Pontifícia Universidade Católica
do Rio de Janeiro

Leandro Tavares Aragão dos Santos

Geração de Mapas de Profundidade Super-resolvidos a partir de Sensores de Baixo Custo e Imagens RGB

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC–Rio.

Orientador: Prof. Alberto Barbosa Raposo

Rio de Janeiro
Fevereiro de 2014

Pontifícia Universidade Católica
do Rio de Janeiro

Leandro Tavares Aragão dos Santos

Geração de Mapas de Profundidade Super-resolvidos a partir de Sensores de Baixo Custo e Imagens RGB

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC–Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Alberto Barbosa Raposo
Orientador
Departamento de Informática — PUC–Rio

Prof. Luciano Pereira Soares
Insper Instituto de Ensino e Pesquisa

Prof. Marcelo Gattass
Departamento de Informática — PUC-Rio

Prof. Manuel Eduardo Loaiza Fernandez
Instituto Tecgraf — PUC-Rio

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

Rio de Janeiro, 27 de Fevereiro de 2014

Leandro Tavares Aragão dos Santos

Bacharel em Engenharia Eletrônica e de Computação pela Universidade Federal do Rio de Janeiro (2009).

Ficha Catalográfica

Santos, Leandro Tavares Aragão dos

Geração de Mapas de Profundidade Super-resolvidos a partir de Sensores de Baixo Custo e Imagens RGB / Leandro Tavares Aragão dos Santos; orientador: Alberto Barbosa Raposo. — Rio de Janeiro : PUC–Rio, Departamento de Informática, 2014.

v., 75 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Dissertação. 2. Kinect. 3. Mapas de Profundidade. 4. Super-Resolução. 5. OpenNI. 6. OpenCV. I. Alberto Barbosa Raposo. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Agradeço àquele que, para ajudar à família, começou a trabalhar com 8 anos de idade e, ainda jovem, voltava da Escola de Especialistas da Aeronáutica de carona em caçambas de caminhão. Que, apesar de todas as dificuldades, se tornou Oficial como Engenheiro e, ao mesmo tempo em que fazia o impossível para que eu e minha irmã tivéssemos as melhores condições de vida e estudo, se formou em Direito e foi aprovado como Delegado Federal. Que concluiu seu exemplo de vida como Doutor em Ciências Jurídicas e Sociais. Devo a ele, meu pai e grande amigo, Carlos Sérgio Aragão dos Santos (in memoriam), esse Mestrado e o homem que sou.

À minha meiga e carinhosa mãe, Maria Suely de Oliveira Tavares, que abdicou de sua vida em prol dos filhos. Que cansada, após um longo dia de trabalho, não titubeava em me ajudar nos estudos o quanto fosse necessário. Devo a ela, minha persistência e dedicação ao conhecimento.

À minha bondosa avó, Maria Ofélia de Oliveira Tavares, que cuidava de mim, enquanto minha mãe trabalhava, e não me deixava escapar pelas ruas, quando, ainda uma criança, insistia em fugir para não ir ao colégio.

À minha querida irmã, Luciana Tavares Aragão dos Santos, que, através de seus conselhos e amizade, mantém presentes os valores que nosso saudoso e querido pai nos ensinou.

À minha amada esposa, Monique da Silva Garcia, que sacrificou diversos momentos de lazer e descontração dos primeiros anos de nosso casamento para que eu pudesse me dedicar às atividades do Mestrado. Mais que isso, me mantém firme diante da vida desde o desenlace de meu pai.

Ao meu orientador Alberto Barbosa Raposo, pela paciência, dedicação e comprenssão diante de todos os intempéries que se abateram sobre minha vida durante o Mestrado.

Resumo

Santos, Leandro Tavares Aragão dos; Alberto Barbosa Raposo. **Geração de Mapas de Profundidade Super-resolvidos a partir de Sensores de Baixo Custo e Imagens RGB**. Rio de Janeiro, 2014. 75p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

As aplicações da reconstrução em três dimensões de uma cena real são as mais diversas. O surgimento de sensores de profundidade de baixo custo, tal qual o Kinect, sugere o desenvolvimento de sistemas de reconstrução mais baratos que aqueles já existentes. Contudo, os dados disponibilizados por este dispositivo ainda carecem em muito quando comparados àqueles providos por sistemas mais sofisticados. No mundo acadêmico e comercial, algumas iniciativas, como aquelas de Tong et al. [1] e de Cui et al. [2], se propõem a solucionar tal problema. A partir do estudo das mesmas, este trabalho propôs a modificação do algoritmo de super-resolução descrito por Mitzel et al. [3] no intuito de considerar em seus cálculos as imagens coloridas também fornecidas pelo dispositivo, conforme abordagem de Cui et al. [2]. Tal alteração melhorou os mapas de profundidade super-resolvidos fornecidos, mitigando interferências geradas por movimentações repentinas na cena captada. Os testes realizados comprovam a melhoria dos mapas gerados, bem como analisam o impacto da implementação em CPU e GPU dos algoritmos nesta etapa da super-resolução. O trabalho se restringe a esta etapa. As etapas seguintes da reconstrução 3D não foram implementadas.

Palavras–chave

Kinect; Mapas de Profundidade; Super-Resolução; OpenNI; OpenCV;

Abstract

Santos, Leandro Tavares Aragão dos; Raposo, Alberto Barbosa (Advisor). **Generating Superresolved Depth Maps using Low Cost Sensors and RGB Images**. Rio de Janeiro, 2014. 75p. M.Sc Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

There are a lot of three dimensions reconstruction applications of real scenes. The rise of low cost sensors, like the Kinect, suggests the development of systems cheaper than the existing ones. Nevertheless, data provided by this device are worse than that provided by more sophisticated sensors. In the academic and commercial world, some initiatives, described in Tong et al. [1] and in Cui et al. [2], try to solve that problem. Studying that attempts, this work suggests the modification of super-resolution algorithm described for Mitzel et al. [3] in order to consider in its calculations coloured images provided by Kinect, like the approach of Cui et al. [2]. This change improved the super resolved depth maps provided, mitigating interference caused by sudden changes of captured scenes. The tests proved the improvement of generated maps and analysed the impact of CPU and GPU algorithms implementation in the superresolution step. This work is restricted to this step. The next stages of 3D reconstruction have not been implemented.

Keywords

Kinect; Depth Maps; Super Resolution; OpenNI; OpenCV;

Sumário

Lista de Figuras

Lista de Tabelas

A sua vida será sempre o que você esteja mentalizando constantemente. Em razão disso qualquer mudança real em seus caminhos, virá unicamente da mudança de seus pensamentos.

Espírito André Luiz, *Respostas da Vida.*

1
Introdução

1.1
Contexto

A reconstrução em três dimensões de uma cena real se aplica às mais diversas áreas: levantamentos de engenharia, como aqueles viabilizados por tecnologias de empresas como a Leica Geosystems [4] e a LFM [5], estudos de ergonomia, navegação de robôs [6], captação do manequim de clientes para lojas de roupas [7] e criação de modelos de monumentos históricos, conforme proposto por Chen et al. [8]. Focando nas aplicações relacionadas ao corpo humano, existem diversas tecnologias que oferecem reconstruções de altíssima qualidade, contudo, seus custos são muito altos. De acordo com Tong et al. [1], o CyberWare Whole Body Color 3D Scanner custa em torno de US$ 240.000,00 e o Swiss Ranger 4000 custa US$ 8.000,00. Considerando esse cenário, uma abordagem que permitisse que os dados do Kinect fossem depurados a um nível de qualidade próximo àquele dos dispositivos mencionados anteriormente, viabilizaria, numa escala mais abrangente, esse tipo de utilização.

Se além de modelos 3D do corpo humano de alta qualidade, tal sistema fosse capaz de ser manipulado pelos movimentos das juntas detectados pelo Kinect, um leque de possibilidades se abriria. Na indústria de óleo e gás, gerar-se-ia o avatar de um operador e o mesmo poderia ser treinado na maquete 3D de determinado empreendimento. Mais que isso, avaliações ergonômicas poderiam ser executadas utilizando-se o modelo virtual do operador. Na medicina, o mesmo avatar poderia ser integrado a modelos volumétricos do organismo do paciente.

Agrupando e processando os mapas de profundidade, as imagens RGB e os movimentos das juntas detectados pelo Kinect, tal sistema objetivaria a geração de avatares controlados em tempo real. Primeiramente, um processo de reconstrução aplicado aos mapas de profunidade e às imagens RGB daria origem a malhas 3D correspondentes ao corpo de cada usuário captado. Posteriormente, por meio dos métodos de Rigging and Skinning, os modelos 3D de cada usuário seriam integrados às juntas captadas. Dessa forma, a movimentação de cada usuário distorceria as respectivas malhas em tempo real. O esquemático desse sistema está representado na figura 1.1.

Esmiuçando o processo de reconstrução, temos três etapas fundamentais:

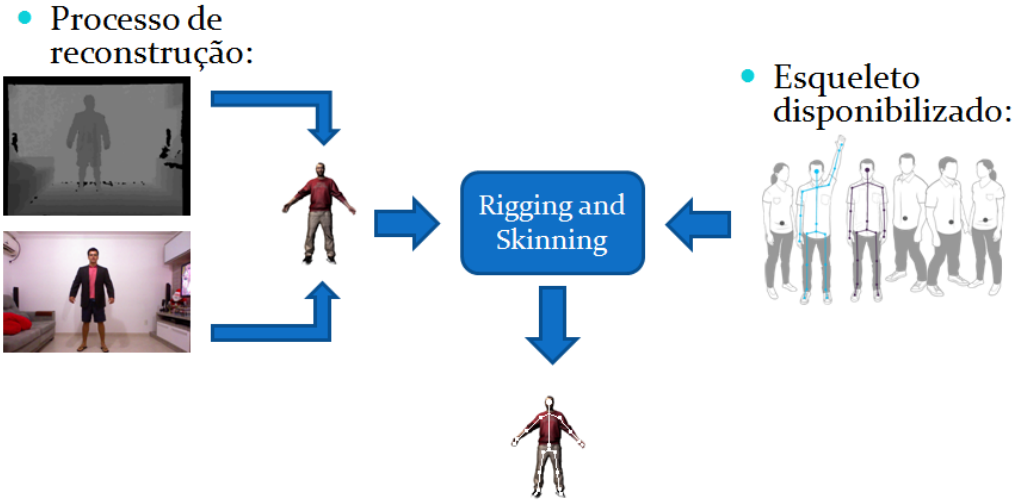


Figura 1.1: Esquemático básico.

a super-resolução, o registro rígido global e o registro não-rígido. A super-resolução gera mapas de profundidade de alta resolução a partir de N mapas de profundidade de baixa resolução. O registro rígido global dá origem a uma malha 3D reconstruída a partir dos mapas de alta resolução. Contudo, tal reconstrução contém diversos artefatos decorrentes da movimentação do usuário durante a captação. Para eliminar tais problemas, a malha preliminar é submetida ao registro não-rígido.

Não existe uma biblioteca aberta que abranja todas as etapas necessárias aos objetivos mencionados. Nesse contexto, essa dissertação se foca na primeira e fundamental das etapas de todo o processo: a super-resolução. Quando se analisam os dados disponibilizados por sensores de profundidade de baixo custo, constatamos que a resolução oferecida é insuficiente para uma série de aplicações relacionadas à reconstrução 3D de alta qualidade. A partir dessa simples observação, é possível entender a razão pela qual as estratégias de reconstrução, a partir desses sensores, dependem fundamentalmente de algoritmos de super-resolução.

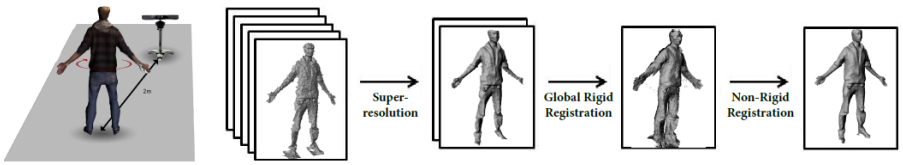
Tais algoritmos compartilham a abordagem representada pela fórmula:

$$minimiza \quad E_{data}(X) + E_{regular}(X),$$

onde $X \in \Re^{\beta n \times \beta m}$ representa a imagem super-resolvida objetivada pelo algoritmo, E_{data} mede a correspondência entre o mapa super-resolvido e os mapas de baixa resolução utilizados no processo de reconstrução e $E_{regular}$ se propõe a preservar as componentes de alta frequência presentes nas cenas originais e suprimir as componentes correspondentes a ruídos. Tal minimização

nos conduz à reconstrução que mais se aproxima da cena real de acordo com os parâmetros adotados em ambos os termos.

Durante as pesquisas, o trabalho de Cui et al. [2], se destacou devido ao aproveitamento das informações das imagens RGB no termo de correspondência E_{data}. Conforme mencionado anteriormente e mostrado por Cui et al. [2], o proceso de reconstrução se divide em três etapas fundamentais: a super-resolução, o registro global rígido e o não-rígido. Um esboço desse processo e da configuração da captação necessária ao mesmo pode ser vizualizado na figura 1.2.



(a) Working setup (b) Processing pipeline

Figura 1.2: Figura retirada de Cui et al. [2] - Sequência de digitalização.

A codificação da abordagem de Cui et al. [2] não está disponível. Daí, decidiu-se buscar nas bibliotecas amplamente utilizadas pela comunidade, porções desse algoritmo que pudessem ser agrupadas e, a partir da implementtação dos módulos restantes, obter um algoritmo que replicasse o trabalho de Cui et al. [2] no tocante à etapa de super-resolução. Decidiu-se utilizar a biblioteca OpenCV, devido à sua integração com a OpenNI e à classe de super-resolução que a mesma disponibiliza para sequências de vídeo. A ideia básica seria modificar tal classe, baseada na proposta de Mitzel et al. [3], incluindo a consideração de cores descrita por Cui et al. [2].

A super-resolução trata os mapas de profundidade crus oferecidos pelos dispositivos de captação. Nesta etapa, o objetivo é, a partir dos dados disponibilizados pelos sensores, gerar imagens de alta resolução. O desafio aqui é encontrar uma sequência de transformações lineares que descrevam corretamente o processo de formação da imagem de alta resolução. Para isso, se analisam as transformações que ocorrem durante a captação de uma cena real e como estas afetam a imagem gerada, conforme mostrado na figura 1.3. Para a obtenção das equações da super-resolução, tal sequência é invertida.

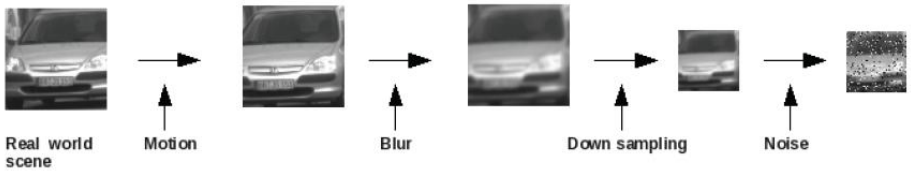


Figura 1.3: Figura retirada de Mitzel et al. [3] - Processo de formação de uma imagem captada de uma cena real.

