UNIVERSIDADE ESTADUAL PAULISTA

Faculdade de Ciências - Bauru

Bacharelado em Ciência da Computação

Henrique Ferraz de Arruda

# PARALELIZAÇÃO DE ALGORITMOS DE TRIMAP UTILIZANDO A ARQUITETURA CUDA

UNESP

2012

Henrique Ferraz de Arruda

# PARALELIZAÇÃO DE ALGORITMOS DE TRIMAP UTILIZANDO A ARQUITETURA CUDA

Orientador: Prof. Adj. Antônio Carlos Sementille

Co-orientador: Prof. Adj. João Fernando Marar

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, campus de Bauru, como parte do Trabalho de Conclusão de Curso.

UNESP

2012

Henrique Ferraz de Arruda

# PARALELIZAÇÃO DE ALGORITMOS DE TRIMAP UTILIZANDO A ARQUITETURA CUDA

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, campus de Bauru, como parte do Trabalho de Conclusão de Curso.

BANCA EXAMINADORA

Prof. Adj. Antônio Carlos Sementille
Professor Adjunto
DCo - FC - UNESP - Bauru
Orientador

Prof. Dr. Eduardo Martins Morgado
Professor Doutor
DCo - FC - UNESP - Bauru
UNESP - Bauru

Profa. Dra. Simone das Graças Domingues Prado
Professora Doutora
DCo - FC - UNESP - Bauru
UNESP - Bauru

Bauru, 29 de Novembro de 2012.

## AGRADECIMENTOS

Agradeço primeiramente a Deus, por permitir que eu tenha condições de estudar. Aos meus pais que sempre me incentivaram e financiaram nos meus estudos. Ao meu irmão, Guilherme, pelo incentivo e por sempre me auxiliar na parte matemática. Ao meu orientador Prof. Adj. Antonio Carlos Sementille, por sempre me auxiliar com o trabalho e me incentivar a cada vez mais procurar melhores resultados. Ao Prof. Dr. João Fernando Marar, por ceder a utilização do laboratório SACI (Sistemas Adaptativos e Computação Inteligente). Ao meu amigo Daniel (Zuna), pela ajuda com o projeto e pelas constantes cobranças quanto a melhores resultados. Ao Júnior, pelo auxilio, tanto em equipamentos, quanto em colaboração com o trabalho. Ao Gustavo (Lampadinha), pelo incentivo e principalmente por servir de modelo para os meus testes. Ao Carlos e o André, pela colaboração e pelo incentivo. À Caroline (Carol), por sempre me incentivar a estudar. Por fim agradeço também a todos os professores que colaboraram com a minha formação.

"A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original."

Albert Einstein

# Resumo

Técnicas de combinações de imagens, com extração de objetos para montar uma cena final, são muito utilizadas em aplicações que vão desde montagens em fotos até produções cinematográficas. Estas técnicas são chamadas de *matting* digital. Com elas é possível diminuir o custo das produções, pois não é necessário que o ator seja filmado no local onde a cena final deveria acontecer. Esta característica também favorece sua utilização em programas realizados para a televisão digital, que demanda uma alta qualidade da imagem.

Muitos algoritmos de *matting* digital utilizam marcações feitas nas imagens, para demarcar o que é o plano da frente, plano de fundo e as áreas de incerteza. Esta marcação é chamada de *trimap*, que é um mapa triplo contendo estas três informações. O *trimap* é feito, geralmente, a partir de marcações manuais.

Neste projeto foram criados métodos que possam ser utilizados em algoritmos de *matting* digital, com restrição de tempo e sem a interação humana, ou seja, a criação de um algoritmo que gera automaticamente o *trimap*. Este que pode ser gerado a partir da diferença entre a cor do plano de um fundo arbitrário e o plano da frente, ou pela utilização de um mapa de profundidade. Foi criado ainda um método de *matting*, baseado no *Geodesic Matting* (BAI; SAPIRO, 2009), que possui um tempo de processamento inferior ao original.

Com o intuito de melhorar o desempenho das aplicações de geração do *trimap* e dos algoritmos geradores do *alphamap* (mapa que associa um valor de transparência a cada pixel da imagem), possibilitando o seu uso em aplicações com restrição de tempo, foi utilizada a arquitetura CUDA. Aproveitando, assim, o poder computacional e as características da GPGPU, que é massivamente paralela.

**Palavras-chave:** *Matting Digital*, CUDA, *Trimap*.

# Abstract

Techniques of image combination, with extraction of objects to set a final scene, are very used in applications from photos montages to cinematographic productions. These techniques are called digital matting. With them is possible to decrease the cost of productions, because it is not necessary for the actor to be filmed in the location where the final scene occurs. This feature also favors its use in programs made to digital television, which demands a high quality image.

Many digital matting algorithms use markings done on the images, to demarcate what is the foreground, the background and the uncertainty areas. This marking is called trimap, which is a triple map containing these three informations. The trimap is done, typically, from manual markings.

In this project, methods were created that can be used in digital matting algorithms, with restriction of time and without human interaction, that is, the creation of an algorithm that generates the trimap automatically. This last one can be generated from the difference between a color of an arbitrary background and the foreground, or by using a depth map. It was also created a matting method, based on the Geodesic Matting (BAI; SAPIRO, 2009), which has an inferior processing time then the original one.

Aiming to improve the performance of the applications that generates the trimap and of the algorithms that generates the alphamap (map that associates a value to the transparency of each pixel of the image), allowing its use in applications with time restrictions, it was used the CUDA architecture. Taking advantage, this way, of the computational power and the features of the GPGPU, which is massively parallel.

**Keywords:** *Matting Digital*, CUDA, *Trimap*.

# Lista de Figuras

# Lista de Tabelas

# Sumário

# Capítulo 1

# Introdução

## 1.1 Contextualização

O "*matting* digital" é uma técnica para combinação de imagens digitais, onde há a extração dos objetos em primeiro plano de imagens fixas ou uma sequência de imagens (vídeo). Essa técnica tem sido amplamente estudada há mais de 20 anos.

A extração consiste em separar os objetos em primeiro plano do plano de fundo, considerando o nível de transparência dos objetos. Para esta finalidade foi desenvolvido o elemento denominado canal alfa ($\alpha$) (PORTER; DUFF, 1984). Esse elemento é utilizado para representar níveis de transparência dos pixeis da imagem em primeiro plano, possibilitando a extração dos elementos de interesse e ainda a preservação da transparência dos objetos. Em termos matemáticos, o canal alfa possibilita controlar a interpolação linear entre as cores das duas camadas (primeiro plano e o fundo) para cada pixel. O canal alfa contém valores que variam em um canal em escala de cinza (PORTER; DUFF, 1984). Aos pixeis com alta transparência, os quais não devem ser exibidos, deve ser atribuido o valor alfa igual a zero, em contrapartida o valor máximo deve ser atribuido aos pixeis opacos, os quais devem ser preservados. Os pixeis com algum nível de transparência devem conter valores intermediários de alfa.

Muitas das imagens de um vídeo, das cenas de filmes ou até mesmo de transmissões de televisão, podem ser obtidas por meio da composição de duas imagens (camadas), uma contendo os objetos em primeiro plano (*foreground*) e a outra o plano de fundo (*background*). Tanto a indústria cinematográfica quanto a de televisão, dispõem de recursos para utilizar a camada em primeiro plano isoladamente, o que gerou a necessidade de separar os objetos em primeiro plano da imagem original.

Esta separação, conhecida como extração de chave, objetiva uma combinação futura dos objetos extraídos em primeiro plano com uma nova camada de plano de fundo. Assim esta técnica é extremamente utilizada em filmes, seriados e programas de TV entre outros, com a finalidade de simular fundos, e consequentemente economizar dinheiro e tempo nas produções. Diversos trabalhos no âmbito da extração de chave são protegidos por patentes (VLAHOS, 1958).

A técnica do *chromakey*, há algum tempo vem sendo utilizada em produções de cinema e televisão, inclusive transmissões ao vivo. Seu princípio baseia-se no reconhecimento de partes da

imagem, em que se encontra uma determinada cor, denominada cor chave. Uma máscara é então gerada para selecionar quais pixeis devem permanecer e quais devem ser substituídos por pixeis de outra imagem, utilizada como plano de fundo (BERGH; LALIOTI, 1999).

O grau de complexidade do problema é evidente quando, por exemplo, todo o processo deve ser realizado em tempo real, se existirem sombras a serem preservadas ou retiradas, quando a superfície do objeto a ser extraido tem alguma transparência, ou ainda, quando na silhueta do objeto em primeiro plano houver elementos como pêlos e cabelos.

Conceitualmente, a operação de *chromakey* é simples, mas apenas a comparação dos pixeis da imagem de entrada com a cor chave, selecionada automaticamente ou por meio do operador, pode conduzir a composições grosseiras e pouco naturais.

Abordagens muito simplificadas produzem uma série de imperfeições e, por esse motivo, os sistemas modernos que utilizam o *chromakey* aplicam algoritmos avançados para preservar detalhes do objeto em primeiro plano a ser isolado. Alguns problemas em potencial são comuns a técnica, caso nenhum controle adicional seja implementado ao sistema. Podem-se observar imperfeições nas bordas ao redor dos objetos extraídos do primeiro plano, sombras sobre o plano de fundo podem ser reconhecidas como elementos em primeiro plano, e a câmera de vídeo deve permanecer imóvel durante o processo para manter a integridade entre as camadas de primeiro e segundo plano (SMITH; BLINN, 1996).

Diversos algoritmos encontrados na literatura fazem uso de um recurso conhecido como *trimap*, para a obtenção de mapas de transparência de maior qualidade. *Trimap* é uma marcação feita na imagem que mostra para cada pixel a que se refere, entre plano da frente, plano de fundo e a região desconhecida, ver Figura 1. "O *trimap* reduz a dimensão do espaço de solução do problema *matting*, o que leva os algoritmos de *matting* a gerar os resultados desejados pelo utilizador" (WANG; COHEN, 2007b).


Figura 1: Exemplo de *trimap.* Fonte: (WANG; COHEN, 2007b).

A aplicação de algoritmos avançados para a extração e o *matting* de imagens/vídeo em tempo real demanda, naturalmente, grande poder de processamento, especialmente em imagens de alta resolução. Para fornecer esta capacidade de processamento, diversas soluções têm sido experimentadas, do ponto de vista do hardware, entre as quais podem ser destacadas: várias placas com múltiplos processadores, arquiteturas com DSPs (*Digital Signal Processor*) e FPGAs (*Field Programmable Gate Array*), e processadores de propósito geral (GPPs – *General-Purpose Processors*).

Além desses sistemas, uma tendência recente na comunidade científica tem sido a de aproveitar o poder de computação paralela massiva das Unidades de Processamento Gráfico (*Graphics Pro-*

*cessing Units* - GPUs), encontradas na maioria dos PCs modernos e computadores portáteis para executar computação intensiva exigida pelos algoritmos de processamento de vídeo (OWENS et al., 2007). Considerando esta última tendência, merece destaque a arquitetura de computação paralela de propósito geral desenvolvida pela NVIDIA, denominada CUDA (*Compute Unified Device Architecture*). A CUDA foi especialmente criada para aproveitar o mecanismo de processamento paralelo das GPUs NVIDIA a fim resolver problemas computacionais complexos. A arquitetura inclui o conjunto de instruções CUDA ISA (*Instruction Set Architecture*) e o mecanismo de computação paralela na GPU. Para programar segundo a arquitetura CUDA™, os desenvolvedores hoje em dia podem usar a linguagem C (CHENG, 2010).

## 1.2 Objetivos do Trabalho

- Objetivo Geral
  - Obter implementações paralelizadas, baseadas na arquitetura CUDA, de técnicas de geração automáticas de *trimaps*, visando a sua utilização em aplicações que envolvam *matting* digital com restrições de tempo.
- Objetivos específicos
  - Analisar técnicas de *matting*, com informações extras;
  - Analisar as funcionalidades do Kinect pertinentes ao projeto;
  - Investigar a utilização da arquitetura CUDA na solução da implementação das técnicas de *matting* digital de imagens e vídeo;
  - Implementar as técnicas de geração automática de *trimaps*, segundo a abordagem sequencial (para processadores de uso geral) e paralelizada (para a arquitetura CUDA).
  - Comparar o desempenho das implementações para a arquitetura CUDA com as implementações para processadores de uso geral.

## 1.3 Organização do Trabalho

Os demais capítulos deste trabalho foram organizados da seguinte forma:

- O Capítulo 2 apresenta os principais conceitos utilizados para o desenvolvimento das técnicas de *matting* digital abordadas neste trabalho.
- O Capítulo 3 apresenta os principais conceitos sobre a arquitetura CUDA.
- O Capítulo 4 descreve os componentes de hardware e software utilizados no trabalho, bem como, os métodos desenvolvidos.
- O Capítulo 5 apresenta quais foram as formas de implementação, tanto na implementação sequencial, quanto na implementação paralela e suas principais características.

- O Capítulo 6 descreve os experimentos, os resultados obtidos durante a realização do projeto e a análise dos mesmos.
- O Capítulo 7 apresenta as conclusões do projeto, discutindo e comentando sobre os resultados obtidos e mostrando as possibilidades de trabalhos futuros.

# Capítulo 2

# Técnicas de *Matting* Digital

Este capítulo descreve as principais técnicas de *matting* digital existentes, as quais podem, ou não envolver a utilização de informações extras e *trimaps*.

## 2.1 Considerações Iniciais

Atualmente estão sendo muito utilizadas técnicas de visão computacional para fazer montagens em imagens e vídeos, com a finalidade de reduzir os custos das produções televisivas e cinematográficas. Uma técnica muito utilizada é o *matting* digital. Por meio desta técnica é possível realizar a combinação de duas imagens para formar a cena, extraindo o plano da frente e colocando um novo plano de fundo.

Uma abordagem muito utilizada é o *trimap*, a qual consiste, basicamente, em uma demarcação na imagem dos pontos de certeza, que são *background* e *foreground*, e pontos de incerteza, estes que devem ser classificados posteriormente pelo algoritmo de *matting*. Na maior parte dos casos o *trimap* é feito manualmente, no caso de vídeos com a marcações em quadros chaves.

## 2.2 Canal alfa

Para que sejam mantidas as transparências da superfície, é utilizado o canal alfa, o qual representa o nível de transparência de cada pixel da imagem. Representa, portanto, a interpolação linear de cores entre as duas camadas (plano de fundo e plano da frente) e varia entre 0 e 1 (PORTER; DUFF, 1984). A utilização do canal alfa proporciona a criação de imagens digitais com alto grau de complexidade e, por esse motivo, a maior parte dos métodos de composição é baseada na "Equação da Composição" (PORTER; DUFF, 1984) (Equação 2.2.1).

$$C = \alpha F + (1–\alpha)B \tag{2.2.1}$$

onde:

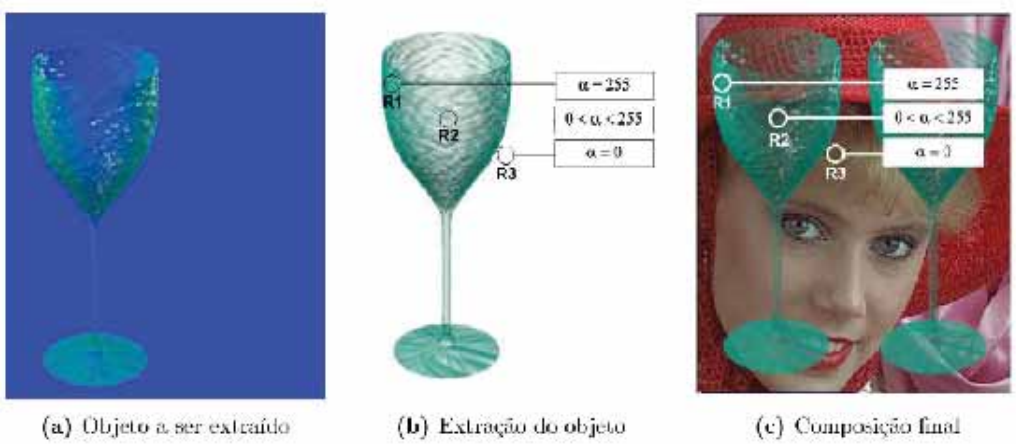
C é a cor resultante representada na composição;

F é a cor do elemento em primeiro plano (foreground);

B é a cor do novo plano de fundo (background);

e $\alpha$ é a componente de transparência do pixel.

