http://journal.unoeste.br/index.php/ce/index
DOI: 10.5747/ce.2022.v14.e389
ISSN on-line 2178-8332

Colloquium Exactarum

Submetido: 29/06/2022 Revisado: 12/09/2022 Aceito: 04/10/2022

RECONSTRUÇÃO 3D DE CENAS E OBJETOS A PARTIR DE IMAGENS DIGITAIS

3D scene and object reconstruction from digital images

Fabrício Milanez[1], Francisco Assis da Silva[1], Flávio Pandur Albuquerque Cabral[1], Leandro Luiz de Almeida[1], Almir Olivette Artero[2], Marco Antônio Piteri[2]

[1] **Universidade do Oeste Paulista – UNOESTE, Faculdade de Informática de Presidente Prudente, Presidente Prudente, SP.**
[2] **Universidade Estadual Paulista – UNESP, Faculdade de Ciências e Tecnologia, Departamento de Matemática e Computação, Presidente Prudente, SP.**
E-mail: fabriciomilanez@protonmail.com, chico@unoeste.br, pandur@unoeste.br, llalmeida@unoeste.br, almir.artero@unesp.br, marco.piteri@unesp.br

RESUMO – Novas tecnologias, como impressoras 3D e carros autônomos por exemplo, advindas de avanços em Visão Computacional e outras áreas, vem impulsionando um interesse cada vez mais elevado em *pipelines* robustos para reconstrução 3D de ambientes e objetos reais. Por meio de métodos de reconstrução 3D, é possível criar uma aplicação que toma fotografias digitais de uma cena observada como entradas e é capaz de obter um modelo 3D que a represente. Este modelo então, pode ser utilizado em uma ampla gama de aplicações, tais como criação de *assets* de jogos digitais, manipulação de vídeos com efeitos especiais ou replicação de objetos com o uso de impressora 3D, por exemplo. Neste trabalho, são abordados, apresentados e discutidos métodos que dizem respeito às diferentes etapas de um *pipeline* tradicional de reconstrução 3D partindo somente de imagens digitais na implementação e uso de ferramentas computacionais que foram capazes de gerar modelos 3D que representam uma cena observada por estas imagens com alta fidelidade.
Palavras-chave: visão computacional; reconstrução 3D; geometria epipolar; *Structure from Motion*; *Multi-View Stereo*.

ABSTRACT – New technologies, such as 3D printers and autonomous cars for instance, originating from advances in Computer Vision and other fields, have been causing an increasingly high interest in robust pipelines for 3D reconstruction of real environments and objects. Through use of 3D reconstruction methods, it is possible to create an application that takes digital photographs of an observed scene as inputs and is capable of obtaining a 3D model that represents it. This model could then be used in a wide range of applications, such as game asset generation, video manipulation with special effects or object replication with the use of a 3D printer, for instance. In this paper, we address, present and discuss methods that concern the different stages of a traditional 3D reconstruction pipeline from digital images only in the implementation and use of computational tools that were capable of generating 3D models that represent a scene observed these images with high fidelity.
Keywords: computer vision; 3D reconstruction; epipolar geometry; Structure from Motion; Multi-View Stereo.

1. INTRODUÇÃO

Com o surgimento de tecnologias emergentes como robôs autônomos, que dependem fortemente de visão computacional para obter mediadas reais do ambiente em que estão inseridos, interesse em metodologias capazes de aferir tais medidas com acurácia vem aumentando. Pesquisas de mercado mostram que o interesse em tecnologias de reconstrução 3D tende a continuar crescendo. Segundo o relatório da Straits Research (2019), o mercado de tecnologias de reconstrução 3D deve alcançar uma taxa de crescimento anual composta de 8,1% no período previsto entre 2019 a 2026.

Uma das abordagens possíveis para resolver o problema da reconstrução 3D é utilizar-se de scanners 3D baseados em tecnologia laser para mapear e reconstruir a área ao redor e obter medidas de distância com precisão. Tal método, porém, resulta em custos elevados que dificultam a acessibilidade e viabilidade da elaboração de sistemas autônomos cientes de seus arredores. Uma outra abordagem utiliza-se de técnicas de fotogrametria – segundo McGlone (2013), uma ciência que envolve o processo de registro, medição e interpretação de imagens fotográficas a fim de se extrair medidas a respeito de objetos e ambientes no meio físico – para mapear o ambiente. Com o uso de técnicas da fotogrametria, é possível reconstruir a cena observada e extrair medidas a partir do modelo tridimensional gerado.

As aplicações de tecnologia de reconstrução 3D envolvem diversos campos e indústrias. Alguns de seus casos de uso incluem aplicações em:

- **Medicina:** Uma das muitas possibilidades de uso desta tecnologia no campo da medicina é na criação de próteses e órteses diversas. Algumas das vantagens do emprego de fotogrametria e reconstrução tridimensional neste campo são custos mais acessíveis, precisão melhorada e tempo de medição menor em relação às técnicas tradicionais (LI; GAO; WANG, 2011);
- **Realidade Virtual:** Utilizada em conjunto com aplicações de realidade virtual (RV), cenas reconstruídas podem proporcionar uma experiência imersiva onde usuários podem observar e interagir com ambientes em meio virtual (EHTEMAMI *et al.*, 2021);
- **Indústria do Entretenimento:** Um caso recente de uso da reconstrução 3D na indústria do entretenimento envolveu a reconstrução das feições do falecido ator Peter Cushing a partir de um molde (*lifecast*) de silicone de seu rosto feito quando ainda em vida. O modelo 3D reconstruído foi sobreposto através de técnicas de efeitos visuais sobre o rosto do ator Guy Henry para o filme "*Rogue One: A Star Wars Story*" (INDIEWIRE, 2017);
- **Arquivamento histórico:** Objetos e marcos com significância histórica, se não preservados, vão naturalmente se deteriorando com o decorrer do tempo. O uso de técnicas de reconstrução 3D permite que estes objetos possam ser preservados digitalmente para fins de arquivamento histórico (BERNADIN *et al.*, 2015);
- **Replicação de objetos:** Em conjunto com softwares gratuitos de modelagem 3D e tecnologias emergentes como impressoras 3D, a tarefa de replicar objetos mundanos quaisquer se torna cada vez mais fácil, até mesmo para pessoas sem experiência prática me modelagem 3D.

Este trabalho busca propor uma sequência (*pipeline*) de soluções computacionais visando a obtenção de modelos 3D que representam uma cena observada, partindo somente de imagens digitais, sem a utilização de informações extra aferidas por sensores laser ou de giroscópios.

As seções que seguem essa introdução estão dispostas da seguinte maneira. Na Seção 2 serão abordados alguns conceitos importantes para o entendimento das metodologias aplicadas. Na Seção 3 estão descritas todas as metodologias empregadas no decorrer de cada etapa do *pipeline*. Os experimentos realizados seguidos de seus resultados são apresentados na Seção 4. Por fim, na Seção 5, encontram-se as conclusões finais aferidas a partir da interpretação dos resultados e discussões a respeito de trabalhos futuros.

2. CONCEITOS CHAVE

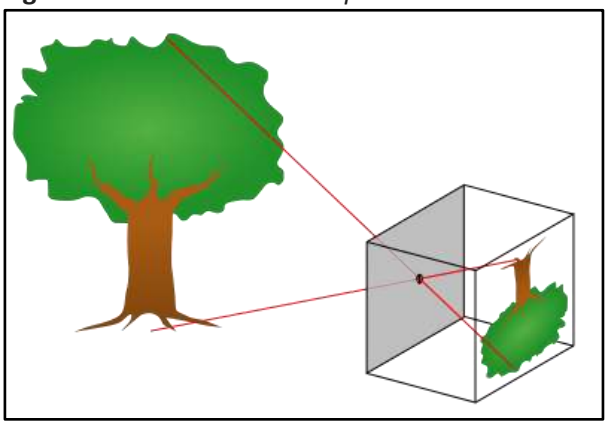
Nesta seção são abordados alguns dos conceitos fundamentais à compreensão dos métodos implementados, dentre eles, o modelo

de câmera *pinhole*, a geometria epipolar e a correspondência de *features* em imagens digitais.

