



UNIVERSIDADE DO ESTADO DO
RIO DE JANEIRO

IPRJ
Instituto Politécnico
Universidade do Estado do Rio de Janeiro

INSTITUTO POLITÉCNICO
GRADUAÇÃO EM ENGENHARIA
DE COMPUTAÇÃO

Pedro Felipe Pena Barata

Experimentos com técnicas atuais de reconstrução 3D visando a
digitalização prática de esculturas suaves do Jardim do Nêgo

Nova Friburgo

2017



UNIVERSIDADE DO ESTADO DO
RIO DE JANEIRO

IPRJ
Instituto Politécnico
Universidade do Estado do Rio de Janeiro

INSTITUTO POLITÉCNICO
GRADUAÇÃO EM ENGENHARIA
DE COMPUTAÇÃO

Pedro Felipe Pena Barata

Experimentos com técnicas atuais de reconstrução 3D visando a digitalização
prática de esculturas suaves do Jardim do Nêgo

Trabalho de Conclusão de Curso apresentado,
como requisito parcial para obtenção
do título de Graduado em Engenharia de
Computação, ao Departamento de Modela-
gem Computacional do Instituto Politécnico,
da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Dr. Ricardo Fabbri

Nova Friburgo

2017

Pedro Felipe Pena Barata

**Experimentos com técnicas atuais de reconstrução 3D visando a digitalização
prática de esculturas suaves do Jardim do Nêgo**

Trabalho de Conclusão de Curso apresentado, como requisito parcial para obtenção do título de Graduado em Engenharia de Computação, ao Departamento de Modelagem Computacional do Instituto Politécnico, da Universidade do Estado do Rio de Janeiro.

Aprovada em 23 de Novembro de 2017.

Banca Examinadora:

Prof. Dr. Ricardo Fabbri (Orientador)
Instituto Politécnico – UERJ

Prof. Dr. Edirlei Everson Soares de Lima
Instituto Politécnico – UERJ

Prof. Dr. Roberto Pinheiro Domingos
Instituto Politécnico – UERJ

Nova Friburgo
2017

AGRADECIMENTOS

A Deus por ter me dado saúde e força para superar as dificuldades.

À minha família por sempre me apoiar, até nos momentos mais dificeis.

À UERJ e todo seu corpo docente, além da direção e administração, que mesmo sem ter condições ideais de funcionamento, realizam seu trabalho com tanto amor e dedicação, trabalhando incansavelmente para que nós, alunos, possamos contar com um ensino de extrema qualidade.

Ao meu professor e orientador Ricardo Fabbri, pelo suporte no pouco tempo que lhe coube, pelas suas correções e incentivos.

Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito.

Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes.

Martin Luther King

RESUMO

BARATA, Pedro Felipe Pena. *Experimentos com técnicas atuais de reconstrução 3D visando a digitalização prática de esculturas suaves do Jardim do Nêgo*. 2017. 96 f. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação), Instituto Politécnico, Universidade do Estado do Rio de Janeiro, Nova Friburgo, 2017.

A área de reconstrução 3D vem sendo amplamente explorada. Sensores de profundidade, tanto aéreos quanto terrestres, têm sido empregados em diferentes aplicações já há muito tempo. Entretanto, constantes melhorias na tecnologia, sobretudo, no *hardware* e software no âmbito da reconstrução, fizeram com que nas últimas duas décadas, novas técnicas surgissem. Muitos cientistas que utilizavam a fotogrametria pura – isto é, usando sensores de imagens convencionais – haviam convertido seus esforços à área dos sensores a laser. Além de executarem uma reconstrução mais confiável de objetos com baixa textura, tais sensores possuem altíssima acurácia, compensando seu alto custo inicial. Isto dificultou e desacelerou o processo de descoberta de novos algoritmos e métodos em fotogrametria pura. Graças a avanços recentes, a fotogrametria, aliada a novos algoritmos, como o *Structure from Motion* (SfM), pontos de interesse e o casamento entre imagens, por exemplo, consegue competir com escaneadores a laser e sensores de profundidade. As características mais bem-reconhecidas das técnicas recentes, no entanto, são a robustez, flexibilidade, e praticidade, não sendo claro se na prática podem competir com escaneadores a laser quanto à precisão em aplicações de interesse. Neste trabalho, investiga-se a praticidade de técnicas atuais combinando SfM e MVS (*Multi-View Stereo*), utilizando softwares como o MVE (**FUHRMANN; LANGGUTH; GOESELE, 2014**) e o VisualSfM (**WU et al., 2011b**). Em particular, relata-se um estudo de até onde é possível chegar apenas utilizando-se a câmera de um *smartphone*, visando aplicação futura à preservação completa do patrimônio do Jardim do Nêgo, em Nova Friburgo. Essa tecnologia é confrontada com outros tipos de técnicas empregadas em grandes projetos, como o Kinect, da Microsoft, e sua aplicação em SfM, bem como técnicas clássicas de escaneamento a laser, tais como as empregadas no projeto *Digital Michelangelo* de preservação de esculturas liderado pela Universidade de Stanford.

Palavras-chave: Fotogrametria. Estrutura por movimento. Estereoscopia multiocular. Reconstrução 3D.

ABSTRACT

BARATA, Pedro Felipe Pena. *Experiments with current 3D reconstruction techniques towards practical digitalization of smooth sculptures from Jardim do Nêgo*. 2017. 96 f. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação), Instituto Politécnico, Universidade do Estado do Rio de Janeiro, Nova Friburgo, 2017.

The field of 3D reconstruction has been widely explored. Range sensors, both aerial and terrestrial, have long been employed in different applications. However, constant improvements in technology, especially in hardware and software in the context of reconstruction, have led to the emergence of new techniques in the past two decades. Many scientists that used pure photogrammetry – *i.e.*, using conventional imaging sensors – have converted their efforts into the area of laser sensors. Such sensors, beyond providing a more reliable reconstruction for objects with low texture information, provide high accuracy, justifying their initial cost. This hindered and slowed down the process of discovering new algorithms and methods in pure photogrammetry. Thanks to recent advances, photogrammetry, coupled with new algorithms such as Structured Motion (SfM), interest points and image matching, for instance, can compete with laser scanners and other range sensors. While Recent techniques are well-known for robustness, flexibility and ease of use, it is not yet clear whether they can compete with laser scanners in terms of precision in applications of interest. In this work, we investigate the practical applicability of current techniques combining SfM and MVS (*Multi-View Stereo*), using softwares such as MVE (**FUHRMANN; LANGGUTH; GOESELE, 2014**) and VisualSfM (**WU et al., 2011b**). In particular, we study how far can one go by using a standard smartphone camera, aiming at a future application to the complete preservation of the sculpture garden Jardim do Nêgo at Nova Friburgo. This technology will be confronted against other types of techniques employed in major projects, such as Microsoft's Kinect and its application in SfM, as well as classical techniques of laser scanning of sculptures, such as those employed in the Digital Michelangelo project, lead by Stanford University.

Keywords: Photogrammetry. Structure from motion. Multiview stereo. 3D reconstruction.

LISTA DE FIGURAS

Figura 1 - Exemplos de Kinect	14
Figura 2 - Reconstrução 3D com o Kinect	15
Figura 3 - Protótipo do escaneador a laser do projeto clássico <i>Digital Michaelangelo</i>	16
Figura 4 - Algumas esculturas do Jardim do Nêgo	17
Figura 5 - Um projetor de padrões utilizando um laser visível.	20
Figura 6 - Escaneamento de David	22
Figura 7 - Escultura “Noite”, de Michelangelo	22
Figura 8 - Alguns tipos de cinzeis que provavelmente foram usados por Michelangelo na escultura de ”St. Matthew”.	24
Figura 9 - Imagem de um Kinect V1 aberto	30
Figura 10 - Saída de uma imagem interpretada pelo Kinect	30
Figura 11 - Representação visual do acerto do deslocamento	33
Figura 12 - Ambiente de calibração do modelo geométrico	33
Figura 13 - Imagens iniciais para reconstrução 3D usando Kinect com câmeras	34
Figura 14 - Resultados da reconstrução SfM	35
Figura 15 - Funcionamento do Kinect Fusion	35
Figura 16 - Processo de escaneamento 3D no Kinect Fusion	36
Figura 17 - Procedimento da maioria dos sistemas SfM	38
Figura 18 - Além dessas etapas, geralmente a reconstrução é densificada usando-se MVS.	38
Figura 19 - Imagens de correspondências do SIFT	39
Figura 20 - Exemplo de filtro gaussiano	41
Figura 21 - Exemplo do resultado obtido do histograma de orientações do gradiente.	42
Figura 22 - Exemplo de um descriptor de pontos SIFT	43
Figura 23 - Exemplo de triangulação	44
Figura 24 - Uma triangulação utilizando um ponto qualquer, X_j . Onde cada câmera C_1, C_2, C_3 possui um feature correspondente a cada uma delas, respectivamente, X_{1j}, X_{2j}, X_{3j}	44
Figura 25 - Funcionamento do MVE	46
Figura 26 - Funcionamento da paralaxe	48
Figura 27 - Interface gráfica (UMVE)	50
Figura 28 - Reconstrução de nuvem de pontos gerada a partir de SfM do MVE	51
Figura 29 - Representação do mapa de profundidade do MVS	53
Figura 30 - Interface gráfica do VisualSfM	55

Figura 31 - Problema de <i>Bundle Adjustment</i>	57
Figura 32 - Descrição da etapa de filtragem	65
Figura 33 - Como foi realizada a varredura das esculturas	67
Figura 34 - Reconstrução esparsa da escultura do Jardim do Nêgo no VisualSfM com 197 imagens.	70
Figura 35 - Resultados da reconstrução densa da escultura do Jardim do Nêgo . . .	70
Figura 36 - Reconstrução esparsa do objeto no VisualSfM com 200 imagens. . . .	71
Figura 37 - Reconstrução densa do objeto no VisualSfM com 200 imagens. . . .	72
Figura 38 - Reconstrução esparsa do objeto com 224 imagens no VisualSfM. . . .	73
Figura 39 - Caso de falha do VisualSfM	74
Figura 40 - Resultado das reconstruções densas do objeto	75
Figura 41 - Procedimento do MVE	76
Figura 42 - Extensão <i>EXIF</i>	76
Figura 43 - Reconstrução usando a interface gráfica do MVE	77
Figura 44 - Linha de comandos do <i>sfmrecon</i>	78
Figura 45 - Malha com ruídos proveniente do comando <i>fssrecon</i>	79
Figura 46 - Resultado final, após a remoção dos ruídos da malha.	80
Figura 47 - Resultado da etapa <i>fssrecon</i> do MVE	81
Figura 48 - Resultado da etapa <i>meshclean</i> , da etapa anterior 47	82
Figura 49 - Caixa do Sense	83
Figura 50 - O Sense	84
Figura 51 - Tela inicial do Sense	85
Figura 52 - Interface do Sense	86
Figura 53 - Resultados do escaneamento com o Sense	87
Figura 54 - Resultados do escaneamento com o Sense	87

LISTA DE TABELAS

Tabela 1 - Métricas do projeto de reconstrução da escultura David	26
Tabela 2 - Valores de deslocamentos e sua média	32
Tabela 3 - Resultados dos testes executados no ambiente descrito anteriormente	34
Tabela 4 - Tempos obtidos da reconstrução da escultura do Jardim do Nêgo usando o VisualSfM	69
Tabela 5 - Tempos obtidos da reconstrução do objeto, com 200 imagens usando o VisualSfM	71
Tabela 6 - Tempos obtidos da reconstrução do objeto, com 224 imagens usando o VisualSfM	72
Tabela 7 - Tempos obtidos usando o MVE em um conjunto de dados do Jardim do Nêgo	78
Tabela 8 - Tempos obtidos usando o MVE em um conjunto de dados em ambiente interno com 200 imagens	80

SUMÁRIO

	INTRODUÇÃO	12
1	RECONSTRUÇÃO 3D ATIVA CLÁSSICA	20
1.1	Introdução	20
1.2	Projeto de preservação das esculturas de Michelangelo	20
1.2.1	<u>Descrição geral</u>	20
1.2.2	<u>Calibração</u>	22
1.2.3	<u>Procedimento de digitalização</u>	24
2	TÉCNICAS DE LUZ ESTRUTURADA E <i>TIME OF FLIGHT</i>	27
2.1	Introdução	27
2.2	Kinect	28
2.2.1	<u>Introdução</u>	28
2.2.2	<u>Processo de calibração</u>	30
2.2.3	<u>Uso do Kinect com <i>Structure from Motion</i></u>	34
3	RECONSTRUÇÃO 3D PASSIVA	38
3.1	Introdução	38
3.2	SIFT – <i>Scale Invariant Feature Transform</i>	39
3.2.1	<u>Detecção de extremos de espaço-escala</u>	40
3.2.2	<u>Atribuição de orientação</u>	41
3.2.3	<u>Descriptor de pontos</u>	42
3.2.4	<u>Casamento de pontos</u>	43
3.3	Triangulação – <i>Full pair-wise image matching</i>	43
3.4	MVE – <i>Multi-View Reconstruction Environment</i>	44
3.4.1	<u>Introdução</u>	44
3.4.2	<u>Guia de reconstrução com o MVE</u>	46
3.4.2.1	<u>Criando uma cena</u>	48
3.4.3	<u>Reconstrução SfM</u>	50
3.4.3.1	<u>Estereoscopia Multiocular – <i>Multi-View Stereo (MVS)</i></u>	51
3.4.3.2	<u>Reconstrução de Superfícies – <i>Surface Reconstruction</i></u>	53
3.5	VisualSfM	54
3.5.1	<u>Introdução</u>	54
3.5.2	<u>Procedimento</u>	55
3.5.3	<u>Bundle Adjustment</u>	55
3.5.3.1	<u>PBA/MCBA – <i>Multi-Core Bundle Adjustment</i></u>	57
3.5.3.2	<u>CMVS/PMVS-2 (<i>Clustering Views for Multi-view Stereo / Patch-based for Multi-view Stereo version 2</i>)</u>	60
3.5.3.3	<u>PMVS-2</u>	63

4	EXPERIMENTOS	67
4.1	Procedimento	67
4.1.1	<u>Resultados da reconstrução com o VisualSfM</u>	68
4.1.2	<u>Resultados da reconstrução com o MVE</u>	73
4.1.3	<u>Resultados da reconstrução com o Sense</u>	82
	CONCLUSÃO	89
	REFERÊNCIAS	93

INTRODUÇÃO

Introdução e Justificativa

A reconstrução 3D de cenas gerais a partir de múltiplos pontos de vista usando-se câmeras convencionais, sem aquisição controlada, é um dos grandes objetivos de pesquisa em visão computacional, ambicioso até mesmo para os dias de hoje. Aplicações incluem a reconstrução de modelos 3D para uso em videogames ([ABLAN, 2007](#)), filmes ([ABLAN, 2007](#)), arqueologia, arquitetura, modelagem 3D urbana (*e.g.*, Google Streetview); técnicas de *match-moving* em cinematografia para fusão de conteúdo virtual e filmagem real ([DOB-BERT, 2012](#)), a organização de uma coleção de fotografias com relação a uma cena (*e.g.*, o sistema *Phototourism* ([AGARWAL et al., 2010](#)) e a funcionalidade *Look Around* do Google Panoramio e Steet View), manipulação robótica, e a metrologia a partir de câmeras na indústria automobilística e metal-mecânica.

Os desafios estão ligados às escolhas de grande escala de representações adequadas e de técnicas que possam modelar simultaneamente com materiais drasticamente diferentes (*e.g.*, não-Lambertianos), modelos geométricos (*e.g.*, variedades curvilíneas gerais, descontinuidades, texturas, deformações, em escalas diferentes), tipos de regiões (com ou sem textura), condições de iluminação variadas, sombras, fortes diferenças de perspectivas, desbalanceamento devido a excesso de detalhes em partes menos importantes, número arbitrário de objetos e câmeras não-calibradas.

Mesmo que um sistema completo esteja fora do alcance da tecnologia atual, um progresso significativo tem sido atingido nos últimos anos. Por um lado, uma tecnologia operacional tem evoluído, mais recentemente para sistemas de grande escala ([AGARWAL et al., 2011](#)), a partir do desenvolvimento da detecção robusta de *features* ([MIKOLAJCZYK, 2002](#)), o *fitting/ajuste* robusto e seleção de correspondências baseados em RANSAC, e o desenvolvimento de métodos de geometria projetiva para calibrar duas ou três imagens e progressivamente adicionar imagens e extrair estrutura 3D dessas *features* na forma de nuvens de pontos. Com o código fonte do sistema Bundler ([SNAVELY et al., 2010](#)) libe-

rado por Noah Snavely, e sua subsequente incorporação ao sistema VisualSfM (WU et al., 2011b), torna-se possível tentar utilizar este sistema para a reconstrução de patrimônio cultural em larga escala, como um jardim de esculturas. Um dos objetivos do presente trabalho é estudar até onde se pode chegar com a aplicação de tais técnicas recentes à futura reconstrução completa do Jardim do Nêgo, em Nova Friburgo, como parte de um projeto maior.

No paradigma usando-se apenas imagens convencionais – denominado **reconstrução estéreo multiocular passiva** – a posição das câmeras são estimadas a partir apenas de imagens, usando pontos de interesse; em seguida, uma nuvem de pontos é reconstruída, Figuras 28, 38 e 36. As câmeras podem então ser utilizadas para obter modelos mais detalhados de reconstrução, como algoritmos de densificação (FURUKAWA; PONCE, 2007) e interpolação (HAIGHT, 1967) da nuvem de pontos, bem como demais algoritmos densos de visão estéreo multi-perspectiva/multi-ocular, como os do grupo de Michel Goesele (FUHRMANN; LANGGUTH; GOESELE, 2014), também com código disponível. Tais algoritmos, no entanto, apresentam problemas, em particular a reconstrução suaviza partes bem-delineadas do objeto, e pode conter buracos em áreas homogêneas. Pode-se, portanto, utilizar no futuro a reconstrução 3D de curvas desenvolvida pelo grupo (USUMEZBAS; FABBRI; KIMIA, 2016; FABBRI; KIMIA, 2016; FABBRI; KIMIA, 2010; FABBRI; GIBLIN; KIMIA, 2012) para auxiliar na reconstrução mais bem-delinada nesses casos problemáticos, bem como para ajudar no problema de escalabilidade quando a reconstrução 3D se torna muito grande.

Um segundo paradigma, denominado **reconstrução estéreo multiocular ativa**, tem se tornado economicamente viável devido à indústria de videogames, e consiste na utilização de sistemas que alteram o funcionamento de câmeras convencionais, típicamente usando-se projetores infra-vermelho, laser ou câmeras ToF (*Time of Flight*), como no caso dos dispositivos Kinect, Figura 1.

A preservação de patrimônio tem sido realizada tradicionalmente com escaneadores dedicados de alto custo, como no famoso projeto de escaneamento *in situ* da escultura David, chamado *Digital Michelangelo* (LEVOY et al., 2000), Figura 3. O projeto teve

Figura 1 - Exemplos de Kinect



Legenda: (a) - Kinect de primeira geração; (b) - projetor infra-vermelho; (c) - tecnologia ToF;
 (d) - escaneamento manual.

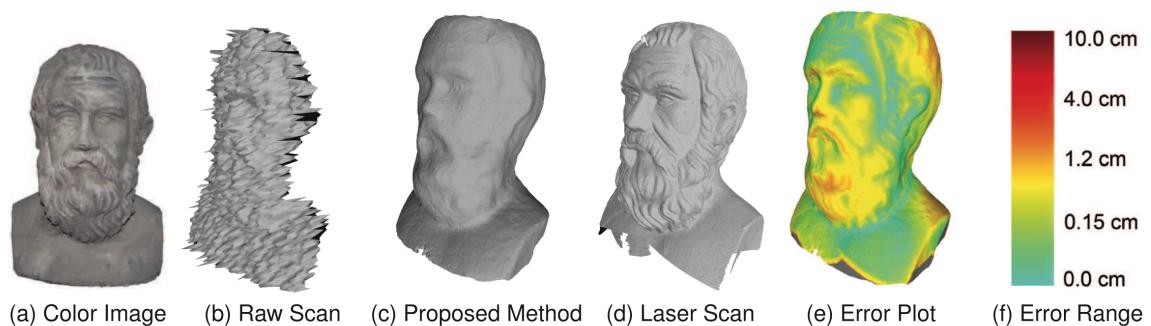
Nota: Kinects de primeira geração consistem em câmeras e projetores de infra-vermelho, os de segunda geração em tecnologia ToF. Ambos Kinects são largamente utilizados para escaneamento em tempo real, formando a base de escaneadores manuais, porém nem sempre são úteis para preservação detalhada de patrimônio.

Fonte: [SMISEK; JANCOSÉK; PAJDLA, 2013](#).

início em 1992 e tem como objetivo a utilização de escaneadores a laser de profundidade (*Rangefinder Scanners*), aliado a algoritmos que combinam diferentes profundidades e cores da imagem, para realizar uma digitalização da parte externa e da superfície de forma acurada da estátua de David. Note-se, porém, que esse método pode ser utilizado em diferentes objetos no mundo real, como partes de máquinas, artefatos culturais e na indústria de video games, por exemplo. Para as partes mais detalhadas, foi utilizado um escaneador de menor escala que faz uma pequena triangulação com laser de profundidade.

Mais recentemente, pode-se considerar tecnologias mais acessíveis, similares às de altíssimo custo do projeto *Digital Michelangelo* e popularizadas na última década pela indústria de entretenimento, notadamente pelo projeto Natal/Kinect ([SMISEK; JANCOSEK; PAJDLA, 2013](#); [WANG et al., 2015](#)). A reconstrução usando-se Kinect (de primeira ou segunda geração) usando software atual de super-resolução, é inferior à de um sistema a laser de alta qualidade, sendo, porém de baixo custo e muito mais versátil devido ao sistema de aquisição manual e ao software amplamente utilizado e desenvolvido ([WANG et al., 2015](#)), Figura 2.

Figura 2 - Reconstrução 3D com o Kinect

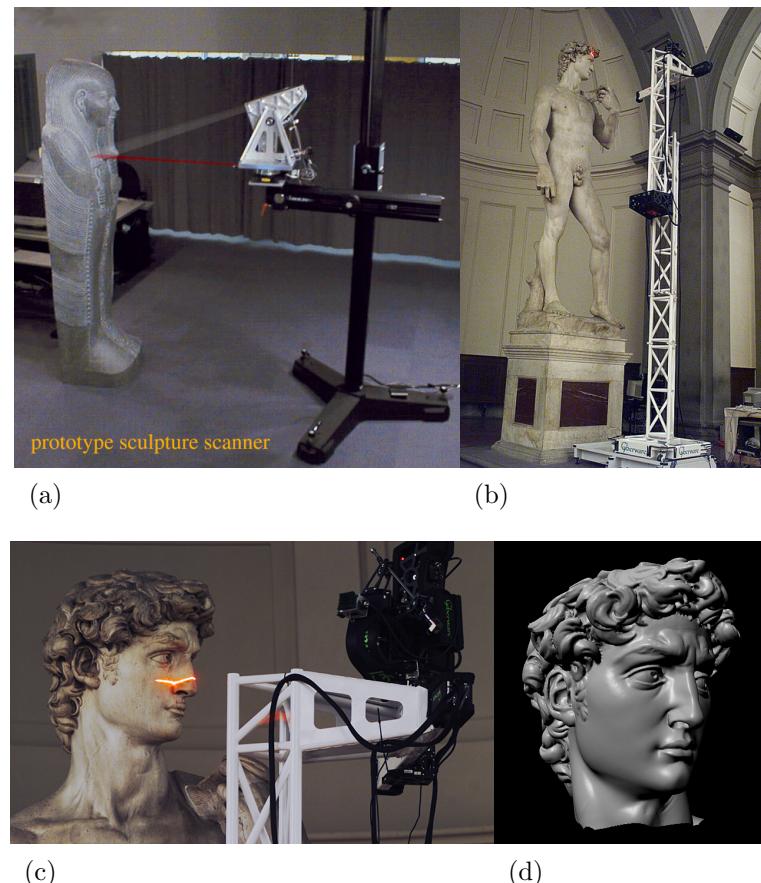


Legenda: (a) - imagem colorida; (b) - escaneamento bruto; (c) - método proposto; (d) - escanamento à laser; (e) - erro.

