UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Mateus de Souza

RASTREAMENTO VISUAL SOB MUDANÇAS EXTREMAS DE ILUMINAÇÃO UTILIZANDO A SOMA DA VARIÂNCIA CONDICIONAL

Florianópolis (SC)

2014

Mateus de Souza

RASTREAMENTO VISUAL SOB MUDANÇAS EXTREMAS DE ILUMINAÇÃO UTILIZANDO A SOMA DA VARIÂNCIA CONDICIONAL

Dissertação submetida ao Programa de Pós-graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciências da Computação.
Orientador: Prof. Dr. Renato Fileto
Coorientador: Dr. Rogério Richa

Florianópolis (SC)

2014

Ficha de identificação da obra elaborada pelo autor através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

A ficha de identificação é elaborada pelo próprio autor

Maiores informações em:
http://portalbu.ufsc.br/ficha

RESUMO

Rastreamento visual direto é resolvido atualmente utilizando principalmente técnicas de otimização baseadas em descida de gradiente. A velocidade de convergência destas técnicas permite utilizar modelos de transformações com vários graus de liberdade. Muitas abordagens utilizam a Soma do Quadrado dos Resíduos como medida de similaridade, mas esta técnica não oferece estabilidade perante mudanças de iluminação na cena. Estas mudanças causam instabilidades na convergência dos métodos se não forem compensadas. Uma das técnicas de compensação de iluminação utiliza um modelo paramétrico de iluminação que aumenta o número de parâmetros a serem calculados. As aplicações que utilizam rastreamento direto precisam de respostas em tempo real e podem tornar-se impraticáveis com a adição do modelo de iluminação. Nesta dissertação é proposto um método de rastreamento visual direto robusto capaz de rastrear sob condições de iluminação extremas. Utilizando a Soma da Variância Condicional como base, a abordagem proposta utiliza sub-imagens para lidar com mudanças de iluminações extremas. O método proposto reduz o esforço computacional quando comparado com técnicas similares da literatura. Resultados experimentais atestam a redução de em média 57,5% em tempo de processamento para sequências coloridas.

Palavras-chave: Rastreamento Visual Direto, Soma da Variância Condicional, Otimização Baseada em Gradiente

ABSTRACT

Direct visual tracking is currently solved mainly with the use of gradient descent optimization. The speed of convergence of these techniques allows the use of transformation models with many degrees of freedom. The most popular similarity measure for direct tracking is the Sum of the Squared Differences, even though this approach is not robust to illumination changes in the scene. These changes, when left uncompensated, can lead to instabilities in the convergence of the algorithms. One technique used to compensate illumination changes uses an illumination model, which increases the number of parameters to be computed. Since most applications that use direct visual tracking need the results to be delivered in real time, the addition of the illumination model can hinder their performance. A novel direct visual tracking approach is presented in this work, being able to cope with extreme illumination conditions. Using the Sum of Conditional Variance as a base, the proposed method uses sub-images to compensate for extreme illumination configurations. The proposed method reduces the computational burden when compared to similar approaches in the literature. Experimental results show that the method is 57,5% faster on average when dealing with color sequences.

Keywords: Direct visual tracking, Sum of Conditional Variance, Gradient-based Optimization

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

CUDA Compute Unified Device Architecture. Plataforma de desenvolvimento para aplicações executadas em placas gráficas.

ESM Minimização Eficiente de Segunda Ordem (do inglês Efficient Second-order Minimization). Um aprimoramento do método de Lucas-Kanade.

GPGPU Computação de Propósito Geral em GPUs (do inglês General-purpose Computing on Graphics Processing Units). Utilização de GPUs para cálculo de operações de natureza não-gráfica.

GPU Unidade de Processamento Gráfico (do inglês Graphics Processing Unit). Arquitetura de processamento que permite processamento paralelo massivo.

LSCV Soma da Variância Condicional Local (do inglês Local Sum of the Conditional Variance). Medida de similaridade baseada no SCV que torna a medida robusta a mudanças locais de iluminação.

MEIA Média do Erro de Intensidade Absoluto. Medida de erro de rastreamento baseado no erro da intensidade dos pixels da imagem atual com a imagem de referência.

MI Informação Mútua (do inglês Mutual Information). Medida de similaridade baseada na entropia das imagens.

NCC Correlação Cruzada Normalizada (do inglês Normalized Cross Correlation). Medida de similaridade proveniente da área médica.

SCV Soma da Variância Condicional (do inglês Sum of Conditional Variance). Uma medida de similaridade robusta a mudanças de iluminação globais.

SIFT Scale-invariant feature transform. Método de detecção de aspectos em imagens.

SSD Soma do Quadrado dos Resíduos (do inglês Sum of Squared Differences). Medida de similaridade simples, utilizada originalmente no método Lucas-Kanade.

SURF Speeded Up Robust Features. Método de detecção de aspectos. Baseado no método SIFT.

TPS Thin Plate Splines. Técnica de rastreamento que utiliza thin plate splines para compensação de iluminação.

TPS-3 Técnica de rastreamento baseada no método TPS que utiliza três thin plate splines para realizar compensação de ilumina-ção em cada canal de cor.

1 INTRODUÇÃO

Nesta dissertação, rastreamento é definido como o problema de determinar a posição de um objeto de interesse em quadros consecutivos de um vídeo a partir de uma configuração inicial. A Figura 1 demonstra um exemplo de rastreamento. Selecionando o sol como objeto de interesse, o algoritmo deve seguí-lo durante a cena, informando ao usuário a posição (e orientação) em todos os quadros que compõem o vídeo.

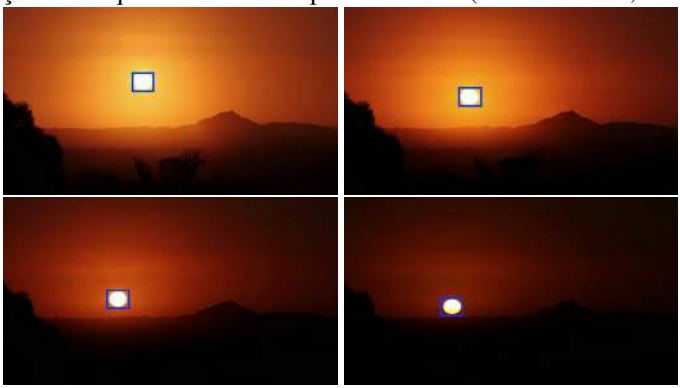
Rastrear objetos em vídeos é um dos grandes problemas em análise de imagens. Este problema possui uma grande complexidade computacional e é um dos elementos fundamentais para muitas aplicações de visão computacional (BAKER et al., 2011). Uma das abordagens comuns é marcar o objeto a ser rastreado com um padrão de fácil reconhecimento pela câmera, de modo que o cálculo da posição do objeto possa ser facilitado (MUNOZ-SALINAS, 2012). Esta modificação é muito útil quando o ambiente de rastreamento pode ser alterado previamente (por exemplo, um jogo que utiliza realidade aumentada). Como muitas aplicações de rastreamento não podem utilizar marcadores (i.e. aplicações de segurança, pós-processamento em vídeos, etc.), outras técnicas evoluíram para realizar rastreamento sem a necessidade de auxílio na cena original. Destas técnicas, duas são as mais utilizadas: rastreamento baseado em aspectos e rastreamento baseados em textura.

Rastreamento baseado em aspectos (do inglês, *feature-based tracking*) busca extrair pontos de interesse na imagem analisada (YILMAZ; JAVED; SHAH, 2006). Rastreamento baseado em textura (do inglês, *texture-based tracking*) utiliza diferenças na imagem atual em relação à imagem anterior para inferir movimento. Este último tipo de rastreamento é o foco deste trabalho.

Como os métodos baseados em textura utilizam a diferença entre duas imagens, eles podem ser considerados problemas de otimização com três principais componentes:

1. Medida de similaridade: esta é a medida utilizada para quantificar o quão parecidas são duas imagens (por exemplo, soma dos quadrados dos resíduos, correlação normalizada cruzada (G.

Figura 1 – Rastreamento de um sol se pondo. O algoritmo atualiza a posição do objeto a cada quadro (retângulo azul), baseado nas informações da sequência. Vídeo disponível *online* (MYNHARDT, 2001).



MEILAND M., 2012) e informação mútua (HERMOSILLO; CHEFD'HOTEL; FAUGERAS, 2002)).

2. Modelo de transformação paramétrica: este modelo descreve como o objeto sendo rastreado se move pela cena. Os mais simples descrevem objetos que se movimentam apenas em duas dimensões com translações, enquanto os mais complexos conseguem modelar até distorções na câmera.

3. Estratégia de otimização: a estratégia que é utilizada para minimizar a diferença entre as imagens (por exemplo, descida de gradiente (MéGRET; AUTHESSERRE; BERTHOUMIEU, 2008)).

Portanto, o algoritmo tenta atualizar os parâmetros do modelo de transformação de modo que a diferença entre a imagem atual e a imagem de referência seja mínima. Este processo é iterativo e espera que o objeto possua pouco movimento entre os quadros da cena sendo analisada.

No mundo real, no entanto, muitos outros desafios são adicionados ao rastreamento. Um dos principais problemas que os algoritmos encontram em ambientes não controlados é a mudança de iluminação. A Figura 2 ilustra como as mudanças de iluminação podem afetar o rastreamento. A presença de uma janela na sala onde o ras-

Figura 2 – Efeitos de mudanças de iluminação em um cenário real. A presença de uma janela na cena faz com que o objeto de interesse sofra mudanças bruscas quando movimentado.

(a) (b) (c)

treamento está acontecendo faz com que o objeto de interesse sofra mudanças extremas, dependendo do ângulo em que se encontra em relação à fonte de luz. Além de haver reflexões da luz no objeto, a própria câmera pode alterar a imagem quando tenta se ajustar às mudanças de iluminação.

As técnicas de rastreamento baseadas em textura utilizam as intensidades dos pixels das imagens para realizar inferências sobre o movimento do objeto. Quando há uma mudança de iluminação, estes valores podem se alterar sem que o objeto se mova. Isso faz com que muitos algoritmos falhem no rastreamento.

Há duas categorias de mudanças de iluminação. As mais fáceis de serem contornadas são as mudanças globais. Nelas, todas as intensidades dos pixels são afetados de maneira uniforme. Mudanças locais (por exemplo, reflexões especulares) requerem uma abordagem mais complexa, já que apenas alguns pixels são afetados pelos efeitos de iluminação.

O problema de rastreamento visual é computacionalmente complexo, já que imagens geralmente possuem muitos dados e o número de imagens a serem analisadas é grande. Além disso, algumas aplicações que utilizam rastreamento visual precisam de uma resposta rápida dos algoritmos para manter a interatividade com o usuário. Para suprir esta necessidade, uma estratégia comum é utilizar placas de processamento gráfico (GPU do inglês *Graphics Processing Unit*). Inicialmente utilizadas para realizar cálculos específicos de visualização, estas placas evoluíram para serem utilizadas para cálculos de propósito

genérico (GPGPU do inglês *General-purpose Computing on Graphics Processing Units).* Este ambiente de execução será também explorado nesta dissertação para os algoritmos de rastreamento.

1.1 MOTIVAÇÃO

Um dos motivadores para a realização deste trabalho é a crescente utilização de realidade aumentada em aplicações de tempo real (ZHOU; DUH; BILLINGHURST, 2008). Este tipo de visualização é diretamente influenciada pelo rastreamento de objetos, já que estimar a pose da câmera é um dos seus principais passos. Muitos das técnicas de realidade aumentada hoje em dia utilizam marcadores para realizar o rastreamento, principalmente pelo fato de que seu uso é controlado (KATO; BILLINGHURST, 1999; MUNOZ-SALINAS, 2012). No entanto, a possibilidade de não utilizar marcadores é muito atraente para algumas aplicações. Além disso, estabilidade e velocidade são dois fatores primordiais para aplicações de realidade aumentada. Estes aplicativos são geralmente utilizados em tempo real, com interação direta com usuários (REITMAYR; DRUMMOND, 2006; VACCHETTI; LEPETIT; FUA, 2004).

Outra área onde o rastreamento pode ser utilizado é no auxílio à médicos para a realização de cirurgias com robôs (RICHA; BO; POIGNET, 2011). É possível, por exemplo, estimar movimento de órgãos durante a cirurgia, de modo a compensar o movimento dos braços mecânicos para que as incisões possam ser mais precisas. Aqui também é necessária uma resposta em tempo real do algoritmo de rastreamento.

Estes algoritmos também são utilizados em aplicações de *mosaicing* (RICHA et al., 2014). Estas aplicações utilizam um conjunto de imagens de um mesmo objeto para, a partir de um processo de *blending*, formar uma imagem de alta definição que mostre todas as características do objeto. O rastreamento direto é utilizado aqui para calcular a disposição relativa das imagens (LOEWKE et al., 2011).

1.2 OBJETIVOS

- Objetivo geral: desenvolver técnicas de rastreamento robustas à mudanças locais de iluminação.
- Objetivos específicos:
  - Implementar e comparar métodos de rastreamento direto do estado da arte.
  - Explorar a métrica de similaridade Soma da Variância Condicional (SCV), tornando-a robusta a mudanças de iluminação locais.
  - Implementar os algoritmos de rastreamento utilizando *Graphic Processing Units* (GPUs).
  - Organizar um *framework* de fácil utilização para rastreamento utilizando métodos baseados em textura.

A métrica de similaridade Soma da Variância Condicional promete ser uma métrica mais robusta à variação de iluminação global, porém ela não é capaz de lidar com mudanças locais. Baseando-se na técnica de registro de imagens apresentada em (HERMOSILLO; CHEFD'HOTEL; FAUGERAS, 2002), será proposta uma técnica de divisão da imagem em sub-regiões que tornará esta medida menos sensível a mudanças de iluminação locais.

A maioria dos algoritmos utilizados para o rastreamento é paralelizável. Por isso, este trabalho prevê a implementação destes métodos utilizando GPUs. As placas gráficas possuem um modelo de execução que permite que algoritmos paralelos sejam executados de maneira muito mais rápida.

Um dos objetivos secundários deste trabalho é prover um *framework* para rastreamento em imagens com as técnicas desenvolvidas. Como a maioria das técnicas utilizadas para rastreamento de imagens são paralelizáveis por natureza, este *framework* fará uso extensivo da GPU de modo a acelerar a resposta ao usuário. Com a criação deste *framework*, será possível verificar a viabilidade de outros métodos com mais facilidade.

1.3 ORGANIZAÇÃO

Esta dissertação está organizada da seguinte maneira. O próximo capítulo descreve uma revisão sistemática da literatura, levantando os principais trabalhos relevantes ao rastreamento visual direto e suas métricas de similaridade.

O Capítulo 3 descreve rastreamento direto de forma abrangente. Aqui serão apresentados as principais dificuldades no rastreamento, assim como aprofundadas as técnicas utilizadas para sua solução. O Capítulo 4 descreve rastreamento utilizando a métrica escolhida para esta dissertação. Aqui também são descritos as mudanças feitas à técnica para torná-la robusta à mudanças de iluminação locais.

O desenvolvimento e implementação das técnicas são apresentados no Capítulo 5. Estas implementações foram então usadas para realizar uma série de experimentos que são apresentados no Capítulo 6.

Finalmente, a dissertação é concluída no Capítulo 7, juntamente com a apresentação de alguns possíveis trabalhos futuros.

