UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

MOLLIOREM: RECONSTRUÇÃO DO TÓRAX FEMININO

MARINA ULIANO

BLUMENAU
2014

2014/1-16

MARINA ULIANO

MOLLIOREM: RECONSTRUÇÃO DO TÓRAX FEMININO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Aurélio Faustino Hoppe, Mestre - Orientador

BLUMENAU
2014

2014/1-16

MOLLIOREM: RECONSTRUÇÃO DO TÓRAX FEMININO

Por

MARINA ULIANO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
	_____
Membro:	Prof. Dalton Solano dos Reis, Mestre – FURB
	_____
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 9 de julho de 2014

Dedico este trabalho a única coisa mais importante no mundo para mim, minha família.

AGRADECIMENTOS

Agradeço ao Google por sempre me prover material de pesquisa e principalmente ao Google *Translate* por sempre traduzi-los.

Agradeço a minha família, minha mãe Maria Margarida e Ricardo Uliano por aguentar meu humor inconstante por conta do nervosismo.

Agradeço ao meu namorado por me aconselhar e me incentivar e sempre estar ao meu lado.

Agradeço ao meu orientador, Aurélio Faustino Hoppe, por me aconselhar e acreditar no meu trabalho.

Agradeço a bancada das lamentações que sempre divertiu os dias em que eu estava cabisbaixa.

Vai trabalhar, vagabundo. Vai trabalhar, vagabundo. Deus permite a todo mundo, ao menos uma loucura.

Chico Buarque

RESUMO

Este trabalho descreve os passos necessários (aquisição, filtro, estimativa de normal e por último a triangulação) para fazer a reconstrução tridimensional de um tórax a partir de uma nuvem de pontos. Para isto foi utilizado o dispositivo Microsoft Kinect e a biblioteca *open source Point Cloud Library* (PCL) para manipular a nuvem de pontos adquirida. Em cada etapa são realizadas comparações dos resultados entre os algoritmos fornecidos pelo PCL para ver qual deles se adapta melhor ao cenário deste trabalho. A partir dos testes realizados encontrou-se o melhor conjunto de algoritmo de filtro, estimativa de normal e triangulação. Também foram realizados testes alterando a profundidade dos pontos para que pudesse ser simulado um implante mamário, porém os resultados não foram satisfatórios.

Palavras-chave: Microsoft Kinect. Nuvem de pontos. Reconstrução 3D. PCL.

ABSTRACT

This work describes the necessary steps to realize the three-dimensional reconstruction of a thorax using point clouds (acquisition, filter, normal estimation and at last the triangulation. It was used the Microsoft Kinect device and the open source library Point Cloud Library (PCL) to manipulate the obtained point cloud. In each step the comparisons of the results are made between the algorithms provided by the PCL, to find which of them adapts better to the scenario of this work. Based on the experimentations the best set of algorithms were found for filter, normal estimation and triangulation. Also tests were done changing the depth of the points to simulate a breast implant, however the results were not satisfactory.

Key-words: Microsoft Kinect. Point cloud. Reconstruction 3D. PCL.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

DSP - *Digital Signal Processing*

E3 - *Electronic Entertainment Expo*

EA - Enterprise Architect

FPS - *Frames Per Second*

ICP - *Iterative Closest Point*

IDE - *Integrated Development Environment*

Ogre - *Open Source 3D Graphics Engine*

OpenCV - *Open Source Computer Vision*

OpenMP - *Open Multi-Processing*

PCD - *Point Cloud Data*

PCL - *Point Cloud Library*

RGB - *Red Green Blue*

ROS - *Robot Operation System*

UML - *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

A reconstrução tridimensional é uma área amplamente estudada nos campos de Visão Computacional e Visão Cientifica (JOSÉ, 2008). Segundo José e Lopes (2007), o objetivo da reconstrução é obter informações tridimensionais baseados em ambientes reais e ambos concordam que a reconstrução 3D pode ser utilizada para auxiliar diversas áreas como medicina, arquitetura, cartografia, automação industrial, entre outros.

Existem duas formas de realizar a reconstrução 3D, baseado em superfícies e baseado em volumes (JOSE; LOPES, 2007). Na reconstrução por superfícies é necessário adquirir as informações por um dispositivo, como *Laser Scanner Cyberware Hires* citado por José e Lopes (2007) ou Microsoft Kinect citado por Jorge e Reis (2013); ambos são capazes de obter uma malha de pontos referente ao mundo real. Para realizar a reconstrução 3D por volumes, podem ser utilizadas imagens adquiridas de qualquer lugar (JOSE; LOPES, 2007).

Segundo Silva (2014), o Kinect, dispositivo criado pelo Microsoft, inicialmente era utilizado como um sensor apenas para jogo, mas cada vez mais os entusiastas tem-se utilizado deste dispositivo como *scanner* 3D. Este fenômeno segundo Borges (2013), se deve pelo custo do aparelho, mais barato que outros *scanners* 3D que estão mercado e a popularidade dele como sensor para entretenimento, ganhando assim mais destaque.

Para realizar a reconstrução 3D existem algumas bibliotecas que facilitam o processo de recebimento de informações do Kinect como *libfreenect* e *Point Cloud Library* (PCL) destacado por Martins (2014) e Kinect SDK mencionado por Mendes (2012). Existem também bibliotecas expostas por Martins (2014) que servem para manipular a nuvem de pontos ou visualizá-las como *libfreenect*, *Open Source Computer Vision Library* (OpenCV) e PCL.

Atualmente, bem como há tempos, as mulheres estão à mercê da tríade: ser bela, ser jovem, ser saudável. (DEL PRIORE 2000 apud MINATO; TRAESEL, 2009). Segundo CRMMedic (2010), as principais dúvidas das mulheres em relação a mamoplastia de aumento são: a prótese pode sair do lugar e qual o tamanho da prótese que se deve.

De acordo com CRMMedic (2010), o primeiro passo de um mamoplastia é saber qual o tamanho dos seios que a mulher gostaria de ter. Após isto, o médico deve fazer uma avaliação completa para sugerir o tamanho ideal. Todavia, mesmo que o médico faça todos os exames necessários para viabilizar o tamanho da prótese e explique à paciente como será o resultado, ela ainda não terá noção do resultado da cirurgia.

Diante do exposto, este trabalho apresenta o desenvolvimento de um protótipo para avaliar e fazer a reconstrução 3D da região torácica das mulheres (seios), para que as mulheres possam analisar melhor a região onde será aplicado o implante.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é fazer a reconstrução 3D de um tórax feminino a partir de uma nuvem de pontos.

Os objetivos específicos do trabalho são:

a) utilizar o Microsoft Kinect como *scanner* 3D para adquirir a nuvem de pontos;
b) utilizar a biblioteca PCL para realizar a manipulação das nuvens de pontos, obtendo no final a reconstrução 3D.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está subdividido em quatro capítulos que serão abordados a seguir.

O primeiro capítulo apresenta a introdução de reconstruções 3D e a justificativa do tema escolhido.

O segundo capítulo aborda os conceitos utilizados para realizar a reconstrução por nuvens de pontos, desde o método de aquisição (Microsoft Kinect) até a biblioteca utilizada (PCL). Este capítulo também descreve alguns trabalhos correlatos que possuem características semelhantes a este trabalho.

O terceiro capítulo aborda os requisitos do protótipo e o diagrama de atividade. Também é explicado como o protótipo foi desenvolvido mostrando os valores utilizados e trechos de código. Este capítulo relata os testes entre os algoritmos do PCL e as discussões sobre os resultados destas comparações.

No quarto capítulo são expostas as conclusões que se chegou a partir das pesquisas e testes realizados durante todo o processo de desenvolvimento. Este capítulo também apresenta possíveis trabalhos futuros que podem ser realizados a partir deste.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado na seção 2.1 o *hardware* utilizado para aquisição da nuvem de pontos. Na seção 2.2 o conceito de reconstrução tridimensional de nuvens de pontos. A seção 2.3 trata da biblioteca PCL. Na última seção (2.4), são apresentados quatro trabalhos correlatos.

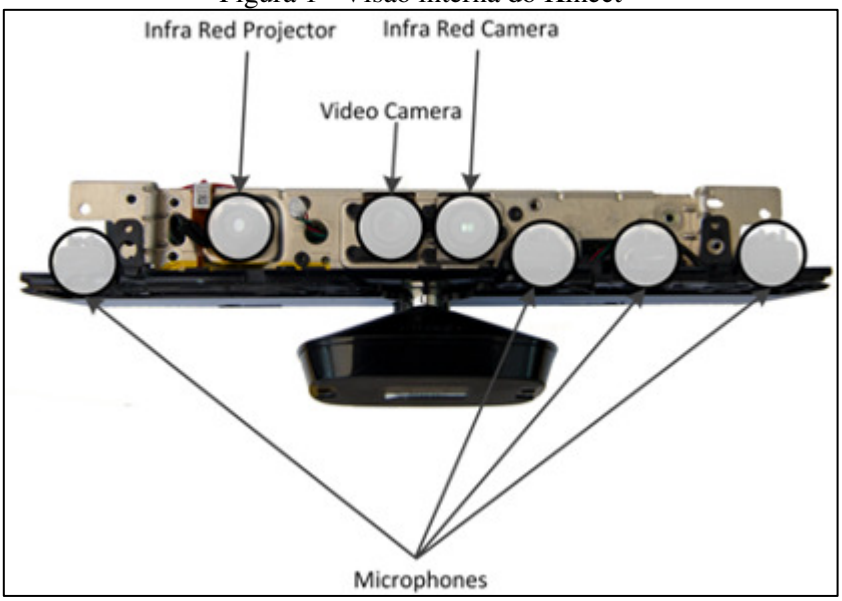
2.1 KINECT

O Kinect é um dispositivo capaz de capturar movimentos de objetos expostos a ele. Foi desenvolvido pela Microsoft e teve sua divulgação oficial anunciada em 2009 na *Electronic Entertainment Expo* (E3). Na época o projeto de codinome Projeto Natal foi apresentado como uma tecnologia para jogos no qual o usuário controlaria os objetos do jogo sem o uso dos tradicionais *joysticks*, mas apenas com movimentos corpóreos (LOWENSOHN, 2011).

O Kinect é um dispositivo que possui uma avançada combinação de hardware e software (Figura 1). Segundo Sá (2011, p. 19), os principais componentes de hardware encontrados no Kinect são:

a) áudio: possui quatro microfones e *Digital Signal Processing* (DSP). Permitindo a distinção da direção da fonte sonora para saber qual fonte sonora está relacionada a um objeto ou pessoa e, também, permitindo ignorar sons indesejáveis e ecos (MILES, 2012);
b) vídeo: possui uma câmera para capturar imagens com resolução de até 1280x1024, mas normalmente utilizada em 640x480 devido a quantidade de *Frames Per Second* (FPS). Na resolução de 1280x1024 são 15 FPS e em 640x480 são 30 FPS (MILES, 2012);
c) profundidade: o Kinect possui dois elementos para obter a profundidade dos objetos a sua frente. O primeiro é um projetor infravermelho que projeta vários pontos infravermelhos. O segundo é uma câmera infravermelha, a qual detecta esses pontos e conforme a distância entre eles é calculada a distância desta área até o Kinect (MILES, 2012).

Figura 1 - Visão interna do Kinect



Fonte: Miles (2012, p. 5).

A verdadeira inovação do sensor está na forma como estes componentes foram unidos para reconhecimento de objetos e pessoas (CRAWFORD, 2010). Com estes dispositivos o Kinect pode prover informações para que um software seja capaz de utilizar movimentos e voz para jogar. Apesar da maioria das pessoas conhecerem ele apenas como mais um dispositivo conectado ao Xbox com o objetivo de proporcionar mais interatividade em jogos, o Kinect vem sendo utilizado juntamente com o computador (Kinect *for* Windows) para desenvolvimento de aplicações, tais como ferramentas de modelagem Autodesk, luvas de controle, rastreamento de mão em 3D, controle para o Google TV, relaxar pacientes, compra de vestuário, entre outras utilidades (CASTRO, 2013).

2.2 RECONSTRUÇÃO 3D DE NUVENS DE PONTOS

Nas próximas seções serão abordados conceitos referentes à reconstrução 3D de nuvens de pontos. Na seção 2.2.1 é clarificado o significado de nuvens de ponto e na seção 2.2.2 será apresentado métodos utilizados para realizar a reconstrução 3D.

2.2.1 Nuvem de pontos

Cousins e Rasu (2011) definem nuvem de pontos como uma estrutura de dados que representa uma coleção de pontos multidimensionais (Figura 2) e é comumente usado para representar dados tridimensionais. Este tipo de representação está cada vez mais comum por causa da popularização dos *scanners* 3D (VALDIVIA, 2013), como o Microsoft Kinect.

Figura 2 - Nuvem de pontos de alguns objetos reais.

Fonte: Cousins e Rasu (2011).

Uma problemática das nuvens de ponto são os ruídos gerados pelo método de aquisição e a quantidade de pontos, que deixa a nuvem de pontos densa. Conforme descrito por Pizzo (2009, p. 6), tratar ruídos não é uma tarefa trivial, pois não se pode simplesmente remover ou adicionar pontos, já que fazendo isso podem ser geradas informações falsas. Para resolver esta situação são definidos parâmetros relacionados ao objeto a ser reconstruído para que o algoritmo possa solucionar o problema de reconstrução de superfícies (PIZO, 2009, p. 6).

2.2.2 Métodos de reconstrução 3D de nuvens de ponto

Os métodos de reconstrução por nuvens têm como objetivo manter o resultado da execução com a mesma topologia da nuvem e também manter a profundidade, além de manter performance e espaço (GOIS, 2004, p. 26)**.**

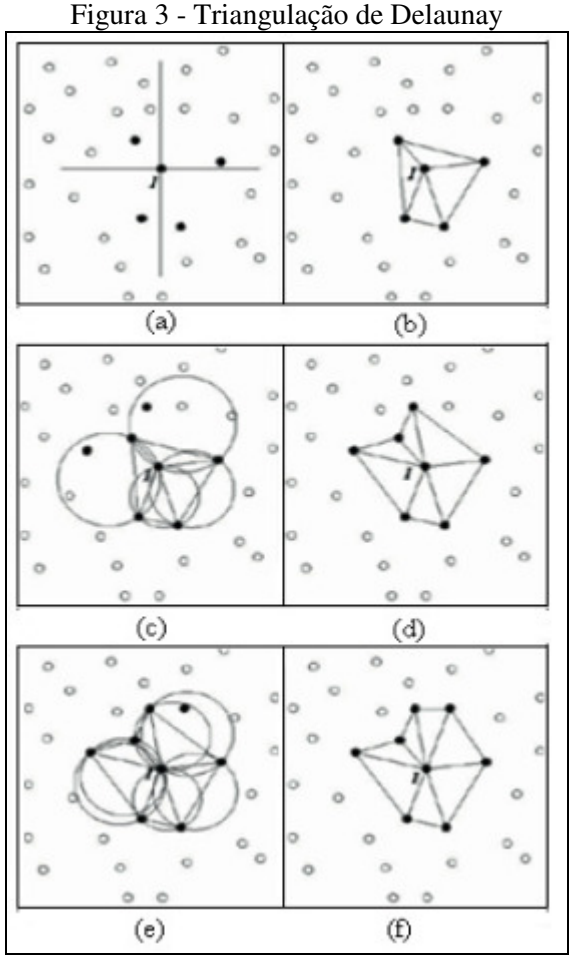
Antes da etapa de triangulação existem algumas etapas que podem ser aplicadas na nuvem, como:

a) suavização de superfícies: seu objetivo é remover o ruídos e diminuir a densidade da nuvem de pontos, mantendo as feições originais tanto quanto possível (VALDIVIA, 2013);
b) estimativa da normal: definem a direção que o polígono irá seguir. Existem basicamente duas abordagens diferentes para a estimativa da normal. A primeira delas é a reconstrução da superfície representada pelos pontos, e calcular as normais a partir daí. A outra abordagem é aproximar os dados normais diretamente na nuvem de pontos (HYVARINEN, 2012). As normais podem servir como entrada de algoritmos para remoção de ruídos, reconstrução de superfícies e detecção de padrões (VALDIVIA, 2013).

Na criação de superfícies, Gois (2008) classificou a reconstrução de nuvem de pontos da seguinte forma:

a) métodos de decomposição espacial: primeiro é feita a triangulação de Delaunay[1] (Figura 3), para que posteriormente sejam removidos os simplexos[2] de forma a conseguir chegar o mais próximo da topologia, como se estivessem esculpindo a malha de triângulos;

Figura 3 - Triangulação de Delaunay



Fonte: Pizo (2009).

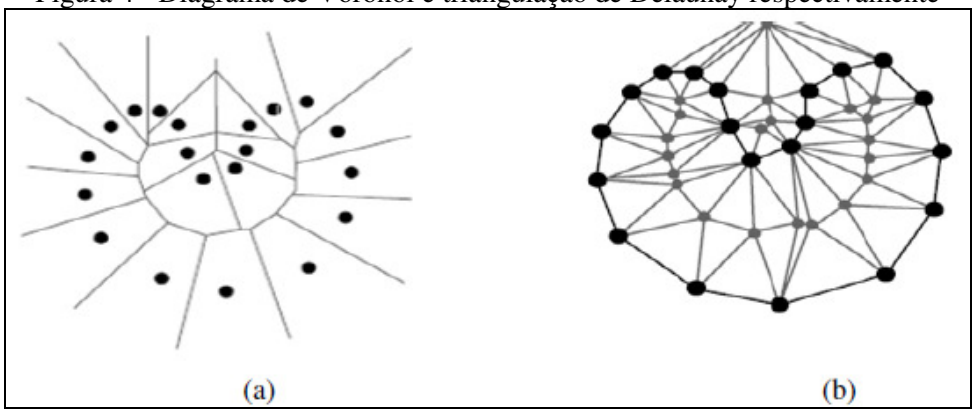
b) métodos incrementais: é o oposto do método anterior, no início é definida uma fronteira composta de um conjunto de *simplexos*, depois a fronteira é ampliada até compreender todos os pontos. Antes de iniciar o método incremental normalmente são utilizadas estruturas de dados *kd-tree* ou *quad-tree* para que seja possível encontrar os próximos pontos candidatos a serem processados;

c) métodos da família *Crust*: tem como principal característica a junção entre o diagrama de Voronoi (Figura 4 item a) e a triangulação de Delaunay (Figura 4 item b);

[1] Segundo Guedes (1996), um triângulo t faz parte da triangulação de Delaunay somente se o círculo formado pelos vértices do triângulo não contém nenhum outro vértice de em seu interior.

