



UNIVERSIDADE DO ESTADO DO  
RIO DE JANEIRO



---

INSTITUTO POLITÉCNICO  
GRADUAÇÃO EM ENGENHARIA  
DE COMPUTAÇÃO

Pedro Felipe Pena Barata

Aplicações práticas em técnicas de reconstrução 3D utilizando  
fotogrametria

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

Aplicações práticas em técnicas de reconstrução 3D utilizando fotogrametria

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.

Orientador: Prof. Dr. Ricardo Fabbri

Nova Friburgo

2017

Pedro Felipe Pena Barata

**Aplicações práticas em técnicas de reconstrução 3D utilizando fotogrametria**

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)  
Departamento de Modelagem Computacional – UERJ

---

Prof. Dr. Edirlei Soares – UERJ

---

Prof. Dr. Roberto Pinheiro – UERJ

Nova Friburgo  
2017

## **AGRADECIMENTOS**

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

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

A 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.

*Marthin Luther King*

## RESUMO

BARATA, Pedro Felipe Pena. *Aplicações práticas em técnicas de reconstrução 3D utilizando fotogrametria*. 2017. 76 f. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) – Departamento de Modelagem Computacional, Instituto Politécnico, Universidade do Estado do Rio de Janeiro, Nova Friburgo, 2017.

A área de reconstrução 3D vem sido amplamente explorada. Sensores de alcance, tanto aéreos quanto terrestres, eram empregados em diferentes aplicações. Porém, constantes melhorias na tecnologia, sobretudo, nos *hardwares* e *softwares* no âmbito da reconstrução, fizeram com que hoje, quase duas décadas depois, novas técnicas surgissem.

Muitos cientistas que utilizavam a fotogrametria converteram seus esforços na área dos sensores à laser. Pois além de executarem uma reconstrução mais rápida, possuem uma altíssima acurácia, compensando seu alto custo inicial. Isto dificultou e desacelerou o processo de descoberta de novos algoritmos e métodos na área da fotogrametria.

Hoje em dia, graças à esse avanço, a fotogrametria, aliada a novos algoritmos, como o *Structure from Motion* (SfM), pontos em comum e de combinação de imagens, por exemplo, consegue competir com scanners à laser e sensores de alcance.

Neste trabalho, abordaremos o uso de técnicas combinando SfM e MVS (*Multi-View Stereo*), utilizando *softwares* como o MVE (??) e o VisualSfM (??), é possível gerar uma reconstrução satisfatória apenas utilizando uma câmera de um *smartphone*. Além disso exploraremos um pouco sobre outros tipos de técnicas empregadas em grandes projetos, como o Kinect, da Microsoft e sua aplicação em SfM e o projeto de reconstrução das esculturas de Michelangelo, feito por um grupo da Universidade de Stanford.

Palavras-chave: Fotogrametria. VisualSfM. MVE – *Multi-View Reconstruction Environment*. Reconstrução 3D.

## ABSTRACT

BARATA, Pedro Felipe Pena. *Practical applications in 3D reconstruction techniques using photogrammetry*. 2017. 76 f. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) – Departamento de Modelagem Computacional, Instituto Politécnico, Universidade do Estado do Rio de Janeiro, Nova Friburgo, 2017.

The 3D reconstruction area has been widely explored. Range sensors, both aerial and terrestrial, were used 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 almost two decades later.

Many scientists which used photogrammetry have converted their efforts into the area of laser sensors. For besides performing a faster reconstruction, they have a very high accuracy, compensating the high initial cost. This hindered and slowed down the process of discovering new algorithms and methods in the field of photogrammetry.

Today, thanks to this advancement, photogrammetry, coupled with new algorithms such as Structured Motion (SfM), common points and image combining, for example, can compete with laser scanners and sensors of reach.

In this work, we will cover the use of techniques combining SfM and MVS (*Multi-View Stereo*), using softwares such as MVE (??) and VisualSfM (??), it is possible to generate a satisfactory reconstruction only using a camera from a smartphone. In addition we will explore a bit about other types of techniques employed in major projects such as Microsoft's Kinect and its application in SfM and the project of reconstruction of the sculptures of Michelangelo, made by a group of Stanford University.

Keywords: Photogrammetry. VisualSfM. MVE – Multi-View Reconstruction

Environment. 3D Reconstruction.

## LISTA DE FIGURAS

|   |    |
|---|----|
| Figura 1 - Kinects de primeira geração (a) consistindo de câmeras e projetores infra-vermelho (b) e de segunda geração, consistindo de tecnologia ToF (c). Ambos os kinects são largamente utilizados para escaneamento em tempo real, formando a base de scanners manuais (d), porém nem sempre são úteis para preservação detalhada de patrimônio. Um dos objetivos deste projeto é explorar os limites desta tecnologia. . . . .   | 13 |
| Figura 2 - A reconstrução usando-se Kinect (de primeira ou segunda geração) usando software atual de super-resolução (c) fornece precisão similar a um sistema estéreo de média resolução, inferior um sistema a laser de alta qualidade (d) porém de baixo custo e muito mais versátil devido ao sistema de aquisição manual e a software amplamente utilizado e desenvolvido(??). . . . .   | 14 |
| Figura 3 - Protótipo do scanner a laser de triangulação. O objeto a ser escaneado é uma réplica em tamanho real de um sarcófago egípcio (a). O scanner foi reconfigurado para escanear objetos maiores, pois a escultura possui 517 centímetros (b), o da cabeça também sofreu uma reconfiguração, este scanner gira em 90 graus, que faz o laser rotacionar, da posição horizontal para a vertical e também roda em torno da cabeça como um todo (c). Para a reconstrução, o primeiro passo foi alinhar cerca de 100 scans em diversas posicoes, após isso, utilizado um alinhamento automatico em pares dos scans, utilizando um algoritmo modificado de iteracoes de pontos próximos (ICP - <i>Iterated-Closest-Points</i> ). Após isso, faz-se um processo de relaxação global a fim de minimizar erros de alinhamento por toda a estátua. Depois de alinhados, usa-se o algoritmo de profundidade volumétrica de processamento de imagens (VRIP - <i>Volumetric Range Image Processing</i> - de Brian Curless) (d). (??) . . . . . | 14 |
| Figura 4 - Algumas esculturas do Jardim do Nêgo . . . . .   | 15 |
| Figura 5 - Exemplo de um projetor de padrões utilizando um laser visível. (??). . . . .   | 17 |
| Figura 6 - Como era feito o escaneamento da estátua de David (a), composto por dois scanners, sendo (b) o scanner principal (da cabeça) e estruturas para alcançar todos os pontos necessários. (??). . . . .   | 18 |
| Figura 7 - Escultura "Noite", de Michelangelo. . . . .  | 19 |
| Figura 8 - Alguns tipos de cinzelões que provavelmente foram usados por Michelangelo na escultura de "St. Matthew". (??). . . . .   | 20 |

|  |    |
|--|----|
| Figura 9 - Imagem de um Kinect V1 aberto, constituído de uma câmera infravermelho (IR - <i>Infra-Red</i> ), uma câmera RGB e um projetor IR. (??).   | 24 |
| Figura 10 - Exemplo de como é a saída de uma imagem interpretada pelo Kinect, onde cada cor disposta na imagem, corresponde à profundidade ou distância da cena para o Kinect. (??).   | 24 |
| Figura 11 - Representação visual do acerto do deslocamento. A parte em preto é a imagem IR e o contorno em branco é a imagem de profundidade do alvo. (??)   | 26 |
| Figura 12 - Posição e orientação do Kinect (com as câmeras IR e RGB) e o par estéreo SLR ( <i>Left, Right</i> ) em conjunto com pontos de calibração 3D reconstruídos em alvos de calibração planar. (??).   | 27 |
| Figura 13 - Procedimento padrão da maioria dos sistemas baseados em SfM e MVS (??).  | 29 |
| Figura 14 - É aplicado um filtro gaussiano na imagem original (a), com $\sigma = 1$ , tendo como resultado a imagem (b). Um outro filtro gaussiano é usado, porém, neste caso, o $\sigma = 2$ (c). Após isso, subtrai-se (b) de (c), obtendo o filtro DoG (d).   | 31 |
| Figura 15 - Exemplo de funcionamento de detecção de espaço-escala extrema  | 31 |
| Figura 16 - Exemplo do resultado obtido do histograma orientado  | 33 |
| Figura 17 - Exemplo de um descritor de pontos-chaves, com uma matriz 2x2 e uma região 8x8  | 34 |
| Figura 18 - Uma triangulação utilizando um ponto qualquer, $X_j$ . Onde cada câmera $C_1, C_2, C_3$ possui um <i>feature</i> correspondente a cada uma delas, respectivamente, $X_{1j}, X_{2j}, X_{3j}$ .  | 35 |
| Figura 19 - Funcionamento do MVE. 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 – <i>Floating Scale Surface Reconstruction</i> ). (??) | 36 |

|   |    |
|---|----|
| Figura 20 - Caso o espaçamento entre as câmeras seja grande, a informação extraída das imagens em comum será menor (a). Se a angulação do efeito de paralaxe seja baixa, terá a mesma informação sobre um ponto em questão (c). Ou seja utilizando ou (a), ou (c). Pode ser que a reconstrução fique incerta. Para que o efeito paralaxe tenha maior proveito das imagens das câmeras, é necessário que as câmeras estejam dispostas como (b), conseguindo extrair uma boa quantidade e qualidade de informações do ponto. (??) . . . . . | 37 |
| Figura 21 - Interface gráfica (UMVE) . . . . .  | 38 |
| Figura 22 - Reconstrução baseada em nuvem de pontos gerada a partir da reconstrução SfM. . . . .  | 39 |
| Figura 23 - Uma imagem de entrada (à esquerda) e sua correspondência em mapas de profundidade (à direita), onde a parte roxa significa que não foi encontrada nenhum mapa naquela região (??). . . . .  | 39 |
| Figura 24 - Botões na parte superior da interface gráfica, este seria o procedimento padrão de funcionamento do <i>software</i> . (??) . . . . .  | 41 |
| Figura 25 - 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. (??) . . . . .   | 42 |
| Figura 26 - Usabilidade dos filtros de qualidade e visibilidade, onde são aplicados fora do núcleo, assim como em paralelo. À esquerda, um ponto MVS $P$ é testado usando os filtros. À direita, temos pseudo-códigos, onde os <i>loops</i> destacados em azul podem ser executados em paralelo. (??). . . . .  | 49 |
| Figura 27 - Botões na parte superior da interface gráfica, este seria o procedimento padrão de funcionamento do <i>software</i> . (??) . . . . .  | 51 |
| Figura 28 - 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. (??) . . . . .   | 53 |
| Figura 29 - Usabilidade dos filtros de qualidade e visibilidade, onde são aplicados fora do núcleo, assim como em paralelo. À esquerda, um ponto MVS $P$ é testado usando os filtros. À direita, temos pseudo-códigos, onde os <i>loops</i> destacados em azul podem ser executados em paralelo. (??). . . . .  | 60 |
| Figura 30 - Exemplo de como foi realizada a varredura da escultura . . . . .  | 62 |
| Figura 31 - Reconstrução esparsa da escultura do Jardim do Nêgo no VisualSfM com 197 imagens. . . . .   | 63 |
| Figura 32 - Resultados da reconstrução densa da escultura do Jardim do Nêgo usando o VisualSfM, em dois ângulos diferentes (a) e (b). . . . .   | 64 |

|  |    |
|--|----|
| Figura 33 - Reconstrução esparsa do objeto no VisualSfM com 200 imagens.   | 64 |
| Figura 34 - Reconstrução densa do objeto no VisualSfM com 200 imagens.   | 64 |
| Figura 35 - Reconstrução esparsa do objeto com 224 imagens no VisualSfM.   | 65 |
| Figura 36 - Foram gerados dois modelos esparsos do objeto a partir do conjunto inicial de 224 imagens, provavelmente, proveniente da falta de parâmetros da câmera.  | 65 |
| Figura 37 - Reconstruções densas do primeiro (a) e do segundo (b) modelo do objeto no VisualSfM com 224 imagens.   | 66 |
| Figura 38 - A figura (a) é um exemplo onde a imagem possui dados na extensão <i>EXIF</i> (destacado em azul). Ao passo que a figura (b) é um frame de um vídeo, que não possui os dados das câmeras (destacado em azul).   | 66 |
| Figura 39 - Final da reconstrução via UMVE, percebe-se que alguns pontos não foram considerados, tendo como resultado uma "nuvem de pontos" mais densa, basicamente.   | 67 |
| Figura 40 - Processos dentro do comando <i>sfmrecon</i> , onde (a) estão sendo detectadas as <i>features</i> do conjunto de imagens. Em (b) está computado o <i>pairwise matching</i> e em (c) está no processo de <i>Bundle Adjustment</i> (??), usando condições-padrão para as câmeras. | 68 |
| Figura 41 - Término do comando <i>sfmrecon</i> , onde demorou cerca de 1 minuto e meio (75509 milisegundos).   | 68 |
| Figura 42 - Término do comando <i>dmrecon</i> , onde demorou cerca de 4 horas (14502576 milisegundos).   | 69 |
| Figura 43 - Execução dos comandos <i>scene2pset</i> , nos níveis -F0, -F1, -F2 e -F3.  | 69 |
| Figura 44 - Progressão do comando <i>fssrecon</i> , onde possui o ETA – <i>Estimated Time of Arrival</i> .   | 70 |
| Figura 45 - Malha com ruídos proveniente do comando <i>fssrecon</i> .  | 70 |
| Figura 46 - Resultado final, após a remoção dos ruídos da malha.   | 71 |
| Figura 47 - Tempo gasto da etapa <i>sfmrecon</i> do MVE  | 72 |
| Figura 48 - Tempo da etapa <i>dmrecon</i> do MVE   | 72 |
| Figura 49 - Tempo da etapa <i>fssrecon</i> do MVE  | 72 |
| Figura 50 - Resultado da etapa <i>fssrecon</i> do MVE  | 73 |
| Figura 51 - Resultado da etapa <i>meshclean</i> , da etapa anterior 50   | 73 |
| Figura 52 - Resultado da etapa <i>sfmrecon</i> , com todas as imagens  | 74 |
| Figura 53 - Resultado da etapa <i>dmrecon</i> , com todas as imagens   | 74 |

## SUMÁRIO

|       |   |    |
|-------|---|----|
| 1     | <b>INTRODUÇÃO</b>   | 12 |
| 1.1   | Objetivos   | 16 |
| 2     | <b>RECONSTRUÇÃO À LASER</b>                                       | 17 |
| 2.1   | Projeto de preservação das esculturas de Michelangelo             | 17 |
| 3     | <b>TÉCNICAS DE RECONSTRUÇÃO BASEADAS EM <i>TIME OF FLIGHT</i></b> | 23 |
| 3.1   | Kinect  | 23 |
| 4     | <b>TÉCNICAS DE RECONSTRUÇÃO BASEADAS EM FOTOGRAFOMETRIA</b>       | 29 |
| 4.1   | MVE – <i>Multi-View Reconstruction Environment</i>                | 35 |
| 4.2   | VisualSfM   | 40 |
| 4.3   | VisualSfM   | 51 |
| 5     | <b>EXPERIMENTOS</b>   | 62 |
| 5.1   | <b>Procedimento</b>   | 62 |
| 5.1.1 | <u>Resultados da reconstrução com o VisualSfM</u>                 | 63 |
| 5.1.2 | <u>Resultados da reconstrução com o MVE</u>                       | 66 |
| 6     | <b>CONCLUSÃO</b>  | 75 |
|       | <b>REFERÊNCIAS</b>  | 76 |

# 1 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 (??), filmes (??), arqueologia, arquitetura, modelagem 3D urbana (*e.g.*, Google Streetview); técnicas de *match-moving* em cinematografia para fusão de conteúdo virtual e filmagem real (??), a organização de uma coleção de fotografias com relação a uma cena (*e.g.*, o sistema *Phototourism* (??) e a funcionalidade *Look Around* do Google Panoramio e Street 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 (??), a partir do desenvolvimento da detecção robusta de *features* (??), o *fitting* 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 (??) liberado por Noah Snavely, e sua subsequente incorporação ao sistema VisualSfM (??), é possível utilizar este sistema para a reconstrução de patrimônio.

