Pontifícia Universidade Católica
do Rio de Janeiro

Gustavo Costa Gomes Moreira

Um método para detecção em tempo real de objetos em vídeos de alta definição

Tese de Doutorado

Tese apresentada como requisito parcial para obtenção do título de Doutor pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Prof. Bruno Feijó

Rio de Janeiro
Abril de 2014

Gustavo Costa Gomes Moreira

Um método para detecção em tempo real de objetos em vídeos de alta definição

Tese apresentada como requisito parcial para a obtenção do grau de Doutor pelo Programa de Pós-graduação em Informática do Departamento de Informática do Centro Técnico e Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Bruno Feijó
Orientador
Departamento de Informática - PUC-Rio

Profa. Soraia Raupp Musse
PUC-RS

Prof. Luiz Eduardo Azambuja Sauerbronn
UFRJ

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática - PUC-Rio

Prof. Gilson Alexandre Ostwald Pedro da Costa
Departamento de Engenharia Elétrica - PUC-Rio

Prof. José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico - PUC-Rio

Rio de Janeiro, 08 de abril de 2014

Gustavo Costa Gomes Moreira

Graduou-se no curso de Bacharelado em Informática pela PUC-Rio em 2005. Obteve o Título de Mestre em Informática em 2008 também pela PUC-Rio. Como pesquisador tem interesse nas áreas de Computação Gráfica e Inteligência Artificial. Sua experiência profissional inclui treze anos de consultoria e desenvolvimento de sistemas tanto para empresas públicas quanto privadas. Atualmente é Professor Agregado do Departamento de Engenharia Industrial da PUC-Rio e Analista de Sistemas da Superintendência Administrativa da mesma Universidade.

Ficha Catalográfica

Moreira, Gustavo Costa Gomes

Um método para detecção em tempo real de objetos em vídeos de alta definição / Gustavo Costa Gomes Moreira ; orientador: Bruno Feijó. – 2014.

85 f. : il. (color.) ; 30 cm

Tese (doutorado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2014.

Inclui bibliografia

1. Informática – Teses. 2. Detecção de objetos. 3. Tempo-real. 4. Vídeos de alta definição. 5. Visão computacional. 6. Análise de imagens. I. Feijó, Bruno. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Para minha família,
A vocês todo o meu amor e carinho sempre.

Agradecimentos

Agradeço primeiramente a Deus pelas oportunidades dadas a mim ao longo de minha vida, por meio de pessoas maravilhosas que Ele tem colocado no meu caminho.

Ao meu professor e orientador, Bruno Feijó, por ter me conduzido de forma precisa durante toda a realização desta pesquisa, e também pela amizade construída ao longo dos últimos anos.

Aos professores membros da Banca, que através de seus conselhos e críticas ajudaram a melhorar o resultado final desta tese.

À minha mãe Elza, que sempre me apoiou e incentivou durante minha vida de estudos.

À minha esposa Claudia, por todo o companheirismo e principalmente paciência durante esta longa fase de estudos e dedicação.

Aos meus amigos na SPADM Floriano Mazini e Gustavo Miranda, e também aos amigos do SGU, que sempre me deram todo o apoio necessário para finalizar esta pesquisa com sucesso.

Agradeço também ao CNPq e à PUC-Rio pelos auxílios financeiros concedidos durante o curso.

Resumo

Moreira, Gustavo Costa Gomes; Feijó, Bruno. **Um método para detecção em tempo real de objetos em vídeos de alta definição.** Rio de Janeiro, 2014. 85p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A detecção e o subsequente rastreamento de objetos em sequencias de vídeo é um desafio no que tange o processamento de vídeos em tempo real. Nesta tese propomos um método de detecção em tempo real adequado para o processamento de vídeos de alta definição. Neste método utilizamos um procedimento de segmentação de quadros usando as imagens integrais de frente, o que permite o rápido descarte de várias partes da imagem a cada quadro, desta maneira atingindo uma alta taxa de quadros processados por segundo. Estendemos ainda o algoritmo proposto para que seja possível detectar múltiplos objetos em paralelo. Além disto, através da utilização de uma GPU e técnicas que podem ter seu desempenho aumentado por meio de paralelismo, como o operador *prefix sum*, conseguimos atingir um desempenho ainda melhor do algoritmo, tanto para a detecção do objeto, como na etapa de treinamento de novas classes de objetos.

Palavras-chave

Detecção de Objetos; Tempo-Real; Vídeos de Alta Definição; Visão Computacional; Análise de Imagens.

Abstract

Moreira, Gustavo Costa Gomes; Feijó, Bruno (Advisor). **A method for real-time Object Detection in HD videos.** Rio de Janeiro, 2014. 85p. DSc. Thesis - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The detection and subsequent tracking of objects in video sequences is a challenge in terms of video processing in real time. In this thesis we propose an detection method suitable for processing high-definition video in real-time. In this method we use a segmentation procedure through integral image of the foreground, which allows a very quick disposal of various parts of the image in each frame, thus achieving a high rate of processed frames per second. Further we extend the proposed method to be able to detect multiple objects in parallel. Furthermore, by using a GPU and techniques that can have its performance enhanced through parallelism, as the operator prefix sum, we can achieve an even better performance of the algorithm, both for the detection of the object, as in the training stage of new classes of objects.

Keywords

Object Detection; Real-time; High Definition Videos; Computer Vision; Image Analysis.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

1
Introdução

Rastreamento de objetos em sequências de vídeo é uma questão central em muitas áreas, tais como vigilância, veículos inteligentes, interação humano-computador, aplicações de realidade aumentada, e TV interativa para citar apenas alguns. É um processo que envolve sempre duas etapas: a detecção e o monitoramento. Uma abordagem comum é detectar o objeto no primeiro quadro e depois segui-lo através do resto do vídeo. No entanto, esse tipo de abordagem depende fortemente da informação espacial. Uma abordagem melhor seria buscar uma integração contínua da informação espacial e temporal, isto é: uma abordagem de detecção e rastreamento integrados. Exemplos de abordagens integradas podem ser encontradas em Viola et al. (2003). Outro ponto de vista sobre métodos de rastreamento é a taxonomia encontrada na pesquisa apresentada por Yilmaz et al. (2006) que compreende três classes: Ponto de Rastreamento (objetos detectados em quadros consecutivos são modelados por pontos); *Kernel Tracking* (objetos são modelados por um *kernel*, por exemplo, uma forma retangular ou uma forma elíptica); Rastreamento por Silhueta (objetos são modelados pelo seu contorno ou a região dentro de um contorno). A primeira classe de métodos é adequada para lidar com pequenos objetos. No entanto, este tipo de método requer um mecanismo para detectar os objetos em todos os quadros. A segunda classe é adequada para objetos rígidos, o que é apropriado para aplicações em tempo real. O terceiro é adequado para objetos complexos e não rígidos.

Em particular, os métodos de *Kernel Tracking* utilizam essencialmente dois modelos: um que reúne informações a partir da observação mais recente sobre o objeto (o que pode fazer com que o objeto sendo rastreado possa ser perdido se aparecer em um ponto de vista diferente) (Comaniciu e Meer (2002)), e outro modelo em que diferentes pontos de vista do objeto podem ser aprendidos e em seguida utilizados para rastreamento, como demonstrado em Avidan (2004) e Viola et al. (2003). Este segundo modelo requer treinamento.

A literatura sobre métodos e algoritmos para rastreamento de objetos em tempo real é vasta, porém há poucos trabalhos nesta área quando queremos alto desempenho em vídeos de alta resolução (HD – *High Definition*). No

presente trabalho, nos dedicamos a esta área, motivados pelas demandas da TV digital e de outras aplicações que envolvem alta qualidade de imagem.

1.1. Motivação e Estado da Arte

Nosso trabalho visa conceber um novo método para o rastreamento de objetos que possui um desempenho computacional adequado para aplicações que lidam com vídeos de alta resolução. Estas aplicações podem acontecer desde o uso de câmeras de vigilância com alta resolução até transmissões de programas de TV interativos, onde alguns objetos são rastreados como "*hot spots*" para interação do usuário ou como regiões para composição em tempo real.

Métodos para rastreamento em tempo real de objetos têm sido propostos por vários pesquisadores (Klein et al., 2010)(Klein e Cremers, 2011)(Stalder et al, 2009) e até mesmo bases de comparação (*benchmarks*) têm sido disponibilizadas: BoBoT (Klein 2010), BoBoT-D[1] (Garcia 2012) e PROST (Grasz UT 2010). Entretanto, estes métodos e bases de comparação trabalham com baixa resolução (tipicamente 320x240 a 25*fps*) e atingem taxas de rastreamento em torno de 20 *fps* e, em alguns casos, menores do que 10 *fps*. O foco destas pesquisas está na qualidade do rastreamento sob condições adversas (tais como movimentos 3D, cameras móveis, illuminação pobre, variações bruscas de iluminação, oclusões, *motion blur*, e mudanças de escala e aparência). O presente trabalho não tem o foco na robustez do rastreamento sob condições adversas, mas no alto desempenho do rastreamento em imagens de alta resolução. Até onde vai o conhecimento do presente autor, não há referências a rastreamento em tempo real nestas condições.

Na área de alta qualidade de imagens, algumas soluções têm sido propostas para lidar com o específico problema de *hot spots* em TV interativa, porém com limitações (tanto de qualidade de imagem como em processos envolvidos). Por exemplo, Bove et al. (2000) apresentam um sistema de televisão *hyperlinked* que requer um processo de autoria (testado em plataforma Web). Este mesmo trabalho depois evoluiu para uma patente (Goldpocket Interactive, 2006), onde a informação *hyperlinked* é transmitida com o sinal (*broadcast*). No presente trabalho, estamos interessados em explorar métodos

[1] *Benchmark* contendo dados RGB-D capturados por um sensor MS Kinect.

mais gerais, com aplicações variadas. Novamente, não foram identificados trabalhos com imagens em alta definição.

A razão pela qual é tão difícil alcançar a detecção e o rastreamento em tempo real de objetos é dupla: em primeiro lugar, a existência de mais de um objeto a ser detectado em cada quadro de vídeo compromete o desempenho dos algoritmos; em segundo lugar, o uso de vídeos de alta resolução afeta diretamente o tempo de processamento, porque quanto maior é a resolução, maior será a área a ser pesquisada para cada objeto.

Uma abordagem usual para procurar um objeto é mover uma janela de busca por toda a área da imagem. No presente trabalho, a ideia básica é a seguinte: ao realizar uma segmentação de quadros e utilizar a respectiva imagem integral de frente - imagem calculada sobre a área que não faz parte do cenário estático de um vídeo, ou seja, o "*foreground*" - somos capazes de trazer à tona as regiões da imagem que podem ser descartadas no que tange a detecção de objetos. Como consequência, podemos alcançar altas taxas de quadros, mesmo em vídeos HD.

1.2. Contribuições

Neste trabalho, propomos um método integrado de detecção e rastreamento, em tempo real, em que as características dos objetos são aprendidas por um algoritmo adaptativo de aprendizado. Nosso método baseia-se parcialmente no algoritmo de detecção de objetos proposto por Viola e Jones (2001). Nossa primeira contribuição é a utilização da imagem integral de frente para descartar a análise de várias partes de cada *frame* de um vídeo. No método proposto, este processo de descarte revela uma segmentação adaptativa de cada *frame* definindo uma área reduzida para a detecção de objetos. Nossa segunda contribuição é uma extensão dessas ideias para lidar com vários objetos em paralelo.

Utilizamos também uma placa gráfica para atingir um desempenho ainda melhor na detecção de objetos em vídeos de alta resolução, assim como para melhorar o tempo de treinamento de novas classes de objetos. Nossos resultados mostram um bom desempenho quando lidamos com imagens de alta definição, como as que podem ser encontradas em vídeos de televisão digital.

A Figura 1.1 ilustra o funcionamento do método proposto, exibindo dois *frames* de um vídeo em alta definição. É possível visualizar ao fundo as cenas

segmentadas, utilizadas para o rápido descarte de regiões que não precisam ser processadas.

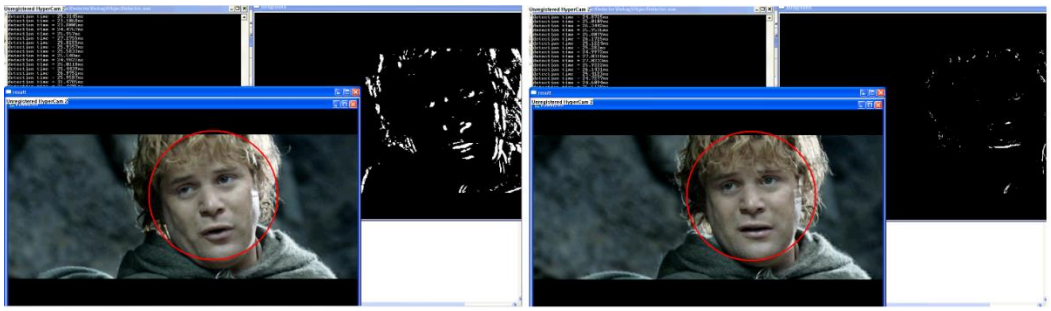


Figura 1.1. Exemplo de rastreamento de objeto em dois quadros de um vídeo de alta definição (New Line Cinema (2003)), usando o nosso algoritmo. Ao fundo estão as cenas segmentadas, utilizadas para o rápido descarte de regiões que não precisam ser processadas. Imagens com direitos autorais reproduzidas no âmbito da política de "*fair use*".

1.3. Estrutura da Tese

A organização dos capítulos desta tese está listada a seguir.

No capítulo 2, são apresentados alguns dos trabalhos mais relevantes na literatura relacionados à questão de rastreamento de objetos. São mostradas as principais características e embasamentos teóricos para cada uma das técnicas, assim como as suas respectivas carências.

O capítulo 3 apresenta o método proposto para detecção de objetos em tempo real, através da aplicação de uma nova heurística para segmentação e descarte de regiões de quadros de um vídeo. Também apresentamos uma forma para paralelização da busca quando for necessária a detecção de vários objetos simultaneamente.

No capítulo 4 apresentamos formas de otimizar o desempenho do método proposto através do poder de paralelismo disponível em uma placa gráfica. Listamos as etapas do método de detecção de objetos que podem ser executadas em uma GPU, assim como as etapas do treinamento de novas classes de objetos que podem ser paralelizadas. Este capítulo também fornece uma breve descrição da arquitetura de uma placa gráfica, levando-se em consideração o framework CUDA.

O capítulo 5 fornece a arquitetura do protótipo desenvolvido ao longo deste trabalho e os resultados obtidos com sua utilização, detalhando cada um dos

módulos desenvolvidos. Apresentamos os experimentos realizados para a detecção e o rastreamento de objetos, assim como o aprendizado de novos objetos, ambos executados tanto na CPU quanto na GPU. São demonstrados também alguns dados comparativos do desempenho do módulo de detecção relacionados a outros trabalhos da área.

Por fim, o capítulo 6 apresenta as conclusões gerais. São listadas as principais contribuições desta tese e analisadas as perspectivas de trabalhos futuros.

2
Trabalhos Relacionados

Para uma seqüência de vídeo com a câmera estática, diversos autores (Gabriel et al. 2003, Porikli e Tuzel 2003, Chen et al. 2007) têm tentado desenvolver um sistema robusto com o intuito de atingir o requisito de detecção e rastreamento em tempo real. Existem também métodos de rastreamento de objetos baseados em filtros de partículas de cor (Lehuger et al. 2006, Nummiaro et al. 2002). Nestes métodos, o modelo alvo do filtro de partículas é definido pela informação de cor do objeto rastreado.

De maneira geral o rastreamento de objetos é computacionalmente muito caro. Neste capítulo apresentamos os principais métodos de detecção e rastreamento existentes na literatura divididos em três abordagens: fluxo ótico, características (*features*) de objetos, e modelos de objetos. Dentro desta última abordagem, damos destaque à solução proposta por Viola e Jones (2001) por ser a precursora da detecção de objetos em vídeos em tempo real, uma vez que o método possui baixo custo computacional em relação aos demais trabalhos. Ao longo de cada subseção, são levantadas de forma geral as vantagens e desvantagens de cada método apresentado, e por fim as carências que ainda não foram supridas.

2.1.
Fluxo Ótico

O Fluxo Ótico é o campo vetorial que descreve como a imagem muda ao longo do tempo, conforme ilustrado pela Figura 2.1. A projeção bidimensional do campo de velocidade observado pela câmera precisa ser estimada. Entretanto, tal tarefa é extremamente difícil de realizar devido a problemas conhecidos, como o problema do "efeito de abertura" (*aperture problem*), que ocorre, por exemplo, quando uma parte de uma barra se move por trás de uma janela, onde apenas é possível detectar o componente do movimento perpendicular à janela, mas a barra pode estar se movimentando em várias direções, sendo impossível detectar a direção real sem visualizar a barra por inteiro, conforme exemplificado na Figura 2.2. Para encontrar o fluxo ótico numa sequência de imagens, autores

tentam usar abordagens baseadas em *features*, ou baseada em gradientes, ou baseada em correlações. A maioria dessas abordagens possui um alto custo computacional, e, portanto é essencial a otimização de tais abordagens.

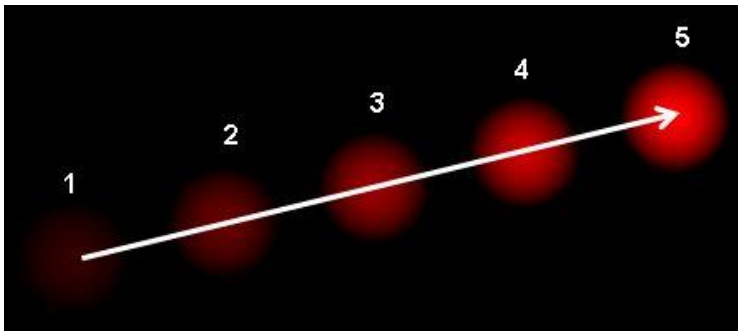


Figura 2.1. A seta em branco indica o deslocamento da esfera verificado entre os cindo quadros analisados. O Fluxo Ótico é dado pelo vetor bidimensional de deslocamento do objeto (reproduzido de Intel (2006b)).

Métodos de fluxo ótico são normalmente utilizados para a geração de campos de fluxo denso computando-se o vetor de fluxo de cada pixel sob a constância de brilho (Schunck et al. 1981). Este cálculo é muitas vezes realizado na vizinhança do pixel algebricamente (Lucas e Kanade 1981) ou geometricamente (Schunck 1986). Estender métodos de fluxo ótico para calcular a translação de uma região retangular é algo pouco viável. Um exemplo foi relatado em Shi e Tomasi (1994), onde foi proposto o bem conhecido rastreador Shi-Tomasi-Kanade (STK) que calcula de forma iterativa a translação de uma região centrada sobre um ponto de interesse. Uma vez que a nova localização do ponto de interesse é obtida, o rastreador STK avalia a qualidade do ponto rastreado pelo cálculo da transformação afim. Este esquema funciona de forma eficaz e rápida na maioria das circunstâncias. Entretanto ainda mais pesquisas são necessárias para se reduzir os pontos incorretos de correspondência.

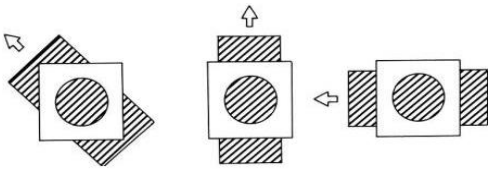


Figura 2.2. Exemplo do problema conhecido como “Problema da Abertura”. Se a estrutura retangular é movida na diagonal, vertical ou horizontal, a imagem será idêntica quando vista através da abertura em todos os três casos. Isso produz ambiguidade quanto à real direção, orientação e velocidade do objeto (reproduzido de Kandel et al. (2000), pag. 554).

2.2. Características de Objetos

Algoritmos de detecção e rastreamento baseados em características de objetos (*features*) foram originalmente desenvolvidos para o acompanhamento de um pequeno número de características marcantes de uma sequência de imagens. Elas envolvem a extração de regiões de interesse (características) nas imagens e, em seguida, a identificação das contrapartes em imagens individuais da sequência. Esta estratégia visa encontrar correspondências muito específicas nos quadros e, portanto, tende a reduzir a perda do rastreamento devido às especificidades únicas das *features* selecionadas; entretanto isto pode não ser verdade na presença de múltiplas similaridades. Algoritmos de rastreamento baseados em *features* típicos são: monitoramento de hipótese múltipla (*Multiple Hypothesis Tracking* - MHT), modelos ocultos de Markov (*Hidden Markov Models* - HMM), rede neural artificial (*Artificial Neural Network* - ANN), filtro de partículas, filtro de Kalman (*Kalman Filtering* - KF) e média de deslocamento (*Mean Shift* - MF).

O algoritmo MHT foi originalmente apresentado por Reid (1979). O conceito por trás do algoritmo é um processo interativo que gera um *feedback* para "ajustar" a correspondência entre dois grupos de *features* até que esta correspondência chegue a um nível ótimo dado um critério particular. Este algoritmo considera vários candidatos para rastreamento e tem a intenção de encontrar o melhor ajuste para os descritores de imagens. Ele funciona bem em rastreamento de múltiplos objetos. No entanto, a técnica clássica de MHT por si só é computacionalmente cara, tanto em termos de tempo quanto de memória (Cox e Hingorani 1996), portanto dificilmente é utilizada em aplicações reais.