[2] Simplexos é o invólucro convexo de (n+1) pontos independentes em $\mathbb{R}^n$

Figura 4 - Diagrama de Voronoi e triangulação de Delaunay respectivamente



Fonte: Pizo (2009).

d) métodos implícitos: como definido por POLIZELLI Junior (2008, p. 10), "A superfície é representada por uma isosuperfície de uma função de distância com sinal".

e) métodos baseados em modelos deformáveis: inicialmente ele busca um modelo que englobe o conjunto de pontos para posteriormente deformar este modelo até que a superfície se ajuste ao conjunto de pontos.

Para facilitar os passos de filtro, estimativa da normal e reconstrução de superfícies foram pesquisadas bibliotecas capazes de suprir todas as necessidades, como o Kinect SDK, *libfreenect* e PCL. A PCL foi a biblioteca escolhida porque apresenta documentação, facilidade de instalação e uso.

2.3 POINT CLOUD LIBRARY

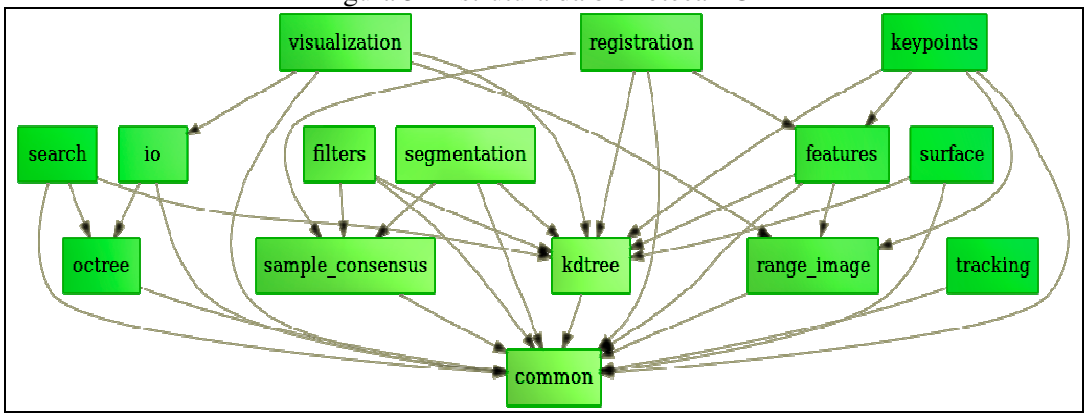
PCL é uma biblioteca *open source*, com integração com a biblioteca para robótica *Robot Operation System* (ROS). O *framework* PCL é escrito em C++ e tem como objetivo a manipulação de nuvens de pontos para modelagens em 3D. Esta biblioteca é compatível com Windows, Mac OS, Linux e Androide a partir da versão 0.6 (COUSINS; RASU, 2011).

Uma das vantagens do PCL é que ele suporta multiprocessamento com o *Open Multi-Processing* (OpenMP), *Threading Building Block* (TBB), conseguindo paralelizar a busca de vizinhos mais próximos deixando o algoritmo mais performático em nuvens densas. Este tipo de processamento pode ser acessado através da classe *Fast Library for Approximate Nearest Neighbors* (FLANN) (COUSINS; RASU, 2011).

Em Point Cloud Library (2014a) é possível encontrar os algoritmos de remoção de ruídos, triangulação, B-Spline, normal e histograma cujo objetivo é manipular nuvens de pontos. Para organizar os algoritmos o PCL foi dividido em módulos tais como: *filter*, *features*, *segmentation*, *surface*, *visualization*, *io*, entre outros (POINT CLOUD LIBRARY,

2014a). A estrutura do PCL com todos os módulos existentes e seus relacionamentos podem ser visto na Figura 5.

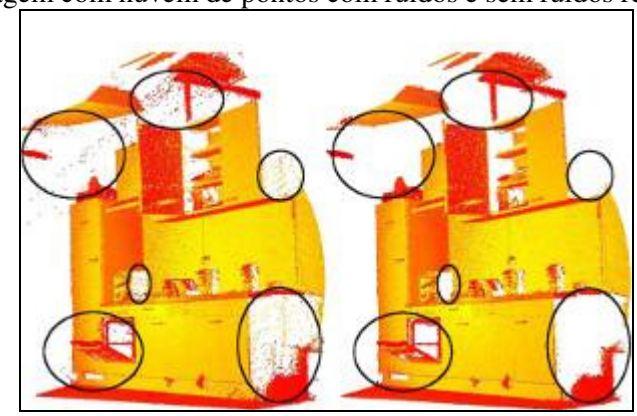
Figura 5 - Estrutura da biblioteca PCL



Fonte: Point Cloud Library (2014b).

Conforme Point Cloud Library (2014b), o principal objetivo do módulo *filter* é diminuir a densidade e/ou remover os ruídos da nuvem de pontos. Estes ruídos podem ser ocasionados por problemas na aquisição, que podem gerar pontos que não refletem o objeto real (Figura 6).

Figura 6 - Imagem com nuvem de pontos com ruídos e sem ruídos respectivamente.



Fonte: Point Cloud Library (2014b).

Alguns dos algoritmos encontrados no módulo *filter* são:

a) `StatisticalOutlierRemoval`: este algoritmo realiza duas varreduras na nuvem de pontos. Na primeira é calculada a distância média que um ponto tem de seus vizinhos (informado no método `setMeanK`) e calcula a média e o desvio padrão de todas as distâncias criando assim um limiar. Na segunda varredura para cada ponto é verificado se a distância dele com seu vizinho se encaixam no limiar definido anteriormente, caso não se encaixe o ponto é removido (POINT CLOUD LIBRARY, 2014b);

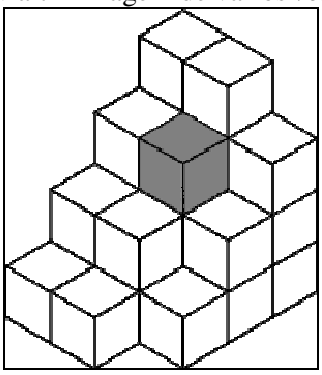
b) `RadiusOutlierRemoval`: verifica num raio informado no método

`setRadiusSearch` se ponto raiz possui a quantidade de vizinhos informados no método `setMinNeighborsInRadius`. Caso esta condição seja falsa o ponto raiz é removido (POINT CLOUD LIBRARY, 2014b);

c) `PassThrough`: verifica se a coordenada informada no método `setFilterFieldName` se encontra entre os limiares informados no `setFilterLimits`. Caso estejam, os pontos são removidos da nuvem de ponto (POINT CLOUD LIBRARY, 2014b);

d) `VoxelGrid`: divide o espaço em *voxel*s (Figura 7) de tamanho fixo informado no método `setLeafSize`, e os pontos que ficam dentro deste cubo se transformam em apenas um ponto (POINT CLOUD LIBRARY, 2014b).

Figura 7 - Imagem de vários *voxels*



Fonte: Voxel (2006).

Segundo Point Cloud Library (2014b) no módulo *features* se encontram algoritmos e estruturas de dados capazes de fazer estimativas a partir de nuvens de ponto. Um dos algoritmos que compõe este módulo é o de estimativa de normal `NormalEstimation`, que usa aproximações para deduzir as normais da superfície de nuvem de pontos.

No módulo de *surface*, segundo Point Cloud Library (2014b), se encontram algoritmos que realizam reconstruções de superfícies tendo como entrada uma nuvem de pontos. Dependendo da tarefa a reconstrução pode usar, por exemplo, um casco (côncavo ou convexo) que representa de forma simplificada uma superfície, malha de polígonos ou uma superfície suavizada com normais.

O *surface* possui o algoritmo capaz de realizar a reconstrução da superfície por triangulação. A classe `GreedyProjectionTriangulation` que executa o algoritmo de triangulação, este é um algoritmo ganancioso (COUSINS; RASU, 2011) que possui sete métodos principais, para auxiliar na criação dos triângulos (POINT CLOUD LIBRARY, 2014b).

No módulo *surface* existe também o algoritmo `MovingLeastSquares`. Ele suaviza a nuvem de pontos, preenchendo com pontos buracos existentes na nuvem para que haja interligações entre os vizinhos. Para isto ele tenta recriar as partes faltantes da superfície por interpolações polinomiais entre os pontos ao redor. Após suavizar a nuvem `MovingLeastSquares` realiza a estimativa de normal (POINT CLOUD LIBRARY, 2014b).

Point Cloud Library (2014b) explica que o módulo *visualization* pode ser comparado com o *Open Source Computer Vision* (OpenCV) HighGUI que é utilizado para visualizações 2D. Mas ao contrário de HighGUI o *visualization* é utilizado para amostras tridimensionais sejam elas polígonos, normais, histograma ou pontos. Neste módulo é possível encontrar a classe `PCLVisualization` que consegue interagir com eventos de teclado ou mouse.

Outra classe que Point Cloud Library (2014b) apresenta é a `CloudViewer`, uma forma mais simples de mostrar nuvens de pontos. Mas ao contrário da `PCLVisualization` não é possível visualizar outras formas que não sejam pontos.

No último módulo citado Point Cloud Library (2014b) explica o módulo *io*, no qual existem classes e funções para nuvens de ponto como: leitura e escrita de arquivos *Point Cloud Data* (PCD), captura de nuvens de pontos a partir de uma variedade de dispositivos de detecção. Os dispositivos suportados pelo PCL são aqueles que compatibilizam com a biblioteca *Open Natural Interaction* (OpenNi), como por exemplo o Microsoft Kinect.

2.4 TRABALHOS CORRELATOS

A seguir são apresentados quatro projetos que utilizam *scanner* 3D para aquisição e manipulação de nuvens de ponto.

Todos os trabalhos utilizam o Kinect como *scanner*, mas o primeiro trabalho "*Surface reconstruction of point clouds captured with Microsoft Kinect*" (HYVARINEN, 2012) utiliza a biblioteca PCL para reconstrução e os trabalhos, "KinectFusion: *Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera*" (IZADI et al., 2011), "Reconstrução de objetos 3D utilizando estruturas de indexação espacial com o Microsoft Kinect" (YAMADA et al., 2013) e "Modelagem tridimensional de ambiente utilizando Kinect" (MENDES, 2012), não fazem referência ao *framework* PCL.

2.4.1 Surface reconstruction of point clouds captured with Microsoft Kinect

O objetivo principal de Hyvarinen (2012) é fazer a reconstrução de nuvens de pontos de um ambiente qualquer, para a representação foi escolhida malha de polígonos. A malha de polígonos é compatível com o modelo *Open Source 3D Graphics Engine* (Ogre) para

posteriormente ser importado para um programa de visualização de ambientes 3D realXtend Tundra (Figura 8).

Figura 8 - Ambiente realXtend Tundra

Fonte: Hyvarinen (2012).

A tese de licenciatura de Hyvarinen (2012) prevê uma nova forma de o usuário interagir com o ambiente 3D. Pessoas que não tem muita habilidade poderão pegar uma cena num ambiente real e modelá-la da forma que preferirem. Para que o resultado final ficasse mais real antes da malha ser exportada, o trabalho utiliza a técnica de coloração de polígono, onde cada polígono recebe uma cor.

Para que o objetivo fosse alcançado foram utilizadas algumas ferramentas, umas voltadas para reconstrução e outras para o projeto, são elas: redmine, git, Qt *framework* e Qt *creator*, realXtend Tundra, módulo Tundra e Ogre.

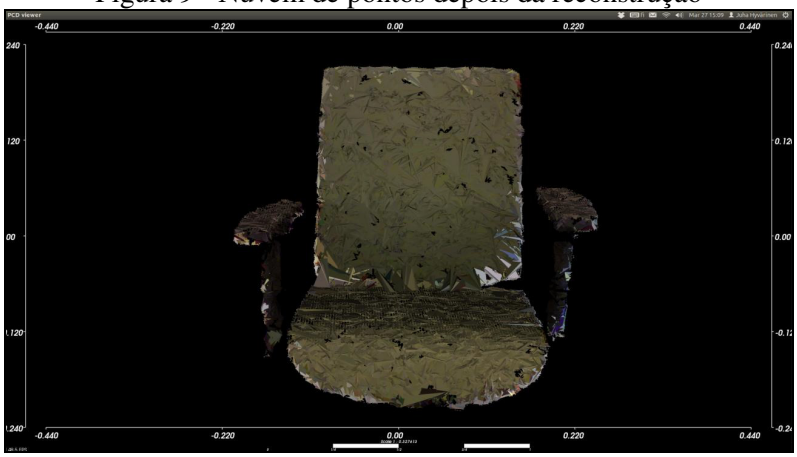
Durante a pesquisa o autor detecta alguns problemas de desenvolvimento com nuvens de pontos. Uma delas são as nuvens adquiridas pelo Microsoft Kinect, pois dependendo da distância entre o sensor e o objeto as nuvens podem conter falhas, como ruídos e buracos deixando a amostragem completamente inadequada, também podem não conseguir capturar detalhes. Estas condições podem ser alteradas para melhorar a condição da nuvem utilizando método como: suavização de ruídos, filtragem da nuvem de pontos, malha simplificada, coloração, preenchimento e reconstrução de superfícies.

Hyvarinen (2012) define como etapas principais no processamento de nuvem de pontos o pré-tratamento, determinação da topologia global para a superfície do objeto, geração da superfície poligonal e pós-processamento. Para a suavização de pontos, normal, remoção de ruídos e reconstrução, Hyvarinen (2012) utilizou a biblioteca PCL.

Para encontrar a melhor solução para o filtro foram implementados três tipos de algoritmos, o primeiro foi por profundidade, segundo por densidade e por último, a extração de *cluster*. No primeiro filtro foi utilizado o algoritmo `PassThrough`, que removeu os pontos entre os limiares 0,4m e 1,4m. No segundo filtro utilizou-se o `VoxelGrid`, que reduziu cerca de 81,1% de pontos. No último algoritmo, extração por *cluster*, a nuvem foi segmentada. Com este algoritmo as partes planas foram quase sempre removidas.

Na reconstrução da malha de polígonos (Figura 9) dois processos foram aplicados, o primeiro a suavização da nuvem de pontos feito pelo algoritmo `MovingLeastSquares` (MLS) fornecida pelo PCL que também estima a normal da nuvem de ponto. Para a criação da malha de polígono o `GreedyProjectionTriangulation` fornecido pela mesma biblioteca.

Figura 9 - Nuvem de pontos depois da reconstrução

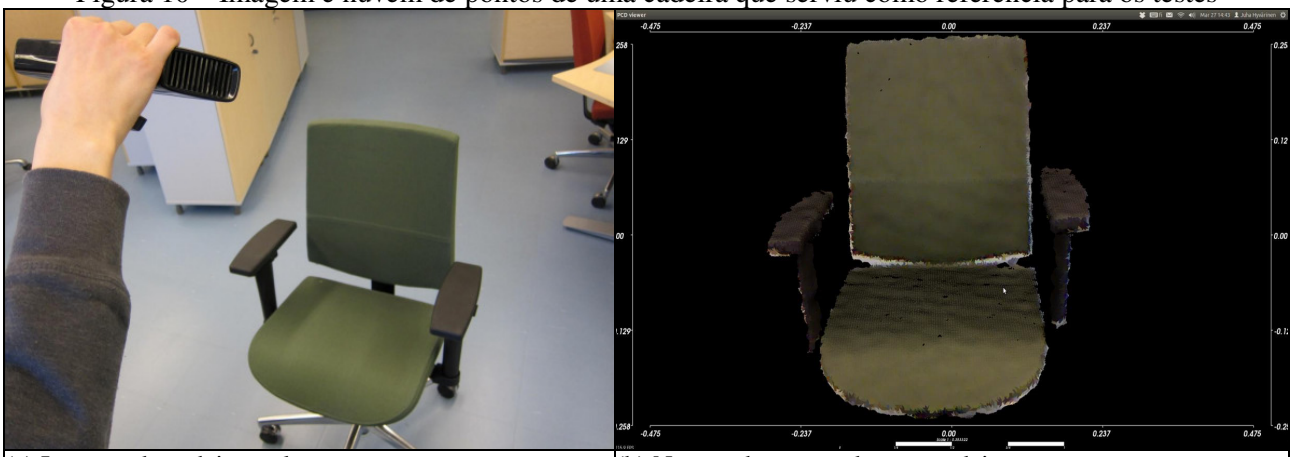


Fonte: Hyvarinen (2012).

Após a triangulação da nuvem de pontos é realizada a conversão da malha de triângulos num arquivo VTK ou Ogre. Para isto o autor utilizou duas formas, com ferramentas externas e utilizando o ManualObject. Depois de convertido é finalmente feita a integração com o módulo Tundra.

Foram realizados testes em cima da nuvem de pontos de uma cadeira (Figura 10 item b), obtido através do *scanner* 3D Microsoft Kinect do objeto real visto na Figura 10 item a. O hardware tinha a seguinte configuração: processador Q8400 a 2.66GHz Intel, com memória RAM de 4GB, placa de vídeo nvidia e um disco rígido com uma capacidade de 500GB. O sistema operacional utilizado nos testes foi o Linux Ubuntu 11.10.

Figura 10 – Imagem e nuvem de pontos de uma cadeira que serviu como referência para os testes

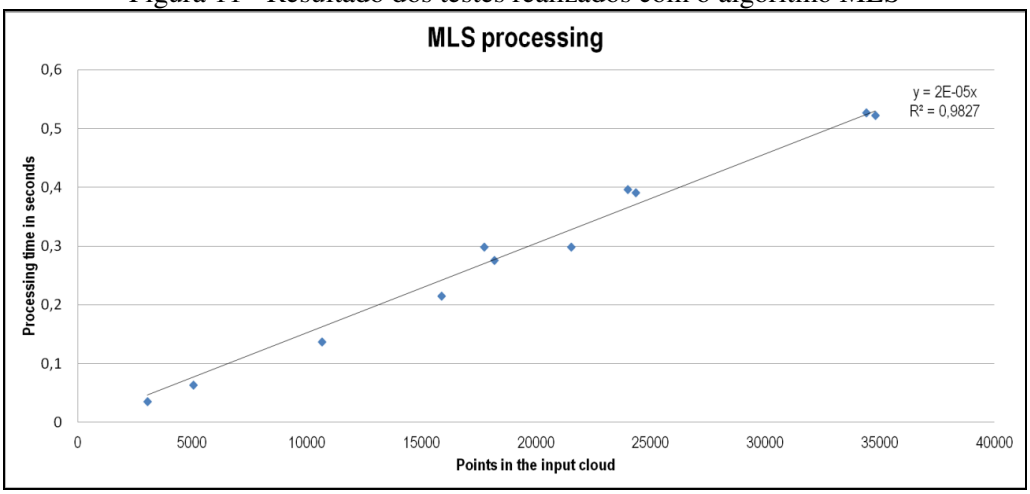


(a) Imagem da cadeira real | (b) Nuvem de pontos de uma cadeira

Fonte: Hyvarinen (2012).