Com uma imagem de entrada, são conhecidos os vetores tridimensionais de cor para cada pixel, porém são desconhecidos os valores das variáveis $\alpha$, B e F. A maioria dos métodos de composição de imagens depende de conhecimentos prévios para restringir o problema, possibilitando uma boa estimativa dessas variáveis. Uma vez estimadas corretamente essas variáveis, o primeiro plano pode ser suavemente integrado ao um novo plano de fundo, por meio da substituição do fundo original B por B' da nova imagem de plano de fundo na Equação 1. O valor de alfa computacionalmente é armazenado em uma variável de 8 bits, dessa forma é necessário fazer uma mudança de escala, na qual o valor máximo passa a ser 255 e deixa de ser real. como mostrado na Figura 2.




(a) Objeto a ser extraído (b) Extração do objeto (c) Composição final

Figura 2: Valores de alfa sobre o elemento em primeiro plano. Fonte: (SMITH; BLINN, 1996)

## 2.3 *Matting* sem *trimap*

Um algoritmo muito simples foi proposto por Van den Bergh & Lalioti (BERGH; LALIOTI, 1999). É um método que faz análise da imagem a partir das cores dos pixeis, em que cada um é analisado separadamente. Para que o algoritmo fique mais eficiente é realizada uma simplificação do cálculo de distância euclidiana (Equação 2.3.1), em que as componentes das cores são representadas pelos eixos no sistema cartesiano, como mostrado na Figura 3.

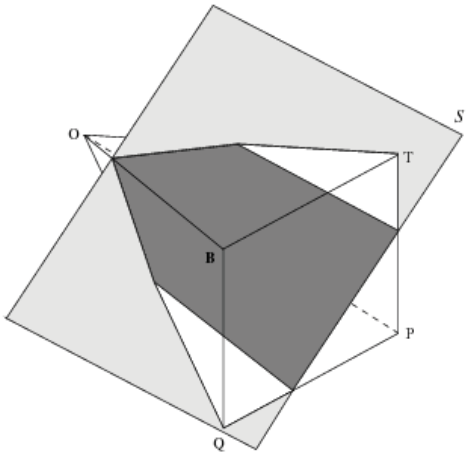



Figura 3: Cubo RGB com a intersecção do plano S com a pirâmide de inclinação OBTPQ. Fonte: (BERGH; LALIOTI, 1999).

$$D = B - R - 2G \tag{2.3.1}$$

Onde: D caracteriza a proximidade entre a cor do pixel com o verde;

B componente azul do pixel;

R componente vermelho do pixel;

G componente verde do pixel (cor do background).

O plano de fundo deve ser controlado e possuir uma cor chave (que deve ser verde ou azul), permitindo a troca ou não dos pixeis. Para todos os pixeis da imagem é calculada ama aproximação de equação da distância, a partir da Equação 1. O resultado encontrado é comparado com um valor de distância pré-definido. Se o valor do resultado for superior ao valor pré-definido, o pixel é trocado pelo pixel novo, pois a distância é maior, de forma contrária não ocorre troca.

O plano de fundo é controlado e possui a cor verde, facilitando assim a troca ou não dos pixeis. Para todos os pixeis da imagem é calculada a distância, a partir da Equação 2.3.1, o valor encontrado é comparado com um valor de distância predefinido, gerando assim uma escala aceitável da cor verde, se o valor for superior o pixel é trocado pelo novo (pixel do novo plano de fundo), de forma contrária não ocorre troca.

Assim é possível isolar os objetos do primeiro plano do plano de fundo, sem o uso do *trimap*, com um desempenho muito grande, devido a simplicidade do algoritmo. Entretanto no objeto a ser recortado não pode haver tons de verde.

Este algoritmo pode ter um resultado não muito natural, pois como é binário não considera níveis de transparência, tendo sempre o valor de $\alpha$ igual a zero ou um. Entretanto possui a vantagem de poder ter um tempo de execução menor.

## 2.4 *Geodesic Matting*

Segundo (OWENS et al., 2007) o *Geodesic Matting* é um método de *matting* digital que demanda menos tempo de processamento que a maioria dos métodos. Ele possui como entrada marcações feitas pelo usuário (chamadas de rabiscos), a partir delas são criados mapas tridimensionais, chamados de mapas de cor, que são semelhantes ao mapa de um relevo. Estes, por sua vez, são utilizados para calcular a distância geodésica dos pixeis desconhecidos para pixeis conhecidos (pontos que estão contidos nos rabiscos). A partir desta distância é estimado um valor para $\alpha$, como mostra a Figura 4 (BAI; SAPIRO, 2009).

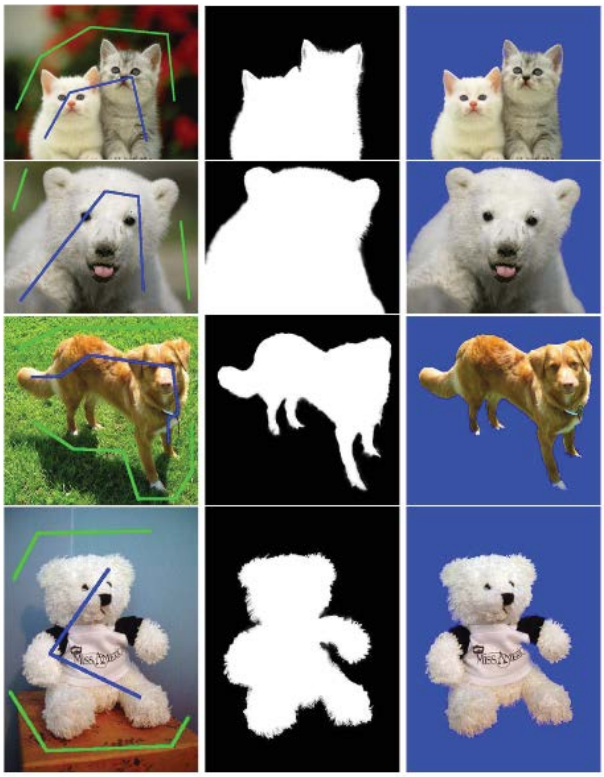


Figura 4: "Exemplos do Geodesic Matting. Coluna da esquerda: imagens originais com fornecidos pelo usuário com rabiscos. Azul para o primeiro plano e verde para o fundo. Coluna do meio: computado o alfa. Coluna da direita: Foregrounds colados em fundos azuis (azuis (constante) fundos são selecionados, uma vez que muitas vezes permitem inspeção muito mais cuidadosa dos resultados de colar em fundos desordenado)". Fonte: (BAI; SAPIRO, 2009).

### 2.4.1 KDE

Segundo (BOWMAN; AZZALINI, 2003) “O conceito de função densidade probabilidade é a ideia central na estatística. O seu papel na modelagem estatística é para encapsular o padrão de variação randômico em dados em que não é explicado por outros termos estruturais no modelo”. O *Kernel Density Estimation* (KDE) estima um gráfico de f(x) a partir de dados amostrais e um *kernel* pré-

definido, na Figura 5 (pode ser utilizado o *kernel* gaussiano). A f(x) gerada é uma função densidade probabilidade, chamade de PDF (*Probability Density Function*).

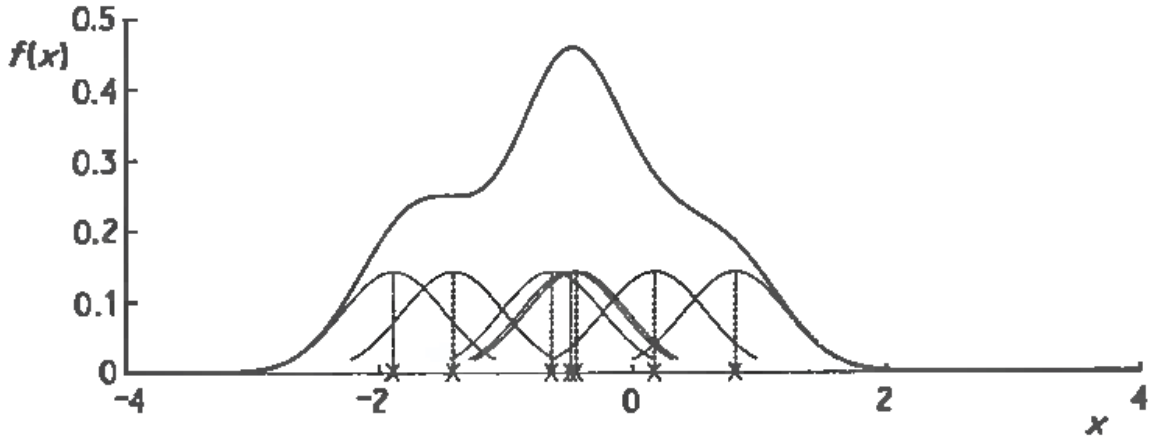



Figura 5: Gráfico maior mostrando a f(x) estimada e os menores mostrando os kernels individuais. Fonte: (SILVERMAN, 1986).

Segundo (BOWMAN; AZZALINI, 2003) função f(x) é estimada a partir da Equação 2.4.1:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) \tag{2.4.1}$$

onde $K_h$ é o *kernel*, n é a quantidade de amostras, x é o ponto da f que está sendo calculado e $x_i$ são os valores das amostras. O parâmetro h controla a variância da função do kernel segundo (SILVERMAN, 1986) é calculado como mostra a Equação 2.4.2:

$$h = \left( \frac{4\hat{\sigma}^5}{3n} \right)^{\frac{1}{5}} \approx 1,06\hat{\sigma}n^{-\frac{1}{5}} \tag{2.4.2}$$

### 2.4.2 Implementação do método

A distância geodésica é o menor caminho entre dois pontos, levando em consideração o relevo, ou seja são consideradas também as deformidades do relevo, como mostra a Figura 6.


Figura 6: A e B são os pontos, a distância desenhada em azul é a distância euclidiana e a distância desenhada em vermelho é a distância geodésica.

A implementação do algoritmo é representada pela Figura 7.

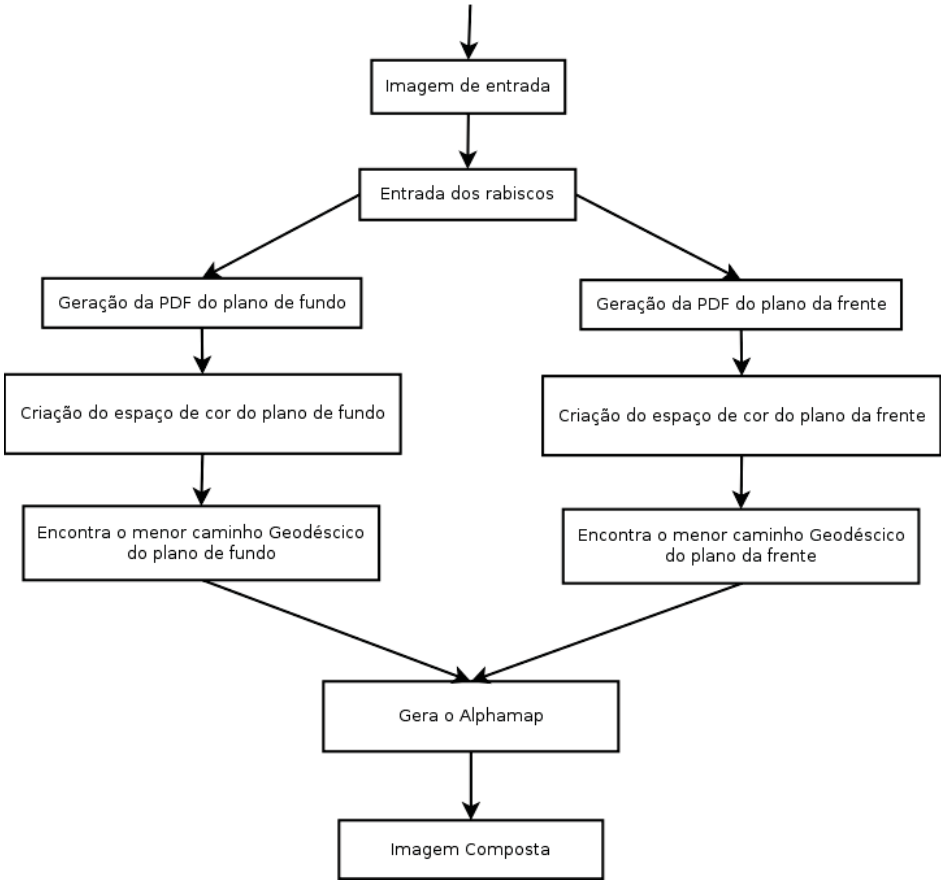



Figura 7: Fluxograma do algoritmo *Geodesic Matting*.

Os espaços de cor gerados pelo algoritmo são feitos a partir do KDE e representam um relevo, para que seja encontrada a distância geodésica. A Equação 2.4.3 é utilizada para gerar o espaço de cor do plano da frente:

$$P_F = \frac{P_r(x|F)}{P_r(x|F) + P_r(x|B)} \tag{2.4.3}$$

em que Pr(x|F) é a PDF do plano da frente Pr(x|B) é a PDF do plano de fundo. O mesmo processo ocorre para o plano de fundo (BAI; SAPIRO, 2009).

As distâncias geodésicas são geradas a partir da Equação 2.4.4:

$$D_l(x) = \min_{s \in \Omega} d(s,x),\ l \in \{F,B\}, \tag{2.4.4}$$

onde d(s,x) é calculado a partir da Equação 2.4.5:

$$d(s_1,s_2) = \min_{C_{s_1 s_2}} \int_0^1 |W\ C_{s_1,s_2}(p)| dp, \tag{2.4.5}$$

onde $C_{s_1,s_2}$ (p) é o caminho que conecta os pixeis e W é o $\nabla P_f$(x).

O valor de $\alpha$ é gerado a partir da Equação 2.4.6:

$$\alpha(x) = \frac{\omega_F(x)}{\omega_F(x) + \omega_B(x)}, \tag{2.4.6}$$

onde $\omega_F(x)$ e $\omega_B(x)$ são calculados pela Equação 2.4.7:

$$\omega_l(x) = D_l(x)^{-r} P_l(x).\ l \in F,B \tag{2.4.7}$$

## 2.5 Métodos que utilizam *trimap*

*Trimap* é uma informação utilizada para muitos algoritmos de *matting* digital, ele consiste na marcação da imagem em pontos que são do plano de fundo, pontos que são do plano da frente e as áreas de incerteza. Por meio dele é possível criar de forma mais precisa a imagem final, pois a área que deve ser feita a separação da imagem é reduzida (WANG; COHEN, 2007b).

Um dos algoritmos que utiliza o *trimap* é o *Baeysian matting* (CHUANG et al., 2001), semelhante ao algoritmo Ruzon and Tomasi (RUZON; TOMASI, 2000), que faz uso de amostras de cor, gerando distribuições gaussianas, entretanto utiliza algumas modificações. A primeira delas é o uso de uma janela, esta que percorre a imagem de forma contínua, para definir as vizinhanças sobre as imagens de fundo e frente, gerando o *alphamap*.

O método utiliza novos conhecimentos e amostras do *background* para construir distribuições de cores, também são utilizadas nas proximidades computando *foreground*, *background* e $\alpha$, de modo que todos os pixeis na vizinhança contribuam para formar as distribuições gaussianas do primeiro plano e plano de fundo, ver Figura 8.