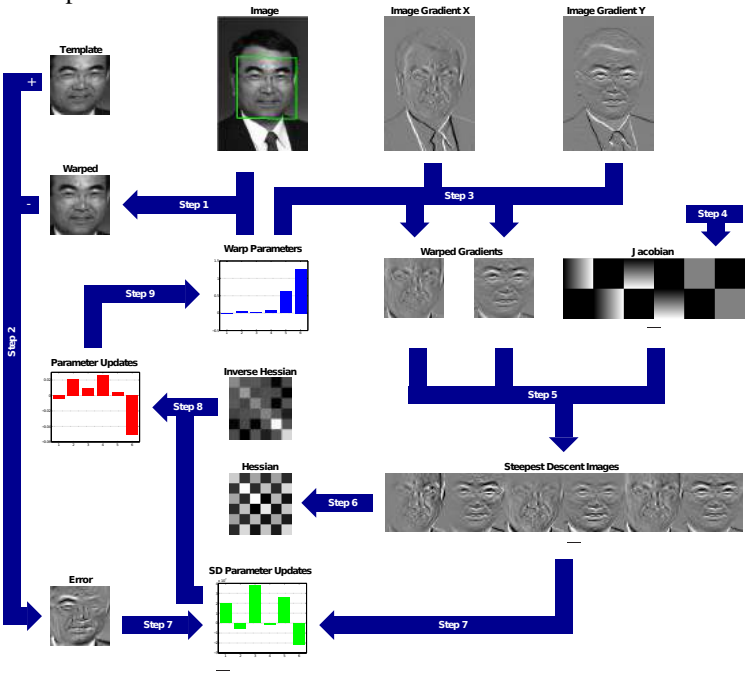
2 REVISÃO BIBLIOGRÁFICA

Como dito no capítulo anterior, as técnicas de rastreamento são divididas principalmente em duas categorias: técnicas baseadas em aspectos e técnicas baseadas em textura. As técnicas que se baseiam em aspectos buscam pontos de interesse na imagem e então inferem movimento baseando-se nas diferentes posições destes pontos em imagens subsequentes. É comum a utilização de técnicas como o SURF (BAY; TUYTELAARS; GOOL, 2006) ou o SIFT (LOWE, 1999) para a extração dos pontos.

Esta dissertação baseia-se principalmente em algoritmos de rastreamento baseados em textura. O primeiro artigo a descrever um método deste tipo foi o de Lucas-Kanade (LUCAS; KANADE, 1981), que é a base para todos os métodos descritos neste capítulo bem como as expansões abordadas nesta dissertação. O artigo descreve um *framework* para rastreamento direto, formulando a tarefa como um problema de registro de imagem. Neste trabalho a Soma do Quadrado dos Resíduos (SSD do inglês *Sum of Squared Differences)* é utilizada como medida de similaridade. Este método toma como base a imagem de referência e tenta, utilizando o método de otimização, encontrar os parâmetros para o modelo de transformação que faça com que a dissimilaridade entre a imagem atual e a de referência seja minimizada. Na Figura 3 encontram-se todos os passos utilizados no método, que serão explicados em maior profundidade no Capítulo 3.

Após a publicação deste primeiro artigo, outros pesquisadores sugeriram melhoras ao *framework* de rastreamento. Uma classificação dos métodos de rastreamento é proposta por Baker e Matthews (2004) utilizando duas características para a classificação: a imagem utilizada como base para o rastreamento (*forward* ou *inverse*) e o método utilizado para atualizar os parâmetros de transformação (*additive* ou *compositional)*. Classificam, portanto, o método de Lucas-Kanade original como sendo *forward additive*, já que ele tenta alinhar a imagem de referência à imagem atual e calcula um Δp que será somado aos parâmetros de transformação. Além desta classificação, neste trabalho também são propostos os métodos de rastreamento inversos, que utilizam a imagem referência como imagem base, fazendo com que a imagem

Figura 3 – Algoritmo de Lucas-Kanade. Este algoritmo será detalhado no Capítulo 3.



atual se alinhe a ela. Este *framework* de rastreamento (classificado na obra como *inverse*) possui a vantagem de poder pré-calcular algumas matrizes que podem ser reutilizadas durante o processo, tornando-o computacionalmente mais eficiente.

Uma forma otimizada do método de Lucas-Kanade também é apresentada por Hager e Belhumeur (1998). Neste trabalho, é proposto que certas matrizes utilizadas pelo algoritmo podem ser pré-calculadas, de modo a diminuir o esforço computacional. Este trabalho também apresenta compensação a mudanças de iluminação utilizando imagens-base com configurações de iluminação distintas, além de apresentar uma abordagem estatística para o tratamento de oclusões.

Como descrito no Capítulo 1, métodos de rastreamento direto podem ser classificados por três componentes: método de otimização, medida de similaridade e modelo de transformação paramétrico. Geralmente os avanços das técnicas de rastreamento abordam principalmente um destes três aspectos e portanto esta revisão avaliará os principais trabalhos em cada área.

2.1 MODELO DE TRANSFORMAÇÃO PARAMÉTRICO

Inicialmente, os modelos de transformação paramétricos utilizados eram simples. Utilizavam, por exemplo, apenas dois graus de liberdade nas transformações (translações). Com o avanço das técnicas de otimização, bem como dos *hardwares* disponíveis para os cálculos, modelos mais complexos foram propostos.

Uma das classificações entre os modelos de transformação divide eles entre transformações rígidas e não-rígidas (elásticas). Os modelos rígidos são os mais usados por serem mais simples que os elásticos. Porém, eles também só conseguem rastrear objetos que não se deformam durante o processo. Ou seja, os objetos rastreados só podem realizar transformações rígidas (translações, rotações, transformações afim, etc.).

Um método que utiliza um modelo de transformação capaz de rastrear objetos deformáveis é apresentado por Silveira e Malis (2010). Como as transformações rígidas são um caso especial de transformação elástica, esse método também é capaz de rastrear objetos rígidos. Além disso, também é descrito um método de compensação de iluminação para aumentar a robustez do rastreamento.

2.2 MÉTODO DE OTIMIZAÇÃO

Naturalmente, um dos modelos de otimização que possui melhor convergência é o de busca exaustiva sobre os parâmetros de transformação (e.g. força-bruta) (ZITOVá; FLUSSER, 2003). No entanto, dado o custo proibitivo desta técnica para modelos de transformação mais complexos, outras técnicas são sugeridas. Algumas técnicas utilizam *particle filtering* (ISARD; BLAKE, 1998) para a otimização. O

método de otimização mais utilizado atualmente é a descida de gradiente (BAKER; MATTHEWS, 2004). Este método é rápido e robusto o suficiente para ser usado em aplicações que requerem rápida resposta do algoritmo (MéGRET; AUTHESSERRE; BERTHOUMIEU, 2008).

Esta dissertação baseia-se na técnica de descida de gradiente chamada Minimização Eficiente de Segunda Ordem (ESM do inglês *Efficient Second-order Minimization*) (BENHIMANE; MALIS, 2004). Este método busca reduzir o custo do cálculo de otimização utilizando uma aproximação para o cálculo de algumas matrizes. Mais detalhes sobre este algoritmo podem ser encontrados no Capítulo 3.

2.3 MEDIDA DE SIMILARIDADE

O método original de Lucas-Kanade utiliza SSD como medida de similaridade, como dito anteriormente. Esta é a medida de similaridade mais simples, pois considera apenas a diferença entre cada par de pixels entre a imagem atual e a referência. Esta medida de similaridade é particularmente sensível a mudanças de iluminação.

A medida de similaridade chamada Informação Mútua (MI do inglês *Mutual Information)* é apresentada por Dame e Marchand (2010). Esta medida permite rastrear objetos com imagens de referência capturadas em modo diferente da imagem atual. Ela baseia-se na entropia de cada imagem, bem como na entropia que as duas imagens compartilham. Como as duas imagens podem ter modalidades diferentes, esta medida também é robusta à mudanças de iluminação e oclusões.

Outra medida robusta a mudanças de iluminação globais é a Correlação Cruzada Normalizada (NCC do inglês *Normalized Cross Correlation)* utilizada por Scandaroli, Meilland e Richa (2012). Esta medida é mais próxima do SSD, mas a fórmula de correlação é robusta a mudanças de brilho e contraste nas imagens.

A medida de similaridade Soma da Variância Condicional (SCV do inglês *Sum of the Conditional Variance*) é utilizada por Richa et al. (2011). Proveniente da área de imagens médicas, esta medida de similaridade se destaca por ser pouco custosa computacionalmente e por ser robusta a mudanças globais de iluminação. Quando comparada com outras medidas de similaridade como a MI ou a NCC, ela necessita de menos iterações para convergir. Além disso, ela possui um

maior raio de convergência, que faz com que a técnica de rastreamento que usa a medida consiga convergir para um resultado ótimo em mais casos. Por estes motivos, esta foi a medida escolhida para ser utilizada nesta dissertação.

A Figura 4 mostra um resumo de como cada uma das medidas de similaridade acima se comportam para um conjunto de imagens diferentes. É possível perceber que medidas de similaridade como o SSD não são muito robustas a mudanças na imagem sendo rastreada. Esta medida aponta pouca semelhança entre as imagens para todos os tipos de mudanças mostrados. A medida SCV, no entanto, consegue lidar com mais tipos de mudanças e é principalmente bem adaptada a mudanças lineares e não-lineares. Porém, como dito anteriormente, ela não é robusta a mudanças locais.

2.4 COMPENSAÇÃO DE ILUMINAÇÃO

Os métodos que utilizam apenas a Soma do Quadrado dos Resíduos como medida de similaridade levam em consideração que o valor dos pixels se manterão constantes, apenas transformações espaciais serão feitas sobre a imagem referência. Essa é uma limitação que obviamente não pode ser mantida em condições reais. Mudanças de iluminação e oclusões fazem com que esses métodos se percam no rastreamento. Uma abordagem comum é adicionar parâmetros de iluminação ao modelo de transformação, fazendo com que a imagem atual seja mais próxima da imagem de referência (BARTOLI, 2008). Alguns modelos tentam estimar o brilho e o contraste que, quando aplicados à imagem atual, minimizam as dissimilaridades com a imagem de referência. Apesar destes métodos funcionarem bem com mudanças de iluminação globais, eles não podem ser aplicados a mudanças locais. Além disso, eles reduzem o raio de convergência do método, já que agora existem mais parâmetros a serem estimados.

Uma técnica de compensação de iluminação tanto para mudanças globais quanto locais é apresentada por Silveira e Malis (2010). Neste trabalho, as mudanças de iluminação são modeladas por uma superfície deformável. Alguns pontos de controle para a superfície são definidos e os pesos de cada ponto são calculados utilizando o mesmo método que para os parâmetros do modelo de transformação. Assim,

Figura 4 – Comportamento das diferentes medidas de similaridade quando confrontadas com diferentes tipos de mudanças nas duas imagens. É possível perceber que o SSD não suporta nenhum dos tipos de mudanças descritos acima, sempre apontando uma dissimilaridade grande entre as imagens.

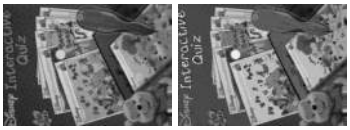
(a) Mudança linear

SSD=1.29e+09
NCC=1
SCV=0
MI=2.02

(b) Mudança não-linear

SSD=6.31e+09
NCC=0.82
SCV=0
MI=1.93

(c) Mudança multi-modal



SSD=4.69e+09
NCC=0.93
SCV=1.24e+06
MI=1.38

(d) Mudança local

SSD=1.30e+10
NCC=0.67
SCV=5.40e+06
MI=0.44

este método apresenta o mesmo ponto fraco dos métodos anteriores.

Naturalmente, o uso de medidas de similaridade que já sejam robustas a mudanças de iluminação globais fazem com que a compensação seja mais direta. É o caso, por exemplo, do SCV (RICHA et al., 2011). O uso do SCV tem a vantagem de ser computacionalmente menos pesado do que os métodos que estimam parâmetros de iluminação.

2.5 IMPLEMENTAÇÃO EM PLACAS GRÁFICAS

Os métodos citados acima são todos de natureza paralela. Seja na compensação de iluminação ou no cálculo das matrizes para a otimização dos parâmetros, a maioria dos algoritmos apresentados possuem passos que podem ser executados em paralelo. Um modelo de execução que é especialmente rápido para implementações de algoritmos paralelos são as placas gráficas. Para aplicações de computação intensa é esperado um ganho de 10 a 20 vezes em tempo de processamento quando comparado com implementações em CPUs (RYOO et al., 2008). Atualmente existem plataformas para facilitar a programação de aplicações para placas gráficas (NICKOLLS et al., 2008; GROUP et al., 2008), mas rastreamento de movimento já vem sendo implementada em GPUs há alguns anos (STRZODKA; GARBE, 2004).

Algumas das técnicas descritas acima já foram implementadas utilizando esta tecnologia. Marzat, Dumortier e Ducrot (2008) apresentam uma implementação de *optical flow* em placas gráficas. Implementações do método Lucas-Kanade em GPUs são apresentados por Sinha et al. (2006) e Duvenhage, Delport e Villiers (2010), sendo que o primeiro trabalho também apresenta uma implementação do algoritmo SIFT. Brown e Capson (2012) apresentam *particle filtering* em placas gráficas. Modelos de transformação não-rígidos também já possuem implementações em GPUs (SCHOENEMANN; CREMERS, 2010; RICHA; POIGNET; LIU, 2010).

3 RASTREAMENTO

Neste capítulo será apresentado o método de Lucas-Kanade, bem como a Otimização Eficiente de Segunda-ordem (ESM), introduzidos no capítulo passado. Primeiramente, será apresentada a notação utilizada tanto neste capítulo quanto nos capítulos seguintes para a descrição dos algoritmos.

3.1 NOTAÇÃO

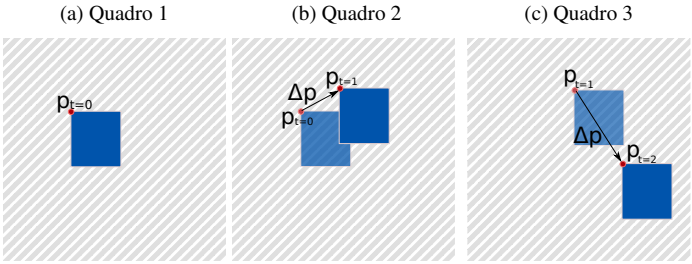
Considerando que a matriz $I \in \mathbb{R}^{r \times c}$ representa uma imagem com r linhas e c colunas, cada elemento de I representa uma intensidade de pixel. A imagem I representa a imagem atual sendo analisada. É denotado como T a imagem de referência do objeto sendo rastreado. Utilizaremos $q = r \times c$ para denotar o número total de pixels na imagem de referência. Seja um vetor de pontos $\mathbf{x} = (x_1, x_2, \ldots, x_q)$ onde $x_i \in \mathbb{R}^2$, então a função $I(\mathbf{x}) : \mathbb{R}^2 \rightarrow \mathbb{R}$ mapeia cada ponto x para uma intensidade de pixel correspondente em I, realizando uma interpolação bilinear se os valores u e v não forem inteiros. Uma função de transformação (do inglês, *warp*), é definida como a função $w(\mathbf{x}, \mathbf{p}) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ que transforma cada ponto x segundo o modelo de transformação escolhido com base no vetor de parâmetros $\mathbf{p}$. Utiliza-se n para denotar o tamanho do vetor $\mathbf{p}$.

3.2 LUCAS-KANADE

O método de Lucas-Kanade (LUCAS; KANADE, 1981) é a base para esta dissertação. Ele assume que a diferença entre quadros consecutivos é pequena para que o rastreamento possa obter sucesso.

