Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

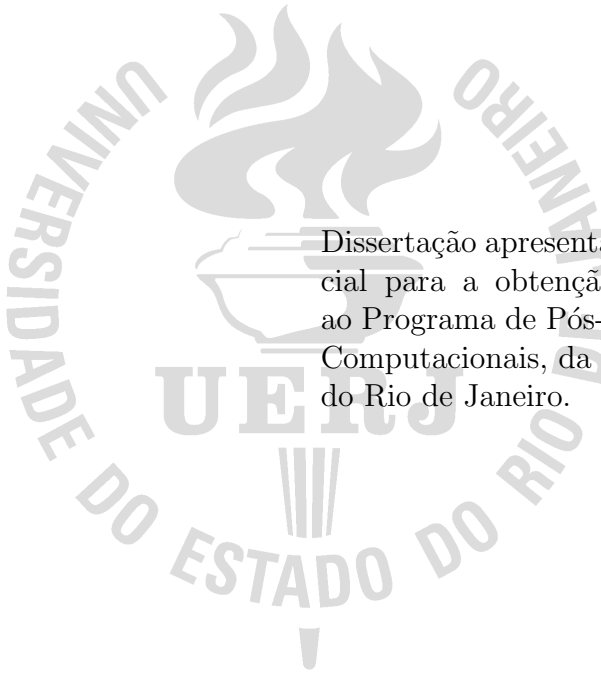
Bruno Cardozo Cotrim da Costa

Avaliação da digitalização 3D obtida com o Kinect One

Rio de Janeiro
2019

Bruno Cardozo Cotrim da Costa

Avaliação da digitalização 3D obtida com o Kinect One



Dissertação apresentada, como requisito parcial para a obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Dr. Guilherme Lucio Abelha Mota

Rio de Janeiro

2019

CATALOGAÇÃO NA FONTE
UERJ / REDE SIRIRUS / BIBLIOTECA CTC-A

C837 Costa, Bruno Cardozo Cotrim da
Avaliação da Digitalização 3D Obtida com o Kinect One / Bruno Cardozo Cotrim da Costa. - 2019.
168 f.

Orientador: Guilherme Lucio Abelha Mota.
Dissertação (Mestrado em Ciências Computacionais) – Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística.

1. Digitalização - Teses. 3. Kinect(Controlador programável) - Avaliação - Teses. 3. Computação gráfica - Teses. I. Mota, Guilherme Lucio Abelha. II. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. III. Título.

CDU 004.92

Rosalina Barros CRB-7 / 4204 - Bibliotecária responsável pela elaboração da ficha catalográfica.

Assinatura Data

Bruno Cardozo Cotrim da Costa

Avaliação da digitalização 3D obtida com o Kinect One

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovado em 22 de fevereiro de 2019.
Banca Examinadora:

Prof. Dr. Guilherme Lucio Abelha Mota (Orientador)
Instituto de Matemática e Estatística – UERJ

Prof. Dr. Gilson Alexandre Ostwald Pedro da Costa
Instituto de Matemática e Estatística – UERJ

Prof.ª Dra. Cristiane Oliveira de Faria
Instituto de Matemática e Estatística – UERJ

Prof. Otávio da Fonseca Martins Gomes
Centro de Tecnologia Mineral/MCTI

Rio de Janeiro
2019

DEDICATÓRIA

A minha mãe, Marcia Elisa Cardozo Cotrim. Pela dedicação e esforço em estar sempre me apoiando, motivando e agindo com rigor quando necessário.

AGRADECIMENTOS

Primeiramente a Deus, por ter me colocado no caminho certo durante uma crise complicada no país.

Ao meu orientador Dr. Guilherme Mota, por ter confiado a mim um trabalho complexo e desafiador, sempre acreditando na minha capacidade.

À minha mãe, Marcia Elisa, que sempre me apoiou e motivou, principalmente quando fiquei muito doente durante a execução desse trabalho.

À Oswaldo Mello Andrade e Roberto Luís Andrade, pelo auxílio na obtenção e construção dos objetos de teste deste trabalho.

A todos vocês, muito obrigado.

Os pequenos atos que se executam
são melhores que todos aqueles grandes
que apenas se planejam.
George C. Marshall

RESUMO

COSTA, Bruno Cardozo Cotrim da. *Avaliação da digitalização 3D obtida com o Kinect One*. 2019. 168 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2019.

Este trabalho avalia a medição de forma possibilitada pelo Kinect One, um sensor baseado na tecnologia *time of flight* comercializado pela Microsoft Corporation originalmente para ser utilizado em conjunto com o console Xbox One. Cinco objetos geométricos diferentes foram submetidos à digitalização 3D utilizando este sensor. No procedimento de digitalização 3D e análise dos resultados produzidos foram utilizadas diferentes peças de software: o aplicativo MeshLab para a reconstrução, a apresentação dos modelos 3D e obtenção de medidas; a biblioteca de alvos codificados ARUCO para a criação de referências para o recorte das nuvens de pontos; a biblioteca PCL (*Point Cloud Library*) para a segmentação e corte das áreas de interesse das nuvens de pontos e manipulação geral das nuvens de pontos; além de aplicativos específicos construídos para o trabalho. As nuvens de pontos finais de cada objeto digitalizado foram comparadas com modelos de referência visualmente e através de métricas objetivas. Dentre as medidas utilizadas encontram-se área, volume dos modelos 3D, assim como variantes da distância de Hausdorff. Os resultados obtidos indicam que a exatidão das nuvens de pontos obtidos pelo Kinect One é maior na faixa mínima de distância de trabalho recomendada (0,5 m). Os valores de erro obtidos nesta faixa variam de 2,8 mm a 6,8 mm no erro médio e com uma faixa de desvio padrão variando de 3,0 mm a 7,2 mm.

Palavras-chave: Digitalização 3D. Kinect One. Avaliação de Qualidade. Nuvens de Pontos.

ABSTRACT

COSTA, Bruno Cardozo Cotrim da. *Evaluation of 3D scan obtained with Kinect One.* 2019. 168 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2019.

This work evaluates the measurement in a way made possible by Kinect One, a sensor based on time of flight technology marketed by Microsoft Corporation originally to be used in conjunction with the Xbox One console. Five different geometric objects were subjected to 3D scanning using this sensor. In the 3D scanning procedure and analysis of the results produced, different pieces of software were used: the MeshLab application for the reconstruction, the presentation of the 3D models and obtaining measurements; the ARUCO coded target library for creating references for the clipping of points clouds; the PCL (Point Cloud Library) library for segmenting and cutting areas of interest of the point clouds and general manipulation of the point clouds; as well as specific applications built for the job. The final point clouds of each scanned object were compared to reference models visually and through objective metrics. Among the measures used are area, volume of 3D models, as well as variants of Hausdorff distance. The results indicate that the accuracy of the point clouds obtained by Kinect One is higher in the minimum recommended working distance range (0.5 m). The error values obtained in this range vary from 2.8 mm to 6.8 mm in the mean error and with a standard deviation range varying from 3.0 mm to 7.2 mm.

Keywords: 3D Scanning. Kinect One. Quality evaluation. Points Cloud.

LISTA DE ILUSTRAÇÕES

Figura 1 - Ilustração do Kinect One 23
Figura 2 - Ilustração do Kinect One com a parte frontal desencapada 23
Figura 3 - Ilustração dos diferentes campos de visão das câmeras do *Kinect One* . 24
Figura 4 - Figura que ilustra o funcionamento de uma câmera tempo de voo . . . 26
Figura 5 - Ilustração de uma nuvem de pontos de um coelho 28
Figura 6 - Ilustração da nuvem de pontos da face de uma pessoa 29
Figura 7 - Ilustração de uma nuvem de pontos de uma região 30
Figura 8 - Ilustração de uma triangulação de Delaunay no plano 33
Figura 9 - Alteração de arestas entre os pontos 34
Figura 10 - Ilustração de um escaneamento com *Kinect Fusion* 35
Figura 11 - Exemplo de marcações ARUCO comumente utilizadas 38
Figura 12 - Exemplo de marcações com o dicionário $DICT\_4X4\_50$ 39
Figura 13 - Exemplo de *setup* de digitalização completo 41
Figura 14 - Exemplo de *setup* de digitalização completo 42
Figura 15 - Exemplo de *setup* de digitalização completo 43
Figura 16 - Exemplo de sistema de coordenadas usado pelo *Kinect One* 45
Figura 17 - Exemplo de cubo 49
Figura 18 - Exemplo de esfera 49
Figura 19 - Exemplo de paralelepípedo de frente 50
Figura 20 - Exemplo de paralelepípedo achatado 50
Figura 21 - Exemplo de paralelepípedo alto 51
Figura 22 - Exemplo de mesa usada na digitalização 52
Figura 23 - Exemplo de nuvem de pontos inicial 53
Figura 24 - Exemplo de cena inicial de digitalização 54
Figura 25 - Exemplo da distância de Hausdorff 56
Figura 26 - Modelo 3D final da esfera 61
Figura 27 - Modelo 3D final do cubo 62
Figura 28 - Modelo 3D final do paralelepípedo 1 63
Figura 29 - Modelo 3D final do paralelepípedo 2 64
Figura 30 - Modelo 3D final do paralelepípedo 3 65
Figura 31 - Modelo 3D final da esfera triangulação de Delaunay 66
Figura 32 - Modelo 3D final do cubo pela triangulação de Delaunay 67
Figura 33 - Modelo 3D final do cubo pela triangulação de Delaunay 68
Figura 34 - Modelo 3D final do paralelepípedo 1 pela triangulação de Delaunay . . 69
Figura 35 - Modelo 3D final do paralelepípedo 1 pela triangulação de Delaunay . . 70
Figura 36 - Modelo 3D final do paralelepípedo 1 pela triangulação de Delaunay . . 71

Figura 37 - Modelo 3D final do paralelepípedo 2 pela triangulação de Delaunay . . 72
Figura 38 - Modelo 3D final do paralelepípedo 2 pela triangulação de Delaunay . . 73
Figura 39 - Modelo 3D final do paralelepípedo 3 pela triangulação de Delaunay . . 74
Figura 40 - Modelo 3D final do paralelepípedo 3 pela triangulação de Delaunay . . 75
Figura 41 - Modelo 3D de referência do cubo 76
Figura 42 - Modelo 3D de referência da esfera 77
Figura 43 - Modelo 3D de referência do paralelepípedo 1 78
Figura 44 - Modelo 3D de referência do paralelepípedo 2 79
Figura 45 - Modelo 3D de referência do paralelepípedo 3 80

LISTA DE TABELAS

Tabela 1 - Informações técnicas sobre o Kinect One 22
Tabela 2 - Resultados com 60 cm e 60.008 pontos 59
Tabela 3 - Resultados com 60cm e 642 pontos para o objeto esfera 59
Tabela 4 - Resultados com 60 cm e 10.008 pontos 59
Tabela 5 - Resultados com 120 cm e 60.008 pontos 60
Tabela 6 - Resultados com 120 cm e 642 pontos para o objeto esfera 60
Tabela 7 - Resultados com 120cm e 10.008 pontos 60
Tabela 8 - Resultados complementares de área e volume 81

LISTA DE ABREVIATURAS E SIGLAS

ICP	*Iterative Closest Point*
PCL	*Point Cloud Library*
CBBS	Confederação Brasileira de Bilhar e Sinuca

SUMÁRIO

	INTRODUÇÃO	15
1	**FUNDAMENTAÇÃO TEÓRICA**	21
1.1	**Microsoft Kinect One**	21
1.2	**Modelo Tempo de Voo**	25
1.3	**Nuvens de Pontos**	27
1.4	**ICP - *Iterative closest point***	27
1.5	***Screened Poisson***	31
1.6	**Triangulação de Delaunay**	32
1.7	***Kinect Fusion***	34
2	**INFRAESTRUTURA DE SOFTWARE**	36
2.1	**libfreenect2 0.2**	36
2.2	**Meshlab 2016.12**	36
2.3	**ARUCO 3.0.6**	37
2.4	**PCL 1.7.1 - *Point Cloud Library***	37
2.5	**OpenCV 3.4.1**	39
3	**MÉTODO**	40
3.1	**Descrição Geral do Método Proposto**	40
3.2	**Etapas do Processo**	40
3.2.1	Etapa 1: Construção do *Setup*	40
3.2.2	Etapa 2: Captura	44
3.2.3	Etapa 3: Detecção de Marcações	44
3.2.4	Etapa 4: Segmentação	44
3.2.5	Etapa 5: Alinhamento	46
4	**PROCEDIMENTO EXPERIMENTAL, RESULTADOS E ANÁLISE**	48
4.1	**Descrição dos Objetos**	48
4.2	**Mesa Digitalizadora**	51
4.3	**Captura**	51
4.4	**Distância de Hausdorff**	55
4.5	**Métricas Adicionais**	57
4.6	**Análise dos Resultados**	57
4.6.1	Avaliação Numérica	81
4.6.2	Avaliação Visual	83
4.6.3	Comparação	84
	CONCLUSÃO	86
	REFERÊNCIAS	88

GLOSSÁRIO 97
APÊNDICE A – Códigos 99
APÊNDICE B – Tutorial Instalação e Configuração 162

INTRODUÇÃO

Motivação

Nos últimos anos, a indústria de consoles de jogos eletrônicos recebeu uma grande melhoria nas tecnologias de reconhecimento de gestos e rastreamento de movimento. Esta evolução pode ser compreendida como uma resposta à crescente demanda por jogos que possibilitem experiências imersivas cada vez mais sofisticadas (PAGLIARI; PINTO, 2015).

O projeto Kinect, desenvolvido pela Microsoft Corporation, disponibiliza uma família de sensores que permite a aquisição simultânea diversos canais de áudio, além de imagens RGB, infravermelho[1] e de profundidade com uma considerável taxa de frames. Devido à natureza complementar das informações providas por este sensor e pelo seu baixo custo financeiro, ele tem se provado um recurso atrativo para pesquisadores das mais diferentes áreas (PAGLIARI; PINTO, 2015).

A primeira versão do Kinect foi lançada no ano de 2010 para ser utilizada com o console Xbox 360. Esta versão, referida a partir de agora como Kinect 360[2], é baseada na tecnologia de luz estruturada. Esta tecnologia é focada no processo de projetar um padrão conhecido (muitas vezes grades, barras horizontais ou verticais) em uma cena ou objeto, permitindo a estimação de sua forma a partir do princípio da triangulação (CHIA; CHEN; YUEH, 1996; PINHEIRO, 2013).

Em 2014, a Microsoft Corporation lançou uma nova versão de console no mercado, o Xbox One[3]. O respectivo sensor de movimento, referido pelo nome Kinect One, utiliza a tecnologia *time-of-flight*, baseada na medição do tempo que um sinal luminoso emitido demora para se locomover até a cena e retornar para o sensor (LEFLOCH et al., 2013).

As versões 360 e One do Kinect fazem parte de uma categoria de sensores para digitalização 3D de baixo orçamento, quando comparados a alternativas mais precisas (e custosas) como os sistema de varredura laser, os *laser scanners*. Atualmente, os sensores de varredura laser custam entre US$ 500 e US$ 410.000 e possuem uma exatidão que varia de 1 a 8 mm, segundo a análise conduzida pelo site all3dp em ⟨https://all3dp.com/1/best-3d-scanner-diy-handheld-app-software/#bq-ciclop⟩ e uma pesquisa de baixo custo conduzida pelo site 3dnatives em

[1] Em processamento digital de imagens, as imagens na banda infravermelha são muito utilizadas em aplicações de microscopia ótica, astronomia, sensoriamento remoto, aplicações industriais e policiamento (WOODS, 2010). A banda infravermelha costuma ser utilizada em conjunto com as bandas visíveis (vermelho, verde e azul) na formação de imagens.

[2] Na literatura, este modelo muitas vezes é referido como Kinect v1.

[3] Este modelo é também denominado na literatura Kinect v2.

⟨https://www.3dnatives.com/en/top-10-low-cost-3d-scanners280320174/⟩, todas efetuadas entre os anos de 2018 e 2019. Apesar da exatidão, o uso dos sensores de varredura laser convencionais, por conta de seu custo, pode ser inviável em aplicações que tenham como requisito valores restritos de orçamento.

Deve ser mencionado o fato de ser possível encontrar-se *laser scanners* por um peço mais modesto, variando entre US$ 500 e US$ 9.000, segundo a pesquisa de preços conduzida por Alexandrea Purvis (3DNATIVES, 2019). Porém, segundo a citada pesquisa, tais sensores apresentam limitações de tamanho dos objetos a serem digitalizados ou, ainda, na exatidão da medida de distância e, consequentemente, forma produzida.

Quanto à plataforma de funcionamento do sensor, naturalmente, o Kinect One pode ser usado em conjunto com o próprio console Xbox One, para o qual o sensor foi originalmente desenvolvido ou mesmo com um computador convencional. Por exemplo, em um computador com sistema operacional Micosoft Windows nas versões 8, 8.1 ou 10 é possível fazê-lo através do driver e da API de desenvolvimento oficiais disponibilizados pela Microsoft Corporation, o chamado Kinect for Windows SDK 2.0 (WEBB; ASHLEY, 2012; KEAN; HALL; PERRY, 2011). Por outro lado, existe a possibilidade de utilização do *Kinect* através de uma plataforma livre para alguma das distribuições do sistema operacional Linux com os drivers e da API libfreenect2 (XIANG; KERL; WIEDEMEYER, 2016; BLAKE; ECHTLER; KERL, 2015) (utilizado neste trabalho) e, também do OpenNI2 (SNYDER, 2013; KAWATA; MURAO, 2017). Dentre os usos do Kinect One mais frequentes na literatura podemos mencionar aplicações na área de robótica (ROSSI, 2016; FANKHAUSER et al., 2015; OLIVER et al., 2012; ZENNARO, 2014).

No âmbito da presente linha de pesquisa, uma investigação anterior (SOUZA, 2016) empregou o Kinect 360 para a digitalização 3D de artefatos arqueológicos metálicos. Uma das limitações deste trabalho pregresso foi a ausência de avaliação quantitativa da exatidão dos modelos geométricos obtidos. Assim, a presente pesquisa se dedica à avaliação da exatidão da digitalização 3D produzida pelo Kinect One.

Contexto

A exatidão da medida de distância fornecida pelo Kinect One foi avaliada e analisada nos trabalhos conduzidos em (WASENMÜLLER; STRICKER, 2016; GONZALEZ-JORGE et al., 2013; CUI; STRICKER, 2011; CLARKSON et al., 2012; LANGE, 2000; ZENNARO, 2014; OLIVER et al., 2012; FANKHAUSER et al., 2015; ANDERSEN et al., 2012; BONNECHERE et al., 2014; MANKOFF; RUSSO, 2013; DUTTA, 2012; YANG et al., 2015).

Diversos experimentos foram conduzidos em (SARBOLANDI; KOLB, 2015) utilizando o kinect one e o kinect 360 em ambientes variados, buscando avaliar sua exatidão

e precisão de distância nesse diferentes ambientes. Os cenários incluíam situações com incidência dos raios solares, objetos de teste transparentes, objetos de teste metálicos com elevado grau de reflectância, interferência com múltiplos dispositivos, super aquecimento do sensor, entre outros. Os resultados foram comparados, atestando a superioridade do kinect one na maioria dos casos.

Também avaliando a exatidão e precisão da medida de distância do kinect one e kinect 360, (PAGLIARI; PINTO, 2015) criou uma função para estimar o erro de profundidade das duas versões do kinect e avaliando os dados de entrada, chegou a conclusão de que o kinect one possui sua taxa de erro aumentando de forma linear de acordo com o crescimento da distância em relação ao objeto medido, enquanto que com o kinect 360, o erro aumenta de forma quadrática.

Essa exatidão de distância já foi verificada e analisada desde o lançamento do Kinect One no mercado, porém existem poucos trabalhos que analisam e verificam a medida de forma obtida na digitalização de objetos, alguns deles são:

(TONG et al., 2012), que digitalizou o corpo de um ser humano, utilizando 3 Kinects diferentes a 1 metro de distância. Ele obteve uma taxa de exatidão que varia de 1,5 cm a 6 cm a longo das diferentes partes do corpo, mesmo para esse objeto de teste complexo que é o corpo humano.

A pesquisa conduzida em (WASENMÜLLER; STRICKER, 2016) apresentou um erro de exatidão constante de 18 mm, comparando as duas versões existentes do Kinect.

Outro trabalho que executou validações com o Kinect muito similares às apresentadas no presente trabalho foi (GONZALEZ-JORGE et al., 2013). Nessa pesquisa, objetos também foram digitalizados e a taxa de erro da exatidão encontrada variou de 5 mm para 15 mm a 1 metro de distância dos objetos de teste. Aumentando a distância para 2 metros, o erro da exatidão variou de 5 mm a 25 mm.

Um trabalho também muito similar ao apresentado aqui é o (LACHAT et al., 2015), avaliando a possibilidade utilização do Kinect One na digitalização 3D de objetos, gerando seus respectivos modelos 3D. Nessa pesquisa, foi utilizado como objeto de teste um fragmento da balaustrada de arenito de dimensões 40 por 20 cm vindo da Catedral de Estrasburgo Notre-Dame (França)[4]. O sensor foi posicionado a 1m de distância do objeto de teste e foi obtida uma taxa de exatidão que variou de 2 a 6mm com um desvio padrão de 2mm.

Em (GONZALEZ-JORGE et al., 2015) foi feita uma comparação metrológica entre as duas versões do kinect. A digitalização foi baseada em cinco esferas distintas e 7 cubos

[4] A Catedral de Estrasburgo ou Catedral de Nossa Senhora de Estrasburgo (em francês Cathédrale Notre-Dame-de-Strasbourg) é uma catedral católica romana em Estrasburgo, França. Tornando-se o mais alto edifício do mundo entre 1625 a 1874.

de tamanhos diferentes como objetos de teste. A distância de aquisição dos dados também foi variada. Com o sensor a 1m de distância, os resultados de exatidão dos dois sensores foram similares, com exatidão variando de 2mm a 6mm dependendo do objeto. Com o aumento da distância para 2m, o Kinect 360 piorou sua exatidão para 12mm, enquanto o Kinect One obteve um resultado constante de 8mm.

(LYSENKOV; ERUHIMOV; BRADSKI, 2013) utilizou objetos transparentes (um grande problema para a digitalização com o kinect) para a digitalização e avaliou os resultados para o reconhecimento de padrões, desenvolvendo no processo uma abordagem de segmentação para a melhoria dos dados adquiridos.

A pesquisa feita em (JIAO et al., 2017) procurou avaliar a precisão e exatidão obtida com o Kinect One digitalizando toda uma cena e a partir daí foi desenvolvido e proposto um método de pós retificação dos dados de profundidade obtidos com o objetivo de melhorar a exatidão e acurácia desses medidas. Os resultados experimentais demostraram que a abordagem dessa pesquisa melhorou consideravelmente a precisão e exatidão dos dados obtidos inicialmente com o Kinect One.

Objetivo

O objetivo deste trabalho é avaliar quantitativamente e qualitativamente a medida de forma fornecida pelo Kinect One. Para isto, serão utilizados objetos de referência com dimensões e modelos tridimensionais conhecidos. Serão digitalizados cindo objetos específicos: uma esfera, um cubo e um paralelepípedo em três posições diferentes (simulando três objetos distintos). Para a geração dos modelos tridimensionais, os corpos de prova sarão cuidadosamente posicionados em um ambiente controlado.

Adicionalmente, a fim de evitar alguns inconvenientes e limitações presentes no licenciamento do Kinect SDK 2.0 (MICROSOFT, 2019), os aplicativos implementados neste trabalho serão desenvolvidos na distribuição Ubuntu (CANONICAL, 2019) com base na biblioteca libfreenect2 (XIANG; KERL; WIEDEMEYER, 2016; BLAKE; ECHTLER; KERL, 2015). Assim sendo,a proposta é fazer uso de diversas bibliotecas com o intuito de facilitar o suporte a muitos dos requisitos do protótipo, permitindo o desenvolvimento de um software completo. Portanto, dentre as bibliotecas utilizadas, destacam-se: **libfreenect2** (XIANG et al., 2016), biblioteca e driver de controle do Kinect One; **ARUCO** (GARRIDO-JURADO et al., 2014; GARRIDO-JURADO R. MUNõZ-SALINAS, 2016), uma biblioteca de alvos codificados flexível, de código aberto e com capacidade de representar até 1.024 alvos diferentes; **OpenCV** (BRADSKI; KAEHLER, 2013), biblioteca de processamento de imagens que possui uma quantidade elevada de algoritmos de visão computacional implementados; **PCL**, biblioteca de processamento e manipulação de nuvens de pontos. Para a fusão das diversas nuvens de pontos parciais, obtidas ao longo

do processo de digitalização 3D, assim como a geração de modelos e malhas 3D a partir das nuvens de pontos finais de cada objeto digitalizado, será utilizado o software ***Meshlab*** (CIGNONI et al., 2008; CIGNONI; ROCCHINI; SCOPIGNO, 1998; KAZHDAN; HOPPE, 2013).

A validação dos modelos 3D obtidos ao longo de todo o processo de digitalização e modelagem 3D é feita pelo método de validação da **Distância de Hausdorff** (HUTTENLOCHER; KLANDERMAN; RUCKLIDGE, 1993) (ASPERT; SANTA-CRUZ; EBRAHIMI, 2002), para cada um dos objetos de teste, verificando a exatidão e eficiência dos modelos gerados a partir das nuvens de pontos capturadas pelo *Kinect One*, além de ser utilizar também 3 outras medidas derivadas da distância de Hausdorff: Distância de Hausdorff Mínima, média dos valores dos erros de cada ponto da nuvem de pontos capturada e desvio padrão da média dos erros de cada ponto da nuvem.

Além da Distância de Hausdorff, também é usado como forma de avaliar a exatidão das nuvens adquiridas as informações de **área** e **volume** dos modelos geradas, para cada objeto de teste, comparando-as as informações de referência de cada objeto.

Um programa foi desenvolvido para este trabalho com o objetivo de executar todos os cálculos necessário para encontrar a distância de hausdorff e suas variações, para cada objeto, enquanto que os cálculos de área e volume foram medidos através do software *meshlab*.

Estrutura do Documento

O restante desse documento está organizado como descrito a seguir:

O capítulo 1 apresenta os fundamentos teóricos necessário para a compreensão do método utilizado e dos experimentos que foram executados.

No capítulo 2 são descritos os software aplicativos e as bibliotecas utilizados no desenvolvimento do trabalho.

O capítulo 3 apresenta o método utilizado na presente dissertação, incluindo todas as etapas que o compõe, começando pela construção do *setup* de digitalização até a segmentação das nuvens de pontos e fusão das nuvens de pontos parciais.

No capítulo 4, é apresentado o procedimento experimental, descrevendo cada um dos objetos escolhidos para teste e incluindo detalhes do processo de digitalização, assim como é informado os programas que foram desenvolvidos para esse trabalho com o objetivo de implementar todas as etapas do método anterior. Nele, em seguida, são exibidos e analisados os resultados obtidos. A análise visa à avaliação da exatidão da medida de forma produzida.

Por fim, nas considerações finais, são apresentadas as conclusões, além das alternativas que não foram exploradas nesse trabalho e que são boas opções para trabalhos

futuros, e que tenham como objetivo dar prosseguimento a esta pesquisa.

1 FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata dos fundamentos teóricos necessários para a compreensão do método utilizado neste trabalho para a obtenção e dos modelos tridimensionais bem como para sua análise.

1.1 Microsoft Kinect One

O Kinect One é um sensor que foi desenvolvido pela Microsoft Corporation para o console de jogos eletrônicos Xbox One. Lançado nos mercados americano e brasileiro em 22 de novembro de 2013, este sensor é uma câmera ativa, diferente dos outros dispositivos lançados pelas empresas concorrentes (como Wii Remote Control comercializado pela Nintendo Co., Ltd. e o Playstation Move produzido pela Sony Corporation), permitindo interagir com o console sem a necessidade do usuário carregar nenhum tipo de controle. Assim, o Kinect possibilita uma interação remota baseada somente na captura e reconhecimento de gestos.

O projeto Kinect disponibiliza um sensor de baixo custo que permite a aquisição de informações de profundidade em tempo real e de imagens RGB e infravermelho a 30 frames por segundo (PAGLIARI; PINTO, 2015). Ele é, basicamente, composto por uma câmera RGB, uma câmera infravermelho, um projetor infravermelho e um microfone. Devido à natureza complementar das informações providas, ele tem se provado um recurso atrativo para pesquisadores das mais diferentes áreas (PAGLIARI; PINTO, 2015).

O Kinect One apresenta uma grande evolução em relação ao seu predecessor, por basear-se na tecnologia *time-of-flight* (PAGLIARI; PINTO, 2015; SARBOLANDI; KOLB, 2015). Dentre as demais melhorias apresentadas, estão o aumento da resolução das câmeras, que agora possuem resoluções de 1920 × 1080 pixels (*FullHD*), para câmera RGB, e 512 × 424 pixels, para a câmera infravermelho. Somadas à performance da tecnologia *time-of-flight*, que daqui para frente será chamada de tempo de voo, tais melhorias proveem uma maior exatidão e aquisição mais completa da cena 3D, assim como melhorias nos algoritmos para rastreamento de esqueleto[5] (*skeleton tracking*) (PAPADOPOU-

[5] O reconhecimento de esqueletos constitui um campo amplamente estudado e um tópico muito ativo à comunidade de pesquisa em visão computacional (JI; LIU, 2010). Isso se deve ao amplo conjunto de campos potenciais onde os resultados da pesquisa podem ser aplicados comercialmente, como vigilância, segurança, interação humano-computador, casas inteligentes, auxilio a idosos e deficientes, entre outros.

Tabela 1 - Informações técnicas sobre o Kinect One

Resolução da câmera RGB	1920 x 1080 pixels
Resolução da câmera infravermelho	512 x 424
Margem de captura de dados de profundidade com o sensor infravermelho	0,5m a 4,5m
Campo de visão horizontal da câmera RGB	70 graus
Campo de visão vertical da câmera RGB	60 graus
Frequências possíveis para o sensor infravermelho	120, 80 e 16 MHz
Tamanho do pixel da câmera RGB	3.1 µm
Tamanho do pixel da câmera infravermelho	10 µm
Entrada	USB 3.0
Preço médio	$199 dólares

Legenda: Tabela com as informações técnicas do Kinect One. Mais detalhes podem ser encontrados em SELL J.; O'CONNOR, 2014.

Fonte: Microsoft, 2014

LOS; AXENOPOULOS; DARAS, 2014) e reconhecimento de gestos[6] (*gesture recognition*) (ROSSI, 2016).

Seguem na tabela 1 algumas das informações técnicas sobre o Kinect One.

O Kinect não funciona corretamente sobre incidência direta de radiação solar, com eficiência dependendo do ângulo de incidência dos raios. Para mais detalhes sobre os problemas e condições adversas encontradas na utilização do Kinect One veja (SARBOLANDI; KOLB, 2015);

A ficha técnica detalhada pode ser acessada em (SELL J.; O'CONNOR, 2014).

A Figura 1 ilustra o Kinect One da forma como vem de fábrica, pronto para uso, enquanto a Figura 2 o ilustra removendo sua capa frontal para que seja possível observar a câmera RGB a esquerda, o emissor infravermelho no centro e o sensor infravermelho no centro da porção esquerda.

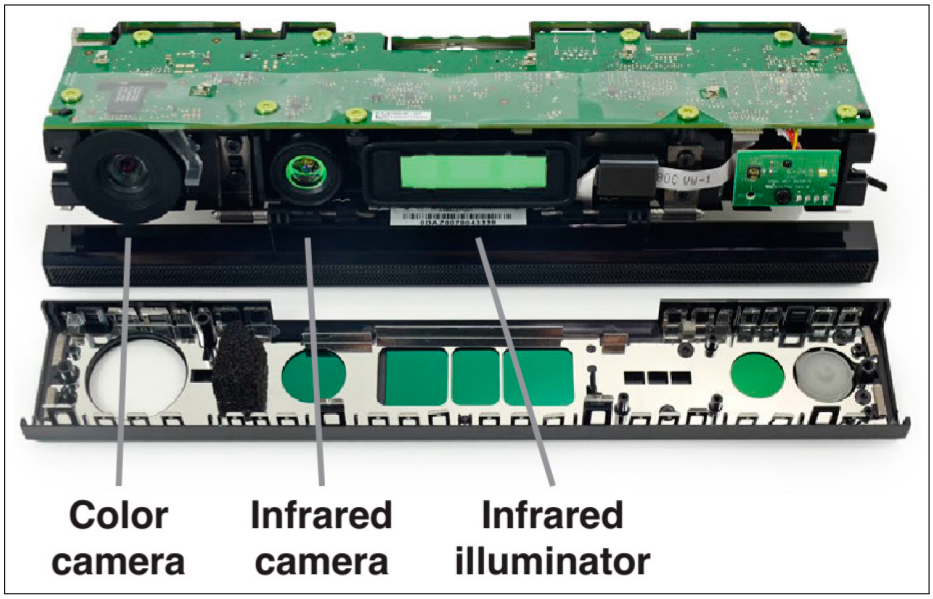
A Figura 3 ilustra os diferentes campos de visão das câmeras RGB e infravermelho. O azul representa o campo de visão da câmera infravermelho, enquanto o verde representa o campo de visão da câmera RGB.

[6] O reconhecimento de gestos é um tópico em ciência da computação e tecnologia do idioma (*language technology*) com o objetivo de interpretar gestos humanos através de algoritmos matemáticos.

Figura 1 - Ilustração do Kinect One

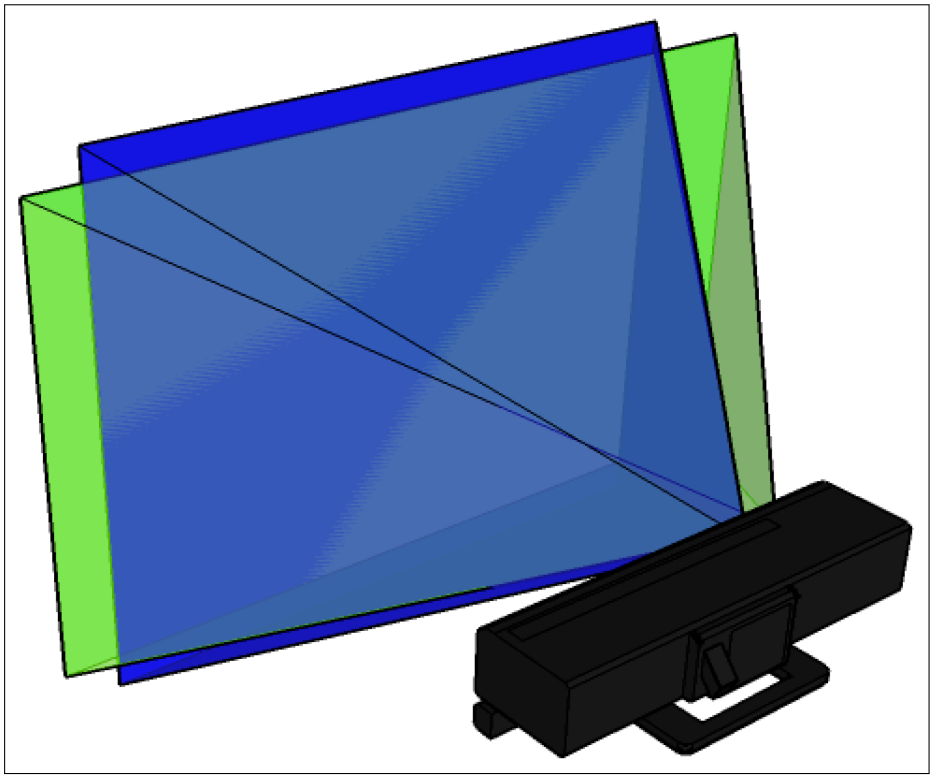
Legenda: Ilustração do Kinect One na forma como vem de fábrica.
Fonte: Microsoft Corporation, 2014

Figura 2 - Ilustração do Kinect One com a parte frontal desencapada



Legenda: Ilustração do Kinect One removendo sua capa frontal para que seja possível observar em detalhes a câmera RGB a esquerda, o sensor infravermelho no meio e o emissor infravermelho no canto direito.
Fonte: CHEN et al., 2018

Figura 3 - Ilustração dos diferentes campos de visão das câmeras do *Kinect One*



Legenda: Ilustração dos diferentes campos de visão das câmeras do *Kinect One*. O azul representa o campo de visão da câmera infravermelho, enquanto o verde representa o campo de visão da câmera RGB.

Fonte: PAGLIARI; PINTO, 2015

1.2 Modelo Tempo de Voo

Conforme mencionado anteriormente, o Kinect One utiliza a tecnologia tempo de voo. Essa tecnologia é baseada na medição do tempo que um sinal emitido demora para se locomover até a cena e retornar para o sensor (LEFLOCH et al., 2013). Na última década, esse princípio encontrou realização em dispositivos microeletrônicos, resultando em novos dispositivos de detecção de alcance, as chamadas câmeras tempo de voo (SARBOLANDI; KOLB, 2015). Nesta seção, será explicado os princípios básicos de funcionamento dessas câmeras. É preciso acrescentar que poucos detalhes técnicos sobre o uso dessa tecnologia no Kinect são conhecidos, devido a segredos industriais.