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 33,33,36, 22. As câmeras podem então ser utilizadas para obter modelos mais detalhados de reconstrução, como algoritmos de densificação (??) e interpolação (??) da nuvem de pontos, bem como demais algoritmos densos de visão estéreo multi-perspectiva/multi-ocular, como os do grupo de Michel Goesele (??), também com código disponível. Tais algoritmos, no entanto, tem problemas, em particular a reconstrução suaviza partes bem-delineadas do objeto, e pode conter buracos em áreas ho-

mogêneas. Pode-se, portanto, utilizar a reconstrução 3D de curvas do pesquisadorponente (?????????) para auxiliar na reconstrução mais bem-delinada nesses casosproblemáticos, bem como para ajudar no problema de escalabilidade quando a reconstrução3D se torna muito grande. Um segundo paradigma, denominado **reconstrução estéreo multiocular ativa**, tem se tornado viável devido à indústria de videogames, e consiste nautilização de sistemas que alteram o funcionamento de câmeras convencionais, típicamenteusando-se projetores infra-vermelho, laser ou câmeras ToF (*Time of Flight*), como no caso dosdispositivos Kinect, figura 9.



Figura 1 - Kinects de primeira geração (a) consistindo de câmeras e projetores infra-vermelho (b) e de segunda geração, consistindo de tecnologia ToF (c). Ambos os kinects são largamente utilizados para escaneamento em tempo real, formando a base de scanners manuais (d), porém nem sempre são úteis para preservação detalhada de patrimônio. Um dos objetivos deste projeto é explorar os limites desta tecnologia.

A preservação de patrimônio tem sido realizada tradicionalmente com scanners dedicados de alto custo, como no projeto David 3. O projeto teve início em 1992 e tem como objetivo a utilização de scanners a laser de profundidade (*Rangefinder Scanners*), aliado com 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 (porém, 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 scanner de menor escala que faz uma pequena triangulação com laser de profundidade.

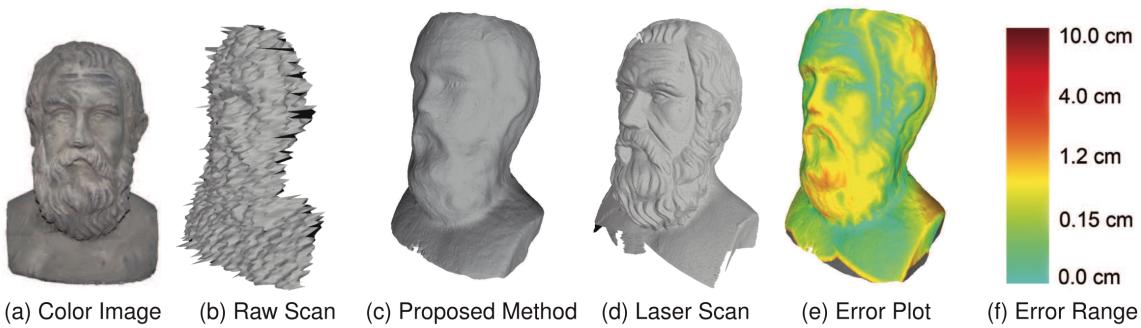


Figura 2 - A reconstrução usando-se Kinect (de primeira ou segunda geração) usando software atual de super-resolução (c) fornece precisão similar a um sistema estéreo de média resolução, inferior a um sistema a laser de alta qualidade (d) porém de baixo custo e muito mais versátil devido ao sistema de aquisição manual e a software amplamente utilizado e desenvolvido(??).

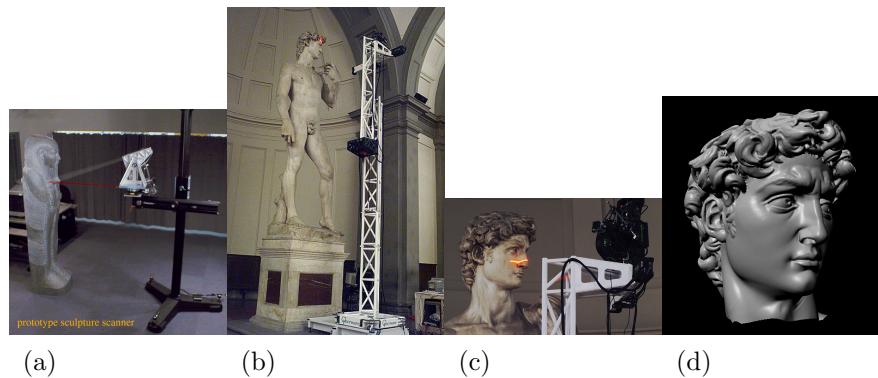


Figura 3 - Protótipo do scanner a laser de triangulação. O objeto a ser escaneado é uma réplica em tamanho real de um sarcófago egípcio (a). O scanner foi reconfigurado para escanear objetos maiores, pois a escultura possui 517 centímetros (b), o da cabeça também sofreu uma reconfiguração, este scanner gira em 90 graus, que faz o laser rotacionar, da posição horizontal para a vertical e também roda em torno da cabeça como um todo (c). Para a reconstrução, o primeiro passo foi alinhar cerca de 100 scans em diversas posicoes, após isso, utilizado um alinhamento automatico em pares dos scans, utilizando um algoritmo modificado de iteracoes de pontos próximos (ICP - *Iterated-Closest-Points*). Após isso, faz-se um processo de relaxação global a fim de minimizar erros de alinhamento por toda a estátua. Depois de alinhados, usa-se o algoritmo de profundidade volumétrica de processamento de imagens (VRIP - *Volumetric Range Image Processing* - de Brian Curless) (d). (??)

O Jardim do Nêgo, Nova Friburgo

No caso de Nova Friburgo, há a necessidade redobrada de preservação do patrimônio, 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 (??), figura 4.



Figura 4 - Algumas esculturas do Jardim do Nêgo

Idealizado e criado por Geraldo Simplicio (Nêgo), artista cearense que mora no local a mais de 30 anos, ganhou notoriedade por suas esculturas de barro, com traços singulares e técnicas únicas. Hoje, trabalha para reconstruir o Jardim da tragédia de 2011 na região serrana, 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 eternizar um patrimônio que é reconhecido no mundo todo.

A preservação das esculturas do Jardim do Nêgo se torna 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, quase sem nenhum padrão de textura/musgo. Seria de grande interesse availiar o potencial de técnicas atuais de reconstrução 3D geral sem controle de aquisição, as quais têm seu código fonte disponível na internet.

## 1.1 Objetivos

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

O objetivo concreto é explorar as tecnologias supracitadas para desenvolver um esquema de escaneamento usando software aberto, câmeras e scanners 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 scanners 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: previamente, introduzimos métodos baseados em reconstrução à laser, no Capítulo 2, apresentando, de uma maneira mais técnica, o projeto das esculturas de Michelangelo na Seção 2.1, que foi um dos pioneiros na técnica de conservação de acervos culturais. No próximo Capítulo 3, discutimos o uso do Kinect, da Microsoft, no que tange a calibração do sistema para o uso em tecnologias Structure from Motion. Adiante, no Capítulo 4, abordaremos o tema central do trabalho: técnicas de reconstrução baseadas em fotogrametria, com o emprego de dois softwares, o MVE 4.1 e o VisualSfM 4.3 e seus respectivos funcionamentos. Finalmente, apresentamos experimentos e conclusões do trabalho, bem como sugestões para implementações e trabalhos futuros.

## 2 RECONSTRUÇÃO À LASER

### Introdução

A técnica de reconstrução baseada em laser é conhecida desde o século passado, pois oferecem uma alta qualidade geométrica de dados, os resultados são em tempo real e requer pouco tempo de captura de dados. Neste caso, abordaremos o projeto de escaneamento da escultura de Michelangelo, David, que utiliza scanners baseados em superfícies, mais especificamente, utilizando uma técnica conhecida como *Time of Flight*, ou tempo de voo 5. Este processo se baseia em projetar um padrão conhecido (muitas vezes, grades ou barras horizontais, via laser) 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.

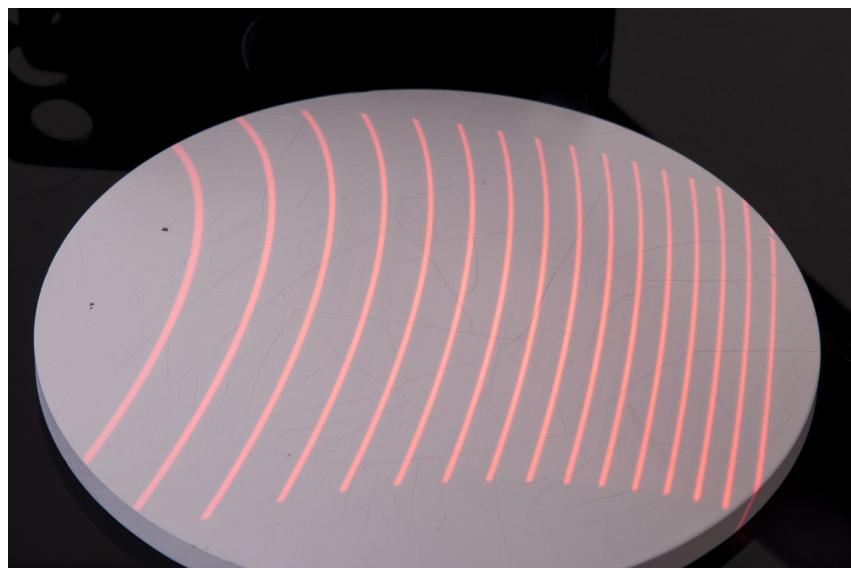


Figura 5 - Exemplo de um projetor de padrões utilizando um laser visível. (??).

### 2.1 Projeto de preservação das esculturas de Michelangelo

#### Introdução

Este projeto tem como motivação avançar a tecnologia de digitalização 3D e criar um acervo digital sobre alguns dos principais artefatos culturais. Foram utilizados sensores de alcance (*range finders*) para triangulação, sensores de alcance baseados em ToF (*Time of Flight*), câmeras digitais e um software de calibração. Com uma equipe de mais

de 30 professores, funcionários e estudantes da Universidade de Stanford desenvolveram algoritmos para combinar imagens de múltiplas gamas e cores, passaram os anos de 1998 e 1999 na Itália escaneando esculturas de Michelangelo.

O principal componente de hardware do sistema é um scanner de triangulação à laser, que é composto por 4 eixos motorizados, um laser, uma câmera de alcance, uma câmera de cores e uma luz branca [6](#), isso tudo acrescido de suportes e estruturas para o escaneamento de estátuas grandes. O objetivo de um scanner deste porte era capturar marcas menores que um milímetro, das ferramentas utilizadas por Michelangelo em suas esculturas. Para isso foram testadas diversas resoluções, na qual foi 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. O que 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 cinzelão 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 Statuario, uma pedra altamente uniforme, não direcional e constituída de grãos finos. Além disso, com exceção da "Noite" [8](#), as esculturas não são polidas e cobertas por terra, o que aumenta a dispersão superficial e reduz a abaixo da superfície. Neste contexto, a dispersão abaixo da superfície causa alguns problemas: não se pode assumir que a superfície é Lambertiana ideal ([??](#)), mudou a forma com que a renderização dos modelos seriam feitos e diminuiu a qualidade de disposição de dados.

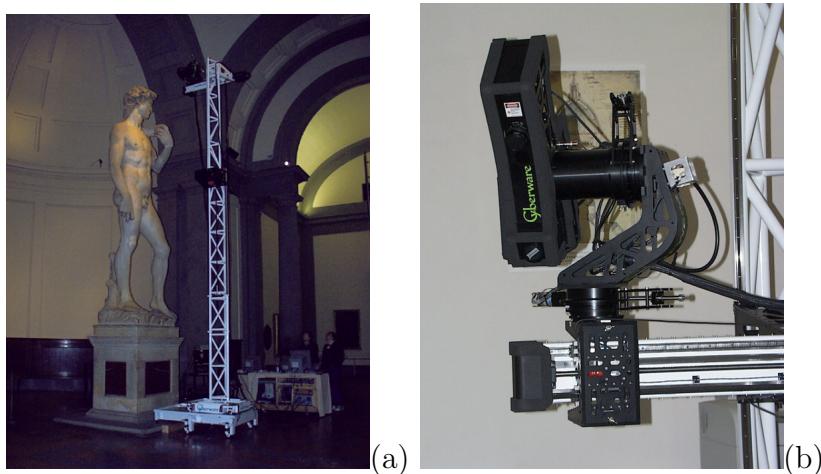


Figura 6 - Como era feito o escaneamento da estátua de David (a), composto por dois scanners, sendo (b) o scanner principal (da cabeça) e estruturas para alcançar todos os pontos necessários. ([??](#)).

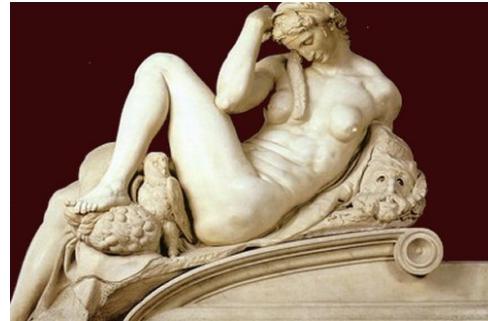


Figura 7 - Escultura "Noite", de Michelangelo.

## 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 não foi rastreado o aparato, ele tornou-se a própria referência. O mapeamento final foi realizado alinhando novas varreduras, com varreduras previamente existentes.