Um HMM é normalmente utilizado para extrair a transformação entre duas imagens ou estruturas 3D em movimento, e a partir desta informação, realizar o rastreamento de objetos. No entanto, o método não é determinístico, e dado que o modelo é “oculto”, pode haver de fato mais do que uma possibilidade de transformação que resulte nas posições de cada *feature*. Assim, a sequencia mais provável de transições é procurada. Utilizando algoritmos, como o algoritmo de Baum - Welch e suas modificações (Rabiner 1989), pode-se treinar HMM ajustando os pesos das transições para melhorar o modelo do relacionamento das amostras de treinamento. Abordagens baseadas em HMM não exigem soluções analíticas para certos problemas, sendo eficaz no tratamento de ambientes muito complicadas. No entanto, a fase de treino necessário em HMM

deve ser supervisionada e é difícil aplicar um HMM previamente treinado para aplicações em geral. Da mesma forma, uma ANN também precisa determinar seus pesos através de uma etapa de treinamento, embora a metodologia ANN tenha sido aplicada com sucesso para o rastreamento de objeto (ou movimento), como visto em Barlow (1972) e Ullman (1979).

Comparando filtro de partículas com filtro de Kalman, o primeiro tem um desempenho mais robusto no caso da não “gaussianidade” e não linearidade devido à distribuição simulada posterior. Em um filtro de partículas, é desejável um grande número de partículas para representar a distribuição a posteriori, especialmente nas situações em que as novas medições aparecem na cauda da anterior, ou, se a verossimilhança é fortemente de picos. Para resolver o problema computacional levantado pelo grande número de partículas, por exemplo, MacCormick e Isard (2000) desenvolveram a amostragem particionada, que exige que o estado-espaço possa ser cortado. Sullivan et al. (1999) propuseram a amostragem em camadas utilizando o processamento de várias escalas de imagens. Estas soluções reduzem significativamente os custos computacionais, mas maiores estudos são necessários para uma melhor eficiência.

Em algoritmos de rastreamento de mudança média (*mean shift*), um histograma de cores é usado para descrever a região alvo. A divergência de Kullback - Leibler, o coeficiente de Bhattacharyya e outras medidas de similaridade da teoria da informação são comumente empregados para medir a similaridade entre a região modelo e a região alvo. O rastreamento é realizado encontrando de forma iterativa o mínimo local das funções de medição de distância. Por exemplo, Yang et al. (2005) propuseram uma medida de similaridade simples de computação e mais discriminativo em espaços de *features*. A nova medida de similaridade permite que o algoritmo de mudança média possa rastrear modelos mais gerais de movimento de forma integrada. Para reduzir a complexidade computacional e fazer um modelo de ordem linear, eles empregaram a transformação rápida de Gauss (Greengard e Strain 1991). O estado da arte de média mudança tem sido popularmente aplicado a problemas práticos, como por exemplo, Cheng (1995), Comaniciu e Meer (2002) e Fashing e Tomasi (2005).

2.3. Modelo de Objetos

O rastreamento baseado em modelo de objetos pode ser entendido como um exemplo de rastreamento baseado em *features*. A razão pela qual ele é descrito de forma independente é devido à exigência de agrupamento, de raciocínio e de renderização, que pode diferir do rastreamento baseado em *features*. Além disso, o conhecimento prévio sobre os modelos investigados é normalmente exigido. Por exemplo, no caso de rastreamento de múltiplos objetos a representação binária (modelos) dos objetos deve ser obtida a priori. Isto pode ser seguido pela aplicação de uma etapa de reconhecimento do modelo.

Lowe (1992) usou o detector de bordas Marr-Hildreth para extrair bordas de uma imagem, que foram, então, encadeadas para formar linhas. Estas linhas foram então combinadas e comparadas às do modelo. A transformada de Hough foi utilizada para alcançar uma ideia semelhante em Wunsch e Hirzinger (1997). Sistemas de rastreamento baseados em modelos compartilham os mesmos desafios que os rastreadores baseados em *features*. Por exemplo, a oclusão é uma importante causa de instabilidades, resultando em mau desempenho do rastreamento. O rastreador RAPID (Harris 1993) tratou esta situação através do uso de uma tabela pré-calculada de características visíveis indexadas. Qualquer característica oclusa será relatada devido a sua ausência da tabela. Para acompanhar o movimento humano em atividades como caminhar, correr e saltar, Gavrila e Davis (1996) combinaram bordas na imagem com as de um modelo de aparência, utilizando transformações de distância.

Viola e Jones (2001) apresentam um novo conjunto de soluções que abordam o problema de detecção de objetos e que minimizam o custo de processamento, enquanto atingem altos níveis de detecção e apresentam resultados em tempo real em vídeos de baixa definição. No trabalho destes autores são demonstrados novos algoritmos, representações, e visões bastante genéricas que podem ter aplicabilidade em outras áreas da visão computacional e no processamento de imagens. Devido a isto detalhamos a seguir um pouco mais os pontos principais de seu trabalho.

Viola e Jones (2001) utilizam a representação de um modelo de objeto por meio de uma cascata de classificadores, criada a partir da seleção das principais *features* retangulares do objeto em questão. Tais *features* são selecionadas após a etapa de aprendizado que utiliza o algoritmo *AdaBoost* (Freund e

Schapire 1995). O objetivo de se utilizar uma cascata de classificadores para representar o modelo de objeto é para que o descarte das regiões que não contém o objeto desejado seja executado de forma extremamente rápida. A ideia por trás da cascata de classificadores é ilustrada pela Figura 2.3.

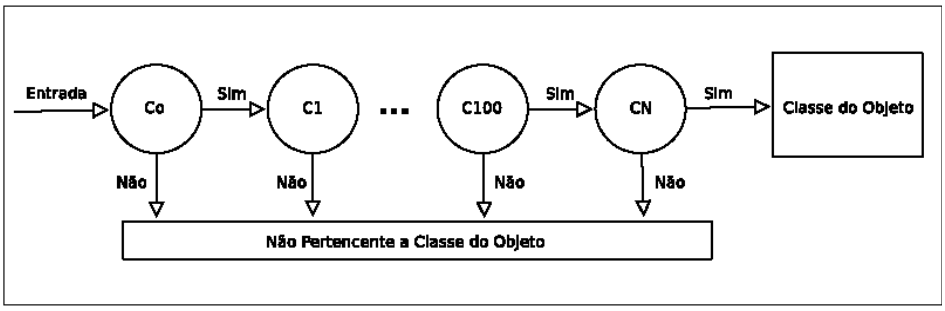


Figura 2.3. Representação de estágios de uma cascata de classificadores. Cada estágio de classificadores (C0, C1,..., CN) tenta descartar ao máximo o número de subjanelas, a fim de diminuir o novo processamento de subáreas da imagem.

Características retangulares, também chamadas de "*Haar Features*" devido a sua semelhança com a definição de "*Haar Wavelets*" (Haar 1910), são extremamente fáceis de serem computadas se for utilizada uma representação intermediária da imagem chamada de **imagem integral**, definida por

$$ii(x,y) = \sum_{x' \le x,\ y' \le y} i(x',y')$$

Nesta forma de representação, apresentada inicialmente por Crow (1984), cada ponto *x,y* da imagem contém o somatório da intensidade dos pixels da origem até a sua localização (Figura 2.4). Desta forma, com uma única passada na imagem, é possível computar a imagem integral. E com estas informações, é possível calcular qualquer soma retangular utilizando apenas quatro referências, conforme ilustrado na Figura 2.5.

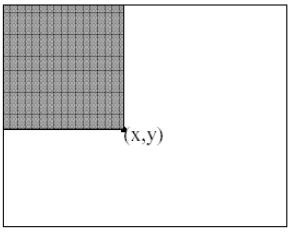


Figura 2.4. O valor da imagem integral no ponto *x,y* é a soma de todos os pixels acima e à esquerda.

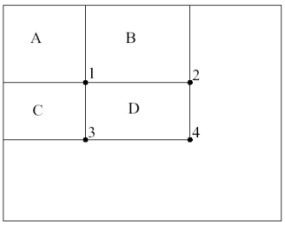


Figura 2.5. É possível calcular o valor da área D fazendo: 4+1-(2+3) (reprodução da Fig. 2 de Viola e Jones (2001)).

Viola e Jones (2001) utilizam apenas três tipos de características retangulares. O valor de uma característica de “dois retângulos” é a diferença entre a soma dos pixels dentro de duas regiões retangulares. Tais regiões têm o mesmo tamanho e são adjacentes. Já o valor de uma característica de “três retângulos” computa a soma de dois retângulos externos a um retângulo central. Por fim, uma característica de “quatro retângulos” computa a diferença entre pares diagonais de retângulos. Estes tipos de características retangulares são ilustrados na Figura 2.6. Com o intuito de melhorar a acurácia da detecção de objetos, Lienhart e Maydt (2002) estenderam a quantidade de tipos possíveis de características retangulares, propondo características novas às previamente sugeridas, mas desta vez rotacionadas em 45 graus. Elas são apresentadas na Figura 2.7.

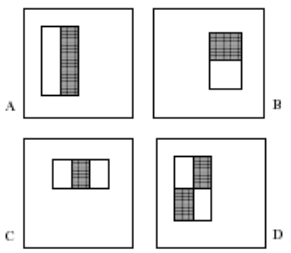


Figura 2.6. (A) e (B) são características de “dois retângulos”. (C) é de “três retângulos” e (D) é de “quatro retângulos”. A soma dos retângulos brancos é subtraída dos retângulos escuros (Extraído de Viola e Jones 2001).

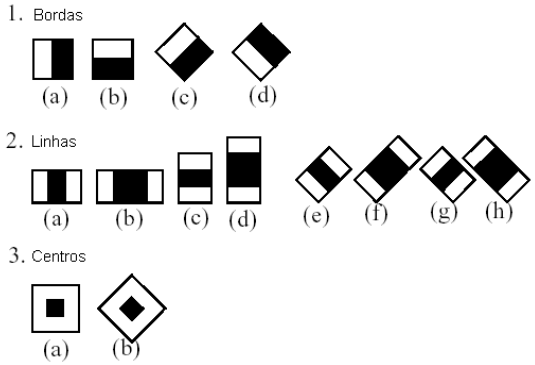


Figura 2.7. Características estendidas (extraído de Lienhart e Maydt 2002)

Um classificador construído sobre um bom conjunto de características passa a avaliar regiões da imagem de forma correta e precisa. No exemplo da Figura 2.8 o classificador percorreria a imagem procurando regiões com os mesmos padrões que as características do objeto desejado.

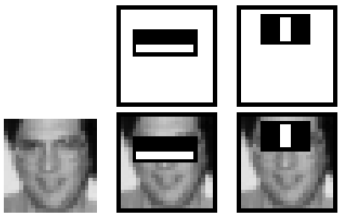


Figura 2.8. Na primeira linha estão exemplos da primeira e segunda característica selecionadas pelo *AdaBoost* para o treinamento do objeto “face humana”. Na linha abaixo elas estão sobrepostas sobre uma face qualquer. Podemos observar que a primeira característica mostra que a região dos olhos é geralmente mais escura que a região das bochechas. Já a segunda característica mostra a diferença entre as intensidades da região dos olhos e do nariz (extraído de Viola e Jones 2001).

Tresadern et al. (2012) propuseram um detector de características faciais em tempo real em dispositivos móveis baseados em imagens integrais. A grande vantagem da solução de Viola e Jones (2001) é que ele é um algoritmo baseado nas características de objetos, ao invés de pixels, o que gera um desempenho muito superior. Viola et al. (2003) apresentam uma extensão do algoritmo de Viola e Jones (2001) para o domínio de movimento. No entanto essa extensão é focada em vídeos de baixa resolução de figuras humanas em condições difíceis, como iluminação. Além disso, a taxa de quadros é muito baixa (cerca de 4 *frames* / segundo).

Uma proposta bastante eficiente inspirada nos trabalhos de Viola e Jones (2001) é apresentada por Stadler et al (2009). Estes autores propõem um método baseado em um classificador múltiplo, no qual detecção (*i.e.* encontrar o objeto de interesse), reconhecimento (*i.e.* distinguir objetos similares na cena) e rastreamento (*i.e.* recuperar o objeto a ser rastreado) são fortemente acoplados. O método de Stadler et al. (*op.cit.*) é uma extensão do rastreamento semi-supervisionado tal como é apresentado em Grabner et al (2008).

Apesar de todo o trabalho inicialmente proposto por Viola e Jones (2001) ser muito eficiente e ter gerado uma série de trabalhos relacionados, ele possui alguns pontos negativos. Por exemplo, ao se utilizar o algoritmo de detecção em vídeos de alta resolução, a taxa de detecção deixa de ocorrer em tempo real. O mesmo problema ocorre com a busca simultânea de vários objetos na cena, pois a taxa de quadros por segundo cai de forma considerável.

2.4. Conclusão

Neste capítulo são analisados os métodos mais estreitamente relacionados com a pesquisa da presente tese. Algumas das técnicas apresentadas possuem um bom resultado para o problema de detecção e rastreamento de objetos em tempo real. Porém, tais técnicas apresentam apenas soluções para problemas muito bem delimitados; ou, então, permitem procurar qualquer tipo de objeto, mas apenas atingem o requisito de tempo real caso sejam utilizados vídeos de baixa resolução.

O trabalho desta tese visa suprir algumas das carências dos trabalhos analisados acima, propondo um novo método de detecção e rastreamento integrados, inspirado na pesquisa de Viola e Jones (2001). Estes autores apresentam uma técnica de detecção de objetos abrangente, sendo capaz de detectar qualquer classe de objeto previamente treinado e com baixo custo computacional, comparado aos demais métodos.

3
Método Proposto

O rastreamento e a detecção de objetos em sequências de vídeos em tempo real é uma questão desafiadora, principalmente em vídeos de alta resolução. A grande quantidade de processamento requerido por diversos algoritmos existentes na literatura acabam gerando uma baixa taxa de quadros por segundo em vídeos HD (High Definition), inviabilizando a detecção em tempo real.

Neste capítulo detalhamos uma abordagem inovadora que permite a detecção em **tempo real**, em vídeos digitais de alta resolução. Para que a detecção do objeto seja possível, é utilizada uma cascata de classificadores construída sobre as características retangulares do objeto em questão, selecionadas por um algoritmo adaptativo de *boosting* conforme descrito em Viola e Jones (2001).

Além disto, também estendemos o método aqui proposto de forma a permitir a busca em paralelo por mais de uma classe de objeto sem a perda de desempenho do algoritmo.

3.1.
Subtração de Fundo em Vídeos

A subtração de fundo é uma abordagem amplamente utilizada para a detecção de objetos em movimento em vídeos de câmeras estáticas. Esta abordagem de detecção de objetos em movimento baseia-se na diferença entre o quadro atual e um quadro de referência, muitas vezes chamado de “imagem de fundo”, ou “modelo de fundo”.

A imagem de fundo deve ser uma representação da cena sem objetos em movimento e deve ser mantida atualizada regularmente, de modo a adaptar-se às condições de iluminação diferentes e configurações de geometria. Uma resenha das técnicas de subtração de fundo pode ser encontrada em McIvor (2000) e em Piccardi (2004). A Figura 3.1 ilustra alguns exemplos de métodos de segmentação de vídeo.

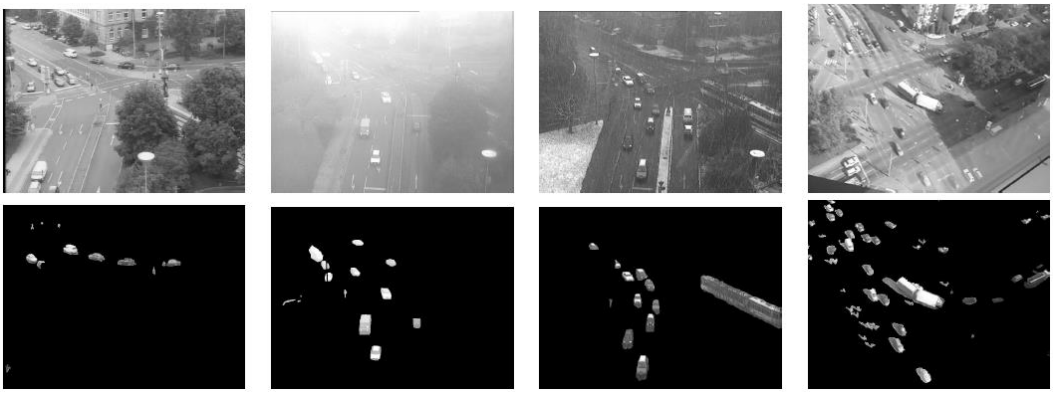


Figura 3.1. Exemplos de Subtração de Fundo por métodos básicos. (extraído de Cheung e Kamath 2004)

Um método baseado na técnica de segmentação de fundo compreende três etapas, sendo elas: treinamento, classificação e atualização. A etapa de treinamento, que não é obrigatória, consiste em coletar informações de um conjunto de N imagens sequenciais para fazer uma estimativa do modelo inicial de fundo. Nos casos em que essa etapa é inexistente, o modelo é calculado nas etapas seguintes de forma dinâmica.

Para definir se um pixel x pertence a um objeto em movimento ou ao plano de fundo, é necessário uma etapa de classificação. Essa etapa utiliza um modelo de fundo calculado previamente.

Por último, a etapa de atualização é a responsável por manter o modelo de fundo coerente, tratando mudanças como iluminação ou geometria que possam ocorrer no cenário.

É importante levar em conta também o desempenho em termos computacionais de cada método de Subtração de Fundo. Dentre a maioria das aplicações há a necessidade de que a subtração seja executada em tempo real, como é o objetivo do tema desta tese. Portanto, é necessário avaliar a quantidade de memória utilizada, e a quantidade de processamento necessária, que impacta diretamente no valor de quadros por segundo (*fps*) da aplicação que utiliza o método.

Dentre os diversos métodos na literatura que tratam sobre a Subtração de Fundo, podemos dividi-los basicamente em duas abordagens: a estatística e a não estatística (Hassanpour et al. 2011). Todas as duas formas buscam um maior custo-benefício entre qualidade da segmentação e o tempo de processamento.

Os métodos que seguem a abordagem não estatística foram os primeiros que surgiram na literatura. O desenvolvimento de tais métodos é relativamente

simples, possuindo baixo custo computacional, e o resultado obtido é bom para casos em que o fundo é relativamente estático e a variação de luz é mais suave. Tais métodos são conhecidos como “Subtração Básica de Fundo” (Hall et al. 2005).

Dentre os diversos métodos para Subtração de Fundo existentes na literatura, propomos, nesta tese, a utilização do método básico que usa um filtro recursivo definido pela equação (3-1):

$$B_{i+1}(x,y) = \begin{cases} I_i(x,y) & se \quad i = 1 \\ (1-\alpha)B_i(x,y) + \alpha I_i(x,y) & se \quad i > 1 \end{cases} \tag{3-1}$$

onde a imagem de fundo B_i é atualizada de maneira acumulativa, adaptativa, integrando a informação do frame que chega I_i na imagem corrente de fundo e α é um coeficiente de adaptação. Filtros recursivos de primeira ordem têm sido utilizados em várias técnicas nos últimos anos (Heikkilä e Silvén, 1999) (Wren et al., 1997). Por exemplo, Wren et al. (1997) utiliza este tipo de filtro para atualizar a média da representação estatística proposta.

Nos métodos básicos de Subtração de Fundo, o *background* é identificado pela seguinte equação:

$$\left| I_i(x,y) - B_i(x,y) \right| > Limiar \tag{3-2}$$

Desta maneira, há dois parâmetros de ajuste que interferem no resultado: *Limiar* (*threshold*) e α. Quando o valor de α é próximo de zero, o fundo B_i irá se adaptar mais lentamente às alterações no cenário. Se α for próximo de um, B_i irá se adaptar rapidamente conforme a cena mude e os objetos se movam.

O método básico de subtração de fundo definido pelas equações (3-1) e (3-2) é muito adequado para a segmentação de vídeos em tempo real, pois é fácil e rápido de ser calculado e também é flexível a mudanças de iluminação e no próprio cenário, onde o modelo de fundo é calculado para cada *frame* do vídeo. Uma desvantagem deste método é encontrar a taxa de adaptação *α* mais adequada à sequência de vídeo, pois como citado anteriormente, caso *α* seja muito perto de zero, o fundo irá se adaptar muito lentamente às alterações no cenário, ou seja, o cenário pode ser alterado e o modelo do fundo pode não corresponder mais a este cenário, podendo-se detectar falsas mudanças entre os *frames* durante longos períodos. Outro problema deste método se dá quando *α* é muito próximo de um, onde o fundo adapta-se mais rapidamente às alterações no cenário e essa velocidade na adaptação pode fazer com que

partes dos alvos sejam perdidas por serem rapidamente consideradas como integrantes do modelo de fundo. Ademais, este método não tem problema de memória, pois não é necessário armazenar informações dos *frames* ao longo do tempo.

