Um Algoritmo para Correspondência Estéreo baseado em GPU

Amilton Sales Reis Junior
Instituto Federal da Bahia
Salvador, Brasil
amiltonjunior3@hotmail.com

Antonio Carlos dos Santos Souza
Instituto Federal da Bahia
Salvador, Brasil
acsantossouza@gmail.com

RESUMO

Nas últimas décadas, vários algoritmos têm sido propostos para resolver o problema de correspondência estéreo. Contudo, a maioria dos métodos propostos foram desenvolvidos com o objetivo de melhorar a precisão das soluções já existentes e pouco foco tem sido dado no que diz respeito ao desempenho dessas soluções. Neste trabalho, será realizado o desenvolvimento de um algoritmo para correspondência estéreo que tenha uma margem de tempo de execução menor que 0,5 segundos para variados tamanhos de imagens. Para tanto, será construída uma nova abordagem com foco em paralelização para correspondência estéreo a qual agregue e adapte soluções já existentes na literatura na Unidade de Processamento Gráfico.

Palavras-chave

Correspondência estéreo; CUDA; *Census Transform*

1. INTRODUÇÃO

Visão estéreo é o ramo da visão computacional que estuda o problema da reconstrução da informação tridimensional de objetos a partir de um par de imagens capturadas simultaneamente, mas com um pequeno deslocamento[7]. No ser humano, a visão estéreo é uma das principais informações de profundidade na visão, uma vez que o olho esquerdo e o olho direito sempre veem imagens diferentes, apesar de muito parecidas, e a partir dessas diferenças (ou disparidades), a noção de profundidade é construída pelo cérebro, levando ao entendimento do volume que está sendo visto.

A tecnologia da visão estéreo é bastante utilizada em várias áreas, como a reconstrução de cenas tridimensionais ou, por exemplo, em robótica no momento em que um robô precisa conhecer a localização de certos objetos(Figuras 1 e 2), assim como a diferença entre eles, a partir de informações de profundidade e isso pode auxiliar seu módulo de navegação.

Dentre os inúmeros passos necessários para recuperar a informação tridimensional a partir das duas imagens dadas pelo sistema de visão estéreo, um deles consiste justamente na correspondência estéreo, ou seja, no cálculo dos *pixels* correspondentes entre as imagens dadas pelo sistema estéreo. A partir da correspondência dos *pixels* nas duas imagens, é possível construir um mapa de disparidade, que consiste justamente em uma imagem que mede a distância, em *pixels*, entre um *pixel* de referência em uma imagem base e o seu *pixel* correspondente na imagem alvo.

Dois dos principais problemas atrelados aos algoritmos de correspondência estéreo são a precisão e o desempenho (em termos de tempo de processamento) desses algoritmos. Apesar de muitos algoritmos serem desenvolvidos a cada ano, ainda não foi desenvolvida uma solução capaz de manter boa precisão, contudo com baixo tempo de processamento. Uma das razões para que isso ocorra está no fato de que as técnicas mais utilizadas para a realização da correspondência estéreo com boa precisão não são facilmente paralelizáveis para uma implementação utilizando-se uma Unidade de Processamento Gráfico (GPU). Dessa forma, desenvolver um método para correspondência estéreo que balanceie precisão e custo computacional ainda permanece um problema desafiador.

As técnicas mais conhecidas na visão estéreo (uma lista está disponível no site do Middlebury[3]) utilizam de métodos computacionalmente custosos para resolver o problema de correspondência estéreo, o que impede o uso deles em aplicações que desejam reconstruir a informação tridimensional em tempo real a partir de uma sequência de imagens. Aplicações como o MobileFusion (Figura 3), o qual é um

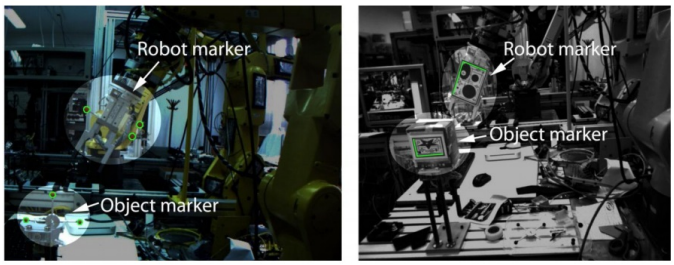

Figura 1: Rastreamento de objeto em um sistema robótico utilizando câmera de visão estéreo.
Imagem retirada de [32]

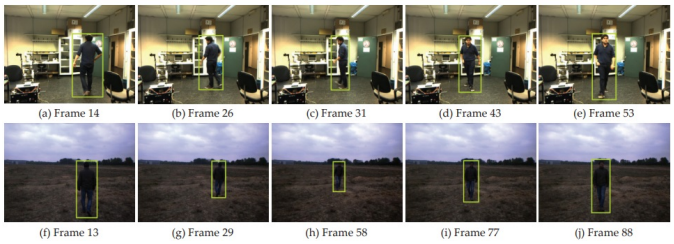

Figura 2: Rastreamento de um ser humano em um sistema robótico.
Imagem retirada de [20]

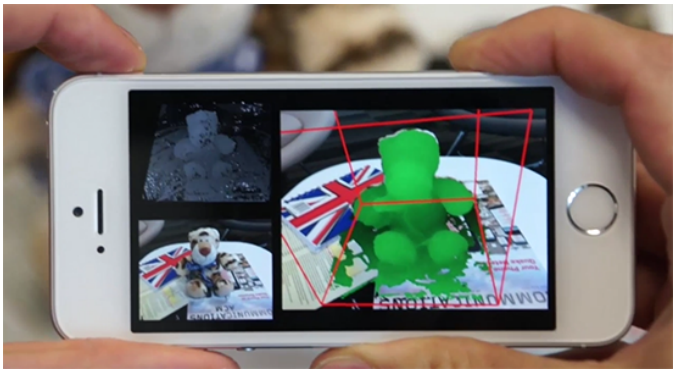

Figura 3: Aplicação para criação de objetos tridimensionais Mobile Fusion.
Imagem retirada de [1]

projeto que tem como objetivo permitir que pessoas criem objetos tridimensionais de alta qualidade em tempo real utilizando um smartphone, precisam realizar muitos cálculos de correspondência estéreo em tempo real[17].

A popularização das GPUs causou um interesse aos pesquisadores em aproveitar seu poder de paralelização para os algoritmos estéreos, resultando na criação de notáveis trabalhos que buscam atingir o tempo real com seus algoritmos[10][13][15][27]. Uma das arquiteturas para programação paralela mais populares é a *Compute Unified Device Architecture* (CUDA), desenvolvida pela NVIDIA[16].

O objetivo deste trabalho é realizar a paralelização de um algoritmo de correspondência estéreo utilizando a plataforma CUDA, buscando atingir resultados relacionados a velocidade de execução muito superiores ao seu correspondente sequencial. Além da comparação dos mesmos algoritmos sendo executado em GPU e em CPU, o tempo de processamento dos algoritmos será medido em imagens de diversos tamanho e, então, feito um comparativo.

A divisão deste trabalho possui a seguinte forma: na Seção 2 é levantada a fundamentação teórica que engloba os conceitos básicos de correspondência estéreo e da arquitetura de programação paralela CUDA; os trabalhos relacionados são estudados logo depois; então, na Seção 3, os algoritmos utilizados para o sistema de correspondência estéreo são apresentados, seguido de como um deles foi paralelizado e, por fim, na Seção 4 é realizada a avaliação do algoritmo paralelizado, a partir da qual é dada a conclusão na Seção 5.

2. FUNDAMENTAÇÃO TEÓRICA

Existem várias maneiras de se desenvolver um algoritmo de correspondência estéreo. As decisões sobre quais métodos utilizar geralmente giram em torno de precisão e velocidade. Nesta seção, serão dadas noções fundamentais para o entendimento desses algoritmos, além da apresentação dos métodos utilizados por algoritmos que priorizam velocidade. Após isto, será apresentada a plataforma para programação paralela CUDA e realizado o estudo sobre trabalhos que a utilizam para resolver o problema de correspondência estéreo.