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, também é reduzido o volume físico total na qual essas medidas de calibração devem ser tomadas. Além disso, uma calibração em etapas é menos suscetível a 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 divida em seis etapas distintas:

1. Um mapeamento 2D a partir de coordenadas de *pixels* das imagens da câmera para locais físicos na camada de laser
2. Transformação rígida 2D/3D do sistema de coordenadas da camada do laser para esferas de aço anexadas ao scanner da cabeça
3. Transformação rígida 3D para acomodar o rolamento do scanner da cabeça em 90° (ao remontá-lo) em relação ao conjunto de inclinação

4. A localização do eixo de rotação de inclinação e o mapeamento não-linear de movimento comandos para ângulos de rotação físicas
5. A localização do eixo de rotação panorâmico e o mapeamento de seus comandos de movimento para ângulos de rotação física
6. A localização do eixo de translação, que também depende de como o conjunto inclinação-panorâmico está montado no braço horizontal

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

$$\begin{bmatrix} \text{translação} \\ \text{horizontal} \end{bmatrix} \begin{bmatrix} \text{rotação} \\ \text{panorâmica} \end{bmatrix} \begin{bmatrix} \text{rotação} \\ \text{de} \\ \text{inclinação} \end{bmatrix} \begin{bmatrix} \text{rotação} \\ \text{de} \\ \text{rolamento} \end{bmatrix} \begin{bmatrix} \text{laser para o} \\ \text{scanner da cabeça} \end{bmatrix} \begin{bmatrix} \text{imagem} \\ \text{para o} \\ \text{laser} \end{bmatrix}$$

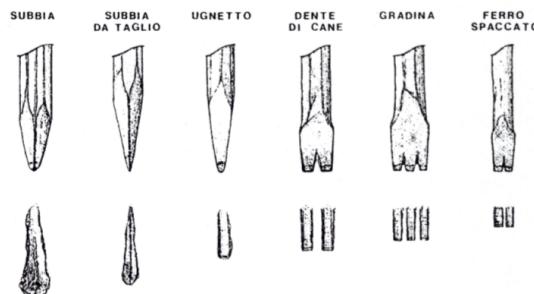


Figura 8 - Alguns tipos de cinzelões que provavelmente foram usados por Michelangelo na escultura de "St. Matthew". (??).

### Procedimento de digitalização

Um operador move, interativamente o scanner da cabeça através de uma sequência de movimentos, definindo os limites de 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 scanner
- 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 alcance (laser) e do escaneamento colorido (câmera digital).

Primeiramente faz-se o escaneamento da geometria da escultura:

- Alinhamento manual
- ICP – *Iterative Closest Point* para uma câmera existente
- ICP automático para todos os pares sobrepostos
- Relaxação global para espalhar o erro
- Reunir utilizando métodos volumétricos

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

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

Limitações:

- Inter-reflexões são ignoradas
- Dispersões subterrâneas são ignoradas
- Tratamento difuso com Lambertiano
- Usa superfícies normais agregadas

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). Um modelo também existe para o Atlas em 0,25 mm (aproximadamente 500 milhões de triângulos), mas contém erros de alinhamento. Após 6 anos de trabalho estudantil remunerado e voluntário, existem modelos para cada um dos 1.186 fragmentos. Esses modelos, que totalizam quase 8 bilhões de polígonos, se encontram no próprio site da Universidade de Stanford

Também foram disponibilizadas algumas métricas sobre este projeto [1](#).

Porém, devido ao seu alto custo com equipamentos, com softwares e sem falar 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, que será abordada ao decorrer deste documento.

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 scanner                       | 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 subcontratantes e colaboradores)       |
| Tempo médio para escaneamento                  | 1 semana (exceto o 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 (até agora)                    |
| Custo do projeto                               | \$2.000.000  |

### 3 TÉCNICAS DE RECONSTRUÇÃO BASEADAS EM *TIME OF FLIGHT*

#### Introdução

*Time of Flight* (??) ou Tempo de Vôo é um **método ótico ativo**, que consegue determinar a localização de objetos no alcance do sensor de infra-vermelho (no caso do Kinect, um sensor ToF invisível), a partir de alterações na matriz do projetor de feixes de infra-vermelho (*e.g.*, Kinect versão 1 (??)). Existe também, um outro método que utiliza fótons, que viajam na velocidade da luz. Onde o valor resultante cálculo da diferença da emissão e da chegada dos fótons aos detectores é utilizado na criação de uma probabilidade de distribuição localizando o evento detectado a uma distância do eixo do scanner (*e.g.*, Kinect versão 2 (????))).

#### 3.1 Kinect

##### Introdução

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

O Kinect V2 é composto de uma câmera RGB-D (*Red, Green, Blue* e *Depth*) e utiliza uma projeção de fótons, é mais robusto que seu antecessor, para ambientes fechados e para fins de escaneamento de formas humanas (como esqueleto, músculos, batimento cardíaco, por exemplo). Devido à técnica empregada para reconhecimento 3D (*Time Of Flight* com fótons), ele é muito sensível às texturas presentes no objeto. Ou seja, para esculturas com diferentes superfícies, diferentes refletâncias, lambertianos ou especulares, por exemplo, podem acarretar em problemas nas reconstruções. Além disso, existem outros obstáculos, como a dificuldade onde o fóton emitido pelo emissor rebate em várias superfícies antes de ser detectado pelo sensor de infra-vermelho. E, portanto, para o uso em áreas externas, a primeira versão se sai melhor.

O Kinect versão 1 é composto por 2 câmeras: uma RGB e outra de profundidade e por um projetor IR (*Infra-Red*) de padrões. E funciona da seguinte maneira: o projetor IR de padrões lança uma matriz que já é previamente conhecida pelo sistema do Kinect, a

partir disso, qualquer deformação deste padrão é captada pelas câmeras, o que identifica se um objeto está no alcance dos sensores ou não. A resposta, é composta por 3 *outputs*: uma imagem IR, uma RGB e a profundidade (inversa) da imagem.

A partir deste momento, referenciamos o Kinect versão 1 apenas como Kinect.



Figura 9 - Imagem de um Kinect V1 aberto, constituído de uma câmera infra-vermelho (IR - *Infra-Red*), uma câmera RGB e um projetor IR. (??).

### Processo de calibração

Sua principal saída da imagem do Kinect é correspondente à profundidade da cena. Em vez de providenciar uma profundidade  $Z$ , ele retorna uma profundidade inversa,  $D$ . A profundidade da imagem é construída a partir da triangulação da imagem IR com o projetor e, consequentemente, ”carregada” pela imagem IR.



Figura 10 - Exemplo de como é a saída de uma imagem interpretada pelo Kinect, onde cada cor disposta na imagem, corresponde à profundidade ou distância da cena para o Kinect. (??).

Foram realizados alguns experimentos associando o uso do Kinect V1 como um scanner em reconstruções. Primeiramente, foi executado uma calibração do Kinect para

este tipo de reconstrução, onde a partir de experimentos, o sistema foi modelado como 1.

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

Onde "z" é a profundidade em metros, e "q" a quantização, em milímetros.

O modelo geométrico do Kinect foi criado com um sistema *multi-view* considerando o RGB, IR e a profundidade.

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

$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = \underbrace{(1 + k_1 r^2 + k_2 r^4 + k_5 r^6)}_{\text{distorção radial}} \begin{bmatrix} p \\ q \\ 0 \end{bmatrix} + \underbrace{\begin{bmatrix} 2k_3 pq + k_4(r^2 + 2p^2) \\ 2k_4 pq + k_3(r^2 + 2q^2) \\ 1 \end{bmatrix}}_{\text{distorção tangencial}} \quad (3)$$

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

Onde  $k_n$  é o parâmetro de distorção, matriz de calibração da câmera  $K$ , rotação  $R$  e centro  $C$ .

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] de 5 e produz uma imagem RGB de 6.

Em 5,  $dis$  é a distorção, proveniente de 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. 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. 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   |

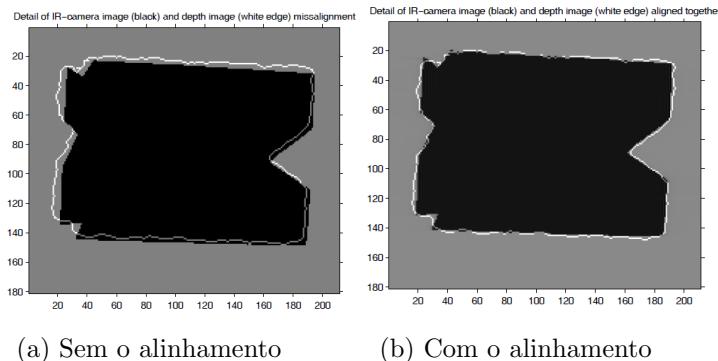


Figura 11 - Representação visual do acerto do deslocamento. A parte em preto é a imagem IR e o contorno em branco é a imagem de profundidade do alvo. (??)

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 coordenada  $Z_{IR}$  de 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 diferentes, no mesmo ambiente do Kinect.

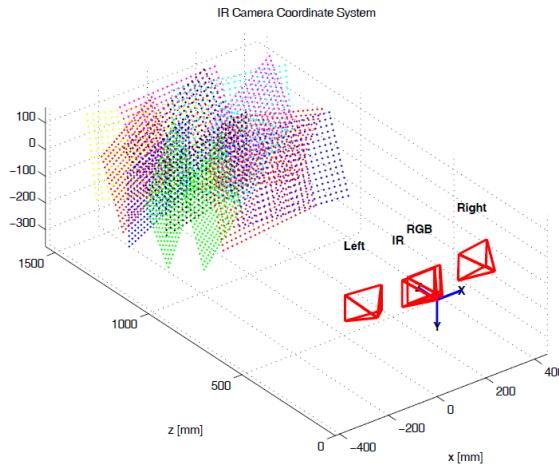


Figura 12 - Posição e orientação do Kinect (com as câmeras IR e RGB) e o par estéreo SLR (*Left*, *Right*) em conjunto com pontos de calibração 3D reconstruídos em alvos de calibração planar. (??).

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    |

### Uso do Kinect com *Structure from Motion*

Com o sistema calibrado, 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 *Multi-View Stereo*.

O Kinect tem a capacidade e, com o procedimento de calibração, é possível combiná-lo com técnicas *SfM* e *multi-view stereo*, o que abre uma nova aplicação na área de reconstrução 3D.

Quanto a qualidade da reconstrução de multi-view, o Kinect ficou melhor que o SR-4000 e perto do 3.5M SLR Stereo 3.

O Kinect é tão utilizado para fins pessoais (e empresariais), que existem alguns softwares para utilizá-lo como uma ferramenta de reconstrução 3D e a maioria deles são bem acessíveis (incluindo um pacote de ferramentas que a própria Microsoft disponibilizou, para o funcionamento do Kinect no Windows). Uma delas é o *Skanect* que tem a versão gratuita onde é possível fazer escaneamentos básicos e a versão paga, que possibilita uma

configuração maior, como por exemplo, a delimitação do objeto que será reconstruído, exportar o arquivo em diferentes formatos (*.PLY*; *.OBJ*: formato para exportação para programas que melhoram o modelo gerado (blender ou sculptris, por exemplo). E escolher o numero de faces a ser exportado; *STL*: próprio para a impressora 3D (software *Cura*); *VRML*: salva 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, com detecção de formas humanas e movimentações. Ou seja, numa aplicação *in situ* já não funcionaria muito bem, pois além de não conseguir projetar os detalhes em alta definição de uma escultura, ele necessita de uma fonte de energia externa, o que dificulta a acessibilidade, e, 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.

## 4 TÉCNICAS DE RECONSTRUÇÃO BASEADAS EM FOTOGRAMETRIA

### Introdução

A reconstrução 3D pelo método da estrutura do 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 usadas na reconstrução. Para encontrar estes pontos, diferentes algoritmos são empregados.

A maioria dos métodos baseados em SfM e MVS (*Multi-View Stereo*) tem uma abordagem similar e funcionam com os seguintes passos 13:

- Obtenção de imagens
- Processamento dos parâmetros de câmera para cada imagem
- Reconstrução da geometria 3D de uma cena com um conjunto de imagens e seus parâmetros correspondentes

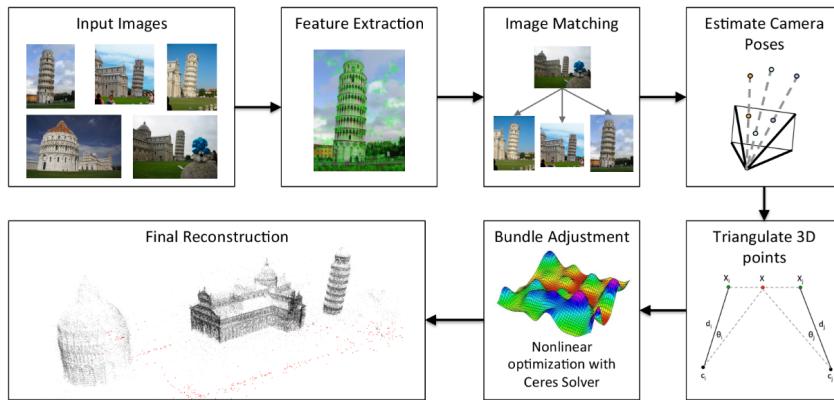


Figura 13 - Procedimento padrão da maioria dos sistemas baseados em SfM e MVS (??).

Neste trabalho, abordaremos o uso de dois softwares baseados em MVS: o MVE (??) e o VisualSfM (??). Ambos utilizam 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 mapas de profundidade enquanto o VisualSfM utiliza um sistema baseado em *Bundle Adjustment*.

O passo de obtenção de imagens, será discutido no Capítulo de experimentos 5. Seguimos adiante com os algoritmos para processamento das imagens de entrada.

## SIFT – *Scale Invariant Feature Transform*

Primeiramente, utiliza-se o SIFT (algoritmo de detecção de pontos de interesse, invariante à escala e à transformações, como rotação, translação e iluminação da imagem, por exemplo). O algoritmo pode ser dividido em cinco etapas, das quais:

- Detecção de espaço-escala extremos – *Scale-space Extrema Detection*
- Localização de pontos-chaves – *Keypoint Localization*
- Atribuição de orientação – *Orientation Assignment*
- Descritor de pontos-chaves – *Keypoint Descriptor*
- Combinação de pontos-chaves – *Keypoint Matching*

### Detecção de espaço-escala extremos

Em casos com cantos pequenos, a detecção funciona bem. Porém, raramente utilizaremos a mesma janela para detectar pontos-chaves em imagens com diferentes escalas, pois utilizamos imagens grandes e, consequentemente, cantos grandes. Para isso, precisamos de janelas grandes também.

Para resolver este problema, o filtro de escala-espacó é usado: o Laplaciano de Gaussiano (*Laplacian of Gaussian* – LoG). O LoG atua como um detector de partículas em diferentes tamanhos  $\sigma$ . (Onde  $\sigma$  é o parâmetro de escala). Por exemplo, o núcleo Gaussiano com  $\sigma$  baixo, tem como resposta um alto valor para um canto pequeno. Enquanto um núcleo gaussiano com alto  $\sigma$ , 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  $\sigma$ .

Porém, 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 Gaussiano de uma imagem, com dois valores diferentes de escala  $\sigma$ .

