UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Juliano Emir Nunes Masson

Geração de mapas de profundidade utilizando redes neurais convolucionais

Florianópolis
2021

Juliano Emir Nunes Masson

Geração de mapas de profundidade utilizando redes neurais convolucionais

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Engenharia de Automação e Sistemas.
Orientador: Prof. Daniel Ferreira Coutinho, Dr.
Coorientador: Prof. Marcelo Roberto Petry, Dr.

Florianópolis
2021

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

Masson, Juliano Emir Nunes
Geração de mapas de profundidade utilizando redes neurais convolucionais / Juliano Emir Nunes Masson ; orientador, Daniel Ferreira Coutinho, coorientador, Marcelo Roberto Petry, 2021.
94 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2021.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Fotogrametria. 3. Multi-View Stereo. 4. Redes neurais. I. Coutinho, Daniel Ferreira. II. Petry, Marcelo Roberto. III. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Engenharia de Automação e Sistemas. IV. Título.

Juliano Emir Nunes Masson

Geração de mapas de profundidade utilizando redes neurais convolucionais

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Eric Aislan Antonelo, Dr.
Universidade Federal de Santa Catarina

Prof. Marcelo Ricardo Stemmer, Dr.
Universidade Federal de Santa Catarina

Prof. Maurício Edgar Stivanello, Dr.
Instituto Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Engenharia de Automação e Sistemas.

Documento assinado digitalmente
Werner Kraus Junior
Data: 10/04/2021 14:10:06-0300
CPF: 531.085.239-53
Verifique as assinaturas em https://v.ufsc.br

Coordenação do Programa de
Pós-Graduação

Documento assinado digitalmente
Daniel Ferreira Coutinho
Data: 09/04/2021 12:59:11-0300
CPF: 379.169.980-68
Verifique as assinaturas em https://v.ufsc.br

Prof. Daniel Ferreira Coutinho, Dr.
Orientador

Documento assinado digitalmente
Marcelo Roberto Petry
Data: 12/04/2021 04:26:56-0300
CPF: 005.992.230-30
Verifique as assinaturas em https://v.ufsc.br

Prof. Marcelo Roberto Petry, Dr.
Coorientador

Florianópolis, 2021.

Dedico este trabalho a todos aqueles que, de alguma forma, auxiliaram para a concretização desta etapa.

AGRADECIMENTOS

Agradeço a minha família pelo apoio durante o mestrado. Ao professor Dr. Daniel Ferreira Coutinho pela ajuda nas decisões acadêmicas e organização das tarefas. Ao professor Dr. Marcelo Roberto Petry pelos conselhos e apoio na execução dos testes. A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro através do Programa de Excelência Acadêmica (PROEX).

RESUMO

A fotogrametria é um problema clássico de visão computacional e consiste na extração de informação tridimensional a partir de imagens. Ela pode ser descrita de maneira simplificada como sendo o processo de gerar um modelo 3D, de alguma cena capturada por duas ou mais imagens, de diferentes pontos de vista. A possibilidade de extrair a geometria de uma cena através de imagens permite uma vasta gama de aplicações, como a criação de ambientes virtuais para a simulação de robôs autônomos, a virtualização de ambientes reais para venda de imóveis, análise de deformação de superfícies, cálculo do deslocamento de barragens, monitoramento de erosões, planejamento e inspeção de construções, etc. Mesmo com o uso massivo de redes neurais em problemas clássicos de visão computacional mostrando um grande avanço quando comparados aos algoritmos tradicionais (principalmente em tarefas de reconhecimento de objetos), os principais *softwares* comerciais e bibliotecas de fotogrametria utilizados para as aplicações citadas anteriormente ainda não fazem uso de redes neurais. A partir dessa constatação, esse trabalho tem como objetivo estudar metodologias de geração de mapas de profundidade a partir de imagens de câmeras calibradas (matriz intrínseca e extrínseca conhecidas) utilizando redes neurais convolucionais. Para alcançar o objetivo foi feita uma revisão sistemática dos trabalhos da área, e com base nas estruturas das redes encontradas, foram propostas algumas modificações na rede CasMVSNet. Como a base da fotogrametria é encontrar correspondências entre as imagens, as modificações propostas focaram na etapa de extração de *features*, trocando as convoluções por convoluções deformáveis e deformáveis moduladas, permitindo uma maior adaptação da rede aos dados de entrada. Para os experimentos foram escolhidos três *datasets*, o DTU, *Tanks and Temples* e o BlendedMVS. Para o treinamento foram utilizados o DTU e o BlendedMVS, e para a avaliação quantitativa dos resultados o DTU e o *Tanks and Temples*. Dentre as modificações propostas, a *dconv_todas* treinada com o *dataset* DTU teve uma redução de 22% no consumo de memória gráfica, melhora na completude e na média do resultado quantitativo do DTU e uma pontuação média maior no *Thanks and Temples*, com uma penalidade de apenas 3.75% no tempo de processamento, em comparação a rede CasMVSNet original.

Palavras-chave: Fotogrametria. *Multi-View Stereo*. Redes neurais.

ABSTRACT

Photogrammetry is a classic computer vision problem and consists of extracting three-dimensional information from images. It can be simply described as the process of generating a 3D model of a scene captured by two or more images from different points of view. The possibility of extracting the geometry of a scene through images enables a wide range of applications, such as the creation of virtual environments for the simulation of autonomous robots, the virtualization of real environments for real estate sales, surface deformation analysis, calculation of dam displacement, erosion monitoring, planning and inspection of construction sites, etc. Even with the massive use of neural networks in classical computer vision problems showing a great advance when compared to traditional algorithms (especially in object recognition tasks), the main commercial softwares and photogrammetry libraries used for the aforementioned applications still do not make use of neural networks. Based on this finding, this work aims to study methodologies for generating depth maps from calibrated camera images (known intrinsic and extrinsic matrix) using convolutional neural networks. To achieve the goal, a systematic review of the works in the area was performed, and based on the structures of the networks found, some modifications to the CasMVSNet network were proposed. As the basis of photogrammetry is to find correspondences between images, the proposed modifications focused on the feature extraction step, changing the convolutions for deformable and modulated deformable convolutions, allowing a better adaptation of the network to the input data. For the experiments three datasets were chosen, the DTU, Tanks and Temples, and BlendedMVS. For training, DTU and BlendedMVS were used, and for quantitative evaluation of the results, DTU and Tanks and Temples were used. Among the proposed modifications, *dconv_todas* trained with the DTU dataset had a 22% reduction in graphics memory consumption, improved completeness and average quantitative DTU results, and a higher average score on Thanks and Temples, with a penalty of only 3.75% in processing time compared to the original CasMVSNet network.

Keywords: Photogrammetry. Multi-View Stereo. Neural networks.

LISTA DE FIGURAS

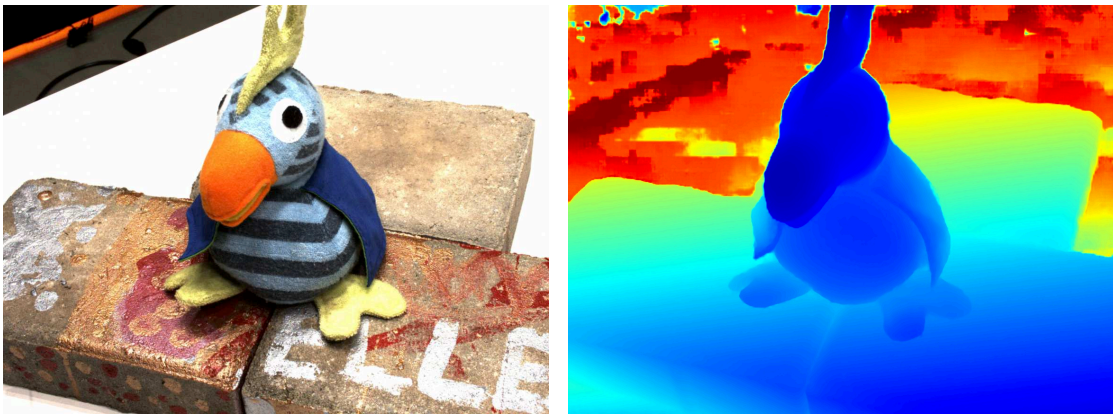
LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A fotogrametria, ou reconstrução 3D a partir de imagens, é um problema clássico de visão computacional e tem suas etapas bem definidas (Seção 1.1). Ela pode ser descrita de maneira simplificada como sendo o processo de gerar um modelo 3D, de alguma cena capturada por duas ou mais imagens, de diferentes pontos de vista. A principal etapa desse problema é a geração da nuvem densa, conhecida como *Multi-View Stereo* (MVS), que é o foco desse trabalho. O MVS pode ser feito de diversas maneiras, mas a abordagem mais comum é o uso do algoritmo *plane sweep stereo* (Seção 2.1.2.3). Nesse algoritmo, o espaço 3D é amostrado em D planos paralelos ao plano da imagem de referência, cada um desses planos é uma hipótese de onde podem estar os objetos na cena. Com os planos definidos, cada um dos pixeis da imagem de referência é projetado nas imagens vizinhas utilizando uma homografia (Seção 2.1.1.5), e é feito o cálculo da similaridade entre o pixel da imagem de referência com os pixeis das imagens vizinhas, caso eles tenham uma intensidade parecida, a similaridade é alta, caso contrário a similaridade é baixa (outra interpretação análoga é usar o custo da correspondência, que seria o inverso da similaridade). Esse processo se repete para todos os D planos, criando um volume de similaridades, que é filtrado e para cada pixel da imagem de referência se extrai o plano com a maior similaridade, criando um mapa de profundidade da imagem de referência. Um exemplo de mapa de profundidade pode ser visto na Figura 1, onde os pontos mais próximos da câmera estão em azul escuro e os pontos mais distantes em vermelho escuro.

Figura 1 – Exemplo de um mapa de profundidade.



(a) Imagem de referência. (b) Mapa de profundidade.

Fonte – Elaborado pelo autor.

A possibilidade de extrair a geometria de uma cena através de imagens permite uma vasta gama de aplicações. Guerra *et al.* (2019) criaram um ambiente de simulação para robôs com modelos 3D gerados através da fotogrametria. Esse ambiente permite que a simulação utilize os robôs reais através de um sistema de captura de movimento,

criando a chamada *vehicle-in-the-loop simulation*. Como o principal objetivo do trabalho é a simulação de robôs autônomos, que usam uma câmera como sensor principal, a fidelidade das imagens é essencial para representar o comportamento dos algoritmos no mundo real. Dentro desse cenário, este trabalho foca na fotogrametria, pois além de recuperar a geometria da cena, ela também consegue gerar texturas de alta resolução. A virtualização de ambientes não se restringe apenas a simulações. A Matterport [1], uma empresa que disponibiliza equipamentos e *softwares* para a virtualização de ambientes, tem como principais aplicações a virtualização de imóveis para aluguel/venda e registro de seguro, de ambientes comerciais para a oferta de produtos e serviços, etc. Durante a pandemia do COVID-19, um exemplo de virtualização de ambientes foi o *showroom* virtual criado pela Acer para a apresentação de seus novos produtos durante a CES 2021, que neste ano foi totalmente virtual [2]. A virtualização de objetos também é um tópico recente com a indústria 4.0, com a criação de *digital twins* (TAO *et al.*, 2019). Um *digital twin* é um modelo 3D de algum equipamento real que tem por objetivo simular sua operação, permitindo a troca de informações entre o equipamento no mundo real e sua simulação, prevendo falhas e possíveis erros. A fotogrametria, quando realizada por um Veículo Aéreo Não Tripulado (VANT), também tem aplicações na área de engenharia civil, como: a análise de deformações em superfícies, útil para o cálculo do deslocamento de barragens, monitoramento de erosões, movimentação do solo, etc (MELO *et al.*, 2020); cálculo de volume e inventário de pilhas de materiais em mineradoras [3]; planejamento e inspeção de construções [4]. Ainda em conjunto com um VANT, a fotogrametria pode trazer dados para a agricultura, com a análise do terreno, para geração de curvas de nível, altimetria, e se utilizada uma câmera multi-espectral, pode também trazer índices de qualidade da plantação [5]. Além das aplicações exemplificadas, algo que ainda não é muito difundido, principalmente no Brasil, é a realidade virtual e realidade aumentada, limitada aqui, por seu preço elevado. Com o amadurecimento de tais tecnologias esse cenário deve mudar, trazendo mais possibilidades de aplicações para a fotogrametria, que seguirá como um problema relevante na área de visão computacional.

1.1 FOTOGRAMETRIA

A fotogrametria pode ser dividida em quatro etapas bem definidas, ilustradas na Figura 2. A primeira etapa é a estimação da postura das câmeras que é feita utilizando algoritmos de *Structure from Motion* (SfM). O primeiro passo é detectar as *features*

[1] https://matterport.com/
[2] https://acertechconnects.com/series/expo-hall-general/series_tradeshow
[3] https://pix4d.com/industry/mining
[4] https://3dsurvey.si/case-studies/test-case
[5] https://mappa.ag/analises-agronomicas

(Seção 2.1.1.3) nas imagens e realizar o *matching* (Seção 2.1.1.4) entre elas. Com os *matches* relacionando cada par de imagens, é possível estimar a postura relativa entre elas. Partindo de um par inicial, a postura das outras imagens é atualizada mantendo o par inicial como referência. Como o processo de detecção de *features* e *matching* não é perfeito, a atualização incremental da postura das imagens vai acumulando erros. Para diminuir esses erros é adicionada uma otimização da postura das câmeras, chamada de *bundle adjustment*, essa otimização atualiza a postura das câmeras de modo que a soma total do erro de reprojeção das *features* seja minimizado. A segunda etapa é a geração da nuvem densa, conhecida como *Multi-View Stereo* (MVS), que é o foco desse trabalho. Diversos métodos podem ser aplicados nessa etapa, mas o mais comum é a geração de mapas de profundidade através do algoritmo *plane sweep stereo* (Seção 2.1.2.3). O princípio básico dele é a aplicação de homografias (Seção 2.1.1.5) para a distorção das imagens seguida da avaliação do custo de correspondência entre os pixeis. Essa correspondência usualmente é calculada com algoritmos de *template matching* (HARTLEY; ZISSERMAN, 2004).

Figura 2 – Etapas da fotogrametria.

Fonte – (MASSON, 2019).

A terceira etapa é a geração de superfícies, onde o objetivo é gerar uma malha 3D que una os pontos da nuvem de pontos. Essa etapa é especialmente útil para reduzir o consumo de recursos computacionais para exibir o modelo 3D gerado. Uma nuvem de pontos pode facilmente ultrapassar os milhões de pontos, algo que pode não ser desejado se o objeto em questão não necessitar de tantos pontos para representar sua superfície. Com uma malha 3D, as partes da nuvem de pontos que forem planas, podem ser substituídas por apenas alguns triângulos, simplificando consideravelmente o custo computacional de armazenamento, transmissão e exibição dos dados. A quarta etapa é a texturização, nela são utilizadas as imagens com as posturas calculadas na primeira etapa para projetar a cena na malha 3D. Caso a nuvem de pontos densa tenha cor, dependendo da aplicação, não é necessário realizar essa etapa. Ainda pode-se adicionar uma etapa de filtragem no final, com filtros para remoção de ruídos, elementos artificiais criados pela etapa de geração de superfície e simplificação de malhas 3D (HARTLEY; ZISSERMAN, 2004).

Apesar de ter diversas aplicações, a fotogrametria não pode ser utilizada para reconstruir qualquer tipo de objeto. Como a base do algoritmo é a detecção de *features*,

a fotogrametria depende das imagens possuírem características únicas, distinguíveis e constantes entre as imagens, para poder realizar o *matching*. Se as imagens não satisfazem essas características, não é possível encontrar correspondências suficientes entre as imagens para relacioná-las, parando o algoritmo já no início. Objetos brilhosos, reflexivos, transparentes e sem textura possuem tais características, fazendo deles um desafio para a fotogrametria. Objetos brilhosos e reflexivos criam pontos de luz que se repetem entre as imagens, mas não na mesma posição e orientação, criando falsas correspondências. Objetos transparentes e sem textura não possuem características distinguíveis para serem relacionadas entre as diferentes imagens. Outra limitação que aparece pela utilização de imagens é a necessidade de iluminação, que também precisa ser constante, pois mudanças de iluminação podem fazer com que correspondências corretas sejam descartadas por apresentarem diferentes intensidades de luz (HARTLEY; ZISSERMAN, 2004).

1.2 MOTIVAÇÃO

As redes neurais tem se tornado um tema cada vez mais presente na área de visão computacional. As redes neurais convolucionais (Seção 2.2.1), em especial, tem trazido novas abordagens para problemas clássicos. Essa mudança de paradigma aconteceu principalmente pelo poder de adaptação e aprendizado das redes, sendo aplicada em algoritmos de reconhecimento e classificação de objetos, detecção de falhas, problemas que tem uma maior dependência da etapa de detecção de *features*. Isso acontece pois algoritmos clássicos de detecção de *features* são feitos de maneira "artesanal", com o desenvolvedor decidindo quais as sequências de operações devem ser feitas para atingir um detector adequado a aplicação. Já a rede neural consegue se adaptar a tarefa e ao conjunto de dados apresentado a ela, e com um treinamento suficientemente grande e variado, ela consegue generalizar o problema. Como visto na seção anterior, a fotogrametria depende da etapa de detecção de *features*, tornando ela uma boa candidata para o uso de redes neurais. Apesar dos avanços nas redes neurais convolucionais, a maioria dos *softwares* comerciais que utilizam a fotogrametria continuam a utilizar os métodos clássicos, motivando esse trabalho a documentar e compreender como esse novo paradigma de redes neurais se aplica no problema da fotogrametria, em especial na etapa do *Multi-View Stereo*.

1.3 OBJETIVOS

O objetivo geral dessa dissertação é desenvolver uma metodologia de geração de mapas de profundidade a partir de imagens de câmeras calibradas (matriz intrínseca e extrínseca conhecidas) utilizando redes neurais convolucionais. Os objetivos específicos do trabalho são os seguintes:

- Aprofundar o conhecimento no tópico de redes neurais.
- Mapear quais são as abordagens utilizadas para algoritmos de *Multi-View Stereo* baseados em redes neurais.
- Modificar um algoritmo para a geração de mapas de profundidade utilizando redes neurais convolucionais.

1.4 DELIMITAÇÃO DO ESCOPO

Esse trabalho foca em algoritmos de *Multi-View Stereo*, sendo assim, será considerado que as câmeras estão corretamente calibradas e sua postura é fixa. Mesmo utilizando redes neurais convolucionais para atenuar algumas das limitações da detecção de *features* da fotogrametria, objetos transparentes ainda serão um problema, não sendo o foco das reconstruções. O trabalho se limitará a modificar a estrutura de uma rede já existente, não sendo proposta uma nova arquitetura de rede neural para a geração de mapas de profundidade.

1.5 ESTRUTURA DO DOCUMENTO

No Capítulo 2 são abordados os conceitos fundamentais de visão computacional e redes neurais. No Capítulo 3 é apresentada a revisão bibliográfica feita para mapear o estado da arte. No Capítulo 4 são apresentadas as modificações propostas na rede neural escolhida. No Capítulo 5 são apresentados os experimentos, como eles foram conduzidos e seus resultados, com uma breve discussão. No Capítulo 6 são descritas as conclusões obtidas com os experimentos, as principais contribuições e sugestões para trabalhos futuros.

2 CONCEITOS FUNDAMENTAIS

Neste capítulo serão apresentados os conceitos fundamentais para compreender o desenvolvimento dessa dissertação. Os conceitos foram divididos em duas grandes áreas, os conceitos de visão computacional e os conceitos de redes neurais artificiais. Os conceitos de visão computacional abrangem desde a formação da imagem até a descrição de algumas técnicas clássicas utilizadas na reconstrução 3D. Os conceitos de redes neurais artificiais focam em explicar de forma simplificada o funcionamento dos principais tipos de redes neurais artificiais.

2.1 VISÃO COMPUTACIONAL

Os conceitos fundamentais de visão computacional foram divididos em visão 2D e visão 3D. Na visão 2D são apresentados os conceitos referentes ao espaço da imagem, como a formação da imagem, detecção de *features*, *matching*, homografia, entre outros. Na visão 3D são apresentadas algumas técnicas clássicas para extrair a informação tridimensional de uma cena, seja por duas câmeras acopladas, uma câmera e um projetor ou uma câmera e vários pontos de vista.

2.1.1 Visão 2D

2.1.1.1 Coordenadas homogêneas

Coordenadas homogêneas, ou coordenadas no espaço projetivo $\mathbb{P}^n$ (CORKE, 2017), são muito utilizados na visão computacional pois permitem que as transformações projetivas (Seção 2.1.1.5) possam ser resolvidas utilizando equações lineares ao invés de não-lineares. Quando utiliza-se as coordenadas homogêneas, um vetor de n dimensões pode ser manipulado em um espaço de $n+1$ dimensões. Esse grau de liberdade extra permite que pontos e linhas no infinito sejam representados apenas com números reais. Isso também significa que a escala não é mais importante. Em outras palavras, seja $\tilde{x}_a$ um vetor pertencente ao $\mathbb{P}^n$, então $\tilde{x}_a$ e $\tilde{x}_b = \alpha\tilde{x}_a$, $\alpha \in \mathbb{R}$, representam o mesmo ponto no espaço euclidiano, para todo $\alpha \neq 0$ (CORKE, 2017) (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

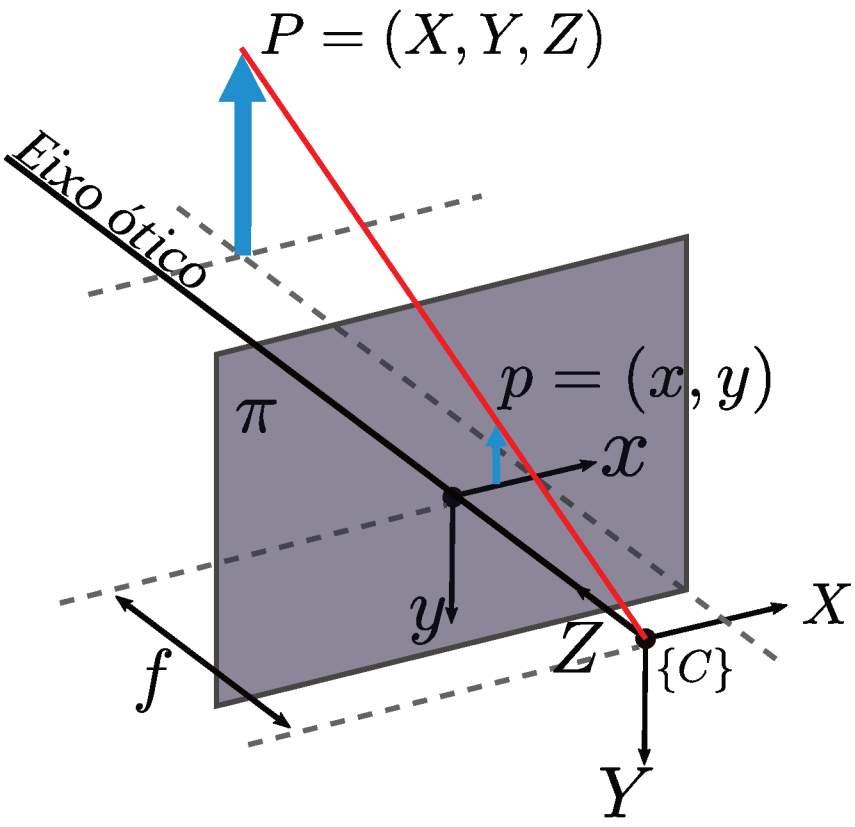
Dado um vetor euclidiano $x = (x_1, x_2, \cdots, x_n)$, ele pode ser construído no espaço projetivo adicionado-se uma dimensão ao vetor $\tilde{x} = (x_1, x_2, \cdots, x_n, 1)$, onde o til representa o vetor em coordenadas homogêneas. Para retornar o vetor para o espaço euclidiano pode-se utilizar a seguinte equação (CORKE, 2017):

$$x_i = \frac{\tilde{x}_i}{\tilde{x}_{n+1}}, i = 1, \cdots, n \tag{1}$$

2.1.1.2 Formação da imagem

O processo de formação da imagem, seja ele em um olho ou em uma câmera, consiste na projeção de um mundo tridimensional em uma superfície bidimensional. A dimensão perdida nessa projeção é relativa a profundidade, dessa forma, não é possível identificar o tamanho de objetos nem a sua distância do olho/câmera. Para recuperar essa informação existem alguns métodos disponíveis, sendo um deles o tema desse trabalho e alguns outros exemplos clássicos descritos na Seção 2.1.2. Para compreender esses métodos é necessário compreender como se representa uma câmera matematicamente. O primeiro modelo de câmera da história foi a câmera *pinhole*, ou câmera obscura. Ela consiste em uma caixa totalmente fechada com apenas um pequeno orifício em uma de suas paredes, a luz passa pelo orifício e é projetada de forma invertida na parede oposta ao orifício. A câmera obscura tem todos os objetos em foco, mas como não possui lente para focar a luz, esse tipo de câmera fica restrita a cenas muito iluminadas. Para permitir o uso da câmera em cenas menos iluminadas é necessária a adição de uma lente. Isso facilita a captação de luz, mas adiciona-se a necessidade de focar os objetos de interesse (CORKE, 2017) (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011) (SZELISKI, 2010).

Figura 3 – Modelo de perspectiva central.



Fonte – Adaptado de (MASSON, 2019).

Para modelar uma câmera (projeção 3D em uma superfície 2D) pode-se usar

diversos modelos, mas o mais comum é o modelo de perspectiva central como ilustrado na Figura 3. Como essa câmera possui lente, os raios são concentrados em um ponto, no caso, a origem da câmera $\{C\}$. A imagem é formada no plano π a uma distância $z = f$ da origem da câmera, sendo f a distância focal. Com esse modelo pode-se relacionar o ponto $P = (X, Y, Z)$ no espaço 3D com sua projeção, o ponto $p = (x, y)$ no espaço 2D, utilizando a similaridade de triângulos:

$$x = f\frac{X}{Z}, y = f\frac{Y}{Z} \tag{2}$$

Pode-se perceber que isso cria um conjunto de equações não-lineares. Para simplificar essa relação pode-se utilizar as coordenadas homogêneas (Seção 2.1.1.1), de modo a obter uma relação linear. Reescrevendo p e P com coordenadas homogêneas tem-se:

$$\tilde{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tilde{P} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

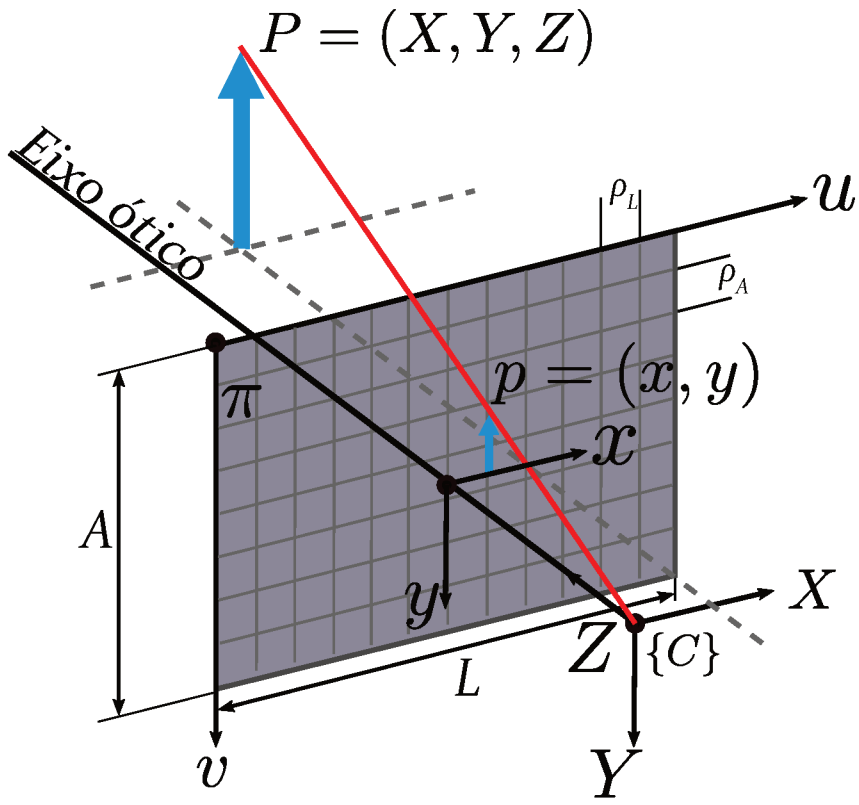
$$\tilde{p} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{P} \tag{4}$$

Essa equação representa uma projeção de perspectiva, que leva os pontos do espaço 3D para o plano da imagem e tem as seguintes propriedades (CORKE, 2017):

- Linhas retas no mundo são projetadas como linhas retas no plano da imagem.
- Linhas paralelas no mundo se encontram no ponto de fuga no plano da imagem, com exceção de linhas paralelas que estão em um plano paralelo ao plano da imagem.
- Cônicas no mundo são mapeadas para cônicas no plano da imagem, mas não necessariamente a mesma.
- A transformação não preserva a forma dos objetos.
- Não existe uma única inversa para a transformação, dado um ponto p, não pode-se determinar um ponto P. A única informação que se sabe, é que o ponto P deve estar em alguma posição da linha vermelha da Figura 3.

Como as imagens capturadas pelas câmeras tem que ser processadas por um computador, é necessário discretizar o plano da imagem. O modelo de perspectiva central discretizado pode ser visto na Figura 4. A discretização divide o plano da

Figura 4 – Modelo de perspectiva central com o plano da imagem discretizado.



Fonte – Adaptado de (MASSON, 2019).

imagem π em um *grid* de $L \times A$ pixeis, sendo L a largura e A a altura da imagem. As coordenadas dos pixeis são inteiros não negativos definidos seguindo os eixos u e v, com a origem definida por convenção no canto superior esquerdo. Os pixeis são considerados idealmente uniformes e os eixos uv e xy estão alinhados, dessa forma a conversão das coordenadas do plano da imagem para os pixeis é dada por (CORKE, 2017):

$$u = \frac{x}{\rho_L} + u_0, v = \frac{v}{\rho_A} + v_0 \tag{5}$$

Onde ρ_L é a largura dos pixeis, ρ_A a altura dos pixeis e (u_0, v_0) é o ponto principal (ponto onde o eixo ótico intercepta o plano da imagem). Assim como anteriormente, pode-se simplificar a equação utilizando coordenadas homogêneas e a representação matricial, relacionando o ponto P diretamente com as coordenadas em pixeis:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} \frac{1}{\rho_L} & 0 & u_0 \\ 0 & \frac{1}{\rho_A} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{P} \tag{6}$$

Sendo o ponto $(\tilde{u}, \tilde{v}, \tilde{w})$ as coordenadas homogêneas em pixeis do ponto P. Como os eixos de referência do espaço 3D não necessariamente coincidem com os

eixos de referência da câmera, é necessário adicionar uma transformação de corpo rígido. Isto é feito adicionado-se a matriz de rotação R e o vetor de translação t da câmera à Equação (6) (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011):

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \underbrace{\underbrace{\begin{bmatrix} \frac{1}{\rho_L} & 0 & u_0 \\ 0 & \frac{1}{\rho_A} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} R & t \\ 0_{1\mathbf{x}3} & 1 \end{bmatrix}}_{E}}_{C} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{7}$$

Onde K é a matriz de parâmetros intrínsecos da câmera, E é a matriz de parâmetros extrínsecos e C a matriz da câmera. Os parâmetros intrínsecos estão relacionados com as características de construção da câmera, como a largura e altura dos pixeis, o ponto onde o eixo ótico intercepta o plano π da imagem (u_0, v_0) e a distância focal f, que dependendo da câmera pode ser fixa ou variável. Os parâmetros extrínsecos estão relacionados com a postura da câmera na cena. No caso da fotogrametria, os algoritmos de *Structure from Motion* (SfM) são responsáveis por estimar os parâmetros intrínsecos e extrínsecos da câmera. A matriz C é uma transformação homogênea 3×4 que escalona, translada, rotaciona e faz projeção de perspectiva (CORKE, 2017).