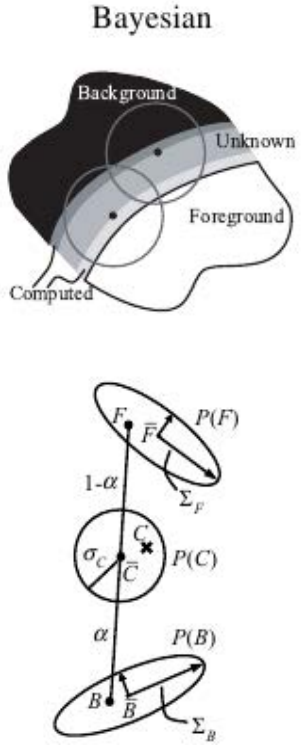



Figura 8: Análise local do pixel desconhecido. Fonte: (CHUANG et al., 2001).

As probabilidades são modeladas pela derivada da distribuição Gaussiana orientada elíptica, é utilizada a matriz de covariância ponderada.

Quando o pixel é atribuído para ambos os planos, é realizado um procedimento de otimização para o par de de pontos "*background*" e "*foreground*" da imagem. O melhor resultado é escolhido pela máxima verossimilhança.

O resultado pode ser mais preciso dependo das áreas do *trimap* serem bem definidas, de forma contrária os resultados podem ser muito ruidosos, como mostrado na Figura 9.

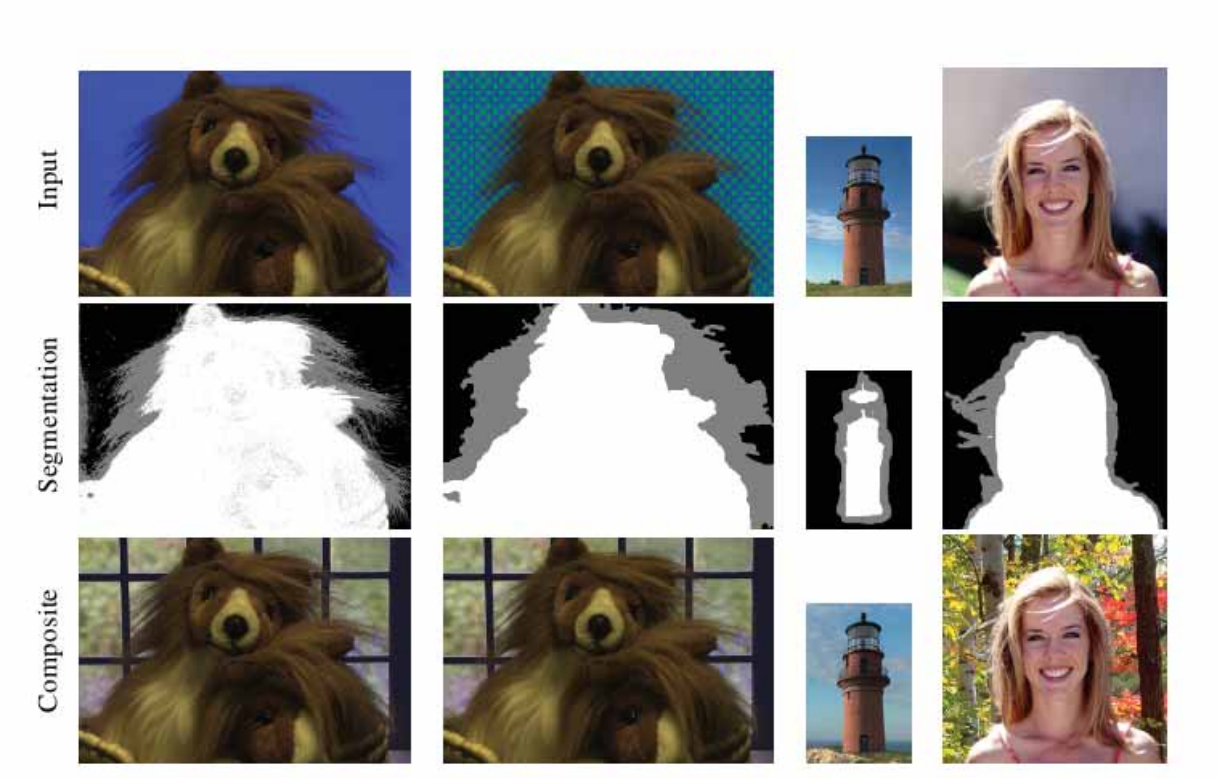



Figura 9: Exemplos de resultados. Fonte: (CHUANG et al., 2001).

Outro método mais recente, que pode ser utilizado é o sistema *Robust matting* (WANG; COHEN, 2007a), que combina a amostragem otimizada em cores com o *matting affinity,* resultando em um sistema bem equilibrado, que é capaz de gerar resultados de alta qualidade, mantendo um razoável grau de robustez com diferentes entradas do usuário. O método utiliza uma função de energia, esta que é minimizada (WANG; COHEN, 2007b).

São criados nós virtuais, estes que representam o "fundo puro", ou a "frente pura", conforme é mostrado na Figura 10, em que ΩF e ΩB representam os nós virtuais de *foreground* e *background*, respectivamente e a cor branca representa os pontos de incerteza. Estes nós ideais são utilizados para encontrar os valores de alfa. Essa metodologia é adotada, pois nem sempre as amostras de cor são totalmente confiáveis, dessa forma os resultados são melhores (OWENS et al., 2007).

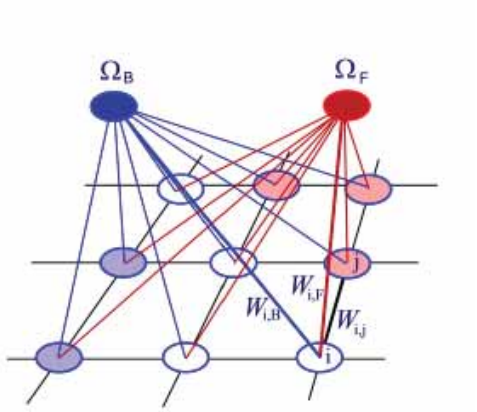



Figura 10: Grafo rotulado do *Robust matting*. Fonte: (WANG; COHEN, 2007a).

É importante frisar que os pontos de certeza de plano da frente ou plano de fundo são marcados pelo usuário, por meio do *trimap*.

## 2.6 Outros métodos de *trimap*

Alguns algoritmos utilizam métodos estatísticos, entre eles está o *matting* de Poisson (WANG; COHEN, 2007b), que são modelos de gradiente, assumindo que as mudanças de intensidade no primeiro plano e plano de fundo são localmente suaves. O algoritmo de passeio aleatório (WANG; COHEN, 2007b) é empregado para calcular os valores de alfa finais com base na afinidade. Semelhante ao *Geodesic Matting* é o *Fuzzy Connectedness for matting* (WANG; COHEN, 2007b), entretanto ao invés de calcular a distancia geodésica, ele calcula a conectividade Fuzzy (FC), que captura a união entre os elementos da imagem, a partir de um pixel conhecido para um desconhecido. O *Closed-form matting* (WANG; COHEN, 2007b) é um método que não faz uso de estimativas de cores e distribuições, é criada uma função de custo a partir de pressupostos de suavidades locais nos dois planos. O *Spectral matting* (WANG; COHEN, 2007b) é uma abordagem que tenta tirar o primeiro plano de uma forma completamente automática.

No *matting* iterativo (WANG; COHEN, 2007b) a entrada é um plano com rabiscos (entradas esparsas) informando o plano da frente e de fundo. Por meio da teoria dos campos randômicos de Markov o *matting* é modelado, e em cada iteração a energia da função é minimizada. Os histogramas utilizados são feitos a partir dos métodos *Global color models* e geram o custo dos dados. O *Easy matting* (WANG; COHEN, 2007b) é muito semelhante ao *matting* iterativo, entretanto não é utilizado nenhum mapeamento exponencial. É utilizada uma função quadrática, assim a energia pode é minimizada.

Existem muitos métodos, cada um possui uma precisão e um tempo de execução diferentes, ver Figura 11.

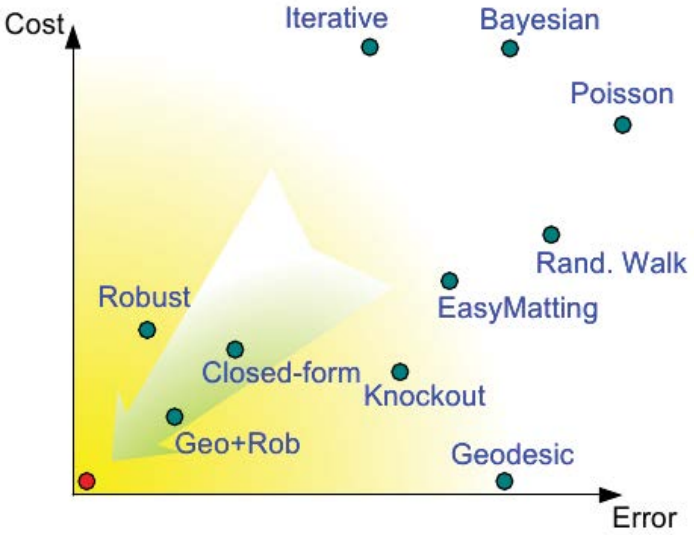



Figura 11: Custo x Erro. Fonte: (WANG; COHEN, 2007b).

Outra abordagem mais moderna é o *Shared Sampling for Real-Time Alpha Matting* (GASTAL; OLIVEIRA, 2010). O qual é um algoritmo, que segundo o autor, que funciona em tempo real. A técnica é baseada em analisar os pixeis da vizinhança, pois, em grande parte das vezes, têm valores muito próximos. Ele possui alguns problemas como em imagens transparentes, que podem aparecer ou desaparecer da cena. Para que possa ser computado em tempo real foi utilizada a GPU, utilizando um algoritmo que permite o processamento.

## 2.7 *Matting* digital com informação extra

O método *Mishima* (WANG; COHEN, 2007b) utiliza o *chroma key* como informação extra, "utiliza o fundo com uma cor definida (azul), todos os pixeis de fundo são cobertos por uma pequena esfera aproximada por um poliedro no espaço de cor. Todos os pixeis de primeiro plano uma outra forma poliedros separado do fundo. O valor de alfa de um pixel desconhecido é estimado por meio do cálculo da sua posição em relação aos dois poliedros".

No método *Knockout* (WANG; COHEN, 2007b) o valor de alfa é estimado através do somatorio dos valores dos pixeis vizinhos, dessa forma os pesos gerados são proporcionais a distâncias espaciais.O valor de alfa é calculado por meio da sua posicão relativa.

## 2.8 Considerações Finais

A partir do estudo mostrado neste capítulo foi possível escolher quais os métodos utilizar para criação automática de *trimap* e geração de *alphamap*, escolher algoritmos para os testes e também algoritmo de geração da imagem composta.

Desta forma foi possível reproduzir e compreender inteiramente o pipeline do *Matting* digital, abordado neste trabalho.

# Capítulo 3

# Arquitetura CUDA

Neste capitulo será apresentada a arquitetura de GPGPU (*General Purpose Graphics Processing Unit*), que é uma placa gráfica para uso geral, desenvolvida pela NVIDIA, serão mostradas as suas características, estas que podem ser utilizadas para diminuir o tempo de processamento em aplicações desenvolvidas com a utilização do paralelismo de dados.

## 3.1 Considerações Iniciais

Devido ao seu paralelismo de dados, envolvendo uma alta quantidade de núcleos de processamento, a arquitetura CUDA é uma grande aliada na resolução de problemas envolvendo grande fluxo de dados, sendo muito útil na implementação de algoritmos de *matting*, principalmente quando o método é aplicado em imagens de alta resolução, podendo ter um desempenho superior ao dos processadores de uso geral.

## 3.2 Arquitetura CUDA

A complexidade dessas técnicas de *matting* exige um alto desempenho do hardware na qual elas são executadas. Cada vez mais se procura soluções para executá-las de forma eficiente.

Uma dessas soluções é a arquitetura de computação paralela CUDA, da empresa NVIDIA. Utilizada na fabricação de GPUs (*Graphic Processor Units*), ela se aproveita da presença de múltiplos processadores e alta banda de memória para realizar operações em grande volume de dados. Sua utilização é adequada para resolver problemas que apresentem paralelismo de dados, em que a mesma instrução é executada paralelamente em dados diferentes (NVIDIA, 2011).

As primeiras CPUs (*Central Processing units*) para o uso em computadores pessoais possuíam em torno de 1MHz, com o passar dos anos e com o avanço da tecnologia os processadoras foram ficando mais rápidos, chegando atualmente a 4GHz. Por restrições dos circuitos eletrônicos não é mais possível que acelerar o processamento por meio do aumento do clock do processador. Como solução para este problema surgiram os processadores com multicores, dessa forma conseguem

melhorar o desempenho pela quantidade de núcleos sem aumentar o clock.

No final dos anos 1980 e início dos anos 1990, com a popularização das interfaces gráficas (como por exemplo o Windows), tornou-se necessário criar um hardware específico para processar as imagens. No início dos anos 1990 foi criada a placa aceleradora 2D, a partir dela eram feitas operações com *bitmaps*.

Em meados dos anos 1990 começaram a surgir os jogos em 3D, assim surgiu a necessidade de um hardware específico para processá-los. Foi nessa época que começaram a surgir as placas aceleradoras 3D. O lançamento da GeForce 256 da NVIDIA aumentou a capacidade do hardware gráfico, pela primeira vez, os cálculos de iluminação foram realizados diretamente no processador de vídeo, aumentando assim o potencial para aplicações, visualmente mais interessantes.

Em 2001 surgiu a GeForce3, que foi a primeira placa a utilizar o DirectX 8, contribuindo para a programação de sombreamento e vértices. Devido a isso, os pesquisadores exploraram computação de propósito geral por meio de APIs gráficas, tentando trazer os seus problemas para a GPU, transformando-os em uma renderização tradicional.

Em novembro de 2006 foi criada a GeForce 8800 GTX, esta que foi lançada com a arquitetura CUDA. “Esta arquitetura incluiu vários novos componentes projetados estritamente para computação GPU com o objetivo de aliviar muitas das limitações que impediram que os processadores anteriores gráficos de ser legitimamente úteis para computação de propósito geral” (SANDERS; KANDROT, 2010).

Diferente das versões anteriores da placas gráficas, as placas com a tecnologia CUDA possuem um único tipo de ULA (Unidade Lógica Arutimética), sendo que a precisão dos cálculos feitos por elas segue as normas do IEEE, outra funcionalidade é que existe a possibilidade de haver acesso a memória do computador através da placa. Todas essas características da arquitetura CUDA foram adicionados a fim de criar uma GPGPU (*General-Purpose computation on Graphics Processing Units*) que faria sobressair em computação, além de um bom desempenho em tarefas tradicionais gráficos (SANDERS; KANDROT, 2010).

A GPU tem uma capacidade muito superior de operações de ponto flutuante do que a CPU (vide Figura 12). Isto porque possui mais transistores, estes que são dedicados ao processamento de dados ao invés de dados cache e controlo de fluxo, tal como esquematicamente ilustrado pela Figura 13. Dados paralelos podem ser processados em *threads* paralelas, como em muitas aplicações gráficas. “Muitas aplicações que processam grandes conjuntos de dados podem usar um modelo de programação paralela de dados para acelerar os cálculos” (NVIDIA, 2011).