3.2. Redução da Área de Busca na Imagem

O algoritmo proposto originalmente por Viola e Jones (2001) para a detecção de objetos em imagens, requer que uma janela de busca, onde seu tamanho inicial é o tamanho mínimo do objeto na imagem, percorra toda a área da imagem verificando se o objeto procurado se encontra na área atual da janela de busca. Sempre que esta janela de busca termina de percorrer a imagem, ela é redimensionada por um fator σ e a busca em toda a imagem é então reiniciada.

Podemos notar com isto que toda a área de imagem é sempre testada para verificar se contém o objeto desejado. No entanto, é conhecido que as alterações entre quadros consecutivos ocorrem apenas em algumas pequenas regiões (exceto quando há uma mudança de cenário ou uma súbita mudança de iluminação, por exemplo). Neste caso, podemos minimizar a área de pesquisa, verificando apenas as regiões que sofreram mudanças entre dois quadros consecutivos.

Esta seção propõe uma técnica para atingir o objetivo de reduzir de forma substancial a área a ser processada em cada quadro de um vídeo, desta maneira tornando a análise de cada *frame* do vídeo rápida e eficiente. Para alcançarmos a meta propomos a combinação de algumas técnicas, entre elas a segmentação de vídeo, a utilização da Imagem Integral, e uma heurística que descarte regiões do *frame* sem a necessidade de um processamento complexo. A Figura 3.2 ilustra de forma geral os passos do método proposto.

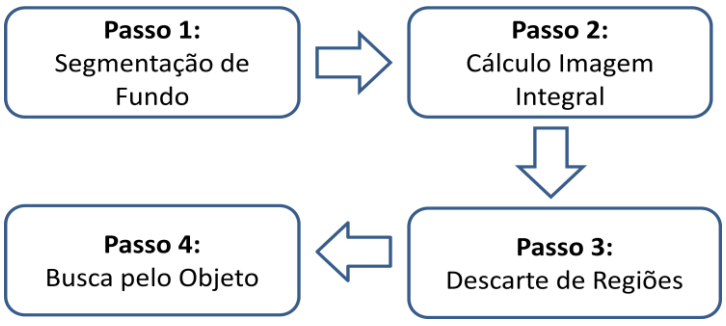
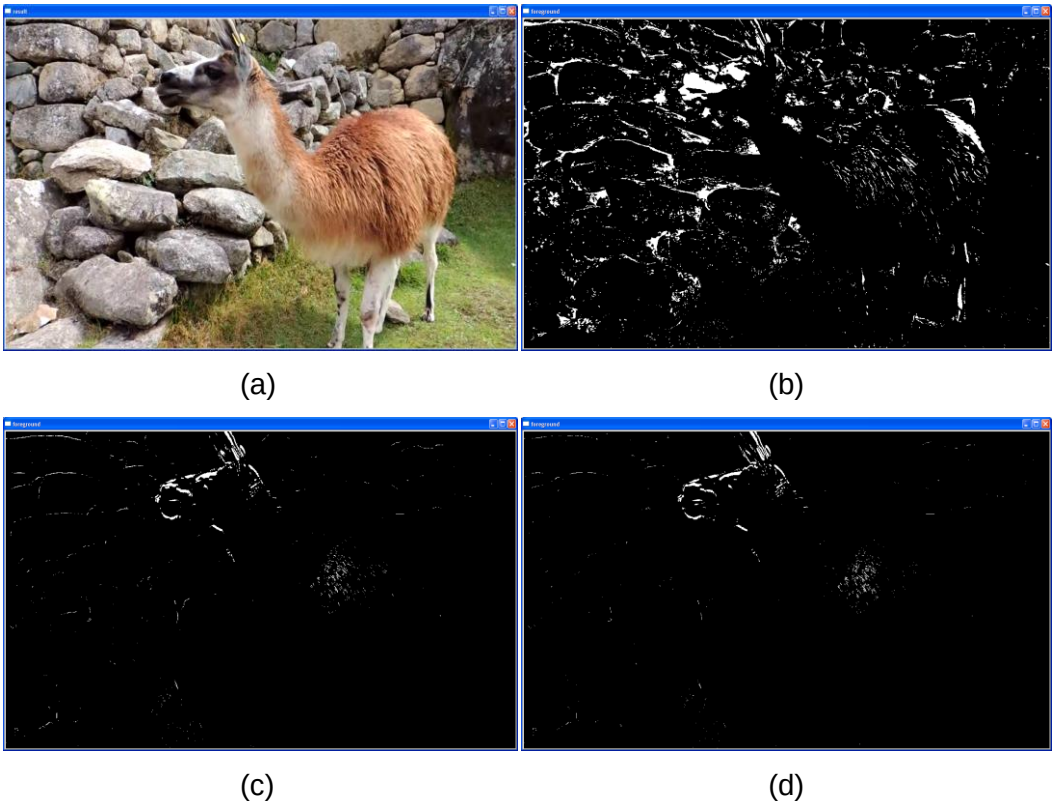


Figura 3.2. Visão geral do método proposto

Através da segmentação do fundo de um vídeo, é possível detectar as regiões do vídeo que sofreram alterações ao longo de seus *frames*. As regiões que sofreram alguma mudança entre os quadros estarão com pixels brancos, e as regiões com cor preta são aquelas que não sofreram alterações ou foram incorporados ao fundo do cenário, de acordo com o modelo de fundo adotado, no caso deste trabalho, o modelo definido pela equação (3-1). Podemos observar que quanto mais próximo de zero for α, o fundo se adaptará mais rapidamente às alterações da cena, e com isto a quantidade de pixels brancos da imagem será menor. Caso contrário, se o fundo se adaptar de forma mais lenta conforme α se aproxima de um, aumentará a quantidade de pixels pretos, ou os pixels que fazem parte do plano de fundo. Este comportamento é ilustrado na Figura 3.3.



(a) (b)

(c) (d)

Figura 3.3. Exemplo de extração de *foreground* sobre um instante *t* de um vídeo (a) utilizando diferentes valores de α, 0.03 (b), 0.7 (c) e 0.9 (d).

Após o plano de fundo ser extraído, a janela de busca do objeto pode ignorar regiões que não tenham alteração no plano de frente (*foreground)*, ou seja, áreas onde todos os pixels foram incorporados ao cenário e são da cor

preta. Para que este teste possa ser feito rapidamente, precisamos inicialmente calcular a imagem integral do *foreground*.

Mediante o teste de se a janela está sobre uma área em que houve uma mudança entre quadros, podemos apenas verificar se a área da sua respectiva imagem integral é maior do que zero. Se assim for, a janela está sobre uma área que pode ou não conter o objeto procurado. Caso contrário, a área pode ser descartada da análise, pois se nela havia um objeto e nenhuma mudança ocorreu nesta região do quadro, o objeto foi encontrado pelo método em algum dos quadros anteriores e esta região deve permanecer marcada ao longo dos quadros subsequentes.

Segmentando o fundo do vídeo e calculando as imagens integrais das partes segmentadas, obtemos um ganho significativo de desempenho. Para conseguir um rápido descarte de áreas da imagem, propomos uma "busca binária" em cada quadro, dividindo a imagem do *foreground* de forma recursiva em duas partes e analisando a sua imagem integral.

Se a imagem integral da janela de busca for igual a zero, toda a área pode ser descartada, uma vez que não há nenhuma mudança entre o quadro anterior e o atual. Se a imagem integral da janela de pesquisa atual for maior do que zero, o *foreground* deve ser dividido e analisado novamente até que seu tamanho seja menor ou igual do que o tamanho mínimo do objeto sendo procurado.

Termos a Imagem Integral previamente calculada para o *foreground* se mostra extremamente eficiente para o rápido descarte de áreas, visto que se têm previamente calculadas todas as possíveis regiões retangulares da imagem, e com isto, podemos descartar regiões de quaisquer tamanhos que fazem parte do *foreground* da cena em **tempo constante**, bastando-se obter o valor dos quatro pixels que formam os vértices da área na Imagem Integral.

No exemplo ilustrado pela Figura 3.4, são analisadas duas regiões do *foreground* de um determinado instante *t* de um vídeo, formadas respectivamente pelos vértices A, B, C e D, e pelos vértices E, F, G e H. Ao analisar a primeira região da imagem formada pelos vértices A, B, C e D, é possível calcular o valor de sua área aplicando a equação $D + A – (B + C)$ sobre a Imagem Integral do *foreground*. Podemos observar que neste caso vamos encontrar o valor da área igual à zero, pois a área formada por tais vértices não possui nenhuma alteração de movimento em relação ao quadro anterior, ou seja, esta região foi incorporada ao cenário estático pelo modelo de fundo utilizado. A área da segunda região é calculada pela equação $H + E – (F + G)$, onde teremos

um valor superior à zero, visto que a área possui alterações de cena com relação ao quadro anterior. Portanto esta região é uma candidata em potencial de possuir o objeto procurado e deve ser processada pelo algoritmo de detecção.

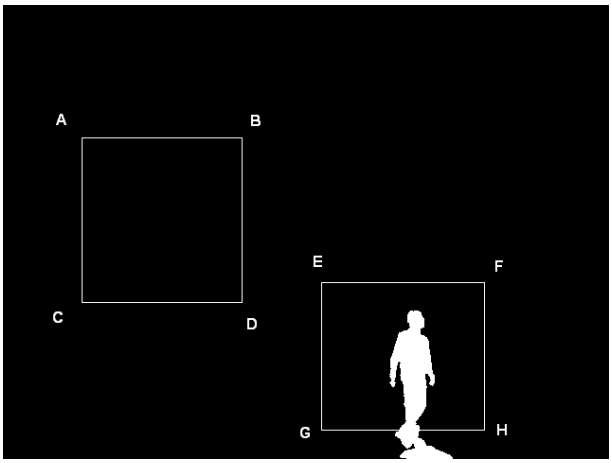


Figura 3.4. Duas regiões retangulares sobre o *foreground* de um instante t do vídeo. O retângulo formado pelos pontos *A*, *B*, *C* e *D* não precisam ser analisados uma vez que a área de sua imagem integral é zero.

Demonstrada a maneira como regiões de um *frame* podem ser rapidamente descartadas, o próximo passo é definir um método para que a quantidade de regiões que a Janela de Busca percorre e analisa seja drasticamente reduzida em cada quadro. Para que este descarte possa ser feito de forma rápida, definimos primeiramente uma máscara binária que contém os pixels que fazem parte do *foreground*, e em seguida calculamos a sua respectiva Imagem Integral. Utilizando o método de Dividir para Conquistar, propomos uma "busca binária" em cada quadro, sempre dividindo a imagem do *foreground* em duas partes, gerando duas sub-regiões e analisando cada sub-região de acordo com a Imagem Integral do plano de frente. Se a área da Imagem Integral de uma sub-região for igual a zero, toda esta região pode ser descartada, uma vez que não há nenhuma mudança entre o quadro anterior e o atual. Se a área da imagem integral da sub-região analisada for maior do que zero, ela deve ser novamente dividida, de forma **recursiva**, até que seu tamanho seja menor ou igual ao tamanho mínimo especificado para o objeto procurado, sendo este o **caso base** da recursão. Ao retornar da recursão, a sub-região deve ser analisada para verificar se ela contém ou não o objeto procurado. É importante verificar também, no retorno da recursão, que a área de limite entre duas sub-regiões pode conter o objeto procurado, pois ao se dividir uma região em outras

duas partes menores, o objeto ser cortado “ao meio”. Porém tal verificação precisa ser feita apenas se ambas as sub-regiões possuírem alterações na cena, caso contrário podemos concluir que o objeto não foi “cortado” pelas duas sub-regiões. O pseudo-algoritmo para o descarte de áreas e detecção de objetos é descrito no Algoritmo 3.1.

A Figura 3.5 ilustra alguns passos do método proposto. Nota-se que a primeira divisão feita no quadro, gerando duas sub-regiões, descarta 50% de toda a imagem em **tempo constante**, uma vez que a área da respectiva Imagem Integral da sub-região à esquerda possui valor igual a zero. A Figura 3.5-c ilustra a nova divisão realizada na imagem, que acabe gerando duas novas sub-regiões que possuem valor de área da Imagem Integral superior a zero. Neste caso, ambas as sub-regiões devem ser novamente divididas. A Figura 3.5-d apresenta a geração de todas as sub-regiões verticais e horizontais, até o momento que o caso base é encontrado, onde o tamanho de uma nova sub-região seria menor ou igual ao tamanho mínimo do objeto procurado. A linha pontilhada representa a análise que é efetivamente realizada pelo algoritmo ao se retornar das recursões. Através da análise da junção das áreas limítrofes das sub-regiões é que o objeto procurado é detectado, tratando-se neste exemplo de um pedestre.

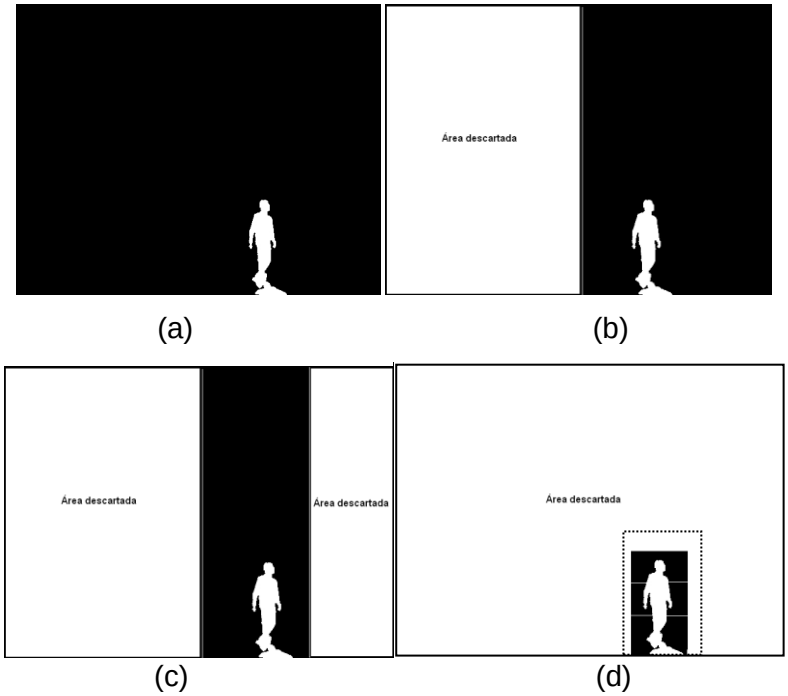


Figura 3.5. Heurística para o descarte de áreas sem necessidade de processamento. (a) imagem inicial, (b) primeira divisão vertical da imagem, (c) segunda divisão vertical da imagem, (d) todas as divisões horizontais e verticais feitas.

Algoritmo 3.1. Rastreamento e Detecção de Objeto com Descarte de Regiões da Imagem segmentada

```
1:  Dado F a imagem segmentada
2:  Dado J, a janela de busca (inicializada com o tamanho mínimo do objeto)
3:  Dado I, a imagem original
4:  Dado λ, o fator de escala
5:  Dado Δ, o deslocamento da janela
6:  Dado tam_min_objeto, o tamanho mínimo do objeto procurado
7:  Chamar função Descartar_Areas_Imagem (F)
8:  Função Descartar_Areas_Imagem (F')
9:      Se tamanho(F') <= tam_min_objeto então
10:       Retornar tam_min_objeto
11:     Fim se
12:     Dividir F' em duas partes, F1 e F2, alternando a forma de divisão, vertical ou horizontal, a
        cada chamada desta função
13:     Calcular II1, a área da imagem integral de F1
14:     Calcular II2, a área da imagem integral de F2
15:     Se II1 > 0 então
16:       tam_ret = Descartar_Areas_Imagem(F1)
17:       Chamar Detectar_Objeto(F1, tam_ret)
18:     Fim se
19:     Se II2 > 0 então
20:       tam_ret = Descartar_Areas_Imagem(F2)
21:       Chamar Detectar_Objeto(F2, tam_ret)
22:     Fim se
23:     Se II1 > 0 E II2 > 0 então
24:       Chamar Detectar_Objeto(F', tamanho(F1))
25:     Fim se
26:     Retornar tamanho(F')
27: Fim função
28: Função Detectar_Objeto(F, tam_min_janela)
29:     Escalonar janela de busca J até que ela caiba dentro de tam_min_janela
30:     Enquanto tamanho(J) < tamanho(F) faça
31:         x=0; y=0;
32:         Enquanto y + Δ < altura(I) faça
33:             Enquanto x + Δ < largura(I) faça
34:                 Se localização de J na imagem original contém o objeto procurado então
35:                     Armazenar a posição corrente de (x,y) da janela de busca
36:                 Fim se
37:                 Incrementar o valor de x
38:             Fim enquanto
39:             Incrementar o valor de y
40:         Fim enquanto
41:         Escalonar o tamanho de J por λ
```

42: **Fim enquanto**
43: Marcar na imagem original as localizações detectadas
44: **Fim função**

3.3. Análise de Complexidade do Algoritmo de Descarte de Regiões

A quantidade de trabalho requerido por um algoritmo, na sua execução, não pode ser descrita simplesmente por um número, pois a quantidade de operações básicas efetuadas, em geral, não é o mesmo para qualquer entrada, dependendo usualmente do tamanho da entrada. No nosso caso, o número de operações efetuadas pelo método de descarte de regiões depende basicamente do cálculo da imagem integral do plano de frente, que servirá de base para orientar como o *frame* será dividido em dois recursivamente até que seja atingido o tamanho mínimo da janela de busca *J*.

É trivial perceber que no melhor caso, todo o *foreground* em um momento *t* do vídeo terá a área de sua imagem integral igual a zero e, portanto, todo o respectivo *frame* poderá ser descartado em tempo constante. Desta forma temos a complexidade do algoritmo no melhor caso igual a $f(n) = \Omega(1)$.

No pior caso, toda divisão realizada sobre o *foreground* terá o cálculo da respectiva área da imagem integral maior que zero, e por isto a região deverá ser analisada. Este caso ocorre, por exemplo, numa mudança brusca de cena ou de iluminação, onde o plano de frente será formado por toda a área do *frame*. Com isto, podemos notar que a complexidade no pior caso será ter de analisar todas as *n* subdivisões, concluindo-se que $f(n) = O(n)$.

Para o cálculo da complexidade do caso médio, deveríamos obter a média (*i.e.* a expectativa estatística) dos tempos de execução de todas as entradas de tamanho *N*. Temos a intuição de que o caso médio deve ser $O(n \log n)$, porém não completamos esta análise no presente trabalho.

3.4. Paralelização de Rastreamento de Objetos

Partindo do princípio que é desejável podermos detectar vários objetos ao longo de um vídeo, ao invés de um único objeto, é preciso aperfeiçoar a quantidade de vezes e a maneira como esses objetos são monitorados em cada quadro. O algoritmo proposto por Viola e Jones (2001) tem uma solução para a

busca de objetos, mas de forma individual. Nele, para cada objeto de interesse, é necessário reconstruir as estruturas dos classificadores em cascata, recalcular a imagem integral e, finalmente, reiniciar a busca pelo objeto no quadro, marcando cada região que possui o objeto. Propomos aqui uma solução relativamente simples, porém eficiente, que pode ser utilizada em vídeos digitais.

O algoritmo pode ser dividido em partes que podem ser executadas em paralelo, cada uma lidando com uma instância de objeto na imagem. São elas:

- A inicialização da estrutura da cascata de classificadores;
- Verificação da existência de um objeto em determinada região da imagem;
- Marcação das regiões da imagem onde foram identificados os objetos.

Uma vez identificados estes passos, podemos modificar o Algoritmo 3.1 para que ele seja executado de forma paralela. Assim, com uma única análise nas regiões da imagem podem ser detectados todos os objetos desejados de forma simultânea, ganhando-se em velocidade proporcionalmente à quantidade de objetos pesquisados no vídeo. Para tanto é preciso que haja um ambiente computacional que permita a paralelização real do algoritmo, seja com mais de um processador ou com múltiplos núcleos de processamento.

A primeira etapa realizada no algoritmo é a inicialização da cascata de classificadores do objeto em questão, por meio da leitura das informações de um arquivo XML para uma estrutura em árvore na memória. Este passo pode ser paralelizado com o uso de *threads* para cada um dos objetos que se deseja detectar, pois cada objeto possui a sua própria cascata de classificadores.

Enquanto no algoritmo proposto por Viola e Jones (2001) é necessário que para cada objeto procurado a imagem seja sempre totalmente percorrida, propomos que cada região seja analisada de forma paralela para cada uma das n cascatas de classificadores. Desta maneira a quantidade de objetos que pode ser processada fica diretamente relacionada ao número de processos/*threads* que podem ser executados paralelamente.

Finalmente, todas as regiões que possuem os objetos procurados devem ser marcadas na imagem original para que o usuário saiba onde foi identificado cada um dos objetos. Para isto, após o processamento de todas as áreas da imagem uma estrutura de dados pode ser armazenada em memória representando todas as regiões que possuem objetos. Para percorrer esta estrutura e realizar as marcações na imagem original, o algoritmo pode ser

executado de forma paralela dividindo-se o tamanho desta estrutura auxiliar pelo número de *threads* existentes e processando cada parte desta estrutura de forma independente entre si. A Figura 3.6 ilustra de maneira geral o método estendido com os trechos que são executados de forma paralela com o objetivo de detectar mais de um objeto simultaneamente.