Conforme já mencionado, partiu-se da abordagem apresentada por Mitzel et al. [3]. A partir de N imagens de baixa resolução $\{I_L^k\}_{k=1}^N$ de tamanho $L_1 \times L_2$, devemos encontrar a imagem de alta resolução I_H de tamanho $H_1 \times H_2$, sendo $H_1 > L_1$ e $H_2 > L_2$, que minimiza o custo da seguinte função:

$$E(I_H) = \sum_{k=1}^{N} \parallel P_k(I_H) - I_L^{(k)} \parallel \tag{1-1}$$

onde $P_k(I_H)$ é a projeção de I_H no sistema de coordenadas e na grade de amostragem da imagem I_L^k. $\parallel \, . \, \parallel$ comporta qualquer norma, contudo, geralmente as normas l_1 ou l_2 são utilizadas. A definição de tais normas é a seguinte:

$$l_1 = \parallel X_1 - X_2 \parallel_1 = \sum \mid X_{1i} - X_{2i} \mid \tag{1-2}$$

$$l_2 = \parallel X_1 - X_2 \parallel_2 = \sum \sqrt{(X_{1i} - X_{2i})^2} \tag{1-3}$$

onde X_{1i} corresponde ao pixel i da matriz X_1 e X_{2i} corresponde ao pixel i da matriz X_2.

A projeção mencionada é modelada por quatro transformações lineares - movimento, blur, subamostragem e adição de ruído. Cada imagem de baixa resolução se relaciona com a de alta resolução através da seguinte fórmula:

$$I_L^k = D_k B_k W_k I_H + e_k \tag{1-4}$$

Sendo D_k a matriz de subamostragem, B_k a matriz de blurring, W_k a matriz de deformação e e_k o vetor de ruído. Apesar de a análise do problema se basear em notações matriciais, a implementação utiliza operações padrões como convolução, deformação e amostragem conforme proposto por Elad e Feuer [9].

Mitzel et al. [3] não estimam W_k e a imagem super-resolvida - computacionalmente custosa e suscetível a um mínimo local - simultaneamente.

Primeiramente, estimam a deformação usando o algoritmo de estimativa de fluxo ótico apresentado por Zach et al. [10]. Posteriormente, utilizam tais resultados para solucionar o problema de minimização. Para regularização, as normas l_1 e l_2 são utilizadas e os resultados são comparados qualitativamente.

Tratando-se especificamente do caso do Kinect, ao final desta etapa, tem-se mapas de profundidade com maior resolução, contudo, ainda com uma porção considerável de ruído. Além disso, os diversos mapas de profundidade ainda não foram integrados ao espaço tridimensional. Essas características demandam o registro global rígido e não-rígido onde os diversos mapas são concatenados dando origem ao modelo tridimensional. Para viabilizarmos a aplicação citada no início do presente capítulo, a saída dessas etapas alimentaria o processo de associação entre o esqueleto detectado pelo Kinect e a malha gerada.

Tal contexto motivou a busca por uma solução que servisse de base para futuros estudos e desenvolvimentos de sistemas que permitissem a utilização de sensores de baixo custo para captação do corpo humano. O já exposto, deixa claro o direcionamento à super-resolução, isto é, à busca por soluções capazes de melhorar os mapas de profundidade disponibilizados por sensores de baixo custo. Conforme já descrito, a utilização de tecnologias abertas e a incorporação de ideias presentes nas soluções fechadas nortearam a pesquisa.

Durante a captação dos diversos mapas de profundidade de baixa resolução necessários ao processo de super-resolução, a roupa de uma pessoa, por exemplo, pode se movimentar. Tal deslocamento gera uma diferença entre os frames de entrada que culmina em artefatos na reconstrução final. Conforme já mencionado, utilizando a biblioteca OpenCV, o presente trabalho experimentou a incorporação das imagens RGB fornecidas pelo dispositivo Kinect à geração de mapas de profundidade super-resolvidos, no intuito de minimizar os problemas mencionados.

A incorporação da consideração de cores proposta por Cui et al. [2] ao termo de correspondência da abordagem de Mitzel et al. [3] se mostrou promissora no tocante à remoção de interferências presentes nos mapas de profundidade captados pelo Kinect. Com isso, o presente trabalho fornece o insumo para as etapas de registro rígido global e não-global citadas anteriormente. Os desafios teóricos e práticos dessa incorporação e os respectivos resultados ficarão claros nos capítulos a seguir.

1.2 Estrutura da Dissertação

A seção anterior contextualizou o escopo da presente pesquisa e expôs a motivação inerente à busca pela melhoria dos mapas de profundidade disponibilizados por sensores de baixo custo.

O restante deste documento se organiza da seguinte maneira:

No Capítulo 2 O esboço geral de um sistema de digitalização 3D é apresentado. Além disso, diversos artigos relacionados a cada módulo de tal sistema são brevemente descritos e relacionados ao presente trabalho.

O Capítulo 3 Se propõe a descrever brevemente as teorias que embasam os algoritmos de super-resolução e estimativa de movimento utilizados na dissertação.

O Capítulo 4 Explicita a teoria que descreve e justifica a inclusão da consideração de cores ao algoritmo de super-resolução descrito no capítulo 3.

O Capítulo 5 Mostra as modificações e inclusões que foram necessárias à biblioteca disponibilizada pela OpenCV para que o o algoritmo de super-resolução proposto no capítulo 4 fosse implementado.

O Capítulo 6 Detalha os testes que foram realizados no intuito de provar a eficácia da modificação proposta e comparar os tempos de execução entre o algoritmo original e o proposto, quando executados na CPU e na GPU.

No Capítulo 7 Se conclui o presente trabalho e se discorre sobre as possibilidades existentes para trabalhos futuros.

O apêndice A concentra alguns dos trechos de código necessários à implementação e citados ao longo do texto. O apêndice B centraliza as representações simbólicas utilizadas nas diversas equações apresentadas ao longo do texto. O apêndice C contém alguns parâmetros da biblioteca OpenCV relacionados ao contexto desse estudo.

2
Trabalhos Relacionados

Um sistema completo de digitalização 3D do corpo humano precisaria de dois elementos básicos. Primeiramente, para uma pessoa específica, deveria ser capaz de capturar as minúcias do seu corpo com razoável qualidade. Além disso, outro módulo associaria a malha gerada às juntas capturadas pelo Kinect e, a cada movimento do usuário, distorceria o modelo. Este capítulo perpassa alguns trabalhos que abordam o sistema completo ou um dos módulos necessários ao mesmo.

2.1
Sistema com Três Kinects

O sistema descrito por Tong et al. [1], propõe a utilização de três Kinects. Dois dos dispositivos, posicionados de forma a evitar qualquer sobreposição dos respectivos infravemelhos, capturam a parte superior e a parte inferior do corpo. O terceiro dispositivo é posicionado de forma diametralmente oposta no intuito de mapear o meio do corpo do usuário. A distância entre os dois conjuntos é de 2 metros. Essa configuração evita a interferência entre o infravermelho destes dispositivos. Para o mapeamento completo, uma plataforma giratória é instalada no centro do sistema permitindo que o corpo analisado seja girado no decorrer da aquisição. Tal arranjo é ilustrado na figura 2.1.

Utilizando a biblioteca OpenNI [11], cada sensor captura imagens RGB de 1280 x 1024 e mapas de profundidade de 640x480, a uma taxa de 15 frames por segundo. Na configuração do sistema, os dados dos três sensores independentes são calibrados e sincronizados automaticamente, conforme descrito por Tomas Svoboda [12]. Como os sensores estão a apenas 1 metro do corpo analisado, a qualidade do mapa de profundidade corresponde àquela necessária ao processo de reconstrução. Para redução de ruído é aplicado um filtro de suavização Laplaciano conforme proposto por Sorkine et al. [13].

O sistema, de início, registra um template bem grosseiro do corpo humano. Este template é utilizado para deformar sucessivos frames. Para distribuir os erros decorrentes das deformações, o registro global das geometrias é utilizado, tratando problemas como a oclusão. Sucessivas iterações, alternando entre o registro emparelhado e o global, ocorrem até que o algoritmo convirja. Por fim, o modelo é reconstruído utilizando o método de reconstrução de Poisson conforme descrito por Kazhdan et al. [14].

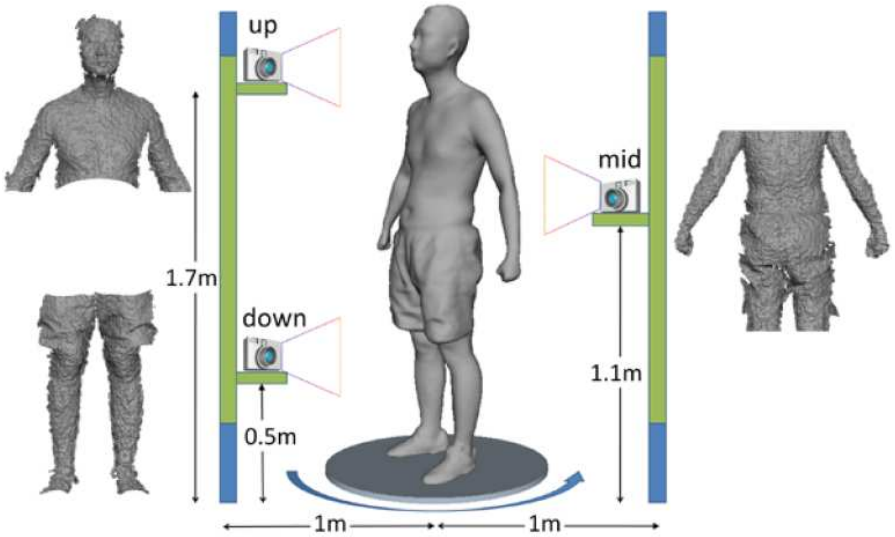


Figura 2.1: Figura retirada de Tong et al. [1] - Arranjo básico do sistema de digitalização do corpo humano.

2.2 Sistema com um único Kinect

Cui et al. [2] utilizam a sequência ilustrada na figura 1.2 para permitir a utilização de um único Kinect para a obtenção de malhas com um nível de detalhe até maior que aqueles alcançados por Tong et al. [1]. Como depende só dos dados fornecidos pelo Kinect e não requer um modelo previamente digitalizado como referência, consegue reproduzir em detalhes as geometrias das faces e dos vestuários. As contribuições de Cui et al. [2] são:

- Uma sequência de procedimentos para digitalizar automaticamente um corpo humano por completo, utilizando o Kinect;
- um algoritmo de super-resolução melhorado que leva em conta as restrições de cor;
- uma formulação unificada de registros rígidos e não-rígidos sob um modelo probabilístico, melhorando os resultados de Cui et al. [15] e Chang e Zwicker [16] no tocante à qualidade em cenários de alto ruído e à solução do problema de fechamento de loop.

A baixa resolução e o alto nível de ruído nos dados de profundidade disponibilizados pelo Kinect tornam necessária uma etapa de suavização das

superfícies geradas a partir do mapa fornecido pelo dispositivo. Newcombe et al. [17] aplicam um filtro bilateral no intuito de obter um mapa com ruído reduzido e descontinuidades preservadas. Schuon et al. [18] desenvolvem um algoritmo de super-resolução capaz de aumentar a resolução da profundidade e a qualidade dos dados gerados por um digitalizador baseado em luz estruturada. Mais tarde, Cui et al. [15] melhoram tal método. Cui et al. [2] elaboram um novo algoritmo para o processamento dos dados de cor e profundidade do Kinect. Tal algoritmo apresenta uma maior resolução, reduz os ruídos e preserva os detalhes da superfície original.

Cui et al. [15] abordaram o registro rígido global através de um modelo probabilísitco do alinhamento da varredura que leva em consideração as caracterísitcas de ruído do sensor. Contudo, essa estratégia só soluciona o alinhamento local. Os mesmos autores propuseram um algoritmo de alinhamento probabilístico global [2].

A deformação não-rígida do corpo humano durante a varredura faz com que o resultado da etapa anterior não seja o ideal. Se aproveitando do fato do corpo humano ser altamente articulado, Cui et al. [2] melhoram o algoritmo apresentado por Chang e Zwicker [16] utilizando um modelo probabilístico de alinhamento da varredura robusto aos ruídos do dispositivo.

Cui et al. [2], computando as devidas transformações a partir de uma função de energia global, solucionam o já conhecido problema de fechamento de loop. Por fim, um mapeamento de texturas é aplicado para acoplar ao modelo a aparência do corpo digitalizado.

2.3 Autorigging

Uma vez obtida a malha, precisamos distorcê-la conforme os movimentos detectados. Baran e Popović [19] adaptam um esqueleto à uma malha. A implementação e a avaliação do método se basearam na generalidade, qualidade e perfomance.

Primeiramente, o esqueleto é escalado e posicionado para caber dentro da malha a qual será associado. Para isso, Baran e Popović [19] constroem um grafo no qual os vértices representam as juntas em potencial e as arestas os ossos em potencial. Esse grafo é construído arrumando esferas cujos centros estão nas superfícies mediais aproximadas da malha e conectando estes centros com as arestas do grafo. A melhor solução é encontrada a partir da solução discreta da função de penalidade descrita em [20] que serve de base para posteriores otimizações.

A discretização se constitui nas seguintes etapas:

- obtenção do campo de distâncias;
- obtenção da superfície medial aproximada;
- arrumação das esferas;
- construção do grafo.

Após a discretização, um grafo reduzido é gerado eliminando-se todos os vértices de grau 2 (ex.: joelhos). Em cima do novo grafo, a função de penalidade é aplicada. Ao esqueleto resultante, são atribuídas as juntas retiradas no início do processo. Tal solução é refinada utilizando uma nova função de penalidade - ver figura 2.2.

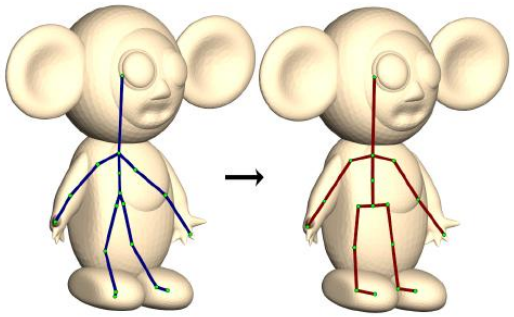


Figura 2.2: Figura retirada de Baran e Popović [19] - Refinamento do esqueleto.

Por fim, a malha deve ser associada ao esqueleto obtido nas etapas anteriores. Para isso, Baran e Popović [19] utilizam o método de Linear Blend Skinning. Se v_j é a posição do vértice j, T^i é a transformação do i^{esimo} osso, e w_j^i é o peso do i^{esimo} osso do vértice j, LBS dá a posição do vértice j transformado segundo $\sum_i w_j^i T^i(v_j)$. Encontrando quanto a transformada de cada osso afeta cada junta se chega ao modelo ideal de como a malha é distorcida pelo esqueleto - ver figura 2.3.

2.4 O sistema completo

Seguindo a linha de tornar o sistema financeiramente viável para uma maior parcela da sociedade, escolheu-se, para a digitalização 3D, o sistema, baseado em um único Kinect, apresentado em 2.2. O mesmo apresenta um maior nível de detalhes em comparação ao 2.1, sendo, contudo, mais lento.

Por fim, associaríamos o esqueleto fornecido pelo Kinect à malha obtida utilizando a estratégia proposta por Baran e Popović [19]. A implementação

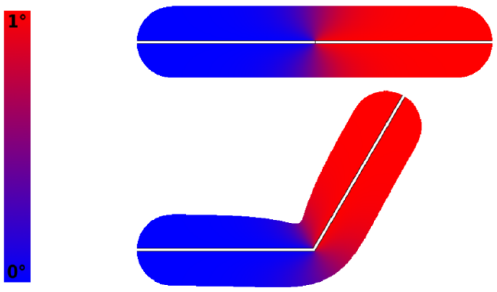


Figura 2.3: Figura retirada de Baran e Popović [19] - Fazendo uma analogia ao equilíbrio de calor.

de todo esse sistema parte da geração de mapas de profundidade super-resolvidos. Devido a isso, conforme já exposto na seção 1.1, o presente trabalho se concentrou no processo de super-resolução. A geração da malha, assim como a associação do esqueleto à mesma, são deixados como trabalho futuro.