A Figura 5 resume o funcionamento do algoritmo de Lucas-Kanade. Inicialmente, temos um objeto de interesse representado pelo retângulo azul, em uma configuração $\mathbf{p}$ na imagem atual. Quando é capturado o próximo quadro, percebe-se que o objeto se movimenta da sua posição original. O objetivo do algoritmo é então encontrar

Figura 5 – Três quadros exemplo do funcionamento do algoritmo de Lucas-Kanade. O objeto de interesse, representado pelo retângulo azul, movimenta-se pela cena. O algoritmo consegue rastreá-lo atualizando o vetor de parâmetros **p**, com as atualizações $\Delta\mathbf{p}$ a cada quadro.



um conjunto de atualizações para os parâmetros **p** (na imagem representados por $\Delta\mathbf{p}$). Isto é feito para cada quadro, de modo que sempre é possível saber a posição atual do objeto de interesse, sendo apenas necessário calcular as modificações que ocorreram nos parâmetros de um quadro para o próximo.

É importante notar que o algoritmo necessita de um valor inicial válido para o vetor de parâmetros **p**. Além disso, a movimentação do objeto de interesse não pode ser muito brusca. Como o algoritmo trabalha em cima de atualizações sobre a posição atual do objeto de interesse, ele espera que o objeto mova-se pouco entre um quadro e outro (o que pode não ser o caso, por exemplo, no quadro 3 da Figura 5). As câmeras modernas geralmente possuem uma taxa de captura suficiente para que o algoritmo consiga acompanhar a maioria dos objetos em velocidades moderadas. No entanto, ainda é possível que o rastreamento torne-se impossível se o objeto de interesse mantiver uma velocidade muito grande entre quadros.

Uma das bases do rastreamento é o modelo de transformação paramétrico $w(\mathbf{x}, \mathbf{p})$. Podemos utilizar, por exemplo quando calculando *optical flow*, uma função que modela apenas translações. Neste caso, por exemplo, a função de transformação pode ser definida da seguinte maneira:

$$w(\mathbf{x},\mathbf{p}) = \begin{pmatrix} u+p_1 \\ v+p_2 \end{pmatrix} \tag{3.1}$$

Da mesma forma, pode ser definida uma função de transformação mais complexa. Como os objetos de interesse estão em um ambiente tri-dimensional que também pode sofrer com distorções da câmera, nesta dissertação usaremos uma função de transformação projetiva (ou homografia). Assim, nossa função é descrita como segue:

$$w(\mathbf{x},\mathbf{p}) = \begin{pmatrix} \frac{u\cdot p_1+v\cdot p_2+p_3}{u\cdot p_7+v\cdot p_8+p_9} & \frac{u\cdot p_4+v\cdot p_5+p_6}{u\cdot p_7+v\cdot p_8+p_9} \end{pmatrix}^T$$

Portanto, nove parâmetros devem ser estimados para cada passo de otimização. O método de Lucas-Kanade baseia-se em encontrar valores para o vetor de parâmetros do modelo de transformação $\mathbf{p}$ que minimizem a dissimilaridade entre a imagem de referência $T(x)$ e a imagem atual transformada pelo modelo de transformação $I(w(x,\mathbf{p}))$. Em outras palavras, o método utiliza a Soma do Quadrado dos Resíduos (SSD) como medida de similaridade. Portanto, o método tenta achar o mínimo para a seguinte medida:

$$SSD = \sum_{i=0}^{q} \left(I\left(w\left(\mathbf{x}_i,\mathbf{p}\right)\right) - T\left(\mathbf{x}_i\right)\right)^2$$

Para minimizar esta equação, o método assume que um $\mathbf{p}$ ótimo já é conhecido para o quadro anterior e então acha, iterativamente, incrementos para os parâmetros. Em outras palavras, a seguinte expressão que é de fato minimizada:

$$\sum_{i=0}^{q} \left(I\left(w\left(\mathbf{x}_i,\mathbf{p}+\triangle\mathbf{p}\right)\right) - T\left(\mathbf{x}_i\right)\right)^2 \tag{3.2}$$

com relação a $\triangle\mathbf{p}$. Então, os parâmetros são atualizados para o próximo passo da iteração:

$$\mathbf{p} \leftarrow \mathbf{p}+\triangle\mathbf{p} \tag{3.3}$$

A iteração é mantida até que os parâmetros $\mathbf{p}$ convirjam. Ou seja, os parâmetros são atualizados até que a norma do vetor de atua-

lizações seja menor que um valor pré-definido ε (i.e.$\| \triangle \mathbf{p} \| < \varepsilon$).

O algoritmo de Lucas-Kanade é derivado, então, da seguinte maneira. A expressão não-linear (3.2) é linearizada utilizando a expansão de primeira ordem da série de Taylor em $I(w(\mathbf{x}, \mathbf{p} + \triangle \mathbf{p}))$ resultando em:

$$\sum_{i=0}^{q} \left[I(w(\mathbf{x}_i, \mathbf{p})) + \nabla I \frac{\partial w}{\partial \mathbf{p}} \triangle \mathbf{p} - T(\mathbf{x}_i) \right]^2 = 0 \tag{3.4}$$

onde $\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$ é a imagem gradiente de I. O termo $\nabla I \frac{\partial w}{\partial \mathbf{p}}$ é a Jacobiana da atualização. Se $w(x, \mathbf{p}) = (w_x(\mathbf{x}, \mathbf{p}), w_y(\mathbf{x}, \mathbf{p}))^T$, então a segunda parte da Jacobiana pode ser escrita como:

$$\frac{\partial w}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial w_x}{\partial p_1} & \frac{\partial w_x}{\partial p_2} & \cdots & \frac{\partial w_x}{\partial p_n} \\ \frac{\partial w_y}{\partial p_1} & \frac{\partial w_y}{\partial p_2} & \cdots & \frac{\partial w_y}{\partial p_n} \end{pmatrix}$$

onde n é o número de parâmetros que compõem $\mathbf{p}$. Por exemplo, usando o exemplo simples da função de transformação que lida apenas com translações, chega-se ao seguinte resultado:

$$\frac{\partial w}{\partial \mathbf{p}} = \left(\frac{\partial w_x}{\partial p_1}, \frac{\partial w_y}{\partial p_2} \right)^T = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

A derivada parcial da expressão (3.4) com relação a $\triangle \mathbf{p}$ é dada por:

$$\sum_{i=0}^{q} \left[\nabla I \frac{\partial w}{\partial \mathbf{p}} \right]^T \left[I(w(\mathbf{x}_i, \mathbf{p})) + \nabla I \frac{\partial w}{\partial \mathbf{p}} \triangle \mathbf{p} - T(\mathbf{x}_i) \right]$$

Igualando esta expressão por zero e resolvendo para $\triangle \mathbf{p}$ resulta na equação:

$$\triangle \mathbf{p} = H^{-1} \sum_{i=0}^{q} \left[\nabla I \frac{\partial w}{\partial \mathbf{p}} \right]^T [I(w(\mathbf{x}_i, \mathbf{p})) - T(\mathbf{x}_i)] \tag{3.5}$$

onde H é uma aproximação da matriz Hessiana $n \times n$, que é calculada da seguinte maneira:

$$H = \sum_{i=0}^{q} \left[\nabla I \frac{\partial w}{\partial \mathbf{p}} \right]^T \left[\nabla I \frac{\partial w}{\partial \mathbf{p}} \right] \tag{3.6}$$

O algoritmo de Lucas-Kanade, então, resume-se em aplicar as Equações (3.5) e (3.3) iterativamente, até que a norma de $\triangle \mathbf{p}$ alcance o limiar ε. Um resumo do algoritmo encontra-se no Algoritmo 1 bem como na Imagem 3. Como o gradiente ∇I deve ser calculado utilizando $w(x, \mathbf{p})$, tanto ele como a matriz Jacobiana $\nabla I \frac{\partial w}{\partial \mathbf{p}}$ geralmente dependem de $\mathbf{p}$. Para algumas funções de transformação mais simples como a de translação da Equação (3.1) a segunda parte da matriz Jacobiana é constante. No entanto, em geral, todos os passos do algoritmo precisam ser executados a cada passo de iteração, já que o vetor de parâmetros $\mathbf{p}$ é modificado em cada iteração.

3.2.1 Custo Computacional do Algoritmo de Lucas-Kanade

Considere que o número de pixels em T é denotado por q e que o número de parâmetros do vetor $\mathbf{p}$ é denotado por n. Os passos descritos nesta sessão estão descritos no Algoritmo (1). O Passo 1 do algoritmo de Lucas-Kanade toma tempo $O(nq)$. Para cada pixel x em T é computada a função $w(\mathbf{x}, \mathbf{p})$ e então o valor resultado é interpolado dos valores de I. O custo de cada cálculo de $w(\mathbf{x}, \mathbf{p})$ depende da função escolhida, mas em geral toma tempo $O(n)$ para cada pixel. O Passo 2 custa $O(q)$. O Passo 3 toma o mesmo tempo que o Passo 1, geralmente $O(nq)$. Calcular a Jacobiana no Passo 4 também depende da função de transformação, mas o custo é geralmente de $O(n)$ por pixel. O custo deste passo é, portanto, $O(nq)$. O Passo 5 toma tempo $O(nq)$, Passo 6 consome tempo $O\left(n^2 q\right)$ e o Passo 7 toma tempo $O(nq)$. O Passo 8 consome tempo $O\left(n^3\right)$ para inverter a matriz Hessiana e tempo $O\left(n^2\right)$ para multiplicá-la pelo resultado do Passo 7. O Passo 9 toma tempo $O(n)$ para atualizar os parâmetros com os incrementos encontrados. O custo total de cada iteração do algoritmo é $O\left(n^2 q + n^3\right)$, tendo como passo mais computacionalmente custoso o Passo 6.

Algoritmo 1 Algoritmo de Lucas-Kanade. Consiste em aplicar as Equações 3.5 e 3.3 até que os parâmetros $\mathbf{p}$ convirjam. Geralmente, um limiar ε é utilizado. Como todos os passos dependem de $\mathbf{p}$, que é atualizado a cada passo de iteração, todos os passos precisam ser executados a cada iteração.

Itere:

1. Transforme I com $w(\mathbf{x},\mathbf{p})$ para calcular $I(w(\mathbf{x},\mathbf{p}))$
2. Calcule a imagem de erro $T(x) - I(w(\mathbf{x},\mathbf{p}))$
3. Transforme o gradiente $\nabla I w(\mathbf{x},\mathbf{p})$
4. Monte a Jacobiana $\frac{\partial w}{\partial \mathbf{p}}$ com $(\mathbf{x},\mathbf{p})$
5. Calcule $\nabla I \frac{\partial w}{\partial \mathbf{p}}$
6. Calcule a Hessiana utilizando a Equação 3.6
7. Calcule $\sum_{i=0}^{q} \left[\nabla I \frac{\partial w}{\partial \mathbf{p}}\right]^T [I(w(\mathbf{x}_i,\mathbf{p})) - T(\mathbf{x}_i)]$
8. Calcule $\triangle\mathbf{p}$ utilizando a Equação 3.5
9. Atualize os parâmetros $\mathbf{p} \leftarrow \mathbf{p} + \triangle\mathbf{p}$

enquanto $\triangle\mathbf{p} < \varepsilon$.

3.3 MINIMIZAÇÃO EFICIENTE DE SEGUNDA ORDEM - ESM

Como visto anteriormente, o método de Lucas-Kanade baseia-se em encontrar parâmetros $\mathbf{p}$ que minimizem o erro de alinhamento entre as imagens $T(\mathbf{x})$ e $I(w(\mathbf{x},\mathbf{p}))$. A Minimização Eficiente de Segunda Ordem (ESM do inglês *Efficient Second-order Minimization*) tenta aumentar a taxa de convergência do algoritmo utilizando mais termos da expansão da série de Taylor. O algoritmo de Lucas-Kanade expande a imagem $I(w(\mathbf{x},\mathbf{p_c}))$, onde $\mathbf{p_c}$ representa o vetor de parâmetros atual, da seguinte maneira:

$$T(\mathbf{x}) = I(w(\mathbf{x},\mathbf{p_c})) + J(\mathbf{p}) \triangle \mathbf{p}$$

onde $J(\mathbf{p})$ representa a matriz Jacobiana. O ESM expande $I(w(\mathbf{x},\mathbf{p}))$ utilizando mais um termo da série de Taylor, diminuindo o erro inerente da aproximação. Utiliza, então, a seguinte expansão:

$$T(\mathbf{x}) = I(w(\mathbf{x},\mathbf{p_c})) + J(\mathbf{p_c}) \triangle \mathbf{p} + \frac{1}{2} H(\mathbf{p_c}, \triangle \mathbf{p}) \triangle \mathbf{p} \tag{3.7}$$

onde $H(\mathbf{p}, \triangle \mathbf{p})$ é a matriz Hessiana que depende de $\triangle \mathbf{p}$ e as derivadas de segunda ordem de $I(w(\mathbf{x},\mathbf{p}))$ em relação a $\mathbf{p}$ avaliadas em $\mathbf{p_c}$. Como demonstrado em (BENHIMANE; MALIS, 2007) uma aproximação da Hessiana pode ser obtida aplicando uma expansão de primeira ordem em $J(\mathbf{p})$ avaliada em $\mathbf{p_0}$, sendo $\mathbf{p_0}$ os parâmetros inicias de rastreamento. A aproximação, então, dá-se como segue:

$$J(\mathbf{p_0}) \approx J(\mathbf{p_c}) + H(\mathbf{p_c}, \triangle \mathbf{p})$$

Isolando $H(\mathbf{p}, \triangle \mathbf{p})$ na equação acima e substituindo em 3.7 resulta em:

$$T(\mathbf{x}) = I(w(\mathbf{x},\mathbf{p})) + \frac{1}{2}(J(\mathbf{p_0}) + J(\mathbf{p_c})) \triangle \mathbf{p}$$

Isolando $\triangle \mathbf{p}$ na equação acima, encontra-se a lei de atualização dos parâmetros para a Minimização Eficiente de Segunda Ordem:

$$\triangle \mathbf{p} = -2\left(J\left(\mathbf{p_0}\right) + J\left(\mathbf{p_c}\right)\right)^{+}\left(I\left(w\left(\mathbf{x},\mathbf{p}\right)\right) - T\left(\mathbf{x}\right)\right) \tag{3.8}$$

onde $(.)^{+}$representa a pseudo-inversa de uma matriz.

A principal vantagem de utilizar uma aproximação para a expansão de segunda ordem sobre os parâmetros de transformação é um aumento na taxa de convergência. Outra vantagem do ESM é que este método evita alguns mínimos locais próximos ao mínimo global que seriam considerados pelo método original de Lucas-Kanade. Estas duas vantagens são demonstradas em (BENHIMANE; MALIS, 2004).

Nesta dissertação, é utilizada esta regra de atualização para todos os métodos implementados. Na formulação original, $J\left(\mathbf{p_0}\right)$ precisa ser calculado apenas uma vez, utilizando a imagem de referência $T\left(\mathbf{x}\right)$. Como será descrito no Capítulo 4, mudanças de iluminação fazem com que a nossa imagem de referência não se mantenha constante durante o rastreamento. Por este motivo, a Jacobiana $J\left(\mathbf{p_0}\right)$ precisa ser recalculada para cara novo quadro. Como esta operação é feita apenas uma vez por quadro, o custo assintótico de nossa implementação do método ESM é o mesmo que o do método original de Lucas-Kanade.