2.1.1.3 Detecção de *features*

Imagens conseguem carregar uma grande quantidade de dados, mas fazem isso de maneira desestruturada. Uma imagem colorida, por exemplo, é apenas um conjunto de três matrizes, uma para cada canal de cor. Quando olhamos para uma imagem de um grupo de pessoas, por exemplo, conseguimos rapidamente extrair informações, como a quantidade de pessoas, a cor das roupas, o tipo das roupas, o ambiente onde a foto foi capturada, etc. Para um computador, essa interpretação de alto nível não é trivial, pois ele apenas "enxerga" um conjunto de matrizes. Para transformar os dados da imagem em informações relevantes, o computador geralmente busca por características específicas, como cantos, quinas, bordas, etc. Nessa etapa que atuam os métodos de detecção de *features*.

Segundo Siegwart, Nourbakhsh e Scaramuzza (2011) uma *feature*, *keypoint* ou característica é um padrão de imagem que difere de seus vizinhos em termos de intensidade, cor e textura, podendo ser formada por pequenos *blobs* (segmento da imagem), bordas ou quinas. Essas *features* podem ser divididas em três categorias relativas a sua interpretação semântica: a primeira categoria seriam as *features* com interpretação semântica, como as linhas da estrada em um automóvel autônomo ou *blobs* em um sistema de detecção de células cancerígenas; as que não tem interpretação semântica, ou seja, o que ela representa não importa, o foco é ter precisão

e repetibilidade na sua detecção; a última categoria é composta pelas *features* que tem interpretação semântica apenas quando estão agrupadas, isso acontece principalmente em aplicações de detecções de cenas. Para o foco desse trabalho serão utilizadas as *features* sem interpretação semântica, já que o desafio é encontrar as mesmas *features* em múltiplas imagens de maneira robusta e precisa. Existem diversos algoritmos que detectam *features*, mas para classificar a qualidade de um detector deve-se analisar os seguintes critérios (TUYTELAARS; MIKOLAJCZYK *et al.*, 2008):

- Repetibilidade: dada duas imagens da mesma cena, capturadas de ângulos diferentes, uma alta porcentagem das *features* detectadas em uma imagem devem ser detectadas novamente na outra imagem.
- Distintividade: os padrões utilizados pelo detector devem ser suficientemente sensíveis a mudanças para distinguir *features* que representam pontos diferentes na cena, evitando falsos positivos durante o processo de correspondência.
- Localidade: as *features* devem ser locais, reduzindo a probabilidade de oclusão (já que apenas uma pequena porção da cena deve ser encontrada em múltiplas imagens) e permitindo que seja utilizado um modelo de planaridade local para aproximar as deformações geométricas e fotométricas de duas imagens com diferenças no ângulo de visão.
- Quantidade: o número de *features* ideal depende da aplicação em foco, para a reconstrução 3D, seja na etapa da estimação da posição das câmeras ou na etapa da geração da nuvem densa, usualmente quanto mais *features* melhor é o resultado.
- Precisão: as *features* devem ser precisamente localizadas, tanto na localização da imagem quanto em escala e formato, já que erros na localização se propagam para as outras etapas, pois o posicionamento das câmeras é estimado através das correspondências de *features*.
- Eficiência: preferencialmente a detecção de *features* deve ser feita no menor tempo possível consumindo pouca memória para armazenar as informações coletadas.

Dentre todos os critérios levantados por Tuytelaars, Mikolajczyk *et al.* (2008), pode-se considerar a repetibilidade o critério mais crítico considerando a aplicação de reconstrução 3D, já que a robustez dos algoritmos está baseada na premissa que as mesmas *features* podem ser encontradas em múltiplas imagens. Isso fica claro observando a Figura 4, com apenas uma imagem não é possível determinar em qual ponto da linha vermelha está posicionado o ponto P. Entretanto, conforme se

incrementa o número de imagens com visão do mesmo ponto, pode-se estimar por triangulação a sua posição. Essa repetibilidade desejada pode ser alcançada através das seguintes formas (TUYTELAARS; MIKOLAJCZYK *et al.*, 2008):

- Invariância: quando se espera grandes deformações, a melhor abordagem é modelar essas deformações matematicamente e desenvolver o detector de modo a ser invariante a essas deformações.
- Robustez: quando se espera pequenas deformações, é suficiente fazer o detector menos sensível a essas deformações. Essa técnica é geralmente utilizada para tornar o detector robusto a ruídos na imagem, efeitos da discretização, artefatos provenientes da compressão, etc.

Sabendo das características de um bom detector de *fetures*, ainda tem-se que escolher entre os três tipos de detectores disponíveis: detectores de *blobs*, bordas ou quinas. Os detectores de bordas e quinas sofrem com mudanças de escala, fazendo com que eles não sejam adequados para aplicações de reconstrução 3D, restando apenas os detectores de *blobs*. *Scale Invariant Feature Transform* (SIFT) (LOWE, 1999) é invariante a rotação e escala, e parcialmente invariante à mudanças em iluminação e no ponto de vista da câmera. Sua grande repetibilidade e alta taxa de *matching* faz dele um dos detectores de *features* mais utilizados; *Speeded-Up Robust Features* (SURF) (BAY; TUYTELAARS; VAN GOOL, 2006), também invariante a escala e fortemente inspirado no SIFT. Ele utiliza uma aproximação da matriz Hessiana através de um conjunto de filtros quadrados e utiliza imagens integrais para as convoluções. Isto acelera a detecção com uma pequena redução na robustez, comparado ao SIFT; *Accelerated*-KAZE (AKAZE) (ALCANTARILLA; SOLUTIONS, 2013), também é invariante a escala, mas ao invés de utilizar o espaço de escala Gaussiano, construído de forma piramidal no SIFT ou através de aproximações das derivadas Gaussianas como no SURF, o AKAZE utiliza um espaço de escala não-linear. Assim, sua suavização é adaptada localmente, suavizando pequenos detalhes mas preservando as bordas dos objetos (MASSON, 2019).

2.1.1.4 *Matching*

Matching se refere ao processo de identificar quais *features* de duas imagens distintas se referem ao mesmo ponto de uma cena. Quando duas *features* representam o mesmo ponto elas criam um *match*/correspondência. No caso da reconstrução 3D, como existe uma grande sobreposição entre as imagens vizinhas, sabe-se que elas devem ter uma grande quantidade de correspondências. Aqui será assumido que os detectores de *features* foram desenvolvidos de maneira que a distância euclidiana entre as *features* pode ser diretamente utilizada para classificar potenciais correspondências. Segundo Szeliski (2010) considerando-se a distância euclidiana como métrica,

a alternativa mais simples para encontrar correspondências é definir um limite máximo de distância e usar esse limite para recuperar todas as possíveis correspondências. Segundo Fawcett (2006) pode-se quantificar a performance de um algoritmo de *matching* em um determinado limite contabilizando o número de correspondências corretas, correspondências erradas e correspondências esquecidas, através das seguintes taxas:

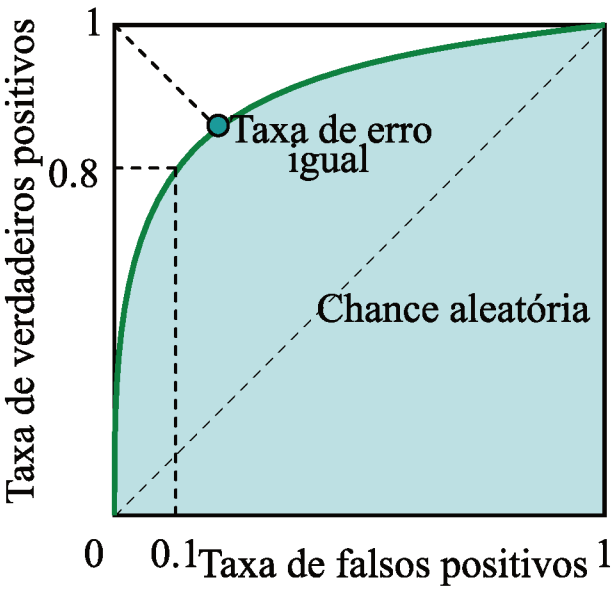
- Taxa de verdadeiros positivos (TVP)

$$TVP = \frac{VP}{VP + FN} = \frac{VP}{P} \tag{8}$$

- Taxa de falsos positivos (TFP)

$$TFP = \frac{FP}{FP + VN} = \frac{FP}{N} \tag{9}$$

Figura 5 – Curva ROC, relacionando as taxas de verdadeiros positivos e falsos positivos.



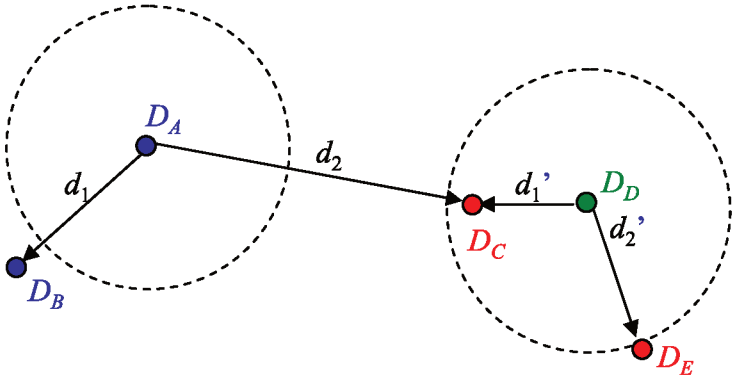
Fonte – Adaptado de (SZELISKI, 2010).

Onde *VP* são os verdadeiros positivos, *FN* falsos negativos, *FP* falsos positivos e *VN* verdadeiros negativos. Qualquer algoritmo de *matching* pode ser classificado utilizando as taxas TVP e TFP, idealmente a taxa de verdadeiros positivos deve se aproximar de 1 e a taxa de falsos positivos de 0. Variando o limite máximo do algoritmo, cria-se uma família de valores para as taxas TVP e TFP, criando uma curva, conhecida como *receiver operating characteristic* (curva ROC), essa curva pode ser visualizada na Figura 5. Quanto mais para o canto esquerdo superior a curva estiver, melhor é o algoritmo de *matching*. O problema desse algoritmo é que o limite tem que ser definido manualmente. Enquanto um limite muito alto cria falsos positivos, um limite muito baixo

cria falsos negativos, assim, uma melhor estratégia é utilizar o vizinho mais próximo no espaço das *features*. Como algumas *features* podem não ter correspondência (por oclusão, por exemplo) um limite ainda é utilizado para evitar falsos positivos. Idealmente, o limite deveria se adaptar a diferentes regiões do espaço de *features*, mas como isso não acontece, uma heurística simples é comparar a distância do vizinho mais próximo com a distância do segundo vizinho mais próximo. Essa heurística é definida como *nearest neighbor distance ratio* (NNDR):

$$NNDR = \frac{d_1}{d_2} = \frac{|D_A - D_B|}{|D_A - D_C|} \tag{10}$$

Figura 6 – Exemplo dos algoritmos de limite fixo, vizinho mais próximo e o NNDR.



Fonte – Adaptado de (SZELISKI, 2010).

Onde d_1 e d_2 são as distâncias do vizinho mais próximo e do segundo vizinho mais próximo, D_A é a *feature* principal, D_B e D_C os dois vizinhos mais próximos. Na Figura 6 tem-se um exemplo dos algoritmos de limite fixo, vizinho mais próximo e o NNDR, a linha pontilhada representa o algoritmo de limite fixo, com ele pode-se perceber que a *feature* D_A falha em ser relacionada com D_B e D_D é erroneamente relacionado com D_C; no algoritmo de vizinho mais próximo, D_A é corretamente relacionado com D_B, mas D_D é erroneamente relacionado com D_C; utilizando o NNDR, D_A ainda é relacionado com D_B, mas D_D não é relacionado com nenhuma das opções, pois a relação de distância $\frac{d_1'}{d_2'}$ é muito alta, corretamente rejeitando qualquer correspondência.

2.1.1.5 Homografia

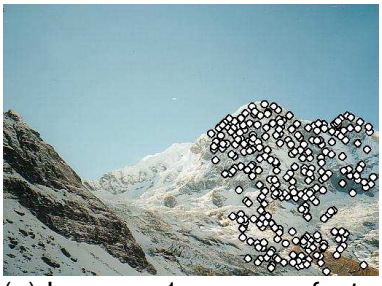
Segundo Hartley e Zisserman (2004) homografia, ou transformação projetiva, é uma transformação linear que opera em vetores homogêneos e possui oito graus de liberdade. Ela mantém as linhas retas da imagem, mas não preserva ângulos,

orientação, comprimentos e paralelismos, e é representada por uma matriz 3×3 não-singular:

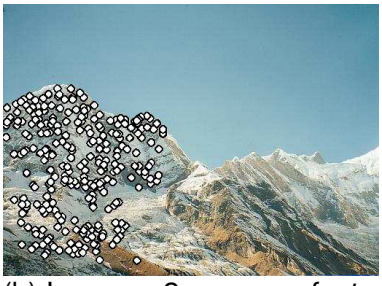
$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{11}$$

A homografia pode ser utilizada para fazer o mapeamento entre diferentes planos, para isso é necessário quatro pontos em cada um dos planos para estimar a transformada. Esse mapeamento é muito útil na área de visão computacional, pois como visto nas seções anteriores, com os algoritmos de detecção de *features* e *matching*, já tem-se a relação dos pontos nos dois planos das imagens. Apesar de qualquer conjunto de quatro pontos ser suficiente para estimar a homografia, a presença de *outliers* pode interferir no resultado. Para resolver isso se utiliza o *RANdom SAmple Consensus* (RANSAC) (CONSENSUS, 1981), um algoritmo que determina o melhor conjunto de pontos para fazer a estimação. A homografia é utilizada em algoritmos de remoção de distorção em imagens (HARTLEY; ZISSERMAN, 2004), construção de panoramas (BROWN; LOWE, 2007), realidade aumentada (WANG; OLSON, 2016), calibração da câmera (ZHANG, 2000), entre outros. Na Figura 7 tem-se um exemplo do algoritmo de construção de panoramas.

Figura 7 – Exemplo da construção de um panorama.



(a) Imagem 1, com as *features* em destaque.



(b) Imagem 2, com as *features* em destaque.

(c) Resultado da aplicação da homografia que leva a imagem 2 para o plano da imagem 1.

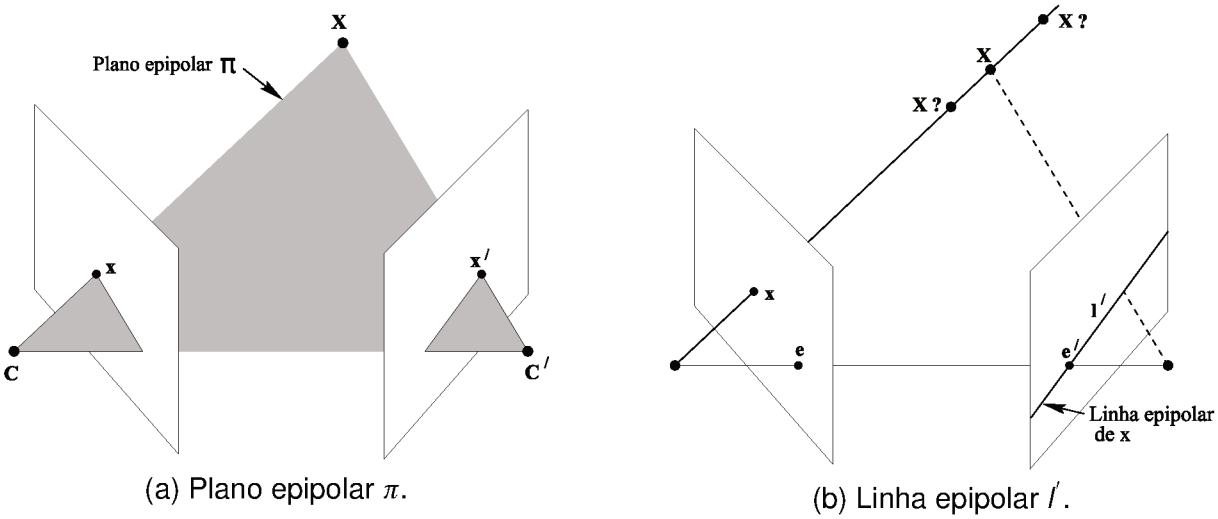
Fonte – Adaptado de (BROWN; LOWE, 2007).

2.1.1.6 Geometria epipolar

Até então, com os conceitos das Seções 2.1.1.3 e 2.1.1.4, é possível encontrar as *features* em um par de imagens e realizar o *matching* entre elas, esse processo resulta em um conjunto de correspondências, que combinados com os dados da matriz da câmera de ambas as imagens permite calcular a posição no espaço 3D de cada uma dessas correspondências. Quando o objetivo é encontrar a posição no espaço 3D de um pixel qualquer, que não possui uma *feature* centrada nele, é necessário achar a correspondência desse pixel na outra imagem sem utilizar os algoritmos de *matching*. A princípio isso não seria uma tarefa difícil, basta utilizar alguma técnica de *template matching* para achar a correspondência desse pixel na outra imagem. O problema dessa abordagem é o tamanho do espaço de busca, que seria a outra imagem inteira, tornando o processo muito demorado. Para reduzir o espaço de busca, e o tempo de processamento, aplica-se a limitação da geometria epipolar.

Segundo Hartley e Zisserman (2004) a geometria epipolar é a geometria projetiva intrínseca entre duas vistas, ela é independente da estrutura da cena e depende apenas dos parâmetros intrínsecos da câmera e suas poses relativas. A geometria epipolar entre duas vistas pode ser simplificada como sendo a geometria de intersecção dos planos das imagens com um conjunto de planos que tem como eixo a *baseline* das câmeras (linha que conecta o centro de uma câmera com o centro da outra câmera). Supondo que o ponto X no espaço 3D é visualizado por duas câmeras, na posição x na primeira câmera e na posição $x^{'}$ na segunda câmera, como esses pontos e o centro das câmeras são coplanares, tem-se o plano π como pode ser observado na Figura 8a.

Figura 8 – Exemplo da geometria epipolar.



(a) Plano epipolar π.

(b) Linha epipolar $l^{'}$.

Fonte – (MASSON, 2019).

A vantagem da geometria epipolar é que não é necessário conhecer todos os

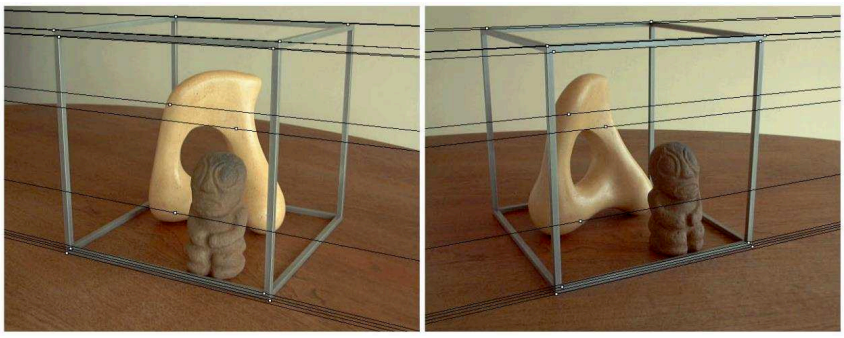
pontos para criar o plano π, essa propriedade é extremamente útil para limitar o espaço de busca de correspondências. Considerando a procura da correspondência do ponto x, sabe-se apenas a posição do ponto x e o centro das câmeras, com isso já se define o plano π. Com a interseção do plano π com o plano da segunda imagem, tem-se a linha epipolar l' e sabe-se que o ponto x' (correspondência do ponto x) estará nessa linha. Essa restrição aplicada pela linha epipolar diminui consideravelmente o espaço de busca, que ao invés de ser a imagem inteira, passa a ser apenas uma linha. A linha epipolar pode ser observada na Figura 8b. Apesar de limitar o espaço de busca deve-se lembrar que não são todos os pixeis que terão uma correspondência, pois o ponto em questão pode não ter sido capturado em na outra imagem, seja por oclusão ou por ser um objeto em movimento.

O *matching* dos pontos nessa estratégia é feito seguindo os princípios do *template matching*, essa técnica tenta encontrar qual porção da imagem se assemelha mais com um *templete* predefinido. No caso da busca pela linha epipolar, o *template* seria uma janela ao redor do ponto x e os testes de semelhança seriam feitos com janelas de pixeis centralizadas em cada um dos pontos que compõem a linha epipolar l' (MASSON, 2019). A geometria epipolar também pode ser representada geometricamente, através da matriz fundamental F, de tamanho 3×3, essa matriz faz o mapeamento de pontos em uma imagem com sua respectiva linha epipolar na outra imagem. A matriz fundamental também traz uma relação muito útil para checar as correspondências, caso x seja a correspondência de x', a equação a seguir deve ser satisfeita (HARTLEY; ZISSERMAN, 2004):

$$x'^T F x = 0 \tag{12}$$

Essa afirmação é verdadeira, pois, se x e x' são correspondentes, então x' está na linha epipolar $l' = Fx$ correspondente a x, em outras palavras, tem-se $0 = x'^T l' = x'^T F x$. De maneira análoga, os raios que ligam o ponto 3D X, com os pontos x e x' são coplanares. A Equação (12) também mostra que é possível encontrar a matriz fundamental utilizando apenas as correspondências, sem a necessidade de conhecer os parâmetros intrínsecos da câmera (ao menos 7 correspondências são necessárias para isso). Na Figura 9a, tem-se um exemplo das linhas epipolares para um par de imagens. Pode-se perceber que apesar de diminuir o espaço de busca da imagem inteira para apenas uma linha, ainda é necessário percorrer a imagem alterando as coordenadas x e y de acordo com a equação da reta. Apesar de ser um cálculo simples de se fazer, terá de ser feito milhões de vezes. Pode-se simplificar esse processo aplicando a retificação estéreo no par de imagens, fazendo com que as linhas epipolares se tornem paralelas, como pode ser visto na Figura 9b.

Figura 9 – Exemplo da linha epipolar e da retificação estéreo.



(a) Exemplos das linhas epipolares.

(b) Exemplos das linhas epipolares após a retificação estéreo.

Fonte – (SZELISKI, 2010).

2.1.2 Visão 3D

A captura da profundidade do mundo pode ser feita de diversas maneiras, seja ela com um sensor específico como o LIDAR, com um conjunto de sensores, como duas câmeras acopladas (câmera estereoscópica), uma câmera e um projetor infravermelho/luz visível ou com apenas uma câmera e diversos pontos de vista. Abaixo algumas dessas abordagens serão apresentadas.

2.1.2.1 Estereoscopia

A estereoscopia é uma técnica que utiliza duas câmeras acopladas para recuperar a informação de profundidade de uma cena. Essa técnica usa o mesmo processo feito pelo cérebro humano para estimar a profundidade, que se baseia na disparidade. A disparidade é a diferença horizontal em pixeis do mesmo ponto 3D observado em duas imagens.

Segundo Siegwart, Nourbakhsh e Scaramuzza (2011) a estereoscopia pode ser dividida em dois problemas: (*i*) o problema da correspondência; e (*ii*) o problema da reconstrução 3D. O problema da correspondência consiste em encontrar os pontos 2D em ambas as imagens que correspondem ao mesmo ponto 3D na cena observada,

essa etapa será detalhada nas Seções 2.1.1.3 e 2.1.1.4. Essa correspondência se baseia na suposição que as imagens tem apenas uma pequena variação de posição entre elas. Apesar disso, essa suposição não é suficiente pois falsos positivos podem surgir e os algoritmos de *matching* não são perfeitos. Para resolver essa característica é adicionada a limitação da geometria epipolar (Seção 2.1.1.6). Essa limitação define que um ponto em uma imagem deve estar em algum ponto de uma determinada linha na outra imagem, reduzindo o espaço de busca à uma dimensão.

Resolvido o problema da correspondência, teremos um conjunto de pares de pontos 2D, cada ponto do par pertencendo a uma das imagens. Antes de iniciar o problema da reconstrução 3D é necessário calibrar ambas as câmeras, de modo a estimar seus parâmetros intrínsecos (distância focal e centro ótico) e extrínsecos (postura relativa entre elas). Para exemplificar esse processo, considere um caso simplificado no qual as câmeras são idênticas e seus eixos óticos são paralelos com uma separação b (*baseline*). Esse caso é exemplificado na Figura 10, sendo que as duas câmeras, C_l e C_r, possuem a mesma distância focal f e observam o mesmo ponto (x, y, z) na cena. O ponto (x, y, z) está na posição 2D u_l na câmera C_l e na posição u_r na câmera C_r, criando a correspondência. Utilizando a similaridade de triângulos, podemos escrever (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011):

$$\frac{f}{z} = \frac{u_l}{x} \tag{13}$$

$$\frac{f}{z} = \frac{-u_r}{b-x} \tag{14}$$

Manipulando as duas equações define-se a profundidade do ponto:

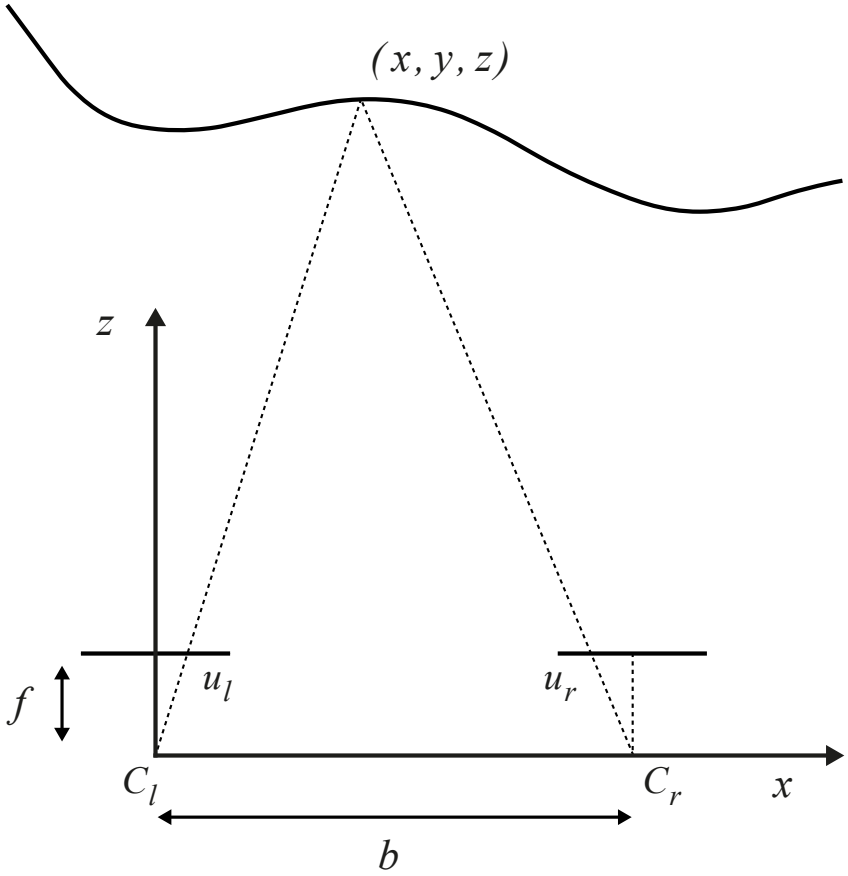
$$z = b\frac{f}{u_l - u_r} \tag{15}$$

Com a Equação (15) definida e com as correspondências encontradas na primeira etapa da estereoscopia, pode-se recuperar o mapa de disparidade/profundidade da cena.

2.1.2.2 Triangulação ativa por luz estruturada

A triangulação ativa por luz estruturada (TALE) é uma técnica que utiliza uma câmera acoplada a um projetor para recuperar a informação de profundidade de uma cena, o projetor emite um padrão luminoso na cena que é detectado pela câmera. O conjunto câmera-projetor é previamente calibrado, de modo que a imagem do padrão a uma determinada distância seja conhecida. Seguindo o funcionamento da estereoscopia, a imagem do padrão na cena é comparada com a imagem do padrão na etapa da calibração, assim, gerando um mapa de profundidade da cena.

Figura 10 – Exemplo de um par estéreo ideal.



Fonte – Adaptado de (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Para exemplificar o funcionamento da TALE, pode-se simplificar o projetor como sendo um único feixe de laser infravermelho como observado na Figura 11. O primeiro passo para utilizar a TALE consiste em calibrar o conjunto câmera-projetor: deve-se posicionar o conjunto para um plano de referência, medir a distância z_o, guardar na memória o pixel que representa o ponto *o* na imagem, calcular a distância focal *f* e a distância *b* (*baseline*) entre a câmera *C* e o projetor *L*. Com esses dados, pode-se posicionar o conjunto em qualquer outra cena para fazer as medições. No exemplo da Figura 11, tem-se o plano do objeto o qual o laser colide no ponto *k*. Como o plano do objeto está a uma distância Z_k do conjunto, que é menor que Z_o, tem-se as disparidades *D* no espaço 3D e a disparidade *d* no espaço 2D. Seguindo a mesma ideia da estereoscopia com a similaridade de triângulos, obtém-se as seguintes equações:

$$\frac{D}{b} = \frac{Z_o - Z_k}{Z_o} \tag{16}$$

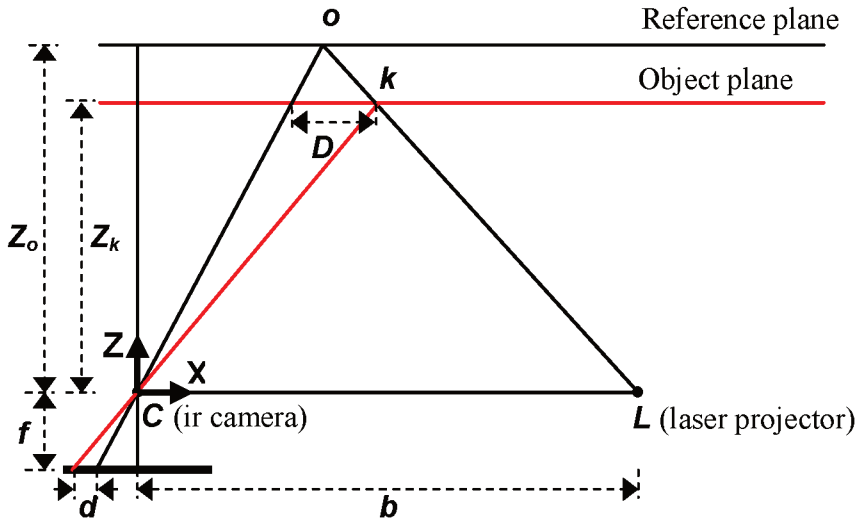
$$\frac{d}{f} = \frac{D}{Z_k} \tag{17}$$

Substituindo *D* da Equação (17) na Equação (16) tem-se que a distância Z_k é

dada por:

$$Z_k = \frac{Z_o}{1 + \frac{Z_o}{bf} d} \tag{18}$$

Figura 11 – Exemplo da triangulação ativa por luz estruturada.