Nota: A reconstrução usando-se Kinect (de primeira ou segunda geração) usando software atual de super-resolução fornece precisão similar a um sistema estéreo de média resolução, inferior um sistema a laser de alta qualidade, porém de baixo custo e muito mais versátil.

Fonte: [WANG et al., 2015](#).

Figura 3 - Protótipo do escaneador a laser do projeto clássico *Digital Michaelangelo*



Legenda: (a) - objeto escaneado com tecnologia à laser; (b) - escaneador reconfigurado para objetos maiores; (c) - reconfiguração que faz a cabeça girar em 90 graus; (d) - reconstrução finalizada;

Nota: Um objeto típico escaneado por um sistema inicial foi uma réplica em tamanho real de um sarcófago egípcio. O escaneador foi reconfigurado para objetos maiores, pois a escultura David possui 517 centímetros, também foi reconfigurado para a cabeça, para girar em 90 graus, que faz o laser rotacionar da posição horizontal para a vertical e em torno da cabeça como um todo. Cerca de 100 *scans* diversas em posições foram alinhados automaticamente por um algoritmo ICP (*iterated-closest-points*). Após uma otimização global para minimizar erros de alinhamento toda a estátua, usa-se um algoritmo VRIP (*volumetric range image processing*)

Fonte: LEVOY et al., 2000.

O Jardim do Nêgo, Nova Friburgo

No caso de Nova Friburgo, há a necessidade redobrada de preservação de patrimônio a céu aberto, em especial devido às chuvas e deslizamentos inerentes à região. O Jardim do Nêgo consiste em grandes esculturas em encostas, cobertas por um tapete de vegetação, as quais desfrutam de grande reconhecimento regional e internacional ([WITNESS, 2014](#)), Figura 4.

Figura 4 - Algumas esculturas do Jardim do Nêgo



Fonte: O autor, 2017.

O Jardim foi idealizado e criado por Geraldo Simplicio (Nêgo), artista cearense que mora no local a mais de 30 anos, e que ganhou notoriedade por suas esculturas de barro, com traços singulares e técnicas únicas. Hoje, Nêgo – como gosta de ser chamado – trabalha para reconstruir o Jardim após a tragédia de 2011 na Região Serrana do Rio de Janeiro, onde algumas estruturas foram destruídas. Portanto, com o consentimento do Nêgo, surgiu a motivação desta pesquisa: além de explorar métodos de reconstrução, também tem o objetivo de ajudar a criar sistemas completos para eternizar um patrimônio que é reconhecido no mundo todo.

A preservação das esculturas do Jardim do Nêgo se mostra um desafio à pesquisa em reconstrução 3D, pois apresentam curvas bem delineadas, que são representadas de maneira suavizada e empobrecida por métodos convencionais. Algumas esculturas apresentam pouca textura, com apenas um leve padrão de musgo. Seria de grande interesse avaliar o potencial de técnicas atuais de reconstrução 3D geral que não exigem controle preciso de aquisição, as quais têm seu código fonte disponível na internet.

Objetivos

Pretende-se, ao longo deste projeto, ganhar experiência com técnicas modernas de reconstrução 3D fotogramétricas, no contexto de uma aplicação bem-definida de preservação de patrimônio.

O objetivo concreto é explorar as tecnologias supracitadas para um dia desenvolver um esquema completo de escaneamento de patrimônio usando software aberto, câmeras e escaneadores de baixo custo, representando o estado da arte em reconstrução 3D sem restrições de aquisição. Perguntas fundamentais a serem respondidas são: que nível de detalhe, facilidade e precisão se pode obter usando-se apenas imagens e software aberto? É possível utilizar escaneadores de baixo custo baseados em Kinect com melhorias significativas em termos de qualidade, conveniência ou tempo de processamento? Quais são as restrições desses sistemas? Seria útil, na prática, uma reconstrução de curvas para auxiliar na reconstrução de nuvem de pontos e de superfícies densas? Onde o estado da arte deve ser avançado de forma a permitir uma solução mais conveniente e completa para a preservação de patrimônio?

O principal objetivo em termos de pesquisa científica será comparar as diferentes abordagens do estado da arte disponíveis para reconstrução 3D e explicitar suas limitações práticas.

Organização deste manuscrito

Este trabalho foi estruturado da seguinte maneira: Introduzimos métodos baseados em reconstrução a laser no Capítulo 1, apresentando, de uma maneira mais técnica, o projeto *Digital Michelangelo* na Seção 1.2, que foi um dos pioneiros na técnica de conservação de acervos culturais. No próximo Capítulo 2, discutimos o uso do Kinect, da Microsoft, no que tange à calibração do sistema para o uso em tecnologias *Structure from Motion*. Adiante, no Capítulo 3, abordaremos o tema central do trabalho: técnicas de

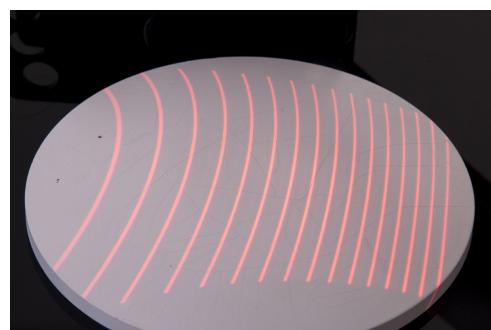
reconstrução baseadas em fotogrametria, com o emprego de dois softwares, o MVE [3.4](#) e o VisualSfM [3.5](#) e seus respectivos funcionamentos. Ao final, apresentamos experimentos e conclusões do trabalho, bem como sugestões para implementações e trabalhos futuros.

1 RECONSTRUÇÃO 3D ATIVA CLÁSSICA

1.1 Introdução

A técnica de reconstrução 3D baseada em laser é conhecida desde o século passado, pois oferece uma alta qualidade geométrica de dados, até mesmo para regiões sem textura. Neste capítulo, abordaremos o projeto clássico de escaneamento da escultura de Michelangelo, Davi, que utiliza escaneadores por laser, também relacionados a técnicas de luz estruturada e uma técnica conhecida como *Time of Flight*, ou tempo de voo, Figura 5. Esse processo se baseia em projetar um padrão conhecido (muitas vezes, grades ou barras horizontais, via laser ou luz infra-vermelho) em uma cena. A forma como o padrão se deforma quando atinge superfícies permite que sistemas de visão calculem a profundidade e informações das superfícies dos objetos na cena.

Figura 5 - Um projetor de padrões utilizando um laser visível.



Fonte: [OPTOENGINEERING, 2017](#).

1.2 Projeto de preservação das esculturas de Michelangelo

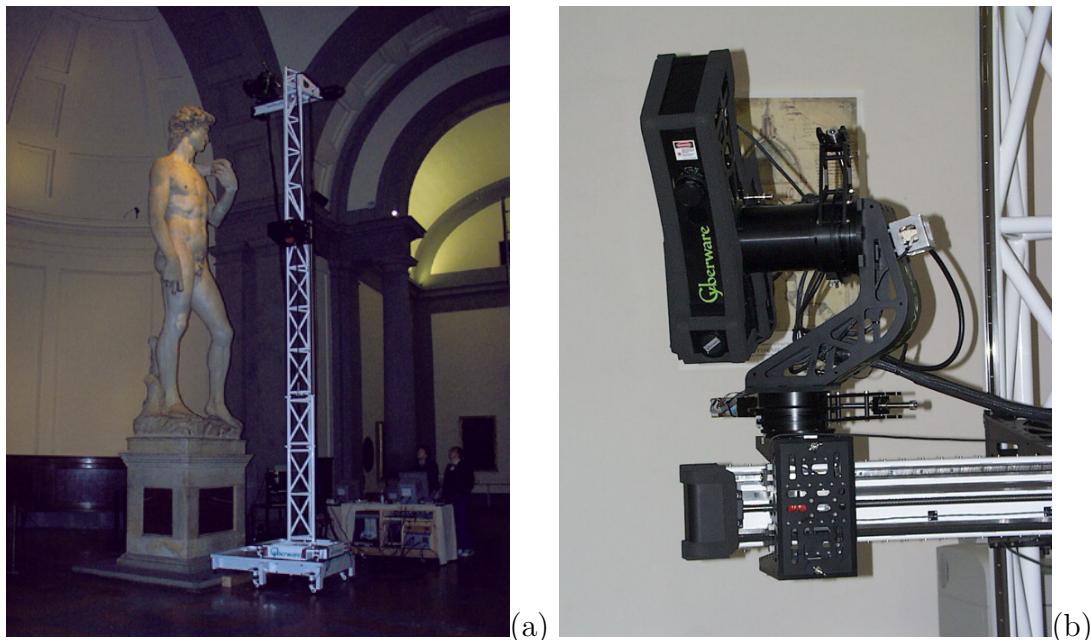
1.2.1 Descrição geral

O projeto Digital Michaelangelo ([LEVOY et al., 2000](#)) tem como motivação avançar a tecnologia de digitalização 3D e criar um acervo digital sobre alguns dos principais ar-

tefatos culturais. Foram utilizados sensores de profundidade (*range finders*) para triangulação, sensores de profundidade baseados em ToF (*Time of Flight*), câmeras digitais e um software de calibração. Uma equipe de mais de 30 professores pesquisadores, funcionários e estudantes das Universidades de Stanford e de Washington, bem como pesquisadores de empresas privadas, passaram os anos de 1998 e 1999 na Itália escaneando esculturas de Michelangelo.

O principal componente de hardware do sistema é um escaneador de triangulação a laser, que é composto por quatro eixos motorizados, um projetor a laser, uma câmera de profundidade, uma câmera de cores RGB e uma luz branca [6](#), isso tudo acrescido de suportes e estruturas para o escaneamento de estátuas grandes. O objetivo de um escaneador deste porte era capturar marcas menores que um milímetro, das ferramentas utilizadas por Michelangelo em suas esculturas, Figura [8](#). Para isso, foram testadas diversas resoluções, tendo sido decidido um espaçamento Y (ao longo da faixa do laser) de 1/4 mm e uma resolução Z (profundidade) de pelo menos duas vezes este valor. Isso resultou em uma visualização de 14 cm de largura (ao longo da faixa do laser) por 14 cm de profundidade. Caso esta resolução fosse menor, as marcas de cinzel ficariam borradas e, se fosse maior, o conjunto de dados produzido seria gigantesco. Felizmente, a maioria das estátuas feitas por Michelangelo foram esculpidas com um mármore encontrado em Carrara, chamado Statuario, uma pedra altamente uniforme, não direcional e constituída de grãos finos. Além disso, com exceção de “Noite”, figura [7](#), as esculturas não são polidas e cobertas por terra, o que aumenta a dispersão superficial e reduz a abaixo da superfície. Nesse contexto, a dispersão abaixo da superfície causa alguns problemas: não se pode assumir que a superfície é Lambertiana ideal (**BASRI; JACOBS, 2003**), o que mudou a forma com que a renderização dos modelos seria feita e diminuiu a qualidade de disposição de dados.

Figura 6 - Escaneamento de David



Legenda: (a) - escaneadores da estátua de David; (b) - escaneador principal.

Nota: O escaneamento de David foi composto por dois escaneadores: um escaneador principal (da cabeça), com estruturas para alcançar todos os pontos necessários para reconstrução, e outro para o restante da escultura.

Fonte: LEVOY et al., 2000.

Figura 7 - Escultura “Noite”, de Michelangelo



Fonte: LEVOY et al., 2000.

1.2.2 Calibração

O objetivo de calibrar o aparato era encontrar um mapeamento de coordenadas 2D no seu alcance e cores para coordenadas 3D em um quadro de referência global. Idealmente, este quadro deve ser a estátua (estacionária). Porém, na prática o aparato

tornou-se a própria referência. O mapeamento final do aparato à estátua foi realizado alinhando-se novas reconstruções com aquelas provenientes de varreduras previamente existentes, conforme detalhado no artigo original (**LEVOY et al., 2000**).

Para calibrar qualquer sistema, primeiramente escolhe-se um modelo matemático que se aproxime do comportamento do sistema, então estima-se parâmetros desse modelo medindo o comportamento do sistema. No caso do projeto de David, o modelo matemático natural era um modelo geométrico 3D parametrizado da cabeça digitalizada e do aparato. Se os componentes do sistema forem suficientemente independentes, a calibração pode ser dividida em estágios, correspondente a cada componente do sistema. Por isso o aparato foi construído pensando na rigidez (independência), pois particionar a calibração em etapas reduz o grau de liberdade em cada etapa, e com isso, o número de medidas necessárias para calibrar este estágio.

Em um sistema mecânico, tal particionamento também reduz o volume físico total no qual essas medidas de calibração devem ser tomadas, o que é relevante dado que o sistema de aquisição é bastante grande. Além disso, uma calibração em etapas é menos suscetível à propagação de erros, pois, caso uma calibração falhasse, seria apenas uma parte da calibração e não o sistema como um todo. No projeto de David, a calibração foi dividida em seis etapas distintas:

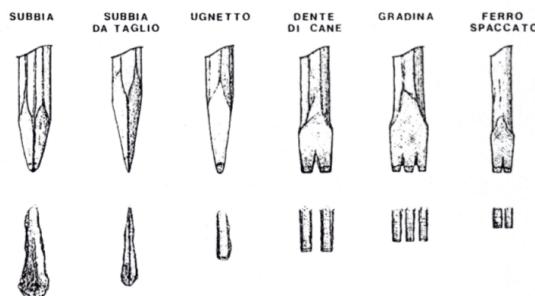
1. Um mapeamento 2D a partir de coordenadas de *pixels* das imagens da câmera de profundidade para localizações físicas na camada de laser;
2. Uma transformação rígida 2D/3D do sistema de coordenadas da camada de laser para esferas de aço anexadas à cabeça do escaneador;
3. Transformação rígida 3D para acomodar o rolamento da cabeça do escaneador em 90° (ao remontá-lo) em relação ao conjunto *pan-tilt*
4. A localização do eixo de rotação de inclinação *tilt* e o mapeamento não-linear dos comandos de movimento para ângulos de rotação físicos;
5. A localização do eixo de rotação *pan* e o mapeamento de seus comandos de movimento para ângulos de rotação físicos;

6. A localização do eixo de translação, que também depende de como o conjunto *pan-tilt* está montado no braço horizontal.

O resultado da calibração pode ser descrito como uma concatenação de seis matrizes 4x4:

$$\begin{bmatrix} \text{translação} \\ \text{horizontal} \end{bmatrix} \begin{bmatrix} \text{rotação} \\ \text{de} \\ \text{pan} \end{bmatrix} \begin{bmatrix} \text{rotação} \\ \text{de} \\ \text{tilt} \end{bmatrix} \begin{bmatrix} \text{rotação} \\ \text{de} \\ \text{rolamento} \end{bmatrix} \begin{bmatrix} \text{laser para a} \\ \text{cabeça do} \\ \text{escaneador} \end{bmatrix} \begin{bmatrix} \text{imagem} \\ \text{para o} \\ \text{laser} \end{bmatrix} .$$

Figura 8 - Alguns tipos de cincéis que provavelmente foram usados por Michelangelo na escultura de "St. Matthew".



Fonte: LEVOY et al., 2000.

1.2.3 Procedimento de digitalização

Um operador move interativamente a cabeça do escaneador através de uma sequência de movimentos, definindo os limites do volume a ser escaneado. O volume que pode ser coberto em uma única varredura foi delimitado por conta de quatro fatores:

- O campo de visão e os limites de movimento do escaneador;
- A queda na qualidade da varredura com o aumento da obliquidade do laser;
- Oclusões, tanto do laser quanto da linha de visão da câmera;
- Obstruções físicas, como paredes, estátuas ou o próprio equipamento.

Uma vez planejada a varredura, um *script* de digitalização é executado automaticamente, levando desde alguns minutos, até horas para terminar, dependendo da largura da área a ser coberta. A partir do *script*, são feitas as etapas de escaneamento de profundidade (laser) e do escaneamento da fotometria (câmera digital RGB).

Primeiramente faz-se o escaneamento fotogramétrico da geometria da escultura:

1. Alinhamento manual;
2. ICP – *Iterative Closest Point* para uma câmera existente;
3. ICP automático para todos os pares sobrepostos;
4. Relaxação global para espalhar o erro;
5. Reunir utilizando métodos volumétricos.

Após isso, ocorre o escaneamento fotométrico e o processamento das cores da escultura:

1. Compensação da iluminação do ambiente;
2. Descarte de pixels com sombra ou reflexos especulares;
3. Mapeia-se os vértices (uma cor por vértice);
4. Correção da irradiância e reflectância difusa.

Limitações:

- Inter-reflexões são ignoradas;
- Dispersões subterrâneas (*subsurface scattering*) são ignoradas;
- Tratamento de efeito difuso como Lambertiano;
- Uso de normais de superfícies suavizadas.

O projeto não teve mais nenhum avanço desde o verão de 2004, por falta de financiamento. Como resultado, modelos de alta qualidade só existem do David na resolução de 1,0 mm (56 milhões de triângulos) e São Mateus a 0,25 mm (372 milhões de triângulos).

Tabela 1 - Métricas do projeto de reconstrução da escultura David

Números de objetos escaneados	10 estátuas + 2 edificações + 1.163 fragmentos de mapa
Menor e maior objetos escaneados	1 polegada (fragmentos de mapa) e 23 pés (David)
Resolução espacial dos dados	0,29mm para geometria, 0,125mm para cor
Complexidade do maior conjunto de dados	2 bilhões de polígonos + 7.000 imagens (David)
Tamanho do maior conjunto de dados	32 <i>gigabytes</i> (David)
Quantia total de dados capturados	250 <i>gigabytes</i>
Tamanho do maior escaneador	24 pés de altura, 1.800 libras de peso
Peso total do equipamento levado para a Itália	4 toneladas
Número de pessoas envolvidas	32 (sem incluir colaboradores)
Tempo médio para escaneamento	1 semana (exceto David, que levou 1 mês)
Tempo total de escaneamento	5.00 horas de trabalho
Total de tempo para processamento de dados	4.000 horas de trabalho
Custo do projeto	\$2.000.000

Fonte: LEVOY et al., 2000.

Um modelo também existe para o Atlas em 0,25 mm (aproximadamente 500 milhões de triângulos), mas contém erros de alinhamento. Também foram disponibilizadas algumas métricas sobre este projeto na Tabela 1.

Porém, devido ao seu alto custo com equipamentos, com softwares especializados e na necessidade de estações robustas para armazenamento dos dados e para escaneamento de patrimônios, outras técnicas foram emergindo com o passar dos anos, como a fotogrametria passiva, usando apenas imagens, que será abordada ao decorrer deste manuscrito.

2 TÉCNICAS DE LUZ ESTRUTURADA E *TIME OF FLIGHT*

2.1 Introdução

A técnica de luz estruturada é um **método fotogramétrico ativo** que consegue determinar a localização de objetos no alcance de uma câmera, a partir de alterações nos padrões conhecidos de um projetor. Tipicamente, o par projetor-câmera tem a geometria análoga a duas câmeras estéreo, e opera no espectro infa-vermelho invisível a olho nu, em vez de laser, para que a aparência do objeto possa ser capturada simultaneamente por outra câmera RGB. Um exemplo recente de escaneador de luz estruturada de baixo custo é o Kinect versão 1 ([SMISEK; JANCOSEK; PAJDLA, 2013](#)), que também serve como base para uma série de scanners que utilizam o mesmo hardware. Uma vantagem clara em relação aos escaneadores a laser é a velocidade (pois mais padrões são projetados simultaneamente em uma área) e o baixo custo; uma desvantagem em potencial é a precisão, conforme discutido mais adiante. Existe também, um outro método fotogramétrico ativo denominado *Time of Flight* ([GOKTURK; YALCIN; BAMJI, 2004](#)), que em vez de padrões infra-vermelho ou laser, emite fôtons a cada pixel do sensor, que viajam na velocidade da luz. O valor resultante da diferença da emissão e do retorno dos fôtons aos detectores, na escala de picosegundos, é utilizado na criação de uma distribuição de probabilidade localizando o evento detectado a uma distância do eixo do escaneador.

Até recentemente, a técnica ToF foi extremamente cara e restrita a baixas resoluções, devido à necessidade de equipamento capaz de medir fôtons refletidos em uma escala de picosegundos. Tal técnica foi barateada em ordens de magnitude pela indústria de entretenimento, no lançamento recente do Kinect versão 2 ([LACHAT et al., 2015; VALGMA, 2016](#)). Além do baixo custo permitido pelo Kinect 2, uma vantagem desta técnica é a alta velocidade de escaneamento e maior robustez dos resultados se comparado à técnica de luz estruturada e mesmo escaneadores a laser. No entanto, ela não pode ser aplicada com confiabilidade em ambientes abertos, e não supera a precisão de um bom escaneador a laser. Note-se que, no entanto, escaneadores a laser Não são adequados para

aplicações em tempo real; porém, para o nosso objetivo de preservação de patrimônio, o requisito de escaneamento em tempo real é secundário, sendo desejável apenas para aplicações em que uma previsão rápida da reconstrução 3D é desejada pelo arqueólogo, ou para realidade aumentada. A seguir, discutimos essas tecnologias de reconstrução 3D ativas em maiores detalhes.

2.2 Kinect

2.2.1 Introdução

Criado pela Microsoft para fins recreativos (como no console de videogame XBox, por exemplo), o Kinect se tornou uma das mais conhecidas ferramentas de reconstrução 3D no cenário atual. Sua primeira versão (Kinect V1) figura 9, utiliza uma técnica similar à empregada no projeto *Digital Michelangelo* da Universidade de Stanford, com luz estruturada, porém, diferentemente dos escaners a laser, o Kinect é mais rápido e tem um custo monetário baixo, acessível ao público em geral (desde entusiastas e amadores até profissionais da área).

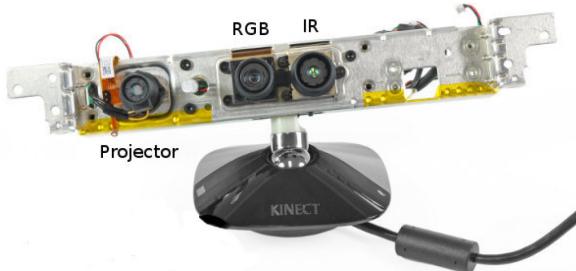
O Kinect versão 2 é composto de uma câmera RGB-D (*red, green, blue* e *depth*) e utiliza uma projeção de fôtons, sendo mais robusto que seu antecessor, para ambientes fechados e para fins de escaneamento de formas humanas (como esqueleto, músculos, e até mesmo para medir o batimento cardíaco, por exemplo). Devido à técnica empregada para reconhecimento 3D *Time of Flight*, ele é muito sensível às texturas presentes no objeto, ou seja, esculturas com diferentes superfícies, diferentes refletâncias, lambertianas ou especulares, por exemplo, podem acarretar em problemas ou buracos nas reconstruções. Além disso, existem outros obstáculos, como a dificuldade em que o fóton emitido rebate em várias superfícies antes de ser detectado pelo sensor. Portanto, para o uso em áreas externas, a primeira versão do Kinect se sai melhor (FREEDMAN et al., 2014).

O Kinect versão 1 é composto por duas câmeras: uma RGB apenas para cores e

aparência, e outra câmera infra-vermelho para profundidade, atuando em sincronia com um projetor de padrões em infra-vermelho (IR ou *infra-red*). O projetor IR lança uma matriz de padrões que já é previamente conhecida pelo sistema do Kinect, e, a partir disso, qualquer deformação deste padrão é captada pela câmera IR, o que identifica se um objeto está no alcance dos sensores ou não. O espectro infra-vermelho é utilizado por não ser percebido pela visão humana. A resposta é composta por 3 saídas: uma imagem de intensidades IR, uma RGB e a profundidade de cada pixel.