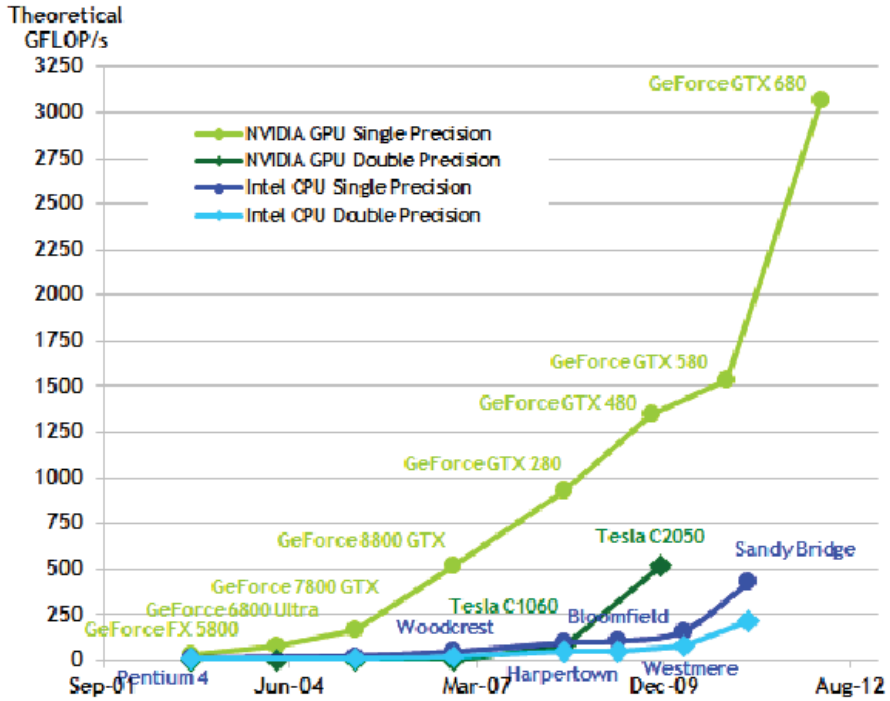



Figura 12: Operações de ponto flutuante por segundo de largura de banda e memória para a CPU e GPU. Fonte: (NVIDIA, 2011).

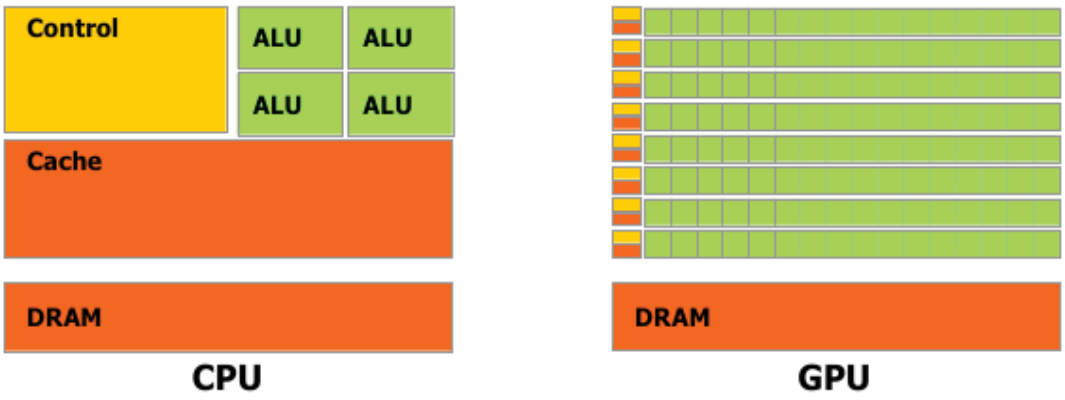



Figura 13: A GPU Dedica mais transistores para Processamento de Dados. Fonte: (NVIDIA, 2011).

A partir de 2007 a indústria começou a utilizar esta tecnologia, pois com o uso da GPGPUs há uma economia no preço do hardware pela quantidade de dados processados, e também utiliza menos energia elétrica, com um maior desempenho por Watt (SANDERS; KANDROT, 2010).

## 3.3 Programação Heterogênea

A tecnologia CUDA é facilitada na sua programação, permitindo que o usuário faça programas utilizando a linguagem C, dessa forma podem ser feito parte do programa utilizando o processador (demarcadas como *host*), e parte paralela, utilizando a GPGPU (demarcadas como *device*), todas as funções que são executadas na GPGPU são chamadas de *kernel*. (NVIDIA, 2011), como mostra a Figura 14.

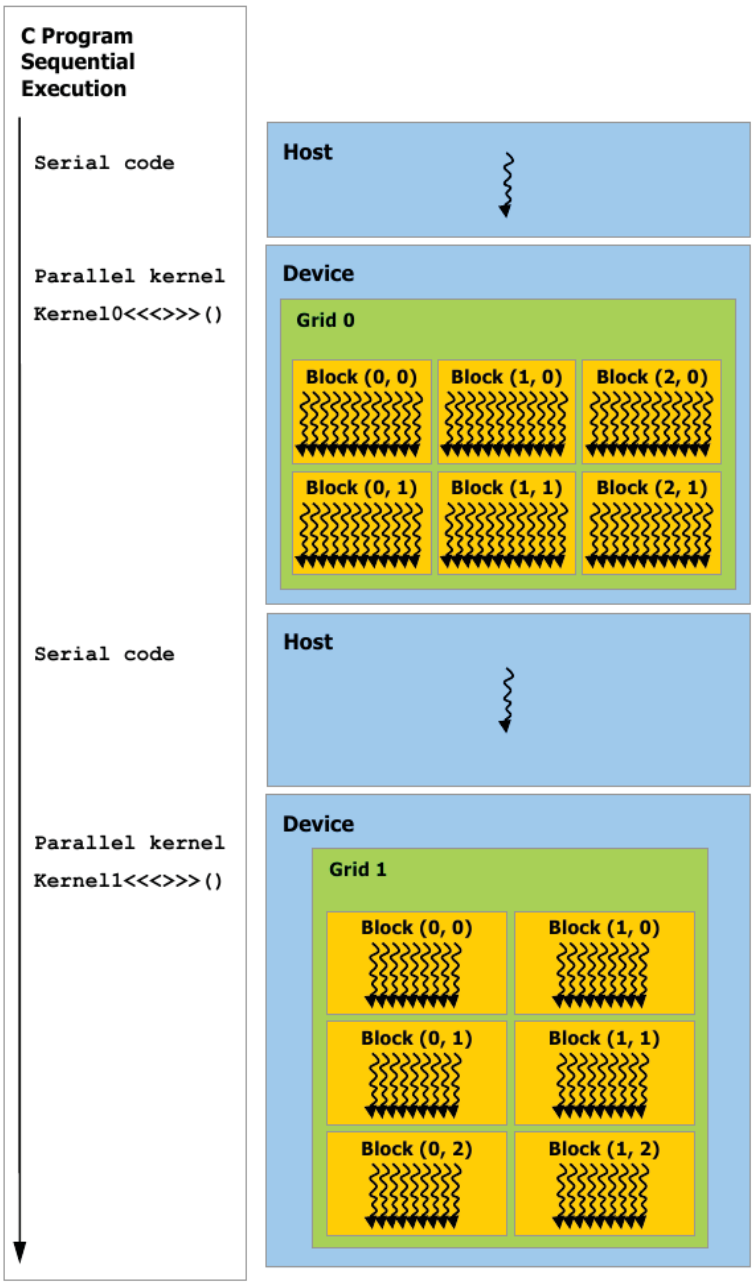



Figura 14: Programação heterogenia. Fonte: (NVIDIA, 2011).

## 3.4 Hierarquias de Memória

Uma característica importante é que as *threads* em CUDA podem acessar dados de múltiplos espaços de memória durante a sua execução. Cada segmento tem memória local, cada bloco tem memória compartilhada, que é visível em todos os blocos e todas as *threads* têm acesso à mesma memória global, (vide Figura 15). Há também as memórias que são somente de leitura, a memória constante e memória de textura (NVIDIA, 2011).

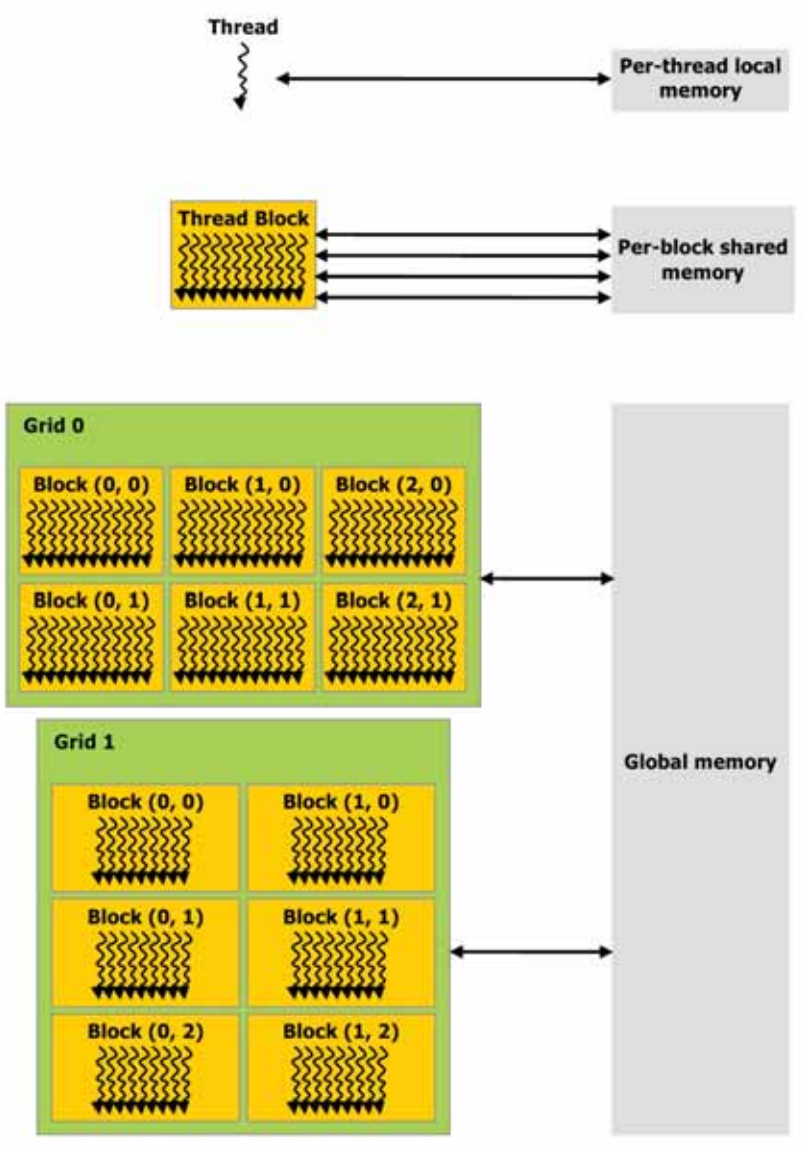



Figura 15: Hierarquia de memória. Fonte: (NVIDIA, 2011).

Cada um desses tipos de memória tem uma determinada função, contendo suas vantagens e desvantagens.

A memória global, que é a memória com maior tamanho na placa gráfica pode ser lida ou

escrita por todas as *threads*, entretanto ela pode ser mais lenta, ou seja demandar um maior número de ciclos para ser lida, ou escrita. Para melhorar o desempenho ela deve ser utilizada de forma correta, permitindo assim que a sua leitura seja feita de forma paralela por diversas *threads*.

Por ser uma memória acessível a todas as *threads* do *kernel*, além de servir para armazenas as variáveis compartilhadas entre todas elas, é utilizada, também, como uma forma de I/O (entrada e saída).

O gerenciamento desse tipo de memória é feito pelo *host*, é nele que devem ser feitas a alocação de um determinado espaço e as cópias de memória, que são do *host* para o *device*, do *device* para o *host*, ou do *device* para ele mesmo. Além disso é por meio do *host* que deve ser desalocada a memória global.

A memória compartilhada é muito mais rápida que a memória global, ou seja utiliza uma quantidade de ciclos muito menor para ser lida, entretanto ela só pode ser lida, ou escrita pelas *threads* do bloco. Outro fator importante é que ela não precisa ser desalocada da memória, é apagada automaticamente quando o *kernel* termina de ser executado. Dessa forma ela não pode ser utilizada para a comunicação entre dois ou mais *kernels*.

A memória local é extremamente rápida, pois ela é armazenada nos registradores, entretanto ela só pode ser lida ou escrita, localmente, pela própria *thread*, não servindo para a comunicação entre as *threads*. Ela também não deve ser desalocada, isso acontece automaticamente no final da execução da *thread*.

Além das memórias já descritas, que podem ser tanto escritas como lidas, existem também as memórias que são apenas para leitura, armazenadas em *cache*. São elas a memória de textura e a memória constante, ambas são muito mais rápidas que a memória global, entretanto devem ser alocadas e desalocadas a partir desta, por comandos executados pelo *host*.

A diferença entre as duas está principalmente nos tamanhos e na forma de acesso. A memória de textura é maior, e permite o acesso de várias *threads* por dados diferentes simultaneamente. Já a memória constante permite o acesso ao mesmo dado por *threads* diferentes, em ambas, quando o acesso é feito de forma diferente do que foi descrito, os dados são lidos de forma serial.

## 3.5 Considerações Finais

Existem vantagens em utilizar a arquitetura CUDA, pois a placa gráfica muitas vezes fica ociosa no computador, consumindo energia elétrica. Além disso, a partir dela é possível aumentar significativamente o desempenho de determinadas aplicações, com funções que são executadas em paralelo. Entretanto não são todos os problemas que podem ser resolvidos com esse tipo de arquitetura, apenas os que possibilitam a codificação utilizando a arquitetura SIMT.

# Capítulo 4

# Materiais e métodos

Neste Capítulo são apresentados os materiais utilizados e a metodologia adotada para o desenvolvimento do trabalho.

## 4.1 Considerações Iniciais

Na tentativa de obterem-se aplicações que executam com restrição de tempo, foram idealizados métodos que não necessitam da intervenção humana, com a criação de algoritmos de geração automática de *trimap*. Para possibilitar o uso destes algoritmos foram criadas também soluções para o restante do *pipeline* do *Matting* digital. Dessa forma o algoritmo do *matting* foi implementado de forma mais eficiente (considerando o tempo de processamento), prevendo também a sua paralelização utilizando a arquitetura CUDA.

## 4.2 Materiais

Os materiais utilizados neste projeto foram:

- Computador Desktop com Processador Intel Core i7 930 @2.80GHz;
- Memória RAM de 12 GB;
- Disco rígido de 3 TB e rotação 5400 RPM.
- Placa gráfica NVIDIA GeForce GTX480, com 480 *cores* - compatível com a arquitetura CUDA, 1536 MB GDDR5 de memória gráfica dedicada, *Memory interface Width* 384-bit;
- Computador Desktop com Processador Intel Core i7 860 @2.80GHz;
- Memória RAM de 8 GB;
- Disco rígido de 1 TB e rotação 5400 RPM.

- Placa gráfica NVIDIA GeForce GTS450, com 196 *cores* - compatível com a arquitetura CUDA, 1024 MB GDDR5 de memória gráfica dedicada, *Memory interface Width* 128-bit;
- Ambos com o sistema operacional Windows 7 - versão 64 bits;
- Microsoft Visual Studio 2008;
- CUDA SDK;
- CUDA *Development toolkit*;
- OpenCV2.2;
- Iluminação controlada;
- Fundos verde e azul para o *chromakey*;
- Webcam Microsoft LifeCam, com resolução máxima de 640 x 480 pixels e taxa de captura de até 30 FPS;
- Câmera FullHD com taxa de captura de 30fps.

Alguns testes iniciais foram no laboratório SACI (Sistemas Adaptativos e Computação Inteligente). Todos os experimentos foram feitos em um local apropriado, com a iluminação controlada e cor de fundo verde ou azul.

## 4.3 Desenvolvimento de novos métodos

Na tentativa de alcançar o tempo real foram desenvolvidos métodos de geração automática de *trimap*, sem utilizar marcações manuais, e uma adaptação do *Geodesic Matting*, uma vez que esta técnica demanda um menor tempo de processamento.

### 4.3.1 *Trimap* automático com *chroma key*

O *trimap* foi gerado automaticamente através de um aperfeiçoamento de uma técnica de *chroma key*. Nela, o pixel a ser investigado como pertencente ou não à cor arbitrária (verde), está limitado entre dois valores a partir da Equação 4.3.1 de (BERGH; LALIOTI, 1999), ou seja limitar o verde praticamente puro das outras tonalidades de verde (vide Figura 16).

$$D = B - R - 2G \tag{4.3.1}$$

Onde: D caracteriza a proximidade entre a cor do pixel com o verde;

B componente azul do pixel;

R componente vermelho do pixel;

G componente verde do pixel (cor do background).