No teste com a suavização de nuvens de pontos (MLS), conforme o gráfico da Figura 11, é possível observar que o tempo do algoritmo cresce à medida que a quantidade de pontos aumenta. Este algoritmo foi o único escolhido para suavização já que nos testes inicias teve o melhor resultado e era bem documentado.

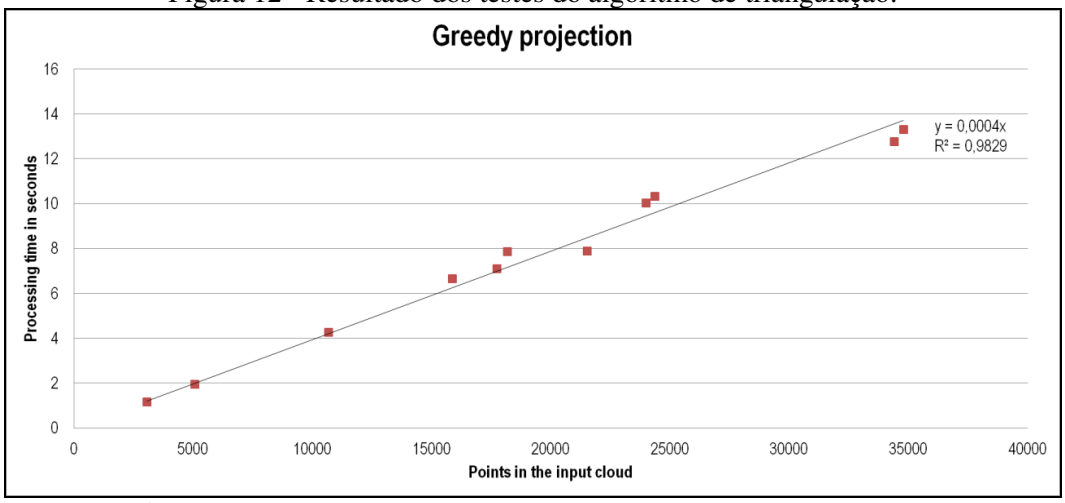
Figura 11 - Resultado dos testes realizados com o algoritmo MLS



Fonte: Hyvarinen (2012).

Nos testes realizados com a triangulação foi possível observar, conforme o gráfico da Figura 12, que o tempo de execução aumenta de forma quase linearmente conforme o numero de pontos aumenta. Para a nuvem utilizada no teste o tempo de execução foi de 10 segundos.

Figura 12 - Resultado dos testes do algoritmo de triangulação.

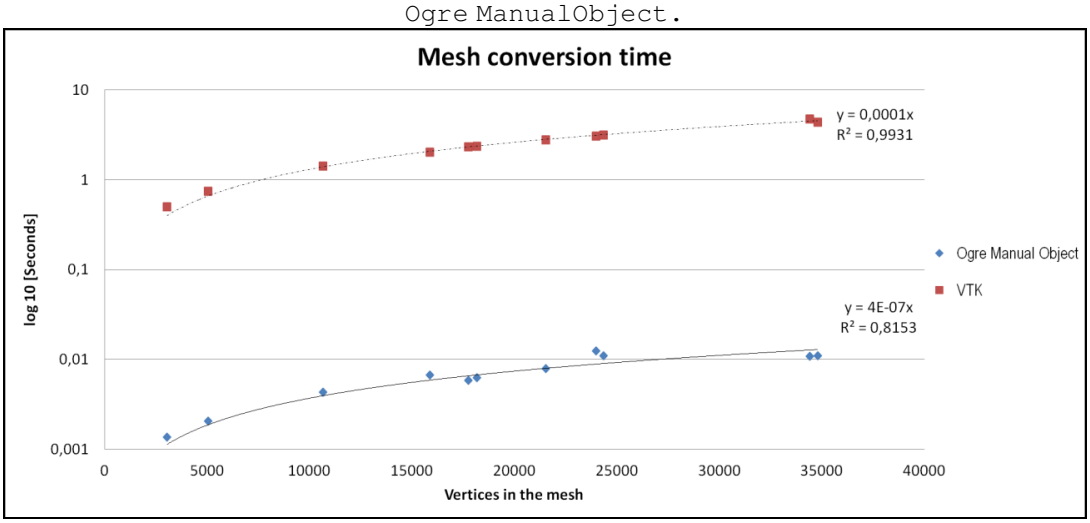


Fonte: Hyvarinen (2012).

Para a conversão de malhas os testes foram mais exaustivos. O autor testou em 11 malhas de polígonos diferentes comparando o tempo que demorou cada conversão, os testes foram feitos tanto com ferramentas externas como com o `Ogre ManualObject`.

Com estes testes viu-se que o tempo aumentou linearmente a medida que a entrada aumentava (Figura 13) em ambos os tipos de conversão, mas com `Ogre ManualObject` o tempo de conversão ficou entre 245 e 442 vezes mais rápidas do que as ferramentas externas.

Figura 13 - Resultados dos testes de conversão da malha de polígonos com ferramentas externas e o `Ogre ManualObject`.



Fonte: Hyvarinen (2012).

Na renderização da malha convertida no Tundra foi observado que malhas utilizando o `Ogre ManualObject` demoram mais tempo para renderizar, quanto maior o arquivo mais demorada a renderização. Nos arquivos convertidos para VTK, mesmo com arquivos de tamanhos maiores o tempo era quase constante.

Alguns problemas encontrados pelo autor, como a criação de polígonos que na versão 1.4 do PCL não era tão eficiente, mas na versão 1.5 já foi melhorada. Ao final da tese o autor

implementou um módulo Tundra, capaz de reconstruir uma malha de polígonos, convertê-la para Ogre e visualizá-la no realXtend de forma automática. O autor também expõe a necessidade de exportar os polígonos para outras extensões, assim podendo abri-los em outros ambientes 3D.

2.4.2 KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera

KinectFusion (IZADI et al., 2011) é uma aplicação que permite a reconstrução de objetos e ambientes através da captura de dados da câmera de profundidade do Kinect. Como dado de entrada o KinectFusion utiliza a nuvem de pontos gerada a partir dos dados de profundidade.

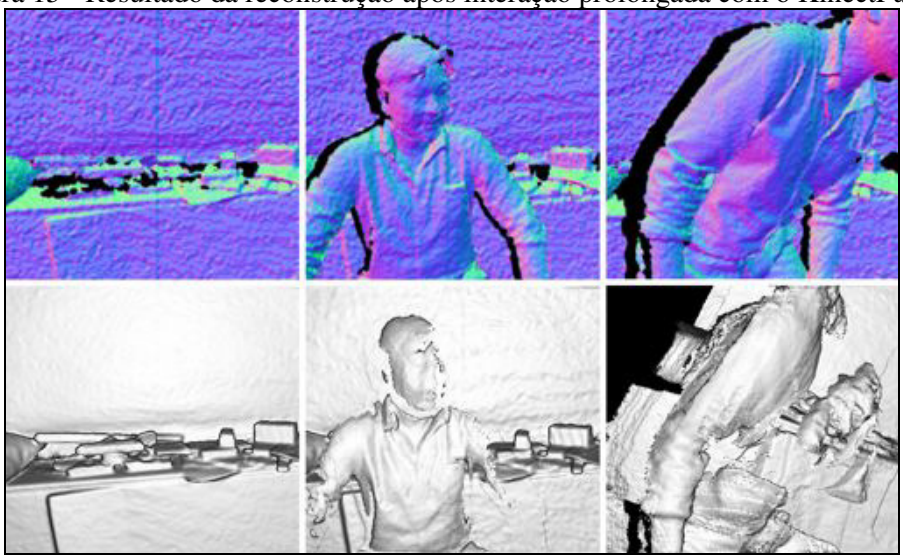
A aplicação é capaz de gerar modelos tridimensionais e aperfeiçoá-los em tempo real. Na Figura 14 é apresentada a reconstrução de um objeto enquanto ele é rotacionado em frente ao Kinect.

Figura 14 - KinectFusion

Fonte: Izadi et al. (2011, p. 3).

O autor menciona que um diferencial do KinectFusion é que, ao contrário da maioria dos trabalhos, ele não espera uma entrada estática, ele é completamente em tempo real, inclusive para melhorar a reconstrução à medida que o *scanner* varre o ambiente, já que desta forma ele reduz a quantidade de buracos que podem existir na nuvem de pontos. Infelizmente se durar por muito tempo esta varredura pode começar a piorar a reconstrução conforme visto na Figura 15**,** por isto é necessário ter alguns cuidados.

Figura 15 - Resultado da reconstrução após interação prolongada com o KinectFusion



Fonte: Izadi et al (2011).

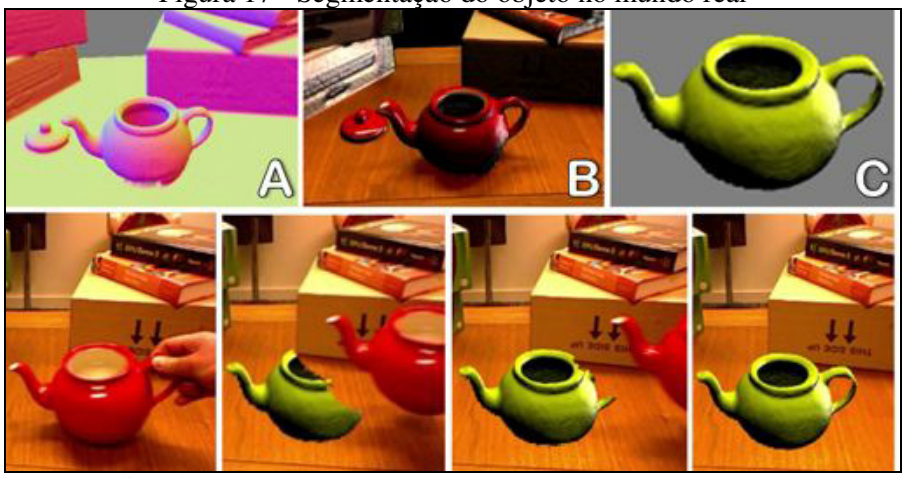
KinectFusion (IZADI et al., 2011, p. 5) permite a interação do mundo real e virtual, no qual objetos complexos se fundem em uma cena real como na Figura 16.

Figura 16 - Simulação realidade aumentada

Fonte: Izadi et al. (2011, p. 4).

Outra faceta da aplicação é a segmentação de objetos, o usuário reconstrói a cena e informa o objeto que deve ser segmentado. Após isto na cena real quando este objeto for removido ele pode ser substituído pelo objeto que foi segmentado antes (Figura 17).

Figura 17 - Segmentação do objeto no mundo real



Fonte: Izadi et al. (2011).

Com todas estas funcionalidades o autor encontrou várias utilidades reais, como na robótica, onde robôs precisam monitorar o ambiente ou visualizar possíveis alterações de cena.

Para implementar o KinectFusion foi focado no paralelismo de GPU e existem quatro etapas principais:

a) conversão do mapa de profundidade: o mapa de profundidade ao vivo é convertido de imagem de coordenada tridimensional e normais no espaço de coordenadas da câmera;
b) movimentação da câmera: para a "mágica" de movimentar o Kinect e a cena seja reconstruída, foi utilizado o algoritmo *Iterative Closest Point* (ICP). Este algoritmo tem como objetivo alinhar duas nuvens de ponto, minimizando a distância entre eles. O ICP é calculado em um GPU paralelamente e quando executado em uma CPU é utilizada a decomposição de Cholesky;
c) integração volumétrica: os pontos são convertidas em coordenadas globais num único *voxel*. Cada *voxel* armazena uma média de execução da sua distância à posição assumida de uma superfície física;
d) *raycasting*: ele é capaz de extrair superfícies implícitas facilitando a visualização do usuário. Ele também é utilizado para remover ruídos das nuvens e melhorar a interação entra nuvens no ICP.

Nas etapas citadas acima pode ser visto o embasamento teórico que tornou possível a interação da aplicação com o mundo real, realizasse reconstrução instantânea com a câmera em movimento, entre outras funcionalidades destacadas.

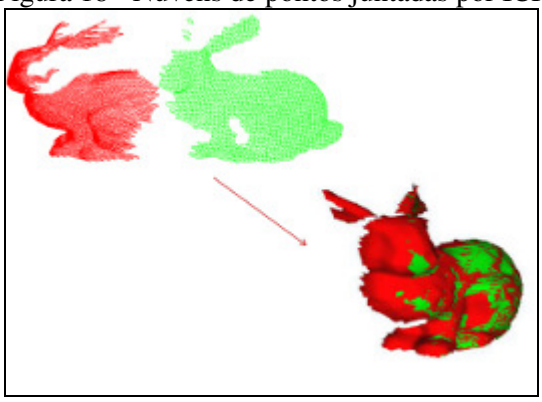
2.4.3 Reconstrução de objetos 3D utilizando estruturas de indexação espacial com o Microsoft Kinect

Com o objetivo de diminuir o custo da aquisição de nuvens de ponto Yamada et al. (2013) utilizaram o *scanner* 3D Microsoft Kinect, inicialmente utilizado como sensor para XBox. Como *driver* responsável por obter os dados do *scanner* os autores utilizaram o *driver open-source* OpenKinect, para que possam obter informações como rotação, RGB de cada ponto e a profundidade, usa-se a biblioteca `libfreekinect`. Esta biblioteca fornece um *array* contendo 307.200 pontos para cada *frame* capturado. As cores estão em outro *array*.

Primeiramente, Yamada et al. (2013) armazenaram os pontos obtidos em um arquivo *.dat* para facilitar a manipulação dos dados. Em seguida, é aplicado um filtro por profundidade que segmenta o objeto desejado, para que isto seja possível é definido um valor baseado na distância que o objeto se encontra do sensor.

Com o *output* do algoritmo de filtro é aplicado o algoritmo de ICP, que tem como objetivo juntar duas nuvens de pontos que estão em perspectivas diferentes como pode ser visto na Figura 18, na qual o coelho da nuvem vermelha está virado para a esquerda e no coelho da nuvem verde está virado com a cabeça para frente.

Figura 18 - Nuvens de pontos juntadas por ICP



Fonte: Yamada et al. (2013, p. 2).

Para fazer o alinhamento por ICP é possível utilizar a cor do ponto, aplicando um peso para cada cor e profundidade, sendo que a cor é uma dimensão diferente, mas o trabalho desses autores se foca apenas na profundidade, por ser mais simples.

O algoritmo implementado pelo autor (Figura 19) basicamente enquanto a distância média entre as duas malhas for menor que um limiar t, a cada iteração o ICP buscará, para cada ponto da malha M, o vizinho mais próximo em P, ou seja, o ponto em P que possui menor distância euclidiana. Com este conjunto de pontos é calculada a distância média e estimada uma transformação rígida que aproxima o conjunto P a M.

Figura 19 - Algoritmo de ICP implementado pelo autor

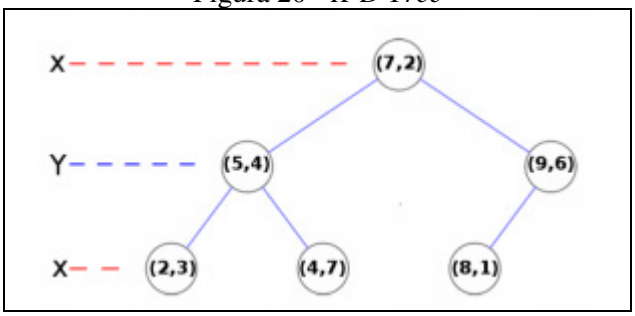
O processo se resume a:

procedure ITERATIVE CLOSEST POINT(M, P, t)
  while $d \leq t$ **do**
    for all $m \in M$ **do**
  Busca vizinho mais próximo em P;
    end for
  Calcula a distância média d;
  Calcula a matriz de transformação T, que aproxima os pontos;
  Aplica T a todos os pontos de M
  end while
end procedure

Fonte: Yamada et al. (2013).

Para executar a busca dos vizinhos mais próximos, os autores utilizam a estrutura de dados *K-D Tree* (Figura 20). Esta estrutura é uma arvore binária balanceada com a complexidade de busca de *log(n)*. Para balancear a arvore é feita a ordenação dos pontos e calculada a mediana, depois, para cada ponto que está depois da mediana é colocado na sub árvore a direita e antes a esquerda.

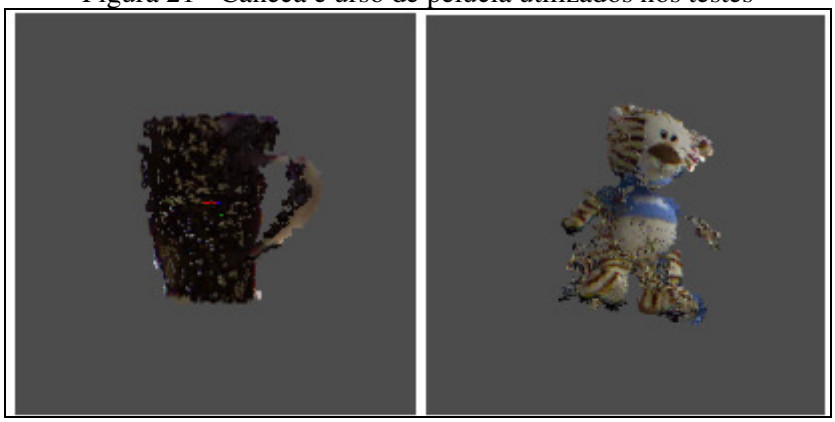
Figura 20 - *K-D Tree*



Fonte: Yamada et al. (2013).

Yamada et al. (2013) fizeram testes de performance realizando três experimentos usando dois modelos distintos: urso de pelúcia e uma caneca (Figura 21).

Figura 21 - Caneca e urso de pelúcia utilizados nos testes



Fonte: Yamada et al. (2013).