O Kinect One utiliza a abordagem de Modulação de Intensidade de Onda (***Continuous Wave Intensity Modulation***[7]), que é a mais comumente usada em câmeras de tempo de voo. Quanto à distância entre os objetos observados e o sensor, assumindo que o sensor e o iluminador estejam posicionados no mesmo local e na mesma posição (são concêntricos) e considerando a velocidade finita da luz c, uma mudança no tempo ϕ[s] é causada no sinal óptico que é equivalente a mudança de fase do sinal periódico. Essa mudança é detectada em cada pixel do sensor através de um processo chamado *mixing* (HAGEBEUKER; MARKETING, 2007) (KOLB et al., 2010). A mudança de tempo pode ser transformada em distância do objeto ao sensor pela equação (1) abaixo, já considerando que a luz emitida precisa se deslocar pela distância duas vezes (SARBOLANDI; KOLB, 2015).

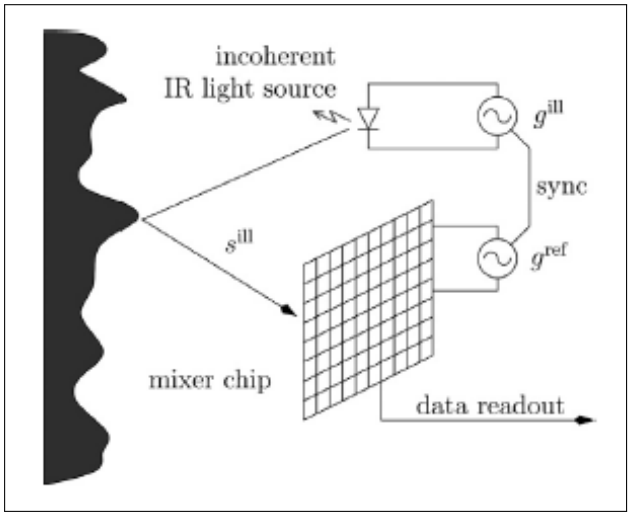
$$d = \frac{c\,\phi}{4\pi} \tag{1}$$

A modulação de intensidade da onda contínua é usada para estimar a distância entre o alvo (objetos observados na cena) e a fonte de iluminação ativa infravermelho (a câmera propriamente dita). Toda câmera tempo de voo que utiliza a abordagem de modulação de intensidade de onda contínua implementa essa função de correlação no seu próprio chip, que é composto do que é conhecido na literatura como pixels inteligentes (LANGE, 2000). A Figura 4 ilustra esse mecanismo de cálculo da distância.

Do ponto de vista técnico, o sinal gerado pelo emissor infravermelho g^{ill}, se desloca até a cena, é refletido de volta por ela e retorna para o sensor como sinal de incidência s^{ill} para cada pixel no chip do sensor. Esse sinal de incidência s^{ill} é correlacionado (correlação

[7] Além dessa abordagem, também existe uma outra abordagem chamada *Pulse Based Approach* (Abordagem Baseada em Pulsos), o conceito básico reside no fato de que a câmera projeta um pulso **NIR** (*near infrared* - infravermelho próximo) de luz com duração conhecida (ou seja, a dimensão é conhecida) e discretiza a frente da iluminação refletida. Essa discretização é realizada antes do retorno de todo o pulso de luz usando um obturador de câmera rápido. A porção do sinal de pulso refletido realmente descreve a forma do objeto observado (LEFLOCH et al., 2013).

Figura 4 - Figura que ilustra o funcionamento de uma câmera tempo de voo



Legenda: Figura que ilustra o funcionamento de uma câmera tempo de voo.
Fonte: SARBOLANDI; KOLB, 2015

espacial (WOODS, 2010)) com o sinal de referência g^{ref}. Essa abordagem produz a função de correlação (2) (SARBOLANDI; KOLB, 2015):

$$C[g^{ill}, g^{ref}] = s \otimes g = \lim_{T\to\infty} \int_{-T/2}^{T/2} s^{ill}(t) \cdot g^{ref}(t)dt \tag{2}$$

O deslocamento de fase é calculado usando várias medições de correlação com iluminação variável e pares de sinal de referencia e incidência g_i^{ref} e g_i^{ill}, respectivamente, utilizando uma função de desmodulação.

$$\phi = G(A_0, A_1, ..., A_n) \tag{3}$$

$$\text{com} \quad A_i = C[g_i^{ill}, g_i^{ref}], i = 1, ..., n \tag{4}$$

Frequentemente A_i é chamado de imagem de fase ou imagem de correlação. Na prática, as imagens de correlação são adquiridas sequencialmente, no entanto, existe a opção teórica de aquisição das imagens de correlação de forma paralela, utilizando diferentes mudanças de fase para cada pixel vizinho no momento de aquisição do sinal s^{ill} (LANGE; SEITZ, 2001). É importante notar que quanto a periodicidade do sinal de referência, cada câmera que utiliza a tecnologia tempo de voo possui seu próprio mecanismo único de desambiguação (SARBOLANDI; KOLB, 2015).

1.3 Nuvens de Pontos

Uma nuvens de pontos é um conjunto de pontos no espaço, que normalmente é definido por um sistema de coordenadas. Nuvens de pontos geralmente são produzidas a partir de digitalizadores 3D (*scanners 3D*), como o Kinect, que calculam e medem uma grande quantidade de pontos da superfície externa de objetos e ambientes.

As nuvens de pontos são utilizadas para diversos propósitos, incluindo a criação de modelos CAD[8] 3D, metrologia[9] (SENIN; COLOSIMO; PACELLA, 2013; LITTLE; JANNING, 1998), navegação de robôs (FANKHAUSER et al., 2015; OLIVER et al., 2012; ZENNARO, 2014; LI et al., 2018), imageamento médico (OLESEN et al., 2012; SITEK; HUESMAN; GULLBERG, 2006), arquitetura (NING et al., 2009; DIMITROV; GOLPARVAR-FARD, 2015), realidade virtual (BONNAFFE; JENNETTE; ANDREWS, 2007; LIN et al., 2008), inspeção de qualidade (SHI et al., 2006; ANIL et al., 2013), animação (REMONDINO, 2003; DOBASHI et al., 2000), entre outros.

As Figuras 5, 6 e 7 são exemplos de nuvens de pontos.

1.4 ICP - *Iterative closest point*

Iterative closest point é um algoritmo utilizado para minimizar a diferença entre duas nuvens de pontos. Ele é comumente utilizado na reconstrução de superfícies 2D e 3D cujos dados foram obtidos através de diferentes digitalizações, também é usado para a navegação de robôs e planejamento de caminhos ótimos, entre outras aplicações (CHEN; MEDIONI, 1992).

Durante a iteração do algoritmo, uma das nuvens de pontos é mantida fixa, chamada de referência ou *target*, enquanto que a outra é transformada para melhor se assemelhar com a primeira nuvem (BESL; MCKAY, 1992). O algoritmo revisa iterativamente a transformação (que é uma combinação de rotação e translação) objetivando minimizar o erro métrico, usualmente a distância da segunda nuvem para a nuvem de referência, como a soma das diferenças quadradas entre as coordenadas de cada um dos pares correspondentes das nuvens comparadas.

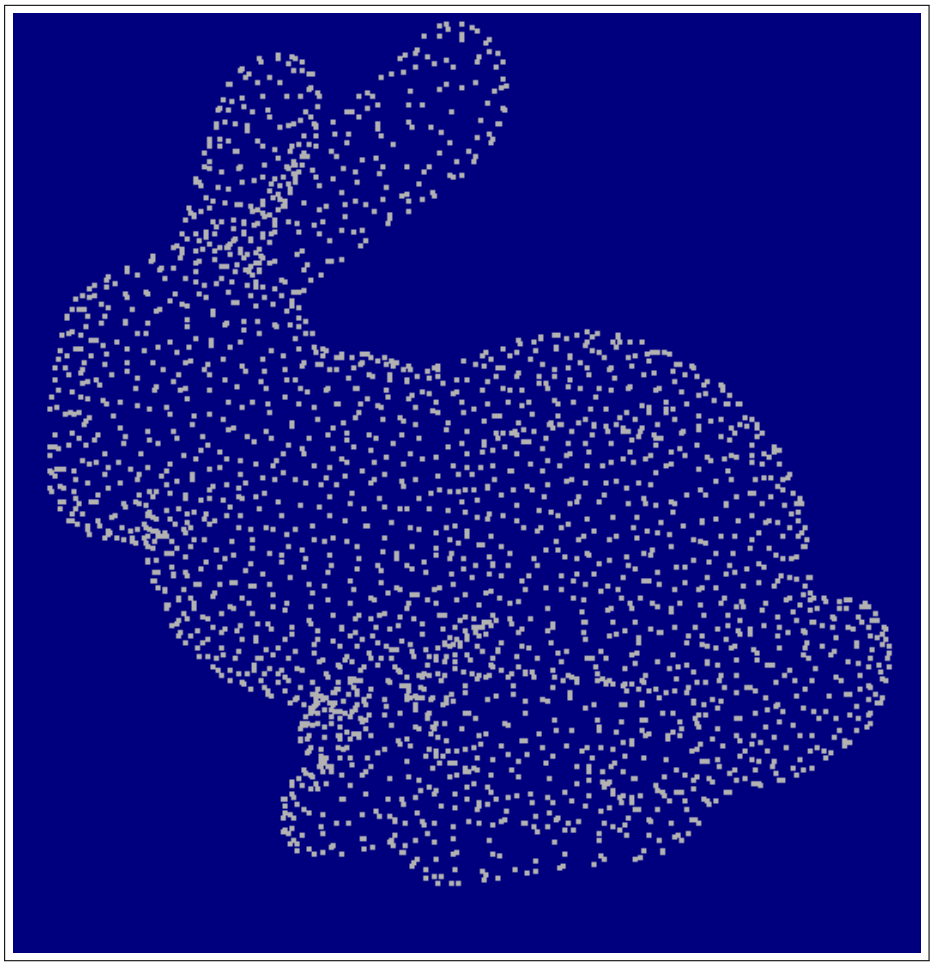
O algoritmo ICP foi primeiramente introduzido por Chen and Medioni (CHEN; MEDIONI, 1992) e Besl and McKay (BESL; MCKAY, 1992).

---

[8] ***Computer-aided design*** (**CAD**) é usado em computadores para auxiliar na criação, modificação, análise e otimização de projetos e modelos. Normalmente utilizados com o objetivo de aprimorar a qualidade, produtividade e comunicação. (NARAYAN K., 2008; DUGGAL, 2000)

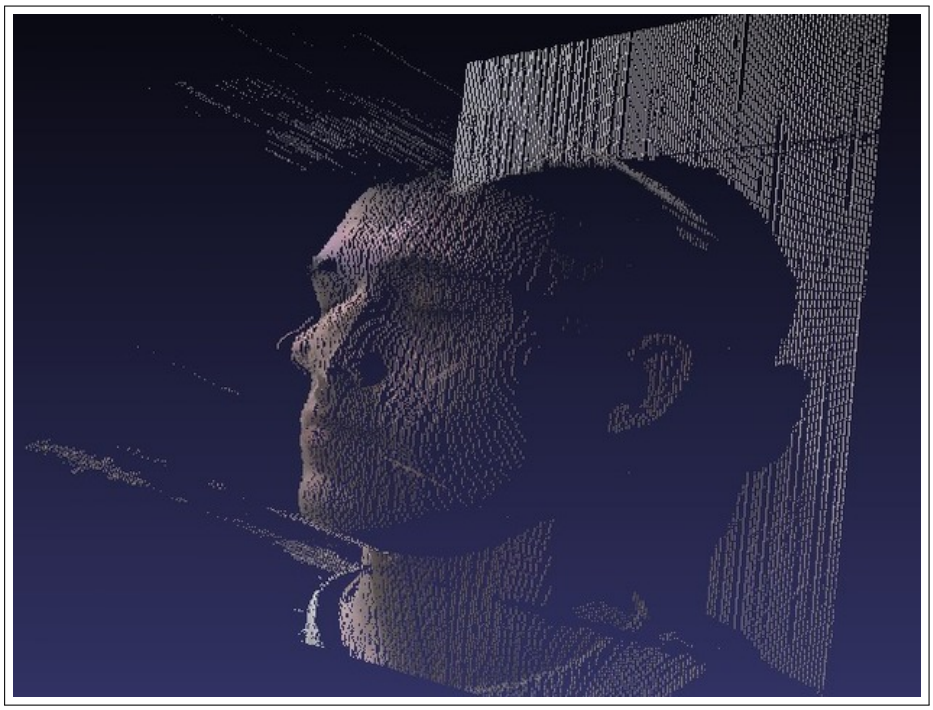
[9] A **Metrologia** é uma área que engloba todos os aspectos teóricos e práticos da medição, qualquer que seja a incerteza da medição e o campo de aplicação.

Figura 5 - Ilustração de uma nuvem de pontos de um coelho



Legenda: Ilustração de uma nuvem de pontos de um coelho de lado.
Fonte: CURLESS; LEVOY, 1996

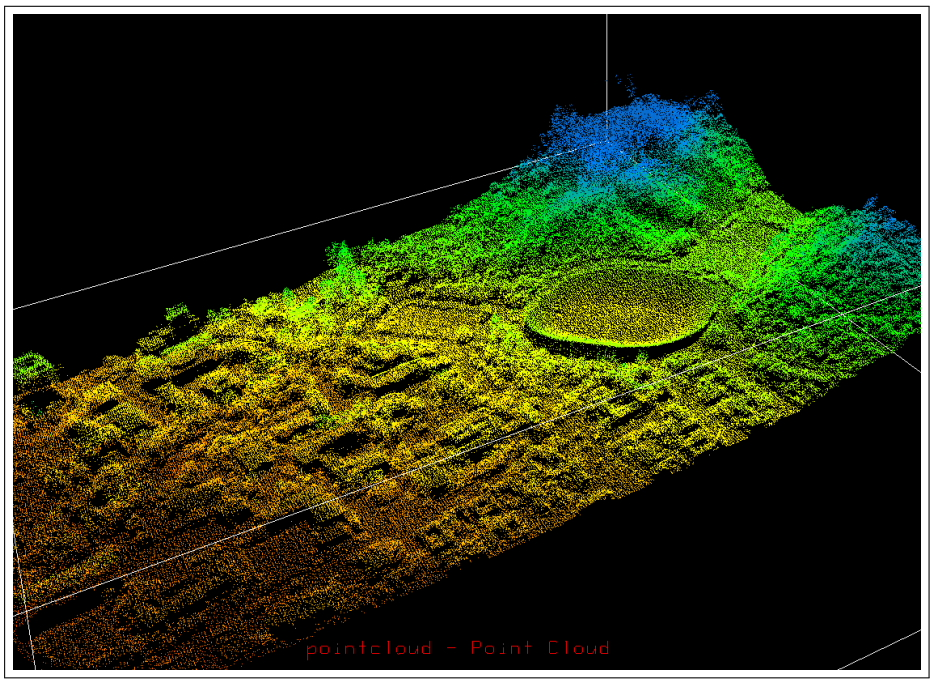
Figura 6 - Ilustração da nuvem de pontos da face de uma pessoa



Legenda: Ilustração da nuvem de pontos da face de uma pessoa logo a frente de um anteparo.

Fonte: https://www.thingiverse.com/thing:1535

Figura 7 - Ilustração de uma nuvem de pontos de uma região



Legenda: Ilustração de uma nuvem de pontos de uma superfície de uma região urbana, digitalizada de uma longa distância.

Fonte: https://www.unavco.org/software/visualization/idv/IDV$_{d}atasource_{p}oint_{c}loud.html$

As etapas para a execução do algoritmo podem ser observadas abaixo:

- Para cada ponto da segunda nuvem de pontos, encontrar o ponto mais próximo na nuvem de pontos de referência;
- Estimas a combinação de rotações e translações usando uma técnica de mínimos quadrados para cada par de pontos, assim será possível encontrar o melhor alinhamento entre o ponto da segunda nuvem com seu respectivo ponto da nuvem de referência;
- Transformar a segunda nuvem de pontos pela transformação encontrada;
- Iterar, reassociando os pontos novamente e repetindo todo o processo.

Após esse processo, as duas nuvens estarão alinhadas. Existem várias variantes do algoritmo ICP, mais variantes e detalhes dos mesmos podem ser encontrados em (POMERLEAU; COLAS; SIEGWART, 2015; LOW, 2004; POMERLEAU et al., 2013).

1.5 *Screened Poisson*

O algoritmo de reconstrução de superfícies **Screened Poisson** é uma técnica bastante conhecida na criação de superfícies a partir de um conjunto de pontos orientados adquiridos de um *scanner* 3D (KAZHDAN MICHAEL, 2013) (como o Kinect). A técnica apresenta resiliência contra ruídos e *outliers* nos objetos digitalizados, porém, segundo algumas análises conduzidas em (ALLIEZ et al., 2007; MANSON; PETROVA; SCHAEFER, 2008; CALAKLI; TAUBIN, 2011; BERGER et al., 2011; DIGNE et al., 2011), apresenta tendências de suavizar os dados trabalhados.

A abordagem da reconstrução de superfície empregada por essa técnica é baseada na observação de que o campo normal (apontando para dentro) do limite de um sólido pode ser interpretado como o gradiente da função indicativa do próprio sólido (KAZHDAN MICHAEL, 2013). Assim, dado um conjunto de pontos orientados que representem esses limites, uma malha pode ser obtida através da equação (5) transformando as amostras de pontos orientados em um campo vetorial contínuo 3D.

$$E(X) = \int |\nabla X(P) - \vec{V}(P)|^2 dp \tag{5}$$

Uma vez que o campo vetorial tenha sido encontrado, é possível encontrar a função escalar cujos gradientes melhor correspondam ao campo vetorial. Equação (6).

$$A_{ij} = [\nabla B_i, \nabla B_j]_{[0,1]^3} \quad e \quad b_i = [\vec{V}, \nabla B_i]_{[0,1]^3} \tag{6}$$

Analisando essa equações mais detalhadamente, o objetivo é resolver a função escalar $X = \Re^3 \rightarrow \Re$, minimizando a equação (5), dado um campo vetorial $\vec{V} : \Re^3 \rightarrow \Re^3$. Usando a formulação de Euler-Lagrange, o mínimo é obtido solucionando a equação de Poisson (7).

$$\Delta X = \nabla \cdot \vec{\nabla} \tag{7}$$

Na equação (6), $B_1, ..., B_n : \Re^3 \rightarrow \Re$, é escolhido inicialmente como base, nomeadamente uma coleção de funções B-Splines tri-variadas (geralmente tri-quadráticas) (KAZHDAN MICHAEL, 2013). Mais detalhes podem ser encontrados em (FLETCHER, 1984).

1.6 Triangulação de Delaunay

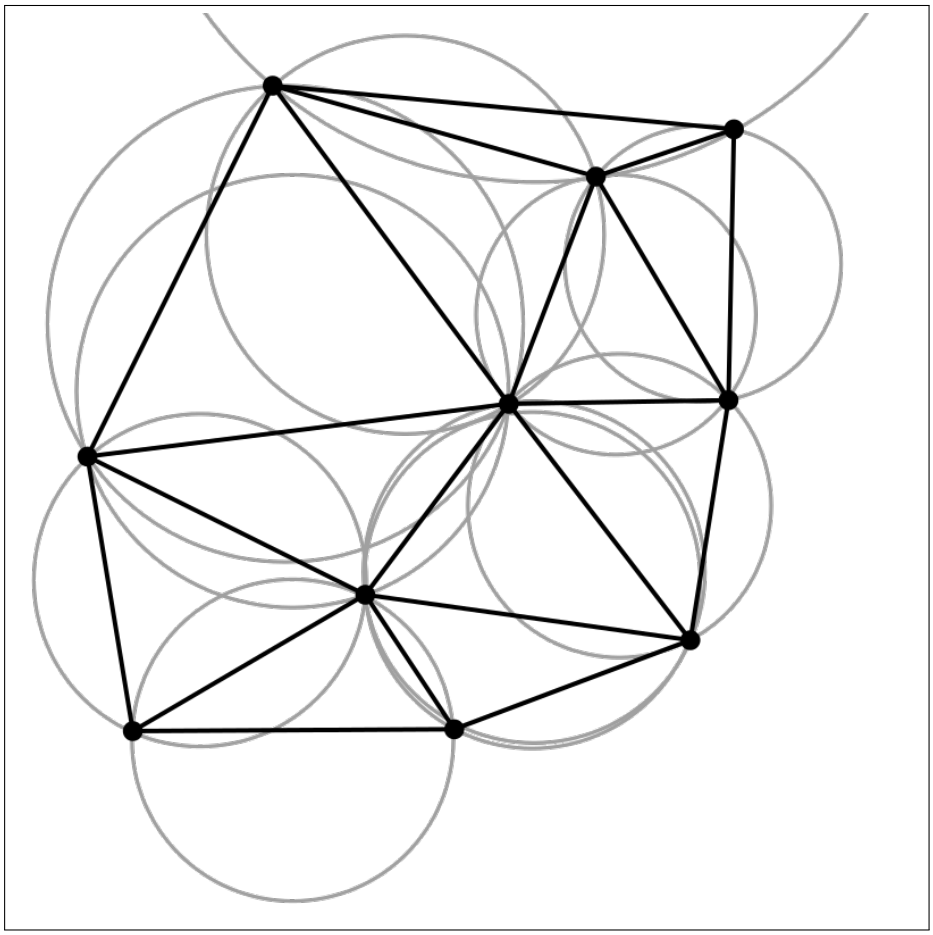
A **triangulação de Delaunay** para um conjunto de pontos **P**, é a triangulação onde nenhum ponto pertencente a **P** está dentro do circuncírculo formado por qualquer triângulo da triangulação (MAGALHÃES; PASSARO; ABE, 2000; PITERI; JUNIOR, 2007), essa condição é chamada de "**condição Delaunay**".

Essa triangulação foi inventada por Boris Delaunay[10] (DELAUNAY et al., 1934). A triangulação de um conjunto de pontos **P** será chamada daqui em diante de DT(**P**). A ideia da triangulação de Delaunay em um plano, junto a condição Delaunay, pode ser observada na Figura 8, onde também é possível observar os circuncírculos formados por cada triângulo.

Uma importante característica da triangulação de Delaunay pode ser observada nos dois triângulos ABD e BCD com a aresta BD em comum (nas Figuras 9a, 9b e 9c), se a soma dos ângulos α e γ for menor ou igual a 180, os triângulos satisfazem a condição de Delaunay. Esta é uma importante propriedade, pois permite o uso da técnica de *flipping* (SHEWCHUK, 2003) (HURTADO; NOY; URRUTIA, 1999) (apenas para duas dimensões). Se dois triângulos não satisfazem a condição de Delaunay, trocando-se a aresta comum BD pela aresta comum AC produzindo dois triângulos que satisfazem a condição Delaunay.

[10] Boris Nikolaevich Delaunay - Matemático russo. Estudou na Universidade de Kiev, de 1909 a 1913, onde doutorou-se. A partir de 1916 foi docente na Escola Politécnica de Kiev e desde 1922 professor em Leningrado. Em 1935 foi professor em Moscou, onde trabalhava desde 1932 no Instituto de Matemática Steklov. Apresentou em 1934 a triangulação de Delaunay.

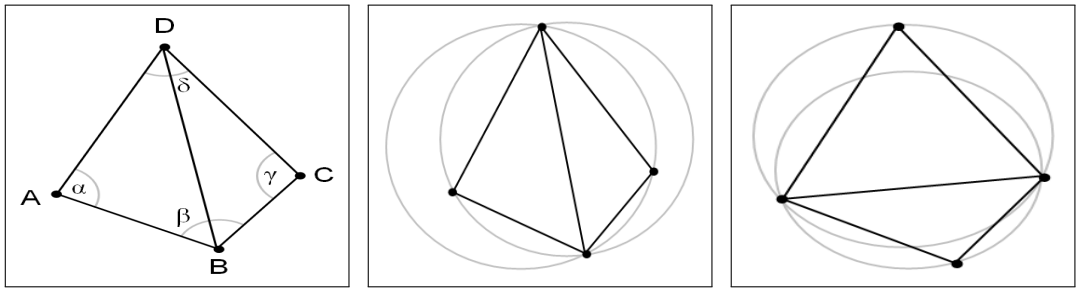
Figura 8 - Ilustração de uma triangulação de Delaunay no plano



Legenda: Ilustração de uma triangulação de Delaunay no plano. É possível observar as circunferências formadas por cada triângulo, onde não podem existir nenhum outro ponto da triangulação em seu interior.

Fonte: https://www.wikipedia.org/ Acesso em 20 de setembro de 2018

Figura 9 - Alteração de arestas entre os pontos



(a) (b) (c)

Legenda: (a) Ilustração de uma triangulação de Delaunay no plano, que não satisfaz a condição de Delaunay. A soma de α e γ é maior que 180∘. (b) Ilustração de uma triangulação de Delaunay no plano, que não satisfaz a condição de Delaunay. As circunferências contêm mais de 3 pontos. (c) Ilustração de uma triangulação de Delaunay no plano. Aplicando *flipping* na aresta comum produz-se uma triangulação de Delaunay para os 4 pontos.

Fonte: ⟨https://www.wikipedia.org/⟩

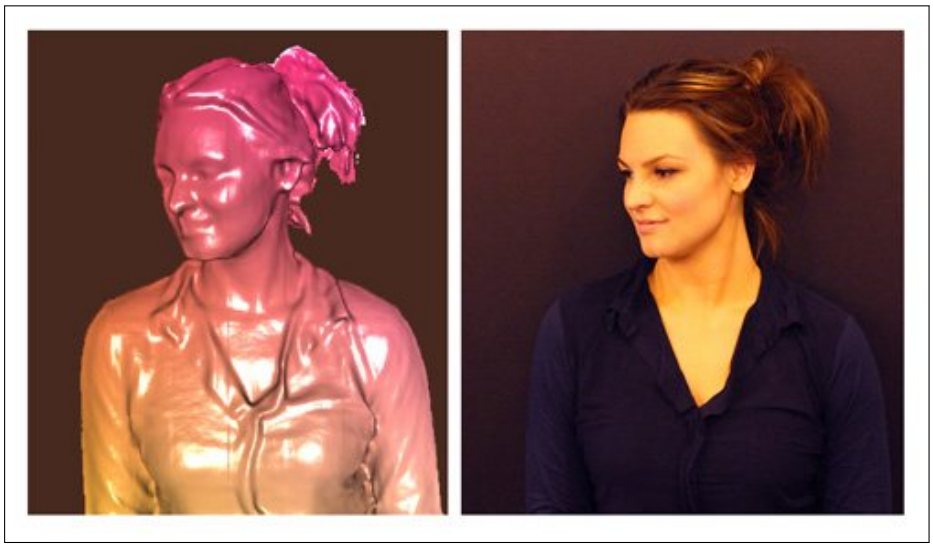
1.7 *Kinect Fusion*

Em adição ao exposto nesta seção, também é importante acrescentar que a *Microsoft* desenvolveu uma biblioteca chamada *Kinect Fusion*, com a capacidade de gerar uma digitalização automática da área escaneada, na forma de nuvens de pontos, em tempo real. Mais detalhes dessa biblioteca e de seu funcionamento, assim como resultados obtidos, podem ser encontrados em (IZADI et al., 2011; BUENO et al., 2015).

A Figura 10 ilustra o modelo de nuvens de pontos que foi obtido com o escaneamento utilizando o *Kinect Fusion*.

Essa biblioteca está disponível apenas para o sistema operacional Microsoft Windows utilizando a biblioteca oficial do sensor, Kinect for Windows SDK 2.0.

Figura 10 - Ilustração de um escaneamento com *Kinect Fusion*



Legenda: Ilustração do escaneamento e geração de um modelo através de nuvem de pontos utilizando a biblioteca *Kinect Fusion* da Microsoft Corporation em uma pessoa.

Fonte: Microsoft Corporation, 2011

2 INFRAESTRUTURA DE SOFTWARE

Conforme mencionado anteriormente, nesse trabalho foi construído um protótipo para realizar a digitalização 3D de objetos utilizando o Kinect One. Este protótipo, implementado na linguagem de programação C++, faz uso de diversas bibliotecas distintas para cobrir as funcionalidades necessárias para o objetivo.

Esta seção trata das diversas bibliotecas que foram utilizadas, oferecendo uma breve explicação sobre elas, suas funcionalidades e sua participação no programa final.

2.1 libfreenect2 0.2

O ***libfreenect2*** é um driver e biblioteca de código fonte aberto para a utilização do *Kinect One* da *Microsoft* em um computador (XIANG et al., 2016). O driver fornece todas as funcionalidades necessárias para o correto funcionamento do dispositivo e também disponibiliza uma biblioteca minimalista para controlar as funcionalidades mais básicas do sensor, como a entrada de imagens com a câmera RGB e *frames* infravermelho com o sensor infravermelho. A documentação da biblioteca está disponível em ⟨https://openkinect.github.io/libfreenect2/⟩ e não oferece algumas descrições de parte de funcionalidades do *Kinect*, como a geração dos modelos 3D a partir da geometrias das duas câmeras combinadas. Isso ocorre devido ao *Kinect* ser um produto comercial fechado pertencente a *Microsoft*.

Esse driver foi desenvolvido por Joshua Blake (SARBOLANDI; LEFLOCH; KOLB, 2015), através de um processo de engenharia reversa. Segundo os testes conduzidos em (SARBOLANDI; KOLB, 2015), essa engenharia reversa não foi feita com perfeição, resultando em um driver que não alcança a capacidade máxima oferecida pelo *Kinect One*, porém próximo disso.

2.2 Meshlab 2016.12

Meshlab é um programa *open source* para o processamento e edição de malhas (*meshes*) 3D e nuvens de pontos (CIGNONI et al., 2008; CORSINI; CIGNONI; SCOPIGNO, 2012). Ele provê uma série de ferramentas de edição, corte, recuperação, inspeção, alteração, renderização, texturização e geração de malhas (CIGNONI; ROCCHINI; SCOPIGNO, 1998). Também oferece ferramentas para digitalização e preparo de modelos para impressão 3D (KAZHDAN; HOPPE, 2013; PIETRONI; TARINI; CIGNONI, 2010).

Este aplicativo foi utilizado no presente trabalho para fazer a junção das nuvens de pontos dos objetos digitalizados, resultando assim em uma única nuvem de pontos final que representa o objeto.

Alguns trabalhos envolvendo essa plataforma podem ser analisados em (CALLIERI et al., 2011; FALKINGHAM, 2013; MENDES; GRIZ; SEDREZ, 2015; SILVESTRE et al., 2012).

O software pode ser encontrado no link ⟨http://www.meshlab.net/#download⟩.

2.3 ARUCO 3.0.6

ARUCO é uma biblioteca de alvos codificados[11] de código fonte aberto (licença BSD) (GARRIDO-JURADO et al., 2014; GARRIDO-JURADO R. MUÑõZ-SALINAS, 2016; GARRIDO-JURADO et al., 2016). Essa biblioteca é bastante flexível e é capaz de representar até 1024 alvos diferentes. Ela é implementada junto a biblioteca *OpenCV*. A Figura 11 mostra exemplos de marcações comumente usadas por essa biblioteca.

Neste trabalho, essa biblioteca foi utilizada para a segmentação e corte das área de interesse de nuvens de pontos. O dicionário específico utilizado foi o $cv :: aruco :: DICT\_4X4\_50$, exemplos de marcações desse dicionário podem ser vistos na Figura 12.

A biblioteca *Aruco* pode ser encontrada em ⟨https://www.uco.es/investiga/grupos/ava/node/26⟩, assim como todos os links para a documentação pertinente.

2.4 PCL 1.7.1 - *Point Cloud Library*

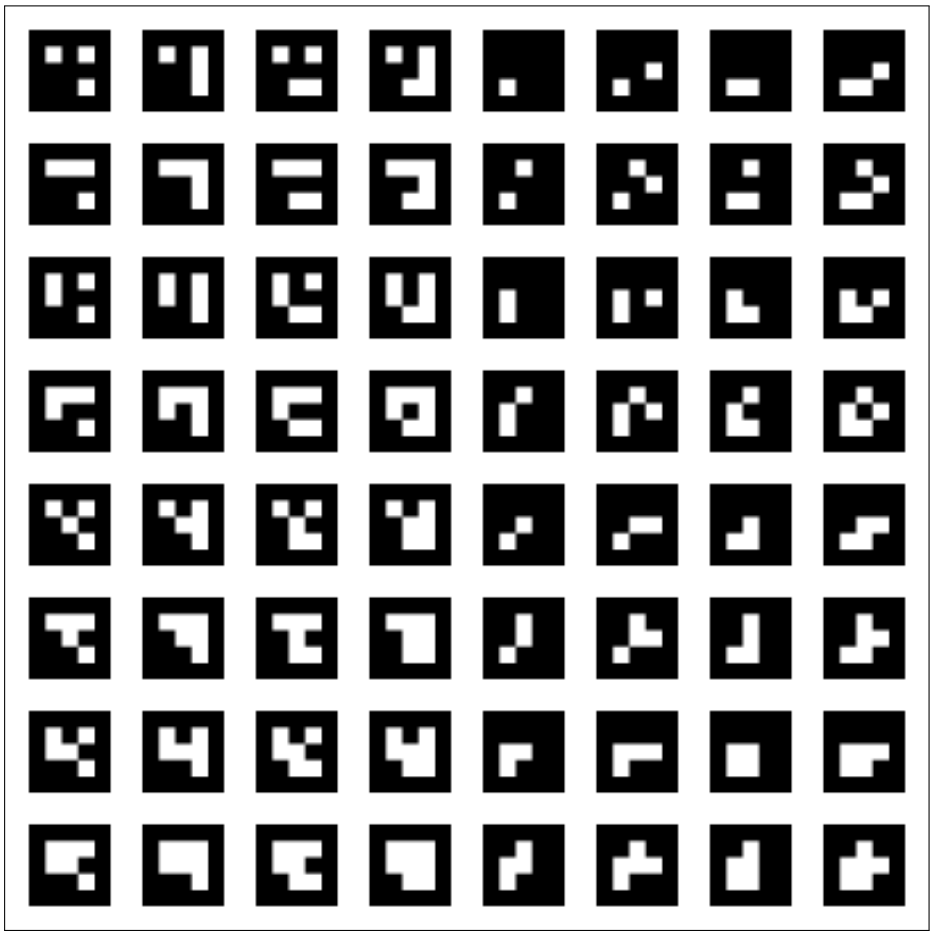
PCL - *Point Cloud Library* é uma biblioteca de código fonte aberto com algoritmos para processamento e manipulação de nuvens de pontos e de geometrias 3D que são muito usados em visão computacional tridimensional (RUSU; COUSINS, 2011). É escrita em C++ e é distribuída sobre a licença BSD.

Essa biblioteca foi utilizada neste trabalho para algumas manipulações e tratamentos de nuvens de pontos obtidas com o Kinect e também será usada para mesclar as nuvens de pontos de um único objeto.

Essa biblioteca pode ser encontrada em ⟨http://pointclouds.org/⟩.

[11] Alvos codificados são figuras pŕe-definidas que possuem dados em sua própria imagem, pois suas dimensões físicas e formatos são previamente conhecidos e padronizados (A.; G.; O., 2014).

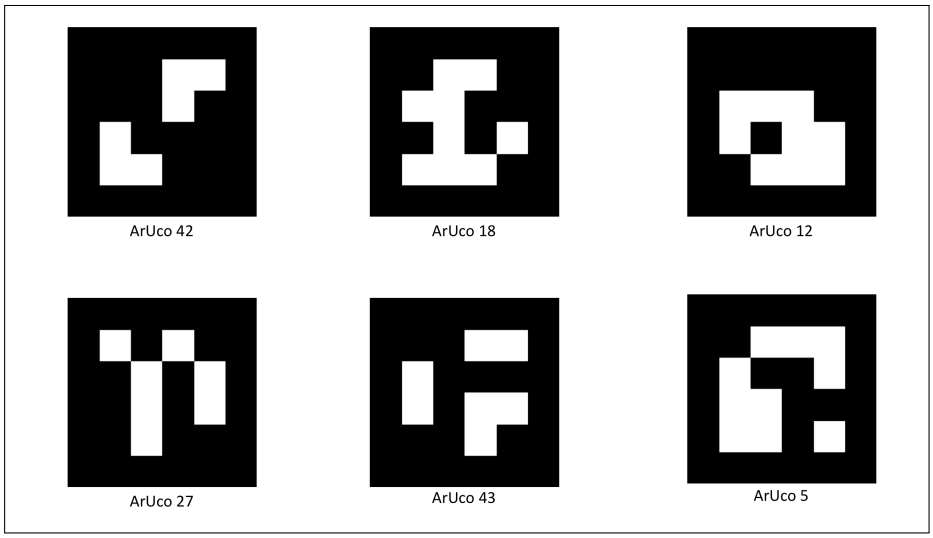
Figura 11 - Exemplo de marcações ARUCO comumente utilizadas



Legenda: Ilustração que exibe diversas marcações Aruco comumente utilizadas em diversas aplicações que utilizam alvos codificados.