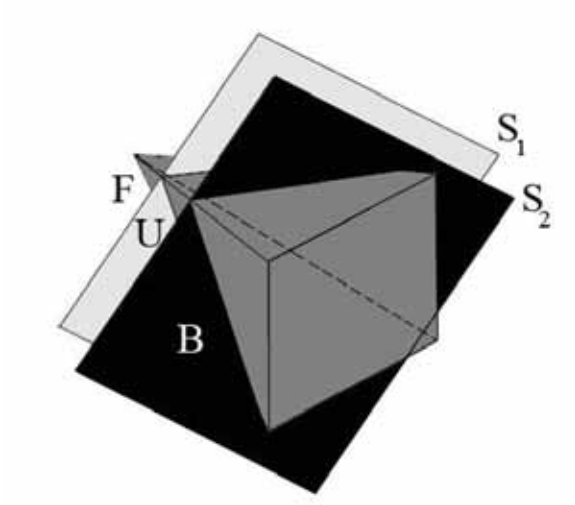



Figura 16: Cubo RGB com a interseção dos planos S1 e S2.

Onde o espaço entre os planos S1 e S2 (representado por U) representa a área de incerteza do *trimap* gerado e as letras F e B representam as áreas de certeza de *foreground* e *background*, respectivamente.

### 4.3.2 Geração automática de *trimap* com uso de Kinect e *chroma key*

Equipamentos de luz estruturada geram e projetam padrões de luz com características conhecidas sobre objetos e, a partir da captura e análise das distorções destes padrões, calculam as distâncias entre a superfície do objeto e o sensor de captura. É possível utilizar diversos padrões de luz, como linhas, grades, círculos, senoidais, etc. As limitações deste processo variam de acordo com a forma de projeção da luz (REISS; TOMMASELLI, 2003).

É importante ressaltar que esta técnica apesar de poder utilizar o infravermelho como luz estruturada é diferente do TOF (*Time of Flight*). Dispositivos, como por exemplo, câmeras TOF, medem o tempo que o sinal infravermelho leva para refletir sobre o objeto alvo e retornar, possibilitando, assim, o cálculo da profundidade para cada pixel da imagem (mapa de profundidade).

O Kinect é um aparelho de luz estruturada criado pela Microsoft, que gera um mapa de profundidade. Foi feito para ser utilizado no controle de jogos do Xbox 360, entretanto pode ser utilizado para outras finalidades. Ele é formado por três componentes óticos, um projetor de infravermelho, uma câmera de infravermelho (a partir dela é gerado o mapa de profundidade) e uma câmera RGB (KÜHN, 2011) (vide Figura 17).


Figura 17: Dentro do Kinect. Fonte: (KÜHN, 2011).

O mapa de profundidade de determinado objeto, obtido por meio do uso do Kinect, pode facilitar a geração automática de *trimaps*, pois a partir dele é possível ter uma noção entre as distâncias do objeto do primeiro plano e do plano de fundo, assim facilitando o recorte da imagem.

Na criação do algoritmo foi considerado que os objetos pertencentes ao primeiro plano devem estar a uma distância máxima do aparelho, ou seja tudo que estiver além dessa distância é considerado como plano de fundo e as áreas de incerteza do *trimap* são os limites entre os planos e as áreas incertas do mapa de profundidade.

Como a imagem do mapa de profundidade é menor que a imagem da câmera RGB, esta é redimensionada, para que desta forma as duas possuam o mesmo tamanho. Outro fator importante é que como as duas câmeras do Kinect ficam em posições diferentes, a imagem da captura de ambas é diferente, sendo assim deve ser feito uma sobreposição das imagens, para que cada pixel tenha o seu correspondente, mesmo assim existem áreas em que não há correspondência enter elas.

Já com as imagens tendo sua correspondência utiliza-se um filtro para retirar os ruído, o filtro gaussiano. Na imagem do mapa de profundidade utiliza-se a dilatação (com valor de 4 pixeis), para que desta forma o pixeis demarcados como indefinidos no centro do objeto sejam excluídos (eles possuem a cor preta).

As duas imagens são analisadas pixel a pixel, gerando uma terceira imagem, que é o *trimap*. Primeiramente todos os pixeis demarcados no mapa de profundidade como desconhecidos são colocados no trimap como plano de fundo. No restante da imagem são feitas duas formas distintas de análise, uma para seres humanos e outra para objetos desconhecidos.

Após utilizar um dos 2 algoritmos, é aplicada uma erosão na imagem do trimap, para que as bordas que os resultados sejam melhores, retirando alguns possíveis ruídos na área de incerteza, que é demarcada pela cor cinza.

#### 4.3.2.1 Análise para seres humanos

O Kinect possui uma ferramenta (o *Player Index*) com a característica de demarcar no mapa de profundidade o que é uma pessoa, dessa forma já foi demarcado no *trimap* como plano de fundo

tudo aquilo que não é demarcado no *player Index*, assim a área a ser analisada fica restrita, sendo que o restante da imagem já é demarcado como plano de fundo.

Para colocar uma ajuste final demarcando o que certamente é plano da frente, ou área de incerteza é feita uma modificação do algoritmo de (BERGH; LALIOTI, 1999) na qual os pontos demarcados como plano da frente são colocados no trimap como plano da frente e os pontos demarcados como plano de fundo são demarcados como área de incerteza, como mostra a Figura 18.

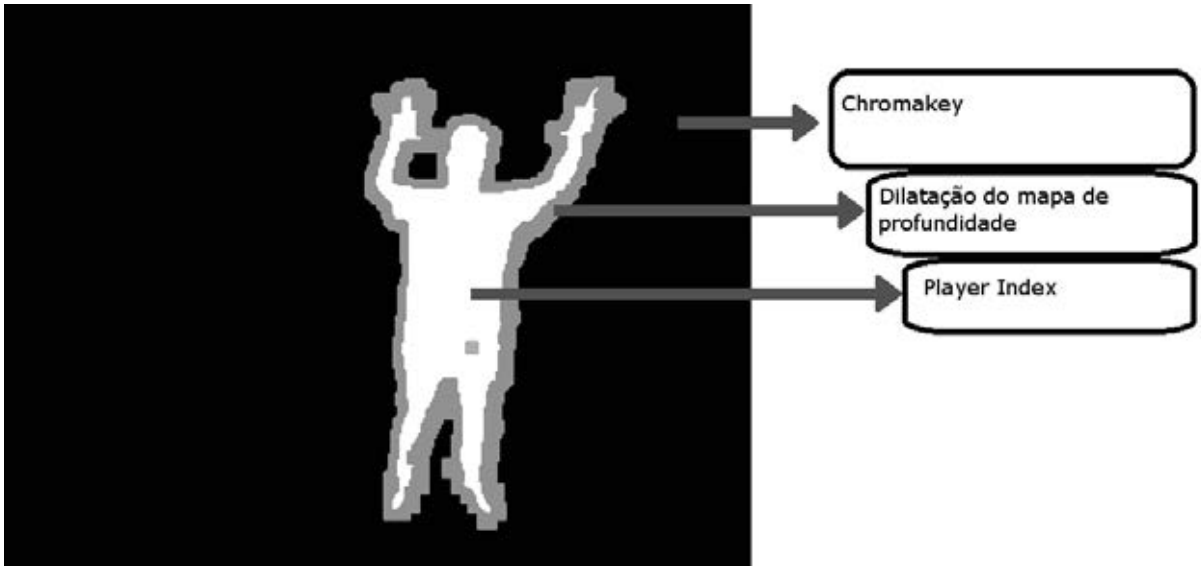



Figura 18: Exemplo de *Trimap gerado pelo* uso do *chroma key* e o *Player Index.*

#### 4.3.2.2 Análise para plano da frente não humano

Nesta abordagem, a imagem é analisada utilizando o algoritmo de Van den Berg & Lalioti (1999), sendo os pixeis detectados como sendo da cor chave marcados como fundo na imagem do trimap. Simultaneamente é feita outra análise, utilizando a imagem de profundidade. Nesta segunda análise são marcados como frente todos os pixeis que estavam marcados na imagem monocromática como frente (branco da imagem resultado). Os pixeis que foram marcados nos dois métodos são os de incerteza.

Para esta análise da imagem de profundidade foi colocada uma faixa de distâncias que classificam o objeto como plano da frente.

### 4.3.3 *Geodesic* modificado

Foram feitas algumas modificações no algoritmo *Geodesic Matting* (BAI; SAPIRO, 2009), para possibilitar o seu uso sem a intervenção do usuário, assim como foram simplificadas algumas etapas para diminuir o tempo de processamento.

O artigo de Bai & Sapiro (2009) sugere que o método seja executado duas vezes, com a fina-

lidade de refinar o resultado. Na primeira vez que o programa é executado são utilizados rabiscos feitos manualmente e, a partir deles é gerado um *alphamap*. Com ele são feitas marcações nos limites entre os planos (da frente e de fundo) e áreas de transparências, estes são considerados os rabiscos ideais, ou seja o mais próximo possível dos pontos de incerteza. Com estes novos rabiscos o método é executado novamente, para obter o resultado final.

No método proposto foi utilizado o *trimap* como entrada, juntamente com a imagem que vai ser recortada, desta forma os rabiscos foram feitos a partir dos pixeis que estão nos limites entre os planos (da frente e de fundo) e as áreas de incerteza, gerando também amostras de cor nas áreas próximas as transparências. Além disso os valores de *alpha* foram calculados apenas nas áreas de incerteza do *trimap*, para diminuir o tempo de processamento.

Outra modificação foi transformar a imagem original, colorida, em monocromática, pois os pixeis coloridos possuem 3 componentes de cor, já na imagem monocromática possuem apenas uma componente. Assim as PDFs são geradas com apenas uma variável, facilitando a busca pelo menor caminho e diminuindo o tempo de processamento.

Foi feita uma modificação na parte mais crítica do código, ou seja o segmento que mais consome tempo de processamento, que é a busca pelos caminhos geodésicos, que foi substituída por uma nova rotina, conforme ilustrado na Figura 19.

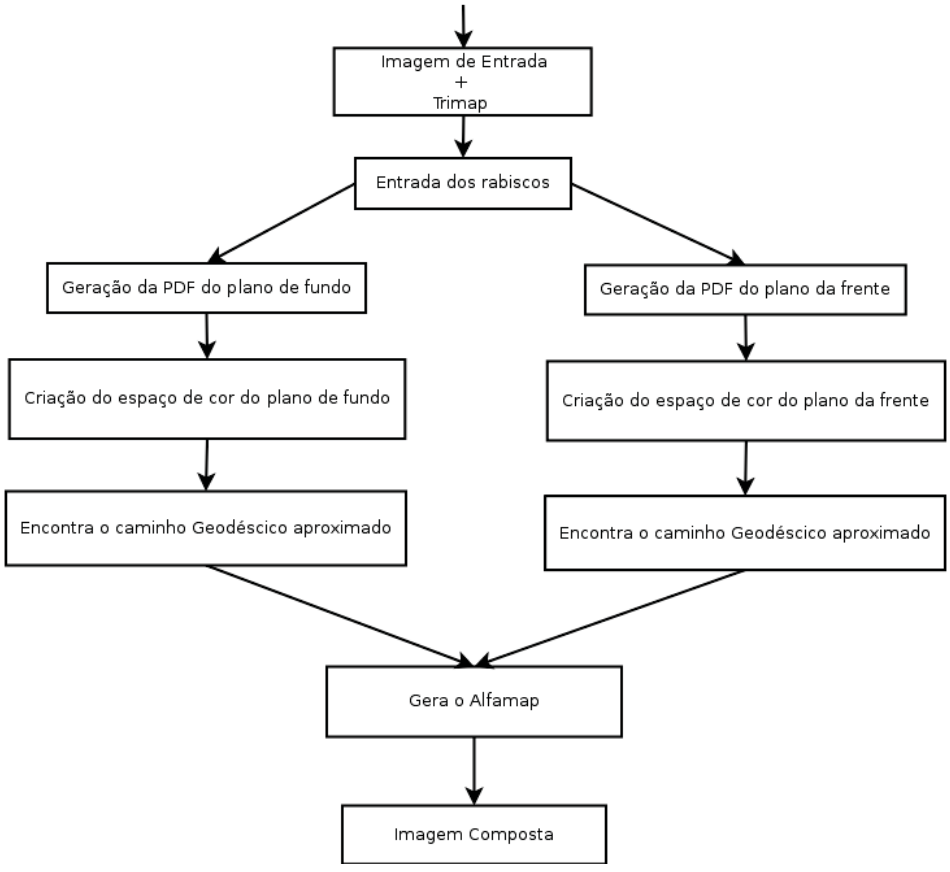



Figura 19: Fluxograma do algoritmo *Geodesic Matting*.

No código original era encontrado o menor caminho geodésico, todavia esta rotina foi substituída por um código simplificado, que consiste em fazer uma busca entre cada ponto de incerteza e o pixel do rabisco mais próximo, utilizando a distância euclidiana. Dessa forma é criado, a partir de um ponto de origem e outro de destino, encontrados na distância euclidiana, uma distância geodésica aproximada. Esta que consiste em caminhar por uma janela criada entre os pontos, caminhando sempre para o ponto com a menor distância entre eles, do ponto de origem até o destino.

## 4.4 Considerações Finais

Foram utilizadas as informações extra de profundidade, do Kinect, e de cor chave, do *chroma key*, para gerar um *trimap* automaticamente, assim permitindo, assim, o uso do método para aplicações com restrição de tempo. Dessa forma é necessário também que o método de geração do *alphamap* também possa ser usado em aplicações com restrição de tempo, por esse motivo foi criada uma modificação do *Geodesic matting*.

Para as demais partes do pipeline, que não necessitam de nenhuma modificação foram utilizados os algoritmos já descritos pela literatura, mostrados no Capítulo 2. As implementações realizadas serão mostradas no Capítulo 5.

# Capítulo 5

# Implementação

Neste Capítulo é apresentada a implementação dos métodos do trabalho, que foi feito integralmente utilizando a linguagem C++.

## 5.1 Considerações Iniciais

Primeiramente os métodos foram implementados de forma sequencial, para que desta forma fossem testados. A partir da análise destes códigos apenas as partes que exibiam a característica do paralelismo de dados foram, efetivamente, implementados para execução em CUDA. Os detalhes desta paralelização são descritos na próxima seção.

## 5.2 *Pipeline* do processo de *Matting Digital*

Os métodos implementados neste trabalho seguiram todo o pipeline do *matting digital*, este que é descrito pela Figura 20.

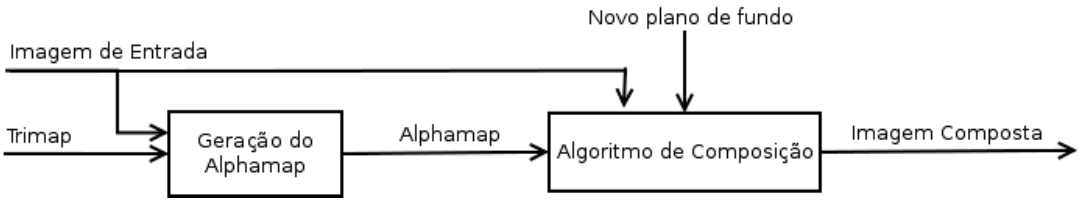



Figura 20: *Pipeline* do *Matting Digital*.

## 5.3 Implementação sequencial

Todos os programas foram implementados de forma sequencial, alguns foram utilizados apenas para auxiliar nos testes, outros foram paralelizados na Linguagem C para CUDA, para que desta

forma tenham um maior desempenho com relação ao tempo de processamento.