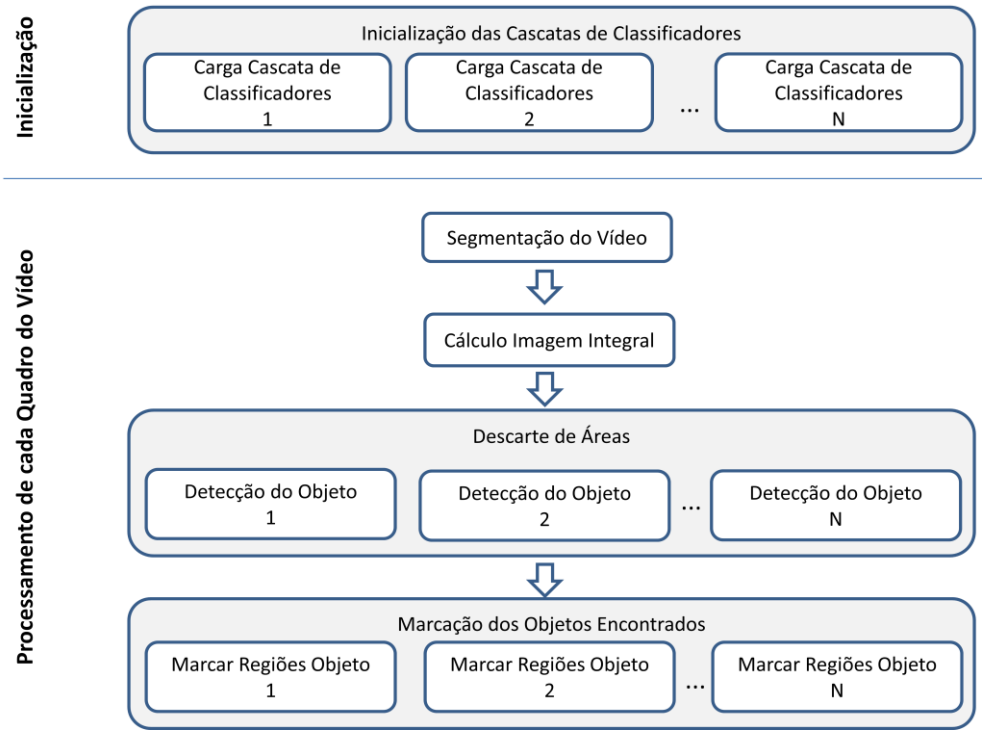


Figura 3.6. Visão geral do método de rastreamento e detecção de objetos utilizando paralelismo num ambiente com N *threads* disponíveis para execução.

3.5. Conclusão

Neste capítulo apresentamos um novo algoritmo de detecção e rastreamento de objetos integrados, capaz de encontrar e rastrear objetos em tempo real durante a execução de vídeos digitais de alta resolução. Enquanto diversos trabalhos relacionados na literatura possuem foco apenas no aumento da velocidade de detecção de objetos em imagens estáticas, apresentamos neste capítulo uma solução genérica que aprimora o problema de desempenho inerente à detecção em vídeos digitais em tempo real.

Em primeiro lugar, propomos a utilização de um método de segmentação de fundo de baixo custo computacional para que o esforço de processamento

seja colocado apenas em regiões do vídeo onde tenham acontecido alterações significativas entre quadros consecutivos.

Em seguida, apresentamos a ideia de utilizar a imagem integral do plano de frente segmentado para que seja possível descartar de forma extremamente rápida várias regiões de *frames* de um vídeo. Podemos então notar que este processo de descarte revela uma segmentação adaptativa de quadros que acaba por definir uma área reduzida para a detecção de objetos.

Portanto, o método proposto tem o potencial de evitar grandes quedas na taxa de quadros por segundo (*fps*) ao se utilizar vídeos de alta resolução, e também ao se procurar por mais de um objeto simultaneamente.

4
Otimizações do método através da GPU

Com o advento e a popularização das chamadas GPUs (*Graphics Processing Units*), percebeu-se que era possível não só utilizar essas placas para o processamento de imagens, mas também programá-las para processar diversos outros cálculos que até então apenas uma CPU fazia. Esta forma de processamento nas placas gráficas é conhecida como GPGPU (*General Purpose Graphics Processing Unit*).

Neste capítulo apresentamos como otimizar o método proposto anteriormente de detecção e rastreamento de objetos através da programação em GPU. Além disto, propomos também a utilização da GPU para realizar o treinamento de novas classes de objetos, tirando proveito do poder de paralelismo inerente às placas gráficas.

4.1.
Arquitetura Básica de uma GPU

Enquanto CPUs dedicam uma grande quantidade de seus circuitos ao controle, uma GPU possui o foco em ALUs (*Arithmetic Logic Units*), o que as torna bem mais eficientes em termos de custo quando rodam um algoritmo paralelo (NVIDIA 2014), conforme ilustrado pela Figura 4.1. A partir do final dos anos 90, as GPUs passaram de um mero *pipeline* de função fixa para geração de imagens para um *pipeline* altamente programável. Essa possibilidade de programar na GPU atraiu pessoas interessadas em executar programas altamente paralelos e de propósito gerais.

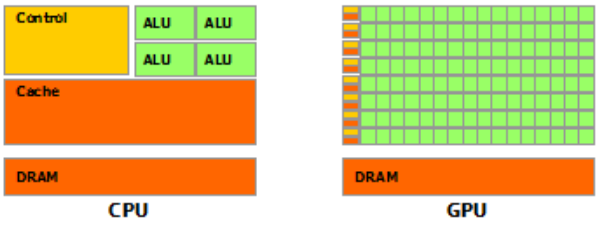
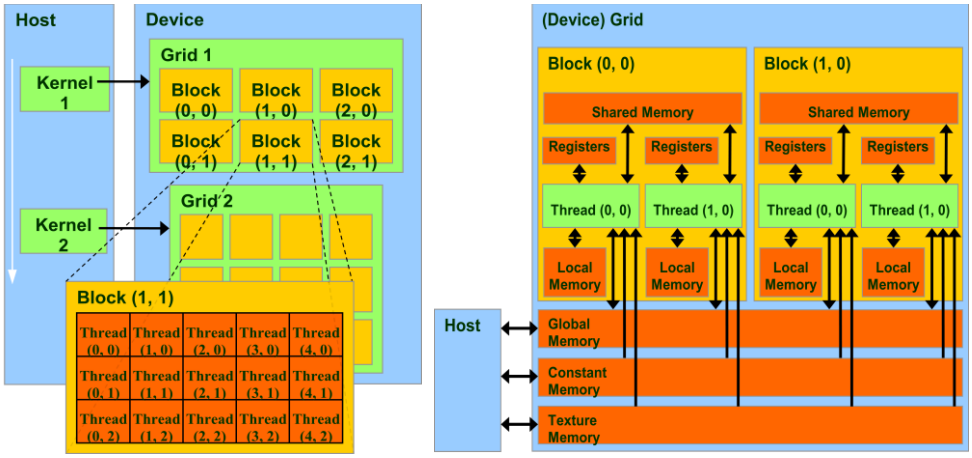


Figura 4.1. Como uma GPU é especializada para o uso intensivo de computação paralela, ela possui mais transistores do que uma CPU para desempenhar tal tarefa (figura extraída de NVIDIA (2013)).

GPUs possuem multiprocessadores com vários núcleos que aplicam a estrutura de uma única instrução para vários dados (SIMD - *Single Instruction, Multiple Data*). O uso de SIMD melhora o desempenho por unidade de custo, permitindo que vários dados sejam processados de forma igual em paralelo simultaneamente. Para melhorar ainda mais o desempenho, GPUs atuais utilizam uma extensão desse conceito que é única instrução e múltiplas *threads* (SIMT - *Single Instruction, Multiple Threads*), o que significa que a mesma instrução é executada em várias *threads* diferentes, ajudando a manter o *pipeline* ocupado. Esse agrupamento de *threads* que executam a mesma instrução recebe o nome de *warp*.

Como cada multiprocessador executa várias *threads*, eles possuem uma memória interna compartilhada pequena e um grande banco de registradores dos quais cada *thread* é dona de alguns. Isso aumenta muito a velocidade da troca de informações entre *threads* de um agrupamento e da troca de contexto, já que não é preciso salvar os dados dos registradores. Mas o fato de a memória local ser compartilhada faz com que possam ocorrer conflitos. Isto ocorre quando mais *threads* do que o possível tentam ler ou escrever da memória que possui um número de "portas" menor do que o número de *threads* que podem ser executadas. O modelo de processamento através de blocos de *threads* e a organização da memória de uma GPU são ilustrados pela Figura 4.2.



(a) (b)

Figura 4.2. Modelo esquemático da divisão da GPU em blocos de *threads* para a execução de *kernels* (a) e organização da memória na GPU com CUDA (b) (figura adaptada de NVIDIA (2013)).

Em um modelo simples de multiprocessador, a técnica SIMT geraria problemas em instruções de desvio condicionais. Para resolver esse problema, o hardware possui um controlador que é capaz de mascarar resultados de instruções executadas em *threads*, permitindo que uma *warp* continue a execução mesmo se houverem resultados divergentes para um desvio condicional. Para isso, basta que os resultados das *threads* que não teriam tomado o caminho sequencial não gerem resultados permanentes, seja gravando em registradores ou na memória. Depois, o trecho de código correspondente ao desvio tomado é executado, impedindo que as outras *threads* gerem resultados. Isso faz com que algum tempo de processamento seja perdido, o que levou à criação de técnicas em software para melhorar o desempenho. Propostas para melhora do desempenho envolvem também instruções condicionais, o que eliminaria vários desvios condicionais, e *warps* dinâmicas, que seriam capazes de se dividir quando suas *threads* tivessem resultados divergentes em desvios condicionais, o que eliminaria a necessidade de mascarar seus resultados.

A NVIDIA criou uma linguagem proprietária chamada CUDA (NVIDIA 2014), baseada nas linguagens C/C++. Por utilizar linguagem semelhante a linguagens já familiares a programadores, CUDA teve uma aceitação muito rápida. Atualmente, mais de trezentas universidades possuem cursos de CUDA em seu catálogo, a página CUDA Zone registra mais de mil aplicações utilizando GPU e também vários trabalhos de pesquisa em andamento (NVIDIA 2014). São bons números, principalmente se considerarmos que a primeira versão da linguagem foi publicada em 2006. Por esta razão, a linguagem CUDA foi a escolhida para a implementação das técnicas propostas neste capítulo.

Para um maior aprofundamento sobre a arquitetura de uma GPU, otimização de transferência dados entre CPU e GPU, configuração de blocos para execução de *kernels*, hierarquia e compartilhamento de memória, tempo de latência, entre outros, recomendamos as leituras de NVIDIA (2013), McGuire (2004) e Nguyen (2007).

4.2. Detecção e Rastreamento em GPU

Analisando os passos do método proposto no capítulo anterior, é possível identificar algumas etapas que poderiam se beneficiar de uma execução em paralelo, além da etapa de detecção de vários objetos que já foi discutida

anteriormente. Esta etapa de detecção de objetos em paralelo é mais bem executada em uma CPU do que em uma GPU, visto que haveria muitos pontos de tomada de decisão (*if*´s) em cada uma das *trheads* durante a avaliação de cada uma das cascatas de classificadores. Isto iria requerer muitos pontos de sincronização e degradaria o desempenho do algoritmo, não sendo uma prática recomendada ao se programar com placas gráficas (NVIDA 2013).

Os passos do método proposto no capítulo anterior que são executados a cada *frame* do vídeo são os que possuem impacto direto no desempenho geral do algoritmo e, portanto, os elegíveis a serem executados na GPU.

Identificamos as etapas de segmentação e de cálculo da imagem integral como sendo as elegíveis para terem as suas execuções feitas pela GPU. A etapa de segmentação por se beneficiar diretamente da característica de SIMD das placas gráficas, e a etapa do cálculo da imagem integral por poder ser efetuado de forma mais eficiente em paralelo. O desempenho geral do algoritmo de detecção pode ser diretamente beneficiado caso estas duas etapas sejam executadas de forma otimizada, principalmente para o caso de vídeos em alta resolução.

4.2.1. Segmentação do vídeo

O método utilizado para segmentação do vídeo é o de subtração básica do fundo, dado pelas equações (3-1) e (3-2). O método é simples e extremamente eficiente para aplicações que necessitam atingir o requisito de tempo real, com baixo custo de processamento e de memória. Entretanto, em vídeos de alta definição, a implementação sequencial do método passa a ter um desempenho mais fraco por contado número de pixels que devem ser processados. Por exemplo, em um vídeo de resolução *Full HD*, com resolução de 1.920 por 1.080 linhas, temos ao todo 2.073.600 pixels que precisam ser processados de forma sequencial a fim de definir quais fazem parte do plano de fundo da cena.

Neste caso, o ideal é paralelizar a execução destas equações na placa gráfica. Podemos, então, com CUDA, criar um bloco de processamento com duas dimensões de *threads*, onde cada *thread* na "posição" (*x,y*) seria responsável por calcular isoladamente o valor de seu pixel correspondente. O Algoritmo 4.1 exemplifica a proposta.

O resultado deste processamento paralelo, que é o *frame* do vídeo segmentado, não precisa ser copiado de volta para a CPU, uma vez que ele será usado como entrada da próxima etapa, o cálculo da imagem integral.

Algoritmo 4.1. Subtração Básica de Fundo

1: {Dados *W e H* a largura e altura do vídeo em questão}
2: Alocar memória na GPU para envio do frame (*W * H * size_of(unsigned char)*);
3: Alocar memória na GPU para resultado (*W * H * size_of(unsigned char)*);
4: Copiar *frame* do vídeo para a GPU;
5: Executar Kernel para subtração básica de fundo (Configuração CUDA: <<<1, (*W*, *H*)>>>).

4.2.2. Operador *Prefix-Sum* Paralelo

Para resolver eficientemente um problema em uma máquina paralela, temos que projetar algoritmos paralelos que são bastante diferentes dos seus equivalentes sequenciais. A maneira mais simples de projetar um algoritmo paralelo é quando o problema pode ser resolvido por um operador básico de algoritmo paralelo. Estes operadores básicos operam em sequências, listas e árvores. Um algoritmo paralelo básico e comum é a operação de *all-prefix-sums* (operação de somas acumulativas de prefixos). Blelloch (1990) descreve *all-prefix-sums* como um bom exemplo de um cálculo que num primeiro momento parece ser inerentemente sequencial, mas para os quais existe um algoritmo paralelo eficiente. Este operador recebe uma sequência de valores e retorna uma sequência de igual comprimento para a qual cada elemento é a soma de todos os elementos anteriores na sequência original. Este operador é usado em um grande número de situações. O cálculo da imagem integral coincide com a aplicação do operador *all-prefix sums*.

Blelloch (1990) define o funcionamento do *all-prefix-sums* como um operador $\oplus$ associativo binário com identidade *D*, e um vetor de *n* elementos, $[a_0, a_1, ..., a_{n-1}]$ e retorna o vetor $[D, a_0, (a_0 \oplus a_1), ..., (a_0 \oplus a_1 \oplus ... \oplus a_{n-1})]$. Por exemplo, se $\oplus$ é uma adição $+$ com identidade 0, a operação de *all-prefix-sums* no vetor [3 1 7 0 4 1 6 3] retornará [0 3 4 11 11 15 16 22]. Outros exemplos de operadores binários $\oplus$ são o *min* (mínimo) e o *max* (máximo), *and* (“e” lógico) e *or* (“ou” lógico).

A operação de *all-prefix-sums* em um vetor de dados é comumente conhecido como *scan*. Há muitos usos para um *scan*, incluindo, mas não limitado

a: classificação, análise léxica (*e.g.* escrever um *parser* que separa um programa em *tokens*), comparação léxica de *strings*, avaliação de polinômios, fluxo de compactação (*e.g.* compactação de *streams*), histogramas de construção e implementações de algoritmos paralelos em estruturas de dados (gráficos, árvores, e listas). Para mais exemplos de aplicações, sugerimos a leitura de Blelloch (1990). Na presente tese vamos nos ater a cobrir o cálculo da imagem integral.

A implementação de uma versão sequencial de *scan* é trivial. Simplesmente precisamos percorrer todos os elementos do vetor de entrada acumulando a soma de cada um dos elementos até o elemento corrente. Esta implementação gera exatamente *n* adições para um vetor de tamanho *n*, e possui complexidade *O(n)*.

Uma implementação paralela deste código nos leva a uma primeira tentativa feita em GPU por Horn (2005), onde assumimos que há tantos processadores quanto elementos de dados, o que por muitas vezes não é o caso. A Figura 4.3 ilustra o método. O principal problema deste método é a sua complexidade, *O(n log n)*, sendo pior que a implementação sequencial de um *scan* que é *O(n)*, portanto não sendo eficiente.

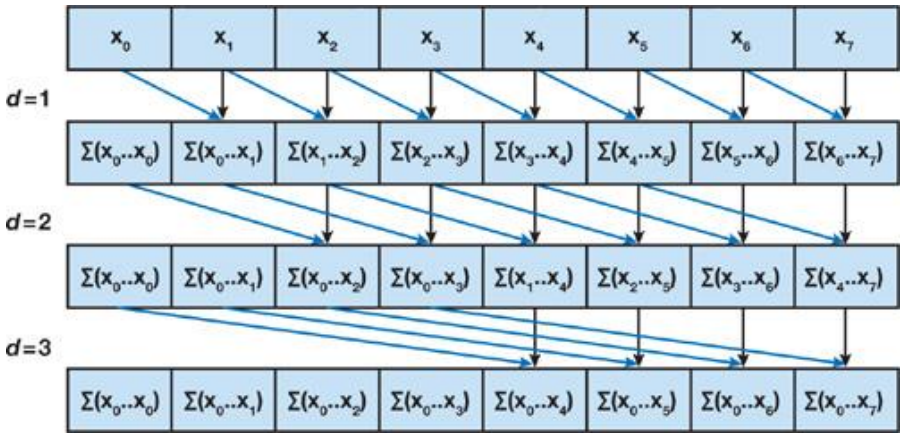


Figura 4.3. Uma implementação “simples” de um *scan* paralelo (figura extraída de Harris et al. 2007).

Uma implementação eficiente é a proposta por Blelloch (1990). Ela utiliza um padrão que usualmente surge quando se trata de paralelismo: árvores balanceadas. A ideia é construir (conceitualmente) uma árvore binária com os dados de entrada e varrê-la a partir da raiz para computar o *prefix-sum*. Esta abordagem possui complexidade de *O(n)*, ou seja, é tão complexo quanto a

versão sequencial do algoritmo. O algoritmo consiste de duas fases: a fase *up-sweep* e a fase *down-sweep*. Na fase *up-sweep* percorremos a árvore das folhas até a raiz computando as somas parciais em nós internos da árvore, como mostrado na Figura 4.4.

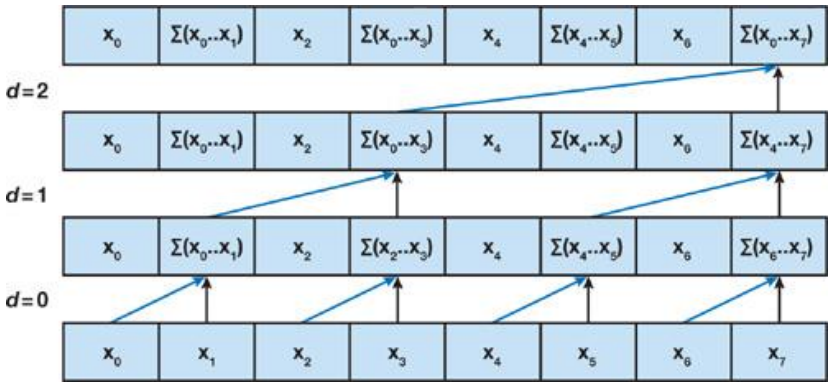


Figura 4.4. Ilustração da fase *up-sweep* de um algoritmo eficiente paralelo (figura extraída de Harris et al. (2007)).

Na fase de *down-sweep*, percorremos a árvore de volta para baixo a partir da raiz, usando as somas parciais da fase anterior para construir a varredura no local do vetor. Começamos inserindo zero na raiz da árvore, e em cada passo, cada nó no nível atual passa o seu próprio valor para o seu filho esquerdo, e a soma do seu valor e do valor anterior de seu filho esquerdo de seu filho à direita. Esta fase é representada na figura Figura 4.5.

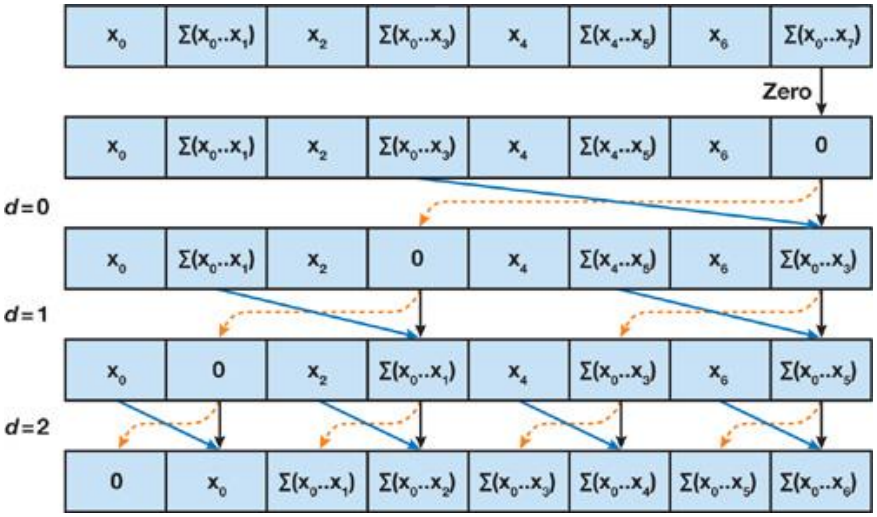


Figura 4.5. Ilustração da fase *down-sweep* de um algoritmo eficiente paralelo (figura extraída de Harris et al. (2007)).

