo de níveis de intensidade que esta imagem pode apresentar. (Gomes, 2001)

Observando a Figura 12 tem-se quatro imagens digitais com diferentes resoluções e profundidades.

As imagens à esquerda possuem menos resolução (128 x 128 *pixels*) e as imagens à direita possuem uma resolução maior (512 x 512 *pixels*).

Da mesma forma, as imagens variam em profundidade, de 4 tons de cima (imagens de cima) e 256 tons de cinza (imagens de baixo).

Figura 12 - Resolução e profundidade.
Fonte: Adaptado de Gomes (2001)

As imagens em tons de cinza geralmente são compostas por 8 *bits*, o que proporcionam 256 tons (variando do preto para o branco).

As imagens coloridas geralmente são compostas por 24 *bits*, formadas pela adição de 3 cores primárias, com 256 níveis de intensidade (8 *bits*) para cada uma.

Assim, uma imagem colorida constitui-se numa composição de 3 imagens de 8 *bits* cada, que podem ser tratadas separadamente.

3.2. ADQUIRINDO IMAGENS PARA O SFM

Como mostrado em Pollefeys (2002), o conjunto de imagens para uma reconstrução pelo método SFM é invariante quanto à translação, escala e iluminação, desde que seja possível determinar as correspondências entre as imagens. Mas para obter-se resultados satisfatórios, um pré-processamento nas imagens contribuem para melhor eficiência da reconstrução.

Como visto em Roberto (2009), a aquisição da imagem é uma das etapas que dependem bastante da forma como o SFM foi implementado. Nas formas mais simples, é necessário que a captura seja feita de forma bem controlada. Desta forma, os parâmetros intrínsecos da câmera (mostrados na subseção 2.3) são conhecidos e permanecerão constantes.

É conveniente a utilização de objetos que estabilizem a imagem (como um tripé ou trilhos) evitando movimentos bruscos e inesperados da câmera.

Mas na reconstrução de um cenário virtual de um ambiente com grandes dimensões, como uma catedral ou teatro, nem sempre é possível conseguir um ambiente controlado.

A captura poderá ser feita com uma câmera de vídeo manipulada manualmente, o que não dá suporte a uma obtenção de imagens tão estabilizadas como em um ambiente controlado.

Nestas condições, poderão ocorrer variações na distância focal, devido a algum zoom que eventualmente possa ser aplicado no momento da captura.

A alteração desta distância focal provoca uma mudança nos cálculos da matriz fundamental (subseção 2.5).

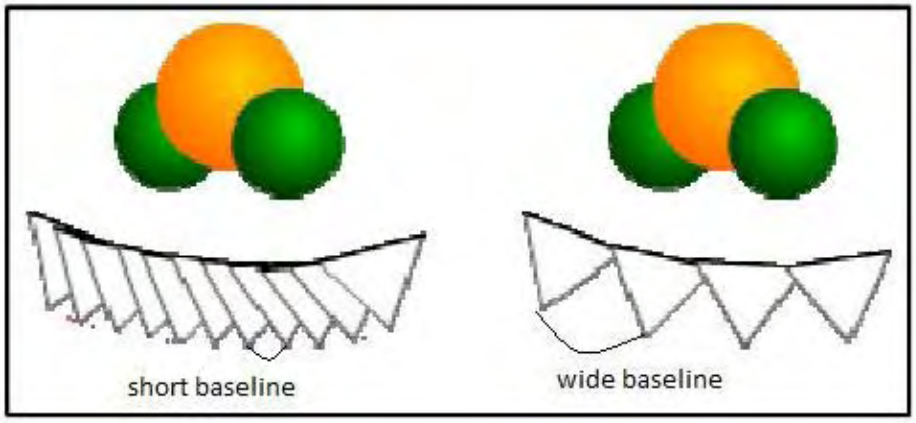
Sendo assim, uma reconstrução mais robusta, com um pré-processamento de imagens será necessária, para eliminar imagens com algum problema por outras mais nítidas, livres de borramentos ou com uma auto calibração, que deverão descobrir o valor da matriz de parâmetros intrínsecos a partir das próprias imagens.

3.3. SHORT BASELINE E WIDE BASELINE

Ainda segundo Roberto (2009) e Pollefeys (2004), seja em ambientes controlados ou não, as imagens podem ser adquiridas de diversas fontes como, por exemplo, câmeras de vídeo ou máquinas fotográficas.

No primeiro caso, a posição da câmera ao longo da sequência de imagens tende a mudar de forma mais suave, onde a diferença entre as imagens tende variar muito pouco. Isso faz com que a *baseline*[4] seja curta, já que a distância do centro de projeção da primeira imagem para a segunda é muito pequena, sendo denominada de *short baseline*.

No segundo caso, quando uma sequência de fotos é utilizada, a variação entre as imagens tendem a ser maior, caracterizando um *wide baseline*.



(a) **(b)**

Figura 13 – Exemplos de *Short baseline* e *Wide baseline*.
Em (a) obtenção das imagens a partir de uma câmera de vídeo. Em (b) obtenção das imagens de uma câmera fotográfica.
Fonte: Adaptado de Roberto (2009)

Analisando a Figura 13 pode-se notar a representação das câmeras a partir de cada centro de projeção. Quando as imagens obtidas provêm de uma câmera de vídeo, um número maior de imagens é obtido, ilustrando a grande quantidade de quadros disponíveis para a reconstrução.

Já quando as imagens provêm de câmeras fotográficas, as distâncias entre os centros de projeção aumentam e diminuem-se a quantidade de quadros disponíveis para a reconstrução.

A busca por pontos característicos entre um as imagens que participarão da reconstrução pelo método SFM, como mostrado na subseção 2.2 deste trabalho, não precisa

[4] *Baseline:* linha imaginária que une os dois centro focais de duas câmeras ou imagens. Hartley e Zisserman (2004).

estar em uma quantidade excessiva para ser bem sucedida. Ela pode conter muitas redundâncias que podem ser desprezadas e com isso reduzir o custo e o tempo de processamento, como vista em Lowe (2004).

3.4. BORRAMENTOS (MOTION BLUR)

Outra característica que pode influenciar na qualidade da reconstrução 3D realizada pelo método SFM é o borramento (em inglês, *motion blur*).

De acordo com Santos et al. (2011) borramentos em uma imagem são ocasionados por movimentos bruscos da câmera ou por um mal ajuste no foco e prejudica o processo de triangulação por dificultar o rastreamento dos pontos característicos da imagem, pois os *pixels* da imagem passam a ter cores similares ao longo de toda a imagem.

Um dos objetivos da aplicação de um filtro de identificação de borramento é conseguir *keyframes* com um menor percentual de borramento, que contenham uma maior preservação dos contrastes.

(a) (b)

Figura 14 - Exemplo de borramento em uma imagem.
Em (a) imagem original. Em (b) imagem com borramento proposital de 3 *pixels* à esquerda.
Fonte: Santos et al. (2011)

Os borramentos podem ser imperceptíveis ao olho humano, mas na detecção dos pontos de correspondências das imagens podem prejudicar pontos cruciais da reconstrução, como visto na Figura 14.

Fulton (2007) descreveu uma métrica para detectar o borramento baseada no conceito de que o borramento de uma imagem i(x,y) é dado pela distorção de um núcleo d(x,y) que produz esta imagem borrada b(x,y).

Calculando a força espectral (*power spectrum*) PS(*x*,*y*), que é uma imagem escalar, entre 0 e 1, dado por $PS(x,y)=log(1+|F(x,y)|^2)/log(1+|F(0,0)|^2)$, onde F é a transformada rápida de *Fourier* (*Fast Fourier Transform*).

Como resultado do método do autor, uma imagem ondulada é produzida quando a imagem está borrada, como na Figura 15.

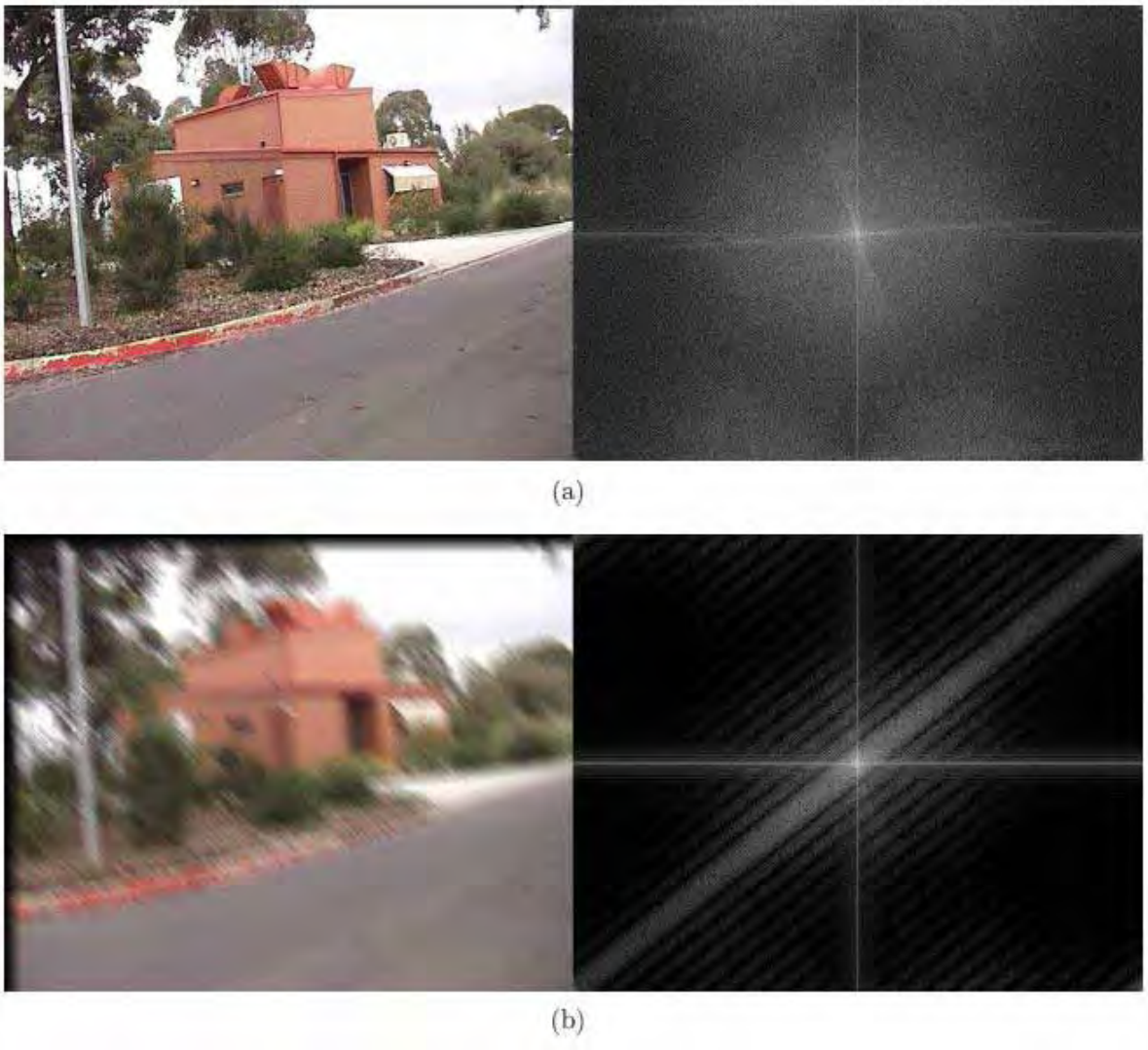


Figura 15 – Detecção de borramento.
Em (a) imagem original e seu *power spectrum*. Em (b) imagem com borramento proposital de 20 *pixels* com 45º de deslocamento e seu *power spectrum*. Fonte:Fulton (2007)

3.5. ILUMINAÇÃO: BRILHO E CONTRASTE

O brilho e o contraste em uma imagem são características que podem ser intuitivamente percebidas. Conforme Gomes (2001), Gonzalez e Woods (2010), em imagens em tons de cinza, os *pixels* com tons mais escuros, próximos ao preto (0) são percebidas como imagens mais escuras, com baixo brilho, assim como imagens em que predominam *pixels* com tons claros, próximo ao branco (255) são imagens mais claras, com alto brilho.

O contraste avalia a quantidade de variação de tons (claros para escuros ou vice-versa) que uma imagem tem. Imagens com baixas variações são denominadas imagens com baixo contraste, e imagens com altas variações, imagens com alto contraste.

O brilho de uma imagem pode ser definido como a média dos tons de cinza de todos os *pixels* da imagem. Dada uma imagem *f(x,y)*, de dimensão *X x Y*, seu brilho é definido na equação 3.1:

$$B = \frac{1}{n}\sum_{y=0}^{y-1}\sum_{x=0}^{x-1} f(x,y) \qquad (3.1)$$

onde *n* é o número total de *pixels* da imagem *f(x,y)*, calculado como *n=X*Y* a partir das dimensões X e Y.

Segundo Weeks (1999 *apud* Gomes 2001), o contraste pode ser definido como o desvio padrão dos tons de cinza de todos os *pixels* da imagem. Dada imagem *f(x,y)*, de dimensão *X x Y*, se contraste é definido pela equação 3.2:

$$C = \sqrt{\frac{1}{n}\sum_{y=0}^{y-1}\sum_{x=0}^{x-1}[f(x,y) - B]^2} \qquad (3.2)$$

onde B é o brilho da imagem *f*(x,y), calculado de acordo com a equação (3.1), e n é o número total de *pixels* da imagem (n=X*Y). Os exemplos são mostrados na Figura 16.

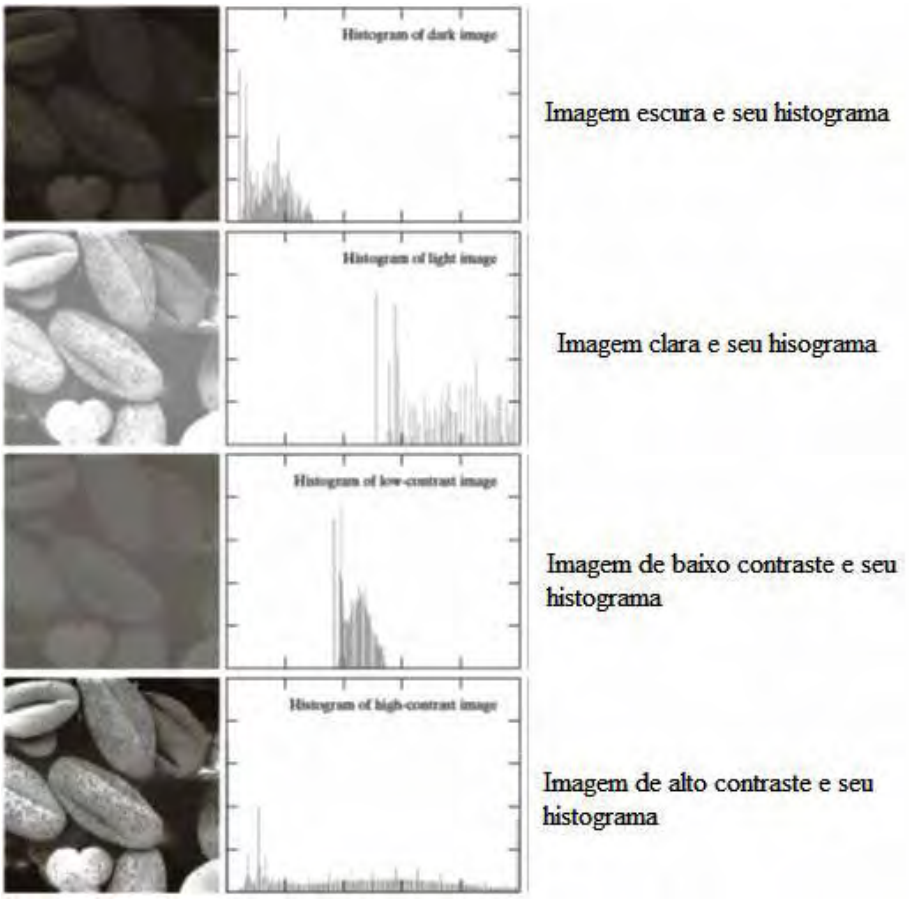


Figura 16 – Brilhos, contrastes e seus respectivos histogramas.
Fonte: Adaptado de Gonzalez e Woods (2010)

3.6. HISTOGRAMA

Uma vez calculadas as intensidades de brilho e contraste, é possível representar as informações obtidas da imagem em forma de um histograma, uma forma rápida de ver graficamente estas informações.