A seguir, será usado o termo “Kinect” para o Kinect versão 1, pois este é ainda o modelo mais utilizado na literatura para o escaneamento de objetos ([HENRY et al., 2012](#)), conforme é o foco deste trabalho, em vez do rastreamento em tempo real para entretenimento, que é dominado pelo Kinect versão 2.

Figura 9 - Imagem de um Kinect V1 aberto



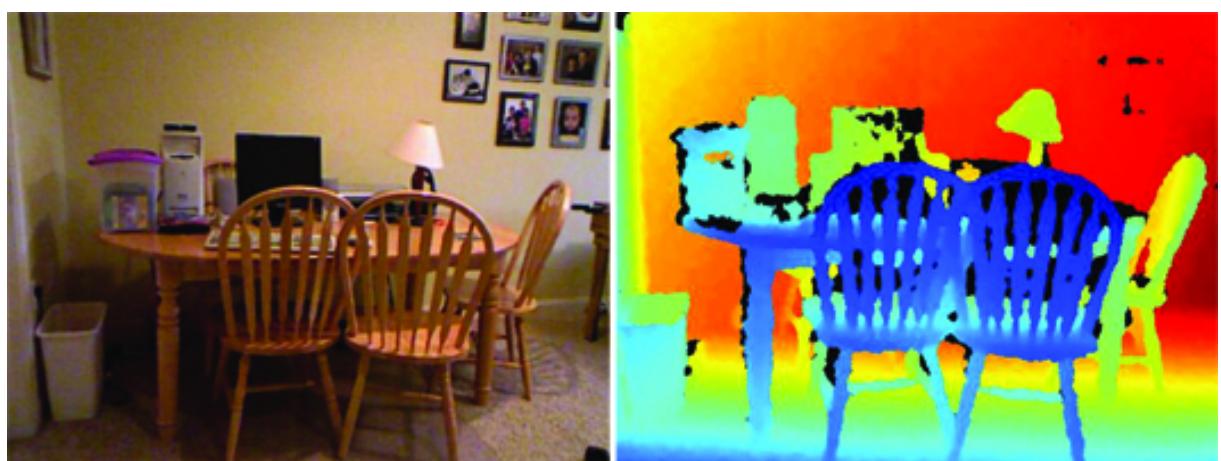
Legenda: Constituído de uma câmera infra-vermelho (IR - *Infra-Red*), uma câmera RGB e um projetor IR.

Fonte: [SMISEK; JANCOSEK; PAJDLA, 2013](#).

2.2.2 Processo de calibração

A principal saída do Kinect é correspondente à profundidade da cena. Na realidade, em vez de providenciar uma profundidade Z , ele retorna uma profundidade inversa, D . Essa informação é construída a partir da triangulação da imagem IR com o padrão conhecido do projetor.

Figura 10 - Saída de uma imagem interpretada pelo Kinect



Legenda: Cada cor disposta na imagem corresponde à profundidade ou distância da cena para o Kinect, e preto corresponde a pontos sem informação confiável.

Fonte: [SILBERMAN DEREK HOIEM; FERGUS, 2012](#).

Na literatura, foram realizados alguns experimentos associando o uso do Kinect V1

como um escaneador de baixo custo em reconstruções (**SMISEK; JANCSEK; PAJDLA, 2013**), tarefa para a qual não foi originalmente projetado. Primeiramente, foi executada uma calibração do Kinect para este tipo de reconstrução, onde, a partir de experimentos, o sistema foi modelado como

$$q(z) = 2.73z^2 + 0.74z - 0.58 \quad (1)$$

onde z é a profundidade em metros, e q a quantização, em milímetros. O modelo geométrico do Kinect foi modelado com um sistema multi-ocular (*multi-view*) considerando o RGB, IR e a profundidade, que projeta um ponto 3D x , no ponto de imagem $[u, v]^T$ 2, onde u e v são os pixels da imagem

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = k \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \quad (2)$$

$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = \underbrace{(1 + k_1r^2 + k_2r^4 + k_5r^6)}_{\text{distorção radial}} \begin{bmatrix} p \\ q \\ 0 \end{bmatrix} + \underbrace{\begin{bmatrix} 2k_3pq + k_4(r^2 + 2p^2) \\ 2k_4pq + k_3(r^2 + 2q^2) \\ 1 \end{bmatrix}}_{\text{distorção tangencial}} \quad (3)$$

$$r^2 = p^2 + q^2, \begin{bmatrix} pz \\ qz \\ z \end{bmatrix} = R(X - C), \quad (4)$$

onde k_n é o parâmetro de distorção, matriz de calibração da câmera K , rotação R e centro de projeção C . A profundidade é associada à geometria da câmera IR, que retorna a profundidade inversa ao longo do eixo z . Os valores de u e de v são dados pela equação 3,

$$X_{IR} = \frac{1}{c_1 d + c_0} dis^{-1} \left(K_{IR}^{-1} \begin{bmatrix} x + u_0 \\ y + v_0 \\ 1 \end{bmatrix}, k_{IR} \right) \quad (5)$$

$$u_{RGB} = K_{RGB} dis(R_{RGB}(X_{IR} - C_{RGB}), k_{RGB}). \quad (6)$$

Associamos o sistema de coordenadas do Kinect com a câmera IR e, consequentemente, $R_{IR} = I$ (identidade) e $C_{IR} = 0$. O ponto 3D X_{IR} é construído a partir da medição de $[x, y, d]^\top$ da equação 5 e produz uma imagem RGB, Equação 6.

Na equação 5, dis é a distorção, proveniente da equação 3, k_{IR} e k_{RGB} são, respectivamente, distorção relacionada à IR e à RGB. K_{IR} é a matriz de calibração da câmera IR, K_{RGB} é a matriz de calibração da câmera RGB. R_{RGB} e C_{RGB} são, a matriz de rotação e de centro da câmera RGB, respectivamente.

A calibração ocorreu usando o mesmo alvo nas câmeras IR e RGB, mesmos pontos 3D, e consequentemente, a mesma posição relativa das câmeras (SMISEK; JANCOSEK; PAJDLA, 2013). O sistema de coordenadas global do Kinect faz a posição relativa da câmera igual a R_{RGB} , C_{RGB} .

Também foi observado que existe um deslocamento entre imagem IR e a imagem da profundidade criada pelo Kinect (SMISEK; JANCOSEK; PAJDLA, 2013). Para contornar este problema, uma série de experimentos foram executados, gerando 2.

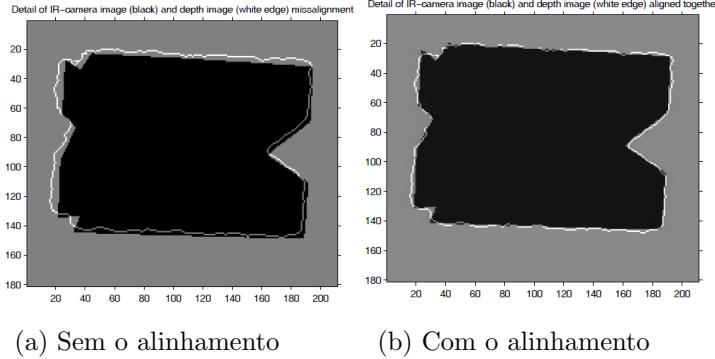
Tabela 2 - Valores de deslocamentos e sua média

Imagen	1	2	3	4	Média
u_0	2,8	2,9	3,0	3,4	3,0
v_0	3,0	2,7	2,8	3,1	2,9

Fonte: SMISEK; JANCOSEK; PAJDLA (2013).

Foi observado que após a calibração, o Kinect gerava erros residuais complexos, que para compensar esse erro residual, foi criada uma correção em z , onde é subtraído da

Figura 11 - Representação visual do acerto do deslocamento

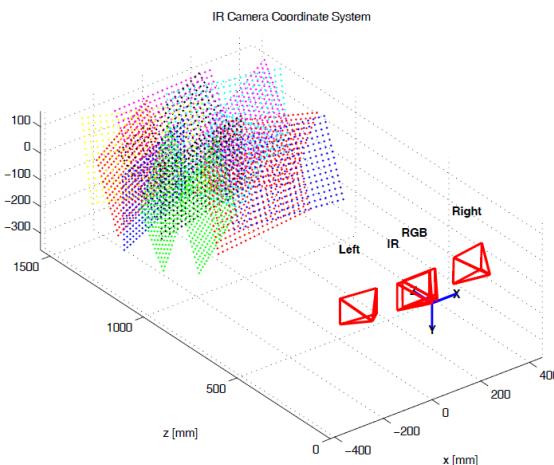


Legenda: A parte em preto é a imagem IR e o contorno em branco é a imagem de profundidade do alvo.

Fonte: [SMISEK; JANCOSEK; PAJDLA, 2013](#).

coordenada Z_{IR} da equação 5. Para validar essa correção, a correção-z das imagens foram construídas a partir dos resíduos das imagens ímpares e aplicadas nas pares, e o vice-versa. Depois da aplicação da correção-z, a media dos erros diminuiu aproximadamente 0,25mm. Como parâmetro de comparação, foram dispostas 2 câmeras, no mesmo ambiente do Kinect.

Figura 12 - Ambiente de calibração do modelo geométrico



Nota: Posição e orientação do Kinect (com as câmeras IR e RGB) e o par estéreo SLR (*Left*, *Right*) em conjunto com pontos de calibração 3D reconstruídos em alvos de calibração planar.

Fonte: [SMISEK; JANCOSEK; PAJDLA, 2013](#).

Tabela 3 - Resultados dos testes executados no ambiente descrito anteriormente

Método	Erro geométrico e [mm]		
	$\mu(e)$	$\sigma(e)$	$\max(e)$
SLR Stereo	1,57	1,15	7,38
Kinect	2,39	1,67	8,64
SR-4000	27,62	18,20	133,85

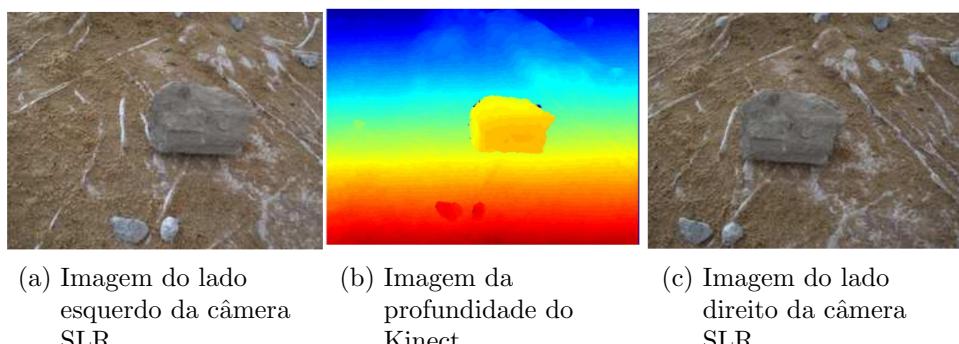
Fonte: [SMISEK; JANCOSEK; PAJDLA \(2013\)](#).

2.2.3 Uso do Kinect com *Structure from Motion*

Com o sistema calibrado, da literatura ([SMISEK; JANCOSEK; PAJDLA, 2013](#)), o Kinect foi testado usando técnicas *Structure from Motion*, onde a figura a seguir compara a superfície 3D de nuvem de pontos com uma com Kinect. O resultado é tão bom quanto ao mais acurado resultado de *Multi-View Stereo*.

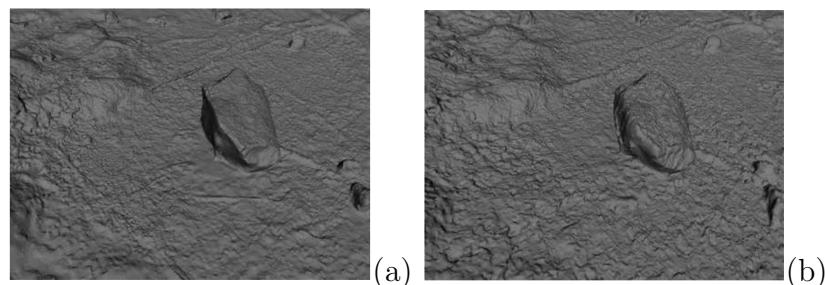
O Kinect, com o procedimento de calibração, pode ser combinado com técnicas *SfM* e *multi-view stereo*, o que abre uma nova metodologia na área de reconstrução 3D. Quanto à qualidade da reconstrução, o Kinect ficou melhor que o SR-4000 e perto do 3.5M SLR Stereo, figuras [13](#), [14](#) e tabela [3](#).

Figura 13 - Imagens iniciais para reconstrução 3D usando Kinect com câmeras



Fonte: [SMISEK; JANCOSEK; PAJDLA \(2013\)](#).

Figura 14 - Resultados da reconstrução SfM



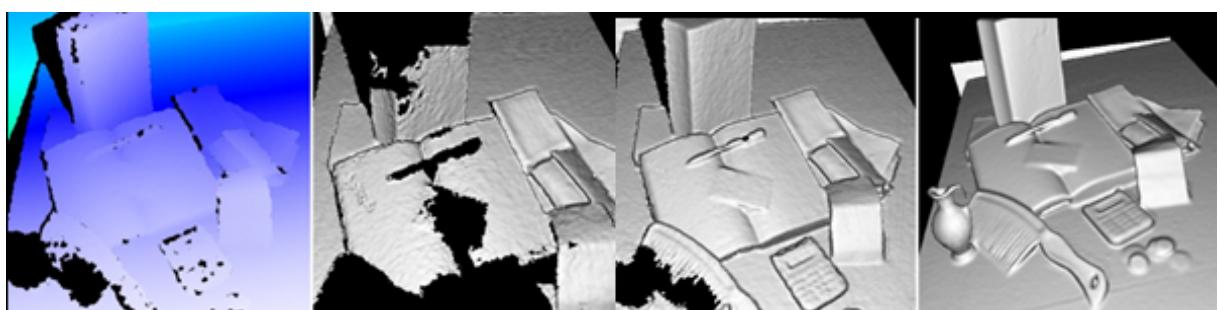
Legenda: (a) - reconstrução do SLR; (b)- resultado do Kinect.

Nota: Qualitativamente ambas são satisfatórias.

Fonte: [SMISEK; JANCOSEK; PAJDLA, 2013](#).

O Kinect é tão utilizado para fins pessoais e empresariais, que existem alguns softwares prontamente disponíveis para utilizá-lo como uma ferramenta de reconstrução 3D, sendo a maioria deles acessíveis (incluindo um pacote de ferramentas que a própria Microsoft disponibilizou para adaptar o Kinect em um computador pessoal convencional). O software chamado Kinect Fusion, da Microsoft, promove escaneamentos usando o Kinect, de um dado objeto e cria um modelo 3D. O usuário pode escanear uma cena “pintando” com o Kinect e, ao mesmo tempo, interagir com o modelo 3D detalhado, Figuras 15 e 16.

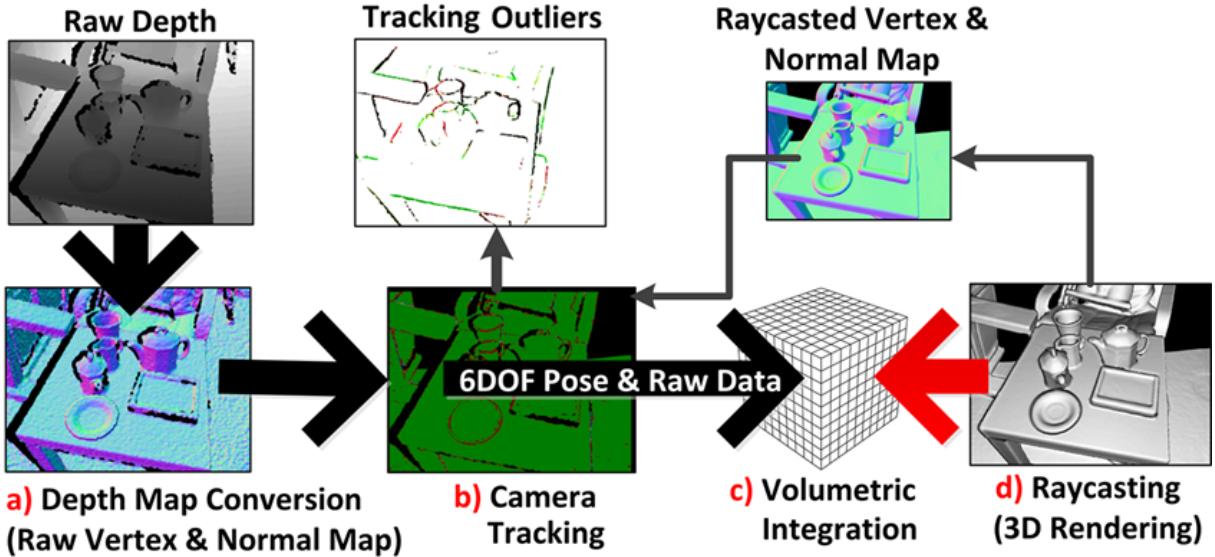
Figura 15 - Funcionamento do Kinect Fusion



Nota: Adquirindo o mapa de profundidade com a câmera do Kinect, onde há várias falhas devido à falta de dados. Em alguns segundos é realizada uma reconstrução 3D realista de uma cena estática apenas movendo o Kinect ao redor da cena. Após isso, uma nuvem de pontos ou uma malha 3D densa pode ser produzida por técnicas adaptadas de *multiview stereo* descritas nos próximos capítulos.

Fonte: [IZADI et al., 2011](#).

Figura 16 - Processo de escaneamento 3D no Kinect Fusion



Fonte: IZADI et al. (2011).

Outro programa conhecido é o *Skanect*, que tem a versão gratuita com a qual é possível fazer escaneamentos básicos, bem como a versão paga, que possibilita uma configuração mais detalhada, como por exemplo: a delimitação do objeto que será reconstruído; exportar o arquivo em diferentes formatos (*.PLY* e *.OBJ*, formatos para exportação para programas que trabalham o modelo gerado, como Blender ou Sculptris); escolher o número de faces a ser exportado; e suporte a demais formatos como *STL* (próprio para a impressora 3D, através de software como o *Cura*) e *VRML*, além de salvar também as cores do modelo.

Entretanto, uma desvantagem que diminui a aplicabilidade do Kinect é que ele foi projetado para funcionar bem em espaços fechados (SCHMID et al., 2013), com detecção de formas humanas e movimentações. Ou seja, numa aplicação *in situ* já não funcionaria muito bem, pois além de não conseguir escanear os detalhes em alta definição de uma escultura, ele necessita de uma fonte de energia externa, diferentemente de um *smartphone* ou câmera convencional, o que dificulta a acessibilidade. Ademais, como é gerada uma reconstrução em tempo real (não tem uma forma de salvar em *cache* ou internamente), ele precisa estar ligado a um computador para fazer o escaneamento. Algumas empresas vendem soluções mais convenientes de escaneamento 3D baseado no hardware

do Kinect, porém o custo desta alternativa pode ser considerável se comparado a câmeras convencionais.

3 RECONSTRUÇÃO 3D PASSIVA

3.1 Introdução

A maioria dos métodos de reconstrução 3D robusta de estrutura por movimento, ou *Structure from Motion* (SfM), tem como base a utilização de pontos de interesse (*features*), que são pontos ou áreas em comum entre as imagens a serem reconstruídas. Para encontrar estes pontos, diferentes algoritmos são empregados. A maioria dos métodos baseados em SfM estima os modelos de câmeras e realiza reconstruções esparsas de nuvens de pontos, e é seguida de abordagens denominadas *Multi-View Stereo* (MVS) para obtenção de informação 3D mais completa. O pipeline SfM+MVS funciona com os seguintes passos, ilustrados na Figura 18:

- Aquisição de imagens;
- Estimação dos parâmetros de câmera para cada imagem;
- Reconstrução da geometria 3D de uma cena em diversos níveis de detalhe.

Figura 17 - Procedimento da maioria dos sistemas SfM

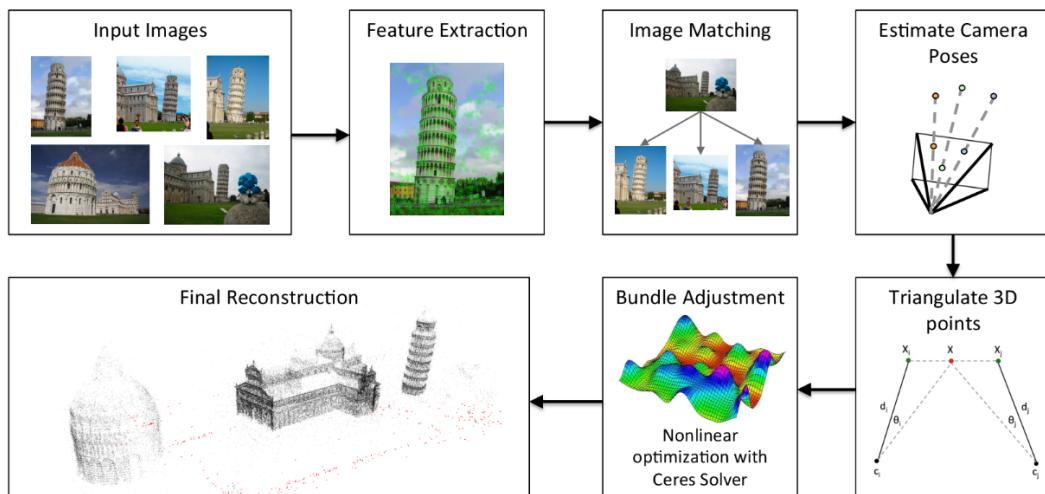


Figura 18 - Além dessas etapas, geralmente a reconstrução é densificada usando-se MVS.

Fonte: [SWEENEY \(2016\)](#).

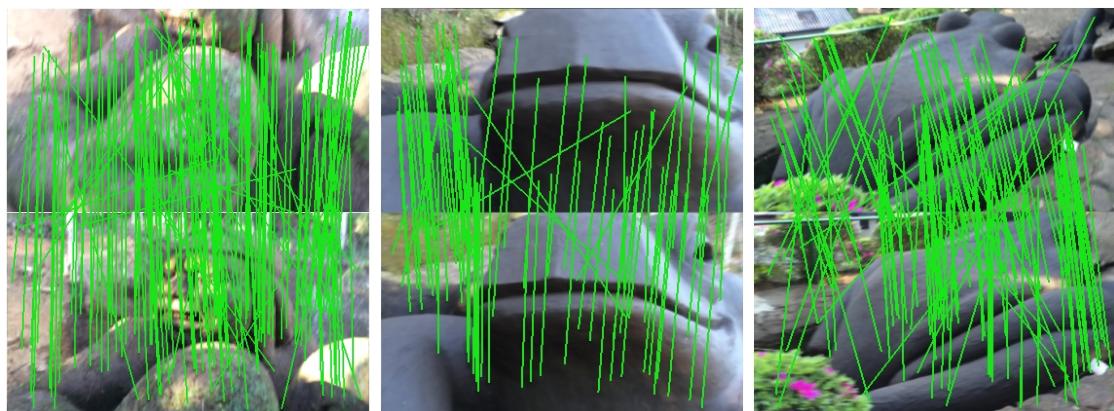
Neste trabalho, abordaremos o uso de dois softwares baseados em SfM+MVS: o MVE ([FUHRMANN; LANGGUTH; GOESELE, 2014](#)) e o VisualSfM ([WU et al., 2011b](#)), que julgamos serem os mais consolidados em pesquisa prática de ponta atualmente. Ambos são grandes pacotes que controlam inúmeros outros softwares e bibliotecas para os passos supracitados, que posteriormente serão comentados. O que difere um do outro é a partir da etapa de reconstrução esparsa e densa, onde o MVE emprega um algoritmo de agregação de mapas de profundidade, enquanto o VisualSfM fornece outros algoritmos de densificação, sendo mais fortemente calcado no resultado esparso do SfM.

O passo de aquisição de imagens será discutido em detalhes no Capítulo de experimentos [4](#). Seguimos adiante com os algoritmos para processamento. Note-se que será usado o termo curto “ponto” para denotar “ponto de interesse”.