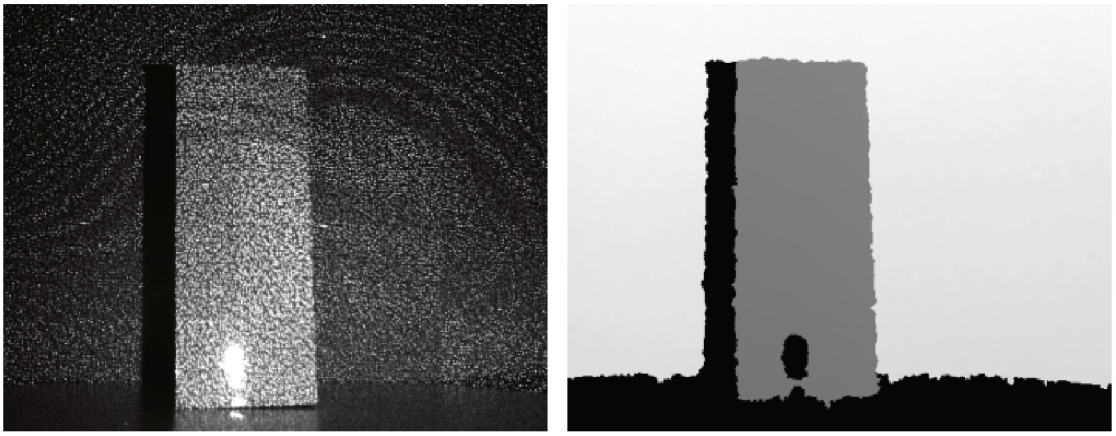
Fonte – Adaptado de (KHOSHELHAM, 2011).

Ao contrário da estereoscopia, que procura as correspondências de forma "aleatória" na imagem, a TALE busca o padrão emitido pelo projetor, isso elimina o problema presente na estereoscopia onde superfícies lisas/sem textura não apresentam características únicas e distinguíveis para se gerar as correspondências entre duas imagens.

Em contra-partida, o principal problema da TALE está associada a sua principal vantagem, pois se ela se destaca em relação a estereoscopia por gerar suas próprias correspondências com o padrão projetado, se esse padrão não é visualizado pela câmera, a TALE também falha em recuperar a profundidade da cena (GUPTA *et al.*, 2011). Um exemplo para esse problema seria o sensor Kinect da Microsoft (ZHANG, 2012), esse sensor não funciona corretamente quando existe uma alta incidência de luz solar na cena pois ele utiliza um projetor infravermelho.

Na Figura 12a, tem-se a imagem da câmera infravermelha do sensor Kinect, onde os pontos luminosos são o padrão emitido pelo projetor e, na Figura 12b, o mapa de profundidade gerado pelo sensor. No canto inferior da imagem da câmera infravermelha a região mais iluminada é o projetor, pode-se perceber que nessa região da imagem o sensor não consegue calcular a profundidade, gerando um buraco no mapa de profundidade. Isso ocorre pelo fato da câmera não conseguir distinguir cada um dos pontos separadamente nessa região, falhando em criar correspondências com o padrão calibrado.

Figura 12 – Exemplo da imagem infravermelha e do mapa de profundidade do Kinect.



(a) Imagem da câmera infravermelha. (b) Mapa de profundidade.

Fonte – Adaptado de (KHOSHELHAM, 2011).

2.1.2.3 *Plane sweep stereo*

Nas técnicas de visão 3D anteriores sempre existem dois equipamentos acoplados e previamente calibrados, sendo duas câmeras (estereoscopia) ou um conjunto de câmera-projetor (TALE). Como visto na introdução (Capítulo 1), é possível recuperar a informação de profundidade de uma cena através de uma câmera e diversos pontos de vista. O *plane sweep stereo* é um dos algoritmos que pode ser utilizado nesse caso, ele atua na etapa do *Multi-View Stereo*, onde se tem como entrada um conjunto de imagens com suas respectivas posturas da câmera, além dos parâmetros intrínsecos da câmera, e tem como saída uma nuvem de pontos.

O *plane sweep stereo* tem por objetivo encontrar o mapa de profundidade de uma imagem de referência através da correspondência de imagens vizinhas. Para isso ele amostra a cena em um conjunto de planos 3D, cada um desses planos será um guia para realizar as correspondências entre as imagens e é uma hipótese de onde pode existir uma superfície na cena real. O algoritmo tem como entrada M planos 3D para os testes de profundidade, $N+1$ imagens em diferentes pontos de vista e suas respectivas matrizes de projeção P_k:

$$P_k = K_k[R|t] \ com \ k = 1,\ldots,N \tag{19}$$

Onde K é a matriz de parâmetros intrínsecos da câmera (distância focal e centro ótico), R é a matriz de rotação e t o vetor de translação. Supõe-se que a câmera de referência se encontra na origem, de tal forma que: $P_{ref} = K_{ref}[I_{3\times3}|0_{3\times1}]$. O conjunto de planos Π_m com $m = 1,\ldots,M$ é definido nas coordenadas da imagem de referência da forma (GALLUP *et al.*, 2007):

$$\Pi_m = [n_m^T - d_m] \ para \ m = 1,\ldots,M \tag{20}$$

sendo $(\cdot)^T$ a operação de transposição, n_m um vetor unitário normal ao plano e d_m a distância do plano até a origem da câmera de referência para $m = 1, \ldots, M$. A distância entre cada um dos planos pode ser fixa ou variável. No caso de considerarmos uma varredura *fronto-parallel*, onde os M planos são paralelos ao plano da imagem de referência, tem-se que $n_1 = \cdots = n_M = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$. As profundidades d_m devem ficar dentro do intervalo $[d_{near}, d_{far}]$, sendo d_{near} a profundidade do primeiro plano e d_{far} a profundidade do último plano. Para testar o conjunto de planos para um dado pixel (x, y) da imagem de referência, esse pixel deve ser projetado em todas as N imagens. Como o mapeamento do plano da imagem de referência P_{ref} para o plano da imagem P_k é planar, ele pode ser definido como uma homografia H_{Π_m, P_k} induzida pelo plano Π_m, essa homografia é definida como (GALLUP *et al.*, 2007):

$$H_{\Pi_m, P_k} = K_k \left(R_k + \frac{R_k t_k n_m^T}{d_m} \right) K_{ref}^{-1} \text{ para } k = 1, \ldots, N \tag{21}$$

A localização (x_k, y_k) na imagem I_k do pixel (x, y) mapeado da imagem de referência é calculado por (GALLUP *et al.*, 2007):

$$[\tilde{x} \ \tilde{y} \ \tilde{w}]^T = H_{\Pi_m, P_k} [x \ y \ 1]^T \text{ para } k = 1, \ldots, N \tag{22a}$$

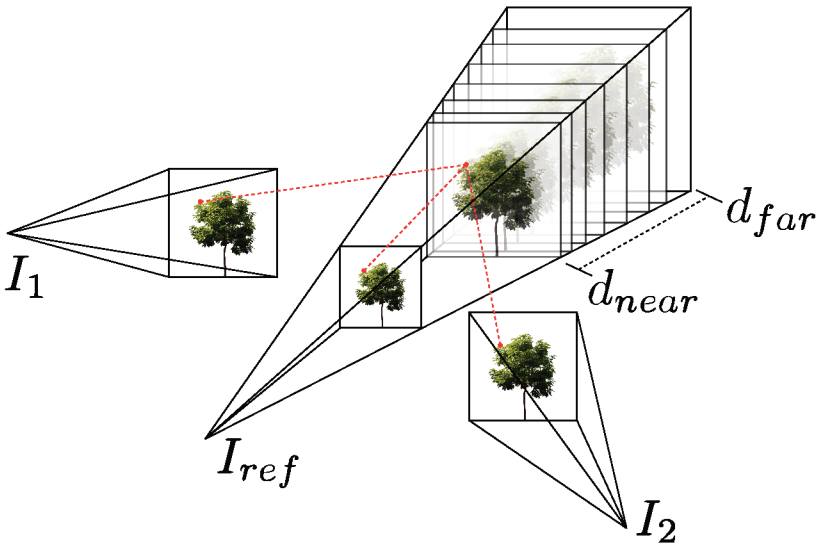
$$\tilde{x}/\tilde{w} = x_k \quad \tilde{y}/\tilde{w} = y_k \tag{22b}$$

Caso as intensidades dos pixeis $I_k((x_k, y_k))$ e $I_{ref}((x, y))$ sejam iguais, existe a possibilidade do plano ter interceptado alguma superfície presente no mundo real. Como se está analisando apenas um pixel não pode-se ter certeza dessa afirmação. Para aumentar o grau de confiança na estimativa, utiliza-se de uma janela retangular W centrada no pixel (x, y) que é comparada com outra janela retangular W centrada no pixel (x_k, y_k) (*template matching*). Uma função de custo pode ser definida em função da localização do pixel (x, y) na imagem de referência e do plano Π_m (GALLUP *et al.*, 2007):

$$C(x, y, \Pi_m) = \sum_{k=0}^{N-1} \sum_{(i,j) \in W} |I_{ref}(x-i, y-j) - \beta_k^{ref} I_k(x_k - i, y_k - j)| \tag{23}$$

sendo (x_k, y_k) obtido através da Equação (22b) e β_k^{ref} corresponde ao ganho entre a imagem k e a imagem de referência, esse ganho é feito para compensar mudanças de iluminação entre as imagens (GALLUP *et al.*, 2007). Após calcular a Equação (23) para todos os pixeis e todos os planos, tem-se o *cost-volume*, um volume que representa o custo para a correspondência entre as imagens. Na Figura 13 pode-se ver um exemplo do algoritmo, o ponto vermelho na imagem I_{ref} indica o ponto (x, y) e o pontos vermelhos nas imagens I_1 e I_2 indicam a reprojeção do (x, y), depois da projeção basta comparar a intensidade dos pixeis (nesse exemplo não se considerou a janela W).

Figura 13 – Exemplo do algoritmo *plane sweep stereo*.



Fonte – Elaborado pelo autor.

2.2 REDES NEURAIS ARTIFICIAIS

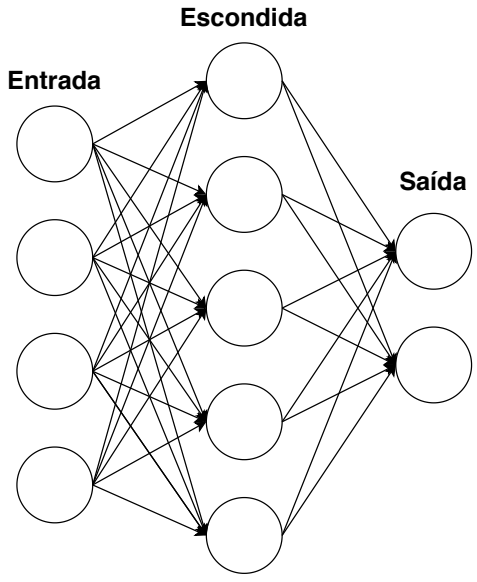
As redes neurais surgiram da vontade de entender e imitar como o cérebro humano resolve e interpreta os problemas. Criar um sistema computacional capaz de realizar tarefas complexas, como limpar uma casa, dirigir um carro, conversar com um humano, não é simples, principalmente partindo das técnicas utilizadas em problemas tradicionais, como desenvolver um sistema gerenciamento de finanças. Os programas tradicionais são especialmente desenhados para fazer cálculos aritméticos de forma rápida e seguir explicitamente uma lista de instruções. A maioria das tarefas realizadas por humanos não podem ser facilmente limitadas a uma porção de instruções de baixo nível a serem seguidas de maneira fixa, um exemplo seria o reconhecimento de objetos. Como criar um programa, que segue uma lista fixa de instruções, que afirma se existe ou não uma cadeira em uma foto? Existem diversas variações de cores e estilos possíveis em uma cadeira, além de outros aspectos, como os diferentes pontos de vista que a foto pode ser capturada (BUDUMA; LOCASCIO, 2017).

Muitos outros problemas caem nessa limitação, como o reconhecimento de escrita, reconhecimento de fala, tradução automática, entre outros. Essa limitação parte do fato de não sabermos quais são as exatas instruções que o cérebro humano utiliza para realizar tais tarefas, pois elas não foram ensinadas como fórmulas matemáticas, mas sim aprendidas por exemplos durante o amadurecimento do cérebro. Seguindo a tarefa do reconhecimento da cadeira, uma criança não nasce sabendo o que é uma cadeira, isso é um conceito ensinado que com o passar do tempo vai sendo aperfeiçoado cada vez que a criança acerta ou erra ao classificar um objeto como uma cadeira. Essa ideia de aprendizado está dentro do campo de inteligência artificial. Ao invés de passar

uma lista de instruções de como resolver o problema para o computador, é definido um modelo (por exemplo, a rede neural) que avalia os exemplos, e um pequenos conjunto de instruções para modificar o modelo quando ele comete um erro. Espera-se que com o passar do treinamento, um bom modelo seja capaz de resolver uma tarefa com uma alta precisão (BUDUMA; LOCASCIO, 2017). Nessa seção serão apresentados três tipos de redes neurais, as redes neurais multicamadas *feed-forward*, as redes neurais convolucionais e as redes neurais recorrentes.

As redes neurais multicamadas *feed-forward* trazem uma descrição mais próxima do funcionamento dos neurônios encontrados em nosso cérebro. Similar a estrutura biológica, as redes neurais definem o neurônio como uma unidade de processamento, que através de uma operação matemática gera uma saída a partir de N entradas. Na Figura 14 pode-se ver um exemplo de uma rede neural multicamadas *feed-forward*, ela pode ser dividida em três camadas, a camada de entrada, constituída pelas entradas, a camada escondida (podendo ser 1 ou N camadas escondidas, com um número variável de neurônios), que faz o processamento da entrada e por fim a camada de saída, que descreverá o resultado. As setas indicam a ligação entre os neurônios e o fluxo da informação, que é sempre da esquerda para a direita (CIABURRO; VENKATESWARAN, 2017).

Figura 14 – Exemplo de uma rede neural multicamadas *feed-forward*.



Fonte – Elaborado pelo autor.

Os neurônios podem ser descritos por 3 elementos, os pesos, o viés e a função de ativação. Os pesos são parâmetros numéricos que definem o impacto de cada conexão de entrada no neurônio, o viés é um parâmetro numérico adicional que é utilizado para ajustar a saída do neurônio em conjunto com a soma dos pesos e a função de ativação é uma função matemática não-linear que define o valor da saída

após os cálculos do neurônio. Esta relação pode ser escrita da forma (CIABURRO; VENKATESWARAN, 2017):

$$y = f\left(\left(\sum_{i=1}^{N} w_i x_i\right) + \theta\right) \tag{24}$$

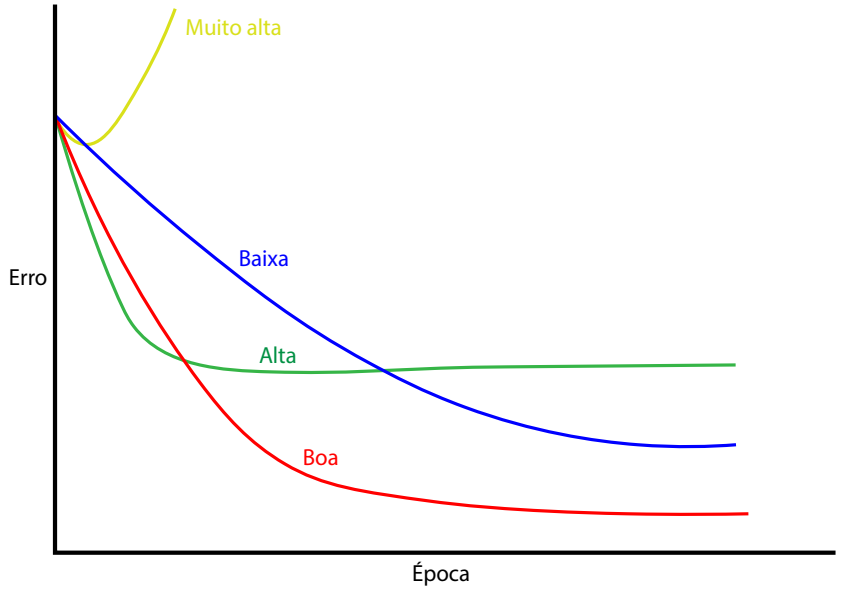
sendo y a saída do neurônio, x_i a i-ésima entrada, w_i o peso da i-ésima entrada, θ o viés, N o número total de entradas e f a função de ativação não-linear. O processo que faz esse cálculo partindo da camada de entrada passando pela(s) camada(s) escondida(s) e finalizando na camada de saída é chamado de propagação *forward*, isso corresponde a uma execução da rede. Durante o processo de treinamento da rede, quando a propagação *forward* chega ao fim, é calculado o erro (o resultado obtido menos o resultado esperado). Com o erro calculado é iniciado o processo de *backpropagation*, nele a derivada parcial da função de custo com relação a cada peso da rede neural é utilizada para identificar a intensidade e direção com que os pesos e vieses devem ser atualizados (CIABURRO; VENKATESWARAN, 2017).

Para controlar a velocidade com que o algoritmo converge existe a taxa de aprendizagem, ela define a intensidade das atualizações que o processo *gradient descent* irá realizar nos pesos e nos vieses. Na Figura 15 tem-se um exemplo da variação do erro durante as épocas (uma época consiste em treinar a rede com o conjunto completo dos dados) para uma mesma rede neural utilizando diferentes taxas de aprendizagem. Uma taxa de aprendizagem muito alta faz com que o algoritmo se desestabilize e não convirja; uma taxa alta faz com que ele chegue rápido a um valor baixo, mas que no caso exemplificado é um mínimo local, fazendo com que o algoritmo não consiga melhorar o erro; uma taxa de aprendizagem muito baixa faz com que o algoritmo demore muito tempo para chegar a um valor baixo de erro; e por fim, uma taxa de aprendizado boa, faz com que o algoritmo convirja lentamente para o menor erro possível (CIABURRO; VENKATESWARAN, 2017).

2.2.1 Redes Neurais Convolucionais

Com o entendimento dos princípios básicos da rede neural multicamadas, podemos partir para a rede neural convolucional. Esse tipo de rede é utilizado para o processamento de imagens, principalmente para a classificação de imagens. As redes neurais multicamadas também podem ser utilizadas para o processamento de imagens, mas a quantidade de pesos a serem treinados pode inviabilizar seu uso, uma imagem colorida com resolução 32×32 teria 3.072 ($32 \times 32 \times 3$) entradas. Para imagens pequenas não existiriam grandes problemas, mas com uma imagem em resolução *Full HD*, 1920×1080, já seriam necessários 2.073.600 entradas, considerando apenas uma camada escondida. A rede neural convolucional resolve esse problema

Figura 15 – Exemplo de diferentes taxas de aprendizado.



Fonte – Adaptado de (CIABURRO; VENKATESWARAN, 2017).

dispondo os neurônios em três dimensões, largura, altura e profundidade (CIABURRO; VENKATESWARAN, 2017).

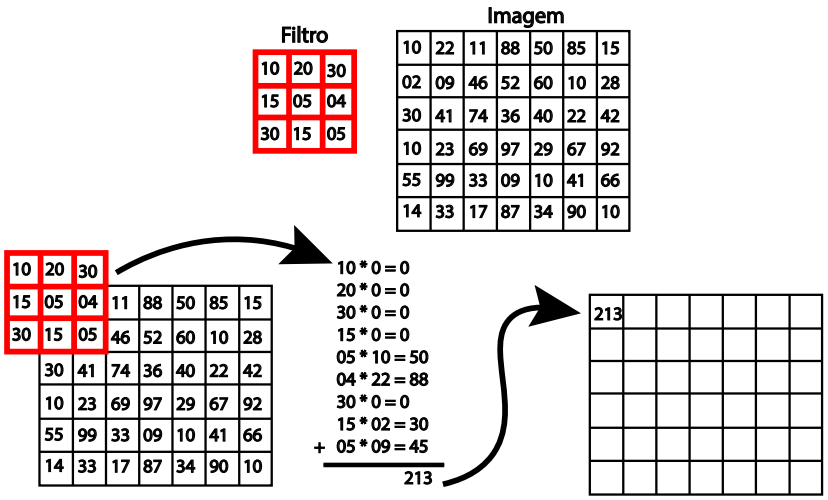
Tem-se três tipos de camadas em uma rede neural convolucional, a camada convolucional, a camada de *pooling* e a camada *fully connected*. A camada convolucional convolve a imagem com um filtro para extrair as características mais importantes de imagem. O operador da convolução pode ser visto a seguir (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \tag{25}$$

onde S seria a imagem resultante da operação de convolução, I a imagem de entrada, K o filtro, i a variável responsável por iterar no eixo x da imagem, j a variável responsável por iterar no eixo y da imagem, m a variável responsável por iterar no eixo x do filtro e n a variável responsável por iterar no eixo y do filtro. Na Figura 16 pode-se observar um exemplo de uma convolução. O filtro deve ter uma largura/altura ímpar, pois ele deve ter um pixel central, utilizado para alinhar o filtro com a imagem. No caso exemplificado, o resultado da operação tem o mesmo tamanho da imagem, sendo assim, o filtro já é alinhado com o primeiro pixel da imagem (canto esquerdo superior), pode-se também evitar as multiplicações por zero (pelo filtro estar fora da imagem), mas isso acaba diminuindo a resolução da imagem de saída. Após o cálculo executado, o filtro é deslocado em x pixeis para a direita, até que toda a imagem seja percorrida, a quantidade que é deslocada é chamada de *stride*. Com um $stride = 1$, todos os pixeis da imagem de referência serão multiplicados pelo pixel central do filtro. Com

um *stride* = 2, apenas metade dos pixeis da imagem de referência serão multiplicados pelo pixel central do filtro, gerando uma saída com metade da resolução da imagem original. Os valores do filtro são os pesos da rede neural convolucional, são eles que serão otimizados para "aprender" a tarefa desejada (CIABURRO; VENKATESWARAN, 2017).

Figura 16 – Exemplo de uma camada de convolução.

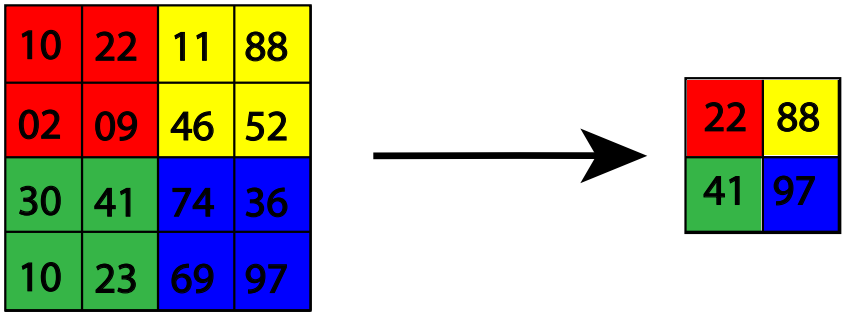


Fonte – Elaborado pelo autor.

A camada de *pooling* é utilizada para reduzir o tamanho da imagem, para isso ela também usa um filtro, geralmente com uma largura/altura de 2 ou 3 pixeis, e assim como na camada de convolução, pode-se definir o deslocamento do filtro com o *stride*. Na Figura 17 pode-se ver o resultado da execução dessa camada com um filtro de *altura*/*largura* = 2 e *stride* = 2, as cores relacionam quais pixeis na imagem foram usados para gerar os pixeis na imagem resultante, pode-se perceber que o resultado da operação é encontrar o valor máximo da região analisada. Após essa camada pode-se adicionar uma etapa de *batch normalization*, essa etapa tenta normalizar os valores dentro de uma faixa ótima de trabalho, isso evita que a rede fique enviesada para algum tipo de especifidade do conjunto de imagens usadas no treinamento (CIABURRO; VENKATESWARAN, 2017).

Por fim tem-se a camada *fully connected*, essa camada pode ser dividida em três camadas, análogas a rede neural multicamadas, a primeira tem por objetivo transformar a representação de matriz até então utilizada por um vetor, similar a camada de entrada da rede neural multicamadas; a segunda etapa funciona como a camada escondida, aplicando pesos para as entradas; por fim, dado uma entrada, a última etapa dá as probabilidades para essa entrada em relação a cada classe (no caso de um problema de classificação) (CIABURRO; VENKATESWARAN, 2017).

Figura 17 – Exemplo de uma camada de *pooling* com *stride* = 2.

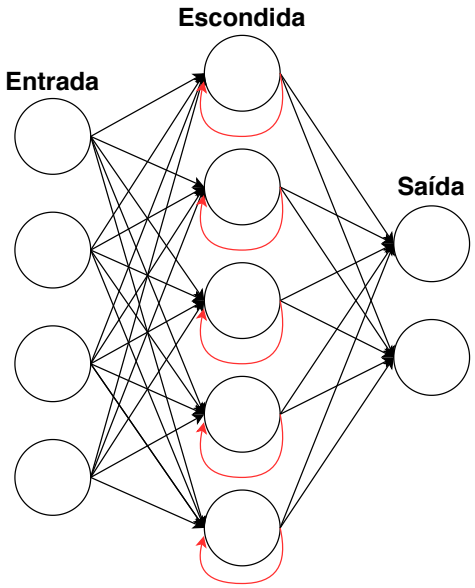


Fonte – Elaborado pelo autor.

2.2.2 Redes Neurais Recorrentes

As redes neurais recorrentes visam o processamento de sequências de dados em contraste com a rede neural convolucional que foca no processamento de imagens. A rede neural recorrente é caracterizada por possuir uma memória interna, isso pode ser expressado através de ciclos que podem se formar entre seus neurônios. Dentre suas aplicações estão a previsão do valor de ações, previsão do clima, traduções, reconhecimento de fala, entre outras (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figura 18 – Exemplo de uma rede neural recorrente.



Fonte – Adaptado de (CIABURRO; VENKATESWARAN, 2017).

As redes apresentadas anteriormente passam a entrada por uma sequência de cálculos de modo a gerar uma saída, a saída $i+1$ não será afetada pelo valor da entrada i. No caso de uma rede neural recorrente, além da entrada, a rede também leva em conta os dados passados. A decisão feita por ela no momento i irá afetar a sua decisão no momento $i+1$, pode-se dizer que a rede neural recorrente tem duas

fontes de entrada, o presente e o passado recente. Na Figura 18 tem-se o exemplo de uma rede neural recorrente, onde as setas vermelhas indicam a memória interna da rede (CIABURRO; VENKATESWARAN, 2017). Esse conceito de memória interna também pode ser aplicado em conjunto com redes neurais convolucionais, permitindo seu uso no processamento de imagens, como exemplificado em algumas redes da Seção 3.2.3.

3 REVISÃO BIBLIOGRÁFICA

A revisão sistemática se mostra uma técnica muito útil para mapear o estado da arte de alguma área específica, conhecer as abordagens que estão sendo mais utilizadas e quais ainda não foram e podem ser aplicadas a um determinado problema. Para executar a revisão sistemática foram seguidas as diretrizes dos trabalhos (COSTA; ZOLTOWSKI, 2014) e (BRASIL. MINISTÉRIO DA SAÚDE. SECRETARIA DE CIÊNCIA, 2012), para organizar as etapas foi utilizado o *software* StArt (FABBRI *et al.*, 2016), um *software* para revisão sistemática desenvolvido pelo laboratório de pesquisa e engenharia de *software* (LaPES) da UFSCar. Essa seção será dividida em duas sub-seções, a primeira irá explicar a metodologia utilizada para realizar a revisão sistemática, enquanto a segunda estará focada em apresentar os resultados encontrados com a revisão.

3.1 METODOLOGIA

A revisão seguiu os seguintes passos: definição da pergunta de pesquisa; definição das bases de busca a serem utilizadas; definição dos critérios de inclusão e exclusão dos artigos e por fim a análise e síntese dos resultados. Com a crescente utilização de redes neurais (convolucionais e recorrentes) na área de visão computacional, principalmente na área de identificação de objetos (interesse impulsionado pelo rápido crescimento de veículos autônomos), surge a dúvida de como essa técnica está sendo utilizada em outros segmentos da visão computacional. Em específico, no foco desse trabalho, a área de *Multi-View Stereo*. Sendo assim, a pergunta de pesquisa que guiará essa revisão sistemática é a seguinte:

- Pergunta de pesquisa: “Quais são as abordagens utilizadas para algoritmos de *Multi-View Stereo* baseados em redes neurais?”.

Para buscar a resposta para essa pergunta foram selecionadas as bases de dados *Scopus* [1], *IEEE Explore* [2] e *Web of Science* [3], selecionadas dentre as bases de dados disponibilizadas pela UFSC. Foram selecionadas as seguintes palavras chaves para a busca: *multi-view*, *multiview*, *stereo*, *stereopsis*, *supervised*, *unsupervised*, *neural*, *network*, *learning*, *convolutional*, cnn, *inference*, *volume* e *cost volume*. As palavras chaves foram agrupadas com conectores lógicos para formar uma *string* de busca que poderia ser utilizada nas bases de dados:

[1] `https://www.scopus.com/`
[2] `https://ieeexplore.ieee.org/`
[3] `https://www.webofknowledge.com`

- *String* de busca: "(*multi-view OR multiview*) *AND* (*stereo OR stereopsis*) *AND* (*supervised OR unsupervised OR neural OR network OR learning OR convolutional OR cnn OR inference OR volume OR* (*cost AND volume*))".

Essa *string* de busca foi aplicada em todas as bases de dados, adicionando-se também os seguintes critérios de inclusão e exclusão:

(i) Tipo de documento: artigo;

(ii) Idioma: inglês;

(iii) Presença da *string* de busca no título, resumo ou palavras chaves;

(iv) Ano de publicação: 2015 até 2020.

O resultado de cada uma das bases de dados por ser observado na Tabela 1. Os resultados das bases de dados foram inseridos no *software* StArt para iniciar a seleção dos artigos, o qual identificou 217 artigos duplicados que foram removidos da análise. Os artigos restantes foram classificados de acordo com a relação com o tema e a disponibilidade de acesso. Nessa etapa apenas o título e o resumo foram utilizados para a seleção, restando 67 artigos. A próxima seleção foi feita avaliando a relevância do artigo para a pergunta de pesquisa. Foram avaliadas a introdução e conclusão dos artigos, restando 19 artigos. O último passo foi a leitura completa dos artigos selecionados. Nessa etapa foi encontrado um plágio, que foi eliminado da revisão. Na Figura 19 pode-se visualizar o processo de seleção dos artigos.