De acordo com Queiroz e Gomes (2001):

> O histograma de uma imagem traduz a distribuição dos seus níveis de cinza. Trata-se, pois, de uma representação gráfica do número de *pixels* associado a cada nível de cinza presente em uma imagem, podendo ser também expressada em termos do percentual do número total de *pixels* na imagem. (Queiroz e Gomes, 2001)

Desta forma, uma imagem *f(x,y)*, com dimensão *X x Y pixels* seu histograma pode ser descrito como na equação 3.3;

$$p(r) = \frac{n_r}{n} \qquad (3.3)$$

onde *r* representa os tons de cinza e varia de 0 a 255, n_r é o número de *pixels* com o tom de cinza *r* e *n* representa o número total de *pixels* da imagem *f(x,y)*, segundo Gomes (2001).

Os níveis de cinza ainda podem ser modificados com o objetivo de melhorar a visualização da imagem e aumentar a quantidade de informações que podem ser extraídas das imagens. Estas técnicas, de acordo com Gonzalez e Woods (2010) são conhecidas realce de contrastes. Estes realces de contrastes produzem uma ampliação no intervalo de níveis de cinza original, de forma que eles são exibidos num intervalo maior. Na Figura 16 apresenta-se alguns histogramas de imagens com diversas intensidades contrastes.

Existem diversos métodos de realces de contrastes, onde foram alguns de Gonzalez e Woods (2010), como o LUT (*Look-up Tables*) que transforma os Níveis de Contraste (NC) em novos valores através de uma função denominada Função de Transferência de Contrastes – FTC – produzindo resultados que podem ser linear (linhas) ou não linear (curvas) no histograma.

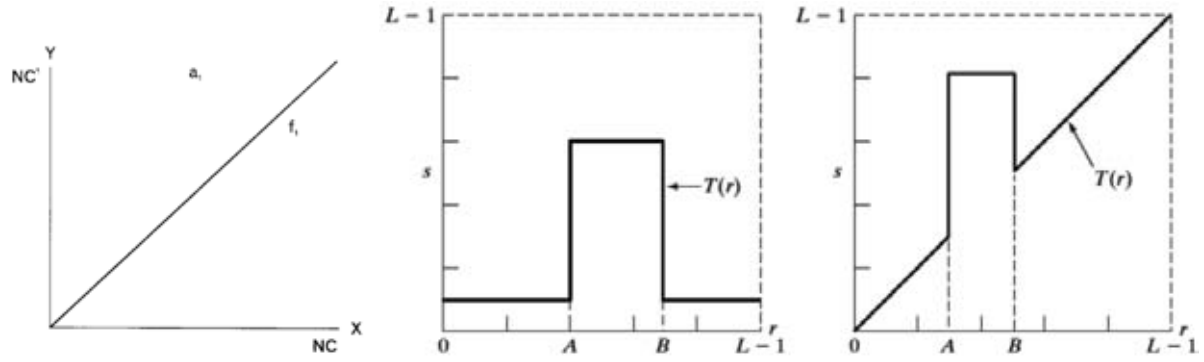
Outro método é o Realce Linear – RL, mostrado na Figura 17. Neste método a FTC linear é uma reta onde apenas dois parâmetros podem ser controlados, sendo a inclinação da reta e os pontos de intersecção com o eixo *x*. Quanto maior a inclinação da reta, maior a expansão do contraste.

Um caso particular de RL é o realce *MinMax*, onde o menor valor do NC da imagem é convertido em 0 e o maior valor em 255.

Gonzalez e Woods (2010) salientam ainda que uma limitação do RL é não levar em consideração a frequência do NC. Por isso, um realce chamado Realce de Equalização do

Histograma, baseado em uma função não linear que leva em consideração os NCs mais frequentes.

No histograma, a transformação acontecerá no sentido de uma expansão (aumento de contraste) dos intervalos centrais e uma compressão (perda do contraste) dos indivíduos dos intervalos laterais.



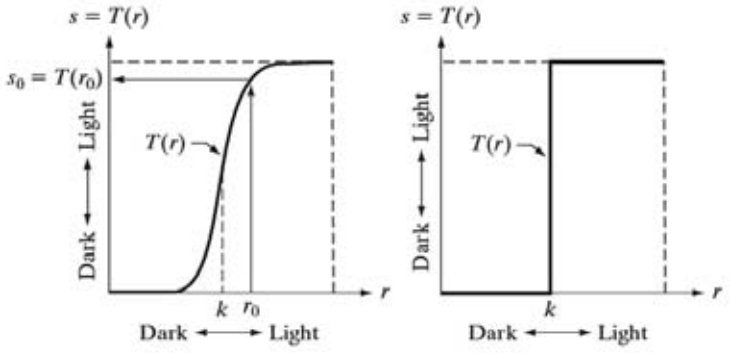
(a) (b) (c)

Figura 17 – Realces de Contraste.
Em (a) exemplo do gráfico de um realce linear, em (b) exemplo de gráfico de realce linear de intensificação entre o intervalo [A, B], e em (c) realce linear do intervalo [A,B] com preservação das intensidades em outros níveis. Fonte: Adaptado de Gonzalez e Wood (2010)

Outro método de realce importante é a Limiarização (*thresholding*), que divide os níveis de cinza de uma imagem em duas categorias, acima e abaixo de um determinado valor dentro do histograma.

Este método é também utilizado para criar uma máscara binária (*bitmap*) para uma imagem. Os valores acima do valor escolhido serão convertidos em branco (255) e os valores abaixo serão convertidos em preto (0). A imagem resultante será composta apenas de uma máscara binária de 1 e 0 (Figura 18).

Ainda podemos citar o Realce Gaussiano (também chamado de normalização do histograma), Realce de Raiz Quadrada (aumentar o brilho da imagem), Realce Negativo (inversão das áreas claras e escuras da imagem), Realce de infrequência (inversão do histograma de entrada da imagem), Realce Logarítmico (para realçar as informações contidas nas áreas mais escuras da imagem), Realce Exponencial (realçando as áreas mais claras da imagem e o Realce Parcial (utilizados para expandir o NC de um determinado segmento da imagem).



(a) (b)

Figura 18 – Realces de Limiarização (*thresholding*).
Em (a) detecta-se a existência de duas populações no histograma, cujo limite está no valor 17. A transformação segundo o realce de limiarização produz em (b) uma população em 0 e a outra em 255.
Fonte: Gonzalez e Woods (2010)

3.7. CONSIDERAÇÕES FINAIS

Este capítulo demonstra a importância de realizar um tratamento em imagens digitais antes de utilizá-las no processamento de uma reconstrução pelo método SFM. Foram mostradas as etapas de a aquisição das imagens para utilização no SFM, onde mesmo sabendo que o método é invariante à translação, escala e iluminação - Pollefeys (2002) – certos parâmetros (como os intrínsecos da câmera) são fundamentais.

Para realizar a reconstrução por SFM as características de visão estéreo são também consideradas, por exemplo, como a *baseline*, e com imagens extraídas de vídeos, a quantidade de centros de projeções podem ser menores (caracterizando o *short baseline*) ou maiores (configurando um *wide baseline*), como mostrado na Figura 13.

Outra característica que pode comprometer o resultado da reconstrução é a presença de borramentos (*motion blur*) nas imagens. Com o cálculo do seu nível de espectro de força (*power spectrum*) é possível determinar um valor e estipular, por um limiar, se a imagem contem visibilidade suficiente para a reconstrução obter sucesso.

As características de brilho e contraste foram também apresentadas, onde é possível observar seus níveis de brilho e contraste e mostrar os resultados em forma de gráficos, chamados histogramas (como na Figura 16).

Estes níveis obtidos das imagens podem ser modificados para melhorar sua visualização e aumentar a quantidade de informação que pode ser extraída para o processamento do SFM.

Alguns métodos de realce destes níveis, com o LUT, RL, *MinMax*, Equalização, dentre outros, são descritos e podem ser aplicados de acordo com a necessidade de melhorar a

qualidade de uma imagem com certas características, onde cada método é indicado para um determinado tipo de situação.

Estes métodos e técnicas apresentados serão fundamentais na escolha dos *keyframes*, pois a cada minuto de um vídeo é possível obter um grande número de *frames* e somente os melhores e essenciais devem ser considerados para obtenção de uma reconstrução por SFM sem erros.

4. SISTEMA DESENVOLVIDO

Como discutido no Capítulo 1, realizar uma reconstrução 3D com um *software* implementado com o método SFM requer muito esforço manual do usuário. Desde o processo de captura das imagens, que é feita com fotos, até a seleção das melhores poses para a obtenção do modelo reconstruído depende do "olho clínico" deste usuário e, muitas vezes, da sorte de obter as imagens correta.

Mesmo que o usuário seja conhecedor do método, em ambientes de grandes dimensões esta tarefa pode ser árdua e pode desencorajá-lo a executá-la.

Os *softwares* mais populares, como os citados na subseção 2.7, necessitam sempre de um conjunto de fotos já selecionados que contenham os elementos fundamentais para a detecção de pontos de correspondência (subseção 2.2) e não analisam nenhum detalhe antes de tentarem executar a reconstrução.

Como exposto, o propósito deste trabalho é investigar, implementar e validar um protótipo de sistema que permita a geração automática de um conjunto de imagens adequado (*keyframes*) para a etapa de construção de cenários virtuais, baseado no algoritmo de reconstrução por SFM.

Sua utilização permite selecionar um vídeo de entrada e, com a aplicação dos módulos de diferença de *pixels* e borramento, individualmente ou combinados, obter um conjunto de imagens (*keyframes*) que sejam satisfatórios para uma reconstrução em 3D pelo método SFM, por meio de um *software* que utilize este tipo de algoritmo.

Considerou-se que os *frames* descartados são aqueles que possuem pouca diferença de movimento de câmera, o que poderia caracterizar um *short baseline* (como descrito na subseção 3.3) ou aqueles *frames* que estiverem abaixo do limiar de borramento estipulado pelo usuário no início do processo de seleção dos *frames*.

A seleção de um *frame* borrado prejudica o processo de triangulação, dificultando o rastreamento de pontos característicos da imagem, uma vez que os *pixels* da imagem passam a ter cores similares ao longo de toda a imagem.

Nem todas as características do método SFM foram analisadas, pois a ideia central deste trabalho não é a realização de uma reconstrução completa. Esta reconstrução será realizada pelo *software* escolhido pelo usuário, estando representada na etapa seguinte do *pipeline* de reconstrução como apresentado no Capítulo 1.

4.1. ESTRUTURA DO PROTÓTIPO DESENVOLVIDO

O protótipo em *software*, cujo principal objetivo é a extração dos *keyframes* de um arquivo de vídeo, foi desenvolvido para oferecer como resultado final um conjunto de imagens que serão inseridas em outro *software* de reconstrução que utilize o método SFM.

Na Figura 19 são mostrados os principais módulos do protótipo, onde a entrada é um arquivo de vídeo de um objeto ou uma cena e a saída é um conjunto de *keyframes* para serem enviados a um *software* de reconstrução pelo método SFM.

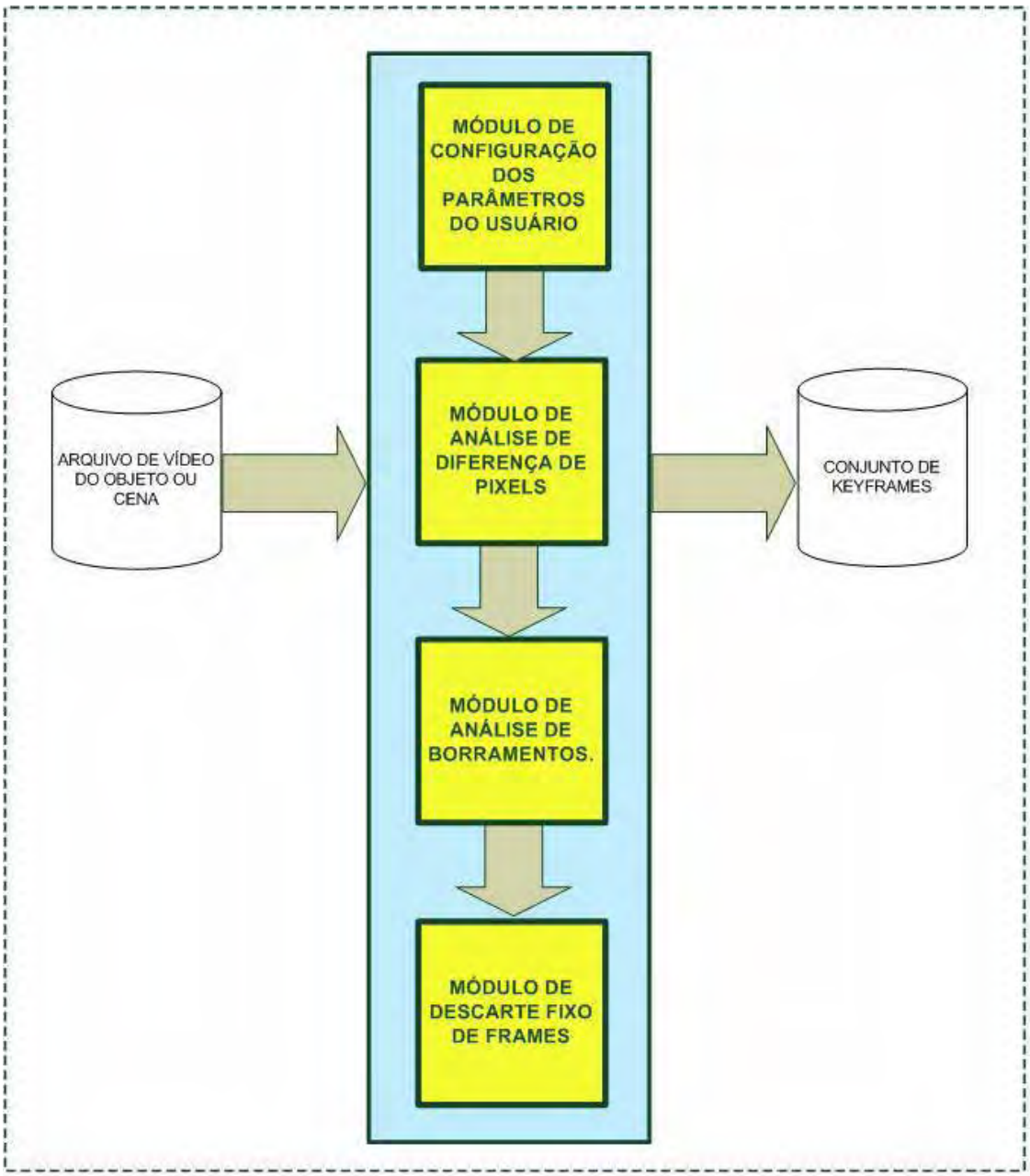


Figura 19 – Módulos principais do *software*.

Inicialmente ele obterá do usuário as informações necessárias ao seu funcionamento, configurando-o quanto a utilização dos filtros disponíveis (borramentos ou saltos por diferença de *pixels*). É possível obter os valores médios para utilização dos filtros no vídeo.

Os filtros podem ser utilizados individualmente, juntos, ou mesmo não serem utilizados, realizando apenas um salto de *frames* a partir de um número fixo especificado.

Em seguida, o vídeo é percorrido *frame* a *frame* e, de acordo com os parâmetros selecionados, é realizada a seleção dos *keyframes*.

O conjunto de *keyframes* obtido ao final da execução será gravado em arquivos individuais, em uma pasta especificada nas configurações, gerando um arquivo de *log*[5] contendo as informações de diferenças obtidas nos *pixels* e borramentos de cada imagem selecionada.

Ao término do processamento do vídeo são exibidas, ao usuário, informações sobre a quantidade total de *frames* do vídeo e da quantidade de *frames* selecionados. Além disto, ainda são apresentadas as informações, quando especificado pelo usuário suas utilizações, da quantidade de *frames* descartados pelo filtro de diferença de *pixels* e da quantidade de *frames* descartados pelo filtro de borramentos.

Ao terminar a execução os *frames* estarão prontos para serem enviados ao *software* de reconstrução que utilize o método SFM desejado.

4.2. IMPLEMENTAÇÃO DO PROTÓTIPO

Para o desenvolvimento do *software* utilizou-se a linguagem C++. A escolha da linguagem deu-se por facilidade de encontrar bibliotecas necessárias às operações de vídeo e imagens necessárias ao projeto.

Como ferramenta de implementação foi utilizado o Microsoft Visual C++ 2010 Express, de versão gratuita para estudos fornecida pela desenvolvedora.

Para acesso aos vídeos e geração dos *frames* utilizou-se a biblioteca OpenCV 2.4 (WILLOW, 2010), de código aberto, para abrir, manipular e salvar imagens ou vídeos.

4.2.1. MÓDULO DE CONFIGURAÇÃO DOS PARÂMETROS DO USUÁRIO

Este módulo é implementado no início do *software* e sua função é obter os parâmetros de configurações para a realização da extração dos *keyframes*.

[5] Arquivo de *log* é uma expressão utilizada para descrever o processo de registro de eventos relevantes num sistema computacional.