3
Marco teórico

3.1
Super-resolução utilizando a regularização anisotrópica e o mapa de cores

Cui et al. [2] utilizando o Kinect, captam mapas de profundidade de pontos de vista levemente deslocados representados por $Y_k \in \Re^{n \times m}$. Utilizando a estimativa de movimentação, descrita por Lucas e Kanade [21], as mesmas são registradas em relação ao primeiro frame. Posteriormente, cada imagem é reamostrada utilizando a interpolação por vizinho mais próximo, originando as correspondentes $D_k \in \Re^{\beta n \times \beta m}$. Tais imagens servem de base ao processo de otimização descrito pela seguinte equação:

$$minimiza \quad E_{data}(X) + E_{regular}(X),$$

onde $X \in \Re^{\beta n \times \beta m}$ representa a imagem super-resolvida objetivada pelo algoritmo. O primeiro termo mede a correspondência entre X e os mapas de baixa resolução, garantindo, assim, a coerência entre o mapa super-resolvido e os frames originais representados por Y_k.

$$E_{data}(X) = \sum_{k=1}^{N} \parallel M_k * T_k * (D_k - X) \parallel_2,$$

onde $*$ denota multiplicação termo-a-termo. $M_k \in \Re^{\beta n \times \beta m}$ é uma matriz que armazena as posições de D_k que se originaram do processo de reamostragem. $T_k \in \Re^{\beta n \times \beta m}$ é uma matriz diagonal contendo 0 para todas as amostras de D_k que não são confiáveis de acordo com as leituras do Kinect. Schuon et al. [18] se baseiam na amplitude do sinal do sensor de profundidade do Swiss Ranger 3000 para tomar tal decisão no tocante à confiabilidade dos dados.

O termo $E_{regular}$ se propõe a preservar as componentes de alta frequência presentes nas cenas originais e suprimir as componentes correspondentes a ruídos. Enfim, busca preservar as superfícies convexas das cenas originais tendo sido elaborado para reconstruções 3D de alta qualidade.

$$E_{regular}(X) = \sum_{u,v} \parallel \nabla X_{u,v} \parallel_2 = \sum_{u,v} \left\| \begin{pmatrix} G_{u,v}(0,1) \\ G_{u,v}(1,0) \\ ... \\ G_{u,v}(j,l) \end{pmatrix} \right\|_2$$

onde $\nabla X_{u,v}$ é um vetor que representa o gradiente da diferença espacial fininita entre o pixel na posição (u, v) e os demais. Sendo $G_{u,v}(j, l)$ a seguinte diferença finita:

$$\frac{X(u,v) - X(u+j, v+l)}{\sqrt{j^2 + l^2}}.$$

onde j varia de 0 a $(\beta n - u)$ e l varia de 0 a $(\beta m - v)$, de forma a computar a diferença finita de todos os pixels com indíces superiores àqueles do pixel em questão. Por fim, temos a seguinte equação:

$$\sum_{k=1}^{N} \parallel M_k * T_k * (D_k - X) \parallel_2 + \lambda \sum_{u,v} \parallel \nabla X_{u,v} \parallel_2$$

Onde λ é um parâmetro de troca entre forçar a similaridade dos dados e a suavidade do resultado, isto é, λ indica o quanto o gradiente espacial $\parallel \nabla X_{u,v} \parallel_2$ influencia a imagem super-resolvida gerada. Em outras palavras, um λ maior diminui as componentes de alta frequência da imagem reconstruída.

3.1.1 Modificação do termo de regularização

Cui et al. [22] modificam o algoritmo do item anterior, incorporando o regularizador anisotrópico não-linear à equação. O novo $E_{regular}$ é $G(\nabla H_l \nabla {H_l}^T)$. Onde a função G está definida como

$$G(\nabla X_{u,v} \nabla {X_{u,v}}^T) = (x1...x_{lm-1}) diag(y_1...y_{lm-1}) (x_1^T...x_{lm-1}^T)^T$$

onde x representa os autovalores e y os autovetores da matriz resultante da operação $\nabla X_{u,v} \nabla {X_{u,v}}^T$. Sendo assim, temos um tensor de difusão no lugar de uma função de difusividade baseada em valores escalares. Esse novo regularizador gera reconstruções mais fiéis das áreas suaves do modelo. Além disso, preserva as características estruturais e detalhes de menor escala.

3.1.2 Acrescentando informações de cor

Cui et al. [2] também modificam o termo E_{data} que mede a correspondência entre os mapas de profundidade originais e as superamostragens de alta resolução.

Partindo de:

$$E_{data}(X) = \sum_{k=1}^{N} \parallel M_k * T_k * (D_k - X) \parallel_2$$

o termo M_k passa a ser:

$$\frac{1}{C_k - \frac{1}{s}\sum_{i=1}^{s} C_i}$$

onde C_k corresponde à imagem RGB captada concomitantemente ao mapa de profundidade Y_k. O índice i varia entre as diversas imagens RGB utilizadas na reconstrução corrente, sendo s a quantidade de imagens utilizadas. M_k produz um valor maior se a cor no frame original é similar à cor média dos demais. Pesos menores indicam alinhamento incorreto.

3.2 Super-resolução em tempo real utilizando o algoritmo TV-L1

Conforme já mencionado, Mitzel et al. [3] dissociam a estimativa da matriz de deformação do processo de resolução da imagem super-resolvida, solução computacionalmente custosa e que tende a um mínimo local. A figura 3.1 descreve o esquema básico desta abordagem.

3.2.1 Optical Flow: a teoria por trás do TV-L1

Conforme descrito por Zach et al. [10], recuperar a movimentação ocorrida em determinado conjunto de imagens é um dos grandes desafios de sistemas de visão tanto biológicos quanto artificiais. Os métodos de estimativa de movimentação se propõem a descrever, através de vetores, a movimentação de cada um dos pixels entre duas imagens diferentes. Este tipo de problema é muito suscetível a ruídos e outros fatores que poluem os níveis de intensidade que descrevem cada pixel. Essa caracterísitca demanda um termo de regularização que objetiva minimizar as inconsistências geradas.

O problema básico perpassa duas imagens I_0 e $I_1 : \Omega \subseteq \mathbb{R}^2$. Baseando-se na minimização de um critério de erro e uma força de regularização pretende-se chegar ao mapa de disparidade $u : \Omega \subseteq \mathbb{R}^2$. Zach et al. [10] consideram

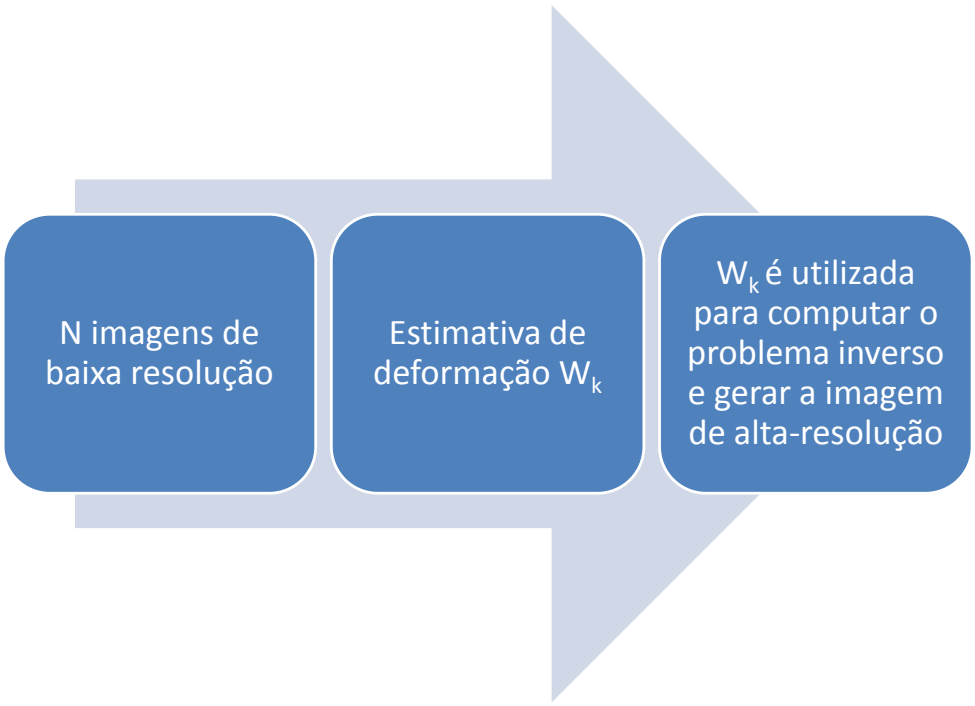


Figura 3.1: Esquemático básico da proposta de Denniz Mitzel [3].

a diferença entre os níveis de intensidade das imagens como o parâmetro de similaridade. Sendo assim, u minimiza a equação

$$\int_{\Omega} \{\lambda\phi(I_0(x) - I_1(x + u(x))) + \psi(u, \nabla u, ...)\}\, dx \tag{3-1}$$

onde $\phi(I_0(x) - I_1(x + u(x))$ é o termo de fidelidade dos dados, e $\psi(u, \nabla u, ...)$é o termo de regularização. λ indica o quanto a fidelidade dos dados e a força de regularização influenciam a solução. Se assumirmos $\phi(x) = x^2$ e $\psi(\nabla u) = \mid \nabla u \mid^2$, temos o modelo proposto por Horn and Schunck [23].

Fazendo-se $\phi(x) = \mid x \mid$ e $\psi(\nabla u) = \mid \nabla u \mid$ temos a seguinte função:

$$E = \int_{\Omega} \{\lambda \mid I_0(x) - I_1(x + u(x)) \mid + \mid \nabla u \mid\}\, dx \tag{3-2}$$

Tendo como base a equação (3-1), limitam-se as disparidades apenas à direção horizontal da imagem, isto é, inicialmente, o problema é restringido a apenas uma dimensão. Linearizando I_1 nas proximidades de $x + u_0$, temos

$$I_1(x + u) = I_1(x + u_0) + (u - u_0)I_1^x(x + u_0)$$

Onde u_0 é um mapa de disparidade inicial e I_1^x é a derivada da imagem I_1 em relação à direção x. Ao utilizar a aproximação de primeira ordem de Taylor para I_1 geramos a necessidade de deformações iterativas para compensar as não-linearidades da imagem. Fixando u_0 e utilizando a aproximação linear

para I_1, temos

$$E = \int_\Omega \{\lambda \mid I_0 - uI_1^x - I_1(x + u_0) + u_0 I_1^x \mid + \mid \nabla u \mid\}\, dx \tag{3-3}$$

Substitui-se $I_1(x + u_0) + (u - u_0)I_1^x - I_0$ por $\rho(u, u_0, x)$. Omite-se sua dependência explícita de u_0 e x e, além disso, se introduz uma variável auxiliar v, propondo-se a minimização da seguinte aproximação:

$$E_\theta = \int_\Omega \left\{ \mid \nabla u \mid + \frac{1}{2\theta}(u - v)^2 + \lambda \mid \rho(v) \mid \right\} dx \tag{3-4}$$

onde θ é uma pequena constante, tal que v é bem próxima a u. Esta nova variável v e o termo $\frac{1}{2\theta}(u - v)^2$ - que, sendo muito pequeno, não impacta o resultado final de E - permitem que a solução do termo de regularização e do termo de fidelidade dos dados, ocorram em momentos separados. Para isso, a cada iteração, incrementamos u ou v conforme descrito a seguir.

Ao se fixar v, a seguinte equação dever ser resolvida:

$$min_u \int_\Omega \left\{ \mid \nabla u \mid + \frac{1}{2\theta}(u - v)^2 \right\} dx \tag{3-5}$$

Esse é o modelo de retirada de ruídos proposto por Rudin et al. [24].

Ao se fixar u, se soluciona:

$$min_v \int_\Omega \left\{ \frac{1}{2\theta}(u - v)^2 + \lambda \mid \rho(v) \mid \right\} dx \tag{3-6}$$

Tal problema de minimização pode ser solucionado ponto a ponto, já que não depende da variação espacial de v.

Chambolle [25] propôs uma maneira eficiente de gerar o primeiro passo, equação 3-5. Zach et al. [10] mencionam essa abordagem e reproduzem os resultados conforme mostrado a seguir.

Proposição 1

A solução da equação (3-5) é dada por

$$u = v - \theta\ div\ p, \tag{3-7}$$

onde o operador div representa a divergência de p e $p = (p^1, p^2)$ leva a

$$\nabla(\theta\ div\ p - v) = \mid \nabla(\theta\ div\ p - v) \mid p, \tag{3-8}$$

que pode ser solucionado pelo seguinte esquema

$$p^{k+1} = \frac{p^k + \tau\nabla(div\ p^k - v/\theta)}{1 + \tau \mid \nabla(div\ p^k - v/\theta) \mid}, \tag{3-9}$$

onde $p^0 = 0$ e o passo $\tau \leq 1/8$.

A seguir, é mostrado com se chegar ao minimizador da equação (3-6).

Proposição 2

A solução da tarefa de minimização da equação (3-6) se dá por:

$$v = u + \begin{pmatrix} \lambda\theta I_1^x & \text{se} & \rho(u) < -\lambda\theta(I_1^x)^2 \\ -\lambda\theta I_1^x & \text{se} & \rho(u) > \lambda\theta(I_1^x)^2 \\ -\rho(u)/I_1^x & \text{se} & \mid \rho(u) \mid \leq \lambda\theta(I_1^x)^2 \end{pmatrix} \tag{3-10}$$

Se a diferença de u para v é suficientemente pequena, $\rho(v)$ pode ser dispensado. Caso contrário, v descreve um passo limitado de u, tal que a magnitude do resíduo descresce.

Conforme Zach et al. [10], a proposição acima pode ser demonstrada diretamente através da análise de três casos:

$$\rho(v) > 0 \ (incluindo\ v = u - \lambda\theta I_1^x)$$
$$\rho(v) < 0 \ (v = u + \lambda\theta I_1^v)$$
$$\rho(v) = 0 \ (v = u - \rho(u)/I_1^x)$$

Agora, para empregar o exposto para estimativa de movimentação de imagens, deve-se generalizar o problema, contemplando um mapa de disparidade $N - dimensional$ $\mathbf{u}$. As alterações se iniciam com o resíduo de primeira ordem $\rho(\mathbf{u}, \mathbf{u}_0, \mathbf{x})$ relacionado a um dado mapa de disparidade $\mathbf{u}_0$. ρ é agora $\mathbf{I}_1(\mathbf{x} + \mathbf{u}_0) + \langle \nabla I_1, \mathbf{u} - \mathbf{u}_0 \rangle - I_0(\mathbf{x})$.

Sendo u_d a d-ésima componente de $\mathbf{u}$ ($d \in \{1, \ldots, N\}$), temos a seguinte generalização da equação $(3-4)$:

$$E_\theta = \int_\Omega \{\textstyle\sum_d \mid \nabla u_d \mid + \sum_d \frac{1}{2\theta}(u_d - v_d)^2 + \lambda \mid \rho(v) \mid\} dx \tag{3-11}$$

Seguindo o mesmo raciocínio do caso unidimensional, a minimização de energia se dá da seguinte forma:

- Para todo d e fixado v_d, soluciona-se

$$min_{u_d} \int_\Omega \left\{ \mid \nabla u_d \mid + \frac{1}{2\theta}(u_d - v_d)^2 \right\} dx, \tag{3-12}$$

  Tal problema pode ser solucionado através do mesmo método utilizado para a equação (3-5). Sendo $u_d = v_d - \theta\ div\ p_d$, seguindo o mesmo raciocínio que nos levou à equação (3-8).