Uma aplicação prática do filtro DoG é a sequência de imagens a seguir:

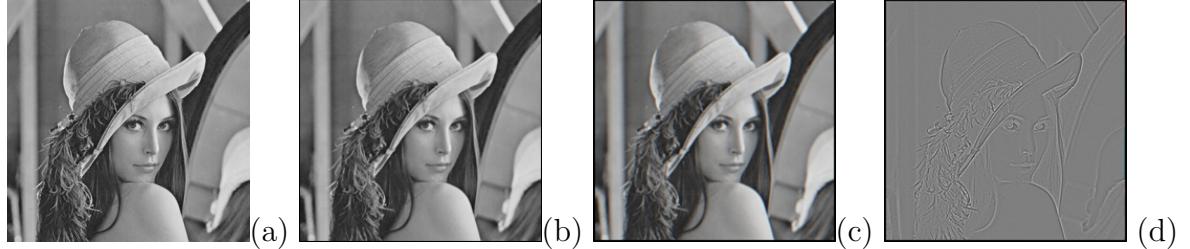


Figura 14 - É aplicado um filtro gaussiano na imagem original (a), com  $\sigma = 1$ , tendo como resultado a imagem (b). Um outro filtro gaussiano é usado, porém, neste caso, o  $\sigma = 2$  (c). Após isso, subtrai-se (b) de (c), obtendo o filtro DoG (d).

Uma vez que o DoG é aplicado, as imagens são utilizadas com o espaço e escala extremos. Por exemplo, na imagem 15, 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. Isto é, este ponto-chave é melhor representado nesta escala.

No caso, as funções ficariam da seguinte forma:

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

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

$$\nabla^2 g_\sigma(x) \quad (9)$$

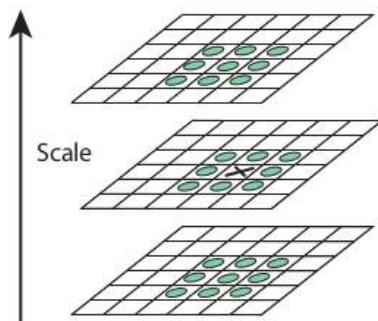


Figura 15 - Exemplo de funcionamento de detecção de espaço-escala extrema

$$DoG_\sigma(o, s) = I_\sigma(o, s + 1) - I_\sigma(o, s) \quad (10)$$

Onde 7 é a função padrão do operador gaussiano (núcleo), a equação 9 é o operador LoG, 10 é o operador DoG e  $\nabla^2$  é o operador Laplaciano.

### Localização de pontos-chaves

A localização dos pontos extremos pode cair em um extremo local e não global. Logo, após a utilização do DoG e com os pontos-chaves em potencial localizados, eles precisam ser refinados para melhorar o resultado. Para isso, são utilizadas Séries de Taylor na escala e no espaço, e, se a intensidade nesse extremo é menor que o valor limite, este é rejeitado. Os *frames* do SIFT (pontos-chaves) são extraídos baseados nos extremos locais (picos) a partir do DoG. Numericamente, extremos locais são elementos que possuem um menor (ou maior) valor em uma vizinhança em um espaço 3x3x3 (em escala e espaço). Depois de extraídos, estes pontos são interpolados quadraticamente (este passo é muito importante, especialmente nas escalas de menor resolução, para ter uma localização precisa do ponto-chave na resolução completa). Finalmente, eles são filtrados para eliminar respostas de baixo contraste ou respostas próximas as bordas.

Picos que são pequenos, na maior parte das vezes são gerados a partir de ruídos e necessitam ser descartados também. Isso é feito com uma comparação de valor absoluto do DoG no pico com o valor do pico limite e é descartado caso este valor é menor que o limite.

Para eliminar respostas em bordas, normalmente os picos mais rasos ou horizontais, são gerados por bordas e não possuem características estáveis, portanto estes picos precisam ser removidos. Para isso, dado um pico  $(x, y, \sigma)$ , o algoritmo avalia a matriz Hessiana  $(x,y)$  do DoG na escala  $\sigma$ . Então é computado um valor para esta equação (11):

$$v = \frac{(T_r D(x, y, \sigma))^2}{\text{Det } D(x, y, \sigma)} \quad (11)$$

Onde,  $T_r$  é o traço, ou seja,  $T_r(H) = D_{xx} + D_{yy}$  e a matriz  $D$  é do tipo

$$D = \begin{bmatrix} \frac{\partial^2 DoG}{\partial x^2} & \frac{\partial^2 DoG}{\partial x \partial y} \\ \frac{\partial^2 DoG}{\partial x \partial y} & \frac{\partial^2 DoG}{\partial y^2} \end{bmatrix}$$

No caso, v possui um valor mínimo (igual a 4) quando os autovalores da Jacobiana são iguais (pico curvado) e aumentam à medida que um dos autovalores aumenta e os

outros permanecem baixos. Os picos são retidos se  $v < \frac{(t_e+1)(t_e+1)}{t_e}$ , onde  $t_e$  é o limite da borda.

### Atribuição de orientação

Agora, uma orientação é atribuída a cada ponto-chave para obter a invariância à rotação da imagem. Uma vizinhança é obtida, dependente da escala, do gradiente da magnitude 12 e da direção 13 (usando diferenças finitas), ao redor da localização do ponto-chave.

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2} \quad (12)$$

$$\theta = \tan^{-1} \left( \frac{(L(x, y + 1) - L(x, y - 1))}{(L(x + 1, y) - L(x - 1, y))} \right) \quad (13)$$

Então, um histograma de 36 orientações (*bins*) cobrindo 360 graus é criado. Onde ele é ponderado pelo gradiente da magnitude e por uma janela Gaussiana circular onde  $\sigma$  vale 1.5 em relação à escala do ponto-chave 16.

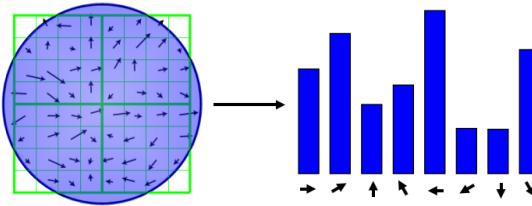


Figura 16 - Exemplo do resultado obtido do histograma orientado

O ponto mais alto do histograma é obtido e qualquer pico acima de 80% é considerado no cálculo da orientação. Pontos-chaves são criados com a mesma localização e escala, mas em diferentes direções, o que contribui para a estabilidade da correspondência.

### Descriptor de pontos-chaves

Com os pontos-chaves criados a partir do histograma orientado, cria-se agora o descriptor de pontos-chaves.

Uma vizinhança 16x16 ao redor do ponto-chave é escolhida e esta mesma vizinhança é dividida em 16 sub-blocos 4x4. Para cada bloco, um histograma orientado com 8 *bin* é

criado. Logo, temos 128 valores válidos de *bin*. Esses valores são representados em forma de vetor para expressar o descritor de pontos-chaves 17.

Além disso, são tomadas algumas medidas para deixar o descritor mais robusto, como, por exemplo, invariante à luminosidade, rotação, etc.

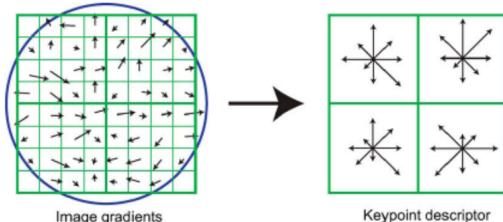


Figura 17 - Exemplo de um descritor de pontos-chaves, com uma matriz 2x2 e uma região 8x8

### Combinação de pontos-chaves

Pontos-chaves entre duas imagens são combinados a partir da identificação da vizinhança mais próxima. Mas, em alguns casos, a segunda combinação mais próxima pode ser parecida com a primeira. Isso se dá por ruídos presentes nas imagens ou algo assim.

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. Esse método elimina cerca de 90% de combinações falsas, enquanto descarta apenas cerca de 5% de combinações corretas.

### Triangulação – *Full pair-wise image matching*

Com os pontos de interesse (*features*) extraídos, podemos agora 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 18.

Infelizmente, não é tão simples assim. Existem muitos fatores que contribuem para aumentar a dificuldade da triangulação: ruídos, posição das câmeras, o feixe das projeções não se encontram no mesmo ponto 3D, não se tem informação da projeções nas 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

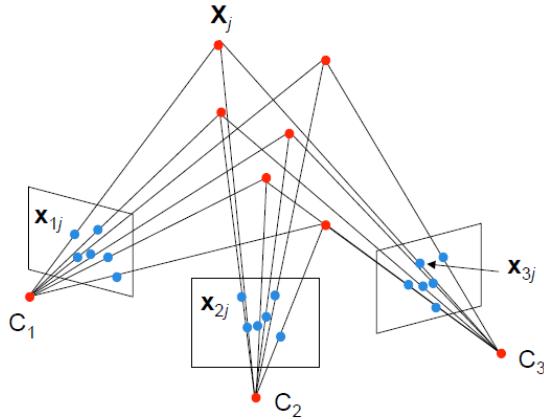


Figura 18 - Uma triangulação utilizando um ponto qualquer,  $X_j$ . Onde cada câmera  $C_1, C_2, C_3$  possui um *feature* correspondente a cada uma delas, respectivamente,  $X_{1j}, X_{2j}, X_{3j}$ .

reconstrução para um dos sistemas se diferem, e a partir de agora, faremos abordagens individuais para cada um deles.

#### 4.1 MVE – *Multi-View Reconstruction Environment*

##### Introdução

Um dos algoritmos utilizados para a técnica de reconstrução densa é o MVE – *Multi-View Reconstruction Environment* (??). 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 (conhecida como UMVE), onde permite a visualização e inspeção das imagens, mapas de profundidade e renderizar cenas e malhas 3D.

Sua base de operação é basicamente 19:

##### 1. Estrutura da formação – *Structure-from-Motion* (SfM)

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

##### 2. Múltiplas visões estéreo – *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.

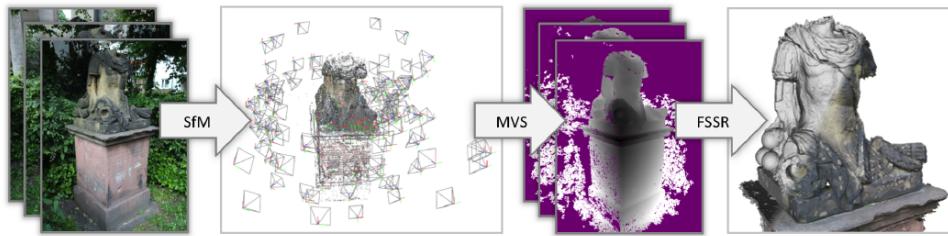


Figura 19 - Funcionamento do MVE. Começando com múltiplas imagens, técnicas SfM são empregadas para reconstruir os parâmetros das câmeras e os conjuntos de pontos esparsos. Mapas de profundidade são computados para cada imagem usando o MVS. Finalmente, uma malha colorida é extraída da união de todos os mapas de profundidade usando um algoritmo de aproximações de reconstruções de superfícies (FSSR – *Floating Scale Surface Reconstruction*). (??)

Como não existem muitas opções para algoritmos de SfM, o MVE permite a utilização de *softwares* externos como o *Bundler* (??) ou o próprio *VisualSfM*.

Uma vez com o passo do 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 no caso, utiliza um algoritmo próprio, feito por um de seus criadores, Michael Goesele (??), que reconstrói um mapa de profundidade para cada foto.

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.

Essa redundância excessiva nos mapas de profundidade pode ser pesado. Não com relação ao armazenamento, mas na questão do processamento computacional exigido nos mapas. Porém, esta abordagem foi capaz de produzir uma geometria detalhada e superar o ruído nos mapas de profundidade individuais.

## Guia de reconstrução com o MVE

Existem algumas recomendações para se ter uma boa reconstrução com o MVE (????). Um bom conjunto de dados é gerado se algumas regras simples forem seguidas:

- Para que o algoritmo do 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 [20](#) (Aparente mudança na posição do objeto). Ou seja, é interessante que o conjunto de imagens seja duplicado.
- A câmera deverá ser reposicionada, de preferência.

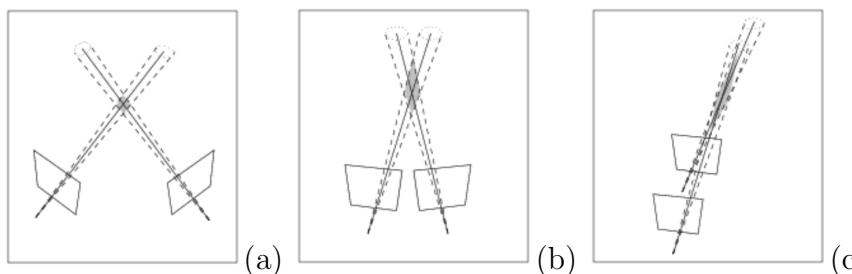


Figura 20 - Caso o espaçamento entre as câmeras seja grande, a informação extraída das imagens em comum será menor (a). Se a angulação do efeito de paralaxe seja baixa, terá a mesma informação sobre um ponto em questão (c). Ou seja utilizando ou (a), ou (c). Pode ser que a reconstrução fique incerta. Para que o efeito paralaxe tenha maior proveito das imagens das câmeras, é necessário que as câmeras estejam dispostas como (b), conseguindo extrair uma boa quantidade e qualidade de informações do ponto. (??)

## Criando uma cena

Uma visualização contém dados por exibição (como imagens, mapas de profundidade ou outros dados). Uma cena é uma coleção de visualizações, que constitui um conjunto de dados. Uma nova cena pode ser criada utilizando a interface gráfica UMVE, 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 visualizações guardadas com uma extensão de arquivos em .MVE.

Criar uma nova cena, criará apenas o diretório (*views*) vazio. A importação de fotos criará arquivos .MVE para cada foto. Esse processo importará meta-dados provenientes das imagens (*tags EXIF*), que é necessário para estimar a distância focal para cada foto. 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, pode vir a acontecer erros no SfM.

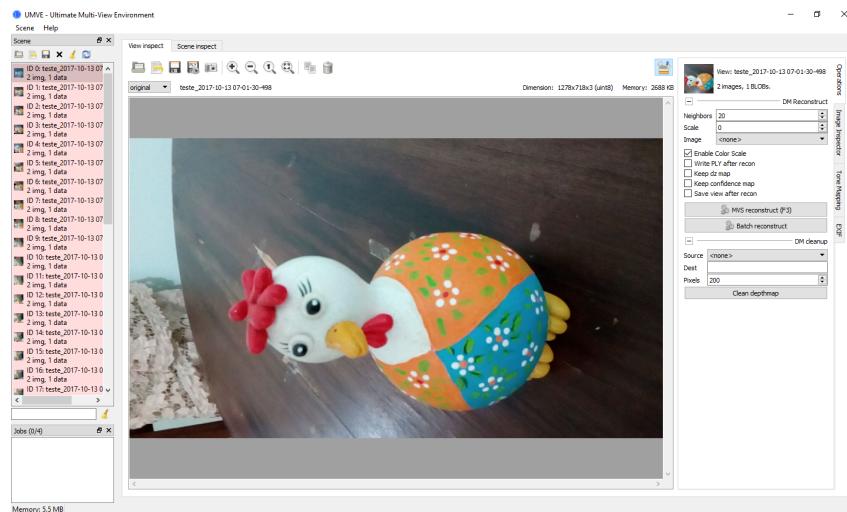


Figura 21 - Interface gráfica (UMVE)

## Reconstrução SfM

Pode ser configurada e iniciada usando a interface gráfica UMVE ou por linha de comando (*sfrmrecon*). A interface guia através da detecção de *features*, combinação emparelhada (*pairwise matching*) e uso incremental do SfM. Que, por sua vez, a reconstrução SfM começa a partir de um par inicial, e adiciona, de forma incremental, mais vistas à reconstrução [22](#).