A implementação proposta por Blelloch (1990) não é muito eficiente nas GPUs NVIDIA, devido à maneira como a memória compartilhada é acessada. Harris et al. (2007) apresentam uma proposta que corrige esta ineficiência, bem como trabalha eficientemente com um vetor dividido em vários blocos. Harris et al. (2007) também otimizam o algoritmo no que se refere à latência da memória global e a sobrecarga de instrução nas repetições (*looping*). Posteriormente, esta proposta foi melhorada por Sengupta et al. (2007). Um algoritmo baseado em Harris et al. (2007) e Sengupta et al. (2007) está disponibilizado na biblioteca CUDPP – CUDA Data Parallel Primitives Library (GPGPU.org, 2013). O uso desta biblioteca requer alguns cuidados de implementação, não apenas para atender especificidades do problema em questão, como também para considerar as constantes atualizações da arquitetura CUDA.

Uma direção futura de pesquisa é a análise do método de *scan* paralelo proposto por Dotsenko et al. (2008), que é mais rápido do que os de Harris et al. (2007) e Sengupta et al. (2007). Este método, entretanto, tem o inconveniente de estar protegido por patente da Microsoft (Dotsenko et al. 2010).

4.2.3. Utilizando *Prefix-Sum* paralelo para calcular a Imagem Integral

Podemos entender uma imagem integral como sendo uma tabela bidimensional gerada a partir de uma imagem de entrada, em que cada posição da tabela armazena a soma de todos os pixels desde o início da imagem (canto superior esquerdo) e a posição desejada, considerando a soma de todos os valores existentes tanto em linhas quanto em colunas (Figura 4.6).

Descrevemos a seguir como uma imagem integral pode ser calculada por meio de operações de *scan*, utilizando a versão paralela eficiente (Blelloch 1990) modificada para a arquitetura CUDA, conforme propostas de Harris et al. (2007) e Sengupta et al. (2007).

4	1	2	2
0	4	1	3
3	1	0	4
2	1	3	2

(a)

4	5	7	9
4	9	12	17
7	13	16	25
9	16	22	33

(b)

Figura 4.6. Dados de entrada (a) e sua respectiva imagem integral (b).

Para calcular imagem integral, podemos aplicar um *scan* de soma de todas as linhas na imagem seguido de outro *scan* de soma de todas as colunas do resultado. Para fazer isso de forma eficiente em CUDA, é preciso executar os *scans* independentes em paralelo.

Graças ao uso de blocos de *threads* fornecido por CUDA, realizar o *scan* das linhas é relativamente fácil; podemos usar um bloco de *threads* de duas dimensões, varrendo cada linha da imagem com cada linha do bloco. Realizar o *scan* em colunas levaria a uma perda de desempenho, porque a varredura em colunas geraria grandes "avanços" na memória pelos *threads*, gerando conflito de acesso entre eles (NVIDIA 2007). Por tanto, ao invés de realizar o *scan* em colunas a alternativa é construir a matriz transposta da imagem, e em seguida realizar um novo *scan* nas linhas da imagem transposta.

Resumindo, o cálculo da imagem integral de uma figura em escala de cinza pode ser feito em quatro etapas:

1. Gerar um vetor com os valores (0 a 255) de cada pixel;
2. Realizar o *scan* de todas as linhas em paralelo;
3. Com o resultado do passo anterior, gerar a respectiva matriz transposta;
4. Realizar o *scan* nas linhas da matriz transposta.

Desta forma, temos apenas duas operações de *scan* sobre um total de *largura* x *altura* elementos cada. A complexidade é mantida em *O(n)*.

A biblioteca CUDPP é utilizada para a execução das etapas descritas acima. Estes passos são, na prática, executados pela rotina *cudppMultiScan*. É importante observar que para alcançarmos um bom desempenho com esta função, é necessário alocar o vetor bidimensional enviado ao dispositivo de forma que cada linha fique "alinhada" com as demais em memória. Isto faz com que as transações em memória sejam realizadas de forma mais rápida ao acessarmos tais vetores com CUDA (NVIDIA 2007). Uma forma simples de se fazer isto é utilizar a função *cudaMallocPitch* da arquitetura CUDA para alocar um vetor de duas dimensões no dispositivo. Esta função aloca ao menos *largura* * *altura* bytes de memória linear na placa gráfica e retorna um ponteiro para a memória alocada. Ela é capaz de deslocar os bytes para garantir que os itens em memória permaneçam corretamente alinhados.

4.3. Treinamento em GPU

Viola e Jones (2001) afirmam que é necessário descobrir um pequeno conjunto de características que se consiga criar um bom classificador de um objeto, isto é, quais são as suas características mais “marcantes” que o diferenciam dos demais objetos.

Para descobrir quais seriam tais características, Viola e Jones propõem utilizar um algoritmo de aprendizado de máquina conhecido por *AdaBoost* (Freund e Schapire 1995). O método para seleção das principais características é resumido a seguir.

4.3.1. Seleção das Principais Características

Um classificador construído sobre um bom conjunto de características (no caso formado por classificadores fracos) passa a avaliar regiões da imagem de forma correta e precisa. Um classificador deve percorrer a imagem procurando regiões com os mesmos padrões que as características do objeto desejado.

Com o objetivo de otimizar a detecção de objetos numa imagem, Viola e Jones utilizam uma cascata de classificadores com N estágios, conforme descrito no Capítulo 2, Figura 2.3. Cada estágio dentro de uma cascata é criado através do algoritmo de aprendizado *AdaBoost*. A cascata é construída seguindo a seguinte heurística: estágios iniciais devem descartar um grande número de imagens que não contém o objeto desejado, e estágios mais avançados devem ser cada vez mais precisos para evitar um falso positivo do objeto. Caso uma área da imagem passe pelo último estágio da cascata de classificadores, então esta área possui o objeto procurado.

Conforme mencionado anteriormente, o treinamento consiste em selecionar as principais características retangulares de um objeto que o diferenciem nas imagens. Tais características são selecionadas através do algoritmo de aprendizado supervisionado conhecido como *AdaBoost*, que consiste em criar um comitê de classificadores “fracos” atribuindo um “peso” a cada um deles.

Tais classificadores são chamados de fracos, pois sozinhos apenas conseguem classificar corretamente pouco mais da metade das imagens. Porém estes classificadores em conjunto formam um classificador forte (um comitê de

classificadores), com taxa de erro tendendo a zero conforme aumenta o número de ciclos de treinamento (Figura 4.9).

Inicialmente, cada imagem que compõe o conjunto de imagens positivas e negativas é inicializada com o mesmo peso (Figura 4.7). A cada turno de treino, as amostras classificadas corretamente têm seu peso diminuído (Figura 4.8). Assim o algoritmo passa a focar mais nas amostras mais difíceis de serem classificadas. O pseudo algoritmo que foi implementado para seleção dos classificadores é apresentado no Algoritmo 4.2.

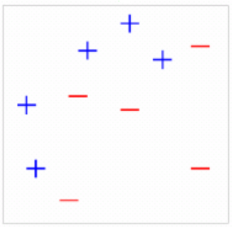


Figura 4.7. Conjunto para treinamento: 10 amostras e 2 classes distintas (símbolos “+” e “-“). Num primeiro momento, temos pesos iguais para todas as amostras.

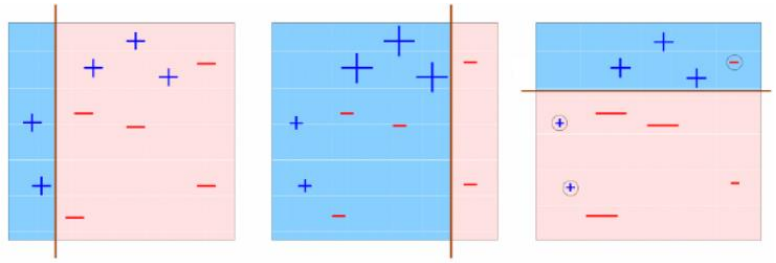


Figura 4.8. A cada iteração do *AdaBoost*, as amostras classificadas erradamente recebem um peso maior, representadas aqui pelo aumento de tamanho. As amostras corretamente classificadas tem seu peso diminuído.

Através do algoritmo apresentado, é possível observar que cada ciclo de treinamento é dependente do resultado do ciclo anterior, ou seja, dos novos pesos das amostras que foram classificadas erradamente ou corretamente. Portanto, os ciclos devem ser treinados de forma sequencial e não podem ser paralelizados.

Entretanto, o passo que é mais custoso computacionalmente é justamente o de descobrir o classificador fraco que separa melhor as amostras em duas classes, ou seja, aquela *feature* retangular que minimiza o erro de classificação.

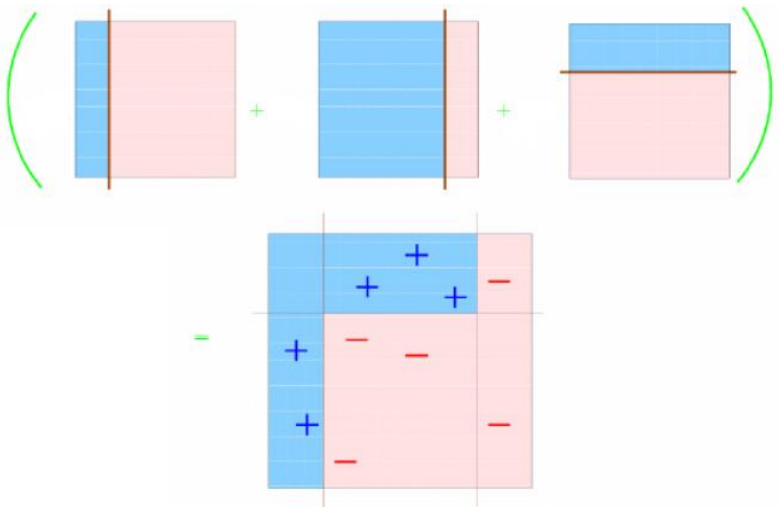


Figura 4.9. O classificador final é obtido integrando os três classificadores fracos, obtendo-se desta forma ao final um classificador forte.

Algoritmo 4.2. Treinamento dos Classificadores

1: Dado os exemplos de imagens $(x_1,y_1), \ldots, (x_n,y_n)$ onde $y_i = 0,1$ para imagens negativas e positivas respectivamente;

2: Inicializar os pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_0 = 0,1$, respectivamente, onde *m* e *l* são o número de imagens negativas e positivas;

3: De $t = 1$ até T, onde T é o número de ciclos de treinamento desejados, repetir:

   a. Normalizar os pesos, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$ para que w_t seja uma distribuição de probabilidade, sendo *n* o número de amostras de treinamento.

   b. Para cada característica, *j*, treinar um classificador h_j que seja restrito a utilizar apenas uma característica. O erro é calculado respeitando-se: $w_t, e_j = \sum_i w_i |h_j(x_i) - y_i|$.

   c. Escolher o classificador, h_t, com o menor erro e_t.

   d. Atualizar os pesos: $w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$ onde $e_i = 0$ se o exemplo x_i for classificado corretamente, e $e_i = 1$ caso contrário, e $\beta_t = \frac{e_t}{1-e_t}$.

4: Ao término, o classificador forte é:

$$h(x) = \begin{cases} 1 & se \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T} \alpha_t \\ 0 & caso\ contrário \end{cases}$$

, onde $\alpha_t = \log\frac{1}{\beta_t}$.

Matematicamente um classificador fraco pode ser definido por

$$h(x, f, p, \theta) = \begin{cases} 1 \; se \; pf(x) > \theta \\ 0 \; caso \, contrário \end{cases}$$

onde *f* é a característica retangular aplicada, *p* é a polaridade (indica a orientação da desigualdade) e *θ* é o *threshold* que decide se *x* deve ser classificado como positivo ou negativo.

Para se encontrar o classificador que minimiza o erro, é preciso utilizar de "força bruta", e para cada uma das possíveis características existentes, compará-la contra cada uma das amostras calculando o erro. Posteriormente, é preciso encontrar dentre todas estas as características calculadas a que possuir o menor erro.

De maneira geral, o treinamento de novos classificadores pode ser realizado através de três passos principais:

1. Calcular imagens integrais das amostras de treinamento;
2. Pré-calcular o valor de cada *feature* sobre cada amostra;
3. Iniciar o algoritmo de *AdaBoost*.

Nas subseções subsequentes apresentamos cada um destes passos.

4.3.2. Cálculo das Imagens Integrais

A primeira etapa para o treinamento é calcular a imagem integral de todas as amostras, uma vez que ela serve de entrada para as demais etapas. A imagem integral de cada amostra precisa ser calculada devido às características retangulares sobre as quais os classificadores serão montados. Uma possibilidade é o cálculo ser realizado utilizando o operador *prefix sum* paralelo apresentado na subseção anterior para cada uma das amostras, a fim de otimizar o tempo total do cálculo de todas as imagens integrais.

Outra possibilidade é realizar o cálculo da imagem integral de cada amostra de forma paralela ao invés de sequencial, uma vez que o cálculo precisa ser realizado apenas uma única vez para cada amostra, seguindo a ideia do Algoritmo 4.3. Nesta listagem exibimos as configurações para execução do *kernel* em CUDA, neste caso sendo um bloco para cada amostra, contendo um *thread* cada bloco. É importante observar que pode não haver memória suficiente na GPU para copiar todas as amostras de uma única vez. Neste caso,

é necessário que as amostras sejam quebradas em conjuntos menores e enviadas sequencialmente para processamento na GPU.

Algoritmo 4.3. Cálculo da Imagem Integral das Amostras

1: {Dados *Q*, *W e H* a quantidade de amostras, largura e altura de cada amostra respectivamente}
2: Alocar memória na GPU para envio das amostras (*Q * W * H * size_of(unsigned char)*);
3: Alocar memória na GPU para resultado (*Q * W * H * size_of(int)*);
4: Copiar dados das amostras para GPU;
5: Executar Kernel para cálculo de imagem integral (Configuração CUDA: <<<*Q*, 1>>>);
6: Copiar dados dos resultados para a CPU;

4.3.3. Pré-cálculo de *features*

Antes de iniciar a etapa de descoberta dos melhores classificadores (treinamento), é possível montar uma estrutura de dados auxiliar com o intuito de otimizar a etapa de cálculo de erro do melhor classificador. Com ela, é possível pré-calcular todos os valores de *features* sobre todas as amostras. Uma *feature* pode ser representada na linguagem C pela seguinte estrutura:

```
typedef struct _HaarFeature
{   char desc[  MAX_DESCRICAO ];
    struct
    {   CvRect r; /* estrutura de um retângulo */
        int weight; /* largura de cada retângulo */
    } rect[ MAX_RETANGULOS ]; /* vetor de retângulos da feature */
} HaarFeature;
```

Cada valor pode ser armazenado nesta estrutura auxiliar, composta de um vetor ordenado pelo valor dos melhores resultados obtidos, de acordo com o pseudo algoritmo do Algoritmo 4.4. A quantidade de blocos sugerida em CUDA neste caso é de *Núm. Features* x *Núm. Amostras*, com uma *thread* por bloco. Desta forma, a etapa seguinte de calcular o erro do melhor classificador pode ser realizada de maneira muito mais rápida, já que todas as *features* estarão previamente calculadas. A estrutura de dados utilizada para esta etapa de pré-cálculo é ilustrada na Figura 4.10.

Algoritmo 4.4. Pré-cálculo de todas as características

1: {Dados *Q*, *W e H* a quantidade de amostras, largura e altura de cada amostra respectivamente}
2: {Dado *F* a quantidade de *features* utilizadas}
3: Alocar memória na GPU para envio das imagens integrais (*Q* * *W* * *H* * size_of(int));
4: Alocar memória na GPU para envio das "features" (*F* * *size_of(Haar Feature*));
5: Alocar memória na GPU para resultado (*F * Q * size_of(int)*);
6: Executar Kernel para Calcular cada *Feature C* sobre cada amostra *A* (Configuração CUDA: <<<(*F, Q*), 1>>>);
7: Copiar dados dos resultados para a CPU;
8: Para cada *feature*, ordenar pelos melhores valores por amostra;

Apesar deste pré-cálculo das *features* poder ser paralelizado, vale lembrar que o pré-cálculo é bastante custoso em termos de memória. Supondo um conjunto de 10.000 características para treino e 10.000 amostras, seria necessária uma matriz de dimensão [10.000][10.000] para armazenar todos os valores pré-calculados. Muitas vezes não há memória suficiente para que tal matriz possa ser alocada na GPU, sendo necessário quebrar a matriz em vários subconjuntos que possam ser processados.

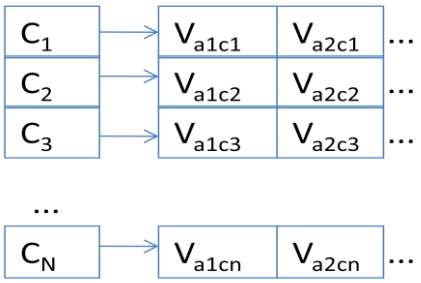


Figura 4.10. Representação dos vetores V ordenados pelos melhores valores de cada *feature* C aplicada em cada amostra A do conjunto de treinamento.

4.3.4. Montagem do Classificador Forte

Esta etapa final consiste na construção do classificador forte, que nada mais é do que a integração de todos os classificadores fracos identificados ao longo de cada iteração do algoritmo *AdaBoost*.

Um classificador fraco pode ser representado pela seguinte estrutura na linguagem C:

```
typedef struct _WeakClassifier
{
  double thresh;
  double error;
  int parity; /* igual a 1 ou -1 */
  HaarFeature feature;
} WeakClassifier;
```

Podemos observar que cada ciclo do treinamento com *AdaBoost* é dependente do resultado do ciclo anterior, ou seja, dos novos pesos das amostras que foram classificadas erradamente ou corretamente. Portanto, os ciclos devem ser treinados de forma sequencial e não podem ser paralelizados.

Entretanto, o passo que é mais custoso computacionalmente é justamente o de descobrir o classificador fraco que melhor separa as amostras em duas classes, ou seja, aquela característica retangular que minimiza o erro de classificação. Tal tarefa pode ser paralelizada, pois o cálculo do erro de cada característica em relação às amostras de treino é totalmente independente entre si. Ao final, precisamos apenas selecionar a característica que apresentar o menor erro.

O pseudo algoritmo é apresentado no Algoritmo 4.5. A configuração CUDA sugerida para execução do *kernel* é de (*Núm. Features*/16) * (*Núm. Amostras*/16) blocos, com 256 *threads* cada bloco. Este número de blocos e *threads* pode variar de acordo com a versão da placa gráfica utilizada.

Algoritmo 4.5. Geração do Classificador Forte

1: {Dado *Q* a quantidade de amostras}
2: {Dado *F* a quantidade de *features* utilizadas}
3: {Dado *N* a quantidade de ciclos desejados de treinamento}
4: Inicializar peso das amostras
5: De *T*=1 até *N*
   a. Normalizar os pesos das amostras;
   b. Executar Kernel que gera os classificadores fracos, utilizando valores pré-calculados (Configuração CUDA: <<<(*F*/16, *Q*/16), (16,16)>>>);
   c. Montar vetor de “classificadores fracos” com resultado do kernel;
   d. Obter o classificador que possuir o menor erro, e que também não tenha sido selecionado em ciclos anteriores;
   e. Aplicar o “classificador fraco” selecionado em cada amostra e diminuir os pesos de cada amostra classificada corretamente;
6: Armazenar o classificador forte.

4.4. Conclusão

As placas gráficas (GPU - *Graphics Processing Unit*) foram inicialmente projetadas para processar apenas gráficos, contendo um *pipeline* fixo para renderização de cenas. Uma vez que elas possuem algumas características como processamento paralelo e alto poder computacional para cálculos aritméticos, começou-se a introduzir alguns estágios programáveis dentro desse *pipeline* das placas gráficas. A evolução natural passou a ser usar a GPU para processamento genérico, o chamado GPGPU (*General Purpose Graphics Processing Unit)*.

Nós apresentamos durante este capítulo otimizações no método de detecção e rastreamento de objetos através de programação na placa gráfica. Para tanto, fizemos uma breve descrição do funcionamento de uma GPU, quais são as vantagens de a utilizarmos, e por fim, uma visão geral do funcionamento da arquitetura CUDA e sua execução de *kernels* em blocos de *threads* (NVIDIA 2013).

Através do alto poder de paralelismo obtido com o uso de uma placa gráfica, é possível melhorar o desempenho geral da detecção e do rastreamento de objetos em relação ao desempenho obtido do mesmo algoritmo executado em uma CPU. Tal feito é atingido levando-se para a placa gráfica alguns passos importantes do algoritmo e que são executadas a cada quadro do vídeo, que são as etapas de Segmentação do Vídeo e do Cálculo da Imagem Integral.