2.1 Processamento de Imagens

Uma imagem é uma distribuição espacial da irradiância

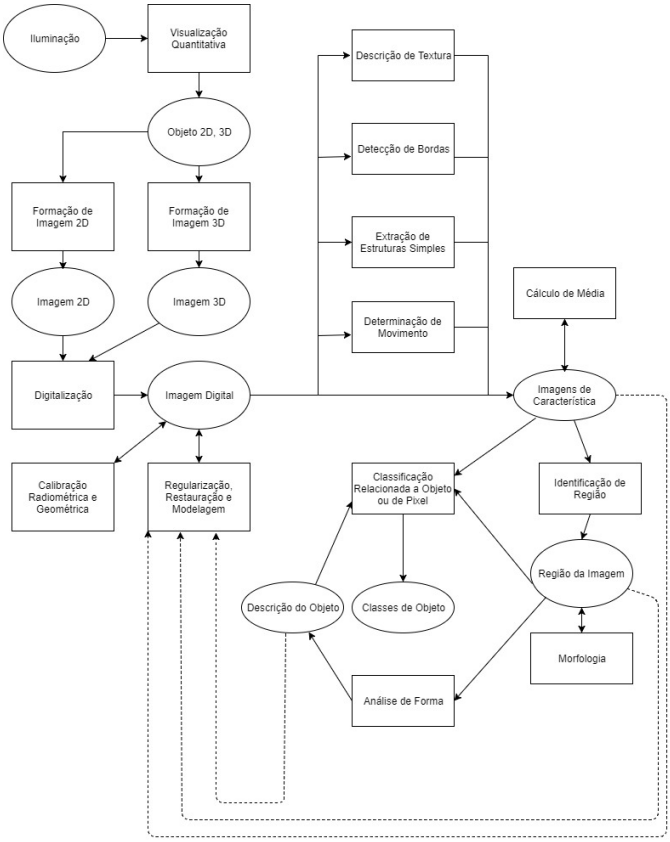

Figura 4: Hierarquia de tarefas de processamento de imagem.
Imagem adaptada de [11]

em um plano e, já que computadores conseguem lidar apenas com *arrays* de números digitais ao invés de imagens contínuas, o processamento de imagens em um computador é feito a partir de representações em *arrays* de duas dimensões, onde cada ponto, ou *pixel*, desses *arrays* representa a irradiância em um ponto da imagem[11].

O processamento de imagens não é algo trivial e geralmente envolve várias etapas, algumas interconectadas, que precisam ser realizadas antes que possamos extrair os dados que nos importam de uma cena observada. Essa sequência de etapas é explicada em detalhes por Jähne[11] como mostrado na Figura 4.

De acordo com Jähne, inicialmente, a aquisição da imagem é feita a partir de um sistema de aquisição para que ela seja tratada computacionalmente em um processo chamado digitalização. O início deste processo se chama de pré-processamento, onde defeitos como distorções geométricas, claridade, contraste etc são melhorados.

Um grande número de etapas é, então, realizado para a análise e identificação de objetos. Essas etapas envolvem métodos de filtragem para a distinção de objetos de interesse e outros que estão no fundo. Então várias características da imagem, como por exemplo a detecção de movimento, são extraídas e algumas das ferramentas básicas e diferentes utilizadas para isso se chamam "Detecção de borda"(Figura 5), "Análise de vizinhanças simples"e "Padrões complexos".

O objeto pode então ser separado dos objetos de fundo

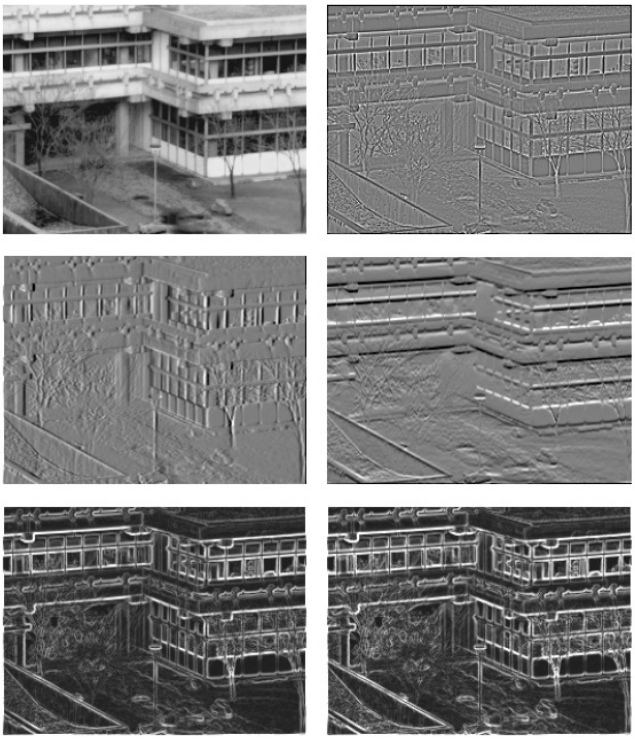

Figura 5: Detecção de bordas utilizando uma variedade de filtros.
Imagem retirada de [11]

em um processo chamado de "Segmentação", onde regiões de características constantes são separadas. Esse pode ser um processo complexo, caso os objetos próximos da imagem não sejam muito distintos uns dos outros, o que na maioria das vezes é o caso.

E, por fim, depois de ter conhecimento das formas dos objetos, operadores morfológicos podem ser usados para analisá-las e modificá-las, ou extrair parâmetros úteis como a área ou escala de cinza do objeto, o que pode ser utilizado para a classificação de objetos.

2.2 Correspondência Estéreo

A partir de um par de imagens de uma mesma cena vista a partir de ângulos levemente diferentes, encontrar qual parte de uma imagem corresponde a qual parte da outra imagem define o problema da correspondência estéreo. O resultado de aplicar correspondência estéreo nesse par de imagens resulta em outra imagem que associa cada um de seus *pixels* com um valor de disparidade. Normalmente, é assumido que as imagens são paralelas em torno do eixo-x para facilitar a descoberta de correspondências entre elas. Desta forma, dado um *pixel p* localizado em *(x, y)* de uma imagem e seu correspondente *p'* localizado em *(x', y)*, a disparidade do *pixel p* é dada pela equação [13]:

$$(p, p') = x - x' \tag{1}$$

Entre os desafios da correspondência estéreo, está o problema de ambiguidade da imagem, o qual é resultado de aparências ambíguas de *pixels* das imagens causadas por ruídos ou texturas semelhantes. Uma forma de reduzir este problema é com o uso de janelas de suporte local (Figura 6) que limitam os cálculos de correspondência em áreas da imagem. Uma janela assume que todos os *pixels* contidos nela possuem profundidade e disparidades similares[28]. Em uma imagem com poucas variações de profundidade, uma janela de suporte de maior tamanho tem melhor confiabilidade para o cálculo de correspondência entre *pixels* e em uma imagem com muitas variações de profundidade, uma janela de suporte menor diminui o número de erros de correspondência, ou seja, existe um *trade off* na escolha do tamanho da janela[9].

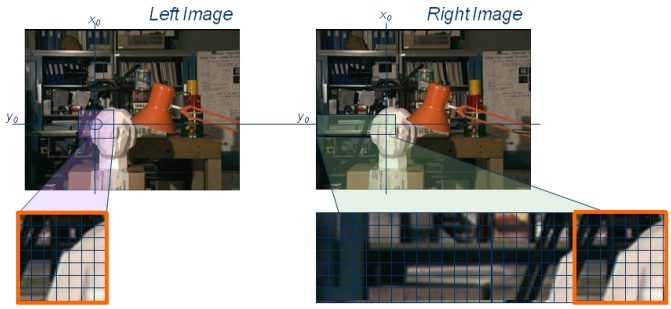

Figura 6: Janela de suporte. Os retângulos laranjas representam o tamanho da janela.
Imagem adaptada de [6]

Algoritmos de correspondência estéreo geralmente são divididos pelas etapas de[14]:

1. Cálculo do custo de correspondência.
2. Agregação de custo.
3. Computação de disparidade.
4. Refinamento de disparidade.

Esses algoritmos podem ser classificados também como locais (os quais utilizam janelas suporte) ou globais, sendo o primeiro grupo mais rápido e o segundo mais preciso. Algumas soluções em tempo real utilizam uma abordagem semi-global[15], as quais misturam características de algoritmos locais e globais. Este trabalho terá maior foco em algoritmos locais para resolver o problema da paralelização de algoritmos de correspondência estéreo.

Nas próximas subseções, serão explicadas em detalhes cada uma das etapas de um algoritmo de correspondência estéreo.

2.2.1 *Cálculo do custo de correspondência*

Esta etapa é responsável por determinar a probabilidade de uma correspondência entre *pixels*. Dois dos algoritmos mais conhecidos para o cálculo de custo de correspondência são *Sum of Absolute Differences* (SAD) e *Sum of Squared Differences* (SSD). Sendo o primeiro mais rápido pois não requer multiplicações comparado ao segundo[9].

O SAD é um algoritmo que foi criado a partir de uma abordagem baseada em *pixels* cujo objetivo é encontrar suas disparidades[19]. Ele calcula a diferença absoluta entre um *pixel* em uma janela de uma imagem e seu *pixel* correspondente em outra imagem do sistema estéreo. Considerando que os *pixels* estão paralelos no eixo horizontal, o algoritmo possui a equação:

$$\sum_{(i,j)\epsilon W} = |I_1(i,j) - I_2(x+i,j)| \quad (2)$$

Onde as coordenadas (i, j) estão dentro de uma janela W, $I_1(i, j)$ representa a localização de um *pixel* em uma janela na primeira imagem e $I_2(x + i, j)$ representa o *pixel* correspondente na janela da segunda imagem.

2.2.2 Agregação de custo

A agregação de custo é importante para criar a suposição de que todos os *pixels* dentro de uma janela possuem o mesmo nível de disparidade.

Algoritmos de correspondência estéreo locais realizam a agregação de custo de correspondências através da soma ou média de valores de uma janela em uma imagem de disparidades[23]. O método local introduzido por Yoon e Kweon[28] chamado *Adaptive support-weights* (ASW), ao provar sua precisão em relação a métodos anteriores, se tornou bastante utilizado como referência para a agregação de custo em algoritmos de correspondência estéreo que buscam o tempo real de processamento. Mais detalhes sobre o ASW serão vistos na Seção 2.4.