- Fixando u, soluciona-se

$$min_v \sum_d \frac{1}{2\theta}(u_d - v_d)^2 + \lambda \mid \rho(v) \mid \tag{3-13}$$

Proposição 3

A solução da tarefa de minimização da equação (3-12) se dá por:

$$v = u + \begin{pmatrix} \lambda\theta\nabla I_1 & \text{se} & \rho(u) < -\lambda\theta \mid \nabla I_1 \mid^2 \\ -\lambda\theta\nabla I_1 & \text{se} & \rho(u) > \lambda\theta \mid \nabla I_1 \mid^2 \\ -\rho(u)\nabla I_1/\mid \nabla I_1 \mid^2 & \text{se} & \mid \rho(u) \mid \leq \lambda\theta \mid \nabla I_1 \mid^2 \end{pmatrix} \tag{3-14}$$

A versão N-dimensional pode ser reduzida a uma dimensão, dado que v sempre se situa na linha $l^{\perp}$ a u com direção ∇I_1 (para todo x). A primeira parte na equação (3-12), $\sum_d \frac{1}{2\theta}$, corresponde a distância quadrada de v para u, e a segunda parte, $\lambda \mid \rho(v) \mid$, é o módulo da distância para a linha l: $\rho(w) = 0$. Se considerarmos todos v_μ como a distância fixa de μ a u, a função na equação (3-12) é minimizada pelo v_μ mais próximo a linha l (com distância normal mínima).

3.2.2
OpticalFlow: Implementação em GPU para N = 2

A linearização dos níveis de intensidade das imagens que compõem a equação (3-2) conduz a um problema de minimização convexa. Dado que tal procedimento só é válido para pequenos deslocamentos, no intuito evitar mínimos locais, o procedimento de minimização de energia é embutido em uma abordagem *"coarse-to-fine"*. Zach et al. [10] empregam pirâmides de imagem com fator de subamostragem igual a 2. Partindo do nível de menor detalhe, soluciona-se a equação (3-2) para cada nível da pirâmide e se propaga a solução para o nível subsequente. Além disso, a função residual linear ρ é computada através das amostragens de I_0 *e* I_1 nos respectivos níveis das pirâmides. Portanto, o passo de deformação para I_1 ocorre uma vez a cada nível. ∇_1 é aproximado por diferenças centrais. No começo do próximo nível, v é incializado com u, e todos os p_d são inicializados com 0. No nível menos detalhado, onde tudo começa, o vetor u é incializado com 0.

A abordagem *"coarse-to-fine"* também proporciona uma aceleração do processo de preenchimento, das regiões sem textura, induzido pela regularização. O procedimento de minimização alterna um passo do esquema de ponto fixo para atualizar todos os p_d, consequentemente u - equação (3-9), com o incremento à v utilizado na *proposição 3* da seção 3.2.1.

Conforme descrito por Zach et al. [10], métodos numéricos dedicados a grades regulares, podem ser potencialmente acelerados pelas unidades de processamento gráfico (GPUs) modernas. Se emprega o alto poder computacional e as capacidades de processamento paralelo para obter uma implementação mais rápida do algoritmo de estimativa de fluxo previamente descrito.

Basicamente, a implementação se realiza com base nas equações (3-9) e

(3-10). As seguintes atualizações se realizam em u, v e p_d:

$$\begin{aligned} 1a.\ v^{k+1} &\longleftarrow TH(u^k) \\ 1b.\ u_d^{k+1} &\longleftarrow v_d^{k+1} - \theta\ div\ p_d^k\ para\ d \in \{1,2\} \\ 2.\ p_d^{k+1} &\longleftarrow \frac{p_d^k + \tau/\theta \nabla u_d^{k+1}}{1+\tau/\theta|\nabla u_d^{k+1}|}\ para\ d \in \{1,2\} \end{aligned} \tag{3-15}$$

onde $TH(.)$ denota o passo limiar da equação (3-10). Tais passos são perfeitamente implementáveis numa GPU por fragmentos apropriados utilizando dois passos de renderização. A primeira etapa implementa os passos $1a$ e $1b$ da equação (3-15). Os valores v^{k+1} são utilizados somente temporariamente no programa *shader* e não precisam ser armazenados. u^{k+1} é escrito para a textura alvo. O segundo programa *shader* corresponde ao passo 2 da equação (3-15). Sendo assim, a utilização dos processadores de fragmento pode ser melhorada atualizando u e p_d para dois pixels simultaneamente. O programa *shader* trabalha nas porções esquerda e direita da imagem em paralelo, com o apropriado tratamento às bordas.

Se utiliza um número de iterações fixo, porém customizável, em cada nível da implementação descrito por Zach et al. [10], pois determinar a atualização máxima de $\mid u^{k+1} - u^k \mid$ ainda demanda uma custosa operação de redução até mesmo nas GPUs modernas.

3.3 Super-resolução variacional

Mitzel et al. [3] incorporam o termo de regularização à equação (1-4), impondo suavidade espacial à imagem estimada I_H. Se baseando em Marquina e Osher [26], a norma L_2 do seu gradiente é penalizada:

$$E(I_H) = 1/2 \sum_{k=1}^{N} \int_{\Omega} \mid D_k B_k W_k I_H - I_L^{(k)} \mid^2 + \lambda \mid \nabla I_H \mid^2 dx \tag{3-16}$$

Sem tal termo de regularização, o problema inverso não possui uma única solução que dependa exclusivamente das medições. O parâmetro λ permite ajustar a importância relativa do regularizador.

Para se chegar a solução I_H, a função de energia (3-16) é minimizada mediante a solução da seguinte equação de Euler-Lagrange:

$$\frac{dE}{dI_H} = \sum_{k=1}^{N} W_k^T B_k^T D_k^T (D_k B_k W_k I_H - I_L^{(k)}) - \lambda \triangle I_H = 0 \tag{3-17}$$

Os operadores lineares D_k^T, W_k^T e B_k^T denotam os operadores inversos associados às operações de sub-amostragem, deformação e "blurring" inerentes

ao processo de formação da imagem. D_k^T é implementado como uma simples super-amostragem sem interpolação. B_k^T pode ser implementado a partir do conjugado do kernel: se $h(i,j)$ é o kernel de "blur", então o conjugado $\tilde{h}$ satisfaz para todo i,j: $\tilde{h}(i,j) = h(-i,-j)$. Mitzel et al. [3] modelam o "blurring" através de uma convolução com um kernel isotrópico Gaussiano. A isometria implica que h é simétrico e, consequentemente, $\tilde{h}$ é idêntico a h. Além disso, assume que os processos de "blurring" e sub-amostragem são idênticos para todas as imagens observadas. Sendo assim, o índice k pode ser descartado nos operadores B e D. O operador W_k^T é implementado por deformação.

A equação (3-17) é solucionada através do método numérico "Steepest Descendent":

$$I_H^{n+1} = I_H^n + \tau \left(\sum_{k=1}^{N} W_k^T B^T D^T (I_L^{(k)} - DBW_k I_H^{(n)}) + \lambda \triangle I_H^{(n)} \right) \qquad (3\text{-}18)$$

onde τ descreve o avanço no tempo a cada iteração. Os dois termos na "evolução" da imagem de alta resolução I_H induzem a mesma (após os processos de deformação, "blurring" e sub-amostragem) a corresponder a todas as observações e, ao mesmo tempo, impõe uma difusão linear das intensidades ajustadas por λ.

3.3.1 O Algoritmo

O algoritmo 3.1 tem o objetivo de, dada uma sequência de N imagens de baixa resolução $\{I_L^k\}_{k=1}^N$, estimar a movimentação entre frames e inferir a imagem de alta resolução I_H da cena em questão.

Algoritmo 3.1 Algoritmo proposto por Mitzel et al. [3].

1: Escolher uma imagem da sequência como referência.

2: Estimar para cada par de frames consecutivos a movimentação do frame atual para o próximo (figura 3.2) utilizando o algoritmo de estimativa de movimento apresentado na seção 3.2.1.

3: Utilizando os campos de movimentação u_i^f e v_i^f compute os campos de movimentação u_i^r, v_i^r relacionados à imagem de referência (figura 3.3), observando-se que os índices f (movimento entre frames mútuos) e r (movimento entre frames de referência e imagens individuais) devem indicar a diferenças entre os mapas de movimentação.

4: Interpolar o campos de movimentação u_i^r e v_i^r de forma atingir a dimensão da imagem I_H.

5: Inicializar I_H, setando todos os valores para 0.

6: de $t = 1$ a T faça

7: $\quad sum := 0$;

8: $\quad$ de $k = 1$ a N faça

9: $b := W_k I_H^t$ (deformação "backward");
10: $c := h(x, y) * b$ (convolução com o kernel Gaussiano);
11: $c := D_c$ (sub-amostragem para a dimensão de I_L);
12: $d := (I_L^k - c)$;
13: $b := D^T d$ (super-amostragem sem interpolação);
14: $c := h(x, y) * b$
15: $d := W_k^T c$ (deformação "forward");
16: $sum := sum + d$;
17: fim para
18: $I_H^{t+1} = I_H^t + \tau(sum - \lambda \triangle I_H^t)$;
19: fim para

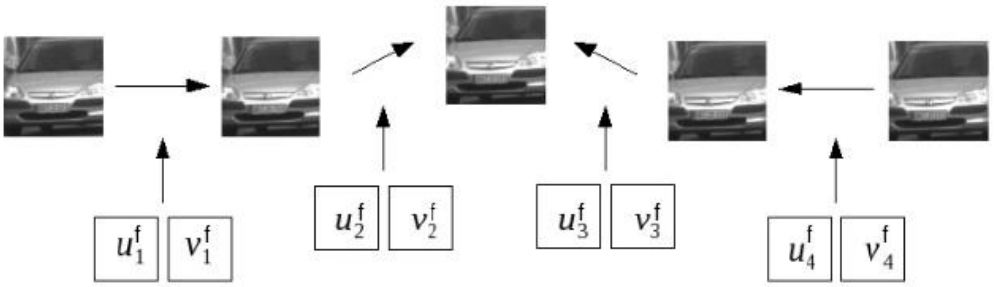


Figura 3.2: Estimativa de movimentos entre frames [3].

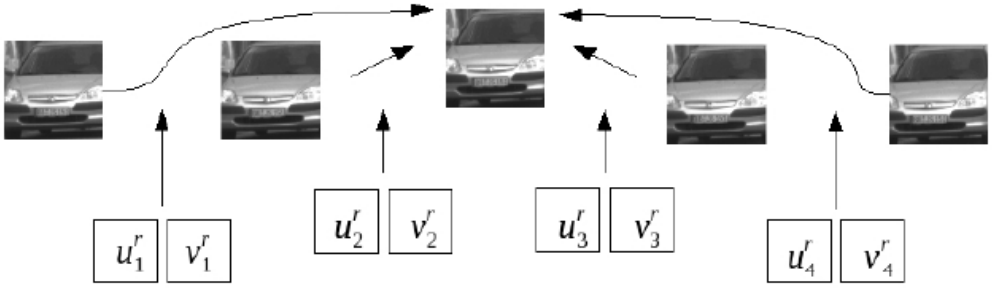


Figura 3.3: Esta figura mostra como a estimativa de movimento entre os frames e as imagens de referência ocorre [3].

Na presente dissertação, serão utilizadas as implementações deste algoritmo tanto em CPU quanto em GPU, disponibilizadas pela biblioteca OpenCV. Tais implementações foram modificadas visando a incorporação das imagens RGB ao algoritmo de super-resolução, conforme será detalhado a seguir.

4
Incorporando as cores à super-resolução

Conforme apontado no capítulo 1, o presente trabalho foca no desenvolvimento de um algoritmo de super-resolução eficiente para ser aplicado aos mapas de profundidade disponibilizados pelo Kinect. Durante as pesquisas, a classe de super-resolução disponibilizada pela biblioteca OpenCV [27] se mostrou a mais promissora como cerne deste trabalho.

Bem documentada e amplamente utilizada pela comunidade, está disponível para uso acadêmico e comercial sob o licenciamento BSD. A respectiva classe de super-resolução tem por base a estratégia apresentada por Mitzel et al. [3] descrita resumidamente no capítulo 3. Contudo, o foco de Mitzel et al. [3] está em imagens contidas em sequências de vídeo. Tal cenário não considera ambientes onde temos a informação de dois sensores, que disponibilizam dados diferentes, acerca da mesma cena, como o ocorre com o Kinect, que disponibiliza dados de profundidade e imagens RGB de uma mesma cena. Conforme já exposto no capítulo 1, o objetivo da pesquisa era, a partir da utilização da OpenCV, se implementar a etapa de super-resolução de Cui et al. [2], também descrita no capítulo 3, que aproveita as imagens RGB na reconstrução dos mapas de profundidade. O objetivo de Cui et al. [2] é gerar malhas do corpo humano, em tempo real, a partir dos dados disponibilizados pelo Kinect. A partir deste quadro, a pesquisa se dedicou a unificar o algoritmo, já implementado na OpenCV [27], aos benefícios do enfoque de Cui et al. [2], no tocante à super-resolução.

Se tratarmos os dados de profundidade disponibilizados pelo Kinect como níveis de intensidade, o algoritmo de super-resolução disponibilizado na OpenCV [27] se adequa perfeitamente a esse mapa de informação. Sendo assim, o primeiro passo da presente implementação foi capturar os dados de profundidade disponibilizados pelo Kinect como níveis de cinza. Dessa forma, o algoritmo de super-resolução proposto por Mitzel et al. [3] não demandaria qualquer alteração que visasse a geração de mapas super-resolvidos.

Contudo, a imagem RGB concomitamente disponiblizada pelo sensor, não era aproveitada de forma alguma neste processo. Utilizando a contribuição teórica de Cui et al. [2], descrita na seção 3.1.2, podemos adaptar a abordagem de Mitzel et al. [3], conforme mostrado na figura 4.1 - as alterações realizadas em relação ao esquemático básico apresentado na figura 3.1 estão destacadas em negrito.

Vale ressaltar que, tanto a utilização da proposta de Cui et al. [2], como a adaptação da abordagem de Mitzel et al. [3], apenas foram viáveis, devido à rotação desprezível do objeto captado entre os frames de determinado agrupamento. No caso da adaptação, por exemplo, tal característica entre frames permitiu uma interpretação dos mapas de profundidade como imagens representadas por níveis de intensidade, sem impactar em nada o algoritmo de super-resolução proposto inicialmente para sequências de frames de vídeo.

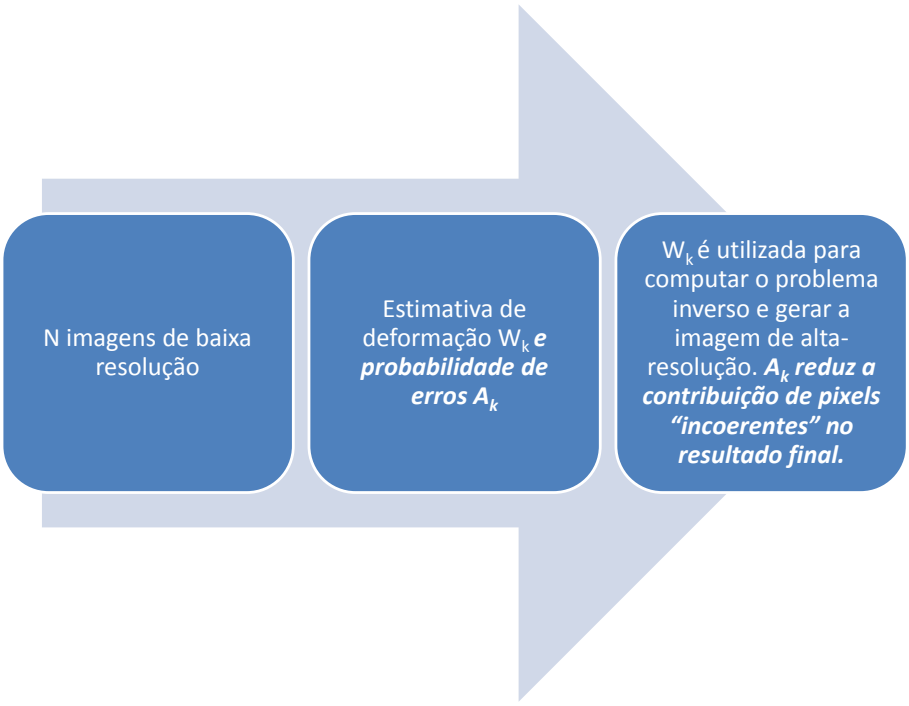


Figura 4.1: Esquemático da adaptação sugerida.