### 5.3.1 *Chroma key*

Primeiramente foi implementado o algoritmo proposto por Van Den Bergh and V. Lalioti (BERGH; LALIOTI, 1999). Este algoritmo é composto por um par de laços aninhados, que lê todos os pixeis da imagem e substitui ou não pelo novo plano de fundo, esta substituição é feita a partir da Equação 4.3.1.

A implementação deste método substitui todo o pipeline mostrado na seção 5.2, descrito na Figura 20.

### 5.3.2 Automatização do *trimap* com uso de *chroma key*

A modificação do pipeline feita pela criação automática de *trimap* é mostrada na Figura 21, representado pela cor azul.

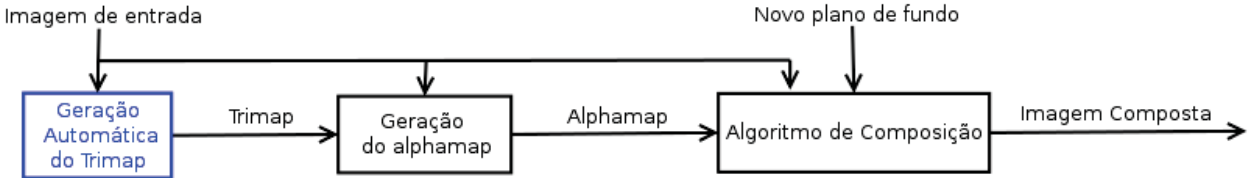



Figura 21: *Pipeline* do *Matting Digital,* com *Trimap* automático .

A partir do programa de *chroma key* já implementado, foi criada uma nova imagem de saída, a qual originou o *trimap*. A diferença entre dois código é que neste foram feitas três condições, ao invés de uma, como no caso anterior. A primeira para limitar os pixeis marcados como plano da frente, a segunda para demarcar os pixeis como plano de fundo, os demais foram marcados como pontos incerteza. Todos eles utilizando valores da Equação 4.3.1.

### 5.3.3 Automatização do *trimap* com uso de Kinect e *chroma key*

A modificação do pipeline feita pela criação automática de *trimap* com o uso do Kinect é mostrada na Figura 22, representado pela cor azul.

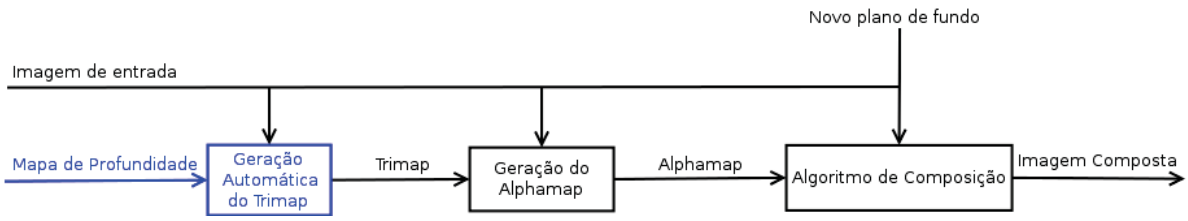



Figura 22: *Pipeline* do *Matting Digital*, com Kinect e *chroma key*.

O trimap automático é gerado a partir da imagem do mapa de profundidade e aplicando a técnica de *chroma key* da imagem RGB, assim foi necessário utilizar uma aproximação entre a imagem do mapa de profundidade e a imagem RGB.

O algoritmo consiste em cinco etapas:

- Tendo em vista a aproximação da imagem monocromática para a RGB e a própria área de incerteza gerada ao redor do objeto pelo próprio dispositivo contribuiu para a presença de ruídos no mapa de profundidade. Com isso a solução encontrada foi a aplicação de dois filtros: dilatação e gaussiana;

As quatro etapas seguintes dependem da comparação do pixel no mapa de profundidade com o pixel correspondente na imagem RGB:

- Caso o pixel no mapa de profundidade represente o ser humano e o seu correspondente na RGB não represente a cor do fundo arbitrário (azul ou verde), então esse pixel será representado no *trimap* como primeiro plano;
- Caso o pixel no mapa de profundidade represente o ser humano e o seu correspondente na RGB represente a cor do fundo arbitrário (azul ou verde), então esse pixel será representado no *trimap* como área de incerteza;
- Caso o pixel no mapa de profundidade não represente o ser humano e o seu correspondente na RGB não represente a cor do fundo arbitrário (azul ou verde), então esse pixel será representado no *trimap* como plano de fundo;
- Caso o pixel no mapa de profundidade não represente o ser humano e o seu correspondente na RGB represente a cor do fundo arbitrário (azul ou verde), então esse pixel será representado no *trimap* como plano de fundo.

Estas cinco etapas estão contidas em um dois laços aninhados, estes que passam por todos os pontos da imagem para executá-los.

Para a aplicação feita para objetos desconhecidos foi utilizada a distância entre o aparelho e o objeto ao invés de utilizar o sistema de detecção de humanos. O restante do código foi igual ao apresentado anteriormente.

### 5.3.4 Geração da imagem a partir do *Alphamap*

A geração da imagem composta é uma parte importante no algoritmo de *matting* digital, é mostrada na Figura 23, representado pela cor azul.

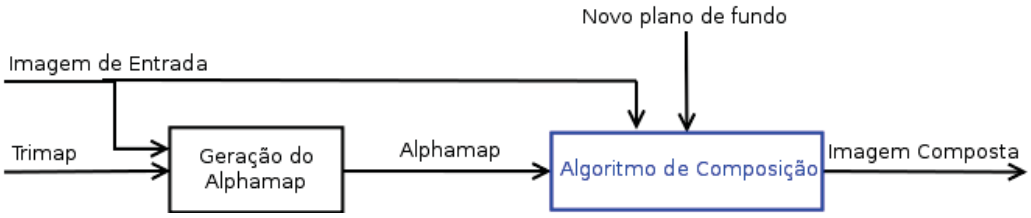



Figura 23: *Pipeline* do *Matting Digital*, composição.

Foi implementada a Equação 5.3.1, que determina o canal alfa (α) de cada pixel, desenvolvida por (PORTER; DUFF, 1984).

$$C = F\ \alpha + B\ (1 - \alpha) \tag{5.3.1}$$

onde: C é a imagem composta;

F é a imagem que esta sendo recortada;

B é a imagem do novo plano de fundo.

Essa rotina consiste em dois laços aninhados, que percorrem todos os pixeis da imagem, a Equação 5.3.1 é aplicada em todos eles e o resultado obtido é salvo em uma terceira imagem, contendo o resultado do *matting*.

### 5.3.5 Programas para testes de resultados

O algoritmo de geração do *alphamap* é a parte mais complexa, ela pode ser representada na cor azul pelo pipeline da Figura 24.

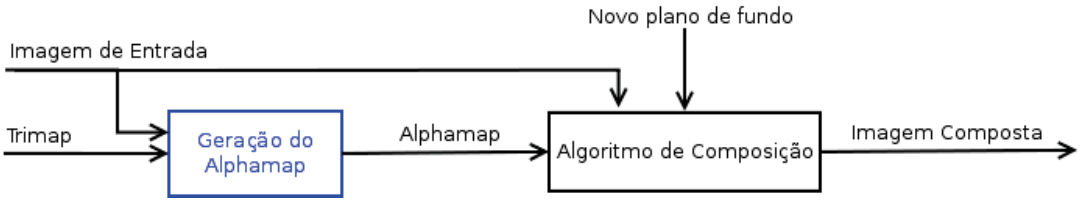



Figura 24: *Pipeline* do *Matting Digital*, algoritmo de *Matting Digital*.

Foram implementados o *Bayesian Matting* (CHUANG et al., 2001) e o *Robust Matting* (WANG; COHEN, 2007a), a partir de códigos cedidos pelos autores de (SUN et al., 2006). Estes programas foram modificados, pois possuíam limitações quanto ao tamanho das imagens e uso de bibliotecas.

Estes códigos foram implementados para auxiliar nos testes e comparações dos resultados.

### 5.3.6 *Geodesic* modificado

O algoritmo proposto para a modificação do *Geodesic Matting* foi utilizado no ponto que está demarcado em azul do *pipeline* da Figura 24.

Primeiramente a imagem de entrada foi transformada de colorida para tons de cinza, ou seja a imagem que antes possuía 3 bytes para cada pixel passou a ter apenas 1 byte, dessa forma facilitando as etapas seguintes do programa.

A segunda parte do código foi a criação dos rabiscos, para isto o *trimap* foi lido na forma de uma imagem, cada pixel analisado separadamente, através de dois laços aninhados que varrem a imagem inteira. Para cada pixel pertencente ao plano da frente foram analisados os seus vizinhos, havendo algum ponto de incerteza este pixel é marcado como um ponto do rabisco do plano da frente. Este processo é repetido por toda a imagem, seguindo o mesmo processo para o rabisco do plano de fundo.

Já com todos os pontos marcados, na imagem referente ao *trimap,* foram criadas estruturas de dados para armazenar os rabiscos da frente e de fundo, separadamente. A imagem em tons de cinza foi lida, armazenando o valor de cada pixel da imagem referentes aos rabiscos marcados no *trimap*, assim como a sua posição.

A partir dos valores armazenados dos rabiscos foram criadas as duas PDFs (descrita do capítulo 2). A implementação de cada PDF foi realizada por um laço que, a partir dos dados do respectivo rabisco e o *kernel* gaussiano, faz o somatório, gerando cada ponto. Para facilitar os cálculos foi utilizado um vetor de 255 posições, no qual a integral definida da PDF de um intervalo unitário é o respectivo ponto do vetor.

Com as duas PDFs geradas foram criados os mapas de cor, chamados de relevo do plano da frente e relevo do plano de fundo. Para a criação do relevo do plano de fundo, foi criada uma imagem auxiliar. Em cada pixel da imagem de tons de cinza há um respectivo na imagem do relevo, dessa forma a imagem de tons de cinza foi lida, ponto a ponto, e foi substituído, na imagem de relevo, o valor da integral da PDF do respectivo ponto. O mesmo processo foi feito com a imagem do plano da frente.

Os caminhos geodésicos aproximados foram encontrados para todos os pixeis nos quais o *trimap* indicou área de incerteza, sendo eles o caminho referente ao relevo do plano da frente e o relevo do plano de fundo. Para encontrar esse caminho primeiramente foi percorrida a estrutura de dados que armazena o rabisco, encontrando o ponto de menor distância euclidiana, a partir disso foi criada uma janela mínima que engloba o ponto inicial e final na imagem do relevo. Com o uso de ou outro laço caminhou-se pelos pixeis, sempre em direção ao caminho geodésico mais curto, respeitando a janela, foi feito um somatório dos pontos percorridos neste caminho, sendo este o valor do caminho geodésico aproximado.

Com os valores obtidos pelos menores caminhos geodésicos foi criado o *alphamap*, de forma equivalente ao método original. Os pontos demarcados no *trimap* como pontos de certeza não foram calculados, apenas foi demarcado o valor do $\alpha$ equivalente a cada um deles.

## 5.4 Implementação paralelizada

Todos os programas que foram paralelizados utilizaram a arquitetura CUDA, que é massivamente paralela, explorando o paralelismo de dados da arquitetura SIMT.

### 5.4.1 Metodologia adotada na implementação

Para facilitar a implementação dos códigos em CUDA e melhorar o seu desempenho foi utilizada a técnica APOD (*Assess, Parallelize, Optimize, Deploy*), ou seja Avaliar, Paralelizar, Otimizar, Desenvolver, descrita pelo (NVIDIA Corporation, 2012). O APOD é um ciclo, no qual é gasto o mínimo de tempo para se obter algum resultado, mas a cada volta o tempo de processamento deve ser menor (vide Figura 25).

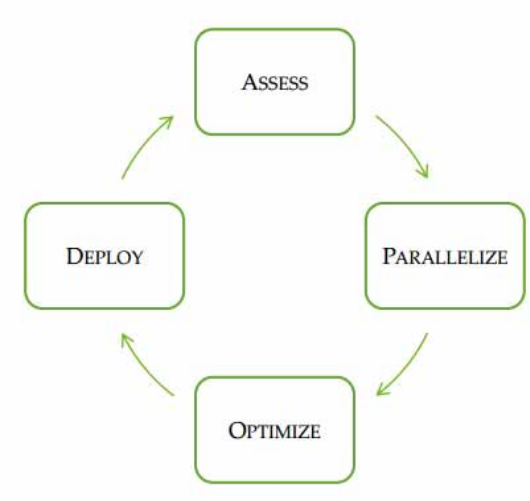



Figura 25: Ciclo APOD. Fonte: (NVIDIA Corporation, 2012).

O primeiro passo é avaliar, ou seja, em uma aplicação já existente, o código fonte deve ser estudado para que as modificações feitas tenham resultados significativos. Devem ser recodificadas apenas os fragmentos que são responsáveis por um tempo de processamento significativo, tendo em vista o programa como um todo.

Depois de feita a avaliação, deve ser feita a paralelização, dependendo do código esta é uma tarefa simples, pois pode ser feita com poucas modificações no código, entretanto esta pode ser uma etapa muito complexa, envolvendo uma remodelagem do problema que possibilite a sua paralelização.

O terceiro passo é a otimização, existem muitas possibilidades de otimizar o código, entretanto essa etapa não necessariamente deve otimizar ao máximo, este processo pode ser feito de forma incremental, otimizando o código a cada iteração.

O último passo, que é desenvolver, é a etapa na qual é possível avaliar o ganho de desempenho obtido, diferente da primeira etapa que somente estima qual pode ser o ganho com determinada modificação.

### 5.4.2 *Chroma key*

O método do *chroma key*, descrito por Van den Berg & Lalioti (1999), analisa cada pixel da imagem de forma independente, por este motivo foi implementado um *kernel* único, que faz a leitura de todos os pixeis da imagem, dessa forma foi criada uma *thread* para cada pixel.

Na calibração dos blocos foi escolhido um valor equivalente à capacidade computacional da placa utilizada, ou seja respeitando o valor máximo de 8 blocos residentes por multiprocessador e também o valor máximo de 1536 *threads* por multiprocessador. Os blocos foram calibrados com duas dimensões, com os valores de (16,16). Como 16 x 16 = 256, são criadas 256 *threads* por bloco e como 1536 / 256 = 6, são alocados 6 blocos no multiprocessador, que é menor que 8. Dessa forma é alcançada uma ocupação máxima dos multiprocessadores, diminuindo o tempo ocioso dos microprocessadores.

Seguindo a metodologia APOD de implementação, percebeu-se que o programa demanda pouco processamento, dessa forma modificações no código para tentar alcançar o acesso coalescido da memória, ou o uso de memórias cache não foram feitos, pois a modificação na estrutura dos dados acarretaria em um tempo de processamento superior ao do programa.

### 5.4.3 Automatização do *trimap* com uso *chroma key*

O algoritmo foi codificado em CUDA, nele foi criado um *kernel* único para ler todos os pontos da imagem, ou seja cada *thread* foi alocada para ler um pixel, pois a leitura e escrita de cada um deles é independente, semelhante ao que ocorre no algoritmo do *chroma key* apresentado.

A configuração dos *kernels* foi implementada para alcançar uma ocupação máxima dos multiprocessadores, semelhante ao que foi descrito na subseção 5.4.2.

### 5.4.4 *Geodesic* modificado

Foram substituídas algumas funções da implementação do método sequencial, para possibilitar o seu funcionamento para o tempo real.

Primeiramente foram alocados na memória da placa gráfica os vetores que para armazenar as imagens. A imagem de entrada, juntamente com a imagem do *trimap*, foram copiados da memória RAM para a memória global da placa.

Foi implementado um *kernel* para transformar a imagem colorida em uma imagem de tons de cinza. Gerando uma *thread* para processar cada pixel. Com um código semelhante ao da implementação sequencial foram gerados os rabiscos, entretanto os dois laços externos foram substituídos pelo *kernel*, no qual cada iteração dos laços foi substituída por uma *thread*. Ambos os *kernels* seguiram a mesma calibração utilizada na implementação do *khroma key*, para obter uma maior ocupação da placa gráfica.