2.2.3 Computação e refinamento de disparidade

O foco dos algoritmos de correspondência estéreo locais geralmente está no cálculo do custo de correspondência e agregação de custo. Logo, para calcular a disparidade, eles apenas escolhem a disparidade de menor custo para cada *pixel*[23]. A agregação nesses algoritmos locais faz com que a escolha do *pixel* seja independente dos *pixels* fora de sua janela, logo é mais rápida dos que nos algoritmos globais, os quais buscam por correspondência por todos os *pixels* das imagens. Então muitos dos algoritmos de correspondência estéreo locais utilizam simplesmente um método *"winner-take-all"* (WTA) para a computação de disparidade, podendo ser calculado da seguinte forma:

$$d_p = \arg\min_{d \epsilon D_d} C(p, d) \quad (3)$$

Onde d_p representa a disparidade para um *pixel* p, D_d representa todas as possíveis disparidades e $C(p, d)$ representa o custo de correspondência quando se associa a disparidade d ao *pixel* p[31].

Dependendo das necessidades do sistema em questão, existem várias formas de se realizar o refinamento de disparidades. Algumas delas incluem ajustes de gradiente ou de curvas para custos de correspondências em certos níveis de disparidades.

2.3 CUDA

A *Compute Unified Device Architecture*, cuja sigla é CUDA, é uma plataforma de computação paralela desenvolvida pela NVIDIA para programação em C/C++/Fortran na Unidade de Processamento Gráfico para propósito geral[16].

A popularização do uso das GPUs para cálculos diversos pode ser associada também ao crescimento imenso no poder de processamento paralelo delas observado há poucos anos atrás, enquanto o crescimento das CPUs continua relativamente pequeno (Figura 7).

A programação CUDA é heterogênea, ou seja, segue um modelo de programação onde há o uso da CPU e GPU (o primeiro sendo geralmente chamado de *Host* e o segundo de *Device*) e segue o seguinte fluxo (Figura 8):

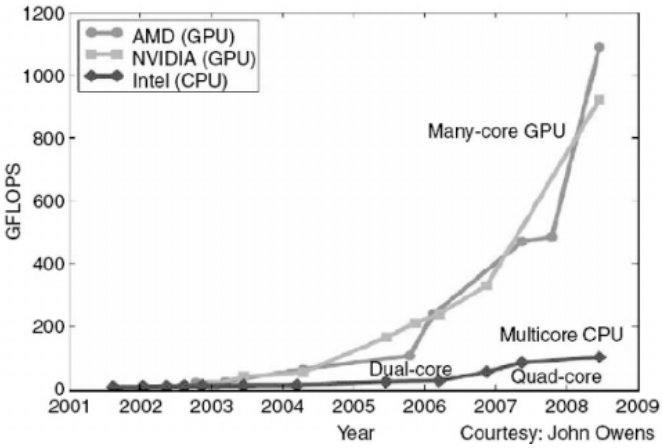

Figura 7: Evolução da CPU x GPU.
Imagem retirada de [12]

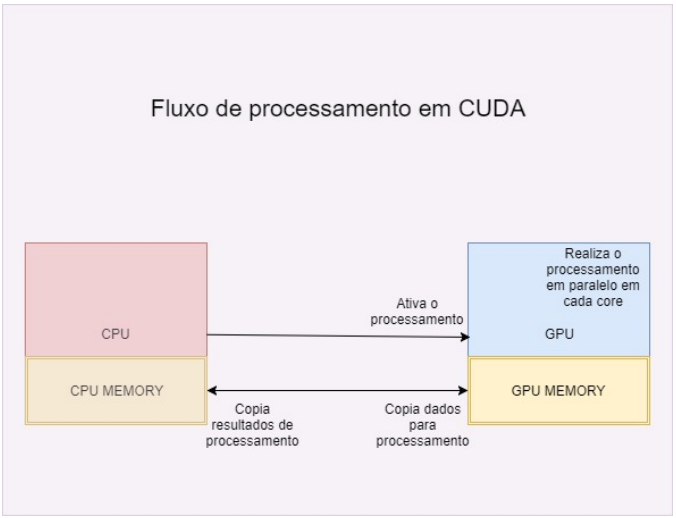

Figura 8: Fluxo de processamento em CUDA.

1. Os dados da aplicação são inicializados no *Host*;
2. Esses dados são transferidos para o *Device*;
3. Ocorre o processamento paralelo desses dados;
4. O resultado do processamento é transferido para o *Host*.

Em relação a terceira etapa, uma função do *Device*, onde ocorre o processamento paralelo dos dados, é chamada de *Kernel* e ela é executada N vezes por N *threads* diferentes. Um *Kernel* suporta a execução de milhares de *threads* simultaneamente.

O modelo CUDA entrega ao programador, de maneira simples, o poder de distinguir as *threads* que são executadas na GPU através de identidades únicas, o que é útil para decisões de controle ou questões relativas ao endereçamento de memória. Um grupo de *threads* forma um bloco e um grupo de blocos forma uma *grid*.

Em relação as hierarquias de memória, cada *thread* possui sua memória local, enquanto um bloco de *threads* pode ter uma memória compartilhada, a qual possui uma latência

maior do que a memória local da *thread*, porém bem menor do que a memória global da GPU, a qual é acessada por todas as *threads* (Figura 9).

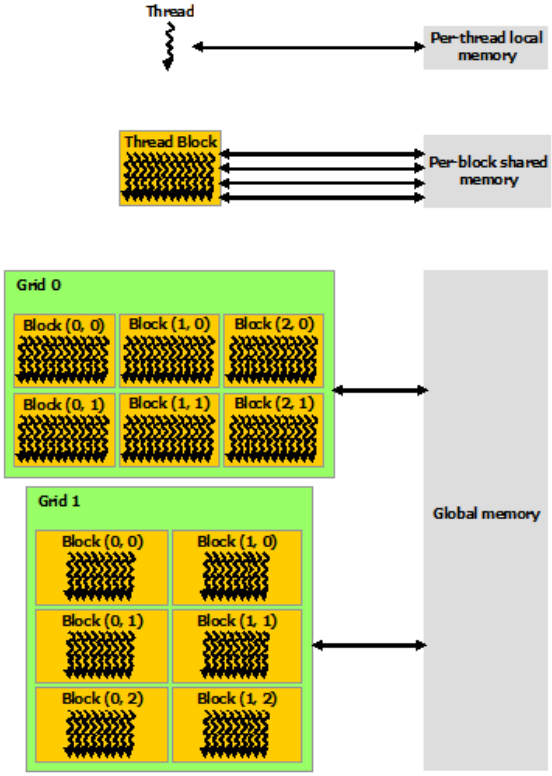

Figura 9: Hierarquia de memória em CUDA.
Imagem retirada de [2]

A partir de sua simplicidade comparada aos métodos de programação em GPU anteriores, a plataforma CUDA atraiu muitos pesquisadores, os quais fazem uso dela para a implementação de seus algoritmos de correspondência estéreo[5][10][13][15][22][26].

2.4 Trabalhos Relacionados

Na última década, com a popularização da programação para GPU provida principalmente pela plataforma CUDA, alguns autores buscaram levar os algoritmos de correspondência estéreo para o tempo real de processamento e, com isso, surgiram algumas maneiras novas e outras variantes de métodos existentes de desenvolvimento para a paralelização dos algoritmos. Então nesta seção serão estudados alguns trabalhos desses autores.

Humenberger et al.[10] desenvolveram um algoritmo de correspondência estéreo que pode ser usado em sistemas embarcados de tempo real. Os dados de entrada que seu algoritmo recebe são, além das imagens estéreo capturadas por duas câmeras em paralelo, parâmetros de calibração delas, o raio de disparidade e limites de confiança e textura. Após passar por várias etapas, incluindo cálculo de custo de correspondência, agregação de custo e computação de disparidade, o algoritmo tem como saída o mapa de disparidade, uma imagem de profundidade, as coordenadas de pontos de nuvem tridimensionais, um mapa de confiança e um mapa de textura.

Após a etapa de captura e configurações de imagens das câmeras, esse algoritmo proposto calcula a provável correspondência de cada *pixel*, tendo como resultado uma imagem de disparidade, a qual é uma estrutura tridimensional com as dimensões *disparidade* × *largura* × *altura*. Para isso os autores utilizam um algoritmo que é uma variação do *Census transform*[29].

Ao assumir que *pixels* vizinhos possuem disparidade semelhante, o algoritmo de correspondência estéreo desenvolvido por esses autores realiza uma agregação de custo que tem correspondências mais distinguíveis. Então realizam uma soma dentro de uma janela de tamanho especificado.

Após realizar todos esses cálculos, o algoritmo busca a melhor correspondência. Isso é feito utilizando-se simplesmente uma abordagem WTA para maior velocidade.

Os autores fazem uso da GPU ao processar cada elemento de dados em diferentes *threads*. Assim, conseguem um considerável paralelismo ao separar dados de imagem em vários blocos que são executados em diferentes multiprocessadores, além de utilizarem memória compartilhada para evitar a maior latência da memória global.