3.2 SIFT – *Scale Invariant Feature Transform*

Primeiramente, calcula-se o SIFT (algoritmo de detecção de pontos de interesse, invariante à escala e à transformações, como rotação, translação e iluminação da imagem [19](#)) em cada imagem ([LOWE, 2004](#)).

Figura 19 - Imagens de correspondências do SIFT



Legenda: Cada linha pode ser interpretada como uma correspondência encontrada entre as duas imagens.

Fonte: O autor, 2017

O algoritmo pode ser dividido em cinco etapas:

- Deteccão de extremos no espaço-escala – *Scale-space Extrema Detection*;
- Localização de pontos – *Keypoint Localization*;
- Atribuição de orientação – *Orientation Assignment*;
- Descritor de pontos – *Keypoint Descriptor*;
- Casamento de pontos – *Keypoint Matching*;

3.2.1 Detecção de extremos de espaço-escala

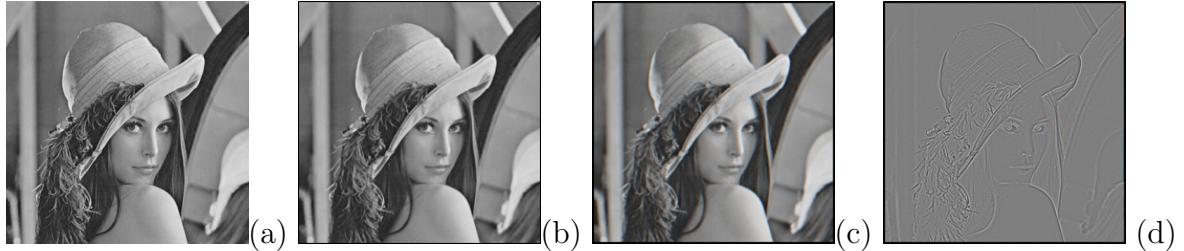
Em casos com cantos pequenos, uma detecção clássica de cantos funciona bem (**HARRIS; STEPHENS, 1988**). Porém, raramente utilizaremos a mesma janela para detectar pontos em imagens com diferentes escalas, pois utilizamos também imagens de maior resolução e, consequentemente, cantos com mais área de suporte. Para isso, precisamos de janelas grandes também.

Para resolver este problema, o filtro de escala-espacó é usado: o laplaciano com uma gaussiana (*Laplacian of Gaussian* – LoG). O LoG atua como um detector de partículas em diferentes tamanhos σ . (Onde σ é o parâmetro de escala). Por exemplo, o núcleo gaussiano com σ baixo, tem como resposta um alto valor para um canto pequeno. Enquanto um núcleo gaussiano com alto σ , se encaixa bem para um canto maior. Com esta lógica, podemos encontrar um máximo local através da escala e o espaço, o que nos fornece uma lista de $(x, y\sigma)$, o que significa que existe um ponto-chave em potencial, com o par (x, y) na escala σ . Por exemplo, um pixel é comparado com seus 8 vizinhos, assim como comparado com os 9 pixels na próxima escala e os 9 pixels na escala anterior. Se esse pixel é um local extremo, ele é um ponto-chave em potencial nessa escala. Sua posição é posteriormente interpolada e refinada antes de produzir os pontos finais.

Como o LoG é um pouco custoso computacionalmente, o SIFT utiliza um algoritmo aproximado do LoG, o DoG (Diferença de Gaussianos – *Difference of Gaussians*). O DoG é a diferença de um filtro de borrimento gaussiano de uma imagem com valores diferentes

de escala σ . Uma aplicação prática do filtro DoG é a sequência de imagens na Figura 20.

Figura 20 - Exemplo de filtro gaussiano



Legenda: (a) - filtro gaussiano aplicado na imagem; (b) - resultado do filtro; (c) - outra filtragem gaussiana; (d) - filtro DoG.

Nota: O filtro gaussiano é aplicado na imagem, com $\sigma = 1$. Após isso, outra filtragem gaussiana com $\sigma = 2$ é aplicada. Subtrai-se as duas imagens e é obtido o filtro DoG.

Fonte: O autor, 2017

Da literatura ([CULJAK et al., 2012](#)), em linguagem matemática, o DoG pode ser expresso a partir da seguinte sequência de expressões:

$$g_\sigma(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}\frac{x^T x}{\sigma^2}} \quad (7)$$

$$I_\sigma = g_\sigma * I, \quad \sigma \geq 0 \quad (8)$$

$$DoG_\sigma(o, s) = I_\sigma(o, s + 1) - I_\sigma(o, s) \approx \nabla^2 g_\sigma(x), \quad (9)$$

Onde a equação 7 é o núcleo gaussiano e ∇^2 é o operador Laplaciano.

3.2.2 Atribuição de orientação

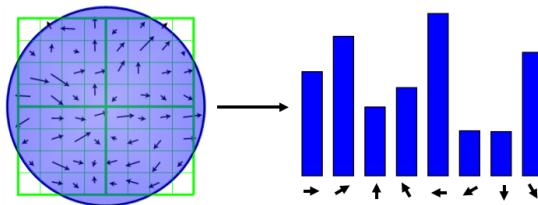
Uma orientação é atribuída a cada ponto-chave para obter a invariância à rotação da imagem. Ao redor da localização do ponto, uma vizinhança é obtida, dependente da escala, com tamanho e orientação definido a partir da magnitude do gradiente, Equação 10 e da direção do gradiente, Equação 11,

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}, \quad (10)$$

$$\theta = \tan^{-1} \left(\frac{(L(x, y + 1) - L(x, y - 1))}{(L(x + 1, y) - L(x - 1, y))} \right), \quad (11)$$

onde $L(x, y) = I_\sigma(x, y)$ para o valor de σ mais próximo possível da escala do ponto de interesse em questão. Em seguida, é montado um histograma de 36 orientações (*bins*) cobrindo 360, que para cada orientação agrupa o número de pixels na vizinhança que têm orientação do gradiente compatível. O histograma é ponderado pela magnitude do gradiente e por uma Gaussiana circular onde σ vale 1.5 em relação à escala do ponto-chave 21.

Figura 21 - Exemplo do resultado obtido do histograma de orientações do gradiente.



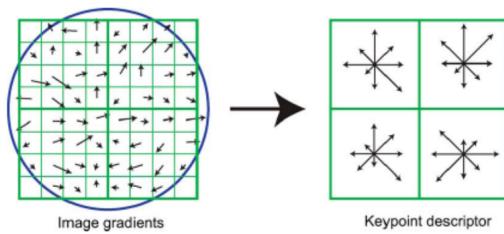
Fonte: JINDAL; VATTA (2010).

O ponto mais alto do histograma é obtido e qualquer pico acima de 80% desse valor é considerado no cálculo da orientação final atribuída ao ponto.

3.2.3 Descriptor de pontos

Com os pontos criados e atribuídos com o histograma supracitado de 36 entradas, cria-se agora o descriptor de pontos-chaves. Uma vizinhança 16x16px ao redor do ponto-chave é escolhida e esta mesma vizinhança é dividida em 16 sub-blocos 4x4px. Para cada bloco, um novo histograma orientado com 8 *bin* é criado. Logo, temos 128 valores válidos. Esses valores são representados em forma de vetor para expressar o descriptor de pontos-chaves 22.

Figura 22 - Exemplo de um descriptor de pontos SIFT



Legenda: Usando uma matriz 2x2 e uma região 8x8

Fonte: [JINDAL; VATTA, 2010.](#)

3.2.4 Casamento de pontos

Em sistemas SfM, pontos-chaves são casados entre duas imagens; as imagens são todas comparadas, duas a duas, exceto em sistemas aproximados de muito larga escala. Tais pontos são casados a partir da identificação da vizinhança mais próxima no espaço de atributos 128-Dimensional. Mas, em alguns casos, a segunda combinação mais próxima pode ser parecida com a primeira. Isso se dá por padrões repetidos e ruídos presentes nas imagens, caso em que os sistemas SfM preferem descartar numa primeira instância, e focar apenas em correspondências confiáveis (sem confusão em potencial). Nesse caso, a razão da distância mais próxima para a segunda distância mais próxima é utilizada. Se essa razão for maior que 0.8, essa combinação é descartada. Apesar de ser uma regra simples, é extremamente eficaz para obter as correspondências confiáveis necessárias para a estimativa das câmeras em sistemas SfM.

3.3 Triangulação – *Full pair-wise image matching*

Com os pontos de interesse (*features*) extraídos, e alguma forma de obter os modelos das câmeras, podemos, a princípio, fazer a triangulação entre os pontos das imagens. A triangulação nada mais é que uma estimativa de um ponto em 3 dimensões, dado pelo menos duas câmeras conhecidas, onde, cada câmera com a projeção do *feature* correspondente àquele ponto 3D [24](#).

Figura 23 - Exemplo de triangulação

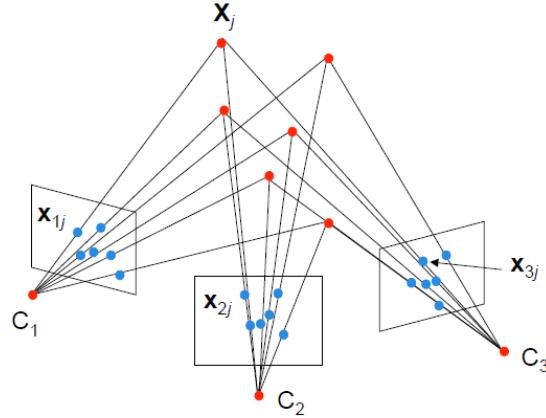


Figura 24 - Uma triangulação utilizando um ponto qualquer, X_j . Onde cada câmera C_1, C_2, C_3 possui um *feature* correspondente a cada uma delas, respectivamente, X_{1j}, X_{2j}, X_{3j} .

Fonte: Fergus (2013)

Infelizmente, não é tão simples assim. Existem muitos fatores que contribuem para aumentar a dificuldade da triangulação: ruídos, erro na estimativa das câmeras, posição inadequada das câmeras, o feixe das projeções podem não se encontrar no mesmo ponto 3D, ou não se tem informação alguma das câmeras, dentre outros. Entretanto, existem diversos algoritmos para resolução de cada um dos problemas enfrentados.

Com a extração dos *features* das imagens selecionadas, as próximas etapas da reconstrução para os sistemas MVE e VisualSfM se diferem, e a partir de agora, faremos abordagens individuais para cada um deles.

3.4 MVE – *Multi-View Reconstruction Environment*

3.4.1 Introdução

Um dos sistemas de pesquisa de ponta mais utilizados para a técnica de reconstrução densa é o MVE – *Multi-View Reconstruction Environment* ([FUHRMANN; LANG-GUTH; GOESELE, 2014](#)). Este algoritmo utiliza fotos e produz uma malha triangular superficial como resultado. Diferentemente das reconstruções baseadas nas geometrias

das imagens, o MVE é focado na reconstrução multi-escala, um quesito importante na reconstrução de esculturas e acervo cultural. Portanto, com esta técnica é possível reconstruir grandes volumes de dados, contendo regiões detalhadas em alta resolução, em comparação com o resto da cena. O sistema ainda possui uma interface gráfica para uma reconstrução baseada no SfM, amigável ao usuário e conhecida como UMVE, que permite a visualização e inspeção das imagens, mapas de profundidade e renderizar cenas e malhas 3D. Sua base de operação é basicamente [25](#):

1. Estrutura por Movimento – *Structure-from-Motion* (SfM):

- Reconstrói os parâmetros externos da câmera (posição e orientação) e seus parâmetros internos de calibração (distância focal e distorção radial), encontrando correspondências esparsas, mas estáveis entre as imagens.

2. Estereoscopia multiocular – *Multi-View Stereo* (MVS):

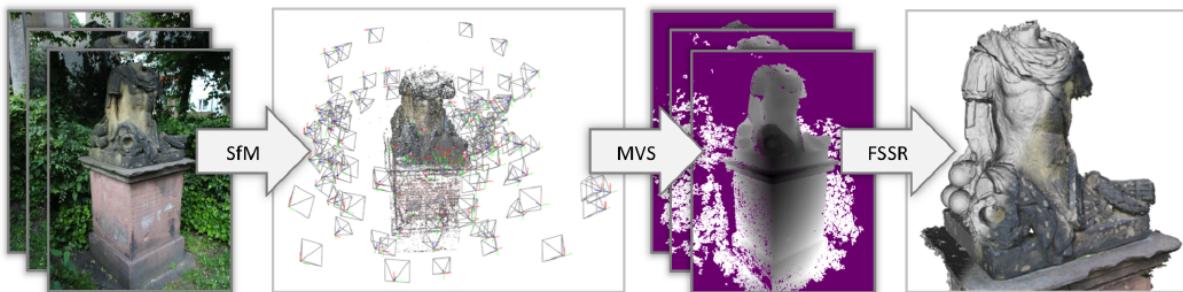
- Utiliza a posição estimada das câmeras, encontrando as correspondências visuais nas imagens. Estas correspondências são trianguladas, produzindo a informação 3D, e, consequentemente a reconstrução 3D densa.

3. Reconstrução de superfícies – *Surface Reconstruction*:

- Tem como entrada uma densa nuvem de pontos, ou mapas de profundidade individuais. Produz uma malha superficial globalmente consistente.

Como não existem muitas opções para algoritmos de SfM, o MVE permite a utilização de softwares externos como o *Bundler* ([SNAVELY et al., 2010](#)) ou o próprio *VisualSfM* para esta etapa. Tendo o SfM feito, partimos para o MVS. Com os parâmetros de câmera conhecidos, a reconstrução densa geométrica é feita. Existem diversos algoritmos para a reconstrução densa; o MVE utiliza um algoritmo próprio, bastante popular, feito por um de seus criadores, Michael Goesele ([GOESELE et al., 2007](#)), que reconstrói um mapa de profundidade para cada foto.

Figura 25 - Funcionamento do MVE



Nota: Começando com múltiplas imagens, técnicas SfM são empregadas para reconstruir os parâmetros das câmeras e os conjuntos de pontos esparsos. Mapas de profundidade são computados para cada imagem usando o MVS. Finalmente, uma malha colorida é extraída da união de todos os mapas de profundidade usando um algoritmo de aproximações de reconstruções de superfícies, FSSR – *Floating Scale Surface Reconstruction*.

Fonte: [FUHRMANN; LANGGUTH; GOESELE, 2014](#).

Embora abordagens baseadas em mapeamentos de profundidade produzam uma grande quantidade de redundâncias, (isso se dá por causa das inúmeras fotos que são sobrepostas e possuem partes similares da mesma cena), este algoritmo é altamente escalável para grandes cenas, pois apenas um pequeno conjunto de fotos vizinhas é necessário para a reconstrução. Outra vantagem da utilização dos mapas de profundidade como representação intermediária é que a geometria é parametrizada em seu domínio natural, e os dados por foto (como a cor, por exemplo) estão diretamente acessíveis nas imagens.

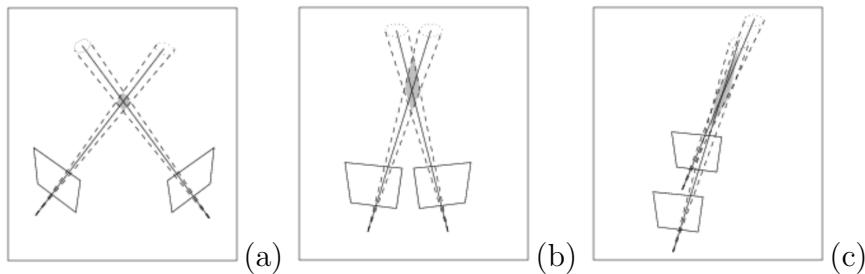
3.4.2 Guia de reconstrução com o MVE

Existem algumas recomendações para se ter uma boa reconstrução com o MVE ([FUHRMANN; LANGGUTH; GOESELE, 2014](#)). Um bom conjunto de dados é gerado se algumas regras simples forem seguidas:

- Para que o algoritmo MVS consiga fazer uma triangulação com qualquer posição 3D, o conjunto de dados terá que ter, no mínimo, cinco fotos.

- As fotos devem ser tiradas com uma boa quantidade de sobreposição. A menos que o conjunto de dados se torne muito grande, uma grande quantidade de fotos não prejudicará a qualidade. Mas terá uma compensação do sistema, no que diz respeito à qualidade e desempenho.
- Para a triangulação funcionar, é necessário que tenha o efeito de paralaxe [26](#).
- A câmera deverá ser reposicionada, de preferência.

Figura 26 - Funcionamento da paralaxe



Legenda: (a) - espaçoamento grande entre câmeras; (b) - disposição correta das câmeras; (c) - espaçoamento pequeno entre câmeras.

Nota: Com espaçoamento grande entre câmeras, a informação extraída das imagens em comum é menor. Se a angulação do efeito de paralaxe for baixa, terá a mesma informação sobre um ponto 3D. Nesses casos, a reconstrução pode ser incerta. Para que o efeito paralaxe tenha maior proveito das imagens, alguns autores argumentam que câmeras devem estar dispostas como, de forma a extrair uma boa quantidade e qualidade de informações do ponto.

Fonte: [STRICKER, 2015](#).

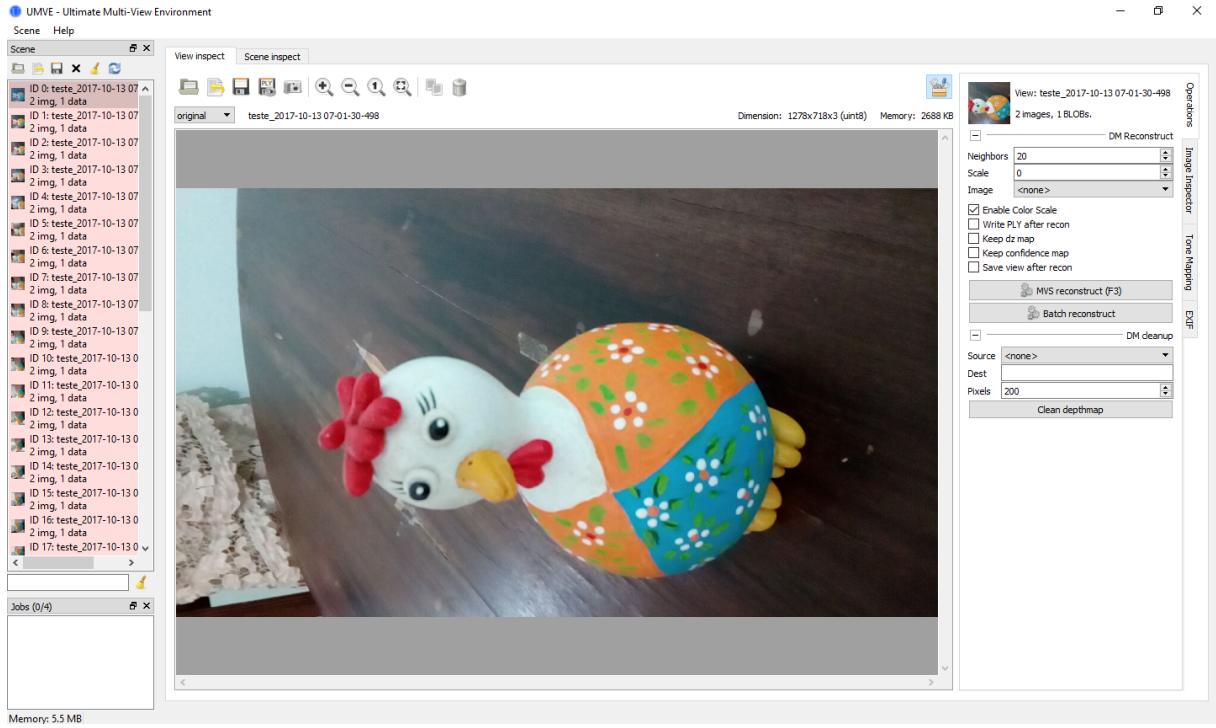
3.4.2.1 Criando uma cena

Uma vista (*view*) no MVE contém dados por *viewport* (como imagens, mapas de profundidade ou outros dados). Uma cena é uma coleção de vistas que formam um conjunto de dados (*dataset*). Uma nova cena pode ser criada utilizando a interface gráfica UMVE, Figura 27, ou por linha de comando (*makescene*). Tecnicamente, a cena é criada como um diretório no sistema de arquivos (com o nome do conjunto de dados). Este, por sua vez, contém outro diretório (*views*), com todas as vistas guardadas com uma extensão de arquivos em .MVE.

Criar uma nova cena também criará apenas o diretório (*views*) vazio. A importação de fotos criará arquivos .MVE para cada foto. Esse processo importará metadados provenientes das imagens (*tags EXIF*), que é necessário para estimar a distância focal para cada foto a partir do modelo de câmera gravado em determinados tipos de arquivos de imagem. Caso estes meta-dados não estejam disponíveis, uma distância focal padrão é assumida pelo sistema, porém se essa distância adotada for uma péssima suposição, com

relação ao conjunto de dados utilizado, podem acontecer erros no SfM.

Figura 27 - Interface gráfica (UMVE)

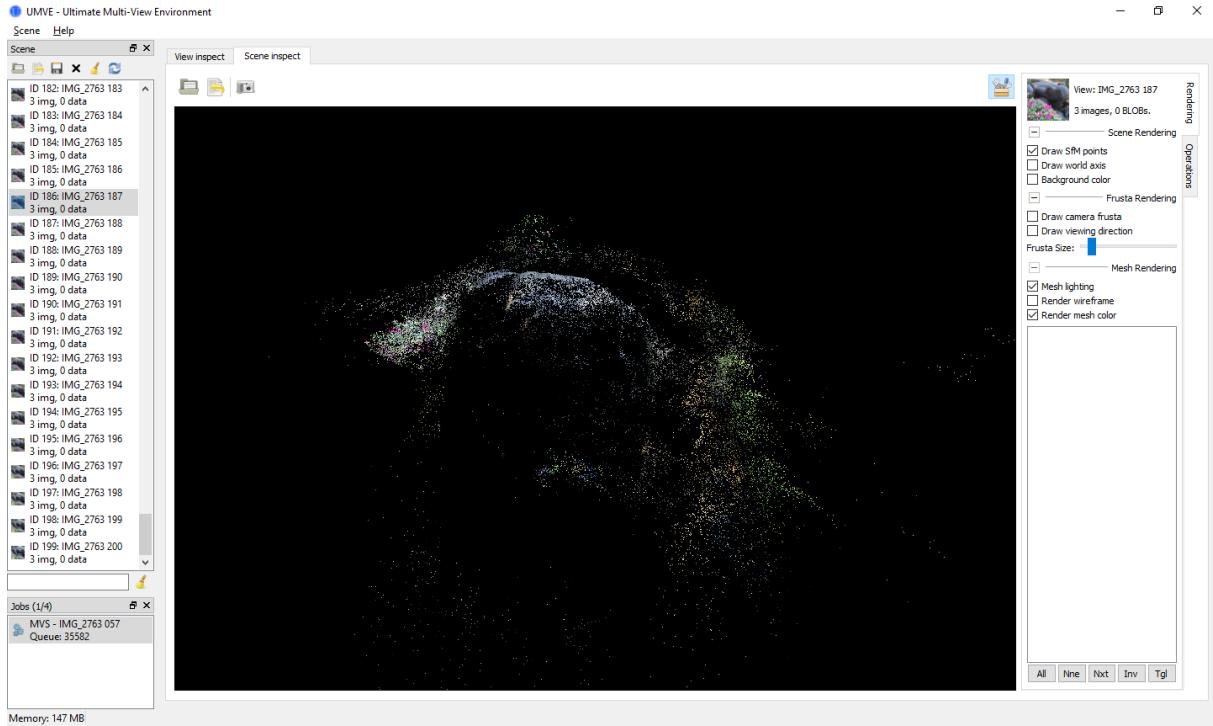


Fonte: O autor (2017).

3.4.3 Reconstrução SfM

Pode ser configurada e iniciada a SfM usando a interface gráfica UMVE ou por linha de comando (`sfmrecon`). A interface guia através da detecção de *features*, casamento de todos os pares de imagens (*pairwise matching*) e uso incremental do SfM. A reconstrução SfM começa a partir de um par inicial, e adiciona, de forma incremental, mais vistas à reconstrução [28](#).