Demonstramos que a segmentação de vídeo pode ser implementada de forma paralela ao invés da sequencial, fazendo com que vídeos de alta resolução tenham seus pixels segmentados de forma extremamente eficiente e rápida.

Vimos também que o cálculo da imagem integral pode ser feito de forma paralela, através do operador *prefix-sum* e o respectivo *scan* otimizado na GPU, utilizando o conceito de árvores balanceadas ao processar a imagem de *foreground* de um vídeo, mantendo a complexidade de *O(n)* no algoritmo apresentado.

Por fim, apresentamos uma metodologia para melhorar o desempenho da fase de treinamento de novos objetos através da GPU, tornando a execução paralela de alguns trechos do algoritmo de *AdaBoost*.

Os tempos de detecção e rastreamento, treinamento, assim como as taxas de convergências encontradas, são apresentados no próximo capítulo.

5
Resultados Experimentais e Discussão

Este capítulo apresenta o protótipo desenvolvido durante esta pesquisa, capaz de realizar o treinamento e a detecção de objetos em um único ambiente integrado, seguindo os métodos propostos nos Capítulos 3 e 4 desta tese.

Apresentamos a arquitetura do protótipo, consistindo de um módulo para treinamento de objetos, um módulo para detecção de objetos em imagens ou vídeos digitais, e por fim um módulo para utilização da placa gráfica para a execução de tais tarefas. São apresentadas também algumas interfaces do protótipo desenvolvido a fim de ilustrar a interação do mesmo por um usuário.

A fim de avaliar a eficiência do método proposto, foram realizados experimentos comparando os resultados obtidos com as novas técnicas aqui propostas com o resultado alcançado pelo algoritmo original proposto por Viola e Jones (2001). A comparação direta dos resultados é possível uma vez que o algoritmo de Viola e Jones encontra-se implementado na biblioteca OpenCV da Intel (Intel 2006).

Também foram realizados experimentos sobre a utilização de placas gráficas para processamentos em paralelo, tanto para a melhoria do desempenho do algoritmo de detecção de objetos, quanto para a melhoria do tempo de treinamento de novas classes de objetos.

5.1.
Protótipo Desenvolvido

Foi construído um protótipo durante esta pesquisa visando criar um ambiente integrado onde seja possível tanto treinar novas cascatas de classificadores, quanto detectar e rastrear em tempo real objetos previamente treinados. O processo de detecção pode ser utilizado tanto em imagens quanto em vídeos digitais. A definição dos módulos que compõe o protótipo foi feita de forma que cada um constitua uma unidade lógica bem definida e coesa e com baixo acoplamento entre si. Para construção do protótipo, foram utilizadas as linguagens C e CUDA, sendo Visual Studio 2010 (Microsoft) o ambiente de desenvolvimento.

O protótipo desenvolvido utiliza algumas bibliotecas muito conhecidas na literatura: a biblioteca OpenCV (*Open Computer Vision*), desenvolvida pela Intel (2006) que contém diversas estruturas de dados e algoritmos para manipulação de vídeos e imagens, a biblioteca IUP (*Portable User Interface*), produzida pelo Tecgraf – PUC-Rio (1994) que é uma API para construção de interfaces gráficas para o usuário, e por fim a biblioteca CUDPP (*CUDA Data Parallel Primitives Library*), de código aberto e construída de forma colaborativa, sendo mantida pela organização GPGPU (GPGPU.org 2013).

Dentre os módulos desenvolvidos, podemos listar:

• *Interface* – Responsável por construir toda a interface gráfica para o usuário. Faz uso direto da biblioteca IUP.

• *Object Learner* – Responsável por realizar o treinamento de novas classes de objetos. Contém rotinas diversas para a geração automática de amostras de imagens para treinamento (aumento, redução, rotação, etc.), assim como rotinas de suporte às demais etapas do treinamento, como parametrização das cascatas de classificadores, marcação de objetos para treinamento em imagens, entre outras. Por fim, é capaz de testar o desempenho de uma cascata de classificadores sobre um conjunto de imagens pré-selecionadas, exibindo diversas informações tais como: percentual de acertos, falhas e falsos positivos.

• *Object Tracker* – Capaz de rastrear um ou mais objetos em um vídeo (ou imagem estática) e informar na imagem em que região cada objeto se encontra.

• *GPU Manager* – Engloba as rotinas de treinamento e rastreamento de objetos para execução na placa gráfica. Este módulo é utilizado apenas quando for parametrizado na interface que o respectivo algoritmo deve ser executado numa GPU para otimização de desempenho. Grande parte deste módulo é escrito em CUDA e utiliza a biblioteca CUDPP.

• *Main* – Responsável por realizar as inicializações em memória necessárias ao programa, como por exemplo, as estruturas de interface com o usuário.

Os módulos aqui apresentados foram desenvolvidos seguindo os padrões de programação modular propostos por Staa (2000), com o intuito gerarmos um artefato de software de alta qualidade, confiável e robusto. A Figura 5.1 ilustra a arquitetura e a interdependência entre os módulos do protótipo.

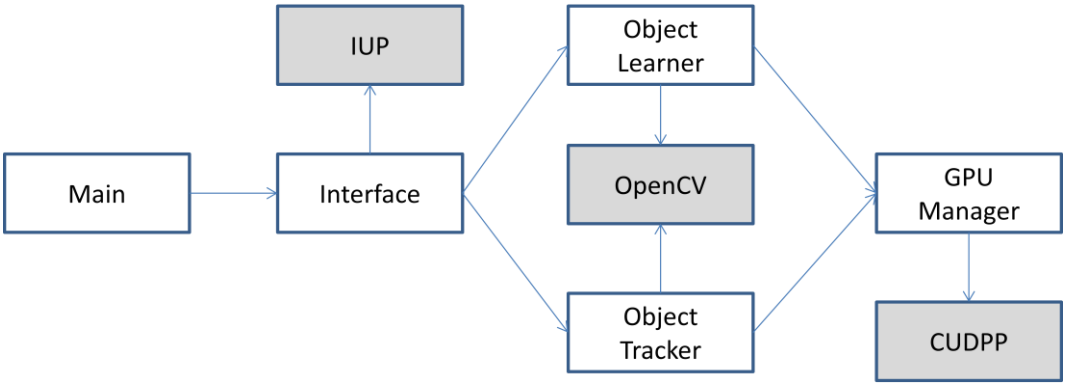


Figura 5.1 Interdependência dos módulos que compõem o protótipo construído. As bibliotecas utilizadas estão representadas com fundo cinza.

5.2. Detecção e Rastreamento de objetos

Para a detecção e rastreamento de objetos é necessário informar dois dados: o arquivo do vídeo que contém o objeto a ser rastreado e a cascata de classificadores previamente treinada, lembrando que pode ser informado uma cascata de classificadores para cada um dos objetos a serem rastreados. Neste caso, onde mais de um objeto é procurado simultaneamente, cada um dos objetos encontrados será circulada com cores distintas, uma para cada classe de objeto. Desta forma o usuário tem como identificar facilmente os diferentes objetos detectados. Um exemplo de detecção da classe de objeto “face” pode ser visto na Figura 5.2.

O módulo de detecção de objetos também está preparado para ao invés de um vídeo, receber como parâmetro o caminho físico de uma imagem. Neste segundo caso, a detecção será feita apenas na imagem enviada.

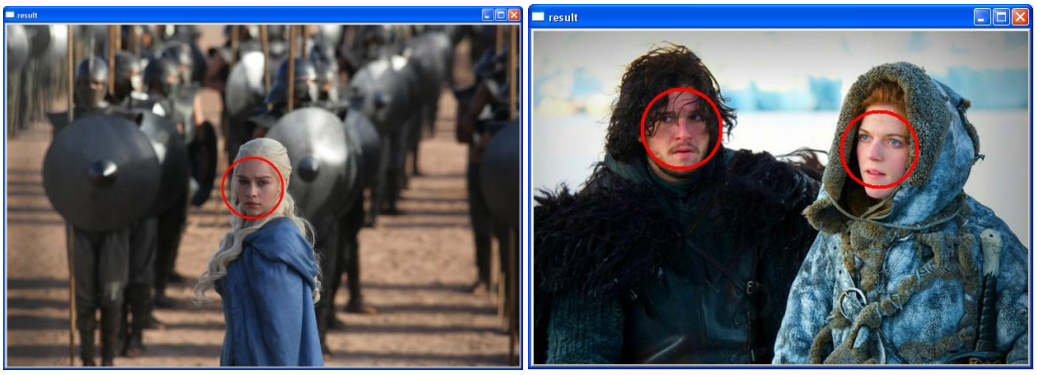


Figura 5.2. Exemplos de detecção da classe de objeto “face”. A região que contém o objeto possui um círculo destacando-a. Imagens com direitos autorais reproduzidas no âmbito da política de "*fair use*” (*Game of Thrones* (2013)).

No caso de vídeos digitais, qualquer resolução é aceita, entretanto o protótipo permite apenas arquivos com extensão “AVI” que não utilizem *codecs* de compactação (e.g. *MPEG*) devido a restrições de compatibilidade na biblioteca OpenCV (Intel 2006). No caso de imagens, as extensões de arquivos mais utilizadas atualmente são aceitas pelo detector (e.g. “*gif*”, “*bmp*” e “*jpeg*”).

A interação com o protótipo para a detecção de objetos é simples, conforme ilustrado pela Figura 5.3. O usuário informa as *N* cascatas de classificadores, o vídeo ou imagem e, por fim, seleciona o método de detecção desejado: em CPU ou em GPU.

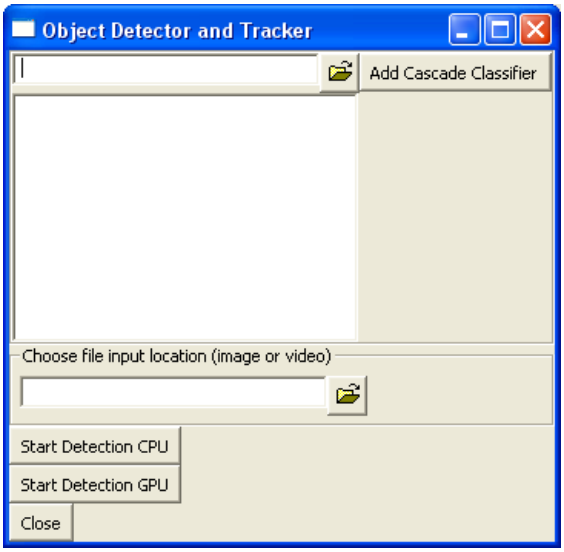


Figura 5.3. Interface para detecção de objetos. Usuário deve selecionar uma ou mais cascatas de classificadores, e uma imagem ou vídeo para subsequente detecção dos objetos desejados. O usuário também escolhe se deseja fazer a detecção na CPU ou em GPU.

5.3. Treinamento

O módulo de treinamento visa proporcionar um ambiente que o usuário seja capaz de treinar novas classes de objetos de maneira simplificada.

Para efetuar o treinamento é preciso que o usuário forneça como entrada dois conjuntos de imagens fundamentais: um de **imagens positivas** e outro de **imagens negativas**. Com isto, o algoritmo é capaz de “aprender” as características mais relevantes do objeto e montar uma cascata de classificadores.

Viola e Jones (2001) especificam que as imagens que formam o conjunto de imagens positivas possuam a mesma dimensão, e que o objeto para treinamento esteja posicionado da mesma maneira em todas as imagens, sendo aceitável ter pequenas transformações dos objetos nas imagens. Também especificam que nas imagens que formam o conjunto de imagens negativas, nenhuma delas pode conter o objeto foco do treinamento, e que o número de imagens deste conjunto deve ser maior que o número de imagens positivas. No artigo de Viola e Jones (2001) para o treinamento do objeto "face humana" foi utilizado um conjunto de imagens positivas contendo 5.000 faces, e um conjunto de imagens negativas contendo 10.000 imagens.

Para o conjunto das imagens negativas, existem duas formas de defini-lo: 1 - os arquivos que o compõe podem estar contidos inteiramente em um único diretório, ou; 2 - ser representado por um arquivo texto que contenha o caminho físico completo de cada uma das imagens que foram o conjunto.

O conjunto de imagens positivas também pode ser informado de duas maneiras: 1 - um arquivo texto contendo o caminho físico completo de cada imagem juntamente com as localizações na imagem do objeto que se deseja treinar, ou; 2 – um arquivo binário com extensão "*vec*" que contém uma sequência de imagens apenas do objeto a ser treinado.

Com o objetivo de facilitar o treinamento de novas classes, o protótipo permite que o usuário crie um conjunto finito de imagens positivas a partir de uma única imagem do objeto, através de transformações aplicadas na imagem original. O arquivo gerado no final deste processo possui a extensão "*vec*" e as imagens contidas dentro dele possuem as dimensões definidas pelo usuário. A Figura 5.4 ilustra este processo.

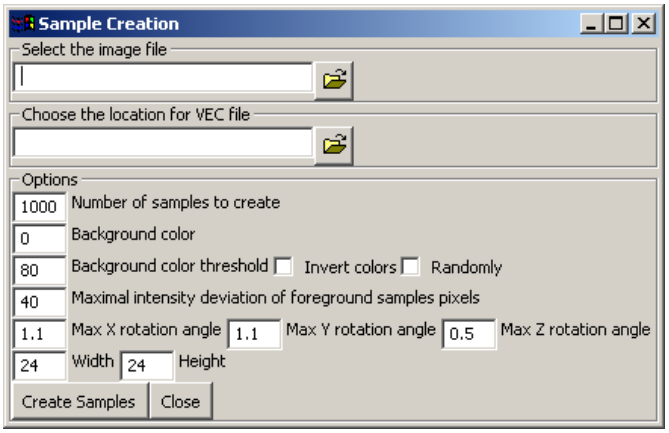


Figura 5.4. Interface para criação de amostras para treinamento a partir de uma imagem original. As amostras geradas possuem as transformações definidas pelo usuário.

Uma vez formados os conjuntos de imagens positivas e negativas, o usuário pode optar por iniciar o processo de treinamento do objeto. A interface permite que alguns parâmetros de como o treinamento deve ser realizado possa ser informado pelo usuário. Uma informação relevante que o usuário pode informar neste momento é se o objeto a ser treinado possui simetria vertical. Caso positivo, o tempo de treinamento pode ser reduzido substancialmente, uma vez que metade de um lado da imagem é um "espelho" da outra metade. O protótipo mostra ao usuário os valores padrão para o treinamento definidos por Viola e Jones (2001), porém o parâmetro que corresponde à resolução de pixels da imagem do objeto a ser treinado deve ser informado conforme cada caso, evitando-se colocar resoluções maiores que 30x30 para não tornar o treinamento demasiadamente demorado. Os parâmetros para o treinamento de um objeto são ilustrados na Figura 5.5.

O usuário pode escolher se executa o treinamento na placa gráfica (GPU) ou não. Para ambos os casos, o processo de treinamento pode ser acompanhado por meio do console da aplicação. Ao término do processo de treinamento é gerado um arquivo *XML* que contém a cascata de classificadores. Esta cascata pode ser utilizada pelo algoritmo de detecção.

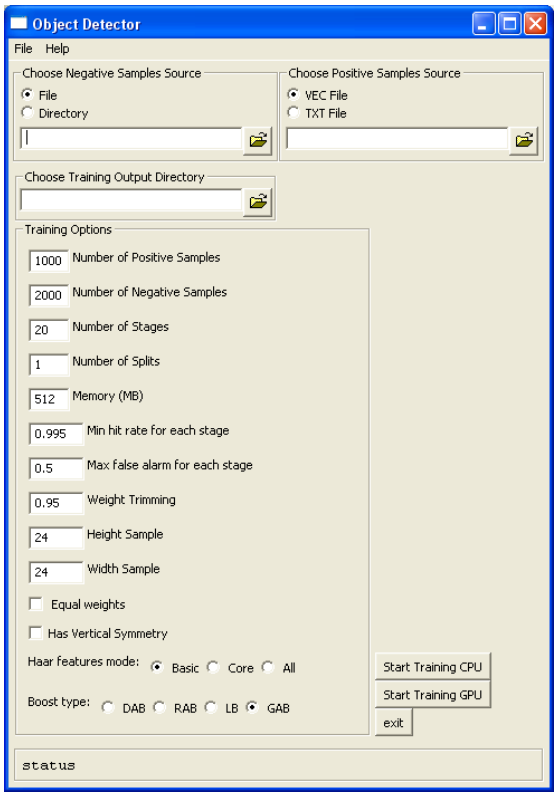
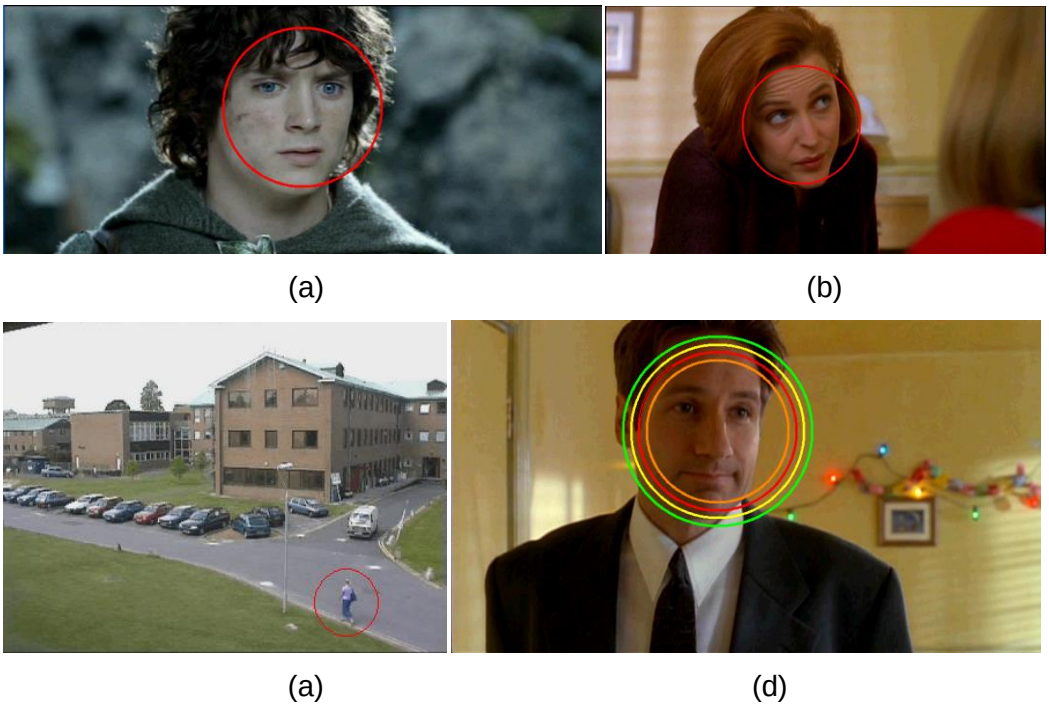


Figura 5.5. Parâmetros utilizados para o treinamento de novas classes de objetos. O treinamento pode ser realizado na CPU ou na GPU.

5.4.
Experimentos e Resultados Obtidos

Esta seção apresenta os resultados experimentais obtidos com diferentes sequências e resoluções de vídeos. O conjunto de dados utilizado para os experimentos são trechos de vídeos conhecidos de entretenimento, como “O Senhor dos Anéis”, “Arquivo X” e “*Game of Thrones*”, e também um vídeo que contém uma cena de pedestres caminhando, popularmente utilizado na literatura relacionada a detecção de objetos, como mostrado na Figura 5.6.



(a) (b)

(a) (d)

Figura 5.6. Exemplos de rastreamentos de objetos em vídeo. (a) e (b) ilustram o rastreamento de um único objeto (face) em vídeos HD. (c) ilustra o rastreamento de um pedestre em um vídeo de resolução padrão. (d) ilustra o rastreamento de múltiplos objetos, neste caso quatro objetos da mesma classe ”face” são rastreados, cada um com um raio e cor diferentes. Imagens com direitos autorais reproduzidas no âmbito da política de "*fair use*” (Senhor dos Anéis: O Retorno do Rei (2001) e Série Arquivo X (2002)).

Os experimentos realizados com o protótipo desenvolvido focaram os seguintes aspectos:

1. Detecção na CPU
    a. De um objeto ao longo de um vídeo;
    b. Detecção de dois a quatro objetos simultaneamente ao longo de um vídeo.
2. Detecção na GPU
    a. De um objeto ao longo de um vídeo;
3. Treinamento de novas classes de objetos em GPU

É importante salientar que apenas foi testado um máximo de 4 classes de objetos a serem detectadas de forma simultânea por ser o número máximo de paralelização real conseguida no ambiente onde foram realizados os experimentos. Em um ambiente onde fosse possível ter um maior poder de paralelização, o número de classes de objetos a serem detectadas simultaneamente poderia ser aumentado de forma proporcional. Na execução do algoritmo de detecção numa GPU, não foram feitos testes com mais de uma classe de objeto simultaneamente, pois como cada classe de objeto é representada por uma cascata diferente, ocorreriam divergências durante a execução dos *threads*, o que não é recomendado de ocorrer num ambiente de modelo SIMD, que é justamente o modelo implementado pelas GPUs.