Tabela 1 – Resultado da pesquisa nas bases de dados (24/08/2020).

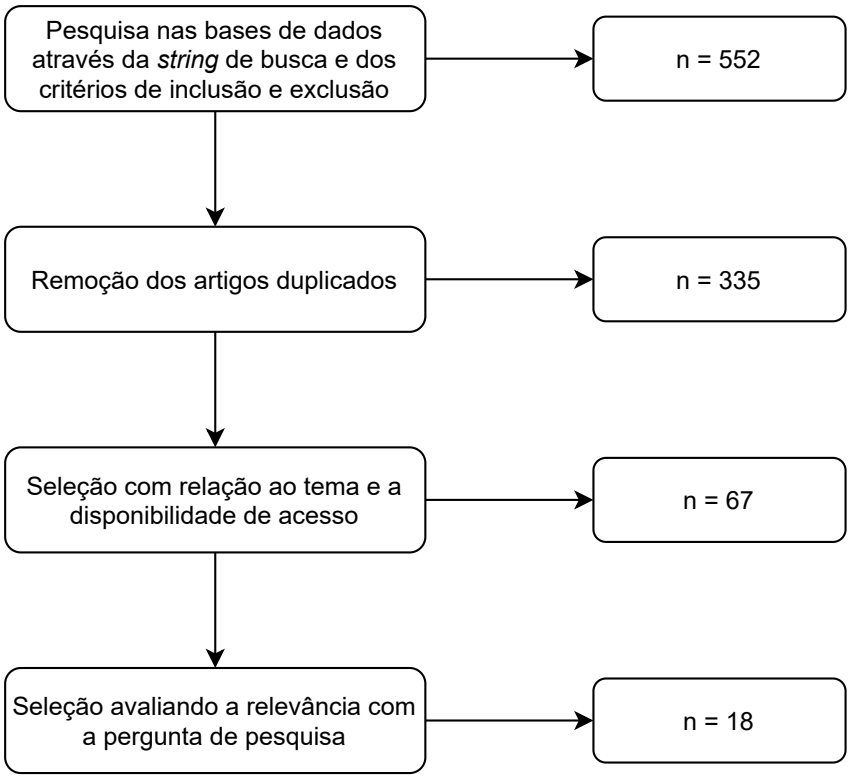
Base de dados	**Número de artigos**
Scopus	201
IEEE Explore	181
Web of Science	170
Total	552

Fonte – Elaborado pelo autor.

3.2 RESULTADOS

Como o foco da revisão sistemática é em algoritmos *Multi-View Stereo*, sabe-se que a entrada será um conjunto de imagens, suas respectivas posturas e os parâmetros intrínsecos da câmera. Esses dados podem ser gerados através da presença de tags fiduciais nas imagens (KROGIUS; HAGGENMILLER; OLSON, 2019), utilizando um ambiente controlado com um braço robótico com as posições previamente calibradas (JENSEN *et al.*, 2014) ou utilizando algoritmos de *Structure from Motion*, que

Figura 19 – Etapas da revisão sistemática.



Fonte – Elaborado pelo autor.

estimam a posição das câmeras através da identificação e *matching* de *features* entre as imagens (SCHÖNBERGER; FRAHM, 2016). Apesar de não ser necessário calcular a postura das câmeras, ainda tem-se o desafio de como codificar os dados de entrada na rede neural, visto que a forma com que o problema é apresentado para a rede implica diretamente na qualidade do resultado e se ele vai ou não ser válido. Esse foi um ponto em comum entre os artigos selecionados, a forma com que a informação de entrada é codificada na rede neural. O problema do *Multi-View Stereo* é, em geral, apresentado para a rede neural seguindo o algoritmo *plane sweep stereo* (Seção 2.1.2.3) com a seguinte divisão de etapas:

- Extração de *features*
- Geração do *cost volume*
- Regularização do *cost volume*
- Regressão da profundidade
- Refinamento (opcional)

Nas seções seguintes cada uma dessas etapas será brevemente explicada, e os artigos mais notáveis serão comentados. Na Tabela 2 encontra-se uma descrição sucinta da abordagem utilizada por cada um dos trabalhos.

3.2.1 Extração de *features*

A extração de *features* segue o mesmo papel da detecção de *features* descrita na Seção 2.1.1.3, encontrar *features* únicas e distinguíveis na imagem, mas ao contrário dos métodos clássicos, aqui as próprias camadas convolucionais são responsáveis por definir quais são os aspectos mais importantes para o problema a ser resolvido. Uma comparação entre os métodos clássicos e as redes neurais de extração de *features* é apresentada pelo trabalho (CHOI *et al.*, 2018).

Nesta etapa a maioria dos trabalhos aplica uma rede de extração de *features* multiescala, variando entre 3 a 4 escalas, com exceção da primeira escala, as outras diminuem a resolução da imagem pela metade. Apesar da imagem ter sua resolução diminuída em 4 vezes (no caso de 3 escalas), as informações da imagem em alta resolução já foram codificadas nos descritores das *features* nas camadas iniciais da rede. Essa abordagem traz um melhor resultado se comparada a utilização das *features* da imagem em tamanho original, como pode ser visto na seção de experimentos do trabalho (YAO; LUO; LI; FANG *et al.*, 2018). A segunda abordagem mais utilizada são as redes do tipo codificadora-decodificadora, ela segue como um complemento das redes de extração de *features* multiescala. Ao invés de retornar o mapa de *features* em uma resolução menor, a rede segue fazendo o processo inverso, o mesmo número de escalas mas agora dobrando a resolução em cada escala. Para que as informações não sejam perdidas entre as escalas, cada escala da etapa de codificação está ligada com sua respectiva escala da etapa de decodificação com uma conexão de passagem, essas conexões de passagem também auxiliam no treinamento da rede (MAO; SHEN; YANG, 2016). Ambas as abordagens podem utilizar apenas o mapa de *features* gerado no final da rede, ou utilizar o mapa de *features* de cada uma das escalas. A última opção é utilizada principalmente em redes que tem a abordagem *coarse-to-fine* como a CasMVSNet (GU *et al.*, 2020) e a UCS-Net (CHENG *et al.*, 2020). Quando se utiliza a abordagem *coarse-to-fine*, o mapa de *features* do nível mais baixo é utilizado para criar um mapa de profundidade de baixa resolução. Com as informações obtidas no mapa de profundidade de baixa resolução, as próximas iteração da rede conseguem reduzir o consumo de memória e tempo de execução.

Duas redes seguem uma abordagem diferente para essa etapa. A rede LDCDE-MVS (CHOI *et al.*, 2018) ao contrário das outras, antes de fazer a extração de *features* das imagens, já distorce elas nas hipóteses de profundidade do algoritmo *plane sweep stereo* (Seção 2.1.2.3). A imagem de referência e cada uma das imagens distorcidas

passa por uma rede de extração de *features* siamesa. A rede CVP-MVSNet (YANG *et al.*, 2020) segue de forma análoga as rede CasMVSNet e UCS-Net na abordagem *coarse-to-fine*, mas ao invés de utilizar mapas de *features* de diferentes escalas, ela cria uma pirâmide de imagens. Cada nível n da pirâmide diminui a resolução da imagem por 2^n. Após a pirâmide ser calculada, cada imagem de cada nível passa por uma rede de extração de *features* multiescala. Algo em comum entre todas as abordagens de redes de extração de *features* é que elas compartilham o peso da rede entre todas as imagens, dessa forma a saída pode ser diretamente comparada.

3.2.2 Geração do *cost volume*

Nesta etapa o algoritmo *plane sweep stereo* (Seção 2.1.2.3) é utilizado. Com os mapas de *features* gerados pela etapa anterior e os parâmetros das câmeras, inicia-se a deformação dos mapas de *features* seguindo as D hipóteses de profundidade (Seção 2.1.2.3). A principal diferença é que como a deformação é aplicada em mapas de *features* ao invés de imagens, é utilizada a interpolação bilinear diferenciável para amostrar os pixeis, dessa forma, a rede pode ser treinada fim-a-fim (YAO; LUO; LI; FANG *et al.*, 2018). Cada mapa de *feature* será deformado D vezes, após isso essas deformações são concatenadas e viram um volume de *features*. Para unir os volumes de *features* é utilizada alguma métrica de custo, usualmente é aplicada a variância. A maioria das redes utiliza essa abordagem diretamente ou com alguma modificação, a seguir alguns trabalhos que se destacam.

A rede DPSNet (IM *et al.*, 2019) ao invés de utilizar uma função de custo como no *plane sweep stereo* clássico, ela concatena as *features* da imagem de referência com as *features* da imagem vizinha atual (o processo gera um volume de *features* para cada imagem vizinha). Esses volumes de *features* passam por um sequência de convoluções para a rede aprender a definir o custo a partir da concatenação das *features*, por fim os *cost volumes* são unidos através de uma média. A rede P-MVSNet (LUO; GUAN; JU; HUANG *et al.*, 2019) inicialmente gera um *cost volume* seguindo o modelo clássico do *plane sweep stereo*, que representa o custo da correspondência de cada pixel. Para aumentar a confiança e precisão, esse *cost volume* passa por uma sub-rede com objetivo de aprender o custo por *patches* (um conjunto de pixeis, no caso uma janela de 3×3), gerando assim um *cost volume* em que cada custo leva em consideração o custo dos vizinhos. A rede AttMVS (LUO; GUAN; JU; WANG *et al.*, 2020) cria o *cost volume* da mesma maneira que a rede P-MVSNet, mas ela otimiza o *cost volume* adicionando a informação contextual da cena através de uma sub-rede.

As redes CasMVSNet e UCS-Net, tentam diminuir a quantidade de memória da rede amostrando de forma mais precisa as hipóteses de profundidade e o intervalo entre a hipótese mínima e máxima. Para isso elas iniciam a rede com uma entrada

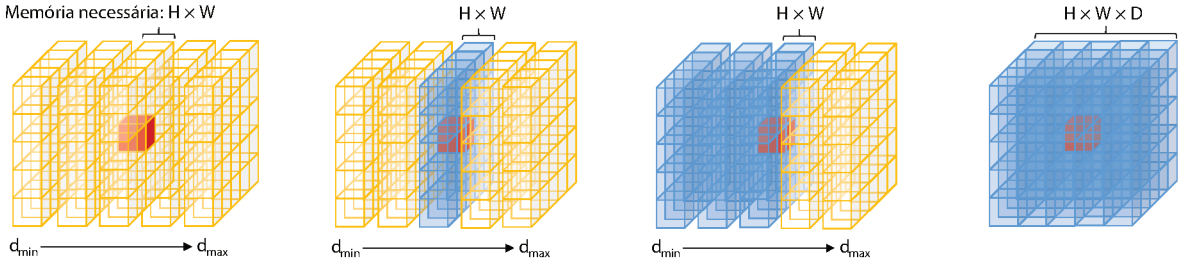
de baixa resolução (*feature map* na CasMVSNet e imagem na UCS-Net) com uma quantidade pequena de hipóteses de profundidade, geram um mapa de profundidade e utilizam essa saída para melhorar a estimação da hipótese de profundidade para a próxima escala de entrada. A UCS-Net se destaca por fazer essa adaptação do intervalo de hipótese de profundidade por pixel ao invés de uma para a entrada inteira. A rede LDCDE-MVS, ao invés de utilizar a variância entre os volumes de *features*, faz a diferença absoluta entre cada um dos volumes de *features* com o volume de *feature* da imagem de referência. Com isso ela gera um *cost volume* para cada imagem vizinha, que passa por uma sub-rede de estimação de confiança. Com todos os *cost volumes* calculados eles são unidos fazendo uma média ponderada considerando a confiança de cada *cost volume*.

3.2.3 Regularização do *cost volume*

A etapa de regularização é utilizada para refinar o *cost volume* que pode estar contaminado por ruídos de superfícies não-lambertianas (superfícies que não refletem a luz igualmente para todas as direções), pela oclusão dos objetos em cena, erros no alinhamento entre as câmeras e outros ruídos que podem diminuir a certeza da estimação de profundidade (YAO; LUO; LI; FANG *et al.*, 2018). Na Figura 20 pode-se observar alguns dos métodos disponíveis para a regularização do *cost volume*. O método mais simples é o ganhador-leva-tudo (Figura 20 (a)), nele a célula que tiver a maior confiança irá definir a profundidade do pixel no mapa de profundidade final, essa abordagem sofre com ruídos, já que quando várias células tem uma confiança próxima, indica uma falta de certeza para a definição da profundidade. O método de regularização espacial (Figura 20 (b)) tenta suavizar o problema do ganhador-leva-tudo filtrando o custo em cada hipótese de plano, agregando informação do contexto espacial para cada estimação de custo. A regularização utilizando redes recorrentes (Figura 20 (c)) segue a ideia do método anterior, mas além de agregar a informação do contexto espacial também agrega a informação na direção da profundidade, levando em consideração a informação de múltiplas hipóteses de profundidade, mas mantendo a mesma quantidade de memória necessária. Por fim tem-se as redes de regularização convolucionais 3D (Figura 20 (d)), essas redes filtram a informação de todo o *cost volume* de uma só vez, trazendo o melhor resultado, mas também o maior consumo de memória, e consequentemente impactando a quantidade de hipóteses de profundidade que pode ser utilizada.

A abordagem mais comum entre os trabalhos é a regularização convolucional 3D utilizando uma rede do tipo codificadora-decodificadora, onde cada uma das escalas da etapa codificadora é ligada com a respectiva escala na etapa decodificadora com uma conexão de passagem. As conexões de passagem são utilizadas para evitar

Figura 20 – Tipos de regularização do *cost volume*, onde *H* é a altura do *cost volume*, *W* largura do *cost volume*, *D* número de hipóteses de planos, d_{min} menor profundidade na hipótese de planos e d_{max} maior profundidade na hipótese de planos.



Fonte – Adaptado de (YAO; LUO; LI; SHEN *et al.*, 2019).

a perda de informações com o processo de codificação-decodificação. A seguir algumas redes que se destacam. A rede DPSNet tem uma abordagem parecida a da rede codificadora-decodificadora, mas ao invés de passar o *cost volume* inteiro pela rede, ele passa o mapa de *features* da imagem de referência concatenado com cada uma das hipóteses de profundidade do *cost volume* na rede codificadora-decodificadora. Isso gera um *cost volume* residual que é somado com o *cost volume* inicial gerando o *cost volume* final. A AttMVS também segue a abordagem da rede codificadora-decodificadora, mas ao invés de utilizar a soma nas conexões de passagem ela utiliza o chamado *ray fusion module* (RFM), dessa forma a rede consegue aprender a melhor forma de unir as diferentes escalas da rede codificadora-decodificadora, ao invés apenas utilizar a soma. As redes R-MVSNet (YAO; LUO; LI; SHEN *et al.*, 2019) e RED-Net (LIU; JI, 2020) são as únicas que utilizam a abordagem da regularização recorrente. O principal impacto dessa abordagem é a redução no consumo de memória. Comparando a rede R-MVSNet com a sua correspondente MVSNet (YAO; LUO; LI; FANG *et al.*, 2018), que usa a regularização com a rede convolucional 3D, a diminuição da memória permite que na R-MVSNet utilize o dobro de hipóteses de profundidade consumindo menos da metade da memória.

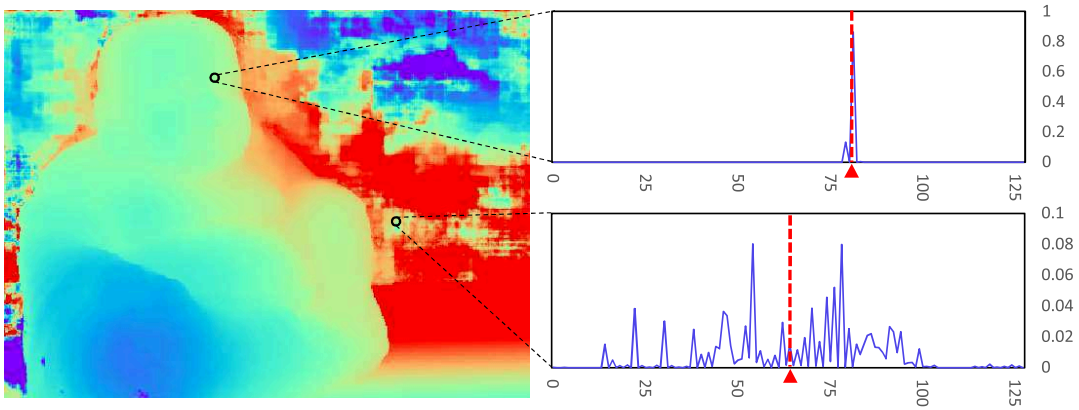
3.2.4 Regressão da profundidade

Essa etapa tem como objetivo transformar o *cost volume* regularizado em um mapa de profundidade. Para isso a maioria dos métodos utiliza duas funções, a *softmax* e a *soft-argmin*. A função *softmax* tem como entrada um conjunto de números e como saída um vetor de probabilidades, essa função é descrita pela equação (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$softmax(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{26}$$

Onde *z* representa o conjunto de números de entrada. Essa função é aplicada no *cost volume* para encontrar qual hipótese de profundidade tem a maior confiança para cada um dos pixeis. Como a saída é um vetor de probabilidades, se não existir um pico bem definido para algum dos pixeis, pode-se afirmar que a estimação de profundidade tem uma baixa confiança. Essa situação pode ser observada nos gráficos da Figura 21 (b), onde o eixo x representa a profundidade e o eixo y a probabilidade. Nota-se que no gráfico superior existe apenas um pico bem definido, aumentando a certeza da estimação, enquanto no gráfico inferior os picos estão espalhados em diferentes profundidades, indicando a baixa confiança.

Figura 21 – Exemplo de um mapa de profundidade e a distribuição de probabilidade de dois pixeis.



(a) Mapa de profundidade (b) Distribuição de probabilidade

Fonte – Adaptado de (YAO; LUO; LI; FANG *et al.*, 2018).

A função *soft-argmin* é a função responsável por reduzir os vetores de probabilidades de cada um dos pixeis a um único valor, seguindo a equação (KENDALL *et al.*, 2017):

$$soft\text{-}argmin = \sum_{d=D_{min}}^{D_{max}} d \times \sigma(-c_d) \tag{27}$$

Onde d é a profundidade, D_{min} é a menor hipótese de profundidade, D_{max} é a maior hipótese de profundidade, σ é a função *softmax* e c_d é o custo do *matching* na profundidade d para o pixel que se está em análise. Essa função é aplicada para cada um dos pixeis. A linha vermelha nos gráficos da Figura 21 (b) representa o valor da função *soft-argmin*. O gráfico inferior, por possuir dois picos separados, a função não consegue estimar com uma alta confiança a profundidade correta.

A rede UCS-Net também utiliza a função *soft-argmin*, mas como o diferencial dessa rede é estimar as hipóteses de profundidade por pixel, cada pixel vai ter um intervalo $[D_{min}, D_{max}]$ diferente. As redes R-MVSNet e RED-Net, por amostrar o in-

verso da profundidade, ao invés de fazer uma amostragem uniforme da profundidade, não podem utilizar a função *soft-agrmin*. Para essas redes, a geração do mapa de profundidade ocorre de maneira diferente na etapa de treinamento e na etapa de teste. Na etapa de treinamento, é aplicada a função *softmax* no *cost volume*, gerando um volume de probabilidades, esse volume de probabilidades é comparado com um volume de ocupação binário gerado através do *ground truth*. Já na etapa de teste, conforme a rede de regularização vai processando cada uma das hipóteses de profundidade do *cost volume*, é aplicado o algoritmo ganhador-leva-tudo. O problema dessa abordagem é que se gera um mapa de profundidade discreto, com baixa precisão, para resolver isso é necessário uma etapa de refinamento. A rede MVSCRF (XUE *et al.*, 2019) também não utiliza a função *soft-argmin*. Ela faz a soma ponderada da saída de todas as escalas da rede decodificadora (da etapa de regularização do *cost volume*) e passa o resultado por uma versão modificada do módulo *Conditional Random Field* (CRF) apresentado por (ZHENG *et al.*, 2015), que tem como saída um mapa de profundidade. O objetivo desse módulo é tratar o problema da definição da profundidade de cada pixel como um problema de *multi-label*, onde cada hipótese de profundidade corresponde a um *label*, isso faz com que pixeis vizinhos tenham uma variação de profundidade suave.

3.2.5 Refinamento

O mapa de profundidade obtido após a regressão de profundidade já é uma possível saída da rede, mas como a etapa de regularização do *cost volume* causa uma super-suavização das bordas dos objetos algumas redes adicionam uma etapa de refinamento (YAO; LUO; LI; FANG *et al.*, 2018). A maioria das redes que tem essa etapa (7 das 8 analisadas) seguem a implementação da MVSNet, nela a imagem de referência é utilizada como uma guia para recuperar as informações das bordas. A imagem de referência é concatenada com o mapa de profundidade e passa por uma sequência de convoluções. A saída da última convolução é somada ao mapa de profundidade inicial, gerando o mapa de profundidade refinado. A única rede que apresenta uma abordagem diferente é a Fast-MVSNet (YU; GAO, 2020). Ela implementa o refinamento através de um algoritmo Gauss-Newton, que tem como objetivo minimizar o erro de reprojeção entre as imagens. Essa abordagem se destaca por utilizar todas as imagens para refinar o mapa de profundidade inicial, não apenas a imagem de referência. Algumas redes aplicam técnicas de pós-processamento, como filtros por erro fotométrico e/ou erro geométrico, mas são algoritmos clássicos que não estão ligados com a rede neural.

3.2.6 Abordagens diferentes

Excepcionalmente, três redes não seguem essa estrutura, a SurfaceNet (JI *et al.*, 2017), a Point-MVSNet (CHEN, R. *et al.*, 2019) e a DeepMVS (HUANG *et al.*, 2018): A SurfaceNet foi a primeira rede criada para resolver o problema do *Multi-View Stereo*. Ela divide a cena em voxels e tem como objetivo afirmar se cada um dos voxels possui ou não uma superfície. Ao invés de utilizar como base o algoritmo *plane sweep stereo*, ela codifica os dados da câmera em cubos chamados de *colored voxel cube* (CVC). Para cada câmera os voxels da cena são projetados na imagem e a cor do pixel atingido na projeção é aplicada no voxel do CVC dessa câmera. O principal problema que surge dessa abordagem é o elevado custo de memória e tempo de execução. Para garantir uma boa resolução de saída deve-se aumentar o número de voxels, e como eles ocupam muito espaço na memória, o algoritmo é obrigado a utilizar uma estratégia dividir para conquistar, fazendo com que o tempo de processamento aumente consideravelmente. A Point-MVSNet (CHEN, R. *et al.*, 2019) se destaca das outras redes por utilizar uma representação de nuvem de pontos ao invés do *cost volume*. Essa rede utiliza como partida o mapa de profundidade de baixa resolução feito pela rede MVSNet (que usa o algoritmo *plane sweep stereo*), e o transforma em uma nuvem de pontos. Cada um dos pontos da nuvem é relacionado com as imagens para entrar no processo de refinamento da rede. A cada iteração do refinamento os pontos são idealmente deslocados para onde seria sua posição real, cada um desses incrementos de posição atualiza o mapa de profundidade inicial até que o processo chegue ao fim. Essa abordagem consegue diminuir o consumo de memória por usar outra representação, mas como o processo é iterativo o tempo acaba sendo consideravelmente maior. A DeepMVS (HUANG *et al.*, 2018), ao contrário das outras redes, já tem como entrada da rede uma imagem de referência e os volumes de *features* das imagens vizinhas (feitos por um algoritmo de *plane sweep stereo*). A rede segue com uma etapa de *patch matching* entre a imagem de referência e cada um dos volumes de *features* das imagens vizinhas, tendo como resultado um volume de *features* com menos canais e um resolução reduzida. Esse volume de *features* passa por uma rede codificadora-decodificadora, que regulariza esse volume e também adiciona informações da imagem de referência. Por fim, os volumes de todas as imagens vizinhas passam por uma camada de *max-pooling*, que seleciona os melhores valores para cada posição, e segue para uma etapa de refinamento. O refinamento é feito utilizando a rede DenseCRF, que tem o mesmo objetivo do módulo CRF da rede MVSCRF, fazer com que os pixeis vizinhos tenham uma profundidade parecida.

As redes *MVS*^{2} (DAI, Y. *et al.*, 2019), MVSNet++ (CHEN, P. *et al.*, 2020) e PruMVS (XIANG *et al.*, 2020), são adaptações da rede MVSNet. A *MVS*^{2} é a única rede com treinamento não-supervisionado dentre os trabalhos selecionados. A princi-

pal diferença em relação a MVSNet é que não se tem um tratamento diferente para a imagem de referência, são selecionadas três imagens vizinhas e o processo de estimação do mapa de profundidade ocorre de maneira simultânea. O processo segue a estrutura da MVSNet até a etapa de refinamento, e para fazer o treinamento é reforçada a consistência fotométrica e geométrica entre os mapas de profundidade encontrados simultaneamente. A MVSNet++ (CHEN, P. *et al.*, 2020) modifica a rede de extração de *features* da MVSNet para uma do tipo codificadora-decodificadora, igual a implementada pelo CasMVSNet, mas a sua principal contribuição está na adição do aprendizado por currículo na etapa de geração do *cost volume*. No aprendizado por currículo, o problema que a rede neural tem que resolver é simplificado e com o passar do treinamento vai aumentando gradativamente de dificuldade. No início do treinamento uma máscara com a profundidade real é disponibilizada para a rede, fazendo com que ela tenha que definir a profundidade de apenas alguns pixeis. Conforme o treinamento evolui, a máscara vai tendo cada vez menos pixeis até que no final, a rede fique responsável por definir a profundidade de todos os pixeis. A PruMVS (XIANG *et al.*, 2020) não faz nenhuma mudança na arquitetura da rede MVSNet, apenas propõe o uso de uma técnica conhecida como *prunning*. Essa técnica consiste em remover as camadas redundantes da rede, com o objetivo de reduzir o consumo de recursos computacionais e o tempo de execução do algoritmo. O trabalho consegue atingir esse objetivo com uma penalidade em precisão e um aumento da completude do mapa de profundidade, quando comparado ao MVSNet original.

Tabela 2 – Artigos selecionados, onde C-D são as redes codificadora-decodificadora e PSS representa o algoritmo *plane sweep stereo*.

Rede	**Extração de *features***	**Geração do *cost volume***	**Regularização do *cost volume***	**Regressão da profundidade**	**Refinamento**
SurfaceNet	-	-	-	-	-
DeepMVS	-	-	-	-	-
LDCDE-MVS	Siamesa	-	C-D	*soft-arming*	-
MVSNet	Multiescala	PSS	C-D	*soft-arming*	Mapa de prof. + features da I_{ref}
DPSNet	Multiescala	PSS, concatenando *features* ao invés da diferença	C-D, cálculo de profundidade residual	*soft-arming*	-
MVS^2	Multiescala	PSS	C-D	*soft-arming*	Mapa de prof. + features da I_{ref}
MVSCRF	C-D	PSS	C-D + Módulo CRF	*soft-arming*	Mapa de prof. + features da I_{ref}
P-MVSNet	C-D, decodificação apenas para a I_{ref}	PSS	C-D	*soft-arming*	Mapa de prof. + features da I_{ref}
Point-MVSNet	-	-	-	-	-
R-MVSNet	Multiescala	PSS	RNN + Módulos GRU	Ganhador-leva-tudo	-
RED-Net	Multiescala	PSS	C-D RNN + Módulos GRU	Ganhador-leva-tudo	-
AttMVS	Multiescala modificada	PSS	C-D + Módulos RFM	*soft-arming*	-
CasMVSNet	C-D	PSS	C-D	*soft-arming*	Mapa de prof. + features da I_{ref}
CVP-MVSNet	Multiescala, com pirâmide de imagens como entrada	PSS	C-D	*soft-arming*	-
UCS-Net	C-D	PSS	C-D	*soft-arming*	-
Fast-MVSNet	Multiescala	PSS	C-D	*soft-arming*	Gauss-Newton
MVSNet++	C-D	PSS, com aprendizado de currículo	C-D, resultado é a soma ponderada das escalas da decodificação	*soft-arming*	Mapa de prof. + features da I_{ref}
PruMVS	Multiescala	PSS	C-D	*soft-arming*	Mapa de prof. + features da I_{ref}

Fonte – Elaborado pelo autor.

4 ABORDAGEM PROPOSTA

Neste capítulo serão apresentados os algoritmos utilizados para modificar as etapas de extração de *features* e geração do *cost volume*. Essas etapas foram escolhidas para o desenvolvimento pois são as que mais se aproximaram do processo clássico da reconstrução 3D. O processo de escolha da rede utilizada como base e as modificações feitas serão descritas na Seção 4.4. O *hardware* utilizado no treinamento será descrito no Capítulo 5.

4.1 CONVOLUÇÃO DEFORMÁVEL

As camadas convolucionais apesar de mostrarem um grande avanço na detecção de características quando comparadas aos detectores de características criados manualmente, como o SIFT, SURF, AKAZE, ainda apresentam uma baixa capacidade de modelar transformações geométricas, visto que elas dependem fortemente de *data augmentation* (geração artificial de dados de treinamento) para criar um modelo genérico. Isso pode ser atribuído ao fato de apesar dela aprender os pesos de cada posição dos filtros, os filtros em si possuem uma estrutura fixa, mas os objetos em cena geralmente são vistos por diversos ângulos, o que modifica sua silhueta. A convolução deformável surge para melhorar a capacidade de modelagem de transformações geométricas das camadas convolucionais, ela adiciona *offsets* 2D na amostragem do *grid* da camada convolucional comum (CHOI *et al.*, 2018) (DAI, J. *et al.*, 2017).