2.1. Modelo de câmera pinhole

O modelo de câmera *pinhole* (Figura 1) é uma simplificação da estrutura básica de câmeras fotográficas reais (ESCRIVÁ *et al.*, 2019). Essencialmente, trata-se de uma caixa escura com um furo em um dos lados, chamado de ponto principal. Os feixes de luz que entram pelo ponto principal projetam uma imagem invertida da cena observada na face interna oposta ao ponto, denominada plano da imagem.

Figura 1. Modelo de câmera *pinhole*.



Fonte: (WIKIMEDIA COMMONS, 2022).

Este modelo descreve as relações matemáticas entre as coordenadas de um ponto no espaço tridimensional e sua projeção bidimensional sobre o plano da imagem em parâmetros intrínsecos, ou seja, parâmetros internos que são inerentes à câmera em si. Tais parâmetros podem ser dispostos em uma matriz 3x3 usada para projetar coordenadas 3D como coordenadas homogêneas 2D.

Outro fator que deve ser levado em consideração é a posição da câmera – sua rotação e translação em relação à origem $O = [0, 0, 0]$. Já que a metodologia proposta utiliza-se de diversas visões de uma mesma cena para poder reconstruí-la, por conveniência, é presumido que a primeira câmera esteja posicionada na origem, sendo assim, deverá ser aplicada uma transformada rígida sobre todas as outras câmeras subsequentes para que se possa descobrir sua localização.

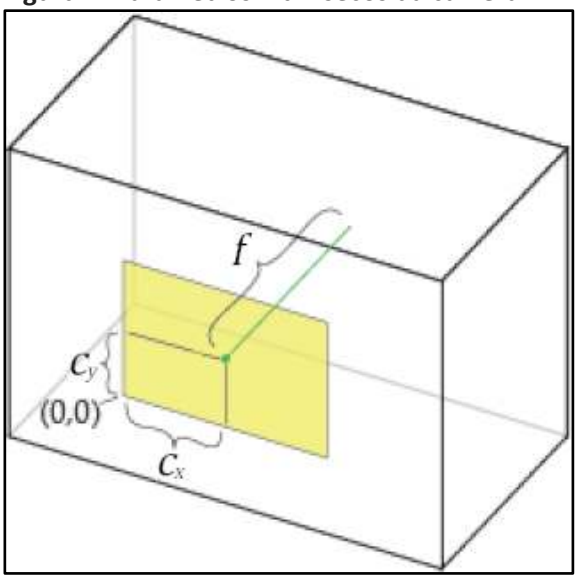
2.1.1. Matriz intrínseca

A matriz intrínseca da câmera, parametrizada por Hartley e Zisserman (2004), é disposta de acordo com a Equação 1.

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}, \qquad (1)$$

Cada parâmetro descreve uma propriedade geométrica da câmera. A distância focal f é a distância entre o ponto principal e o plano da imagem em pixels. As coordenadas $[c_x, c_y]$ representam o deslocamento em pixels do ponto principal da câmera, por onde passa o eixo principal em relação à origem do plano da imagem. Na Figura 2 são exemplificados os parâmetros intrínsecos da câmera.

Figura 2. Parâmetros intrínsecos da câmera.



Fonte: Os autores.

2.1.2. Matriz extrínseca

A matriz extrínseca descreve a localização de uma câmera no espaço tridimensional e a direção para qual ela está voltada. Ela é composta por duas componentes: uma matriz de rotação R 3x3 e um vetor linha de translação t de 3 dimensões. Juntas, essas componentes formam a matriz de transformação rígida $[R|t]$ 3x4 que descreve a posição da câmera em relação à origem.

A matriz $[R|t]$ pode ser modelada de acordo com a Equação 2 (HARTLEY; ZISSERMAN, 2004).

$$[R|t] = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix}, \qquad (2)$$

2.2. Projeção perspectiva

Segundo Escrivá *et al.* (2019), dado um ponto $W = [u, v, w]$ em coordenadas 3D do mundo, sua projeção perspectiva 2D sobre o plano da imagem Π dado pelo ponto $X = [x, y]$ pode ser obtida "deslizando-se" o ponto W pelo eixo principal da câmera até que ele se sobreponha ao plano da imagem. Essa interação é descrita por meio da Equação 3.

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}, \quad (3)$$

onde s é um fator de escala arbitrário que se faz necessário, pois a projeção perspectiva não preserva a informação de profundidade. Logo, dois objetos que se diferenciam apenas por sua profundidade e escala aparentam ter o mesmo tamanho quando projetados em perspectiva 2D por uma câmera *pinhole*.

Tendo como $P = K[R|t]$ a matriz da câmera (ou matriz de projeção) com ambos seus parâmetros extrínsecos e intrínsecos, podem simplificar a expressão anterior conforme descreve a Equação 4.

$$sX = PW, \quad (4)$$

2.3. Geometria epipolar

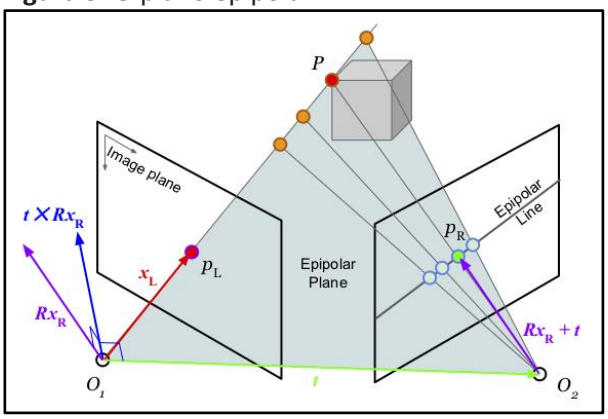
Nas subseções anteriores, foi apresentado um modelo em que se considera apenas uma câmera no sistema. Considere agora duas câmeras com origens O_1 e O_2, que têm visão do mesmo ponto real W sobre perspectivas diferentes. É possível projetar este ponto real como um ponto 2D, logo, se houverem duas câmeras neste sistema, o ponto real será projetado nas visões da esquerda e da direita como P_E e P_D, respectivamente, e que neste processo a informação da profundidade é perdida. Através da geometria epipolar, é possível impor uma restrição rígida sobre todos os pares de pontos P_E e P_D que são projeções de um ponto real.

2.3.1. O plano epipolar

Conforme discutido nessa seção, é possível "deslizar" um ponto real W sobre o eixo principal de uma câmera. À medida que ele cruza o eixo principal de uma das câmeras, sua projeção nesta câmera sempre se manterá fixa, enquanto a projeção na outra câmera irá se deslocar sobre uma reta. Dessa forma, qualquer ponto projetado no plano da câmera da esquerda vai corresponder a uma reta no plano da direita, e vice-versa. Tal reta é conhecida como linha epipolar, que se encontra sobre o plano epipolar.

O plano epipolar (Figura 3) é um plano triangular formado entre o ponto real W, e as origens O_1 e O_2 das câmeras. Além disso, verifica-se que a linha epipolar é dada pela interseção do plano epipolar com o plano da imagem.

Figura 3. O plano epipolar.



Fonte: (ESCRIVÁ *et al.*, 2019).

2.3.2. A restrição epipolar e obtenção da matriz fundamental

A matriz fundamental fornece a restrição epipolar sobre todos os pares de pontos que são projeções do mesmo ponto real sobre perspectivas diferentes em um sistema com múltiplas câmeras. Todos estes pares de pontos, portanto, devem respeitar a restrição epipolar para serem considerados pares válidos que representam o mesmo ponto real.