Fonte: GARRIDO-JURADO R. MUNõZ-SALINAS, 2016

Figura 12 - Exemplo de marcações com o dicionário $DICT\_4X4\_50$



Legenda: Ilustração que exibe diversas marcações Aruco do dicionário $DICT\_4X4\_50$ que foram utilizadas neste trabalho, assim como o identificador de cada marcação.
Fonte: GARRIDO-JURADO R. MUNõZ-SALINAS, 2016

2.5 **OpenCV 3.4.1**

Por fim, o **OpenCV** é uma biblioteca multiplataforma, totalmente livre para o uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de Visão computacional, também sobre a licença BSD (BRADSKI, 2000; PULLI et al., 2012). Essa biblioteca possui uma grande quantidade de algoritmos para processamento de imagens e visão computacional, fazendo dela uma das bibliotecas mais utilizadas na área. Possui interfaces com as linguagens C/C++, Python e Java, além de conseguir tirar vantagem de processamento paralelo.

Ela foi usada aqui neste trabalha para fazer o tratamento necessário em diversas imagens durante o processo de digitalização. Essa biblioteca supriu diversas funções de entrada e saída de imagens, assim como exibição e realização de pequenas tarefas necessárias que as outras bibliotecas não conseguiam realizar.

Essa biblioteca pode ser encontrada em ⟨https://opencv.org/⟩.

3 MÉTODO

O presente capítulo apresenta o método empregado nesse trabalho que executa uma abordagem direta e objetiva para o processo de digitalização.

3.1 Descrição Geral do Método Proposto

O método utilizado neste trabalho visa a digitalização sistemática e controlada e geração de modelos tridimensionais de objetos com o *Kinect One*.

3.2 Etapas do Processo

O método é dividido em 5 etapas distintas, executadas sequencialmente de forma sistemática até a obtenção do modelo final do objeto digitalizado no final de todo o processo. As etapas são:

3.2.1 Etapa 1: Construção do *Setup*

A primeira etapa a ser executada é a mais simples, a construção do *setup* de digitalização, incluindo a mesa personalizada, o posicionamento das marcações ARUCO, o posicionamento do objeto e o posicionamento do sensor propriamente dito.

Cada marcação possui 14 cm de tamanho de lado e estão dispostas sobre a mesa de forma não simétrica, para evitar qualquer tipo de oclusão das marcações por parte do objeto digitalizado ou de algum ângulo simétrico em relação a mesa que o sensor, por falta de sorte, possa ter assumido durante o processo de digitalização.

As marcações também estão posicionadas na mesa de forma a serem utilizadas como pontos de controle para as etapas seguintes. Uma das marcações é definida como origem do sistema objeto (0,0,0) e então a orientação dos eixos é definida. As demais marcações são medidas em função da origem com precisão milimétrica, tentando evitar ao máximo erros humanos e manuais nessa etapa, pois se propagariam ao longo de todo o processo.

O objeto é posicionado no centro de 4 marcações e o sensor a 60 cm de distância desse objeto. As Figuras 13, 14 e 15 ilustram esse esquema do *setup* com diferentes ângulos. Para uma segunda tomada de experimentos, o objeto será posicionado como nas figuras anteriores, porém o sensor estará a 120 cm do objeto.

Figura 13 - Exemplo de *setup* de digitalização completo

Legenda: Figura que ilustra o *setup* de digitalização completo, incluindo o sensor *Kinect One* a 60 cm do objeto, o objeto propriamente dito e a mesa personalizada.

Fonte: O autor, 2018

Figura 14 - Exemplo de *setup* de digitalização completo

Legenda: Figura que ilustra o *setup* de digitalização completo, incluindo o sensor Kinect One a 60 cm do objeto, o objeto propriamente dito e a mesa personalizada.

Fonte: O autor, 2018

Figura 15 - Exemplo de *setup* de digitalização completo

Legenda: Figura que ilustra o *setup* de digitalização completo, incluindo o sensor *Kinect One* a 60 cm do objeto, o objeto propriamente dito e a mesa personalizada.

Fonte: O autor, 2018

3.2.2 Etapa 2: Captura

Essa etapa trata da aquisição inicial das informações (frames) referentes ao cenário e objeto que será digitalizado.

Com o *setup* de digitalização já construído, e o objeto de interesse já devidamente posicionado, essa etapa do processo de digitalização trata da digitalização propriamente dita. O Kinect é então ligado a um computador e é posicionado de forma que as marcações e o objeto fiquem dentro do campo de visão das câmeras do sensor, possibilitando que as marcações sejam detectadas para a delimitação da área de interesse e o objeto digitalizado.

Os frames capturados pelo sensor serão enviados para o programa que irá processá-los na etapa seguinte, porém, com o intuito de obter um modelo 3D do objeto com alta exatidão, o Kinect é posicionado em vários ângulos diferentes na cena em relação ao objeto, cobrindo quase que totalmente a superfície do objeto e aumentando a exatidão do modelo gerado. Assim sendo, frames de todas essas poses diferentes são capturados e enviados para o processamento. Inicialmente o sensor é rotacionado em 90 graus horizontalmente em relação ao objeto de interesse, posteriormente a rotação se repete até que quase toda a superfície do objeto já tenha sido coberta pelo sensor, resultando em 4 frames diferentes de cada objeto de teste, cada um deles rotacionado em 90 graus.

Essa etapa é executada por um programa que utiliza o driver libfreenect2 e as bibliotecas OpenCV e PCL.

3.2.3 Etapa 3: Detecção de Marcações

Nessa terceira etapa, com todos os frames já adquiridos, as marcações ARUCO são detectadas. Possibilitando a detecção da mesa e do objeto, assim como o estabelecimento das marcações como pontos de controle.

Um arquivo é gerado nessa etapa, para cada uma das quatro imagens de entrada, contendo os valores dos pixels onde se encontram todas as marcações detectadas. Esse arquivo será lido pela etapa seguinte.

Um programa separado executa toda essa etapa. Ele utiliza as bibliotecas ARUCO e OpenCV.

3.2.4 Etapa 4: Segmentação

Um outro programa resolvere a etapa 4 para a segmentação. Este programa recebe como entrada o arquivo com as marcações detectadas que foi gerado pelo programa de detecção da etapa anterior e a nuvem de pontos inicial capturada pelo *Kinect* na etapa

Figura 16 - Exemplo de sistema de coordenadas usado pelo *Kinect One*

Legenda: Figura que ilustra a disposição do sistema de coordenadas do *Kinect One*.
Fonte: Microsoft Corporation, 2014

2. A partir da detecção das marcações, é criada uma área de interesse contendo apenas os pontos que compõe o objeto. Primeiramente, é identificado a marcação que está mais a direita do objeto, assim como a marcação que está mais a esquerda, mais acima e mais abaixo, que a partir deste momento passarão a ser chamados de **limite superior**, **limite inferior**, **limite direito** e **limite esquerdo**, respectivamente. Em seguida, planos são criados através desses 4 pontos limites, qualquer ponto da nuvem de pontos original que estiver à direita do **limite direito** é descartado. O mesmo é válido para os pontos acima do **limite superior**, pontos abaixo do **limite inferior** e pontos que estejam mais a esquerda do **limite esquerdo**.

Os limites simulam uma caixa ao redor do objeto de interesse para que seja possível descartar todo o resto da nuvem de pontos. A Figura 16 exibe o sistema de coordenadas utilizado pelo Kinect One para facilitar a compreensão desse corte e desse limites impostos.

Após o corte pelos limites laterais do objeto, é preciso fazer a remoção dos pontos pertencentes a mesa digitalizadora, para isso é criado uma equação do plano utilizando 3 dos pontos das marcações ARUCO detectadas na mesa (3 dos 4 pontos usados como limites laterais até então). A quação do plano criada a partir desses pontos segue o formato da equação (8).

$$a(X - X_0) + b(Y - Y_0) + c(Z - Z_0) = 0 \tag{8}$$

Onde a, b e c representam os coeficientes do vetor normal ao plano, X_0, Y_0 e Z_0 são as coordenadas de um ponto conhecido pertencente ao plano e que é a interseção do vetor normal com o próprio plano, por fim, X, Y e Z são as coordenadas de todos os pontos pertencentes ao plano.

Uma vez que o plano tenha sido construído a partir das marcações, é possível comparar a coordenada z de cada um dos pontos restantes da nuvem com a coordenada z dos respectivos pontos pertencentes ao plano. O ponto da nuvem é excluído no caso em que sua coordenada z é maior que a mesma coordenada do respectivo ponto pertencente ao plano. Essa comparação é possível mediante uma modificação na equação do plano que isola a coordenada z do plano. A modificação pode ser vista na equação (9).

$$Z = \frac{-a(X - X_0) - b(Y - Y_0) + cZ_0}{c} \tag{9}$$

Quanto à parte superior do objeto, assume-se pontos pertencentes a toda parte superior do objeto até a origem do sistema de coordenadas, assim sendo, os pontos não são limitados na região que vai dessa parte do sensor até o objeto propriamente dito. Isso ocorre devido ao fato de que em condições normais, como as condições controladas para execução dos experimentos, não haverá nenhum obstáculo ou interferência entre o sensor e o objeto digitalizado, tornando desnecessária a remoção de pontos nesta região.

O resultado da segmentação na nuvem de pontos é um pequeno pedaço central contendo apena o objeto de interesse, descartando todo o resto.

Essa etapa é executada para cada uma das 4 nuvens de pontos, das 4 posições diferentes do sensor, assim sendo, a saída desse programa são 4 nuvens de pontos, que representa as faces do objeto que está sendo digitalizado no momento.

Esse programa utiliza as bibliotecas OpenCV e PCL.

3.2.5 Etapa 5: Alinhamento

A quinta etapa recebe como entrada as 4 nuvens de pontos de um único objeto da etapa anterior. As 4 nuvens já sofreram o processo de segmentação e cada uma delas representa uma das faces do objeto que está sendo digitalizado. Esta última etapa visa alinhar todas essa nuvens de pontos para que seja obtida uma única nuvem de ponto do objeto em questão. Essa nuvem de pontos final é usada para validação da exatidão da nuvem completa do objeto, assim como as nuvens de pontos de cada uma das faces do objeto digitalizado.

Essa é a única etapa de todo o processo que não é automatizada por um programa desenvolvido durante este trabalho. Ela é executada manualmente no *Meshlab*, pois é uma plataforma bastante completa para esse tipo de fusão de nuvens de pontos, além de

possuir várias ferramentas para evitar ruídos e *outliers*[12].

O *Meshlab* utiliza internamento, durante o processo de alinhamento das nuvens de pontos, o algoritmo conhecido como **(*Iterative closest point*)** (BESL; MCKAY, 1992; CHEN; MEDIONI, 1992). Esse algoritmo é empregado para minimizar a diferença entre duas nuvens de pontos e é comumente usado na reconstrução de superfícies 2D e 3D (JIN et al., 1995) (NIESSNER; DAI; FISHER, 2014), navegação de robôs (YOSHITAKA et al., 2006; ZHOU; LIN, 2011), entre outras aplicações.

Todo o processo, em suas 5 etapas descritas acima, é executado para cada um dos 5 objetos de teste, então é repetido novamente variando-se a distância que o sensor se encontra do objeto, 60 cm para 120 cm, por fim é repetindo novamente variando a quantidade de pontos nas nuvens de pontos de referência de cada objeto, necessárias para uma comparação posterior.

[12] Valor aberrante ou valor atípico. É uma observação que apresenta um grande afastamento das demais da série, ou que é uma observação inconsistente (HAWKINS, 1980). A existência de *outliers* implica, tipicamente, em prejuízos à interpretação dos resultados dos testes estatísticos aplicados às amostras. (GLADWELL; KORYTOWSKI,)

4 PROCEDIMENTO EXPERIMENTAL, RESULTADOS E ANÁLISE

Esse capítulo trata do procedimento experimental conduzido durante a implementação do método proposto, incluindo todas as suas etapas e detalhes. Também exibe todos os resultados obtidos nesse trabalho durante todo o processo de digitalização com os objetos escolhidos para teste, assim como a análise dos mesmos.

Esse trabalho apresenta um cenário único para a avaliação e validação dos resultados. Envolvendo a utilização do driver libfreenect2, obtido através de engenharia reversa, dos objetos geométricos selecionados para teste e de toda a disposição do *setup*.

Vários componentes são necessários para a execução dos experimentos. O componente mais importante, evidentemente, é o Kinect One. Além do Kinect, também são necessários a mesa personalizada onde estarão dispostas as marcações ARUCO, o objeto que será digitalizado, o computador ao qual o Kinect estará ligado e o programa que fará todo o processamento dos frames capturados pelo Kinect e produzirá o modelo 3D.

O Kinect deverá permanecer fixado a um tripé devidamente posicionado de forma que o campo de visão do sensor contenha toda a mesa personalizada com as marcações e o objeto de interesse no centro da mesma. Também estará conectado ao computador com o programa que fará o processamento.

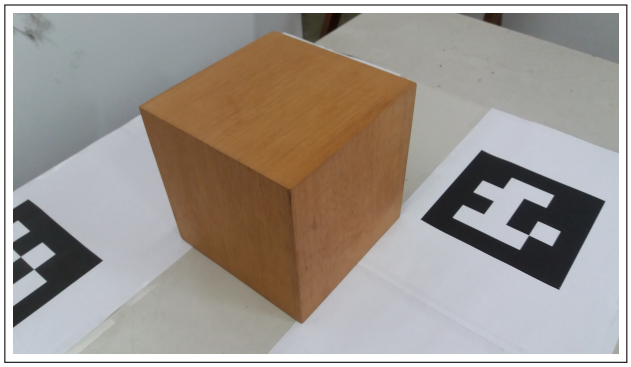
O programa que faz a aquisição dos frames do objeto na verdade foi dividido em 4 programas distintos. Isso foi feito com o objetivo de reduzir os requisitos de bibliotecas do programa, tornando-o mais simples e tratável (uma vez que o programa final utilizaria diversas bibliotecas e drivers), facilitando a execução de todas as etapas ao longo de todo o processo.

4.1 Descrição dos Objetos

Os objetos que foram testados são um cubo de 15 cm de lado, de madeira, ilustrado na Figura 17, uma bola de sinuca de 54 mm de diâmetro, tamanho padrão das bolas de sinuca usadas no Brasil[13], ilustrada na Figura 18 e um paralelepípedo de madeira maciça, de dimensões 20 cm, 13,1 cm e 5,4 cm, que foi utilizado em 3 posições diferentes para que fosse possível simular 3 objetos distintos, ele pode ser observado nas Figuras 19, 20 e 21. A digitalização desses 5 objetos foi feita através do posicionamento do sensor a 60 cm e 120 cm de distância do próprio objeto.

[13] Segundo a empresa SALUC (⟨http://www.saluc.com/html/home.htm⟩), da Bélgica, e aprovadas pela CBBS (Confederação Brasileira de Bilhar e Sinuca).

Figura 17 - Exemplo de cubo



Legenda: Ilustração que exibe o cubo de madeira de 15 cm de lado que foi utilizado como objeto de teste nesse trabalho.
Fonte: O autor, 2018

Figura 18 - Exemplo de esfera

Legenda: Ilustração que exibe a esfera (bola de sinuca) de 54 mm de diâmetro que foi utilizado como objeto de teste nesse trabalho.
Fonte: O autor, 2018

Figura 19 - Exemplo de paralelepípedo de frente

Legenda: Ilustração que exibe o paralelepípedo de madeira maciça, de frente, que foi utilizado como objeto de teste nesse trabalho.

Fonte: O autor, 2018

Figura 20 - Exemplo de paralelepípedo achatado

Legenda: Ilustração que exibe o paralelepípedo de madeira maciça, achatado, que foi utilizado como objeto de teste nesse trabalho.

Fonte: O autor, 2018

Figura 21 - Exemplo de paralelepípedo alto

Legenda: Ilustração que exibe o paralelepípedo de madeira maciça, alto, que foi utilizado como objeto de teste nesse trabalho.
Fonte: O autor, 2018

A escolha de 5 figuras geométricas simples, para objetos de teste, foi feita porque dessa forma é possível a geração de **nuvens de pontos de referências** com o *Meshlab*, permitindo a comparação e verificação da exatidão entre as nuvens de referência e as nuvens obtidas.

4.2 Mesa Digitalizadora

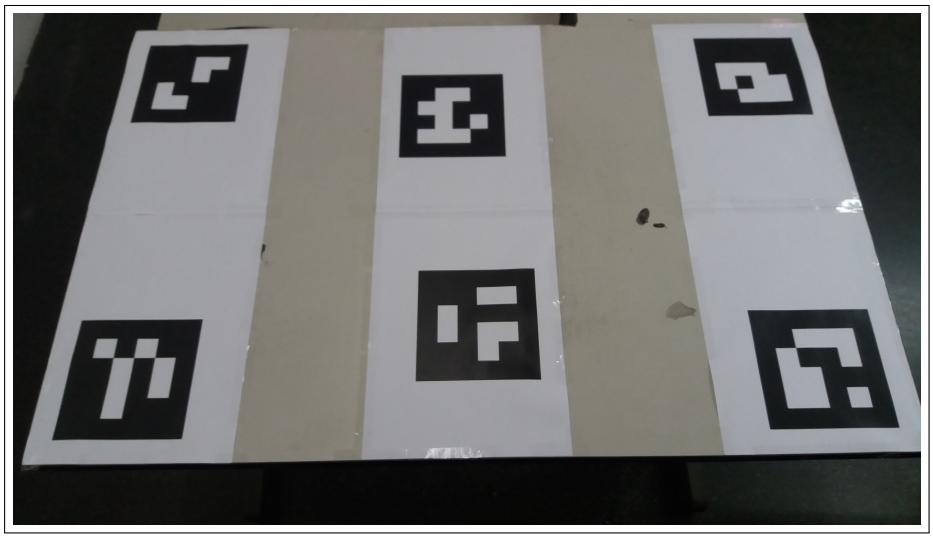
Além dos objetos utilizados, também foi construído uma mesa personalizada contendo marcações ARUCO. As marcações da mesa são usadas para o corte e segmentação das nuvens de pontos, assim como auxiliam na fusão das nuvens para o modelo final. As marcações funcionam como pontos de controle na mesa. A Figura 22 ilustra a mesa que foi usada para digitalização.

O dicionário das marcações utilizadas é o $cv :: aruco :: DICT\_4X4\_50$. Todo o *setup* de digitalização com a mesa e o objeto pode ser visto nas Figuras 13, 14 e 15.

4.3 Captura

De forma completa, os *frames* capturados pelo sensor durante a etapa 2 do método (captura) são os seguintes:

Figura 22 - Exemplo de mesa usada na digitalização



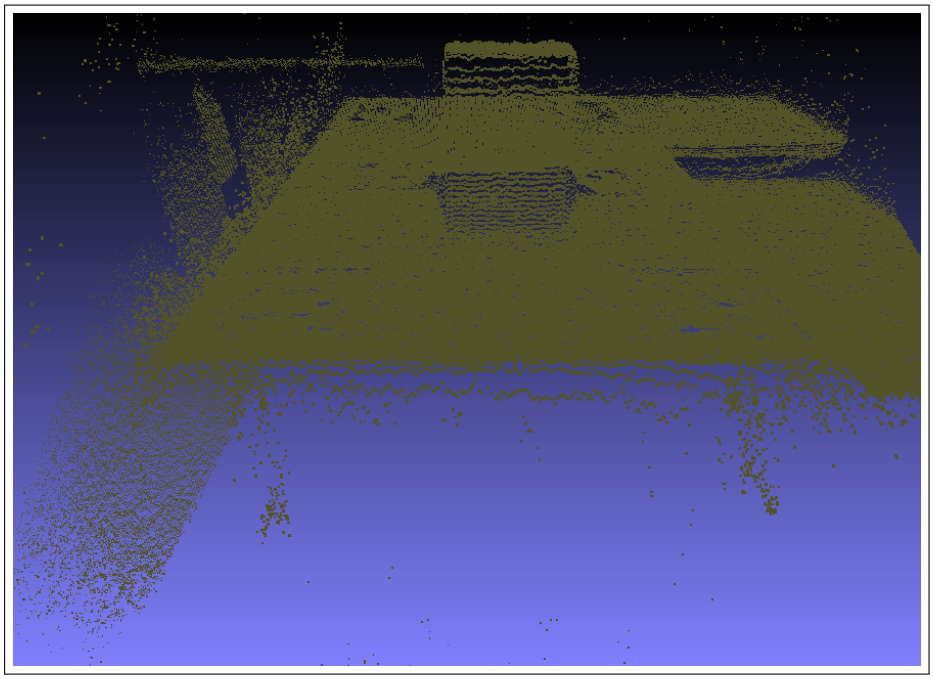
Legenda: Ilustração que exibe a mesa digitalizadora usada para digitalizar os 5 objetos. Nota-se as 6 marcações ARUCO especificamente posicionadas sobre a mesa.
Fonte: O autor, 2018

- frame RGB, obtido da câmera RGB *fullHD* (1920x1080). É usado na construção do frame de registro, descrito mais a frente;
- frame infravermelho, obtido pela câmera infravermelho;
- frame de profundidade obtido pela geometria interna do Kinect e frame infravermelho. Mais detalhes da tecnologia tempo de voo podem ser encontrados na seção 1.2;
- frame de profundidade sem distorções de lentes. O Kinect utiliza sua calibração interna para remover as distorções de suas próprias lentes;
- frame de profundidade sem distorções de lentes e com textura. Também utilizando sua geometria interna, o Kinect gera o mesmo frame de profundidade sem distorções anterior com adição de texturas construídas com o frame RGB.

A Figura 23 exibe as nuvens de pontos adquirida inicialmente sem qualquer tipo de processamento por parte dos programas. Ela representa a nuvem de pontos da cena mostrada na Figura 24.

Como mencionado anteriormente, a captura também será feita com duas distância diferentes do sensor ao objeto, visando avaliar o desempenho nos dois cenários diferentes. As distâncias são 60 cm e 120 cm.

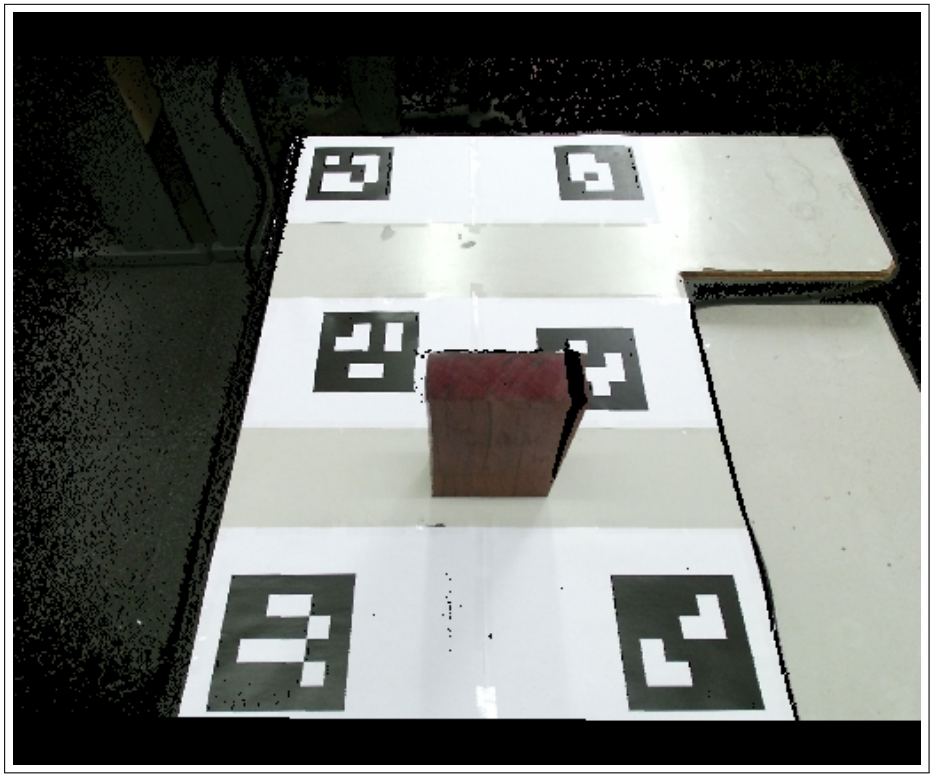
Figura 23 - Exemplo de nuvem de pontos inicial



Legenda: Figura que ilustra uma nuvem de pontos de uma digitalização inicial, sem qualquer tipo de manipulação.

Fonte: O autor, 2018

Figura 24 - Exemplo de cena inicial de digitalização



Legenda: Figura que ilustra a cena inicial de digitalização. A imagem é oriunda do frame de registro do Kinect One.

Fonte: O autor, 2018

4.4 Distância de Hausdorff

A **Distância de Hausdorff** foi utilizada nesse trabalho como método de validação para a verificação da exatidão dos modelos 3D finais obtidos pelo programa de digitalização. A comparação é feita em oposição a outras nuvens de pontos consideradas ideais dos objetos de teste que foram usadas como referência para efetuar todos os cálculos necessários.

A Distância de Hausdorff, ou Métrica de Hausdorff, também chamada de **Distância de Pompeiu–Hausdorff** (ROCKAFELLAR; WETS, 2015), calcula o quão distante dois conjunto de pontos estão um do outro. Dois conjuntos estão próximos do ponto de vista da distância de Hausdorff se cada ponto do primeiro conjunto também está próximo de algum ponto do segundo conjunto e reciprocamente. A Distância de Hausdorff é a maior de todas as distâncias que é preciso trafegar de um ponto do primeiro conjunto para o ponto mais próximo do segundo conjunto.

Essa distância foi introduzida por Felix Hausdorff[14] em seu livro *Grundzüge der Mengenlehre*[15] (BLUMBERG, 1920; HAUSDORFF, 1914), publicado pela primeira vez em 1914, embora um texto muito similar tenha aparecido na tese de doutorado de Maurice Fréchet[16] em 1906, em seu estudo do espaço das curvas contínuas de $[0,1] \rightarrow \Re^3$.

Quanto a definição, sejam X e Y dois conjuntos não vazios, a Distância de Hausdorff $d_H(X,Y)$ pode ser definida por (ROCKAFELLAR; WETS, 2015; ASPERT; SANTA-CRUZ; EBRAHIMI, 2002):

$$d_H = max \quad \{sup \quad inf \quad d(x,y) \quad , \quad sup \quad inf \quad d(y,x)\} \tag{10}$$

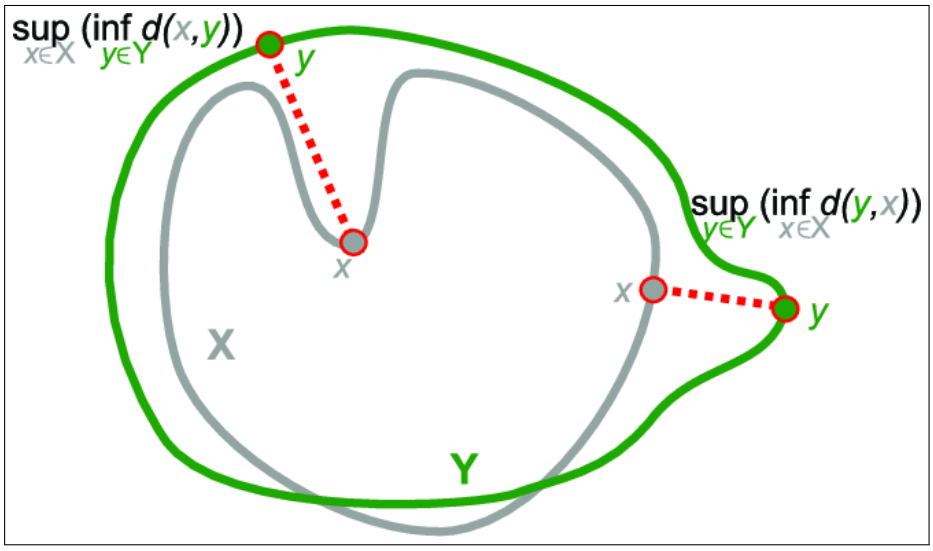
Onde *sup* representa o supremo e *inf* o ínfimo. Alternativamente, de forma mais simples, a Distância de Hausdorff também pode ser definida por (JESORSKY; KIRCH-

[14] Felix Hausdorff (8 de novembro de 1868 - 26 de janeiro de 1942) foi um matemático alemão considerado um dos fundadores da topologia moderna e que contribuiu significativamente para a teoria dos conjuntos, teoria dos conjuntos descritivos, teoria das medidas, teoria das funções e análise funcional.

[15] *Grundzüge der Mengenlehre* (alemão para *Basics of Set Theory*) é um influente livro sobre teoria dos conjuntos escrito por Felix Hausdorff. Foi a primeira introdução abrangente à teoria dos conjuntos. Além do tratamento sistemático de resultados conhecidos na teoria dos conjuntos, o livro também contém capítulos sobre teoria de medida e topologia, que ainda eram considerados partes da teoria dos conjuntos na época.

[16] Maurice Fréchet (2 de setembro de 1878 - 4 de junho de 1973) foi um matemático francês. Ele fez grandes contribuições para a topologia de conjuntos de pontos e introduziu todo o conceito de espaços métricos. Ele também fez várias contribuições importantes para o campo da estatística e probabilidade, bem como cálculo.

Figura 25 - Exemplo da distância de Hausdorff



Legenda: Ilustração que exibe a Distância de Hausdorff a ser calculada em dois conjuntos X e Y

Fonte: https://www.researchgate.net/figure/Hausdorff-distance-between-two-lines-X-and-Y-supF-is-the-superior-bound-of-F-and$_fig16_280881453$

BERG; FRISCHHOLZ, 2001):

$$H(X,Y) = max(h(X,Y), h(Y,X)) \tag{11}$$

$$h(X,Y) = max \quad min||a-b||, \quad com \quad a \in X \quad e \quad b \in Y \tag{12}$$

Desta forma, *h(X,Y)* é chamada Distância Dirigida de Hausdorff do conjunto X para Y (JESORSKY; KIRCHBERG; FRISCHHOLZ, 2001).

Portanto, para cada ponto do primeiro conjunto X é calculado a distância até o ponto mais próximo do segundo conjunto Y. Então a distância de X para Y é definida como a maior das distâncias calculadas de todos esses pontos. O processo também é executado na direção contrária, de Y para X, e então o maior valor de Y para X também é armazenado. A Distância de Hausdorff final entre os dois conjuntos é a maior dentre as duas distâncias calculadas nas duas direções.

A Figura 25 ilustra essa ideia da Distância de Hausdorff e da diferença dos dois conjuntos.

4.5 Métricas Adicionais

Em adição a distância de Hausdorff já exposta, medidas adicionais derivadas também são verificadas com o intuito de obter uma melhor compreensão dos resultados. Essas medidas incluem: Distância de Hausdorff Mínima; Valor Médio; Desvio Padrão; Área; Volume.

A distância de Hausdorff Mínima, descrita pela equação (13), é modificada da distância de Hausdorff comum. Essa distância representa a menor distância possível medida de um ponto da primeira nuvem de pontos para o ponto mais próximo da segunda nuvem de pontos.

$$d_H = min \quad \{inf \quad inf \quad d(x,y) \quad , \quad inf \quad inf \quad d(y,x)\} \tag{13}$$

O valor médio calculado é a média aritmética de todas as distância encontradas de cada ponto da nuvem.

Para avaliar o desvio padrão, primeiro é preciso compreender o conceito de variância: Ela é a medida de dispersão que mostra o quão distante cada valor do conjunto de distâncias calculadas encontra-se do valor médio (MONTGOMERY; RUNGER, 2010; TRIOLA, 2005). Quanto menor é a variância, mais próximo os valores estão do valor médio. Inversamente, quanto maior for a variância, mais os valores estão distantes do valor médio.

O desvio padrão é obtido através da raiz quadrada positiva da variância. Ele representa o quão confiável é o valor medido e informa o erro em um conjunto de dados, caso fosse substituído um dos valores do conjunto pela média aritmética (MONTGOMERY; RUNGER, 2010; TRIOLA, 2005).

A área e o volume dos objetos de teste correspondem a aplicação padrão dessa medidas e uma vez obtidas nas nuvens capturadas pelo *kinect*, serão comparadas com cada um de seus respectivos valores de referência obtidos através do tamanho real dos objetos.

4.6 Análise dos Resultados

A última etapa de todo o processo de digitalização é a validação dos resultados obtidos. O programa final recebe como entrada a nuvem de pontos final do objeto, que foi gerada pela etapa anterior, de alinhamento, e a compara com uma nuvem de pontos de referência do respectivo objeto, que foi gerada pelo próprio Meshlab. Através dessa comparação, é possível aplicar a distância de Hausdorff e verificar o quão exata a nuvem final é. Assim sendo, é obtido como resultado final a distância de Hausdorff da nuvem

final para a nuvem de referência.

A comparação também será feita com uma nuvem de pontos de referência de 60.008 pontos e 10.008 pontos, para os objetos cubo e paralelepípedos, visando avaliar dois cenários de referências diferentes, para cada uma das tomadas de todos os objetos de teste. O objeto esfera será avaliado com nuvens de referência contendo 642 pontos e 10.008 pontos, isso é feito devido à bola de sinuca utilizada como objeto esférico ser bem menor em comparação com os demais objetos de teste.

A nuvem de pontos final do objeto está disponível no formato .PCD[17], pelo programa, ao término de todo o processo. A partir daí ele poderá ser manipulado novamente pelo usuário ou aberto em outros programas para processamento de nuvens de pontos, para os mais variados fins.

Os resultados dos objetos testados são anotados e verificados em comparação a suas respectivas nuvens de pontos de referência (nuvens com 60.008 e 10.008 pontos para cubo e paralelepípedos e 642 e 10.008 pontos para esfera) através do método de validação da distância de Hausdorff e variações, descrita anteriormente.

A Tabela 2 exibe todos os resultados do cálculo da distância de Hausdorff, distância de Hausdorff Mínima, erro médio e desvio padrão para os objetos cubo e paralelepípedos em comparação com suas respectivas nuvens de pontos de referência. O sensor foi posicionado a 60cm dos objetos durante o processo de digitalização e as nuvens de pontos de referência possuem 60.008 pontos para o cubo e os paralelepípedos.

A tabela 3 exibe os resultados para a esfera, utilizando, porém, 642 pontos para a nuvem de referência. As demais características permanecem similares aos objetos anteriores, 60cm de distância do sensor a esfera e cálculo da distância de Hausdorff, distâncoa de Hausdorff mínima, erro médio e desvio padrão.

Ainda mantendo o sensor posicionado a 60cm de distância dos objetos, a Tabela 4 exibe os resultados da distância de Hausdorff variando a quantidade de pontos da nuvem de referência para 10.008, para todos os objetos.

Avançando para os resultados obtidos com o sensor posicionado a 120 cm de distância dos objetos, a Tabela 5 exibe esses resultados utilizando 60.008 pontos para compor a nuvem de pontos de referência dos objetos cubo, paralelepípedos 1, 2 e 3. A tabela 6 apresentado os resultados com a esfera, cuja nuvem de referência possui 642 novamente.

A Tabela 7 exibe os resultados obtidos também a uma distância de 120cm dos objetos, porém com 10.008 pontos para as nuvens de pontos de referência para todos os objetos.

As Tabelas 2, 4, 5 e 7 estão organizadas de forma que a primeira coluda identifica

[17] Formato padrão para nuvens de pontos da biblioteca PCL.

Tabela 2 - Resultados com 60 cm e 60.008 pontos

Objeto	**Distância de Hausdorff**	**Mínimo**	**Erro Médio**	**Desvio Padrão**
Cubo	68,1 mm	0,01 mm	6,0 mm	6,7 mm
Paralelepípedo 1	26,9 mm	0,003 mm	2,8 mm	3,2 mm
Paralelepípedo 2	27,0 mm	0,009 mm	6,8 mm	7,2 mm
Paralelepípedo 3	21,9 mm	0,005 mm	2,9 mm	3,0 mm

Legenda: Tabela com os resultados dos objetos cubo e paralelepípedos 1, 2 e 3 com o sensor posicionado a 60 cm de distância e com 60.008 pontos para cada uma das nuvens de referência dos objetos.

Fonte: O autor, 2018

Tabela 3 - Resultados com 60cm e 642 pontos para o objeto esfera

Objeto	**Distância de Hausdorff**	**Mínimo**	**Erro Médio**	**Desvio Padrão**
Esfera	18,9 mm	0,07 mm	3,9 mm	3,7 mm

Legenda: Tabela com os resultados do objeto esfera com o sensor posicionado a 60 cm de distância e com 642 pontos na nuvem de referência.