A camada convolucional pode ser dividida em duas etapas, a primeira consiste na amostragem de um *grid* regular R sobre o mapa de características de entrada x. A segunda etapa é a somatória dos valores amostrados pelo *grid* ponderados por w (os pesos que a rede aprende). Para cada pixel p_0 no mapa de características de saída y, tem-se:

$$y(p_0) = \sum_{p_n \in R} w(p_n) \cdot x(p_0 + p_n) \tag{28}$$

sendo p_n os pixeis que estão sobrepostos pelo *grid* R. Para a convolução deformável, algumas alterações devem ser feitas. O *grid* R é modificado com *offsets* $\{\Delta p_n | n = 1, \ldots, N\}$, onde $N = |R|$. Desta forma, a Equação (28) é reescrita:

$$y(p_0) = \sum_{p_n \in R} w(p_n) \cdot x(p_0 + p_n + \Delta p_n) \tag{29}$$

Como Δp_n é tipicamente fracional, a Equação (29) é implementada via interpolação bilinear:

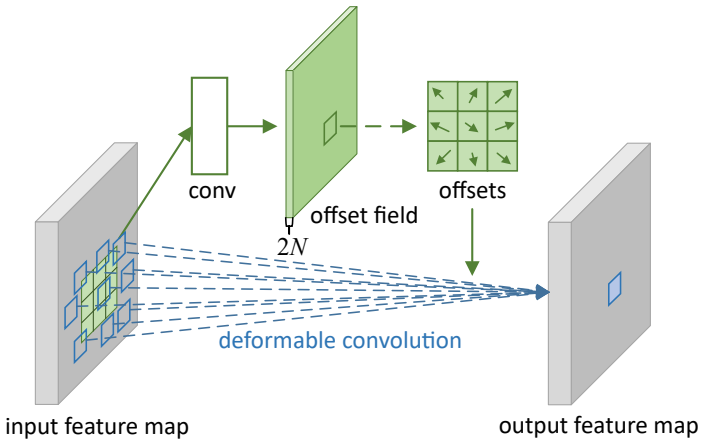
$$x(p) = \sum_{q} G(q, p) \cdot x(q) \tag{30}$$

onde p denota uma posição arbitrária (podendo ser fracional), q enumera todas as posições inteiras do mapa de características de entrada e $G(\cdot,\cdot)$ é a interpolação bilinear. Apesar de G ser bidimensional, ele é separado por dois filtros unidimensionais:

$$G(q,p) = g(q_x,p_x)\cdot g(q_y,p_y) \tag{31}$$

onde $g(a,b) = \max(0,1-|a-b|)$. Na Figura 22 pode-se ver um exemplo de uma convolução deformável 3×3. Os *offsets* são obtidos aplicando uma convolução sobre o mapa de *features* de entrada. Essa convolução tem o mesmo tamanho que a convolução dessa camada (3×3 no caso da Figura 22). O *offset field* tem o mesmo tamanho da mapa de *features* de entrada e possui $2N$ canais, sendo N o número de *offsets* 2D. O treinamento dos *offsets* acontece simultaneamente com o treinamento dos pesos da camada convolucional (DAI, J. *et al.*, 2017).

Figura 22 – Convolução deformável 3×3.



Fonte – Adaptado de (DAI, J. *et al.*, 2017).

4.2 CONVOLUÇÃO DEFORMÁVEL MODULADA

Apesar da melhora na modelagem de transformações geométricas da convolução deformável, os *offsets* aprendidos pela rede podem se estender além da área desejada, influenciando erroneamente os resultados obtidos pela rede. Para resolver isso é apresentada a convolução deformável modulada, ela adiciona um termo que permite a rede definir pesos para cada uma das posições dos filtros, fazendo com que regiões muito distantes do objeto sejam descartadas. Desta forma, a Equação (29) pode ser reescrita (ZHU *et al.*, 2019):

$$y(p_0) = \sum_{p_n\in R} w(p_n)\cdot x(p_0+p_n+\Delta p_n)\cdot \Delta m_n \tag{32}$$

sendo Δm_n o termo utilizado para modular o resultado, variando de 0 a 1. Os valores iniciais de Δp_n e Δm_n são 0 e 0.5 respectivamente. Um exemplo da diferença dos resultados de uma camada convolucional comum, deformável e deformável modulada pode ser visto na Figura 23. O ponto verde nas imagens indicam o centro do filtro da convolução, tanto a convolução comum quanto a deformável padrão consideram alguns detalhes da mesa e do prato, enquanto a deformável modulada foca no objeto selecionado.

Figura 23 – Exemplo da diferença entre as camadas convolucionais, da esquerda pra direita, uma camada convolucional comum, deformável e deformável modulada.

Fonte – Adaptado de (ZHU *et al.*, 2019).

4.3 SIMILARIDADE MÉDIA DE CORRELAÇÃO POR GRUPO

Um dos principais problemas da reconstrução 3D utilizando redes neurais é o consumo de memória. Como pode ser visto na revisão bibliográfica (Capítulo 3), a etapa que mais gera esse problema é a regularização do *cost volume*, principalmente quando utiliza-se redes 3D convolucionais. Para contornar esse problema algumas redes trouxeram diferentes abordagens, seja por um processo iterativo de construção do resultado (Point-MVSNet), gradativa melhora da resolução do resultado (CasMVSNet e CVP-MVSNet) ou com redes recorrentes (R-MVSNet e RED-Net). Apesar dessas abordagens serem válidas, outra opção que pode ter um efeito parecido é a similaridade média de correlação por grupo (SMCG) apresentada por (XU; TAO, 2019). Essa abordagem entra como uma etapa anterior a construção do *cost volume* para reduzir a quantidade de canais no momento da construção do *cost volume*. Dessa maneira, não é necessário criar o *cost volume* com todos os canais para em seguida fazer a redução. Para cada mapa de *features* da imagem de referência F_{ref} e cada mapa de de *features* da i-ésima imagem vizinha na profundidade d_j, $\tilde{F}_{i,j}$, as *features* são divididas igualmente em G grupos na dimensão dos canais, então cria-se um mapa de similaridade $S^g_{i,j}$ da forma:

$$S^g_{i,j} = \frac{1}{H/G}\left\langle F^g_{ref}, \tilde{F}^g_{i,j} \right\rangle \tag{33}$$

sendo H o número de canais originais das *features* (32 no caso do CasMVSNet), $g = 0, \ldots, G-1$, F_{ref}^{g} é a g-ésima *feature* de F_{ref}, $\tilde{F}_{i,j}^{g}$ é a g-ésima *feature* de $\tilde{F}_{i,j}$ e $\langle \cdot, \cdot \rangle$ é o produto interno. Com o mapa de similaridade $S_{i,j}$ calculado para todos os G grupos e todas as D profundidades, pode-se criar um *cost volume* V_i da i-ésima imagem vizinha. Para adaptar esse método para N imagens vizinhas, os *cost volumes* são unidos seguindo a equação:

$$V = \frac{1}{N-1} \sum_{i=1}^{N-1} V_i \tag{34}$$

De acordo com os autores $G = 8$ consegue manter uma boa precisão e reduzir o consumo de memória da rede neural, além de evitar a construção do *cost volume* com mais canais para reduzi-lo em sequência.

4.4 MODIFICAÇÕES PROPOSTAS

Para aplicar os algoritmos propostos anteriormente foi necessário escolher alguma rede como base para as modificações. A rede base deve possuir as etapas de extração de *features* e geração do *cost volume* descritas no capítulo anterior, já que as modificações vão atuar nessas etapas, disponibilizar o código em algum repositório aberto e ser implementada na linguagem Python utilizando a biblioteca PyTorch [1]. A limitação do PyTorch foi imposta já que a implementação da convolução deformável e deformável modulada utiliza essa biblioteca [2]. Na Figura 24 tem-se a seleção das redes encontradas na revisão sistemática (Tabela 2) com base nesses critérios.

Dentre as redes da revisão sistemática, apenas duas cumprem todos os critérios, a DPSNet e a CasMVSNet. Para aplicar os algoritmos propostos anteriormente foi escolhida como base a rede CasMVSNet. Essa rede foi selecionada por possuir uma implementação de código aberto [3] que aplica a similaridade média de correlação por grupo (SMCG) e pelo consumo de memória permitir o treinamento no *hardware* disponível. Os autores da rede DPSNet não disponibilizam o consumo de memória no artigo ou no repositório online, o que dificulta a análise da viabilidade do uso da rede.

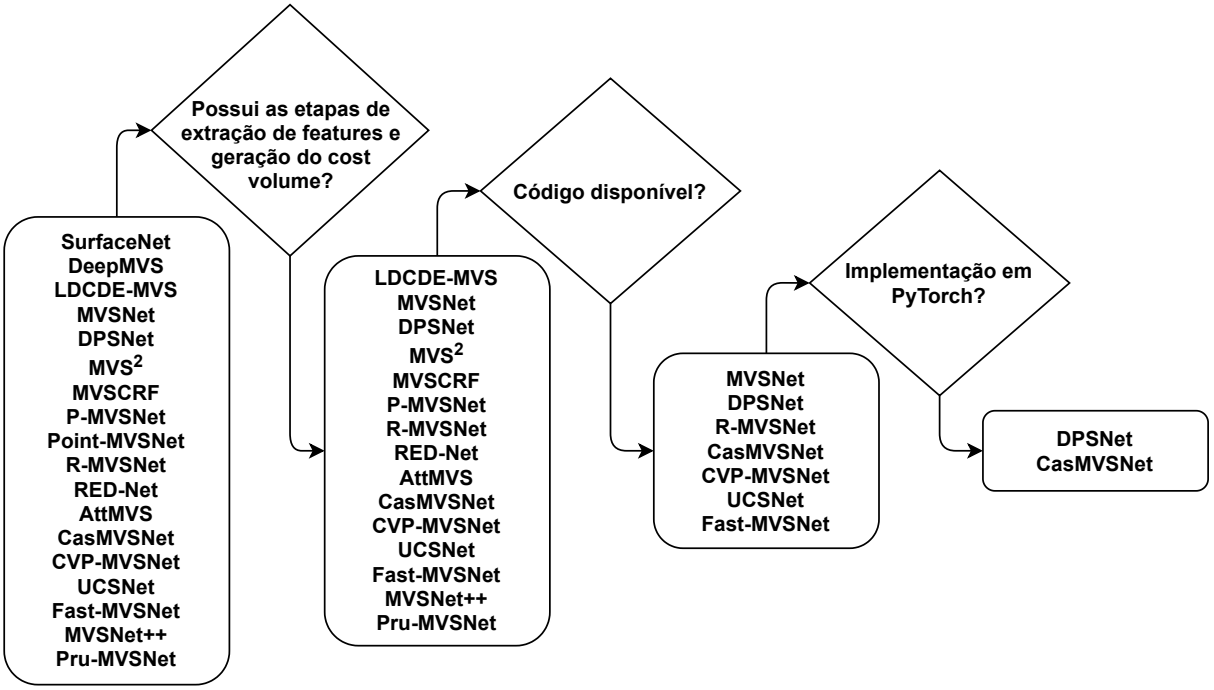
A arquitetura da rede CasMVSNet pode ser visualizada na Figura 25. A principal vantagem dessa rede está no consumo de memória reduzido. Isso é alcançado através da abordagem *coarse-to-fine*, onde a rede cria um mapa de profundidade de alta resolução de maneira iterativa. Como visto na seção anterior, a etapa de geração do *cost volume* (Seção 3.2.2) é baseado no algoritmo *plane sweep stereo* (Seção 2.1.2.3), onde a cena é dividida em D hipóteses de planos. Essas hipóteses de planos são uma forma de amostrar a cena para encontrar onde pode ou não existir uma superfície.

[1] `https://pytorch.org/`

[2] `https://github.com/chengdazhi/Deformable-Convolution-V2-PyTorch/tree/pytorch_1.0.0`

[3] `https://github.com/kwea123/CasMVSNet_pl`

Figura 24 – Processo de seleção da rede base para as modificações.

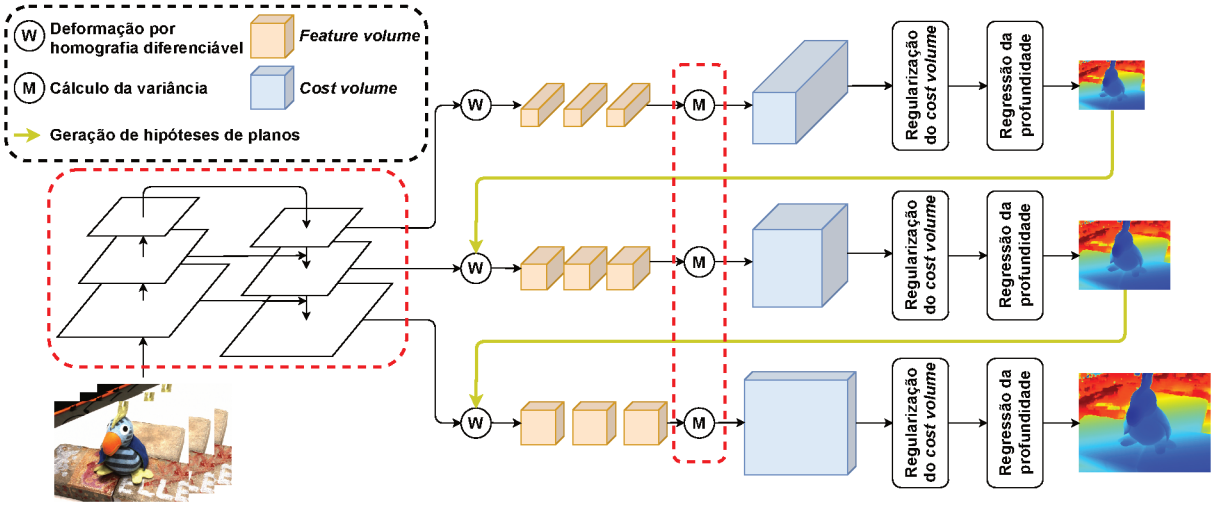


Fonte – Elaborado pelo autor.

Como a princípio não se sabe onde estão as superfícies da cena, é necessário um número maior de hipóteses de planos, aumentando o consumo de memória. Quando se utiliza a abordagem *coarse-to-fine*, o mapa de *features* do nível mais baixo da rede de extração de *features* é utilizado para criar um mapa de profundidade de baixa resolução. Como a entrada tem uma resolução baixa, o impacto de mais hipóteses de planos não é tão grande, permitindo uma maior amostragem da cena. Com as informações obtidas no mapa de profundidade de baixa resolução, pode-se posicionar menos hipóteses de planos na próxima iteração da rede. Dessa maneira, as próximas iterações necessitam de menos hipóteses de planos para gerar um mapa de profundidade de alta resolução com uma boa acurácia.

Para aplicar a convolução deformável e deformável modulada, foi modificada a rede de extração de *features*. A extração de *features* da rede CasMVSNet (destacada com o pontilhado vermelho a esquerda na Figura 25) pode ser vista na Figura 26, ela é do tipo codificadora-decodificadora de 3 níveis. Na etapa de codificação (Conv 0, Conv 1 e Conv 2) estão as convoluções responsáveis por extrair as características da imagem. A primeira camada convolucional da Conv 1 e Conv 2 possuem $stride = 2$ e $padding = 2$ para diminuir a resolução da imagem pela metade (blocos em amarelo claro). Para o processo de decodificação, é retirado um mapa de *features* em cada um dos níveis da rede através de uma camada convolucional com filtro de tamanho 1×1. Para agregar as informações dos níveis mais baixos até o nível mais alto, é aplicada a interpolação bilinear no mapa de *features* do nível $i-1$, para duplicar sua resolução,

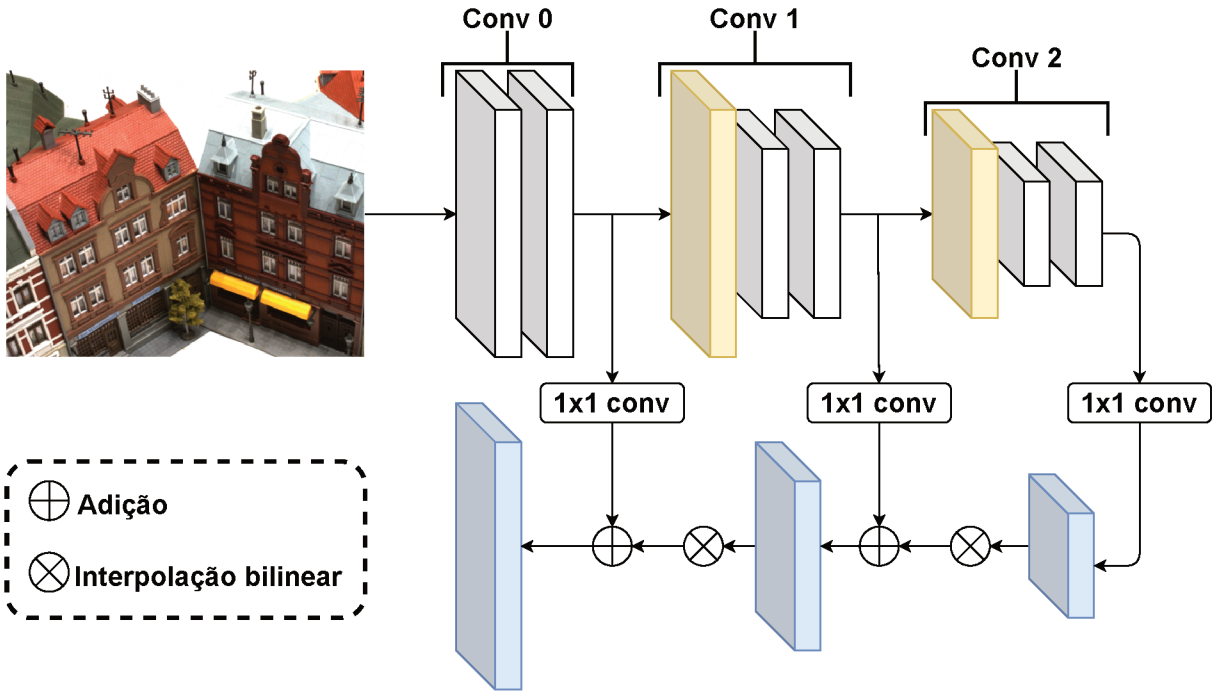
Figura 25 – Arquitetura da rede CasMVSNet.



Fonte – Adaptado de (GU *et al.*, 2020).

que é então somado com o mapa de *features* do nível *i*, esse processo permite que as características locais dos níveis mais baixos sejam levadas até o nível mais alto.

Figura 26 – Extração de *features* da rede CasMVSNet.



Fonte – Elaborado pelo autor.

Para adicionar o SMCG na etapa de geração do *cost volume* do CasMVSNet (destacada com o pontilhado vermelho no centro da Figura 25), basta em aplicar as Equações 33 e 34 antes de iniciar o processo de regularização do *cost volume*. Pela diminuição considerável no consumo de memória (Tabela 4), a SMCG foi mantida em

todos os testes, exceto na rede original, que servirá como base de comparação. A convolução deformável (dconv) foi aplicada na segunda camada da Conv 0; em todas as camadas da Conv 0; todas as camadas da Conv 1; todas as camadas da Conv 2; na segunda camada da Conv 0 e todas as camadas da Conv 2 e 1. Essas mesmas configurações foram aplicadas para a convolução deformável modulada (mdconv), como pode ser visto na Tabela 3. Em alguns testes envolvendo a camada Conv 0, a primeira camada convolucional não foi modificada, essa decisão foi feita para analisar o impacto da modificação dessa camada, já que ela é a primeira camada pela qual a imagem de entrada passa.

Tabela 3 – Configurações utilizadas para os testes. A marcação "X" indica quais convoluções foram modificadas.

Nome	**Conv0**		**Conv1**			**Conv2**		
Original								
SMCG								
dconv_conv0		X						
dconv_todas_conv0	X	X						
dconv_todas_conv1			X	X	X			
dconv_todas_conv2						X	X	X
dconv_todas		X	X	X	X	X	X	X
mdconv_conv0		X						
mdconv_todas_conv0	X	X						
mdconv_todas_conv1			X	X	X			
mdconv_todas_conv2						X	X	X
mdconv_todas		X	X	X	X	X	X	X

Fonte – Elaborado pelo autor.

5 EXPERIMENTOS E RESULTADOS

Neste capítulo serão descritos como os experimentos foram preparados, quais equipamentos foram utilizados e serão apresentados os resultados quantitativos e qualitativos das modificações descritas na Tabela 3. Os treinamentos serão realizados com dois *datasets*, o *DTU* (Seção 5.1.1) e o BlendedMVS (Seção 5.1.3). Os testes quantitativos vão seguir as avaliações disponíveis pelos *datasets DTU* e *Tanks and Temples* (Seção 5.1.2). Os testes qualitativos serão feitos com o auxílio de um *software* (desenvolvido pelo autor) para capturar imagens das diferentes reconstruções de um mesmo ponto de vista. Para realizar os experimentos foram utilizados dois computadores de bancada. O primeiro com as configurações: CPU: Intel Core i7-6700k @ 4.0 GHz; memória RAM: 64 GB DDR4 @ 2133 MHZ; GPU: NVIDIA GTX 1080 8 GB; Sistema operacional: Ubuntu 18.04 LTS. O segundo com as configurações: CPU: Intel Core i9-10900k @ 3.7 GHz; memória RAM: 64 GB DDR4 @ 3200 MHZ; GPU: NVIDIA Quadro RTX 6000 24 GB; Sistema operacional: Ubuntu 18.04 LTS. O primeiro computador foi utilizado para o treinamento com o *dataset* DTU. Os testes quantitativos e os testes de consumo de memória e tempo de estimação foram feitos com esse treinamento, pois todos os parâmetros se mantiveram inalterados. Como o conjunto de imagens de baixa resolução do BlendedMVs possui uma resolução maior que o conjunto do DTU (768×576 contra 640×512), os 8 GB de memória gráfica disponíveis na GTX 1080 não foram suficientes para realizar os testes. Sendo assim, o segundo computador foi utilizado para o treinamento com o *dataset* BlendedMVS. A mudança do computador utilizado no treinamento modificou apenas a memória gráfica e o tempo consumido na etapa do treinamento. Para testar a consistência entre os computadores um treinamento de validação com os mesmos parâmetros foi executado em ambos os computadores, gerando o mesmo resultado. O aluguel de servidores remotos com a capacidade de rodar os treinamentos foi considerado, mas o custo de realizar todos os treinamentos necessários tornou essa opção inviável com o orçamento disponível.

5.1 *DATASETS*

Nesta seção serão apresentados os *datasets* utilizados para o treinamento e para as avaliações das redes neurais, assim como as métricas utilizadas para a avaliação dos resultados.

5.1.1 DTU

O DTU (JENSEN *et al.*, 2014) é um *dataset* feito para a avaliação de métodos *Multi-View Stereo*. Ele consiste em 124 cenas diferentes, sendo que cada cena possui de 49 a 64 imagens com resolução de 1600×1200 capturadas de diferentes posições.

O sistema de captura consiste em um conjunto de luz estruturada (Seção 2.1.2.2) acoplado a um robô que fica em um ambiente fechado com controle de luminosidade. Como o robô não tem uma boa precisão na definição da posição, apenas uma boa repetibilidade, os parâmetros intrínsecos e extrínsecos das câmeras são estimados utilizando uma *toolbox* de calibração de câmera para o MATLAB [1], e o robô apenas repete essa trajetória calibrada. Na Figura 27 tem-se algumas cenas disponíveis no *dataset*.

Figura 27 – Exemplo de cenas do *dataset* DTU.

Fonte – (JENSEN *et al.*, 2014)

Para permitir a comparação dos resultados obtidos com o *ground truth*, o *dataset* disponibiliza um código para calcular a acurácia e a completude (também chamada de *recall*) do resultado. Como o movimento do robô é limitado, nem todos os objetos podem ser capturados de todos os ângulos possíveis, fazendo com que algumas superfícies sofram com a falta de dados. Para eliminar essa variável da avaliação, o *dataset* cria máscaras para cada uma das imagens, eliminando partes das cenas que não representam a qualidade do resultado. Como a saída das redes neurais pode variar, principalmente em densidade de pontos, o DTU faz uma redução no número de pontos dos resultados. Para se aproximar da densidade do *ground truth*, o algoritmo escolhe aleatoriamente pontos na nuvem e elimina qualquer ponto que esteja a menos de 0.2 mm do ponto selecionado.

A acurácia e a completude são calculadas somando as distância entre a nuvem de pontos do *ground truth* e a nuvem a ser testada. Na acurácia, cada ponto da nuvem

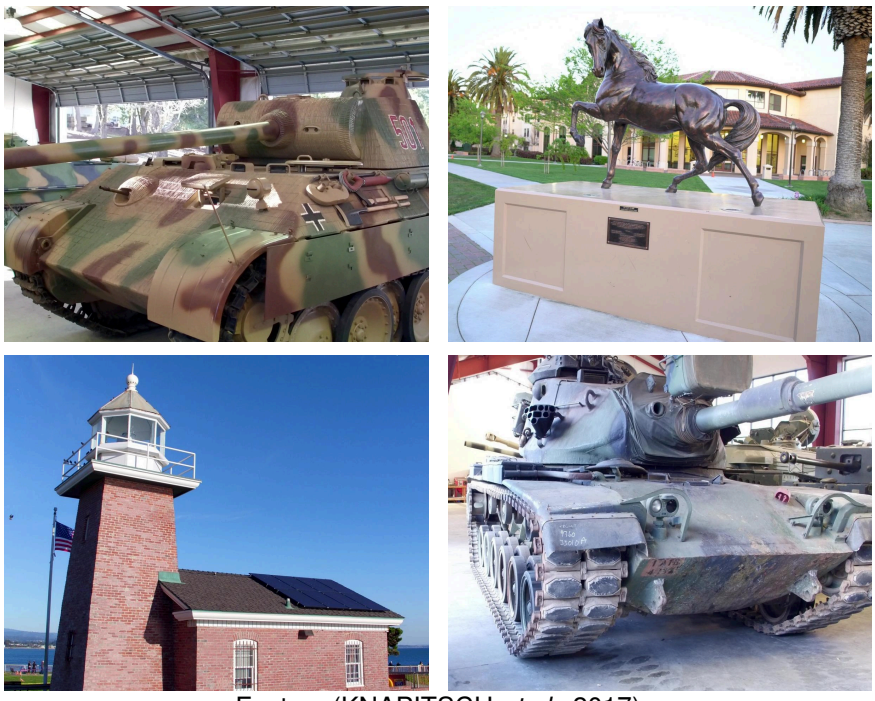
[1] `http://www.vision.caltech.edu/bouguetj/calib_doc/`

de pontos do *ground truth* é visitado e o vizinho mais próximo da nuvem de pontos a ser testada é encontrado. A distância entre os pontos é calculada, e caso seja maior que 20 mm essa distância é considerada um *outlier* e é ignorada. Para a completude o processo é o mesmo, mas a ordem das nuvens de pontos é invertida. Cada ponto da nuvem de pontos a ser testada é visitado e o vizinho mais próximo da nuvem de pontos do *ground truth* é encontrado. A distância entre os pontos é calculada e o mesmo limite de 20 mm se aplica. Um conjunto de cenas padrões é definido para a avaliação, permitindo a comparação entre diferentes trabalhos, que é feita utilizando a média e a mediana da acurácia e completude das cenas.

5.1.2 *Tanks and Temples*

Ao contrário do DTU, o *Tanks and Temples* (KNAPITSCH *et al.*, 2017) é um *dataset* focado em ambientes não controlados. O *dataset* é dividido em dois grupos, o grupo intermediário com cenas de esculturas, veículos e construções, e o grupo avançado com cenas de ambientes internos e ambientes externos, com trajetórias de câmeras mais complexas. A captura dos dados foi feita utilizando uma câmera acoplada a um *gimbal*, e um sensor de varrimento laser 3D industrial, com alcance de 330 metros, que pode operar em ambientes internos e externos com a presença da luz solar. Na Figura 28 tem-se algumas cenas disponíveis no *dataset*.

Figura 28 – Exemplo de cenas do *dataset Tanks and Temples*.



Fonte – (KNAPITSCH *et al.*, 2017)

Assim como o *dataset* DTU, o *Tanks and Temples* também disponibiliza um algoritmo para a avaliação dos resultados, mas ao contrário do DTU, ele não disponibiliza

a posição das câmeras do *ground truth*, sendo necessária uma etapa de alinhamento anterior a comparação dos resultados. O alinhamento parte da posição das câmeras e é refinado através de um algoritmo ICP (CORKE, 2017). Para ajustar a densidade da nuvem de pontos a ser avaliada, é feito uma re-amostragem utilizando um *voxel grid*. Caso mais que um ponto caia no mesmo *voxel*, é feita a média da posição dos pontos. Por fim, é feita a delimitação da nuvem de pontos utilizando o mesmo limite usado no *ground truth*, evitando que pontos que não tem *ground truth* sejam considerados *outliers*. Para comparar as nuvens de pontos são seguidas as métrica de acurácia e completude definidas no *dataset* DTU, mas aqui elas são normalizadas e ficam em um intervalo de $[0,100]$. Para realizar o cálculo de acurácia, é necessário encontrar a menor distância entre o ponto $r \in R$ da nuvem de pontos a ser avaliada até algum ponto $g \in G$ da nuvem de pontos de referência (*ground truth*), esse problema pode ser definido através da seguinte equação (KNAPITSCH *et al.*, 2017):

$$e_{r \to G} = \min_{g \in G} |r - g| \tag{35}$$

onde G é a nuvem de pontos de referência e R a nuvem de pontos a ser avaliada. A acurácia é a média dessa distância para todos os pontos de R (KNAPITSCH *et al.*, 2017):

$$P(d) = \frac{100}{|R|} \sum_{r \in R} [e_{r \to G} < d] \tag{36}$$

onde $P(d)$ é a acurácia a um determinado limite d para *outliers*, e $[\cdot]$ são os colchetes de Iverson (GRAHAM, 1994). De forma análoga, essas equações se repetem para a completude (KNAPITSCH *et al.*, 2017):

$$e_{g \to R} = \min_{r \in R} |g - r| \tag{37}$$

$$R(d) = \frac{100}{|G|} \sum_{g \in G} [e_{g \to R} < d] \tag{38}$$

onde $R(d)$ é a completude (também chamado de *recall*) a um determinado limite d para *outliers*. Após calcular ambas as métricas, elas são unidas em um único indicador, o *F-score*, que é definido por (KNAPITSCH *et al.*, 2017):

$$F(d) = \frac{2P(d)R(d)}{P(d) + R(d)} \tag{39}$$

Esse indicador a um dado limite d é a média harmônica entre a acurácia e completude, ele consegue garantir que ambas métricas tenham igual importância, pois se $P(d) \to 0$ ou $R(d) \to 0$ então $F(d) \to 0$. Um diferencial desse *dataset* é que ele não disponibiliza o *ground truth* de algumas cenas especificamente separadas

para a avaliação. Para conseguir o resultado nesse *dataset* é necessário submeter os resultados para o servidor dos autores do *dataset*. Essa característica dificulta a análise dos resultados pois apenas o *F-score* é disponibilizado pelos autores, não sendo possível acessar as métricas de acurácia e completude.

5.1.3 BlendedMVS

Ao contrário dos outros *datasets*, o BlendedMVS (YAO; LUO; LI; ZHANG *et al.*, 2020) não capturou as próprias cenas, mas selecionou 113 cenas disponíveis na plataforma online Altizure (que não está mais disponível). As cenas variam entre construções, veículos, esculturas e pequenos objetos, contendo de 20 a 1000 imagens por cena, totalizando 17818 imagens. A contribuição desse trabalho está na geração do *ground truth*, cada uma das cenas (modelo 3D gerado pelo Altizure) é renderizada, e para cada uma das câmeras é gerado um mapa de profundidade e uma imagem RGB. Como essa imagem RGB parte de uma cena renderizada, muitos dos reflexos que acontecem naturalmente em uma fotografia não são capturados. Para isso é aplicada uma técnica de *blending*, onde um filtro de passa-baixa é utilizado na imagem original da câmera para recuperar a luz ambiente, e isso é somado com o resultado de um filtro passa-alta na imagem renderizada para realçar as bordas dos objetos, gerando a imagem final. Ao contrário do DTU que usa um robô para repetir a mesma trajetória da câmera, o BlendedMVS não mantém uma trajetória padrão, permitindo uma maior generalização dos pontos de vista observados pela rede neural. Na Figura 29 tem-se algumas cenas do *dataset*. Ao contrário dos outros *datasets*, o BlendedMVS não disponibiliza nenhum algoritmo para a avaliação dos resultados.