As seguintes resoluções foram utilizadas nos experimentos: 320 x 240, 640 x 480, 1280 x 720 e 1920 x 1080. O ambiente utilizado para os testes foi um Intel Core 2 Quad 2.4 GHz, com 4 núcleos de processamento, 4 GB de RAM, Windows XP. A placa gráfica utilizada para a execução dos algoritmos, quando escolhida a opção de execução em GPU, foi uma placa GeForce 8600 GT da Nvidia.

O valor médio de $\alpha = 0,7$ foi utilizado para a segmentação adaptativa do fundo. O valor de α foi definido empiricamente. Observamos que um nível muito baixo do valor de α diminui o tempo de processamento, pois quanto menor for o seu valor, o *foreground* calculado fica maior e a área a ser avaliada aumenta. Um valor muito alto para α não captura todas as mudanças necessárias do cenário, fazendo com que alguns objetos não sejam detectados entre os quadros de um vídeo (elevado número de "*missing positives*"). Seguindo a mesma ideia, os valores de λ (fator de escala da janela de pesquisa) e Δ (deslocamento da janela de pesquisa) foram escolhidos após experimentos empíricos.

O nosso método tem a mesma precisão do algoritmo inicialmente proposto por Viola e Jones (2001), uma vez que não foi alterado o processo de detecção de objetos através de uma cascata de classificadores. Durante nossos experimentos observamos que mesmo com a mudança do fator de escala e o deslocamento da janela de busca, respectivamente λ e Δ, conseguimos exatamente o mesmo número de taxas de objetos encontrados e falsos positivos para ambos os métodos.

Devido ao baixo custo computacional do método proposto e o elevado número de áreas entre os quadros descartados de um vídeo, é possível atingir uma elevada taxa de quadros por segundo, incluindo imagens de alta definição, tal como mostrado na Tabela 5.1.

Na experiência da Figura 5.6, foram utilizados quatro objetos da mesma classe, ao invés de quatro objetos totalmente diferentes, principalmente para evitar a introdução de novas variáveis que poderiam causar uma avaliação tendenciosa do processamento paralelo proposto.

Também devemos notar uma redução substancial na quantidade de áreas processadas durante a busca por objetos, como mostrado na Tabela 5.2. No entanto, é importante notar que o ganho com a área de busca é reduzida de maneira proporcional à quantidade de movimento entre os quadros do vídeo. Atualmente o método proposto não lida com oclusão. O tempo de processamento obtidos usando a técnica paralela proposta para detecção de vários objetos são mostradas nas Figura 5.7 e Figura 5.8. Nessas duas figuras, podemos ver que o nosso algoritmo produz um comportamento praticamente linear com realação ao número de objetos procurados.

Resolução do Vídeo	FPS Método Proposto	FPS Viola e Jones
320 x 240	94,7	54,2
640 x 480	72,6	28,3
1.024 x 720	41,3	14,4
1.920 x 1.080	21,4	5,9

Tabela 5.1. Tempo de processamento médio em quadros por segundo (FPS) procurando um único objeto em um vídeo, para cada resolução especificada.

Resolução do Vídeo	Método Proposto	Viola e Jones
320 x 240	1.984	81.246
640 x 480	4.206	162.642
1.024 x 720	7.628	305.324
1.920 x 1.080	15.842	620.976

Tabela 5.2. Número total de áreas processadas procurando por um objeto segundo a técnica originalmente proposta por Viola e Jones (2001) e a técnica proposta, usando uma janela de pesquisa inicial de tamanho 20x20, λ = 1,3 e Δ = 1. Valores extraídos de acordo com o quadro da Figura 3.4.

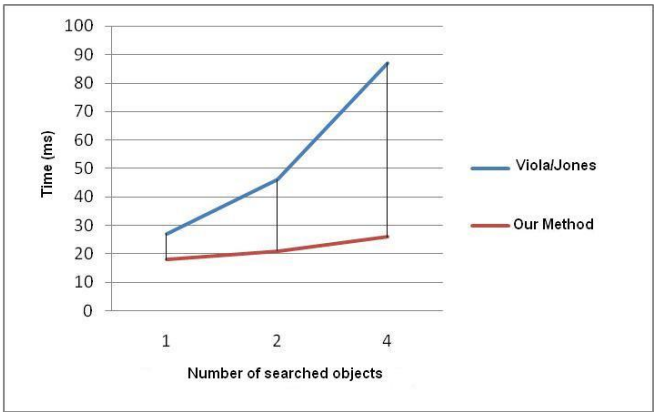


Figura 5.7. Desempenho médio do algoritmo para a detecção de objetos com e sem as otimizações propostas com uma resolução de vídeo de 640x480.

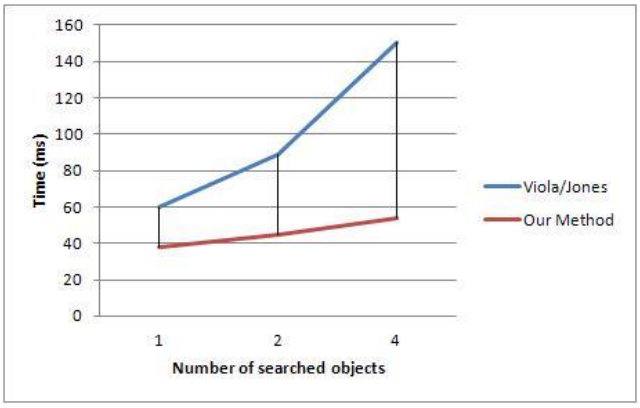


Figura 5.8. Desempenho médio do algoritmo para a detecção de objetos com e sem as otimizações propostas com uma resolução de vídeo de 1.024 x 720.

Podemos observar que há uma diminuição da quantidade de *frames* processados por segundo conforme aumenta a resolução do vídeo. No caso de vídeos na resolução *Full HD*, podemos observar na Tabela 5.1 que temos uma taxa abaixo de 30 FPS, que seria a ideal a ser atingida em vídeos. Sabendo que um vídeo de alta definição possui um número muito grande de pixels (uma imagem *Full HD* possui 2.073.600 pixels) a ser processado, foram refeitos os experimentos de tempo médio de processamento por *frame* utilizando uma placa gráfica, para desta maneira, podermos avaliar o possível ganho de desempenho ao se executar algumas das etapas do algoritmo de detecção de forma paralela. Os resultados obtidos estão descritos na Tabela 5.3. É interessante observar pelos resultados desta tabela que não houve a melhora que era esperada no desempenho de todas as resoluções, pelo contrário, em algumas delas houve degradação no desempenho obtido quando comparado com o algoritmo original em CPU. Isto deve-se pela frequente troca de dados entre a CPU e a GPU. A alocação e envio de dados para uma GPU é uma operação relativamente lenta, e dependendo da situação, é mais custoso fazer o envio da informação para a placa gráfica do que realizar o processamento na própria GPU. Esta é a situação identificada nos vídeos de baixa resolução. Entretanto, para vídeos de alta resolução, o ganho obtido com o processamento numa GPU compensa o custo do envio dos dados. Em vídeos *Full HD*, conseguimos uma melhora no desempenho superior a 40%.

Resolução do Vídeo	FPS Método Proposto	FPS com GPU
320 x 240	94,7	76,3
640 x 480	72,6	61,2
1.024 x 720	41,3	42,6
1.920 x 1.080	21,4	31,7

Tabela 5.3. Tempo de processamento médio em quadros por segundo (FPS) procurando um único objeto em um vídeo, para cada resolução especificada, utilizando uma GPU.

Fizemos também uma análise adicionando ruído de forma aleatória nos *frames* já segmentados dos vídeos analisados, com o objetivo de avaliar o desempenho do método proposto em imagens muito ruidosas. O ruído adicionado equivale a um pixel branco, o que faz com que a imagem integral da região não seja igual a zero. O ruído foi adicionado de maneira uniforme pela

imagem. A Tabela 5.4 contém os resultados obtidos. É possível observar que quanto mais ruidosa for a imagem, maior será o tempo de processamento do *frame*, aproximando-se do pior caso que é quando todo o *frame* precisa ser analisado na busca pelo objeto desejado.

Resolução do Vídeo	N° de Pixels de Ruído			
	Sem Ruído	50	100	300
320 x 240	76,3	74,1	66,2	57,3
640 x 480	61,2	58,3	48,8	32,4
1.024 x 720	42,6	36,4	29,1	17,8
1.920 x 1.080	31,7	26,8	16,4	9,5

Tabela 5.4. Tempo de processamento médio em quadros por segundo (FPS) procurando um único objeto em um vídeo, para cada resolução especificada, variando a quantidade de ruído aleatório, utilizando uma GPU.

Apesar de termos obtido um bom desempenho para a detecção de objetos, o treinamento de novas classes de objetos é algo extremamente dispendioso. De acordo com Viola e Jones (2001), são necessários dias para realizar um treinamento de uma nova classe de objeto por completo. Nossa intenção é verificar a melhoria no desempenho do treinamento de novos objetos com o auxílio de uma GPU. Para realizar tal tarefa, utilizamos a base de imagens de faces do MIT *Center For Biological and Computation Learning* (CBCL 2000), consistindo de 2.429 faces e 4.548 não faces, todas com resolução de 19x19 pixels. Foi escolhida para treino uma base de faces devido a ser uma classe de objeto muito estudada ao longo dos anos e, portanto, com muitas bases disponíveis na internet.

Num. Ciclos / Tempo de Processamento	CPU	GPU
10	56,3 s	3,9 s
100	432,9 s	25,6 s
200	875,1 s	54,1 s
500	1.622,6 s	90,3 s
1.000	3.420,2 s	162,8 s
5.000	10.612,1 s	727,3 s

Tabela 5.5. Quadro Comparativo do desempenho da GPU x CPU em vários ciclos de treinamento. Tempo em segundos.

Na Tabela 5.5 podemos observar que a redução do tempo de treinamento teve uma melhora significativa para o treinamento composto por 5.000 ciclos na GPU. Concluímos que o aumento do número de ciclos aumenta o tempo de execução na CPU exponencialmente, ocasionando a demora de dias para o treinamento de vários estágios da cascata conforme constatado no trabalho de Viola e Jones (2001).

O tempo total de treinamento também está relacionado com a quantidade de amostras do treino. Quanto maior o número de amostras, maior será o tempo para treinar e identificar os melhores classificadores de cada estágio. Além disto, quanto maior o número de ciclos treinados menor será o erro do aprendizado, tendendo a zero conforme provado em Freund e Schapire (1995). De fato, com o conjunto de treino especificado contendo um total de 6.977 amostras, aproximadamente no ciclo 70 já é possível obter um classificador forte capaz de classificar todas as amostras sem erros. Os demais classificadores apenas geram mais complexidade sem necessidade. Para que os novos classificadores tivessem utilidade o número de amostras do conjunto de treino deveria ser maior.

Além do treinamento da classe face, também efetuamos o treinamento da classe “olhos”. As amostras foram criadas a partir da própria base de faces do CBCL, recortando a região de onde se localizam os olhos. Isto é possível de ser feito uma vez que todas as faces da base estão aproximadamente na mesma posição. Na Tabela 5.6 podemos observar o tempo total de treinamento para estas novas classes de classificadores tanto em CPU quanto em GPU. Foram utilizados os parâmetros *default* de treinamento existentes na biblioteca OpenCV. Para realizar os testes de acurácia sobre as novas cascatas treinadas, foi utilizada a base de teste MIT + CMU, e os resultados obtidos encontram-se na Tabela 5.7. Nós usamos nesta tabela o número de falsos positivos como oposição à taxa de falsos positivos para facilitar a comparação com as outras classes de objetos. Para calcular a taxa de falsos positivos, é necessário dividir o valor pelo número total de subjanelas percorridas. Em relação aos resultados da cascata para detecção de faces, é possível observar um desempenho inferior ao obtido pela cascata já existente na biblioteca OpenCV, provavelmente devido ao número inferior de amostras utilizadas para a realização do treinamento. Um desempenho inferior pode ser observado para a detecção de olhos, provavelmente devido à geração do conjunto positivo de imagens para treino ter sido feita considerando que todas as faces estão exatamente na mesma

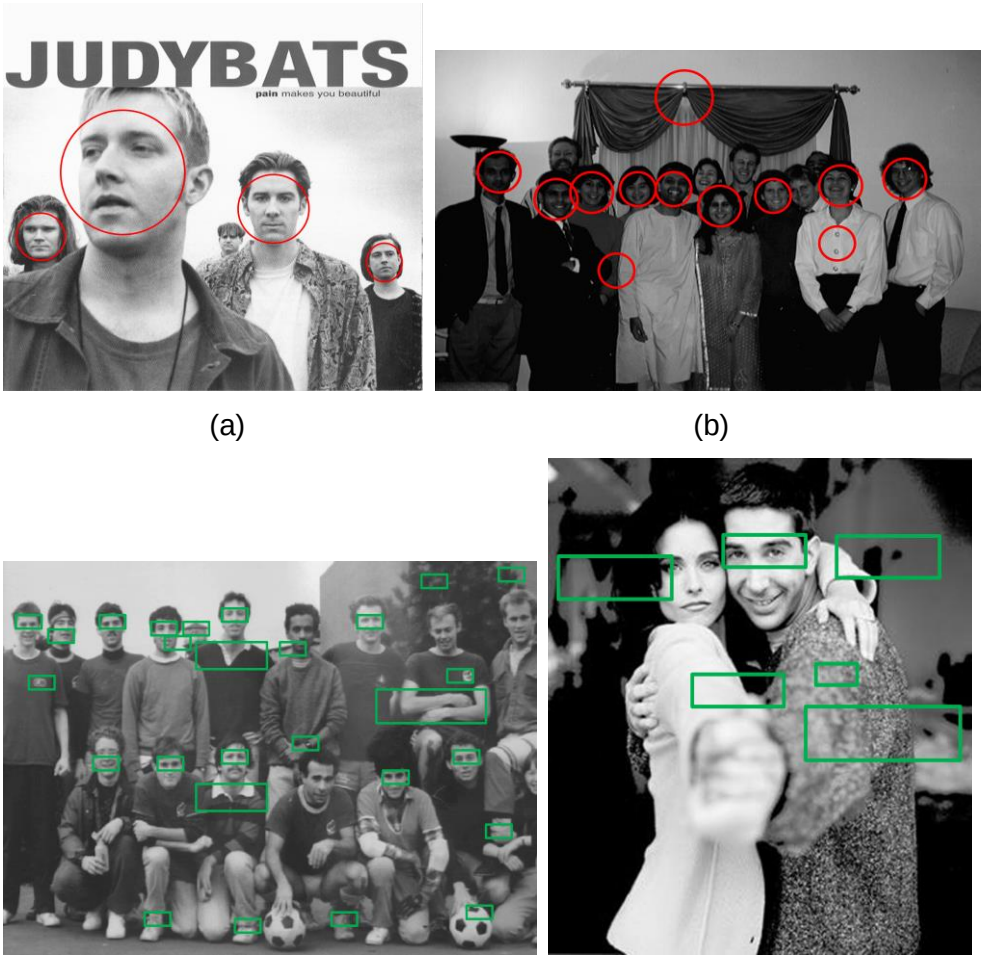
posição, e, além disto, o número de pixels que formam a região dos olhos é ainda menor do que o de uma face. Exemplos de detecção de ambas as classes treinadas em GPU estão ilustrados na Figura 5.9. É importante salientar que se utilizarmos os mesmos parâmetros para treinamento na CPU e na GPU, o resultado em termos de desempenho de classificação será o mesmo.

Classe	Tempo em horas	
	CPU	GPU
Face	26,3	5,6
Olhos	24,1	4,7

Tabela 5.6. Quadro comparativo do tempo total do treinamento das classes “Face” e “Olhos” realizados com uma GPU e CPU.

Cascata	Falsos Positivos				
	10	50	80	100	150
Face (OpenCV)	82,3%	87,8%	91,1%	93,4%	96,2%
Face	74,6%	77,8%	79,4%	84,8%	87,6%
Olhos	43,6%	47,4%	50,7%	55,1%	59,3%

Tabela 5.7. Taxa de detecção para vários números de falsos positivos sobre a base de teste MIT + CMU contendo 130 imagens e 507 faces. A primeira linha apresenta o resultado obtido com uma cascata para detecção de faces já existente na biblioteca OpenCV. As demais linhas apresentam os resultados das cascatas treinadas no protótipo construído utilizando uma GPU.



(a) (b)

(c) (d)

Figura 5.9. Exemplos de teste com as cascatas para detecção de faces e olhos treinada em GPU. (a) e (b) contém exemplos de detecção de objetos da classe face, enquanto (c) e (d) contém exemplos de detecção de objetos da classe olhos. É possível observar um maior número de falsos positivos e objetos não detectados na classe olhos. Imagens contidas na base de testes MIT + CMU.

5.5. Conclusão

Este capítulo apresenta os resultados obtidos através dos experimentos relacionados ao treinamento e à detecção de objetos. Com a utilização do protótipo desenvolvido, o usuário dispõe de um ambiente integrado para o treinamento de novas classes de objetos e suas respectivas detecções em imagens ou vídeos digitais.

O protótipo disponibiliza uma maneira flexível para que o usuário possa criar um conjunto de imagens positivas e negativas de acordo com a sua necessidade. Além disto, é possível testar o desempenho da cascata de

classificadores recém-criada para verificar se o seu desempenho em termos de taxa de acerto, taxa de falso positivo e taxa de não detecção são satisfatórios. Caso não sejam, o usuário pode reiniciar o treinamento com novos parâmetros.

Para realizar a detecção de objetos, é necessário apenas que o usuário informe os objetos são procurados, através das respectivas cascatas de classificadores, e que o usuário informe também uma imagem ou um vídeo onde o objeto deve ser procurado. Para cada classe de objeto encontrado na cena é desenhado um círculo em sua volta indicando sua localização na mesma.

Apresentamos também os resultados dos testes de desempenho obtidos pelo algoritmo proposto nesta tese em relação ao algoritmo original de Viola e Jones (2001). Tal comparação é possível uma vez que a biblioteca OpenCV (Intel 2006) implementa o método de Viola e Jones. Pelos dados apresentados, concluímos que o desempenho do nosso método apresenta uma redução no tempo de detecção de objetos, considerando o método de Viola e Jones, de até 80%, possibilitando desta forma o rastreamento de objetos em tempo real mesmo em vídeos de alta definição. Não foi encontrado na literatura trabalhos com foco na detecção de objetos em vídeos de alta definição, o que impede a criação de um *bechmarking* com outras abordagens existentes.

Entretanto, para vídeos na resolução de *Full HD* (1.920 x 1.080 linhas), o requisito de tempo real só é atingido ao se utilizar uma placa gráfica para realizar o processamento em paralelo de algumas etapas do algoritmo aqui proposto.

Também demonstramos o ganho obtido através da utilização de uma GPU para a realização do treinamento de novas classes de objetos. Enquanto Viola e Jones (2001) citam em seu trabalho que o treinamento da classe de objeto “face” demora dias para ser realizado, com uma base de 5.000 imagens positivas e 10.000 imagens negativas, conseguimos obter uma boa taxa de detecção com o algoritmo de aprendizado sendo executado em GPU, com a vantagem de levarmos algumas poucas horas para a execução do treinamento, ao invés de vários dias.

Com a execução do treinamento e a detecção de objetos realizados com o auxílio de uma GPU, conseguimos construir um artefato de software que apresenta um alto desempenho para realizar tais tarefas, algo que não poderia ser possível apenas com a utilização de uma CPU.

6
Conclusão e Trabalhos Futuros

Embora avanços na área de processamento de imagens e visão computacional venham possibilitando o reconhecimento e a deteção de objetos, fazê-los em tempo real em quaisquer condições ainda é um problema com inúmeros desafios. Enquanto grande parte das pesquisas focam na robustez do método, pouca atenção é dada ao caso de vídeos de alta definição. Neste trabalho apresentamos um método de detecção e rastreamento integrados, alcançando o requisito de tempo real para a detecção de objetos em vídeos de alta definição. Chegamos, nesta tese, à resolução Full HD de 1920 x 1080, enquanto que a literatura trabalha usualmente com 320 x 240. Entretanto, chegar a Full HD ainda está longe da resolução 2K do cinema (2048 x 1080). Neste particular aspecto, devemos lembrar que as TVs de resolução 4K (4096 x 2160), lançadas para a Copa do Mundo 2014, deverão se tornar populares nos próximos 5 a 10 anos.

O método proposto nesta tese pode facilitar o desenvolvimento de aplicações interativas sincronizadas com um vídeo de alta resolução em tempo real. Neste caso particular, os seguintes exemplos podem ser mencionados: publicidade interativa, e-commerce, ou mesmo a venda de produtos que são detectados durante as transmissões de programas de TV, como novelas, reality shows e filmes. Aplicações de storytelling interativo são um caso especial de aplicação do método proposto.

6.1.
Principais Contribuições

Em primeiro lugar, apresentamos a idéia de utilizar a imagem integral do plano de frente (*foreground*) para evitar a análise desnecessária de várias regiões de cada *frame* de um vídeo. Este processo de descarte revela uma segmentação adaptativa de cada quadro, que acaba por delimitar uma área reduzida para a detecção de objetos. Outra contribuição é expandir essas idéias para lidar com vários objetos em paralelo. Finalmente, nosso método apresenta alto desempenho em termos de tempo processamento sem perder as qualidades

apresentadas pelo método proposto por Viola e Jones (2001), como alta taxa acerto na detecção de objetos e baixo índice de falsos positivos.