Figura 28 - Reconstrução de nuvem de pontos gerada a partir de SfM do MVE



Fonte: O autor (2017).

3.4.3.1 Estereoscopia Multiocular – *Multi-View Stereo* (MVS)

Usando as imagens junto com os parâmetros obtidos das câmeras, é possível reconstruir a geometria densa utilizando o MVS. Isso pode ser feito utilizando a interface gráfica (UMVE) ou por linha de comando (*dmrecon*). O parâmetro mais importante é o nível de resolução em que os mapas de profundidade são reconstruídos: Caso seja nível 0 (ou L0), a reconstrução é feita usando o tamanho original das imagens. Se for nível 1 (ou L1), a reconstrução corresponde à metade do tamanho (um quarto dos números de pixels), e assim por diante.

Com a alta resolução das câmeras atuais, uma reconstrução L0 é raramente usada, pois geram mapas de profundidade mais dispersos com um custo computacional elevado, o que acarreta em dificuldades para encontrar as correspondências densas das imagens. Geralmente utiliza-se o L2, pois o processo é mais rápido, gerando mapas de profundidades

completos, já que utiliza resoluções menores, Figura 29.

Figura 29 - Representação do mapa de profundidade do MVS



Legenda: Uma imagem de entrada (esquerda) e correspondência em mapas de profundidade (direita), em roxo as partes sem nenhum mapa naquela região.

Fonte: [FUHRMANN; LANGGUTH; GOESELE \(2014\)](#).

O algoritmo do MVS pode ser dividido em duas partes

- Uma estrutura regional-crescente que tem uma fila de candidatos correspondentes, Q , ordenada pelas localizações dos pixels na câmera acrescido de seus valores para profundidade e normais;
- Um sistema de correspondências que leva um candidato correspondente como entrada e calcula profundidade, normal e uma confiança de correspondência usando vistas vizinhas fornecidas pela seleção de exibição local. Se a correspondência for bem sucedida, os dados são armazenados em mapas de profundidade, normais e de confiança e os pixels vizinhos na câmera são adicionados como novos candidatos a Q .

3.4.3.2 Reconstrução de Superfícies – *Surface Reconstruction*

Utiliza-se o programa em linha de comando `scene2pet`, que combina todos os mapas de profundidade em uma única e grande nuvem de pontos. Nesta fase, um valor de

escala é atribuído a cada ponto, que indica o tamanho atual da região da superfície na qual o ponto foi mensurado. Esta informação adicional permite o uso de várias propriedades benéficas usando a abordagem de reconstrução de superfície por FSSR (**FUHRMANN; GOESELE, 2014**). A seguir, as ferramentas FSSR calculam uma representação volumétrica de escala múltipla a partir dos pontos (na qual não precisa de nenhum ajuste de parâmetros explícitos) e uma malha final é extraída. Esta malha pode parecer desordenada devido a regiões não confiáveis e a componentes isolados, oriundos de medidas imprecisas. Logo, a malha é limpa, retirando pequenos componentes isolados e regiões não confiáveis da superfície.

3.5 VisualSfM

3.5.1 Introdução

O VisualSfM (**WU et al., 2011b**) é um sistema de software baseado em fotogrametria que faz todo o processo de reconstrução 3D de um objeto e que pode ser usado por linha de comando ou então pela interface gráfica. No presente projeto, constatou-se que a interface gráfica é de fato excelente na prática, sendo altamente customizável, podemos utilizar o CUDA da NVIDIA, ou OpenGL, especificar a lista de pares para correspondência de imagens, usar diversos detectores de *features* próprios, além de possuir detecção de *features* eficiente, algoritmos de reconstrução densa, dentre outros. Trata-se dum software robusto, que pode ser usado em Linux, Windows ou Mac. Ademais, o VisualSfM é capaz de mostrar a matriz de correspondência de *features*, número de *features*, rodar um *Bundle Adjustment* independente, usar PVMS, alterar a memória necessária de GPU usada na reconstrução, deletar uma reconstrução indesejável ou até mesmo alterar parâmetros.

3.5.2 Procedimento

O *pipeline* de reconstrução do VisualSfM é parecido com o MVE 3.4, porém é mais intuitivo. Em sua interface, possui um *log* de mensagens e erros que por ventura venham a acontecer e, na parte de cima, alguns botões, Figura 30. O funcionamento segue os seguintes passos:

Figura 30 - Interface gráfica do VisualSfM



Legenda: Botões na parte superior da interface gráfica, este seria o procedimento padrão de funcionamento do software.

Fonte: WU et al., 2011b.

1 - Adicionar imagens. Este é o primeiro passo para começar uma reconstrução: adiciona-se imagens ao software. Pode ser uma única foto, um conjunto de fotos, incrementar o conjunto já existente ou então abrir um arquivo de extensão .nvm, que é interpretado como uma reconstrução esparsa previamente feita.

2 - Correspondência de imagens. O programa roda o algoritmo SIFT, realizando todas as correspondências entre os pontos de interesse, conforme discutido anteriormente.

3 - Reconstrução esparsa. O VisualSfM roda o algoritmo de reconstrução esparsa (PBA/MCBA) (WU et al., 2011a) em todos os pontos de interesse descobertos no passo anterior.

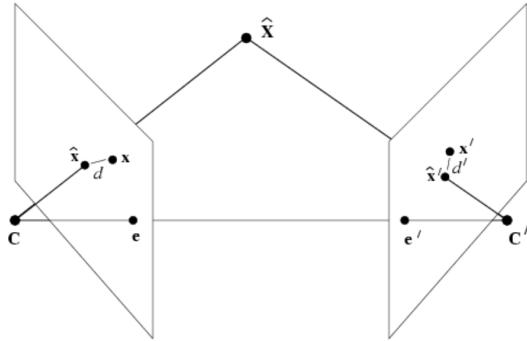
3.5.3 Bundle Adjustment

O termo *Bundle* refere-se aos feixes de luz que refletem em cada *feature* 3D e vão para o centro de projeção de cada câmera da cena a ser reconstruída. Com isso, o *Bundle Adjustment*, ou BA, é o problema de refinar uma reconstrução visual para produzir

estimativas de calibração interna e externa de câmeras, juntamente com as estruturas 3D.

Solvers de problemas de BA (**CHOUDHARY**,) resumem-se em minimizar o erro de reprojeção, Figura 31, entre as posições de imagem dos pontos de observados e previstos, o que é expresso como a soma de quadrados de um grande número de funções não-lineares de valor real. Assim, a minimização é obtida usando algoritmos de mínimos quadrados não-lineares. Destes, Levenberg-Marquardt (**MORÉ, 1978; PRESS, 1992**), que emprega um método híbrido de iterações de Newton e *Gradient Descent*, provou ser um dos mais bem sucedidos devido à sua facilidade de implementação e ao uso de uma estratégia de amortecimento eficaz que lhe confere a capacidade de convergir rapidamente de uma ampla gama de suposições iniciais, na qual, o desafio de se ter uma boa performance é, basicamente, escolher uma boa suposição inicial. Ao linearizar iterativamente a função a ser minimizada na vizinhança da estimativa atual, o algoritmo de Levenberg-Marquardt envolve a solução de sistemas lineares denominados equações normais. Ao resolver os problemas de minimização que surgem na estrutura do BA, as equações normais têm uma estrutura de bloco esparsa devido à falta de interação entre os parâmetros para diferentes pontos 3D e câmeras. Isso pode ser explorado para obter enormes benefícios computacionais ao empregar uma variante esparsa do algoritmo de Levenberg-Marquardt que aproveita explicitamente o padrão de zeros de equações normais, evitando armazenar e operar em elementos zero. A seguir detalharemos essas etapas e as implementações utilizadas (**FABBRI, 2011**).

Figura 31 - Problema de *Bundle Adjustment*



Nota: Dado um ponto de um objeto 3D \hat{X} , sua reprojeção nas câmeras C e C' , são, respectivamente, x e x' . Esses pontos possuem um erro de reprojeção d e d' e, ao utilizar o PBA/MCBA, os pontos x e x' serão reajustados como \hat{x} e \hat{x}' , respectivamente.

Fonte: [STRICKER, 2015](#).

Consideremos o seguinte problema de estimativa de um conjunto de parâmetros $x \in \mathbb{R}^M$ a partir de um conjunto de observações $\bar{Z} \in \mathbb{R}^N$, com $M \ll N$, relacionados por um modelo $Z(x)$. O objetivo é calcular x tal qual $Z(x)$ é o mais próximo de \bar{Z} possível, i.e., para minimizar o erro de previsão residual $\|\Delta Z\| = \|\bar{Z} - Z(x)\|$.

3.5.3.1 PBA/MCBA – *Multi-Core Bundle Adjustment*

O PBA/MCBA ([FURUKAWA; PONCE, 2009; WU et al., 2011a](#)) é um algoritmo que utiliza, de forma eficiente, os núcleos do computador, sendo até 10 vezes mais rápido em CPU e 30 vezes em GPU, em comparação ao estado da arte anterior. É o primeiro sistema publicado baseado em GPU que escala para maiores problemas de Bundle Adjustments. Isto abriu a porta para resolver problemas ainda maiores. Ele funciona observando o passo inexato de resolução não-linear usando Levenberg Mardquardt, que pode ser implementado sem armazenar nenhuma matriz (hessiana, complemento de Schur ou jacobiana) na memória. Para sistemas de núcleo único, isso se traduzia em trocar memória por tempo, mas em GPUs, isso leva a um surpreendente ganho, no espaço e no tempo.

Outra surpresa é que a aritmética de precisão única, quando combinada com técnicas de normalização adequadas, dá resultados comparáveis aos obtidos por um solucionador de problemas usando aritmética de dupla precisão. Isso resulta em mais espaço livre e economia de tempo.

Assumindo que x é um vetor de parâmetros e $f(x) = [f_1(x), \dots, f_k(x)]$ é o vetor de erros de projeção de uma reconstrução 3D. O problema de otimização pode ser formulado como um problema de mínimos quadrados não-linear:

$$x^* = \operatorname{argmin}_x \sum_{i=1}^k \|f_i(x)\|^2. \quad (12)$$

O Levenberg-Marquadt supracitado funciona resolvendo uma série de aproximações lineares regularizadas ao problema não-linear original. Seja $J(x)$ a Jacobiana de $f(x)$, então em cada iteração LM resolve um problema linear de mínimos quadrados da forma:

$$\delta^* = \operatorname{argmin}_{\delta} \|J(x)\delta + f(x)\|^2 + \lambda \|D(x)\delta\|^2, \quad (13)$$

e atualiza x da forma:

1: caso($\|f(x + \delta^*)\| < \|f(x)\|$), então

2: $x = x + \delta^*$.

Aqui, $D(x)$ é uma matriz diagonal não-negativa, tipicamente a raíz quadrada da diagonal da matriz $J(x)^T J(x)$, e λ são parâmetros não-negativos que controlam a limite da regularização, que, por sua vez, é necessária para garantir um algoritmo convergente. O LM atualiza o valor de λ a cada passo, baseado no quanto a jacobiana ($J(x)$) está próxima da função original ($f(x)$). Resolver 13 é equivalente à resolver as equações normais

$$(J^T J + \lambda D^T D)\delta = -J^T f \quad (14)$$

onde retiramos a dependência de x . A matriz $H_\lambda = J^T J + \lambda D^T D$ é conhecida como a matriz Hessiana aumentada.

No *Bundle Adjustment*, o parâmetro ”vetor” é tipicamente organizado como $x = [x_c; x_p]$, onde x_c é o vetor de parâmetros da câmera e x_p é o vetor de parâmetros do ponto. Similarmente para D , δ e J , usaremos subscritos c e p para denotar parâmetros da câmera e do ponto, respectivamente. Seja $U = J_c^T J_c$, $V = J_p^T J_p$, $U_{\lambda} = U + \lambda D_c^T D_c$, $V_{\lambda} = V + \lambda D_p^T D_p$ e $W = J_c^T J_p$, então 14 pode ser re-escrita como um bloco de sistema linear estruturado

$$\begin{bmatrix} U_{\lambda} & W \\ W^T & V_{\lambda} \end{bmatrix} \begin{bmatrix} \delta_c \\ \delta_p \end{bmatrix} = - \begin{bmatrix} J_c^T f \\ J_p^T f \end{bmatrix}$$

Vale ressaltar que, para a maioria dos problemas de *Bundle Adjustment*, U_{λ} e V_{λ} são matrizes diagonais de blocos. Esta observação está no fundamento do truque do complemento de Schur usado para resolver esse sistema linear de forma eficiente, onde, ao aplicar a eliminação Gaussiana nos parâmetros do ponto, obtemos um sistema linear consistindo apenas nos parâmetros da câmera:

$$(U_{\lambda} - WV_{\lambda}^{-1}W^T)\delta_c = -J_c^T f + WV_{\lambda}^{-1}J_p^T f, \quad (15)$$

onde $S = U_{\lambda} - WV_{\lambda}^{-1}W^T$ é o complemento de Schur ou a matriz reduzida da câmera. Com a solução para a Equação 15, conseguimos obter os parâmetros do ponto com substituição regressiva:

$$\delta_p = -V_{\lambda}^{-1}(J_p^T f + W^T \delta_c). \quad (16)$$

Caso S for simétrica positiva-definida, a fatoração de Cholesky é indicada para a resolução da Equação 15. O algoritmo MCBA, une o paralelismo da GPU e técnicas de uso de multi-núcleos da GPU e a combinação de algoritmos de gradientes de conjugados pré-condicionados e algoritmos inexatos LM para resolução dessas equações com alguns pré-condicionadores simples e computacionalmente baratos.

4 - Reconstrução densa. A reconstrução é finalizada executando-se o algoritmo de reconstrução densa (**GAVA**,) CMVS/PMVS-2 embutido no próprio VisualSfM.

3.5.3.2 CMVS/PMVS-2 (*Clustering Views for Multi-view Stereo / Patch-based for Multi-view Stereo version 2*)

Muitos algoritmos Multi-View Stereo (MVS) não escalam tão bem com um grande número de imagens de entrada ou em uma alta resolução, pois necessitam de muita memória e recursos computacionais. A palavra-chave do CMVS (TEERAVECH, 2013; WU, 2013; LI et al., 2013a) é escalabilidade, pois seu propósito é utilizar imagens provenientes de sites na internet, em diferentes resoluções, como o [Flickr.com](#). Ele utiliza a saída do structure from Motion – SfM (mais especificamente, a saída do passo anterior, do PBA) como entrada. Após isso, decompõe as imagens de entrada como um conjunto de *clusters* de imagens com tamanhos gerenciáveis. O MVS pode ser usado para processar cada *cluster* de forma independente e em paralelo, onde a fusão das reconstruções de todos os *clusters* não deve perder detalhes que poderiam ser obtidos através do conjunto de imagens como um todo. A formulação dos *clusters* é projetada para satisfazer três restrições:

1. As imagens redundantes são excluídas dos *clusters* (compacidade)
2. Cada *cluster* é pequeno o suficiente para uma reconstrução MVS (restrição de tamanho)
3. As reconstruções MVS destes *clusters* resultam em uma perda mínima de conteúdo e detalhes em comparação com o que pode ser obtido através do processamento do conjunto completo de imagens (cobertura).

A compacidade é importante para a eficiência computacional, mas também para melhorar precisão, pois as coleções de fotos da Internet ou de vídeo geralmente contêm centenas ou milhares de fotos adquiridas de quase mesmo ponto de vista, ou seja, um conjunto composto inteiramente informações redundantes. A sobreposição de *clusters* é definida por:

- Minimizar $\sum_k |C_k|$ (compacidade);

- $|C_k| \leq \alpha$, $\forall k$, onde α é determinado por recursos computacionais, principalmente por limitações de memória. (tamanho);
- $\frac{\# \text{ pontos cobertos em } I_i}{\# \text{ pontos em } I_i} \geq \delta$, $\forall i$, onde δ é uma constante de proporção de pontos cobertos. (cobertura).

O CMVS pode ser resumido em quatro passos, detalhados adiante:

- Filtro SFM – agrupamento de pontos SFM;
- Seleção de imagens – remove imagens redundantes;
- Divisão de *cluster* – reforça a restrição de tamanho;
- Adição de imagens – reforça a cobertura.

Filtro SFM. Os recursos da imagem não detectados ou incomparáveis levam a erros nas estimativas de visibilidade do ponto V_j (geralmente na forma de imagens que estão faltando). Estimativas de visibilidade mais confiáveis são obtidas ao agregar dados da visibilidade em uma vizinhança local. A posição do ponto mesclado é a média de seus vizinhos, enquanto a visibilidade se torna a união. Este passo também reduz significativamente o número de pontos SFM e melhora o tempo de execução das três etapas restantes. Especificamente, a partir de um conjunto de pontos SFM, um ponto é selecionado, combinado com seus vizinhos (mesclado) e, este ponto mesclado é emitido. Após isso, o ponto original e seus vizinhos são removidos do conjunto de entrada. Esse procedimento é repetido até o conjunto de entrada estar vazio. O conjunto de pontos mesclados torna-se o novo conjunto de pontos, que pode ser denotado por P_j^2 .

Seleção de imagens. Começando com o conjunto completo de imagens, cada imagem é testada, e removida se a restrição de cobertura ainda for satisfeita após a remoção. O teste de remoção é realizado para todas as imagens enumeradas em ordem crescente de resolução de imagem (# de pixels), de modo que as imagens de baixa resolução sejam removidas primeiro. Observe que as imagens são descartadas permanentemente nesta etapa para acelerar as seguintes etapas principais de otimização.

Divisão de *cluster*. Em seguida, é aplicada a restrição de tamanho; *cluster* de imagens é dividido em componentes menores caso viole a restrição de tamanho, usando-se um algoritmo de *Normalized-Cuts* em um grafo de visibilidade, onde os nós são imagens. O peso da borda entre um par de imagens (I_l, I_m) mede o quanto I_l e I_m contribuem, juntas, para a reconstrução MVS em pontos SFM relevantes: $e_{lm} = \sum_{P_j \in \Theta^{lm}} \frac{f(P_j, I_l, I_m)}{f(P_j, V_j)}$, onde Θ^{lm} denota um conjunto de pontos SFM visíveis em I_l e I_m . Intuitivamente, as imagens com alta contribuição no MVS têm pesos altos entre eles e são menos propensos a serem cortados. A divisão de um *cluster* se repete até que a restrição de tamanho seja satisfeita para todos os *clusters*.

Adição de imagens. A restrição de cobertura pode ter sido violada na etapa anterior, e agora são adicionadas imagens a cada *cluster* para cobrir mais pontos SFM e restabelecer a cobertura. Nesta etapa, primeiro é construída uma lista de ações possíveis, onde cada ação mede a eficácia de adicionar uma imagem a um *cluster* para aumentar a cobertura. Para cada ponto SFM que está descoberto, P_j , seja $C_k = \text{argmax}_{Cl} f(P_j, Cl)$ o *cluster* com a máxima precisão de reconstrução. Então, para P_j , é criada uma ação $(I \rightarrow C_k), g$ que adiciona a imagem $I (\in V_j, \notin C_k)$ a C_k , onde g mede a eficácia. Só são consideradas ações que adicionam imagens ao C_k em vez de cada *cluster* que poderia cobrir P_j , para eficiência computacional. Como as ações com a mesma imagem e com o mesmo *cluster* são geradas a partir de vários pontos SFM, ocorre uma mescla dessas ações ao resumir a eficácia medida g . As ações na lista são classificadas em uma ordem decrescente de sua eficácia. Tendo construído uma lista de ações, uma abordagem seria tomar a ação com a pontuação mais alta, então refazer a lista novamente, o que é computacionalmente muito caro.

Alternativamente, são consideradas ações com pontuações maiores que 0, 7 vezes a pontuação mais alta na lista; em seguida, repete-se a ação a partir do topo da lista. Como uma ação pode alterar a eficácia de outras ações semelhantes, depois de tomar uma ação, remove-se quaisquer conflito da lista, onde duas ações $(I \rightarrow C), g, (I' \rightarrow C'), g'$ estão em conflito se I' e I são vizinhos. A construção da lista e a adição da imagem são repetidas até que a restrição de cobertura seja satisfeita.

Após a adição da imagem, a restrição de tamanho pode ser violada e, neste caso, as duas últimas etapas são repetidas até que ambas as restrições sejam satisfeitas. O passo seguinte, depois de obtido o *cluster* das imagens, é um algoritmo de reconstrução MVS, neste caso, o PMVS-2 (Patch-based Multiview Stereo Versão 2) (TEERAVECH, 2013; LI et al., 2013b; FURUKAWA et al., 2010), extensamente utilizado.

3.5.3.3 PMVS-2

O PMVS-2 utiliza a técnica de DoG descrita anteriormente e cantos de Harris (HARRIS; STEPHENS, 1988). O DoG é utilizado para detecção de bordas, e o operador de Harris emprega uma auto-correlação local para melhorar a consistência da borda, extraindo a borda e os cantos dos *features* das imagens. A resposta de Harris é positiva em regiões com cantos, negativa em bordas e pequena em regiões planas. Além disso, no PMVS-2, usando pontos de amostras das imagens como sementes, as linhas epipolares são usadas para encontrar a região correspondente (dentro de uma área de 2x2px) em outra imagem, gerando *patches* (cada uma definida com seu centro, normal e visibilidade) para atender às restrições na visibilidade, e levando a uma correspondência baseada em *patches* 2D entre imagens. A correspondência *Multi-view* no PMVS-2 depende da consistência fotométrica média de todos os pares visíveis. Um *patch* é reconstruído maximizando o valor consistência média da foto e, em seguida, aceitando somente se o número de imagens visíveis for maior ou igual a três.

A superfície do objeto é aproximada por um pequeno retângulo, o *patch* em 3D. O *patch*(p) é um retângulo modelado pela posição central $c(p)$, pelo vetor normal $n(p)$, pelos eixos x e y e pela imagem de referência $R(p)$, que é a imagem que melhor representa a visibilidade do *patch*. Seu tamanho é determinado por sua projeção na imagem de referência $R(p)$. A imagem é dividida em células (*grid*), de $\beta \times \beta$ pixels (usualmente 2x2). O ideal é reconstruir um patch por célula. Quanto menor a célula, maior será a densidade na nuvem de pontos final.

O PMVS-2 pode ser dividido em algumas etapas:

1. Inicialização:

- Detecção de *features*;

- Correspondência guiada;

2. Expansão;

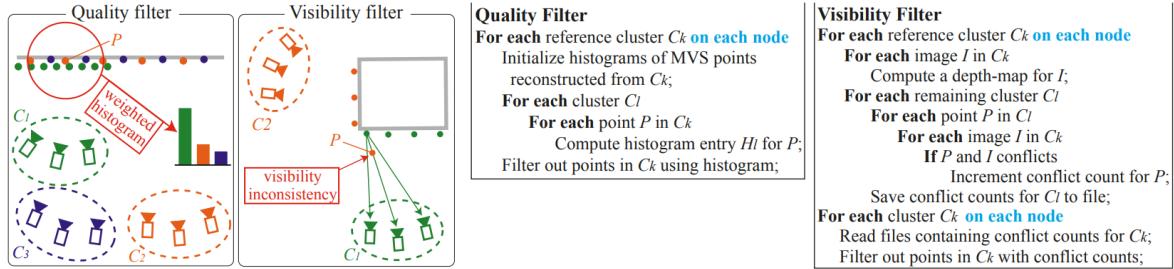
3. Filtragem.

Detecção de *features*. O PMVS-2 padrão utiliza o DoG em conjunto com o algoritmo de cantos de Harris, onde é criada uma linha epipolar, e todos os pontos em comum nesta linha são considerados consistentes para a reconstrução. Em seguida cria-se o *patch*, onde o $c(p)$ é calculado pela triangulação dos *features* detectados das imagens. A normal $n(p)$ é o cálculo da relação do vetor $c(p)$, multiplicado pela centro óptico da imagem, pelo módulo do numerador. E $R(p)$ é a imagem de referência propriamente dita. São otimizadas as orientações e posições de todos os *patches*. Com a inicialização já foi finalizada, temos como parte da entrada o resultado da reconstrução esparsa. No caso do VisualSfM, como a reconstrução esparsa já é feita em passos anteriores (com o PBA, Seção 3.5.3.1), na realidade o código do PMVS-2 só é empregado para a reconstrução densa, ou seja, a inicialização (SfM) não é feita pelo PMVS-2 no VisualSfM.