5.2 TREINAMENTO

Para o treinamento foram escolhidos os *datasets* DTU e BlendedMVS, já que o *Tanks and Temples* possui apenas 7 cenas com *ground truth*. Como visto no Capítulo 3, as rede neurais tem como saída um mapa de profundidade (com exceção da SurfaceNet), mas o *ground truth* disponibilizado pelo *dataset* DTU está no formato de nuvem de pontos, sendo necessária uma etapa de conversão. Para utilizar esse *dataset* no treinamento, foram utilizados os dados pré-processados disponibilizados pela rede MVSNet, onde a nuvem de pontos passa pelo algoritmo *Screened Poisson* (KAZHDAN; HOPPE, 2013) para gerar uma superfície. Com a superfície pronta, ela é renderizada em cada uma das posições das câmeras da cena, gerando um mapa de profundidade para cada câmera, num processo parecido ao realizado pelo BlendedMVS.

Os treinamentos seguiram os parâmetros padrões da rede CasMVSNet: a duração do treinamento foi de 16 épocas (uma época corresponde a uma vez que todo

Figura 29 – Exemplo de cenas do *dataset* BlendedMVS.

Fonte – (YAO; LUO; LI; ZHANG *et al.*, 2020)

o *dataset* passou pelo processo de inferência –*foward propagation*– e cálculo de erro –*backpropagation*–); o número de hipóteses de planos de profundidade foi de 8, 32 e 48, variando por escala, já que o CasMVSNet gera o mapa de profundidade de maneira iterativa; um multiplicador do intervalo entre as hipótese de plano de profundidade de 1.0, 2.0 e 4.0, variando por escala; 3 imagens RGB de entrada, sendo 1 imagem de referência e 2 vizinhas. Para o *dataset* DTU, o intervalo entre as hipóteses de plano de profundidade foi definido em 2.65 mm e para o BlendedMVS 192.0 mm. Como o BlendedMVS possui cenas de objetos pequenos e de grandes construções, ela tem sua escala normalizada para ficar aproximadamente com os mesmos parâmetros do *dataset* DTU. Para diminuir o consumo de memória durante o treinamento, os *datasets* disponibilizam uma versão com uma resolução menor, as imagens do *dataset* DTU ficam resolução de 640×512, e as imagens do BlendedMVS em 768×576. Para poder comparar a memória entre os treinamentos das diferentes modificações na rede, o *batch size* (número de amostras que vão ser propagadas pela rede por vez) foi definido como 2 para todo o treinamentos com o DTU. No treinamento com o BlendedMVS, que foi feito com o segundo computador (RTX 6000), o *batch size* variou de 4 até 6, dependendo do consumo de memória da rede.

5.3 RESULTADOS

Os resultados foram divididos em duas sub-seções, cada uma referente a um *dataset* utilizado no treinamento. Essa divisão foi feita para se observar o impacto dos

diferentes *datasets* nas avaliações quantitativas, visto que o DTU é um *dataset* que contém apenas cenas de objetos pequenos em um ambiente interno controlado, enquanto o BlendedMVS possui cenas em ambientes não controlados com diferentes escalas. Para avaliar os resultados, foram escolhidos os *datasets* DTU e *Tanks and Temples*, já que o BlendedMVS não disponibiliza uma avaliação quantitativa. De maneira análoga ao início do treinamento, onde as nuvens de pontos tiveram que ser convertidas para mapas de profundidade, agora, para poder fazer a análise dos resultados é necessário transformar os mapas de profundidade em nuvens de pontos. Para isso, é utilizado o algoritmo de fusão de mapas de profundidade (MERRELL *et al.*, 2007) disponibilizado pelo MVSNet.

Ao contrário do treinamento, a resolução das imagens não precisa ser tão restritiva, sendo assim, as imagens do DTU e do *Tanks and Temples* foram redimensionadas para 1152×864. Para comparar os dados de memória e tempo de execução foram utilizados apenas os testes do treinamento com o *dataset* DTU, pois ele foi treinado em apenas um computador com um *batch size* fixo. Como a única mudança na rede entre os dois treinamentos foi o *dataset*, as conclusões feitas continuam válidas para a rede treinada com o BlendedMVS. Nas tabelas das seções a seguir, o melhor resultado estará em negrito enquanto o segundo melhor resultado estará sublinhado. Em caso de empate, ambos os dados serão sublinhados ou negritados.

5.3.1 Rede treinada com o *dataset* DTU

Para avaliar o treinamento foram calculadas as seguintes métricas: erro absoluto, que representa a diferença absoluta entre o mapa de profundidade estimado e o mapa de profundidade do *ground truth*; as acurácias de 1, 2 e 4 mm, representando a porcentagem dos pixeis do mapa de profundidade que tem um erro menor que 1, 2 e 4 mm; o consumo de memória gráfica e o tempo de treinamento, que é o tempo total para treinar a rede durante as 16 épocas. Na Tabela 4 tem-se os resultados do treinamento. As diferenças de erro absoluto e das acurácias não mudaram drasticamente com as diferentes configurações da rede, sendo a maior diferença de erro absoluto de apenas $0.38mm$ $(4.35-3.97)$ e 0.94% $(72.11-71.17)$, 1.11% $(84.73-83.72)$ e 0.76% $(90.97-90.20)$ para as acurácias de 1, 2 e 4 mm, respectivamente. A maior variação a ser observada é no consumo de memória e no tempo de treinamento. O SMCG conseguiu reduzir 1.5 GB de memória se comparado a rede original. Isto justifica ele ter sido mantido nas outras modificações, visto que não seria possível executá-las com os 8.0 GB de memória gráfica disponível na GTX 1080 de outro modo. Em geral, as modificações que estão focadas nas convoluções com uma resolução maior (Conv 0), consomem mais memória e levam mais tempo para concluir o treinamento, e o maior impacto nessas métricas fica com as redes que modificam quase todas as camadas (dconv_todas e mdconv_todas), consumindo quase 1 dia a mais que as

outras modificações.

Tabela 4 – Resultado do treinamento com o *dataset* DTU.

Método	**Erro absoluto (mm)**	**Acurácia 1mm (%)**	**Acurácia 2mm (%)**	**Acurácia 4mm (%)**	**Memória**	**Tempo treinamento**
Original	4.04	71.94	84.32	90.67	7.7 GB	3d 22h
SMCG	4.10	71.41	84.08	90.50	**6.2 GB**	**3d 08h**
dconv_conv0	4.00	72.05	84.46	90.71	6.5 GB	3d 16h
dconv_todas_conv0	4.08	71.24	83.92	90.43	7.0 GB	3d 19h
dconv_todas_conv1	4.08	71.97	84.57	90.87	6.5 GB	3d 22h
dconv_todas_conv2	4.03	71.64	84.17	90.50	6.3 GB	3d 14h
dconv_todas	4.15	72.01	84.53	90.83	6.9 GB	4d 14h
mdconv_conv0	4.17	71.28	83.76	90.21	7.6 GB	3d 18h
mdconv_todas_conv0	4.09	71.45	84.09	90.53	7.0 GB	3d 22h
mdconv_todas_conv1	**3.97**	**72.11**	**84.73**	**90.97**	7.6 GB	4d 03h
mdconv_todas_conv2	4.35	71.17	83.72	90.20	6.5 GB	3d 18h
mdconv_todas	4.13	71.58	84.07	90.50	8.0 GB	4d 23h

Fonte – Elaborado pelo autor.

Tabela 5 – Resultados da estimação com o *dataset* DTU de uma imagem em diferentes resoluções.

	Memória (GB)			**Tempo (s)**		
Método	**1536x1152**	**1152x864**	**640x512**	**1536x1152**	**1152x864**	**640x512**
Original	6.5	4.6	1.9	1.55	0,80	0,26
SMCG	7.4	4.3	1.8	**1.20**	**0,64**	**0,21**
dconv_conv0	6.1	4.3	**1.6**	1.31	0,73	0,24
dconv_todas_conv0	7.7	4.2	1.8	1.34	0,75	0,24
dconv_todas_conv1	6.8	4.8	1.9	1.37	0,76	0,25
dconv_todas_conv2	6.3	4.8	2.0	1.28	0,71	0,23
dconv_todas	**6.0**	**3.6**	**1.6**	1.56	0,83	0,27
mdconv_conv0	7.4	5.0	2.0	1.36	0,73	0,24
mdconv_todas_conv0	7.5	4.8	2.0	1.38	0,76	0,25
mdconv_todas_conv1	7.3	4.3	1.8	1.42	0,79	0,26
mdconv_todas_conv2	7.2	4.7	1.9	1.30	0,72	0,24
mdconv_todas	7.4	5.1	2.0	1.64	0,92	0,29

Fonte – Elaborado pelo autor.

Na Tabela 5 tem-se os resultados do consumo de memória gráfica e tempo para a estimação do mapa de profundidade de apenas uma imagem com resolução de 1536×1152, 1152×864 e 640×512. A imagem utilizada é um exemplo, as conclusões podem ser estendidas para outras imagens se seguidas as resoluções definidas. Comparando os valores da estimação com o treinamento, pode-se ver que a memória não é proporcional, ou seja, o menor consumo de memória no treinamento não garante o menor consumo de memória na estimação. Isso acontece pois o processo de cálculo de erro também consome memória, e as convoluções deformáveis, além de aprenderem os pesos dos filtros, também aprendem a posição de cada amostra do filtro e no

caso da convolução deformável modulada, os peso dessas amostras. O tempo pode ser relacionado, visto que os melhores tempos em ambas as tabelas foram feitos pelas mesmas configurações.

Comparando os resultados das resoluções 1152×864 e 640×512, pode-se perceber que a memória seguiu uma tendência de descida parecida e o tempo seguiu um comportamento parecido ao observado no treinamento. Entre as resoluções 1536×1152 e 1152×864 existiu uma diferença mais notável, a SMCG falhou em reduzir o consumo de memória, quando comparado a rede original, o que pode ter prejudicado os outros métodos, visto que todos implementam o SMCG. O consumo de memória teve um aumento maior nas redes que modificam as convoluções com uma resolução maior (Conv 0), com destaque para a *dconv_todas_conv*0, que nas outras resoluções teve o segundo menor consumo de memória.

Os resultados quantitativos do *dataset* DTU na resolução 1152×864 são apresentados na Tabela 6, onde a acurácia e completude são os resultados obtidos pelo algoritmo do *dataset* DTU e a última coluna da tabela é a média entre essas duas métricas, em todas as métricas, o menor número é o melhor. A acurácia das redes com a convolução deformável ficou igual ou piorou em relação a rede original. A completude teve uma maior diferença, fazendo com que a média melhorasse na maioria dos casos. Apesar de existir uma diferença entre os métodos, ela não é muito grande. Em geral essa diferença da completude pode ser vista nas extremidades das nuvens de pontos.

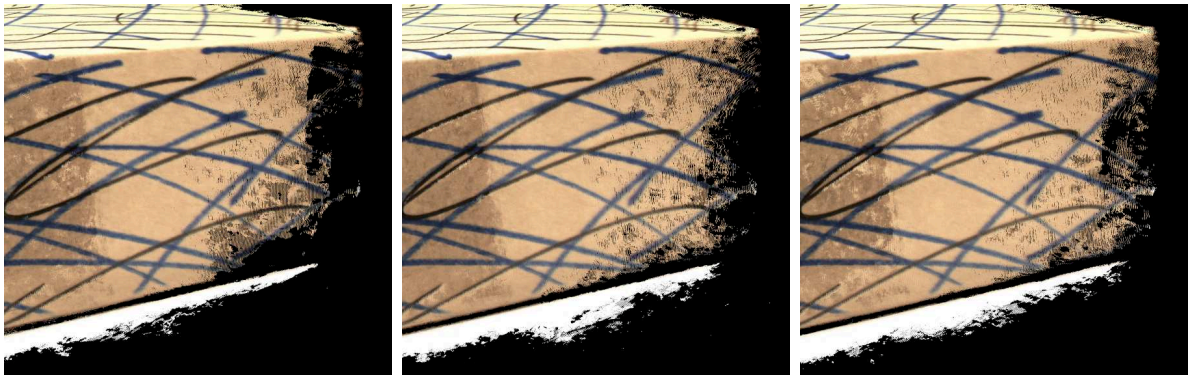
Tabela 6 – Resultados quantitativos do DTU para as redes treinadas com o DTU.

Método	**Acurácia (mm)**	**Completude (mm)**	**Média (mm)**
Original	**0,348**	0,363	0,355
SMCG	**0,348**	0,344	0,346
dconv_conv0	0,349	0,340	**0,345**
dconv_todas_conv0	0,356	0,352	0,354
dconv_todas_conv1	0,360	**0,335**	0,348
dconv_todas_conv2	0,351	0,345	0,348
dconv_todas	**0,348**	0,350	0,349
mdconv_conv0	0,350	0,346	0,348
mdconv_todas_conv0	0,350	0,345	0,347
mdconv_todas_conv1	0,355	0,357	0,356
mdconv_todas_conv2	0,349	0,352	0,351
mdconv_todas	0,359	0,342	0,350

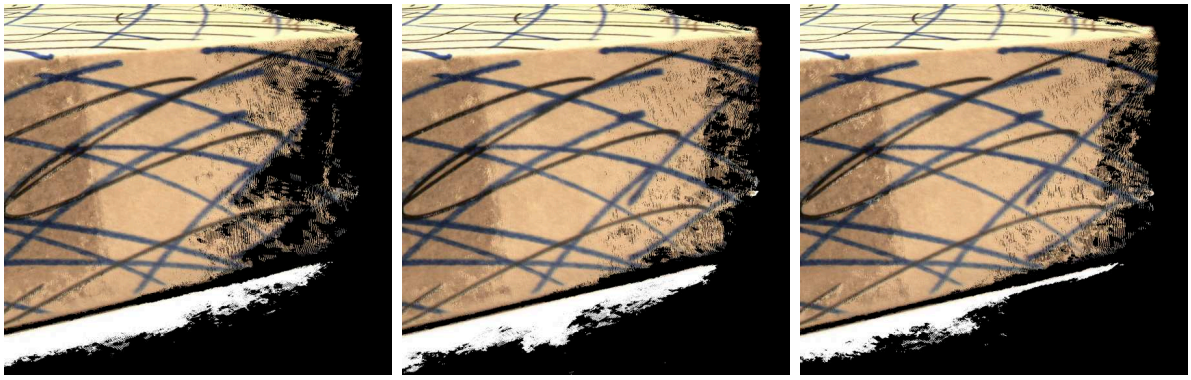
Fonte – Elaborado pelo autor.

Nas Figuras 30 e 31 pode-se observar como as diferentes redes impactam as nuvens de pontos das cenas 10 e 32, respectivamente. A cena 10 consiste em uma caixa de papelão com marcações desenhadas enquanto a cena 32 consiste em um conjunto de alimentos em uma mesa. Ambas as cenas podem ser observadas na Figura 27.

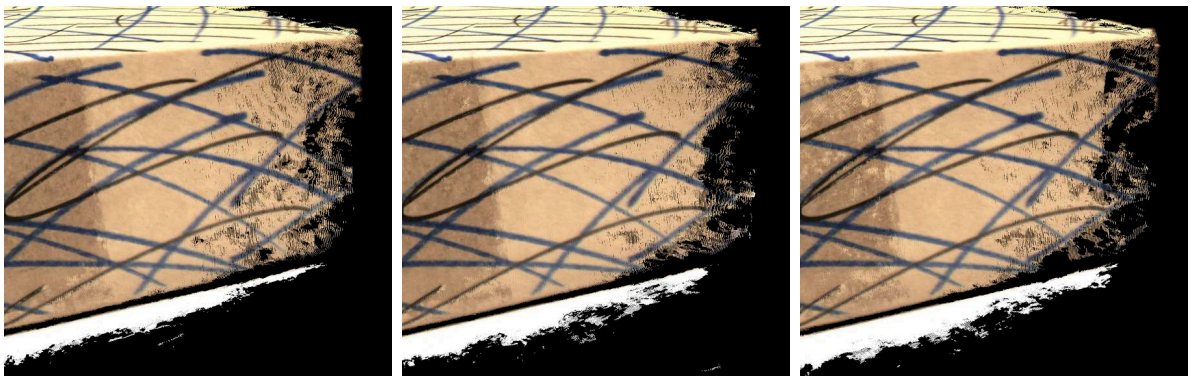
Figura 30 – Nuvens de pontos da cena 10 do *dataset* DTU.



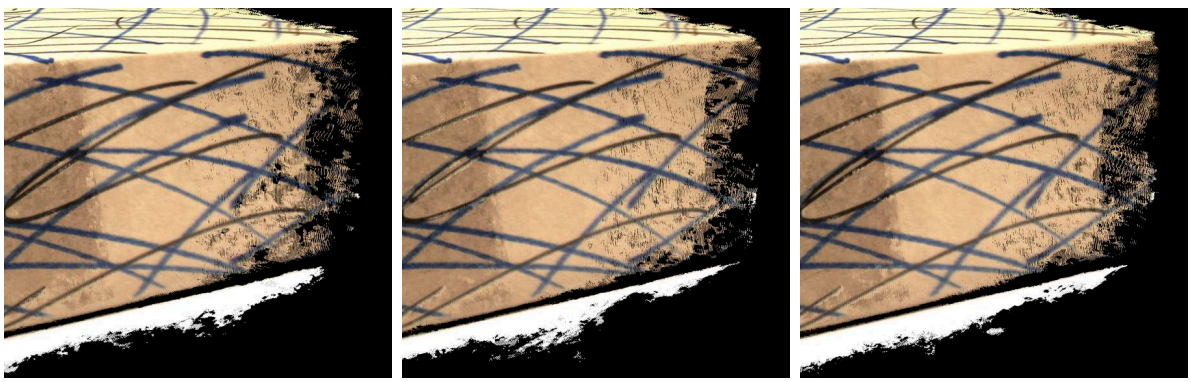
(a) Original. (b) SMCG. (c) dconv_conv0.



(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.



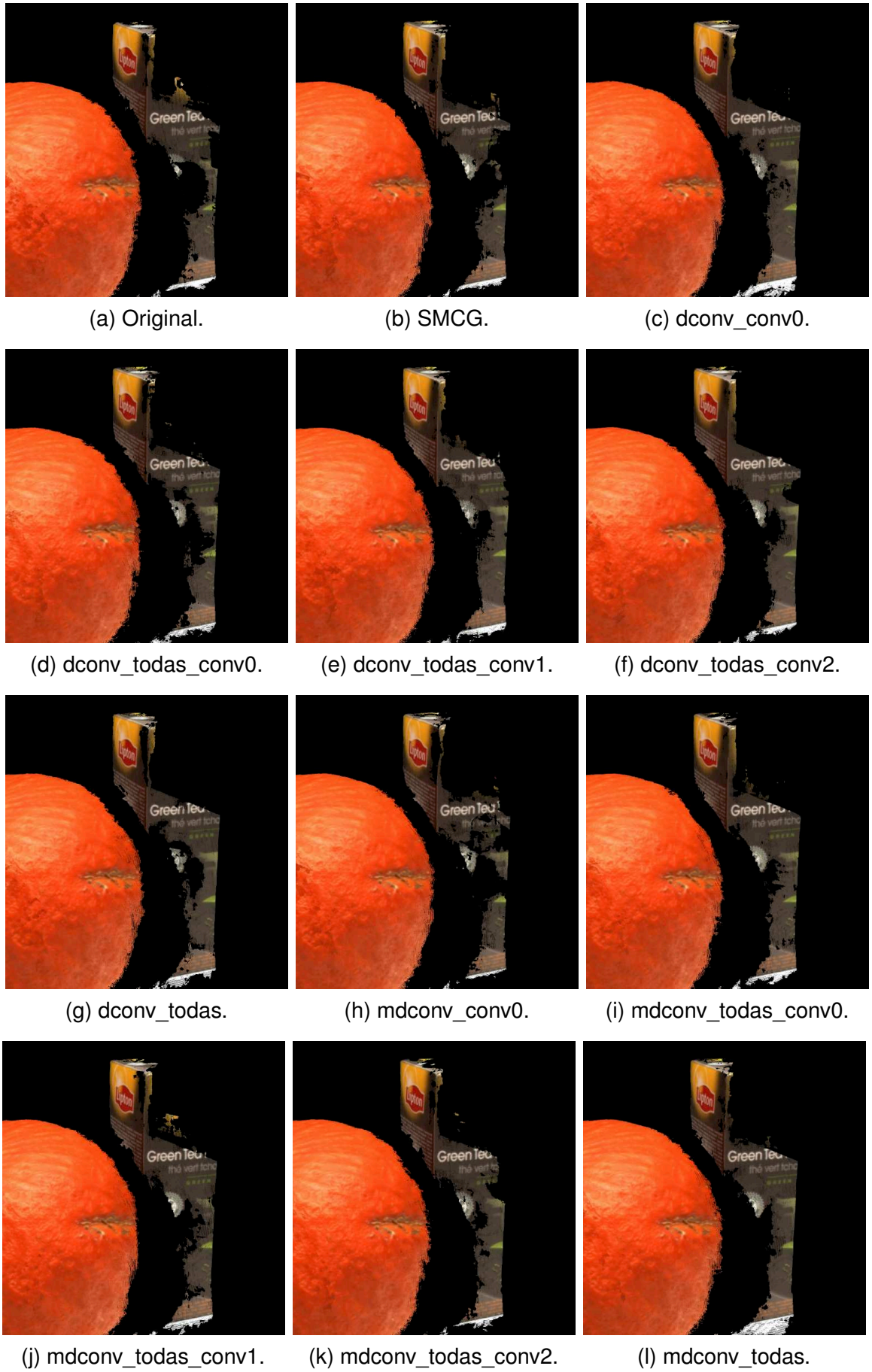
(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.



(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

Figura 31 – Nuvens de pontos da cena 32 do *dataset* DTU.



(a) Original. (b) SMCG. (c) dconv_conv0.

(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.

(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.

(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

A cena 10 (Figura 30) consiste em uma caixa de papelão com marcações desenhadas. Pode-se perceber a dificuldade da rede original em reconstruir as bordas, isso se deve por ela não ter uma boa delimitação das bordas dos objetos, e esses pontos se tornarem *outliers*. A rede SMCG consegue melhorar a área reconstruída, mas ainda fica atrás das outras abordagens. Para as outras modificações, o pior resultado foi a *dconv_todas_conv*0 criando falhas que não estavam presentes nem na rede original. As outras redes seguiram um padrão de melhor para pior da *dconv_conv*0, *dconv_todas_conv*1, *dconv_todas_conv*2 e *dconv_todas*. Esse padrão se repetiu de maneira análoga para as redes com a *mdconv*. A cena 32 é um conjunto de frutas e alimentos em uma mesa, e na Figura 31 fica em destaque uma caixa de suco. Ao contrário da cena anterior, a rede SMCG tem um resultado pior que a rede original, enquanto as modificações com a *dconv* e *mdconv* conseguem recuperar as partes perdidas pelo SMCG e reconstruir outras partes perdidas pela rede original, lembrando que as modificações da rede SMCG também estão nas redes *dconv* e *mdconv*.

Apesar de não terem sido geradas as nuvens de pontos de todas as cenas nas resoluções 1536×1152 e 640×512, na Figura 32 pode-se ver três nuvens de pontos da cena 4 geradas pela rede *dconv_conv*0, onde cada nuvem foi gerada com uma resolução de imagem de entrada diferente. A rede *dconv_conv*0 foi escolhida por ter a melhor média em relação aos outros métodos (Tabela 6). Existem algumas diferenças de qualidade, as extremidades da nuvem de pontos gerada com as imagens 640×512 são menos densas, e existe uma quantidade maior de ruídos (pontos pretos na cabeça da pelúcia), mas a principal diferença entre elas está no número de pontos. A nuvem gerada com a resolução 1536×1152 possui 37.7 milhões de pontos, a nuvem da resolução 1152×864 possui 23 milhões de pontos e a nuvem gerada com a resolução 640×512 tem apenas 7.5 milhões de pontos. Essa diferença na quantidade de pontos não indica necessariamente a qualidade do resultado.

Para uma comparação quantitativa, apenas essa cena foi levada em consideração, gerando uma acurácia média de $0{,}309$, $0{,}265$ e $0{,}276$, e uma completude de $0{,}396$, $0{,}367$ e $0{,}433$ para a resolução 1536×1152, 1152×864 e 640×512, respectivamente. Como pode-se observar, apesar de possui mais pontos, a resolução 1536×1152 acabou perdendo em acurácia para as resoluções mais baixas e em completude para a resolução 1152×864, principalmente por reconstruir uma porção menor da mesa (parte branca na frente da base). Como a qualidade da nuvem gerada com a resolução 640×512 não foi tão impactada, essa resolução se torna mais interessante para algumas aplicações, pois economiza memória, tempo de processamento e espaço de armazenamento. Para as aplicações que necessitam passar pela etapa de geração de superfície, a quantidade de pontos gerada na resolução 640×512 é mais que suficiente para descrever as superfícies da cena, não justificando o gasto extra de memória e tempo das resoluções maiores. Utilizando como exemplo o método de geração de

Figura 32 – Nuvens de pontos da cena 4 do *dataset* DTU geradas pela rede *dconv_conv*0 com diferentes resoluções da imagem de entrada.

(a) Resolução 1536 × 1152. (b) Resolução 1152 × 864. (c) Resolução 640 × 512.

Fonte – Elaborado pelo autor.

Figura 33 – Superfícies criadas pelo *Poisson* a partir nuvens de pontos geradas com diferentes resoluções.

(a) Resolução 1536 × 1152. (b) Resolução 1152 × 864. (c) Resolução 640 × 512.

Fonte – Elaborado pelo autor.

superfície *Screened Poisson* (KAZHDAN; HOPPE, 2013), os resultados podem ficar muito parecidos, como pode ser visto na Figura 33. A diferença mais notável entre as superfícies fica na qualidade da cor das faces, já que são os pontos que amostram a cor das imagens, com mais pontos, o resultado final fica mais próximo do real. Caso se utilize o processo de texturização, a resolução 640 × 512 ainda é mais indicada, pois o menor número de faces fará que o processo aconteça mais rapidamente, e como a superfície irá receber a textura, não importam as cores das faces.

Os resultados quantitativos do *dataset Tanks and Temples* são apresentados na Tabela 7, quanto mais próximo de 100 melhor é o resultado. Como o processo é feito no servidor dos autores do *dataset*, não são todas as informações que são

disponibilizadas, apenas o *F-score* de cada cena é retornado no processo. A falta das outras métricas compromete a análise, pois como visto no dados quantitativos do *dataset* DTU, uma mesma média pode vir de resultados de acurácia e completude diferentes, não ficando claro qual métrica a mudança da estrutura da rede impactou.

Tabela 7 – Resultados quantitativos do *Tanks and Temples* para as redes treinadas com o DTU.

Método	**Family**	**Francis**	**Horse**	**Light-house**	**M60**	**Panther**	**Play-ground**	**Train**	**Média**
Original	74,23	51,75	50,11	52,30	58,86	54,07	54,82	50,51	55,83
SMCG	73,45	51,74	49,57	50,87	58,72	53,42	55,93	49,65	55,42
dconv_conv0	73,84	52,70	48,46	52,17	58,09	53,22	55,80	50,81	55,64
dconv_todas_conv0	73,95	**52,96**	49,09	50,91	58,42	**54,72**	56,72	49,88	55,83
dconv_todas_conv1	73,00	51,61	48,41	51,77	58,16	52,97	55,79	50,12	55,23
dconv_todas_conv2	73,84	52,40	**50,35**	52,08	59,08	53,44	55,92	49,46	55,82
dconv_todas	73,37	51,39	49,49	**52,55**	59,45	54,18	56,56	50,24	**55,90**
mdconv_conv0	73,79	51,45	47,57	50,20	58,89	53,77	56,24	50,16	55,26
mdconv_todas_conv0	72,83	52,87	44,46	51,41	58,53	53,10	55,53	49,77	54,81
mdconv_todas_conv1	73,30	51,27	47,82	52,15	58,70	54,19	**57,12**	49,32	55,48
mdconv_todas_conv2	73,15	49,99	48,19	49,62	58,12	53,10	55,59	48,52	54,54
mdconv_todas	**74,27**	51,56	49,79	50,93	**59,92**	53,31	56,14	**50,83**	55,84

Fonte – Elaborado pelo autor.

Figura 34 – Nuvens de pontos do *Playground* do *dataset Tanks and Temples*.

(a) Rede original.

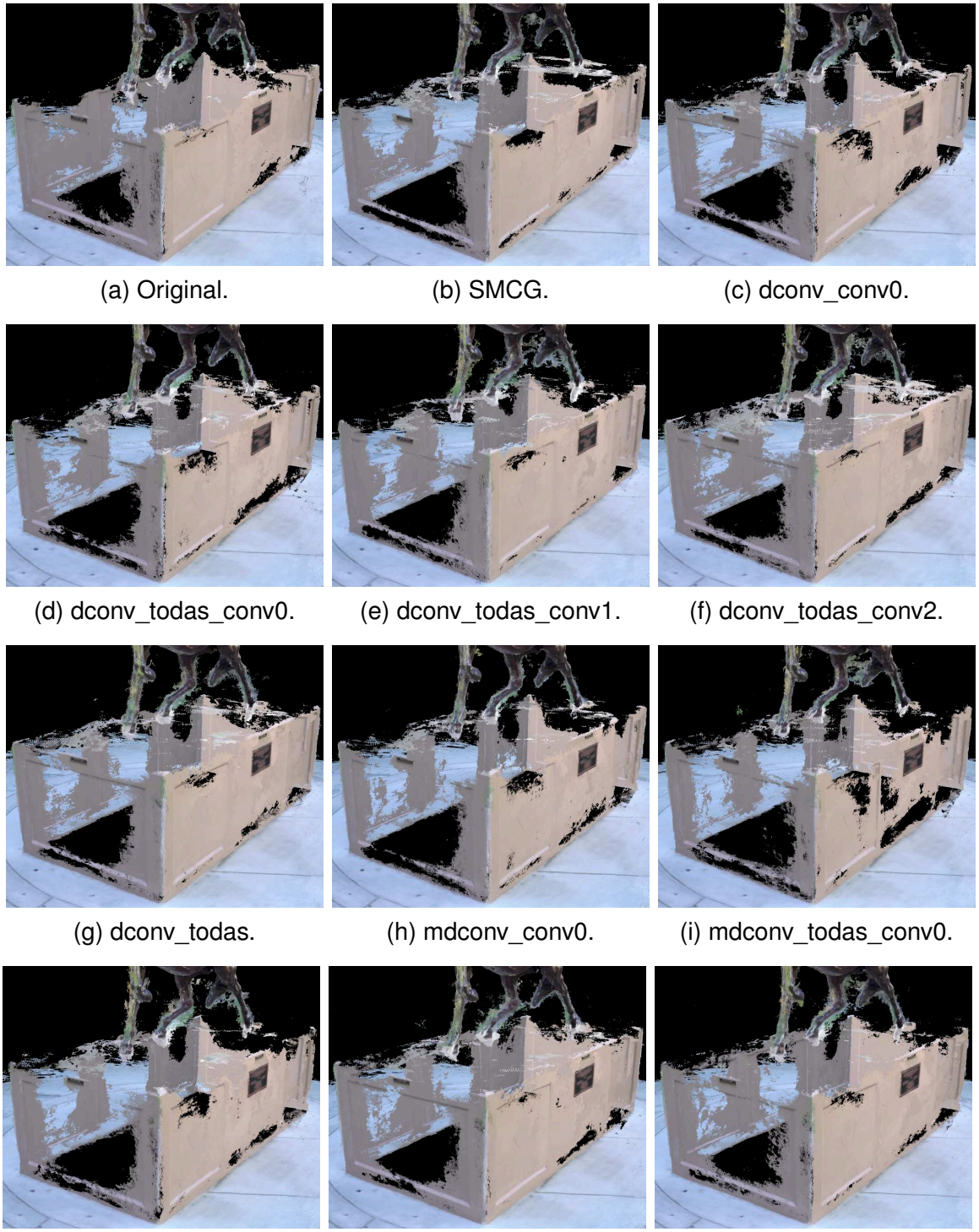
(b) Rede *mdcvon_todas_conv*1.