Fonte: O autor, 2018

Tabela 4 - Resultados com 60 cm e 10.008 pontos

Objeto	**Distância de Hausdorff**	**Mínimo**	**Erro Médio**	**Desvio Padrão**
Cubo	68,1 mm	0,01mm	6,3 mm	6,5 mm
Esfera	45,9 mm	22,5 mm	29,8 mm	4,3 mm
Paralelepípedo 1	26,9 mm	0,05 mm	3,2 mm	3,0mm
Paralelepípedo 2	27,0 mm	0,08 mm	7,0 mm	7,0 mm
Paralelepípedo 3	21,9 mm	0,04 mm	3,2 mm	2,9 mm

Legenda: Tabela com os resultados de cada um dos objetos de teste com o sensor posicionado a 60 cm de distância e com 10.008 pontos para cada uma das nuvens de referência dos objetos.

Fonte: O autor, 2018

Tabela 5 - Resultados com 120 cm e 60.008 pontos

Objeto	**Distância de Hausdorff**	**Mínimo**	**Erro Médio**	**Desvio Padrão**
Cubo	84,3 mm	0,02 mm	13,9 mm	6,8 mm
Paralelepípedo 1	39,8 mm	0,1 mm	5,9 mm	3,4 mm
Paralelepípedo 2	40,2 mm	0,1 mm	15,9 mm	7,4 mm
Paralelepípedo 3	32,8 mm	0,1 mm	6,0 mm	3,2 mm

Legenda: Tabela com os resultados dos objetos cubo, paralelepípedos 1, 2 e 3 com o sensor posicionado a 120 cm de distância e com 60.008 pontos para cada uma das nuvens de referência dos objetos.

Fonte: O autor, 2018

Tabela 6 - Resultados com 120 cm e 642 pontos para o objeto esfera

Objeto	**Distância de Hausdorff**	**Mínimo**	**Erro Médio**	**Desvio Padrão**
Esfera	78,9mm	0,3 mm	66,7 mm	7,1 mm

Legenda: Tabela com os resultados do objeto esfera com o sensor posicionado a 120 cm de distância e com 642 pontos para a nuvem de referência.

Fonte: O autor, 2018

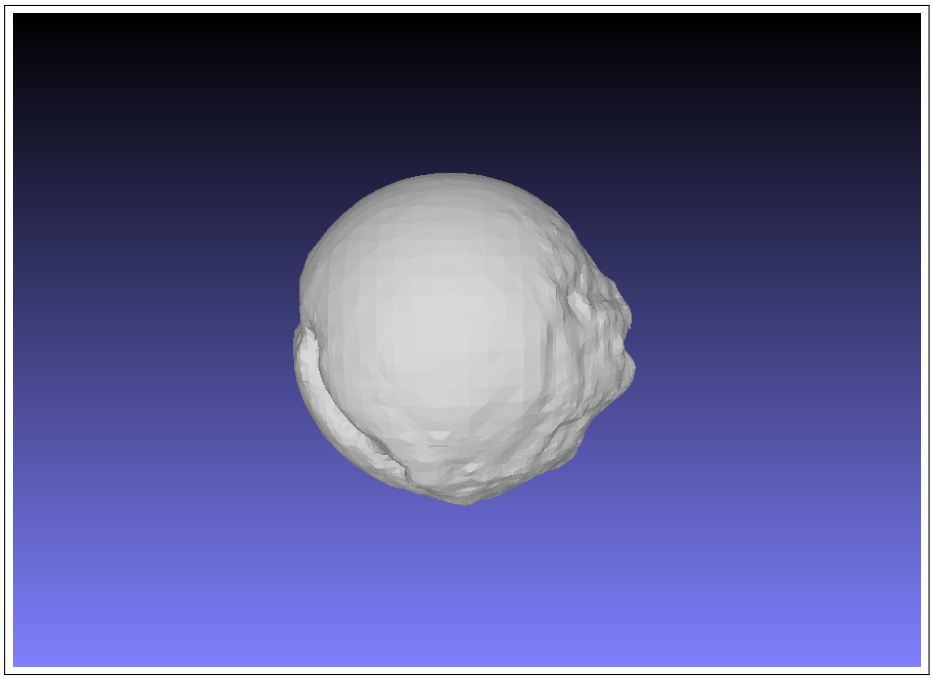
Tabela 7 - Resultados com 120cm e 10.008 pontos

Objeto	**Distância de Hausdorff**	**Mínimo**	**Erro Médio**	**Desvio Padrão**
Cubo	84,9mm	0,1mm	14,9mm	4,5mm
Esfera	79,2mm	22,9mm	67,7mm	5,3mm
Paralelepípedo 1	40,1mm	0,1mm	6,2mm	1,0mm
Paralelepípedo 2	40,7mm	0,1mm	15,9mm	5,1mm
Paralelepípedo 3	33,3mm	0,1mm	7,2mm	1,1mm

Legenda: Tabela com os resultados de cada um dos objetos de teste com o sensor posicionado a 120 cm de distância e com 10.008 pontos para cada uma das nuvens de referência dos objetos.

Fonte: O autor, 2018

Figura 26 - Modelo 3D final da esfera



Legenda: Figura que ilustra o modelo 3D final do objeto esférico de teste, construído através do algoritmo de reconstrução *screened poisson*.

Fonte: O autor, 2018

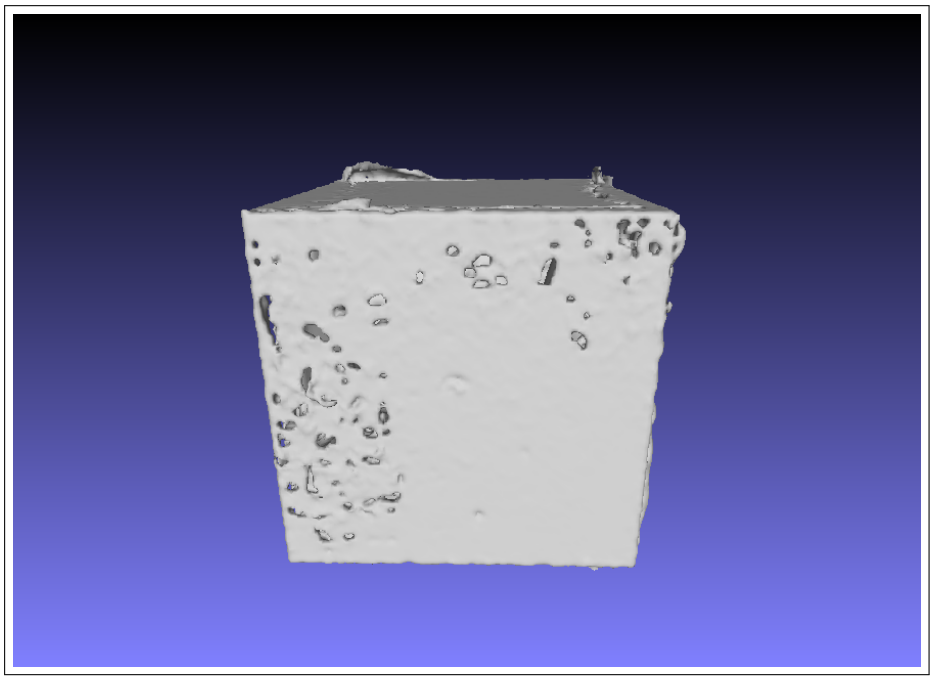
o objeto em questão, a segunda coluda apresenta a distância de Hausdorff final (mais detalhes na sessão 4.4), a terceira coluda apresenta o valor da média de todos os pontos da nuvem do respectivo objeto, enquanto que a última coluna exibe o desvio padrão dos erros encontrados em toda a nuvem. Todas as medições da tabela estão na ordem de milímetros.

Quanto aos Paralelepípedos, o número 1 representa o paralelepípedo de frente, como foi mostrado na Figura 19, o número 2 representa o paralelepípedo mais alto, com altura de 20 cm, largura de 13,1 cm e profundidade de 5,4 cm, enquanto que o terceiro representa o paralelepípedo mais achatado, já ilustrado na Figura 20.

A Figura 26 ilustra a nuvem de pontos final da esfera coberta por uma malha, gerando seu modelo 3D final. A construção do modelo final foi feita pelo algoritmo de reconstrução *Screened Poisson* (KAZHDAN; HOPPE, 2013) (ESTELLERS et al., 2015) (WU et al., 2014). Da mesma forma, os modelos 3D dos objetos cubo, paralelepípedo 1, paralelepípedo 2 e paralelepípedo 3 podem ser observados pelas Figuras, 27, 28, 29 e 30, respectivamente.

Além da malha, também é possível visualizar a construção do modelo final de cada

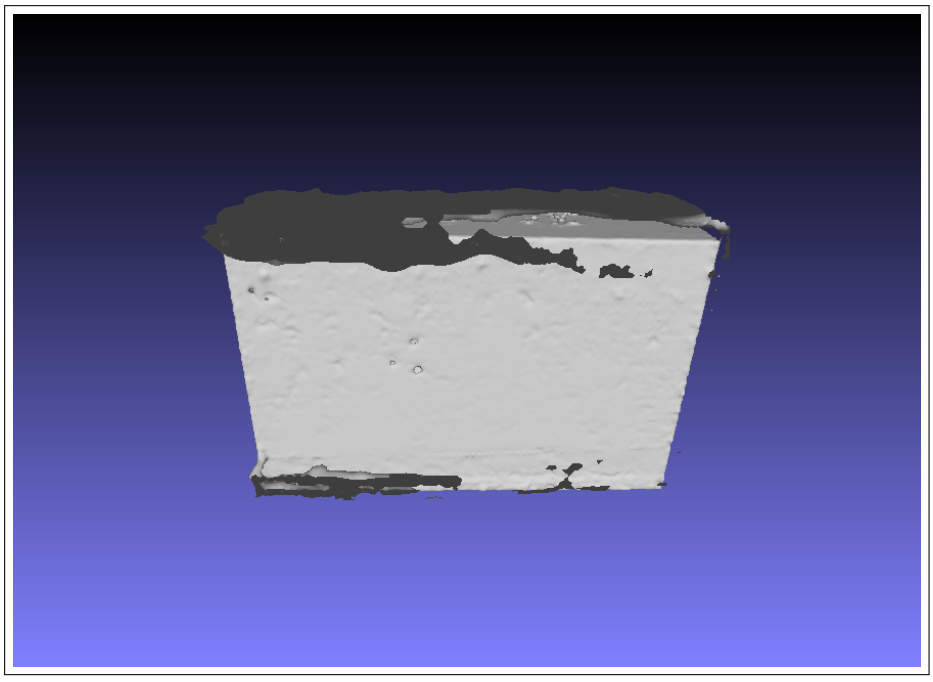
Figura 27 - Modelo 3D final do cubo



Legenda: Figura que ilustra o modelo 3D final do cubo de teste, construído através do algoritmo de reconstrução *screened poisson*.

Fonte: O autor, 2018

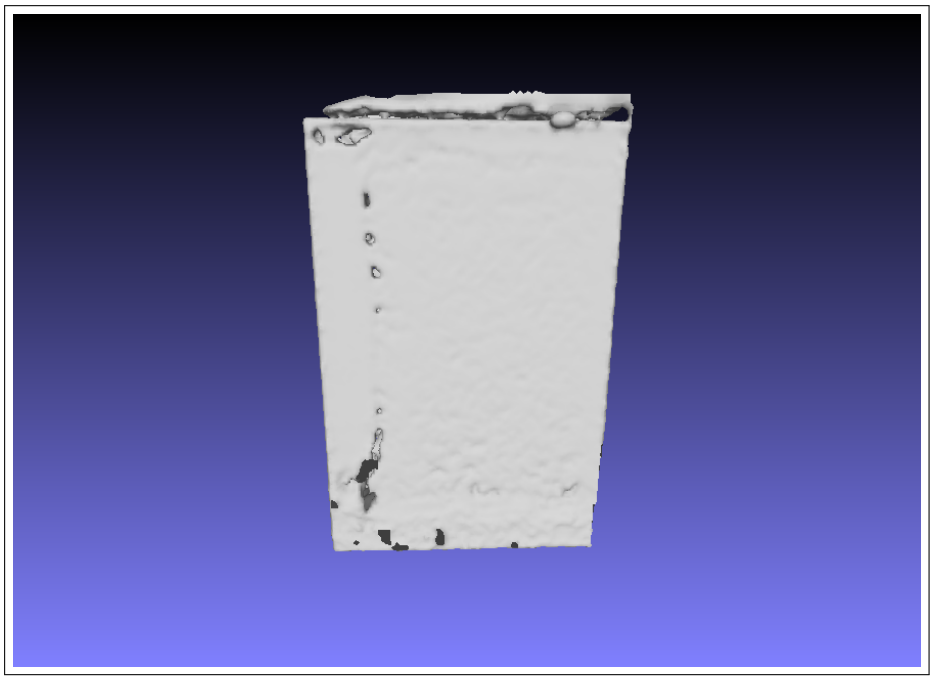
Figura 28 - Modelo 3D final do paralelepípedo 1



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 1 (de frente), construído através do algoritmo de reconstrução *screened poisson*.

Fonte: O autor, 2018

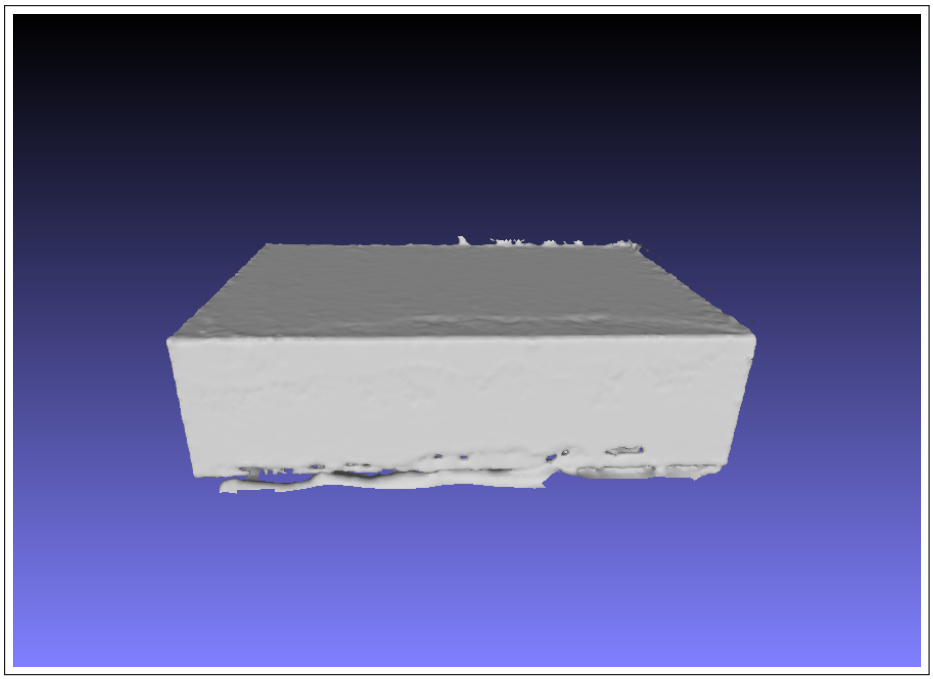
Figura 29 - Modelo 3D final do paralelepípedo 2



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 2 (mais alto), construído através do algoritmo de reconstrução *screened poisson*.

Fonte: O autor, 2018

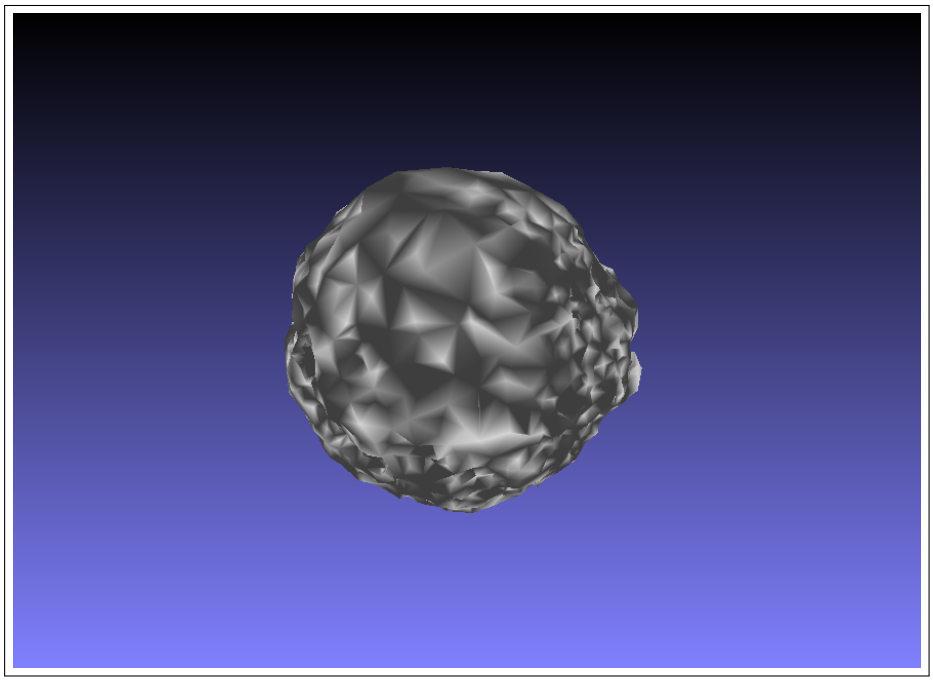
Figura 30 - Modelo 3D final do paralelepípedo 3



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 3 (achatado), construído através do algoritmo de reconstrução *screened poisson*.

Fonte: O autor, 2018

Figura 31 - Modelo 3D final da esfera triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final da esfera construído pela triangulação de Delaunay.
Fonte: O autor, 2018

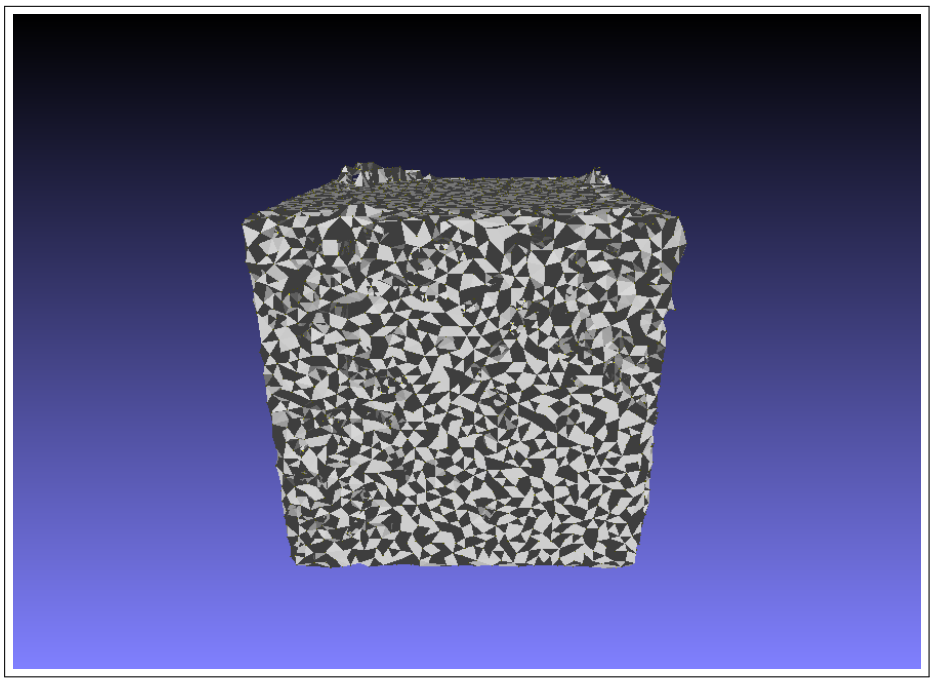
objeto através de uma triangulação de Delaunay (DELAUNAY et al., 1934) (PITERI; JUNIOR, 2007) (MAGALHÃES; PASSARO; ABE, 2000). O modelo da esfera pode ser visualizada na Figura 31, o do cubo nas Figuras 32 e 33, o do paralelepípedo 1 nas Figuras 34, 35 e 36, o do paralelepípedo 2 na Figura 37 e 38 e por fim o modelo do paralelepípedo 3 pode ser observado nas Figuras 39 e 40.

Em adição aos modelos gerados pelos algoritmos *screened poisson* e triangulação de Delaunay, as Figuras 41, 42, 43, 44 e 45 também ilustram a malha 3D de cada uma das nuvens de pontos dos objetos de referência, com o objetivo de facilitar a comparação visual entre essas malhas de referência e àquelas geradas com o algoritmo *screened poisson*.

Essas malhas de referência foram todas geradas com 60.008 pontos (a exceção da esfera que foi construído com 642 pontos, por ser um objeto bem menor em relação aos demais) e representam os objetos, cubo, esfera, paralelepípedo 1, paralelepípedo 2 e paralelepípedo 3, respectivamente.

Além da distância de Hausdorff, também foram medidas a área e o volume de cada uma das nuvens de pontos dos objetos de teste obtidas com o Kinect. Esses valores de área e volume fornecem uma ideia adicional à distância de Hausdorff do quão bom são as medida obtidas, fornecendo uma análise complementar.

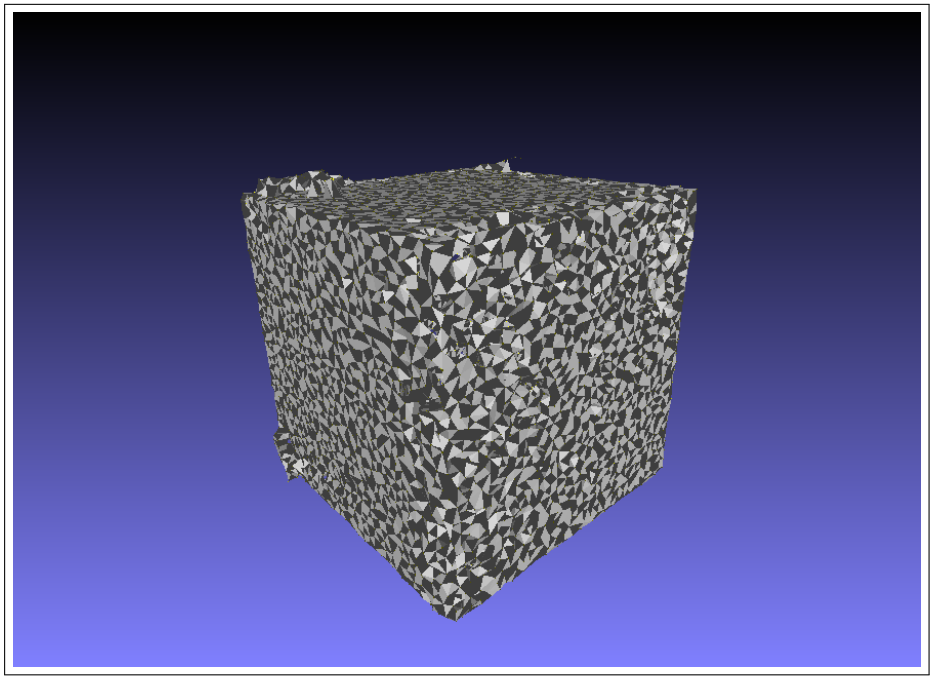
Figura 32 - Modelo 3D final do cubo pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do cubo construído pela triangulação de Delaunay.

Fonte: O autor, 2018

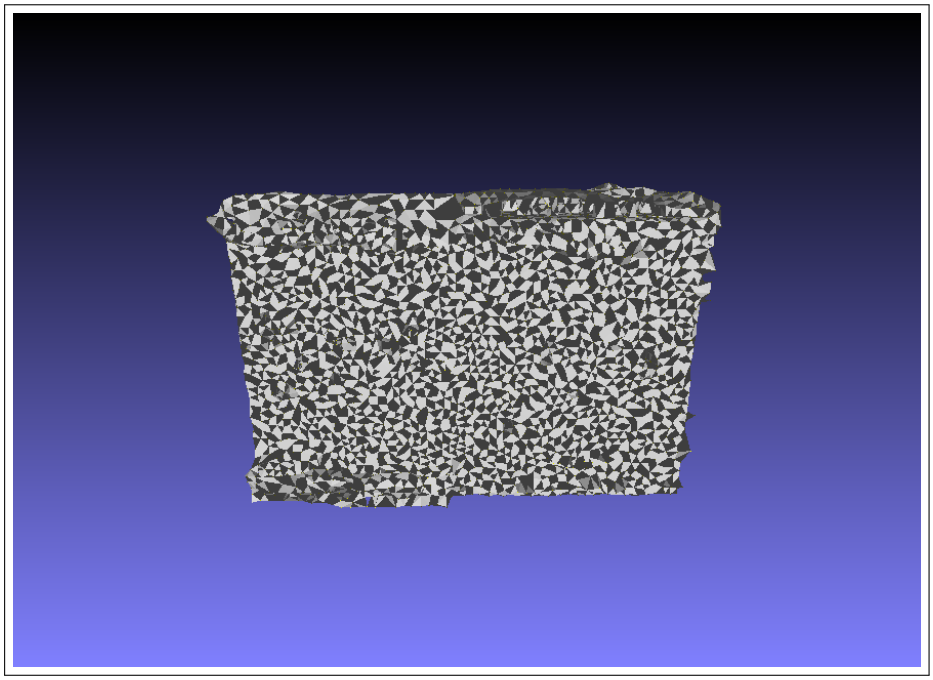
Figura 33 - Modelo 3D final do cubo pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do cubo construído pela triangulação de Delaunay.

Fonte: O autor, 2018

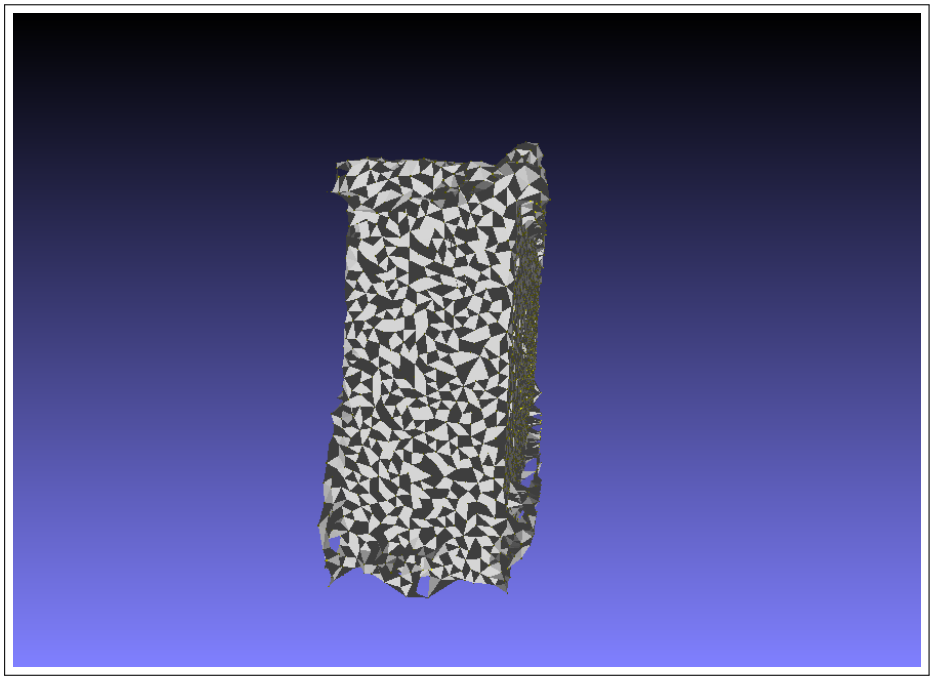
Figura 34 - Modelo 3D final do paralelepípedo 1 pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 1 construído pela triangulação de Delaunay.

Fonte: O autor, 2018

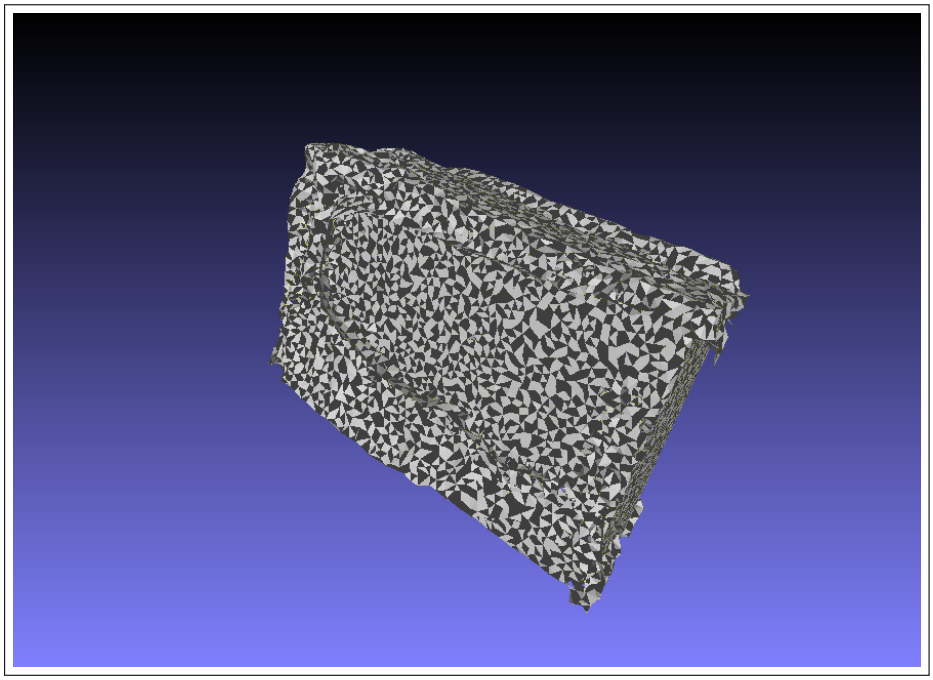
Figura 35 - Modelo 3D final do paralelepípedo 1 pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 1 construído pela triangulação de Delaunay.

Fonte: O autor, 2018

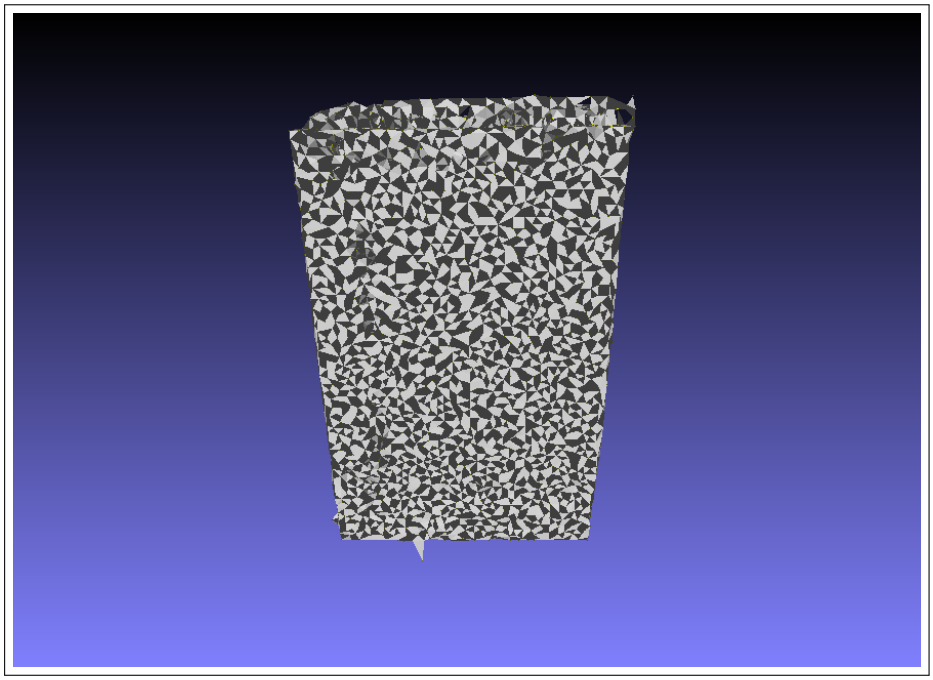
Figura 36 - Modelo 3D final do paralelepípedo 1 pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 1 construído pela triangulação de Delaunay.

Fonte: O autor, 2018

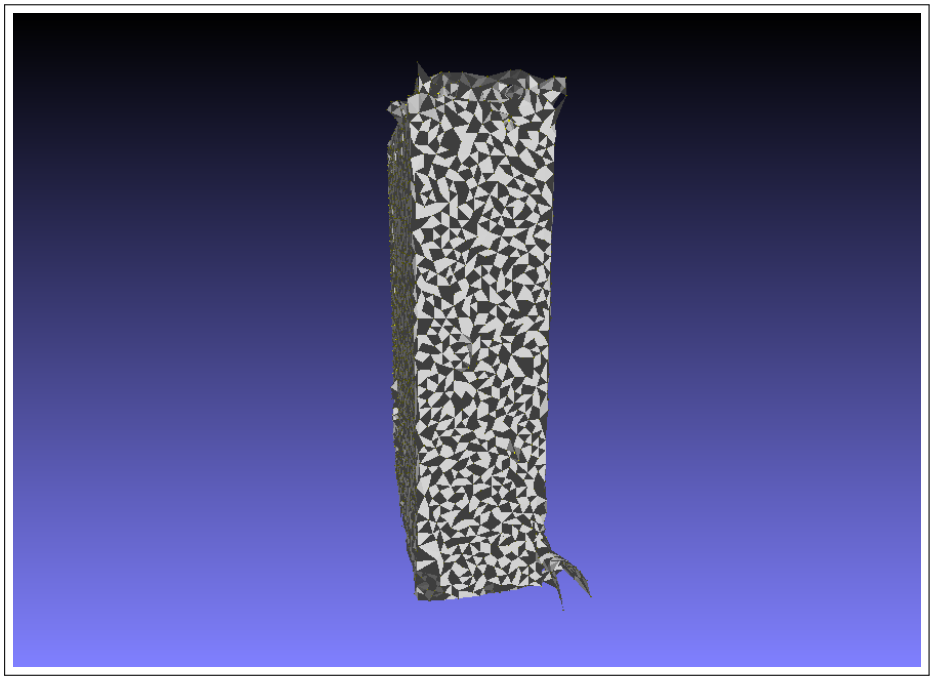
Figura 37 - Modelo 3D final do paralelepípedo 2 pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 2 construído pela triangulação de Delaunay.

Fonte: O autor, 2018

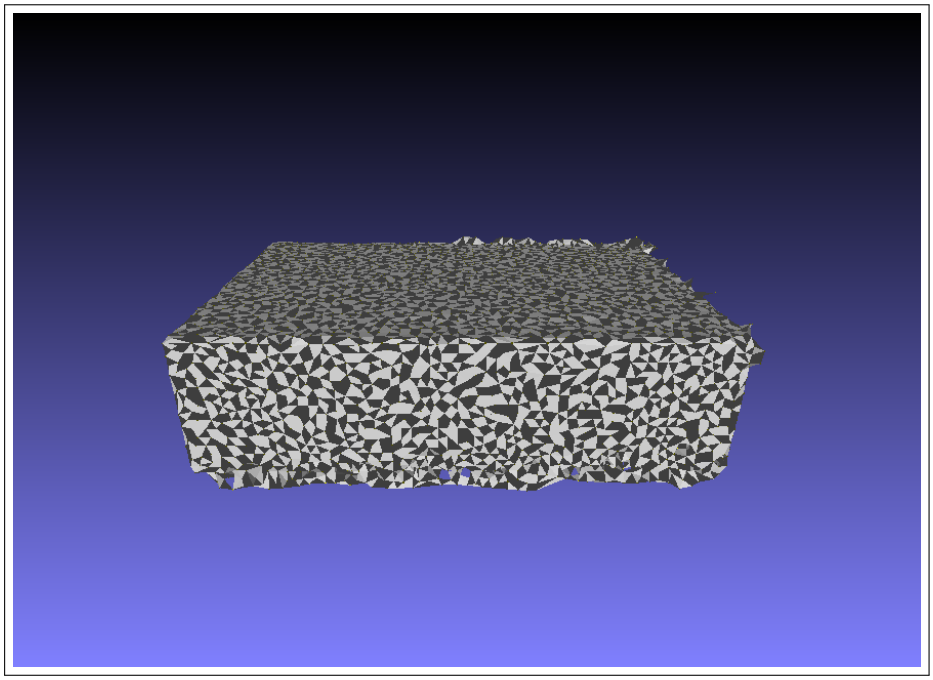
Figura 38 - Modelo 3D final do paralelepípedo 2 pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 2 construído pela triangulação de Delaunay.