Conforme foi descrito nesta seção, num sistema com duas câmeras, de uma câmera para outra é aplicada uma transformação rígida que é dada pela matriz extrínseca $[R|t]$ e que os dois pontos projetados sobre o plano das duas câmeras representam o mesmo ponto real. Segundo Escrivá *et al.* (2019), o ponto da direita pode ser representado sobre as coordenadas da imagem da esquerda somando-se a componente de translação ao produto entre componente de rotação e o ponto da direita ($Rx_D + t$). Fazendo o produto vetorial $t \times Rx_D$ obtém-se um vetor perpendicular ao plano epipolar, sendo assim, tem-se que $x_E \cdot t \times Rx_D = 0$, pois o produto interno entre dois vetores perpendiculares resulta em 0. Esta é a restrição epipolar sobre o sistema, pares de pontos em câmeras diferentes

que não respeitam essa restrição, portanto, não podem ser projeções do mesmo ponto real. Indo mais a diante, pode-se tomar a forma bilinear simétrica do produto vetorial e reescrever a equação da restrição epipolar como $x_E^T[t] \times Rx_D = 0$ e então combinar o produto vetorial entre as componentes de translação e rotação em uma matriz, chegando na Equação 5, onde E é a matriz essencial que descreve a restrição epipolar para o sistema.

$$x_E^T E x_D = 0, \tag{5}$$

A matriz essencial, porém, assume que todas as câmeras estão normalizadas, com sua matriz intrínseca K igual a matriz identidade. A não ser que a câmera fotográfica usada para capturar as imagens tenha uma distância focal de exatamente 1 pixel e que seu ponto principal se localize na origem do plano da imagem (luz incide apenas sobre um dos cantos do sensor ou filme), isso não se aplica na prática. Portanto, deve-se levar em conta os parâmetros intrínsecos reais. Isso pode ser feito aplicando-se a inversa da matriz intrínseca aos pontos de ambos os lados, conforme o desenvolvimento a seguir, que resulta na Equação 6.

$$\begin{gathered}(K^{-1}x_E)^T E K^{-1} x_D \\ x_E^T K^{-1} E K^{-1} x_D \\ x_E^T F x_D = 0,\end{gathered} \tag{6}$$

Esta nova matriz F é a matriz fundamental, que define a restrição epipolar levando em conta os parâmetros intrínsecos reais da câmera empregada na captura das imagens.

2.3.3. Triangulando os pontos 3D

Considerando que os eixos principais que passam pelos pontos projetados nas imagens convergem exatamente no ponto real, é possível, portanto, usá-los para triangular as coordenadas tridimensionais originais do ponto.

Uma das formas que existem para resolver o problema da triangulação de um ponto 3D é o método linear direto, onde são igualadas as equações de projeção de ambos os pontos 2D projetados, já que ambas compartilham o mesmo ponto real. Dessa forma, dadas a matrizes de projeção P_E e P_D dos pontos projetados na imagem da esquerda e da direita, respectivamente, e assumindo que estes pontos respeitam a restrição epipolar tem-se a Equação 7.

$$P_E W = P_D W, \tag{7}$$

Com isso, tem-se um sistema de equações lineares homogêneo que pode ser resolvido para encontrar as coordenadas de W.

2.4. Correspondência de *features* em imagens digitais

Uma *feature* (ou ponto chave) de uma imagem digital é um padrão ou região da imagem que se diferencia de seus vizinhos em uma ou mais características, tais como cor, intensidade ou orientação (TUYTELAARS; MIKOLAJCZYK, 2008). Para comparar estas *features*, é necessário um descritor, um valor numérico obtido através de medições feitas sobre uma região próxima à *feature*.

O processo de extração e descrição de *features* em imagens digitais e suas subsequentes correspondências entre pares de imagens é o que possibilita a aplicação dos conceitos de geometria epipolar, vistos anteriormente nesta seção, no problema da obtenção dos pontos 3D originais. No contexto de reconstrução 3D, cada *feature* de uma correspondência representa a projeção de um ponto real sobre visões diferentes. É este conceito que permite ser feita a aplicação da restrição epipolar sobre os pares de *features*, a obtenção da matriz extrínseca que descreve a posição de uma câmera em relação a outra, e o uso de técnicas de triangulação para a obtenção das coordenadas do ponto 3D.

A seguir, são apresentados alguns algoritmos de detecção de *features* utilizados neste trabalho, e uma visão geral do algoritmo de busca dos k-vizinhos mais próximos (knn) aplicado ao problema da correspondência de *features*.

2.4.1. SIFT (*Scale-Invariant Feature Transform*)

Proposto por Lowe (2004), o algoritmo SIFT apresenta um método para extrair *features* altamente distintas, invariantes à escala e rotação da imagem. O algoritmo basicamente está disposto em quatro etapas principais: encontrar pontos chave em potencial através da diferença de filtros Gaussianos aplicados sobre diferentes escalas da imagem de entrada; filtragem dos pontos chave com pouco contraste em sua vizinhança ou que estão localizados sobre bordas; atribuição de orientação aos pontos chave restantes; e, por fim, computação dos descritores de cada ponto chave.

Para a computação dos descritores, é tomada uma vizinhança de 16x16 pixels ao redor do ponto chave, que é então subdividida em 16 blocos de 4x4 pixels. Para cada bloco é calculado um histograma de orientação de 8 *bins*. Portanto, ao final tem-se um descritor com $4 \times 4 \times 8 = 128$ valores, que pode ser representado como um vetor linha de 128 dimensões.

2.4.2. SURF (*Speeded Up Robust Features*)

O algoritmo SURF, proposto por Bay *et al.* (2008) é parcialmente inspirado no SIFT e, similarmente, extrai *features* invariantes à escala e rotação da imagem. Segundo os autores, o algoritmo é capaz de se aproximar ou até mesmo superar outros algoritmos similares em respeito à repetibilidade, distinção e robustez das *features* extraídas, porém, podendo ser computadas e comparadas muito mais rapidamente.

O algoritmo trabalha sobre imagens integrais (LEWIS, 1995) para obter custos computacionais menores e seu detector é baseado em determinantes de matrizes Hessianas. Os descritores são obtidos a partir da transformada de Haar, podendo ter comprimento de 64 ou 128 dimensões.

2.4.3. AKAZE (*Accelerated*-KAZE)

Proposto por Alcantarilla *et al.* (2013), o algoritmo AKAZE busca a detecção e descrição de *features* invariantes à escala e rotação da imagem em espaços de escala não lineares de maneira eficaz e extremamente eficiente em termos de tempo de computação. Tais espaços de escala não lineares são construídos com a utilização de um esquema denominado *Fast Explicit Diffusion* (FED), descrito no trabalho de Grewenig, Weickert e Bruhn (2010).

Assim como o SURF, o detector baseia-se em determinantes de matrizes Hessianas. Os descritores são binários e o tamanho da sequência de bits varia de acordo com a imagem de entrada. O método empregado para a computação dos descritores proposto pelos autores é conhecido como M-LDB (*Modified-Local Difference Binary*), uma versão modificada do método LDB (*Local Difference Binary*), proposto por Yang e Cheng (2012).

2.4.4. *Brute-Force Macther* (BFM)

Esta é uma abordagem amplamente utilizada para identificar *features* correspondentes em um par de imagens. Consiste em método de força bruta onde, para cada *feature* na imagem A, buscam-se as *features* na imagem B cujos descritores mais se aproximam do descritor da *feature* em A e, geralmente, para a formação de um par correspondente elege-se a *feature* em B mais próxima da *feature* em A.

Tal busca é feita através da criação de uma vizinhança ao redor da *feature* em A povoada pelos k-descritores de B com valores mais próximos do descritor em A, onde k é uma constante arbitrária que delimita o número máximo de *features* na vizinhança. Para a tarefa de construção dessa vizinhança, é necessário o emprego de uma métrica de distância para medir a magnitude dos vetores que compõe os descritores. Neste trabalho, foi empregado o uso das medidas Norma-L0 (distância Hamming) e Norma-L1 (distância Manhattan).

2.4.5. Medidas de distância entre descritores

Em álgebra linear, uma norma refere-se à magnitude (comprimento) de todos os vetores não nulos de um espaço vetorial.

Matematicamente, a Norma-L0 não é uma norma em si, mas sim a quantidade de elementos não nulos de um vetor. Para comparação de descritores binários como os do AKAZE, a distância Hamming pode ser utilizada. A distância Hamming de duas sequências de bits de mesmo tamanho é definida pelo número de posições em que seus bits diferem. Essa distância pode ser implementada contando-se o número de bits 0 obtidos de uma operação XOR *bitwise* sobre os dois descritores que se desejam comparar.