(c) Imagem de referência.

Fonte – Elaborado pelo autor.

Tendo como referência a rede original, a maioria das cenas ficaram piores nas redes modificadas, com exceção de alguns casos. O único padrão que se mantém é o resultado do *Playground*, que é melhor em todas as modificações. Isso acontece pois essa cena tem várias estruturas finas e complexas, como os balanços e os suportes dos brinquedos, fazendo com que as convoluções deformáveis utilizem sua capacidade de adaptação para amostrar de maneira mais precisa essas estruturas. Na Figura 34 pode-se ver esse efeito comparando a rede original e a *mdcvon_todas_conv*1. Tendo como referência a rede SMCG, todas as redes modificadas tiveram uma pontuação menor no *Horse*, com exceção da *dconv_todas_conv*2 (que obteve o melhor resultado) e da *mdconv_todas*. O principal motivo dessa diferença ficou na completude da base abaixo do cavalo, como pode ser observado na Figura 35. O contrário aconteceu na *Lighthouse*, todas as redes modificadas foram melhores, com exceção da *mdconv_conv*0 e da *mdconv_todas_conv*2, na Figura 36 pode-se ver um comparativo dos resultados. Apesar do resultado ter melhorado com as redes modificadas, ainda exite uma grande quantidade de ruído, principalmente nas áreas que fazem fronteira com o céu. Isso pode ser explicado pelo fato do *dataset* DTU ter apenas cenas capturadas em um ambiente interno e controlado, fazendo com que as redes não aprendessem como interpretar o céu nas imagens, que deve ser tratado como um elemento no infinito que não possui profundidade válida na escala do objeto. Analisando a média, as diferenças não foram expressivas, a rede *dconv_todas* foi a melhor, mas por uma margem pequena. Uma imagem de referência da cena do *Horse* e da *Lighthouse* podem ser observadas na Figura 28.

Figura 35 – Nuvens de pontos do *Horse* do *dataset Tanks and Temples*.



(a) Original. (b) SMCG. (c) dconv_conv0.

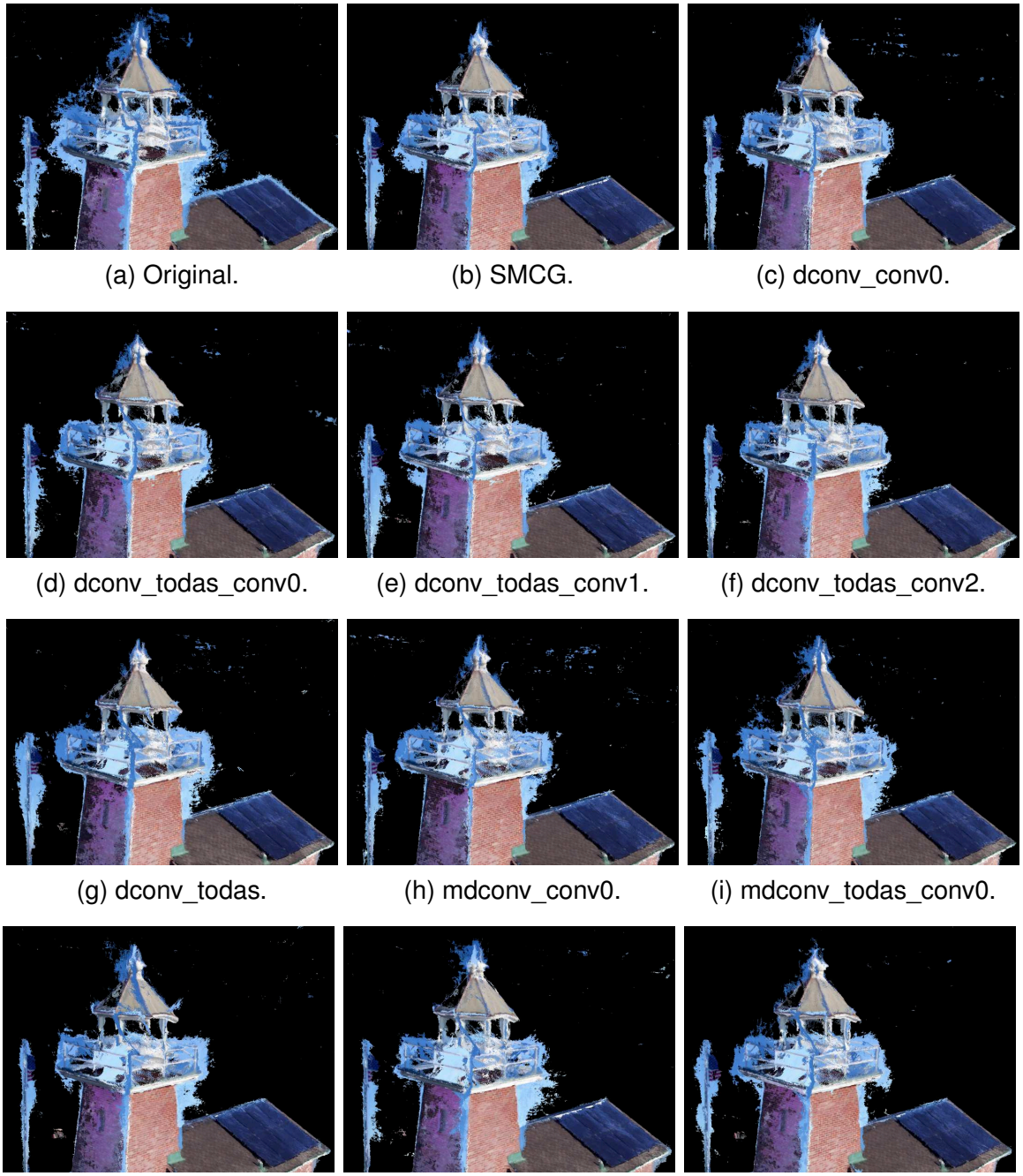
(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.

(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.

(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

Figura 36 – Nuvens de pontos da *Lighthouse* do *dataset Tanks and Temples*.



(a) Original. (b) SMCG. (c) dconv_conv0.

(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.

(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.

(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

5.3.2 Rede treinada com o *dataset* BlendedMVS

Os resultados quantitativos das redes treinadas com o *dataset* BlendedMVS e avaliadas com o *dataset* DTU são apresentados na Tabela 8. Se comparadas diretamente as métricas dos testes quantitativos do DTU, das redes treinadas com o DTU e das redes treinadas com o BlendedMVS, as redes treinadas com o próprio DTU se saem melhor. Isto já era previsto, pois mesmo não sendo treinadas com as cenas usa-

das na avaliação, as cenas do treino estão no mesmo ambiente controlado. A principal diferença entre os treinamentos fica na melhora das métricas das redes modificadas com as convoluções deformáveis e deformáveis moduladas. Além de melhorar a completude, o treinamento do BlendedMVS tornou o resultado mais consistente e com maior acurácia do que as redes original e SMCG. Como a melhora foi em ambas as métricas, a média das redes com a convolução deformável e deformável modulada ficou melhor do que as redes original e SMCG em todas as configurações, com uma margem significativamente maior que a do treinamento anterior.

Tabela 8 – Resultados quantitativos do DTU para as redes treinadas com o BlendedMVS.

Método	**Acurácia (mm)**	**Completude (mm)**	**Média (mm)**
Original	0,398	0,473	0,436
SMCG	0,376	0,480	0,428
dconv_conv0	0,402	0,432	0,417
dconv_todas_conv0	0,388	**0,425**	**0,406**
dconv_todas_conv1	0,388	0,441	0,415
dconv_todas_conv2	**0,365**	0,467	0,416
dconv_todas	0,374	0,450	0,412
mdconv_conv0	0,378	0,467	0,422
mdconv_todas_conv0	0,385	0,444	0,414
mdconv_todas_conv1	0,393	0,441	0,417
mdconv_todas_conv2	0,392	0,459	0,425
mdconv_todas	0,395	0,437	0,416

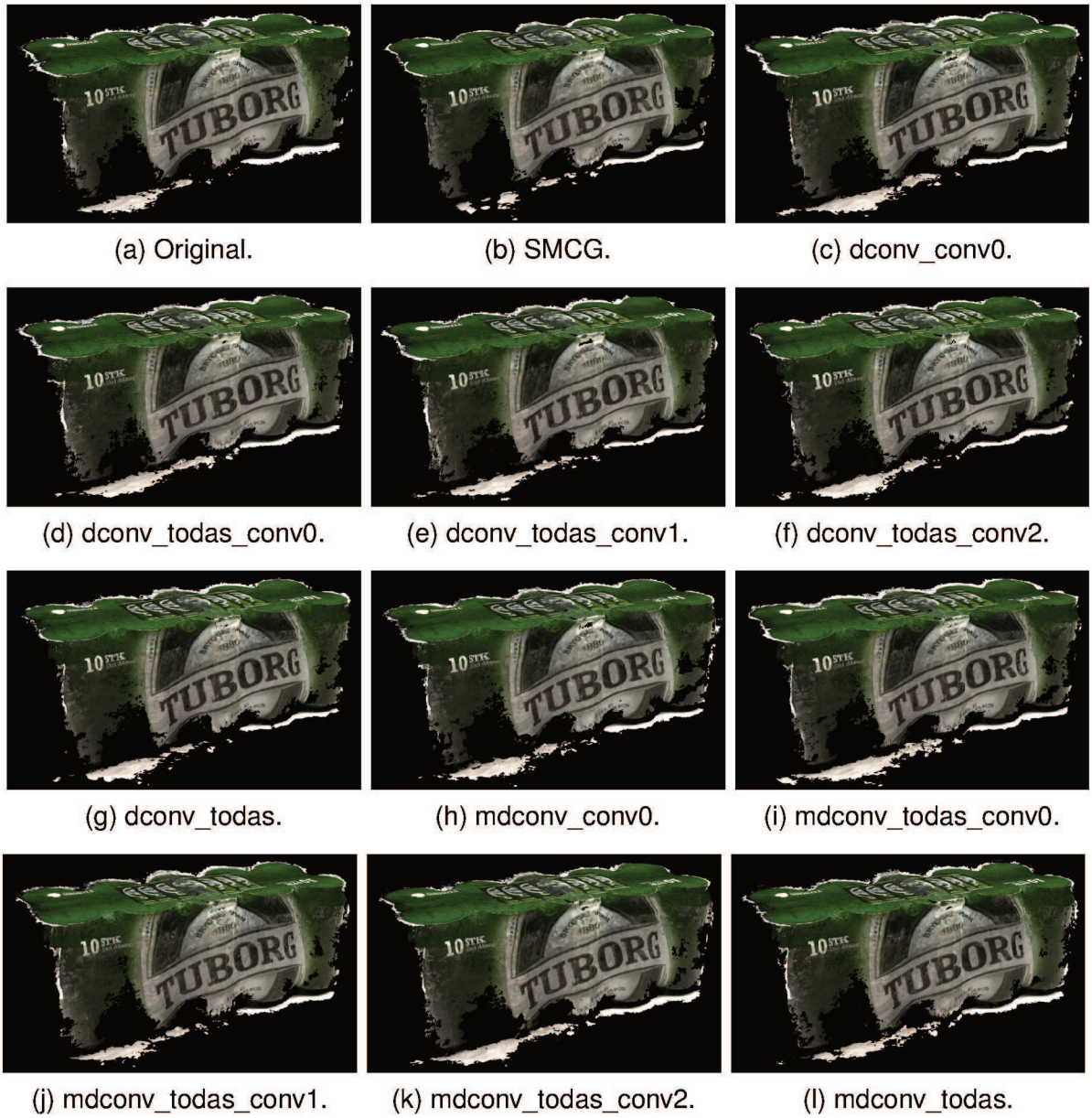
Fonte – Elaborado pelo autor.

Partindo para a análise qualitativa, as melhoras mais visíveis aconteceram na completude das regiões com baixa textura. Nas Figuras 37 e 38 tem-se os exemplos das cenas 12 e 13, respectivamente. Uma imagem de referência de ambas as cenas pode ser observada na Figura 27. A cena 12 é particularmente interessante pois a embalagem do *pack* de cerveja é levemente reflexiva e apresenta variações de áreas com e sem textura. A rede original falhou em algumas partes, mas a completude do objeto não ficou ruim. O principal destaque é a dificuldade da rede SMCG em manter o resultado alcançado pela rede original, tendo uma quantidade maior de falhas principalmente nas regiões da embalagem sem texto e na mesa. Mesmo as redes convolucionais deformáveis e deformáveis moduladas tendo como base a rede SMCG, elas conseguiram recuperar as partes perdidas e completar outras partes que a rede original falhou.

A cena 13 é a outra face da caixa de papelão da cena 10 (Figura 30), com uma imagem colada nessa face. Nessa cena, duas regiões se destacam, a parte vermelha da roupa do homem e a parte branca no canto direito da imagem. A parte vermelha da roupa segue o mesmo comportamento da cena anterior. A rede SMCG diminui a completude se comparada a rede original. As modificações com as convoluções

deformável e deformável modulada recuperam o que foi perdido pela SMCG e em alguns casos tem uma completude maior (*dconv_todas_conv*1). Na parte branca, a ordem entre a rede original e a SMCG se inverte, agora a SMCG tem uma completude maior e mais contínua se comparada a rede original. As modificações com as convoluções deformável e deformável modulada não tem um resultado uniforme. As convoluções deformáveis ficaram piores principalmente nas redes que modificam a conv 2 (*dcvon_todas_conv2* e *dconv_todas*), já as convoluções deformáveis moduladas tiveram um resultado mais próximo da SMCG.

Figura 37 – Nuvens de pontos da cena 12 do *dataset* DTU.



(a) Original. (b) SMCG. (c) dconv_conv0.

(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.

(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.

(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

Figura 38 – Nuvens de pontos da cena 13 do *dataset* DTU.

(a) Original. (b) SMCG. (c) dconv_conv0.

(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.

(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.

(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

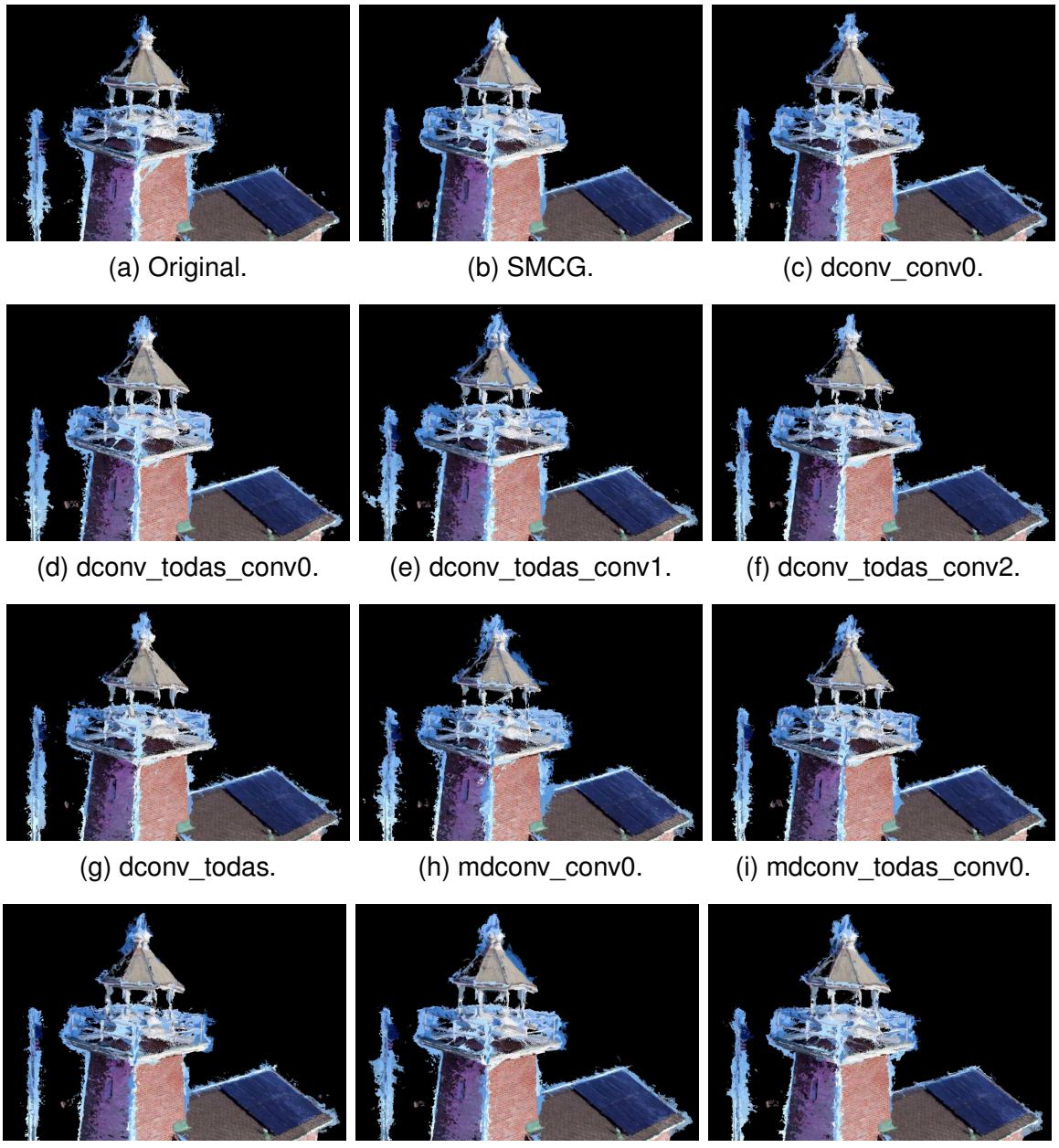
Os resultados quantitativos das redes treinadas com o *dataset* BlendedMVS e avaliadas com o *dataset Tanks and Temples* são apresentados na Tabela 9. Comparando os resultados com o treinamento anterior as métricas melhoraram, principalmente pelo fato do *dataset* utilizado no treinamento possuir cenas em ambientes externos, que é o foco do *Tanks and Temples* e não estão presentes no DTU. Tendo como referência a rede original, a maioria das cenas ficaram piores nas redes modificadas, com exceção de alguns casos, em especial, o resultado da *Lighthouse*, que é melhor em todas as modificações. A análise das nuvens de pontos não deixa claro qual é a causa da melhora, pois os ruídos percebidos nas extremidades da torre continuam, e em alguns casos são piores nas redes modificadas, como pode ser observado na Figura 39. Esses ruídos continuam pois como no *dataset* DTU, o BlendedMVS também elimina o céu no treinamento. Tendo como referência a rede SMCG, a maioria das redes modificadas teve uma pontuação maior no *Horse*, na *Lighthouse* e no *Panther*. Assim como no treinamento anterior, a melhora no *Horse* ocorreu pela maior completude na base abaixo do cavalo, na *Lighthouse* e no *Panther* a diferença na pontuação não ficou clara na nuvem de pontos. Essas cenas podem ser observadas nas Figuras 40, 39 e 41, respectivamente. Analisando a média, as diferenças não foram expressivas e a rede original foi a melhor. Uma imagem de referência da cena do *Horse*, da *Lighthouse* e do *Panther* podem ser observadas na Figura 28.

Tabela 9 – Resultados quantitativos do *Tanks and Temples* para as redes treinadas com o BlendedMVS.

Método	**Family**	**Francis**	**Horse**	**Light-house**	**M60**	**Panther**	**Play-ground**	**Train**	**Média**
Original	**75,72**	**58,56**	**47,43**	53,69	58,69	**56,05**	**60,53**	50,66	**57,67**
SMCG	74,53	57,71	45,64	54,40	58,61	54,39	59,76	**51,62**	57,08
dconv_conv0	73,85	57,09	46,21	54,72	58,65	54,69	59,72	50,52	56,93
dconv_todas_conv0	74,76	57,97	47,31	54,18	58,63	54,37	59,66	50,54	57,18
dconv_todas_conv1	74,03	58,20	47,03	55,13	58,41	54,64	59,67	49,70	57,10
dconv_todas_conv2	74,36	57,68	46,40	54,44	58,44	54,40	59,74	50,08	56,94
dconv_todas	74,30	58,52	46,94	**55,92**	**59,04**	54,59	59,65	51,17	57,52
mdconv_conv0	74,51	58,24	46,58	55,34	58,50	54,72	59,48	51,00	57,29
mdconv_todas_conv0	74,36	58,27	47,51	54,70	58,41	54,87	59,85	51,28	57,41
mdconv_todas_conv1	74,43	57,65	47,60	54,98	58,79	54,62	59,42	51,30	57,35
mdconv_todas_conv2	73,91	56,77	46,42	55,06	58,06	54,17	58,99	50,70	56,76
mdconv_todas	73,87	58,10	47,67	55,37	58,72	53,63	59,78	51,29	57,30

Fonte – Elaborado pelo autor.

Figura 39 – Nuvens de pontos da *Lighthouse* do *dataset Tanks and Temples*.



(a) Original. (b) SMCG. (c) dconv_conv0.

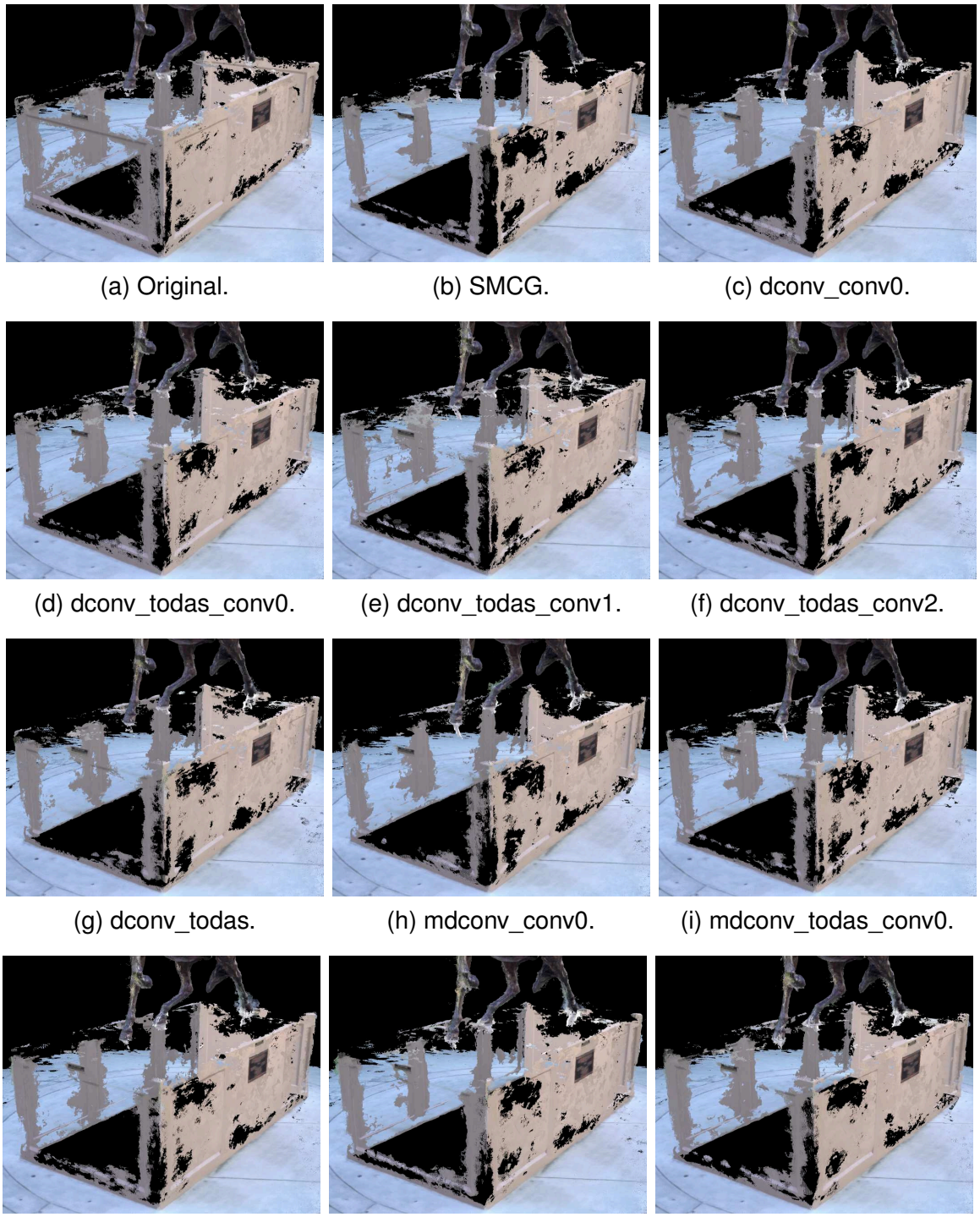
(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.

(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.

(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

Figura 40 – Nuvens de pontos do *Horse* do *dataset Tanks and Temples*.



(a) Original. (b) SMCG. (c) dconv_conv0.

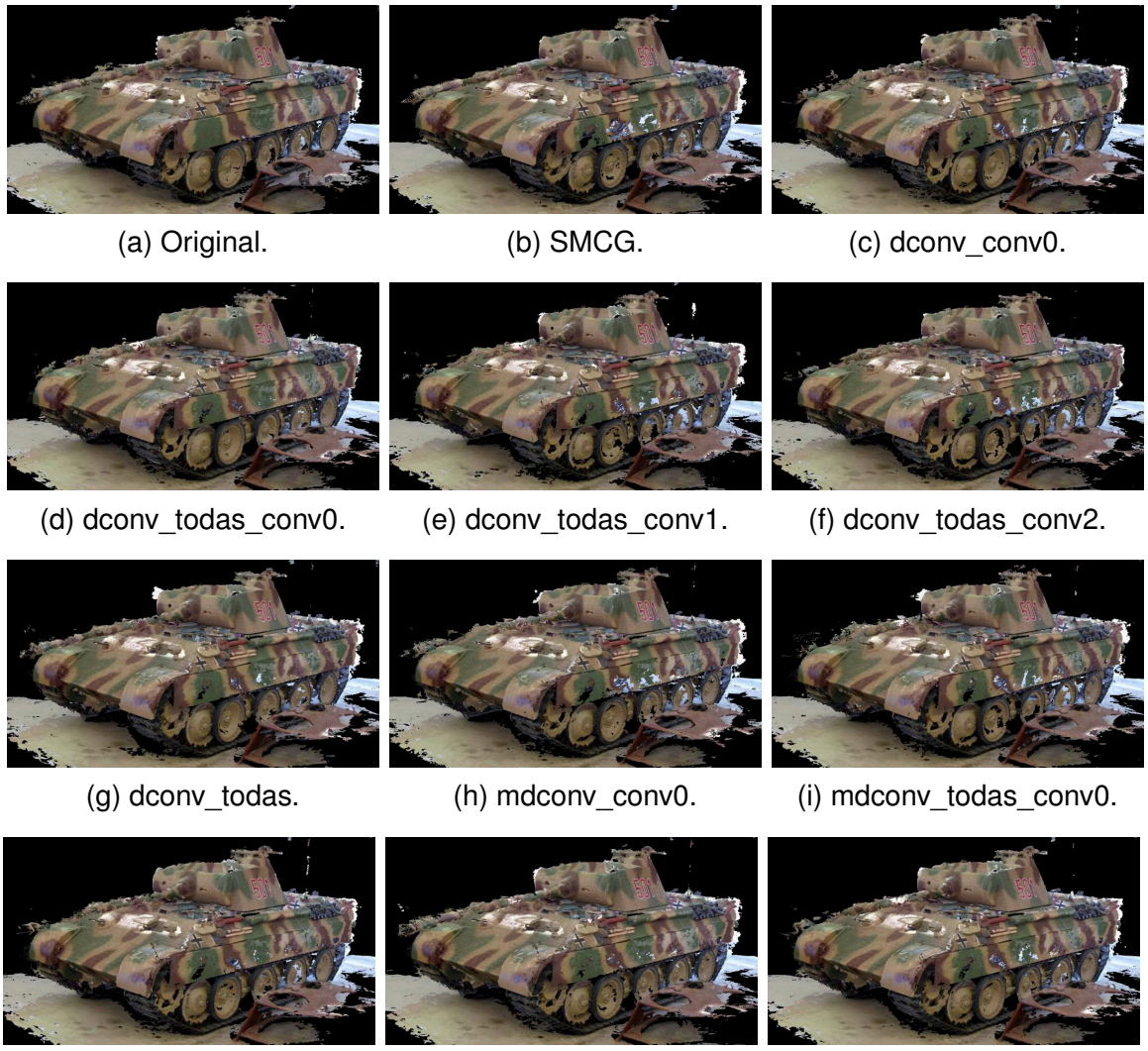
(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.

(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.

(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

Figura 41 – Nuvens de pontos do *Panther* do *dataset Tanks and Temples*.



(a) Original. (b) SMCG. (c) dconv_conv0.

(d) dconv_todas_conv0. (e) dconv_todas_conv1. (f) dconv_todas_conv2.

(g) dconv_todas. (h) mdconv_conv0. (i) mdconv_todas_conv0.

(j) mdconv_todas_conv1. (k) mdconv_todas_conv2. (l) mdconv_todas.

Fonte – Elaborado pelo autor.