Para a geração das PDFs foi implementada a mesma lógica da implementação sequencial, entretanto neste *kernel* foi colocado todo o conteúdo que antes era interno a um laço, este que executava 256 vezes, desta forma ele foi calibrado com apenas um bloco contendo 256 *threads*.

Com as PDFs já criadas foi implementada também o *kernel* de geração dos relevos, este que por substituir dois laços aninhados também foi calibrado de forma equivalente ao código do *chroma key*.

Posterior a isso foram copiados todos os dados gerados na memória da placa gráfica para a memória RAM e foi desalocado da placa tudo o que foi alocado anteriormente.

O restante do programa foi mantido igual a implementação sequencial. É importante ressaltar que esta implementação foi feita visando o funcionamento do programa com restrição de tempo para uma imagem pequena, dessa foram os caminhos geodésicos calculado em CPU são feitos rapidamente, diferente do que ocorre com imagens maiores.

## 5.5 Considerações Finais

A partir da implementação dos métodos propostos foi possível desenvolver ferramentas para todo o *pipeline* do *matting* digital. O uso da programação CUDA auxilia no uso dessas ferramentas em aplicações com restrição de tempo. No próximo capítulo serão apresentados e analisados a partir das implementações descritas no presente capítulo.

# Capítulo 6

# Experimentos e Análise de Resultados

Nesta capítulo serão mostrados os testes realizados, assim como os seus resultados e análises.

## 6.1 Considerações Iniciais

Foram feitos testes comparativos, para mostrar qual a qualidade das imagens comparando com o *Robust Matting*, que é considerado por OWENS et al. (2007) como o melhor algoritmo. Além disso, como o objetivo do projeto é aplicação em tempo real, foram comparados também os tempos de processamento dos métodos desenvolvidos com esta finalidade.

## 6.2 Métricas utilizadas

Para analisar os resultados foram utilizadas as métricas descritas nessa subseção.

### 6.2.1 Speedup

*Speedup* é uma forma de medir a eficiência, comparando dois programas, um paralelo e outro sequencial. Ele é definido pela Equação 6.2.1 (ALBA, 2005).

$$S_p = \frac{T_1}{T_p} \tag{6.2.1}$$

Onde: $S_p$ é o resultado do speedup;

$T_1$ é o tempo de execução do algoritmo sequencial;

$T_1$ é o tempo de execução do algoritmo paralelo.

### 6.2.2 SSIM

O método SSIM (*Structural Similarity*) é um índice que mede a similaridade entre duas imagens. Para fazer a medida é comparada uma imagem sem distorção, com a imagem que deve ser analisada. Este método é uma tentativa de analisar a imagem de acordo com o olho humano, faz análises na mudança estrutural das imagens (WANG et al., 2004). O cálculo é feito pela Equação 6.2.2, em cada janela .

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \tag{6.2.2}$$

Onde: x representa a janela da primeira imagem;

y representa a janela da segunda imagem;

$\mu_x$ é a média de x;

$\mu_y$ é a média de y;

$\sigma_x^2$ é a variância de x;

$\sigma_y^2$ é a variância de y;

$\sigma_{xy}$ é a covariância entre x e y;

$c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$;

L é o valor máximo dos pixeis (255 para imagens em escala de cinza);

$k_1$e $k_2$ são constantes que devem ter valores muito pequenos.

Para medir a qualidade da imagem como um todo é utilizada a média dos valores calculados em todas as janelas, sendo que é criada uma para cada pixel. O resultado varia entre -1 e 1, sendo que quanto mais próximo de 1, as imagens são mais parecidas.

### 6.2.3 UIQI

O método UIQI (*Universal Image Quality Index*), que mede as distorções entre duas imagens, combinando três fatores, que são: perda de correlação, distorção da luminância e distorção do contraste (ZHOU; BOVIK, 2002). O UIQI é definido pela Formula 6.2.3.

$$Q = \frac{4\sigma_{xy}\bar{x}\bar{y}}{(\sigma_x^2 + \sigma_y^2)[(\bar{x}^2 + \bar{y}^2)]}, \tag{6.2.3}$$

Onde: x representa os pixeis da primeira imagem;

y representa os pixeis da segunda imagem;

$\bar{x}$ é o valor médio de x;

$\bar{y}$ é o valor médio de y;

$\sigma_x^2$ é a variância de x;

$\sigma_y^2$ é a variância de y;

$\sigma_{xy}$ é a covariância entre x e y;

O valor de Q varia entre -1 e 1, sendo 1 as imagens totalmente iguais e -1com as imagens muito diferentes.

## 6.3 Testes iniciais

Para todos os testes em CUDA a quantidade de blocos e *threads* por bloco foi escolhida para manter a maior ocupação possível dos multiprocessadores, assim como com a criação mínima de *threads* ociosas, como foi mostrado no Capítulo 5.

Os primeiros testes foram feitos a partir do algoritmo de Van den Bergh & Lalioti (BERGH; LALIOTI, 1999), com fundo azul (vide Figuras 26, 27 e 28), para serem comparados os tempos de processamento na arquitetura CUDA, com os tempos de processamento seriais. No computador com a placa gráfica GTS 450 e processador Intel Core i7 860, foram feitos testes com imagens de diversos tamanhos. Em todos eles o processamento sequencial foi mais rápido, pois a quantidade de instruções do algoritmo é muito pequena, não compensando assim a transferência dos dados para a placa gráfica. É importante frisar que o processamento em CUDA sem medir transferência de dados é muito superior e quanto maior a imagem menores são as diferenças de tempo, conforme mostra a Tabela 1. Para a medida de tempo, o programa foi executado 100 vezes e os valores são as médias dos tempos.

Tabela 1: Comparação de tempos Van den Bergh & Lalioti (tempos em milissegundos).

| Tamanho | CPU | GPU | GPU sem transferência de memória |
|---|---|---|---|
| 2560x920 | 13,09 | 20,58 | 3,06 |
| 1280x960 | 2,98 | 5,50 | 0,79 |
| 640x480 | 0,78 | 1,46 | 0,24 |
| 320x240 | 0,16 | 0,45 | 0,09 |

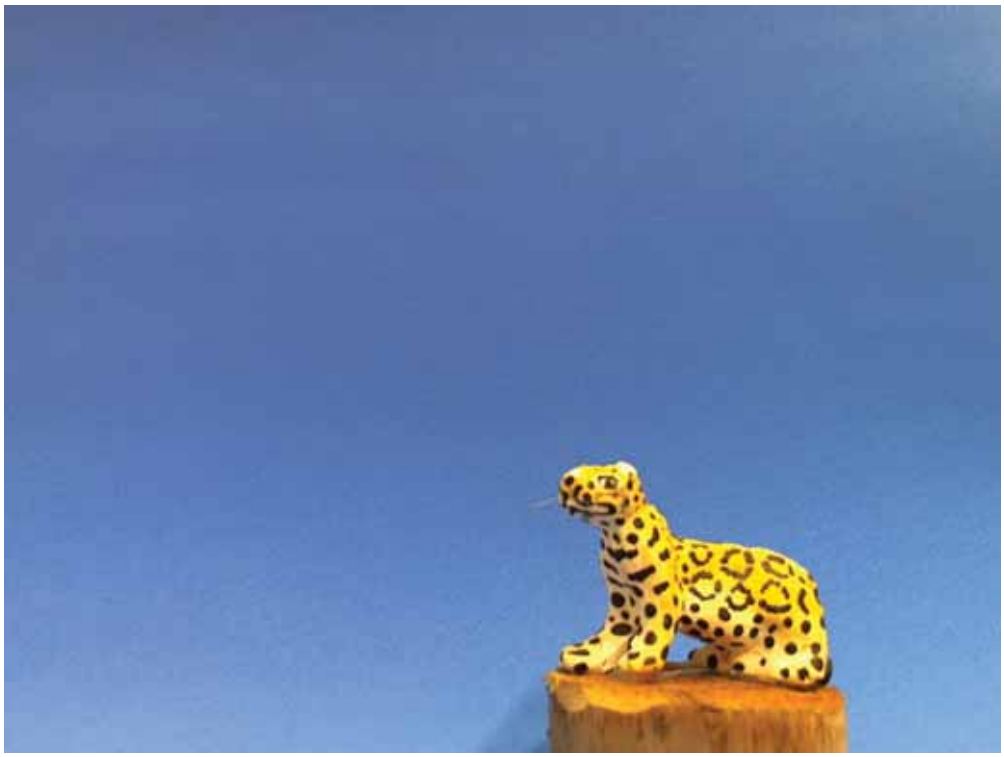


Figura 26: Imagem original.

Figura 27: Nova imagem de fundo.

Figura 28: Composição com o algoritmo de Van den Bergh & Lalioti (BERGH; LALIOTI, 1999).

Para os vídeos (vide figura 29) foi utilizando o computador com a placa gráfica GTX 480 e o processador Intel Core i7 930, foi calculado o tempo médio de processamento de todos os quadros de em cada um deles. Diferente do teste anterior, neste caso foi considerado também o tempo de aquisição da imagem.

Os resultados reforçam o que ocorreu nos testes anteriores, conforme mostra a Tabela 2, como o processamento é com poucas instruções, não é viável fazer as cópias entre as memórias, dessa forma este método só deve ser utilizado em GPU se for associado a algum outro método. Outro fator importante é que quanto maior o tamanho da imagem o ganho de desempenho foi maior.

Tabela 2: Comparação de tempos em vídeos Van den Bergh & Lalioti (tempos em milissegundos).

| Vídeo | Tamanho | CPU | GPU | GPU sem transferência de memória |
|---|---|---|---|---|
| Vídeo 1 | 160x120x353 | 0,099 | 0,278 | 0,023 |
| Vídeo 2 | 160x120x2191 | 0,099 | 0,280 | 0,024 |
| Vídeo 1 | 320x240x353 | 0,387 | 0,489 | 0,022 |
| Vídeo 2 | 320x240x2191 | 0,381 | 0,511 | 0,022 |
| Vídeo 1 | 640x480x353 | 1,429 | 1,441 | 0,033 |
| Vídeo 2 | 640x480x2191 | 1,456 | 1,460 | 0,035 |

Figura 29: Comparação entre as imagens original e com troca do plano de fundo.

No algoritmo de geração automática de *trimap*, os valores para as distâncias do algoritmo de Van den Bergh & Lalioti (BERGH; LALIOTI, 1999) foram geradas a partir de diversas tentativas, para encontrar um valor ideal. Percebeu-se que, em um mesmo ambiente sem mudanças na iluminação e na tonalidade do plano de fundo não foi necessário modificar estes valeres, ou seja, para um determinado ambiente, após ser calibrado, não há necessidade de intervenção humana. Os resultados foram iguais para ambos os programas (vide Figura 30).

Figura 30: Geração automática de trimap, a figura da esquerda é gerada pela CPU, a figura da direita gerada pela GPU.

Para este algoritmo todos os testes com a arquitetura CUDA obtiveram resultados melhores (ver Tabela 3) do que utilizando o código sequencial, ou seja todos os tempos foram menores.

Tabela 3: Comparação de tempos geração automática de trimap (tempos em milissegundos).

| Tamanho | CPU | GPU | GPU sem transferência de memória |
|---|---|---|---|
| 2560x920 | 21,52 | 13,96 | 5,60 |
| 1280x960 | 5,45 | 3,72 | 1,40 |
| 640x480 | 1,1 | 0,99 | 0,38 |
| 320x240 | 0,47 | 0,31 | 0,13 |

## 6.4 Resultados utilizando o *trimap* gerado pelo kinect e *chroma key*

Foram feitos testes utilizando a automatização do *trimap* utilizando kinect e *chroma key.* A Figura 31 mostra a foto que será recortada, a Figura 32 mostra o mapa de profundidade, a Figura 33 mostra o *trimap* gerado pelo método, a Figura 34 mostra o matting feito com o *Baeysian Matting* (CHUANG et al., 2001) e a Figura 35 mostra o *matting* feito com o *Robust Matting* (WANG; COHEN, 2007a).

Figura 31: Foto com plano de fundo azul.

Figura 32: Mapa de profundidade gerado pelo Kinect.

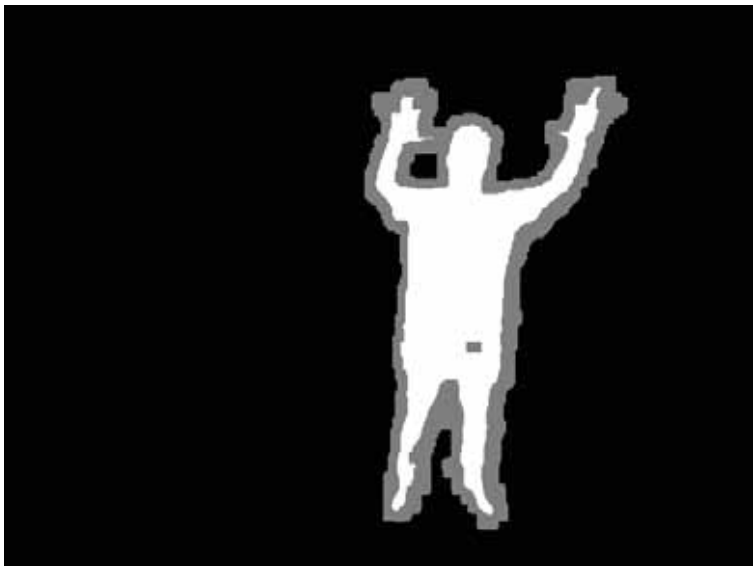


Figura 33: *Trimap* gerado pelo Kinect.

Figura 34: Recorte feito pelo *Baeysian Matting* (CHUANG et al., 2001), com Kinect.

Figura 35: Recorte feito pelo *Robust Matting* (WANG; COHEN, 2007a), com Kinect.

A partir dos resultados mostrados foi possível perceber que as imagens não ficaram visualmente muito boas, pois a qualidade da câmera RGB do Kinect é baixa e o *trimap* gerado possui uma área de incerteza muito grande. Foi possível observar também que a partir dos resultados mostrados o *Robust Matting* (WANG; COHEN, 2007a) é menos eficiente que o *Baeysian matting* (CHUANG et al., 2001), quando é utilizado esse tipo de *trimap*. Por esses motivos não foi feita uma implementação paralela do *trimap* gerado pelo Kinect e *chroma key.*

## 6.5 Resultados utilizando o *trimap* gerado por *chroma key*

Foi utilizado um estúdio de *chroma key* com fundo verde, com uma cor uniforme, conforme mostrado na Figura 36. Neste ambiente a câmera fullHD foi acoplada a um tripé com regulagem de altura e nível. Os testes foram realizados com o ator a uma distância de 90cm da câmera e a 100cm da parede, gerando o *trimap* da Figura 37.

Figura 36: Foto original.

Figura 37: Trimap gerado automaticamente.

É possível observar que o *trimap* gerado possui uma pequena área de incerteza, dessa forma diminuindo a área em que o cálculo do alfa será feito.

Foram feitos testes utilizando as implementações dos métodos *Baeysian matting* (CHUANG et al., 2001) e o *Robust Matting* (WANG; COHEN, 2007a). Estes que foram realizados para demonstrar a eficiência do *trimap* gerado nos algoritmos de *Matting digital*, conforme mostram as Figuras 38 e 39, foi colocado um fundo branco para ficarem visíveis os defeitos presentes nas imagens.

Figura 38: Recorte utilizando o *Robust Matting* (WANG; COHEN, 2007a), com novo plano de fundo branco.

Figura 39: Recorte utilizando o *Baeysian matting (CHUANG et al., 2001),*com novo plano de fundo branco.

A partir dos testes foi possível observar que o *Baeysian Matting* gera resultados visualmente melhores do que no *Robust matting* quando há um *trimap* pouco preciso, entretanto os resultados do *Robust matting* são visualmente mais precisos quando a área de incerteza do *trimap* é menor, como mostrado nas Figuras 38 e 39.