## Múltiplas visões estéreo – *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 pro-

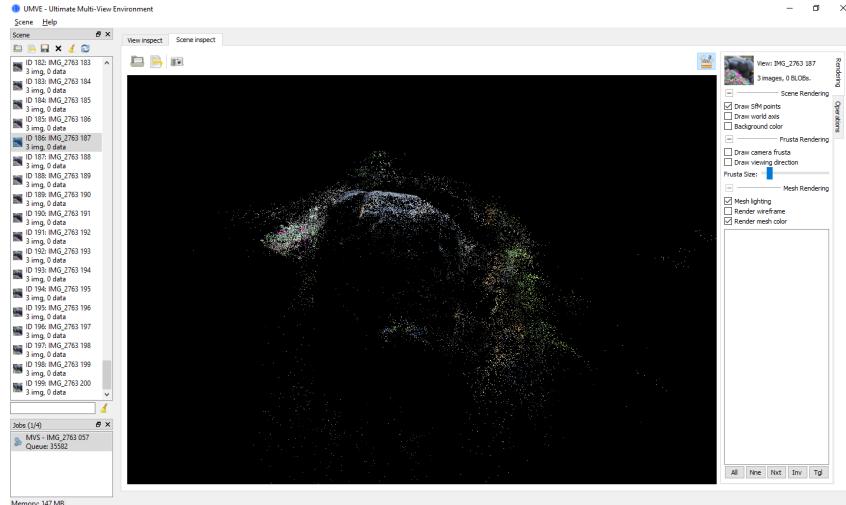


Figura 22 - Reconstrução baseada em nuvem de pontos gerada a partir da reconstrução SfM.

fundidade 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 a metade do tamanho (um quarto dos números de pixels), e assim por diante.

Com a 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 imagens menores [23](#).



Figura 23 - Uma imagem de entrada (à esquerda) e sua correspondência em mapas de profundidade (à direita), onde a parte roxa significa que não foi encontrada nenhum mapa naquela região (??).

### Reconstrução de superfícies – *Surface Reconstruction*

Utiliza-se a 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 usando FSSR (??). 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 à regiões não confiáveis e à componentes isolados, oriundos de medidas imprecisas. Logo, a malha é limpa, retirando pequenos componentes isolados e regiões não confiáveis da superfície.

## 4.2 VisualSfM

### Introdução

VisualSfM (??) é um *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, que é ótima, por sinal. É altamente customizável, podemos utilizar o CUDA da NVIDIA, ou OpenGL, especificar a lista de pares para correspondência de imagens, usar detectores de *features* próprios, velocidade da detecção de *features*, da reconstrução densa, dentre outros parâmetros. Ou seja, é um *software* robusto, que pode ser usado em Linux, Windows ou até mesmo Mac.

Além disso, o VisualSfM é capaz de mostrar a matriz de correspondência de *features*, número de *features*, rodar um *Bundle Adjustment* independente, usar um Level 0 no PVMS, alterar a memória de GPU usada na reconstrução, deletar uma reconstrução indesejável ou até mesmo alterar parâmetros.

### Procedimento

Sua linha de reconstrução é parecida com o MVE 4.1, porém é mais intuitiva. Em sua interface, possui um Log de mensagens e erros que por ventura venham a acontecer e na parte de cima, alguns botões 27

Como demonstrado na imagem 27, o funcionamento seria da seguinte forma:

- **1 - Adicionar algumas imagens.** Este é o primeiro passo, para começar uma reconstrução, primeiro 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.

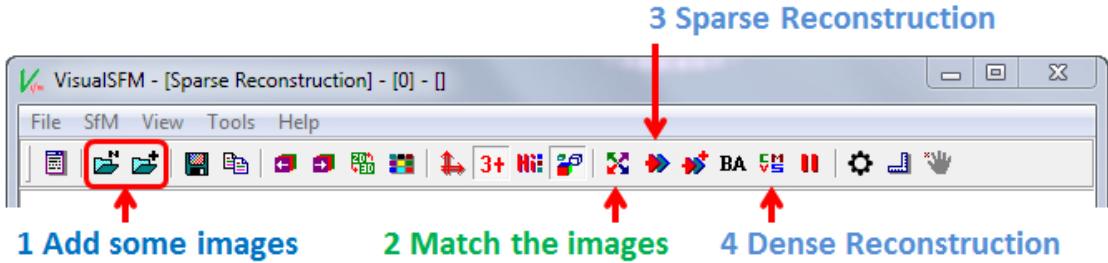


Figura 24 - Botões na parte superior da interface gráfica, este seria o procedimento padrão de funcionamento do *software*. (??)

- **2 - Correspondência de imagens.** Agora, o *software* roda o algoritmo SIFT, realizando todas as correspondências entre os *features*, que já foi discutido anteriormente.
- **3 - Reconstrução esparsa.** Neste passo, o VisualSfM roda o algoritmo de reconstrução esparsa (PBA/MCBA) (??) em todos os *features* descobertos no passo passado.

### *Bundle Adjustment*

O termo *Bundle* refere-se aos feixes de luz que refletem em cada *feature* 3D e vão para o centro de cada câmera da cena a ser reconstruída. Com isso, o *Bundle Adjustment* é o problema de refinar uma reconstrução visual para produzir estimativas com a visualização de parâmetros (estimativa e/ou calibração de câmeras), juntamente com as estruturas 3D.

Solucionadores de problemas de *Bundle Adjustment* (??) resumem-se em minimizar o erro de reprojeção 28 entre as posições de imagem dos pontos de imagem 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 (????), 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 *Bundle Adjustment*, as equações normais tem 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 descrição ?? é uma das abordagens mais comuns sobre o *Bundle Adjustment* e por isso, usaremos ela como exemplo.

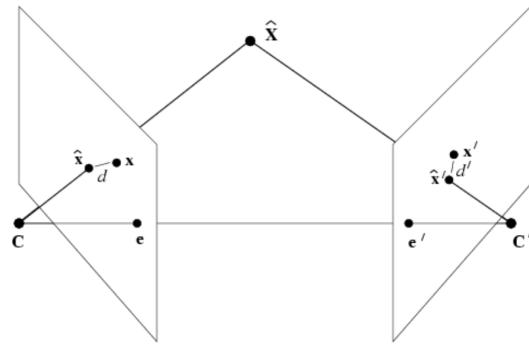


Figura 25 - Dado um ponto de um objeto 3D  $\hat{X}$ , sua reprojeção nas câmeras  $C$  e  $C'$ , são, respectivamente,  $x$  e  $x'$ . Esses pontos possuem um erro de reprojeção  $d$  e  $d'$  e ao utilizar o PBA/MCBA, os pontos  $x$  e  $x'$  serão reajustados como  $\hat{x}$  e  $\hat{x}'$ , respectivamente. (??)

### Formulação inicial

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

### PBA/MCBA – Multi-Core Bundle Adjustment

Neste caso, o PBA/MCBA (????) 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 utilizado na conferência ECCV de 2010. É o primeiro sistema publicado, baseado em GPU que escala com 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 Marquardt, 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 memória de troca 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 e economia de tempo.

Este algoritmo recebe como entrada conjunto de localizações e correspondências de *features* de imagens, o objetivo do *Bundle Adjustment* é encontrar posições de 3 pontos e parâmetros de câmera que minimizem o erro de reprojeção. Esse problema de otimização geralmente é formulado como um problema de mínimos quadrados não-lineares, onde o erro é a norma L2 quadrada da diferença entre a localização da característica observada e a projeção do ponto 3D correspondente no plano da imagem da câmera.

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:

$$x^* = \operatorname{argmin}_x \sum_{i=1}^k \|f_i(x)\|^2 \quad (14)$$

O algoritmo de Levenberg-Marquardt (LM) é o mais conhecido para resolução de problemas de mínimos quadrados não-lineares. O LM 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:

$$\delta^* = \operatorname{argmin}_{\delta} \|J(x)\delta + f(x)\|^2 + \lambda \|D(x)\delta\|^2 \quad (15)$$

e, atualiza  $x$ , da forma:

- 1: caso( $\|f(x + \delta^*)\| < \|f(x)\|$ )
- 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  $\lambda$  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  $\lambda$  a cada passo, baseado no quanto a Jacobiana ( $J(x)$ ) está próxima da função original ( $f(x)$ ). Resolver 20 é equivalente à resolver as equações normais

$$(J^T J + \lambda D^T D)\delta = -J^T f \quad (16)$$

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  $D$ ,  $\delta$  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 21 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*,  $U_{\lambda}$  e  $V_{\lambda}$  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:

$$(U_{\lambda} - WV_{\lambda}^{-1}W^T)\delta_c = -J_c^T f + WV_{\lambda}^{-1}J_p^T f \quad (17)$$

Onde  $S = U_{\lambda} - WV_{\lambda}^{-1}W^T$  é o complemento de Schur ou a matriz reduzida da câmera. Com a solução para 22, conseguimos obter os parâmetros do ponto com substituição regressiva:

$$\delta_p = -V_{\lambda}^{-1}(J_p^T f + W^T \delta_c) \quad (18)$$

Caso S for simétrica positiva-definida, a fatorização de Cholesky é indicado para a resolução de 22.

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.** Finalmente, acaba a reconstrução rodando o algoritmo de reconstrução densa (??) CMVS/PMVS-2 embutido no próprio VisualSfM.

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 (???????) é escalabilidade, pois seu propósito é utilizar imagens provenientes de sites na internet, em diferentes resoluções, como o Flickr.com. Ele usa a saída do Structure from Motion – SfM (mais especificamente a saída do passo anterior, do PBA) e utiliza 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 união das reconstruções de todos os *clusters* não deve perder detalhes que poderiam ser obtidos através do conjunto de imagens.

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 geralmente contêm centenas ou milhares de fotos adquiridas de quase mesmo ponto de vista, ou seja, um conjunto composto inteiramente informações duplicadas.

Em outras palavras, a sobreposição de *clusters* é definida por:

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

O CMVS pode ser resumido em quatro passos:

- 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

A obtenção de medidas precisas de visibilidade de pontos é fundamental para o sucesso do procedimento de visualização baseado em *clusters*. 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). Obtemos estimativas de visibilidade mais confiáveis ao agregar dados da visibilidade em uma vizinhança local, e mesclando os pontos nessa vizinhança. 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 realizada 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 dividindo os *clusters*, ignorando a cobertura. Mais especificamente, um *cluster* de imagens é dividido em componentes menores caso viole a restrição de tamanho. A divisão de um *cluster* é realizada pelo algoritmo *Normalized-Cuts* [23] em um gráfico de visibilidade, onde os nós são imagens. O peso dA borda entre um par de imagens ( $I_l, I_m$ ) mede o quanto a  $I_l$  e  $I_m$  contribuem, juntos, para a reconstrução MVS em pontos SFM relevantes:

$$e_{lm} = \sum_{P_j \in \Theta^{lm}} \frac{f(P_j, I_l, I_m)}{f(P_j, V_j)}, \text{ onde } \Theta^{lm} \text{ denota um conjunto de pontos SFM visíveis em } L_l \text{ e } I_m. \text{ Intuitivamente, as imagens com alta contribuição no MVS têm pesos}$$

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,  $P_j$ , deixe  $C_k = \text{argmax}_{Cl} f(P_j, Cl)$  ser 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. Uma vez que 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.

Em vez disso, consideramos ações cujas pontuações são mais de 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 qualquer 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 empregado algum algoritmo de reconstrução MVS, neste caso, o PMVS-2 (Patch-based Multiview Stereo Versão 2) (??????).

### PMVS-2

O PMVS-2 utiliza a técnica de DoG e cantos de Harris. O DoG é utilizado para detecção de bordas, subtraindo o resultado de dois Gaussianos com escalas dife-

rentes 4. O operador de Harris emprega uma auto-correlação local para melhorar a consistência da borda, extraíndo a borda e os cantos dos *features* das imagens. A resposta de Harris é positiva em regiões com cantos, negativas em bordas e pequenas em regiões planas. Além disso, no PMVS-2, usando pontos de amostras das imagens como sementes, as linhas epipolares são usadas para decidir a região correspondente (dentro de uma área 2x2 pixels) em outra imagem, gerando *patches* (cada uma definida com seu centro, normal e visibilidade) para atender às restrições na visibilidade, e levando à uma correspondência baseada em *patches* entre imagens. A correspondência *Multi-view* no PMVS-2 é baseada em *patches* e depende da consistência fotográfica média de todos os pares visíveis. Um *patch* é reconstruído usando maximizando o valor médio da consistência 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*). 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  $R(p)$ , onde a imagem é a 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:

- Inicialização
  - Detecção de *features*
  - Correspondência guiada
- Expansão
- 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. Após isso 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 finalizada, temos como resultado a reconstrução esparsa da escultura. No caso do VisualSfM, como a reconstrução esparsa já é feita em passos anteriores (com o PBA 4.3), na realidade o PMVS-2 só é empregado para a reconstrução densa, ou seja, a inicialização 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 ??.

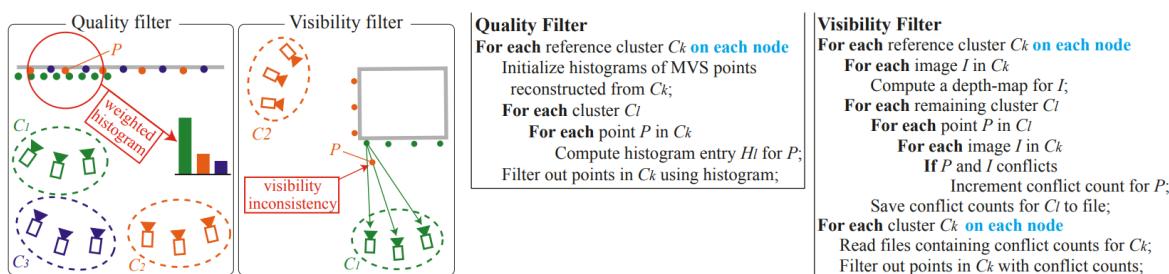


Figura 26 - Usabilidade dos filtros de qualidade e visibilidade, onde são aplicados fora do núcleo, assim como em paralelo. À esquerda, um ponto MVS  $P$  é testado usando os filtros. À direita, temos pseudo-códigos, onde os *loops* destacados em azul podem ser executados em paralelo. (??).

## 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 cachos distantes produzem pontos escassos e ruidosos. Queremos filtrar o último, que é realizado pelo seguinte filtro de qualidade. Assumimos que  $P_j$  e  $V_j$  denotarem um ponto MVS e suas informações de visibilidade estimadas pelo algoritmo MVS, respectivamente. Supomos que  $P_j$  foi reconstruído a partir do *cluster*  $C_k$ . Primeiramente coletamos pontos MVS  $Q_m$  e sua informação de visibilidade  $V_m$  de todos os *clusters* que possuam normais compatíveis com  $P_j$ , isto é, a diferença de ângulo menores que  $90^\circ$  e seus locais projetados estejam dentro de  $n$  pixels de  $P_j$  em cada imagem em  $V_j$ , (no caso, experimentalmente,  $n = 8$ ). A partir dos pontos MVS obtidos, calculamos 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  $C_l$ . Uma vez que um *cluster* possua pontos acurados e densos, este 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:  $H_k < 0,5\max_l H_l$ . Repetimos 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 [7, 9]. A diferença é que o PMVS reforça a consistência intra-cluster dentro de cada cluster, enquanto nosso 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, contamos 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.