3.4 IMAGENS COLORIDAS

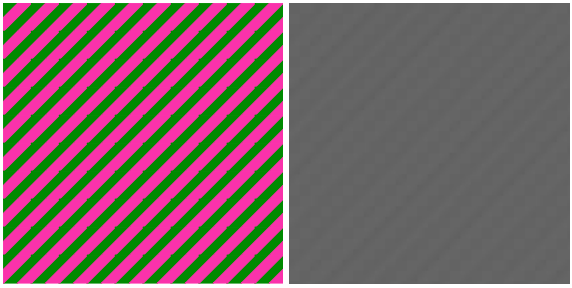
Como assumimos que a imagem I é representada por uma matriz de r linhas por c colunas, os métodos acima assumem que a imagem possui apenas um canal de cor. Inicialmente, estas imagens eram usadas pois diminuíam o esforço computacional para montar as matrizes necessárias para o cálculo dos parâmetros de transformação.

Imagens em tons de cinza, no entanto, carregam menos informação do que imagens coloridas. A Figura 6 demonstra esta diferença. As cores geralmente são definidas por três canais de cores. Como existem menos valores de cinza possíveis, algumas cores são mapeadas para o mesmo tom de cinza. Já que o rastreador utiliza o gradiente das imagens, é importante manter o máximo de informação possível nas imagens utilizadas para o rastreamento. Na Figura 6 ainda é possível extrair uma diferença grande entre as faixas com cor, mas esta informação é bem menos presente na imagem da direita.

Figura 6 – As imagens abaixo demonstram a perda de informação quando se transforma uma imagem colorida em uma imagem em tons de cinza. A imagem da esquerda contém mais informação útil para o rastreador do que a imagem da direita.

(a) Imagem com cor

(b) A imagem ao lado convertida para tons de cinza



Algumas pequenas mudanças são necessárias para que os métodos descritos aceitem imagens coloridas para o rastreamento. Em vez de utilizarmos uma matriz bidimensional para representar a imagem I, utilizaremos uma matriz tridimensional. Assim, a dimensão da matriz torna-se $r \times c \times t$, onde t representa o número de canais de cores utilizado pela imagem. Além disso, agora os pontos utilizados como posição de pixels possuem três coordenadas, duas para indicar a posição espacial na imagem e uma para indicar o canal de cor sendo acessado. As funções de transformação paramétricas não alteram esta terceira coordenada adicionada aos pontos.

Todas as fórmulas continuam sendo válidas quando se utilizando imagens coloridas. É importante notar que a quantidade de informação a ser processada aumenta quando se adicionam mais canais de cores. Tipicamente são utilizados três canais de cores (vermelho, verde e azul), o que faz com que o número de informação triplique.

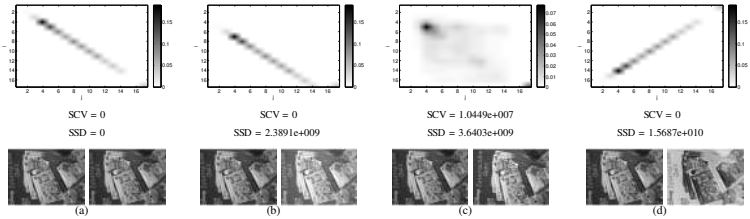
4 SOMA DA VARIÂNCIA CONDICIONAL

Como visto no Capítulo 3, o método de Lucas-Kanade utiliza a medida de similaridade Soma do Quadrado dos Resíduos. Esta medida de similaridade é bastante utilizada em algoritmos de rastreamento devido a sua simplicidade. Como muitas aplicações necessitam de uma resposta rápida para o rastreamento, a velocidade com que é possível calcular o SSD é um grande vantagem. No entanto, em ambientes naturais, mudanças de iluminação fazem com que esta medida se mostre pouco efetiva. Como ela considera apenas os valores absolutos das diferenças entre intensidade de cada pixel na imagem atual e na imagem de referência, ela não possui nenhuma robustez para mudanças de iluminação.

A Soma da Variância Condicional (SCV), no entanto, adiciona cálculos de distribuição conjunta à medida de similaridade para remediar esta falha do SSD. Esta medida de similaridade busca montar uma distribuição de probabilidade conjunta entre a imagem atual e a imagem de referência. Assim, uma das imagens pode ser compensada, de modo a remover as mudanças de iluminação.

Uma comparação entre estas duas medidas de similaridade é demonstrada na Figura 7. A imagem mostra o resultado das duas medidas quando aplicadas às duas imagens em cada caso, bem como a distribuição de probabilidade conjunta para o SCV. Em (a), as duas imagens são idênticas. Como esperado, as duas medidas apontam similaridade completa. Em (b), todos os pixels da imagem de referência foram somados de 50 tons de cinza. Aqui é possível perceber que a distribuição de probabilidade se assemelha muito ao do caso (a). Por este motivo, a medida SCV aponta similaridade completa entre as imagens. Enquanto isso, o SSD aponta uma grande dissimilaridade, já que considera os valores absolutos dos pixels. Em (c), as duas imagens não estão alinhadas, sem nenhuma modificação na iluminação. Neste caso, as duas medidas de similaridade apontam para dissimilaridades nas imagens, como esperado. Finalmente, em (d), a imagem de referência é a inversa da imagem atual. Aqui também a distribuição é parecida com a do caso (a), já que os pixels da imagem atual possuem uma relação direta com os da imagem de referência.

Figura 7 – Comparação entre SSD e SCV. É possível perceber que a medida SSD não consegue descrever bem a similaridade entre as imagens quando há mudanças na intensidade dos pixels (imagens b, c e d). A medida SCV descreve as similaridades com mais facilidade, até quando os valores dos pixels são invertidos (imagem d). Figura retirada de (RICHA et al., 2011).



Após a compensação de iluminação, o SCV se comporta similarmente ao SSD, fazendo a diferença dos erros quadráticos entre as duas imagens. Assim, o SCV pode ser descrito pela seguinte equação:

$$SCV = \sum_{x=0}^{q} \left(\hat{I}\left(w\left(\mathbf{x},\mathbf{p}\right)\right) - T\left(\mathbf{x}\right) \right)^2$$

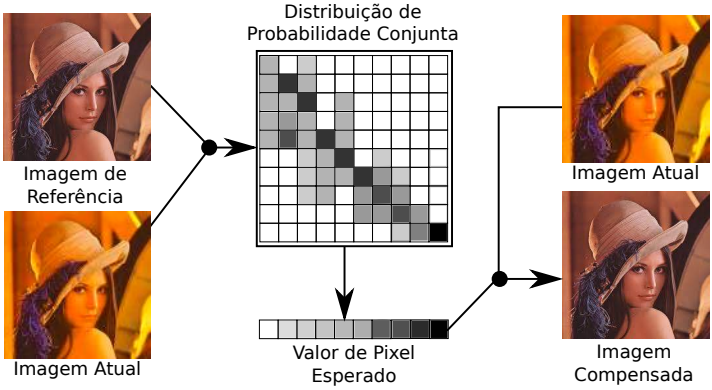
Sendo $\hat{I}\left(w\left(\mathbf{x},\mathbf{p}\right)\right)$ a imagem atual compensada utilizando a distribuição de probabilidade conjunta. Ou seja:

$$\hat{I}\left(w\left(\mathbf{x},\mathbf{p}\right)\right) = \xi\left(I\left(w\left(\mathbf{x},\mathbf{p}\right)\right) \mid T\left(\mathbf{x}\right)\right)$$

onde $\xi\left(.\right)$ representa o operador de expectância. É possível notar que o SCV é um caso genérico do SSD. Se tomarmos $\hat{I}\left(w\left(\mathbf{x},\mathbf{p}\right)\right) = \xi\left(I\left(w\left(\mathbf{x},\mathbf{p}\right)|T\left(\mathbf{x}\right)\right)\right) = I\left(w\left(\mathbf{x},\mathbf{p}\right)\right)$, a equação acima se torna equivalente ao SSD apresentado no Capítulo 3.

Para calcular a distribuição de probabilidade conjunta, utiliza-se tanto a imagem atual como a imagem de referência. Esta distribuição é uma matriz $d_I \times d_T$, onde d_I representa o número de níveis de cinza da imagem atual e d_T este mesmo número para a imagem de referência. Cada elemento desta matriz representa a probabilidade de

Figura 8 – Compensação de iluminação utilizada pelo SCV. A imagem de referência e a imagem atual são utilizadas para o cálculo da distribuição de probabilidade conjunta. A partir desta distribuição é calculado o valor esperado para cada pixel da imagem. A imagem compensada que será usada no rastreamento é montada utilizando os valores de pixel esperados e a imagem atual.



co-ocorrência ($I(w(\mathbf{x},\mathbf{p})) = i, T(\mathbf{x}) = j$) para cada pixel x:

$$P_{ij} = \frac{1}{q} \sum_{x=0}^{q} \phi\left(I(w(\mathbf{x},\mathbf{p})) - i\right) \phi\left(T(\mathbf{x}) - j\right)$$

onde $\phi(s) = 1$ para $s = 0$ e $\phi(s) = 0$ para $s \neq 0$. Além disso, $i \in [0, d_I]$ e $j \in [0, d_T]$. Utilizando esta distribuição é possível, então, calcular a imagem esperada utilizando:

$$\xi\left(I(w(\mathbf{x},\mathbf{p}))\,|T(\mathbf{x})\right) = \alpha^{-1} \sum_{i=0}^{d_I} i \cdot P_{ij}\left(i, I(w(\mathbf{x},\mathbf{p}))\right)$$

onde $\alpha = \sum_{i=0}^{d_I} P_{ij}\left(i, I(w(\mathbf{x},\mathbf{p}))\right)$.

Em outras palavras, esta medida de similaridade encontra o valor esperado para um valor de intensidade de pixel utilizando a imagem atual e a imagem de referência. A abordagem é ilustrada na Figura 8.

Após o cálculo da distribuição de probabilidade, a imagem atual é compensada utilizando esta informação. Durante o processo de rastreamento, a imagem atual compensada $\hat{I}(w(\mathbf{x},\mathbf{p}))$ precisa ser calculada uma vez para cada iteração da otimização. Como este é um problema linear sobre o tamanho das imagens $I(w(\mathbf{x},\mathbf{p}))$ e $T(\mathbf{x})$, a análise assintótica do método de Lucas-Kanade continua o mesmo após a adição desta medida de similaridade.

O SCV possui a característica de recalcular a imagem atual compensada $\hat{I}(w(\mathbf{x},\mathbf{p}))$ a cada iteração, diferentemente do SSD que utiliza apenas a imagem atual original. Esta compensação torna o SCV invariante a mudanças de iluminação. Como é possível perceber na Imagem 7(b), há apenas uma translação vertical da reta que representa a imagem, mas a forma da distribuição continua a mesma.

O SCV possui a vantagem de poder usar imagens de modalidades diferentes para o cálculo da semelhança, desde que a relação entre as intensidades dos pixels possa ser descrita como uma função.

4.1 SOMA DA VARIÂNCIA CONDICIONAL LOCAL

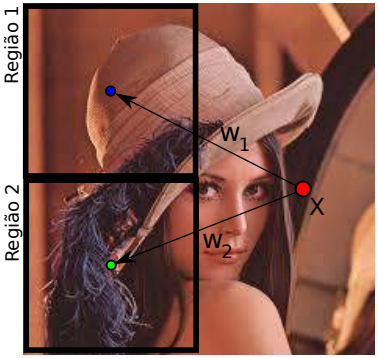
Como visto na seção anterior, a medida de similaridade Soma da Variância Condicional é invariante a mudanças de iluminação globais, mas não é invariante a mudanças locais. Isso se dá pelo fato de a distribuição de probabilidade ser calculada utilizando toda a imagem. Assim, quando há uma mudança em uma região de pixels da imagem, a distribuição de probabilidade é afetada como um todo, o que afeta a compensação de pixels que não sofreram alteração nenhuma.

Nesta dissertação, é proposto um método que busca contornar esta limitação. A compensação de iluminação agora levará em conta não a distribuição de probabilidade da imagem inteira, mas sim de distribuições menores, baseadas em sub-regiões da imagem. Assim, a compensação de iluminação que afeta cada pixel será uma interpolação das distribuições de probabilidade mais próximas.

Assim, a Soma da Variância Condicional Local (LSCV do inglês *Local Sum of the Conditional Variance*) é definida como segue:

$$LSCV = \sum_{x=0}^{q} \left(\hat{I}(w(\mathbf{x},\mathbf{p})) - \hat{T}(\mathbf{x})\right)^2$$

Figura 9 – Para realizar a interpolação do valor compensado do pixel X são utilizados todos as regiões da imagem. Neste exemplo, apenas duas regiões estão visíveis por simplicidade. O valor do pixel X então depende dos valores esperados das duas regiões. Os valores esperados são interpolados utilizando os pesos w_i baseados na distância euclidiana do ponto X para o centro das regiões (representados pelo ponto verde e pelo ponto azul).



A formulação geral dele é parecida com a do SCV, com a mesma $\hat{I}(w(\mathbf{x},\mathbf{p}))$ representando a imagem compensada utilizando as distribuições de probabilidades:

$$\hat{I}(w(\mathbf{x},\mathbf{p})) = \xi\left(I(w(\mathbf{x},\mathbf{p})) \mid T(\mathbf{x})\right)$$

Cada sub-região da imagem possui uma distribuição de probabilidade própria para a compensação de iluminação. Os valores esperados dos pixels são calculados a partir desta distribuição, mas apenas para esta porção da imagem. O valor compensado de um pixel específico da imagem é calculado a partir de uma interpolação dos valores esperados para aquele pixel de todas as regiões da imagem. Esta interpolação é feita utilizando o inverso da distância euclidiana como peso associado a cada distribuição. Deste modo, distribuições próximas possuem mais influência no cálculo da compensação de iluminação. A Figura 9 demonstra esta abordagem.

As distribuições de probabilidade são calculadas da mesma ma-

neira que no SCV:

$$P_k(i,j) = \frac{1}{q_k} \cdot \sum_{x=0}^{q_k} \phi\left(I\left(w\left(\mathbf{x},\mathbf{p}\right)\right) - i\right) \phi\left(T\left(\mathbf{x}\right) - j\right)$$

onde P_k é a distribuição de probabilidade da sub-região k e q_k representa o número de pixels na sub-região k. Utilizando estas distribuições, é possível calcular o valor esperado de cada pixel utilizando a seguinte fórmula:

$$\xi\left(I\left(w\left(\mathbf{x},\mathbf{p}\right)\right) \mid T\left(\mathbf{x}\right)\right) = \rho\left(\mathbf{x}\right)^{-1} \cdot \sum_{k=0}^{n} d_k\left(\mathbf{x}\right) \cdot P_k\left(k, T\left(\mathbf{x}\right)\right)$$

onde $\rho(x) = \sum_{k=0}^{n} d_k(\mathbf{x})$, ou seja, a soma dos pesos associados a todas as sub-regiões e $d_k(\mathbf{x})$ representa o inverso da distância euclidiana do ponto x para o centro da sub-região k.

Assim como no SCV, a imagem de referência e a imagem atual são usadas para a criação de uma distribuição de probabilidade conjunta. No entanto, com o LSCV, apenas uma porção das imagens são utilizadas para cada distribuição. O número de divisões pode ser escolhido pelo usuário. A escolha do número é uma troca entre a velocidade de cálculo (números maiores precisam de mais distribuições de probabilidade conjuntas) e qualidade de compensação. É esta divisão das imagens que faz com que a medida de similaridade seja robusta à mudanças de iluminação locais. Se, por exemplo, apenas o canto inferior direito for afetado por uma mudança de iluminação, o resto das distribuições de probabilidade não serão afetadas pela mudança. Isto não acontece no SCV pois a mesma distribuição é utilizada para toda a imagem.