Para a etapa de cálculo de custo de correspondência, a GPU aloca o processamento de cada *pixel* em uma *thread* diferente. Já a etapa de agregação de custo, de acordo com os autores, foi melhor paralelizada utilizando-se uma abordagem de janelas-quadradas proposta por [25]. A etapa de computação de disparidades não utilizou memória compartilhada por causa do grande volume de dados, então os autores buscaram utilizar a memória global da maneira mais eficiente possível.

Como resultado, o algoritmo proposto pelos autores chega a atingir uma velocidade de execução de 1,74 ms para imagens de tamanho 320x240 *pixels*.

Em um trabalho publicado pouco tempo depois, Kowalczuk et al.[13] desenvolvem um novo método para correspondência estéreo em tempo real. Eles utilizam um método de agregação de custo de correspondência que faz uma simplificação do método *Adaptive support-weights* e uma técnica de refinamento iterativo que tem foco na consistência das disparidades.

De acordo com esses autores, o ASW imita o processo de agrupamento do sistema de visão do ser humano a partir de alguns princípios de percepção, entre os quais estão os princípios de proximidade e semelhança, os quais fazem parte do problema de correspondência estéreo. Então, a partir desse primeiro princípio, o algoritmo pode assumir que a probabilidade de um *pixel* p pertencer a mesma profundidade que um *pixel* q diminui de acordo com o aumento da distância entre eles e, a partir do segundo princípio, o algoritmo pode assumir que o *pixel* provavelmente p está na mesma superfície de q caso compartilhem de uma mesma cor.

Para uma maior precisão, o método ASW, assim como o algoritmo desenvolvido por Humenberger et al. e os algoritmos que utilizam métodos locais em geral, precisa considerar a existência uma janela de suporte. Esta janela atribui um peso-suporte a cada *pixel* localizado dentro dela e para cada *pixel* p, o cálculo de custo de correspondência é feito entre p e todos os *pixels* localizados dentro do domínio dessa janela.

Para a paralelização do algoritmo proposto, Kowalczuk et al. separam blocos de *threads* na GPU de forma que cada *thread* seja responsável pelo cálculo de custo de correspondência de um único par de *pixels*. Além disso, já que *pixels* de certas *threads* vizinhas, por estarem na mesma janela,

precisam acessar os mesmos dados, cada bloco de *threads* aloca uma memória compartilhada para evitar uma possível lentidão provinda de muitos acessos a memória global.

Após os cálculos de custo de correspondência, da mesma forma que no algoritmo estudado anteriormente, uma abordagem WTA é utilizada para escolher os *pixels* de acordo com seus menores custos de correspondência. Kowalczuk et al. fazem uso da GPU para o WTA e teste de consistência, resultando nos mapas de disparidade iniciais e níveis de confiança de cada *pixel*. O volume de custo, os mapas de disparidade e os níveis de confiança passam então por um refinamento iterativo, o qual, de acordo com os autores, foi uma adição relativamente pequena a complexidade do algoritmo em comparação a agregação de custo de correspondência. Então finalmente o mapa de disparidade é pós-processado. O fluxo de processamento pode ser visto na Figura 10.

Os autores conseguem atingir um tempo de execução de 16,1 ms para imagens de tamanho 320x240 *pixels*.

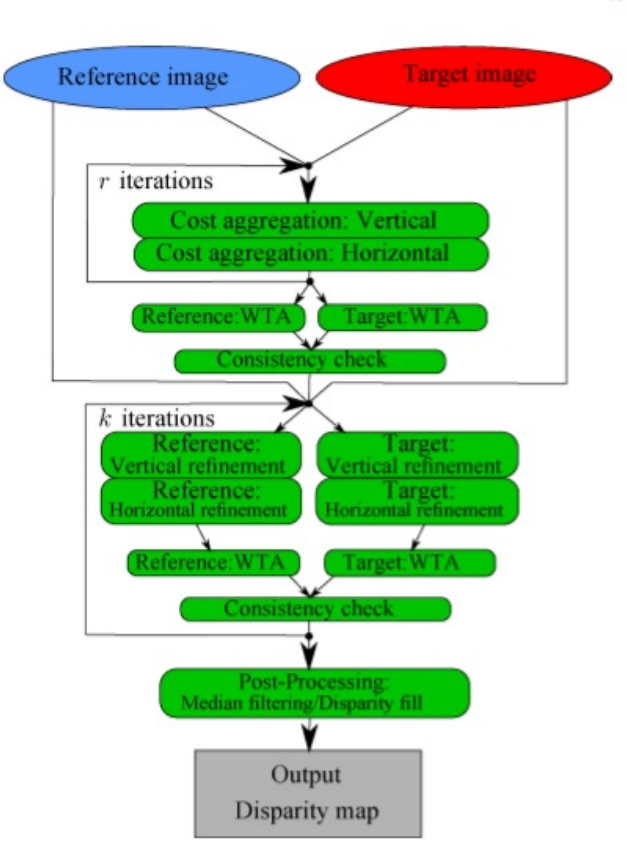

Figura 10: Fluxo de processamento em CUDA do método ASW simplificado.
Imagem retirada de [13]

Os autores Michael et al.[15] e Sah e Jotwani[22] utilizaram abordagens um pouco mais distantes dos métodos locais, chamadas de métodos semi-globais, os quais buscam trazer a baixa complexidade dos métodos locais e a alta qualidade dos métodos globais para a correspondência estéreo.

Para a paralelização do algoritmo de correspondência, Michael et al. começam calculando o custo de correspondência inicial de uma maneira bem direta: é criada uma *thread* para cada entrada de uma matriz C de dimensões *largura x altura x disparidade*. Essa *thread* faz uma busca em duas imagens paralelas, que foram carregadas previamente para a memória da GPU, pela diferença absoluta de níveis de cinza. O algoritmo, a partir da matriz C, então utiliza um método de computação de caminho chamado *path traversal* para percorrer a imagem de disparidade através de oito direções diferentes ao mesmo tempo que minimiza o problema de paralelização criado pelo fato de que cada etapa das direções percorridas dependem de predecessores. O resultado desse método é uma matriz de caminhos E de mesmo tamanho que a matriz C. Por fim, a disparidade d de E para cada *pixel* é determinada e isto pode ser feito para cada *pixel* separadamente. Essa paralelização do algoritmo atinge um tempo de execução de 85,4 ms utilizando tamanhos de imagem de 640x480 *pixels*.

Os autores Sah e Jotwani desenvolveram o algoritmo de correspondência estéreo em paralelo na GPU a partir de uma abordagem de programação dinâmica, fazendo uso de memória compartilhada e global e realizando os passos comuns aos algoritmos semi-globais de cálculo do custo de correspondência, agregação de custo e computação de caminho. Porém não conseguiram alcançar o tempo real de processamento, então, para alcançá-lo, utilizaram um método chamado de multi resolução. Este método, paralelamente, busca três *pixels* e tira uma média deles. Estas médias são então guardadas em uma imagem reduzida e é feita uma correlação nela. Com a imagem reduzida, o tempo de execução diminui consideravelmente, atingindo assim o tempo real de processamento, chegando a um tempo de execução de até 15,1 ms para imagens de tamanho 640x480 *pixels*.

3. SOLUÇÃO DESENVOLVIDA

O algoritmo para correspondência estéreo proposto neste trabalho consiste na paralelização em GPU de um método de cálculo de custo de correspondência chamado *Census Transform* proposto por Zabih e Woodfill[29], mas será apresentado também um método para agregação de custo proposto por Rhemann et al.[21] chamado de *Guided Image Filter* para uma melhor plenitude do sistema de correspondência estéreo utilizado.

Nas subseções a seguir, primeiramente serão apresentados os algoritmos utilizados para todo o processo de correspondência estéreo que são executados em CPU e, logo depois, será apresentada a solução deste trabalho em GPU.

3.1 Algoritmo de Correspondência estéreo

3.1.1 Cálculo de custo de Correspondência

O algoritmo a ser paralelizado neste trabalho é chamado de *Census Transform* (Figura 11) e ele é um algoritmo para cálculo de custo de correspondência.