Seguindo a proposta de Cui et al. [2], o termo de ajuste $A^{(k)}$ é igual a

$$\frac{1}{C_k - \frac{1}{s}\sum_{i=1}^{s} C_i,}$$

onde C_k representa o frame RGB correspondente a cada mapa de profundidade k, isto é, relacionado ao respectivo $I_L^{(k)}$. O índice i varia entre as diversas imagens RGB utilizadas na reconstrução corrente, sendo s a quantidade de imagens utilizadas. Considerando-se que a equação (3-18) descreve o processo de reconstrução da imagem super-resolvida a partir do método numérico "Steepest Descendent", esta deve ser alterada de maneira a comportar o ajuste fornecido pela contribuição do mapa de cores. Tal adaptação nos leva à seguinte equação:

$$I_H^{n+1} = I_H^n + \tau \left(\sum_{k=1}^{N} A^{(k)} * \left(W_k^T B^T D^T (I_L^{(k)} - DBW_k I_H^{(n)}) \right) + \lambda \triangle I_H^{(n)} \right) \quad (4\text{-}1)$$

onde $*$ representa o produto termo-a-termo de $A^{(k)}$ com o termo de regularização proposto por Mitzel et al. [3] originalmente. A matriz C_k dita o dimensionamento da matriz de ajuste $A^{(k)}$. Contudo, esta é multiplicada termo-a-termo com o valor resultante de $(W_k^T B^T D^T (I_L^{(k)} - DBW_k I_H^{(n)})$, isto é, precisa ter dimensão $\beta n \times \beta m$. No presente trabalho, utilizamos β - fator de aumento da resolução - igual a 2. O mapa de profundidade de baixa resolução disponibilizado pelo Kinect nos remete a $n = 640$ e $m = 480$. Sendo assim, a dimensão de nosso termo de ajuste deve ser 1280×960. Contudo, o Kinect nos fornece imagens RGB de 1280×1024. Sendo assim, o segundo passo da incorporação de cores foi adaptar tais imagens, de forma que cada matriz C_k contivesse a respectiva imagem com resolução de 1280×960. Detalhes dessa implementação estão descritos na seção 5.3.

Posteriormente, após se garantir que, para o instante k, a captura do mapa de baixa resolução I_L e a geração da respectiva imagem RGB já processada C ocorrem de forma concomitante, as matrizes $I_L^{(k)}$ e C_k são processadas conforme o algoritmo descrito na seção 4.1. Partindo-se do algoritmo de Mitzel et al. [3], a base para este novo algoritmo foi a equação (4-1).

Em resumo, antes do processamento do novo algoritmo, temos as etapas preliminares descritas no algoritmo 4.1.

Algoritmo 4.1 Processamento preliminar

1: Adapatar o mapa de profunidade de baixa resolução à representação por níveis de intensidade.
2: Adapatar a resolução das imagens RGB de 1280×1024 para 1280×960.
3: Garantir concomitância entre a captura do mapa de baixa resolução I_L e a geração da respectiva imagem RGB já processada C.

O novo algoritmo garante que, se a imagem colorida correspondente ao instante k, apresenta alguma diferença quando comparada às demais do agrupamento, o respectivo mapa $I_L^{(k)}$, contribuirá menos que os demais para o resultado da super-resolução. Isso se deve a multiplicação termo-a-termo incorporada pois, o termo de ajuste $A^{(k)}$, apresenta valores menores para os pixels que são muito diferentes em relação aos seus correspondentes nos demais frames.

4.1 Novo algoritmo

O objetivo do algoritmo 4.2 é: dada uma sequência de N imagens de baixa resolução $\{I_L^k\}_{k=1}^N$ estimar a movimentação entre frames e inferir a imagem de alta resolução I_H da cena em questão. Caso alguma disparidade no mapa de cores $C^{(k)}$ correspondente a cada frame $I_L^{(k)}$ seja detectada, tal imagem de baixa resolução não deve contribuir para o resultado final.

Algoritmo 4.2 Novo algoritmo - as partes em negrito destacam as diferenças em relação ao algoritmo 3.1

1: Escolher uma imagem da sequência como referência.
2: Estimar para cada par de frames consecutivos a movimentação do frame atual para o próximo (figura 3.2) utilizando o algoritmo de estimativa de movimento apresentado na seção 3.2.1.
3: Utilizando os campos de movimentação u_i^f e v_i^f compute os campos de movimentação u_i^r, v_i^r relacionados à imagem de referência (figura 3.3), observando-se que os índices f (movimento entre frames mútuos) e r (movimento entre frames de referência e imagens individuais) devem indicar a diferenças entre os mapas de movimentação.
4: Interpolar o campos de movimentação u_i^r e v_i^r de forma atingir a dimensão da imagem I_H.
5: Inicializar I_H, setando todos os valores para 0.
6: $\mathbf{C_{sum} := 0}$;
7: **de $\mathbf{k = 1}$ a $\mathbf{N}$ faça**
8: $\quad \mathbf{C_{sum} = C_{sum} + C^{(k)}}$
9: **fim para**
10: **de $\mathbf{k = 1}$ a $\mathbf{N}$ faça**
11: $\quad \mathbf{A^{(k)} = \frac{1}{C^{(k)} - \frac{1}{s} C_{sum}}}$
12: **fim para**
13: de $t = 1$ a T faça
14: $\quad sum := 0$;
15: $\quad$ de $k = 1$ a N faça
16: $\quad\quad b := W_k I_H^t$ (deformação "backward");
17: $\quad\quad c := h(x, y) * b$ (convolução com o kernel Gaussiano);
18: $\quad\quad c := D_c$ (sub-amostragem para a dimensão de I_L);
19: $\quad\quad d := (I_L^k - c)$;
20: $\quad\quad b := D^T d$ (super-amostragem sem interpolação);
21: $\quad\quad c := h(x, y) * b$
22: $\quad\quad d := W_k^T c$ (deformação "forward");
23: $\quad\quad \mathbf{sum := sum + A^{(k)} d}$ **(consideração do mapa de cores);**
24: $\quad$ fim para
25: $\quad I_H^{t+1} = I_H^t + \tau(sum - \lambda \triangle I_H^t)$;
26: fim para

Reiterando, com essas modificações, a contribuição de determinado frame $I_L^{(k)}$ para o resultado final será menor nos casos em que o C_k correspondente apresentar um padrão de cores diferente quando comparado à cor média dos demais.

5
Implementação

A firura 5.1 descreve, de maneira sucinta, as etapas do algoritmo 4.2 após a aquisição das imagens e dos mapas de profundidade pelo Kinect.

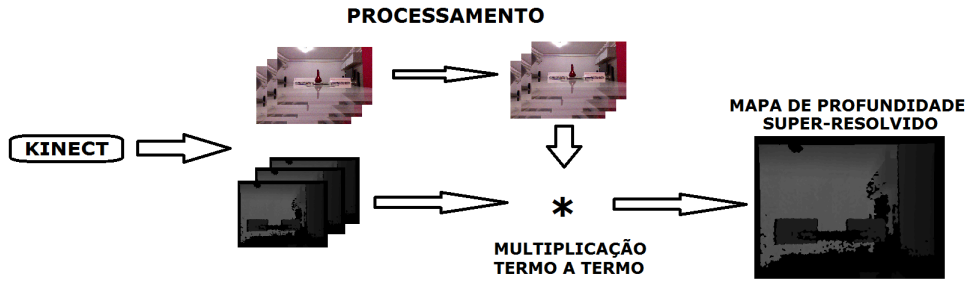


Figura 5.1: Etapas após aquisição.

Os detalhes da implementação de cada um destes passos são descritos a seguir.

5.1
Integração OpenCV x OpenNI: visão geral

Conforme já exposto, definiu-se que a biblioteca OpenCV [27] seria utilizada para o desenvolvimento das funcionalidades propostas. Identificou-se que havia uma integração já implementada nesta biblioteca totalmente dedicada à OpenNI. Tal interface permite que a aplicação seja integrada a sensores de profundidade compatíveis com a OpenNI - Kinect, XtionPRO,... - acessando-se os mesmos através da classe *VideoCapture* amplamente utilizada pela comunidade OpenCV.

Os dados do sensor de profundidade e os dados do sensor RGB disponibilizados pela integração estão descritos nas seções C.1 e C.2.

Após se indicar à classe *VideoCapture* através da flag CV_CAP_OPENNI, podemos obter os diversos tipos de informação para cada frame captado pelo respectivo sensor. Através do método *grab*(), "congelamos" a leitura da captação em determinado frame. A seguir, basta indicar o tipo de dado a ser capturado, conforme mostrado no trecho de código A.1, no apêndice A.

Para o mesmo instante são capturados o mapa de profundidade e a imagem colorida captada pelo sensor RGB. Como dois tipos de sensores são

suportados, é necessário que haja como alternar entre um ou outro no momento em que setam-se as respectivas propriedades. Isso pode ser feito através das flags descritas na seção C.3.

A classe $VideoCapture$ disponibiliza métodos *get* e *set* para manipulação das propriedades mencionadas conforme exemplificado no trecho A.2, no apêndice A. Nesse algoritmo o gerador de imagens é configurado para o formato VGA a uma frequência de 30Hz e logo depois a taxa de frames por segundo, do mesmo sensor, é consultada.

Quando nenhuma flag é indicada, se assume CV_CAP_OPENNI_DEPTH _GENERATOR como padrão. Tais flags devem ser combinadas àquelas respectivas às propriedades desejadas. As opções disponíveis para o gerador de imagens e para o sensor de profundiade estão representados nas seções C.4 e C.5.

5.2 OpenCV: Super-resolução

A classe dedicada à super-resolução disponibilizada pela OpenCV [27] se baseia na teoria descrita na seção 3.3. A base da mesma se propõe a fornecer uma interface comum aos diversos algoritmos de super-resolução providos pela biblioteca.

Através do método $setInput$ fornecemos a sequência de frames a ser super-resolvida. Através do método $nextFrame$ disparamos o processamento do próximo frame da nossa fonte. Os dados não mais necessários são removidos através do método *collectGarbage*.

O algoritmo descrito na seção 3.3 é carregado e executado na CPU através do método $createSuperResolution\_BTVL1$ e, na GPU, através do método $createSuperResolution\_BTVL1\_GPU$.

Antes do algoritmo ser executado, os membros descritos na seção C.6 devem ser inicializados.

No tocante a estimativa de movimento, os seguintes algoritmos podem ser acessados através da interface mencionada:

- **tvl1** - algoritmo descrito na seção 3.3;
- **farneback** - algoritmo descrito por Farnebäck [28];
- **pyrlk** - algoritmo descrito por Lucas e Kanade [21], com melhorias brevemente propostas por Nagy e Vámossy, em [29];
- **simple** - algoritmo descrito por Tao et al. [30];
- **brox** - algoritmo descrito por Brox et al. [31].

5.3 Adaptando as imagens coloridas fornecidas pelo Kinect

Os mapas de profundidade disponibilizados pelo Kinect possuem resolução de 640 x 480. Para esta dissertação, setou-se o fator de aumento da resolução β com o valor 2. Ou seja, o mapa de profundidade gerado teria uma resolução de 1280 x 960. Utilizando-se a flag CV_CAP_OPENNI_SXGA_15HZ, apresentada na seção 5.1, configurou-se o dispositivo para gerar imagens RGB a uma resolução de 1280 x 1024.

A matriz $A^{(k)}$, presente na equação (4-1), é multiplicada termo-a-termo por uma parcial do frame super-resolvido. Sendo assim, teríamos uma incompatibilidade de dimensões entre a imagem RGB captada - 1280 x 1024 -, que serve de base para a geração de $A^{(k)}$, e o mapa de profundiade parcial super-resolvido - 1280 x 960. Para sanar tal problema, a cada captação das imagens RGB, o seguinte procedimento foi adotado:

1. Aplicação de um filtro de média objetivando a redução de ruídos;
2. Remoção das 64 últimas linhas da imagem filtrada.

Tal padrão de adaptação é sugerido no fórum de desenvolvimento Kinect da Microsoft. Testes foram realizados no intuito de validar o mesmo. Um exemplo de uma imagem RGB original e o resultado final são apresentados nas figuras 5.2(a) e 5.2(b).

Após tal processamento, cada canal da imagem RGB é multiplicado por 33%. Se gera uma nova matriz composta pela soma dos três resultados - representada por $C^{(k)}$ no algoritmo 4.2. Essa soma gera um único valor de intensidade para cada pixel e, graças à multiplicação anterior, não ocorrem overflows.

5.4 OpenCV: Modificações

Algumas modificações se fizeram necessárias para que a interface de super-resolução disponibilizada pela OpenCV [27] atendesse aos propósitos da pesquisa em questão. Além das alterações geradas pelas modificações teóricas, conforme descrito no capítulo 4, o sistema demandou certas mudanças de natureza prática.

5.2(a): Imagem original.

5.2(b): Imagem processada.

Figura 5.2: Processamento do mapa RGB.

5.4.1 A classe FrameSource

Essa classe permite a leitura de sequências de frames originárias de arquivos de vídeo e sensores de captura. Dado que o foco está em sensores compatíveis com a OpenNI, não se alteraram os métodos referentes à leitura de vídeos. No tocante ao acesso à informação, nenhuma alteração foi necessária pois a integração descrita na seção 5.1 permite que se lide com o Kinect como se fosse uma câmera comum ou qualquer outro dispositivo para captura de imagens. Contudo, tal classe não comportava a aquisição de dois frames simultâneos de dois sensores diferentes do mesmo dispositivo - no presente caso, o gerador de imagens e o sensor de profundidade.

No intuito de suprir essa necessidade, modificou-se a definição inicial da classe $FrameSource$, através da inclusão de um novo método, o $nextOpenNIFrame$, conforme mostrado no trecho A.3, no apêndice A.

É importante destacar que a declaração do novo método foi codificada de forma a permitir que as classes que implementassem ou herdassem a FrameSource não fossem obrigadas a ter tal método presente. Com isso, todas as classes que se enquadram nessa característica compilaram sem problemas e por outro lado permitiu-se a inclusão do método $nextOpenNIFrame$ - trecho apresentado em A.4 (apêndice A) - onde era desejado.

A partir dessa alteração, se tornou possível "congelar" a captação do Kinect e obter dois frames concomitantes: um associado ao modo $deviceMode$ e outro ao modo $deviceMode1$.

5.4.2 As classes $BTVL1$ e $BTVL1\_base$ dedicadas à CPU

As modificações necessárias à incorporação da consideração de cores às classes dedicadas à CPU estão detalhadas na seção A.2 do apêndice A.

5.4.3 As classes $BTVL1$ e $BTVL1\_base$ dedicadas à GPU

As modificações necessárias à incorporação da consideração de cores às classes dedicadas à GPU foram praticamente as mesmas descritas em A.2 (apêndice A) destinadas à CPU. Contudo, para o processamento em paralelo na GPU, é necessário que se garanta que os cálculos de cada etapa não se iniciem antes do término do cálculo dos valores das etapas anteriores dos quais depende.