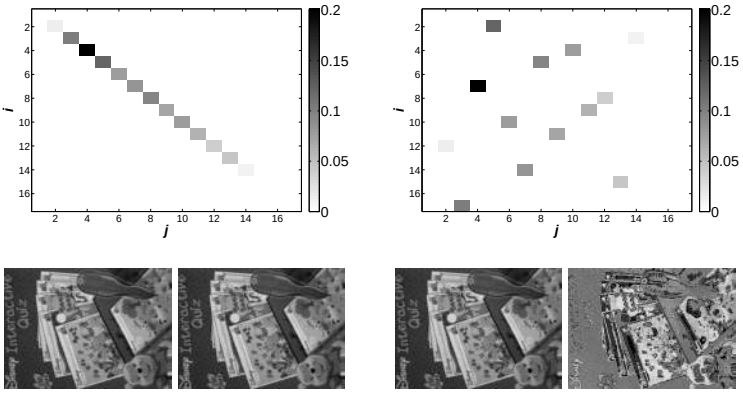
4.2 RESTRIÇÕES À SOMA DA VARIÂNCIA CONDICIONAL LOCAL

Algumas medidas de similaridade possuem um raio menor de convergência do que outras. Isto significa que elas possuem um raio menor de dissimilaridades entre a imagem de referência e a imagem atual para que possa haver convergência. É o que acontece entre o

Figura 10 – O método de compensação utilizado pelo LSCV lida bem com imagens do tipo (a). No entanto, existem configurações (como a mostrada na imagem (b)) que recebem uma baixa pontuação da medida de similaridade. A segunda configuração não acontece geralmente em cenários reais e portanto são consideradas configurações degeneradas.

(a) Duas imagens iguais

(b) Intensidades da imagem atual são uma permutação da imagem de referência

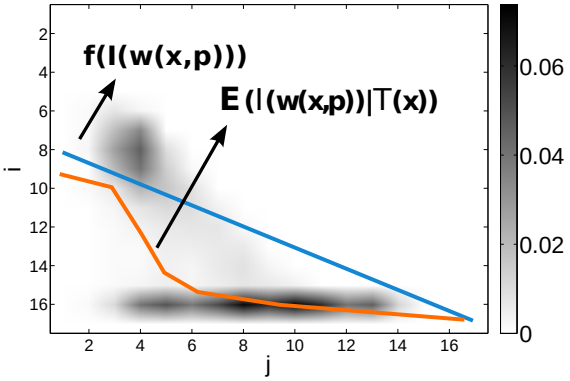


SCV e o SSD. A primeira medida de similaridade possui um raio de convergência menor que a segunda. Isto se dá por o SCV possuir invariância a permutações.

Como é possível perceber na Figura 10, as duas configurações da imagem atual recebem uma avaliação igual pela medida de similaridade. Apesar de esta característica permitir o rastreamento utilizando imagens multimodais, ele é prejudicial ao rastreamento direto realizado neste trabalho. O segundo tipo de configuração da Figura 10 não acontece em situações reais, nem com modificações extremas de iluminação. No entanto, como a medida de similaridade a considera como uma configuração válida, ela cria um mínimo local que pode atrapalhar o rastreamento.

Portanto, algumas restrições ao cálculo de compensação utilizado na Soma da Variância Condicional Local são adicionadas para

Figura 11 – Restrições da estratégia de compensação de iluminação para o LSCV. A curva laranja mostra a compensação calculada a partir do histograma. No entanto, a linha azul é a função aproximadora que é utilizada para realizar a compensação de iluminação. Esta abordagem evita estados degenerados, como mostrados na Figura 10.



evitar que o método considere as configurações degeneradas como válidas. Em vez de calcular a compensação de iluminação diretamente utilizando as distribuições conjuntas $P_k(i,j)$ como nas equações acima, será utilizada uma função de aproximação linear $f(I(w(\mathbf{x},\mathbf{p})))$ de modo que:

$$\hat{I}(w(\mathbf{x},\mathbf{p})) = f(I(w(\mathbf{x},\mathbf{p}))) = aI(w(\mathbf{x},\mathbf{p})) + b$$

Os parâmetros a e b são calculados a partir de P a cada quadro de modo a minimizar:

$$min_{a,b}||P(i,I(w(\mathbf{x},\mathbf{p}))) - (aI(w(\mathbf{x},\mathbf{p})) + b)||$$

A diferença entre o método original e a utilização desta restrição pode ser melhor visualizada na Figura 11. Em vez de utilizarmos os valores obtidos pela distribuição de probabilidade, utiliza-se uma aproximação linear que faz com que todos os pixels sigam a mesma tendência.

Este método possui a vantagem de utilizar todos os valores de

intensidade de pixel quando realizando a compensação de cada pixel, já que todos os valores influenciam na formulação da reta aproximadora. Isto faz com que os estados degenerados sejam evitados e aumenta a robustez da medida de similaridade.

Como dito, esta abordagem é uma tentativa de evitar estados degenerados que resultam em um pior rastreamento. No entanto, esta restrição é uma troca entre evitar estes estados e aumentar o erro médio da compensação de iluminação de cada pixel. Como cada pixel influencia a compensação de todos os pixels na sub-imagem (já que a reta de aproximação é formulada com base em todas os pixels da sub-imagem), a tendência é que o erro médio de compensação aumente. No entanto, é esperado que o aumento na robustez do rastreamento seja mais relevante do que o pequeno aumento no erro de compensação (que não se traduz diretamente em um aumento no erro do rastreamento).

5 DESENVOLVIMENTO

Com os dois métodos descritos nos Capítulos 3 e 4, torna-se possível a implementação de um método de rastreamento robusto tanto a mudanças de iluminação globais quanto mudanças locais.

A maioria das operações realizadas pelos algoritmos é de natureza paralela. Temos, por exemplo, as operações de transformação das imagens (mapeamento da imagem atual utilizando a função de transformação paramétrica). Esta operação aplica uma função a cada pixel da imagem. No entanto, não há interdependência entre pixels diferentes, de modo que todos podem ser calculados de maneira independente. Esta característica é comum em muitas outras operações, como o cálculo das imagens de gradiente e a operação de compensação de iluminação.

Outro grupo de operações presentes nos algoritmos são operações com matrizes. Estas dominam o tempo de execução, principalmente as operações de inversão e multiplicação de matrizes de passos intermediários. Assim como as operações com imagens, algumas destas operações são paralelas. Por exemplo, o cálculo das matrizes Jacobianas são independentes para cada pixel. Ou seja, cada entrada da matriz pode ser calculada independentemente.

Dada a natureza dos algoritmos, é natural implementá-los em um ambiente paralelo. Atualmente, um dos meios mais fáceis de implementar paralelismo em massa é utilizando placas de processamento gráfico (GPU do inglês *Graphics Processing Unit*). Inicialmente dedicadas a operações de cunho gráfico, hoje em dia estas placas expandiram seu mercado de atuação para qualquer aplicação que necessite de paralelismo massivo. Esta prática de utilizar GPUs para algoritmos gerais foi cunhada de Computação de Propósito Geral em GPUs (GPGPU do inglês *General-purpose Computing on Graphics Processing Units*).

Com o aumento de interesse em GPGPU, principalmente para aplicações científicas, foram criados *frameworks* e bibliotecas para o auxílio da programação nestes ambientes. As duas plataformas mais prevalentes são o OpenCL (GROUP et al., 2008) e o CUDA (NICKOLLS et al., 2008).

Por contar com um ambiente mais completo de desenvolvimento (*profiler, debugger,* etc.) e por fornecer um desempenho melhor em placas NVIDIA, a plataforma CUDA foi escolhida para a implementação dos algoritmos de rastreamento desta dissertação. Na próxima seção encontra-se um breve resumo desta plataforma, bem como um resumo da arquitetura da placa de vídeo utilizada para os experimentos. Estas informações são importantes para que o leitor compreenda as otimizações realizadas, bem como os benefícios e restrições que esta decisão traz à implementação realizada.

5.1 CUDA

CUDA, do inglês *Compute Unified Device Architecture*, é uma plataforma para computação paralela que fornece ao usuário não só funções e bibliotecas prontas para o uso, mas também um ambiente de programação completo. O principal objetivo da plataforma é tornar possível a programação GPGPU de forma rápida e eficaz.

Este *framework* fornece uma linguagem de programação baseada em C++, de modo que os programadores não precisem aprender uma linguagem nova para produzir programas. Este código é compilado com um compilador fornecido pelo *framework.* A principal diferença entre CUDA C++ e a linguagem em que se baseia é o modo de execução de algumas funções. Como parte do código será executado na placa gráfica, algumas funções precisam ser executadas por um número massivo de *threads* ao mesmo tempo. Estas funções são chamadas de *kernels* pela plataforma. Existe também uma diferenciação entre código que é executado na CPU e código executado em GPUs. Três palavras-chave denotam onde o código será executado: *host, device* e *global. Host* é a denominação usada para a CPU, o código que será utilizado para criar os *kernels* de execução paralela. *Device* são os *kernels* que serão executados exclusivamente em GPUs. Finalmente, quando uma função pode ser executada tanto em um *host* quando em um *device,* ela é chamada *global.*

Como utiliza uma linguagem própria, o *framework* também fornece um compilador próprio, chamado *nvcc*. Já que o objetivo desta dissertação é criar um *framework* de rastreamento que seja de fácil utilização e extensão, a utilização de um compilador diferente é

problemática. Este problema será abordado na seção que descreve a arquitetura do *framework.*

Por lidar naturalmente com dois núcleos de processamento (CPU e GPU), esta plataforma também fornece ao programador funções de utilidade para gerenciamento de memória nos dois núcleos. Todas as funções de gerência de memória no *host*, por exemplo, possuem uma função similar para gerência de memória no *device* (*e.g. malloc* e *cudaMalloc*). O comportamento destas funções tenta ser o mais próximo das funções originais o possível, de modo que um programador C++ possa produzir código mais facilmente.

Outro conceito importante de CUDA são blocos, *threads* e *grid.* O modelo de execução de CUDA executa uma mesma função *kernel* várias vezes em paralelo. Cada *thread*, portanto, executa o *kernel* uma vez. As *threads* são organizadas em blocos e os blocos são organizados em um *grid.* Isto é feito de modo que cada *thread* possua uma identidade própria, que o programador pode utilizar para modificar a execução do código. Os *grids* e os blocos podem ter de uma a três dimensões, atualmente. Esta organização foi escolhida porque o número de *threads* em um *grid* bidimensional pode ser mapeado facilmente para cada pixel de uma imagem, por exemplo. Como veremos na seção seguinte, é importante escolher uma distribuição adequada de *threads* por bloco e de blocos por *grid* para otimizar o desempenho das placas gráficas.

Por placas gráficas utilizadas para os experimentos do Capítulo 6 serem fabricadas pela empresa NVIDIA, este *framework* foi escolhido para a implementação por apresentar um desempenho superior. Além disso, o ambiente de desenvolvimento oferecido pelo CUDA é muito útil para programadores. Depurar erros de programação é possível utilizando o *debugger* fornecido, ainda sendo possível analisar o desempenho de cada chamada de função individualmente com o *profiler* do *framework*. Finalmente, existe muita literatura sobre as melhores práticas de programação em se tratando de programação paralela utilizando CUDA. Estes foram os principais motivos na escolha deste *framework.*

5.2 ARQUITETURA FERMI

Fermi é o nome da arquitetura das placas gráficas utilizadas nos experimentos do próximo capítulo. Nesta seção, os aspectos da arquitetura que afetam nossa implementação ou otimizações realizadas serão ressaltados. Uma descrição completa da arquitetura pode ser encontrado em (NVIDIA, 2011).

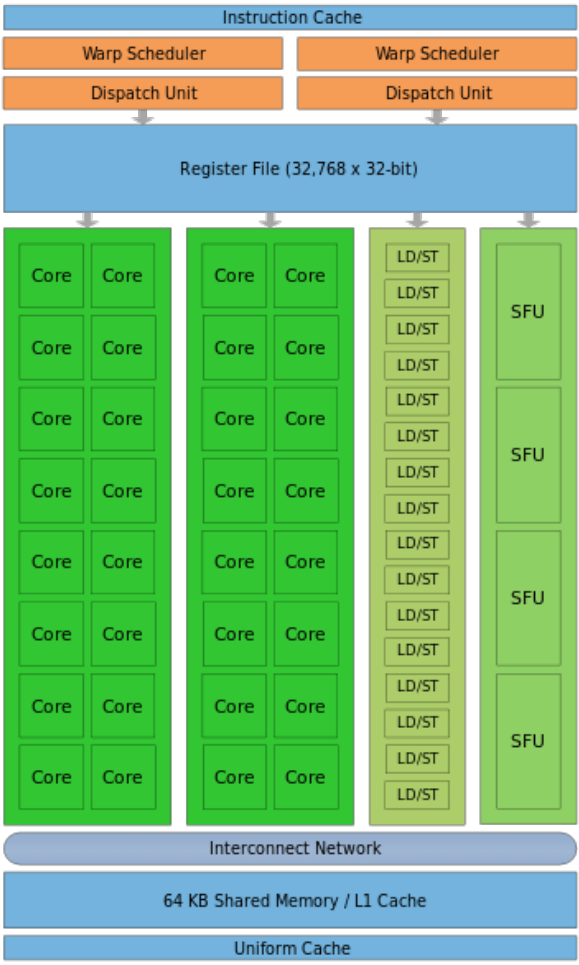
O modelo de execução de CUDA (dividido em *threads,* blocos e *grids*) é refletido nas placas gráficas desta arquitetura. Nela, cada GPU executa um ou mais *grids.* As GPUs são então divididas em *Stream Multiplocessors* (SM), que executam um conjunto de blocos de um determinado *grid*. Finalmente, as SMs são divididas em CUDA *cores*, que executam as *threads*. Esta arquitetura é montada para que *threads* de ID próximo compartilhem o mesmo banco de memória. É esperado que estas *threads* realizem leituras de memória em faixas parecidas da memória, de modo que estas buscas possam ser aceleradas pela utilização de *caches.*

Os *Stream Multiprocessors* realizam sua execução em conjuntos de 32 *threads*. Este conjunto de *threads* é denominado *warp*. As *threads* de um *warp* são então divididas em dois grupos, chamados de *semi-warps*. A execução das *threads* então acontece executando a mesma instrução para todas as *threads* de um *semi-warp.* Um programador não precisa se preocupar em tratar as *threads* como *semi-warps* se ele busca apenas um código que funcione corretamente. Porém, criar *kernels* que levem em consideração este número pode fornecer um ganho de performance significativo. Como todas as *threads* de um *semi-warp* executam a mesma instrução, há também um mecanismo para indicar quais *threads* estão ativas naquela instrução. Isto permite que *threads* de um mesmo *semi-warp* executem caminhos diferentes de uma estrutura de escolha de fluxo. É preciso notar, no entanto, que uma *thread* inativa ainda vai ocupar um *core* na execução, o que deteriora o desempenho do processador.

A Figura 12 mostra uma visão geral de um *Stream Multiprocessor*, com a divisão dos *cores* em dois *semi-warps.* Também é possível perceber que existe um *warp scheduler* para cada *semi-warp.* Este *scheduler* que escolhe qual *semi-warp* será executada em seguida.