Uma das normas que podem ser utilizadas na comparação de descritores numéricos como os do SIFT e SURF é a Norma-L1, mais conhecida como distância Manhattan ou distância *Taxicab*. O nome desta medida está relacionado à distância que um taxi deve percorrer sobre uma grade retangular de ruas para viajar do ponto X ao ponto Y. Matematicamente, essa distância é computada somando-se os comprimentos de todos os vetores que passam por este caminho, de acordo com a Equação 8.

$$\|x\| := \sum_{i=1}^{n} |x_i|, \qquad (8)$$

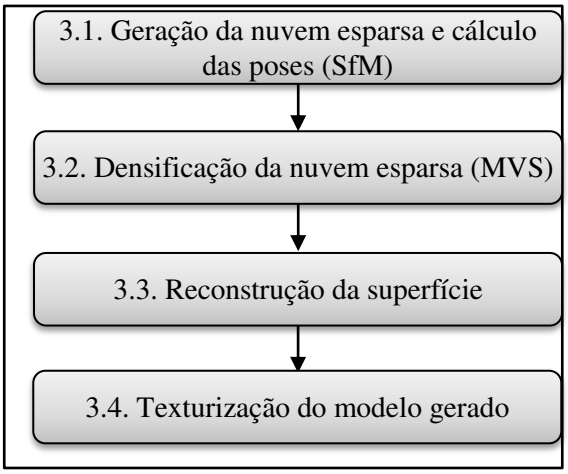
3. METODOLOGIA

A metodologia empregada no *pipeline* proposto está disposta em quatro etapas distintas, descritas a seguir:

i. Geração da nuvem de pontos esparsa e estimação das poses de cada câmera;
ii. Densificação da nuvem esparsa;
iii. Reconstrução da malha triangular da superfície a partir da nuvem densa;
iv. E por fim, opcionalmente, texturização da superfície.

O fluxograma apresentado na Figura 4 ilustra as etapas do *pipeline*.

Figura 4. *Pipeline* para reconstrução 3D a partir de imagens digitais.



Fonte: Os autores.

Foram implementadas ferramentas englobando as etapas 1ª (geração da nuvem esparsa e estimação das poses) e 3ª (reconstrução da superfície) do *pipeline*. Para os estágios restantes, fez-se o uso das ferramentas disponibilizadas pela biblioteca OpenMVS (OPENMVS, 2022), especificamente, "*DensifyPointCloud*", "*ReconstructMesh*" e "TextureMesh".

3.1. Geração da nuvem esparsa e estimação das poses das câmeras

Nesta etapa foi aplicada uma técnica de reconstrução conhecida como *Structure from Motion* (SfM) que utiliza conceitos da geometria epipolar para triangular pontos 3D a partir da correspondência de pares de pontos 2D presentes em imagens de diferentes perspectivas de uma mesma cena.

O processo tem início na correspondência de *features* entre pares de imagens do *dataset*, após várias etapas de filtragens sobre as correspondências obtidas inicialmente, são construídas as *tracks* (um único ponto de interesse visto sobre várias visões), e finalmente, estas são usadas para calcular as poses das câmeras em relação à origem e triangular os pontos 3D a partir de suas projeções 2D em diferentes visões usando geometria epipolar. Com isso obtém-se uma nuvem de pontos que representa a cena e uma sequência de matrizes extrínsecas.

3.1.1. Extração e correspondência de *features* das imagens

Esta etapa tem seu início com a detecção de *features* e a subsequente computação de seus descritores para cada imagem do *dataset*. Na literatura, existem vários algoritmos que podem ser utilizados para tais fins. Neste trabalho, foram utilizados alguns dos extratores de *features* mais robustos disponibilizados pela biblioteca OpenCV (OPENCV, 2022), mais especificamente, AKAZE, SIFT e SURF.

Tendo computado os descritores para todas as imagens, pode-se prosseguir com a correspondência (*matching*) das *features* extraídas. Para este objetivo, foi utilizado o *matcher* de força bruta fornecido pelo OpenCV, que compara cada imagem com todas as subsequentes. Esse processo é repetido até que seja iterado sobre todo o conjunto de imagens. Dependendo do tipo do descritor empregado pelo algoritmo usado para extrair as *features*, a obtenção das k melhores correspondências para um dado descritor é diferente. Caso o descritor possua valores numéricos (SIFT e SURF) o cálculo é feito com a distância Manhattan (Norma-L1), já para descritores binários (AKAZE), a distância Hamming (Norma-L0) é utilizada.

Após o *matching* inicial, é feito um teste de proporção sobre as *features* correspondentes em potencial. São buscadas as duas *features* da imagem B que melhor correspondem à *feature* de busca da imagem A, se estas duas *features* da imagem B estiverem muito próximas uma da outra, ou seja, forem muito similares, a correspondência em potencial é descartada a fim de evitar confusão em relação a qual *feature* encontrada na imagem B realmente corresponde à *feature* de busca da imagem A.

Prosseguindo, é feita uma filtragem de reciprocidade, onde uma correspondência (com teste de proporção) é considerada válida se, e somente se, ela for uma correspondência simétrica, ou seja, se uma *feature* da imagem A corresponde à outra de B e essa *feature* em B corresponde à mesma *feature* em A.

Por último, mais uma etapa de filtragem de correspondências em que é aplicada a

restrição epipolar definida pela matriz fundamental. Tal restrição determina que cada ponto x_E na imagem da esquerda deve ser observado na imagem da direita como um ponto x_D que passa pela linha epipolar, e pode ser denotada por $x_E^T F x_D = 0$, onde F é a matriz fundamental. O não respeito à essa restrição indica que um dos pontos não passa pela linha epipolar do outro e, portanto, muito provavelmente não são uma boa correspondência e podem contribuir para ruído. A aplicação da restrição epipolar é feita a partir da estimação da matriz fundamental usando o algoritmo RANSAC (FISCHLER; BOLLES, 1981) sobre os pontos das correspondências (com *threshold* máximo de 3 pixels de distância da linha epipolar para ser considerado um *inlier* e nível de confiança de 99%) e verificando a proporção de *inliers* para *outliers*. Se, ao final, a razão entre o número de *inliers* obtidos pelo RANSAC e o número de correspondências originais for menor que um dado *threshold* ($0{,}5$ por padrão), o par de imagens é descartado e suas *features* não serão utilizadas na construção das *tracks*. Um exemplo de todo o processo pode ser visualizado na Figura 5.

Figura 5. Correspondências iniciais sobre um par de imagens, seguido dos resultados das etapas de filtragem por teste de proporção, filtro de reciprocidade e aplicação da restrição epipolar, respectivamente.



Fonte: Os autores.

3.1.2. Construção das *tracks*

Como dito anteriormente, uma *track* simplesmente rastreia um único ponto do objeto visto sobre várias visões diferentes. *Tracks* são especialmente importantes neste trabalho, pois com o módulo "*sfm*" do OpenCV, é possível reconstruir a cena a partir delas. Essa abordagem permite um maior controle sobre todo o processo de correspondência das *features* e a obtenção de melhores resultados finais do que se tivesse apenas fornecidas as imagens diretamente para o módulo.

Para simplificar o processo de obtenção das *tracks* é construído um grafo de correspondência, onde os vértices representam as *features* extraídas e as arestas representam as correspondências de uma *feature* ao longo das imagens do *dataset*. Para encontrar uma *track*, basta percorrer o caminho de componentes conectadas ao primeiro vértice que representa um ponto de interesse em questão. Como são garantidas correspondências simétricas 1-para-1 através da filtragem de reciprocidade feita na etapa anterior (Subseção 3.1.1), este grafo pode ser unidirecional, facilitando sua construção. Para a construção dos grafos de correspondências, foi utilizada a implementação da estrutura de dados que utiliza uma lista de adjacência disponível na biblioteca Boost (BOOST, 2022) no módulo "*Boost Graph Library*". Na Figura 6 são demonstradas algumas das *tracks* encontradas em um dos *datasets* utilizados.

Figura 6. Algumas das *tracks* que rastreiam um ponto de interesse em um dos *datasets* utilizados para testes.



Fonte: Os autores.

3.1.3. Triangulação dos pontos 3D e cálculo das poses de cada câmera

O último passo desta etapa é delegado ao módulo *sfm* do OpenCV, que em sua implementação utiliza uma versão mínima do *libmv*, uma biblioteca para reconstrução 3D que é integrada ao software de computação gráfica Blender.