No primeiro teste utilizaram-se as nuvens de pontos originais dos objetos. A caneca possuía no primeiro ponto de vista 6.820 pontos e no segundo 7.621 pontos, já o urso de pelúcia tinha no primeiro ponto de vista 10.243 pontos e no segundo 11.202 pontos. Com estas entradas o algoritmo de ICP foi testado primeiramente sem a estrutura de indexação *K-D Tree* tendo o resultado de 7.738s para a caneca e 15.740s para o urso de pelúcia. Utilizando a estrutura *K-D Tree* o tempo para a caneca passou para de 0.109s e para o urso de pelúcia 0.100s. Observando estes dados é fácil notar que a estrutura de indexação *K-D Tree* deixa o algoritmo de ICP mais rápido.

No segundo teste as entradas foram reduzidas em três vezes a quantidade de pontos, com isto o tempo do ICP para a caneca sem *K-D Tree* foi de 0.873s, já com o *K-D Tree* o tempo diminuiu para 0.047s. Utilizando o urso de pelúcia os tempos foram 1.77s sem a estrutura de indexação 0.100s com a estrutura.

Para o último teste os autores reduziram as entradas em cinco vezes, com a amostragem menor os tempos também foram reduzidos. Para o algoritmo de ICP sem o uso da *K-D Tree* demorou 0.343s e com *K-D Tree* demorou 0.031s. Na amostra utilizando o urso de pelúcia os tempos foram de 0.010s e 0.655s, utilizando o *K-D Tree* e sem utilizá-lo respectivamente.

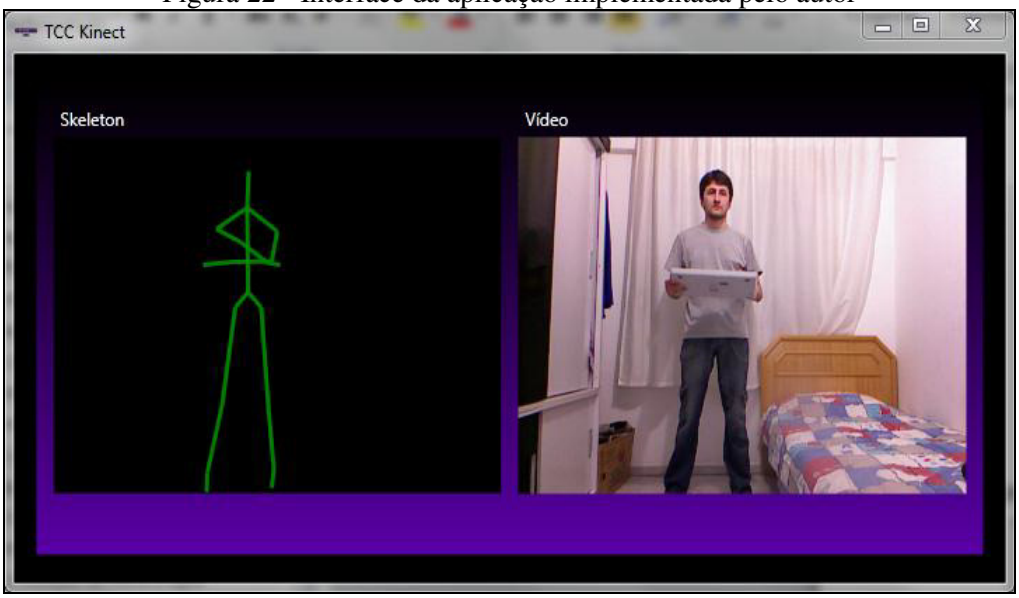
Diante dos três testes realizados, Yamada et al. (2013) notaram que o algoritmo *K-D Tree* com o ICP é muito mais eficiente do que o ICP fazendo a busca de vizinhos mais próximos sem o auxilio da estrutura de indexação. Com isso foi concluído que a junção entre eles torna o Kinect viável para capturar geometrias.

2.4.4 *Modelagem tridimensional de ambiente utilizando Kinect.*

O trabalho desenvolvido por Mendes (2012) tem como objetivo a reconstrução virtual de um ambiente tridimensional. Para isto ele utilizou o Microsoft Kinect como *scanner* 3D para obter a nuvem de pontos de um ambiente real para posteriormente reconstruí-lo.

Neste aplicativo o usuário conseguia interagir com o modelo reconstruído e solicitar a visualização de um vídeo gravado pelo Kinect (Figura 22). Para desenvolver a aplicação foi utilizada a biblioteca OpenTK, linguagem C# e o *framework* para *design* de telas *Windows Presentation Fundation* (WPF).

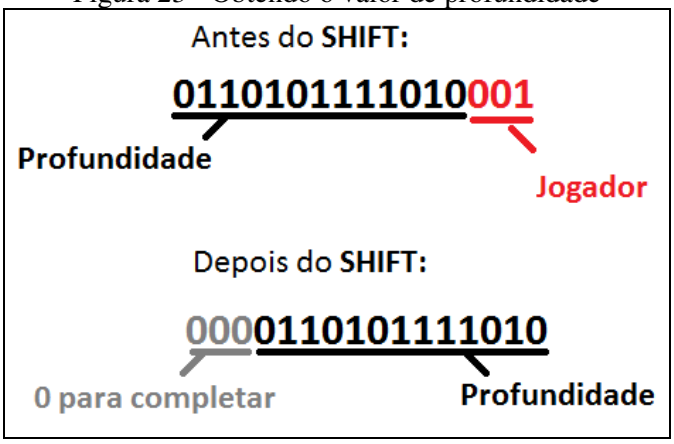
Figura 22 - Interface da aplicação implementada pelo autor



Fonte: Mendes (2012).

Para obter a nuvem de pontos Mendes (2012) utilizou a câmera infravermelha e o projetor infravermelho. Com eles é possível obter as coordenadas do ambiente real e desta forma é possível obter um *frame* e deste *frame* obter um *array* onde estão as informações de profundidade conforme visto na Figura 23. Com estes dados é criado um *array* que seria a nuvem de pontos.

Figura 23 - Obtendo o valor de profundidade

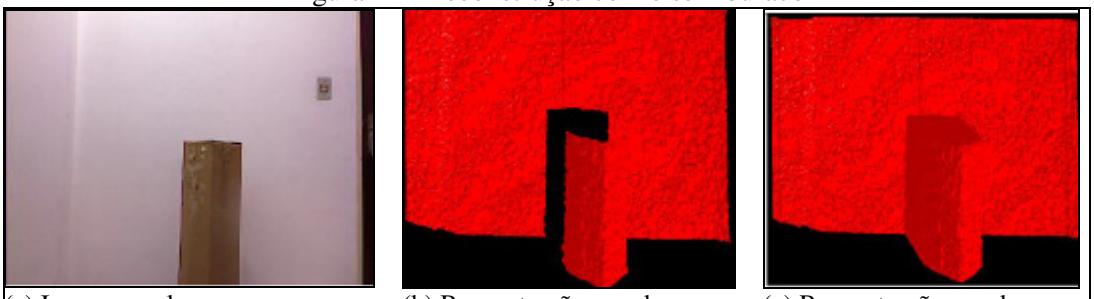


Fonte: Mendes (2012).

Na nuvem criada o autor começa a aplicar o algoritmo de reconstrução de superfícies por triangulação. O algoritmo obtém três pontos iniciais e com base neles é formado um triângulo. Depois do triângulo formado é analisado se o vértice possui distância maior ou igual a 8.191, caso seja, o vértice é removido. É feita analise também de triângulos com arestas maiores que 250. Caso exista, o triangulo é removido para que desta forma sejam criados buracos e, desta forma poder diferenciar o objeto do fundo.

Mendes (2012) utilizou a cor vermelha para rachurar os triângulos e dar textura aos objetos reconstruídos. Na Figura 24 é apresentada a reconstrução de uma caixa defronte a uma parede com e sem buracos.

Figura 24 - Reconstrução com e sem buraco

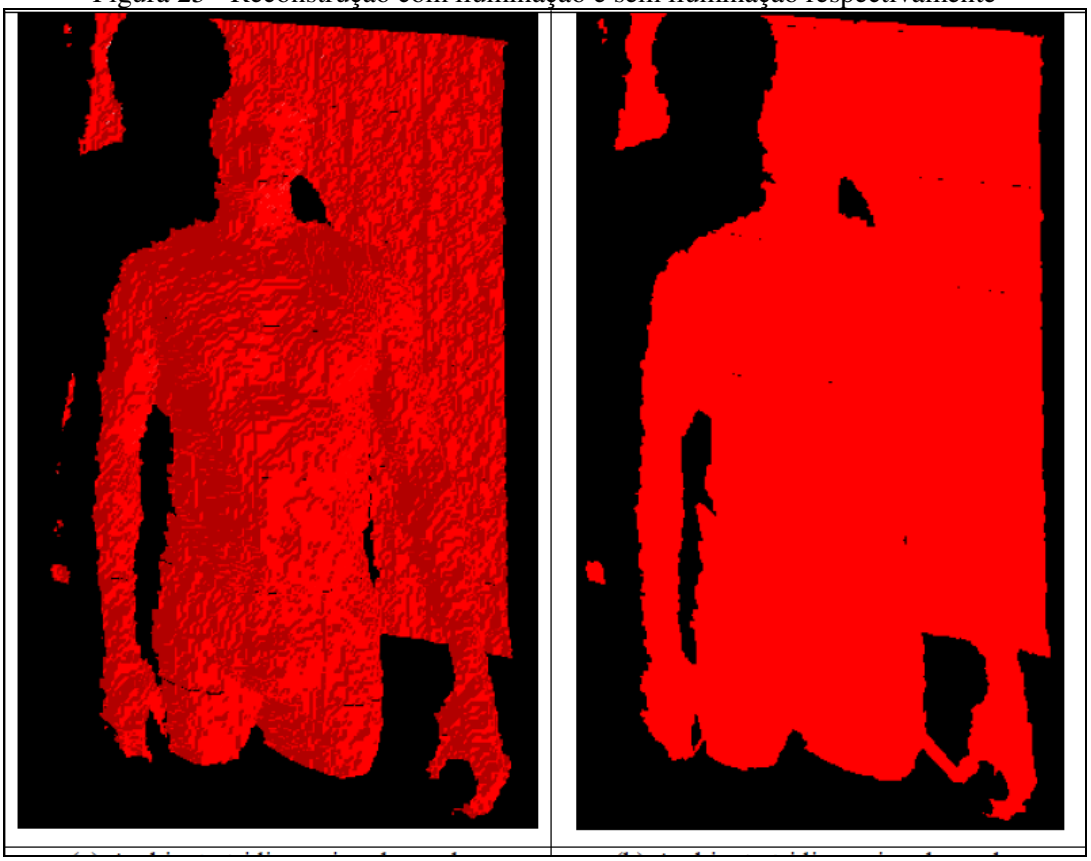


(a) Imagem real (b) Reconstrução com buracos (c) Reconstrução sem buracos

Fonte: Mendes (2012, p. 43).

Durante a implementação o autor percebeu que a iluminação afeta na forma como é reconstruído o ambiente, como pode ser notado na Figura 25.

Figura 25 - Reconstrução com iluminação e sem iluminação respectivamente



Fonte: Mendes (2012).

Mendes (2012) observou que as imagens reconstruídas são muito semelhantes à nuvem de pontos, mas a reconstrução ainda possui imperfeições. Um dos motivos apresentados por Mendes (2012) é a falta de cor de fundo que dificulta o entendimento de onde os objetos se delimitam.

A partir dos testes realizados, Mendes (2012) constatou que a aplicação desenvolvida não possui bons resultados quando um objeto esta próximo a outra superfície, já que o modelo e o fundo sempre são apresentados em vermelho.

2.4.5 Comparação entre os trabalhos correlatos

Nesta seção será apresentado um quadro (Quadro 1) comparativo entre os trabalhos relacionados, no qual as linhas se referem às características extraídas dos trabalhos correlatos, tendo em vista o objetivo e a metodologia desta monografia. Cada coluna é um trabalho correlato descrito na seção anterior.

Quadro 1 - Comparação entre os trabalhos correlatos

características/ trabalho relacionado	Hyvarinen (2012)	Izadi et al. (2011)	Yamada et al. (2013)	Mendes (2012)
utiliza MicrosoftKinect como *scanner* 3D	X	X	X	X
manipulação de nuvens de ponto	X	X	X	X
utiliza PCL	X			
utiliza ICP (incremental)		X	X	
utiliza *K-D Tree*	X		X	
reconstrução de superfícies	X	X		X
reconstrução por triangulação	X			X
filtra a nuvem de pontos	X		X	
interage com mundo real		X		
exporta malha de triângulos para aplicações externas	X			

Conforme visualizado no quadro acima todos os trabalhos tem em comum a nuvem de pontos obtida pelo dispositivo Kinect como entrada.

Yamada et al. (2013) e Hyvarinen (2012) são os únicos que focam na filtragem de nuvem de pontos (remover ruídos e diminuir a densidade) para melhorar os resultados de algoritmos posteriores e Izadi et al. (2011) é o único capaz de interagir com o mundo real.

O trabalho de Yamada et al. (2013) utiliza explicitamente o *K-D Tree*, mas o trabalho de Hyvarinen (2012) utiliza o *framework* PCL que internamente utiliza esta estrutura para fazer a busca de vizinhança. O objetivo de Yamada et al. (2013) é testar o *K-D Tree* juntamente com algoritmo de ICP, mas o trabalho de Izadi et al. (2011) utiliza este algoritmo para que a nuvem de pontos adquirida seja melhorada.

Quase todos os trabalhos correlatos buscam realizar reconstrução 3D, menos Yamada el al. (2013) que apenas realiza testes com *K-D Tree* e o ICP e apenas Hyvarinen (2012) e Mendes (2012) utilizam triangulação para a criação de superfícies.

3 DESENVOLVIMENTO

O capítulo atual tem como objetivo apresentar as etapas realizadas no desenvolvimento do protótipo. A seção 3.1 aborda o levantamento de requisitos, a seção 3.2 apresenta as especificações do protótipo, já a seção 3.3 é visto como o protótipo foi desenvolvido, apresentando partes do código e ferramentas utilizadas. A seção 3.4 aborda como o usuário interage com o protótipo para que a reconstrução tridimensional seja feita. Por fim, na seção 3.5 são abordados testes com vários algoritmos de filtro, estimativa de normal e triangulação para descobrir qual conjunto favorece a reconstrução 3D.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo de reconstrução 3D do tórax feminino deverá possuir os seguintes requisitos:

a) obter a nuvem de pontos do tórax feminino (Requisito Funcional – RF);
b) realizar a reconstrução tridimensional utilizando o PCL (RF);
c) ser implementado utilizando a linguagem C++ (Requisito Não-Funcional – RNF);
d) utilizar o ambiente Visual Studio 2010 para o desenvolvimento (RNF);
e) utilizar o equipamento Microsoft Kinect como *scanner* 3D (RNF);
f) utilizar o *driver* OpenNI juntamente com a biblioteca PCL (RNF);
g) executar no sistema operacional Windows 7 (RNF).

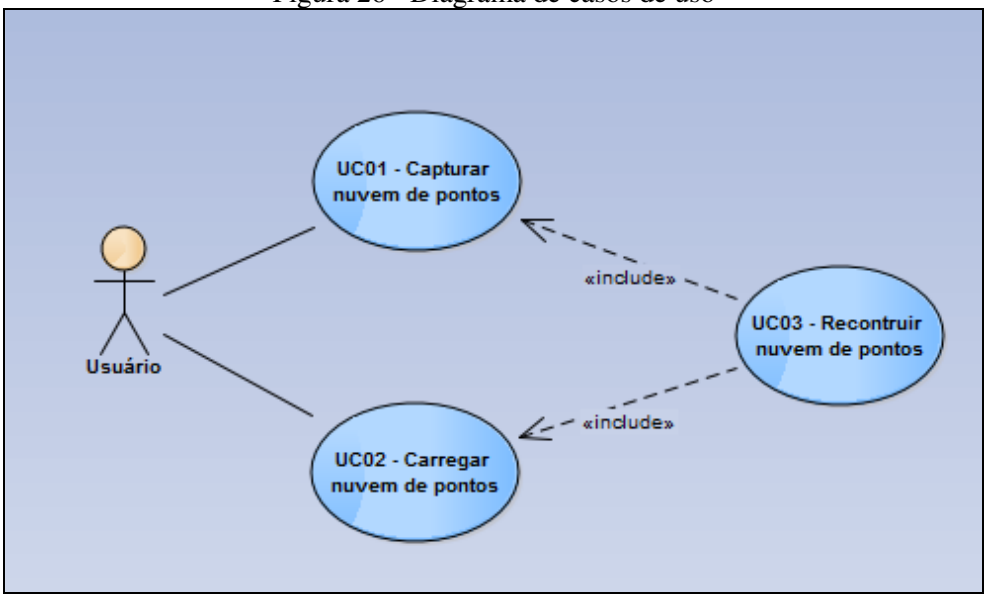
3.2 ESPECIFICAÇÃO

As especificações do sistema foram feitas utilizando a *Unified Modeling Language* (UML) na ferramenta *Enterprise Architect* (EA). Para as especificações foram criados os diagramas de caso de uso, classe e atividade.

3.2.1 Diagrama de casos de uso

Nesta seção são apresentadas as interações que o único ator identificado, `Usuário`, tem com o sistema. Na Figura 26 se encontra o caso de uso modelado para o protótipo.

Figura 26 - Diagrama de casos de uso



Nas próximas seções serão detalhados os casos de uso da aplicação. Na seção 3.2.1.1 é detalhado o caso de uso `UC01` no qual o `Usuário` deseja capturar uma nuvem de pontos. Na seção 3.2.1.2 é detalhado o `UC02` no qual o `Usuário` deseja apenas carregar uma nuvem de pontos existente e, por fim, na seção 3.2.1.3 é feito o detalhamento do caso de uso `UC03` responsável por todas as etapas da reconstrução 3D.

3.2.1.1 UC01 - Capturar nuvem de pontos

Este caso de uso descreve as ações que o `Usuário` deve executar para capturar uma nuvem de pontos a partir do dispositivo Microsoft Kinect. Detalhes do caso de uso podem ser vistos no Quadro 2.