O usuário deve especificar o nome do arquivo de vídeo, o nome de saída das imagens selecionadas, a pasta em que elas serão armazenadas e dos parâmetros para os filtros de diferença de *pixels* e borramento. É possível especificar ainda quais filtros serão utilizados.

O módulo pode determinar para o usuário os valores médios dos limiares das diferenças de *pixels* e borramento para auxiliá-lo na escolha empírica que deve ser realizada. Na Figura 20 é mostrado o fluxo detalhado do processo:

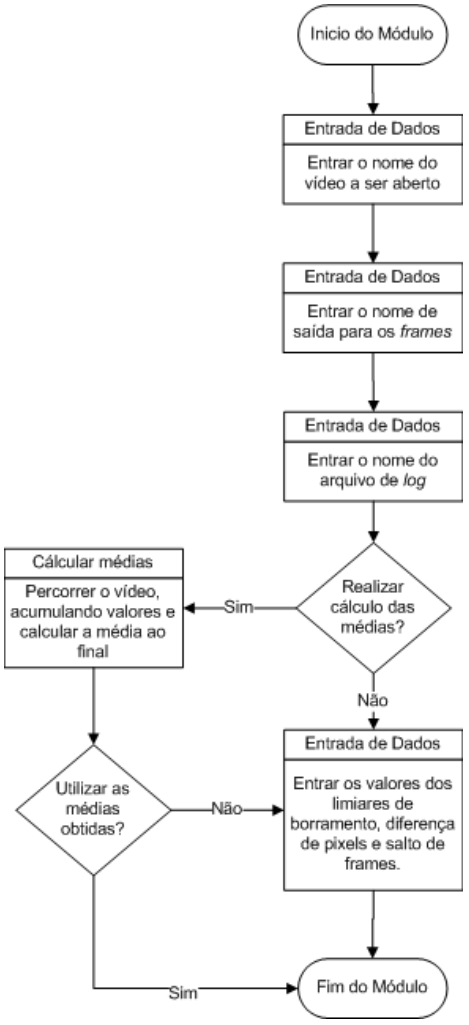


Figura 20 – Fluxograma do módulo de configuração.

Inicia-se o processo abrindo uma cópia do vídeo a ser avaliado, que não será exibido no dispositivo de saída do usuário. Percorrem-se todos os *frames* do vídeo acumulando os valores dos módulos de diferença de *pixels* e borramento e armazenando-os em variáveis apropriadas. A saída pode ser vista na Figura 20.

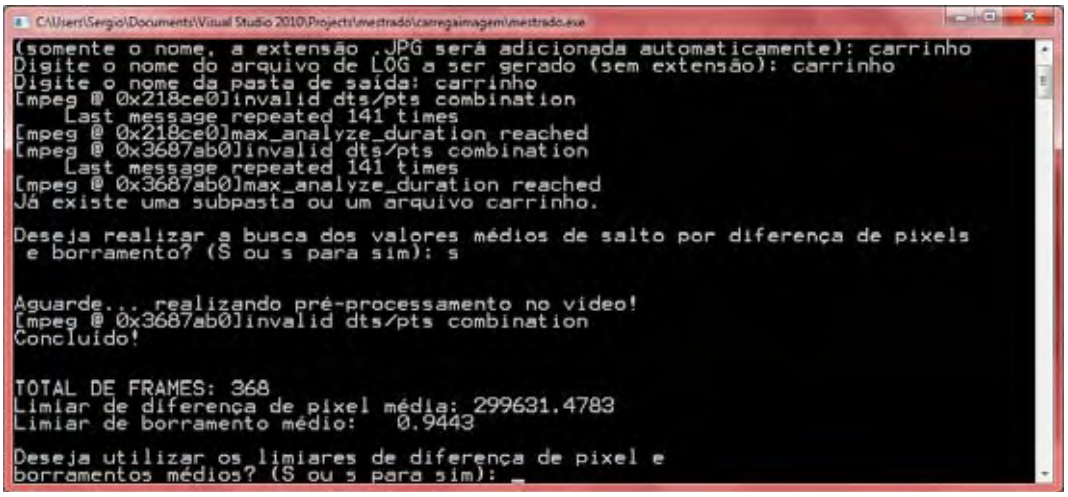


Figura 21 – Resultados do cálculo das médias de borramento e diferença de *pixels*.

Ao final é feita uma média aritmética simples para obter o valor médio de diferença dos *pixels* e limiar de borramento presentes no vídeo. Estes valores podem ser utilizados como parâmetros iniciais ou podem ser modificados de acordo com a necessidade do usuário. Na Figura 22 apresenta-se um trecho do código do protótipo.

```
if(cmedia=='S'||cmedia=='s') {
printf("\n\nAguarde... realizando pr%c-processamento no v%cdeo!\n",130,161);
while(frame = cvQueryFrame(tresh)){ //executa ate o fim do vídeo
width  = frame->width;
height = frame->height;
IplImage *frameprepproc = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 ); //define a imagem cinze
cvCvtColor( frame, frameprepproc, CV_RGB2GRAY ); //converte para cinza
IplImage *frameproxgray = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 ); //define segundo frame cinze
do {
frameprox = cvQueryFrame(tresh); //recupera o proximo frame e transforma em uma imagem na variavel frameprox
IplImage *frameproxgray = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 ); //define segundo frame cinze
 if(!frameprox) //se não obter o frame termina
    break;
cvCvtColor( frameprox, frameproxgray, CV_RGB2GRAY ); //converte para cinza o segundo frame
//calcular a diferença de grayscale de duas imagens
IplImage *framegraydif = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
cvAbsDiff (frameprepproc, frameproxgray, framegraydif);
sugilumina=sugilumina+countNonZero(framegraydif);
i++;
sugborra=sugborra+espectro(frameprox);
if (i%4==1) //mostrar uma animação de tempo enquanto processa
```

Figura 22 – Trecho de código com o cálculo das médias dos filtros de borramento e diferença de *pixels*.

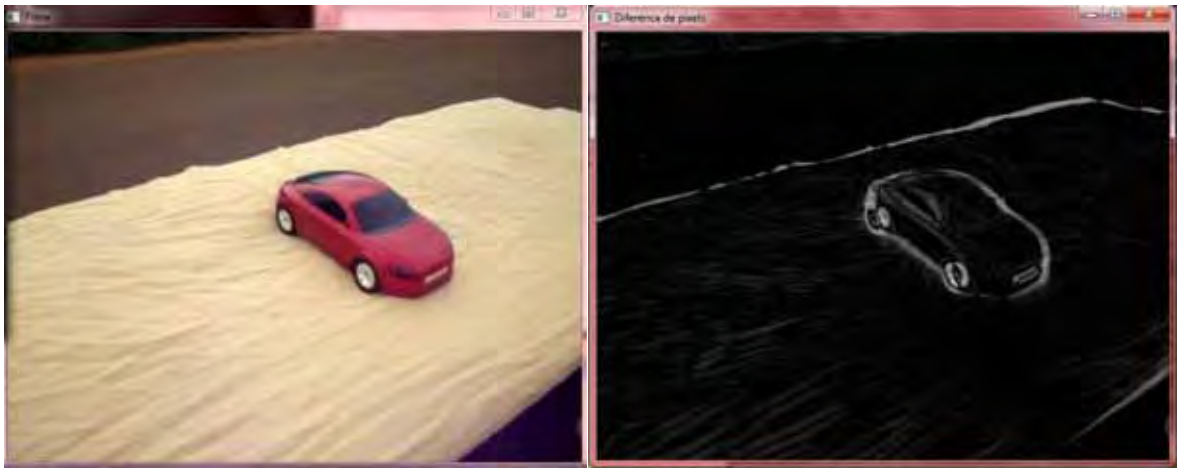
4.2.2. FILTRO DE DIFERENÇA DE *PIXELS*

O filtro de diferença de *pixels* realiza a comparação das quantidades de *pixels* que sofreram alterações, de um *frame* para outro, para determinar se houve uma mudança significativa da posição da câmera ou não, o que não justifica sua escolha como um *keyframe*. Neste último caso, onde o deslocamento de *pixels* de um *frame* para outro é pequeno, teremos *frames* quase idênticos e não será possível ao método de reconstrução SFM a obtenção da *baseline*, explicada na subseção 3.3. Isto implicará na criação de um *short baseline*, ou uma linha base muito pequena, o que pode não ser muito significativo para o método.

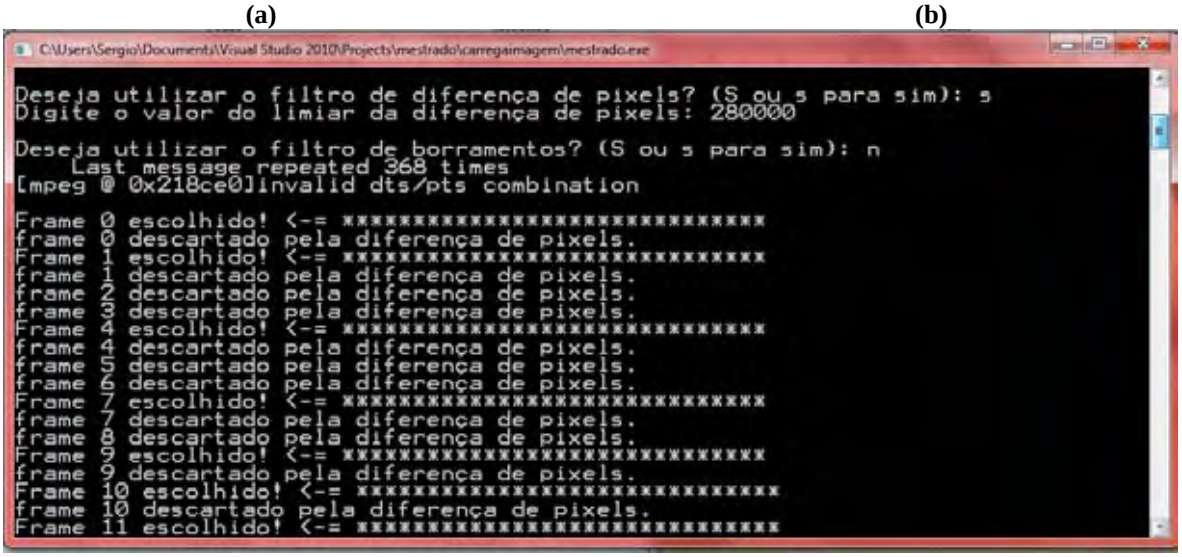
A imagem original é convertida em escala preto e branco e então comparada com a imagem selecionada anteriormente como *keyframe* e, de acordo com um limiar definido pelo usuário, poderá ser selecionada e enviada para apreciação do filtro seguinte ou descartada.

Caso seja descartado, um novo *frame* é enviado à função para as mesmas comparações. Este *frame* será o próximo *frame* da sequência, permanecendo o primeiro *frame* inicialmente selecionado como o *frame* base da comparação.

Na Figura 23 podemos ver um *frame*, uma imagem em escala cinza que representa a diferença entre dois *frames* e um exemplo em execução do protótipo.



(a) **(b)**



(c)

Figura 23 – Filtro de descarte por diferença de *pixels* em execução.
O *frame* a ser comparado (a) é convertido em escala cinza e subtraído do *frame* base também em escala cinza. O resultado (b) é uma imagem com traços brancos, onde, superado o limiar estipulado, será selecionada. Em (c) pode-se ver a tela em que o usuário acompanha a execução do filtro.

Na Figura 25 é apresentado o fluxo do módulo de diferença de *pixels*.

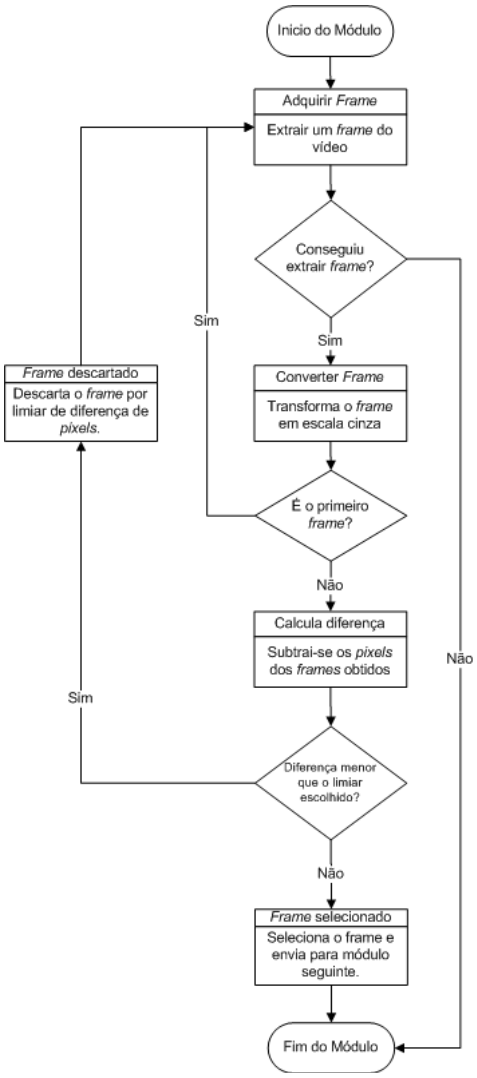


Figura 24 – Fluxograma do módulo de descarte por diferença de *pixels*.

Se o *frame* for selecionado, ele será enviado à etapa seguinte e passará a ser, inicialmente, a base para a próxima comparação na sequência do programa. Na Figura 25 apresenta-se um trecho do código do protótipo.

```
do {
if((ublur=='s'||ublur=='S')||(uilumina=='s'||uilumina=='S')) //se utilizar qualquer filtro
    frameprox = cvQueryFrame(capture); //recupera o proximo frame e transforma em uma imagem na variavel frameprox
IplImage *frameproxgray = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 ); //define segundo frame cinza

 if(!frameprox||(uilumina!='S'&&uilumina!='s')) //se não obter o frame ou nao for utilizar o filtro termina
    break;

cvCvtColor( frameprox, frameproxgray, CV_RGB2GRAY ); //converte para cinza o segundo frame
//calcular a diferença de grayscale de duas imagens
IplImage *framegraydif = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
cvAbsDiff (framegray, frameproxgray, framegraydif);
qtddif=countNonZero(framegraydif);
if(qtddif<=treshilumina) {
printf("\nframe %d descartado pela diferen%ca de pixels.",i,135);
i++;
dilumina++;
}
cvShowImage("Filme",frame); //mostra a imagem
}while(qtddif<=treshilumina); //fim da procura pela diferenca de pixels
```

Figura 25 – Trecho de código da comparação da diferença de *pixels*.

4.2.3. FILTRO DE BORRAMENTO

Este filtro realiza uma checagem do nível de borramento (*motion blur*), ocasionado normalmente pela rápida movimentação da câmera, o qual pode provocar efeitos indesejados na reconstrução. A base para a construção desta função está descrita na subseção 3.3.

Inicialmente, ao receber a imagem, será criada uma imagem em escala de cinza (tal como no filtro de diferença de *pixels*). Em seguida, esta imagem é submetida ao método de detecção de borramento descrito na seção 3.4.

Para detectar se uma imagem está borrada é feito o cálculo da força espectral (*power spectrum*) da imagem, onde o resultado é um valor entre 0 e 1, que é o valor médio do borramento da imagem como um todo.

Quanto menor este valor, menor o índice de borramento da imagem. Comparando-se este valor com o limiar do usuário (que pode ser obtido pela média no início do processamento ou estipulado empiricamente) é possível selecionar ou descartar esta imagem.

Descartando-se a imagem, aguarda-se para receber uma nova imagem onde será realizado o cálculo novamente. Em caso de aproveitamento, ela será enviada à etapa seguinte. Na Figura 26 é apresentado o fluxo do módulo de filtro de borramentos, enquanto na Figura 27 pode-se ver um trecho de código onde é feito o cálculo pela busca do borramento.