De acordo com Zabih e Woodfill, os algoritmos de correspondência tem dificuldades em realizar cálculos em regiões de disparidade, as quais geralmente representam as bordas dos objetos. Esses algoritmos são normalmente baseados em métodos estatísticos comuns que são melhores aplicados a um único objeto. Porém, é comum muitos *pixels* em uma região local corresponderem a vários objetos diferentes. Então, a ideia principal da abordagem proposta pelos autores é definir uma transformação local, a qual é aplicada em uma imagem, que contorne o problema de sub-populações distintas de objetos, cada qual possuindo seus próprios parâmetros

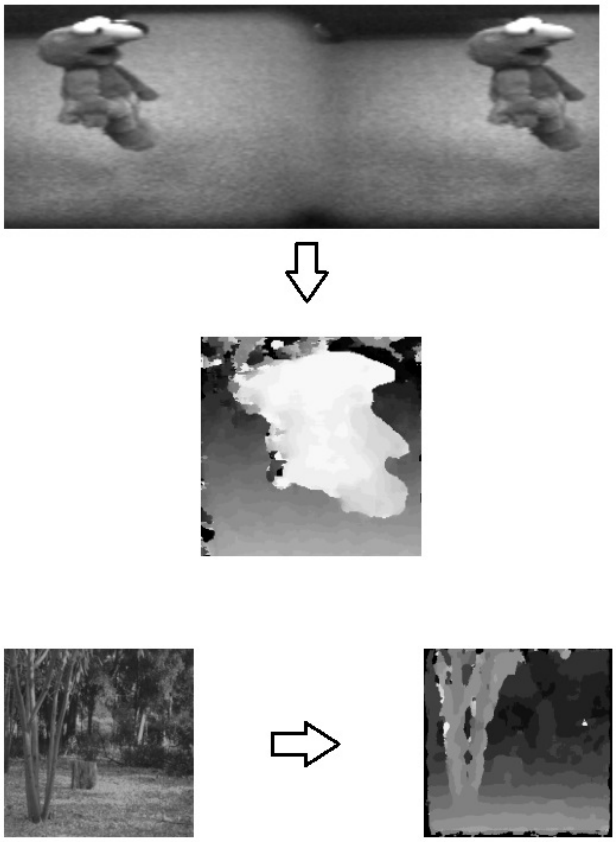

Figura 11: Aplicação do *Census Transform* em uma imagem estéreo e em uma foto.
Imagens retiradas de [29]

de medidas paramétricas, como média ou variância.

20	67	345		0	1	1
23	56	67		0	1	1
45	37	34	->	1	x	0
56	76	57		1	1	1
23	56	78		0	1	1

Figura 12: Exemplo da aplicação do *Census Transform* em uma janela que contém os *pixels* vizinhos a um *pixel x*.

O *Census Transform* é um índice não paramétrico de uma estrutura espacial local. Ele mapeia a intensidade da vizinhança local em volta de um *pixel* p para um bit de tipo String que representa um conjunto de *pixels* vizinhos cuja intensidade é menor que a de p (Figura 12). Tal algoritmo pode ser definido pela seguinte equação:

$$C(p) = \underset{[i,j]\epsilon D}{\otimes} \xi(p, p + [i, j]) \quad (4)$$

Onde $\otimes$ representa uma concatenação dada por uma soma de Minkowski, D representa uma janela em volta do *pixel* p e ξ denota a transformação que pode ser definida como[4]:

$$\xi(p, p + [i, j]) = \left\{ \begin{array}{c} \text{1, onde } p > p + \text{[i,j]} \\ \text{0, caso contrário} \end{array} \right\} \quad (5)$$

3.1.2 Agregação de Custo

A agregação de custo é uma etapa computacionalmente custosa de uma solução de correspondência estéreo. Para a completude do sistema deste trabalho, será utilizado um algoritmo proposto por Rhemann et al. que é chamado de *Guided Image Filter*.

Os autores utilizam uma abordagem chamada de multi-rotulagem para o desenvolvimento de tal algoritmo. Em uma abordagem de rótulo, é feita a construção do volume de custo em três dimensões a partir dos dados de entrada. Em correspondência estéreo, os valores de disparidade de um *pixel* x localizado em *(x,y)* são armazenados nesse volume de custo em um rótulo r.

O nome do algoritmo e a base que Rhemann et al. utilizaram para o desenvolvimento dele vieram de uma solução de filtro proposta por [8] de mesmo nome. Esta solução de filtro foi utilizada pelos autores por ter propriedades que preservam as bordas (Figura 13) das imagens e um tempo de execução independente do tamanho do filtro. Assim poderiam obter uma melhor qualidade em seu algoritmo sem perder velocidade.

Figura 13: Exemplo da preservação de bordas de uma imagem durante a utilização do *Guided Filter*.
Imagem original retirada de [24]; Imagem filtrada retirada de [8]

O algoritmo proposto por Rhemann et al. é dividido em

três etapas:

1. Construção do volume de custo;
2. Filtragem do volume de custo;
3. Seleção de rótulo.

Os autores consideram um problema de rótulo geral, cujo objetivo é associar cada *pixel p* com coordenadas *(x,y)* de uma imagem I a um rótulo r de um conjunto $Cr = 1, ..., R$.

Inicialmente é criado um array tridimensional que representa o volume de custo C. Nesse array são armazenados os custos para um rótulo r de um *pixel* $p = (x,y)$.

Logo depois, as R partições do volume de custo são filtradas. Para isso, os autores realizam uma média ponderada de todos os *pixels* de uma mesma partição, a qual é resultado da filtragem em um *pixel* de índex i em um rótulo r. Isso pode ser representado pela equação:

$$C'_{i,r} = \sum_{j} W_{i,j}(I) C_{j,r} \tag{6}$$

Onde C' representa o volume de custo filtrado, enquanto i e j representam os índices dos *pixels*. O peso do filtro W depende da imagem I de referência.

Após a filtragem de volume de custo, é feita uma seleção de rótulo simples com uma abordagem *winner-take-all*, como mostrada a seguir:

$$f_i = \arg\min_{r} C'_{i,r} \tag{7}$$

No contexto da correspondência estéreo, Rhemann et al. estabeleceram que os rótulos r correspondem a vetores *(u,v)*, os quais representam deslocamentos nos eixos *(x,y)*. A disparidade d corresponde a u e não há deslocamento no eixo y, o que assume que as imagens estéreo estão totalmente paralelas no eixo x. O volume de custo mostra o quanto um *pixel p* em uma imagem I corresponde a um *pixel p'* em uma imagem I' deslocada pelo vetor r.

3.2 Census Transform em GPU

O código-fonte do algoritmo *Census Transform* utilizado para sua paralelização, assim como o de todo o sistema de correspondência estéreo, foi retirado de uma abordagem proposta por Zhang et al. [30].O algoritmo paralelizado foi testado em uma Unidade de Processamento Gráfico NVidia GT720 utilizando o Visual Studio 2010 da ©Microsoft.

O *Census Transform* implementado pelos autores utiliza alguns formatos de objetos que não tem suporte em CUDA, então para a implementação do algoritmo em GPU proposto por este trabalho, foi necessária a conversão de algumas variáveis para formatos que possuem suporte nessa plataforma. Entre eles, estão os formatos *Mat*, o qual é próprio do *OpenCV* [18], e o *Bitset*. O formato *Mat* foi então transformado em formato *GpuMat*, o qual foi transformado em *PtrStepSz* para seu reconhecimento dentro do *kernel*. Estes formatos são utilizados para representar a imagem digitalmente. Ambos *GpuMat* e *PtrStepSz* também são formatos do OpenCV, o qual foi recompilado para ter suporte a CUDA neste trabalho. Já o formato *Bitset* foi utilizado por Zhang et al. por ser uma classe que é basicamente um *array* de *bools*, o qual otimiza espaço em memória. Esse formato não possui suporte em CUDA atualmente, então, ao invés dele, foram utilizados *arrays* comuns de *bools*.

Zhang et al. utilizam o *Bitset* com duas dimensões para implementar o *Census Transform*, sendo uma delas a altura multiplicada pela largura em *pixels* da imagem, o que dificultou a cópia dos *arrays* para a GPU. Então estes *arrays* de duas dimensões foram mapeados em *arrays* de uma dimensão apenas, os quais foram convertidos novamente para *arrays* de duas dimensões quando o *kernel* termina seu processamento.

O processamento dos *pixels* no algoritmo desenvolvido pelos autores é feito a partir de quatro iterações aninhadas, o que gera um tempo de processamento exponencial, dependendo do número de *pixels* a serem processados. Então a paralelização do *Census Transform* neste trabalho foi feita de forma que cada *pixel* em posição *(x,y)* é processado em uma *thread* diferente, cada qual com duas iterações aninhadas necessárias para a comparação do *pixel* de interesse com a sua vizinhança dentro de uma janela.

4. RESULTADOS E DISCUSSÃO

Para a paralelização do algoritmo *Census Transform*, foram utilizados o ©Microsoft Visual Studio 2010 e a biblioteca de visão computacional OpenCV.

O algoritmo desenvolvido em GPU foi testado junto a sua contraparte em CPU utilizando-se tamanhos de imagens de 128x128px, 256x256, 384x384, 512x512px, 640x640, 768x768 e 800x800px. Em todos os testes, o algoritmo desenvolvido em GPU teve um tempo de processamento mais de 10 vezes menor do que o desenvolvido por Zhang et al. em CPU (Figuras 14 e 15). Porém, como limitação do trabalho, o algoritmo não consegue processar imagens maiores do que 800x800, suspeitando-se de que o motivo foi a falta de possibilidade do uso do formato *Bitset* em CUDA, o qual otimiza espaço de memória para *arrays* de *bools*, enquanto os *arrays* comuns de duas dimensões utilizados precisariam armazenar cerca de 83.886.080 posições, considerando-se apenas uma das variáveis. Então por o tamanho desses *arrays* serem muito grandes, não foi utilizado o *shared memory*.

A verificação de integridade dos dados podem ser vistos na figura 16 onde o algoritmo é testado em uma imagem 768x768 tanto em CPU quanto em GPU e é mostrado em tela as posições iniciais dos *arrays* resultados. O mapa de disparidade resultante da execução do sistema completo de correspondência estéreo em imagens do dataset do Middlebury pode ser visto nas Figuras 17, 18 e 19.

5. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho, um algoritmo de *Census Transform*, o qual estava inserido em um sistema de correspondência estéreo, foi paralelizado em GPU, utilizando-se a plataforma CUDA, e mostrou-se muito mais eficiente do que sua contraparte em CPU por ter tempo de execução mais de dez vezes menor em todos os tamanhos de imagem testados. Porém o algoritmo possui limitações relacionadas ao tamanho máximo de imagem a qual pode processar.

Em trabalhos futuros, poderão ser realizadas otimizações no algoritmo para que ele atinja um possível tempo real de processamento, além de otimizações do uso da memória com o intuito encontrar soluções para a limitação deste trabalho e paralelizar outros algoritmos para que seja formado um sistema completo de correspondência estéreo que seja executado em GPU.

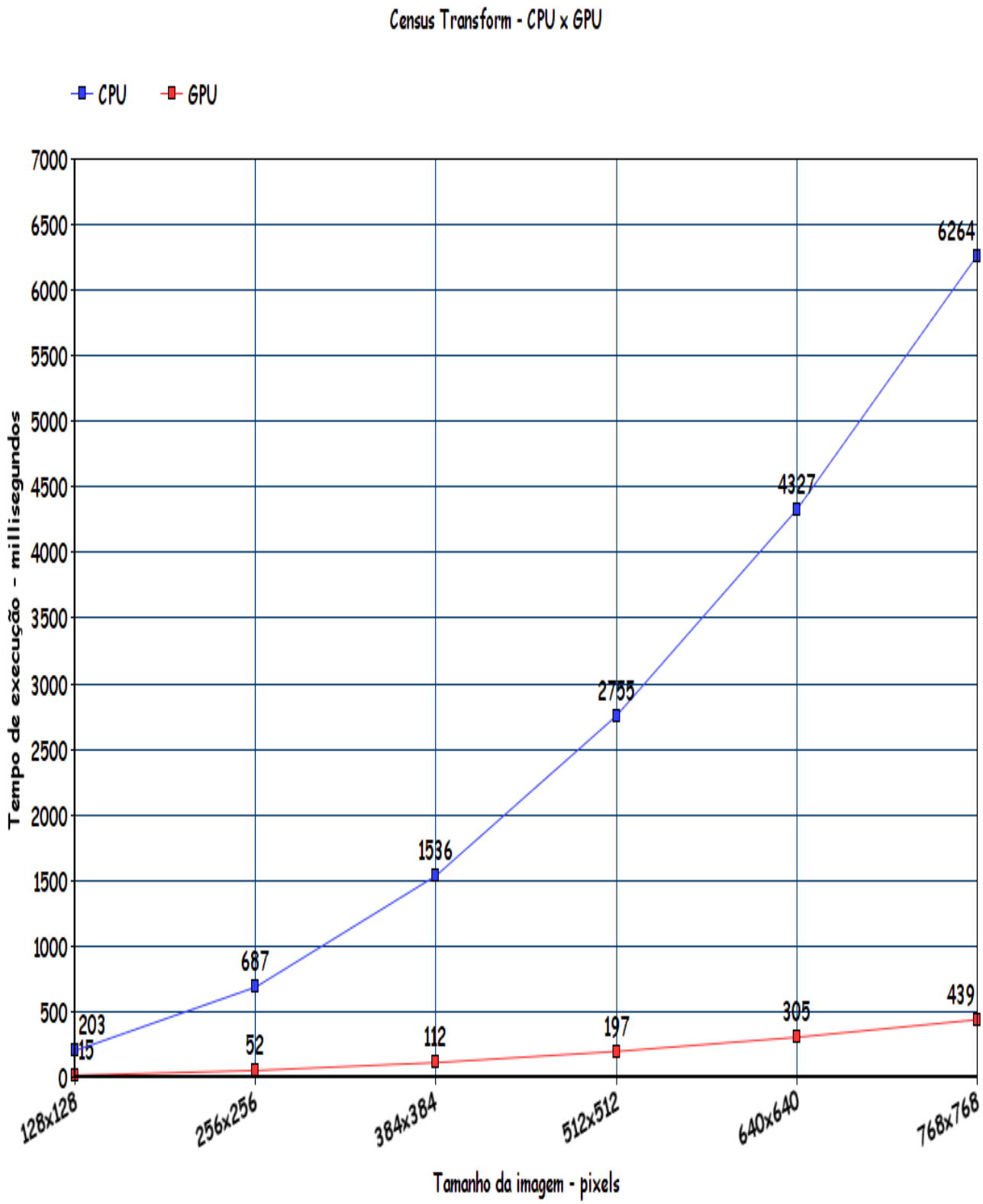

Figura 14: Comparação do tempo de execução do algoritmo *Census Transform* em CPU e GPU.

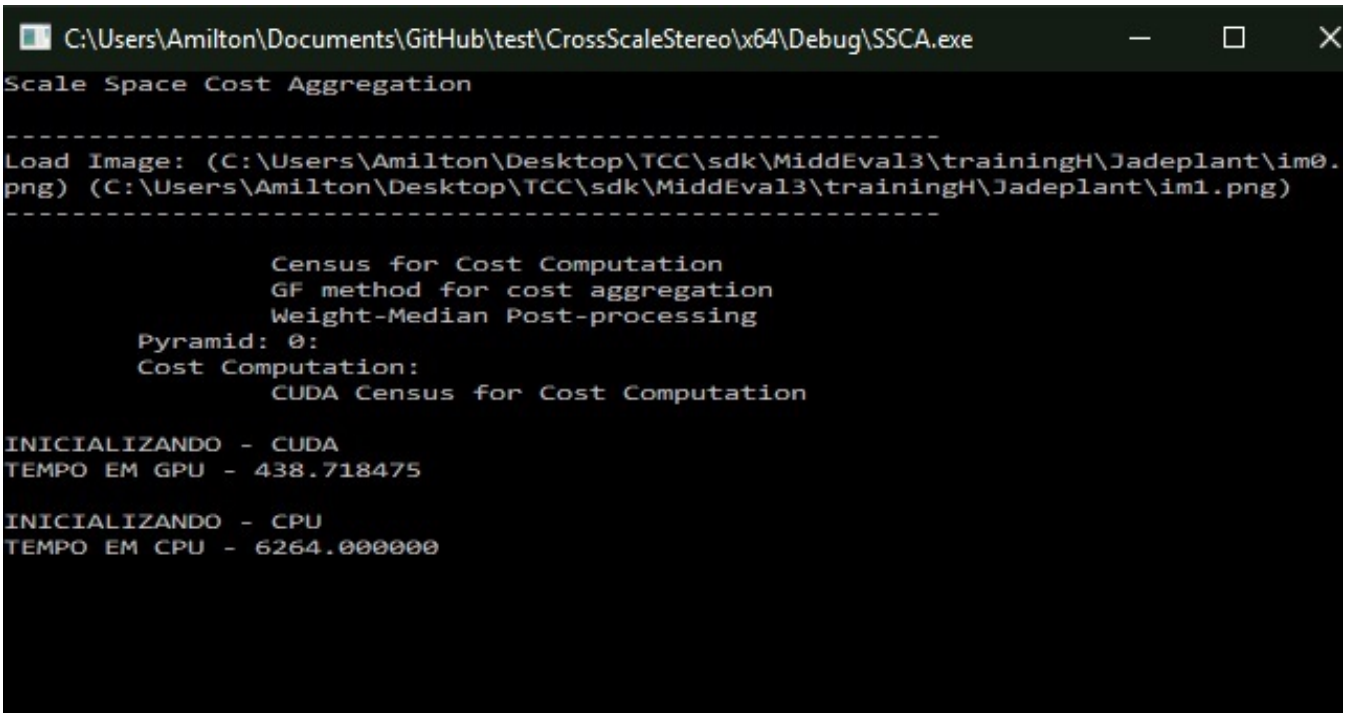

Figura 15: Comparação do tempo de execução em milissegundos do algoritmo. *Census Transform* em CPU e GPU para uma imagem 768x768.