Assim como o *hardware* de execução, a hierarquia de memória

Figura 12 – Visão de um *Stream Multiprocessor* da arquitetura Fermi. Os *cores* são divididos em dois grupos (*semi-warps*), que executam a mesma instrução. É possível perceber que existe um *warp scheduler* para cada *semi-warp.* Apesar de existirem 32 *cores*, existem apenas 4 *special function units* (SFU) que realizam operações especiais (por exemplo, funções aritméticas).



das placas gráficas desta arquitetura refletem o modelo de execução de CUDA. Cada *thread* possui um banco de registradores para usar como memória privada. Cada bloco possui uma memória compartilhada para todas as suas *threads*. Finalmente, existe a memória global que é acessível a todos os *grids.* Deste modo, uma *thread* pode escolher qual memória utilizar para guardar uma informação, dependendo da natureza do dado. Quando mais próxima da *thread* a memória, mais rápido é o seu acesso. Por este motivo, é preferível que dados de cálculo intermediários sejam guardados na memória local ou na memória compartilhada. A memória global é utilizada, geralmente, apenas para buscar os dados no início do cálculo e para escrever os resultados ao fim.

As arquiteturas das placas gráficas oferecem algumas soluções para comunicação entre *threads* durante a execução de um *kernel* paralelo. *Thread*s em um bloco podem se comunicar utilizando a memória compartilhada. Como a memória compartilhada é relacionada a um bloco, *threads* em um bloco não podem acessar dados de blocos diferentes, a não ser que estes sejam colocados em memória global (o que é muito custoso). Além disso, não há garantia na ordem de execução dos blocos. Por este motivo, é aconselhado criar algoritmos e funções em que as *threads* cooperem apenas com outras *threads* de seu bloco. Esta plataforma não possui nenhuma garantia na ordem de execução das *threads*, porém é possível sincronizá-las dentro de um bloco utilizando barreiras (em CUDA, existe a função *__syncthreads()* para este fim). O único modo de sincronizar blocos diferentes é esperar que a execução do *kernel* termine.

5.3 *FRAMEWORK* ANNIE

Utilizando as tecnologias descritas acima, um *framework* denominado Annie foi implementado para facilitar a utilização das técnicas de rastreamento direto. A principal preocupação para o design deste *framework* foi esconder a utilização de CUDA ou da placa gráfica dos usuários.

Das técnicas de rastreamento direto, algumas foram escolhidas para serem implementadas. O algoritmo de Lucas-Kanade, utilizando a ESM, foi implementado utilizando várias funções de transformação

paramétricas. Inicialmente, as funções implementadas contemplavam apenas translação e rotação de objetos na tela (três graus de liberdade). Estas funções foram expandidas até suportarem homografias. A medida de similaridade implementada nestes métodos foi a SSD. Como visto anteriormente, estas implementações não possuem nenhum tipo de compensação de iluminação.

O algoritmo descrito por Silveira e Malis (2010) também foi implementado, de modo a fornecer rastreamento com compensação de iluminação. Este método é robusto a mudanças de iluminação locais, porém acarreta num aumento do número de parâmetros a serem estimados, já que adiciona parâmetros de iluminação ao passo de minimização.

Como alternativa mais leve, é fornecido também um rastreamento utilizando a medida de similaridade SCV. Como visto no Capítulo 4, esta medida é apenas robusta a mudanças de iluminação globais, porém possui a vantagem de não precisar estimar parâmetros de compensação de iluminação. Finalmente, o método proposto de iluminação local LSCV foi adicionado ao *framework.*

Todos estes métodos possuem variações em suas implementações que permitem ao usuário rastrear tanto imagens em níveis de cinza quanto imagens coloridas.

O *framework* se organiza de modo a esconder do usuário a linguagem de programação específica de CUDA. Isto é feito de modo que o usuário possa compilar seu código utilizando apenas o compilador de C++ usual, sem precisar recorrer ao compilador fornecido pela plataforma. Um exemplo desta filosofia são as classes *Matrix* e *CUDAMatrix*. A primeira é uma abstração para matrizes, feita em C++, que lida com os dados em memória de *host.* Associada a cada *Matrix* está uma classe *CUDAMatrix* que realiza as operações em memória de *device*, além de realizar o controle de memória (alocação e desalocação), de forma automática. Assim, o usuário precisa apenas se preocupar em definir o tamanho da matriz a ser utilizada e o *framework* realiza o resto das operações necessárias.

Além de funções de rastreamento direto, foram criadas funções de utilidade para programação de *kernels*. Assim, é possível para o usuário construir seus próprios *kernels* e incorporá-los às implementações. Uma destas funções é a rotina de escolha de tamanhos de blocos e *grids* para a execução de *kernels.* É necessário escolher um número

de *threads* que vão executar determinado *kernel*, bem como sua disposição entre *grids* e blocos. Portanto, foi adicionado ao *framework* uma função que calcula este número dado o tamanho de uma imagem ou matriz que serão usados em um *kernel.* Além disso, existem funções para detalhamento de erros que podem ocorrer em tempo de execução, funções para realizar o *binding* de texturas na placa gráfica, etc.

5.4 OTIMIZAÇÕES

O desenvolvimento de algoritmos para execução em placas gráficas é diferente do desenvolvimento convencional. Como o modelo de execução é diferente, são necessárias estratégias específicas para realizar otimização de código. Já que a aplicação de rastreamento direto precisa ser feita da maneira mais rápida possível, de modo que se possa alcançar uma boa interação com os usuários, foram aplicadas estas otimizações às principais funções que são executadas pelo *framework* nas placas gráficas.

Primeiramente, é necessário escolher um número ótimo de *threads* e sua distribuição. Elas precisam ser distribuídas de modo que ocupem a maior porcentagem possível da GPU. Como cada *thread* precisa de um número de recursos (registradores, processadores, etc) e cada *Stream Multiprocessor* possui um número máximo de blocos a serem executados simultaneamente, existe uma configuração de *threads* em blocos que garante o maior espalhamento possível. Esta configuração é dependente do modelo da placa de vídeo utilizada (no caso desta dissertação, foi utilizada uma GTX 570). A função de utilidade fornecida pelo *framework* Annie é uma função genérica, que não garante que a distribuição seja ótima. Isto acontece pois cada *kernel* possui um número de registradores específico como requerimento para sua execução. Por isso, para as funções que não atingiam uma distribuição ótima com a função genérica, foram escolhidos valores específicos, de maneira empírica.

Além disso, existem funções que não possuem sua execução determinada por entradas em uma imagem ou em uma matriz. Estas funções também receberam um número de *threads* por blocos e blocos por *grid* específico.

Como descrito anteriormente, as placas gráficas possuem três

tipos de memórias. A memória global é acessível a todas as *threads*, porém é uma memória de acesso lento. Existem, no entanto, a memória compartilhada (entre *threads* de um bloco) e a memória privada, que são de acesso mais rápido. Uma das maiores preocupações quanto a otimização de *kernels* são os acessos de memória.

Um aspecto muito importante para a otimização de acessos é a coalescência de memória. Este conceito trata do agrupamento de *threads* para que elas acessem endereços de memória consecutivos. Isto é benéfico à execução do *kernel* pois é mais rápido para a memória trazer um bloco de endereços sequenciais. Se as *threads* acessassem endereços aleatórios de memória, cada *thread* gastaria um ciclo de memória para buscar sua informação da memória global. Quando há coalescência de memória, é possível servir mais de uma *thread* por ciclo de busca, diminuindo assim o tempo em que as *threads* ficam esperando pela informação para o cálculo. Coalescência de memória afeta mais memória global do que as memórias compartilhadas e privadas, dado seu tempo de acesso superior. É importante notar que coalescência de memória também acontece na escrita, não apenas na leitura da memória global.

Outra técnica para acelerar o acesso de memória por *threads* é o *caching* em memória compartilhada. Esta otimização também baseia-se na diferença entre o tempo de acesso das memórias global e compartilhada. *Kernels* que precisam acessar dados globais mais do que uma vez durante sua execução podem realizar o acesso em memória global apenas uma vez e realizar o *caching* destes valores em memória compartilhada. É importante lembrar que estes acessos em memória global também devem ser coalescidos. É comum, dividir um *kernel* em três etapas. Primeiramente, todas as *threads* trabalham em conjunto para trazer da memória global os valores que serão utilizados por este bloco. Depois, realizam os cálculos específicos do *kernel*. Ao fim dos cálculos, os resultados podem ser escritos na memória global novamente, de preferência também aproveitando da coalescência de memória.

Além das otimizações de código, algumas técnicas utilizam seleção de pixels para tornar os métodos menos custosos. Apesar de não ser explorado nesta dissertação, esta abordagem mostra-se bastante efetiva em diminuir o custo de rastreamento (MEILLAND; COMPORT; RIVES, 2011).

6 RESULTADOS EXPERIMENTAIS

6.1 DESCRIÇÃO DOS EXPERIMENTOS

No Capítulo 5 foi apresentada a implementação das técnicas descritas nos Capítulos 3 e 4. O objetivo deste Capítulo é utilizar esta implementação para atestar a velocidade, robustez e efetividade do método proposto quando comparado com métodos estabelecidos. Para tanto, será utilizado um método presente na literatura que utiliza *Thin Plate Splines* (doravante denominado TPS) (SILVEIRA; MALIS, 2010) para realizar a compensação de iluminação local. Este método foi implementado no *framework* Annie. Assim, neste Capítulo serão comparados os métodos que utilizam as seguintes estratégias de compensação de iluminação:

- Soma da covariância local (LSCV)
- *Thin Plate Splines* (TPS)
- Nas sequências coloridas também será executada uma variação do TPS denominada TPS-3

Os métodos acima serão analisados de modo a comparar os seguintes quesitos:

- velocidade de processamento por quadro
- erro fotométrico de rastreamento
- número de iterações necessárias para a convergência do rastreamento
- um vídeo com o resultado de rastreamento dos métodos abordados para comparação visual
- tempo de processamento de cada função crítica para o rastreamento
- utilização do *framework* proposto para a implementação de uma aplicação exemplo

Tabela 1 – Parâmetros de rastreamento utilizados para todas as sequências.

Parâmetro	Valor
Número máximo de iterações	20
Valor mínimo de atualização (ε)	0.01

O *framework* Annie foi utilizado para a execução de rastreamento direto em sequências de vídeo utilizando os diferentes métodos mencionados. Os testes serão divididos em duas categorias: vídeos em cores e vídeos em tons de cinza. Esta divisão é feita porque o método TPS utiliza apenas uma *Thin Plate Spline* para realizar a compensação de iluminação. O método LSCV gera um conjunto de histogramas para cada canal, de modo que a compensação de iluminação é feita separadamente para cada um deles. De modo a fazer uma comparação semelhante, também é comparado o método LSCV com um método chamado TPS-3 nas sequências com cores. O método TPS-3 se assemelha muito ao método TPS convencional, mas utiliza três *Thin Plate Splines* para realizar a compensação em cada canal das imagens coloridas. Como essa adição de duas *splines* a mais para serem estimadas adiciona muitos parâmetros ao passo de otimização, é esperado uma queda de velocidade de cálculo significativa.

Os vídeos utilizados nos experimentos foram retirados em sua maioria de (SILVEIRA; MALIS, 2010). As sequências retiradas deste trabalho foram, em tons de cinza: Pooh, Pooh-2, Vulcan e Lunchbox, e em cores: Cat e Vangogh. Além destes vídeos, a sequência Jonas foi capturada para estes experimentos, utilizando uma câmera GoPro. As imagens foram retificadas antes de servirem de entrada para o rastreamento. Portanto, serão usadas quatro sequências em tons de cinza e três sequências em cores. A Tabela 2 mostra um resumo das sequências.

É necessário definir um conjunto de parâmetros para a execução do rastreamento. Todas as sequências possuem tamanhos de quadro diferentes, isso foge do controle do usuário e depende apenas do vídeo capturado. Além disso, para cada sequência também foram escolhidos tamanhos de imagens de referência diferentes. Isto foi feito para que fosse possível verificar a diferença entre os algoritmos

Tabela 2 – Configuração das sequências utilizadas para os experimentos. O tamanho da referência foi escolhido baseado na configuração do objeto de interesse no primeiro quadro da sequência.

Sequência	Nome	Tamanho do quadro	Tamanho da referência
	Pooh	(800x600)	(370x260)
	Pooh-2	(800x600)	(440x350)
	Lunchbox	(800x600)	(140x140)
	Vulcan	(800x600)	(300x400)
	Jonas	(1920x1080)	(400x400)
	Vangogh	(800x600)	(280x180)
	Cat	(640x480)	(250x300)

quando há variação nestas dimensões. Os parâmetros que mantiveram o mesmo valor para todas as sequências são mostrados na Tabela 1. O número máximo de iterações é um compromisso entre a qualidade do rastreamento e do tempo de cálculo. Foi escolhido o valor de 20 iterações empiricamente. Assim, quando um rastreador chega ao número de 20 iterações de otimização, o *loop* é interrompido e o rastreamento continua para o próximo quadro. Finalmente, o parâmetro ε, também utilizado para interromper o *loop* de otimização, permaneceu constante para todos os experimentos. O valor empiricamente escolhido foi de 0.01. Assim, quando a soma da atualização dos parâmetros de rastreamento for menor que 0.01, o algoritmo passa para o próximo quadro da sequência.

A imagem de referência para cada rastreamento é escolhida utilizando o primeiro quadro de cada sequência. Um ponto foi escolhido manualmente para cada vídeo, de modo que a imagem de referência com os parâmetros escolhidos englobe o máximo do objeto de interesse possível. Este ponto foi então utilizado para todos os rastreamentos que utilizam o vídeo.

Nas Seções 6.3 e 6.4 são descritos os resultados dos experimentos de rastreamento. Neles, são descritas três estatísticas da execução: tempo médio gasto por quadro, média do erro de rastreamento e média do número de iterações. O erro de rastreamento é medido fazendo a média das diferenças absolutas entre os pixels da imagem de referência e da imagem obtida após a aplicação da função de transformação na imagem atual. Este erro está descrito como Média do Erro de Intensidade Absoluto (MEIA) nas tabelas. Como este erro é apenas uma comparação entre as duas imagens, ele é um indicador apenas de quão bem o método de compensação altera a imagem de referência para refletir as mudanças de iluminação presentes na imagem atual do método. O número médio de iterações é relativo às iterações necessárias para a convergência da otimização para cada quadro.

6.2 AMBIENTE DE EXECUÇÃO

O mesmo ambiente foi utilizado para realizar os experimentos das Seções 6.3, 6.4, 6.5 e 6.6. Foi utilizado um computador com processador AMD FX(tm)-6100 com seis *cores*, com 12 GB de memória

Tabela 3 – Resultados de rastreamento para sequências em tons de cinza.

Sequência (Tamanho da Referência)	**Algoritmo**	**Tempo (ms)**	**MEIA**	**Iterações**
Pooh (370x260)	LSCV	85.45	15.23	11.08
	TPS	123.57	7.61	11.35
Pooh-2 (440x350)	LSCV	86.29	9.10	8.00
	TPS	134.69	5.80	8.73
Vulcan (300x400)	LSCV	103.73	20.61	13.68
	TPS	186.05	5.43	16.74
Lunchbox (140x140)	LSCV	43.31	11.04	8.08
	TPS	60.70	7.5	24.20

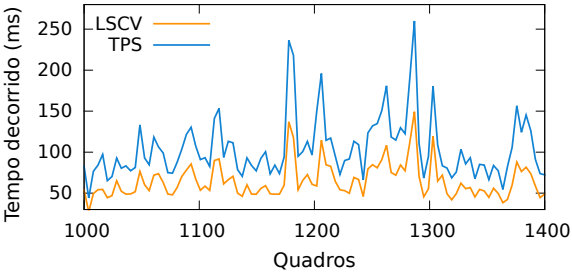
RAM e uma placa de vídeo Nvidia GeForce GTX 570. O ambiente utilizado para a Seção 6.7 é diferente dos outros experimentos, utilizando um computador Intel i7 3.8GHz com 4 *cores* (com *hyper-threading*), 32GBytes de RAM e uma placa gráfica Nvidia GeForce GTX 570.