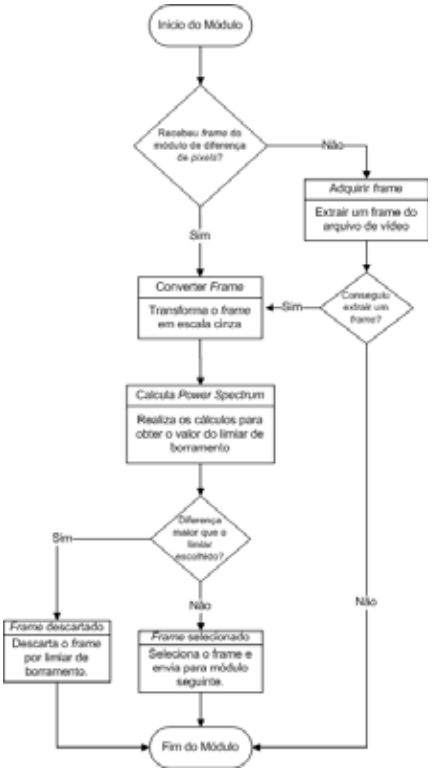


Figura 26 – Fluxo do módulo de borramentos.

```
//printf("\nApos a transformada de fourier\n");
for( j = 0; j < 4; j++ )
{
for( i = 0; i < 4; i++ )
{ //ver valores apos a transformada
Re = ((double*)(Fourier->imageData +
Fourier->widthStep*j))[i*2];
Im = ((double*)(Fourier->imageData +
Fourier->widthStep*j))[i*2+1];
//printf("(%.2f,%.2fi) ", Re, Im);
}
//printf("\n");
}
// Dividir Fourier em partes real e imaginária
cvSplit( Fourier, image_Re, image_Im, 0, 0 );
//Calcular a magnitude do espectro Mag = sqrt(Re^2 + Im^2)
cvPow( image_Re, image_Re, 2.0);
cvPow( image_Im, image_Im, 2.0);
cvAdd( image_Re, image_Im, image_Re);
cvPow( image_Re, image_Re, 0.5 );

// Calcular log(1 + Mag)
cvAddS( image_Re, cvScalar(1.0), image_Re ); // 1 + Mag
cvLog( image_Re, image_Re ); // log(1 + Mag)
```

Figura 27 – Trecho do código do filtro de borramentos.

4.2.4. FUNÇÃO DE SALTO DE *FRAMES*

Esta função realiza o salto de um determinado número de *frames* de acordo com a escolha do usuário. Utilizando-se o valor 1, serão analisados todos os *frames* do vídeo. Quando este valor é um número maior do que 1, os *frames*, após serem selecionados por qualquer um dos filtros do protótipo, serão saltados na quantidade de vezes deste valor.

Caso não tenha *frames* para serem retornados, por exemplo, quando atingir o fim do arquivo de vídeo, a função retornará um valor nulo, que será analisado por quem a acionou e reconhecer que não existem mais *frames* disponíveis. Um trecho do código é mostrado na Figura 28.

```
IplImage* skipNFrames(CvCapture* capture, int n) //salto de franes ***************************
{
    for(int i = 1; i < n; i++)
    {
        if(cvQueryFrame(capture) == NULL)
        {
            return NULL;
        }
    }

    return cvQueryFrame(capture);
}
//*****************************************************************
```

Figura 28 – Função de salto de *frames*.

4.3. CONSIDERAÇÕES FINAIS

Neste capítulo apresentou-se o processo de desenvolvimento do protótipo de *software* de seleção automática de *keyframes*, seus módulos e principais trechos de códigos desenvolvidos.

Foram mostradas as principais funções do protótipo, iniciando pelo módulo de configuração, que permite ao usuário a seleção do vídeo desejado, qual pasta salvar o resultado e o nome dos arquivos gerados. Também é possível escolher quais dos filtros serão utilizados (individualmente), onde é possível ainda a estipulação dos valores empíricos dos limiares a serem considerados.

Em seguida foi apresentado o detalhamento do filtro de diferença de *pixels*, o filtro de borramentos e o filtro de salto de *frames*, que compõe a estrutura principal do protótipo.

5. TESTES E ANÁLISES DE RESULTADOS

Neste capítulo são descritos os testes realizados com o protótipo do *software* de extração automática de *keyframes*, além de analisados os resultados obtidos. Os testes foram realizados a fim de obter a comprovação de que os *keyframes* selecionados possam realizar uma reconstrução com resultados satisfatórios em comparação à utilização de todos os *frames* ou de uma seleção manual.

Foram realizados dois tipos de testes para verificar a eficácia do protótipo de extração de *keyframes*. No primeiro teste foram realizadas reconstruções com os *softwares* Autodesk 123D Catch (AUTODESK, 2012) e Visual SFM (WU, 2011), onde foram comparados os números de pontos e faces gerados e observação visual da qualidade da reconstrução obtida. No segundo teste foi utilizado o *software* Visual SFM (WU, 2011) e comparado o tempo de processamento para realizar uma reconstrução com todos os *frames* do vídeo e apenas com os *keyframes* selecionados pelo protótipo.

Todos os testes foram realizados utilizando o mesmo *hardware* descrito nesta seção.

O *hardware* utilizado foi um *notebook* Dell Inspiron 5010 dotado de um processador Intel Core i5 M450, com 2 núcleos de 2.40Mhz, 4Gb de memória RAM, com sistema operacional Windows 7 Ultimate, 64bits.

Para captura dos vídeos foram utilizadas: uma câmera Sony H9, que gera vídeos em arquivos *Mpeg* com resolução de 640x480 *pixels* e uma câmera Nikon Coolpix S3100, que gera vídeos *Avi* com resolução de 1280x720 *pixels*.

Para realizar as reconstruções nos testes foram selecionados os *softwares* Autodesk 123D Catch (AUTODESK, 2012) e Visual SFM (WU, 2011) descritos na subseção 2.7.5 e subseção 2.7.1 respectivamente.

A escolha do Autodesk 123D Catch (AUTODESK, 2012) foi feita devido ao resultado final da reconstrução estar em forma de uma malha de triângulos, texturizados com as imagens obtidas dos *keyframes*. Este formato é totalmente compatível com o *software* do estúdio virtual, utilizado para a construção da cena virtual.

Já a escolha do Visual SFM (WU, 2011), apesar de não resultar em uma malha poligonal, se justifica por ser uma aplicação que pode ser executada localmente e resulta em uma nuvem de pontos, que podem ser comparados com os pontos resultantes na reconstrução com o Autodesk 123D Catch (AUTODESK, 2012). Por executar localmente, também é

possível apurar o tempo gasto para executar a reconstrução, pois não depende do uso de servidores existentes na internet, como o Autodesk 123D Catch.

Para realizar a contagem de pontos e triângulos foi utilizado o *software* Meshlab (CIGNONI et al., 2008) na versão 1.3.2, da Visual Computing Labs, ISTI – CNR. Ele é gratuito e permite manipulação de malhas, pontos e texturas, bem como trabalha com uma grande variedade de formatos de arquivos.

5.1. NÚMERO DE PONTOS E POLÍGONOS

No primeiro teste foram capturados diferentes tipos de vídeos para serem aplicados no protótipo desenvolvido. Foi escolhido um objeto plástico, um objeto de gesso, uma pessoa e dois ambientes fechados, um com largura maior do que altura e outro com altura maior do que largura.

Foram utilizadas as duas câmeras de vídeo para verificar se a resolução e aspecto de vídeos diferentes entre as reconstruções influenciariam no desempenho do filtro.

Nos objetos e na pessoa que foram reconstruídos, a trajetória aplicada à câmera seguiu sempre o sentido horário, capturando o objeto com uma volta de 360º em torno do mesmo, com uma inclinação de aproximadamente 15º para baixo.

Nos ambientes reconstruídos aplicou-se uma trajetória diferente, em forma de “S” com a última passada de captura com aproximadamente 45º para baixo. Esta trajetória será melhor detalhada na sequência dos testes.

5.1.1. CARRINHO

O objeto reconstruído Carrinho é feito de plástico, com pintura metálica. Foi utilizado um vídeo no formato *MPegG* de 12 segundos, realizando uma volta completa de 360° ao seu redor em um ambiente fechado com iluminação artificial branca.

O limiar de diferença de *pixels* utilizado foi de 285000 e o limiar de borramento utilizado foi de 0,6, determinados empiricamente.

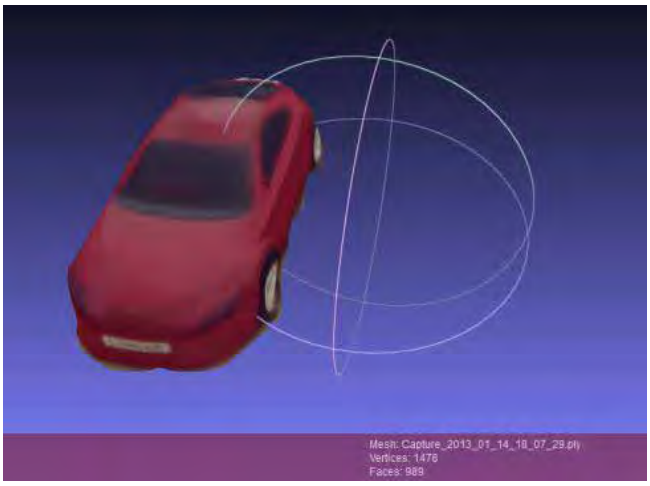
A trajetória de câmera aplicada foi a circular, em sentido horário, até completar 360° em torno do objeto, situado fixo no centro.

Na

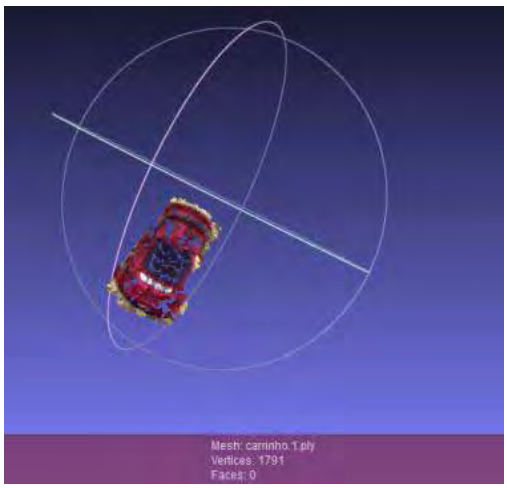
Figura 29 são mostradas algumas imagens da reconstrução nos diversos *softwares* utilizados.

(a)

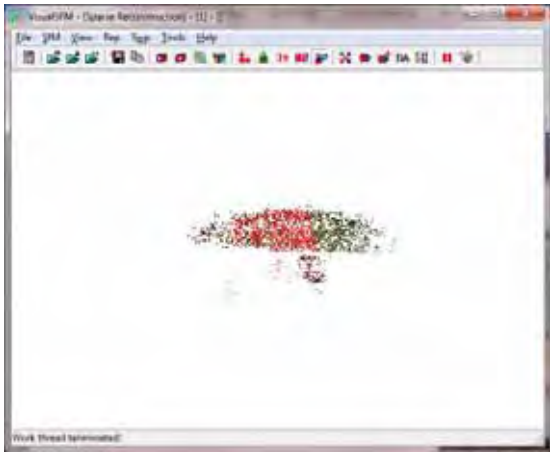
(b)



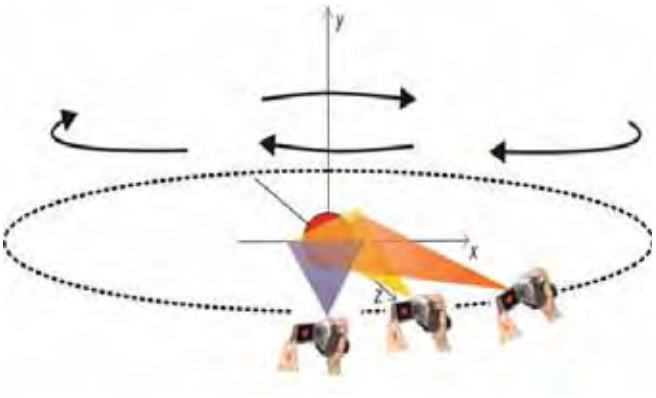
(c)



(d)



(e)



(f)

Figura 29 – Reconstrução Carrinho.
Em (a) um *keyframe* selecionado. Em (b) reconstrução no Autodesk 123D Catch, é possível ver a trajetória de câmeras . Em (c) reconstrução do Autodesk 123D Catch no Meshlab, em (d) a reconstrução do Visual SFM vista no Meshlab. Em (e) a tela do Visual SFM e os pontos gerados. Em (f) a trajetória de câmera utilizada na reconstrução.

5.1.2. ESFINGE

A reconstrução Esfinge foi obtida a partir de imagens de um objeto de gesso com pintura dourada metálica em sua maioria. Foi utilizado um vídeo no formato *MPegG* de 27 segundos, em um ambiente fechado com iluminação artificial amarela e um foco de luz branca incidindo diretamente sobre ele, acompanhando o movimento da câmera.

O limiar empírico para os filtros de diferença de *pixels* e de borramento foram 260000 e 0,85, respectivamente. A trajetória da câmera foi circular, 360° em sentido horário.

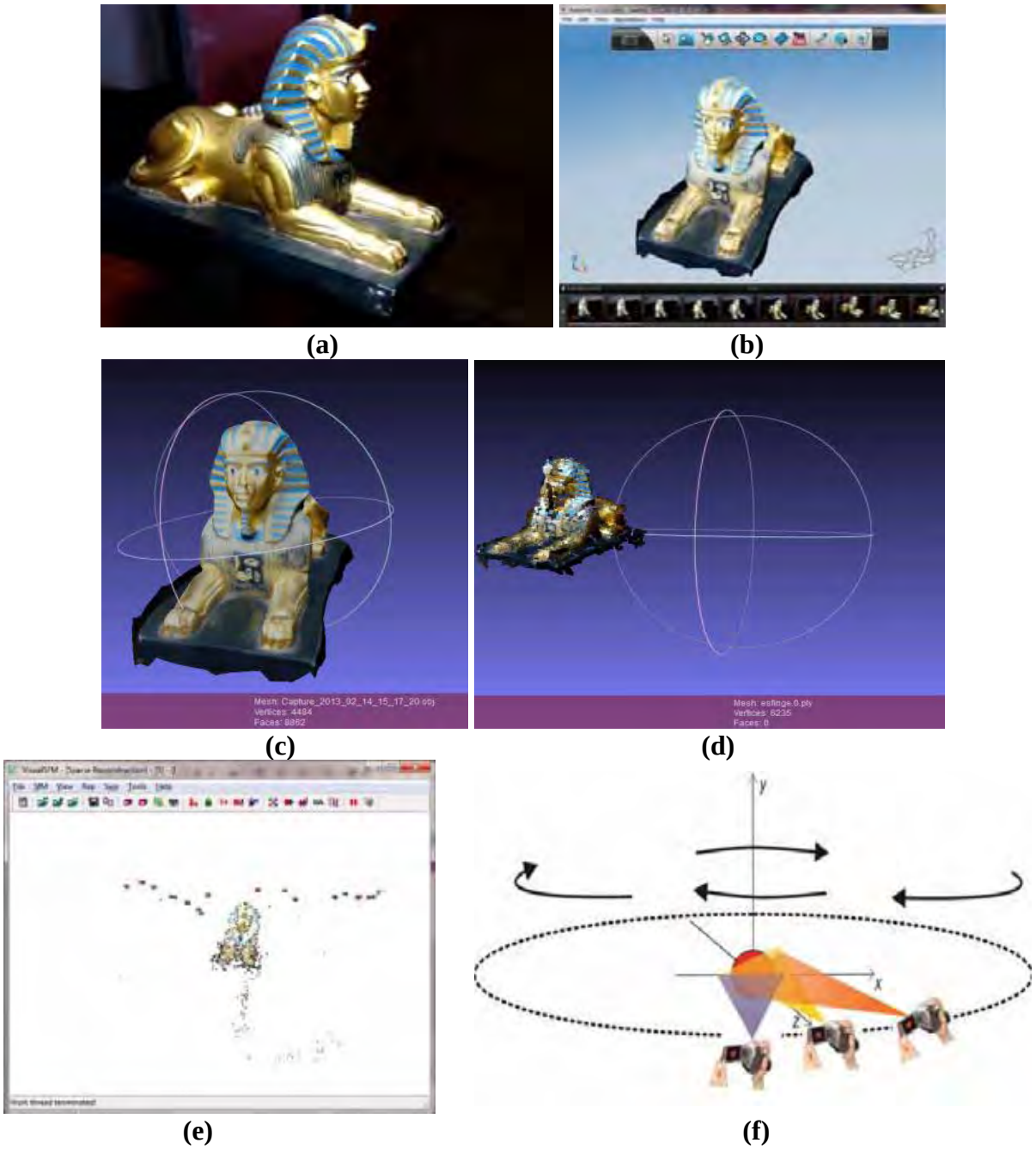


Figura 30 – Reconstrução Esfinge.
Em (a) um *keyframe* selecionado. Em (b) a tela do Autodesk 123D Catch. Em (c) reconstrução do Autodesk 123D Catch vista no Meshlab. Em (d) reconstrução do Visual SFM vista no Meshlab e em (e) a tela do Visual SFM. Em (f) o movimento de câmera utilizado.

5.1.3. SERGIO

O objeto Sergio exibe uma reconstrução realizada de uma pessoa. Foi utilizado um vídeo no formato MPegG de 22 segundos, realizando uma volta completa de 360° ao seu redor, em um ambiente aberto, com iluminação natural.

Apenas utilizado o limiar de borramento, com valor empírico de 0,85, analisando-se todos os *frames* do vídeo, sem nenhum salto. A reconstrução pode ser vista na Figura 31.