### 4.3 VisualSfM

#### Introdução

VisualSfM (??) é um *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, que é ótima, por sinal. É altamente customizável, podemos utilizar o CUDA da NVIDIA, ou OpenGL, especificar a lista de pares para correspondência de imagens, usar detectores de *features* próprios, velocidade da detecção de *features*, da reconstrução densa, dentre outros parâmetros. Ou seja, é um *software* robusto, que pode ser usado em Linux, Windows ou até mesmo Mac.

Além disso, o VisualSfM é capaz de mostrar a matriz de correspondência de *features*, número de *features*, rodar um *Bundle Adjustment* independente, usar um Level 0 no PVMS, alterar a memória de GPU usada na reconstrução, deletar uma reconstrução indesejável ou até mesmo alterar parâmetros.

#### Procedimento

Sua linha de reconstrução é parecida com o MVE 4.1, porém é mais intuitiva. Em sua interface, possui um Log de mensagens e erros que por ventura venham a acontecer e na parte de cima, alguns botões 27



Figura 27 - Botões na parte superior da interface gráfica, este seria o procedimento padrão de funcionamento do *software*. (??)

Como demonstrado na imagem 27, o funcionamento seria da seguinte forma:

- **1 - Adicionar algumas imagens.** Este é o primeiro passo, para começar uma reconstrução, primeiro 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.** Agora, o *software* roda o algoritmo SIFT,

realizando todas as correspondências entre os *features*, que já foi discutido anteriormente.

- **3 - Reconstrução esparsa.** Neste passo, o VisualSfM roda o algoritmo de reconstrução esparsa (PBA/MCBA) (??) em todos os *features* descobertos no passo passado.

### *Bundle Adjustment*

O termo *Bundle* refere-se aos feixes de luz que refletem em cada *feature* 3D e vão para o centro de cada câmera da cena a ser reconstruída. Com isso, o *Bundle Adjustment* é o problema de refinar uma reconstrução visual para produzir estimativas com a visualização de parâmetros (estimativa e/ou calibração de câmeras), juntamente com as estruturas 3D.

Solucionadores de problemas de *Bundle Adjustment* (??) resumem-se em minimizar o erro de reprojeção 28 entre as posições de imagem dos pontos de imagem 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 (????), 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 *Bundle Adjustment*, as equações normais tem 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 descrição ?? é uma das abordagens mais comuns sobre o *Bundle Adjustment* e por isso, usaremos ela como exemplo.

### **Formulação inicial**

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

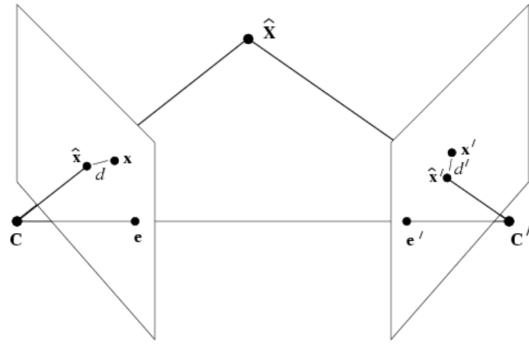


Figura 28 - Dado um ponto de um objeto 3D  $\hat{X}$ , sua reprojeção nas câmeras  $C$  e  $C'$ , são, respectivamente,  $x$  e  $x'$ . Esses pontos possuem um erro de reprojeção  $d$  e  $d'$  e ao utilizar o PBA/MCBA, os pontos  $x$  e  $x'$  serão reajustados como  $\hat{x}$  e  $\hat{x}'$ , respectivamente. (??)

### PBA/MCBA – Multi-Core Bundle Adjustment

Neste caso, o PBA/MCBA (????) 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 utilizado na conferência ECCV de 2010. É o primeiro sistema publicado, baseado em GPU que escala com 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 memória de troca 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 e economia de tempo.

Este algoritmo recebe como entrada conjunto de localizações e correspondências de *features* de imagens, o objetivo do *Bundle Adjustment* é encontrar posições de 3 pontos e parâmetros de câmera que minimizem o erro de reprojeção. Esse problema de otimização geralmente é formulado como um problema de mínimos quadrados não-lineares, onde o erro é a norma L2 quadrada da diferença entre a localização da característica observada e a projeção do ponto 3D correspondente no plano da imagem da câmera.

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:

$$x^* = \operatorname{argmin}_x \sum_{i=1}^k \|f_i(x)\|^2 \quad (19)$$

O algoritmo de Levenberg-Marquardt (LM) é o mais conhecido para resolução de problemas de mínimos quadrados não-lineares. O LM 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:

$$\delta^* = \operatorname{argmin}_{\delta} \|J(x)\delta + f(x)\|^2 + \lambda \|D(x)\delta\|^2 \quad (20)$$

e, atualiza  $x$ , da forma:

- 1: caso( $\|f(x + \delta^*)\| < \|f(x)\|$ )
- 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  $\lambda$  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  $\lambda$  a cada passo, baseado no quanto a Jacobiana ( $J(x)$ ) está próxima da função original ( $f(x)$ ). Resolver 20 é equivalente à resolver as equações normais

$$(J^T J + \lambda D^T D)\delta = -J^T f \quad (21)$$

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  $D$ ,  $\delta$  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 21 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*,  $U_\lambda$  e  $V_\lambda$  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:

$$(U_\lambda - WV_\lambda^{-1}W^T)\delta_c = -J_c^T f + WV_\lambda^{-1}J_p^T f \quad (22)$$

Onde  $S = U_\lambda - WV_\lambda^{-1}W^T$  é o complemento de Schur ou a matriz reduzida da câmera. Com a solução para 22, conseguimos obter os parâmetros do ponto com substituição regressiva:

$$\delta_p = -V_\lambda^{-1}(J_p^T f + W^T \delta_c) \quad (23)$$

Caso S for simétrica positiva-definida, a fatorização de Cholesky é indicado para a resolução de 22.

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.** Finalmente, acaba a reconstrução rodando o algoritmo de reconstrução densa (??) CMVS/PMVS-2 embutido no próprio VisualSfM.

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 (???????) é escalabilidade, pois seu propósito é utilizar imagens provenientes de sites na internet, em diferentes resoluções, como o Flickr.com. Ele usa a saída do Structure from Motion – SfM (mais especificamente a saída do passo anterior, do PBA) e utiliza 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 união das reconstruções de todos os *clusters* não deve perder detalhes que poderiam ser obtidos através do conjunto de imagens.

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 geralmente contêm centenas ou milhares de fotos adquiridas de quase mesmo ponto de vista, ou seja, um conjunto composto inteiramente informações duplicadas.

Em outras palavras, a sobreposição de *clusters* é definida por:

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

O CMVS pode ser resumido em quatro passos:

- 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

A obtenção de medidas precisas de visibilidade de pontos é fundamental para o sucesso do procedimento de visualização baseado em *clusters*. 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). Obtemos estimativas de visibilidade mais confiáveis ao agregar dados da visibilidade em uma vizinhança local, e mesclando os pontos nessa vizinhança. 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 realizada 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 dividindo os *clusters*, ignorando a cobertura. Mais especificamente, um *cluster* de imagens é dividido em componentes menores caso viole a restrição de tamanho. A divisão de um *cluster* é realizada pelo algoritmo *Normalized-Cuts* [23] em um gráfico de visibilidade, onde os nós são imagens. O peso da borda entre um par de imagens ( $I_l, I_m$ ) mede o quanto a  $I_l$  e  $I_m$  contribuem, juntos, 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  $\Theta^{lm}$  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 co-

bertura. 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,  $P_j$ , deixe  $C_k = \text{argmax}_{Clf}(P_j, Cl)$  ser 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. Uma vez que 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.

Em vez disso, consideramos ações cujas pontuações são mais de 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 empregado algum algoritmo de reconstrução MVS, neste caso, o PMVS-2 (Patch-based Multiview Stereo Versão 2) (??????).

## PMVS-2

O PMVS-2 utiliza a técnica de DoG e cíntios de Harris. O DoG é utilizado para detecção de bordas, subtraindo o resultado de dois Gaussianos com escalas diferentes 4. O operador de Harris emprega uma auto-correlação local para melhorar a consistência da borda, extraíndo a borda e os cíntios dos *features* das imagens. A resposta de Harris é positiva em regiões com cíntios, negativa em bordas e pequenas em regiões planas. Além disso, no PMVS-2, usando pontos de amostras das imagens como sementes, as linhas epipolares são usadas para decidir a região correspondente (dentro de uma área 2x2 pixels) em outra imagem, gerando *patches* (cada uma definida com seu centro, normal e visibilidade) para atender às restrições na visibilidade, e levando à uma correspondência baseada em *patches* entre imagens. A correspondência *Multi-view* no PMVS-2 é baseada em *patches* e depende

da consistência fotográfica média de todos os pares visíveis. Um *patch* é reconstruído usando maximizando o valor médiada consistência 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*). 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  $R(p)$ , onde a imagem é a 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:

- Inicialização
  - Detecção de *features*
  - Correspondência guiada
- Expansão
- 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. Após isso 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 finalizada, temos como resultado a reconstrução esparsa da escultura. No caso do VisualSfM, como a reconstrução esparsa já é feita em passos anteriores (com o PBA [4.3](#)), na realidade o PMVS-2 só é empregado para a reconstrução densa, ou seja, a inicialização 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 ??.

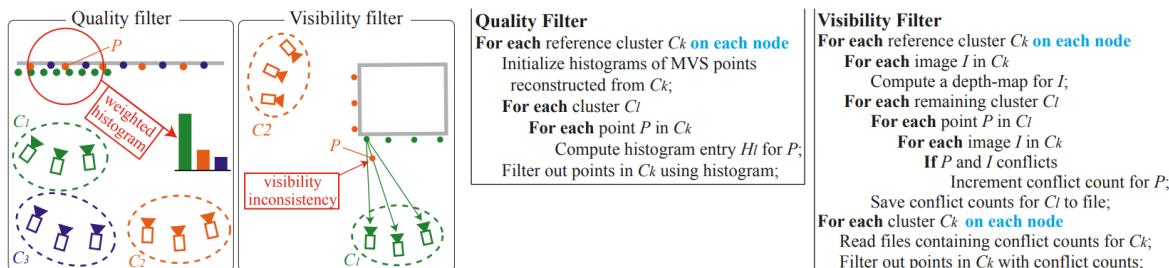


Figura 29 - Usabilidade dos filtros de qualidade e visibilidade, onde são aplicados fora do núcleo, assim como em paralelo. À esquerda, um ponto MVS  $P$  é testado usando os filtros. À direita, temos pseudo-códigos, onde os loops destacados em azul podem ser executados em paralelo. (??).

## 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 cachos distantes produzem pontos escassos e ruidosos. Queremos filtrar o último, que é realizado pelo seguinte filtro de qualidade. Assumimos que  $P_j$  e  $V_j$

denotarem um ponto MVS e suas informações de visibilidade estimadas pelo algoritmo MVS, respectivamente. Supomos que  $P_j$  foi reconstruído a partir do *cluster*  $C_k$ . Primeiramente coletamos pontos MVS  $Q_m$  e sua informação de visibilidade  $V_m$  de todos os *clusters* que possuam normais compatíveis com  $P_j$ , isto é, a diferença de ângulo menores que  $90^\circ$  e seus locais projetados estejam dentro de  $n$  pixels de  $P_j$  em cada imagem em  $V_j$ , (no caso, experimentalmente,  $n = 8$ ). A partir dos pontos MVS obtidos, calculamos 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  $C_l$ . Uma vez que um *cluster* possua pontos acurados e densos, este 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:  $H_k < 0,5\max_l H_l$ . Repetimos 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 [7, 9]. A diferença é que o PMVS reforça a consistência intra-cluster dentro de cada cluster, enquanto nosso 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, contamos 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.

## 5 EXPERIMENTOS

No presente trabalho, nos concentramos no objetivo proposto descrito na Seção 1.1.

### 5.1 Procedimento

Primeiramente, filmamos algumas esculturas utilizando a câmera de um *smartphone* convencional na resolução de 1920x1080 pixels. Esta filmagem foi realizada varrendo toda (ou maioria) da superfície da escultura em 360° com o intuito de ter toda a escultura reconstruída [30](#). Após isso, fizemos mais alguns vídeos, pegando alguns pontos que possuíam mais detalhes e que, com uma única varredura, não era capaz de reproduzir uma boa reconstrução.

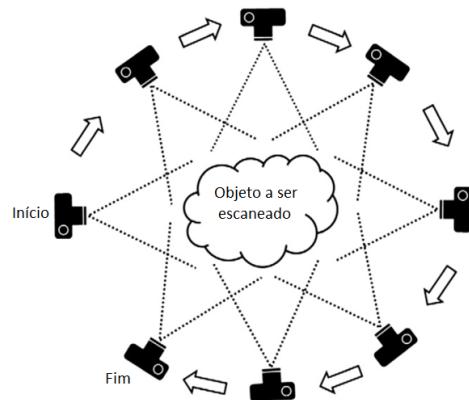


Figura 30 - Exemplo de como foi realizada a varredura da escultura

Com este material, foram feitos "cortes" em determinados *frames* do vídeo, com atenção para não cortar em *frames* muito juntos, pois aumentaria o número de correspondências ambíguas entre as imagens e com isso, o processamento da reconstrução demoraria mais. E não usar *frames* muito distantes, que ocorreria o inverso: com menos correspondências, ficariam buracos (partes sem a informação necessária) na reconstrução, como descrito na Seção 4.1.

Com isso em mente, foram reconstruídas duas esculturas: empregando o Visu-alSfM, usamos um único vídeo, totalizando 197 imagens. Com o MVE, utilizamos dois vídeos, que, ao cortá-los, totalizou cerca de 280 imagens.

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) na qual possui uma superfície propícia (Lambertiana) ([??](#)) para uma reconstrução.

Com o procedimento descrito anteriormente, a partir dos vídeos feitos, obtemos um total de 200 imagens em um vídeo superficial e mais 24 imagens mais detalhadas

do objeto, ambos numa resolução de 1080x1920 pixels. E, para um mesmo conjunto de imagens, rodamos tanto o VisualSfM quanto o MVE.

### 5.1.1 Resultados da reconstrução com o VisualSfM

Seguindo o passo-a-passo de reconstrução do software, obtivemos os seguintes resultados, para a escultura de 197 imagens do Jardim do Nêgo.

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> | 6.643 segundos |
| Gerar a reconstrução esparsa do modelo            | 220 segundos   |
| Gerar a reconstrução densa do modelo              | 1.385 segundos |

Com as seguintes reconstruções [31](#), [32](#). Percebemos que foi gerada uma nuvem de pontos bem consistente, a partir da reconstrução esparsa do algoritmo PBA. Por conta disso, nossa reconstrução 3D densa, empregando o MCBA/PMVS-2 obteve uma qualidade adequada para o conjunto de imagens usada.