Para essa sincronização, se utilizam vetores que armazenam instâncias da classe $Stream$. Esta encapsula uma fila de chamadas assíncronas.

Outra diferença em relação à CPU foi a utilização da classe $GpuMat$ em substituição à classe Mat. Além disso, todas as operações termo a termo, necessárias à execução do algoritmo foram substituídas por implementações dedicadas à GPU disponibilizadas pela biblioteca $OpenCV$.

Os trechos de código que se diferenciam daqueles apresentados para implementação em CPU são apresentados em A.3 (apêndice A).

6
Testes e Resultados

Após toda teoria e implementação descritas nos capítulos anteriores, eram necessários testes que comprovassem alguma melhoria provocada pela inclusão de cores ou até mesmo indicasse que tal alteração no algoritmo de super-resolução fosse ineficaz. Além disso, precisava-se mensurar o quanto a mudança impactava o desempenho do algoritmo.

Basicamente, realizaram-se três testes:

- Teste com objetos inanimados presentes em poucos frames;
- Teste com seres humanos;
- Comparativo de desempenho entre o algoritmo original e o novo - alternado entre CPU e GPU.

Os procedimentos e resultados de cada teste serão descritos nas seções a seguir.

6.1
Verificando a contribuição da inclusão dos frames RGB

Objetivando a constatação de alguma melhoria advinda da inclusão da consideração de cores no algoritmo proposto por Mitzel et al. [3], arquitetou-se o teste representado na figura 6.1.

Uma plataforma circular giratória foi posicionada a 2 metros de distância do Kinect conforme mostrado na figura 6.2(a). A mesma foi demarcada de forma a se dividir em 8 fatias idênticas - destacadas por círculos vermelhos na figura 6.3. A cada frame gerado, a plataforma é rotacionada a 45 °. A marcação, destacada em lilás, serve de referência para o posicionamento da plataforma a cada rotação. Além disso, duas marcações - destacadas por círculos verdes - foram feitas na superfície da plataforma giratória para indicar a posição, em que os dois objetos apresentados na figura 6.2(b), seriam posicionados no decorrer dos testes. A sequência dos testes é a seguinte:

1. Com a plataforma giratória vazia, captam-se os mapas de profunidade da cena apresentada na figura 6.4 até que seja gerado o primeiro frame super-resolvido;

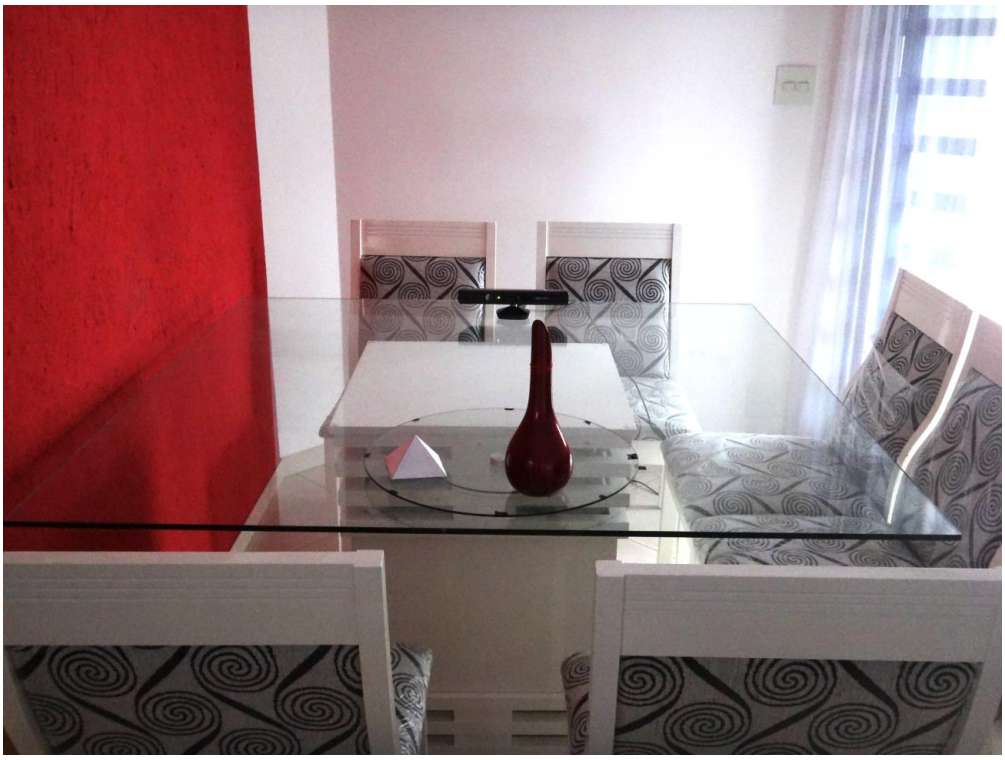


Figura 6.1: Teste para verificação dos resultados consequentes à inserção da consideração das imagens coloridas.

6.2(a): Visão detalhada do ambiente de testes.

6.2(b): Objetos utilizados para "interferir" na captação.

Figura 6.2: Alguns dos elementos utilizados.

2. Os objetos apresentados na figura 6.2(b) são posicionados nas respectivas marcações conforme indicado pelos destaques em verde na figura 6.3;

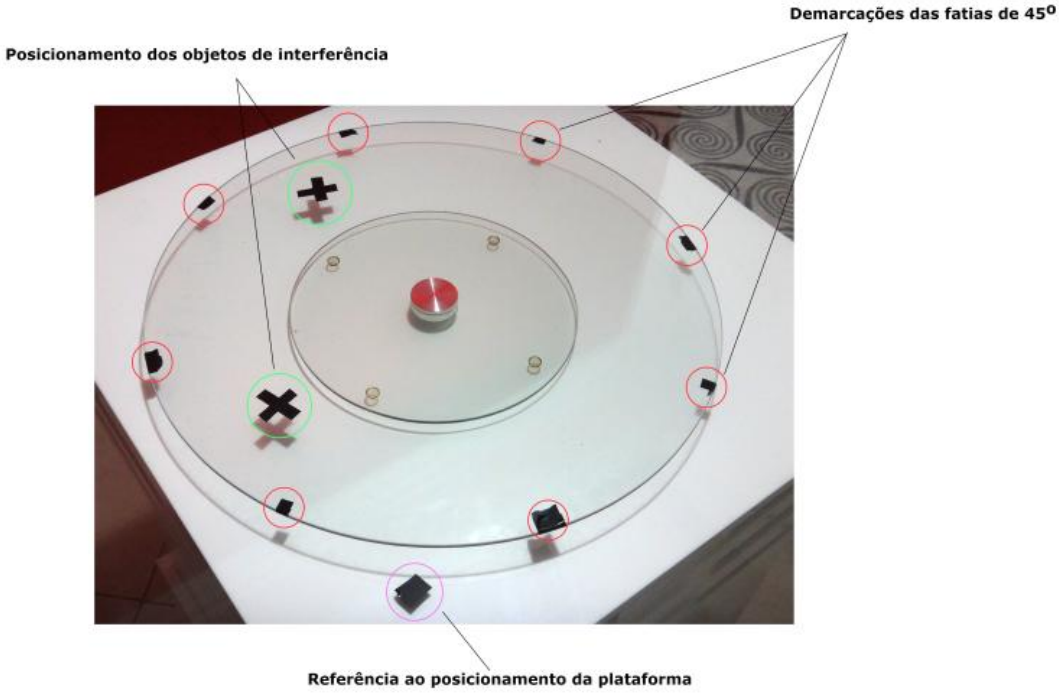


Figura 6.3: Plataforma giratória

Figura 6.4: Cena inicial capturada durante os testes.

3. A cada frame super-resolvido gerado, a plataforma é rotacionada de 45 °.

4. A etapa anterior é repetida por 25 vezes.

Figura 6.5: Cena com objetos de interferência capturada durante os testes.

A figura 6.6 ilustra a sequência descrita. Tal procedimento foi executado uma vez para o algoritmo 3.1 - implementação original sem consideração de cores - e outra vez para o algoritmo 4.2 - implementação proposta que incorpora as imagens RGB ao processo. Os objetos acrescentados após a geração do

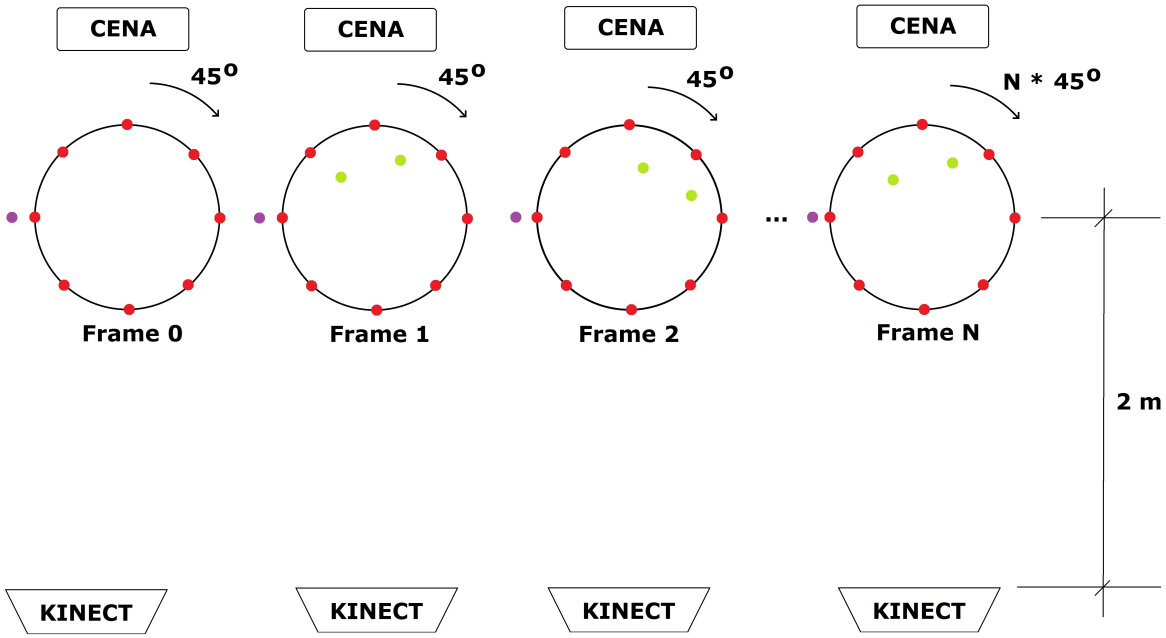
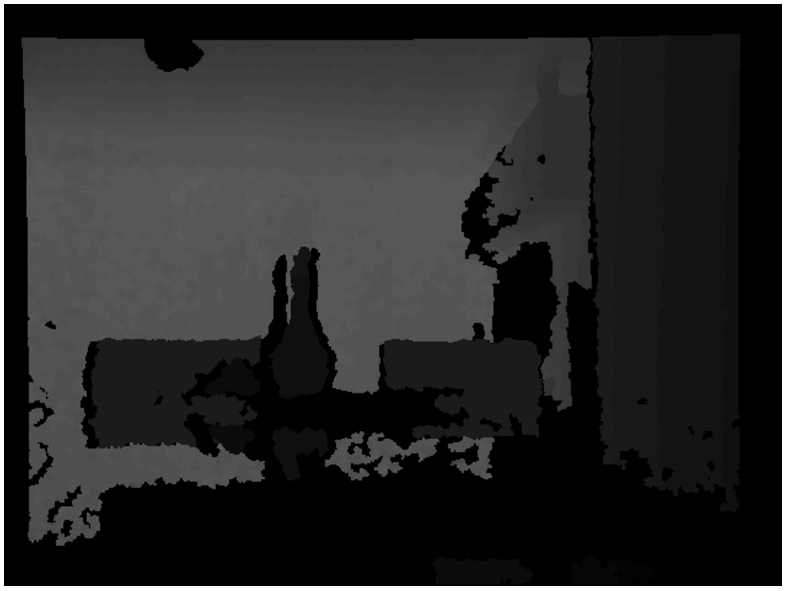


Figura 6.6: Esquemático da sequência de testes.

primeiro frame super-resolvido geram discrepâncias entre as imagens RGB captadas. Devido à rotação da plataforma, essas diferenças se deslocam entre os pixels a cada frame. Sendo assim, a inclusão da consideração de cores deveria impedir que as profundidades dos objetos que geram essas discrepâncias influenciassem os frames super-resolvidos.

A primeira cena super-resolvida gerada pelo algoritmo proposto por 3.1 é apresentada na figura 6.7(a). Já na décima segunda imagem de maior resolução gerada apresentou os objetos que não deveriam aparecer, conforme mostrado na figura 6.7(b).

6.7(a): Primeiro frame super-resolvido gerado.



6.7(b): Décimo-segundo frame super-resolvido gerado.

Figura 6.7: Resultados apresentados pelo algoritmo proposto por Mitzel et al. [3].

A execução do algoritmo 4.2 gerou como primeiro frame super-resolvido, a imagem mostrada na figura 6.8(a). Após as 25 repetições da etapa 3, nenhum sinal dos objetos de interferência surgiu. Tal resultado é ilustrado através da figura 6.8(b) que mostra o frame de número 24 gerado pelo novo algoritmo.

6.8(a): Primeiro frame super-resolvido gerado.

6.8(b): Vigésimo-quarto frame super-resolvido gerado.

Figura 6.8: Resultados apresentados pelo algoritmo proposto no presente trabalho.

Sendo assim, conclui-se que a consideração de cores impede que mudanças bruscas na profundidade dos elementos captados influenciem as distâncias apresentadas no mapa super-resolvido.

6.2 Testes com seres humanos

Como, no presente trabalho, não se executará a reconstrução da malha composta pelos mapas de profundidade super-resolvidos, seria impossível detectar alterações ocasionadas por pequenas dobras das roupas durante o processo de captação conforme realizado por Cui et al. [2], devido à pequena escala de tais imperfeições. Contudo, precisava-se comprovar que o algoritmo proposto mitigava a interferência causada por modificações bruscas da cena captada, movimentação das roupas de um ser humano por exemplo, durante o processo de geração das imagens super-resolvidas.

Para tal avaliação, um ser humano se posicionou no centro da cena captada, conforme figura 6.9(a). Tal ambiente foi captado de forma contínua. Em média, a cada 10s, a captação era congelada e a pessoa vestia um paletó conforme figura 6.9(b). Após a captação de três novos frames, o processo era congelado e a pessoa retirava tal paletó. Tal procedimento se repetiu por cerca de 120s.

Se o algoritmo proposto fosse capaz de eliminar artefatos causados por mudanças pontuais entre os frames captados, conforme o resultado alcançado por Cui et al. [2], quando os mapas super-resolvidos fossem gerados considerando as imagens RGB, o paletó não deveria ser detectado.

6.9(a): Cena principal.

6.9(b): Cena com interferência.

Figura 6.9: Teste proposto para captação de um ser humano.

Os resultados da execução do algoritmo descrito por Mitzel et al. [3], sofreram interferência dos frames que continham o paletó, conforme mostrado nas figuras 6.10(a) e 6.10(b).

6.10(a): Primeiro frame super-resolvido gerado.

6.10(b): Vigésimo frame super-resolvido gerado.

Figura 6.10: Resultados apresentados pelo algoritmo proposto por Mitzel et al. [3], durante a captação de uma pessoa.

Já ao utilizar o algoritmo proposto no presente trabalho, o paletó não surgiu em nenhum dos frames super-resolvidos. Tal resultado é ilustrado pelas figuras 6.11(a) e 6.11(b).