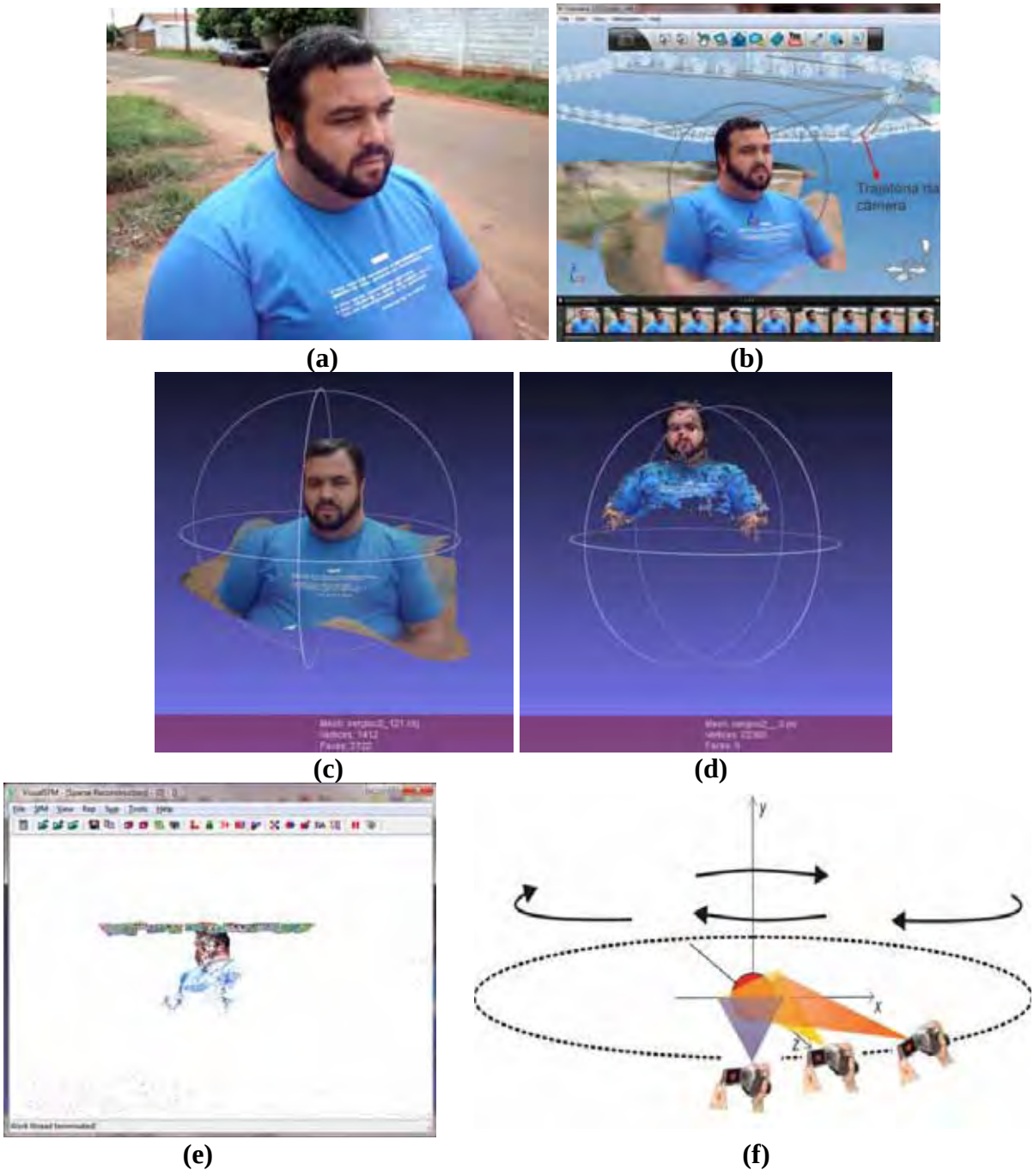


Figura 31 – Reconstrução Sergio.
Em (a) um *keyframe* selecionado. Em (b) a tela do Autodesk 123D Catch. Em (c) reconstrução do Autodesk 123D Catch vista no Meshlab. Em (d) reconstrução do Visual SFM vista no Meshlab e em (e) a tela do Visual SFM. Em (f) o movimento de câmera utilizado.

5.1.4. CPD

O objeto CPD é uma reconstrução realizada em ambiente fechado, com iluminação branca artificial. Foi utilizado um vídeo no formato *AVI* de 37 segundos, com resolução HD (1280 x 720 *pixels*), realizando uma meia volta, em formato de "2" de 180°.

A reconstrução foi feita utilizando o limiar de diferença dos *pixels* em 270000 e o limiar de borramento, com valor de 0,70, valores estes obtidos empiricamente.

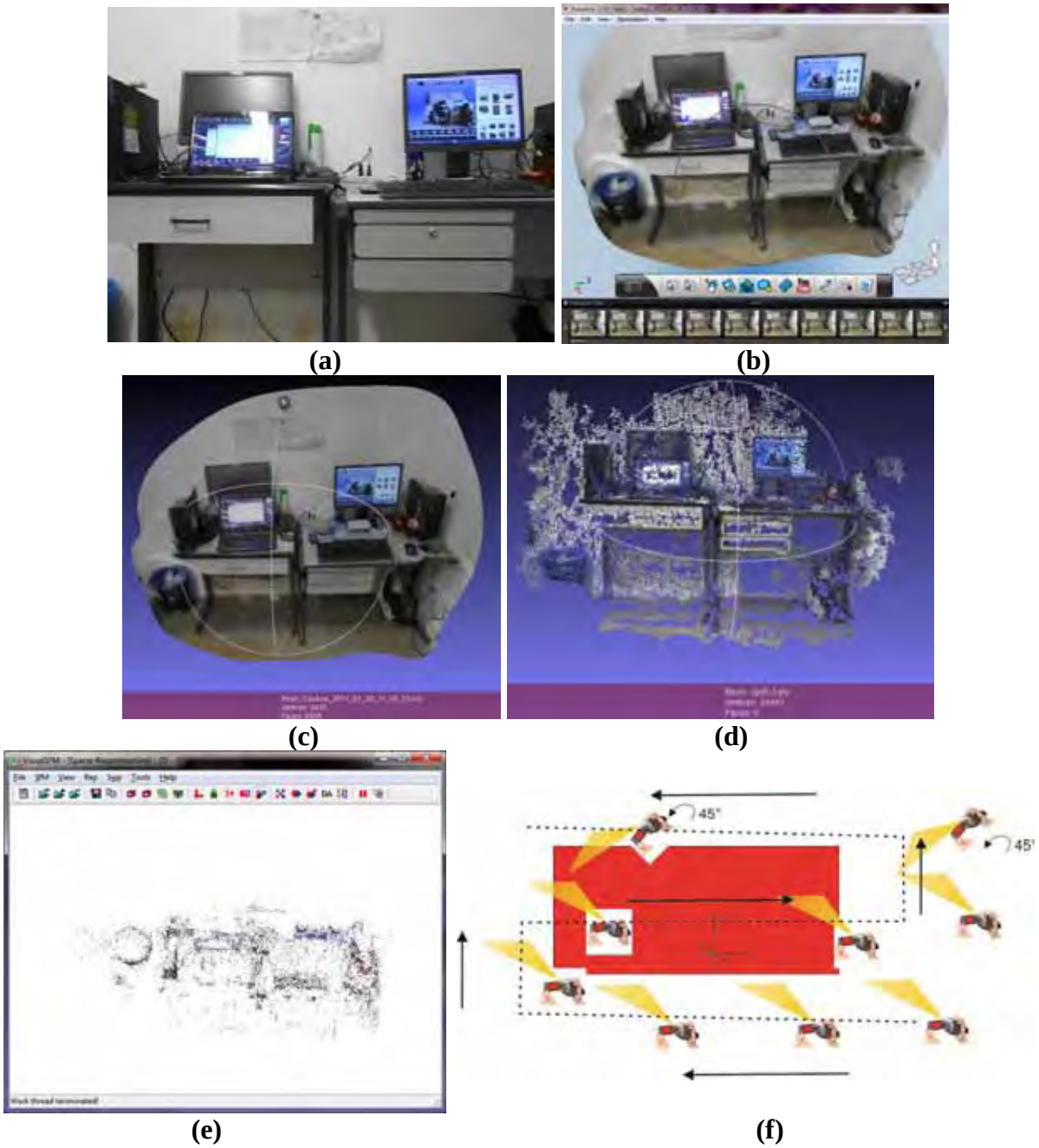


Figura 32 – Reconstrução CPD.
Em (a) um *keyframe* selecionado. Em (b) a tela do Autodesk 123D Catch. Em (c) reconstrução do Autodesk 123D Catch vista no Meshlab. Em (d) reconstrução do Visual SFM vista no Meshlab e em (e) a tela do Visual SFM. Em (f) o movimento de câmera utilizado.

Devido às propriedades do método SFM, as reconstruções de ambientes de grandes dimensões que obtiveram sucessos, precisaram ser capturadas seguindo um fluxo contínuo de vídeo, de maneira que os *frames* mantenham uma correlação entre os seus vizinhos.

A partir desta característica do método SFM obteve-se reconstruções satisfatórias seguindo uma trajetória de câmera em forma de "2", de baixo para cima, com a última linha de captura tendo a câmera inclinada em 45º para baixo, conforme visto na Figura 32 (f).

É possível realizar quantas sequências de "2" quanto sejam necessários até que toda a área do objeto seja capturada. O importante a destacar é que a elevação da câmera seja feita sempre nas extremidades e com movimentos retilíneos.

Apenas a última vez em que o movimento horizontal for ser feita, passando a câmera por toda a extensão do objeto, quando estiver capturando a região superior, deverá inclinar a câmera para baixo com o ângulo de 45º.

5.1.5. SERVIDOR

O objeto Servidor é uma reconstrução em um ambiente fechado, com iluminação artificial.

Foi utilizado um vídeo com resolução HD (1280x720 *pixels*), no formato *AVI,* de 21 segundos, seguindo a trajetória de câmera do "2" de filmagem, com 180º em torno do objeto.

O limiar de diferença de *pixels* utilizado foi de 270000 e o limiar de borramento utilizado foi de 0,7, ambos empiricamente determinados.

Esta reconstrução também utilizou a inclinação de 45° da câmera no momento de capturar a última sequência horizontal sobre o objeto. A reconstrução pode ser observada na Figura 33.

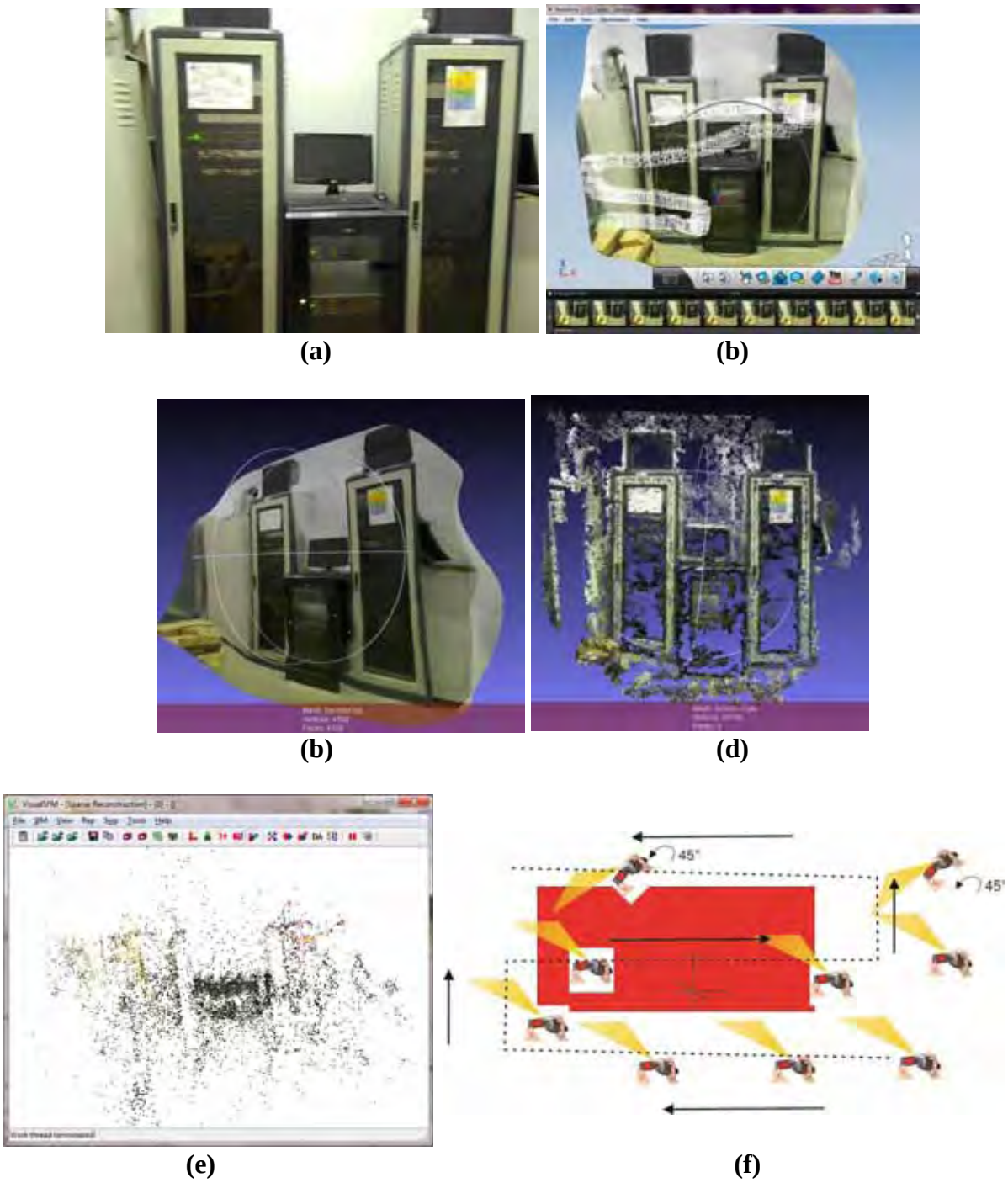


Figura 33 – Reconstrução Servidor.
Em (a) um *keyframe* selecionado. Em (b) a tela do Autodesk 123D Catch. Em (c) reconstrução do Autodesk 123D Catch vista no Meshlab. Em (d) reconstrução do Visual SFM vista no Meshlab e em (e) a tela do Visual SFM. Em (f) o movimento de câmera utilizado.

A Tabela 3 exibe os dados das reconstruções realizadas no primeiro experimento.

Tabela 3 – Resultados das reconstruções

Objeto	Câmera	Resolução	Aspecto	Nº de frames	Software	Pontos	Faces
Carrinho	Sony H9	640 x 480	3x4	47 (12,8%)	Autodesk 123D Catch	989	1478
					Visual SFM	1791	-
Esfinge	Sony H9	640 x 480	3x4	49 (6,0%)	Autodesk 123D Catch	4484	8862
					Visual SFM	6235	-
Sergio	Sony H9	640 x 480	3x4	121 (18,4%)	Autodesk 123D Catch	1412	2722
					Visual SFM	22300	-
CPD	Nikon S3100	1280 x 720	16x9	206 (18,6%)	Autodesk 123D Catch	4436	8669
					Visual SFM	24461	-
Servidor	Nikon S3100	1280 x 720	16x9	162 (26,1%)	Autodesk 123D Catch	4182	8129
					Visual SFM	25144	-

A Figura 34 mostra a relação entre as quantidades de frames que foram selecionados, descartados pelo filtro da diferença de pixels e pelo filtro de borramento.