Expansão. Cada ponto 3D na nuvem de pontos é usado como semente para um algoritmo de expansão (aumento da região). Um *patch* utilizado como semente é expandido da seguinte forma. Um novo *patch* é projetado em uma célula vizinha. A posição é definida para a interseção do raio projetado para trás e no plano de seu patch pai, usando a mesma orientação (o vetor normal é propagado) e a mesma imagem de referência. Novamente, são otimizadas as orientações e posições, porém, do novo *patch*.

Filtragem. É aplicada uma consistência de visibilidade global, onde os *patches* que não são visíveis pelos centros ópticos das imagens, são descartados (estão dentro da superfície). Para isso, são utilizados dois filtros: filtro de qualidade e filtro de visibilidade, Figura 32.

Figura 32 - Descrição da etapa de filtragem



Nota: Usabilidade dos filtros de qualidade e visibilidade, aplicados fora do núcleo, assim como em paralelo. À esquerda, um ponto MVS P é testado usando os filtros. À direita, temos pseudocódigos, onde os *loops* em azul podem ser executados em paralelo.

Fonte: FURUKAWA et al., 2010.

Filtro de Qualidade. A mesma região de superfície pode ser reconstruída em múltiplos clusters com qualidade de reconstrução variável: grupos próximos produzem pontos densos e precisos, enquanto os distantes produzem pontos escassos e ruidosos. Deseja-se filtrar o último. Sejam P_j e V_j um ponto MVS e suas informações de visibilidade estimadas pelo algoritmo MVS, respectivamente. Supondo que P_j foi reconstruído a partir do *cluster* C_k , primeiramente coletamos pontos MVS Q_m e a informação de visibilidade V_m de todos os *clusters* que possuam normais compatíveis com P_j , isto é, a diferença de ângulo menores que 90° e com seus locais projetados dentro de n pixels de P_j em cada imagem em V_j . A partir dos pontos MVS obtidos, calcula-se um histograma (H_l), onde H_l é a soma da acurácia das precisões $f(Q_m, V_m)$ associadas a pontos MVS reconstruídos a partir de C_l . Se um *cluster* possui pontos acurados e densos, deverá ter um valor significativamente maior do que os outros. P_j é filtrado se o valor do histograma H_k correspondente for inferior a metade do máximo: $H_k < 0,5 \max_l H_l$. Repete-se este procedimento examinando cada *cluster* de referência, que pode ser executado em paralelo.

Filtro de Visibilidade. O filtro de visibilidade reforça a consistência das informações de visibilidade associadas aos pontos MVS durante toda a reconstrução. O filtro é, de fato, muito semelhante ao usado no PMVS original. A diferença é que o PMVS reforça a consistência intra-cluster dentro de cada cluster, enquanto o novo filtro reforça a consistência de visibilidade inter-cluster em uma reconstrução inteira comparando as saídas PMVS de

todos os clusters. Mais concretamente, para cada ponto MVS, conta-se o número de vezes que ele conflita com reconstruções de outros clusters. O ponto é eliminado se a contagem de conflitos for superior a três (FURUKAWA et al., 2010; FURUKAWA; PONCE, 2010).

4 EXPERIMENTOS

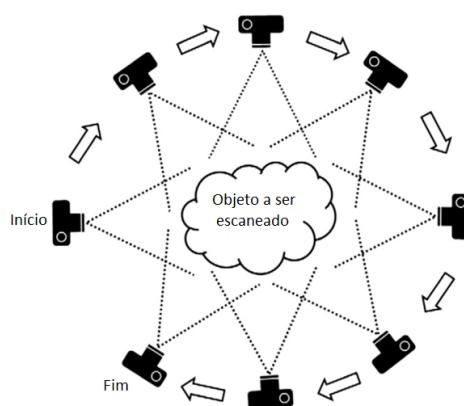
Os experimentos se concentram em explorar, ao longo deste Capítulo, os objetivos propostos na Seção .

4.1 Procedimento

Primeiramente, foram realizadas filmagens iniciais de algumas esculturas, utilizando-se a câmera de um *smartphone* convencional, primariamente um iPhone 5s, na resolução *Full HD* de 1920x1080 pixels. Esta filmagem foi realizada de forma conveniente a um usuário leigo, sem grandes restrições, procurando-se varrer a maioria da superfície da escultura em 360°, Figura 33. Em seguida, fizemos mais alguns vídeos, capturando alguns pontos que possuíam mais detalhes mais de perto.

Vale ressaltar que no dia das filmagens, estava ensolarado com boa iluminação para a captura das esculturas

Figura 33 - Como foi realizada a varredura das esculturas



Fonte: O autor (2017).

Com este material, foram feitos videoclipes extraiendo um subconjunto de *frames* do vídeo, para ao menos inicialmente não utilizarmos uma quantidade muito grande de imagens. Houve atenção para não extrair apenas *frames* muito juntos, pois aumentaria o

número de correspondências ambíguas entre as imagens e com isso, o processamento da reconstrução demoraria mais. Também procurou-se evitar usar apenas *frames* muito distantes, pulando os muito próximos, pois ocorreria o inverso: com menos correspondências, ficariam buracos (partes sem a informação necessária) na reconstrução, como descrito na Seção 3.4. Com isso em mente, nos experimentos iniciais foram reconstruídas duas esculturas empregando o VisualSfM, totalizando 197 imagens, e, com o MVE, utilizamos dois vídeos, que, ao selecionar os frames, totalizou cerca de 280 imagens. Para critério de comparação, usamos o tempo de reconstrução nos dois programas que é importante para termos um parâmetro de quanto demoraria para reconstruirmos uma escultura com mais, ou com menos imagens. E também as áreas faltantes nas reconstruções, avaliando a qualidade da reconstrução e o nível de detalhes obtidos com os softwares.

Além de esculturas ao ar livre, fizemos alguns testes em ambiente fechado, dentro de uma casa, por exemplo. Foi utilizado um objeto feito de cabaça (casca de abóbora), a qual possui uma reflectância propícia (Lambertiana) para uma reconstrução (BASRI; JACOBS, 2003), ao mesmo tempo sendo um objeto de arte e suave, com certos desafios similares às esculturas do Jardim do Nêgo. Com o procedimento descrito anteriormente, a partir dos vídeos feitos, selecionamos um total de 200 imagens em um vídeo superficial e mais 24 imagens mais detalhadas do objeto, ambos numa resolução de 1080x1920 pixels. Para um mesmo conjunto de imagens, foi executado tanto o VisualSfM quanto o MVE.

4.1.1 Resultados da reconstrução com o VisualSfM

Seguindo o passo-a-passo de reconstrução do software, obtivemos os seguintes resultados em relação ao tempo, Tabela 4. Para a escultura de 197 imagens do Jardim do Nêgo, com as reconstruções mostradas nas Figuras 34 e 35.

Percebe-se que foi gerada uma nuvem de pontos bastante consistente, a partir da reconstrução esparsa do algoritmo PBA. Por conta disso, nossa reconstrução 3D densa, empregando o MCBA/PMVS-2, obteve uma qualidade razoável para o conjunto de ima-

Tabela 4 - Tempos obtidos da reconstrução da escultura do Jardim do Nêgo usando o VisualSfM

Procedimento	Tempo (aprox.)
Carregamento de imagens	10 segundos
Calcular pares correspondentes de <i>features</i>	1 hora e 50 minutos
Gerar a reconstrução esparsa do modelo	4 minutos
Gerar a reconstrução densa do modelo	24 minutos

Fonte: O autor (2017).

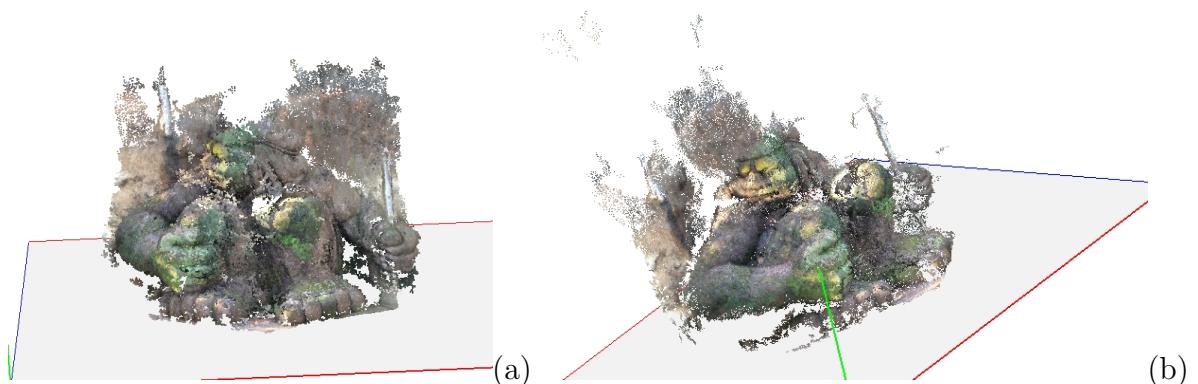
gens usado, figura (??), apesar de apresentar ainda ruídos e falta de resolução para uma preservação de patrimônio completa como no projeto *Digital Michaelangelo* de escaneamento a laser. Nota-se que o resultado é bastante surpreendente, dado que apenas os vídeos foram fornecidos, e o software teve que calcular as posições das câmeras, além da e reconstrução geométrica e fotométrica.

Figura 34 - Reconstrução esparsa da escultura do Jardim do Nêgo no VisualSfM com 197 imagens.



Fonte: O autor (2017).

Figura 35 - Resultados da reconstrução densa da escultura do Jardim do Nêgo



Legenda: Usando o software VisualSfM, em dois ângulos diferentes (a) e (b).

Fonte: O autor, 2017.

Com o objeto em ambiente fechado, conseguimos os resultados a seguir. Para o primeiro vídeo, convertido em 200 imagens, os tempos estão relatados na Tabela 5.

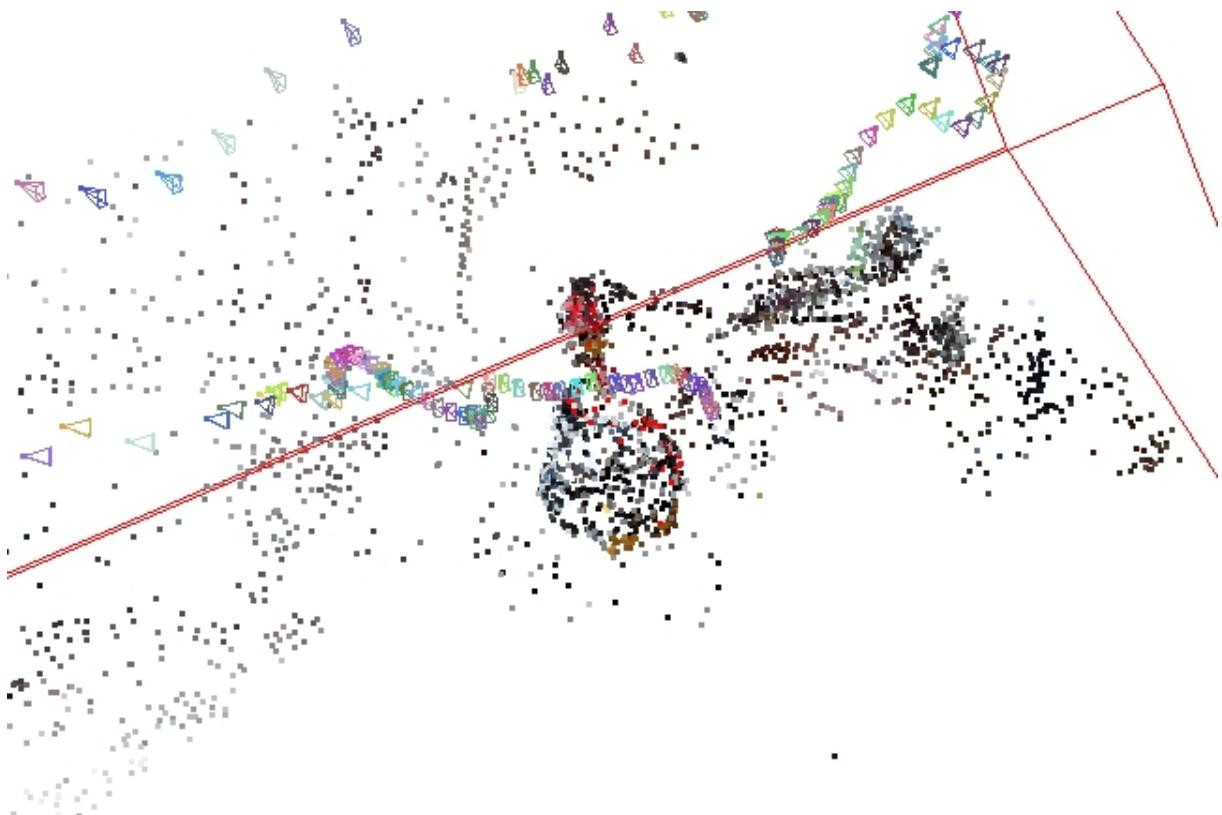
Tabela 5 - Tempos obtidos da reconstrução do objeto, com 200 imagens usando o VisualSfM

Procedimento	Tempo (aprox.)
Carregamento de imagens	50 segundos
Calcular pares correspondentes de <i>features</i>	2 horas e 40 minutos
Gerar a reconstrução esparsa do modelo	2 minutos e 25 segundos
Gerar a reconstrução densa do modelo	24 minutos

Fonte: O autor (2017).

A Figura 36 mostra o resultado da reconstrução esparsa pelo algoritmo PBA. Nota-se que não é tão nítida como na reconstrução densa, Figura 37, dada a quantidade de ruídos provenientes de outros objetos presentes na cena (o VisualSfM só identifica objetos estáticos). Só é possível limpar a malha manualmente, pressionando a tecla F1 e selecionando a área desejada para ser deletada. Não é muito prático, pois podemos excluir alguns pontos importantes. O ideal seria fazer esta limpeza por meio de programas externos.

Figura 36 - Reconstrução esparsa do objeto no VisualSfM com 200 imagens.



Fonte: O autor (2017).

Figura 37 - Reconstrução densa do objeto no VisualSfM com 200 imagens.



Fonte: O autor (2017).

Outra reconstrução também foi realizada neste trabalho, utilizando os dois vídeos do segundo objeto (cabaca), gerando 224 imagens, onde quando era usado um conjunto maior o programa parava de funcionar por falta de memória, mesmo após ajustar parâmetros (como o número de vizinhos, número de *cores* do processador, *level* do PMVS, entre outros) para tentar melhorar esse problema. Portanto, o experimento seguiu da forma: os tempos estão relatados na Tabela 6.

Tabela 6 - Tempos obtidos da reconstrução do objeto, com 224 imagens usando o VisualSfM

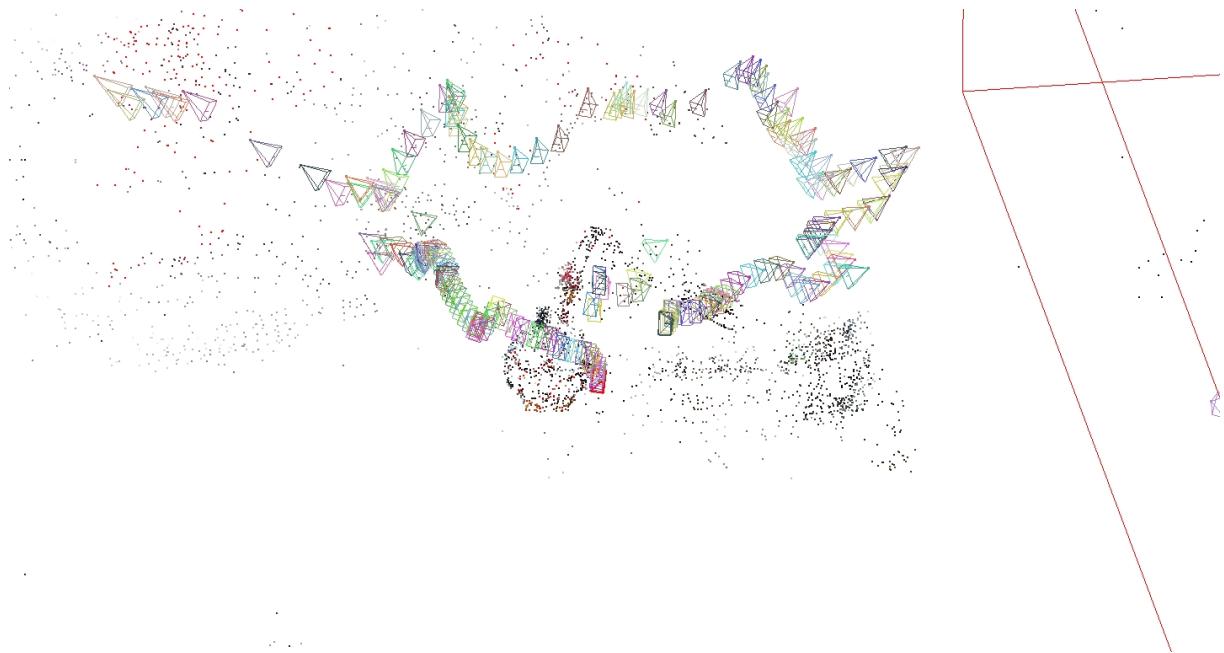
Procedimento	Tempo (aprox.)
Carregamento de imagens	1 minuto
Calcular pares correspondentes de <i>features</i>	3 horas
Gerar a reconstrução esparsa do modelo	3 minutos
Gerar a reconstrução densa do modelo	32 minutos

Fonte: O autor (2017).

Percebe-se que não foi tão proveitoso (qualitativamente) usar mais imagens neste

caso; inclusive, o algoritmo perdeu a referência do objeto e gerou um segundo modelo espelhado na reconstrução esparsa, Figura 38, e, consequentemente, na reconstrução densa, Figuras 40 e 40, o que gerou uma certa incoerência na reconstrução.

Figura 38 - Reconstrução esparsa do objeto com 224 imagens no VisualSfM.

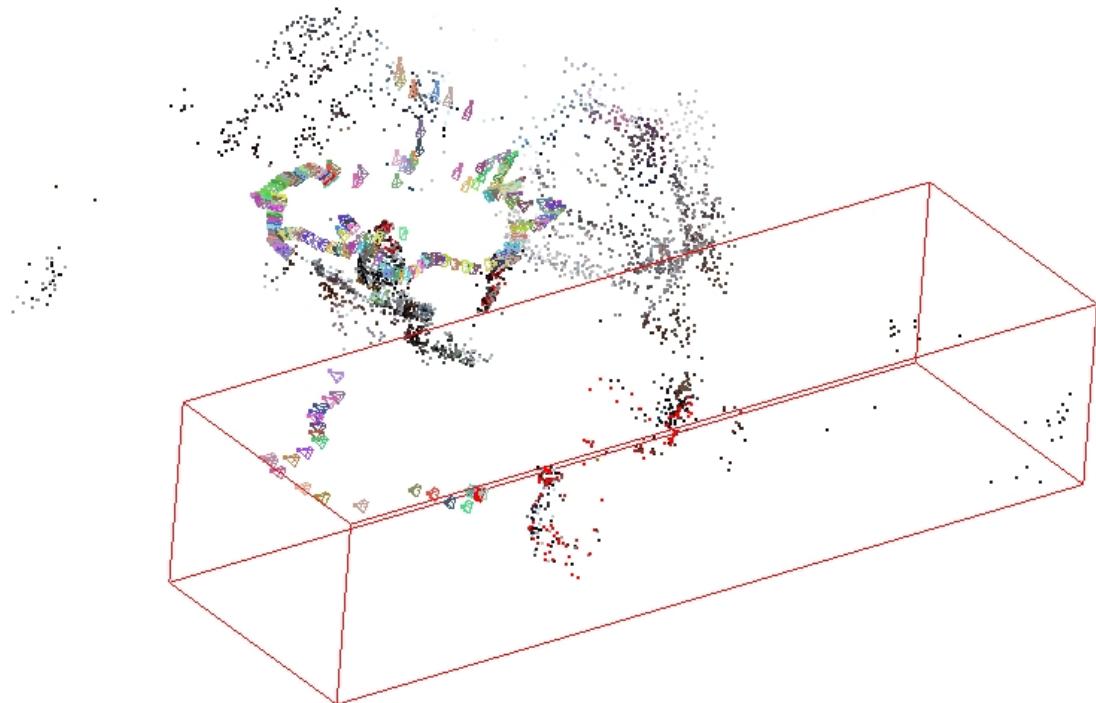


Fonte: O autor (2017).

4.1.2 Resultados da reconstrução com o MVE

A utilização do software é bem intuitiva, seja por linha de comando ou pela interface gráfica (neste modo, fica mais fácil visualizar cada etapa da reconstrução). O programa é amplamente configurável, podendo escolher a vizinhança, escala, manter o mapa de profundidade, ver os dados *EXIF* de cada imagem, dentre outras configurações.

Figura 39 - Caso de falha do VisualSfM



Nota: Foram gerados dois modelos refletidos esparsos do objeto a partir do conjunto inicial de 224 imagens, provavelmente, proveniente da falta de informações para estimar as câmeras.

Fonte: O autor, 2017.

Entretanto, para a aplicação proposta neste projeto, o MVE não é muito interessante, visto que utiliza fortemente a informação *a priori* das câmeras inseridas nas imagens (*EXIF*) e, como as imagens empregadas na reconstrução são, tecnicamente, vídeos cortados em determinados *frames*, não é fácil obter essa informação das câmeras [42](#). Logo o, software não tem tanta aplicabilidade neste caso, pois pode recair no problema dos parâmetros padrões adotados para as câmeras não serem bons o suficiente para estes conjuntos de dados. A menos que sejam tiradas fotos sequenciais de alguma escultura ou objeto que se deseja gerar a reconstrução densa, pois dessa forma, as informações necessárias das câmeras estarão armazenadas no *EXIF*.

Foi gerada uma reconstrução no MVE de um vídeo gravado de uma escultura no Jardim do Nêgo. O vídeo foi reduzido a *frames* de onde foram selecionadas 200 imagens base, com os parâmetros de câmera gerados pelo próprio software, mesmo sem o *EXIF* para ajudar na estimativa inicial. A partir disso, foram executados todos os passos de uma reconstrução utilizando o MVE, de forma que foram utilizadas as duas opções, tanto

Figura 40 - Resultado das reconstruções densas do objeto



Legenda: (a) - reconstrução densa do primeiro modelo;(b) - reconstrução densa do segundo modelo.

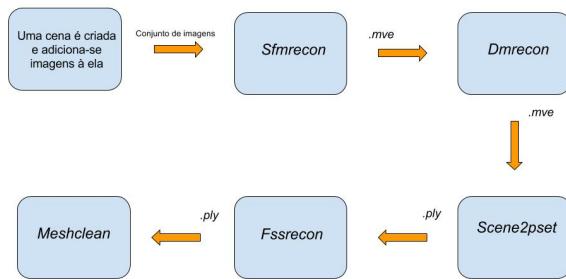
Nota: Reconstrução do objeto no VisualSfM com 224 imagens.

Fonte: O autor, 2017.

por linha de comando, quanto pela interface gráfica (UMVE).