Os experimentos apresentados (seção 5.4) revelaram que é possível reduzir substancialmente o tempo de detecção dependendo da resolução de vídeo e do tipo do movimento dos objetos. Portanto, o método proposto tem o potencial de evitar grandes quedas na taxa de quadros (FPS) quando utilizado um vídeo de alta resolução e na busca por mais de um objeto simultaneamente.

Nosso algoritmo pode alcançar taxas de quadro ainda mais elevadas, em vídeos de alta definição, se utilizarmos uma GPU para executar alguns passos do método proposto de forma paralela, como por exemplo o cálculo da imagem integral através do operador "*all-prefix-sums*" (seção 4.2.2). Esta operação (também conhecida como um *scan* paralelo) pode ser aplicada a todas as linhas de imagem seguido pela aplicação de mesma operação para todas as colunas do resultado. Além da utilização da GPU para a detecção de objetos, também foi utilizado o mesmo princípio para o treinamento de novas classes de objetos, paralelizando parte do algoritmo *AdaBoost*.

Desenvolvemos também um protótipo que permite ao usuário selecionar quais objetos devem ser procurados em um vídeo (através da seleção de uma cascata de classificadores previamente treinada). Além disto, o protótipo também permite que o usuário facilmente treine novas classes de objetos e analise a sua respectiva performance.

6.2. Trabalhos Futuros

A heurística proposta neste trabalho para o rápido descarte de regiões que não sofreram alterações entre dois *frames* consecutivos poderia ser revista quando o vídeo a ser processado estiver compactado através do padrão MPEG. Para a compressão de dados, o padrão MPEG define o conceito de *macroblocks,* que são blocos de 16 x 16 pixels. Um *macroblock* pode ser de três tipos: I (*intra*), P (*predicted*) e B (*bi-directionally-predicted*).

Um bloco do tipo I possui toda a sua informação codificada dentro dela mesma, não dependendo da informação temporal relacionada a *frames* anteriores ou posteriores. Um bloco do tipo P possui vetores de movimento, que indicam a diferença entre o bloco atual a ser descodificado e um bloco semelhante de um quadro anterior. Isto é chamado de compensação de

movimento, e pode acontecer em uma cena de vídeo onde muitos objetos permanecem parados enquanto outros se movem apenas uma curta distância. Um bloco do tipo B pode ter a predição de movimentos com relação a um *frame* para frente e para trás, ou apenas predição para um *frame* anterior. Conhecendo os tipos de *macroblocks* existentes, a informação de quais áreas efetivamente sofreram alterações entre dois *frames* consecutivos poderia ser extraída diretamente dos *macroblocks* do vídeo, evitando assim a necessidade de outro algoritmo para descobrir tal informação. Desta forma, o desempenho geral para a detecção de objetos poderia ser reduzido, entretanto a solução estaria restrita a vídeos compactados em MPEG. O trabalho aqui apresentado possui uma solução mais genérica neste sentido, funcionando em vídeos sem qualquer tipo de compressão de dados. Entretanto, seria interessante avaliar o custo-benefício deste tipo de solução que pode se beneficiar de informações temporais inerentes ao padrão MPEG.

Com relação à utilização de uma GPU para otimizar o desempenho tanto da etapa de detecção de objetos quanto da etapa de treinamento, dois pontos são interessantes de serem desenvolvidos.

O primeiro ponto é relacionado à detecção de objetos em vídeos. Para cada *frame* do vídeo, a imagem é enviada para a GPU para subsequente paralelização da segmentação do plano de frente e do cálculo da imagem integral. Como o envio de informações para a GPU é lenta, o ideal é que esta transferência constante de dados fosse evitada. Uma forma de solucionar este problema seria se a própria GPU pudesse “descompactar” o vídeo onde o objeto é procurado. Desta maneira, o *frame* já estaria disponível para processamento na própria placa gráfica e não teríamos o *overhead* de envio da imagem para a GPU. A arquitetura CUDA possui um decodificador de vídeo (NVIDIA 2013b) que seria capaz de realizar tal tarefa. Experimentos seriam necessários para comprovar o fato, e também para comprovar o ganho ao se evitar constantes envios de dados para a GPU.

O segundo ponto se refere ao treinamento de novas classes de objetos. Seria interessante analisar se paralelizar a etapa de treinamento dos vários estágios da cascata de classificadores seria útil, ou seja, se reduziria ainda mais o tempo despendido para treinar novos objetos.

Permitir o reconhecimento automático de qualquer objeto previamente treinado sem a necessidade de uma pessoa informar que objetos devem ser procurados seria de extrema importância. Isto tornaria o algoritmo de detecção mais “inteligente” e independente de informações adicionais. Uma possível

solução para este problema seria a utilização de cascatas que formassem uma hierarquia de padrões, similar ao trabalho apresentado por Epshtein e Ullman (2005). Com isto, formas geométricas básicas como triângulos e quadrados estariam nos níveis mais próximos à raiz das cascatas, enquanto formas mais particulares do objeto estariam nos níveis mais próximos às folhas.

Outra questão para trabalho futuro seria desenvolver um método que permitisse a detecção de objetos rotacionados pela cascata de classificadores. Desta forma, objetos que estejam em formas distintas das que foram originalmente treinadas poderiam ser detectados. Um passo nesta direção foi dado no trabalho realizado por Messom e Barczak (2006), em que os autores apresentam com sucesso a utilização de "características Haar" rotacionadas entre 0^0 e 90^0 para a detecção de objetos, porém com a desvantagem do aumento do custo computacional do algoritmo e a consequente perda de parte do desempenho. Sobre a possibilidade de detecção de objetos parcialmente oclusos também através de cascatas de classificadores, o trabalho realizado por Barczak (2004) considera a hipótese de que é possível atingir bons resultados de detecção treinando novas cascatas a partir de amostras do objeto parcialmente oclusos de forma aleatória. Entretanto os resultados apresentados não são muito promissores, sendo necessários mais estudos a respeito desta técnica.

Quanto á robustez do método há bastante espaço para futuras investigações sobre rastreamento de objetos em vídeos de alta resolução em condições adversas, tais como oclusões extensas e mudanças bruscas na iluminação, no movimento de câmera e na escala dos objetos. Neste caso, seria interessante comparar o método proposto com os resultados encontrados em bancos de comparação, tais como BoBoT (Klein 2010) e PROST (Grasz UT 2010).

Por fim, uma vez que o método aqui apresentado possui um baixo custo computacional, seria interessante desenvolver uma versão do algoritmo para utilização em dispositivos móveis, como celulares e *tablets*. Esta versão "móvel" do método poderia ter seu desempenho avaliado em tais dispositivos; e, caso o resultado dos experimentos seja satisfatório, ela poderia ser utilizada em larga escala em aplicativos de diversos fins, tais como: detecção de expressões faciais, reconhecimento de pontos turísticos, entre outros. Os principais problemas com os dispositivos móveis seriam com a placa gráfica e a adaptação do algoritmo à arquitetura destes dispositivos (que têm limitações de bateria, dissipação de calor, e custo mais alto de comunicação CPU-GPU).

Referências Bibliográficas

AVIDAN, S., 2004 “Support vector tracking,” Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 26, no. 8, pp. 1064–1072.

BLELLOCH, GUY E. 1990. "Prefix Sums and Their Applications." Technical Report CMU-CS-90-190, *School of Computer Science*, Carnegie Mellon University.

BARCZAK, ANDRE L. C. 2004. Evaluation of a Boosted Cascade of Haar-Like Features in the Presence of Partial Occlusions and Shadows for Real Time Face Detection. PRICAI, volume 3157 of Lecture Notes in Computer Science, page 969-970, Springer.

BARLOW, H. 1972. Single units and sensation: a neuron doctrine for perceptual psychology. *Perception,* 1, pp. 371–394.

BOVE, V. M., DAKSS, J., CHALOM, E., AND AGAMANOLIS, S. 2000. IBM Systems Journal, Vol. 39, Nos. 3 & 4, pp. 470-478.

CBCL, MIT, 2000. Face Database. Disponível em http://cbcl.mit.edu/software-datasets/FaceData2.html [Acessado em 01 de novembro de 2013].

CHEN T. P., BUDNIKOV D., HUGHES C. J., E CHEN Y.-K., 2007. “Computer vision on multi-core processors: Articulated body tracking,” *in Multimedia and Expo, 2007 IEEE International Conference on. IEEE*, pp. 1862–1865.

CHENG Y. 1995. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17 (8), pp. 790–799

CHEUNG, SEN-CHING S., E CHANDRIKA, KAMATH. 2004. "Robust techniques for background subtraction in urban traffic video", Proc. SPIE 5308, Visual Communications and Image Processing, 881 (January 7, 2004);

COMANICIU, D. E MEER, P. 2002. “Mean shift: A robust approach toward feature space analysis,” *Pattern Analysis and Machine Intelligence, IEEE Transactions* on, vol. 24, no. 5, pp. 603–619.

COX, I. E HINGORANI, S. 1996. An efficient implementation of reid’s multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18 (2), pp. 138–150

CROW, F. 1984. Summed-area tables for texture mapping. *In Proceedings of SIGGRAPH*, volume 18(3), pages 207–212.

DOTSENKO, Y., GOVINDARAJU, N.K., SLOAN, P-P., BOYD, C. & MANFERDELLI, J. 2008. Fast scan algorithms on graphics processors. ACM ICS'08, June 7-12, Grécia, p. 205-213.

DOTSENKO, Y., GOVINDARAJU, N.K., BOYD, C., MANFERDELLI, J. & SLOAN, P-P. 2010. Matrix-based scans on parallel processors. Patent US 2010/0076941, Assignee: Microsoft, Mar. 25, 2010.

EPSHTEIN, B. AND ULLMAN, S. 2005. Identifying Semantically Equivalent Object Fragments. *In Computer Vision and Pattern Recognition*, IEEE Computer Society Conference on Volume 1, Issue , 20-25 June. Page(s): 2 - 9 vol. 1.

FASHING, M. E TOMASI, C., 2005. Mean shift is a bound optimization. *IEEE Trans. Pattern Anal. Mach. Intell*., 27 (3), pp. 471–474.

FREUND, Y. AND SCHAPIRE, R.E. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. *In: Computational Learning Theory*, Springer-Verlag, pp. 23–37.

GABRIEL, P. F., VERLY, J. G., PIATER, J. H., E GENON, A., 2003. "The state of the art in multiple object tracking under occlusion in video sequences," *in Advanced Concepts for Intelligent Vision Systems*. Citeseer, pp. 166–173.

GAME OF THRONES, 2013. *Série de TV*, rede de televisão por assinatura HBO, EUA.

GARCIA, G. M. 2012. BoBoT-D Benchmark, Univ. of Bonn, available at: http://www.iai.uni-bonn.de/~martin/tracking.html [acessado 16 Dez 13].

GAVRILA, D. E DAVIS, L. 1996. 3D model-based tracking of humans in action: a multi-view approach, in: *Proceedings of the Computer Vision and Pattern Recognition,* pp. 73-80.

GOLDPOCKET INTERACTIVE. 2006. Method and apparatus for receiving a hyperlinked television broadcast. US 7120924 B1 Patent, Inventors: Katcher, D., Bove, V.M., Sarachik, K., Milazzo, P. and Dakss, J., Publication date: Oct 10, 2006.

GPGPU.ORG. 2013. CUDPP – CUDA Data Parallel Primitives Library. Disponível em http://gpgpu.org/developer/cudpp [acessado em 14/07/2013].

GRABNER, H., LEISTNER, C. E BISCHOF, H. 2008. Semi-supervised on-line boosting for robust tracking. In: D. Forsyth, P. Torr, and A. Zisserman (Eds.): ECCV 2008, Part I, LNCS 5302, pp. 234-247.

GRAZ UT. 2010. PROST – Parallel Robust Online Simple Tracking, Graz Technical University, available at: http://gpu4vision.icg.tugraz.at/index.php?content=subsites/prost/prost.php [acessado 16 Dez 2013].

GREENGARD, L. E STRAIN J., 1991. The fast Gauss transform. *SIAM J. Sci. Statist. Comput.*, 12(1):79–94.

HAAR, A., 1910. Zur Theorie der orthogonalen Funktionensysteme. In *Mathematische Annalen*, 69, pp 331-371.

HALL, D., NASCIMENTO, J., RIBEIRO, P., ANDRADE, E., MORENO, P. PESNEL, S., LIST T., EMONET, R., FISHER, R.B., SANTOS VICTOR, J. AND CROWLEY, J.L. 2005. Comparison of target detection algorithms using adaptive background models. *In: Proc. 2nd Joint IEEE Int. Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, Beijing, October. pp 113-120.

HARRIS, C. 1993. Geometry from Visual Motion in Active Vision. *MIT Press, Cambridge, MA, USA*.

HARRIS, M., SENGUPTA, S. & OWENS, J.D. 2007. Parallel Prefix Sum (Scan) with CUDA. In: H. Nguyen (Ed.), GPU Gems 3, Addison-Wesley, Chapter 39, p. 851-876.

HORN, DANIEL. 2005. "Stream Reduction Operations for GPGPU Applications." In *GPU Gems 2*, edited by Matt Pharr, pp. 573–589. Addison-Wesley.

HASSANPOUR H., SEDIGHI M., MANASHTY A. R., 2011. Video Frame's Background Modeling: Reviewing the Techniques. In: Journal of Signal and Information Processing, 2011, 2, 72-78.

INTEL, 2006. *OpenCV*. (1.0) [computer vision library]. Intel Corporation. Disponível em: http://sourceforge.net/projects/opencvlibrary. [Acessado 15 Jan 2008].

INTEL, 2006b. *OpenCV*. [Developer Documentation]. Intel Corporation. Disponível em: docs.opencv.org/trunk/doc/py_tutorials/py_video/ py_lucas_kanade/ py_lucas_kanade.html. [Acessado 02 Fev 2014].

KANDEL, E., SCHWARTZ,J. E JESSELL,T. 2000. Principles of Neural Science. McGraw-Hill Medical, 4th Ed.

KLEIN, D., SCHULZ, D., FRINTROP, S. E CREMERS, A. B. 2010. Adaptive Real-Time Video-Tracking for Arbitrary Objects. In IEEE Int. Conf. on Intelligent Robots and Systems (IROS), Oct 2010, pp. 772-777.

KLEIN, D.A. 2010. BoBoT – Bonn Benchmark on Tracking, Univ. of Bonn, Disponível em: http://www.iai.uni-bonn.de/~kleind/tracking [acessado 16 Dez 13].

KLEIN, D. A. E CREMERS, A. B. 2011. Boosting Scalable Gradient Features for Adaptive Real-Time Tracking. In: IEEE Int. Conf. on Robotics and Automation (ICRA 2011), May 2011, pp. 4411-4416.

LEHUGER A., LECHAT P., E PEREZ P., 2006. “An adaptive mixture color model for robust visual tracking,” *in Image Processing, 2006 IEEE International Conference* on. IEEE, pp. 573–576.

LIENHART, R. AND MAYDT, J. 2002. An Extended Set of Haar-like Features for Rapid Object Detection. *In IEEE ICIP*, Vol. 1, pp. 900-903, September.

LIENHART, R., KURANOV, A. AND PISAREVSKY, V. 2003. Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection, *DAGM'03, 25th Pattern Recognition Symposium*, Madgeburg, Germany, pp. 297-304, September.

LIENHART, R., LIANG, L. AND KURANOV A.. 2003. A Detector Tree of Boosted Classifiers for Real-time Object Detection and Tracking, *IEEE International Congress on Mathematical Education*, Vol. 2, pp. 277-280, July.

LOWE, D., 1992. Robust model-based motion tracking through the integration of search and estimation Int. *Journal Comput. Vision*, 8, pp. 113–122

LUCAS, B. E KANADE, T., 1981. An iterative image registration technique with an application to stereo vision, in: *International Joint Conference on Artificial Intelligence*, pp. 674–679.

MACCORMICK, J. E ISARD, M., 2000. Partitioned sampling, articulated objects, and interface-quality hand tracking, in: *Proceedings of the European Conference on Computer Vision*, pp. 390–395.

MCGUIRE, M. 2004. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. EUA: Addison-Wesley Professional.

MCIVOR, ALAN. 2000. *Background subtraction techniques*. s.l.: Proceedings of Image and Vision Computing.

MESSOM, C.H. AND BARCZAK, A.L.C., 2006, Fast and Efficient Rotated Haar-like Features Using Rotated Integral Images, *Australian Conference on Robotics and Automation*, pp. 1-6.

NEW LINE CINEMA, 2003. The Lord of the Rings: The Return of the King.

NGUYEN, H. 2007. *GPU Gems 3*. EUA: Addison-Wesley Professional.

NUMMIARO K., KOLLER-MEIER E., E GOOL L. V., 2002, “A color-based particle filter,” *in First International Workshop on Generative Model Based Vision*, pp. 53–60.

NVIDIA, 2007. *CUDA Compute Unified Device Architecture Programming Guide.* Disponível em http://moss.csc.ncsu.edu/~mueller/cluster/nvidia/0.8/NVIDIA_CUDA_Programming_Guide_0.8.2.pdf [Acessado em 10 de dezembro de 2013].

NVIDIA, 2013. *CUDA C Programming Guide*. Disponível em http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html [Acessado em 20 de outubro de 2013].

NVIDIA, 2013b. *CUDA Video Decoder*. Disponível em http://docs.nvidia.com/cuda/video-decoder/index.html [Acessado em 16 de dezembro de 2013].

NVIDIA, 2014. *O que é CUDA – Histórico*. Disponível em: http://www.nvidia.com.br/object/cuda_home_new_br.html [Acessado em 5 de Janeiro de 2014].

PICCARDI, M. 2004. Background subtraction technique: a review, *In: Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics*, Vol. 4, pp. 3099-3104.

PORIKLI F. E TUZEL O., 2003. "Human body tracking by adaptive background models and mean-shift analysis," *in IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*.

RABINER, L., 1989. A tutorial on hidden markov models and selected applications in speech recognition*. Proc. IEEE*, 77, pp. 257–286

REID, D., 1979. An algorithm for tracking multiple targets. *IEEE Trans. Auto. Control*, 24 (6), pp. 843–854

SENGUPTA, S., HARRIS, M., ZHANG, Y. AND OWENS, J. D. 2007. Scan primitives for GPU computing. In: Proc. Graphics Hardware 2007, August 2007, San Diego, CA, pp. 97-106.

SHI, J. E TOMASI, C., 1994. Good features to track, in: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 593–600.

SCHUNCK, B. E HORN, B. 1981. Determining optical flow. In: *DARPA81*, pp. 144-156.

SCHUNCK, B. 1986. The image flow constraint equation. *Comput. Vis. Graph. Image Process.*, 35 (1), pp. 20–46

STAA, ARNDT VON. 2000. *Programação Modular*. Rio de Janeiro, Brasil: Campus, 690p.

STALDER, S., GRABNER, H. AND VAN GOOL, L. 2009. eyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition. In: IEEE 12th Int Conf on Computer Vision Workshops (ICCV 2009), Sep 27- Oct 4 2009, pp. 1409-1416.

SULLIVAN, J., BLAKE, A., ISARD, M., MACCORMICK, J., 1999. Object localization by bayesian correlation, in: *Proceedings of the Seventh International Conference on Computer Vision*, pp. 1068–1075.

TECGRAF, 1994. *IUP – Portable User Interface*. (2.5) [computer interface library]. Tecgraf/PUC-Rio. Disponível em http://www.tecgraf.puc-rio.br/iup. [Acessado 15 Nov 2012].

TRESADERN, P., IONITA, M. E COOTES, T., 2012. “Real-time facial feature tracking on a mobile device,” *International Journal of Computer Vision*, vol. 96, no. 3, pp. 280–289.

ULLMAN, S., 1979. The Interpretation of Visual Motion. *MIT Press, Cambridge,* MA.

VIOLA, P. E JONES, M. 2001. Rapid object detection using a boosted cascade of simple features. *In IEEE Conference on Computer Vision and Pattern Recognition*.

VIOLA, P. E JONES, M. 2003. Fast Multi-view Face Detection, *Mitsubishi Electric Research Laborotories, TR2003-96*, July.

VIOLA, P., JONES, M.J E SNOW, D., 2003. “Detecting pedestrians using patterns of motion and appearance,” in Computer Vision, 2003. *Proceedings. Ninth IEEE International Conference on. IEEE*, pp. 734–741.

WREN, C., AZARBAYEJANI, A., DARRELL, T., AND PENTLAND, A. 1997. PFinder: real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 19, no. 7, pp. 780-785.

WUNSCH, P. E HIRZINGER, G., 1997. Real-time visual tracking of 3D objects with dynamic handling of occlusion, in: *Proceedings of 97 International Conference on Robotics and Automation*, pp. 2868–2879.

YANG, C., DURAISWAMI, R. E DAVIS, L., 2005. Efficient mean-shift tracking via a new similarity measure, in: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, San Diego, pp. 176–183.

YILMAZ, A., JAVED, O. E SHAH, M., 2006. “Object tracking: A survey,” *Acm Computing Surveys (CSUR)*, vol. 38, no. 4, p. 13.