Fonte: O autor, 2018

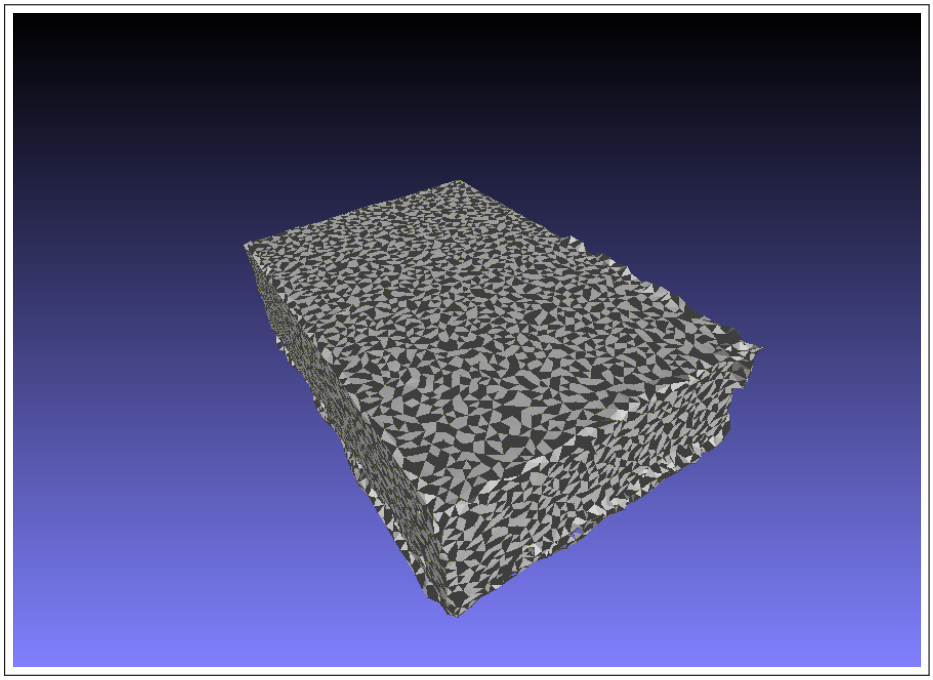
Figura 39 - Modelo 3D final do paralelepípedo 3 pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 3 construído pela triangulação de Delaunay.

Fonte: O autor, 2018

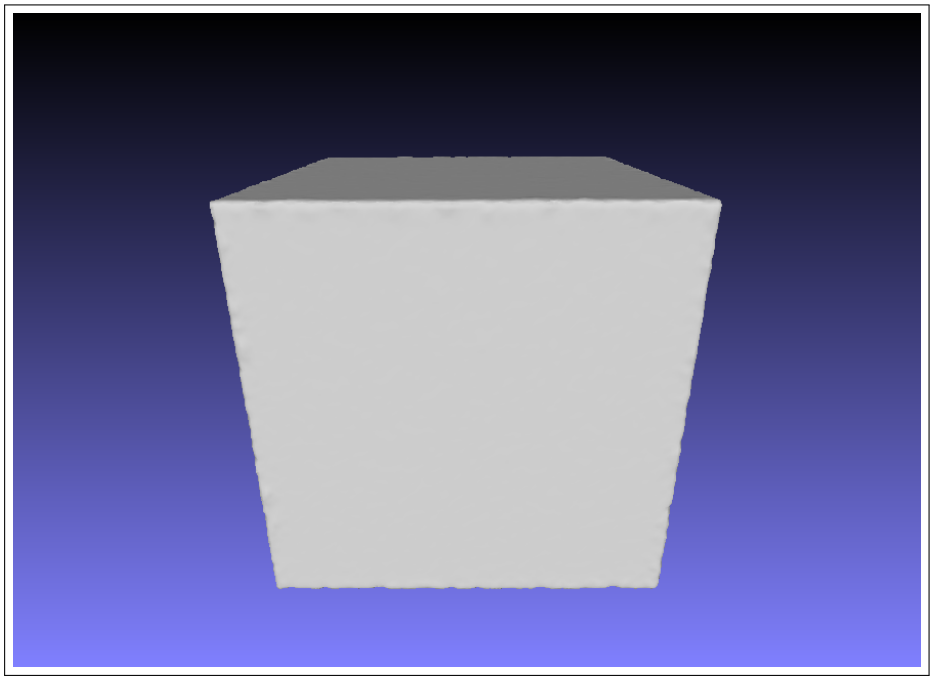
Figura 40 - Modelo 3D final do paralelepípedo 3 pela triangulação de Delaunay



Legenda: Figura que ilustra o modelo 3D final do paralelepípedo 3 construído pela triangulação de Delaunay.

Fonte: O autor, 2018

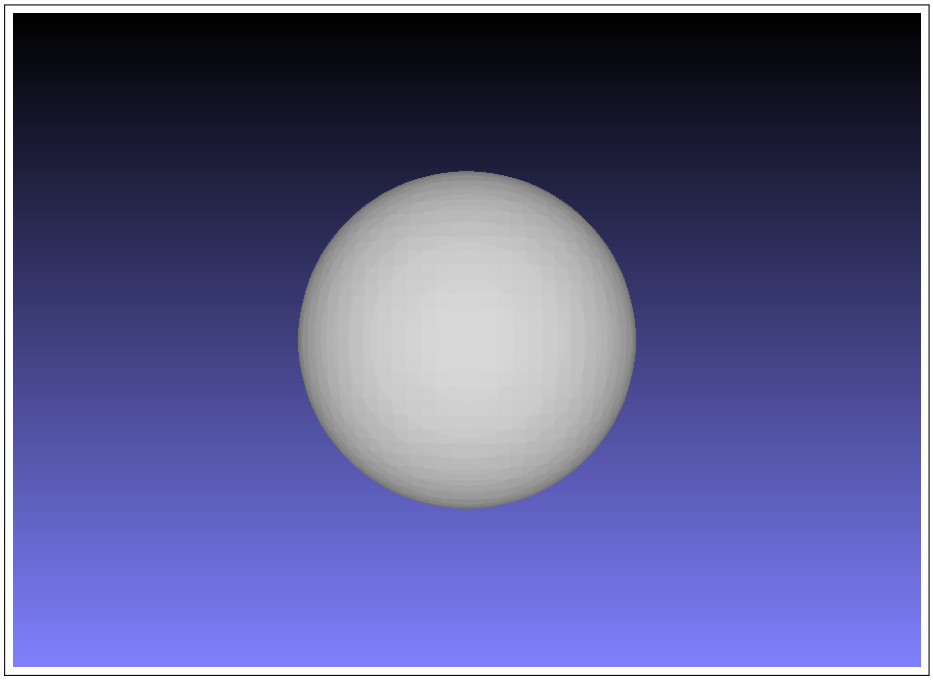
Figura 41 - Modelo 3D de referência do cubo



Legenda: Figura que ilustra o modelo 3D da nuvem de pontos de referência do cubo, gerada através do algoritmo *screened poisson*.

Fonte: O autor, 2018

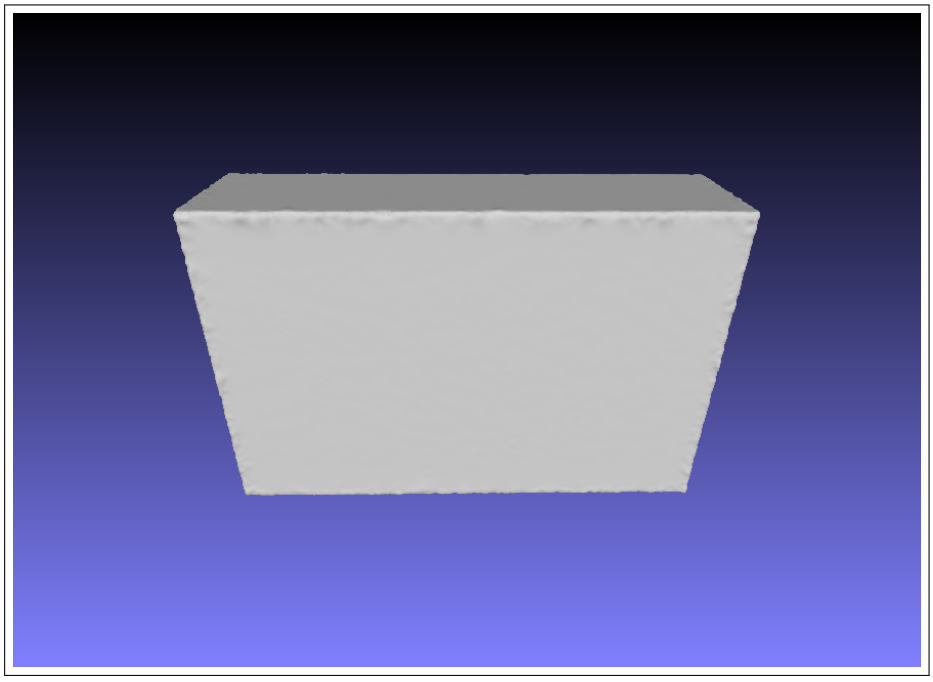
Figura 42 - Modelo 3D de referência da esfera



Legenda: Figura que ilustra o modelo 3D da nuvem de pontos de referência da esfera, gerada através do algoritmo *screened poisson*.

Fonte: O autor, 2018

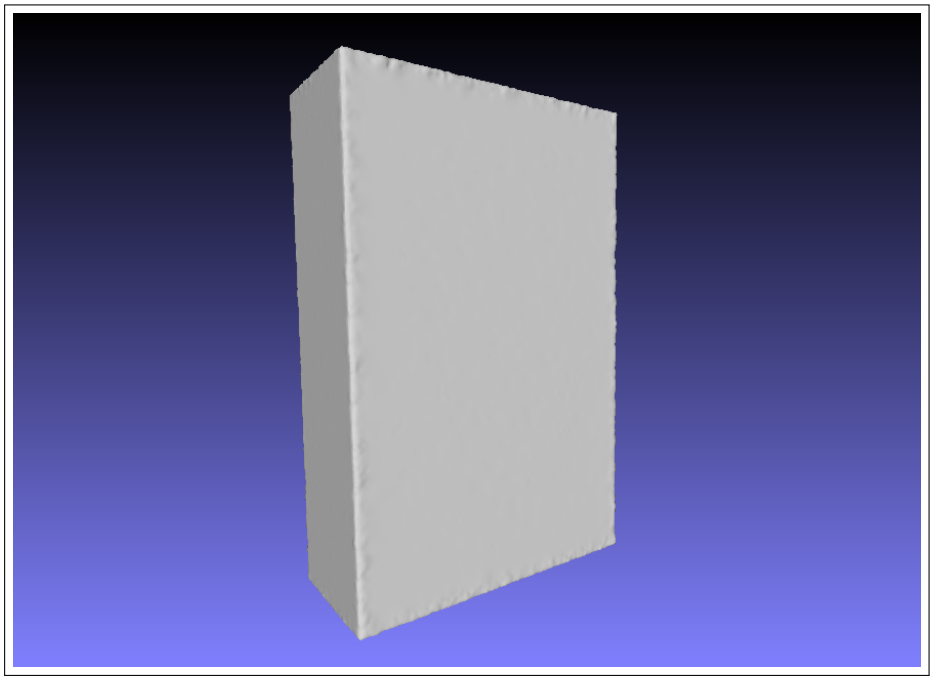
Figura 43 - Modelo 3D de referência do paralelepípedo 1



Legenda: Figura que ilustra o modelo 3D da nuvem de pontos de referência do paralelepípedo 1, gerada através do algoritmo *screened poisson*.

Fonte: O autor, 2018

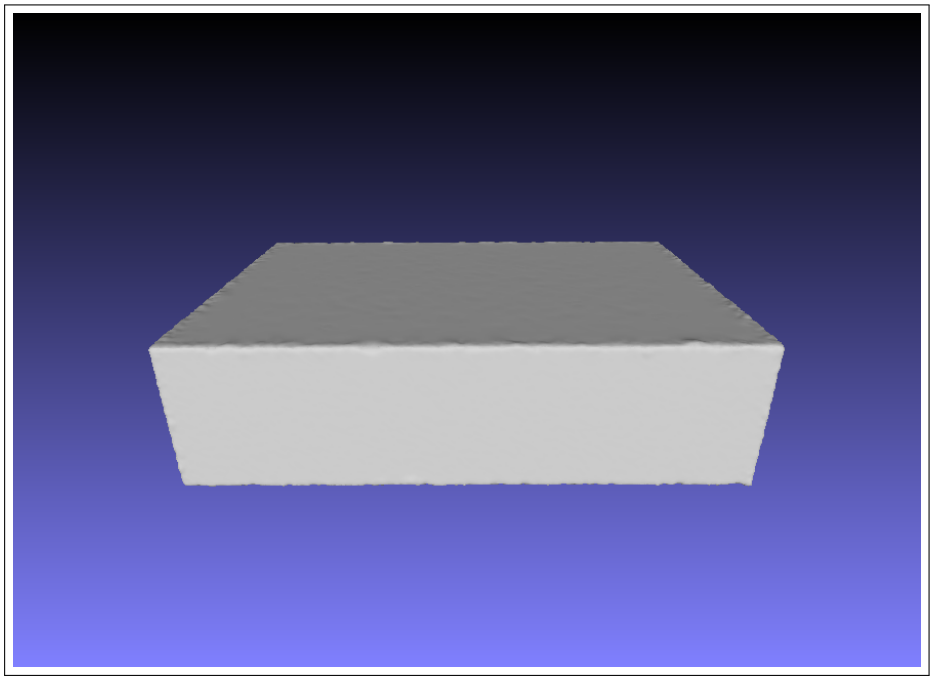
Figura 44 - Modelo 3D de referência do paralelepípedo 2



Legenda: Figura que ilustra o modelo 3D da nuvem de pontos de referência do paralelepípedo 2, gerada através do algoritmo *screened poisson*.

Fonte: O autor, 2018

Figura 45 - Modelo 3D de referência do paralelepípedo 3



Legenda: Figura que ilustra o modelo 3D da nuvem de pontos de referência do paralelepípedo 3, gerada através do algoritmo *screened poisson*.

Fonte: O autor, 2018

Tabela 8 - Resultados complementares de área e volume

Objeto	**Área**	**Área Referência**	**Volume**	**Volume Referência**
Cubo	$1019cm^2$	$1350cm^2$	$3136{,}11cm^3$	$3375cm^3$
Esfera	$149{,}43cm^2$	$91{,}60cm^2$	$74cm^3$	$82{,}44cm^3$
Paralelepípedo 1	$1716{,}77cm^2$	$881{,}48cm^2$	$1250cm^3$	$1414{,}8cm^3$
Paralelepípedo 2	$1815{,}07cm^2$	$881{,}48cm^2$	$1031cm^3$	$1414{,}8cm^3$
Paralelepípedo 3	$1389{,}59cm^2$	$881{,}48cm^2$	$1327cm^3$	$1414{,}8cm^3$

Legenda: Tabela com os resultados do cálculo de área e volume de cada uma das nuvens de pontos dos objetos de teste.

Fonte: O autor, 2018

Os resultados de área e volume podem ser observados na Tabela 8.

Todos os resultados da Tabela 8 estão na ordem de centímetros. A tabela está organizada de forma que a primeira coluna identifica o objeto de teste em questão; A segunda coluna exibe a área total calculada através da nuvem de pontos do objeto; Na terceira coluna é possível observar o valor da área da nuvem de pontos do objeto de referência, para que seja possível efetuar a comparação com a área obtida; a quarta coluna exibe o volume e a quinta coluna o volume do objeto de referência.

4.6.1 Avaliação Numérica

Avaliando inicialmente os resultados da Tabela 2, com o valor máximo da distância de Hausdorff, observa-se que os valores não são muito satisfatórios. A taxa de erro dos objetos varia entre 1 centímetro (com a esfera) até 6 cm (com o cubo), fazendo com que o *Kinect* não seja uma boa opção para a exatidão de objetos a uma distância similar a que foi testada nesse trabalho, se fosse levado em conta apenas a distância máxima. O objeto de teste com melhor exatidão foi a esfera com 18,9 mm pela distância de Hausdorff e o pior foi o cubo com 68,1 mm. As variações de paralelepípedo ficaram entre 21 e 26 mm. Porém a distância máxima de afastamento entre duas nuvens pode passar a falsa sensação de inexatidão no caso em que uma parte ou região da nuvem de pontos do objeto digitalizado apresente um erro muito grande e as demais regiões apresentem bastante semelhança com o objeto real, levando o modelo como um todo a assumir a pior exatidão dentre todos os seus pontos. Devido a essa situação, os demais resultados métricos devem ser analisados.

A distância de Hausdorff Mínima apresentou um cenário muito semelhante para todos os objetos, em que a distância mínima medida é muito próxima de zero. Isso indica que provavelmente existem pontos da nuvem gerada pelo Kinect que são coincidentes a

outros pontos pertencentes a nuvem de referência. Novamente, esse resultado praticamente perfeito de distanciamento pode levar a uma interpretação errônea da exatidão do modelo alcançado. Por razões inversas a da distância de Hausdorff comum, é atribuído a melhor exatidão dentre todos os pontos da nuvem ao modelo inteiro, ignorando ou camuflando possíveis falhas e imperfeições das demais regiões. A análise deve prosseguir para uma melhor distribuição das taxas de erro ao longo de todo o modelo.

O valor do erro médio fornece uma ideia melhor da distribuição do erro como um todo no modelo. Dessa forma é possível observar que a distância mínima dos pontos que tente a zero é totalmente absurda, uma vez que o erro de exatidão varia de 2 a 6 mm para a distância da nuvem do objeto para a nuvem de referência, para todos os objetos. Os paralelepípedos 1 e 3 já apresentam bons resultados nessa análise mais geral, com exatidão de 2 mm. Com essa mesma ideia, a avaliação da distância de Hausdorff pela distância máxima também é incabível, pois apresentou um erro que variava na escala dos centímetros, enquanto que a média dos erros está na escala dos milímetros, apresentando erros bem menores distribuídos ao longo de todo o modelo.

Partindo para uma análise mais bem distribuída ao longo de toda a nuvem de pontos utilizando medidas de dispersão, no desvio padrão observa-se um pequeno aumento no erro em relação ao erro médio. A diferença ainda é bastante pequena, fazendo com que o desvio padrão varie de 3 a 7 mm. Esse resultado informa que os valores dos erros médios são bastantes confiáveis.

Os paralelepípedos 1 e 3 apresentaram o melhor resultado dentre todos os objetos de teste digitalizados, com 2,8 mm e 2,9 mm no erro médio, respectivamente. Os objetos que apresentaram os piores resultados foram o cubo e o paralelepípedo 2, com 6,0 mm e 6,8 mm no erro médio, respectivamente.

Quanto aos resultados obtidos com a redução da quantidade de pontos nas nuvens de referência (reduzindo de 60.008 para 10.008), exibidos na Tabela 4, os valores são muito similares em todos os objetos. Apenas no erro médio que existe uma alteração de aproximadamente 3 mm de imprecisão quando se reduz a quantidade de pontos da nuvem de referência para 10.008 pontos. Esses resultados parecidos são previsíveis, visto que o objetivo da nuvem de pontos de referência é simular e representar o objeto real.

Na Tabela 5 são exibidos os resultados utilizando 10.008 pontos para as nuvens de referência de todos os objetos e o sensor encontra-se posicionado a 120 cm de distância dos mesmos objetos. É possível observar uma piora em todas as medidas. Como a distância do sensor dobrou, é possível fazer uma analogia com a imprecisão das medidas que praticamente dobraram em todos os objetos. O cubo por exemplo, possuía um erro médio em todo o modelo de 6 mm e passou para 13,9 mm quando a distância do sensor dobrou. Essa situação se repete para todos os objetos de teste. A partir dessa análise é possível deduzir uma taxa de imprecisão linear em relação a distância entre o objeto e o sensor, ou seja, quanto mais o sensor se distancia dos objetos, a imprecisão aumenta de

forma linear.

A Tabela 7 exibe os resultados utilizando 10.008 pontos para as nuvens de pontos de referência e o sensor posicionado a 120 cm de distância dos objetos. Assim como nos resultados com o sensor posicionado a 60 cm dos objetos, a variação da quantidade de pontos das nuvens de pontos de referência não acarretou uma grande mudança nos valores obtidos.

Os resultados do objeto esférico seguiram uma linha diferente em termos de quantidade de pontos que compunham a nuvem de pontos de referência, ela foi usada com 642 e 10.008 pontos. Isso foi feito devido a esfera ser um objeto bem menor em comparação aos demais objetos de teste. Os resultados avaliados desse objeto seguem o mesmo padrão dos demais, perdendo exatidão em proporções lineares a medida que o sensor se distância do objeto e reduzindo sua exatidão a medida que a quantidade de pontos da nuvem de pontos de referência é reduzida.

Seguindo para os resultados obtidos através do cálculo de volume e área dos objetos, os valores parecem não ser muito satisfatórios. A área calculada dos paralelepípedos é bastante ruim em relação a referência, apresentando quase o dobro da área buscada. No entanto, os valores de volumes já são bem melhores e próximos das referências para todos os objetos. A esfera apresentou os piores resultados de forma geral, provavelmente devido ao fato de ser o menor objeto entre os testados.

4.6.2 Avaliação Visual

Analisando visualmente, percebe-se que os modelos 3D gerados pelo algoritmo *screened poisson* apresentam distorções e deformações mais próximo das bordas dos objetos e na junção com a mesa. A esfera não possui limiares com 90 graus de mudança da superfície como o cubo e os paralelepípedos possuem, razão pelo qual apresentou resultados melhores para a distância de Hausdorff comum, que sempre assume o pior erro do modelo inteiro, possivelmente nessas bordas.

O paralelepípedo 1 apresentou grandes deformações nas bordas da parte superior do objeto, enquanto que o paralelepípedo 3 apresentou mais deformações na parte de junção com a mesa digitalizadora.

Também observa-se que os objetos que possuem superfícies retas mais longas, como o cubo e os paralelepípedos, eventualmente surgem alguns pequenos orifícios, as vezes buracos, provavelmente devido a falta de informação das nuvens de pontos naquela parte.

Quanto aos modelos gerados pela triangulação de Delaunay, observa-se o aspecto triangular característico do algoritmo em todos os modelos. A esfera parece um conglomerado triangular desforme, embora ainda mantenha sua forma ligeiramente esférica. Existe uma saliência na lateral direita do objeto, exatamente onde existe também um calombo

no modelo do mesmo objeto pelo algoritmo *screened poisson*.

O cubo e os paralelepípedos aparentemente receberam uma grande melhora na triangulação de Delanay, pois suas superfícies tornaram-se melhores onde outrora existiam orifícios e buracos. As bordas que antes possuía bastante imperfeição em *screened poisson*, ainda mantém essas imperfeições e incongruências, embora pareçam um pouco mais suaves.

Por fim, a triangulação de Delaunay parece ter apresentado um modelo melhor para o cubo e os paralelepípedos, apesar da evidente perda de detalhamento. O algoritmo *screened poisson* apresentou mais detalhes no modelo, mas também ressaltou possível imperfeições na nuvem de pontos, gerando deformidades.

4.6.3 Comparação

A comparação dos resultados desse trabalho com os resultados de outros trabalhos é complicada, pois cada um optou por um cenário e *setup* diferente, assim como métodos de validação distintos, impossibilitando uma comparação direta.

O trabalho (SARBOLANDI; KOLB, 2015) apresentou vários testes com o *Kinect One* em vários ambientes, comparando-o com o *Kinect 360*. Obtendo bons resultados para alguns dos cenários, porém apenas para testes de cálculo de distância e em cenários específicos, não para verificação de exatidão da forma de objetos a uma determinada distância do sensor. Outros trabalhos para a avaliação da exatidão do *Kinect One* também podem ser encontrados em (GONZALEZ-JORGE et al., 2013), (WASENMÜLLER; STRICKER, 2016) e (ZENNARO, 2014).

O trabalho apresentado em (WASENMÜLLER; STRICKER, 2016) registrou um erro de exatidão constante de 18 mm para o *Kinect One*, porém apenas para cálculo de distância novamente, não para toda a estrutura de um objeto. Já o trabalho apresentado em (GONZALEZ-JORGE et al., 2013), executou testes com objetos, 5 esferas e 7 cubos de alumínio. Os *frames* foram capturados com o sensor a um ângulo de 45, 90 e 135 graus dos objetos, obtendo um erro de exatidão da nuvens de pontos que varia de 5 mm a 15 mm, a uma distância de 1 metro do sensor. Aumentando a distância do sensor para o objeto, a exatidão dos dados adquiridos passa a variar de 5 mm para 25 mm. Como pode-se observar, os resultados obtidos no presente trabalho são similares aos resultados de (GONZALEZ-JORGE et al., 2013).

Outro trabalho que verificou a exatidão dos dados adquiridos pelo *Kinect* é o (TONG et al., 2012). Nesse trabalho, o corpo completo de um humano é digitalizado, utilizando 3 *Kinects* simultaneamente para isso. Foi obtido um erro de 2,5 cm na região do pescoço até o quadril, 1,5 cm na largura entre os ombros, 2 a 3 cm no tamanho dos braços e pernas e 6,2 cm na largura da cintura. Observa-se nesse caso que os resultados são uma

escala maior que os resultados obtidos no presente trabalho, porém isso provavelmente se deve ao corpo humano ser um objeto de estudo bem mais complexos que os objetos geométricos aqui utilizados.

CONCLUSÃO

O Kinect One apresentou resultados bons e aceitáveis para a digitalização 3D de objetos para cenários em que não é necessária uma exatidão muito grande de detalhes do objeto e cenários com objetos não muito distantes. A digitalização da forma do objeto pode ser alcançada, porém com uma quantidade eleva de ruídos nos arredores do objeto, reduzindo consideravelmente sua exatidão para detalhes.

Ele apresentou uma exatidão que varia de 2 a 6 mm de erro na forma a 60 cm de distância do objeto, o que foi suficiente para reconhecer as formas dos objetos propostos nesse trabalho e nas condições de distância testadas. A distâncias maiores de um metro, os erros de exatidão foram muito grandes e problemáticos. Objetos com muitos vértices e arestas tendem a apresentar uma redução considerável na exatidão também.

Na análise do cálculo de área e volume dos modelos 3D, as áreas dos modelos demostraram ser muito ruins, com grandes defasagens, chegando a possuir até mesmo o dobro da área buscada. O cálculo do volume já apresentou resultados melhores, com valores bem próximos do esperado.

Com os resultados obtidos nesse trabalho, torna-se previsível que ao utilizar objetos mais complexos que os simples objetos geométricos aqui usados, a exatidão se tornará pior e os modelos 3D mais deformados, porém as formas dos objetos são nitidamente reconhecidas, possibilitando que o Kinect One seja utilizado em aplicações que utilizam o reconhecimento dos objetos, mas não de muita exatidão de detalhes dos mesmos, como em propostas de reconhecimento facial apresentadas nos trabalhos (KOVACS et al., 2006) e (MOSS et al., 1987).

O Kinect One é um sensor de baixo orçamento e não deve ser comparado diretamente, em termos de exatidão de detalhes, a sensores de alto desempenho como *laser scanners* que, em termos financeiros, custam dezenas de vez mais o preço do *Kinect*. Porém, em adição a tudo isso, o *Kinect One* ainda representa uma boa alternativa de baixo orçamento para projetos que não demandam muita exatidão em objetos digitalizados e aplicações de engenharia de curto alcance.

Trabalhos futuros podem contemplar testes em uma nova gama de objetos, incluindo objetos complexos, com frestas, buracos e pequenos detalhes, assim como posicionados a diferentes distâncias do sensor. O driver libfreenect2 para controle do sensor também pode ser substituído por outro, como o OpenNI2, JuneSDK ou até mesmo uma versão mais recente do próprio libfreenect2. Todos os programas desenvolvidos neste trabalho também podem ser fundidos em um único programa, acoplando todos os requisitos de bibliotecas, tornando o código mais complexo e difícil de receber manutenção, porém mais fácil de ser utilizado por um eventual usuário que tenha interesse em replicar os experimentos aqui executados ou utilizar os programas para novas aplicações. Por fim,

é possível replicar os experimentos deste trabalho, executar e avaliar novos experimentos com o novo sensor da *microsoft*, e sucessor do *Kinect*, Azure Kinect, que é voltado para aplicações industriais e comerciais de inteligência artificial e utiliza a nova plataforma Azure da *microsoft*.

REFERÊNCIAS

3DNATIVES. *TOP 13 Best Low Cost 3D Scanners (2019 Update)*. 2019. Disponível em: <https://www.3dnatives.com/en/top-10-low-cost-3d-scanners280320174/>.

A., SÉRGIO LEANDRO; G., ANTONIO MARIA; O., ALMIR. UtilizaÇÃo de alvos codificados do tipo aruco na automaÇÃo do processo de calibraÇÃo de cÂmaras. *Boletim de Ciências Geodésicas*, v. 20, p. 626–646, 2014. ISSN 1413-4853. Disponível em: <http://www.redalyc.org/articulo.oa?id=393933953009>.

ALLIEZ, Pierre et al. Voronoi-based variational reconstruction of unoriented point sets. In: *Symposium on Geometry processing*. [S.l.: s.n.], 2007. v. 7, p. 39–48.

ANDERSEN, MR et al. Kinect depth sensor evaluation for computer vision applications. *Aarhus University*, p. 1–37, 2012.

ANIL, Engin Burak et al. Deviation analysis method for the assessment of the quality of the as-is building information models generated from point cloud data. *Automation in Construction*, v. 35, p. 507 – 516, 2013. ISSN 0926-5805. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0926580513001003⟩.

ASPERT, N.; SANTA-CRUZ, D.; EBRAHIMI, T. Mesh: measuring errors between surfaces using the hausdorff distance. In: *Proceedings. IEEE International Conference on Multimedia and Expo*. [S.l.: s.n.], 2002. v. 1, p. 705–708 vol.1.

BERGER, Matt et al. An end-to-end framework for evaluating surface reconstruction. *Sci Comput Imag Inst*, 2011.

BESL, P. J.; MCKAY, N. D. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 14, n. 2, p. 239–256, February 1992. ISSN 0162-8828.

BLAKE, J; ECHTLER, F; KERL, C. *libfreenect2: Open source drivers for the Kinect for Windows v2 device*. 2015.

BLUMBERG, Henry. Hausdorff's grundzüge der mengenlehre. *Bulletin of the American Mathematical Society*, v. 27, n. 3, p. 116–129, 1920.

BONNAFFE, Florence; JENNETTE, Dave; ANDREWS, John. A method for acquiring and processing ground-based lidar data in difficult-to-access outcrops for use in three-dimensional, virtual-reality models. *Geosphere*, Geological Society of America, v. 3, n. 6, p. 501–510, 2007.

BONNECHERE, Bruno et al. Determination of the precision and accuracy of morphological measurements using the kinect™ sensor: Comparison with standard stereophotogrammetry. *Ergonomics*, Taylor & Francis, v. 57, n. 4, p. 622–631, 2014.

BRADSKI, Gary. The opencv library. November 2000. Disponível em: ⟨http://www.drdobbs.com/open-source/the-opencv-library/184404319⟩.

BRADSKI, Gary; KAEHLER, Adrian. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. 2nd. ed. [S.l.]: O'Reilly Media, Inc., 2013. ISBN 1449314651, 9781449314651.

BUENO, M. et al. Metrological evaluation of kinectfusion and its comparison with microsoft kinect sensor. *Measurement*, v. 73, p. 137 – 145, 2015. ISSN 0263-2241. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0263224115002778⟩.

CALAKLI, Fatih; TAUBIN, Gabriel. Ssd: Smooth signed distance surface reconstruction. In: WILEY ONLINE LIBRARY. *Computer Graphics Forum*. [S.l.], 2011. v. 30, n. 7, p. 1993–2002.

CALLIERI, M et al. Processing a complex architectural sampling with meshlab: the case of piazza della signoria. *Proceedings of 3D-ARCH*, v. 4, 2011.

CANONICAL. *Ubuntu Linux*. 2019. Disponível em: ⟨https://www.ubuntu.com⟩.

CHEN, Yang; MEDIONI, Gérard. Object modelling by registration of multiple range images. *Image and Vision Computing*, v. 10, n. 3, p. 145 – 155, 1992. ISSN 0262-8856. Range Image Understanding. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/026288569290066C⟩.

CHIA, Tsorng-Lin; CHEN, Zen; YUEH, Chaur-Jou. Curved surface reconstruction using a simple structured light method. In: IEEE. *Proceedings of 13th International Conference on Pattern Recognition*. [S.l.], 1996. v. 1, p. 844–848.

CIGNONI, Paolo et al. MeshLab: an Open-Source Mesh Processing Tool. In: SCARANO, Vittorio; CHIARA, Rosario De; ERRA, Ugo (Ed.). *Eurographics Italian Chapter Conference*. [S.l.]: The Eurographics Association, 2008. ISBN 978-3-905673-68-5.

CIGNONI, Paolo; ROCCHINI, Claudio; SCOPIGNO, Roberto. Metro: measuring error on simplified surfaces. In: BLACKWELL PUBLISHERS. *Computer Graphics Forum*. [S.l.], 1998. v. 17, n. 2, p. 167–174.

CLARKSON, Sean et al. Calculating body segment inertia parameters from a single rapid scan using the microsoft kinect. In: *Proceedings of the 3rd international conference on 3D body scanning technologies*. [S.l.: s.n.], 2012. p. 153–163.

CORSINI, Massimiliano; CIGNONI, Paolo; SCOPIGNO, Roberto. Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transaction on Visualization and Computer Graphics*, v. 18, n. 6, p. 914–924, 2012. Http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.34. Disponível em: ⟨http://vcg.isti.cnr.it/Publications/2012/CCS12⟩.

CUI, Yan; STRICKER, Didier. 3d shape scanning with a kinect. In: ACM. *ACM SIGGRAPH 2011 Posters*. [S.l.], 2011. p. 57.

DELAUNAY, Boris et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, v. 7, n. 793-800, p. 1–2, 1934.

DIGNE, Julie et al. Scale space meshing of raw data point sets. In: WILEY ONLINE LIBRARY. *Computer Graphics Forum*. [S.l.], 2011. v. 30, n. 6, p. 1630–1642.

DIMITROV, Andrey; GOLPARVAR-FARD, Mani. Segmentation of building point cloud models including detailed architectural/structural features and mep systems. *Automation in Construction*, v. 51, p. 32 – 45, 2015. ISSN 0926-5805. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0926580514002593⟩.

DOBASHI, Yoshinori et al. A simple, efficient method for realistic animation of clouds. In: ACM PRESS/ADDISON-WESLEY PUBLISHING CO. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. [S.l.], 2000. p. 19–28.

DUGGAL, Vijay. *Cadd Primer: A General Guide to Computer Aided Design and Drafting-Cadd, CAD*. 1st. ed. [S.l.]: Mailmax Pub, 2000. 210 p. ISBN 0962916595.

DUTTA, Tilak. Evaluation of the kinect™ sensor for 3-d kinematic measurement in the workplace. *Applied ergonomics*, Elsevier, v. 43, n. 4, p. 645–649, 2012.

ESTELLERS, Virginia et al. Robust poisson surface reconstruction. In: SPRINGER. *International Conference on Scale Space and Variational Methods in Computer Vision*. [S.l.], 2015. p. 525–537.

FALKINGHAM, Peter L. Generating a photogrammetric model using visual sfm, and post-processing with meshlab. *Brown University, Tech. Rep.*, 2013.

FANKHAUSER, P. et al. Kinect v2 for mobile robot navigation: Evaluation and modeling. In: *2015 International Conference on Advanced Robotics (ICAR)*. [S.l.: s.n.], 2015. p. 388–394. ISBN 978-1-4673-7509-2.

FLETCHER, Clive AJ. Computational galerkin methods. In: *Computational Galerkin Methods*. [S.l.]: Springer, 1984. p. 72–85.

GARRIDO-JURADO R. MUNõZ-SALINAS, F.J. Madrid-Cuevas R. Medina-Carnicer S. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, v. 51, p. 481 – 491, 2016. ISSN 0031-3203. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0031320315003544⟩.

GARRIDO-JURADO, S. et al. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, v. 47, n. 6, p. 2280 – 2292, 2014. ISSN 0031-3203. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0031320314000235⟩.

_____. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, v. 51, p. 481 – 491, 2016. ISSN 0031-3203. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0031320315003544⟩.

GLADWELL, M.; KORYTOWSKI, I. *Fora De Série - Outliers*. SEXTANTE. ISBN 9788575424483. Disponível em: ⟨https://books.google.com.br/books?id=xvN4PgAACAAJ⟩.

GONZALEZ-JORGE, Higinio et al. Metrological evaluation of microsoft kinect and asus xtion sensors. *Measurement*, Elsevier, v. 46, n. 6, p. 1800–1806, 2013.

GONZALEZ-JORGE, H et al. Metrological comparison between kinect i and kinect ii sensors. *Measurement*, Elsevier, v. 70, p. 21–26, 2015.

HAGEBEUKER, Dipl-Ing Bianca; MARKETING, Product. A 3d time of flight camera for object detection. *PMD Technologies GmbH, Siegen*, 2007.

HAUSDORFF, Felix. *Grundzüge der Mengenlehre*. 1st. ed. [S.l.]: Veit, 1914. 624 p. ISBN 978-0-8284-0061-9.

HAWKINS, Douglas M. *Identification of outliers*. [S.l.]: Springer, 1980. v. 11.

HURTADO, F.; NOY, M.; URRUTIA, J. Flipping edges in triangulations. *Discrete & Computational Geometry*, v. 22, n. 3, p. 333–346, Oct 1999. ISSN 1432-0444. Disponível em: ⟨https://doi.org/10.1007/PL00009464⟩.

HUTTENLOCHER, D. P.; KLANDERMAN, G. A.; RUCKLIDGE, W. J. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 15, n. 9, p. 850–863, Sep. 1993. ISSN 0162-8828.

IZADI, Shahram et al. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: ACM, 2011. (UIST '11), p. 559–568. ISBN 978-1-4503-0716-1. Disponível em: ⟨http://doi.acm.org/10.1145/2047196.2047270⟩.