Quadro 2 - UC01 - capturar a nuvem de pontos

Número	01
Caso de Uso	Capturar nuvem de pontos
Descrição	O objetivo deste caso de uso é capturar uma nuvem de pontos a partir do Kinect e salvá-lo no formato pcd.
Ator	`Usuário`
Pré-condições	Que o Kinect já esteja conectado ao computador.
Cenário Principal	1. Usuário inicia o protótipo. 2. O protótipo pergunta se deseja capturar uma nuvem de pontos a partir do Microsoft Kinect. 3. Usuário informa SIM. 4. Usuário pressiona enter. 5. O protótipo pede para informar o nome da nova nuvem de pontos. 6. Usuário informa o caminho e nome do arquivo. 7. Usuário pressiona enter. 8. O protótipo mostra uma tela com a nuvem de pontos. 9. O usuário pressiona a tecla s. 10. O protótipo salva o arquivo em formato pcd. 11. O protótipo fecha a tela. 12. O usuário efetua a reconstrução tridimensional.

3.2.1.2 Carregar nuvem de pontos

No Quadro 3 é apresentado o caso de uso que descreve como o `Usuário` deve proceder para carregar uma nuvem de pontos existente.

Quadro 3 - Caso de uso carregar nuvem de pontos existente

Número	02
Caso de Uso	Carregar nuvem de pontos
Descrição	O objetivo deste caso de uso é carregar um arquivo pcd que já foi salvo anteriormente pelo protótipo ou por qualquer outro software que salve nesta mesma extensão.
Ator	`Usuário`
Pré-condições	Ter um arquivo pcd existente.
Cenário Principal	1. Usuário inicia o protótipo. 2. O protótipo pergunta se o usuário deseja capturar uma nuvem de pontos a partir do Microsoft Kinect. 3. Usuário informa NAO. 4. Usuário pressiona enter. 5. O protótipo pede para informar informe o nome do arquivo contendo a nuvem de pontos. 6. Usuário informa o caminho e nome do arquivo. 7. Usuário pressiona enter. 8. O protótipo efetua a reconstrução tridimensional.

3.2.1.3 Reconstruir nuvem de pontos

Este caso de uso é executado depois dos `UC01` e `UC02`. Ele é responsável por fazer a reconstrução tridimensional da nuvem de pontos. No Quadro 4 é detalhado o caso de uso responsável pela reconstrução 3D.

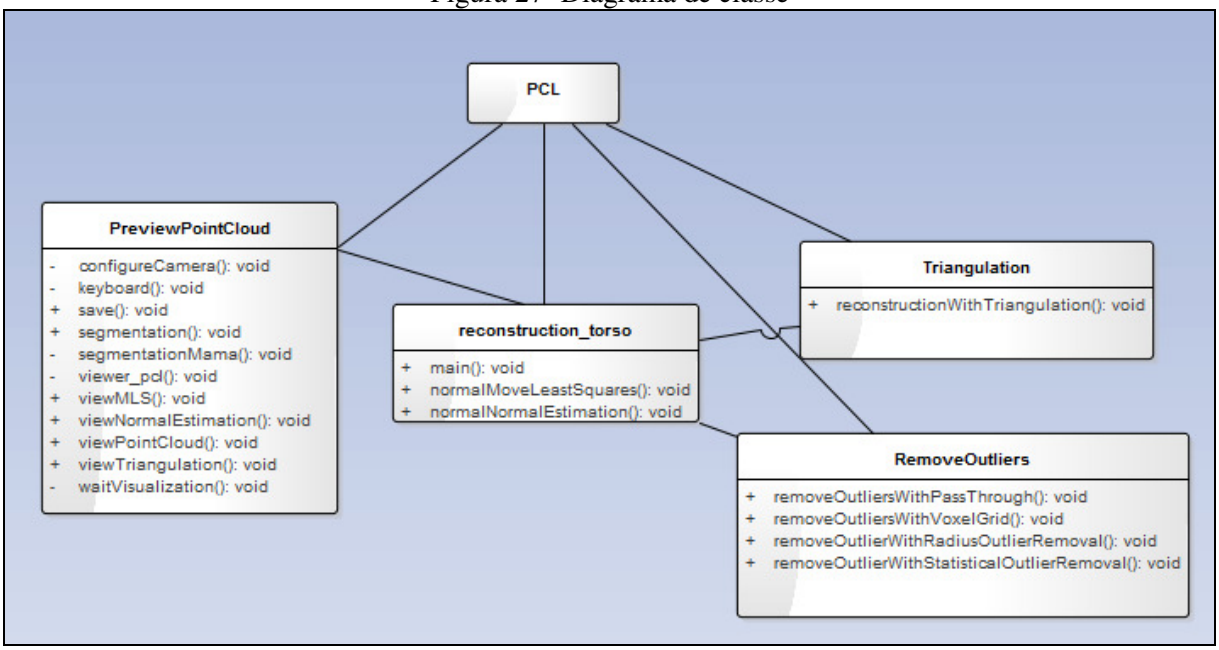
Quadro 4 - Caso de uso reconstruir nuvem de pontos

Número	03
Caso de Uso	Reconstruir nuvem de pontos
Descrição	O objetivo deste caso de uso é fazer a reconstrução 3D de nuvens de pontos que foi informada pelos caso de usos `UC01` ou `UC02`.
Ator	`Usuário`
Pré-condições	Ter a nuvem de ponto salva em arquivo de extensão pcd
Cenário Principal	1. O protótipo lê o arquivo pcd 2. O protótipo remove fundo da nuvem 3. O protótipo diminuir densidade da nuvem 4. O protótipo estima a normal 5. O protótipo concatenar normal com ponto 6. O protótipo realiza triangulação 7. O protótipo apresenta reconstrução 3D

3.2.2 Diagrama de classe

Nesta seção será descrita a estrutura de classes do protótipo desenvolvido. A Figura 27 apresenta as classes utilizadas no desenvolvimento do protótipo.

Figura 27- Diagrama de classe



A classe principal é a `reconstruction_torso`, nela são executados todos os passos para a triangulação tridimensional. No método `main` é aonde o usuário decide se deseja capturar uma nova nuvem de pontos ou apenas carregar uma já existente. Nela, também são realizados os procedimentos de filtro, estimativa de normal e triangulação, chamado método de outras classe.

Para capturar uma nova nuvem de pontos é necessário utilizar o método `save` da classe `PreviewPointCloud`. Este método cria um arquivo no formato PCD contendo as informações da nuvem de pontos, mais especificamente as coordenadas X, Y, Z de cada ponto capturado.

A classe `RemoveOutilier` é responsável por implementar quatro algoritmos de filtro, capazes de remover ruídos e densidade. Todos eles foram utilizados para testes, mas na execução final foram utilizados somente dois algoritmos: `removeOutlierWithPassThroug` para remover o fundo do objeto a ser reconstruído e `removeOutliersVoxelGrid` para diminuir a densidade da nuvem de pontos.

Para realizar a estimativa de normal foi chamado o método `normalMoveLeastSquares` da classe `reconstruction_torso`. Este método executa o algoritmo `MoveLeastSquares` que além de estimar a normal suaviza a nuvem de pontos completando buracos e removendo ruídos.

No último passo a ser executado, a triangulação, foi chamado o método `reconstructionWithTriangulation` e está dentro da classe `Triangulation`. Este método é responsável por receber a nuvem de ponto já com as informações da normal e, a partir disto, reconstruir a superfície utilizando o algoritmo de triangulação.

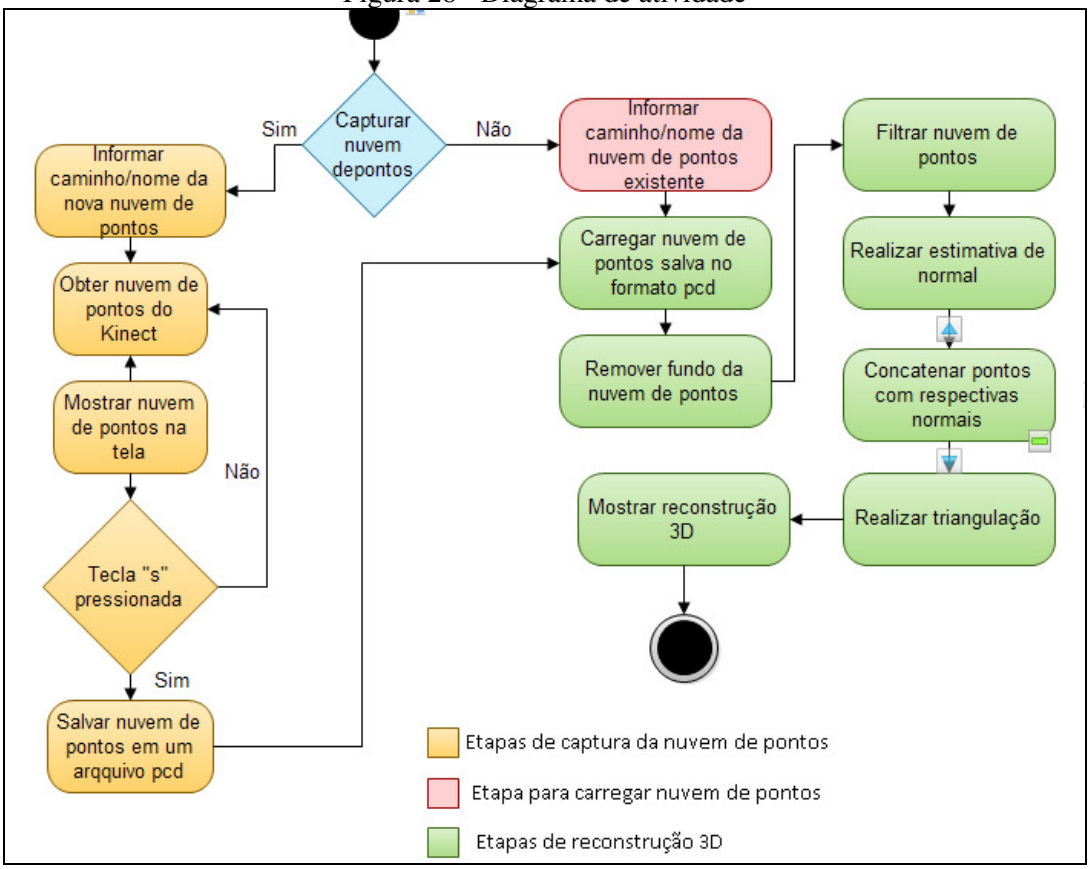
3.2.3 Diagrama de atividade

A Figura 28 apresenta o diagrama de atividade com as etapas que devem ser executadas para realizar a reconstrução tridimensional.

No diagrama, a primeira etapa representa a opção que o usuário deseja realizar, se ele deseja ou não capturar uma nuvem de pontos, caso a resposta seja positiva as etapas para a aquisição de uma nova nuvem de pontos será iniciada.

Primeiramente deve-se informar o local (caminho/nome) onde se pretende salvar a nuvem de pontos. Na próxima etapa é feita a comunicação com o Kinect para capturar a nuvem de pontos e depois apresentá-la. Enquanto o usuário não pressionar a tecla "s" para salvar a nuvem de pontos, o protótipo mantém a comunicação com o Kinect. Quando o usuário pressiona "s" é executada a próxima etapa, no qual a nuvem é salva no formato PCD, que armazena as informações de cada ponto, para uso posterior.

Figura 28 - Diagrama de atividade



Caso o usuário deseje apenas carregar uma nuvem de pontos existente, ele deve informar o local (caminho/nome) do arquivo de extensão PCD que será carregado. As próximas etapas são comuns, independente da escolha da primeira etapa. Primeiramente se carrega a nuvem de pontos e depois se faz a remoção do fundo. Na próxima etapa a nuvem de pontos é filtrada para diminuir a densidade e, posteriormente é realizada a etapa responsável pela estimativa de normal.

Após realizar a estimativa da normal é executada a etapa que concatena cada ponto com sua normal. Na penúltima etapa é realizada triangulação e por fim, é apresentado ao usuário à reconstrução 3D.

3.3 IMPLEMENTAÇÃO

Na seção 3.3.1 será apresentadas as ferramentas e *frameworks* utilizados no desenvolvimento do protótipo. A seção 3.3.2 é responsável por explicar cada uma das etapas (aquisição, filtro, estimativa de normal e triangulação) realizadas no desenvolvimento do protótipo.

3.3.1 Técnicas e ferramentas utilizadas

A aplicação foi desenvolvida no sistema operacional Windows com a linguagem C++ e a *Integrated Development Environment* (IDE) Microsoft Visual Studio 2010, utilizando-se das seguintes tecnologias:

a) Microsoft Kinect v1 como *scanner* 3D, para obtenção da nuvem de ponto;
b) biblioteca PCL 1.6 para manipulação de nuvens de ponto;
c) Microsoft Visual Studio 2010 como ambiente de desenvolvimento.

3.3.2 Implementação da aplicação

Esta seção é responsável por detalhar de forma técnica as etapas apresentadas no diagrama de atividade na seção 3.2.3.

Na seção 3.3.2.1 é detalhado como o protótipo faz para capturar uma nuvem de pontos para leitura posterior. Na seção 3.3.2.2 é explicado como funciona a etapa de filtro que engloba a retirada do fundo e a diminuição da densidade da nuvem. A seção 3.3.2.3 apresentada explicações sobre a estimativa de normal. Por fim, a seção 3.3.2.4 explica como é utilizado o algoritmo de triangulação para fazer reconstrução tridimensional.

3.3.2.1 Capturando uma nova de nuvem de pontos

Para manipular o Kinect foi utilizado a interface `OpenNIGrabber` (Quadro 5 linha 3) que consegue obter a nuvem de ponto e registrar algumas funções para que durante a captura sejam realizadas outras ações, como no caso, receber a nuvem de pontos do Kinect (Quadro 5 linha 11 até 13).

A visualização da nuvem resgatada é feita utilizando `PCLVisualizer` (Quadro 5 linha 2), na qual é possível adicionar polígonos, normais, nuvens de pontos, linhas, histogramas, etc. Esta biblioteca também suporta a inserção de funções para capturar a tecla pressionada (Quadro 5 linha 13) ou o clique do mouse.

O arquivo é salvo com a extensão `PCD`, um arquivo binário que armazena a posição de pontos tridimensionais, para isto foi utilizada a classe `PCDWriter` (Quadro 5 linha 22 até 23) para que a nuvem seja salva nesta extensão.

Quadro 5 - Armazenando a nuvem de pontos

```
1  void PreviewPointCloud::save(std::string fileName) {
2      pcl::visualization::PCLVisualizer viewer("Salvar Nuvem de Pontos");
3      pcl::Grabber* kinect = new pcl::OpenNIGrabber();
4      savecloud = false;
5      boost::function< void(const pcl::visualization::KeyboardEvent&)> keyboardFunc =
6                                          boost::bind (&PreviewPointCloud::keyboard, this, _1);
7      configureCamera(viewer);
8      viewer.registerKeyboardCallback(keyboardFunc);
9
10     //adicionado listiner em que o kinect chama o visualizador
11     boost::function<void (const pcl::PointCloud<pcl::PointXYZ>::ConstPtr&)> viewerFunction =
12                                         boost::bind (&PreviewPointCloud::viewer_pcl, this, _1);
13     kinect->registerCallback(viewerFunction);
14     kinect->start();
15
16     while (!viewer.wasStopped()) {
17         if (global_cloud && m_mutex.try_lock()) {
18             if (!viewer.updatePointCloud(global_cloud)){
19                 viewer.addPointCloud(global_cloud);
20             }
21             if (savecloud) {
22                 pcl::PCDWriter writer;
23                 writer.writeBinary(fileName, *global_cloud);
24                 std::cout << "Salvo o arquivo " + fileName << std::endl;
25                 savecloud = false;
26             }
27         }
28         m_mutex.unlock ();
29         viewer.spinOnce(100);
30     }
31     kinect->stop ();
32     viewer.close();
33 }
```

3.3.2.2 Filtro

A primeira coisa a ser feita após a leitura do arquivo com a nuvem de pontos é a retirada do fundo. Esta etapa pode ser considerada uma segmentação, já que deixaremos apenas a área de interesse, o tórax.

Para a remoção do fundo foi utilizado o algoritmo `PassThrough` que verifica se as coordenada x, y ou z se encontram nos limiares passado por parâmetro. Para adicionar a coordenada é utilizado o método `setFilterFieldName` (Quadro 6 linha 3) e os limiares são informados no método `setFilterLimits` (Quadro 6 linha 4).

Os melhores limiares encontrados para a coordenada z foram 0.0 e 0.7 metros, estes valores podem ser utilizados em qualquer nuvem, já que um pré-requisito do protótipo é que se mantenha uma distância de 50 a 57 centímetros do objeto até o Kinect.

Quadro 6 - Filtrando a nuvem de pontos

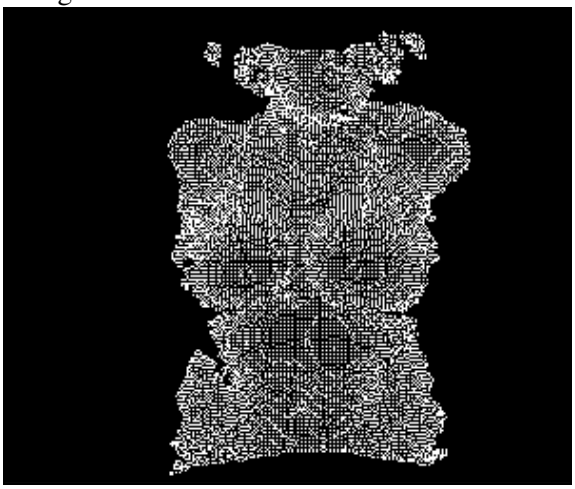
```
1       pcl::PassThrough<pcl::PointXYZ> pass;
2       pass.setInputCloud(cloud);
3       pass.setFilterFieldName(point);
4       pass.setFilterLimits(lMin, lMax);
5       pass.filter(*output);
```

Mesmo com a remoção do fundo, a nuvem continua densa. Fato que deixa oos algoritmos posteriores menos performáticos e eficazes. Para diminuir a densidade foi utilizado o algoritmo `VoxelGrid` que tem como objetivo dividir o espaço em cubos de tamanho fixo, e os pontos que ficam dentro deste cubo se transformam em apenas um ponto. O valor dos cubos é 0.005 metros (Quadro 7 linha 3), para que não existam espaços largos entre um ponto e outro. O resultado obtido pelo filtro pode ser observado na Figura 29.