6.3 RESULTADOS DAS SEQUÊNCIAS EM TONS DE CINZA

Os resultados do rastreamento utilizando as sequências em tons de cinza estão resumidos na Tabela 3. Eles demonstram claramente uma diminuição no tempo de resposta do algoritmo. Em todas as séries o algoritmo LSCV possui um tempo de processamento menor. Isto se dá principalmente pelo fato de ele não precisar estimar parâmetros de iluminação na etapa de otimização. Em média, o método LSCV é 34.8% mais rápido que o método TPS para as sequências em tons de cinza. O exemplo de maior ganho é a sequência **Vulcan**, que possui uma diminuição de 44,2% no tempo de resposta do algoritmo quando com parado com o método TPS.

Também é possível notar que o número de iterações necessárias para convergência é significativamente menor quando utilizando o método LSCV, especialmente na sequência **Lunchbox**. Isto também é

Figura 13 – Gráfico comparando o desempenho dos métodos LSCV e TPS na sequência **Pooh**. É possível perceber que o método LSCV se mantém mais rápido durante todo o rastreamento, mas a diferença é mais perceptível em partes difíceis do rastreamento.



causado pela estimação de parâmetros de iluminação no método TPS. Como existem mais parâmetros a serem estimados, mais iterações são necessárias para atingir o valor limiar ε.

No entanto, a média do erro (MEIA) é maior quando se utiliza o método LSCV. Felizmente, este aumento no erro não se traduz em uma diminuição na qualidade do rastreamento, como será demonstrado na Seção 6.6. Este aumento no erro já era esperado. O método TPS utiliza uma abordagem mais flexível para estimar a compensação de iluminação, utilizando uma superfície deformável. Como também são adicionadas restrições à compensação de iluminação do método LSCV, um aumento no erro é natural.

A Figura 13 mostra uma comparação entre o tempo gasto por quadro entre os métodos LSCV e TPS. É possível perceber que o primeiro é mais rápido em todas as partes do rastreamento. Além disso, a diferença entre o tempo gasto por quadro é maior nas partes de rastreamento mais difícil. Já que o método LSCV gasta menos tempo por iteração de otimização, quando é necessário mais iterações para convergência, o ganho no uso deste método é ainda maior.

Em imagens de tons de cinza, a utilização do método LSCV já se mostra benéfica. O aumento de velocidade é significativo e o aumento do erro não se traduz diretamente em erros de rastreamento.

6.4 RESULTADOS DAS SEQUÊNCIAS EM CORES

Os resultados das sequências em cores encontram-se na Tabela 4. É possível perceber que os resultados seguem a mesma tendência que os resultados para as sequências em tons de cinza. Diferentemente das sequências em tons de cinza, aqui são comparados os desempenhos dos três métodos: LSCV, TPS e TPS-3.

Os resultados mostram que o método LSCV é o mais rápido para o rastreamento. Esta diferença é ainda mais aparente do que nas sequências anteriores. Quando compara-se o método LSCV com o TPS, o método proposto é, em média, 57.5% mais rápido nestas sequências. O caso onde houve maior ganho foi a sequência **Vangogh**, que obteve melhora de 78,3% no tempo de cálculo. Como dito anteriormente, esta diferença se dá pelo fato de que o método LSCV não precisa estimar parâmetros de iluminação. Isto é confirmado pela diferença ainda mais acentuada de desempenho quando compara-se o método LSCV com o TPS-3. Este segundo método precisa estimar três vezes mais parâmetros que o método TPS convencional. Assim, o método LSCV é, em média, 91,2% mais rápido que o método TPS-3. O fato de segundo utilizar três *Thin Plate Splines* para realizar a compensação de luz afeta muito seu desempenho, tornando impraticável a sua utilização.

O número de iterações necessárias pra convergência também é menor para o método LSCV nas sequências coloridas. Isto também pode ser explicado pelo menor número de parâmetros a serem estimados. Esta diferença é bastante perceptível quando comparando os métodos com o TPS-3.

Estes resultados diferem dos resultados para sequências em tons de cinza em um ponto importante. O erro do método LSCV nestas sequências é muito mais próximo do método TPS que nas sequências em tons de cinza. Em duas sequências, **Cat** e **Vangogh**, o primeiro alcançou um erro menor do que o segundo. Esta diferença pode ser explicada pelo modo como o método TPS faz compensação de iluminação. Ele utiliza apenas uma superfície para realizar compensação nos três canais de cor. Isto não era um problema em imagens em tons de cinza, já que elas só possuem um canal para ser compensado. Em imagens coloridas, no entanto, esta abordagem não é flexível o

Tabela 4 – Resultados de rastreamento para sequências em cores

Sequência	Algoritmo	Tempo (ms)	MEIA	Iterações
Cat (250x300)	LSCV	111.68	14.88	11.93
	TPS	191.00	15.07	12.11
	TPS-3	1609.75	11.39	36.1
Jonas (400x400)	LSCV	277.97	10.87	10.73
	TPS	588.39	7.86	17.32
	TPS-3	1949.97	7.39	24.07
Vangogh (280x180)	LSCV	100.36	10.04	9.47
	TPS	463.17	10.86	42.76
	TPS-3	2064.22	10.51	75.98

suficiente. O método LSCV realiza a compensação em cada canal, utilizando um conjunto de histogramas por canal. Assim, é possível atingir um erro menor. Isto é reforçado pelo erro calculado quando utilizando o método TPS-3. Este método é a implementação que realiza compensação de iluminação em cada canal separadamente (uma superfície para cada canal). Como é possível perceber na tabela, este método possui o menor erro dos métodos testados. No entanto, seu desempenho em tempo deixa muito a desejar quando comparado com o método LSCV.

O uso do método LSCV se torna ainda mais vantajoso em sequências de imagens coloridas. O método possui um tempo de resposta menor que o método TPS, ainda assim conseguindo atingir um erro de intensidade menor. Para atingir a mesma precisão que o método LSCV, seria preciso utilizar o método TPS-3. Porém, este método possui um tempo de resposta proibitivo para a maioria das aplicações que são o foco deste trabalho.

6.5 TEMPO DE EXECUÇÃO POR FUNÇÃO

Utilizando o tempo gasto por iteração já é possível constatar que o método LSCV é mais rápido que o método TPS. No entanto, comparando o tempo gasto em cada função crítica para o rastreamento

Tabela 5 – Comparação do tempo de execução de funções críticas para o rastreamento utilizando os métodos LSCV e TPS. O tempo é medido em milissegundos. São utilizadas duas sequências: **Pooh** e **Vangogh** (em tons de cinza e colorida, respectivamente). É possível perceber que, apesar de utilizar mais tempo para realizar a compensação de iluminação, o método LSCV é mais eficiente ao calcular a matriz Jacobiana.

Função	LSCV	TPS
Compensação de iluminação (**Pooh**)	0.503	0.314
Compensação de iluminação (**Vangogh**)	0.768	0.181
Cálculo da matriz Jacobiana (**Pooh**)	0.966	3.817
Cálculo da matriz Jacobiana (**Vangogh**)	1.375	5.415

por cada método é possível perceber onde está o ganho principal do método LSCV. A Tabela 5 mostra estes valores para as duas técnicas. Nela estão comparados os tempos de compensação de iluminação e de cálculo da matriz Jacobiana para uma sequência em tons de cinza (**Pooh**) e uma sequência colorida (**Vangogh**). As outras funções críticas (como *warp* e cálculo de gradiente) não são mostradas nesta tabela pois são idênticas para ambos os métodos.

É possível perceber que a compensação de iluminação é mais custosa. O método LSCV precisa calcular os parâmetros para a compensação de iluminação, enquanto o método TPS faz este passo dentro da atualização de parâmetros a cada iteração do passo de otimização. O contrário acontece, como esperado, para o cálculo da matriz Jacobiana. Como o método TPS possui mais parâmetros para calcular, a sua matriz Jacobiana é significativamente maior que a do método LSCV. Na sequência em cores, o método LSCV é quase quatro vezes mais rápido que o método TPS para o cálculo desta função.

Além de tornar possível contabilizar onde se encontra o ganho de tempo do método LSCV, a medição do tempo de cálculo de funções críticas pode nos informar a melhoria da implementação utilizando GPUs sobre uma implementação tradicional. Na Tabela 6 encontram-se os tempos de funções críticas utilizando o *framework* Annie (GPU) e utilizando o *framework* Naya (CPU) (RICHA, 2013). Em ambos os casos utiliza-se o método TPS com uma sequência em preto e branco

(**Pooh**), já que este era o método implementado pelo *framework* Naya. Neste caso contabilizaremos o tempo gasto para o cálculo das seguintes funções:

- Função *warp*
- Compensação de iluminação
- Cálculo das imagens gradiente
- Cálculo da matriz Jacobiana
- Transferência de memória (no caso da implementação em GPU)

É possível perceber que há um ganho em velocidade de cálculo em todas as funções analisadas. A compensação de iluminação é a mais afetada pela implementação em GPU, sendo mais do que 30 vezes mais rápida que a implementação em CPU. Com os resultados obtidos é possível perceber que a implementação em GPU é preferível quando comparada com a implementação em CPU. A natureza paralela de muitas das funções críticas para o rastreamento faz com que o ganho de desempenho seja grande. O único ponto onde a GPU é menos eficiente do que a CPU é na transferência de memória. É necessário transferir o quadro atual para a memória da GPU, o que faz com que a aplicação possua um *overhead* quando implementada nesta arquitetura. No entanto, esse tempo de transferência é pequeno quando comparado com os tempos necessários para a realização dos cálculos de rastreamento. Além disso, esta transferência precisa ocorrer apenas uma vez por quadro.

6.6 VÍDEO DE RESULTADOS

Os resultados tanto das sequências em tons de cinza quanto das sequências em cores foram compilados em um vídeo. Nele, é possível acompanhar o rastreamento realizado por todas as técnicas descritas acima, simultaneamente. Além das técnicas já descritas nas seções anteriores, foi adicionado ao vídeo o rastreamento utilizando uma técnica de compensação de iluminação global. O método escolhido foi

Tabela 6 – Comparação do tempo de execução de funções críticas para o rastreamento utilizando duas implementações, uma executada na CPU e outra na GPU. Aqui as duas implementações utilizam o algoritmo TPS, rastreando a sequência **Pooh**. A última coluna mostra qual a porcentagem do tempo de execução na CPU é gasto pela implementação da GPU. No caso da compensação de iluminação, a GPU executa a mesma função em apenas 2% do tempo da CPU. As execuções são medidas em milissegundos. A implementação em GPU executa todas as funções mais rapidamente que a implementação em CPU.

Função	**CPU**	**GPU**	**GPU/CPU**
Compensação de Iluminação	10.779	0.314	2%
Cálculo da matriz Jacobiana	20.845	3.817	18%
Cálculo das imagens gradiente	0.895	0.039	5%
Função *warp*	3.616	0.117	3%
Transferência de memória	-	0.369	-
Total das funções críticas	36.135	4.287	11%

um método baseado na medida de similaridade SCV. Assim, é possível verificar a melhoria que o LSCV proporciona ao rastreamento, comparando com esta técnica.

O vídeo está disponível *online* [1]. Analisando o vídeo, é possível perceber que as três técnicas de compensação de iluminação local possuem uma precisão muito próxima. Os resultados dos três métodos são idênticos na maioria dos quadros. Como dito na Seção 6.3, o aumento do erro do método LSCV em relação ao TPS não se traduz diretamente numa diminuição na precisão do rastreamento. No entanto, também é possível perceber que o método de compensação global SCV não é robusto o suficiente para rastrear as sequências. Este método falha (perde o objeto de interesse) na maioria das sequências. O vídeo demonstra claramente a necessidade de se utilizar uma abordagem de compensação de iluminação local.

Todas as sequências foram rastreadas com sucesso pelos métodos de compensação local. A única exceção sendo a sequência **Pooh-**

[1] http://youtu.be/DebOYdLGebo

2, onde o método LSCV falha ao final. Nesta etapa da sequência, há uma mudança de iluminação muito drástica que o método proposto não foi capaz de superar. Isto provém das restrições descritas no Capítulo 4. Dada a grande diferença entre os valores dos pixels nesta parte da sequência, a aproximação linear proposta não é flexível o suficiente para compensar estas imagens. Como a compensação de iluminação não alcança os valores desejados, o erro entre a imagem atual e a imagem de referência cresce muito, o que faz com que o rastreamento falhe.

6.7 RESULTADOS UTILIZANDO UMA APLICAÇÃO EXEMPLO

Os experimentos descritos acima são apenas uma aplicação direta do algoritmo de rastreamento visual direto. Para verificar o desempenho da implementação do *framework* Annie em placas gráficas, este Capítulo descreve um experimento utilizando uma aplicação real. A aplicação *Trackscan* (RICHA et al., 2014), utilizada para criar mapas de retina durante exames oftalmológicos, foi escolhida para este experimento. Aqui será feita uma substituição do módulo de rastreamento da aplicação pelos rastreadores do *framework* Annie. Com isso será possível calcular a diferença dos tempos de rastreamento das funções críticas para as duas implementações.

O *Trackscan* possui uma implementação que utiliza a CPU. Como visto nas seções passadas, é esperado que o *framework* Annie possua um desempenho melhor, já que é implementado na GPU. No entanto, esta implementação em CPU foi paralelizada com a biblioteca OpenMP para utilizar os diferentes processadores presentes na máquina onde o experimento foi executado. Esta paralelização, apesar de ser muito menor em granularidade do que a efetuada nas GPUs, já confere ao algoritmo um tempo de resposta muito mais rápido. Nesta seção compara-se os tempos dos seguintes rastreadores:

- *framework* Annie utilizando GPUs
- rastreador *Trackscan* utilizando a CPU com apenas uma *thread* (denominado *Trackscan(1)*)
- rastreador *Trackscan* utilizando a CPU com seis *threads* (deno-

minado *Trackscan(6)*)

Nesta Seção, foram contabilizadas as mesmas funções críticas utilizadas na Seção 6.5. Como o rastreamento da retina para o *Trackscan* utiliza apenas translações e rotações, todas as implementações utilizam um algoritmo com apenas três graus de liberdade. Além disso, utilizam apenas dois canais de cores: vermelho e verde. Os parâmetros de rastreamento (número máximo de iterações e limiar de atualização) são os mesmos utilizados pelos experimentos das seções passadas. A Tabela 7 mostra os resultados de tempos médio de execução de funções críticas para as três implementações. A Figura 14 resume os resultados da Tabela 7 comparando os tempos de execução das implementações Annie e *Trackscan(6)* com a implementação *Trackscan(1).*

É possível notar que a utilização do *framework* Annie para o rastreamento nesta aplicação faz com que todas as funções sejam executadas mais rapidamente. A função *warp* é a única que possui um desempenho pior, mas apenas quando comparada com a versão *multithread* da implementação do *Trackscan*. É importante lembrar, no entanto, que a implementação em GPU possui um *overhead* de transferência de memória que não está presente nas implementações em CPU. Como as imagens estão originalmente na memória principal do computador, elas precisam ser transferidas para a memória da GPU antes que esta possa realizar os cálculos. Apesar do tempo de transferência ser significativo (neste caso é uma operação quase tão cara quanto um *warp*), ela é feita apenas uma vez por quadro. Por este motivo, quando maior o número de iterações de otimização, menor será o impacto da transferência de memória no custo total do processamento do quadro.