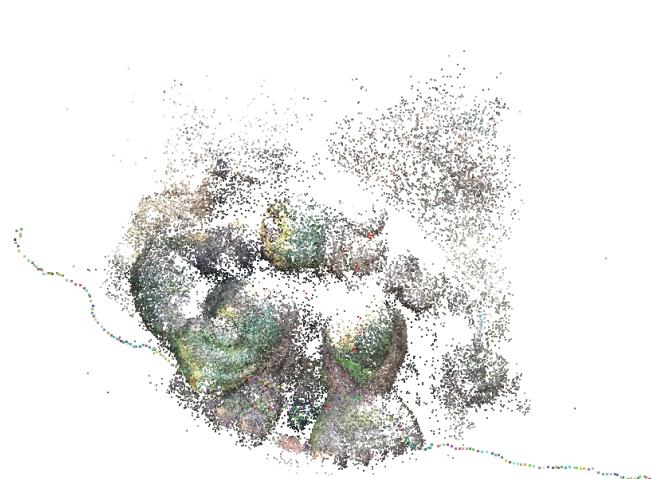


Figura 31 - Reconstrução esparsa da escultura do Jardim do Nêgo no VisualSfM com 197 imagens.

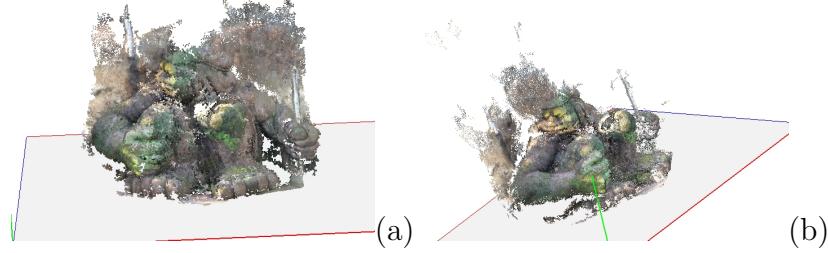


Figura 32 - Resultados da reconstrução densa da escultura do Jardim do Nêgo usando o VisualSfM, em dois ângulos diferentes (a) e (b).

Com o objeto em ambiente fechado, conseguimos os resultados a seguir.

Para o primeiro vídeo, convertido em 200 imagens:

Tabela 5 - Tempos obtidos da reconstrução do objeto usando o VisualSfM

| Procedimento                                      | Tempo (aprox.) |
|---|----------------|
| Carregamento de imagens                           | 50 segundos    |
| Calcular pares correspondentes de <i>features</i> | 9.540 segundos |
| Gerar a reconstrução esparsa do modelo            | 135 segundos   |
| Gerar a reconstrução densa do modelo              | 1.416 segundos |

A figura 33 mostra o resultado da reconstrução esparsa do algoritmo PBA. Não é tão nítida como na reconstrução densa 34, 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.

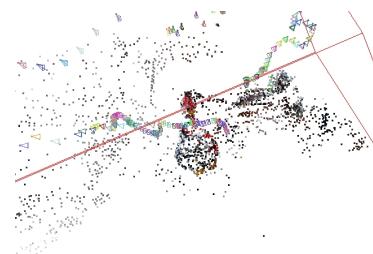


Figura 33 - Reconstrução esparsa do objeto no VisualSfM com 200 imagens.



Figura 34 - Reconstrução densa do objeto no VisualSfM com 200 imagens.

Fizemos uma outra reconstrução, utilizando os dois vídeos (gerando 224 imagens). Caso usássemos 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 melhorar esse problema. Portanto, o experimento seguiu da forma:

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

| Procedimento                                      | Tempo (aprox.)  |
|---|-----------------|
| Carregamento de imagens                           | 60 segundos     |
| Calcular pares correspondentes de <i>features</i> | 10.451 segundos |
| Gerar a reconstrução esparsa do modelo            | 162 segundos    |
| Gerar a reconstrução densa do modelo              | 1920 segundos   |

Percebemos que não foi tão proveitoso (qualitativamente) usar mais imagens neste caso, inclusive o algoritmo perdeu a referência do objeto e gerou um segundo modelo na reconstrução esparsa [36](#), e consequentemente, na reconstrução densa [36a](#) e [36b](#). O que gerou uma certa incoerência na reconstrução.



Figura 35 - Reconstrução esparsa do objeto com 224 imagens no VisualSfM.

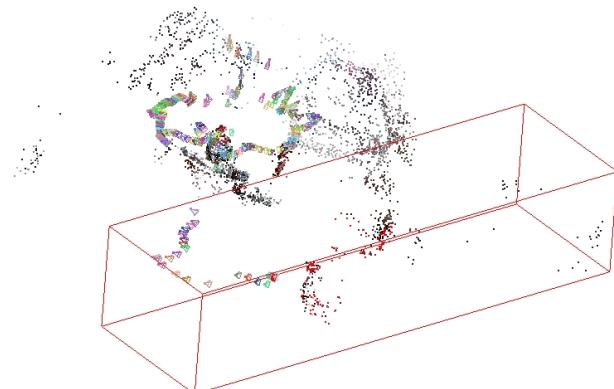


Figura 36 - Foram gerados dois modelos esparsos do objeto a partir do conjunto inicial de 224 imagens, provavelmente, proveniente da falta de parâmetros da câmera.

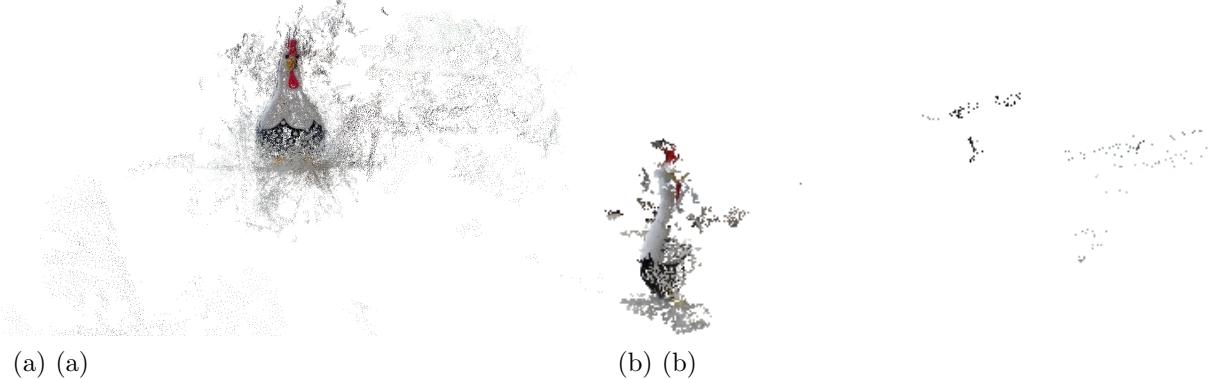


Figura 37 - Reconstruções densas do primeiro (a) e do segundo (b) modelo do objeto no VisualSfM com 224 imagens.

### 5.1.2 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). Amplamente configurável, podendo escolher a vizinhança, escala, manter o mapa de profundidade, ver os dados *EXIF* de cada imagem, dentre outras configurações.

Entretanto, para a aplicação proposta neste projeto, não é muito interessante, visto que ele utiliza a informação das câmeras, inseridas nas imagens (*EXIF*) e como as imagens empregadas na reconstrução são, tecnicamente, vídeos cortados em determinados *frames*, não é possível obter a informação das câmeras [38](#). 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.

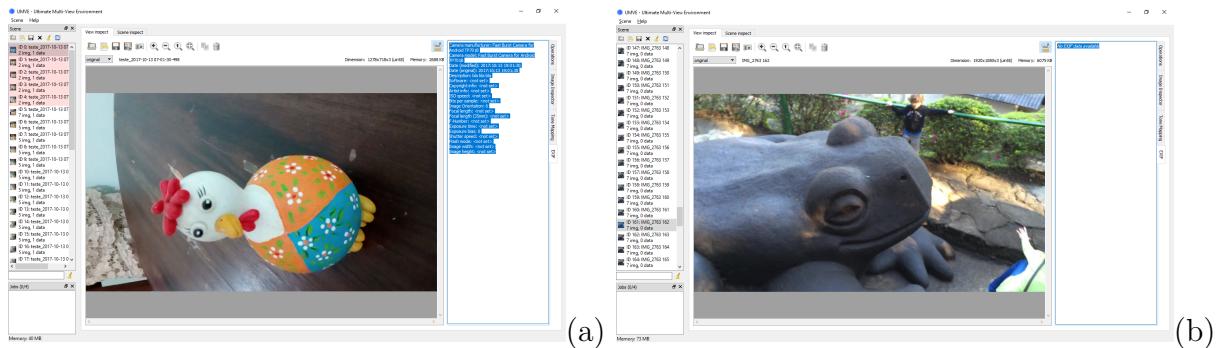


Figura 38 - A figura (a) é um exemplo onde a imagem possui dados na extensão *EXIF* (destacado em azul). Ao passo que a figura (b) é um frame de um vídeo, que não possui os dados das câmeras (destacado em azul).

Foi gerada uma reconstrução de um vídeo gravado de uma escultura no Jardim do Nêgo. O vídeo foi cortado em *frames* onde foram geradas 200 imagens base, com os parâmetros de câmera gerados pelo próprio software.

A partir disso, foi executado, todos os passos de uma reconstrução utilizando o MVE, de forma que, foram utilizadas as duas opções, tanto por linha de comando, quanto pela interface gráfica (UMVE).

Pela interface gráfica, o processo todo de reconstrução foi rápido (cerca de 30 minutos) 39, 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 7.

O UMVE não sinaliza quando o processo em execução termina, então, a explicação para essa discrepância no tempo é devido à execução de outro comando, sobrepondo o que já estava sendo executado, sem que o primeiro tivesse terminado.

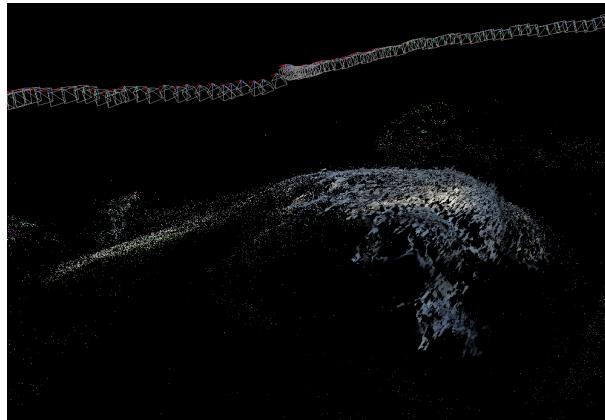


Figura 39 - Final da reconstrução via UMVE, percebe-se que alguns pontos não foram considerados, tendo como resultado uma "nuvem de pontos" mais densa, basicamente.

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

| Comando           | Tempo (aprox.)  |
|-------------------|-----------------|
| <i>sfmrecon</i>   | 78 segundos     |
| <i>dmrecon</i>    | 14.503 segundos |
| <i>scene2pset</i> | 600 segundos    |
| <i>fssrecon</i>   | 25.293 segundos |
| <i>meshclean</i>  | 60 segundos     |

Na reconstrução por linha de comando também é possível visualizar em qual etapa da execução o algoritmo está (figura 40), 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 41.



Figura 40 - Processos dentro do comando *sfmrecon*, onde (a) estão sendo detectadas as *features* do conjunto de imagens. Em (b) está computado o *pairwise matching* e em (c) está no processo de *Bundle Adjustment* (??), usando condições-padrão para as câmeras.

```
Selecionar Prompt de Comando
Rejected 201 tracks with large error.
Rejected 70 tracks behind cameras.
Rejected 674 tracks with unstable angle.
Deleted 0 of 23278 tracks (0.00%) above a threshold of 0.0159941.
Skipping full bundle adjustment (skipping 5 views).

Adding next view ID 65 (143 of 200)...
Collected 12 2D-3D correspondences.
Selected 10 2D-3D correspondences outliers (83%).
Reconstructed camera 65 with focal length 1
Running single camera bundle adjustment...
pBA: MSE 5.00345e-07 > 5.00159e-07 (S), 3 iter, 327ms.
Triangulated 1 new tracks, rejected 945 bad tracks.
Rejected 201 tracks with large error.
Rejected 70 tracks behind cameras.
Rejected 674 tracks with unstable angle.
Deleted 0 of 23279 tracks (0.00%) above a threshold of 0.0159941.
SFM reconstruction finished.

Running final bundle adjustment...
pBA: MSE 5.0055e-07 > 4.9486e-07 (S), 26 iter, 75509ms.
Camera 56, focal length: 0.935861 -> 0.935935, distortion: -0.0506553
Camera 57, focal length: 0.935864 -> 0.938748, distortion: -0.0554248
Camera 58, focal length: 0.955135 -> 0.953858, distortion: -0.0439308
Camera 59, focal length: 0.957488 -> 0.956769, distortion: -0.0595935
Camera 60, focal length: 0.963651 -> 0.964237, distortion: -0.0612002
Camera 61, focal length: 0.955873 -> 0.956757, distortion: -0.076264
Camera 62, focal length: 0.955131 -> 0.956728, distortion: -0.085152
Camera 63, focal length: 0.960946 -> 0.963876, distortion: -0.0631885
Camera 64, focal length: 0.970816 -> 0.975268, distortion: -0.0136306
```

Figura 41 - Término do comando *sfmrecon*, onde demorou cerca de 1 minuto e meio (75509 milisegundos).

O próximo comando, *dmrecon* demorou cerca de 4 horas, usando como configuração um nível L2, com 20 vizinhos 42. Usando um nível L0, o algoritmo rodou durante 6 horas aproximadamente e foi cancelado devido à demora na execução.

```

Selecionar Prompt de Comando
Count: 178570 filled: 67000 Queue: 22340
Count: 221173 filled: 75000 Queue: 6875
Count: 174699 filled: 68000 Queue: 22135
Count: 178779 filled: 69000 Queue: 21589
Count: 228433 filled: 76000 Queue: 3431
Count: 183540 filled: 70000 Queue: 20540
Filled 76658 pixels, i.e. 59.1 %.
MVS took 204 seconds.
Count: 187961 filled: 71000 Queue: 20256
Count: 191974 filled: 72000 Queue: 20095
Count: 196084 filled: 73000 Queue: 19739
Count: 202042 filled: 74000 Queue: 17974
Count: 206923 filled: 75000 Queue: 17279
Count: 210777 filled: 76000 Queue: 16925
Count: 216323 filled: 77000 Queue: 15475
Count: 221385 filled: 78000 Queue: 13979
Count: 226689 filled: 79000 Queue: 12274
Count: 234770 filled: 80000 Queue: 10341
Count: 240287 filled: 81000 Queue: 8352
Count: 248127 filled: 82000 Queue: 4630
Count: 254555 filled: 83000 Queue: 3094
Filled 83611 pixels, i.e. 64.5 %.
MVS took 216 seconds.
Reconstruction took 14502576ms.
Saving views back to disc...
Saving views to MVE files... done.

C:\Users\Desktop\Documents\UERJ\mve-20160517-win64>
C:\Users\Desktop\Documents\UERJ\mve-20160517-win64>
```

Figura 42 - Término do comando *dmrecon*, onde demorou cerca de 4 horas (14502576 milisegundos).