Quadro 7 - Aplicando o filtro `VoxelGrid`

```
1       pcl::VoxelGrid<sensor_msgs::PointCloud2> vg;
2       vg.setInputCloud(input);
3       vg.setLeafSize(0.005f, 0.005f, 0.005f);
4       vg.filter(*output);
```

Figura 29 - Resultado do filtro `VoxelGrid`



3.3.2.3 Normal

A normal é um passo importante para que a triangulação seja bem feita, pois ela indica a direção dos polígonos. O algoritmo utilizado é o `MovingLeastSquares`, que conforme os testes obteve os melhores resultados.

Foram utilizados dois métodos para melhorar o resultado da normal, o método `setPolynomialFit` com o valor `true` (Quadro 8 linha 9) que aproxima a normal da superfícies utilizando polígonos, com isto o algoritmo fica mais lento, mas com eficácia maior. Outro método utilizado foi o `setSearchRadius` com o valor de 0.02 (Quadro 8 linha

11). Este método busca os vizinhos mais próximos em um raio de 0.02. Quanto maior o valor, menos definição terão as normais.

O resultado do algoritmo `MoveLeastSquares` são normais que não possuem espaços quando as normais mudam de direção. Com isto, é possível ver de forma nítida a silhueta do objeto captura, no caso, uma mulher (Figura 30).

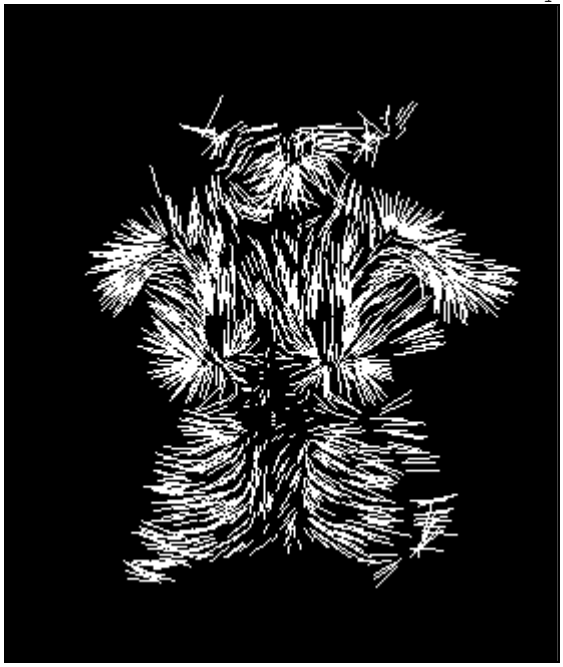
Quadro 8 - Código utilizado para executar a `MoveLeastSquares`

```
pcl::PointCloud<pcl::PointNormal> normalMoveLeastSquares
                    (pcl::PointCloud<pcl::PointXYZ>::Ptr cloud, pcl::PointCloud<pcl::PointNormal> normal) {
    double ti, tf;
    pcl::MovingLeastSquares<pcl::PointXYZ, pcl::PointNormal> normalEstimation;
    pcl::search::KdTree<pcl::PointXYZ>::Ptr searchToNormal (new pcl::search::KdTree<pcl::PointXYZ>);

    normalEstimation.setComputeNormals(true);
    normalEstimation.setInputCloud(cloud);
    normalEstimation.setPolynomialFit(true);
    normalEstimation.setSearchMethod(searchToNormal);
    normalEstimation.setSearchRadius(0.02);

    normalEstimation.process(normal);
    return normal;
}
```

Figura 30 - Resultado da normal com `MoveLeastSquares`



3.3.2.4 Reconstrução da superfície

Depois de estabelecer a estimativa de normal é necessário relacionar a normal a um ponto. Para isto, é utilizado o método `concatenateFields` que adiciona o valor da normal ao campo `point[i].curvature`.

Para a reconstrução da superfície foi utilizada a classe `GreedyProjectionTriangulation` que executa o algoritmo de triangulação ganancioso. Ele mantém uma lista de pontos e a partir destes pontos ele começa a conectá-los em forma de triângulos até que o último ponto seja conectado.

Para fazer a triangulação, o algoritmo depende nos valores passados por parâmetro de alguns métodos principais como:

a) `setSearchRadius`: define a distância máxima do vértice e, como a distância entre um ponto e outro é pequena, o valor ideal para o caso deste trabalho foi de 0.025 (Quadro 9 linha 10);

b) `setMu`: multiplicador da distância do vizinho mais próximo. Serve para obter o raio final de busca do vizinho mais próximo, sendo que o valor que mais se ajustou ao caso deste trabalho foi de 2.5 (Quadro 9 linha 11);

c) `setMaximumNearestNeighbors`: máximo de vizinhos considerados na busca. O valor definido foi de 190, pois com mais vizinhos o algoritmo não achava mais pontos em relação ao raio de busca (Quadro 9 linha 12);

d) `setMaximumSurfaceAngle`: se o desvio da normal for maior que o ângulo definido a normal não é considerada. O valor definido foi de π/4 (Quadro 9 linha 13);

e) `setMinimumAngle`: define o ângulo mínimo do triângulo (Quadro 9 linha 14);

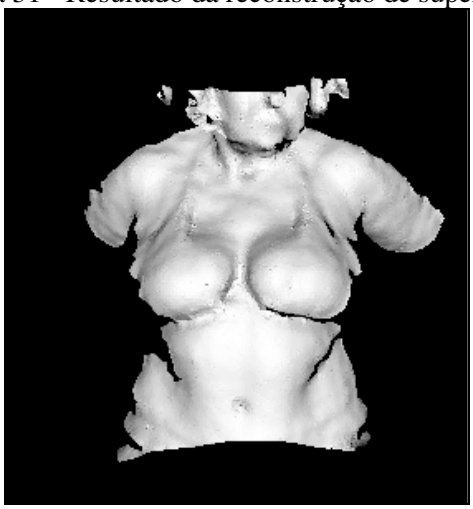
f) `setMaximumAngle`: define o ângulo máximo de cada triangulo (Quadro 9 linha 15).

Quadro 9 - Chamada do algoritmo de reconstrução de superfícies

```
1  pcl::PolygonMesh Triangulation::reconstructionWithTriangulation(pcl::PointCloud<pcl::PointNormal>::Ptr input)
2  {
3      pcl::search::KdTree<pcl::PointNormal>::Ptr searchToReconstruction
4                                                  (new pcl::search::KdTree<pcl::PointNormal>);
5      pcl::GreedyProjectionTriangulation<pcl::PointNormal> greedyTriangulation;
6      pcl::PolygonMesh triangulation;
7
8      searchToReconstruction->setInputCloud(input);
9
10     greedyTriangulation.setSearchRadius(0.025);
11     greedyTriangulation.setMu(2.5);
12     greedyTriangulation.setMaximumNearestNeighbors(190);
13     greedyTriangulation.setMaximumSurfaceAngle(M_PI/4);
14     greedyTriangulation.setMinimumAngle(M_PI/18);
15     greedyTriangulation.setMaximumAngle(3*M_PI/3);
16     greedyTriangulation.setNormalConsistency(false);
17     greedyTriangulation.setInputCloud(input);
18     greedyTriangulation.setSearchMethod(searchToReconstruction);
19     greedyTriangulation.reconstruct(triangulation);
20
21     return triangulation;
22 }
```

Após executar todos os passos, inclusive a triangulação, chega-se a reconstrução 3D, conforme pode ser visto na Figura 31.

Figura 31 - Resultado da reconstrução de superfícies



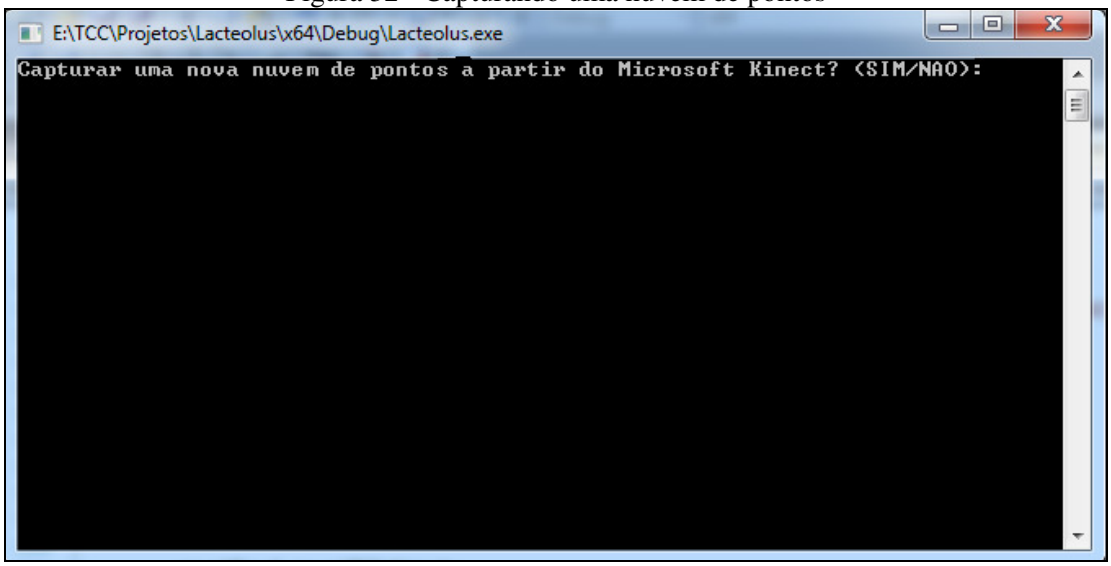
3.4 OPERACIONALIDADE

Nesta seção é apresentada a operacionalidade do protótipo, ou seja, como o usuário deve interagir com a aplicação.

A *interface* com o usuário ficou pouco intuitiva porque não faz parte do escopo deste trabalho. O usuário interage somente no início e não pode encerrar a aplicação assim que ela for iniciada, somente finalizando o processo. Outro requisito para o protótipo funcionar é ter o *driver* do OpenNi instalado.

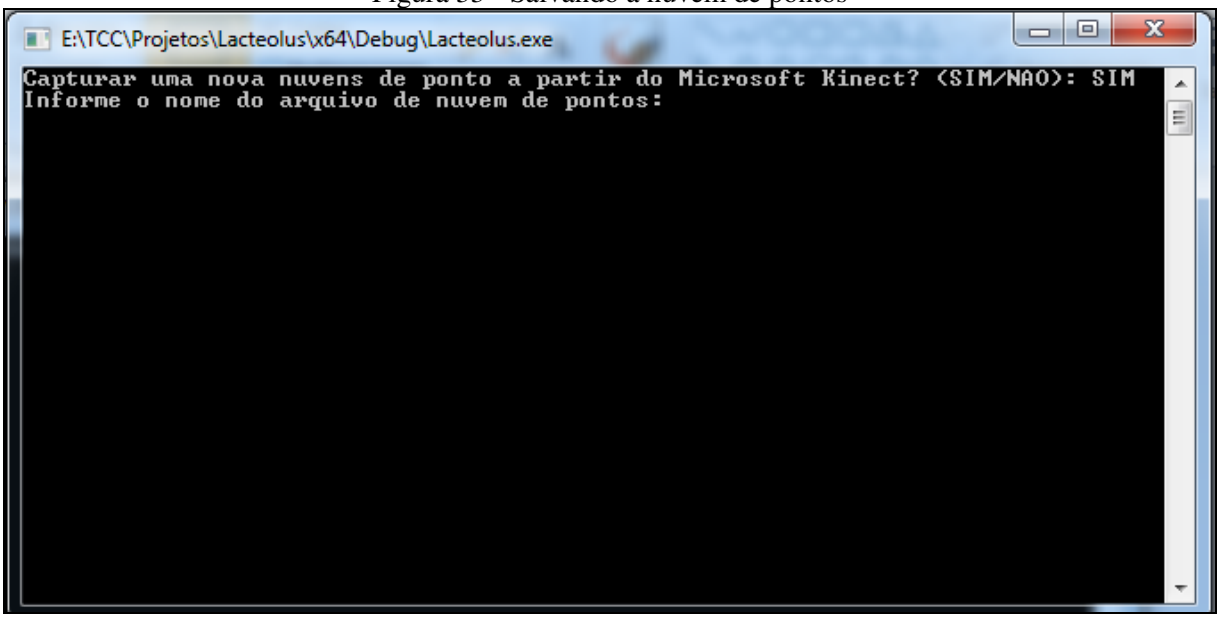
Quando o protótipo é iniciado aparece uma tela (Figura 32) perguntando se o usuário deseja capturar uma nova nuvem de pontos a partir do Microsoft Kinect.

Figura 32 - Capturando uma nuvem de pontos



Se o usuário informar `SIM` então nesta mesma tela aparecerá uma nova linha pedindo o caminho e o nome do arquivo .pcd (Figura 33).

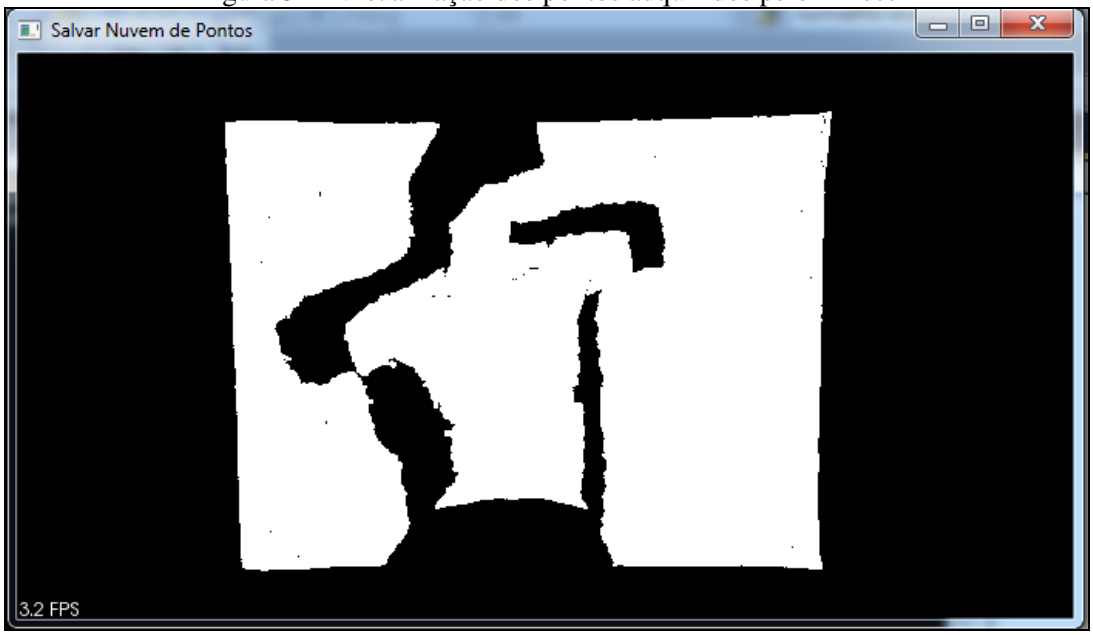
Figura 33 - Salvando a nuvem de pontos



Ao adicionar as informações requeridas, o protótipo iniciará a comunicação com o Microsoft Kinect, mostrando em uma nova tela (Figura 34) a nuvem de pontos adquirida.

Para salvar uma amostra de qualidade é necessário posicionar o objeto de 25 a 30 centímetros da parede e de 50 a 57 centímetros do sensor. Com o objeto na posição desejada o usuário deve pressionar a tecla "`s`" para salvar a nuvem de pontos, após o salvamento o processo de reconstrução é iniciado.

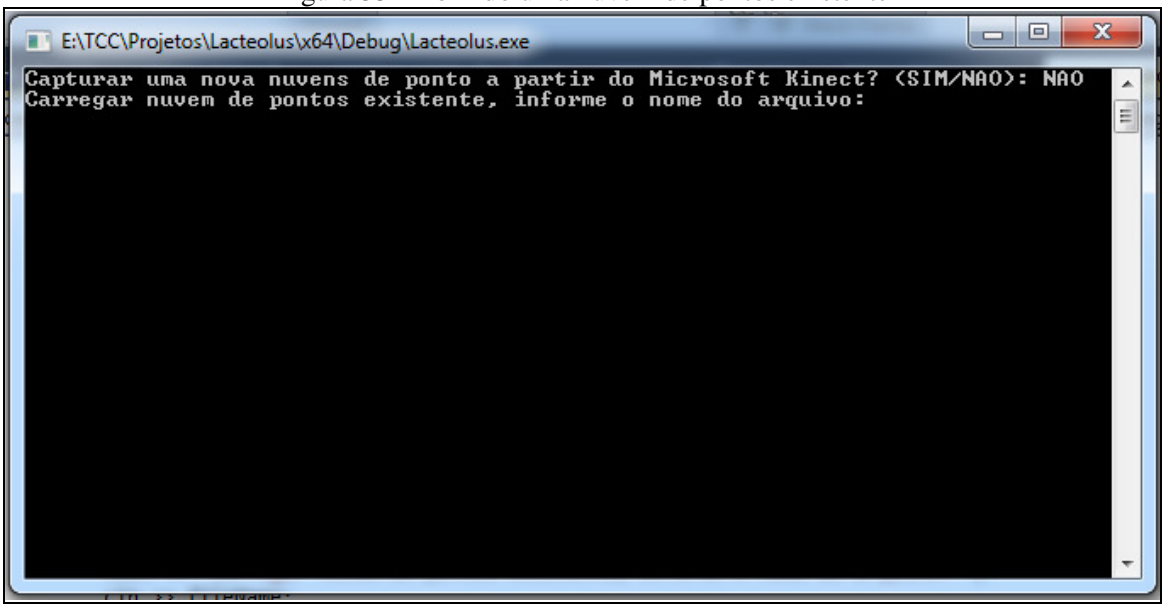
Figura 34 - Visualização dos pontos adquiridos pelo Kinect



Em alguns casos o usuário já pode ter salvo a nuvem de pontos e deseja apenas reconstruí-la. Neste caso, o usuário deve informar `NAO` na tela inicial vista na Figura 32. Após

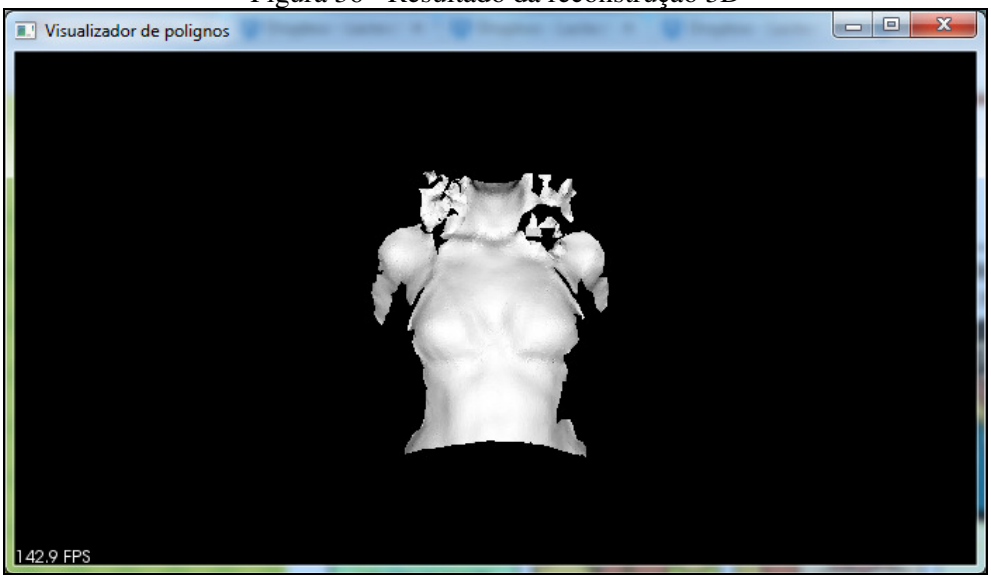
isto, uma nova linha será adicionada perguntando o caminho e nome da nuvem de pontos já existente (Figura 35).