Para este passo, é necessário saber a matriz de valor intrínsecos K da câmera usada para capturar as imagens. Caso estes valores não sejam conhecidos de antemão, e como eles serão refinados durante o processo de SfM, pode ser obtida uma aproximação inicial destes valores a partir das dimensões das imagens. A distância focal f pode ser igualada à altura ou largura da imagem em pixels, qualquer que for maior. Os deslocamentos cx e cy do ponto principal podem ser obtidos assumindo-se que este encontra-se exatamente no centro do plano da imagem. Sendo assim, sejam w e h a largura e altura das imagens em pixels, respectivamente, a estimação da matriz intrínseca é dada pela Equação 8.

$$K = \begin{bmatrix} max\|w,h\| & 0 & w/2 \\ 0 & max\|w,h\| & h/2 \\ 0 & 0 & 1 \end{bmatrix}, \quad (8)$$

Após a estimação dos valores intrínsecos iniciais (caso necessário), seus valores, em conjunto com as *tracks* recuperadas, são encaminhadas para o módulo *sfm* do OpenCV, que, utilizando de princípios da geometria epipolar, irá triangular pontos 3D para cada *track*, refinar a matriz intrínseca e recuperar as componentes de rotação e translação de cada câmera em relação à origem (assume-se que a primeira câmera está posicionada na origem) que compõem as matrizes extrínsecas $[R|t]$.

Somente caso o algoritmo obtenha sucesso em reconstruir a cena a partir de todas as câmeras, a nuvem de pontos é exportada como um arquivo ".obj" ou ".ply".

Porém, como a nuvem gerada é esparsa, os resultados são pouco impressionantes e sua gama de aplicações é pequena. Na próxima subseção, é abordada uma metodologia para densificar a nuvem de pontos que pode ser então introduzida à um algoritmo de reconstrução de superfície para se obter a cena reconstruída final.

3.2. Densificação da nuvem esparsa

Na segunda etapa do *pipeline*, foi utilizada a ferramenta "*DensifyPointCloud*", fornecida pelo projeto OpenMVS para gerar uma nuvem de pontos densa através de técnicas de *Multi-View Stereo* (MVS) partindo da nuvem de pontos esparsa e das informações extrínsecas das câmeras obtidas durante o processo de SfM. Esta ferramenta do OpenMVS implementa o algoritmo *PatchMatch* (BARNES *et al.*, 2009) para densificar a nuvem esparsa, de forma a obter uma nuvem completa e que mais se aproxime dos objetos na cena possível.

Para usar a ferramenta, primeiramente, os resultados obtidos na primeira etapa devem ser dispostos no formato esperado. São adicionadas três informações à interface: parâmetros intrínsecos da câmera, a pose de cada câmera e, por fim, a nuvem de pontos esparsa.

Começando pelas informações intrínsecas, estas são compartilhadas por todas as câmeras e logo, são adicionadas apenas uma vez à interface. O formato necessita das dimensões da imagem em pixels e da matriz intrínseca.

Para as poses de cada câmera, o formato não usa a matriz extrínseca $[R|t]$ inteira, utilizando apenas a componente de rotação R diretamente e o centro da câmera c. O centro da câmera pode ser obtido aplicando-se a rotação

inversa sobre a componente de translação t de acordo com a Equação 9.

$$c = -R^T t, \quad (9)$$

Por fim, a nuvem de pontos esparsa é adicionada à interface, juntamente da informação de cor de cada ponto.

3.3. Reconstrução da superfície

No processo de reconstrução da malha triangular da superfície dos objetos em cena a partir da nuvem de pontos densa, foram empregadas três abordagens diferentes; triangulação por projeção gulosa, reconstrução Poisson, e a ferramenta "*ReconstructMesh*". Sendo as duas primeiras possuindo implementação na biblioteca Point Cloud Library (PCL, 2022) e incorporadas à uma ferramenta própria, e a última implementada em outra ferramenta do OpenMVS.

3.3.1. Triangulação por projeção gulosa

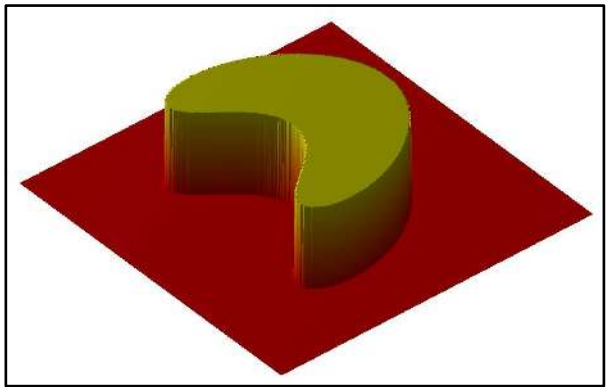
Este método, proposto por Marton *et al.* (2009), funciona sobre um princípio de crescimento de superfície incremental. O algoritmo tem seu início selecionando-se os vértices de um triângulo inicial e conectando-se pontos a ele até que não haja mais pontos disponíveis ou não seja mais possível criar mais triângulos válidos. No segundo caso, um novo triângulo inicial é criado na parte não conectada e o algoritmo é reiniciado. Neste contexto, triangulação se refere à conexão de pontos da nuvem, criando triângulos e recriando uma superfície que se aproxima o mais fiel possível do original.

3.3.2. Reconstrução Poisson

Essa abordagem, apresentada no trabalho de Kazhdan *et al.* (2006), expressa o problema da reconstrução da superfície para uma nuvem de pontos como a solução de uma equação de Poisson.

O método busca se aproximar da função indicadora que modela a superfície do objeto partindo dos vetores normais orientados dos pontos da nuvem (Figura 7).

Figura 7. Exemplo de função indicadora plotada no gráfico.



Fonte: (WIKIMEDIA COMMONS, 2022).

3.3.3. ReconstructMesh

A ferramenta "*ReconstructMesh*" do OpenMVS implementa o algoritmo proposto por Jancosek e Pajdla (2014) para reconstruir a superfície original dos objetos em cena.

Ao invés de modelar a superfície a partir dos pontos da nuvem, o algoritmo modela o espaço vazio em cena baseando-se na visibilidade dos pontos. A superfície é estimada verificando-se onde existe transição entre o espaço vazio modelado e seu complemento, que representa um espaço preenchido.

3.4. Texturização da superfície

Para este passo final e opcional, fez-se o uso da ferramenta "*TextureMesh*" do OpenMVS, que implementa o algoritmo descrito no trabalho de Waechter *et al.* (2014), capaz de gerar a textura para modelos reconstruído de grande escala partindo das imagens de entrada. Como este processo depende especificamente do resultado da reconstrução da superfície no formato da biblioteca, somente será possível utilizar o algoritmo de texturização sobre a malha obtida pela ferramenta "*ReconstructMesh*".

4. EXPERIMENTOS E RESULTADOS

Para a implementação de parte das ferramentas do *pipeline* proposto, foi utilizada a linguagem de programação *C++* (*standard* C++20) estendida pelas bibliotecas *OpenCV 3.4* (compilada com os módulos extra "*sfm*" e "*xfeatures2d*"), *OpenMVS 2.0.1*, *Boost 1.71* e *PCL 1.12.1*, todas disponíveis para uso acadêmico de forma livre e gratuita conforme suas respectivas licenças. Os binários das bibliotecas e executáveis das ferramentas foram construídos utilizando o compilador *gcc 11.1.0* para a plataforma GNU/Linux *x86-64*. Mais informações e instruções

detalhadas sobre todo o processo de compilação das bibliotecas e ferramentas podem ser acessadas no repositório online <link omitido para revisão às cegas>.

Todos os experimentos foram realizados em um computador disposto de uma CPU Intel Core i5-3210M com *clock* em 2,50 GHz, 6 GB de RAM DDR3 com *clock* em 1.333 MHz, placa de vídeo NVIDIA GeForce GT 630M com 4 GB de memória de vídeo, em uma instalação 64-bits do SO Ubuntu 20.04 virtualizada sobre o *Windows Subsystem for Linux* (WSL2) em uma instalação 64-bits do SO *Windows 10 Pro* (compilação do sistema operacional 19043.1706).

4.1. Considerações sobre as imagens de entrada

Para a realização dos testes, foi utilizada uma combinação de *datasets* disponíveis na internet, produzidos especificamente com reconstrução 3D em mente, e fotos capturadas pelos autores.