Usando o *scene2pset*, é necessário especificarmos em qual nível estamos recons-truindo e também uma saída válida. Por exemplo: ”*scene2pset.exe -Fnivel cena output*”. Onde o nível poderá ser um 0 (-F0), 1 (-F1) e assim por diante, a cena é o *input* e o *out-put* é um arquivo de extensão configurável, neste caso *.ply* 43. Este comando foi rápido, demorou cerca de 10 minutos, levando em conta todos os níveis.

```

Prompt de Comando - fssrecon pset_L0.ply pset_L1.ply pset_L2.ply pset_L3.ply final-output-mesh.ply
Processing view "IMG_2763_198" (with colors)...
Processing view "IMG_2763_191" (with colors)...
Processing view "IMG_2763_192" (with colors)...
Processing view "IMG_2763_194" (with colors)...
Processing view "IMG_2763_193" (with colors)...
Processing view "IMG_2763_195" (with colors)...
Processing view "IMG_2763_196" (with colors)...
Processing view "IMG_2763_197" (with colors)...
Processing view "IMG_2763_198" (with colors)...
Processing view "IMG_2763_199" (with colors)...
Processing view "IMG_2763_200" (with colors)...
Writing final point set (13543640 points)...
Writing PLY file (13543640 verts, with colors, with normals, with confidences, with values, 0 faces, with colors)... done.

C:\Users\Desktop\Documents\UERJ\mve-20160517-win64>scene2pset.exe -F3 teste pset_L3.ply
MVE Scene to PointSet (built on May 17 2016, 12:36:22)
Using depthmap "depth-L3" and color image "undist-L3"
Initializing scene with 200 views...
Initialized 200 views (max ID is 199), took 104ms.
Writing final point set (0 points)...
Writing PLY file (0 verts, with colors, with normals, with confidences, with values, 0 faces, with colors)... done.

C:\Users\Desktop\Documents\UERJ\mve-20160517-win64>fssrecon pset_L0.ply pset_L1.ply pset_L2.ply pset_L3.ply final-output-mesh.ply
Floating Scale Surface Reconstruction (built on May 17 2016, 12:36:32)
Loading: pset_L0.ply...
Loading samples took 0ms.
Loading: pset_L1.ply...
```

Figura 43 - Execução dos comandos *scene2pset*, nos níveis -F0, -F1, -F2 e -F3.

Para juntar todos os níveis do *scene2pset*, foi usado o *fssrecon*, que gera uma única reconstrução. Este processo demorou bastante, cerca de 7 horas 44. Que teve como resultado a malha 45.

```

Selecionar Prompt de Comando
Loading: pset_l3.ply...
Loading samples took 0ms.
Limiting octree to 20 levels...
Octree contains 19019463 samples in 5267793 nodes on 16 levels.
    Level 7: 3 samples
    Level 8: 1330 samples
    Level 9: 34013 samples
    Level 10: 509845 samples
    Level 11: 3294553 samples
    Level 12: 8815390 samples
    Level 13: 6128926 samples
    Level 14: 235361 samples
    Level 15: 42 samples
Computing sampling of the implicit function...
Sampling the implicit function at 6796595 positions, fetch a beer...
Processing voxel 6796595 (100.00%, 421:19, ETA 307445734561825:51)...
Generated 6796595 voxels, took 25292543ms.
Extracting isosurface...
    Sanity-checking input data... took 430 ms.
    Computing Marching Cubes indices... took 8041 ms.
    Computing isovertices... took 14438 ms.
    Computing isopolygons... took 41304 ms.
    Computing triangulation... took 5218 ms.
    Done. Surface extraction took 69698ms.
Deleting zero confidence vertices... took 219ms.
Mesh output file: final-output-mesh.ply
Writing PLY file (1826729 verts, with colors, with confidences, with values, 3640878 faces)... done.
All done. Remember to clean the output mesh.

C:\Users\Desktop\Documents\UERJ\mve-20160517-win64>

```

Figura 44 - Progressão do comando *fssrecon*, onde possui o ETA – *Estimated Time of Arrival*.

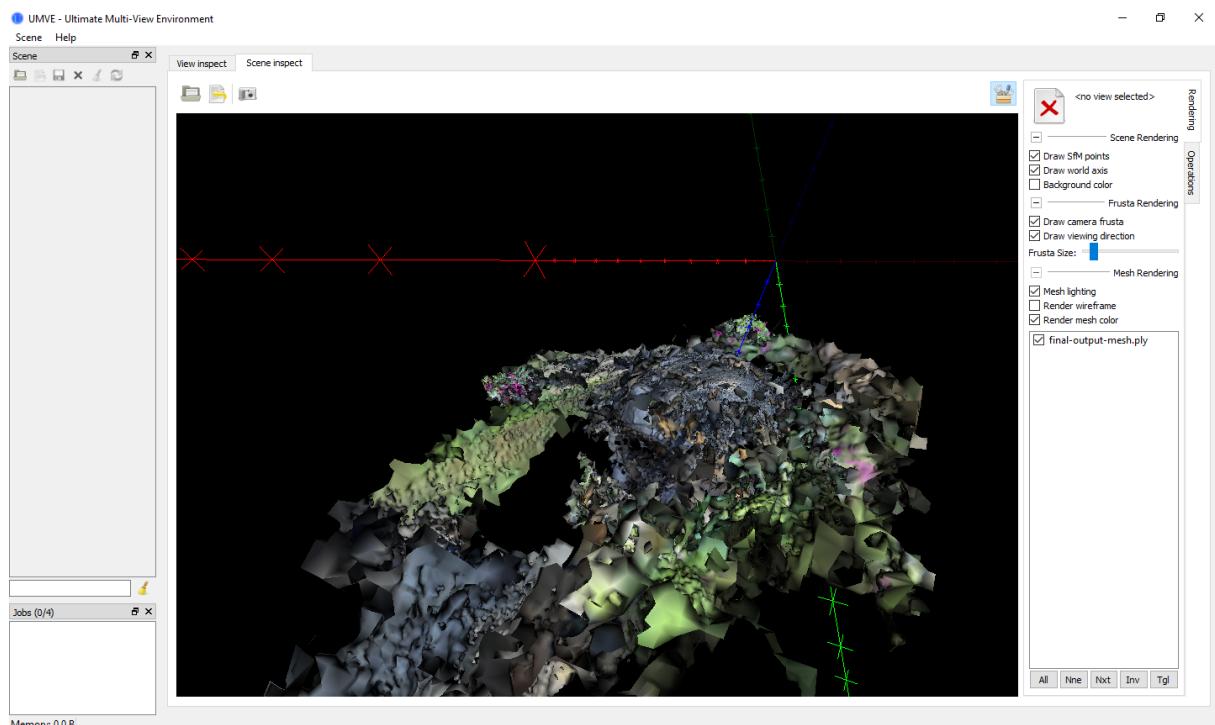


Figura 45 - Malha com ruídos proveniente do comando *fssrecon*.

Finalmente, basta limpar a malha atual com o comando *meshclean*, onde foi obtido o resultado [46](#).

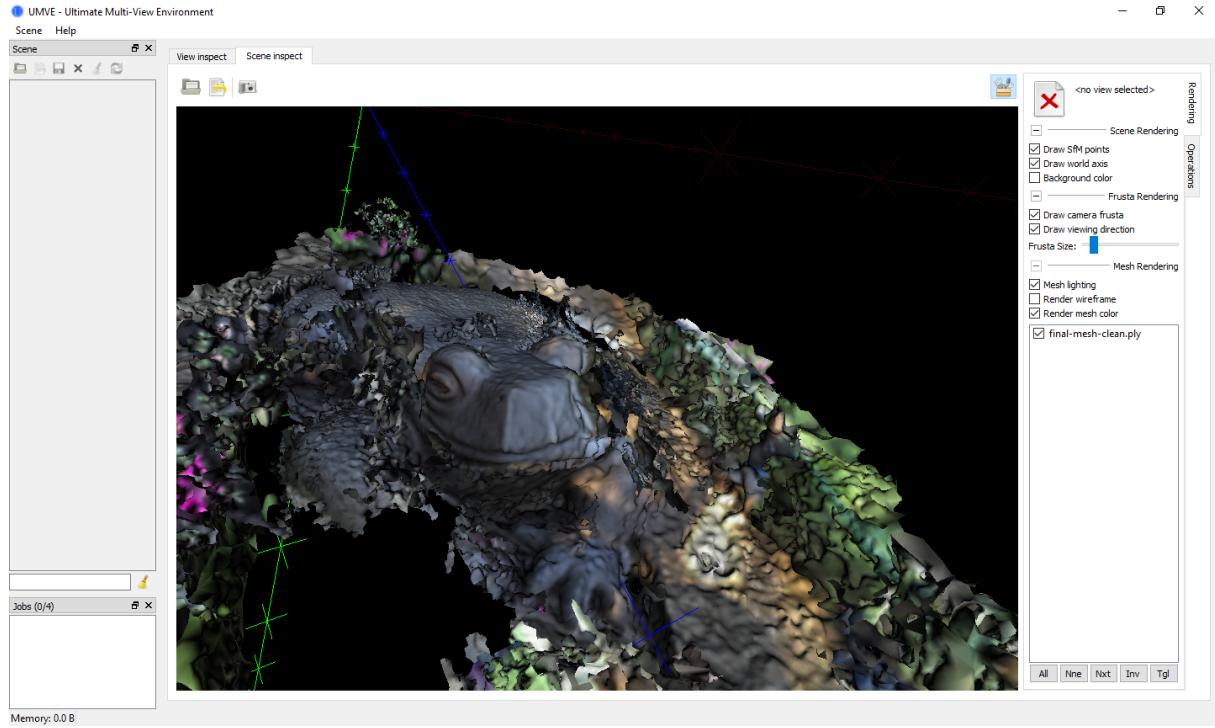


Figura 46 - Resultado final, após a remoção dos ruídos da malha.

Para comparação, usamos o mesmo objeto utilizado na reconstrução do VisualSfM (a galinha) e fizemos o passo a passo com o MVE: com a interface gráfica (UMVE), criamos uma nova cena e inserimos, primeiramente, as 200 fotos do objeto. Em seguida, utilizando as linhas de comando do MVE, fizemos o procedimento padrão de reconstrução do software. E, obtivemos os seguintes resultados [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>   | 371 segundos   |
| <i>dmrecon</i>    | 3.716 segundos |
| <i>scene2pset</i> | 300 segundos   |
| <i>fssrecon</i>   | 1.695 segundos |
| <i>meshclean</i>  | 45 segundos    |

Figura 47 - Tempo gasto da etapa *sfmrecon* do MVE

Figura 48 - Tempo da etapa *dmrecon* do MVE

Figura 49 - Tempo da etapa *fssrecon* do MVE

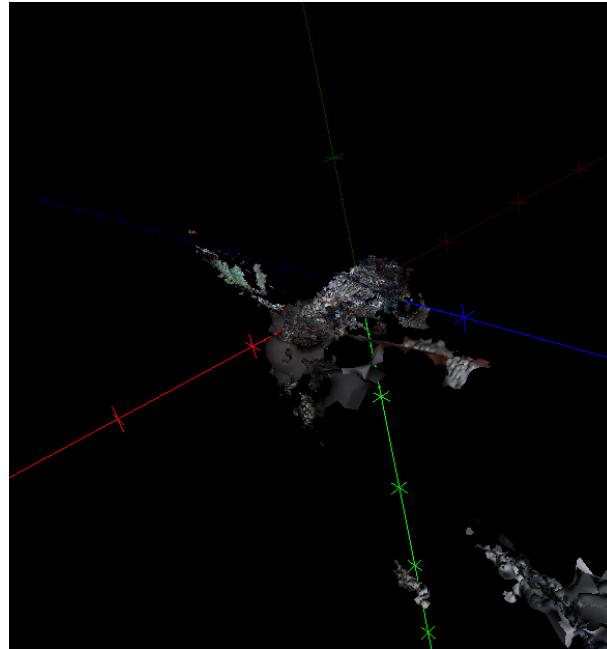


Figura 50 - Resultado da etapa *fssrecon* do MVE

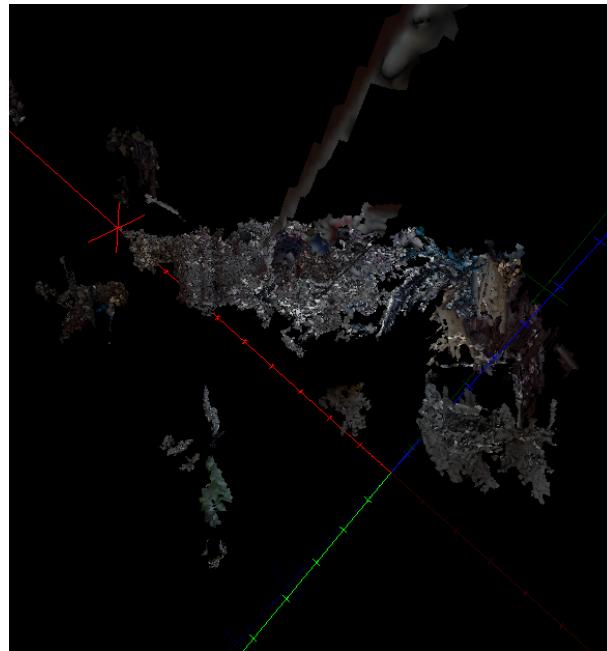


Figura 51 - Resultado da etapa *meshclean*, da etapa anterior 50

A etapa de *scene2pset* demorou cerca de 20 segundos (total). Porém percebemos que 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 essa reconstrução acima 50 e 51.

Rodamos também, com as 224 fotos, só foi possível executar o passo *sfmrecon* 52, pois o MVE não conseguiu rodar o comando *dmrecon* por algum motivo, e não gerou nenhum resultado para a continuação do algoritmo 53.

Figura 52 - Resultado da etapa *sfmrecon*, com todas as imagens

Figura 53 - Resultado da etapa *dmrecon*, com todas as imagens

## 6 CONCLUSÃO

Apresentamos técnicas de reconstruções utilizando fotogrametria, mais especificamente para esculturas à céu aberto, aprendendo sobre a calibração de equipamentos, software a serem utilizados e sobre como fazer uma boa varredura, cobrindo toda a escultura com uma câmera comum de celular.

### Trabalhos futuros

Identificamos os seguintes caminhos para a evolução deste projeto:

- **Realizar uma varredura com o Kinect.** Embora seja custoso, tanto fisicamente quanto computacionalmente, seria interessante ter um parâmetro de comparação com as técnicas de fotogrametria 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, o melhor método de varredura da escultura.** Verificamos que um dos melhores modos de se escanear uma escultura de grande porte seria escaneá-la várias vezes, a fim de que se pegue todos os detalhes, cobrindo toda a área a ser reconstruída. Mas será que este é realmente o melhor método?
- **Realizar uma reconstrução de curvas.** Será possível utilizar uma reconstrução baseada em curvas para auxiliar na reconstrução de nuvem de pontos e superfícies densas?
- **Concretizar o objetivo proposto neste trabalho.** Ir algumas 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.

## REFERÊNCIAS