Figura 35 - Abrindo uma nuvem de pontos existente



Assim que o processo de reconstrução for iniciado o usuário perde o controle da aplicação e só recobrará quando uma tela com a reconstrução 3D aparecer, conforme mostra a Figura 36.

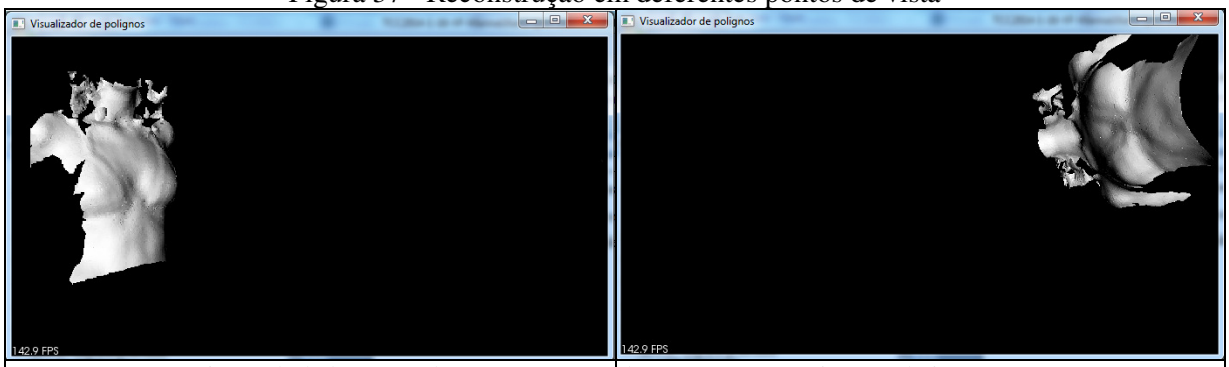
Figura 36 - Resultado da reconstrução 3D



Também é possível rotacionar a imagem reconstruída para que se consiga visualizar todos os lados da reconstrução como na Figura 37 item a.

Ao rotacionar a imagem, se consegue deixar a profundidade (coordenada z) evidente. Na Figura 37 item b é visualizado o volume do seio, esta visualização só é possível, pois é uma reconstrução é em 3D.

Figura 37 - Reconstrução em deferentes pontos de vista



(a) Reconstrução vista pelo lado esquerdo | (b) Reconstrução vista por baixo

3.5 RESULTADOS E DISCUSSÕES

Nesta seção serão apresentados os testes feitos em cada etapa da reconstrução 3D. Será discutido o motivo pelo qual foi utilizado um determinado algoritmo ao invés do outro e porque alguns passos foram ignorados/excluídos da reconstrução final.

Na seção 3.5.1 serão demonstrados como se chegou ao pré-requisito na hora de adquirir nuvens de pontos. Na seção 3.5.2 aborda os testes entre os algoritmos de filtros. Na seção 3.5.3 são realizados os testes entre os algoritmos de estimativa de normal.

A seção 3.5.4 realiza testes com a triangulação para a reconstrução 3D final. Na seção 3.5.5 é demonstrado como foi implementada a segmentação por região de interesse e porque foi removida do protótipo. A seção 3.5.6 são demonstrados alguns testes com a nuvem de ponto colorida e porque não foi possível utilizá-la.

A última seção (3.5.7) apresenta comparações entre o protótipo desenvolvido e os trabalhos correlatos.

3.5.1 Aquisição da nuvem de pontos

Com o Microsoft Kinect é possível obter a nuvem de pontos da cena capturada, da mesma forma que um *scanner* 3D, sendo a forma de aquisição escolhida neste trabalho.

Ao adquirir a nuvem de pontos pelo Kinect a nuvem sempre é densa (muitos pontos) e, dependendo alguns ruídos e buracos podem ser gerados. Isto depende de fatores externos como a posição, distância, iluminação e reflexo. Por este motivo foram feitos testes para analisar qual seria o melhor cenário para aquisição da nuvem de pontos.

Para executar os testes foram utilizados duas amostras do tórax, a primeira é um manequim (Figura 38 item a) e a segunda uma mulher (Figura 38 item b). Com estas amostras foram feitos testes com três distâncias diferentes para ver qual delas se aproxima do ideal.

Figura 38 - Foto do manequim e da mulher respectivamente

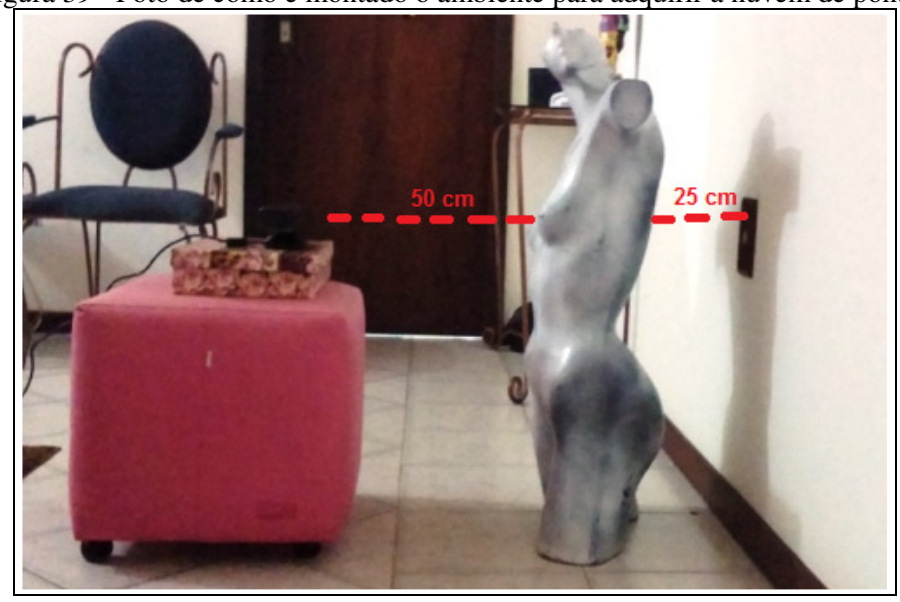
(a) Foto do tórax de um manequim (b) Foto do tórax de uma mulher

Na

Figura 39 é possível perceber como o ambiente foi montado para capturar da nuvem de pontos. O Kinect deve estar na altura dos seios para que o ponto de vista da cena seja de alguém olhando com a cabeça reta para uma direção fixa.

Para controlar a região do corpo que será incluída na nuvem de pontos é necessário variar a distância do sensor, já que dependendo da distância, a amostra pode vir com falhas ou mais informações do que o necessário.

Figura 39 - Foto de como é montado o ambiente para adquirir a nuvem de pontos



Inicialmente foi definida a distância que o tórax deve ficar da parede. Isto se fez necessário para que o protótipo tenha um ambiente controlado e, com isto, a segurança de utilizar valores fixos nos algoritmos, extinguindo as preocupações com ambientes variados.

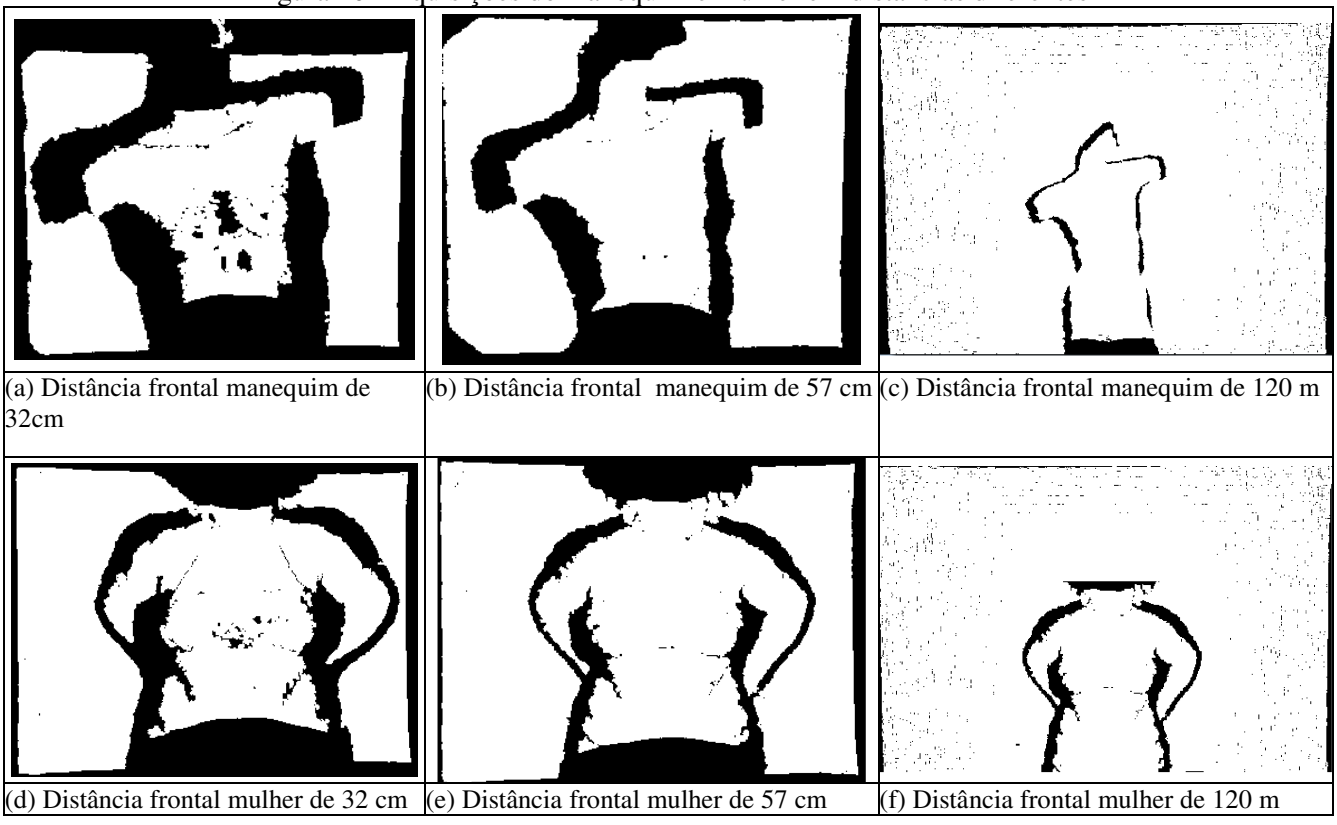
O limite definido foi de 25 a 30 (Figura 39) centímetros do tórax até a parede, pois desta forma se cria uma sombra ao redor do tórax, delimitando bem a região de interesse. Este valor é válido somente para quando o limite de distância frontal é inferior a 120 centímetros.

Com o limite entre a parede e o tórax já definido, foram feitos testes com três distâncias entre o tórax e o dispositivo. No primeiro teste foi posicionado o Kinect a 30 centímetros (Figura 40 item a e d) do tórax tanto do manequim quanto da mulher e em ambos os casos veio somente o tórax, mas as malhas continham buracos na região torácica e ruídos no fundo da nuvem.

Na segunda aquisição foi definido o valor da distância de 57 centímetros (Figura 40 item b e e) para ambas as amostras. A nuvem de pontos adquirida continha mais informações do que o necessário, como a cabeça e a região um pouco mais abaixo da cintura, mas a qualidade da nuvem melhorou, ou seja, não continha buracos ou ruídos.

No último teste usou-se o valor de 120 centímetros (Figura 40 item c e f). Com esta distância, a maior parte do corpo do manequim e da mulher foi adicionada a nuvem de ponto, alguns buracos aparecem no fundo, mas nada que comprometesse a região torácica.

Figura 40 - Aquisições do manequim e mulher em distâncias diferentes



(a) Distância frontal manequim de 32cm	(b) Distância frontal manequim de 57 cm	(c) Distância frontal manequim de 120 m
(d) Distância frontal mulher de 32 cm	(e) Distância frontal mulher de 57 cm	(f) Distância frontal mulher de 120 m

Baseando-se nos testes com a distância frontal, foi concluído que a melhor distância é a de 50 centímetros. Valores entre 30 e 50 centímetros foram descartados, pois existem variações de resultados entre as amostras.

Foram realizadas também testes com diferentes iluminações: iluminando as laterais do objeto e iluminando a parte frontal do objeto. Foi constatado que a iluminação não afeta a aquisição da nuvem de ponto, pois os resultados foram sempre os mesmos.

3.5.2 Filtros

Esta seção foi dividida pelas amostras capturadas na seção anterior e, para cada uma das amostras capturadas serão apresentados os resultados alcançados nos quatro algoritmos de filtro (`StatisticalOutlierRemoval`, `RadiusOutlierRemoval` , `PassThrough` e `VoxelGrid`) . O objetivo de todos os algoritmos testados é diminuir a quantidade de pontos e remover os possíveis ruídos gerados pelo *scanner* 3D.

Na seção 3.5.1.1 serão apresentados os resultados obtidos na nuvem de pontos de um manequim (amostra 1). A seção 3.5.1.2 apresenta os resultados obtidos na nuvem de pontos de uma mulher (amostra 2) e por último, na seção 3.5.1.3 serão discutidos os resultados dos algoritmos com base nos testes realizados.

3.5.2.1 Amostra 1: Manequim

Nesta seção são demonstrados os experimentos realizados em duas amostras diferentes do mesmo manequim. O primeiro teste foi realizado na amostra vista na Figura 38 item a (seção 3.5.1) e para o segundo teste foi utilizado a amostra vista na Figura 38 item b (seção 3.5.1).

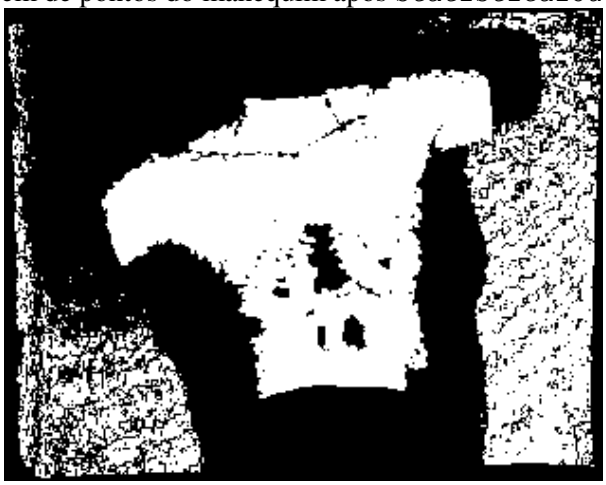
3.5.2.1.1 Nuvem de pontos do manequim com alta densidade e buracos

O primeiro algoritmo utilizado para filtro foi o `StatisticalOutlierRemoval`, sendo inseridos os seguintes valores:

a) `setMeanK`: a busca será aplicada considerando apenas 10 vizinhos. Com um valor maior o processamento demoraria mais e o resultado seria basicamente o mesmo;
b) `setStddevMulThresh`: foi informado valor de 0.5, pois desta forma o valor da média e desvio é divido pela metade, aumentando a quantidade de pontos excluídos.

A execução demorou 7 segundos e o resultado obtido pode ser visto na Figura 41.

Figura 41 - Nuvem de pontos do manequim após `StatisticalOutlierRemoval`



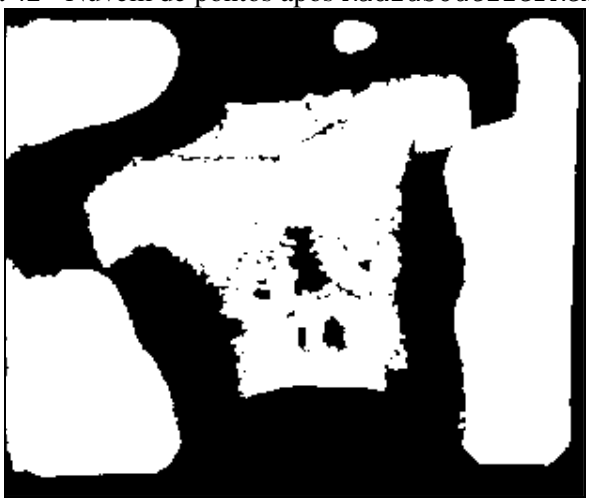
Comparando o resultado da Figura 41 com a da Figura 40 item b é possível ver que ocorreram mudanças na densidade, mas apenas no fundo da amostra. Entretanto, alguns ruídos foram criados após a aplicação do algoritmo.

Ainda utilizando a amostra da Figura 40 item b foi utilizado o algoritmo `RadiusOutlierRemoval`, com o mesmo objetivo. Para utilizar este algoritmo foram utilizados os seguintes métodos e os respectivos parâmetros:

a) `setMinNeighborsInRadius`: foi usado o valor de 2000 vizinhos. Como a nuvem é densa, o fato de ter menos vizinhos aumenta possibilidade de ser um ruído;
b) `setRadiusSearch`: foi utilizado o valor 0.09, pois num espaço pequeno com vários vizinhos é um indicativo que aquele ponto não é um ruído;

O resultado do filtro pode ser observado na Figura 42, onde se percebe que o resultado não foi satisfatório, ele só removeu os pontos das extremidades. O algoritmo removeu alguns ruídos encontrados na nuvem, mas em contrapartida a nuvem continuou densa e o tempo de execução foi de 12 minutos. Caso mudasse o valor de `setMinNeighborsInRadius` para 1000, o tempo de execução e o resultado seriam os mesmos.