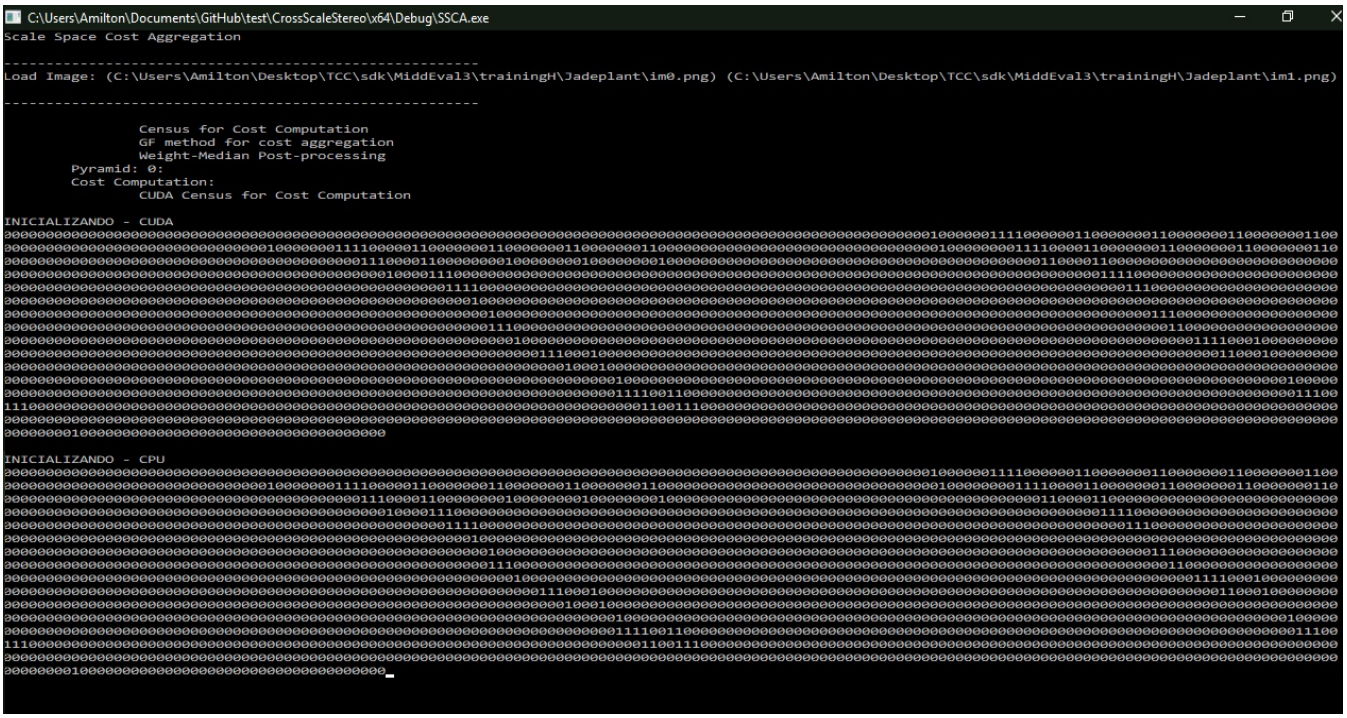

Figura 16: Verificação de integridade dos dados.

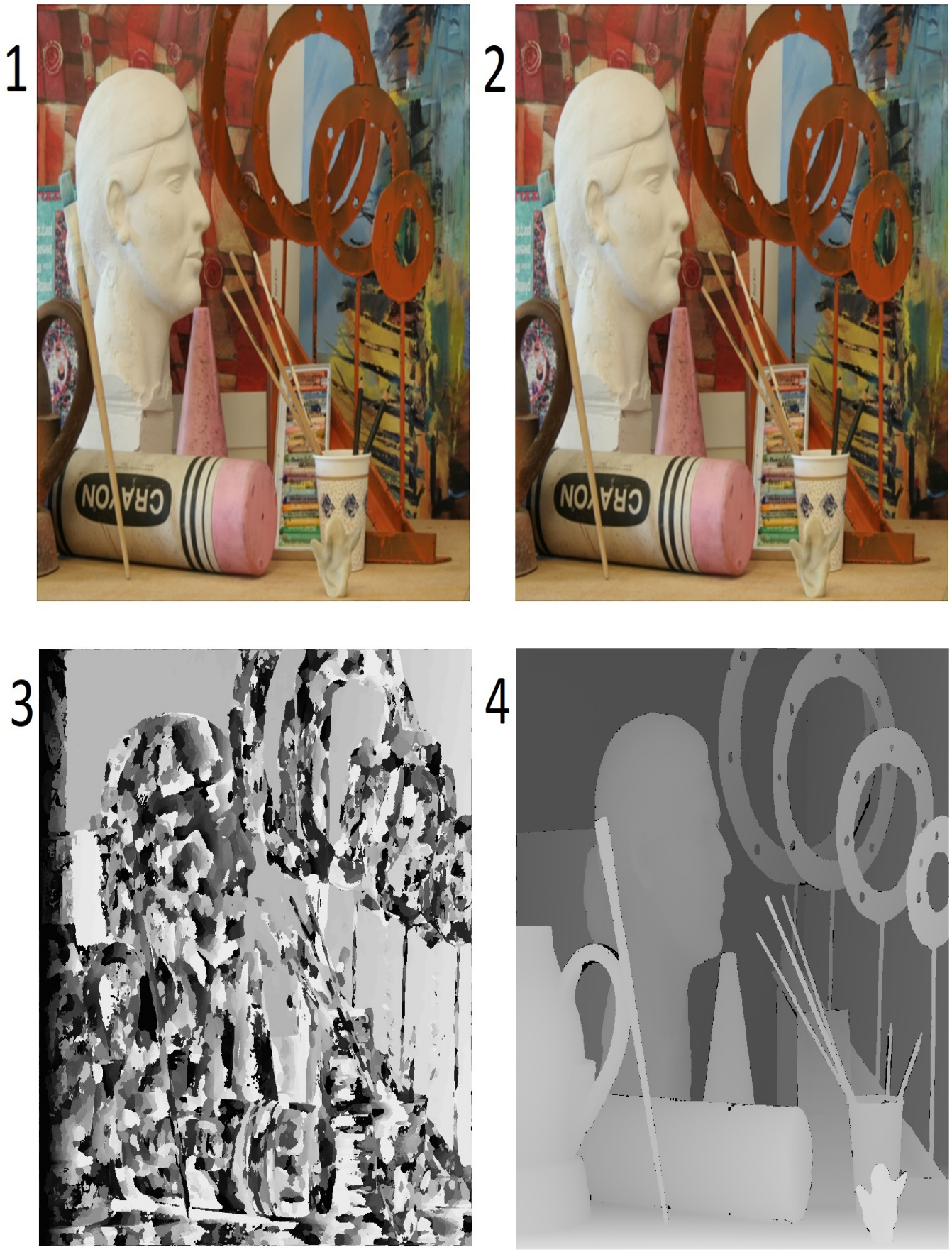

Figura 17: Mapa de disparidade (3) resultante das imagens fonte (1) e (2). Ground Truth representado por (4).

Imagens fonte retiradas de [3]

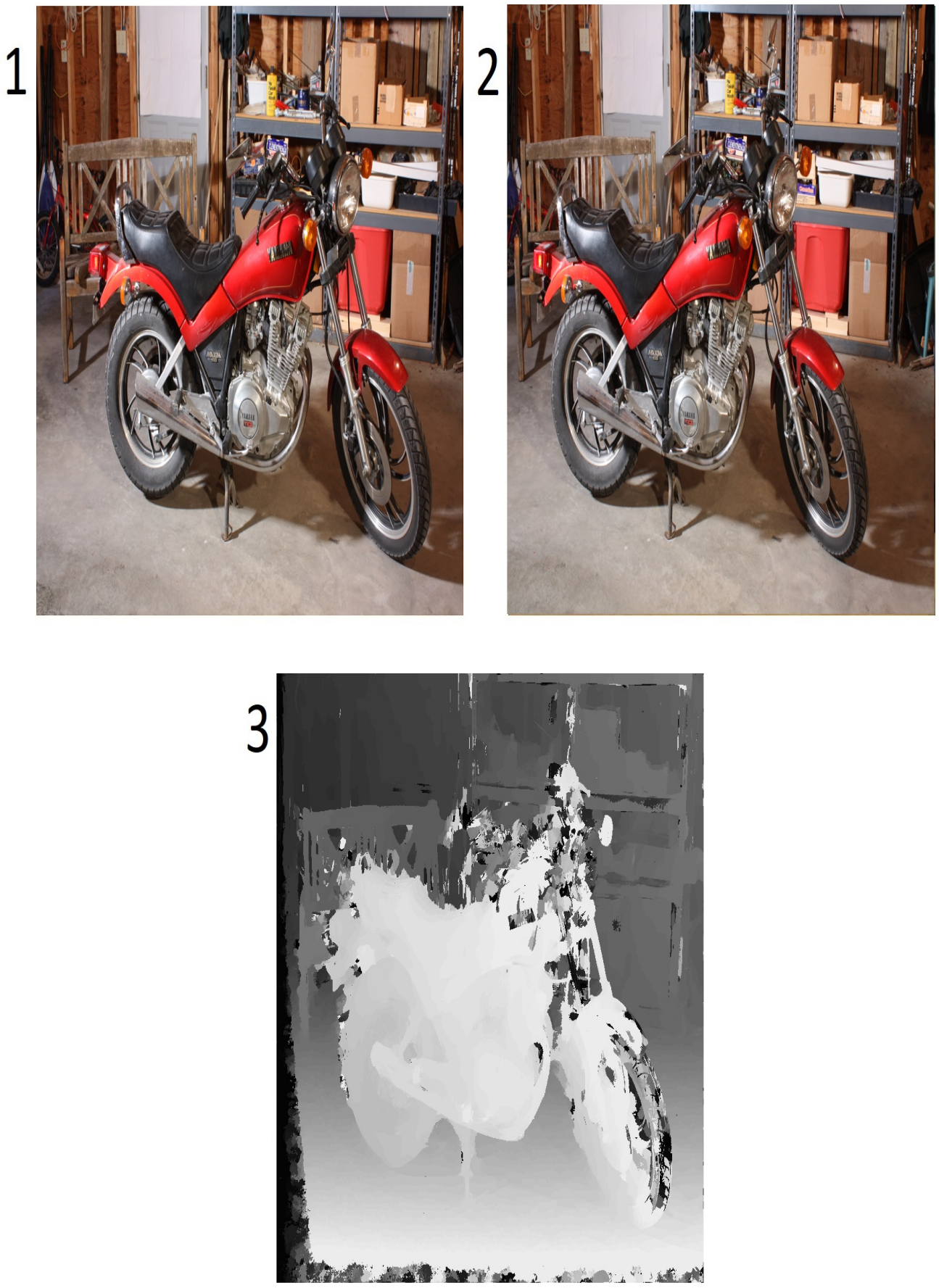

Figura 18: Mapa de disparidade (3) resultante das imagens fonte (1) e (2).
Imagens fonte retiradas de [3]