Isto é, comprovamos que, dentro de um sistema de digitalização 3D dedicado ao corpo humano, a utilização das imagens RGB evitaria que movimentações das roupas do corpo captado, por exemplo, prejudicassem os níveis

6.11(a): Primeiro frame super-resolvido gerado.

6.11(b): Vigésimo-quarto frame super-resolvido gerado.

Figura 6.11: Resultados apresentados pelo algoritmo proposto no presente trabalho, durante a captacção de uma pessoa.

de profundiade disponibilizados para reconstrução da malha final. Tais testes indicam que, com a consideração das cores, os mapas de profundidade super-resolvidos estarão menos sujeitos a interferências durante o processo de captação.

6.3 Tempo de execução: CPU x GPU

Para os testes utilizou-se um computador com as seguintes configurações:

- Processador Intel Core i7 de 1.73GHz

- 6 GB de memória RAM
- Placa de vídeo GEFORCE GT 425M

Foram computados os tempos de execução relativos aos 10 primeiros frames para quatro combinações distintas:

- Utilizando CPU e não considerando as cores;
- Utilizando GPU e não considerando as cores;
- Utilizando CPU e considerando as cores;
- Utilizando GPU e considerando as cores.

A configuração utilizada para a execução do algoritmo foi:

- **int scale** - 2;
- **int iterations** - 10;
- **double tau** - 1.3;
- **double lambda** - 0.03;
- **double alpha** - 0.7;
- **int btvKernelSize** - 7;
- **int blurKernelSize** - 5;
- **double blurSigma** - 0.0;
- **int temporalAreaRadius** - 4;
- **Ptr<DenseOpticalFlowExt>** - TVL1.

A resolução dos mapas de profundidade captados foi de 640 x 480 e a das imagens coloridas foi de 1280 x 1024. Dado que o parâmetro *scale* utilizado foi igual a 2, a resolução da saída super-resolvida foi de 1280 x 960.

Em todas as combinações, o tempo consumido para a geração do *frame 0* foi maior que aquele dos demais. Isso já era previsto pois, para a primeira saída, a estimativa de movimento é processada para $2 * temporalAreaRadius + 1$ - nesse caso 9 - vezes, e, utilizando tais valores, $temporalAreaRadius + 1$ - nesse caso 5 - frames passam pelas demais etapas, linhas 3 a 26, do algoritmo 4.2. Por fim, mais um frame é captado e processado, gerando-se o *frame 0* com base nos cálculos anteriores. A partir daí, para cada novo frame, apenas a nova captação é processada, descartando-se os resultados da última armazenada.

Os testes também mostraram que o processamento na GPU é cerca de 7 vezes mais rápido que na CPU.

Tabela 6.1: Tempos de execução CPU: impato da inclusão de cores.

Frame	Tempo sem cores (s)	Tempo com cores	Aumento
Frame 0	178.75	195.57	9%
Frame 1	24.75	28.66	16%
Frame 2	24.42	28.80	18%
Frame 3	24.35	27.92	15%
Frame 4	23.83	28.29	19%
Frame 5	24.09	28.27	17%
Frame 6	24.06	28.19	17%
Frame 7	24.23	28.09	16%
Frame 8	24.03	30.09	25%
Frame 9	24.16	28.13	16%
Média - frames 1 a 9	-	-	18%

Tabela 6.2: Tempos de execução GPU: impato da inclusão de cores.

Frame	Tempo sem cores (s)	Tempo com cores	Aumento
Frame 0	19.49	27.58	40%
Frame 1	2.21	3.08	40%
Frame 2	2.26	3.16	40%
Frame 3	2.3	3.12	36%
Frame 4	2.22	3.17	43%
Frame 5	2.2	3.09	40%
Frame 6	2.11	3.14	49%
Frame 7	2.16	3.04	41%
Frame 8	2.17	2.99	38%
Frame 9	2.11	3.10	47%
Média - frames 1 a 9	-	-	41%

Analisando o impacto da inclusão da consideração de cores, para a CPU, o tempo de execução aumentou em aproximadamente 9% - para o *frame 0* - e 18% - para os demais, conforme tabela 6.1. Já para GPU, o tempo de execução aumentou em aproximadamente 42% - para o *frame 0* - e 41% - para os demais, conforme tabela 6.2.

7
Conclusão e trabalhos futuros

Conforme exposto no capítulo 1, o presente trabalho objetivava a confecção de um sistema de super-resolução que fosse capaz de aumentar a resolução dos mapas de profundidade disponibilizados por sensores de baixo custo. Tal sistema serviria de base para digitalização 3D de corpos humanos. Para mitigar problemas gerados por possíveis movimentações dos corpos captados, no capítulo 4, propôs-se a utilização de imagens coloridas da mesma cena que fornecia os mapas de profundidade.

A contribuição teórica do presente trabalho se deu ao incoporar a proposta de Cui et al. [2], ao algoritmo de Mitzel et al. [3], que se traduz na equação 4-1. Dado que o código da solução de Cui et al. [2] não está disponível, a junção dos dois conceitos, aqui proposta, permitiu a utilização da implementação do algoritmo de Mitzel et al. [3] disponibilizada na biblioteca OpenCV como base para o desenvolvimento da estratégia descrita no capítulo 4

As classes, dedicadas à super-resolução, disponiblizadas pela biblioteca OpenCV demandaram certas modificações para contemplar dois aspectos:

- aquisição concomitante do mapa de profundidade e da imagem colorida;
- incorporação da consideração das imagens coloridas no algoritmo de super-resolução original.

Tais modificações, descritas no capítulo 5, culminaram nos resultados expostos no capítulo 6. Os testes descritos na seções 6.1 e 6.2 comprovaram que a utilização das imagens coloridas melhorou os mapas de profundidade super-resolvidos no sentido de evitar interferências provocadas por mudanças repentinas na cena captada.

Os testes da seção 6.3 mostraram que a execução em GPU supera em sete vezes a equivalente na CPU. Tal resultado, justifica que os desenvolvimentos futuros sejam focados em algoritmos dedicados à GPU. Um aumento de aproximadamente 40% provocado pela inclusão de cores nos leva à primeira oportunidade para trabalhos futuros. Seria interessante explorar duas frentes:

- otimização da implementação das modificações propostas no capítulo 4 e descritas no capítulo 5;

- proposição de uma nova abordagem para a consideração as cores que possa ser computada mais rapidamente.

O desenvolvimento dos módulos de registro global e não global descritos na seção 2.2 daria continuidade ao sistema de digitalização 3D de baixo custo e permitira testes volumétricos capazes de fornecer novos dados acerca da eficiência da consideração de cores. Mais que isso, tal implementação permitiria uma comparação aos resultados de reconstrução alcançados por Cui et al. [2].

Além disso, futuros testes poderiam analisar dois aspectos:

- Qual o comportamento do algoritmo proposto em cenas onde o objeto de interferência tivesse um contraste pequeno em relação ao restante da cena captada?
- Se utilizarmos a imagem de baixa resolução como insumo para os registros rígido e não-rígido qual o impacto na malha 3D reconstruída?

Referências Bibliográficas

[1] Jin Tong, Jin Zhou, Ligang Liu, Zhigeng Pan, and Hao Yan. Scanning 3d full human bodies using kinects. Technical report, Hohai University, 2012.

[2] Yan Cui, Will Chang, Tobias Nöll, and Didier Stricker. Kinectavatar: Fully automatic body capture using a single kinect. *AACV, Workshop on Color Depth Fusion in Computer Vision*, 2012.

[3] Dennis Mitzel, Thomas Pock nad Thomas Schoenemann, and Daniel Cremers. Video super resolution using duality based tv-l1 optical flow. *DAGM*, 2009.

[4] http://www.leica-geosystems.com/en/index.htm.

[5] http://www.lfm-software.com/lfm-products.

[6] D. L. Cardon, W. S. Fife, J. K. Archibald, and D. J. Lee. Fast 3d reconstruction for small autonomous robots. In *Proceedings of the 31st Annual Conference of IEEE Industrial Electronics Society (IECON '05)*, 2005.

[7] http://www.bodymetrics.com/.

[8] Jiawen Chen, Dennis Bautembach, and Shahram Izadi. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph.*, 32(4):113:1–113:16, July 2013.

[9] M. Elad and A. Feuer. Super-resolution reconstruction of image sequences. *IEEE Transactions on Pattern Analysis and Machine Ingelligence*, 21:817–834, 1999.

[10] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l1 optical flow. In *Pattern Recognition Proc. DAGM Heidelberg, Germany*, pages 214–223, 2007.

[11] www.openni.org.

[12] http://cmp.felk.cvut.cz/ svoboda/selfcal/index.html.

[13] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rossl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184, 2004.

[14] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing, SGP'06*, pages 61–70, Aire-la-Ville, Switzerland, 2006. Eurographics Association.

[15] Y. Cui, S. Schuon, D. Chan, S. Thrun, and C. Theobalt. 3d shape scanning with a time-offlight camera. In *IEEE Proc. CVPR*, 2010.

[16] W. Chang and M. Zwicker. Global registration of dynamic range scans for articulated model reconstruction. *ACM Trans. Graph. 30*, 2011.

[17] R.A Newcombe, D. Kim, and P. Kohli. Kinectfusion: Real-time dense surface mapping and tracking. *ISMAR*, 2011.

[18] S. Schuon, C. Theobalt, J. Davis, and S. Thrun. Lidarboost: Depth superresolution for tof 3d shape scanning. In *CPVR 2009*, 2009.

[19] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3D characters. *ACM Transactions on Graphics*, 26(3):72:1–72:8, jul 2007.

[20] Ilya Baran and Jovan Popović. Penalty function for automatic rigging and animation of 3d characters. *http://people.csail.mit.edu/ibaran/papers/2007-SIGGRAPH-Pinocchio-Penalty.pdf*, 2007.

[21] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *In International Joint Conference on Artificial Intelligence*, 3, 1981.

[22] Yan Cui, Sebastian Schuon, Sebastian Thrun, Didler Stricker, and Christian Theobalt. Algorithms for 3d shape scanning with a depth camera. *IEEE Transactions on Patt*, 2012.

[23] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[24] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[25] A. Chambolle. An algorithm for total variation minimization and aapplication. *Journal of Mathematical Imaging and Vision*, 20(1-2):89–97, 2004.

[26] A. Marquina and S. J. Osher. Image super-resolution by tv-regularization and bregman iteration. *Journal of Scientific Computing*, 3:37, 2008.

[27] www.opencv.org.

[28] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. *SCIA13: Gothenburg, Sweden*, pages 363–370, 2003.

[29] A. Nagy and Z. Vámossy. Opencv c# wrapper based video enhancement using different optical flow methods in the super-resolution. In *6th International Symposium on Intelligent Systems and Informatics*, pages 1–6, 2008.

[30] Michael Tao, Jiamin Bai, Pushmeet Kolli, and Sylvain Paris. Simpleflow - a non-iterative, sublinear optical flow algorithm. *EUROGRAPHICS*, 31, 2012.

[31] Thomas Brox, André Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *8th European Conference on Computer Vision*, 2004.

A
Códigos em C++

A.1
Trechos de código referenciados na dissertação

Código A.1 Exemplo de captura simultânea dos mapas de profundidade e das imagens RGB

```
VideoCapture capture(CV_CAP_OPENNI);

for (;;) {
    Mat depthMap;
    Mat rgbImage

    capture.grab();

    capture.retrieve( depthMap, CV_CAP_OPENNI_DEPTH_MAP );
    capture.retrieve( bgrImage, CV_CAP_OPENNI_BGR_IMAGE );

    if( waitKey( 30 ) >= 0 )
        break;
}
```

Código A.2 Métodos *get* e *set* para manipulação de propriedades

```
VideoCapture capture( CV_CAP_OPENNI );

capture.set( CV_CAP_OPENNI_IMAGE_GENERATOR_OUTPUT_MODE,↩
    CV_CAP_OPENNI_VGA_30HZ );

cout << ``FPS '' << capture.get( ↩
    CV_CAP_OPENNI_IMAGE_GENERATOR + CV_CAP_PROP_FPS ) << ↩
    endl;
```

Código A.3 Modificação da declaração da classe $FrameSource$

```
class CV_EXPORTS FrameSource
{
        public:
        virtual ~FrameSource();

        virtual void nextFrame(OutputArray frame) = 0;
        virtual void reset() = 0;
        virtual void nextOpenNIFrame (OutputArray frame, ↩
           OutputArray frame1, int deviceMode, int ↩
           deviceMode1) {};
};
```

Código A.4 Método $nextOpenNIFrame$

```
void CaptureFrameSource::nextOpenNIFrame (OutputArray ↩
   _frame, OutputArray _frame1, int deviceMode, int ↩
   deviceMode1)
{
        vc_.grab();
        vc_.retrieve(frame_,deviceMode);
        arrCopy(frame_, _frame);
        vc_.retrieve(frame1_,deviceMode1);
        arrCopy(frame1_, _frame1);
}
```

A.2
As classes $BTVL1$ e $BTVL1\_base$ dedicadas à CPU

O primeiro passo para a modificação da classe BTVL1 seria indicar para mesma quando deveria rodar o algoritmo de super-resolução considerando o mapa de cores e quando deveria executar o método original. Para evitar grandes modificações, decidiu-se utilizar a mesma abordagem empregada para a alteração dos parâmetros de configuração do algoritmo original, conforme mostrado no trecho A.5.

Código A.5 Inicialização do atributo $withColorWeights$

```
        superRes->set("withColorWeights", true);
```

Onde $superRes$ é uma instância da classe $BTVL1$. O atributo $wihColorWeights$ é um booleano que funciona como um flag mudando o comportamento dos métodos responsáveis pelo processo de super-resolução. Caso tal valor não seja indicado, o valor padrão é falso, isto é, o algoritmo de super-resolução se comporta conforme indicado na seção 3.3. Para permitir tal comportamento, foi necessária a inclusão da chamada ao método de inicialização, herdado da classe $Algorithm$ da biblioteca OpenCV, conforme mostrado no trecho A.6.

Código A.6 Método de inicialização

```
CV_INIT_ALGORITHM(BTVL1, "SuperResolution.BTVL1",
        obj.info()->addParam(obj,
                             "colorWeights",
                             obj.withColorWeights_,
                             false,
                             0,
                             0,
                             "Consider or not color ↩
                                weights.");
                        ...
```

O próximo método a ser modificado é o $initImpl$. O vetor $colorFrames$ - que armazena os frames disponibilizados pelo gerador de imagens - é redimensionado, conforme mostrado em A.7.

Código A.7 Redimensionamento

```
if (withColorWeights_)
{
        colorFrames_.resize(cacheSize);
}
```

$cacheSize$ é uma unidade maior que o dobro do parâmetro $temporalAreaRadius$ de forma a comportar todas imagens utilizadas no processamento do algoritmo. O método $initImpl$ invoca dois métodos que demandam modificações: o $readNextFrame$ - captura os frames a serem processados e faz os devidos cálculos relacionados à estimativa de movimento - e o $processFrame$ - processa as imagens previamente armazenadas. No primeiro, adicionamos a

chamada ao método $nextOpenNIFrame$, descrito na seção 5.4.1, conforme trecho A.8.

Código A.8 Chamando o método $nextOpenNIFrame$

```
if (withColorWeights_) {
        frameSource->nextOpenNIFrame(curFrame_,↩
            curColorFrame_,CV_CAP_OPENNI_DEPTH_MAP,↩
            CV_CAP_OPENNI_BGR_IMAGE);
} else {
        frameSource->nextFrame(curFrame_);
}
```

Além disso, precisamos garantir que, caso haja algum problema na captação da imagem, uma imagem "vazia" não seja utilizada no processamento. Isso é verificado e encaminhado pela condição A.9.