Outro teste equivalente foi feito, entretanto com um novo plano de fundo, para demonstrar que em uma aplicação real as falhas do método são menos perceptíveis, como mostrado nas Figuras 40 e 41.

Figura 40: Recorte utilizando o *Robust Matting* (WANG; COHEN, 2007a), com novo plano de fundo.

Figura 41: Recorte utilizando o *Baeysian matting (CHUANG et al., 2001),* com novo plano de fundo.

## 6.6 Análise da qualidade das imagens

Para todos os testes utilizando o algoritmo *Geodesic* Modificado, o valor da variável r é de 1,5 e o *trimap* utilizado foi gerado a partir do método de geração automática de *trimap* a partir do *chroma key.*

Segundo o WANG & COHEN (2007b) o método *Robust Matting* (WANG; COHEN, 2007a) é considerado como o de melhor resultado, dessa foram foram aplicados testes para comparar as qualidades das imagens, comparando-as com o resultado do *Robust Matting* (Figura 38), como mostrado na Tabela 4. Os testes utilizados foram UIQI e SIIM, para comparação dos resultados foram utilizadas as Figura 42 e 43.

Tabela 4: Comparação de qualidade entre o *Robust Matting* e os demais métodos.

| Método utilizado | UIQI | SSIM |
|---|---|---|
| *Bayesian Matting* | 0,9950 | 0,9997 |
| *Geodesic Matting* modificado (implementação em CPU) | 0,9993 | 0,9894 |
| *Geodesic Matting* modificado (implementação em GPU) | 0,9993 | 0,9894 |

Figura 42: Recorte utilizando o *Geodesic Matting* modificado, com implementação squencial.

Figura 43: Recorte utilizando o *Geodesic Matting* modificado, com implementação paralela.

Com estes resultados é possível perceber que as imagens resultantes ficaram muito parecidas, pois para todos os testes feitos os resultados se aproximaram do valor 1, em ambos os testes.

Para saber se a imagem gerada pelo programa paralelo (Figura 42) é equivalente a imagem gerada pelo programa em sequencial (Figura 43), foram feitos os mesmos testes, comparando as imagens geradas em CPU, com as imagens geradas pela GPU, nos quais o resultados do UIQI e do SSIM foram de 0,9999 e de 1,0000, respectivamente.

Com estes resultados é possível mostrar que as imagens são muito semelhantes, a diferença mostrada pelo UIQI é devido ao fato de as aproximações numéricas feitas pela placa gráfica não serem iguais as do processador, outro fator que justifica isso é a forma como a imagem RGB foi transformada em uma imagem de tons de cinza, neste processo houve uma pequena diferença entre as duas.

Foi criada também a imagem com um novo plano de fundo, para demonstrar o seu uso em aplicações reais, mostrado na Figura 43. Foi mostrada apenas a imagem feita com o programa paralelo, pois os resultados são muito semelhantes, como já foi mostrado pela Tabela **??**.

Figura 44: Recorte utilizando o *Geodesic Matting* modificado, com implementação paralela e novo plano de fundo.

Todos os testes mostram que as imagens resultantes são muito semelhantes, independente do método utilizado, isso pode ser auxiliado, também, por todos os testes executados usarem o *trimap* gerado automaticamente pelo *chroma key*, dessa forma fica evidente que o *trimap* é preciso e gera resultados semelhantes, independente do método aplicado.

## 6.7 Testes em vídeo utilizando utilizando todo o pipeline

Foram feitos testes comparando as implementações sequencial e paralela em vídeos. As Tabelas 5 e 6 mostram o número de FPS (*Frames Per Second*) de cada um, ou seja a taxa de quadros por segundo na qual o vídeo é mostrado na tela. O resultado pode ser visto na Figura 45, foi colocado o fundo na cor rosa para mostrar que as partes brancas da imagem não foram recortadas.

Tabela 5: Implementação sequencial.

| Tamanho (largura x altura) | FPS |
|---|---|
| 320 x 240 | 42,4 |
| 640 x 480 | 13,4 |

Tabela 6: Implementação paralela.

| Tamanho (largura x altura) | FPS |
|---|---|
| 320 x 240 | 63,3 |
| 640 x 480 | 19,6 |

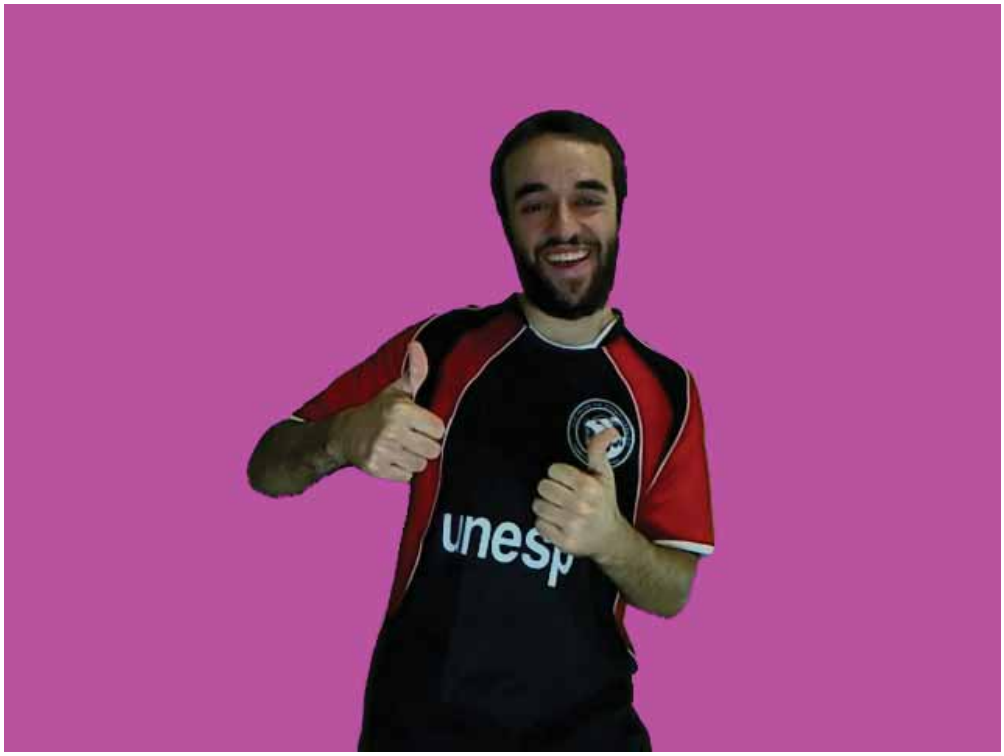



Figura 45: Figura de um quadro gerado a partir do vídeo.

Com esses testes é possível observar que a implementação utilizando a arquitetura CUDA foi mais eficiente que a implementação sequencial, dessa forma possibilitando uma exibição de um número maior FPS em ambos os vídeos.

## 6.8 Considerações Finais

O *trimap* gerado automaticamente por *chroma key* mostrou-se muito eficiente. A qualidade das imagens geradas pelo método de *matting* digital proposto são semelhantes ao método considerado

como melhor, por (OWENS et al., 2007), tanto na implementação sequencial quanto na paralela. Além disso com a utilização de todo o *pipeline* é possível fazer o matting de vídeos com uma quantidade aceitável de quadros por segundo, com uma maior taxa de FPS na implementação utilizando CUDA.

# Capítulo 7

# Conclusões e Trabalhos Futuros

Neste capítulo são apresentadas as conclusões e sugestões para trabalhos futuros.

## 7.1 Conclusões

Os testes iniciais foram feitos com a utilização de um método simples, que não prevê, por exemplo, transparências ou fumaça. Na comparação entre a CPU e a GPU, a segunda foi mais eficiente que a primeira no processamento da imagem, desconsiderando o tempo de transferência de memória. Dessa forma o tempo de processamento total, utilizando CUDA, é maior.

Já no algoritmo de geração automática do *trimap* por *chroma key*, por haver mais instruções e uma quantidade maior de acesso a memória, o algoritmo paralelizado mostrou-se mais eficiente para todos os testes realizados.

Existem métodos de *matting* digital que são implementadas utilizando a arquitetura CUDA, como descrito por (OWENS et al., 2007). Dessa forma é uma vantagem utilizar o algoritmo de geração automática de *trimap* já implementado nesta arquitetura, pois assim a transferência de dados da CPU para a GPU é feita apenas uma vez. Como o que foi feito com o *Geodesic* modificado, possibilitando o seu uso em aplicações com restrição de tempo.

Utilizando o Kinect como informação extra, os algoritmos de não geram a mesma qualidade de *trimap*, entretanto, a partir dele é possível gerar o *trimap* em ambientes de teste mais simples do que o que foi utilizado na geração automática a partir do *chroma key*.

As análises feitas mostram que o uso da arquitetura CUDA pode acelerar o algoritmo do *chroma key*, todavia a transferência de dados é o principal fator limitante. Assim os resultados sugerem que com processamentos maiores e algoritmos mais complexos o seu uso pode ser justificado.

## 7.2 Trabalhos Futuros

Como desdobramento deste trabalho, sugere-se os seguintes trabalhos futuros:

- Aplicação de outros métodos de resolução de área de incerteza (tais como, Poisson *Matting* e o *Random Ralks*);

- Aperfeiçoar o algoritmo Geodesic modificado, visando maior desempenho com imagens de alta resolução, utilizando, para isto, a continuação da abordagem APOD no método paralelizado;

- Uso do Kinect como informação extra, acoplado juntamente com uma câmera fullHD, utilizando ou não a técnica de chroma key. Isto viabilizaria sua utilização em aplicações de Estúdios Virtuais e Videoconferência.

- Uso do Kinect como informação extra, acoplado juntamente com uma câmera fullHD, utilizando ou não o *chroma key*. Dessa forma possibilita o seu uso em estúdios de baixo custo ou em conferências.

# Referências Bibliográficas


ALBA, E. *Parallel Metaheuristics: A New Class of Algorithms*. NJ, USA: John Wiley & Sons, 2005. ISBN 0-471-67806-6. Disponível em: <http://www.ebookmall.com/ebooks/parallel-metaheuristics-a-new-class-of-algorithms-alba-ebooks.htm; http://www.bibsonomy.org/bibtex/2499f9a511ff28e6ccf55017ddc2d080a/brazovayeye>.

BAI, X.; SAPIRO, G. Geodesic Matting: A Framework for Fast Interactive Image and Video Segmentation and Matting. *International Journal of Computer Vision*, v. 82, n. 2, p. 113–132, 2009. Disponível em: <http://dblp.uni-trier.de/db/journals/ijcv/ijcv82.html; http://dx.doi.org/10.1007/s11263-008-0191-z; http://www.bibsonomy.org/bibtex/2f67478ee6152bf618598ce03c4ba9be6/dblp>.

BERGH, F. van den; LALIOTI, V. Software chroma keying in an immersive virtual environment. *South African Computer Journal*, Citeseer, v. 24, p. 155–162, 1999.

BOWMAN, A. W.; AZZALINI, A. Computational aspects of nonparametric smoothing with illustrations from the sm library. *Computational Statistics & Data Analysis*, v. 42, n. 4, p. 545–560, 2003. ISSN 0167-9473. Disponível em: <http://www.sciencedirect.com/science/article/B6V8V-4619K7B-1/2/2bdc293a677140f136b4bc7144e906d7; http://www.bibsonomy.org/bibtex/2a19b91b2c2a57b3b1f835f20c1dc7a6a/joelotz>.

CHENG, J. Programming Massively Parallel Processors. A Hands-on Approach. *Scalable Computing: Practice and Experience*, v. 11, n. 3, 2010. Disponível em: <http://dblp.uni-trier.de/db/journals/scpe/scpe11.html>.

CHUANG, Y. et al. A bayesian approach to digital matting. In: IEEE. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. [S.l.], 2001. v. 2, p. II–264.

GASTAL, E. S. L.; OLIVEIRA, M. M. Shared Sampling for Real-Time Alpha Matting. *Comput. Graph. Forum*, v. 29, n. 2, p. 575–584, 2010. Disponível em: <http://dblp.uni-trier.de/db/journals/cgf/cgf29.html; http://dx.doi.org/10.1111/j.1467-8659.2009.01627.x; http://www.bibsonomy.org/bibtex/2f8d2538cd899e3c98836b4c22397bee3/dblp>.

KÜHN, T. The Kinect Sensor Platform. *ADVANCES IN MEDIA TECHNOLOGY*, p. 1, 2011.

NVIDIA, C. Nvidia cuda c programming guide. *NVIDIA Corporation*, 2011.

NVIDIA Corporation. *NVIDIA CUDA C Best Practices Guide*. 2012. Version 3.2.

OWENS, J. et al. A survey of general-purpose computation on graphics hardware. *Warning: the year was guessed out of the URL*, 2007.

PORTER, T.; DUFF, T. Compositing digital images. *ACM Siggraph Computer Graphics*, ACM, v. 18, n. 3, p. 253–259, 1984.

REISS, M.; TOMMASELLI, A. Reconstrução 3D por Luz Estruturada: Calibração dos Vetores Diretores dos Feixes de Padrões Projetados. In: *XXI CONGRESSO BRASILEIRO DE CARTOGRAFIA, Belo Horizonte, MG*. [S.l.: s.n.], 2003.

RUZON, M. A.; TOMASI, C. Alpha Estimation in Natural Images. In: *CVPR*. IEEE Computer Society, 2000. p. 1018–1025. ISBN 0-7695-0662-3. Disponível em: <http://dblp.uni-trier.de/db/conf/cvpr/cvpr2000.html; http://computer.org/proceedings/cvpr/0662/Volume; http://06621018abs.htm; http://www.bibsonomy.org/bibtex/2bde4f3a5e4c5b1f43e381fa3f6943844/vonolfen>.

SANDERS, J.; KANDROT, E. *CUDA by example: an introduction to general-purpose GPU programming*. [S.l.]: Addison-Wesley Professional, 2010.

SILVERMAN, B. W. *Density estimation for statistics and data analysis*. London: Chapman and Hall, 1986. Disponível em: <http://www.bibsonomy.org/bibtex/209d072207f82bbc4d702174f670a2f02/josephausterwei>.

SMITH, A. R.; BLINN, J. F. Blue screen matting. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1996. (SIGGRAPH '96), p. 259–268. ISBN 0-89791-746-4. Disponível em: <http://doi.acm.org/10.1145/237170.237263>.

SUN, J. et al. Flash matting. *ACM Trans. Graph.*, v. 25, n. 3, p. 772–778, 2006. Disponível em: <http://dblp.uni-trier.de/db/journals/tog/tog25.html; http://doi.acm.org/10.1145/1141911.1141954; http://www.bibsonomy.org/bibtex/22489127bd201582390f5a156ccdde50d/dblp>.

VLAHOS, P. *Composite photography utilizing sodium vapor illumination (us patent 3,095,304)*. [S.l.]: May, 1958.

WANG, J.; COHEN, M. Optimized color sampling for robust matting. In: IEEE. *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. [S.l.], 2007. p. 1–8.

WANG, J.; COHEN, M. F. Image and Video Matting: A Survey. *Foundations and Trends in Computer Graphics and Vision*, v. 3, n. 2, p. 97–175, 2007. Disponível em: <http://dblp.uni-trier.de/db/journals/ftcgv/ftcgv3.html; http://dx.doi.org/10.1561/0600000019; http://www.bibsonomy.org/bibtex/2db3549f69d8ab8638c942f4dadcc43b4/dblp>.

WANG, Z. et al. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, v. 13, n. 4, p. 600–612, 2004. Disponível em: <http://dblp.uni-trier.de/db/journals/tip/tip13.html; http://dx.doi.org/10.1109/TIP.2003.819861; http://www.bibsonomy.org/bibtex/2d235adedbae104af6b40f76d4f50e2d8/lychen1109>.

ZHOU, W.; BOVIK, A. C. A Universal Image Quality Index. *IEEE SIGNAL PROCESSING LETTERS*, XX, mar. 2002.