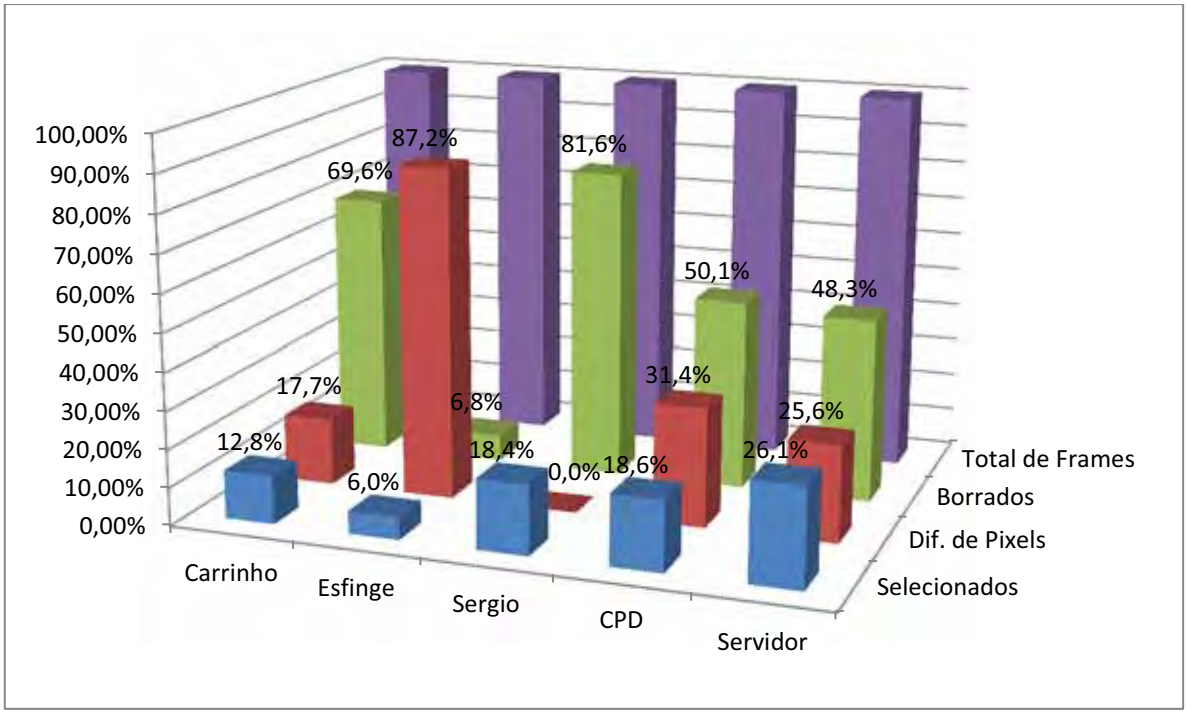


Figura 34 – Valores percentuais de frames nas reconstruções.

5.2. DIFERENÇA DE TEMPO EM PROCESSAMENTO LOCAL

Este teste consiste em avaliar o tempo de processamento necessário para realizar uma reconstrução. Para isso foram utilizados dois conjuntos de *frames* extraídos dos vídeos utilizados nos testes anteriores. No primeiro conjunto de *frames* foram utilizados todos os *frames* que compõe o vídeo e no segundo conjunto foram utilizados apenas os *frames* selecionados pelo protótipo.

Para realizar este teste foi necessário alterar uma propriedade de configuração do Visual SFM (WU, 2011) devido à grande quantidade de *frames* na reconstrução sem nenhum descarte.

No arquivo de configurações do Visual SFM, existe uma propriedade para estipular o máximo de pontos detectados dentro de uma mesma reconstrução. O padrão definido pelo autor do programa é de 100 pontos, o que para nosso teste foi considerado pequeno depois dos testes iniciais realizados. O valor foi alterado para 50000. Segundo as instruções no site do desenvolvedor, a quantidade máxima de pontos depende da memória da máquina, e nos testes realizados no *hardware* escolhido foram aceitáveis.

Os vídeos foram submetidos ao protótipo do *software* sem aplicar nenhum filtro, apenas para extração de todos os *frames*. Estes *frames* foram processados pelo Visual SFM (WU, 2011).

Com os dados obtidos com estas novas reconstruções, é possível avaliar as quantidades de pontos e o tempo que foi necessário para realizar a reconstrução.

A Tabela 4 exibe os dados obtidos nos testes.

Tabela 4 – Resultados das reconstruções com todos os *frames*

Objeto	Câmera	Resolução	Aspecto	Tempo com Todos os frames (segs)	Nº de Pontos todos os frames	Tempo com os Frames Seleciona-dos (segs)	Nº Pontos c/ frames seleciona-dos
Carrinho	Sony H9	640 x 480	4:3	5211	2031	188 (3,6%)	1791 (88,2%)
Esfinge	Sony H9	640 x 480	4:3	43751	14738	304 (0,7%)	8862 (60,1%)
Sergio	Sony H9	640 x 480	4:3	42250	23937	333 (0,79%)	22300 (93,2%)
CPD	Nikon S3100	1280 x 720	16:9	44568	13025	446 (1%)	8669 (66,5%)
Servidor	Nikon S3100	1280 x 720	16:9	28498	25660	295 (1%)	25144 (97,9%)

Na Figura 35 pode-se visualizar graficamente os dados da Tabela 4

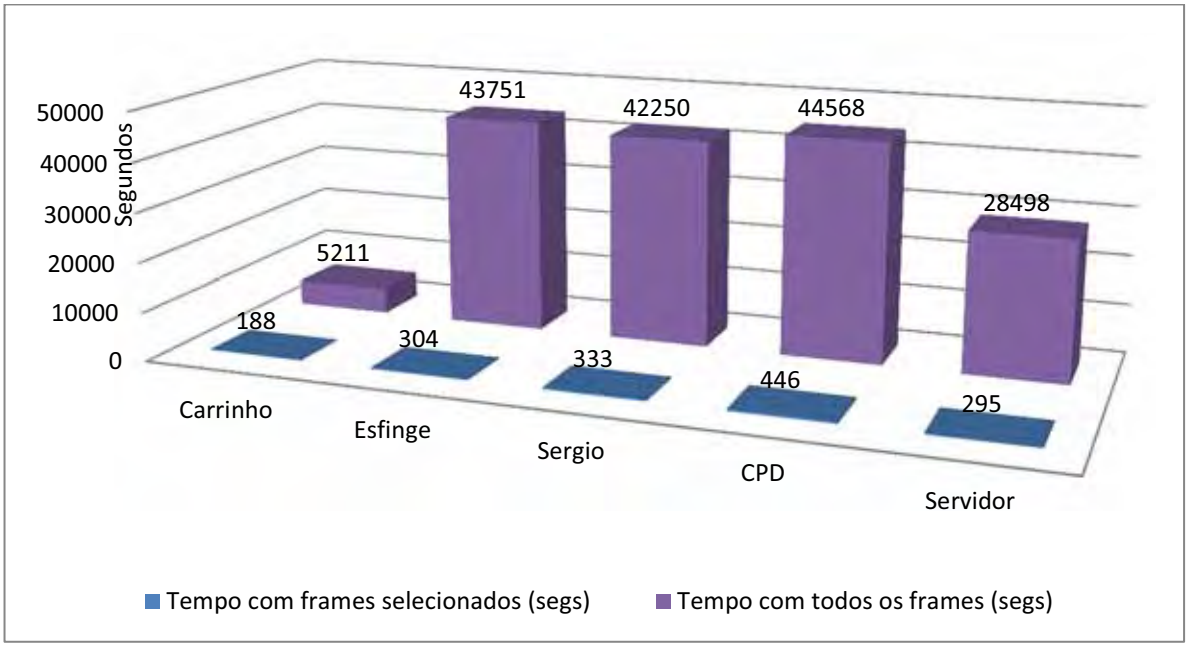
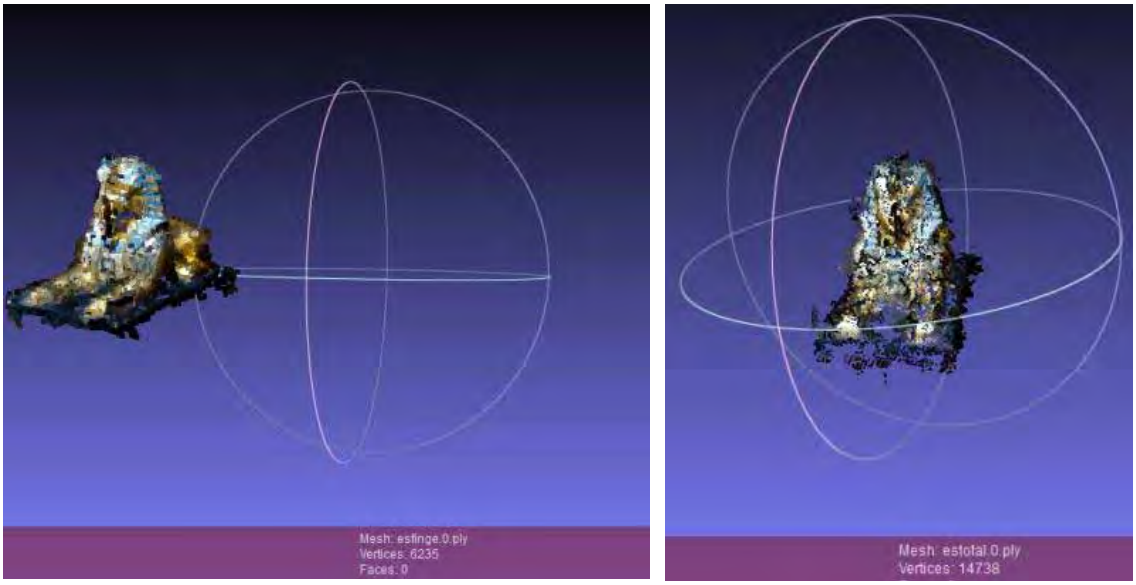


Figura 35 – Valores obtidos nas reconstruções com Visual SFM.

Com os testes realizados ficou claro a redução de tempo no processamento efetuado com todos os *frames* em relação ao processamento com apenas os *keyframes* selecionados pelo protótipo. Por exemplo, a reconstrução Esfinge, exibida na Figura 36, necessitou de mais de doze horas de processamento com todos os *frames* e com os *keyframes* foram precisos apenas cinco minutos.



(a) **(b)**

Figura 36 – Reconstruções com quantidades de *frames* diferentes.
Em (a) a reconstrução no Visual SFM do objeto Esfinge apenas com os *keyframe* selecionados. Em (b) reconstrução com todos os *frames* do vídeo também no Visual SFM.

5.3. TESTES COM UTILIZAÇÃO DOS MODELOS

Para comprovar a utilidade dos modelos gerados em ambientes 3D utilizou-se o sistema de estúdio virtual aumentado, desenvolvido por Sementille et al. (2012).

Este sistema carrega modelos reconstruídos em 3D para a construção de cenários virtuais em TV Digital, cinema ou qualquer estúdio que necessite auxiliar o diretor ou atores na pré-produção em tempo real no momento da captura de imagens.

O sistema possui diversas características que permitem a reprodução total dos controles existentes em estúdios virtuais de grande porte, como controle de níveis e cores de *chroma-key*, associação de objetos virtuais na cena em tempo real, total controle sobre manipulação do posicionamento dos objetos (rotação translação e escala) e trabalha com diversos formatos de arquivos.

Sementille et al. (2012 *apud* Gobbi et al. 2012) descreveu o sistema:

> Consiste na criação de cenários virtuais, bem como de quaisquer objetos virtuais tridimensionais, os quais podem ser integrados digitalmente às cenas capturadas em um estúdio real. Por meio de técnicas como o chroma-key, computação gráfica, realidade aumentada e a realidade virtual é possível flexibilizar a produção de conteúdo para a TV digital, cinema e internet, bem como baratear seu custo. (Sementille et al., 2012)

Na Figura 37 é possível ver o *layout* do sistema em execução, onde o fundo foi inserido por *chroma-key* e o objeto associado a um marcador.

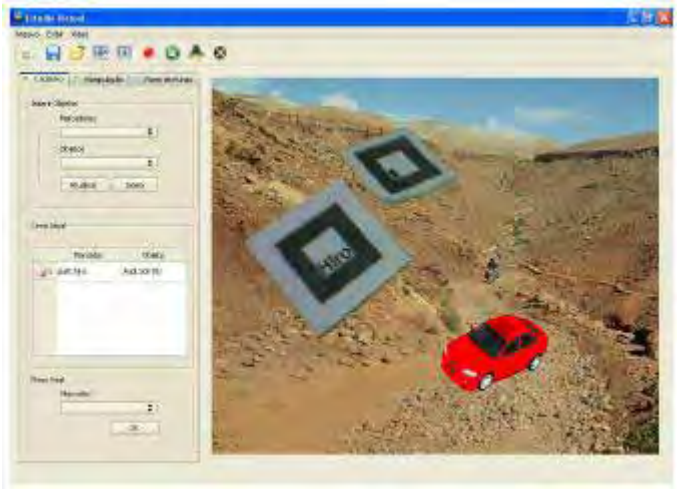
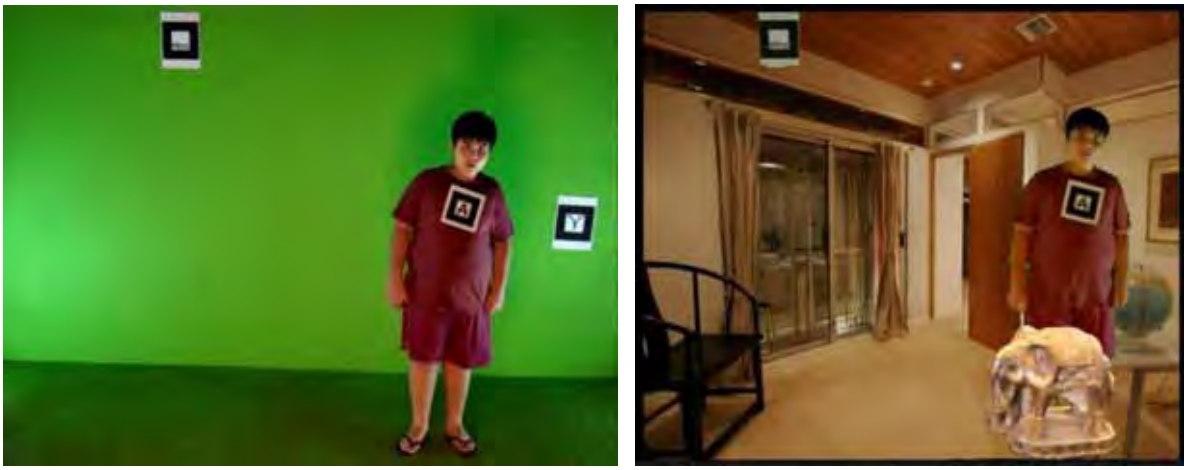


Figura 37 – Sistema de Estúdio Virtual Aumentado.
É possível ver os marcadores, o objeto virtual reconstruído e o fundo substituído pelo *chroma-key* no ambiente do *software* do estúdio virtual. Fonte: Campos et al (2010).

Os marcadores são utilizados para associar os objetos virtuais que serão apresentados na cena e compor o cenário virtual. Eles podem ser removidos da cena, quando são confeccionados com as mesmas cores de fundo utilizada no *chroma-key*, passando a ficar invisíveis em cena. Nos testes apresentados eles foram feitos propositalmente em preto e branco para demonstrar sua utilização.

Foram realizados dois testes no *software* do estúdio virtual aumentado para ilustrar a possibilidade de trabalhar com um vídeo gravado, como pós-produção, e outro teste com a captura de vídeo em tempo real.

No primeiro teste utilizou-se escala real, onde um ator foi filmado em um estúdio com a adição dos marcadores. O ator utilizava um marcador fixado no peito e outros dois marcadores foram colocados, um em cada lado, para permitirem a inserção dos objetos virtuais na cena, como visto na Figura 38.



(a) **(b)**

Figura 38 – Imagem do vídeo com objeto virtual.
O ator foi originalmente capturado em frente a um fundo verde (b). É possível ver os marcadores, um objeto virtual reconstruído (elefante) e o fundo substituído pelo *chroma-key* (b).

No segundo teste, com um vídeo em tempo real de captura, utilizou-se um modelo em escala reduzida, onde foram colocados vários marcadores em frente a um modelo em escala reduzida de um estúdio. Neste teste foi utilizado um modelo em escala com fundo azul, capturado em tempo real com resolução HD e vários objetos simultaneamente foram inseridos. Um dos marcadores foi colocado na cor do fundo azul para ilustrar que pode estar presente na cena e ser removido pelo filtro de *chroma-key* sem necessitar estar presente na cena final.

(a) **(b)**

Figura 39 – Imagem em tempo real com objetos virtuais na cena.
É possível ver alguns marcadores, o primeiro à esquerda desaparece quando o fundo do *chroma-key* é substituído. Em (a) a cena com o fundo removido e substituído por uma imagem. Em (b) cena sem o fundo removido e com os objetos nas mesmas posições.

5.4. CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os testes realizados com o protótipo proposto. Dois testes foram realizados. Primeiramente foram selecionados os *keyframes* de alguns vídeos e enviados ao Autodesk 123D Catch, que executa em servidores *web*, e os mesmos *keyframes* foram submetidos ao Visual SFM, executado em um computador local.

Os modelos reconstruídos possuem qualidades aceitáveis para utilização em ambientes de estúdios virtuais para composição de cenários virtuais.

No segundo teste utilizou-se o Visual SFM para verificar se houve ganho de tempo de processamento, com a utilização dos *keyframes* selecionados pelo protótipo em relação à utilização de todos os *frames* disponíveis nos vídeos. Nos testes realizados, as reconstruções passaram de várias horas (quando utilizados todos os *frames*) para poucos minutos (quando utilizado apenas os *keyframes* selecionados). Os resultados em número de pontos também foram consideravelmente menores, porém no resultado final da reconstrução não apresentaram diferença visual, podendo ser aplicados nos estúdios virtuais.