Pela interface gráfica, o processo todo de reconstrução foi rápido (cerca de 30 minutos), Figura 43, ao passo que por linha de comando, levou cerca de 11 horas e 30 minutos. Portanto, vamos nos atentar somente à reconstrução por linha de comando, onde, resumindo, tivemos os resultados na Tabela 7. O UMVE não sinaliza quando o processo em execução termina; a explicação para essa discrepância no tempo é devido

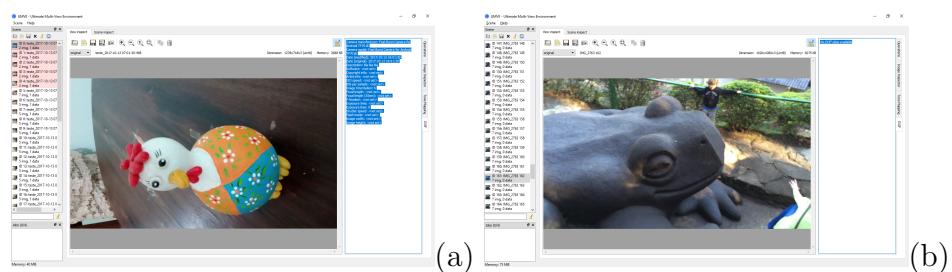
Figura 41 - Procedimento do MVE



Nota: Processo de reconstrução usando o MVE por linha de comando: O *sfmrecon* executa os passos do SfM no conjunto de imagens e gera um arquivo *.mve* para cada imagem. Após isso, são associados mapas de profundidade a cada *.mve* com o *dmrecon*. Em seguida o *scene2pset* cria uma nuvem de pontos densa com a união de todas as amostras de todos os mapas de profundidade. O *fssrecon* executa o algoritmo de reconstrução de superfícies 3D e gera uma malha daquele objeto a ser reconstruído e, por fim, o *meshclean* limpa os ruídos do modelo e retira algumas triangulações defeituosas da etapa do *fssrecon*.

Fonte: O autor, 2017.

Figura 42 - Extensão EXIF

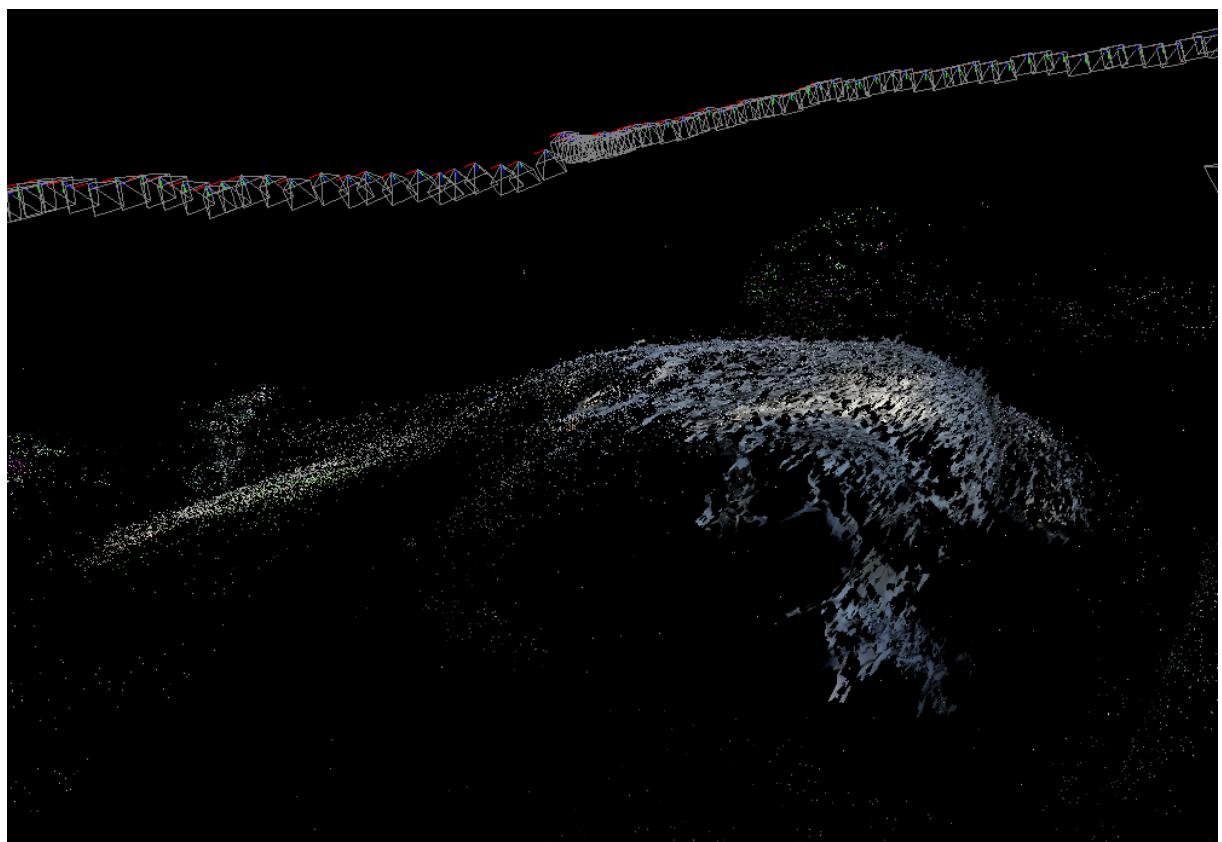


Legenda: (a) - imagem com dados na extensão EXIF (em azul);(b) -um frame de um vídeo, que não possui os dados das câmeras (em azul).

Fonte: O autor, 2017.

à execução de outro comando, sobrepondo o que já estava sendo executado, sem que o primeiro tivesse terminado.

Figura 43 - Reconstrução usando a interface gráfica do MVE



Nota: Reconstrução final via UMVE; percebe-se que alguns pontos não foram considerados, mas o resultado geral foi uma nuvem de pontos mais densa.

Fonte: O autor, 2017.

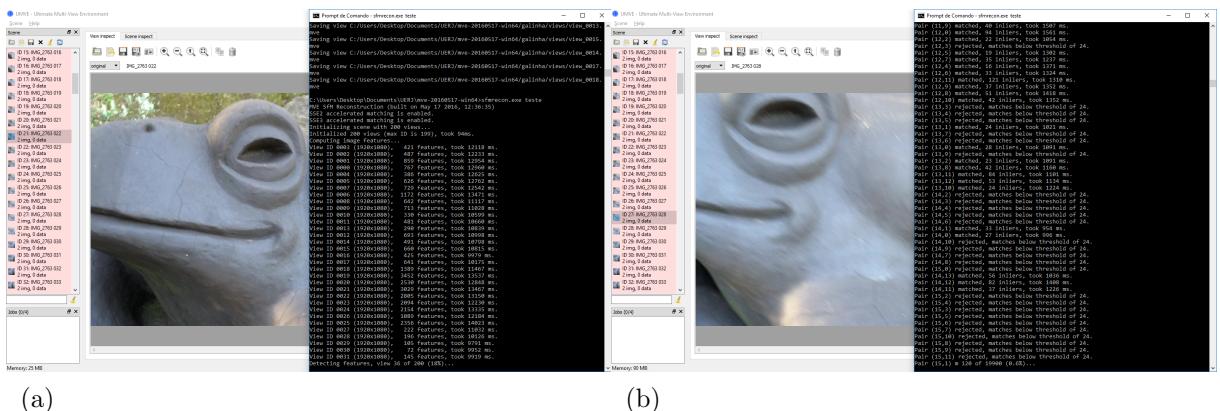
Tabela 7 - Tempos obtidos usando o MVE em um conjunto de dados do Jardim do Nêgo

Comando	Tempo (aprox.)
<i>sfmrecon</i>	1 minuto e 28 segundos
<i>dmrecon</i>	4 horas
<i>scene2pset</i>	10 minutos
<i>fssrecon</i>	7 horas
<i>meshclean</i>	1 minuto

Fonte: O autor (2017).

Na reconstrução por linha de comando também é possível visualizar em qual etapa da execução o algoritmo está (Figura 44), configurar alguns parâmetros e inclusive mostrar a porcentagem de progresso do comando em execução. Foram executados os comandos declarados nesta seção. O *sfmrecon* demorou cerca de 1 minuto e meio.

Figura 44 - Linha de comandos do *sfmrecon*



Nota: Processos dentro do comando *sfmrecon*, onde (a) estão sendo detectadas as *features* do conjunto de imagens e em (b) está computado o *pairwise matching*.

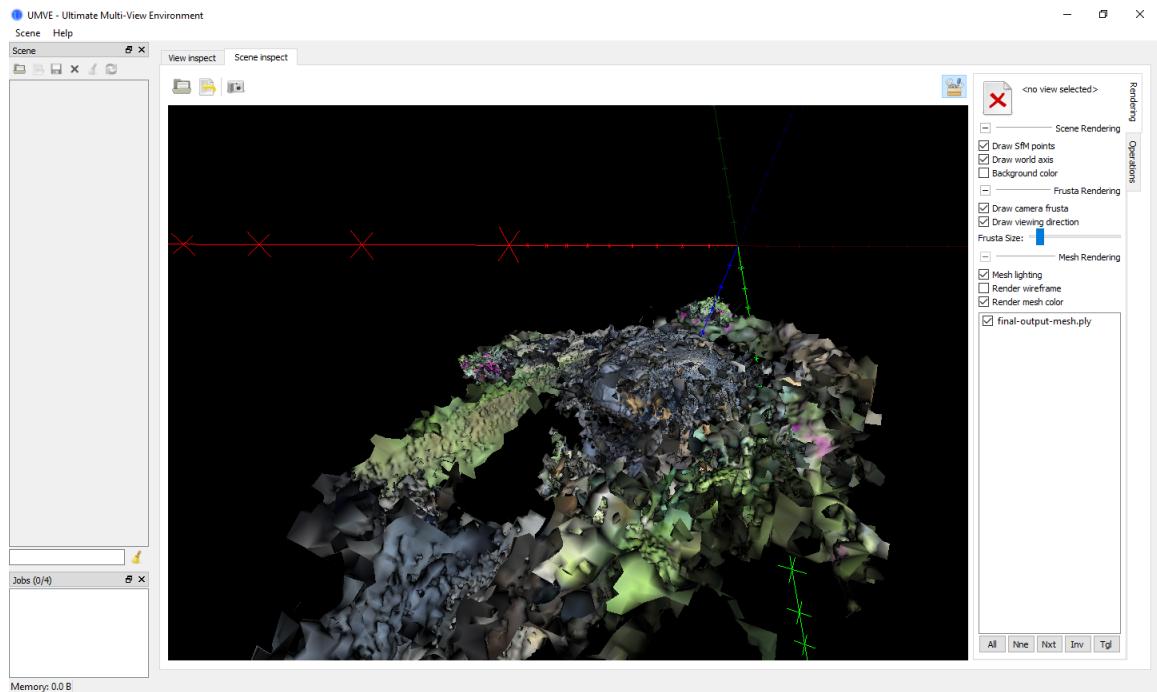
Fonte: O autor, 2017.

O próximo comando, *dmrecon* demorou cerca de 4 horas, usando como configuração um nível L2, com 20 vizinhos. Usando um nível L0, o algoritmo rodou durante 6 horas aproximadamente e foi cancelado devido à demora na execução.

Usando o *scene2pset*, é necessário especificar em qual nível estamos reconstruindo e também uma saída válida. Por exemplo: *scene2pset.exe -Fnivel cena output*, onde o nível (escala do tamanho das imagens) poderá ser um 0 (-F0), 1 (-F1) e assim por diante; a cena é o *input* e o *output* é um arquivo de extensão configurável, neste caso .ply. Este comando foi rápido, demorou cerca de 10 minutos, levando em conta todos os níveis.

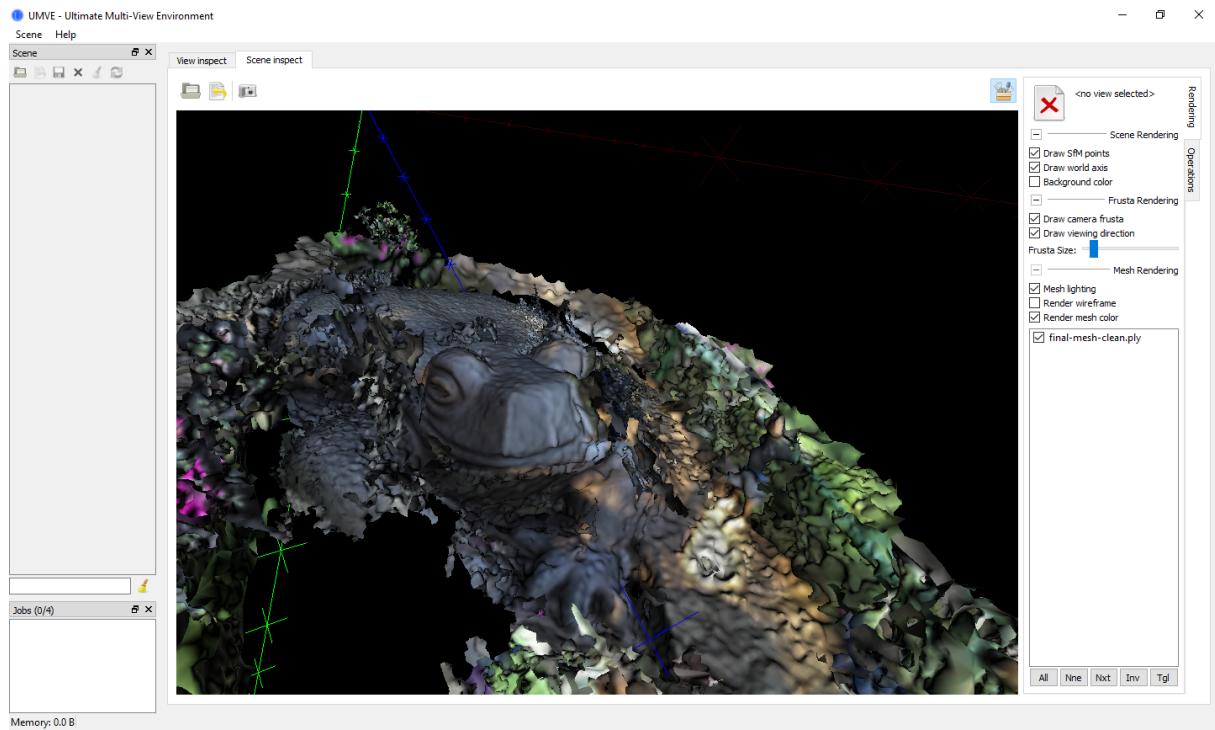
Para juntar todos os níveis do `scene2pset`, foi usado o `fssrecon`, que gera uma única reconstrução. Este processo demorou bastante, cerca de 7 horas e teve como resultado a malha da Figura 45. Por fim, basta limpar a malha atual com o comando `meshclean`, onde foi obtido o resultado 46.

Figura 45 - Malha com ruídos proveniente do comando *fssrecon*.



Fonte: O autor (2017).

Figura 46 - Resultado final, após a remoção dos ruídos da malha.



Fonte: O autor (2017).

Para comparação, usou-se o mesmo objeto utilizado na reconstrução do VisualSfM (a galinha) e foi feito o passo a passo com o MVE: com a interface gráfica (UMVE), criou-se uma nova cena inserido-se, primeiramente, as 200 fotos do objeto. Em seguida, utilizando as linhas de comando do MVE, foi feito o procedimento padrão de reconstrução do software. Foram obtidos os resultados na Tabela 8:

Tabela 8 - Tempos obtidos usando o MVE em um conjunto de dados em ambiente interno com 200 imagens

Comando	Tempo (aprox.)
<i>sfmrecon</i>	6 minutos
<i>dmrecon</i>	1 hora
<i>scene2pset</i>	5 minutos
<i>fssrecon</i>	29 minutos
<i>meshclean</i>	45 segundos

Fonte: O autor (2017).

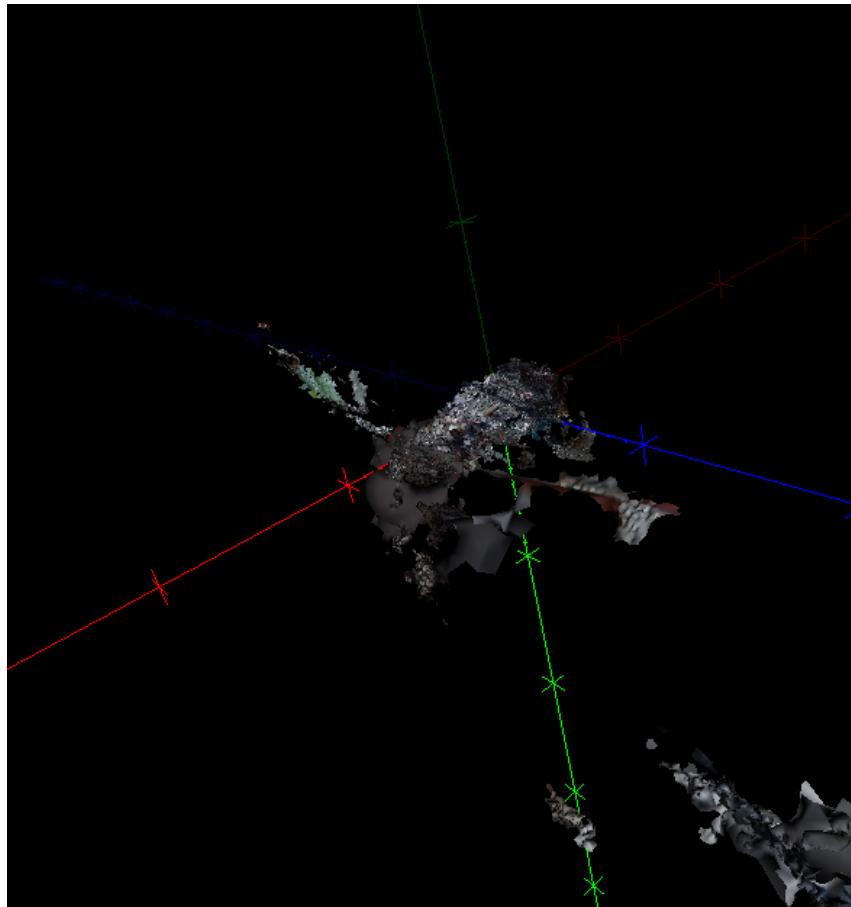
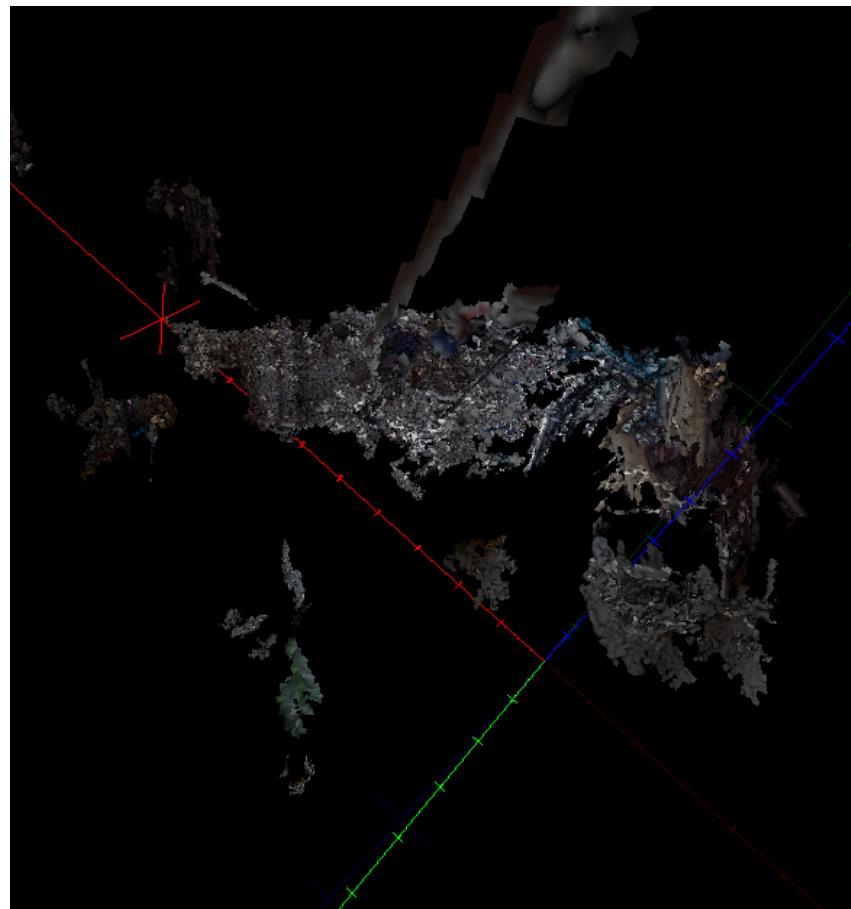


Figura 47 - Resultado da etapa *fssrecon* do MVE

A etapa de *scene2pset* demorou cerca de 20 segundos (total). Porém a reconstrução não foi satisfatória, o software se confundiu, e não conseguiu obter os parâmetros corretos das câmeras utilizadas. A partir disso, o erro se propagou e gerou a reconstrução da Figura 48. Rodamos também com as 224 fotos, mas só foi possível executar o passo *sfm-recon*, pois o MVE não conseguiu rodar o comando *dmrecon* provavelmente o SIFT falhou na correspondências entre as imagens, e não gerou nenhum resultado para a continuação do algoritmo.

Figura 48 - Resultado da etapa *meshclean*, da etapa anterior 47



Fonte: O autor (2017).

4.1.3 Resultados da reconstrução com o Sense

Para a reconstrução com o Kinect, utilizamos uma ferramenta de escaneamento que utiliza o mesmo princípio do Kinect versão 1 (baseado em luz estruturada): o Sense versão 1 ([3DSYSTEMS, 2017](#)).

Seu formato facilita o manuseio do scanner para conseguirmos realizar uma boa varredura (figuras 49, 50).

Figura 49 - Caixa do Sense



Fonte: O autor (2017).

Primeiramente devemos ativar e cadastrar o hardware no *site* do proprietário e a partir disso, baixar o software.

O software do Sense é de fácil entendimento (figura 51), caso necessário, pode-se usar a tradução para o português e possui um *link* para o guia do usuário onde é explicado todo o funcionamento da ferramenta e do seu software, bem como boas práticas para ter um bom resultado na reconstrução. O programa também conta com uma boa customização (como tamanho do objeto a ser escaneado, resolução, orientação, se desejamos reconstruir as cores, entre outros parâmetros). Além disso, ele faz uma rápida avaliação

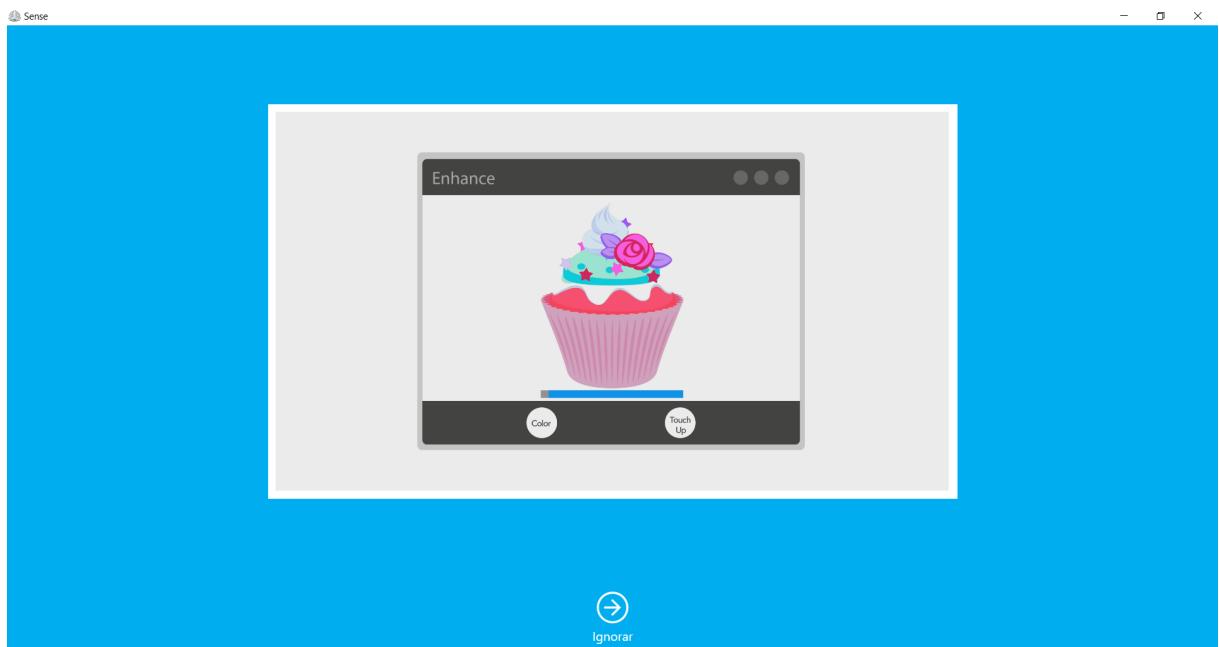
Figura 50 - O Sense



Fonte: O autor (2017).

do computador e recomenda fazer a reconstrução a nível de GPU ou CPU, dependendo do resultado.