JESORSKY, Oliver; KIRCHBERG, Klaus J.; FRISCHHOLZ, Robert W. Robust face detection using the hausdorff distance. In: BIGUN, Josef; SMERALDI, Fabrizio (Ed.). *Audio- and Video-Based Biometric Person Authentication*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. p. 90–95. ISBN 978-3-540-45344-4.

JI, X.; LIU, H. Advances in view-invariant human motion analysis: A review. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 40, n. 1, p. 13–24, Jan 2010. ISSN 1094-6977.

JIAO, Jichao et al. A post-rectification approach of depth images of kinect v2 for 3d reconstruction of indoor scenes. *ISPRS International Journal of Geo-Information*, v. 6, p. 349, 11 2017.

JIN, Hubert et al. Surface reconstruction from misregistered data. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Vision Geometry IV*. [S.l.], 1995. v. 2573, p. 324–329.

KAWATA, Megumi; MURAO, Hajime. Study on kinect-based sonification system for blind spot warning. In: IEEE. *2017 International Conference on Information, Communication and Engineering (ICICE)*. [S.l.], 2017. p. 496–497.

KAZHDAN, Michael; HOPPE, Hugues. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, ACM, v. 32, n. 3, p. 29, 2013.

KAZHDAN MICHAEL, Hoppe Hugues. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, ACM, v. 32, n. 3, p. 29, 2013.

KEAN, Sean; HALL, Jonathan C; PERRY, Phoenix. Microsoft's kinect sdk. In: *Meet the Kinect*. [S.l.]: Springer, 2011. p. 151–173.

KOLB, A. et al. Time-of-flight cameras in computer graphics. *Computer Graphics Forum*, v. 29, n. 1, p. 141–159, 2010. Disponível em: ⟨https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01583.x⟩.

KOVACS, L et al. Three-dimensional recording of the human face with a 3d laser scanner. *Journal of plastic, reconstructive & aesthetic surgery*, Elsevier, v. 59, n. 11, p. 1193–1202, 2006.

LACHAT, Elise et al. Assessment and calibration of a rgb-d camera (kinect v2 sensor) towards a potential use for close-range 3d modeling. *Remote Sensing*, v. 7, p. 13070–13097, 2015.

LANGE, Robert. *3D Time-of-flight distance measurement with custom solid-state image sensors in CMOS/CCD-technology*. 207 p. Tese (Doctor of Technical Sciences) — Department of Eletrical Engineering and Computer Science, June 2000.

LANGE, R.; SEITZ, P. Solid-state time-of-flight range camera. *IEEE Journal of Quantum Electronics*, v. 37, n. 3, p. 390–397, March 2001. ISSN 0018-9197.

LEFLOCH, Damien et al. Technical foundation and calibration methods for time-of-flight cameras. In: _____. *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications: Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New Modalities*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 3–24. ISBN 978-3-642-44964-2. Disponível em: ⟨https://doi.org/10.1007/978-3-642-44964-2_1⟩.

LI, Huixia et al. Research into kinect/inertial measurement units based on indoor robots. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 18, n. 3, p. 839, 2018.

LIN, Ta-Te et al. Development of a virtual reality gis using stereo vision. *Computers and Electronics in Agriculture*, Elsevier, v. 63, n. 1, p. 38–48, 2008.

LITTLE, Francis H; JANNING, John C. *Computed tomography metrology*. [S.l.]: Google Patents, 1998. US Patent 5,848,115.

LOW, Kok-Lim. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, Citeseer, v. 4, n. 10, 2004.

LYSENKOV, Ilya; ERUHIMOV, Victor; BRADSKI, Gary. Recognition and pose estimation of rigid transparent objects with a kinect sensor. *Robotics*, MIT Press, v. 273, 2013.

MAGALHÃES, Guilherme M; PASSARO, Angelo; ABE, Nancy Mieko. Geração de malha de delaunay orientada a objetos. *Anais do Worcomp*, p. 17–18, 2000.

MANKOFF, Kenneth David; RUSSO, Tess Alethea. The kinect: A low-cost, high-resolution, short-range 3d camera. *Earth Surface Processes and Landforms*, Wiley Online Library, v. 38, n. 9, p. 926–936, 2013.

MANSON, Josiah; PETROVA, Guergana; SCHAEFER, Scott. Streaming surface reconstruction using wavelets. In: WILEY ONLINE LIBRARY. *Computer Graphics Forum*. [S.l.], 2008. v. 27, n. 5, p. 1411–1420.

MENDES, Leticia Teixeira; GRIZ, Cristiana; SEDREZ, Maycon. O uso de digitalização 3d em experiências de documentação digital de patrimônio histórico: o caso da sede social do metropolitano de lisboa. *Blucher Design Proceedings*, v. 2, n. 3, p. 776–779, 2015.

MICROSOFT. *Microsoft Kinect for Windows Software Development Kit (SDK) 2.0 End User License Agreement*. 2019. Disponível em: ⟨download.microsoft.com/download/0/D/C/0DC5308E-36A7-4DCD-B299-B01CDFC8E345/kinect-sdk2.0-eula_en-us.pdf⟩.

MONTGOMERY, Douglas C; RUNGER, George C. *Applied statistics and probability for engineers*. [S.l.]: John Wiley & Sons, 2010.

MOSS, JP et al. Three-dimensional visualization of the face and skull using computerized tomography and laser scanning techniques. *European Journal of Orthodontics*, Oxford University Press, v. 9, n. 4, p. 247–253, 1987.

NARAYAN K., Mallikarjuna Rao M.M.M. Sarcar Lalit. *Computer Aided Design and Manufacturing*. 1st. ed. [S.l.]: Asoke K. Ghosh, Prentice-Hall of India Private Limited, 2008. 699 p. ISBN 812033342X.

NIESSNER, Matthias; DAI, Angela; FISHER, Matthew. Combining inertial navigation and icp for real-time 3d surface reconstruction. In: CITESEER. *Eurographics (Short Papers)*. [S.l.], 2014. p. 13–16.

NING, Xiaojuan et al. Segmentation of architecture shape information from 3d point cloud. In: *Proceedings of the 8th International Conference on Virtual Reality Continuum and Its Applications in Industry*. New York, NY, USA: ACM, 2009. (VRCAI '09), p. 127–132. ISBN 978-1-60558-912-1. Disponível em: ⟨http://doi.acm.org/10.1145/1670252.1670280⟩.

OLESEN, O. V. et al. Motion tracking for medical imaging: A nonvisible structured light tracking approach. *IEEE Transactions on Medical Imaging*, v. 31, n. 1, p. 79–87, Jan 2012. ISSN 0278-0062.

OLIVER, Ayrton et al. Using the kinect as a navigation sensor for mobile robotics. In: *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*. New York, NY, USA: ACM, 2012. (IVCNZ '12), p. 509–514. ISBN 978-1-4503-1473-2. Disponível em: ⟨http://doi.acm.org/10.1145/2425836.2425932⟩.

PAGLIARI, Diana; PINTO, Livio. Calibration of kinect for xbox one and comparison between the two generations of microsoft sensors. *Sensors*, MDPI AG, v. 15, n. 12, p. 27569–27589, Oct 2015. ISSN 1424-8220. Disponível em: ⟨http://dx.doi.org/10.3390/s151127569⟩.

PAPADOPOULOS, Georgios Th.; AXENOPOULOS, Apostolos; DARAS, Petros. Real-time skeleton-tracking-based human action recognition using kinect data. In: GURRIN, Cathal et al. (Ed.). *MultiMedia Modeling*. Cham: Springer International Publishing, 2014. p. 473–483. ISBN 978-3-319-04114-8.

PIETRONI, Nico; TARINI, Marco; CIGNONI, Paolo. Almost isometric mesh parameterization through abstract domains. *IEEE Transaction on Visualization and Computer Graphics*, v. 16, n. 4, 2010.

PINHEIRO, Marcelino. *Digitalização 3d e Suas Aplicações no Desenvolvimento ESX-Engenharia*. 2013.

PITERI, Marco Antônio; JUNIOR, MESSIAS MENEGUETTE. Triangulação de delaunay e o princípio de inserção randomizado. *II Simpósio Brasileiro de Geomática-V Colóquio Brasileiro de Ciências Geodésicas, Presidente Prudente-SP*, p. 9, 2007.

POMERLEAU, François; COLAS, Francis; SIEGWART, Roland. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics*, v. 4, p. 1–104, 05 2015.

POMERLEAU, François et al. Comparing icp variants on real-world data sets. *Autonomous Robots*, Springer, v. 34, n. 3, p. 133–148, 2013.

PULLI, Kari et al. Realtime computer vision with opencv. *Queue*, ACM, New York, NY, USA, v. 10, n. 4, p. 40:40–40:56, abr. 2012. ISSN 1542-7730. Disponível em: ⟨http://doi.acm.org/10.1145/2181796.2206309⟩.

REMONDINO, Fabio. From point cloud to surface: the modeling and visualization problem. *International Archives of photogrammetry, Remote Sensing and spatial information sciences*, ETH, Swiss Federal Institute of Technology Zurich, Institute of Geodesy and ..., v. 34, 2003.

ROCKAFELLAR, R. Tyrrell; WETS, Roger J-B. Variational analysis. In: _____. [S.l.]: Springer-Verlag, 2015. p. 117. ISBN 3-540-62772-3.

ROSSI, D. Pagliari; L. Pinto; M. Reguzzoni; L. Integration of kinect and low-cost gnss for outdoor navigation. *THE INTERNATIONAL ARCHIVES OF THE PHOTOGRAMMETRY, REMOTE SENSING AND SPATIAL INFORMATION SCIENCES*, XLI, p. 565 – 572, 2016. Disponível em: ⟨http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B5/565/2016/⟩.

RUSU, Radu Bogdan; COUSINS, Steve. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: [s.n.], 2011.

SARBOLANDI, Damien Lefloch Hamed; KOLB, Andreas. Kinect range sensing: Structured-light versus time-of-flight kinect. *Computer Vision and Image Understanding*, v. 139, p. 1 – 20, 2015. ISSN 1077-3142. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1077314215001071⟩.

SARBOLANDI, Hamed; LEFLOCH, Damien; KOLB, Andreas. Kinect range sensing. *Comput. Vis. Image Underst.*, Elsevier Science Inc., New York, NY, USA, v. 139, n. C, p. 1–20, out. 2015. ISSN 1077-3142. Disponível em: ⟨http://dx.doi.org/10.1016/j.cviu.2015.05.006⟩.

SELL J.; O'CONNOR, P. The xbox one system in a chip and kinect sensor. *Micro IEEE*, v. 34, p. 44 – 53, 2014.

SENIN, N.; COLOSIMO, B.M.; PACELLA, M. Point set augmentation through fitting for enhanced icp registration of point clouds in multisensor coordinate metrology. *Robotics and Computer-Integrated Manufacturing*, v. 29, n. 1, p. 39 – 52, 2013. ISSN 0736-5845. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0736584512000877⟩.

SHEWCHUK, Jonathan Richard. Updating and constructing constrained delaunay and constrained regular triangulations by flips. In: *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*. New York, NY, USA: ACM, 2003. (SCG '03), p. 181–190. ISBN 1-58113-663-3. Disponível em: ⟨http://doi.acm.org/10.1145/777792.777821⟩.

SHI, Quan et al. Registration of point clouds for 3d shape inspection. In: IEEE. *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. [S.l.], 2006. p. 235–240.

SILVESTRE, Ivo et al. Modelaçao 3d de grutas. In: *6th International Conference on Digital Arts–ARTECH 2012*. [S.l.: s.n.], 2012. p. 461–463.

SITEK, A.; HUESMAN, R. H.; GULLBERG, G. T. Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud. *IEEE Transactions on Medical Imaging*, v. 25, n. 9, p. 1172–1179, Sep. 2006. ISSN 0278-0062.

SNYDER, Benn. Computer vision: Object recognition and human-computer interaction. 2013.

SOUZA, Thais Rodrigues de. Curso de Mestrado em Ciências Computacionais, *Digitalização 3D do patrimônio arqueológico metálico utilizando o Kinect*. 2016. 66 p.

TONG, Jing et al. Scanning 3d full human bodies using kinects. *IEEE transactions on visualization and computer graphics*, IEEE, v. 18, n. 4, p. 643–650, 2012.

TRIOLA, Mario F. Introdução à estatística.(tradução vera regina lima de farias e flores). *Revisão técnica Ana Maria Lima de Farias e Flores. Rio de Janeiro: LTC*, 2005.

WASENMÜLLER, Oliver; STRICKER, Didier. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In: SPRINGER. *Asian Conference on Computer Vision*. [S.l.], 2016. p. 34–45.

WEBB, Jarrett; ASHLEY, James. *Beginning Kinect Programming with the Microsoft Kinect SDK*. [S.l.]: Apress, 2012.

WOODS, Rafael C. Gonzalez; Richard E. *Processamento Digital de Imagens*. 3rd. ed. [S.l.]: Pearson Prentice Hall, 2010. 624 p. ISBN 978-85-7605-401-6.

WU, Shihao et al. Quality-driven poisson-guided autoscanning. *ACM Transactions on Graphics*, v. 33, n. 6, 2014.

XIANG, FE Lingzhu; KERL, Christian; WIEDEMEYER, Thiemo. *Lars, hanyazou, Alistair: libfreenect2: Release 0.2 [Data set], Zenodo*. 2016.

XIANG, Lingzhu et al. *libfreenect2: Release 0.2*. 2016. Disponível em: ⟨https://doi.org/10.5281/zenodo.50641⟩.

YANG, Lin et al. Evaluating and improving the depth accuracy of kinect for windows v2. *IEEE Sensors Journal*, IEEE, v. 15, n. 8, p. 4275–4285, 2015.

YOSHITAKA, Hara et al. Mobile robot localization and mapping by scan matching using laser reflection intensity of the sokuiki sensor. In: IEEE. *IEEE Industrial Electronics, IECON 2006-32nd Annual Conference on*. [S.l.], 2006. p. 3018–3023.

ZENNARO, Simone. Corso di Laurea Magistrale in Ingegneria Informatica, *Evaluation of Microsoft Kinect 360 and Microsoft Kinect One for robotics and computer vision applications*. 2014. 74 p.

ZHOU, Jia-Heng; LIN, Huei-Yung. A self-localization and path planning technique for mobile robot navigation. In: IEEE. *Intelligent Control and Automation (WCICA), 2011 9th World Congress on*. [S.l.], 2011. p. 694–699.

GLOSSÁRIO

Kinect	Sensor desenvolvido pela *Microsoft*, que permite a aquisção de imagens RGB, infravermelho e de profundidade simultaneamente, com uma alta taxa de *frames*.
Kinect One	Nova versão do *Kinect* da *Microsoft* baseada na tecnologia *time of flight*.
time of flight	Tecnologia baseada na medição do tempo que um sinal emitido demora para se locomover até a cena e retornar para o sensor.
laser scanner	Sensor de alto desempenho e elevado custo financeiro. Seu preço normalmente varia de US$50.000 a US$410.000.
XBOX One	Console e plataforma padrão de funcionamento do *Kinect One*.
Windows	Sistema operacional principal da *Microsoft*, para o qual o *Kinec One* recebe suporte da empresa.
JuneSDK	Driver oficial do *Kinect One* disponibilizado pela *Microsoft*. Também conhecido como *Kinect for Windows SDK 2.0*.
luz estru.	Luz Estruturada: Tecnologia da primeira versão do *Kinect*, é focada no processo de projetar um padrão conhecido (muitas vezes grades ou barras horizontais) em uma cena.
libfreenect2	Driver e biblioteca para a utilização do *Kinect One* no linux. Obtida através de engenharia reversa por John Blake.
open source	É um modelo de desenvolvimento criado em 1998, que promove o licenciamento livre para o *design* ou esquematização de um produto, e a redistribuição universal desses, com a possibilidade de livre consulta, examinação ou modificação do produto, sem a necessidade de pagar uma licença comercial, promovendo um modelo colaborativo de produção intelectual.
ARUCO	Biblioteca de alvos codificados flexível, de código aberto e com capacidade de representar até 1.024 alvos diferentes.
OpenCV	Biblioteca muito conhecida de processamento de imagens, que possui uma quantidade elevada de algoritmos de visão computacional implementados.
PCL	Biblioteca de processamento e manipulação de nuvens de pontos.
Meshlab	Software para fusão e manipulação de nuvens de pontos, assim com geração de modelos 3D. Possui uma grande quantidade de algoritmos para manipulação das nuvens de pontos.
R. Esqueleto	Rastreamento de Esqueleto: Campo amplamente estudado e um tópico muito ativo à comunidade de pesquisa em visão computacional.
R. de Gestos	Reconhecimento de Gestos: Tópico em ciência da computação e tecnologia de idioma (*language technology*) com o objetivo de interpretar gestos humanos através de algoritmos matemáticos.

N. de Pontos Nuvens de Pontos: Conjunto de pontos no espaço, que normalmente é definido por um sistema de coordenadas.

Kinect Fusion Biblioteca desenvolvida pela *Microsoft* com capacidade de gerar uma digitalização automática da área escaneada em tempo real, incluindo na forma de nuvens de pontos.

l. superior Limite Superior: marcação ARUCO que se encontra mais elevada em relação as demais detectadas durante o processo de segmentação.

l. inferior Limite Inferior: marcação ARUCO que se encontra mais abaixo em relação as demais detectadas durante o processo de segmentação.

l. direito Limite Direito: marcação ARUCO que se encontra mais a direita em relação as demais detectadas durante o processo de segmentação.

l. esquerdo Limite Esquerdo: marcação ARUCO que se encontra mais a esquerda em relação as demais detectadas durante o processo de segmentação.

ICP *Iterative Closest Point*, é o algoritmo empregado para minimizar a diferença entre duas nuvens de pontos e é comumente usado na reconstrução de superfícies 2D e 3D.

outliers Valor aberrante ou valor atípico. É uma observação que apresenta um grande afastamento das demais da série, ou que é uma observação inconsistente.

Distância H. Distância de Hausdorff: Métrica que calcula o quão distante dois conjuntos de pontos estão um do outro.

F. Hausdorff Felix Hausdorff: Matemático alemão considerado um dos fundadores da topologia moderna e que contribuiu significativamente para a teoria dos conjuntos.

M. Fréchet Maurice Fréchet: Matemático Francês que fez grandes contribuições para a topologia de conjuntos de pontos e introduziu todo o conceito de espaços métricos.

erro médio média aritmética de todos os erros encontrados durante o cálculo da distância de uma nuvem para a outra.

variância Medida de dispersão que mostra o quão distante cada valor do conjunto de distâncias calculadas encontra-se do valor médio.

desvio padrão Raiz quadrada positiva da variância. Ele representa o quão confiável é o valor medido e informa o erro em um conjunto de dados, caso fosse substituído um dos valores do conjunto pela média aritmética.

S. Poisson Screened Poisson: Algoritmo para reconstrução de um modelo 3D a partir de uma nuvem de pontos, gerando uma malha.

T. Delaunay Triangulação de Delaunay: Algoritmo para reconstrução de um modelo 3D a partir de uma nuvem de pontos, gerando uma triangulação.

APÊNDICE A – Códigos

Este apêndice apresenta todos os programas que foram desenvolvidos para esse trabalho e citados ao longo do texto. Todos eles foram escritos da linguagem C++ e utilizam as bibliotecas descritas no capítulo 2.

A.1 Captura Inicial

O programa faz a captura inicial dos dados com o kinect. Após o *setup* já preparado com a mesa e o kinect já conectado ao computador, o programa fará a captura das nuvens de pontos e imagens da posição relativa do sensor.

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <signal.h>
4  #include <fstream>
5  #include <string>
6  #include <sstream>
7  #include <vector>
8  #include <pcl/io/pcd_io.h>
9  #include <pcl/point_types.h>
10 #include <pcl/filters/passthrough.h>
11 #include <opencv2/opencv.hpp>
12 #include <opencv2/aruco.hpp>
13 //#include "limiar.cpp" //classe designada para a deteccao dos
      limites de uma area de interesse, baseado nas marcacoes aruco
      detectadas
14 //#include "corte.cpp"
15 #include "MatrizInversa.cpp"
16
17
18 /// [headers]
19 #include <libfreenect2/libfreenect2.hpp>
20 #include <libfreenect2/frame_listener_impl.h>
21 #include <libfreenect2/registration.h>
22 #include <libfreenect2/packet_pipeline.h>
23 #include <libfreenect2/logger.h>
24 /// [headers]
25 #ifdef EXAMPLES_WITH_OPENGL_SUPPORT
```

```
26 #include "viewer.h"
27 #endif
28
29 //#include <convert.cpp>
30
31 using namespace cv;
32
33 bool protonect_shutdown = false; ///< Whether the running
      application should shut down
34 bool protonect_paused   = false;
35 libfreenect2::Freenect2Device *devtopause;
36
37 /// [main]
38 /**
39  * Main application entry point.
40  */
41 int main(int argc, char *argv[])
42 /// [main]
43 {
44
45 /// [context]
46   libfreenect2::Freenect2       freenect2;
47   libfreenect2::Freenect2Device *dev = 0;
48   libfreenect2::PacketPipeline  *pipeline = 0;
49 /// [context]
50
51   std::string serial = "";
52
53   bool   viewer_enabled = true;
54   bool   enable_rgb     = true;
55   bool   enable_depth   = true;
56 //  int    deviceId       = -1;
57   size_t framemax       = -1;
58
59
60 /// [discovery]
61   if(freenect2.enumerateDevices() == 0)
62   {
63     std::cout << "no device connected!" << std::endl;
64     return -1;
65   }
```

```
66
67    if (serial == "")
68    {
69      serial = freenect2.getDefaultDeviceSerialNumber();
70    }
71  /// [discovery]
72
73    if(pipeline)
74    {
75  /// [open]
76      dev = freenect2.openDevice(serial, pipeline); //abre o device
          com um pipeline especifico, como openGL, openCL, CPU e etc
77  /// [open]
78    }
79    else
80    {
81      dev = freenect2.openDevice(serial); //abre o device com o
          pipeline padrao
82    }
83
84    if(dev == 0) //se nao conseguir abrir o device por qualquer
        motivo
85    {
86      std::cout << "failure opening device!" << std::endl;
87      return -1;
88    }
89
90    devtopause = dev;
91
92    protonect_shutdown = false;
93
94  /// [listeners]
95    int types = 0;
96
97    if (enable_rgb)
98      types |= libfreenect2::Frame::Color;
99    if (enable_depth)
100     types |= libfreenect2::Frame::Ir | libfreenect2::Frame::Depth
          ;
101
102   libfreenect2::SyncMultiFrameListener listener(types);
```

```
103   libfreenect2::FrameMap frames;
104
105   dev->setColorFrameListener(&listener);
106   dev->setIrAndDepthFrameListener(&listener);
107 /// [listeners]
108
109 /// [start]
110   if (enable_rgb && enable_depth)
111   {
112     if (!dev->start())
113       return -1;
114   }
115   else
116   {
117     if (!dev->startStreams(enable_rgb, enable_depth))
118       return -1;
119   }
120
121   std::cout << "device serial: " << dev->getSerialNumber() << std
        ::endl;
122   std::cout << "device firmware: " << dev->getFirmwareVersion()
        << std::endl;
123 /// [start]
124
125 /// [registration setup]
126   libfreenect2::Registration* registration = new libfreenect2::
        Registration(dev->getIrCameraParams(), dev->
        getColorCameraParams());
127   libfreenect2::Frame undistorted(512, 424, 4), registered(512,
        424, 4);
128 /// [registration setup]
129
130 viewer_enabled = false;
131
132   size_t framecount = 0;
133 #ifdef EXAMPLES_WITH_OPENGL_SUPPORT
134   Viewer viewer;
135   if (viewer_enabled)
136     viewer.initialize();
137 #else
138   viewer_enabled = false;
```

```
139 #endif
140
141
142 ///aquisicao de frames
143
144
145         std::cout << "Criacao e aquisicao de Frames
                !!!!!!!!!!!!!!!!!!!!!!!!!!!!\n";
146
147
148         ///criacao das imagens que receberao os frames
149         cv::Mat rgbMat;
150         cv::Mat irMat;
151         cv::Mat depthMat;
152
153         ///informa ao kinect o inicio da captura de frame
154         listener.waitForNewFrame(frames);
155
156         ///instancia os frames com os valores capturados do
                kinect
157         libfreenect2::Frame *rgb = frames[libfreenect2::Frame::
                Color];
158         libfreenect2::Frame *ir = frames[libfreenect2::Frame::Ir
                ];
159         libfreenect2::Frame *depth = frames[libfreenect2::Frame::
                Depth];
160
161         ///copia os frames ja capturados para as respectivas
                imagens
162         cv::Mat(rgb->height, rgb->width, CV_8UC4, rgb->data).
                copyTo(rgbMat);
163         cv::Mat(ir->height, ir->width, CV_32FC1, ir->data).copyTo
                (irMat); //a faixa de valores do infravermelho    de 0
                a 65535 (maior que a faixa tons de cinza 0 a 255)
164         cv::Mat(depth->height, depth->width, CV_32FC1, depth->
                data).copyTo(depthMat);
165
166         ///salva as imagens
167         cv::imwrite("imagemRGB.png",rgbMat);
168         cv::imwrite("imagemIR.png",irMat);
169         cv::imwrite("imagemDEPTH.png",depthMat);
```

```
170
171         ///aplica o registry para que seja gerado os frames '
                undistorted' e 'registered'
172         libfreenect2::Frame bigDepth(1920, 1082, 4);
173         registration->apply(rgb, depth, &undistorted, &registered
                , true, &bigDepth, 0);
174
175         ///cria as imagens que receberao os frames 'undistorted',
                 'registered' e 'bigDepth'
176         cv::Mat undistortedMat;
177         cv::Mat registeredMat;
178         cv::Mat bigDepthMat;
179
180         ///copia os frames gerados pelo aply do registry para as
                respectivas imagens
181         cv::Mat(undistorted.height, undistorted.width, CV_32FC1,
                undistorted.data).copyTo(undistortedMat);
182         cv::Mat(registered.height, registered.width, CV_8UC4,
                registered.data).copyTo(registeredMat);
183         cv::Mat(bigDepth.height, bigDepth.width, CV_32FC1,
                bigDepth.data).copyTo(bigDepthMat);
184
185         ///salva as imagens 'undistorted', 'registered' e '
                bigDepth'
186         cv::imwrite("imagemUNDISTORTED.png",undistortedMat);
187         cv::imwrite("imagemREGISTERED.png",registeredMat);
188         cv::imwrite("imagemBIGDEPTH.png",bigDepthMat); //por
                algum motivo n o funciona
189
190         framecount++;
191         std::cout << "framecount: " << framecount << "\n\n";
192
193
194
195
196 ///teste calculo da distancia
197 /*
198         float x,y,z;
199         libfreenect2::Frame * ponteiroFrame;
200         ponteiroFrame = &undistorted;
201         registration->getPointXYZ(ponteiroFrame,253,352,x,y,z);
```

```
202         std::cout << "as coordenadas espaciais do pixel(253,352)
               no frame undistorted sao: " << x << " " << y << " " <<
               z << "\n\n";
203
204         float bx,by,bz;
205         libfreenect2::Frame * ponteiroFrame2;
206         ponteiroFrame2 = &bigDepth;
207         registration->getPointXYZ(ponteiroFrame2,253,352,bx,by,bz
               );
208         std::cout << "as coordenadas espaciais do pixel(253,352)
               no frame bigDepth sao: " << x << " " << y << " " << z
               << "\n\n"; //possui um erro que exibe x,y,z e nao bx,by
               ,bz como deveria
209
210 */
211
212
213
214
215 ///teste de exibicao dos parametros de calibracao
216 /*
217 libfreenect2::Freenect2Device::ColorCameraParams parametrosCor =
       dev->getColorCameraParams();
218 libfreenect2::Freenect2Device::IrCameraParams parametrosIr = dev
       ->getIrCameraParams();
219
220 std::cout << "\n\nOs parametros internos da camera colorida sao:
       \ncoordenadas do ponto principal em pixels (x,y): " <<
       parametrosCor.cx << " " << parametrosCor.cy << "\n"
221           << "distancia focal em pixels (x,y): " << parametrosCor
                 .fx << " " << parametrosCor.fy << "\n\n";
222
223 std::cout << "Os parametros da camera infra-vermelho sao: \
       ncoordenadas do ponto principal em pixels (x,y): " <<
       parametrosIr.cx << " " << parametrosIr.cy << "\n"
224       << "dist ncia focal em pixels (x,y): " << parametrosIr.fx
             << " " << parametrosIr.fy << "\n\n";
225 */
226
227
228 ///deteccao das marcacoes aruco na imagem
```

```
229 /*
230
231         //cv::Mat imagem = cv::imread( argv[1] );
232
233         //if ( !imagem.data )
234         //{
235         //      printf("No image data \n");
236         //      return -1;
237         //}
238
239         cv::Mat imagem = registeredMat; //utilizando o frame
               registered que tem a informacao de cor para a deteccao
               das marcacoes e possui o mesmo tamanho do frame de
               profundidade, permitindo um mapeamento direto
240
241         //deteccao das marcacoes na imagem
242         cv::aruco::Dictionary dicionario = cv::aruco::
               getPredefinedDictionary(cv::aruco::DICT_4X4_50);
243         std::vector< int > idsMarcas;
244         std::vector< std::vector<Point2f> > cantosMarcas;
245         std::vector< std::vector<Point2f> > marcasRejeitadas;
246         cv::aruco::DetectorParameters parametrosDetector;
247
248         cv::aruco::detectMarkers( imagem , dicionario ,
               cantosMarcas , idsMarcas , parametrosDetector ,
               marcasRejeitadas );
249
250         //desenha marcas na imagem que teve marcacoes detectadas:
251
252         cv::Mat imagemMarcada = imagem;
253
254         cv::aruco::drawDetectedMarkers(imagemMarcada,
               cantosMarcas, idsMarcas);
255         cv::imwrite( "imagemMarcada.png" , imagemMarcada );
256
257         if(cantosMarcas.empty()) std::cout << "\n\nVETOR DE
               CANTOS DAS MARCAS VAZIO!!!!!!\n";
258         std::cout << "\n\nA quantidade de marcas detectadas e: "
               << cantosMarcas.size() << "\n\n";
259
```

```
260         //imprime os valores dos cantos das marcas detectadas,
               apenas se o vetor de marcas conter marcacoes
261
262         if( cantosMarcas.size() > 0 )
263         {
264                 std::cout << "O vetor de cantos das marcas
                       detectadas contem:\n\n";
265
266                 cv::Point2f * ponteiroCantos;
267                 std::vector<Point2f> * ponteiroMarcas;
268                 ponteiroMarcas = &(cantosMarcas.front()); //
                       recebe a primeira marca do vetor
269                 ponteiroCantos = &((*ponteiroMarcas).front()); //
                       recebe o primeiro canto da primeira marca
270                 for( int i = 1 ; i <=cantosMarcas.size() ; i++ )
271                 {
272                         for( int j = 1 ; j <= 4 ; j++ )
273                         {
274                                 std::cout << *ponteiroCantos <<
                                       "\n";
275                                 ponteiroCantos++;
276                         }
277                         ponteiroMarcas++;
278                         ponteiroCantos = &((*ponteiroMarcas).
                               front());
279                         std::cout << "\n";
280                 }
281
282
283                 //imprime os valores dos ids das marcas
                       detectadas
284
285                 std::cout << "O vetor de ids das marcas
                       detectadas contem:\n\n";
286
287                 int * ponteiroIds;
288                 ponteiroIds = &(idsMarcas.front());
289                 for( int i = 1 ; i <= idsMarcas.size() ; i++ )
290                 {
291                         std::cout << *ponteiroIds << "\n";
292                         ponteiroIds++;
```

```
293                 }
294                 std::cout << "\n\n";
295         }
296
297         //imprime os valores dos cantos das marcas rejeitadas, se
                houver alguma
298         if( marcasRejeitadas.size() > 0 )
299         {
300                 std::cout << "O vetor de cantos das marcas
                    rejeitadas contem:\n\n";
301
302                 cv::Point2f * ponteiroCantos2;
303                 std::vector<Point2f> * ponteiroMarcas2;
304                 ponteiroMarcas2 = &(marcasRejeitadas.front());
305                 ponteiroCantos2 = &((*ponteiroMarcas2).front());
306                 for( int i = 1 ; i <=marcasRejeitadas.size() ; i
                    ++ )
307                 {
308                         for( int j = 1 ; j <= 4 ; j++ )
309                         {
310                                 std::cout << *ponteiroCantos2 <<
                                    "\n";
311                                 ponteiroCantos2++;
312                         }
313                         ponteiroMarcas2++;
314                         ponteiroCantos2 = &((*ponteiroMarcas2).
                            front());
315                         std::cout << "\n";
316                 }
317
318                 //imprime imagem marcada com marcacoes rejeitadas
319                 cv::Mat imagemMarcadaRejeitada = imagem;
320
321                 for(std::vector< vector<Point2f> >::iterator it =
                     marcasRejeitadas.begin(); it !=
                    marcasRejeitadas.end(); ++it)
322                 {
323                         vector<Point2f> sqPoints = *it;
324                         //cout << sqPoints.size() << endl;
325                         //Point pt2(it[1].x, it[1].y);
```

```
326                         line(imagemMarcadaRejeitada, sqPoints[0],
                                sqPoints[1], CV_RGB(255, 0 , 0));
327                         line(imagemMarcadaRejeitada, sqPoints[2],
                                sqPoints[1], CV_RGB(255, 0 , 0));
328                         line(imagemMarcadaRejeitada, sqPoints[2],
                                sqPoints[3], CV_RGB(255, 0 , 0));
329                         line(imagemMarcadaRejeitada, sqPoints[0],
                                sqPoints[3], CV_RGB(255, 0 , 0));
330                 }
331
332                 cv::imwrite( "imagemMarcadaRejeitada.png" ,
                        imagemMarcadaRejeitada );
333         }
334 */
335
336
337
338 ///Calculo dos pontos detectados, corte e filtro
339 /*
340         //utilizar o frame infravermelho para mapeamento direto
                com frame undistorted e evitar as conversoes
341
342         std::vector<Point2f> * ponteiroMarcasDetectadas;
343         ponteiroMarcasDetectadas = &(cantosMarcas.front()); //
                recebe a primeira marca do vetor
344
345         cv::Point2f pontos[3];
346
347         for( int i = 1 ; i <= 3 ; i++ ) //pega apenas as 3
                primeiras marcacoes detectadas, (canto superior
                esquerdo)
348         {
349                 pontos[i-1] = (*ponteiroMarcasDetectadas).front()
                        ;
350                 ponteiroMarcasDetectadas++;
351         }
352
353         //calculo da matriz X
354
355         double X[9];
356         double x, y, z;
```

```
357
358         libfreenect2::Frame * ponteiroFrame;
359         ponteiroFrame = &undistorted;
360
361         for( int i = 0 ; i < 3 ; i++ ) //percorre os 3 pontos do
               vetor de pontos
362         {
363                 registration->getPointXYZ( ponteiroFrame , pontos
                    [i].y , pontos[i].x , x , y , z ); //acertar os
                     pontos
364                 X[i]   = x;
365                 X[i+3] = y;
366                 X[i+6] = z;
367         }
368
369
370         double inversaX[9];
371         int ordem = 3;
372
373         MatrizInversa(inversaX, X, ordem);
374
375         // Exibe a matriz inversa de A
376         printf ("\n\nInversa de X =");
377         for (int i = 0; i < 9; i++)
378         {
379                 if( i % 3 == 0) printf("\n");
380                 //printf("%13G", inversaA[i]);
381                 std::cout << inversaA[i] << "  ";
382         }
383         std::cout << "\n\n";
384 */
385
386
387
388 ///Criacao da nuvem de pontos completa e do arquivo com os
       valores de profundidade de cada ponto do frame
389
390         libfreenect2::Frame * ponteiroFrameU; //cria um ponteiro
               para o frame undistorted
391         ponteiroFrameU = &undistorted;
392
```

```
393         pcl::PointCloud<pcl::PointXYZ> cloud;
394 
395         pcl::PointXYZ pontoNovo;
396 
397         float x2, y2, z2;
398 
399         std::ofstream arquivoNuvem;
400         arquivoNuvem.open("arquivoNuvem.txt",ios::out);
401 
402         for ( int i = 0 ; i < depth->height ; i++ ) //i=linha
403         {
404                 for ( int j = 0 ; j < depth->width ; j++) //j=
                        coluna
405                 {
406 
407                 registration->getPointXYZ( ponteiroFrameU , i , j
                         , x2 , y2 , z2 ); //ponteiro frame U est
                        apontando para undistorted
408                 // os valores adquiridos de x2,y2,z2 est o em
                        metros
409                 pontoNovo.x = x2;
410                 pontoNovo.y = y2;
411                 pontoNovo.z = z2;
412 
413                 cloud.push_back(pontoNovo); //adiciona o ponto
                        novo a nuvem de pontos
414 
415                 //adicao dos pontos ao arquivo
416                 arquivoNuvem << x2 << " " << y2 << " " << z2 << "
                        \n"; //os pontos sao armazenados no arquivo de
                        forma seguencial
417                 //o loop percorre a matriz de linha em linha
418                 }
419         }
420 
421         arquivoNuvem.close();
422 
423         pcl::io::savePCDFileASCII ("nuvem_completa.pcd", cloud);
424         std::cerr << "\nSaved " << cloud.points.size () << " data
                points to nuvem_completa.pcd." << std::endl;
425 
```