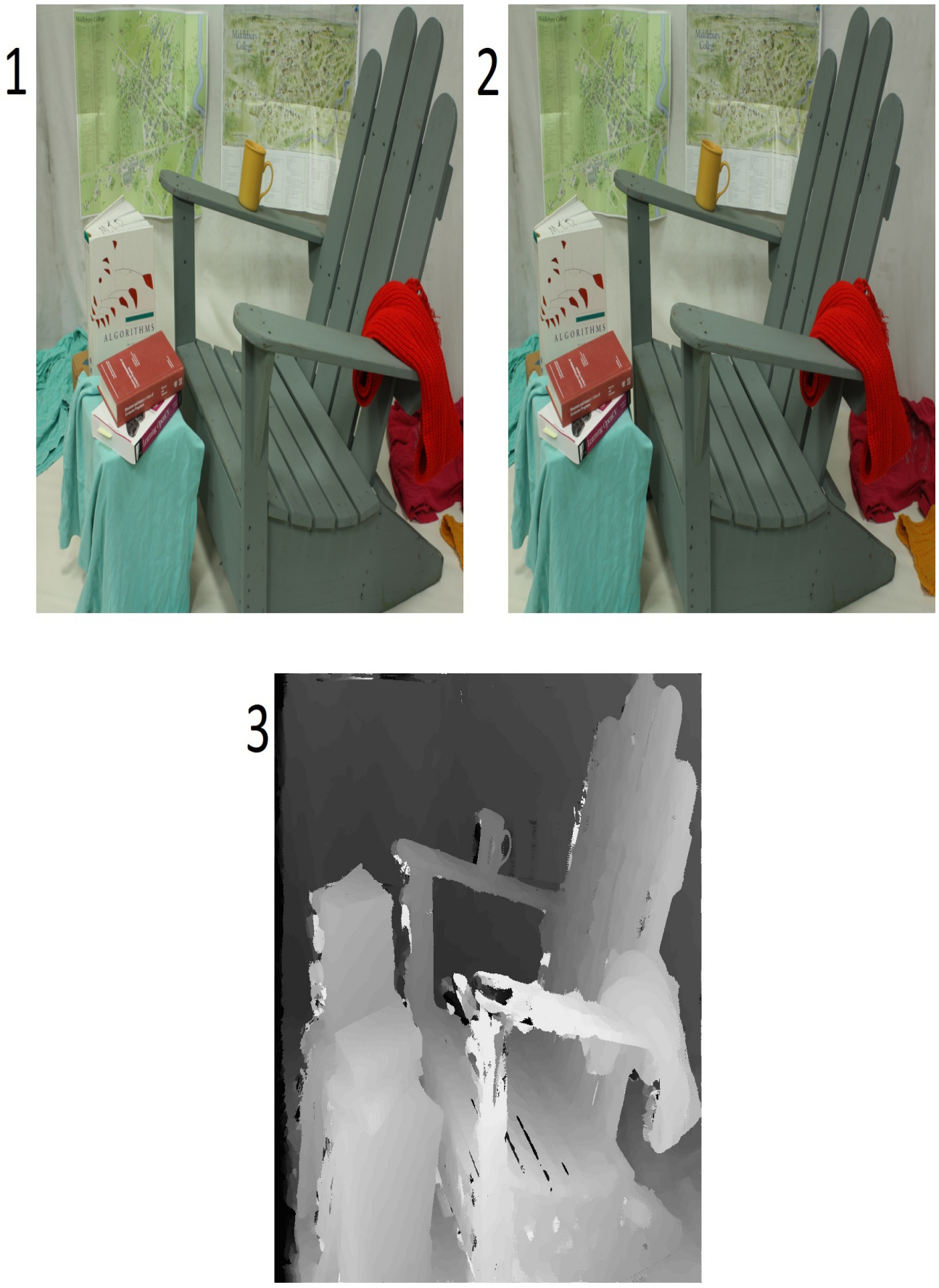

Figura 19: Mapa de disparidade (3) resultante das imagens fonte (1) e (2).
Imagens fonte retiradas de [3]

6. REFERENCES

[1] Mobilefusion: Research project turns regular mobile phone into 3d scanner. https://www.microsoft.com/en-us/research/blog/mobilefusion-research-project-turns-regular-mobile-phone-into-3d-scanner/, 2015 (acessado Julho 28, 2017).

[2] Cuda c programming guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.htmlaxzz4rM91QfP1, 2017 (acessado Agosto 27, 2017).

[3] Middlebury stereo vision page. http://vision.middlebury.edu/stereo/, 2017 (acessado Julho 28, 2017).

[4] Y. K. Baik, J. H. Jo, and K. M. Lee. Fast census transform-based stereo algorithm using sse2, 2006.

[5] T. Chen, Y. Liu, J. Li, and P. Wu. *Fast Narrow-Baseline Stereo Matching Using CUDA Compatible GPUs*, pages 10–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[6] G. D. Evangelidis. Subpixel stereo correspondence. https://sites.google.com/site/georgeevangelidis/encc, 2017 (acessado Novembro 14, 2017).

[7] R. A. Hamzah and H. Ibrahim. Literature survey on stereo vision disparity map algorithms. *Journal of Sensors*, 2016:1–23, 2016.

[8] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, June 2013.

[9] H. Hirschmüller, P. R. Innocent, and J. Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *International Journal of Computer Vision*, 47(1):229–246, Apr 2002.

[10] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze. A fast stereo matching algorithm suitable for embedded real-time systems. *Computer Vision and Image Understanding*, 114(11):1180–1202, 2010.

[11] B. Jähne. *Digital Image Processing*. Springer-Verlag, 2002.

[12] D. B. Kirk and W.-m. W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[13] J. Kowalczuk, E. T. Psota, and L. C. Perez. Real-time stereo matching on cuda using an iterative refinement method for adaptive support-weight correspondences. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(1):94–104, 2013.

[14] D. Kumari and K. Kaur. A survey on stereo matching techniques for 3d vision in image processing. *International Journal of Engineering and Manufacturing*, 6(4):40–49, 2016.

[15] M. Michael, J. Salmen, J. Stallkamp, and M. Schlipsing. Real-time stereo vision: Optimizing semi-global matching. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 1197–1202, June 2013.

[16] NVIDIA. Cuda. http://www.nvidia.com/object/cuda_home_new.html, 2017 (acessado Julho 28, 2017).

[17] P. Ondrúška, P. Kohli, and S. Izadi. Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1251–1258, Nov 2015.

[18] opencv. Open source computer vision library. https://github.com/opencv/opencv, 2017.

[19] C. S. Panchal and A. B. Upadhyay. Depth estimation analysis using sum of absolute difference algorithm. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 3(1), 2014.

[20] E. Petrović, A. Leu, D. Ristić-Durrant, and V. Nikolić. Stereo vision-based human tracking for robotic follower. *International Journal of Advanced Robotic Systems*, 10(5):230, 2013.

[21] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 3017–3024, Washington, DC, USA, 2011. IEEE Computer Society.

[22] S. Sah and N. Jotwani. Stereo matching using multi-resolution images on cuda. *International Journal of Computer Applications*, 56(12):47–55, 2012.

[23] D. Scharstein, R. Szeliski, and R. Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, pages 131–140, 2001.

[24] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, Jan 1998.

[25] O. Veksler. Fast variable window for stereo correspondence using integral images. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–556–I–561 vol.1, June 2003.

[26] J. Žbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. *J. Mach. Learn. Res.*, 17(1):2287–2318, Jan. 2016.

[27] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–211–I–217 vol.1, June 2003.

[28] K.-J. Yoon and I.-S. Kweon. Locally adaptive support-weight approach for visual correspondence search. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 924–931 vol. 2, June 2005.

[29] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *Proceedings of the Third European Conference on Computer Vision (Vol. II)*, ECCV '94, pages 151–158, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[30] K. Zhang, Y. Fang, D. Min, L. Sun, S. Yang, S. Yan, and Q. Tian. Cross-scale cost aggregation for stereo matching. In *CVPR*, 2014.

[31] K. Zhang, J. Li, Y. Li, W. Hu, L. Sun, and S. Yang.

Binary stereo matching. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 356–359, Nov 2012.

[32] F. Šuligoj, B. Šekoranja, M. Švaco, and B. Jerbić. Object tracking with a multiagent robot system and a stereo vision camera. *Procedia Engineering*, 69:968 – 973, 2014. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013.