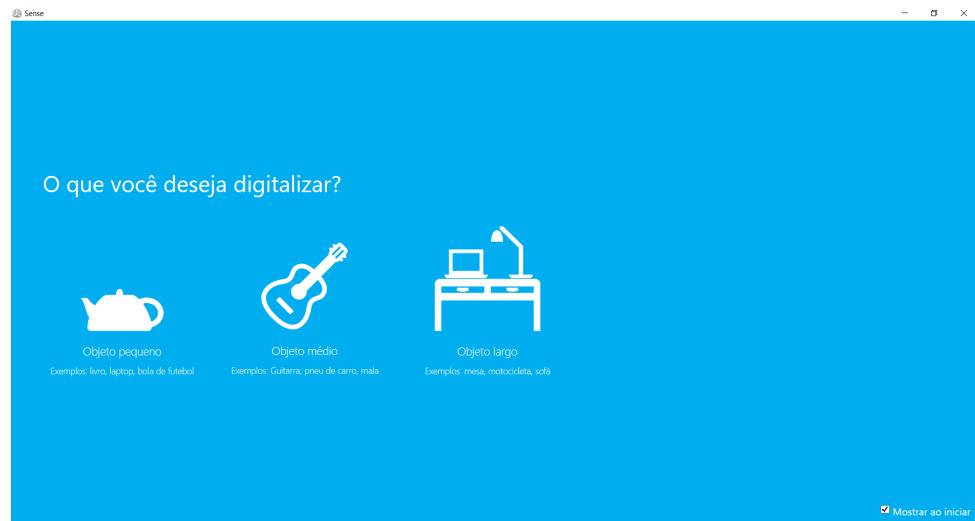
Figura 51 - Tela inicial do Sense



Nota: Quando abrimos o Sense, temos à disposição um vídeo explicativo com algumas informações, como dicas para um bom escaneamento e sobre como manusear o software da maneira correta.

Fonte: O autor, 2017.

Figura 52 - Interface do Sense



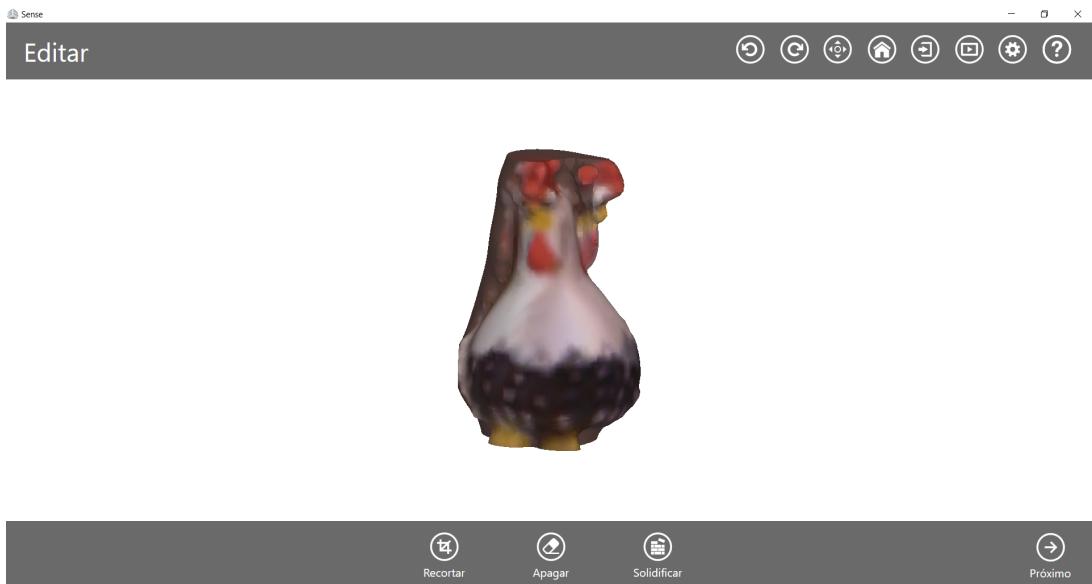
Nota: Assim que passamos o vídeo introdutório, o software nos dá a opção de escanearmos uma pessoa ou um objeto. Onde ao escolhermos um objeto, podemos selecionar o tamanho do objeto a ser escaneado. Notamos que o próprio software indica que os menores objetos seriam um livro ou um laptop, por exemplo.

Fonte: O autor, 2017.

Para comparação com os softwares que foram descritos anteriormente, fizemos o escaneamento apenas do objeto em ambiente fechado. O Sense tem dificuldades de escaneamento em áreas abertas e pode não funcionar tão bem, como descrito no *site* do proprietário ([3DSYSTEMS, 2017](#)).

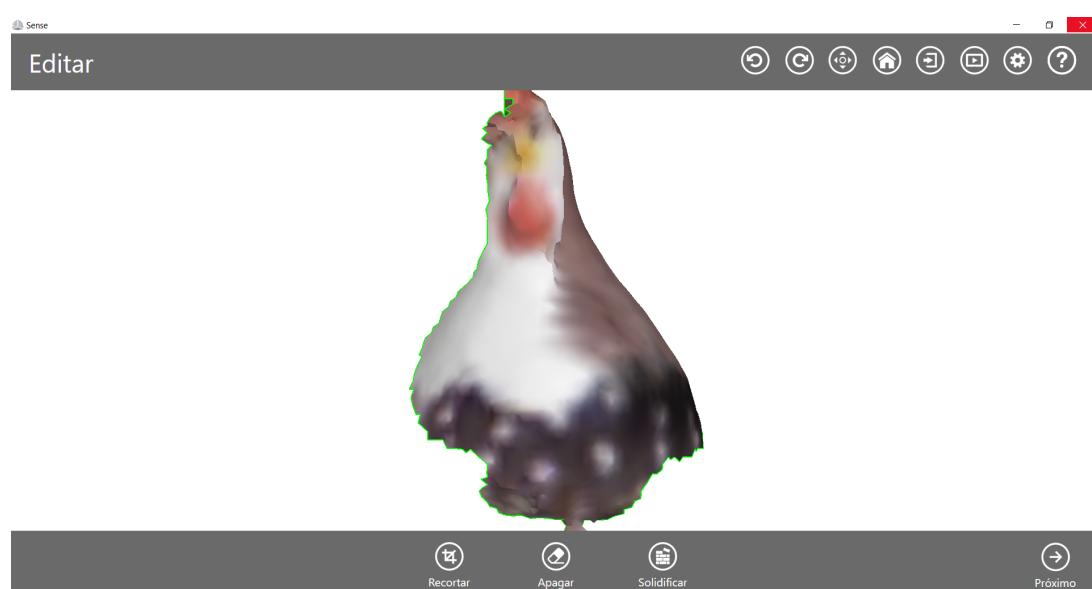
Para a galinha, obtivemos os resultados abaixo:

Figura 53 - Resultados do escaneamento com o Sense



Fonte: O autor, 2017.

Figura 54 - Resultados do escaneamento com o Sense



Nota: Como resultado da digitalização, o Sense não se mostrou muito eficaz, pois gerou uma reconstrução com buracos e com poucas informações.

Fonte: O autor, 2017.

Os buracos e a falta de informações são decorrentes da perda de referência do objeto pelo software, solicitando ao usuário que volte ao ponto anterior para retomar a referência do objeto, o que geralmente é mais fácil recomeçar a digitalização do zero pois dificilmente o software consegue achar novamente a referência do objeto.

Entrando em contato com o suporte do proprietário ([3DSYSTEMS, 2017](#)) fui noti-

ficado de que é necessário um computador com um processador Intel Core i5 ou superior e mesmo assim o software trava constantemente, o que torna o seu uso muito frustrante e que também colabora para a perda de referência do objeto.

Não avaliamos o tempo de reconstrução, pois o Sense (assim como o Kinect) faz o escaneamento em tempo real.

CONCLUSÃO

Técnicas práticas recentes de reconstrução 3D utilizando fotogrametria foram apresentadas e exploradas, visando ajudar a desenvolver os primórdios de uma metodologia prática para digitalizar jardins de esculturas a céu aberto. A partir dos resultados dos experimentos, averiguamos que é possível, do ponto de vista qualitativo, utilizar softwares de código aberto e apenas uma câmera comum de celular ou uma scanner de baixo custo (Kinect) para conseguirmos ter uma boa reconstrução de uma escultura ou objeto. Um pseudocódigo poderia ser executado da seguinte maneira, com o intuito de gerar uma reconstrução satisfatória:

Foi adquirida experiência sobre a calibração de equipamentos, sistemas de software recentes e esquemas para realizar uma boa varredura, cobrindo toda a escultura com uma câmera comum de celular. As esculturas friburguenses visadas neste trabalho possuem geometria peculiar, com curvas delineadas e regiões suaves, com pouca textura e poucos pontos de interesse, sendo desafiadoras para o estado da arte em escaneamento 3D; dessa forma, procuramos relatar as dificuldades e problemas encontrados, para justificar a eventual pesquisa em novas técnicas mais avançadas.

Dentre esses problemas, a nível de software, podemos explicitar a dificuldade de instalação do MVE e do VisualSfM pois possuem uma série de algoritmos embutidos no seu passo-a-passo e, portanto, várias bibliotecas deverão ser instaladas e algumas geraram conflitos, problemas para configuração dos parâmetros (como memória, número de núcleos utilizados na aplicação, número de vizinhos e escala da reconstrução, por exemplo). E a nível prático, temos todo o custo utilização do Kinect (ou do Sense) na reconstrução das esculturas, necessitando de uma infra-estrutura adequada para seu funcionamento (como computador e/ou alguma fonte de energia próxima).

Algoritmo 1 - Modelo de escaneamento

```

1: se Utilizarmos o Sense então
2:   Criar uma conta e Baixar o programa no site do proprietário (3D Systems)
3:   Cadastrar e ativar o Sense no próprio software;
4:   Configurar os parâmetros de acordo com o objeto a ser escaneado;
5:   Realizar o escaneamento, usando as dicas de boas práticas de digitalização descriptas no guia do usuário;
6:   enquanto A reconstrução não estiver satisfatória faça
7:     Repassar o scanner para tentar preencher as partes com falta de informações;
8:     Usar as ferramentas de edição (como apagar pontos indesejáveis, preencher os furos provenientes do escaneamento para obter um modelo sólido, entre outros);
9:   fim enquanto
10:  Exportar o modelo criado;
11: senão
12:   Faz-se um escaneamento da escultura ou do objeto, varrendo toda a superfície visível;
13:   Seleciona-se frames do vídeo;
14:   se Utilizarmos o VisualSfM então
15:     Pré-configurar o numero de cores do processador e escala da reconstrução a ser feita;
16:     Adicionar esse conjunto de frames ao programa;
17:     Executar o comando de correspondência entre as imagens;
18:     Executar o comando da reconstrução esparsa;
19:     Executar o comando da reconstrução densa;
20:     devolve Modelo com pontos densificados;
21:
22:   se Utilizar o modelo com pontos densificados como entrada no MVE então
23:     Gerar a reconstrução baseada em superfície;
24:     devolve Modelo com uma malha baseada em superfícies reconstruída do objeto;
25:   fim se
26: fim se

```

```

27: se Utilizarmos o MVE então
28:     Abrir a interface gráfica (pois é mais fácil, se compararmos este passo utilizando
    linha de comando) e cria-se uma cena;
29:     Adicionar imagens a essa cena criada e fechar a interface gráfica;
30:     enquanto Não executamos todos os procedimentos por linha de comando faça
31:         Executar sfmrecon, gerando conjuntos de extensões .mve;
32:         Executar dmrecon, onde são criados mapas de profundidade nas extensões
    .mve;
33:         Executar scene2pset, indicando o nível (escala da reconstrução), que com-
    bina todos os mapas de profundidade em uma única e grande nuvem de pontos;
34:         Executar fssrecon que utiliza a união dos vértices dos mapas de profun-
    didade e gera uma malha superficial do objeto;
35:     fim enquanto
36: fim se
37:     Exportar o modelo 3D gerado;
38:     Utilizar algum programa para tratamento do modelo 3D (Blender, Meshlab, entre
    outros);
39: fim se

```

Trabalhos futuros

Identificamos os seguintes caminhos para a evolução deste projeto:

- **Realizar uma varredura com o Kinect ou com o Sense.** Embora seja relativamente custoso, tanto fisicamente quanto computacionalmente, seria interessante ter um parâmetro de comparação prática das técnicas fotogramétricas passivas com as técnicas de fotogrametria por Kinect, que se mostrou muito promissor em um ambiente fechado.
- **Validação adicional.** Ter resultados mais expressivos, em questão quantitativa e não só qualitativa, para realizar uma engenharia mais completa do sistema, comparando valores em diferentes técnicas empregadas.
- **Constatar na prática, a melhor estratégia de varredura de esculturas.** Verificamos que um dos melhores modos de se escanear uma escultura de grande porte seria escaneá-la várias vezes, de perto e de longe, a fim de que se pegue todos os detalhes, cobrindo toda a área a ser reconstruída, e a fim de que a geometria global seja bem restringida. Mas será que este é realmente o melhor método?
- **Realizar uma reconstrução de curvas.** Utilizar uma reconstrução baseada em

curvas para auxiliar na reconstrução de nuvem de pontos e superfícies densas, já que as esculturas do Jardim do Nêgo não possuem ampla informação pontual, mas sim de curvas e bordas. Nossos resultados indicam que aliar essas técnicas em um sistema de software ainda maior seria benéfico, além de constituir uma possível contribuição científica para publicação a curto prazo.

- **Concretizar o objetivo proposto neste trabalho.** Ir mais vezes ao Jardim do Nêgo com o intuito de aumentar o acervo de filmes/imagens das esculturas de modo que seja possível ter uma reconstrução 3D satisfatória de todo o jardim, eternizando todo o patrimônio cultural.
- **Estratégia de larga escala.** De certa forma, os sistemas utilizados já são de larga escala (da ordem de centenas de imagens *Full HD*), mas para escanear todo um jardim de esculturas em alta resolução, com uma metodologia prática facilmente replicável, será necessário empregar mais fortemente técnicas relacionadas a *big data*, envolvendo o cluster do IPRJ e algoritmos de mais larga escala ([AGARWAL et al., 2009](#)). Pode-se pensar em um sistema em que o *smartphone* faz upload dos vídeos na medida em que são capturados, os quais são reconstruídos incrementalmente usando o *cluster* do IPRJ como nuvem computacional.

REFERÊNCIAS

- 3DSYSTEMS. 2017. Visitado em 16 Out. 2017. Disponível em: <<https://www.3dsystems.com>>.
- ABLAN, D. *Digital Photography for 3D Imaging and Animation*. Wiley, 2007. Visitado em 05 Set. 2017. Disponível em: <<https://books.google.com.br/books?id=nLSVKEavyPcC>>.
- AGARWAL, S. et al. Reconstructing rome. *Computer*, ieee, v. 43, n. 6, p. 40–47, 2010.
- AGARWAL, S. et al. Building rome in a day. *Communications of the ACM*, ACM, v. 54, n. 10, p. 105–112, 2011.
- AGARWAL, S. et al. Building rome in a day. In: IEEE. *International Conference On Computer Vision 12., 2009 IEEE 12th International Conference On Computer Vision*. IEEE, 2009. [S.l.], 2009. p. 72–79.
- BASRI, R.; JACOBS, D. W. Lambertian reflectance and linear subspaces. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 25, n. 2, p. 218–233, 2003.
- CHOUDHARY, S. Bundle adjustment : A tutorial.
- CULJAK, I. et al. A brief introduction to opencv. In: IEEE. *MIPRO, 2012 proceedings of the 35th international convention*. [S.l.], 2012. p. 1725–1730.
- DOBBERT, T. *Matchmoving: The Invisible Art of Camera Tracking*. Wiley, 2012. (Books 24x7 IT PRO). ISBN 9781118529669. Disponível em: <<https://books.google.com.br/books?id=Jr4YsaTrIBgC>>.
- FABBRI, R. *Multiview differential geometry in application to computer vision*. Tese (Doutorado) — Citeseer, 2011.
- FABBRI, R.; GIBLIN, P. J.; KIMIA, B. B. Camera pose estimation using first-order curve differential geometry. In: *Proceedings of the IEEE European Conference in Computer Vision*. [S.l.: s.n.], 2012.
- FABBRI, R.; KIMIA, B. B. 3D curve sketch: Flexible curve-based stereo reconstruction and calibration. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. San Francisco, California, USA: IEEE Computer Society Press, 2010.
- FABBRI, R.; KIMIA, B. B. Multiview differential geometry of curves. *International Journal of Computer Vision*, Springer, v. 117, p. 1–23, 2016. Disponível em: <<http://dx.doi.org/10.1007/s11263-016-0912-7>>.
- FREEDMAN, D. et al. SRA: fast removal of general multipath for tof sensors. *CoRR*, abs/1403.5919, 2014. Disponível em: <<http://arxiv.org/abs/1403.5919>>.
- FUHRMANN, S.; GOESELE, M. Floating scale surface reconstruction. *ACM Transactions on Graphics (TOG)*, ACM, v. 33, n. 4, p. 46, 2014.

- FUHRMANN, S.; LANGGUTH, F.; GOESELE, M. Mve: A multi-view reconstruction environment. In: *Proceedings of the Eurographics Workshop on Graphics and Cultural Heritage*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2014. (GCH '14), p. 11–18. ISBN 978-3-905674-63-7. Disponível em: [\(\)](http://dx.doi.org/10.2312/gch.20141299).
- FURUKAWA, Y. et al. Towards internet-scale multi-view stereo. In: IEEE. *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. [S.l.], 2010. p. 1434–1441.
- FURUKAWA, Y.; PONCE, J. *Dense patch models for motion capture from synchronized video streams*. [S.l.], 2007.
- FURUKAWA, Y.; PONCE, J. Accurate camera calibration from multi-view stereo and bundle adjustment. *International Journal of Computer Vision*, Springer, v. 84, n. 3, p. 257–268, 2009.
- FURUKAWA, Y.; PONCE, J. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 32, n. 8, p. 1362–1376, 2010.
- GAVA, C. Dense 3d reconstruction.
- GOESELE, M. et al. Multi-view stereo for community photo collections. In: IEEE. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. [S.l.], 2007. p. 1–8.
- GOKTURK, S. B.; YALCIN, H.; BAMJI, C. A time-of-flight depth sensor-system description, issues and solutions. In: IEEE. *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*. [S.l.], 2004. p. 35–35.
- HAIGHT, F. A. *Handbook of the poisson distribution*. Wiley, 1967.
- HARRIS, C.; STEPHENS, M. A combined edge and corner detector. In: *Alvey Vision Conference*. [S.l.: s.n.], 1988. p. 189–192.
- HENRY, P. et al. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, v. 31, n. 5, p. 647–663, 2012. Disponível em: [\(<https://doi.org/10.1177/0278364911434148>\)](https://doi.org/10.1177/0278364911434148).
- IZADI, S. et al. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: ACM, 2011. (UIST '11), p. 559–568. ISBN 978-1-4503-0716-1. Disponível em: [\(<http://doi.acm.org/10.1145/2047196.2047270>\)](http://doi.acm.org/10.1145/2047196.2047270).
- JINDAL, R.; VATTA, S. Sift: scale invariant feature transform. 2010.
- LACHAT, E. et al. First experiences with kinect v2 sensor for close range 3d modelling. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, Copernicus GmbH, v. 40, n. 5, p. 93, 2015.
- LEVOY, M. et al. The digital michelangelo project: 3d scanning of large statues. In: ACM PRESS/ADDISON-WESLEY PUBLISHING CO. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. [S.l.], 2000. p. 131–144.

- LI, B. et al. Improving pmvs algorithm for 3d scene reconstruction from sparse stereo pairs. In: SPRINGER. *Pacific-Rim Conference on Multimedia*. [S.l.], 2013. p. 221–232.
- LI, B. et al. Improving pmvs algorithm for 3d scene reconstruction from sparse stereo pairs. In: _____. *Advances in Multimedia Information Processing – PCM 2013: 14th Pacific-Rim Conference on Multimedia, Nanjing, China, December 13-16, 2013. Proceedings*. Cham: Springer International Publishing, 2013. p. 221–232. ISBN 978-3-319-03731-8. Disponível em: [⟨https://doi.org/10.1007/978-3-319-03731-8_21⟩](https://doi.org/10.1007/978-3-319-03731-8_21).
- LOWE, D. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, Springer, v. 60, n. 2, p. 91–110, 2004.
- MIKOLAJCZYK, K. *Detection of local features invariant to affines transformations*. Tese (Doutorado) — Institut National Polytechnique de Grenoble-INPG, 2002.
- MORÉ, J. J. The levenberg-marquardt algorithm: implementation and theory. In: *Numerical analysis*. [S.l.]: Springer, 1978. p. 105–116.
- OPTOENGINEERING. 2017. Disponível em: [⟨http://www.opto-engineering.com/⟩](http://www.opto-engineering.com/).
- PRESS, W. H. *The art of scientific computing*. [S.l.]: Cambridge university press, 1992.
- SCHMID, K. et al. Stereo vision based indoor/outdoor navigation for flying robots. In: IEEE. *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. [S.l.], 2013. p. 3955–3962.
- SILBERMAN DEREK HOIEM, P. K. N.; FERGUS, R. Indoor segmentation and support inference from rgbd images. In: *ECCV*. [S.l.: s.n.], 2012.
- SMISEK, J.; JANCOSEK, M.; PAJDLA, T. 3d with kinect. In: *Consumer depth cameras for computer vision*. [S.l.]: Springer, 2013. p. 3–25.
- SNAVELY, N. et al. Bundler: Structure from motion (sfm) for unordered image collections. Available online: *phototour. cs. washington. edu/bundler/*(accessed on 12 July 2013), v. 1, 2010.
- STRICKER, D. Structure from motion ii. 2015.
- SWEENEY, C. *Theia Multiview Geometry Library: Tutorial & Reference*. 2016. [⟨http://theia-sfm.org⟩](http://theia-sfm.org).
- TEERAVECH, K. An introduction to 3d reconstruction using visualsfm and pmvs2/cmvs. In: *Remote Sensing and Geographic Information Systems*. [S.l.: s.n.], 2013. ISBN 978-3-905674-63-7.
- USUMEZBAS, A.; FABBRI, R.; KIMIA, B. B. From multiview image curves to 3D drawings. In: *Proceedings of the European Conference on Computer Vision*. [S.l.]: Springer, 2016. (Lecture Notes in Computer Science).
- VALGMA, L. *3D reconstruction using Kinect v2 camera*. Tese (Doutorado) — Tartu Ülikool, 2016.

WANG, G. et al. Research on features matching method for visual odometry based on depth image. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Sixth International Conference on Electronics and Information Engineering*. [S.I.], 2015. v. 9794, p. 97941U.

WITNESS, G. Best of brazil – readers' tips. 2014. Disponível em: <<https://www.theguardian.com/travel/2014/jun/23/best-of-brazil-readers-tips>>.

WU, C. Towards linear-time incremental structure from motion. In: IEEE. *3DTV-Conference, 2013 International Conference on*. [S.I.], 2013. p. 127–134.

WU, C. et al. Multicore bundle adjustment. In: IEEE. *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. [S.I.], 2011. p. 3057–3064.

WU, C. et al. Visualsfm: A visual structure from motion system. 2011.