```
426
427
428
429 ///criacao da imagem cujas marcacoes serao detectadas
430
431         //cv::Mat imagemParaDeteccao = cv::Mat::zeros( depthMat.
               size(), depthMat.type() ); //talvez remover o type caso
                nao funcione
432         cv::Mat imagemParaDeteccao = cv::Mat::zeros( cv::Size(
               depth->width , depth->height) , CV_8UC3 );
433
434         libfreenect2::Frame * ponteiroFrameR; //cria um ponteiro
               para o frame registered
435         ponteiroFrameR = &registered;
436
437         float valorRGB;
438
439         for ( int i = 0 ; i < depth->height ; i++ )
440         {
441                 for ( int j = 0 ; j < depth->width ; j++)
442                 {
443
444                 registration->getPointXYZRGB( ponteiroFrameU ,
                       ponteiroFrameR , i , j , x2 , y2 , z2 ,
                       valorRGB ); //ponteiro frame U esta apontando
                       para undistorted e ponteiroFrameR esta
                       apontando para registered
445                 // os valores adquiridos de x2,y2,z2 estao em
                       metros
446
447                 const uint8_t *p = reinterpret_cast<uint8_t*>(&
                       valorRGB);
448                 uint8_t b = p[0];
449                 uint8_t g = p[1];
450                 uint8_t r = p[2];
451
452                 imagemParaDeteccao.at<Vec3b>(i,j)[0] = b;
453                 imagemParaDeteccao.at<Vec3b>(i,j)[1] = g;
454                 imagemParaDeteccao.at<Vec3b>(i,j)[2] = r;
455
456                 }
```

```
457         }
458
459         cv::imwrite("imagemParaDeteccao.png",imagemParaDeteccao);
460
461
462
463
464 /// [stop]
465   dev->stop();
466   dev->close();
467 /// [stop]
468
469   delete registration;
470
471   return 0;
472 }
```

```
1  CMAKE_MINIMUM_REQUIRED(VERSION 3.1)
2
3  IF(NOT DEFINED CMAKE_BUILD_TYPE)
4    # No effect for multi-configuration generators (e.g. for Visual
       Studio)
5    SET(CMAKE_BUILD_TYPE RelWithDebInfo CACHE STRING "Choose:
       RelWithDebInfo Release Debug MinSizeRel None")
6  ENDIF()
7
8  PROJECT(libfreenect2_examples)
9
10 SET(MY_DIR ${libfreenect2_examples_SOURCE_DIR})
11 SET(DEPENDS_DIR "${MY_DIR}/../depends" CACHE STRING "Dependency
     directory")
12
13 OPTION(ENABLE_OPENGL "Enable OpenGL support" ON)
14
15 # The example build system is standalone and will work out-of-
     tree with these files copied
16 SET(freenect2_ROOT_DIR ${MY_DIR}/..)
17 SET(flextGL_SOURCES ${freenect2_ROOT_DIR}/src/flextGL.cpp)
18 SET(flextGL_INCLUDE_DIRS ${freenect2_ROOT_DIR}/src) # for flextGL
     .h
19
20 FIND_PACKAGE(PkgConfig)    # try find PKGConfig as it will be
```

```
   used if found
LIST(APPEND CMAKE_MODULE_PATH ${freenect2_ROOT_DIR}/cmake_modules
   ) # FindGLFW3.cmake

find_package( OpenCV REQUIRED )
#SET("OpenCV_DIR" "C:/opencv/opencv320/build/x64/vc14/lib")


#find_package( aruco   REQUIRED )
find_package(PCL 1.3 REQUIRED COMPONENTS common io)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})

include_directories("${PROJECT_SOURCE_DIR}")



IF(TARGET freenect2)
  MESSAGE(STATUS "Using in-tree freenect2 target")
  SET(freenect2_LIBRARIES freenect2)
  SET(freenect2_DLLS ${LIBFREENECT2_DLLS})
ELSE()
  FIND_PACKAGE(freenect2 REQUIRED)
  # Out-of-tree build will have to have DLLs manually copied.
ENDIF()

INCLUDE_DIRECTORIES(
  ${freenect2_INCLUDE_DIR}
)

include_directories( ${OpenCV_INCLUDE_DIRS} )

SET(Protonect_src
  Protonect.cpp
)

SET(Protonect_LIBRARIES
  ${freenect2_LIBRARIES}
)

```

```
SET(Protonect_DLLS
  ${freenect2_DLLS}
)

SET(GCC_COVERAGE_COMPILE_FLAGS "-fexceptions")

add_definitions(${GCC_COVERAGE_COMPILE_FLAGS})

IF(ENABLE_OPENGL)
  FIND_PACKAGE(GLFW3)
  FIND_PACKAGE(OpenGL)
  IF(GLFW3_FOUND AND OPENGL_FOUND)
    INCLUDE_DIRECTORIES(
      ${GLFW3_INCLUDE_DIRS}
      ${flextGL_INCLUDE_DIRS}
    )

    LIST(APPEND Protonect_DLLS ${GLFW3_DLL})
    LIST(APPEND Protonect_src
      viewer.cpp
      ${flextGL_SOURCES}
    )
    LIST(APPEND Protonect_LIBRARIES
      ${GLFW3_LIBRARIES}
      ${OPENGL_gl_LIBRARY}
    )
    ADD_DEFINITIONS(-DEXAMPLES_WITH_OPENGL_SUPPORT=1)
  ENDIF()
ENDIF(ENABLE_OPENGL)

ADD_EXECUTABLE(Protonect
  ${Protonect_src}
)

TARGET_LINK_LIBRARIES(Protonect
  ${Protonect_LIBRARIES}
)
target_link_libraries(Protonect ${PCL_COMMON_LIBRARIES} ${
   PCL_IO_LIBRARIES})

target_link_libraries( Protonect ${OpenCV_LIBS} )
```

```
100 #target_link_libraries( Protonect ${aruco_LIBS}  )
```

A.2 Detecção de Marcações

O segundo programa recebe como entrada as imagens capturadas pelo primeiro programa e detecta as marcações ARUCO contidas nelas. Pelo menos 3 marcações precisam ser detectadas para que seja possível prosseguir com o corte e segmentação no prgrama seguinte.

Esse segundo programa irá gerar um arquivo contendo todas as marcações detectadas que serão usadas no programa seguinte.

```
1  #include <opencv2/aruco/charuco.hpp>
2  #include <opencv2/opencv.hpp>
3  #include <opencv2/core/core.hpp>
4  #include <opencv2/imgcodecs.hpp>
5  #include <opencv2/highgui/highgui.hpp>
6  #include <iostream>
7  #include <fstream>
8  #include <string>
9  //#include "types.hpp"
10
11 using namespace std;
12 using namespace cv;
13
14 int main( int argc, char** argv )
15 {
16 ///deteccao das marcacoes aruco na imagem
17
18
19         //leitura da imagem de entrada
20         cv::Mat imagem = cv::imread( argv[1] );
21
22         if ( !imagem.data )
23         {
24                 printf("No image data \n");
25                 return -1;
26         }
27
28         //deteccao das marcacoes na imagem
29         cv::aruco::Dictionary dicionario = cv::aruco::
```

```
        getPredefinedDictionary(cv::aruco::DICT_4X4_50);
30      std::vector< int > idsMarcas;
31      std::vector< std::vector<Point2f> > cantosMarcas;
32      std::vector< std::vector<Point2f> > marcasRejeitadas;
33      cv::aruco::DetectorParameters parametrosDetector;
34
35      cv::aruco::detectMarkers( imagem , dicionario ,
        cantosMarcas , idsMarcas , parametrosDetector ,
        marcasRejeitadas );
36
37      //desenha marcas na imagem que teve marcacoes detectadas:
38
39      cv::Mat imagemMarcada = imagem;
40
41      cv::aruco::drawDetectedMarkers(imagemMarcada,
        cantosMarcas, idsMarcas);
42      cv::imwrite( "imagemMarcada.png" , imagemMarcada );
43
44      if(cantosMarcas.empty()) std::cout << "\n\nVETOR DE
        CANTOS DAS MARCAS VAZIO!!!!!!\n";
45      std::cout << "\n\nA quantidade de marcas detectadas e: "
        << cantosMarcas.size() << "\n\n";
46
47      //imprime os valores dos cantos das marcas detectadas,
        apenas se o vetor de marcas conter marcacoes
48
49      if( cantosMarcas.size() > 0 )
50      {
51              std::cout << "O vetor de cantos das marcas
                detectadas contem:\n\n";
52
53              cv::Point2f * ponteiroCantos;
54              std::vector<Point2f> * ponteiroMarcas;
55              int * ponteiroIdsTemp; //ponteiro que percorre o
                vetor de ids
56              ponteiroMarcas  = &(cantosMarcas.front()); //
                recebe a primeira marca do vetor
57              ponteiroCantos  = &((*ponteiroMarcas).front());
                //recebe o primeiro canto da primeira marca
58              ponteiroIdsTemp = &(idsMarcas.front());
59
```

```
60                  std::ofstream arquivoPontos;
61                  arquivoPontos.open( "arquivoPontos.txt" , ios::
                       out );
62                  cv::Point2f cantos[4];
63
64                  for( int i = 1 ; i <=cantosMarcas.size() ; i++ )
65                  {
66                          //insere os pontos detectados no arquivo
67                          if( *ponteiroIdsTemp == 27 || *
                               ponteiroIdsTemp == 42 || *
                               ponteiroIdsTemp == 43 )
68                          {
69                                  arquivoPontos << (*ponteiroCantos
                                       ).y << " " << ( (*
                                       ponteiroCantos).x) << "\n";
70
71                                  for( int j = 1 ; j <= 4 ; j++ )
72                                  {
73                                          cantos[j-1] = *
                                               ponteiroCantos;
74                                          ponteiroCantos++;
75                                  }
76
77
78                          }
79
80
81                          ponteiroCantos = &((*ponteiroMarcas).
                               front());
82
83                          //faz a impressao propriamente dita dos
                               pontos detectados
84                          for( int j = 1 ; j <= 4 ; j++ )
85                          {
86                                  std::cout << *ponteiroCantos << "
                                       \n";
87                                  ponteiroCantos++;
88                          }
89                          ponteiroMarcas++;
90                          ponteiroCantos = &((*ponteiroMarcas).
                               front());
```

```
91                      ponteiroIdsTemp++;
92                      std::cout << "\n";
93              }
94
95
96              //imprime os valores dos ids das marcas
                   detectadas
97
98              std::cout << "O vetor de ids das marcas
                   detectadas contem:\n\n";
99
100             int * ponteiroIds;
101             ponteiroIds = &(idsMarcas.front());
102             for( int i = 1 ; i <= idsMarcas.size() ; i++ )
103             {
104                     std::cout << *ponteiroIds << "\n";
105                     ponteiroIds++;
106             }
107             std::cout << "\n\n";
108
109             arquivoPontos.close();
110     }
111
112     //imprime os valores dos cantos das marcas rejeitadas, se
            houver alguma
113     if( marcasRejeitadas.size() > 0 )
114     {
115             std::cout << "O vetor de cantos das marcas
                   rejeitadas contem:\n\n";
116
117             cv::Point2f * ponteiroCantos2;
118             std::vector<Point2f> * ponteiroMarcas2;
119             ponteiroMarcas2 = &(marcasRejeitadas.front());
120             ponteiroCantos2 = &((*ponteiroMarcas2).front());
121             for( int i = 1 ; i <=marcasRejeitadas.size() ; i
                   ++ )
122             {
123                     for( int j = 1 ; j <= 4 ; j++ )
124                     {
125                             std::cout << *ponteiroCantos2 <<
                                   "\n";
```

```
126                                 ponteiroCantos2++;
127                         }
128                         ponteiroMarcas2++;
129                         ponteiroCantos2 = &((*ponteiroMarcas2).
                               front());
130                         std::cout << "\n";
131                 }
132
133                 //imprime imagem marcada com marcacoes rejeitadas
134                 cv::Mat imagemMarcadaRejeitada = imagem;
135
136                 for(std::vector< vector<Point2f> >::iterator it =
                        marcasRejeitadas.begin(); it !=
                       marcasRejeitadas.end(); ++it)
137                 {
138                         vector<Point2f> sqPoints = *it;
139                         //cout << sqPoints.size() << endl;
140                         //Point pt2(it[1].x, it[1].y);
141                         line(imagemMarcadaRejeitada, sqPoints[0],
                                sqPoints[1], CV_RGB(255, 0 , 0));
142                         line(imagemMarcadaRejeitada, sqPoints[2],
                                sqPoints[1], CV_RGB(255, 0 , 0));
143                         line(imagemMarcadaRejeitada, sqPoints[2],
                                sqPoints[3], CV_RGB(255, 0 , 0));
144                         line(imagemMarcadaRejeitada, sqPoints[0],
                                sqPoints[3], CV_RGB(255, 0 , 0));
145                 }
146
147                 cv::imwrite( "imagemMarcadaRejeitada.png" ,
                       imagemMarcadaRejeitada );
148         }
149
150
151         return 0;
152 }
```

```
1 cmake_minimum_required(VERSION 3.1)
2 project(deteccao_marcas)
3 find_package( OpenCV  REQUIRED )
4 find_package( aruco   REQUIRED )
5 add_executable(deteccao_marcas codigo.cpp)
6
```

```
target_link_libraries(deteccao_marcas ${OpenCV_LIBS} )
target_link_libraries(deteccao_marcas ${aruco_LIBS})
```

A.3 Segmentação e Corte

O terceiro programa executa a segmentação e corte da nuvem de pontos. Ele recebe como entrada as nuvens de pontos que serão trabalhadas e o arquivo com as marcações detectada que foi gerado pelo segundo programa.

A saída desse programa é uma nuvem de pontos já cortada e segmentada que contém apenas o objeto de interesse que está sendo digitalizado.

```
#include <pcl/io/pcd_io.h>
#include <pcl/io/ply_io.h>
#include <pcl/console/print.h>
#include <pcl/console/parse.h>
#include <pcl/console/time.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <stdlib.h>
#include <string>
#include <math.h>
#include "MatrizInversa.cpp"

using namespace pcl;
using namespace pcl::io;
using namespace pcl::console;
using namespace std;

struct ponto
{
        double x;
        double y;
        double z;
};


int main (int argc, char** argv)
{
cout << "!!!!!!!!!!!!!!!!!!!!!!!!!!1\n\n";
```

```
30
31
32
33  ///leitura do arquivo de nuvem e armazenamento dos valores dos
        pontos em uma estrutura de dados (MatrizNuvem)
34
35          ponto matrizNuvem[424][512];
36          string frase;
37          float x, y, z;
38
39          ifstream arquivoNuvem;
40          arquivoNuvem.open("arquivoNuvem.txt",ios::in);
41
42          for( int i = 0 ; i < 424 ; i++ ) //i representa as linhas
43          {
44                  for( int j = 0 ; j < 512 ; j++ ) //j representa
                        as colunas
45                  {
46                          getline(arquivoNuvem,frase);
47                          std::stringstream ss(frase);
48                          ss >> x >> y >> z; //pega os valores
                                numericos da linha para as variaveis
49
50                          matrizNuvem[i][j].x = x;
51                          matrizNuvem[i][j].y = y;
52                          matrizNuvem[i][j].z = z;
53                  }
54          }
55
56          arquivoNuvem.close();
57
58          //neste ponto, a atribuicao da matriz de pontos da nuvem
                esta completa
59
60  cout << "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!2\n\n";
61
62          //geracao de uma nuvem de pontos da matrizNuvem para
                conferencia
63
64          //nuvem normal
65
```

```
66          pcl::PointCloud<pcl::PointXYZ>::Ptr cloudNormal(new pcl::
               PointCloud<pcl::PointXYZ>);
67
68          pcl::PointXYZ pontoNovoNormal;
69
70          for( int i = 0 ; i < 424 ; i++ )
71          {
72                  for( int j = 0 ; j < 512 ; j++ )
73                  {
74                          pontoNovoNormal.x = matrizNuvem[i][j].x;
75                          pontoNovoNormal.y = matrizNuvem[i][j].y;
76                          pontoNovoNormal.z = matrizNuvem[i][j].z;
77
78
79                          cloudNormal->width = 1;
80                          cloudNormal->height = cloudNormal->points
                               .size() + 1;
81                          cloudNormal->points.push_back(
                               pontoNovoNormal);
82                  }
83          }
84
85          pcl::io::savePCDFileASCII ("nuvem_intermediaria_normal.
               pcd", *cloudNormal);
86          std::cerr << "\nSaved " << cloudNormal->points.size () <<
                " data points to nuvem_intermediaria_normal.pcd." <<
               std::endl;
87
88
89
90  ///leitura do arquivo de pontos (marcacoes) detectados e
       armazenamento dos valores em uma matriz 3x3 (27|43|42)
91
92          //o arquivo dos pontos das marcacoes fornece as
               informacoes em termos de pixel da imagem detectada
93
94          ifstream arquivoPontos;
95          arquivoPontos.open("arquivoPontos.txt",ios::in);
96
97          double X[3][3];
98          double pixelLinhaFloat, pixelColunaFloat, fractPart,
```

```
            pixelLinha, pixelColuna;
99          int id;
100
101         for( int i = 0 ; i < 3 ; i++)
102         {
103                 getline(arquivoPontos,frase);
104                 std::stringstream ss(frase);
105                 ss >> pixelLinhaFloat >> pixelColunaFloat;
106
107                 fractPart = modf(pixelLinhaFloat , &pixelLinha);
                        //cast para inteiro para remover a parte
                       fracionada
108                 fractPart = modf(pixelColunaFloat , &pixelColuna)
                       ;
109
110                 int pixelLinhaFinal  = (int)pixelLinha;
111                 int pixelColunaFinal = (int)pixelColuna;
112
113 //              if(id==27)
114 //              {
115                         X[0][i] = matrizNuvem[pixelLinhaFinal][
                               pixelColunaFinal].x;
116                         X[1][i] = matrizNuvem[pixelLinhaFinal][
                               pixelColunaFinal].y;
117                         X[2][i] = matrizNuvem[pixelLinhaFinal][
                               pixelColunaFinal].z;
118 //              }
119 /*
120                 if(id==43)
121                 {
122                         X[0][1] = matrizNuvem[pixelLinhaFinal][
                               pixelColunaFinal].x;
123                         X[1][1] = matrizNuvem[pixelLinhaFinal][
                               pixelColunaFinal].y;
124                         X[2][1] = matrizNuvem[pixelLinhaFinal][
                               pixelColunaFinal].z;
125                 }
126
127                 if(id==42)
128                 {
129                         X[0][2] = matrizNuvem[pixelLinhaFinal][
```

```
                                pixelColunaFinal].x;
130                         X[1][2] = matrizNuvem[pixelLinhaFinal][
                                pixelColunaFinal].y;
131                         X[2][2] = matrizNuvem[pixelLinhaFinal][
                                pixelColunaFinal].z;
132                 }*///faz o armazenamento de X na ordem 27|43|42
133         }
134
135         //imprime a matriz adquirida do arquivo para verificar se
                esta correta
136
137         cout << "\nMatriz de Pontos Detectados, (adquirida do
               arquivo):\n\n";
138
139         for( int i = 0 ; i < 3 ; i++ )
140         {
141                 for( int j = 0 ; j < 3 ; j++ )
142                 {
143                         cout << X[i][j] << " ";
144                 }
145                 cout << "\n";
146         }
147
148                 arquivoPontos.close();
149
150 cout << "\n!!!!!!!!!!!!!!!!!!!!!3\n";
151
152
153
154 //Calculo da Nuvem automatica
155
156         //nuvem manipulada automatica
157
158         pcl::PointCloud<pcl::PointXYZ>::Ptr cloudAutomatica(new
               pcl::PointCloud<pcl::PointXYZ>);
159
160         pcl::PointXYZ pontoNovoAutomatico;
161
162         double maisAlto, maisBaixo, maisADireita, maisAEsquerda;
163         int idAlto, idDireita, idEsquerda, idBaixo;
164
```

```
165         maisAlto      = -50;
166         maisBaixo     =  50;
167         maisADireita  = -50;
168         maisAEsquerda =  50;
169
170         //marcacao mais alta
171         if( X[1][0] > maisAlto) //X[1]    o y
172         {
173                 maisAlto = X[1][0];
174                 idAlto = 27;
175         }
176         if( X[1][1] > maisAlto)
177         {
178                 maisAlto = X[1][1];
179                 idAlto = 43;
180         }
181         if( X[1][2] > maisAlto)
182         {
183                 maisAlto = X[1][2];
184                 idAlto = 42;
185         }
186
187         cout << "\nA marcacao mais alta e a de id=" << idAlto <<
               "\n\n";
188
189         //marcacao mais baixa
190         if( X[1][0] < maisBaixo) //X[1]    o y
191         {
192                 maisBaixo = X[1][0];
193                 idBaixo = 27;
194         }
195         if( X[1][1] < maisBaixo)
196         {
197                 maisBaixo = X[1][1];
198                 idBaixo = 43;
199         }
200         if( X[1][2] < maisBaixo)
201         {
202                 maisBaixo = X[1][2];
203                 idBaixo = 42;
204         }
```

```
205
206         cout << "\nA marcacao mais baixa    a de id=" << idBaixo
                << "\n\n";
207
208         //marcacao mais a direita
209         if( X[0][0] > maisADireita) //X[0]    o x
210         {
211                 maisADireita = X[0][0];
212                 idDireita = 27;
213         }
214         if( X[0][1] > maisADireita)
215         {
216                 maisADireita = X[0][1];
217                 idDireita = 43;
218         }
219         if( X[0][2] > maisADireita)
220         {
221                 maisADireita = X[0][2];
222                 idDireita = 42;
223         }
224
225         cout << "\nA marcacao mais a direita e a de id=" <<
                idDireita << "\n\n";
226
227         //marca  o mais a esquerda
228         if( X[0][0] < maisAEsquerda) //X[0]    o x
229         {
230                 maisAEsquerda = X[0][0];
231                 idEsquerda = 27;
232         }
233         if( X[0][1] < maisAEsquerda)
234         {
235                 maisAEsquerda = X[0][1];
236                 idEsquerda = 43;
237         }
238         if( X[0][2] > maisAEsquerda)
239         {
240                 maisAEsquerda = X[0][2];
241                 idEsquerda = 42;
242         }
243
```

```
244         cout << "\nA marcacao mais a esquerda    a de id=" <<
                idEsquerda << "\n\n";
245
246         //criacao do plano com os pontos das marcacoes detectadas
247
248         double vetor1[3]; //27 para 43
249         vetor1[0] = X[0][1] - X[0][0];
250         vetor1[1] = X[1][1] - X[1][0];
251         vetor1[2] = X[2][1] - X[2][0];
252
253         double vetor2[3]; //27 para 42
254         vetor2[0] = X[0][2] - X[0][0];
255         vetor2[1] = X[1][2] - X[1][0];
256         vetor2[2] = X[2][2] - X[2][0];
257
258         double a, b, c;
259
260         a = vetor1[1]*vetor2[2] - vetor1[2]*vetor2[1]; //produto
                vetorial entre os dois vetores para encontrar o vetor
                normal do plano
261         b = vetor1[2]*vetor2[0] - vetor1[0]*vetor2[2];
262         c = vetor1[0]*vetor2[1] - vetor1[1]*vetor2[0];
263
264         double zDoPlano;
265         double x0, y0, z0;
266
267         x0 = X[0][0];
268         y0 = X[1][0];
269         z0 = X[2][0];
270
271         for( int i = 0 ; i < 424 ; i++ ) //percorre toda
                matrizNuvem
272         {
273                 for( int j = 0 ; j < 512 ; j++ )
274                 {
275                         pontoNovoAutomatico.x = matrizNuvem[i][j
                                ].x;
276                         pontoNovoAutomatico.y = matrizNuvem[i][j
                                ].y;
277                         pontoNovoAutomatico.z = matrizNuvem[i][j
                                ].z;
```

```
278
279                         zDoPlano = ( -a*(matrizNuvem[i][j].x - x0
                               ) - b*(matrizNuvem[i][j].y - y0) + c*z0
                                )/c;
280
281                         if( pontoNovoAutomatico.y <= maisAlto &&
                               pontoNovoAutomatico.y >= maisBaixo &&
282                             pontoNovoAutomatico.x <= maisADireita
                                 && pontoNovoAutomatico.x >= (
                                maisAEsquerda - 0.2) &&
283                             pontoNovoAutomatico.z <= 1.2 &&
284                             pontoNovoAutomatico.z < (zDoPlano -
                                0.01))
285 //)
286                         {
287                             cloudAutomatica->width = 1;
288                             cloudAutomatica->height =
                                cloudAutomatica->points.size()
                                + 1;
289                             cloudAutomatica->points.push_back
                                (pontoNovoAutomatico);
290                         }
291                 }
292         }
293
294         pcl::io::savePCDFileASCII ("
            nuvem_intermediaria_automatica.pcd", *cloudAutomatica);
295         std::cerr << "\nSaved " << cloudAutomatica->points.size
            () << " data points to nuvem_intermediaria_automatica.
            pcd" << std::endl;
296
297
298
299 //Definicao da matriz B
300 /*
301         double B[3][3]; // |27|43|42|
302
303         B[0][0] = 0 + 1; //o x    invertido devido a imagem ser
            espelhada horizontalmente
304         B[1][0] = 0 + 1;
305         B[2][0] = 0 + 1;
```

```
306         B[0][1] = -0.367 + 1;
307         B[1][1] = 0 + 1;
308         B[2][1] = -0.054332 + 1;
309         B[0][2] = 0 + 1;
310         B[1][2] = 0 + 1;
311         B[2][2] = -0.408 + 1;
312
313         // Exibe a matriz B
314         printf ("\n\nMatriz B =\n");
315         for (int i = 0; i < 3; i++)
316         {
317                 for(int j = 0 ; j < 3 ; j++)
318                 {
319                         cout << B[i][j] << " ";
320                 }
321                 cout << "\n";
322         }
323         std::cout << "\n\n";
324
325 cout << "\n\n!!!!!!!!!!!!!!!!!!!!!!!!!5\n\n";
326
327
328
329 //Calculo da matriz inversa de B (nova formulacao por mudanca de
       base)
330
331         double inversaB[9];
332         double Bfinal[9];
333         int ordem = 3;
334
335         //passa X para a forma linear exigida pela funcao '
              MatrizInversa'
336
337         for( int i = 0 ; i < 3 ; i ++ )
338         {
339                 for( int j = 0 ; j < 3 ; j++ )
340                 {
341                         //Bfinal[(i * 3) + j] = X[i][j][0];
342                         Bfinal[(i*3) + j] = B[i][j];
343                 }
344         }
```

```
345
346         //imprime a matriz Bfinal para comparacao com a matriz B
               e verificacao
347
348         cout << "\n\nMatriz Bfinal:\n\n";
349         for( int i = 0 ; i < 9 ; i++ )
350         {
351                 cout << Bfinal[i] << " ";
352
353                 if( (i + 1) % 3 == 0 ) cout << "\n";
354         }
355         cout << "\n\n";
356
357         //calculo da matriz inversa
358
359         MatrizInversa(inversaB, Bfinal, ordem); //para chamar
               essa funcao, a matriz X precisa estar armazenada de
               forma linear (por linhas)
360
361         // Exibe a matriz inversa de B
362         printf ("\n\nInversa de B =");
363         for (int i = 0; i < 9; i++)
364         {
365                 if( i % 3 == 0) printf("\n");
366                 std::cout << inversaB[i] << "  ";
367         }
368         std::cout << "\n\n";
369
370         //armazenando a inversa de X em forma matricial novamente
371
372         double inversaFinalB[3][3];
373
374         inversaFinalB[0][0] = inversaB[0];
375         inversaFinalB[0][1] = inversaB[1];
376         inversaFinalB[0][2] = inversaB[2];
377         inversaFinalB[1][0] = inversaB[3];
378         inversaFinalB[1][1] = inversaB[4];
379         inversaFinalB[1][2] = inversaB[5];
380         inversaFinalB[2][0] = inversaB[6];
381         inversaFinalB[2][1] = inversaB[7];
382         inversaFinalB[2][2] = inversaB[8];
```

```
383
384         cout << "\n\nInversa Final de B: \n\n";
385         for( int i = 0 ; i < 3 ; i++ )
386         {
387                 for( int j = 0 ; j < 3 ; j++ )
388                 {
389                         cout << inversaFinalB[i][j] << " ";
390                 }
391                 cout << "\n";
392         }
393
394 cout << "\n\n!!!!!!!!!!!!!!!!!!!!!!!!!!4\n\n";
395
396
397
398 //Calculo da matriz M, que e a multiplicacao da inversa de B por
       X (na formulacao X    A)
399
400         double M[3][3];
401         double auxiliar;
402
403         int linhasM, colunasM, linhasX, colunasX, i, j, k;
404
405         linhasX = 3;
406         colunasX = 3;
407         int linhasInversaB = 3;
408         int colunasInversaB = 3;
409
410
411 //multiplicacao das duas matrizes
412
413     for ( i = 0 ; i < linhasInversaB ; i++ )
414     {
415         for( j = 0 ; j < colunasX ; j++ )
416         {
417             auxiliar = 0;
418             for( k = 0 ; k < colunasInversaB ; k++ )
419             {
420                 auxiliar = auxiliar + inversaFinalB[i][k]*X[k][j
                       ];
421             }
```

```
422             M[i][j] = auxiliar;
423         }
424     }
425     printf("\n");
426
427 //impressao da matriz M
428
429         cout << "\nMatriz M que fara a transformacao dos pontos
                de camera (X) para o espa o objeto (B):\n\n";
430         for( int i = 0 ; i < 3 ; i++ )
431         {
432                 for( int j = 0 ; j < 3 ; j++ )
433                 {
434                         cout << M[i][j] << " ";
435                 }
436                 cout << "\n";
437         }
438
439 cout << "\n\n!!!!!!!!!!!!!!!!!!!!!!!!6\n\n";
440
441
442
443 //Calculo e criacao da nuvem de pontos do objeto extraido
444
445         pcl::PointCloud<pcl::PointXYZ>::Ptr cloudTransformada(new
                pcl::PointCloud<pcl::PointXYZ>);
446         pcl::PointXYZ pontoNovoTransformado;
447
448         //pcl::PointCloud<pcl::PointXYZ> cloud; //cria a nuvem
449         pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::
                PointCloud<pcl::PointXYZ>);
450
451         pcl::PointXYZ pontoNovo;
452
453         double pontoTransformado[3];
454         double pontoNuvem[3];
455         auxiliar = 0;
456
457         for( int i = 0 ; i < 424 ; i++ ) //percorre toda a
                matrizNuvem
458         {
```

```
459                 for( int j = 0 ; j < 512 ; j++ )
460                 {
461                         //transforma um ponto da matrizNuvem em
                               um ponto do espaco
462
463                         auxiliar = 0;
464
465                         pontoNuvem[0] = matrizNuvem[i][j][0].x;
466                         pontoNuvem[1] = matrizNuvem[i][j][0].y;
467                         pontoNuvem[2] = matrizNuvem[i][j][0].z;
468
469                         for( int w = 0 ; w < 3 ; w++ ) //percorre
                                uma 3 por 3
470                         {
471                                 for( int k = 0 ; k < 3 ; k++ )
472                                 {
473                                         auxiliar = auxiliar + (M[
                                            k][w] * pontoNuvem[k]);
474                                 }
475                                 pontoTransformado[w] = auxiliar;
476                         }
477
478
479                         //armazena a nuvem transformada sem
                               filtragem
480
481                         pontoNovoTransformado.x =
                               pontoTransformado[0];
482                         pontoNovoTransformado.y =
                               pontoTransformado[1];
483                         pontoNovoTransformado.z =
                               pontoTransformado[2];
484
485                         cloudTransformada->width = 1;
486                         cloudTransformada->height =
                               cloudTransformada->points.size() + 1;
487                         cloudTransformada->points.push_back(
                               pontoNovoTransformado);
488
489
490                         //faz a checagem do ponto e inclusao na
```

```
                        nuvem final propriamente dita (precisa
                        verificar o filtro)
491
492                     if( pontoTransformado[1] >= 0 &&
                        pontoTransformado[1] <= 1   && //y
493                      pontoTransformado[0] <= 0 &&
                            pontoTransformado[0] >= -1 && //x,
                            z abaixo
494                      pontoTransformado[2] <= 0 &&
                            pontoTransformado[2] >= -1) //est
                             deslocado em 1 unidade devido a
                            origem ser o (1,1,1)
495                     {
496                          pontoNovo.x = pontoTransformado
                                [0];
497                          pontoNovo.y = pontoTransformado
                                [1];
498                          pontoNovo.z = pontoTransformado
                                [2];
499
500                          cloud->width = 1;
501                          cloud->height = cloud->points.
                                size() + 1;
502                          cloud->points.push_back(pontoNovo
                                );
503
504                          //cout << "ponto adicionado!!!!\n
                                \n";
505                     }
506             }
507     }
508
509     //pcl::io::savePCDFileASCII ("nuvem_final.pcd", *cloud);
510     //std::cerr << "\nSaved " << cloud->points.size () << "
           data points to nuvem_final.pcd." << std::endl;
511
512     pcl::io::savePCDFileASCII ("nuvem_transformada.pcd", *
           cloudTransformada);
513     std::cerr << "\nSaved " << cloudTransformada->points.size
            () << " data points to nuvem_transformada.pcd." << std
           ::endl;
```

```
514 */
515 cout << "\n\nConcluido!!\n\n";
516
517
518         return 0;
519 }
```

```
1  /*
2   * Matriz_Inversa.cpp
3   *
4   * Copyright 2014 Rafael Andrade <rafaelsandrade@gmail.com>
5   *
6   * This program is free software; you can redistribute it and/or
       modify
7   * it under the terms of the GNU General Public License as
       published by
8   * the Free Software Foundation; either version 2 of the License,
        or
9   * (at your option) any later version.
10  *
11  * This program is distributed in the hope that it will be useful
       ,
12  * but WITHOUT ANY WARRANTY; without even the implied warranty of
13  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14  * GNU General Public License for more details.
15  *
16  * You should have received a copy of the GNU General Public
       License
17  * along with this program; if not, write to the Free Software
18  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
19  * MA 02110-1301, USA.
20  *
21  * 16/10/2011
22  * Calcula a matriz inversa
23  */
24
25
26 #include <cstdio>
27 #include <cstdlib>
28
29 ////    Calcula a Inversa da matriz
30 void MatrizInversa( double *r, const double *m, int ordem)
```

```
31 {
32     double *inv = new double[ordem * ordem];
33     double *temp = new double[ordem * ordem];
34 
35     if (!inv || !temp)
36     {
37         printf("\n\nERRO: falha na alocacao de memoria\n\n");
38         exit(1);
39     }
40 
41     // Copia para 'temp' o conteudo de 'm'
42     for( int i = 0; i < ordem * ordem; i++ )
43     {
44         temp[ i ] = m[ i ];
45     }
46 
47     // Transforma 'inv' na matriz identidade
48     int j = 0;
49 
50     for( int i = 0; i < ordem * ordem; i++ )
51     {
52         if( i == (j * (ordem + 1)) )
53         {
54             inv[i] = 1.0;
55             j++;
56         }
57         else inv[i] = 0.0;
58     }
59 
60     //////  Escalona a parte inferior
61     j = 0;
62     double pivo;
63 
64     for ( int i = 0; i < ordem; i++ )
65     {
66         if ( temp[ i * ( ordem + 1 ) ] == 0.0 ) /// Verifica se o
               pivo    nulo
67         {
68             for ( j = i + 1; j < ordem; j++ )   /// Procura o
                   valor nao nulo abaixo do pivo
69             {
```

```
70                  if ( temp[ i + j * ordem ] != 0.0 )
71                  {
72                      for ( int k = 0; k < ordem; k++ )
73                      {
74                          /// Soma a linha do pivo nulo com a linha
                                abaixo
75                          temp[ k + i * ordem ] += temp[ k + j *
                               ordem ];
76                          inv[ k + i * ordem ] += inv[ k + j *
                               ordem ];
77                      }
78                      break;
79                  }
80              }
81
82              if ( j == ordem )   /// Se nao achar retorna matriz
                    nula
83              {
84                  delete[] temp;
85                  delete[] inv;
86                  printf("\n\nERRO: Matriz nao possui inversa\n\n")
                        ;
87                  return;
88              }
89          }
90
91          for ( j = i + 1; j < ordem; j++ )   /// Zera os elementos
                abaixo do pivo
92          {
93              if ( temp[ i + j * ordem ] != 0.0 ) /// Ignora se
                    elemento e nulo
94              {
95                  pivo = temp[ i + j * ordem ] / temp[ i * ( ordem
                        + 1 ) ];
96                  for( int k = 0; k < ordem; k++ )
97                  {
98                      temp[ k + j * ordem ] -= temp[ k + i * ordem
                            ] * pivo;
99                      inv[ k + j * ordem ] -= inv[ k + i * ordem ]
                            * pivo;
100                 }
```

```
101             }
102         }
103     }
104
105     //////// Escalona a parte superior
106     for( int i = ordem - 1; i >= 0; i-- )
107     {
108         for( j = i - 1; j >= 0; j-- )   /// Zera os elementos
               abaixo do pivo
109         {
110             if( temp[ i + j * ordem ] != 0.0 )  /// Ignora se
                   elemento e nulo
111             {
112                 pivo = temp[ i + j * ordem ] / temp[ i * ( ordem
                       + 1 ) ];
113                 for( int k = ordem - 1; k >= 0; k-- )
114                 {
115                     temp[ k + j * ordem ] -= temp[ k + i * ordem
                           ] * pivo;
116                     inv[ k + j * ordem ] -= inv[ k + i * ordem ]
                           * pivo;
117                 }
118             }
119         }
120     }
121
122     //////// Transformando os elementos da coluna principal de '
           temp' em '1'
123     for( int i = 0; i < ordem; i++ )
124     {
125         pivo = temp[ i * ( ordem + 1 ) ];
126         for( j = 0; j < ordem; j++ )
127         {
128             temp[ j + i * ordem ] /= pivo;
129             inv[ j + i * ordem ] /= pivo;
130         }
131     }
132
133     // Copia os elementos de 'inv' para 'r'
134     for ( int i = 0; i < ordem * ordem; i++ )
135     {
```

```
136         r[i] = inv[i];
137     }
138 
139     delete[] inv;
140     delete[] temp;
141 }
142 
143 /*
144 int main(int argc, char **argv)
145 {
146     double M[9], A[9] = {    -1, 5, -7, // Matriz de ordem 3 a
        ser invertida
147                               3, -2, 3,
148                              -2, 3, 7      };
149 
150     // Exibe a matriz A
151     printf("A = ");
152     for (int i = 0; i < 9; i++)
153     {
154         if( i % 3 == 0) printf("\n");
155         printf("%13G", A[i]);
156 
157     }
158 
159     MatrizInversa(M, A, 3);
160 
161     // Exibe a matriz inversa de A
162     printf ("\n\n\n  -1\n(A) =");
163     for (int i = 0; i < 9; i++)
164     {
165         if( i % 3 == 0) printf("\n");
166         printf("%13G", M[i]);
167 
168     }
169 
170 
171     return 0;
172 }
173 */
```

```
1 cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
2 
```

```
project(corte)

find_package(PCL 1.2 REQUIRED)

include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})

add_executable (corte Corte.cpp)
target_link_libraries (corte ${PCL_LIBRARIES})
```

A.4 Conversões

Programas que executam uma série de transformações necessárias para converter do formato .pcd, da biblioteca PCL, para o formato .ply, do meshlab e também efetuar a conversão na direção contrária.