As imagens de entrada devem ser obtidas sequencialmente ao redor do objeto ou cena que se deseja reconstruir, sempre se atentando para manter uma variação pequena de perspectiva entre as imagens, a fim de preservar pontos chave suficientes entre uma imagem e a subsequente. Não é necessário ter visão de todos os ângulos do objeto nas imagens de entrada, entretanto, nota-se que será impossível triangular pontos de faces que não estão presentes nas imagens. Todas as fotos do *dataset* devem estar localizadas dentro do mesmo diretório. Um fator importante que deve ser verificado é que os arquivos de imagem devem ser renomeados de forma que a sequência em que as fotos foram obtidas seja estritamente preservada quando ordenadas de forma crescente por seus nomes, caso contrário, o algoritmo de correspondência implementado para a primeira etapa do processo pode obter resultados não ideais para a estimação da matriz fundamental e subsequente triangulação dos pontos 3D.

Para a visualização dos resultados, é recomendado o uso de uma ferramenta capaz de visualizar objetos 3D que não possuem faces nem arestas (nuvem de pontos). Neste trabalho foi utilizado o *MeshLab*, uma ferramenta para manipulação de nuvens de pontos e malhas de superfície de objetos 3D.

4.2. Datasets

Foram utilizados três *datasets* para a realização dos experimentos, sendo que dois deles estão disponíveis na internet e o terceiro é de elaboração dos autores. A seguir, são detalhados cada um dos *datasets* utilizados.

O primeiro *dataset*, disponível em https://github.com/openMVG/SfM_quality_evaluation/tree/master/Benchmarking_Camera_Calibration_2008/fountain-P11/images, intitulado "*fountain-P11*" consiste em 11 imagens de 3072x2048 pixels de dimensão (Figura 8).

Figura 8. Exemplos de imagens do *datasets* "*fountain-P11*".



Fonte: Os autores.

O segundo *dataset* está disponível em https://github.com/TrainingByPackt/Building-Computer-Vision-Projects-with-OpenCV-4-and-CPP/tree/master/Chapter_14/crazyhorse e é intitulado "*crazyhorse*". Trata-se de uma sequência de sete fotos que retratam o Memorial *Crazy Horse*, um monumento em construção sobre a face de uma montanha localizada no condado de *Custer*, Dakota do Sul, Estados Unidos (Figura 9). As fotos possuem dimensões 1024x768 pixels e as informações intrínsecas da câmera são desconhecidas.

Figura 9. Exemplos de imagens do *datasets* "*crazyhorse*".



Fonte: Os autores.

Por fim, o *dataset* "*deck*", elaborado pelos autores deste trabalho, consiste em seis imagens que retratam bancos e uma mesa de madeira. As imagens têm 4624x3472 pixels de dimensão e estão disponibilizadas em <link omitido para revisão às cegas>.

Figura 10. Exemplos de imagens do *datasets* "*deck*".



Fonte: Os autores.

4.3. Experimentos

Os experimentos foram realizados através da execução de cada etapa proposta para o *pipeline*, partindo do conjunto de imagens de cada *dataset* e executando a próxima etapa, tomando os resultados da anterior como entrada. Foi levado em consideração a combinação de diferentes abordagens para as etapas de geração da nuvem esparsa e reconstrução da superfície.

Para cada *dataset*, os seguintes procedimentos foram realizados:

i. Geração da nuvem esparsa usando três algoritmos de detecção de *features* diferentes: AKAZE, SIFT e SURF;
ii. Utilização da ferramenta "*DensifyPointCloud*" sobre a nuvem esparsa gerada de cada método da etapa anterior;
iii. Para a obtenção da superfície reconstruída, sobre as nuvens densas, foram utilizados três métodos: reconstrução Poisson, triangulação por projeção gulosa e "*ReconstructMesh*";
iv. Ao final, a ferramenta "*TextureMesh*" foi utilizada para geração e aplicação da textura sobre o resultado do método "*ReconstructMesh*".

Com o intuito de manter a legibilidade das tabelas apresentadas (Tabela 1, 2 e 3), alguns métodos tiveram seus nomes abreviados: *DensifyPointCloud* (*Dense*), reconstrução Poisson (Poisson) e triangulação por projeção gulosa (*Greedy*).

4.4. Resultados

Para fins de comparação dos resultados foi utilizada uma abordagem qualitativa sobre a fidelidade do objeto adquirido ao final de cada experimento em relação à cena original, bem como uma abordagem quantitativa sobre o número de pontos da nuvem esparsa, número de pontos da nuvem densa, número de vértices da superfície, número de faces da superfície e tempo de execução de cada método para cada passo.

4.4.1. Geração da nuvem de pontos esparsa

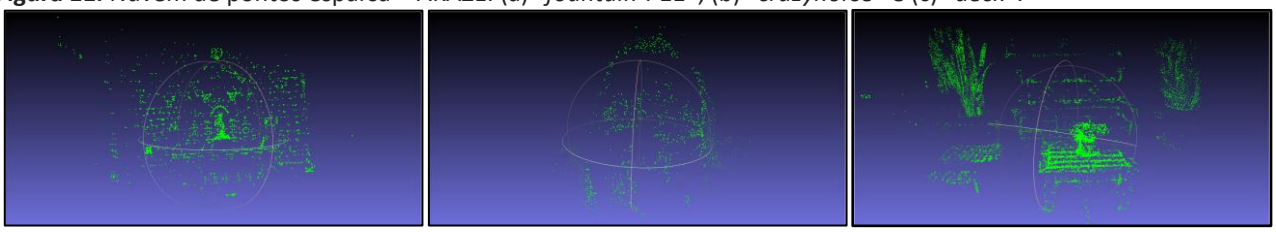
Nesta seção são apresentados os resultados dos experimentos realizados sobre o método de geração da nuvem de pontos esparsa por meio da utilização do algoritmo SfM implementado. Foram escolhidos os algoritmos de detecção AKAZE, SIFT e SURF para extrair os pontos chave e sua subsequente comparação entre pares de imagens. Na Tabela 1 e na Figura 11 são apresentados os resultados obtidos nessa etapa, comparando o tempo de execução e número de pontos na nuvem esparsa gerada. Nota-se que para o *dataset* "*deck*", os testes realizados com os algoritmos SIFT e SURF não conseguiram reconstruir a cena para todas as visões e seus resultados foram desconsiderados para as próximas etapas, portanto, somente os resultados do detector AKAZE darão prosseguimento no *pipeline* para este *dataset*.

Tabela 1. Resultados da geração da nuvem esparsa.

Dataset	Método	Tempo de execução	Número de pontos
fountain-P11	AKAZE	1m 30s	4.768
	SIFT	6m 41s	8.809
	SURF	39m 15s	38.766
crazyhorse	AKAZE	18s	1.521
	SIFT	48s	2.667
	SURF	55s	3.843
deck	AKAZE	8m 54s	16.970
	SIFT	38m 25s	11.127[1]
	SURF	82m 50s	20.803[1]
	[1]Para estes dados, não foi possível triangular os pontos de todas as câmeras, portanto, serão desconsiderados para as próximas etapas.		

Fonte: Os autores.

Figura 11. Nuvem de pontos esparsa – AKAZE. (a) "*fountain-P11*", (b) "*crazyhorse*" e (c) "*deck*".



(a) (b) (c)

Fonte: Os autores.

4.4.2. Densificação da nuvem esparsa

A seguir, na Tabela 2 e Figura 12, estão dispostos os resultados dos experimentos realizados sobre a segunda etapa do *pipeline*, comparando o tempo de execução e número de pontos obtidos para a nuvem densa. Nesta etapa, foi executada a ferramenta "*DensifyPointCloud*" sobre os arquivos obtidos ao final da etapa anterior.

Tabela 2. Resultados da geração da nuvem densa.

Dataset	Método	Tempo de execução	Número de pontos
fountain-P11	AKAZE + Dense	16m 35s	2.479.740
	SIFT + Dense	17m 03s	2.618.556
	SURF + Dense	16m 13s	1.935.326[1]
crazyhorse	AKAZE + Dense	3m 59s	261.391
	SIFT + Dense	3m 43s	264.060
	SURF + Dense	3m 42	261.547
deck	AKAKE + Dense	20m 49s	1.440.754
	[1]Diferentemente dos outros resultados, a combinação SURF + Dense sobre o *dataset* fountain-P11 obteve comparativamente menos pontos		

Fonte: Os autores.

Figura 12. Nuvem de pontos densa – AKAZE + *DensifyPointCloud*. (a) "*fountain-P11*", (b) "*crazyhorse*" e (c) "*deck*".



(a) (b) (c)

Fonte: Os autores.