Realizou-se também um teste com os objetos gerados em um *software* de estúdio virtual aumentado para demonstrar a viabilidade da utilização dos modelos para composição de um cenário virtual, com um vídeo gravado para pós-produção e um vídeo capturado em tempo real. Em todos os casos, os testes obtiveram sucesso.

A manutenção das características das reconstruções dos objetos, que permitem sua utilização na construção de cenários virtuais, quando reconstruídos apenas com os *keyframes*

selecionados pelo protótipo e o ganho em tempo de processamento que foi proporcionado, em relação à reconstrução com utilização de todos os *frames* mostram a viabilidade de utilização do protótipo de extração automática de *keyframes* para reconstruções pelo método SFM.

6. CONCLUSÕES E TRABALHOS FUTUROS

A partir do trabalho apresentado, é possível concluir que o protótipo do *software* de extração automática de *keyframes* é viável para reconstruções que utilizam o método SFM. Com base nos testes realizados, demonstrou-se que, com uma quantidade adequada de *frames* corretamente selecionados há uma economia de tempo e processamento na etapa de reconstrução tridimensional.

Determinou-se, também, que para reconstruir ambientes de grandes dimensões é necessário utilizar uma trajetória de câmera em forma de “2”, mantendo um fluxo de vídeo constante durante a captura, como mostrado na seção 5.1.4.

Os testes realizados mostraram que a pré-seleção na escolha dos *frames* antes de realizar uma reconstrução resulta em objetos com menores erros em sua malha. Também verificou-se que há uma melhoria na malha obtida, quando são analisadas as informações de borramento e diferença de *pixels* para proporcionar um salto de *frames* eficiente. Obtém-se, assim, maiores taxas de acertos em comparação com escolhas manuais feitas pelo método tradicional ou com a utilização de todos os *frames*.

No melhor resultado obtido nos experimentos obteve-se uma redução de 94% no número de *frames* e no pior caso, uma redução de 75%.

Em relação ao tempo de processamento, necessitou-se de 0,7% do tempo no melhor caso e 3,6% do tempo no pior caso, comparando com a reconstrução com todos os *frames*.

Foi demonstrado que os modelos gerados (como na Figura 40) nos testes são adequados à utilização em estúdios virtuais e que puderam claramente ser utilizados em uma composição de cena virtual, utilizando um *software* específico para estúdios virtuais.

Figura 40 – Objeto reconstruído com os *frames* selecionados pelo protótipo de um vídeo.

Destacam-se, como trabalhos futuros, a elaboração de uma forma automática para o cálculo dos limiares de borramento e diferença de *pixels*, uma vez que os mesmos foram estabelecidos de forma empírica; a implementação de uma interface gráfica amigável para o *software*, seguindo as tendências de visualização dos sistemas operacionais modernos; e aperfeiçoamentos nos filtros de borramento e diferença de *pixels*, onde o objeto de interesse deve ser isolado do restante da imagem para ser efetuada uma análise mais precisa.

REFERÊNCIAS BIBLIOGRÁFICAS

APOLINÁRIO, Eduadrdo L.; TEICHRIEB, Veronica; KELNER, Judith. **RECONSTRUÇÃO 3D EM AMBIENTES COM LUZES ESTRUTURADAS.** II Workshop de Realidade Virtual e Aumentada. Recife. Nov. 2006.

AUMONT, Jacques; MARIE, Michel: **"Dicionário teórico e crítico de cinema"**, Tradução de Eloísa Araújo Ribeiro. Campinas-SP. Editora Papirus, 2003.

AUTODESK. **123D Catch.** Disponível em: <http://www.123dapp.com/catch>. 2012. Acesso em: 18 agosto 2012.

AZEVEDO, Eduardo; Conci, Aura. **Computação Gráfica: Teoria e Prática.** 2 ed. São Paulo. Editora Campus. 2003.

AZEVEDO, Tereza C. Souza; TAVARES, João Manoel R. S.; VAZ, Mário A. P. **Reconstrução e Caracterização de Estruturas Anatómicas Exteriores usando Visão Activa.** Relatório Interno do Laboratório de Óptica e Mecânica Experimental (LOME) Faculdade de Engenharia, Universidade do Porto, Porto, set. 2005.

BAY, Herbert; TUYTELAARS, Tinne; VAN GOOL, Luc. **SURF: Speeded Up Robust Features.** European conference on computer vision. p. 404-417. 2006.

BETHENCOURT, Aymeric; JAULIN, Luc. **3D Reconstruction Using Interval Methods on The Kinect Device Coupled With an IMU.** International Journal of Advanced Robotic Systems. Vol. 10, p. 93-103. 2013.

BJØRNSTAD, Jørgen B.; FEDREHEIM , Stein A. **Automatisk 3D-modellering ved hjelp av stillbilder.** 112f. Disseração (Mestrado em Ciências Matemáticas e Tecnologia) Norwegian University of Life Sciences. 2012.

CAMPOS, G. M.; MUKUDAI, L. M.; IWAMURA, V. S.; SEMENTILLE, A. C. **Sistema de Composição de Estúdios Virtuais Utilizando Técnicas de Realidade Aumentada**. In: Proceedings - XII Symposium on Virtual and Augmented Reality- SVR 2010, v.01, p.22 – 30. Natal, 2010

CIGNONI, P.; CALLIERI, M.; CORSINI, M.; DELLEPIAINE; GANOVELLI, F.; RANZUGLIA, G. **MeshLab: an open-source mesh processing tool.** Eutographics Italian Chapter Conference. V, Sacrano R. De Chiara, U. Erra (Eds.). Eurographics Association, pp. 129-136. Italia, 2008.

CLARK, J. J.; **Active photometric stereo.** IEEE International Conference on Computer Vision. p. 29-34. USA, 1992.

COURCHAY, J.; PONS, J.; MONASSE, P.; KERIVEN, R. **Dense and Accurate Spatio-Temporal Multi-View Stereovision**. Computer Vision ACCV 2009. Lectures Notes, p 11-22. 2010.

DELLAERT, Frank, et al.. **Structure From Motion without Correspondence,** Computer Vision and Pattern Recognition. Proceedings of IEEE Conference, v. 2, p. 557-564. 2000.

FULTON, John R. **Sensor Orientation in Image Sequence Analysis.** 299f. Tese (Mestrado em Filosofia) Melbourne University. Melbourn, 2007.

GOMES, Otávio F. M. **Processamento e Análise de Imagens Aplicados à Caracterização Automática de Materiais.** 141f. Dissertação (Mestrado em Ciências da Engenharia Metalúrgica) Pontifícia Universidade Católica do Rio de Janeiro (PUC/RIO). Rio de Janeiro. 2001.

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento Digital de Imagens.** 3. ed. Pearson Pretince Hall. São Paulo, 2010.

GRAU, O.; PRICE, M.; THOMAS, G. A. **Use of 3-D Techniques for Virtual Production**. BBC Research & Development, White Paper WHP 033, Londres, 2002.

HÄMING, Klaus; PETERS, Gabriele; **The Structure-From-Motion Reconstruction Pipeline – A Survey With Focus On Short Image Sequences**. Kybernetika. v. 46 n. 5 p. 926-937. Czech Republic, 2010.

HARTLEY, Richard; ZISSERMAN, Andrew. **Multiple view geometry in computer vision.** 2. ed. Cambridge University Press. Cambridge, 2004.

HARRIS, Chris.; STEPHENS, Mike; **A combined corner and edge detector.** Forth Alvey Vision Conference, University of Manchester, vol. 15, p. 147-151. England, 1988.

HUANG, Peng; HILTON, Adrian; STARCK, Jonathan. **Automatic 3D Video Summarization: Key Frame Extraction from Self-Similarity.** Fourth International Symposium on 3D Data Processing, Visualization and Transmission. Atlanta, jun. 2008.

KANNAN, P.; RAMAKRISHNAN, R.; **A Versatile Algorithm for Reconstruction of Sports Video Sequences.** IEEE International Conference on Computational Intelligence and Computing Research. Coimbatore, dez. 2010.

KERSTEN, Th; LINDSTAEDT, M.; MECHELKE, K.; ZOBEL, K. **Automatische 3D-Objektrekonstruktion aus unstrukturierten digitalen Bilddaten für Anwendungen in Architektur, Denkmalpflege und Archäologie.** Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation. Proceedings on CD-ROM, p 137-148. Potsdam, mar. 2012.

KERSTEN, Th; LINDSTAEDT, M. **Automatic 3D Object Reconstruction from Multiple Images for Architectural, Cultural Heritage and Archaeology Applocations Using Open-Source Software and Web Services.** International Journal of Photogrammetry, Remote Sensing and Geospatial Sciences. Heft 6. Jahrgang, 2012.

LOWE, David;. **Distinctive image features from scale-invariant keypoints**. International Journal of Computer Vision, n. 60, p. 91-110. 2004.

MARQUES, Clarissa C. S. C. **Um Sistema de Calibração de Câmera.** 90f. Dissertação (Mestrado em Computação Gráfica) Universidade Federal de Alagoas. Maceió. 2007.

MILLERSON, Gerald; OWENS, Jim. **Television Production.** 14. ed. Oxford: Focal Press, 2009.

OZANIAN, T., **Approaches for Stereo Matching**. A Review, Modeling,Identification and Control, vol. 16, no. 2, p. 65-94, 1995.

PAULA FILHO, WILSON DE PÁDUA. **Multimídia: Conceitos e Aplicações**. 2ª edição. Editora LTC. Rio de Janeiro, 2011.

POLLEFEYS, Marc. **Visual 3D Modeling from Images**. Tutorial Notes, University of North Carolina. Chapel Hill, USA, 2002.

POLLEFEYS, Marc, et al. **Visual modeling with a hand held camera.** Journal of Visualization and Computer Animation, v. 59, n. 13, p. 207-232. 2004.

QUEIROZ, José E.R, GOMES, Herman M. **Introdução ao Processamento Digital de Imagens**. Revista de Informática Teórica e Aplicada (RITA) Vol VIII, nº 1. (2001)

ROBERTO, Rafael. **Retificação cilíndrica: um método eficiente para a retificação de par de imagens.** 66f. Trabalho de conclusão de curso (Engenharia da Computação) Universidade Federal de Pernambuco. Recife. 2009.

SANTOS, Andréa; et al.. **Refinamento de Reconstrução 3D através de Seleção Apropriada de Keyframes.** VIII Workshop de Realidade Virtual e Aumentada. Uberaba. Nov. 2011.

SEMENTILLE, A. C. ; MARAR, J. F. ; SANCHES, S. R. R. ; YONEZAWA, Wilson M. ; CAVENAGHI, Marcos A. ; ALBINO, João Pedro ; ANDRADE, Ândrea Barreto de **. Inovação em Televisão Digital: A Aplicação de Realidade Aumentada e Virtual na Criação de Estúdios Virtuais.** In: Maria Cristina Gobbi; Osvando J. de Morais. (Org.). Televisão Digital na América Latina: avanços e perpectivas. 1ed: Sociedade Brasileira de Estudos interdisciplinares da Comunicação - INTERCOM, v. 2, p. 655-680. São Paulo, 2012.

VAN DENBERGH, F.; LALIOTI, V. **Software chroma keying in an immersive virtual environment**. In: South African Computer Journal, n. 24, p. 155-162, África do Sul, 1999.

WILLOW, Garage. **Opencv Library**. Disponível em <http://opencv.willowgarage.com>. 2010. Acesso em 18 de agosto de 2012.

WU, Changchang. **VisualSFM: A Visual Structure From Motion System**. Disponível em <http://homes.cs.washington.edu/~ccwu/vsfm/>. 2011. Acesso em 18 de agosto de 2012.