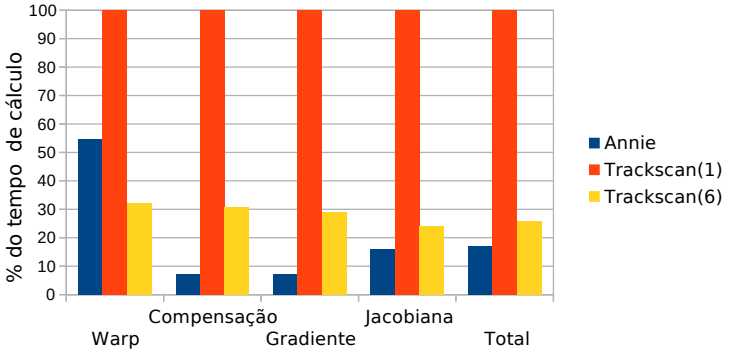
A implementação em GPU mostra-se a mais eficiente das três avaliadas, principalmente em quadros que requerem um número maior de iterações para convergência. Em média, a implementação em GPU é quase 9 vezes mais rápida nestas funções que a implementação em CPU que utiliza uma *thread.* Quando comparada com a versão que utiliza seis *threads,* o *framework* é duas vezes mais lento para realizar o *warp*, mas é três vezes mais rápido nas outras funções.

A implementação atual do *framework* Annie já é mais eficiente que as implementações em CPU com as quais foi comparada. No entanto, o *framework* Annie não utiliza técnicas complexas de otimi-

Tabela 7 – Tempos médio gasto por execução de funções críticas para o rastreamento. A implementação Annie é feita em GPU, enquanto as duas implementações *Trackscan* são executadas em CPU. Apesar de o *Trackscan* ser paralelizado, são apresentados também os tempos quando executado com apenas uma *thread*. A última linha da tabela mostra a soma de todas as funções críticas para cada método. Todos os tempos mostrados estão em milissegundos.

	Implementação		
Função	Annie	*Trackscan(1)*	*Trackscan(6)*
Função *warp*	0.41	0.75	0.24
Compensação de iluminação	0.11	1.525	0.47
Cálculo das imagens gradiente	0.05	0.69	0.20
Cálculo da matriz Jacobiana	1.22	7.59	1.82
Transferência de memória	0.36	-	-
Total das funções críticas	1.79	10.555	2.73

Figura 14 – Gráfico de porcentagens dos tempos apresentados na Tabela 7. Aqui compara-se os tempos da implementação Annie e da *Trackscan(6)* com a implementação *Trackscan(1)*. Apesar de apresentar um desempenho pior que o *Trackscan(6)* na função *warp*, a implementação Annie se mostra mais eficiente quando comparando o tempo total das funções.



zação. É possível que esta abordagem torne-se ainda mais eficiente quando otimizada de maneira mais completa.

7 CONCLUSÃO

Neste trabalho foram apresentados métodos de rastreamento visual direto e métodos de compensação de iluminação globais e locais. Como visto, em cenários reais, diferenças de iluminação podem aumentar o erro fotométrico. Isto faz com que o rastreamento utilizando métodos de compensação de iluminação globais torne-se inviável. Portanto, é necessário empregar métodos de compensação de iluminação locais, que consigam lidar com variações de iluminação que não afetam todos os pixels uniformemente.

Com o objetivo de desenvolver um método de compensação de iluminação local, foram descritos os métodos de rastreamento direto originais no Capítulo 3. A medida de similaridade Soma da Variância Condicional foi escolhida por sua simplicidade e rapidez. Uma vantagem da compensação de iluminação utilizada no SCV é que ela não adiciona parâmetros de iluminação ao método de rastreamento. Assim, apenas os parâmetros de transformação precisam ser estimados.

Esta medida de similaridade foi expandida para tornar-se robusta a mudanças de iluminação locais. Em vez de utilizar apenas um histograma para realizar a compensação sobre toda a imagem, vários histogramas são criados baseados em sub-imagens. Cada pixel é então compensado utilizando uma interpolação dos histogramas. Algumas restrições forem adicionadas aos histogramas para evitar configurações degeneradas de compensação.

Os métodos citados foram organizados em um *framework* de rastreamento direto. Com ele, é possível instanciar um rastreador apenas definindo alguns parâmetros necessários (como tipo de compensação de iluminação, graus de liberdade, etc). Todos os métodos foram implementados utilizando algoritmos que são executados em placas gráficas. Esta implementação é transparente para o usuário do *framework*. As decisões de projeto focaram a facilidade em se utilizar o *framework*, sem que o usuário precisasse utilizar compiladores especiais para o desenvolvimento de novas aplicações.

No Capítulo 6 foram mostrados os resultados de comparações entre os métodos implementados. Com os resultados, ficou clara a vantagem em velocidade do método proposto sobre os outros métodos.

Também foi demonstrada uma queda na precisão quando comparando a imagem de referência com a imagem atual. Porém, esta diferença não faz com que o rastreamento seja pior, como demonstrado no vídeo de resultados.

Artigos desenvolvidos durante a formulação desta dissertação foram avaliados pela comunidade científica e aceitos para publicação, atestando que o método proposto é de interesse para a área de processamento de imagens. Os dois artigos produzidos são descritos abaixo:

- SOUZA, M.; RICHA, R.; PUEL, A.; CAETANO, J.; COMUNELLO, E.; WANGENHEIM, A.V. Robust Visual Tracking for Retinal Mapping in Computer-assisted Slit-lamp Imaging, *IEEE International Symposium on Computer-Based Medical Systems,* 2014

- RICHA, R.; SOUZA, M.; SCANDAROLI, G.; COMUNELLO, E.; WANGENHEIM, A.V. Direct Visual Tracking Under Extreme Illumination Variations Using the Sum of Conditional Variance, *IEEE International Conference on Image Processing,* 2014.

O método proposto de compensação de iluminação local oferece, portanto, uma melhora no tempo de resposta para o rastreamento direto. Este método é especialmente útil para aplicações que necessitem de respostas em tempo real do rastreamento, podendo sacrificar um pouco de precisão para tanto.

7.1 TRABALHOS FUTUROS

Como visto no Capítulo 6, o método LSCV ainda possui alguns pontos fracos. A falha de rastreamento no final da sequência **Pooh-2** foi causada pelas restrições adicionadas aos histogramas locais criados para a compensação. Assim, se torna necessário explorar outras aproximações para os histogramas. A solução utilizada, aproximar os histogramas com funções lineares, é muito simples para a tarefa que enfrenta. É possível que a utilização de funções de maior grau possam fornecer resultados melhores para a aproximação.

Atualmente, todos os pixels da imagem estão sendo utilizados

para a estimação de parâmetros. Alguns estudos apontam a possibilidade de utilizar apenas uma parte dos pixels para a estimação de movimento. Com esta abordagem, poderia ser possível diminuir ainda mais o tempo de cálculo. Para isso seria necessário adaptar todas as funções em GPUs para utilizar uma lista de pixels ativos.

As otimizações feitas para os *kernels* executados na placa gráfica foram essenciais para a velocidade dos algoritmos. Porém, ainda existem algumas otimizações que podem ser feitas. Não existe, atualmente, nenhum controle sobre as transferências de memória entre a memória *host* e a memória do *device.* Estas transferências tomam uma boa parte do tempo de processamento, já que sem os dados da imagem na placa de vídeo é impossível fazer os cálculos do deslocamento do objeto. É possível, por exemplo, executar o *upload* de um novo quadro para a placa gráfica enquanto o quadro anterior está sendo processado pela mesma. As versões mais novas da plataforma CUDA permitem esse paralelismo de execução e transferência de memória. Além de transferências de memória em paralelo, existe a possibilidade de executar mais de um *kernel* em paralelo em uma mesma placa gráfica. Estas otimizações podem melhorar ainda mais a velocidade do algoritmo.

REFERÊNCIAS

BAKER, S.; MATTHEWS, I. Lucas-kanade 20 years on: A unifying framework. **Int. J. Comput. Vision**, v. 56, n. 3, p. 221–255, fev. 2004.

BAKER, S. et al. A database and evaluation methodology for optical flow. **International Journal of Computer Vision**, Springer US, v. 92, n. 1, p. 1–31, 2011.

BARTOLI, A. Groupwise geometric and photometric direct image registration. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v. 30, n. 12, p. 2098–2108, Dec 2008.

BAY, H.; TUYTELAARS, T.; GOOL, L. Surf: Speeded up robust features. In: LEONARDIS, A.; BISCHOF, H.; PINZ, A. (Ed.). **Computer Vision - ECCV 2006**. [S.l.]: Springer Berlin Heidelberg, 2006, (Lecture Notes in Computer Science, v. 3951). p. 404–417.

BENHIMANE, S.; MALIS, E. Real-time image-based tracking of planes using efficient second-order minimization. In: **Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on**. [S.l.: s.n.], 2004. v. 1, p. 943–948 vol.1.

BENHIMANE, S.; MALIS, E. Homography-based 2d visual tracking and servoing. **Int. J. Rob. Res.**, Sage Publications, Inc., v. 26, n. 7, p. 661–676, jul. 2007.

BROWN, J.; CAPSON, D. A framework for 3d model-based visual tracking using a gpu-accelerated particle filter. **Visualization and Computer Graphics, IEEE Transactions on**, v. 18, n. 1, p. 68–80, Jan 2012.

DAME, A.; MARCHAND, E. Accurate real-time tracking using mutual information. In: **ISMAR**. [S.l.]: IEEE, 2010. p. 47–56.

DUVENHAGE, B.; DELPORT, J. P.; VILLIERS, J. de. Implementation of the lucas-kanade image registration algorithm on a

gpu for 3d computational platform stabilisation. In: **Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa**. [S.l.]: ACM, 2010. (AFRIGRAPH '10), p. 83–90.

G. MEILAND M., R. R. S. Improving ncc-based direct visual tracking. **European Conference on Computer Vision (ECCV)**, 2012.

GROUP, K. O. W. et al. The opencl specification. **A. Munshi, Ed**, 2008.

HAGER, G.; BELHUMEUR, P. Efficient region tracking with parametric models of geometry and illumination. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v. 20, n. 10, p. 1025–1039, Oct 1998.

HERMOSILLO, G.; CHEFD'HOTEL, C.; FAUGERAS, O. Variational methods for multimodal image matching. **International Journal of Computer Vision**, Kluwer Academic Publishers, v. 50, n. 3, p. 329–343, 2002.

ISARD, M.; BLAKE, A. Condensation—conditional density propagation for visual tracking. **International Journal of Computer Vision**, Kluwer Academic Publishers, v. 29, n. 1, p. 5–28, 1998.

KATO, H.; BILLINGHURST, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In: **Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on**. [S.l.: s.n.], 1999. p. 85–94.

LOEWKE, K. et al. In vivo micro-image mosaicing. **Biomedical Engineering, IEEE Transactions on**, v. 58, n. 1, p. 159–171, Jan 2011.

LOWE, D. Object recognition from local scale-invariant features. In: **Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on**. [S.l.: s.n.], 1999. v. 2, p. 1150–1157 vol.2.

LUCAS, B. D.; KANADE, T. An iterative image registration technique duvenhage:2010:ili:1811158.1811172,with an application to stereo vision (darpa). In: **Proceedings of the 1981 DARPA Image Understanding Workshop**. [S.l.: s.n.], 1981. p. 121–130.

MARZAT, J.; DUMORTIER, Y.; DUCROT, A. Real-time Dense and Accurate Parallel Optical Flow using CUDA. In: **International Workshop on Computer Vision and Its Application to Image Media Processing**. Tokyo, Japan: [s.n.], 2008.

MEILLAND, M.; COMPORT, A.; RIVES, P. Dense visual mapping of large scale environments for real-time localisation. In: **Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on**. [S.l.: s.n.], 2011. p. 4242–4248.

MUNOZ-SALINAS, R. **ArUco: A minimal library for Augmented Reality applications based on OpenCV**. [S.l.]: Universidad de Córdoba, 2012.

MYNHARDT, J. **Time lapse sunset Cape Town Africa**. 2001. Disponível em: <https://archive.org/details/TimeLapseSunsetCapeTownAfrica>.

MéGRET, R.; AUTHESSERRE, J.-B.; BERTHOUMIEU, Y. The bi-directional framework for unifying parametric image alignment approaches. In: FORSYTH, D.; TORR, P.; ZISSERMAN, A. (Ed.). **Computer Vision – ECCV 2008**. [S.l.]: Springer Berlin Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5304). p. 400–411.

NICKOLLS, J. et al. Scalable parallel programming with CUDA. In: **ACM SIGGRAPH 2008**. [S.l.: s.n.], 2008.

NVIDIA. **Fermi Compute Architecture Whitepaper**. [S.l.], 2011.

REITMAYR, G.; DRUMMOND, T. Going out: robust model-based tracking for outdoor augmented reality. In: **Mixed and Augmented Reality, 2006. ISMAR 2006. IEEE/ACM International Symposium on**. [S.l.: s.n.], 2006. p. 109–118.

RICHA, R. Biblioteca naya para rastreamento direto - acessado em maio de 2014. 2013. Disponível em: <https://sites.google.com/site/rogerioricha/resources>.

RICHA, R.; BO, A. P.; POIGNET, P. Towards robust 3d visual tracking for motion compensation in beating heart surgery. **Medical Image Analysis**, v. 15, n. 3, p. 302 – 315, 2011.

RICHA, R. et al. Fundus image mosaicking for information augmentation in computer-assisted slit-lamp imaging. **Medical Imaging, IEEE Transactions on**, v. 33, n. 6, p. 1304–1312, June 2014.

RICHA, R.; POIGNET, P.; LIU, C. Three-dimensional motion tracking for beating heart surgery using a thin-plate spline deformable model. **Int. J. Rob. Res.**, Sage Publications, Inc., v. 29, n. 2-3, p. 218–230, fev. 2010.

RICHA, R. et al. Visual tracking using the sum of conditional variance. In: **Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on**. [S.l.: s.n.], 2011. p. 2953–2958.

RYOO, S. et al. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In: **Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming**. [S.l.]: ACM, 2008. (PPoPP '08), p. 73–82.

SCANDAROLI, G.; MEILLAND, M.; RICHA, R. Improving ncc-based direct visual tracking. In: FITZGIBBON, A. et al. (Ed.). **Computer Vision – ECCV 2012**. [S.l.]: Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7577). p. 442–455.

SCHOENEMANN, T.; CREMERS, D. A combinatorial solution for model-based image segmentation and real-time tracking. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v. 32, n. 7, p. 1153–1164, July 2010.

SILVEIRA, G.; MALIS, E. Unified direct visual tracking of rigid and deformable surfaces under generic illumination changes in grayscale and color images. **Int. J. Comput. Vision**, v. 89, n. 1, p. 84–105, ago. 2010.

SINHA, S. N. et al. **GPU-based Video Feature Tracking and Matching**. [S.l.], 2006.

STRZODKA, R.; GARBE, C. Real-time motion estimation and visualization on graphics cards. In: **Proceedings IEEE Visualization 2004**. [S.l.: s.n.], 2004. p. 545–552.

VACCHETTI, L.; LEPETIT, V.; FUA, P. Combining edge and texture information for real-time accurate 3d camera tracking. In: **Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality**. [S.l.]: IEEE Computer Society, 2004. (ISMAR '04), p. 48–57.

YILMAZ, A.; JAVED, O.; SHAH, M. Object tracking: A survey. **ACM Comput. Surv.**, ACM, v. 38, n. 4, dez. 2006.

ZHOU, F.; DUH, H. B.-L.; BILLINGHURST, M. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In: **Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality**. [S.l.]: IEEE Computer Society, 2008. (ISMAR '08), p. 193–202.

ZITOVá, B.; FLUSSER, J. Image registration methods: a survey. **Image and Vision Computing**, v. 21, n. 11, p. 977 – 1000, 2003.