Código A.9 Evitando a utilização de imagens corrompidas

```
if (curColorFrame_.empty() && withColorWeights_) {
        return;
}
```

Uma vez captada, precisamos converter o tipo de dado de cada pixel da imagem para o CV_32F utilizado em todos os cálculos do algoritmo.

Código A.10 Conversão

```
if (withColorWeights_)  {
        curColorFrame_.convertTo(at(storePos_, ↩
            colorFrames_), CV_32F);
}
```

Onde $storePos$ contém a posição da matriz $colorFrames$ em que a imagem corrente será armazenada. Em relação ao método $processFrame$ adicionou-se o redimensionamento do vetor $srcColorFrames\_$ - armazena as imagens a serem utilizadas no processamento corrente - conforme trecho A.11.

Código A.11 Redimensionamento do vetor $srcColorFrames$

```
if (withColorWeights_) {
        srcColorFrames_.resize(count);
}
```

As imagens captadas foram copiadas para o vetor $srcColorFrames\_$, conforme trecho A.12.

Código A.12 Populando o vetor $srcColorFrames$

```
if (withColorWeights_) {
        srcColorFrames_[k] = at (i, colorFrames_);
}
```

Onde k itera sobre o intervalo de imagens a serem consideradas no processamento e i corresponde ao frame captado correspondente. Além disso, incluiu-se a chamada ao método $process$ modificado de forma a considerar os mapas de cores captados pelo Kinect, conforme destacado no trecho A.13.

Código A.13 Chamada ao método $process$

```
if (withColorWeights_) {
        process(srcFrames_, srcColorFrames_, at(idx, ↩
           outputs_), srcForwardMotions_, ↩
           srcBackwardMotions_, baseIdx);
} else {
        process(srcFrames_, at(idx, outputs_), ↩
           srcForwardMotions_, srcBackwardMotions_, ↩
           baseIdx);
}
```

Após todos os preparativos explicitados anteriormente, o método $process$, o cerne da classe $BTVL1\_base$, teve seus parâmetros e corpo alterados. Como entrada para os mapas de cores, adicionou-se o vetor $\_srcColorFrames$ como parâmetro. As primeiras modificações no corpo do método foram o redimensionamento e a inicialização da matriz $colorWeights\_$ que corresponde à variável $A^{(k)}$ utilizada no algoritmo proposto na seção 4, conforme trecho A.14.

Código A.14 Redimensionamento e incialização da variável $colorWeights$

```
colorWeights_.create(srcColor[0].rows,srcColor[0].cols,src↩
    [0].type());
colorWeights_.setTo(Scalar::all(0));
```

Incluiu-se o cálculo e o armazenamento da variável s que corresponde ao valor de $\frac{1}{s}$ usado na geração da média entre os mapas de cores.

Código A.15 Cálculo da variável s

```
double s = 1 / src.size();
```

As matrizes $colorMap$, que armazena o somatório do mapa de cores (C_{sum}), e cK foram criadas.

Código A.16 Criação das variáveis $colorMap$ e cK

```
Mat colorMap(srcColor[0].rows,srcColor[0].cols,src[0].type↩
    ());
Mat cK(srcColor[0].rows,srcColor[0].cols,src[0].type());
```

O vetor $layers$ - armazenamento dos canais RGB - foi criado.

Código A.17 Vetor para o armazenamento dos canais RGB.

```
vector<Mat> layers;
```

Através da iteração A.18, calculamos a matriz $colorMap$. Os três canais de cada mapa de cor são condensados em um único valor, cada qual contribuindo com aproximadamente 33% para a soma final.

Código A.18 Iteração para o cálculo da variável $colorMap$

```
for (size_t k = 0; k < src.size(); ++k)
{
        split(srcColor[k],layers);

        add(layers[0],layers[1],colorMap,noArray(),-1);
```

```
        add(layers[2], colorMap, colorMap, noArray(), -1);

        divide(3.0f, colorMap, colorMap, -1);
}
```

Em seguida, uma nova sequência de iterações gera o denominador de $A^{(k)}$.

Código A.19 Iteração para o cálculo do denominador de $A^{(k)}$

```
for (size_t k = 0; k < src.size(); ++k)
{
        cK.setTo(Scalar::all(0));

          add(layers[0], layers[1], cK, noArray(), -1);
        add(layers[2], cK, cK, noArray(), -1);
        divide(3.0f, cK, cK, -1);

        addWeighted(cK, 1.0, colorMap, -s, 0.0, colorWeights_ ↩
            , -1);
}
```

A útlima alteração demandada pelo método $process$ é a inserção da multiplicação termo a termo de $A^{(k)}$ - inverso da variável $colorWeights\_$ - e o resultado intermediário do algoritmo proposto conforme mostrado no trecho A.20.

Código A.20 Incorporando o termo $A^{(k)}$

```
for (int i = 0; i < iterations_; ++i)
{
        diffTerm_.setTo(Scalar::all(0));

        for (size_t k = 0; k < src.size(); ++k)
        {
                remap(highRes_, a_, backwardMaps_[k], ↩
                    noArray(), INTER_NEAREST);
                filter_->apply(a_, b_);
                resize(b_, c_, lowResSize, 0, 0, ↩
                    INTER_NEAREST);

                diffSign(src[k], c_, c_);
```

```
            upscale(c_, a_, scale_);
            filter_->apply(a_, b_);
            remap(b_, a_, forwardMaps_[k], noArray(), ↩
                INTER_NEAREST);

            divide(a_,colorWeights_,a_,1.0,-1);

            add(diffTerm_, a_, diffTerm_);
        }

        if (lambda_ > 0)
        {
            calcBtvRegularization(highRes_, regTerm_, ↩
                btvKernelSize_, btvWeights_);
            addWeighted(diffTerm_, 1.0, regTerm_, -↩
                lambda_, 0.0, diffTerm_);
        }

        addWeighted(highRes_, 1.0, diffTerm_, tau_, 0.0, ↩
            highRes_);
}
```

O método $collectGarbage$ da classe $BTVL1$ também foi alterado de forma a garantir que todos os dados dispensáveis sejam removidos da memória.

Código A.21 Método $collectGarbage$ da classe $BTVL1$.

```
void BTVL1::collectGarbage()
{
    curFrame_.release();
    prevFrame_.release();

    frames_.clear();
    forwardMotions_.clear();
    backwardMotions_.clear();
    outputs_.clear();

    srcFrames_.clear();
    srcColorFrames_.clear();
    srcForwardMotions_.clear();
```

```
        srcBackwardMotions_.clear();
        finalOutput_.release();

        colorFrames_.clear();
        srcColorFrames_.clear();

        SuperResolution::collectGarbage();
        BTVL1_Base::collectGarbage();
}
```

O mesmo foi feito para o método $collectGarbage$ da classe $BTVL1\_base$.

Código A.22 Método $collectGarbage$ da classe $BTVL1\_base$.

```
void BTVL1_Base::collectGarbage()
{
        filter_.release();

        lowResForwardMotions_.clear();
        lowResBackwardMotions_.clear();

        highResForwardMotions_.clear();
        highResBackwardMotions_.clear();

        forwardMaps_.clear();
        backwardMaps_.clear();

        highRes_.release();

        diffTerm_.release();
        regTerm_.release();
        a_.release();
        b_.release();
        c_.release();

        colorWeights_.release();
}
```

As variáveis $layers$, $colorMap$ e cK são liberadas a cada finalização do cálculo do denominador de $A^{(k)}$.

A.3
As classes $BTVL1$ e $BTVL1\_base$ dedicadas à GPU

Código A.23 Criação das variáveis $colorMap$ e cK

```
GpuMat colorMap(srcColor[0].rows, srcColor[0].cols,↩
    src[0].type());
GpuMat cK(srcColor[0].rows, srcColor[0].cols, src↩
    [0].type());
```

Código A.24 Vetor para o armazenamento dos canais RGB.

```
vector<GpuMat> layers;
```

Código A.25 Iteração para o cálculo da variável $colorMap$

```
for (size_t k = 0; k < src.size(); ++k)
{
        gpu::split(srcColor[k], layers);

        gpu::add(layers[0], layers[1], colorMap, GpuMat(), -1,↩
            colorStreams_[k]);
        gpu::add(layers[2], colorMap, colorMap, GpuMat(), -1,↩
            colorStreams_[k]);

        gpu::divide(colorMap, 3.0f, colorMap, 1.0f, -1,↩
            colorStreams_[k]);
}
```

Código A.26 Exemplo de trecho utilizado para sincronização.

```
for (size_t k = 0; k < src.size(); ++k)
{
        colorStreams_[k].waitForCompletion();
}
```

Código A.27 Iteração para o cálculo do denominador de $A^{(k)}$

```
for (size_t k = 0; k < src.size(); ++k)
{
        cK.setTo(Scalar::all(0));

        gpu::add(layers[0],layers[1],cK,GpuMat(),-1,↩
            colorStreams_[k]);
        gpu::add(layers[2],cK,cK,GpuMat(),-1,colorStreams_↩
            [k]);

        gpu::divide(cK,3.0f,cK,1.0f,-1,colorStreams_[k]);

        gpu::addWeighted(cK,1.0f,colorMap,-s,0.0,↩
            colorWeights_,-1,colorStreams_[k]);
}
```

Código A.28 Incorporando o termo $A^{(k)}$

```
for (int i = 0; i < iterations_; ++i)        {

        for (size_t k = 0; k < src.size(); ++k)
        {
                gpu::remap(highRes_, a_[k], backwardMaps_[↩
                    k].first, backwardMaps_[k].second, ↩
                    INTER_NEAREST, BORDER_REPLICATE, Scalar↩
                    (), streams_[k]);
                filters_[k]->apply(a_[k], b_[k], Rect↩
                    (0,0,-1,-1), streams_[k]);
                gpu::resize(b_[k], c_[k], lowResSize, 0, ↩
                    0, INTER_NEAREST, streams_[k]);

                diffSign(src[k], c_[k], c_[k], streams_[k↩
                    ]);

                upscale(c_[k], a_[k], scale_, streams_[k])↩
                    ;
                filters_[k]->apply(a_[k], b_[k], Rect↩
                    (0,0,-1,-1), streams_[k]);
                gpu::remap(b_[k], diffTerms_[k], ↩
                    forwardMaps_[k].first, forwardMaps_[k].↩
```

```
                second, INTER_NEAREST, BORDER_REPLICATE↩
                , Scalar(), streams_[k]);

            gpu::divide(diffTerms_[k],colorWeights_,↩
                diffTerms_[k],1.0,-1,streams_[k]);
        }

        if (lambda_ > 0)
        {
            calcBtvRegularization(highRes_, regTerm_, ↩
                btvKernelSize_);
            gpu::addWeighted(highRes_, 1.0, regTerm_, ↩
                -tau_ * lambda_, 0.0, highRes_);
        }

        for (size_t k = 0; k < src.size(); ++k)
        {
            streams_[k].waitForCompletion();
            gpu::addWeighted(highRes_, 1.0, diffTerms_↩
                [k], tau_, 0.0, highRes_);
        }

}
```

As variáveis $layers$, $colorMap$ e cK são liberadas a cada finalização do cálculo do denominador de $A^{(k)}$.

B
Lista de Símbolos

B.1
Tabela

Tabela B.1: Tabela de símbolos.

Símbolo	Significado
$\sum$	Somatório
$\parallel f(x) \parallel$	Aplicação de norma a $f(x)$
$A \in B$	A pertence a B
$*$	Multiplicação termo-a-termo
∇	Gradiente
$diag(X)$	Matriz diagonal cujos elementos correspondem aos elementos de X
$A \subseteq B$	A está contido ou é igual a B
$\int$	Integral
div	Divergência
$min_u f(u)$	Valor de u que gera o mínimo da função f
X^T	Transposta da matriz X

C
Parâmetros da biblioteca OpenCV

C.1
Dados do sensor de profundidade

- **CV_CAP_OPENNI_DEPTH_MAP** - dados de pronfundidade em milímetros disponibilizados através do tipo de dado CV_16UC1;
- **CV_CAP_OPENNI_POINT_CLOUD_MAP** - coordenadas XYZ em metros disponibilizadas através do tipo de dado CV_32FC3;
- **CV_CAP_OPENNI_DISPARITY_MAP** - disparidade entre os pixels disponibilizada através do tipo de dado CV_8UC1;
- **CV_CAP_OPENNI_DISPARITY_MAP_32F** - disparidade entre os pixels disponibilizada através do tipo de dado CV_32FC1;
- **CV_CAP_OPENNI_VALID_DEPTH_MASK** - máscara de pixels válidos (ex.: não oclusos) disponibilizada através do tipo de dado CV_8UC1.

C.2
Dados do sensor RGB

- **CV_CAP_OPENNI_BGR_IMAGE** - imagem colorida disponibilizada através do tipo de dado CV_8UC3;
- **CV_CAP_OPENNI_GRAY_IMAGE** - imagem em tons de cinza disponibilizada através do tipo de dado CV_8UC1.

C.3
Flags para alternância de propriedades

- **CV_CAP_OPENNI_IMAGE_GENERATOR** - quando se deseja o acesso ao sensor que capta imagens RGB;
- **CV_CAP_OPENNI_DEPTH_GENERATOR** - quando se deseja acesso ao sensor de profundidade.

C.4
Opções para o gerador de imagens

- **CV_CAP_OPENNI_OUTPUT_MODE** - três modos de saída são suportados:
  - **CV_CAP_OPENNI_VGA_30HZ** - retorna imagens no modo VGA a 30 frames por segundo;
  - **CV_CAP_OPENNI_SXGA_15HZ** - retorna imagens no modo SXGA a 15 frames por segundo;
  - **CV_CAP_OPENNI_SXGA_30HZ** - retorna imagens no modo SXGA a 30 frames por segundo.

C.5
Opções para o sensor de profundidade

- **CV_CAP_OPENNI_REGISTRATION** - se está ativo, o mapa de profundidade é remapeado para o mesmo ponto de vista do gerador de imagens; caso contrário, o ponto de vista do mapa de profundidade se mantém inalterado; após o mencionado processo de registro cada pixel da imagem RGB está necessariamente alinhado ao respectivo pixel do mapa de profundidade.
- **CV_CAP_OPENNI_FRAME_MAX_DEPTH** - profundidade má-xima suportada em mílimetros;
- **CV_CAP_OPENNI_BASELINE** - distância entre os sensores CMOS e a fonte IR em mm;
- **CV_CAP_OPENNI_FOCAL_LENGTH** - distância focal em pixels;
- **CV_CAP_FRAME_WIDTH** - largura do frame em pixels;
- **CV_CAP_FRAME_HEIGHT** - altura do frame em pixels;
- **CV_CAP_FRAME_FPS** - taxa de frames em fps.

C.6
Membros a serem inicializados

- **int scale** - fator de escala que descreve em quantas vezes a resolução dos frames originais será aumentada;
- **int iterations** - número de iterações do processo de regularização;
- **double tau** - valor assintótico do método "Steepest Descendent";
- **double lambda** - parâmetro que ajusta a contribuição entre o termo de dados e o termo de regularização para o resultado final;
- **double alpha** - parâmetro que regula a distribuição espacial utilizada na execução do algoritmo;
- **int btvKernelSize** - tamanho do $kernel$ utilizado para regularização;
- **int blurKernelSize** - tamanho do $kernel$ de suavização Gaussiano;
- **double blurSigma** - parâmetro $sigma$ utlizado para suavização Gaussiana;
- **int temporalAreaRadius** - indica quantos frames anteriores e posteriores ao frame corrente devem ser utilizados para o processamento do frame corrente;
- **Ptr<DenseOpticalFlowExt>** - através desse membro indicamos o algoritmo de opticalFlow que estimará a movimentação entre os frames.