ANEXO I

```
//1. Create a default Win32 Console project
//2. Delete all files in default project
//3. Open Configuration Properties
//4. C/C++ -> General ->Additional Includes Directories
//                C:\OpenCV2.2\include;
//5. C/C++ -> Advanced ->Compiles As
//                Compile as C++ Code (/TP)
//6. Linker -> General ->Additional Library Directories
//                C:\OpenCV2.2\lib
//7. Linker -> Input -> Additional Dependencies
//                opencv_calib3d220d.lib
//                opencv_contrib220d.lib
//                opencv_core220d.lib
//                opencv_features2d220d.lib
//                opencv_ffmpeg220d.lib
//                opencv_flann220d.lib
//                opencv_gpu220d.lib
//                opencv_highgui220d.lib
//                opencv_imgproc220d.lib
//                opencv_legacy220d.lib
//                opencv_ml220d.lib
//                opencv_objdetect220d.lib
//                opencv_video220d.lib
//


// PROGRAMA DO MESTRADO
*****************************************************************
//Autor: Sérgio Carlos Portari Júnior
//Criado em: 06/05/2012
//Alterado em 29/01/2013

#include <opencv\cxcore.h>
#include <opencv\highgui.h>
#include <opencv\cv.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

//#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <iostream>


using namespace std;
using namespace cv;
IplImage* skipNFrames(CvCapture* capture, int n);

double espectro(IplImage *image = 0);
//gerar um log sobre os dados analisados do espectro
FILE *fptr; //ponteiro para arquivo de log

int main(void)
{
int i=1,fr=0,largura,altura,qtddif=0;
char comando[40],pasta[40],nome[40],j[35],nomelog[40],temp[40],ublur='a',uilumina='a';
char nomefilme[255],nomeimagens[255],filmesaida[255],autotresh='a',cmedia='a';
int qtframes=0,width, height,auxconv=0;
double treshborra=0,treshilumina=0; //limiares
```

```
double minretorno=0;
double sugborra=0;
int key = 0;
double fps=30,sugilumina=0;
int dilumina=0, dborra=0,faval=-1;

//entrada de dados
printf("          =-> SISTEMA DE EXTRA%c%cO AUTOM%cTICO DE KEYFRAMES PARA SFM <-
=\n\n\n",128,199,181);
printf("Digite o nome do arquivo de v%cdeo a ser aberto: ",161);
gets(nomefilme);
printf("Digite o nome das imagens dos frames extra%cdos\n(somente o nome, a extens%co
.JPG ser%c adicionada automaticamente): ",161,198,160);
gets(nomeimagens);
//printf("Digite o nome do filme de saida\n(somente o nome, a extensao .AVI será
adicionada automaticamente): "); //se quiser gerar um vídeo com os keyframes
//gets(filmesaida);
//printf("Digite a largura do filme de saida (320, 640, 800, 1024): "); //se quiser
alterar a resolução de saída
//scanf("%d",&largura);
//printf("Digite a altura do filme de saida (240, 480, 600, 768): ");
//scanf("%d",&altura);
fflush(stdin);
printf("Digite o nome do arquivo de LOG a ser gerado (sem extens%co): ",198);
fflush(stdin);
gets(nomelog);
printf("Digite o nome da pasta de sa%cda: ",161);
fflush(stdin);
gets(pasta);
strcpy(temp,pasta);
strcat(temp,"/");
strcat(temp,nomelog);
strcat(temp,".txt");
strcpy(nomelog,temp);
fflush(stdin);
char* video_name = nomefilme; //abre o vídeo
char* preproc = nomefilme; //abre uma cópia do vídeo para contagem do limiar

IplImage* frame = 0, *frameprox=0; //variáveis para os frames

CvCapture* capture = cvCaptureFromAVI(video_name); //busca um frame do arquivo de
vídeo

CvCapture* tresh = cvCaptureFromAVI(preproc); //busca um frame do arquivo de vídeo



if (!capture) { //se não capturou
       printf("Nenhum filme encontrado");
       return 0; }
else
{
//strcat(filmesaida,".avi"); //adiciona a extensão avi no arquivo do vídeo
//CvSize size = cvSize(largura,altura); //define a dimensão do vídeo

//CvVideoWriter* writer =
cvCreateVideoWriter(filmesaida,CV_FOURCC('D','I','V','X'),fps,size); //cria um arquivo
de vídeo para salvar nossos frames



//CRIAR PASTA
```

```
strcpy(comando,"md ");
strcat(comando,pasta);
system(comando); //cria pasta
fptr=fopen(nomelog,"w");  //w significa arquivo para gravação



fflush(stdin);
printf("\nDeseja realizar a busca dos valores m%cdios de salto por diferen%ca de
pixels\n e borramento? (S ou s para sim): ",130,135);
scanf("%c",&cmedia);
if(cmedia=='S'||cmedia=='s') {
printf("\n\nAguarde... realizando pr%c-processamento no v%cdeo!\n",130,161);
while(frame = cvQueryFrame(tresh)){ //executa ate o fim do vídeo
width  = frame->width;
height = frame->height;
IplImage *framepreproc = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
//define a imagem cinze
cvCvtColor( frame, framepreproc, CV_RGB2GRAY ); //converte para cinza
IplImage *frameproxgray = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
//define segundo frame cinze
do {
frameprox = cvQueryFrame(tresh); //recupera o proximo frame e transforma em uma imagem
na variavel frameprox
IplImage *frameproxgray = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
//define segundo frame cinze
 if(!frameprox) //se não obter o frame termina
    break;
cvCvtColor( frameprox, frameproxgray, CV_RGB2GRAY ); //converte para cinza o segundo
frame
//calcular a diferença de grayscale de duas imagens
IplImage *framegraydif = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
cvAbsDiff (framepreproc, frameproxgray, framegraydif);
sugilumina=sugilumina+countNonZero(framegraydif);
i++;
sugborra=sugborra+espectro(frameprox);
if (i%4==1) //mostrar uma animação de tempo enquanto processa
{
   printf("\r| %d",i);
}
else
{
  if (i%4==2)
{
   printf("\r/ %d",i);
}
  else
  {
  if (i%4==3)
  {
      printf("\r- %d",i);
   }
  else
  {
     printf("\r\\ %d",i);
  }
  }
}
}while(frameprox);
printf("\rConclu%cdo!\n",161);
} //fim do while do treshold
sugilumina=sugilumina/i;
```

```
sugborra=sugborra/i;
printf("\n\nTOTAL DE FRAMES: %d\nLimiar de diferen%ca de pixel m%cdia:
%8.4f\n",i,135,130,sugilumina);
printf("Limiar de borramento m%cdio: %8.4f\n",130,sugborra);
fflush(stdin);
printf("\nDeseja utilizar os limiares de diferen%ca de pixel e\nborramentos m%cdios?
(S ou s para sim): ",135,130);
scanf("%c",&autotresh);
} //if para calculo das medias de iluminação e borramento
if(autotresh!='S'&&autotresh!='s')
{
fflush(stdin);
printf("\nDeseja utilizar o filtro de diferen%ca de pixels? (S ou s para sim): ",135);
scanf("%c",&uilumina);
if(uilumina=='S'||uilumina=='s') {
fflush(stdin);
printf("Digite o valor do limiar da diferen%ca de pixels: ",135);
scanf("%lf",&treshilumina);
fprintf(fptr,"Limiar da diferen%ca de pixels %4.2f\n",135,treshilumina);
} //if utiliza iluminacao
fflush(stdin);
printf("\nDeseja utilizar o filtro de borramentos? (S ou s para sim): ");
scanf("%c",&ublur);
if(ublur=='S'||ublur=='s') {
fflush(stdin);
printf("Digite o valor do limiar para o borramento: ");
scanf("%lf",&treshborra);
printf("Digite o numero de intervalos de frames a ser extra%cdo (1 para todos):
",161);
do { fflush(stdin);
scanf("%d",&qtframes);
if(qtframes<=0)
       printf("\nValor inv%clido. Digite novamente o n%cmero de intervalos a ser
extra%cdo: ",160,163,161);
}while(qtframes<=0); //nao deixa entrar valor negativo
fprintf(fptr,"Limiar de borramentos %4.2f\n",treshborra);
}//if utiliza borramento
}//if utilizar valores não padrões
else
{
    printf("\nUtiilizando os valores m%cdios determinados para sele%c%co de
keyframes\n",130,135,198);
   treshilumina=sugilumina;
   treshborra=sugborra;
   fprintf(fptr,"Limiar de diferen%ca de pixels. M%cdia
%4.2f\n",135,130,treshilumina);
   fprintf(fptr,"Limiar de borramento. M%cdia %4.2f\n",130,treshborra);
   ublur='s'; uilumina='s';
   qtframes=1;
}
if(uilumina!='s'&&uilumina!='S'&&ublur!='s'&&ublur!='S'&&autotresh!='s'&&autotresh!='S
')
{
       printf("Digite o n%cmero de intervalos de frames a ser extra%cdo (1 para todos):
",163,161);
    do { fflush(stdin);
       scanf("%d",&qtframes);
       if(qtframes<=0)
              printf("\nValor inv%clido. Digite novamente o n%cmero de intervalos a ser
extra%cdo: ",160,163,161);
    }while(qtframes<=0); //nao deixa entrar valor negativo
}
```

```
i=0; fr=0;

cvReleaseCapture(&tresh); //fecha o video do limiar
cvNamedWindow("Filme",CV_WINDOW_AUTOSIZE); //abre uma janela para mostrar o frame na
tela
frame = cvQueryFrame(capture); //recupera o frame e transforma em uma imagem na
variavel frame
frameprox = frame;
while(key != 'q'){ //executa ate que q seja pressionado ou termine o arquivo
frame=frameprox;
faval++;
//frameprox = cvQueryFrame(capture); //recupera o proximo frame e transforma em uma
imagem na variavel frameprox

if(!frameprox) //se não obter um dos frame termina
   break;
if(fr!=0) {  //faz o primeiro frame ser gravado antes de teste
//criar imagem grayscale do frame
width  = frame->width;
height = frame->height;
IplImage *framegray = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
//define a imagem cinze
cvCvtColor( frame, framegray, CV_RGB2GRAY ); //converte para cinza
IplImage *frameproxgray = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
//define segundo frame cinze

do {
if((ublur=='s'||ublur=='S')||(uilumina=='s'||uilumina=='S')) //se utilizar qualquer
filtro
    frameprox = cvQueryFrame(capture); //recupera o proximo frame e transforma em uma
imagem na variavel frameprox
IplImage *frameproxgray = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
//define segundo frame cinza

 if(!frameprox||(uilumina!='S'&&uilumina!='s')) //se não obter o frame ou nao for
utilizar o filtro termina
    break;

cvCvtColor( frameprox, frameproxgray, CV_RGB2GRAY ); //converte para cinza o segundo
frame
//calcular a diferença de grayscale de duas imagens
IplImage *framegraydif = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
cvAbsDiff (framegray, frameproxgray, framegraydif);
//mostrar diferença
//cvNamedWindow("Diferenca de pixels",CV_WINDOW_AUTOSIZE);
//cvShowImage("Diferenca de pixels",framegraydif); //mostra a imagem
//fimmostra
qtddif=countNonZero(framegraydif);
if(qtddif<=treshilumina) {
printf("\nframe %d descartado pela diferen%ca de pixels.",i,135);
i++;
dilumina++;
}
cvShowImage("Filme",frame); //mostra a imagem
}while(qtddif<=treshilumina); //fim da procura pela diferenca de pixels


 if(!frameprox||!frame) //se não obter o frame termina
    break;
}
else
```

```
{
   frameprox=frame;
} //fim do if se não for o primeiro frame

//if(i%qtframes==0) //verifica se é um quadro que será extraído
//{

   if(ublur=='S'||ublur=='s') //se for utilizar filtro de borramentos...
   {
          if(i%qtframes!=0) {
                    frameprox = skipNFrames(capture, qtframes-i%qtframes);
                    i=i+qtframes-i%qtframes;
              } //salta o intervalo selecionado
         // int controla=0;
          do {

                if(!frameprox) //se não obter o frame termina
                break;
        minretorno=espectro(frameprox); // ***********************************
analisar o espectro para detectar o borramento

             if(minretorno>treshborra)
           {
           printf("\nframe %d descartado por borramento",i);
                 i++;
                 dborra++;
                 //controla=1;
                 frameprox = cvQueryFrame(capture); //recupera o frame e transforma em
uma imagem na variavel frame
          if(!frameprox) //se não obter o frame termina
                break;
          cvShowImage("Filme",frameprox); //mostra a imagem obtida do frame na tela de
saída
           } //if do espectro
              else
              {
              //if(controla==0)
                    //i++; //else espectro
              }
          } while (minretorno>treshborra); //compara o limiar da imagem com o
especificado para realizar o salto se preciso
   } //if se utilizar borramentos

   //se nao utiliza nenhum filtro, gera todos
   if(ublur!='s'&&ublur!='S'&&uilumina!='s'&&uilumina!='S')
      cvShowImage("Filme",frameprox); //mostra a imagem obtida do frame na tela de
saída

       if(!frameprox) //se não obteve o frame termina
        break;
auxconv=10000+i;
_itoa(auxconv,j,10); //converte o inteiro em caractere para gravar na string do nome
do arquivo
j[0]='0';
//printf("%s",auxconv);
strcpy(nome,pasta);
strcat(nome,"\\");
strcat(nome,nomeimagens); //pega o nome das imagens
strcat(nome,j); //coloca numero sequencial no arquivo
strcat(nome,".jpg"); //adiciona a extensão jpg no arquivo de imagem
strcpy(j,"");
```

```
   cvSaveImage(nome, frameprox); //grava o frame no arquivo jpg
   printf("\nFrame %d escolhido! <-= ****************************",i); //mostra o
numero do frame da tela
   if((ublur=='s'||ublur=='S')&&(uilumina=='s'||uilumina=='S')) //se utiliza qualquer
filtro
      i++;
  // cvWriteFrame( writer, frame); //grava o frame no vídeo de saída avi
  fprintf(fptr,nome); //grava log
  fprintf(fptr," - Limiar diferenca pixels %d/%d: %d - Limiar borramento
%f\n",fr,i,qtddif,minretorno);
  // fprintf(fptr,": min = %g\n\n", minretorno); //gravalog
//} //if do salto de frames

key = cvWaitKey(40); //espera pela letra q para terminar

//se nao utiliza nenhum filtro, gera todos
if(ublur!='s'&&ublur!='S'&&uilumina!='s'&&uilumina!='S'){
       if(qtframes!=0) {
          frameprox = skipNFrames(capture, qtframes);
          i=i+qtframes;
       }
       else
             i++;
}
fr=i;
fr++; //aumenta o número sequencial do frame
}// while dos frames
//cvReleaseVideoWriter( &writer ); //fecha o vídeo criado
}

printf("\n\n=============================================================================
======");
fprintf(fptr,"\n\n=============================================================================
============");
printf("\nTOTAL DE FRAMES: %d",i);
fprintf(fptr,"\nTOTAL DE FRAMES: %d",i);
printf("\nTOTAL DE FRAMES SELECIONADOS: %d - %4.2f%%",faval,(float) ((float)
faval*100.0/(float) i));
fprintf(fptr,"\nTOTAL DE FRAMES SELECIONADOS: %d - %4.2f%%",faval,(float) ((float)
faval*100.0/(float) i));
printf("\nTOTAL DE FRAMES DESCARTADOS POR DIFEREN%CA DE PIXELS: %d -
%4.2f%%",128,dilumina,(float) ((float) dilumina*100.0/(float) i));
fprintf(fptr,"\nTOTAL DE FRAMES DESCARTADOS POR DIFEREN%CA DE PIXELS: %d -
%4.2f%%",128,dilumina,(float) ((float) dilumina*100.0/(float) i));
printf("\nTOTAL DE FRAMES DESCARTADOS POR BORRAMENTO: %d - %4.2f%%",dborra,(float)
((float) dborra*100.0/(float)i));
fprintf(fptr,"\nTOTAL DE FRAMES DESCARTADOS POR BORRAMENTO: %d -
%4.2f%%",dborra,(float) ((float) dborra*100.0/(float)i));


cvDestroyWindow("Filme"); //fecha a janela
cvReleaseCapture(&capture); //fecha o video aberto
fclose(fptr); //fecha arquivo de log
printf("\n\nTECLE ALGO PARA TERMINAR!");
getch();
return 0;

}


IplImage* skipNFrames(CvCapture* capture, int n) //salto de franes
**************************
```

```
{
    for(int i = 1; i < n; i++)
    {
        if(cvQueryFrame(capture) == NULL)
        {
            return NULL;
        }
    }

    return cvQueryFrame(capture);
}
//*****************************************************************

double espectro(IplImage *entrada)
//*****************************************************************
{
IplImage *image_Re = 0, *image_Im = 0, *Fourier = 0;
int nRow, nCol, i, j, cy, cx, width, height;
double elem, Re, Im, scale, shift, tmp13, tmp24;

CvPoint minLoc, maxLoc; //<---- usada pela função cvMinMaxLoc
double minVal = 0, maxVal = 0; //<-----------|

// Inicia contador de tempo
clock_t start = clock();


width  = entrada->width;
height = entrada->height;
IplImage *image = cvCreateImage( cvSize( width, height ), IPL_DEPTH_8U, 1 );
cvCvtColor( entrada, image, CV_RGB2GRAY );

//cvNamedWindow("Image", CV_WINDOW_AUTOSIZE);
//cvShowImage("Image", image);


//Imagem real
image_Re = cvCreateImage(cvGetSize(image), IPL_DEPTH_64F, 1);
//Imagem imaginaria
image_Im = cvCreateImage(cvGetSize(image), IPL_DEPTH_64F, 1);
//Imagem em escala cinza
Fourier = cvCreateImage(cvGetSize(image), IPL_DEPTH_64F, 2);


// Converte a parte real de u8 para 64f (double)
cvConvertScale(image, image_Re, 1, 0);

// Parte imaginária (zerada)
cvZero(image_Im);

// Junção das partes real e imaginária e armazenalas em uma imagem para Fourier
cvMerge(image_Re, image_Im, 0, 0, Fourier);

// Checar o que temos antes da Fourier (apenas os primeiros 16 valores)
//printf("\nValores antes da transformada de Fourier\n");
for( j = 0; j < 4; j++ )
{
for( i = 0; i < 4; i++ )
{
// Obtem os valores reais e imaginarios
Re = ((double*)(Fourier->imageData +
Fourier->widthStep*j))[i*2];
```

```
Im = ((double*)(Fourier->imageData +
Fourier->widthStep*j))[i*2+1];
//printf("(%.2f,%.2fi) ", Re, Im);
}
//printf("\n");
}

//Aplicação da transformada de Fourier DTC
cvDFT(Fourier, Fourier, CV_DXT_FORWARD);
// Checar o que temos depois de Fourier
//printf("\nApos a transformada de fourier\n");
for( j = 0; j < 4; j++ )
{
for( i = 0; i < 4; i++ )
{ //ver valores apos a transformada
Re = ((double*)(Fourier->imageData +
Fourier->widthStep*j))[i*2];
Im = ((double*)(Fourier->imageData +
Fourier->widthStep*j))[i*2+1];
//printf("(%.2f,%.2fi) ", Re, Im);
}
//printf("\n");
}
// Dividir Fourier em partes real e imaginária
cvSplit( Fourier, image_Re, image_Im, 0, 0 );
//Calcular a magnitude do espectro Mag = sqrt(Re^2 + Im^2)
cvPow( image_Re, image_Re, 2.0);
cvPow( image_Im, image_Im, 2.0);
cvAdd( image_Re, image_Im, image_Re);
cvPow( image_Re, image_Re, 0.5 );

// Calcular log(1 + Mag)
cvAddS( image_Re, cvScalar(1.0), image_Re ); // 1 + Mag
cvLog( image_Re, image_Re ); // log(1 + Mag)


// -Reorganizar os quadrantes da imagem de Fourier, de modo que a origem está no
// centro da imagem
nRow = image->height;
nCol = image->width;
cy = nRow/2; // centro da imagem
cx = nCol/2;
for( j = 0; j < cy; j++ )
{
for( i = 0; i < cx; i++ )
{
tmp13 = CV_IMAGE_ELEM( image_Re, double, j, i);
CV_IMAGE_ELEM( image_Re, double, j, i) = CV_IMAGE_ELEM(
image_Re, double, j+cy, i+cx);
CV_IMAGE_ELEM( image_Re, double, j+cy, i+cx) = tmp13;

tmp24 = CV_IMAGE_ELEM( image_Re, double, j, i+cx);
CV_IMAGE_ELEM( image_Re, double, j, i+cx) =
CV_IMAGE_ELEM( image_Re, double, j+cy, i);
CV_IMAGE_ELEM( image_Re, double, j+cy, i) = tmp24;
}
}


// Localize os valores minimos e maximos
cvMinMaxLoc( image_Re, &minVal, &maxVal );
//fprintf(fptr,"\nmin = %g, max = %g\n", minVal, maxVal);
```

```
//printf("\nmin = %g, max = %g\n", minVal, maxVal);

//printf("\n Tamanho da imagem : %i x %i pixels\n", image_Re->width,image_Re->height);

// Normalização da imagem (0 - 255) para ser observada como uma imagem u8
scale = 255/(maxVal - minVal);
shift = -minVal * scale;
cvConvertScale( image_Re, image, scale, shift);

//cvNamedWindow("Spectrum", CV_WINDOW_AUTOSIZE); //habilite para ver o espectro

//cvShowImage("Spectrum", image); //habilite para ver o espectro

cvReleaseImage(&image);
cvReleaseImage(&image_Re);
cvReleaseImage(&image_Im);
cvReleaseImage(&Fourier);

clock_t end = clock();
const double temps = (end - start); //calculo do tempo
//printf("\nComputation time: %lf seconds", temps/CLOCKS_PER_SEC);

//cvWaitKey(); //habilite para parar antes de continuar

//cvDestroyWindow("Image");
//cvDestroyWindow("Spectrum"); //habilite para ver o espectro
return minVal;
}

//termina aqui
```