```
/*pcd2ply.cpp
 * Software License Agreement (BSD License)
 *
 *  Point Cloud Library (PCL) - www.pointclouds.org
 *  Copyright (c) 2011-2012, Willow Garage, Inc.
 *
 *  All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or
    without
 *  modification, are permitted provided that the following
    conditions
 *  are met:
 *
 *   * Redistributions of source code must retain the above
    copyright
 *     notice, this list of conditions and the following
    disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the
    following
 *     disclaimer in the documentation and/or other materials
    provided
```

```
18  *      with the distribution.
19  *    * Neither the name of the copyright holder(s) nor the names
    of its
20  *      contributors may be used to endorse or promote products
    derived
21  *      from this software without specific prior written
    permission.
22  *
23  *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
    CONTRIBUTORS
24  *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT
     NOT
25  *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
    FITNESS
26  *  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
    THE
27  *  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
    INDIRECT,
28  *  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (
    INCLUDING,
29  *  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
    SERVICES;
30  *  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    HOWEVER
31  *  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
    STRICT
32  *  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
    ARISING IN
33  *  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
    THE
34  *  POSSIBILITY OF SUCH DAMAGE.
35  *
36  * $Id$
37  *
38  */
39
40 #include <pcl/io/pcd_io.h>
41 #include <pcl/io/ply_io.h>
42 #include <pcl/console/print.h>
43 #include <pcl/console/parse.h>
44 #include <pcl/console/time.h>
```

```
45
46 using namespace pcl;
47 using namespace pcl::io;
48 using namespace pcl::console;
49
50 void
51 printHelp (int, char **argv)
52 {
53   print_error ("Syntax is: %s [-format 0|1] [-use_camera 0|1]
       input.pcd output.ply\n", argv[0]);
54 }
55
56 bool
57 loadCloud (const std::string &filename, pcl::PCLPointCloud2 &
     cloud)
58 {
59   TicToc tt;
60   print_highlight ("Loading "); print_value ("%s ", filename.
       c_str ());
61
62   tt.tic ();
63   if (loadPCDFile (filename, cloud) < 0)
64     return (false);
65   print_info ("[done, "); print_value ("%g", tt.toc ());
       print_info (" ms : "); print_value ("%d", cloud.width * cloud
       .height); print_info (" points]\n");
66   print_info ("Available dimensions: "); print_value ("%s\n", pcl
       ::getFieldsList (cloud).c_str ());
67
68   return (true);
69 }
70
71 void
72 saveCloud (const std::string &filename, const pcl::PCLPointCloud2
     &cloud, bool binary, bool use_camera)
73 {
74   TicToc tt;
75   tt.tic ();
76
77   print_highlight ("Saving "); print_value ("%s ", filename.c_str
       ());
```

```
78
79    pcl::PLYWriter writer;
80    writer.write (filename, cloud, Eigen::Vector4f::Zero (), Eigen
          ::Quaternionf::Identity (), binary, use_camera);
81
82    print_info ("[done, "); print_value ("%g", tt.toc ());
          print_info (" ms : "); print_value ("%d", cloud.width * cloud
          .height); print_info (" points]\n");
83  }
84
85  /* ---[ */
86  int
87  main (int argc, char** argv)
88  {
89    print_info ("Convert a PCD file to PLY format. For more
          information, use: %s -h\n", argv[0]);
90
91    if (argc < 3)
92    {
93      printHelp (argc, argv);
94      return (-1);
95    }
96
97    // Parse the command line arguments for .pcd and .ply files
98    std::vector<int> pcd_file_indices =
          parse_file_extension_argument (argc, argv, ".pcd");
99    std::vector<int> ply_file_indices =
          parse_file_extension_argument (argc, argv, ".ply");
100   if (pcd_file_indices.size () != 1 || ply_file_indices.size ()
          != 1)
101   {
102     print_error ("Need one input PCD file and one output PLY file
          .\n");
103     return (-1);
104   }
105
106   // Command line parsing
107   bool format = true;
108   bool use_camera = true;
109   parse_argument (argc, argv, "-format", format);
110   parse_argument (argc, argv, "-use_camera", use_camera);
```

```
111   print_info ("PLY output format: "); print_value ("%s, ", (
          format ? "binary" : "ascii"));
112   print_value ("%s\n", (use_camera ? "using camera" : "no camera"
          ));
113
114   // Load the first file
115   pcl::PCLPointCloud2 cloud;
116   if (!loadCloud (argv[pcd_file_indices[0]], cloud))
117     return (-1);
118
119   // Convert to PLY and save
120   saveCloud (argv[ply_file_indices[0]], cloud, format, use_camera
          );
121
122   return (0);
123 }
```

```
1  #pcd2ply
2  cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
3
4  project(pcd2ply)
5
6  find_package(PCL 1.2 REQUIRED)
7
8  include_directories(${PCL_INCLUDE_DIRS})
9  link_directories(${PCL_LIBRARY_DIRS})
10 add_definitions(${PCL_DEFINITIONS})
11
12 add_executable (pcd2ply pcd2ply.cpp)
13 target_link_libraries (pcd2ply ${PCL_LIBRARIES})
```

```
1  /*ply2pcd.cpp
2   * Software License Agreement (BSD License)
3   *
4   *  Point Cloud Library (PCL) - www.pointclouds.org
5   *  Copyright (c) 2011-2012, Willow Garage, Inc.
6   *
7   *  All rights reserved.
8   *
9   *  Redistribution and use in source and binary forms, with or
        without
10  *  modification, are permitted provided that the following
```

```
    conditions
11  *  are met:
12  *
13  *   * Redistributions of source code must retain the above
    copyright
14  *     notice, this list of conditions and the following
    disclaimer.
15  *   * Redistributions in binary form must reproduce the above
16  *     copyright notice, this list of conditions and the
    following
17  *     disclaimer in the documentation and/or other materials
    provided
18  *     with the distribution.
19  *   * Neither the name of the copyright holder(s) nor the names
    of its
20  *     contributors may be used to endorse or promote products
    derived
21  *     from this software without specific prior written
    permission.
22  *
23  *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
    CONTRIBUTORS
24  *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT
     NOT
25  *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
    FITNESS
26  *  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
    THE
27  *  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
    INDIRECT,
28  *  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (
    INCLUDING,
29  *  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
    SERVICES;
30  *  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    HOWEVER
31  *  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
    STRICT
32  *  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
    ARISING IN
33  *  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
```

```
    THE
34  *  POSSIBILITY OF SUCH DAMAGE.
35  *
36  * $Id$
37  *
38  */
39
40 #include <pcl/io/pcd_io.h>
41 #include <pcl/io/ply_io.h>
42 #include <pcl/console/print.h>
43 #include <pcl/console/parse.h>
44 #include <pcl/console/time.h>
45
46 using namespace pcl;
47 using namespace pcl::io;
48 using namespace pcl::console;
49
50 void
51 printHelp (int, char **argv)
52 {
53   print_error ("Syntax is: %s [-format 0|1] input.ply output.pcd\
       n", argv[0]);
54 }
55
56 bool
57 loadCloud (const std::string &filename, pcl::PCLPointCloud2 &
     cloud)
58 {
59   TicToc tt;
60   print_highlight ("Loading "); print_value ("%s ", filename.
       c_str ());
61
62   pcl::PLYReader reader;
63   tt.tic ();
64   if (reader.read (filename, cloud) < 0)
65     return (false);
66   print_info ("[done, "); print_value ("%g", tt.toc ());
       print_info (" ms : "); print_value ("%d", cloud.width * cloud
       .height); print_info (" points]\n");
67   print_info ("Available dimensions: "); print_value ("%s\n", pcl
       ::getFieldsList (cloud).c_str ());
```

```
68
69    return (true);
70  }
71
72  void
73  saveCloud (const std::string &filename, const pcl::PCLPointCloud2
        &cloud, bool format)
74  {
75    TicToc tt;
76    tt.tic ();
77
78    print_highlight ("Saving "); print_value ("%s ", filename.c_str
          ());
79
80    pcl::PCDWriter writer;
81    writer.write (filename, cloud, Eigen::Vector4f::Zero (), Eigen
        ::Quaternionf::Identity (), format);
82
83    print_info ("[done, "); print_value ("%g", tt.toc ());
        print_info (" ms : "); print_value ("%d", cloud.width * cloud
        .height); print_info (" points]\n");
84  }
85
86  /* ---[ */
87  int
88  main (int argc, char** argv)
89  {
90    print_info ("Convert a PLY file to PCD format. For more
        information, use: %s -h\n", argv[0]);
91
92    if (argc < 3)
93    {
94      printHelp (argc, argv);
95      return (-1);
96    }
97
98    // Parse the command line arguments for .pcd and .ply files
99    std::vector<int> pcd_file_indices =
        parse_file_extension_argument (argc, argv, ".pcd");
100   std::vector<int> ply_file_indices =
        parse_file_extension_argument (argc, argv, ".ply");
```

```
101   if (pcd_file_indices.size () != 1 || ply_file_indices.size ()
          != 1)
102   {
103     print_error ("Need one input PLY file and one output PCD file
          .\n");
104     return (-1);
105   }
106
107   // Command line parsing
108   bool format = 1;
109   parse_argument (argc, argv, "-format", format);
110   print_info ("PCD output format: "); print_value ("%s\n", (
          format ? "binary" : "ascii"));
111
112   // Load the first file
113   pcl::PCLPointCloud2 cloud;
114   if (!loadCloud (argv[ply_file_indices[0]], cloud))
115     return (-1);
116
117   // Convert to PLY and save
118   saveCloud (argv[pcd_file_indices[0]], cloud, format);
119
120   return (0);
121 }
```

```
1  #ply2pcd
2  cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
3
4  project(ply2pcd)
5
6  find_package(PCL 1.2 REQUIRED)
7
8  include_directories(${PCL_INCLUDE_DIRS})
9  link_directories(${PCL_LIBRARY_DIRS})
10 add_definitions(${PCL_DEFINITIONS})
11
12 add_executable (ply2pcd ply2pcd.cpp)
13 target_link_libraries (ply2pcd ${PCL_LIBRARIES})
```

A.5 Hausdorff Modificado

Programa que executa todos os cálculos da distância de Hausdorff, incluindo todas as modificações feitas para esse trabalho, como a distância de Hausdorff mínima, média dos erros e desvio padrão.

```
/*
 * Software License Agreement (BSD License)
 *
 *  Point Cloud Library (PCL) - www.pointclouds.org
 *  Copyright (c) 2009-2012, Willow Garage, Inc.
 *  Copyright (c) 2012-, Open Perception, Inc.
 *  Copyright (c) 2014, RadiantBlue Technologies, Inc.
 *
 *  All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or
    without
 *  modification, are permitted provided that the following
    conditions
 *  are met:
 *
 *   * Redistributions of source code must retain the above
    copyright
 *     notice, this list of conditions and the following
    disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the
    following
 *     disclaimer in the documentation and/or other materials
    provided
 *     with the distribution.
 *   * Neither the name of the copyright holder(s) nor the names
    of its
 *     contributors may be used to endorse or promote products
    derived
 *     from this software without specific prior written
    permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
    CONTRIBUTORS
```

```
26  *   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT
        NOT
27  *   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
        FITNESS
28  *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
        THE
29  *   COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
        INDIRECT,
30  *   INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (
        INCLUDING,
31  *   BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
        SERVICES;
32  *   LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
        HOWEVER
33  *   CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
        STRICT
34  *   LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
        ARISING IN
35  *   ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
        THE
36  *   POSSIBILITY OF SUCH DAMAGE.
37  *
38  * $Id$
39  */
40
41 #include <pcl/point_types.h>
42 #include <pcl/io/pcd_io.h>
43 #include <pcl/console/print.h>
44 #include <pcl/console/parse.h>
45 #include <pcl/console/time.h>
46 #include <pcl/search/kdtree.h>
47
48 using namespace std;
49 using namespace pcl;
50 using namespace pcl::io;
51 using namespace pcl::console;
52 using namespace pcl::search;
53
54 typedef PointXYZ PointType;
55 typedef PointCloud<PointXYZ> Cloud;
56
```

```
57 void
58 printHelp (int, char **argv)
59 {
60   print_error ("Syntax is: %s cloud_a.pcd cloud_b.pcd\n", argv
       [0]);
61 }
62
63 bool
64 loadCloud (const std::string &filename, Cloud &cloud)
65 {
66   TicToc tt;
67   print_highlight ("Loading "); print_value ("%s ", filename.
       c_str ());
68
69   tt.tic ();
70   if (loadPCDFile (filename, cloud) < 0)
71     return (false);
72   print_info ("[done, "); print_value ("%g", tt.toc ());
       print_info (" ms : "); print_value ("%d", cloud.width * cloud
       .height); print_info (" points]\n");
73   print_info ("Available dimensions: "); print_value ("%s\n", pcl
       ::getFieldsList (cloud).c_str ());
74
75   return (true);
76 }
77
78 void compute (Cloud &cloud_a, Cloud &cloud_b)
79 {
80   // Estimate
81   TicToc tt;
82   tt.tic ();
83
84   print_highlight (stderr, "Computing ");
85
86   // compare A to B
87   pcl::search::KdTree<PointType> tree_b;
88   tree_b.setInputCloud (cloud_b.makeShared ());
89
90   float max_dist_a = -std::numeric_limits<float>::max ();
91   float min_dist_a =  std::numeric_limits<float>::max ();
92
```

```
93          int numeroPontos_a = cloud_a.points.size();
94          int numeroPontos_b = cloud_b.points.size();
95
96          float array_a[numeroPontos_a];
97          float array_b[numeroPontos_b];
98
99
100 for (size_t i = 0; i < cloud_a.points.size (); ++i)
101 {
102   std::vector<int> indices (1);
103   std::vector<float> sqr_distances (1);
104
105   tree_b.nearestKSearch (cloud_a.points[i], 1, indices,
        sqr_distances);
106
107   if (sqr_distances[0] > max_dist_a)
108     max_dist_a = sqr_distances[0];
109
110   if (sqr_distances[0] < min_dist_a)
111     min_dist_a = sqr_distances[0];
112
113
114         array_a[i] = std::sqrt(sqr_distances[0]); //preenche o
            array 'a' com todos as distancia da nuvem 'a'
115 }
116
117 // compare B to A
118 pcl::search::KdTree<PointType> tree_a;
119 tree_a.setInputCloud (cloud_a.makeShared ());
120
121 float max_dist_b = -std::numeric_limits<float>::max ();
122 float min_dist_b =  std::numeric_limits<float>::max ();
123
124
125 for (size_t i = 0; i < cloud_b.points.size (); ++i)
126 {
127   std::vector<int> indices (1);
128   std::vector<float> sqr_distances (1);
129
130   tree_a.nearestKSearch (cloud_b.points[i], 1, indices,
        sqr_distances);
```

```
131
132     if (sqr_distances[0] > max_dist_b)
133       max_dist_b = sqr_distances[0];
134
135     if (sqr_distances[0] < min_dist_b)
136       min_dist_b = sqr_distances[0];
137
138
139         array_b[i] = std::sqrt(sqr_distances[0]); //preenche o
              array 'b' com todos as distancia da nuvem 'b'
140   }
141
142   max_dist_a = std::sqrt (max_dist_a); //calcula a raiz quadrada
       das distancias calculadas
143   max_dist_b = std::sqrt (max_dist_b);
144   min_dist_a = std::sqrt (min_dist_a);
145   min_dist_b = std::sqrt (min_dist_b);
146
147   float dist    = std::max (max_dist_a, max_dist_b);
148   float distMin = std::min (min_dist_a, min_dist_b);
149
150   print_info ("[done, "); print_value ("%g", tt.toc ());
       print_info (" ms : ");
151   print_info ("A->B: "); print_value ("%f", max_dist_a);
152   print_info (", B->A: "); print_value ("%f", max_dist_b);
153   print_info (", Hausdorff Distance: "); print_value ("%f", dist)
       ;
154   print_info (" ]\n");
155
156   cout << "\n";
157
158   print_info ("[done min, "); print_value ("%g", tt.toc ());
       print_info (" ms : ");
159   print_info ("A->B: "); print_value ("%f", min_dist_a);
160   print_info (", B->A: "); print_value ("%f", min_dist_b);
161   print_info (", Minimum Hausdorff Distance: "); print_value ("%f
       ", distMin);
162   print_info (" ]\n");
163
164
```

```
165         //calculo do somatorio dos arrays de distancias e da
               media dos valores
166
167         float sum_a = 0;
168         float sum_b = 0;
169
170         for(size_t i = 0; i < cloud_a.points.size(); ++i)
171         {
172                 sum_a = sum_a + array_a[i];
173         }
174
175         for(size_t i = 0; i < cloud_b.points.size(); ++i)
176         {
177                 sum_b = sum_b + array_b[i];
178         }
179
180         float media_a = 0;
181         float media_b = 0;
182
183         media_a = sum_a/numeroPontos_a;
184         media_b = sum_b/numeroPontos_b;
185
186         cout << "\n\n";
187
188         print_info ("A media das distancias dos pontos de A para
               B e: "); print_value ("%f", media_a);
189
190         cout << "\n\n";
191         print_info ("A media das distancias dos pontos de B para
               A e: "); print_value ("%f", media_b);
192
193         cout << "\n\n";
194
195         //calculo da variancia e do desvio padrao
196
197         //para A
198
199         float sd_a  = 0;
200         float var_a = 0;
201
202         for( int n = 0 ; n < numeroPontos_a ; n++ )
```

```
203         {
204                 var_a += (array_a[n] - media_a) * (array_a[n] -
                        media_a);
205         }
206         var_a /= numeroPontos_a;
207         sd_a = std::sqrt(var_a);
208 
209         //para B
210 
211         float sd_b  = 0;
212         float var_b = 0;
213 
214         for( int n = 0 ; n < numeroPontos_b ; n++ )
215         {
216                 var_b += (array_b[n] - media_b) * (array_b[n] -
                        media_b);
217         }
218         var_b /= numeroPontos_b;
219         sd_b = std::sqrt(var_b);
220 
221         cout << "\n\n";
222 
223         print_info ("A variancia das distancias dos pontos de A
                para B e: "); print_value ("%f", var_a);
224 
225         cout << "\n\n";
226 
227         print_info ("A variancia das distancias dos pontos de B
                para A e: "); print_value ("%f", var_b);
228 
229         cout << "\n\n";
230 
231         print_info ("O DESVIO PADRAO das distancias dos pontos de
                A para B e: "); print_value ("%f", sd_a);
232 
233         cout << "\n\n";
234 
235         print_info ("O DESVIO PADRAO das distancias dos pontos de
                B para A e: "); print_value ("%f", sd_b);
236 
237         cout << "\n\n";
```

```
238
239 }
240
241 /* ---[ */
242 int
243 main (int argc, char** argv)
244 {
245   print_info ("Compute Hausdorff distance between point clouds.
        For more information, use: %s -h\n", argv[0]);
246
247   if (argc < 3)
248   {
249     printHelp (argc, argv);
250     return (-1);
251   }
252
253   // Parse the command line arguments for .pcd files
254   std::vector<int> p_file_indices;
255   p_file_indices = parse_file_extension_argument (argc, argv, ".
        pcd");
256   if (p_file_indices.size () != 2)
257   {
258     print_error ("Need two PCD files to compute Hausdorff
        distance.\n");
259     return (-1);
260   }
261
262   // Load the first file
263   Cloud::Ptr cloud_a (new Cloud);
264   if (!loadCloud (argv[p_file_indices[0]], *cloud_a))
265     return (-1);
266
267   // Load the second file
268   Cloud::Ptr cloud_b (new Cloud);
269   if (!loadCloud (argv[p_file_indices[1]], *cloud_b))
270     return (-1);
271
272   // Compute the Hausdorff distance
273   compute (*cloud_a, *cloud_b);
274 }
```

```
1 cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
```

```
2
3  project(hausdorff)
4
5  find_package(PCL 1.2 REQUIRED)
6
7  include_directories(${PCL_INCLUDE_DIRS})
8  link_directories(${PCL_LIBRARY_DIRS})
9  add_definitions(${PCL_DEFINITIONS})
10
11 add_executable (hausdorff hausdorff.cpp)
12 target_link_libraries (hausdorff ${PCL_LIBRARIES})
```

A.6 Gerador de Nuvem de Referência

Programa que gera uma nuvem de referência perfeita para os objetos cubo e paralelepípedos (todos). A modificação dos parâmetros diretamente no código modifica a nuvem gerada dependendo do objeto. A criação e modificação é feita diretamente no respectivo arquivo pcd.

```
1  #include<iostream>
2  #include<fstream>
3  #include<string>
4  #include<cstdlib>
5
6  using namespace std;
7
8  int main()
9  {
10         std::ofstream arquivo;
11         arquivo.open( "arquivoReferencia.txt" , ios::out | ios::
              app );
12         arquivo.seekp( 0 , ios::end); //posiciona o ponteiro de
              insercao no final do arquivo
13
14         //arquivo << "\n"; //pula uma linha no arquivo
15
16         float r = 0;
17         float s = 0;
18
19         //pontos FRENTE
```

```
20          arquivo << 0 << " " << 0.054 << " " << 0.20 << "\n";
               //canto superior esquerdo
21          arquivo << 0.131 << " " << 0.054 << " " << 0.20 << "\n";
               //canto superior direito
22          arquivo << 0 << " " << 0 << " " << 0.20 << "\n";       //
               canto inferior esquerdo
23          arquivo << 0.131 << " " << 0 << " " << 0.20 << "\n";
               //canto inferior direito
24
25          for( int i = 0 ; i < 10000 ; i++ )
26          {
27                  r = static_cast <float> (rand()) / (static_cast <
                       float> (RAND_MAX/0.131)); //gera um numero
                       aleatorio de 0 a 0.15
28                  s = static_cast <float> (rand()) / (static_cast <
                       float> (RAND_MAX/0.054)); //gera um numero
                       aleatorio de 0 a 0.15
29
30                  arquivo << r << " " << s << " " << 0.20 << "\n";
31          }
32
33          //pontos TRAS
34          arquivo << 0 << " " << 0.054 << " " << 0 << "\n";    //
               canto superior esquerdo
35          arquivo << 0.131 << " " << 0.054 << " " << 0 << "\n"; //
               canto superior direito
36          arquivo << 0 << " " << 0 << " " << 0 << "\n";       //
               canto inferior esquerdo
37          arquivo << 0.131 << " " << 0 << " " << 0 << "\n";    //
               canto inferior direito
38
39          for( int i = 0 ; i < 10000 ; i++ )
40          {
41                  r = static_cast <float> (rand()) / (static_cast <
                       float> (RAND_MAX/0.131)); //gera um numero
                       aleatorio de 0 a 0.15
42                  s = static_cast <float> (rand()) / (static_cast <
                       float> (RAND_MAX/0.054)); //gera um numero
                       aleatorio de 0 a 0.15
43
44                  arquivo << r << " " << s << " " << 0 << "\n";
```

```
45          }
46
47          //pontos DIREITA
48
49          for( int i = 0 ; i < 10000 ; i++ )
50          {
51                  r = static_cast <float> (rand()) / (static_cast <
                        float> (RAND_MAX/0.20)); //gera um numero
                        aleatorio de 0 a 0.15
52                  s = static_cast <float> (rand()) / (static_cast <
                        float> (RAND_MAX/0.054)); //gera um numero
                        aleatorio de 0 a 0.15
53
54                  arquivo << 0.131 << " " << s << " " << r << "\n";
55          }
56
57          //pontos ESQUERDA
58
59          for( int i = 0 ; i < 10000 ; i++ )
60          {
61                  r = static_cast <float> (rand()) / (static_cast <
                        float> (RAND_MAX/0.20)); //gera um numero
                        aleatorio de 0 a 0.15
62                  s = static_cast <float> (rand()) / (static_cast <
                        float> (RAND_MAX/0.054)); //gera um numero
                        aleatorio de 0 a 0.15
63
64                  arquivo << 0 << " " << s << " " << r << "\n";
65          }
66
67          //pontos CIMA
68
69          for( int i = 0 ; i < 10000 ; i++ )
70          {
71                  r = static_cast <float> (rand()) / (static_cast <
                        float> (RAND_MAX/0.131)); //gera um numero
                        aleatorio de 0 a 0.15
72                  s = static_cast <float> (rand()) / (static_cast <
                        float> (RAND_MAX/0.20)); //gera um numero
                        aleatorio de 0 a 0.15
73
```

```
74                  arquivo << r << " " << 0.054 << " " << s << "\n";
75          }
76
77          //pontos BAIXO
78
79          for( int i = 0 ; i < 10000 ; i++ )
80          {
81                  r = static_cast <float> (rand()) / (static_cast <
                        float> (RAND_MAX/0.131)); //gera um numero
                        aleatorio de 0 a 0.15
82                  s = static_cast <float> (rand()) / (static_cast <
                        float> (RAND_MAX/0.20)); //gera um numero
                        aleatorio de 0 a 0.15
83
84                  arquivo << r << " " << 0 << " " << s << "\n";
85          }
86
87          arquivo.close();
88
89          return 0;
90  }
```

APÊNDICE B – Tutorial Instalação e Configuração

Esse apêndice trás um roteiro de como instalar a kinect one e configura-lo corretamente para uso, assim como também instrui como fazer a instalação das bibliotecas necessárias para replicar os experimentos deste trabalho.

```
Instalacao do Kinect One no Ubuntu

Requisitos:

- USB 3.0;
- Ubuntu 16.04 ou maior (recomendavel 18.04);

Etapas e Serem Seguidas:

- Instalar o driver de video proprietario mais adequado e recente
   , seguindo a passo a passo no link: https://www.edivaldobrito.
   com.br/recentes-drivers-graficos-proprietarios-no-ubuntu/

        Download do codigo fonte do libfreenect2:

git clone https://github.com/OpenKinect/libfreenect2.git
cd libfreenect2

Instalacao de build tools

sudo apt-get install build-essential cmake pkg-config

Instalacao do libusb. A versao precisa ser maior ou igual a
   1.0.20.

sudo apt-get install libusb-1.0-0-dev

Instalacao TurboJPEG

1. (Ubuntu 16.04) sudo apt-get install libturbojpeg libjpeg-
   turbo8-dev
2. (Ubuntu 18.04 ) sudo apt-get install libturbojpeg0-dev

Instalacao OpenGL

```

```
32 sudo apt-get install libglfw3-dev
33
34 Instalacao OpenCL (opcional)
35
36 1. Intel GPU:
37
38 sudo apt-get install beignet-dev
39
40 Build
41
42 voltar para a pasta raiz do libfreenect2 primeiro;
43
44 mkdir build && cd build
45 cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2
46 make
47 make install
48
49 Aplicacoes de terceiro que utilizarem o libfreenect2 (o kinect
      one) precisam especificar a linha abaixo no arquivo Cmake das
      respectivas aplicacoes, para que seja possivel encontrar o
      driver atraves da aplicacao:
50
51  -Dfreenect2_DIR=$HOME/freenect2/lib/cmake/freenect2
52
53 configurar as regras do udev para acesso de dispositivos:
54
55  sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.
       d/, entao reconecte o Kinect
56
57 Executar o programa teste: ./bin/Protonect
58
59 OBS: O primeiro programa  Captura  Inicial desenvolvido neste
       trabalho encontra-se dentro da pasta examples novo dos
      arquivos desse tutorial. Essa pasta examples novo deve
      substituir a pasta examples original na hierarquia de pastas do
       libfreenect2 ap s esse ser baixado e devidamente instalado,
      da forma como mencionado anteriormente.
60
61
62 Documentacao libfreenect2: https://openkinect.github.io/
      libfreenect2/
```

```
Mais detalhes da instalacao do libfreenect2 em: https://github.com/OpenKinect/libfreenect2/


Instalacao do PCL (1.7.1 ou maior):

sudo add-apt-repository ppa:v-launchpad-jochen-sprickerhof-de/pcl
sudo apt-get update
sudo apt-get install libpcl-all

Download e instalacao do OpenCV (3.4.1 ou maior) com o modulo adicional ARUCO (3.0.6 ou maior):

utilizar o arquivo de instalacao do opencv na mesma pasta desse arquivo.

        bash install-opencv.sh

apos a instala  o do opencv, entrar no site oficial do aruco, fazer o download da versao mais recente e descompactar o arquivo baixado. Apos a descompactacao, ir para a pasta descompactada do aruco e executar os seguintes comandos no terminal:

        mkdir build
        cd build
        cmake ..
        make
        sudo make install

Apos a instalacao e preciso alterar algumas configuracoes para o correto funcionamento:

executar no terminal:

        sudo gedit /etc/ld.so.conf.d/aruco.conf

adicionar a seguinte linha no final do arquivo que abrir e salvar logo em seguida:

        /usr/local/lib
```

```
95
96  no terminal novamente, executar:
97
98          sudo ldconfig
99
100 novamente executar:
101
102         sudo gedit /etc/bash.bashrc
103
104 verificar se as seguintes duas linhas estao no final do arquivo
        que abrir. Se nao estiverem, adicione-as e salve o arquivo:
105
106         PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
107         export PKG_CONFIG_PATH
108
109 por fim, reinicie o computador.
```

```
1   #install-opencv.sh
2   ##########################################
3   # INSTALL OPENCV ON UBUNTU OR DEBIAN #
4   ##########################################
5
6   # |           THIS SCRIPT IS TESTED CORRECTLY ON          |
7   # |-------------------------------------------------------|
8   # | OS                | OpenCV       | Test | Last test   |
9   # |-------------------|--------------|------|-------------|
10  # | Ubuntu 18.04 LTS | OpenCV 3.4.2 | OK   | 18 Jul 2018 |
11  # | Debian 9.5        | OpenCV 3.4.2 | OK   | 18 Jul 2018 |
12  # |-------------------------------------------------------- |
13  # | Debian 9.0        | OpenCV 3.2.0 | OK   | 25 Jun 2017 |
14  # | Debian 8.8        | OpenCV 3.2.0 | OK   | 20 May 2017 |
15  # | Ubuntu 16.04 LTS | OpenCV 3.2.0 | OK   | 20 May 2017 |
16
17
18  # VERSION TO BE INSTALLED
19
20  OPENCV_VERSION='3.4.2'
21
22
23  # 1. KEEP UBUNTU OR DEBIAN UP TO DATE
24
25  sudo apt-get -y update
```

```
# sudo apt-get -y upgrade        # Uncomment this line to install the newest versions of all packages currently installed
# sudo apt-get -y dist-upgrade   # Uncomment this line to, in addition to 'upgrade', handles changing dependencies with new versions of packages
# sudo apt-get -y autoremove     # Uncomment this line to remove packages that are now no longer needed


# 2. INSTALL THE DEPENDENCIES

# Build tools:
sudo apt-get install -y build-essential cmake

# GUI (if you want to use GTK instead of Qt, replace 'qt5-default' with 'libgtkglext1-dev' and remove '-DWITH_QT=ON' option in CMake):
sudo apt-get install -y qt5-default libvtk6-dev

# Media I/O:
sudo apt-get install -y zlib1g-dev libjpeg-dev libwebp-dev libpng-dev libtiff5-dev libjasper-dev libopenexr-dev libgdal-dev

# Video I/O:
sudo apt-get install -y libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev yasm libopencore-amrnb-dev libopencore-amrwb-dev libv4l-dev libxine2-dev

# Parallelism and linear algebra libraries:
sudo apt-get install -y libtbb-dev libeigen3-dev

# Python:
sudo apt-get install -y python-dev python-tk python-numpy python3-dev python3-tk python3-numpy

# Java:
sudo apt-get install -y ant default-jdk

# Documentation:
sudo apt-get install -y doxygen
```

```
56
57
58 # 3. INSTALL THE LIBRARY
59
60 sudo apt-get install -y unzip wget
61 wget https://github.com/opencv/opencv/archive/${OPENCV_VERSION}.
      zip
62 unzip ${OPENCV_VERSION}.zip
63 rm ${OPENCV_VERSION}.zip
64 mv opencv-${OPENCV_VERSION} OpenCV
65 cd OpenCV
66 mkdir build
67 cd build
68 cmake -DWITH_QT=ON -DWITH_OPENGL=ON -DFORCE_VTK=ON -DWITH_TBB=ON
      -DWITH_GDAL=ON -DWITH_XINE=ON -DBUILD_EXAMPLES=ON -
      DENABLE_PRECOMPILED_HEADERS=OFF ..
69 make -j4
70 sudo make install
71 sudo ldconfig
72
73
74 # 4. EXECUTE SOME OPENCV EXAMPLES AND COMPILE A DEMONSTRATION
75
76 # To complete this step, please visit 'http://milq.github.io/
      install-opencv-ubuntu-debian'.
```

ÍNDICE DE ASSUNTOS

Índice de assuntos

Abordagem Baseada em Pulsos, 25
alinhamento, 47
alvos codificados, 37
animação, 27
arquitetura, 27
ARUCO, 18, 37

bola de sinuca, 48
Boris Delaunay, 32

câmeras tempo de voo, 25
campos de visão, 22
condição Delaunay, 32
construção do setup, 40
correlação espacial, 25
cubo, 48

desvio padrão, 57
digitalização, 44
digitalizadores 3D, 27
Distância de Hausdorff, 55
distância de Hausdorff Mínima, 57

equação do plano, 45

flipping, 32

geração de malhas, 36

ICP, 47
icp, 27
imageamento médico, 27
inspeção de qualidade, 27
iterative closest point, 27

Kinect, 15
Kinect Fusion, 34
Kinect One, 15, 21

laser scanners, 15
libfreenect2, 18, 36
licença BSD, 37
limite direito, 45
limite esquerdo, 45
limite inferior, 45
limite superior, 45
luz estruturada, 15

Métrica de Hausdorff, 55
Meshlab, 19, 36
metrologia, 27
Microsoft Corporation, 15
Modulação de Intensidade de Onda Contínua, 25

nuvem de pontos, 27

OpenCV, 18, 39
outliers, 47

paralelepípedo de madeira maciça, 48
PCL, 18, 37
Playstation Move, 21

realidade virtual, 27
reconhecimento de esqueletos, 21
reconhecimento de gestos, 22
renderização, 36
robótica, 16

Screened Poisson, 61
segmentação, 44
sreened poisson, 31

tempo de voo, 25
texturização, 36
time-of-flight, 15, 21
triangulação de Delaunay, 32, 66

valor médio, 57
variância, 57

Wii Remote Control, 21

Xbox One, 16, 21