6 CONCLUSÃO

6.1 CONSIDERAÇÕES FINAIS

Os testes de consumo de memória no treinamento das redes com o *dataset* DTU (Tabela 4), mostraram que a adição do SMCG foi tanto efetiva quanto necessária, pois permitiu o treinamento das redes com apenas os 8 GB de memória gráfica disponíveis na GTX 1080. Os dados de tempo de treinamento mostraram como algumas modificações na rede podem afetar drasticamente o tempo consumido, algo a ser observado, visto que os testes foram feitos utilizando apenas 16 épocas e alguns artigos passam de 50 épocas. Considerando um aumento de tempo linear, o maior e menor tempo do treinamento com 16 épocas, a diferença que foi de 39 horas (*mdconv_todas* - SMCG) passaria para 141 horas em 50 épocas, um aumento considerável que deve ser observado, principalmente quando o treinamento é feito em servidores remotos que cobram a hora de processamento. As outras métricas disponíveis no treinamento, como o erro absoluto e as acurácias de 1, 2 e 4 mm, não se mostraram tão expressivas, pois a rede que teve as melhores métricas (*mdconv_todas_conv*1) não traduziu esses valores como os melhores resultados nos testes quantitativos.

Os testes de estimação (Tabela 5), feitos nas resoluções 1536×1152, 1152×864 e 640×512, mostraram que a memória pode variar consideravelmente dependendo da resolução, sem seguir necessariamente um padrão, já o tempo se manteve consistente entre as resoluções e com o tempo do treinamento. Aqui a rede *dconv_todas* se destacou por ter o menor consumo de memória entre todas as redes testadas. Essa melhora permite que a rede seja utilizada na resolução 1152×864 em sistemas com uma restrição maior de memória gráfica (VRAM), como em uma Jetson Nano [1], que possui apenas 4 GB de VRAM. Os testes foram feitos com resoluções diferentes para medir o impacto dessa mudança no resultado final, e ao contrário do que se esperava, os testes quantitativos favoreceram a rede 1152×864 ao invés da 1536×1152. Uma explicação para esse mudança foi o fato das redes terem sido treinadas com a resolução 640×512, fazendo com que a imagem de entrada para as últimas camadas da Conv 2 fosse de 160×128, já para a estimação na resolução 1536×1152, as últimas camadas da Conv 2 recebem uma imagem de 384×288, fazendo com que os filtros (de tamanho 3×3) não “enxerguem” a mesma porção da cena. Além do pior resultado nos testes quantitativos, as mudanças visuais entre as resoluções não se mostraram tão significantes quanto o aumento de memória e tempo de processamento para as resoluções maiores. A resolução 640×512 se mostrou ideal para a maioria das aplicações, principalmente as que continuam a *pipeline* da fotogrametria, com as etapas de geração de superfície e texturização.

Nos testes quantitativos do *dataset* DTU, as redes treinadas com o DTU tiveram

[1] www.nvidia.com/en-us/autonomous-machines/embedded-systems/

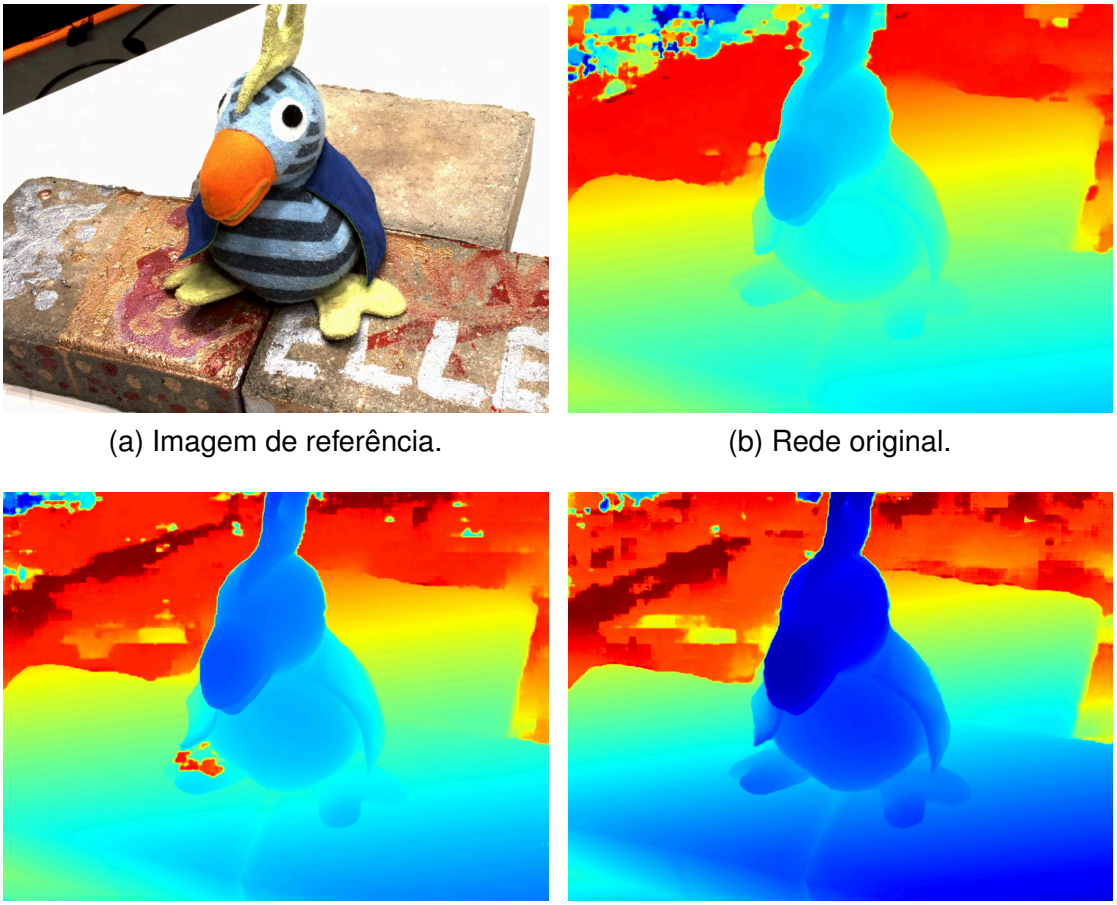
resultados melhores se comparadas as redes treinadas com o BlendedMVS, algo já esperado, pois a rede treinada com o DTU, apesar de não ter visto as mesmas cenas, foi treinada com cenas nas mesmas condições de iluminação e posição das câmeras. As redes modificadas treinadas com o DTU mantiveram a acurácia ou tiveram uma leve piora, na completude todas as modificações trouxeram uma melhora, se comparadas a rede original. As redes modificadas treinadas com o BlendedMVS tiveram uma melhora tanto em acurácia quanto em completude, sendo as únicas exceções a SMCG que piorou a completude e a *dconv_conv*0 que piorou a acurácia. Ambos os treinamentos trouxeram melhoras nas médias para todas as modificações. Nos testes qualitativos, quando comparadas as nuvens de pontos das redes modificadas com a rede original, a diferença mais perceptível ficou na completude das cenas. Em geral, as redes modificadas apresentando superfícies com menos falhas e mais pontos nas áreas que são vistas por poucas imagens (extremidade das cenas).

Nos testes quantitativos do *dataset Tanks and Temples* as redes treinadas com o BlendedMVS tiveram resultados melhores se comparadas as redes treinadas com o DTU. Isto se deve principalmente pelo fato do BlendedMVS conter cenas em ambientes externos, algo que o DTU não possui. Para o treinamento do DTU todas as redes modificadas conseguiram uma pontuação maior no *Playground*. As outras cenas não mostraram tal consistência, fazendo com que a média em geral ficasse pior que a da rede original. As únicas exceções foram as redes *dcvon_todas* e *mdconv_todas* que tiveram uma média levemente melhor. Para o treinamento do BlendedMVS a *Lighthouse* teve um resultado melhor em todas as modificações. Como a melhora ocorreu apenas na *Lighthouse*, a média ficou pior para todas as modificações, fazendo da rede original o melhor resultado. O fato do *Tanks and Temples* disponibilizar apenas o *F-score* (média harmônica da acurácia e completude) das cenas dificultou a avaliação qualitativa dos resultados, pois apesar de existir uma diferença numérica, se essa fosse em acurácia, dificilmente seria visível comparando as nuvens de pontos.

Os testes quantitativos e qualitativos dos *datasets* foram feitos utilizando as nuvens de pontos, e como ela é o objetivo final dos algoritmos de *Multi-View Stereo*, analisar ela não é uma abordagem incorreta. Entretanto, a saída da rede é um mapa de profundidade e sua análise traz outras diferenças que são ofuscadas pela etapa de fusão dos mapas. Na Figura 42 tem-se uma comparação entre o mapa de profundidade da rede original, da SMCG e da dconv_conv0. A rede original tem dificuldade em delimitar as bordas da base e do objeto, principalmente no topo do objeto. A rede SMCG consegue melhorar a delimitação, mas alguns pontos da base ainda se mesclam com o fundo, além de adicionar um erro no pé esquerdo (pontos vermelhos). A rede dconv_conv0 consegue melhorar a delimitação da base, se comparada a SMCG, e retira o erro que ocorria antes. A falha de delimitação da rede original e o erro no pé esquerdo da rede SMCG são resolvidos com mais mapas de profundidade de diferen-

tes pontos de vista, pois esses erros não ocorrem da mesma maneira em mais de uma vista, fazendo com que seus pontos sejam considerados *outliers*.

Figura 42 – Mapas de profundidade da cena 4 do *dataset* DTU.



(a) Imagem de referência. (b) Rede original.

(c) Rede SMCG. (d) Rede dconv_conv0.

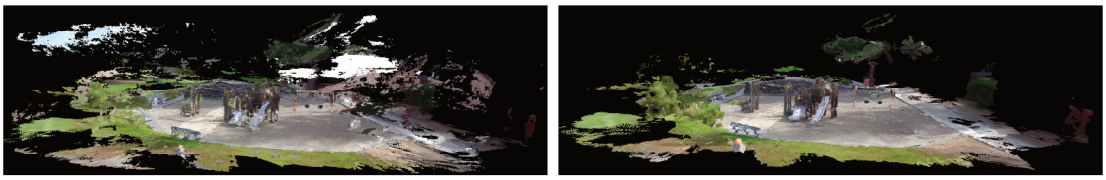
Fonte – Elaborado pelo autor.

A principal diferença entre os treinamentos ficou na quantidade de ruído presente nas nuvens de pontos. Na Figura 43 tem-se uma comparação dos resultados da rede *mdconv_todas_conv2* para a cena do *M60* e do *Playground*, apenas modificando o *dataset* utilizado no treinamento. Pode-se perceber que a rede treinada com o DTU não sabe lidar com um ambiente externo não controlado, criando ruídos que ocorrem nas bordas dos objetos e flutuando pela cena. Isso acontece pois todos o fundo das imagens do *dataset* DTU são mascarados e quando ele aparece, é totalmente preto ou branco. No resultado da rede treinada com o BlendedMVS esses ruídos diminuem significativamente, principalmente pelo fato do BlendedMVS possuir cenas em ambientes externos com uma maior presença de ruídos.

Independente da rede ou do treinamento utilizado, o ruído nas bordas dos objetos que fazem fronteira com o céu foi um problema. Isso é mais visível nas redes treinadas com o DTU no resultado da *Lighthouse* do *Tanks and Temples* (Figura 36).

Figura 43 – Diferença no nível de ruídos das cenas do *Tanks and Temples* da rede *mdconv_todas_conv2* treinada com o DTU e o BlendedMVS.

(a) Treinado com DTU. (b) Treinado com BlendedMVS.



(c) Treinado com DTU. (d) Treinado com BlendedMVS.

Fonte – Elaborado pelo autor.

O treinamento com o DTU piorou esse problema por conter apenas objetos em um ambiente controlado. Apesar do *dataset* BlendedMVS possuir cenas em ambientes externos, o céu é sempre mascarado durante o treinamento, como pode ser visto nas Figuras 44b e 44e . Os problemas do *dataset* BlendedMVS não se limitam apenas a mascarar o céu, como ele é criado a partir de modelos 3D disponíveis na internet, a qualidade entre os modelos varia muito. Por fazer um *blending* entre a imagem original (que criou a cena) e a imagem renderizada (Seção 5.1.3), a qualidade da imagem final fica muito dependente da qualidade do processo de texturização aplicado na cena. Isso pode ser visto analisando a árvore na Figura 44a, que possui uma grande quantidade de borrões e galhos sem continuidade, possivelmente ocasionados por algum movimento da árvore durante a captura das imagens. Observando a imagem mascarada da árvore (Figura 44b), a superfície reconstruída se estende muito além dos galhos da árvore, reforçando a dificuldade da rede em aprender como se comportar na presença do céu, pois os únicos momentos que ele é visto, são quando alguma superfície foi gerada erroneamente. O segundo problema, que está relacionado com a falta de eliminação de ruídos das cenas fica claro na Figura 44e, onde um pedaço do céu foi reconstruído e não foi eliminado/filtrado por algum processo automático/manual.

Figura 44 – Exemplos de cenas do *dataset* BlendedMVS.

(a) Imagem original. (b) Imagem mascarada. (c) Mapa de profundidade.

(d) Imagem original. (e) Imagem mascarada. (f) Mapa de profundidade.

Fonte – (YAO; LUO; LI; ZHANG *et al.*, 2020).

6.2 PRINCIPAIS CONTRIBUIÇÕES

A principal contribuição desse trabalho foi fazer um levantamento de como as redes neurais artificiais podem ser utilizadas para o problema do *Multi-View Stereo*, dividindo esse problema clássico em algumas etapas bem definidas e mostrando como diferentes estratégias podem ser utilizadas para traduzir esse problema para as redes neurais artificiais. As modificações propostas (Tabela 3) também trouxeram melhoras expressivas na rede CasMVSNet. A adição da SMCG diminuiu o consumo de memória e tempo durante o treinamento e a estimação. As convoluções deformáveis e deformáveis moduladas conseguiram melhorar os resultados, principalmente a métrica de completude nos testes qualitativos. Considerando a modificação *dconv_todas* treinada com o *dataset* DTU, existiu uma redução de 22% no consumo de memória gráfica (resolução 1152×864), melhora na completude e na média do resultado quantitativo do DTU e uma pontuação média maior no *Thanks and Temples*, com uma penalidade de apenas 3.75% no tempo de processamento. Na Tabela 10 tem-se uma comparação dos resultados quantitativos do *dataset* DTU entre as duas melhores redes desenvolvidas nesse trabalho e as redes encontradas na revisão bibliográfica (os resultados foram retirados dos artigos). Algumas redes da revisão bibliográfica não aparecem pois não foram testadas no *dataset* DTU.

Tabela 10 – Resultados quantitativos do *dataset* DTU para as diferentes arquiteturas de redes.

Método	**Acurácia (mm)**	**Completude (mm)**	**Média (mm)**
SurfaceNet	0,450	1,043	0,746
MVSNet	0,396	0,527	0,462
R-MVSNet	0,383	0,452	0,417
MVS^2	0,760	0,515	0,637
MVSCRF	0,371	0,426	0,398
P-MVSNet	0,406	0,434	0,420
Point-MVSNet	0,361	0,421	0,391
AttMVS	0,383	**0,329**	0,356
CVP-MVSNet	**0,296**	0,406	0,351
UCS-NET	0,338	0,349	**0,344**
Fast-MVSNet	0,336	0,403	0,370
MVSNet++	0,407	0,345	0,376
CasMVSNet	0,325	0,385	0,355
dconv_conv0 (DTU)	0,349	0,340	0,345
dconv_todas (DTU)	0,348	0,350	0,349

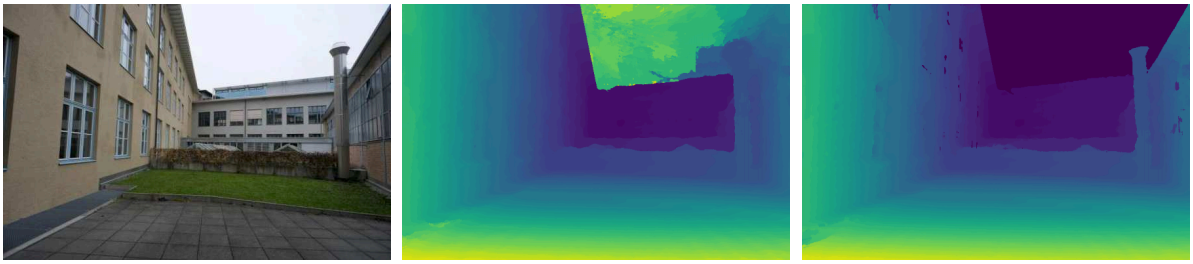
Fonte – Elaborado pelo autor.

6.3 TRABALHOS FUTUROS

Como visto, a modificação da etapa de extração de *features* com as convoluções deformáveis trouxeram melhoras tanto quantitativas quanto qualitativas, mas não é só nessa etapa que elas podem ser utilizadas. No caso das redes R-MVSNet e RED-Net, por utilizarem uma RNN na etapa de regularização do *cost volume*, cada "fatia" do *cost volume* é processada por vez, sendo possível utilizar as convoluções deformáveis diretamente na regularização. Para aplicar essa estratégia na rede CasMVSNet (modificada nesse trabalho), poderia ser utilizada uma implementação 3D da convolução deformável, como no trabalho (YING *et al.*, 2020), onde as camadas deformáveis 3D foram aplicadas para aumentar a resolução de vídeos, trazendo resultados mais nítidos.

Apesar do *dataset* BlendedMVS focar em ambientes externos, a qualidade dos seus modelos 3D não permite que todos os desafios encontrados em tais ambientes sejam resolvidos. Como visto na Figura 44b, a delimitação da árvore não foi correta e todo o céu foi eliminado. Para endereçar esse problema, a rede DeepMVS criou o *dataset* MVS-Synth, um *dataset* sintético criado a partir de cenas do jogo GTA V. Seu principal objetivo é adicionar no treinamento os objetos que são difíceis de se capturar em um ambiente real, como o céu, superfícies reflexivas e estruturas muito finas (como as árvores). Analisando uma comparação disponibilizada pelo DeepMVS, disposta na Figura 45, a adição do *dataset* sintético eliminou as falhas presentes no céu e conseguiu delimitar a chaminé de metal presente na cena, onde a rede treinada sem o *dataset* sintético falhou. Esse *dataset* não substituiria completamente os outros, mas poderia ser utilizado de maneira a complementar o treinamento da rede.

Figura 45 – Diferença do resultado do DeepMVS com e sem o *dataset* sintético no treinamento.



(a) Imagem de referência para gerar os mapas.

(b) Mapa de profundidade com o *dataset* sintético.

(c) Mapa de profundidade sem o *dataset* sintético.

Fonte – (HUANG *et al.*, 2018).

REFERÊNCIAS

ALCANTARILLA, Pablo F; SOLUTIONS, T. Fast explicit diffusion for accelerated features in nonlinear scale spaces. **IEEE Trans. Patt. Anal. Mach. Intell**, v. 34, n. 7, p. 1281–1298, 2013.

BAY, Herbert; TUYTELAARS, Tinne; VAN GOOL, Luc. Surf: Speeded up robust features. *In*: SPRINGER. EUROPEAN conference on computer vision. [*S.l.*: *s.n.*], 2006. P. 404–417.

BRASIL. MINISTÉRIO DA SAÚDE. SECRETARIA DE CIÊNCIA, Tecnologia e Insumos Estratégicos. Departamento de Ciência e Tecnologia. Diretrizes metodológicas: elaboração de revisão sistemática e metanálise de ensaios clínicos randomizados. **Brasília; Ministério da Saúde**, 2012.

BROWN, Matthew; LOWE, David G. Automatic panoramic image stitching using invariant features. **International journal of computer vision**, Springer, v. 74, n. 1, p. 59–73, 2007.

BUDUMA, N.; LOCASCIO, N. **Fundamentals of Deep Learning: Designing Next-generation Machine Intelligence Algorithms**. [*S.l.*]: O'Reilly Media, 2017. ISBN 9781491925614.

CHEN, P. *et al.* MVSNet++: Learning Depth-Based Attention Pyramid Features for Multi-View Stereo. **IEEE Transactions on Image Processing**, v. 29, p. 7261–7273, 2020. DOI: 10.1109/TIP.2020.3000611.

CHEN, Rui *et al.* Point-based Multi-view Stereo Network. *In*: THE IEEE International Conference on Computer Vision (ICCV). [*S.l.*: *s.n.*], 2019.

CHENG, Shuo *et al.* Deep Stereo Using Adaptive Thin Volume Representation With Uncertainty Awareness. *In*: PROCEEDINGS of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], jun. 2020.

CHOI, Sungil *et al.* Learning descriptor, confidence, and depth estimation in multi-view stereo. *In*: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. [*S.l.*: *s.n.*], 2018. P. 276–282.

CIABURRO, G.; VENKATESWARAN, B. **Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles**. [*S.l.*]: Packt Publishing, 2017. ISBN 9781788399418.

CONSENSUS, Random Sample. A paradigm for model fitting with applications to image analysis and automated cartography. **MA Fischler, RC Bolles**, v. 6, p. 381–395, 1981.

CORKE, P. **Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised, Extended And Updated Edition**. [*S.l.*]: Springer International Publishing, 2017. (Springer Tracts in Advanced Robotics). ISBN 9783319544137.

COSTA, Angelo; ZOLTOWSKI, Ana. Como escrever um artigo de revisão sistemática. *In*: [*S.l.*: *s.n.*], jun. 2014.

DAI, J. *et al.* Deformable Convolutional Networks. *In*: 2017 IEEE International Conference on Computer Vision (ICCV). [*S.l.*: *s.n.*], 2017. P. 764–773. DOI: 10.1109/ICCV.2017.89.

DAI, Y. *et al.* MVS2: Deep Unsupervised Multi-View Stereo with Multi-View Symmetry. *In*: 2019 International Conference on 3D Vision (3DV). [*S.l.*: *s.n.*], 2019. P. 1–8. DOI: 10.1109/3DV.2019.00010.

FABBRI, S. *et al.* Improvements in the StArt tool to better support the systematic review process. *In*: EASE '16. [*S.l.*: *s.n.*], 2016.

FAWCETT, Tom. An introduction to ROC analysis. **Pattern recognition letters**, Elsevier, v. 27, n. 8, p. 861–874, 2006.

GALLUP, D. *et al.* Real-Time Plane-Sweeping Stereo with Multiple Sweeping Directions. *In*: 2007 IEEE Conference on Computer Vision and Pattern Recognition. [*S.l.*: *s.n.*], 2007. P. 1–8.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [*S.l.*]: MIT Press, 2016.

GRAHAM, R.L. **Concrete Mathematics: A Foundation for Computer Science**. [*S.l.*]: Pearson Education, 1994. ISBN 9788131708415.

GU, Xiaodong *et al.* Cascade Cost Volume for High-Resolution Multi-View Stereo and Stereo Matching. *In*: PROCEEDINGS of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], jun. 2020.

GUERRA, Winter *et al.* FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality. **2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**, IEEE, nov. 2019. DOI: 10.1109/iros40897.2019.8968116.

GUPTA, M. *et al.* Structured light 3D scanning in the presence of global illumination. *In*: CVPR 2011. [*S.l.*: *s.n.*], 2011. P. 713–720. DOI: 10.1109/CVPR.2011.5995321.

HARTLEY, Richard; ZISSERMAN, Andrew. **Multiple View Geometry in Computer Vision**. [*S.l.*]: Cambridge University Press, 2004.

HUANG, Po-Han *et al.* DeepMVS: Learning Multi-View Stereopsis. *In*: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], 2018.

IM, Sunghoon *et al.* DPSNet: End-to-end Deep Plane Sweep Stereo. **International Conference on Learning Representations (ICLR)**, mai. 2019.

JENSEN, R. *et al.* Large Scale Multi-view Stereopsis Evaluation. *In*: 2014 IEEE Conference on Computer Vision and Pattern Recognition. [*S.l.*: *s.n.*], 2014. P. 406–413. DOI: 10.1109/CVPR.2014.59.

JI, Mengqi *et al.* SurfaceNet: An End-to-End 3D Neural Network for Multiview Stereopsis. **2017 IEEE International Conference on Computer Vision (ICCV)**, IEEE, out. 2017. DOI: 10.1109/iccv.2017.253.

KAZHDAN, Michael; HOPPE, Hugues. Screened poisson surface reconstruction. **ACM Transactions on Graphics (ToG)**, ACM New York, NY, USA, v. 32, n. 3, p. 1–13, 2013.

KENDALL, A. *et al.* End-to-End Learning of Geometry and Context for Deep Stereo Regression. *In*: 2017 IEEE International Conference on Computer Vision (ICCV). [*S.l.*: *s.n.*], 2017. P. 66–75. DOI: 10.1109/ICCV.2017.17.

KHOSHELHAM, Kourosh. Accuracy analysis of kinect depth data. *In*: 1. ISPRS workshop laser scanning. [*S.l.*: *s.n.*], 2011.

KNAPITSCH, Arno *et al.* Tanks and temples: Benchmarking large-scale scene reconstruction. **ACM Transactions on Graphics (ToG)**, ACM New York, NY, USA, v. 36, n. 4, p. 1–13, 2017.

KROGIUS, Maximilian; HAGGENMILLER, Acshi; OLSON, Edwin. Flexible Layouts for Fiducial Tags. *In*: PROCEEDINGS of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). [*S.l.*: *s.n.*], out. 2019.

LIU, J.; JI, S. A Novel Recurrent Encoder-Decoder Structure for Large-Scale Multi-View Stereo Reconstruction From an Open Aerial Dataset. *In*: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], 2020. P. 6049–6058. DOI: 10.1109/CVPR42600.2020.00609.

LOWE, David G. Object recognition from local scale-invariant features. *In*: IEEE. COMPUTER vision, 1999. The proceedings of the seventh IEEE international conference on. [*S.l.*: *s.n.*], 1999. P. 1150–1157.

LUO, Keyang; GUAN, Tao; JU, Lili; HUANG, Haipeng *et al.* P-MVSNet: Learning Patch-Wise Matching Confidence Aggregation for Multi-View Stereo. *In*: PROCEEDINGS of the IEEE/CVF International Conference on Computer Vision (ICCV). [*S.l.*: *s.n.*], out. 2019.

LUO, Keyang; GUAN, Tao; JU, Lili; WANG, Yuesong *et al.* Attention-Aware Multi-View Stereo. *In*: PROCEEDINGS of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], jun. 2020.

MAO, Xiao-Jiao; SHEN, Chunhua; YANG, Yu-Bin. Image Restoration Using Very Deep Convolutional Encoder-Decoder Networks with Symmetric Skip Connections. *In*: PROCEEDINGS of the 30th International Conference on Neural Information Processing Systems. Barcelona, Spain: Curran Associates Inc., 2016. (NIPS'16), p. 2810–2818.

MASSON, Juliano. Desenvolvimento de um algoritmo para a reconstrução 3D a partir de imagens RGB. Trabalho de Conclusão de Curso, Departamento de Engenharia de Controle, Automação e Computação, UFSC, Blumenau, SC, 2019.

MELO, Aurelio G *et al.* 3D Correspondence and Point Projection Method for Structures Deformation Analysis. **IEEE Access**, IEEE, v. 8, p. 177823–177836, 2020.

MERRELL, P. *et al.* Real-Time Visibility-Based Fusion of Depth Maps. *In*: 2007 IEEE 11th International Conference on Computer Vision. [*S.l.*: *s.n.*], 2007. P. 1–8. DOI: 10.1109/ICCV.2007.4408984.

SCHÖNBERGER, Johannes Lutz; FRAHM, Jan-Michael. Structure-from-Motion Revisited. *In*: CONFERENCE on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], 2016.

SIEGWART, Roland; NOURBAKHSH, Illah Reza; SCARAMUZZA, Davide. **Introduction to autonomous mobile robots**. [*S.l.*]: MIT press, 2011.

SZELISKI, Richard. **Computer vision: algorithms and applications**. [*S.l.*]: Springer Science & Business Media, 2010.

TAO, F. *et al.* Digital Twin in Industry: State-of-the-Art. **IEEE Transactions on Industrial Informatics**, v. 15, n. 4, p. 2405–2415, 2019. DOI: 10.1109/TII.2018.2873186.

TUYTELAARS, Tinne; MIKOLAJCZYK, Krystian *et al.* Local invariant feature detectors: a survey. **Foundations and trends® in computer graphics and vision**, Now Publishers, Inc., v. 3, n. 3, p. 177–280, 2008.

WANG, John; OLSON, Edwin. AprilTag 2: Efficient and robust fiducial detection. *In*: PROCEEDINGS of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). [*S.l.*: *s.n.*], out. 2016.

XIANG, Xiang *et al.* Pruning multi-view stereo net for efficient 3D reconstruction. **ISPRS Journal of Photogrammetry and Remote Sensing**, v. 168, p. 17–27, 2020. ISSN 0924-2716.

XU, Qingshan; TAO, Wenbing. **Learning Inverse Depth Regression for Multi-View Stereo with Correlation Cost Volume**. [*S.l.*: *s.n.*], 2019. arXiv: 1912.11746 [cs.CV].

XUE, Youze *et al.* MVSCRF: Learning Multi-View Stereo With Conditional Random Fields. *In*: PROCEEDINGS of the IEEE/CVF International Conference on Computer Vision (ICCV). [*S.l.*: *s.n.*], out. 2019.

YANG, Jiayu *et al.* Cost Volume Pyramid Based Depth Inference for Multi-View Stereo. *In*: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], jun. 2020.

YAO, Yao; LUO, Zixin; LI, Shiwei; FANG, Tian *et al.* Mvsnet: Depth inference for unstructured multi-view stereo. *In*: PROCEEDINGS of the European Conference on Computer Vision (ECCV). [*S.l.*: *s.n.*], 2018. P. 767–783.

YAO, Yao; LUO, Zixin; LI, Shiwei; SHEN, Tianwei *et al.* Recurrent mvsnet for high-resolution multi-view stereo depth inference. *In*: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition. [*S.l.*: *s.n.*], 2019. P. 5525–5534.

YAO, Yao; LUO, Zixin; LI, Shiwei; ZHANG, Jingyang *et al.* BlendedMVS: A Large-scale Dataset for Generalized Multi-view Stereo Networks. **Computer Vision and Pattern Recognition (CVPR)**, 2020.

YING, X. *et al.* Deformable 3D Convolution for Video Super-Resolution. **IEEE Signal Processing Letters**, v. 27, p. 1500–1504, 2020. DOI: 10.1109/LSP.2020.3013518.

YU, Z.; GAO, S. Fast-MVSNet: Sparse-to-Dense Multi-View Stereo With Learned Propagation and Gauss-Newton Refinement. *In*: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], 2020. P. 1946–1955. DOI: 10.1109/CVPR42600.2020.00202.

ZHANG, Z. A flexible new technique for camera calibration. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 11, p. 1330–1334, 2000. DOI: 10.1109/34.888718.

ZHANG, Z. Microsoft Kinect Sensor and Its Effect. **IEEE MultiMedia**, v. 19, n. 2, p. 4–10, 2012. DOI: 10.1109/MMUL.2012.24.

ZHENG, Shuai *et al.* Conditional Random Fields as Recurrent Neural Networks. **2015 IEEE International Conference on Computer Vision (ICCV)**, IEEE, dez. 2015. DOI: 10.1109/iccv.2015.179.

ZHU, X. *et al.* Deformable ConvNets V2: More Deformable, Better Results. *In*: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). [*S.l.*: *s.n.*], 2019. P. 9300–9308. DOI: 10.1109/CVPR.2019.00953.