4.4.3. Reconstrução da superfície

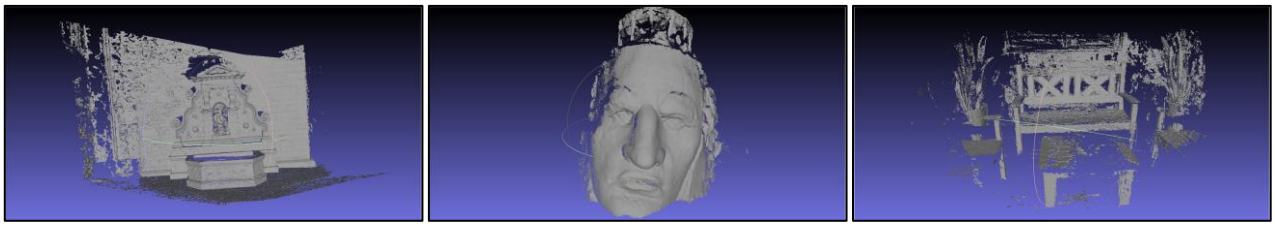
Para a realização dos experimentos da terceira etapa do *pipeline*, foram utilizados os métodos "*ReconstructMesh*", triangulação por projeção gulosa e reconstrução Poisson para a geração da superfície. A tabela a seguir (Tabela 3) compara os resultados em termos do tempo de execução, número de vértices (pontos conectados a uma face) e número de faces. Mais adiante, nas Figura 13, 14 e 15, é feito um comparativo qualitativo dos modelos obtidos por cada método.

Tabela 3. Resultados da reconstrução da superfície.

Dataset	Método	Tempo de execução	Número de vértices	Número de faces
fountain-P11	AKAZE + Dense + *ReconstructMesh*	2m 33s	938.569	1.876.554
	AKAZE + Dense + *Greedy*	2m 43s	2.479.740	4.637.689
	AKAZE + Dense + Poisson	52s	303.782	607.476
	SIFT + Dense + *ReconstructMesh*	6m 29s	988.220	1.975.741
	SIFT + Dense + *Greedy*	1m 25s	2.618.556	5.064.581
	SIFT + Dense + Poisson	51s	324.253	648.386
	SURF + Dense + *ReconstructMesh*	3m 23s	763.436	1.526.390
	SURF + Dense + *Greedy*	2m 19s	1.935.326	3.476.326
	SURF + Dense + Poisson	52s	356.227	712.370
crazyhorse	AKAZE + Dense + *ReconstructMesh*	10s	32.276	64.436
	AKAZE + Dense + *Greedy*	11s	261.391	451.358
	AKAZE + Dense + Poisson	19s	172.293	344.363
	SIFT + Dense + *ReconstructMesh*	12s	33.714	67.315
	SIFT + Dense + *Greedy*	16s	264.060	398.670
	SIFT + Dense + Poisson	4s	1.763	3.496
	SURF + Dense + *ReconstructMesh*	10s	33.596	67.098
	SURF + Dense + *Greedy*	13s	261.547	403.594
	SURF + Dense + Poisson	3s	1.971	3.906
deck	AKAZE + Dense + *ReconstructMesh*	2m 31s	671.029	1.340.697
	AKAZE + Dense + *Greedy*	1m 39s	1.440.754	2.652.736
	AKAZE + Dense + Poisson	1m 12s	582.524	1.164.278

Fonte: Os autores.

Figura 13. Superfície reconstruída – AKAZE + *DensifyPointCloud* + *GreedyProjection*. (a) "*fountain-P11*", (b) "*crazyhorse*" e (c) "*deck*".



(a) (b) (c)

Fonte: Os autores.

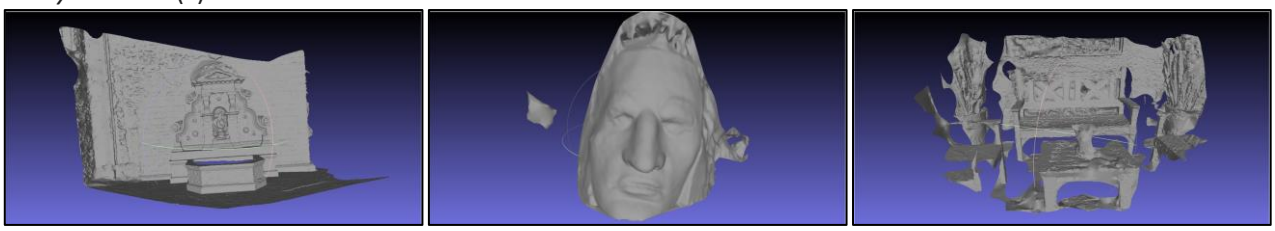
Figura 14. Superfície reconstruída – AKAZE + *DensifyPointCloud* + Poisson. (a) "*fountain-P11*", (b) "*crazyhorse*" e (c) "*deck*".



(a) (b) (c)

Fonte: Os autores.

Figura 15. Superfície reconstruída – AKAZE + *DensifyPointCloud* + *ReconstructMesh*. (a) "*fountain-P11*", (b) "*crazyhorse*" e (c) "*deck*".



(a) (b) (c)

Fonte: Os autores.

4.4.4. Texturização da superfície

Por fim, a ferramenta "*TextureMesh*" gera e aplica uma textura sobre o objeto obtido pelo método *ReconstructMesh*. Nota-se que o número de faces e vértices do modelo texturizado é exatamente o mesmo do modelo sem textura. Na Figura 16 é apresentando um exemplo de resultado da aplicação da textura sobre a superfície reconstruída nos três *datasets* utilizados.

Figura 16. Aplicação da textura sobre a superfície reconstruída – AKAZE + *DensifyPointCloud* + *ReconstructMesh* + *TextureMesh*. (a) "*fountain-P11*", (b) "*crazyhorse*" e (c) "*deck*".



(a) (b) (c)

Fonte: Os autores.

5. CONCLUSÕES

Neste trabalho, foi proposto um encadeamento de soluções computacionais (*pipeline*) empregado na obtenção de um modelo 3D que represente a cena observada por um conjunto de fotografias digitais capturadas pela mesma câmera com visão dos mesmos objetos sobre perspectivas diferentes. Foram

desenvolvidas duas ferramentas com interface de linha de comando (CLI): uma que implementa um algoritmo *Structure from Motion* (SfM) para geração de uma nuvem de pontos e estimação da pose da câmera que capturou cada imagem; e outra que reconstrói a superfície dos objetos em cena, empregando os algoritmos de triangulação por projeção gulosa e reconstrução Poisson.

5.1. Discussão dos resultados

A análise dos resultados experimentais mostra que o *pipeline* proposto consegue gerar modelos 3D que representam a cena e objetos observados com alta fidelidade, especialmente para o método de reconstrução de superfície *ReconstructMesh*, que mostrou obter um bom balanço entre tempo de execução e a qualidade da malha gerada. Porém, o tempo total de execução é relativamente alto para imagens de entrada com dimensões maiores que 1024x768 pixels.

Na etapa de geração da nuvem esparsa e estimação das posições, o método SURF foi capaz de recuperar o maior número de pontos, entretanto seu tempo de execução foi ordens de magnitude maior que o dos métodos AKAZE e SIFT.

Dentro do mesmo *dataset*, foi possível notar que, indiferentemente do número de pontos da nuvem esparsa e do método para obtê-la, o processo de densificação da nuvem obteve tempo de execução e quantidade de pontos densos muito similares. Foi possível verificar também que, em contraste aos resultados obtidos pela mesma combinação nos outros *datasets*, a combinação "AKAZE + *Dense*" sobre o dataset "fountain-P11" obteve a nuvem densa com a menor quantidade de pontos.

A respeito da reconstrução da superfície, o tempo de execução é diretamente proporcional ao número de pontos da nuvem densa de entrada. Dentre os três métodos, a reconstrução por Poisson foi o que obteve melhores resultados em relação ao tempo de execução (sendo trivial para nuvens pouco densas, e variando entre 51s e 1m 12s para volumes maiores), porém suas malhas reconstruídas possuem a menor resolução, mesmo a superfície não apresentando falhas.

O método *GreedyProjection* preserva todos os pontos originais da nuvem densa e, logo, apresenta excelente resolução e preservação de detalhes, porém, a superfície resultante é de qualidade pobre pois se fazem presentes diversas falhas onde o algoritmo não conseguiu propagar triângulos novos.

5.2. Conclusão e trabalhos futuros

Como mencionado anteriormente, o método *ReconstructMesh* obteve o melhor balanço entre riqueza de detalhes e qualidade da superfície gerada, assim como habilita a possibilidade de aplicação de textura sobre o modelo gerado, aumentando ainda mais a fidelidade. Entretanto, foi a abordagem que obteve maior tempo de execução médio.

Durante o processo de elaboração de um *dataset* de teste pelos autores, notou-se que todo o *pipeline* é extremamente sensível a qualidade do conjunto de imagens de entrada. Levando em consideração a natureza encadeada de um *pipeline*, qualquer falha apresentada em uma das etapas é propagada às próximas. Foi possível verificar que *datasets* que possuem muitas vistas de um mesmo objeto muito próximo produzem uma nuvem de pontos esparsa que, mesmo sendo fiel ao objeto em questão, não é adequada à etapa de reconstrução da superfície e apresenta falhas durante o processo de densificação. Entretanto, conjuntos de imagens que representam uma cena em sua totalidade, observada de uma distância maior, apresentam melhores resultados.

Em trabalhos futuros, seria de interesse propor um método mais amplo que funcione em ambos os casos descritos no parágrafo anterior, a fim de se ampliarem as aplicações dos modelos obtidos ao final.

REFERÊNCIAS

ALCANTARILLA, P.; NUEVO, J.; BARTOLI, A. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In: BRITISH MACHINE VISION CONFERENCE, 2013, Bristol, Reino Unido, **Procedings [...].** Bristol: British Machine Vision Association 2013, p. 13.1-13.11.. https://dx.doi.org/10.5244/c.27.13.

BARNES, C.; SHECHTMAN, E.; FINKELSTEIN, A.; GOLDMAN, D. B. PatchMatch: a randomized correspondence algorithm for structural image editing. **Acm Transactions On Graphics**, v. 28, n. 3, p. 1-11, 27 jul. 2009. https://dx.doi.org/10.1145/1531326.1531330.

BAY, H.; ESS, A.; TUYTELAARS, T.; VAN GOOL, L. Speeded-Up Robust Features (SURF). **Computer**

Vision and Image Understanding, v. 110, n. 3, p. 346-359, jun. 2008. https://dx.doi.org/10.1016/j.cviu.2007.09.014.

BERNADIN, S. L.; THARPE, A.; WAGNER, T.; DAUPIN, R.; CARDENAS, N.. Work-in-progress: designing a 3d scanner for digital reconstruction of rare cultural artifacts. **Southeastcon 2015**, abr. 2015. IEEE. https://dx.doi.org/10.1109/secon.2015.7132878.

BOOST. **The Boost Graph Library (BGL)**. Disponível em: https://www.boost.org/doc/libs/1_79_0/libs/graph/doc/index.html. Acesso em: 10 jun. 2022.

EHTEMAMI, A.; PARK, S. B.; BERNADIN, S.; LESCOP, L.; CHIN, A.. Overview of Visualizing Historical Architectural Knowledge through Virtual Reality. **Southeastcon, 2021**,10 mar. 2021. IEEE. https://dx.doi.org/10.1109/southeastcon45413.2021.9401850.

ESCRIVÁ, D. M.; JOSHI, P.; MENDONÇA, Vinícius G.; SHILKROT, R. **Building Computer Vision Projects with OpenCV 4 and C++**: implement complex computer vision algorithms and explore deep learning and face detection. [S. l.]: Packt Publishing, 2019.

FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. **Communications Of The Acm**, v. 24, n. 6, p. 381-395, jun. 1981. https://dx.doi.org/10.1145/358669.358692.

GREWENIG, S.; WEICKERT, J.; BRUHN, A. From Box Filtering to Fast Explicit Diffusion. **Lecture Notes In Computer Science**, p. 533-542, 2010. https://dx.doi.org/10.1007/978-3-642-15986-2_54.

HARTLEY, R.; ZISSERMAN, A. **Multiple View Geometry in Computer Vision**. 2. ed. Cambridge, Reino Unido: Cambridge University Press, 2004. https://doi.org/10.1017/CBO9780511811685

INDIEWIRE, **'Rogue One': How ILM Created CGI Grand Moff Tarkin and Princess Leia**. Disponível em: https://www.indiewire.com/2017/01/rogue-one-visual-effects-ilm-digital-grand-moff-tarkin-cgi-princess-leia-1201766597/. Acesso em: 12 out. 2020.

JANCOSEK, M.; PAJDLA, T. Exploiting Visibility Information in Surface Reconstruction to Preserve Weakly Supported Surfaces. **International Scholarly Research Notices**, v. 2014, p. 1-20, ago. 2014. https://dx.doi.org/10.1155/2014/798595.

KAZHDAN, M.; BOLITHO, M.; HOPPE, H. Poisson surface reconstruction. **Sgp '06**: *In*: EUROGRAPHICS SYMPOSIUM ON GEOMETRY PROCESSING, 4. 2006. **Proceedings** […]. Association for Computing Machinery (ACM), . 2006. p. 61-70https://dl.acm.org/doi/10.5555/1281957.1281965.

LEWIS, J. P. Fast Template Matching. **Proceedings of Vision Interface 95**, Quebec, Canada, p. 120-123, 15 maio 1995.

LI, G-K; GAO, Fan; WANG, Zhi-Gang. A photogrammetry-based system for 3D surface reconstruction of prosthetics and orthotics. **2011 Annual International Conference Of The Ieee Engineering In Medicine And Biology Society**, [S.L.], ago. 2011. IEEE. https://dx.doi.org/10.1109/iembs.2011.6092087.

LOWE, D. G. Distinctive Image Features from Scale-Invariant Keypoints. **International Journal Of Computer Vision**, v. 60, n. 2, p. 91-110, nov. 2004.. https://dx.doi.org/10.1023/b:visi.0000029664.99615.94

MARTON, Z. C.; RUSU, R. B.; BEETZ, M. On fast surface reconstruction methods for large and noisy point clouds. **2009 Ieee** INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2009. Anais […].Institute of Electrical and Electronics Engineers (IEEE). 2009.p. 3218-3223, https://dx.doi.org/10.1109/ROBOT.2009.5152628

MCGLONE, J. C. **Manual of Photogrammetry**. 6. ed. [S. l.]: American Society For Photogrammetry And Remote Sensing (Asprs), 2013. 1372 p.

OPENCV. **OpenCV**. Disponível em: https://opencv.org. Acesso em: 10 jun. 2022.

OPENMVS. **OpenMVS**. Disponível em: https://github.com/cdcseacave/openMVS. Acesso em: 10 jun. 2022.

PCL. **Point Cloud Library**. Disponível em: https://pointclouds.org. Acesso em: 10 jun. 2022.

SIMEK, K. Dissecting the Camera Matrix. **Sightations,** 14 ago. 2012. Disponível em: https://ksimek.github.io/2012/08/14/decompose/. Acesso em: 10 jun. 2022

STRAITS Reserch. **3D Reconstruction Technology Market**. [S. L.], 12 abril 2019. Disponível em: https://straitsresearch.com/report/3d-reconstruction-technology-market. Acesso em: 26 set. 2022.

TUYTELAARS, T; MIKOLAJCZYK, K. Local Invariant Feature Detectors: a survey. **Foundations And Trends® In Computer Graphics And Vision**, [S.L.], v. 3, n. 3, p. 177-280, 15 jun. 2008. Now Publishers. https://dx.doi.org/10.1561/0600000017.

WAECHTER, M.; MOEHRLE, N.; GOESELE, M. Let There Be Color! Large-Scale Texturing of 3D Reconstructions. **Computer Vision – Eccv 2014**, [S.l.], p. 836-850, 2014. Springer International Publishing. https://dx.doi.org/10.1007/978-3-319-10602-1_54.

WIKIMEDIA FOUNDATION. **Wikimedia Commons**. 2022. Disponível em: https://commons.wikimedia.org/wiki/Main_Page. Acesso em: 10 jun. 2022.

YANG, X.; CHENG, K-T. LDB: an ultra-fast feature for scalable augmented reality on mobile devices. *In***:** INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY (ISMAR**)**, 2012. **Anais** […]. IEEE, 2012, p. 49-57, https://dx.doi.org/10.1109/ismar.2012.6402537