Figura 42 - Nuvem de pontos após `RadiusOutlierRemoval`



O terceiro algoritmo é o `PassThrough`. Ao qual, foram adicionados os seguintes valores como parâmetro:

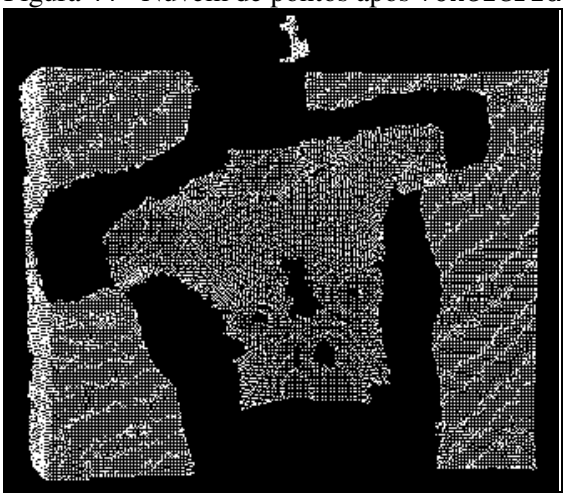
a) `setFilterFieldName`: foi informada a coordenada z, visto que se pretende remover possíveis pontos com profundidade não condizente com o resto do objeto;
b) `setFilterLimits`: o limiar testado foi de 0.0 e 0.5. Com estes valores, a parede é removida do fundo. Se for alterado este limiar o tórax será afetado negativamente.

O resultado apresentado na Figura 43 não era o esperado. É possível ver que a densidade da nuvem não diminui e, em contrapartida, removeu a maior parte do fundo, ou seja, segmentou o objeto de interesse deste trabalho, o tórax.

Figura 43 - Nuvem de pontos após `PassThrough`

O último algoritmo testado foi o `VoxelGrid`. No parâmetro que atribui o tamanho do *voxel*, `setLeafSize`, foram incluídos os seguintes valores 0.05, 0.05 e 0.05. Com estes valores, a nuvem ficou menos densa, conforme visto na Figura 44.

Figura 44 - Nuvem de pontos após `VoxelGrid`



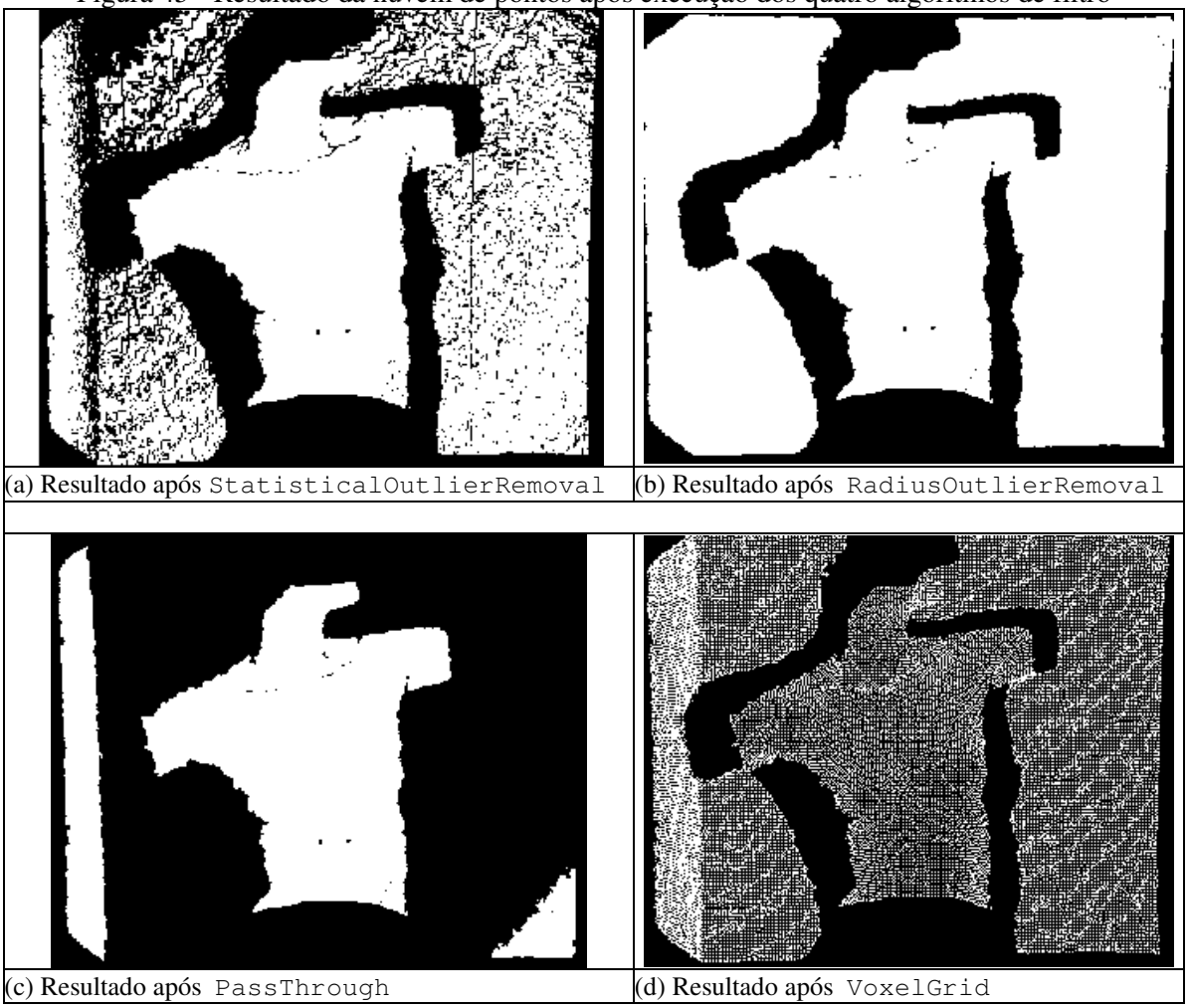
3.5.2.1.2 Nuvem de pontos do manequim com alta densidade e sem buracos

Nesta seção serão executados os mesmos algoritmos da seção 3.5.2.1.1 (`StatisticalOutlierRemoval`, `RadiusOutlierRemoval`, `PassThroug` e `VoxelGrid`), porém aplicados em uma amostra sem buracos, mas com alta densidade. Os valores de entrada desses algoritmos não sofreram alterações, já que a amostra é semelhante. O valor de entrada do método `setFilterLimits` (`PassThroug`) foi alterado para 0.0 e 0.8. O resultado desta execução pode ser visto na Figura 45 item c.

Ao executar o algoritmo `RadiusOutlierRemoval` com os mesmos valores. Ele demorou 10 minutos a mais em relação ao tempo alcançado na seção 3.5.2.1.1, pois a quantidade de pontos era maior. O resultado pode ser visto na Figura 45 item b.

Todos os algoritmos tiveram os mesmos resultados que a seção 3.5.2.1.1. O algoritmo `StatisticalOutlierRemoval` visto na Figura 45 item a, teve alguns pontos do fundo removido. No algoritmo `RadiusOutlierRemoval` apenas os pontos das extremidades foram removidos, conforme visto na Figura 45 item b. O algoritmo de `PassThrough` removeu apenas o fundo da nuvem (Figura 45 item c). Por fim, o algoritmo de `VoxelGrid` apenas diminuiu a densidade da nuvem (Figura 45 item d).

Figura 45 - Resultado da nuvem de pontos após execução dos quatro algoritmos de filtro



(a) Resultado após `StatisticalOutlierRemoval` (b) Resultado após `RadiusOutlierRemoval`

(c) Resultado após `PassThrough` (d) Resultado após `VoxelGrid`

3.5.2.2 Amostra 2: Mulher

Neste teste, foram executados novamente os quatro algoritmos de filtro. Porém, agora em duas amostras diferentes da mesma mulher. Na seção 3.5.2.2.1 são demonstrados os testes feitos em uma amostra de alta densidade e com buracos, conforme visto na Figura 40 item d (seção 3.5.1) e na seção 3.5.2.2.2 são apresentados os resultados obtidos em uma amostra de alta densidade e sem buracos.

3.5.2.2.1 Nuvem de pontos de uma amostra mulher com alta densidade e buracos

O primeiro algoritmo executado foi o `StatisticalOutlierRemoval`, tendo como parâmetro os seguintes valores:

a) `setMeanK`: a busca será aplicada considerando apenas 10 vizinhos. Com um valor maior o processamento demoraria mais e o resultado seria basicamente o mesmo;

b) `setStddevMulThresh`: foi informado valor 0.5, pois desta forma o valor da média e desvio é divido pela metade, ao qual aumentaria a quantidade de pontos

excluídos.

Com os valores acima, os resultados obtidos (Figura 46) foram semelhantes as amostras do manequim, ou seja, partes do fundo foram removidos e a densidade da nuvem diminuiu apenas nos braços. Nota-se que o algoritmo também gerou ruídos ao remover o fundo. Em comparação com a nuvem de pontos do manequim, esta amostra apresentou mais buracos na região do tórax e braços.

Figura 46 - Nuvem de pontos da mulher após `StatisticalOutlierRemoval`

No algoritmo `RadiusOutlierRemoval` foram utilizados os parâmetros `setRadiusSearch` com o valor de 0.05 e `setMinNeighborsInRadius` com o valor de 2000 vizinhos. Estes valores foram escolhidos por causa da densidade de pontos na nuvem, cujo um dos principais objetivos é a redução de pontos. Para isto, foi fixado um valor de vizinhos para que os pontos que não se encaixassem nestes valores fossem removidos.

A execução do `RadiusOutlierRemoval` demorou 6 minutos. O resultado pode ser observado na Figura 47. O algoritmo se comporta como fosse corroendo a nuvem de ponto, iniciando pelas extremidades até atingir o meio da nuvem.

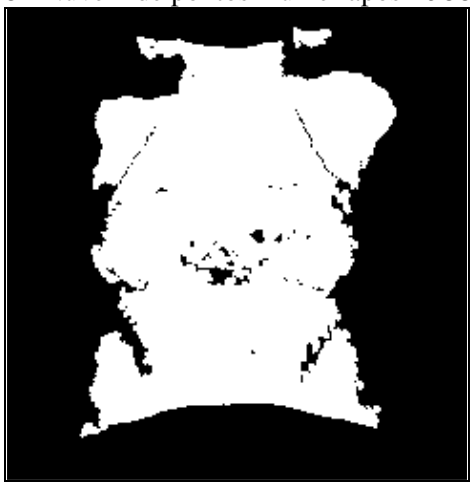
Figura 47 - Nuvem de pontos da mulher após `RadiusOutlierRemoval`

Como terceiro algoritmo foi usado o `PassThroug`, ao qual, foram passados os seguintes parâmetros:

a) `setFilterFieldName`: a coordenada informada é a "z", já que se pretende remover possíveis pontos com a profundidade não condizente com o resto do objeto;
b) `setFilterLimits`: o limiar testado foi de 0.0 e 0.6. Estes valores são em metros, ou seja, pontos que estão a uma profundidade de 60cm serão removidos.

Na Figura 48 pode ser observado que o algoritmo removeu somente o fundo e os braços, pois estavam posicionados mais atrás. Este algoritmo também removeu alguns ruídos que estavam mais próximos ao fundo, porém não diminuiu a densidade da nuvem.

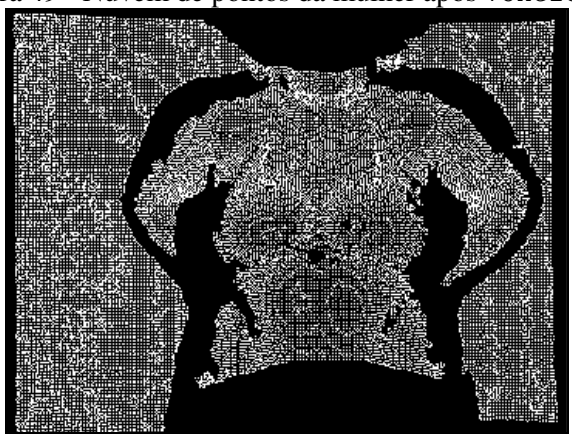
Figura 48 - Nuvem de pontos mulher após `PassThroug`



O último algoritmo utilizado foi o `VoxelGrid` e os valores informados no `setLeaftSize` foram de 0.005, 0.005 e 0.005. Estes valores servem para que os *voxels* possam diminuir a densidades da nuvem.

Na Figura 49 é possível ver que a densidade da nuvem foi reduzida.

Figura 49 - Nuvem de pontos da mulher após `VoxelGrid`

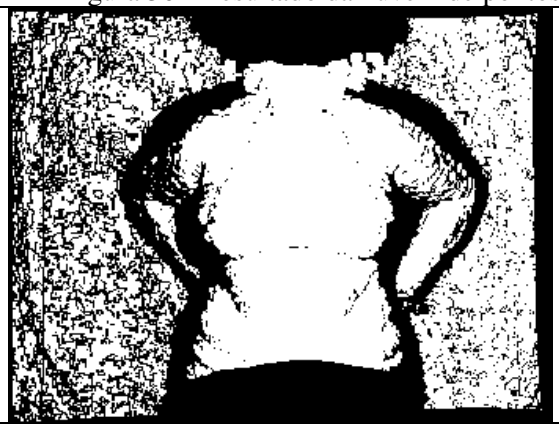


3.5.2.2.2 Nuvem de pontos de uma mulher com alta densidade e sem buracos

Os resultados obtidos nesta nuvem de pontos são praticamente os mesmos da seção 3.5.2.2.1, porém, foi necessário alterar os parâmetros do algoritmo `PassThrough` para que ele se adaptasse a distância da amostra vista na Figura 40 item e. No método `setFilterLimits` foi alterado o valor de 0.6 para 0.8, para que os valores não afetassem a área de interesse.

Os resultados obtidos a partir da nuvem da Figura 40 item e podem ser vistos na Figura 50. O algoritmo `StatisticalOutlierRemoval` está Figura 50 item a, o algoritmo `RadiusOutlierRemoval` na Figura 50 item b, o algoritmo `PassThrough` na Figura 50 item c e o algoritmo `VoxelGrid` na Figura 50 item d.

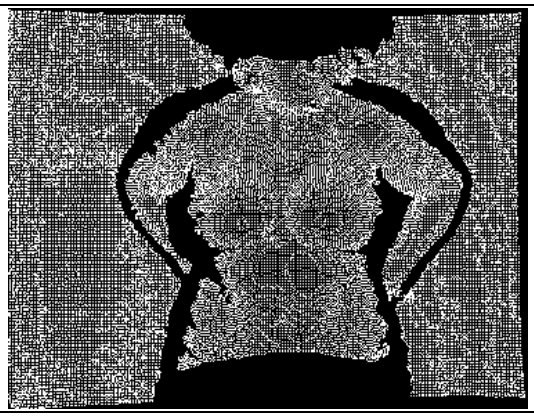
Figura 50 - Resultado da nuvem de pontos após execução dos quatro algoritmos de filtro



(a) Resultado após `StatisticalOutlierRemoval`

(b) Resultado após `RadiusOutlierRemoval`

(c) Resultado após `PassThrough`



(d) Resultado após `VoxelGrid`

3.5.2.3 Discussões dos resultados

Levando em consideração os objetivos destes algoritmos (remover ruídos e diminuir a densidade) é possível tirar algumas conclusões, tais como:

a) o algoritmo `RadiusOutliersRemoval` não se mostrou viável para todas as nuvens de teste. Em nenhum caso, ele executou em tempo hábil. No manequim ele demorou 12 minutos e, na amostra da mulher ele demorou 6 minutos. Ele também não removeu os ruídos e nem diminui a densidade da nuvem de pontos;
b) o algoritmo de `StatisticOutliersRemoval` reduziu a quantidade de ruídos e densidade da nuvem, mas apenas na região do fundo;
c) o algoritmo `PassThrough` reduziu por completo o fundo da nuvem, por consequência os ruídos que existiam neste local, entretanto, a região do tórax ainda continua densa. Com o resultado deste algoritmo surgiu a idéia de fixar a distância do objeto até o sensor e, executar o `PassThrough` para segmentar o tórax;
d) o algoritmo `VoxelGrid`, não reduziu os ruídos, mas diminuiu a densidade da nuvem de forma que os detalhes não fossem perdidos. Este foi o algoritmo

escolhido para a reconstrução tridimensional, mas como ele não é eficiente em remover os ruídos, foi executado antes o `PassThrough` para remover os ruídos e o fundo.

3.5.3 Normal

Nesta seção serão apresentados dois algoritmos de estimativa de normal utilizado no protótipo, `NormalEstimation` (seção 3.5.3.1) e `MovingLeastSquares` (seção 3.5.3.2). O objetivo desta seção é comparar o tempo de execução e os resultados das estimativas de normais com cada algoritmo executado na seção 3.5.2.

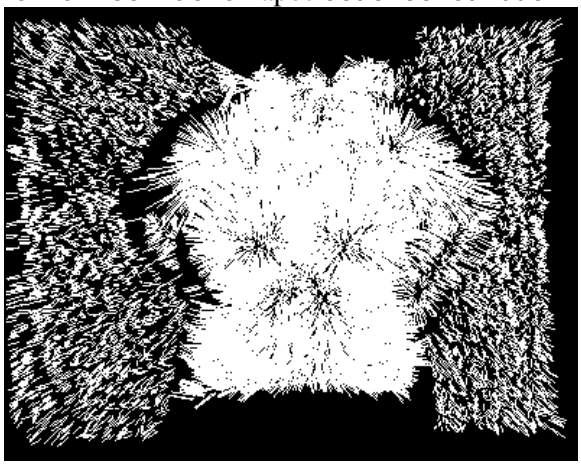
A amostra utilizada nos testes foi a nuvem de pontos sem buracos de uma mulher (Figura 40 item e). Não existe a necessidade de utilizar outras amostras, pois conforme a seção 3.5.2 os resultados pouco se diferem.

3.5.3.1 NormalEstimation

O algoritmo utilizado nos testes foi o de `NormalEstimation` que estima as normais baseado na matriz de covariância e na consulta de vizinhos mais próximos.

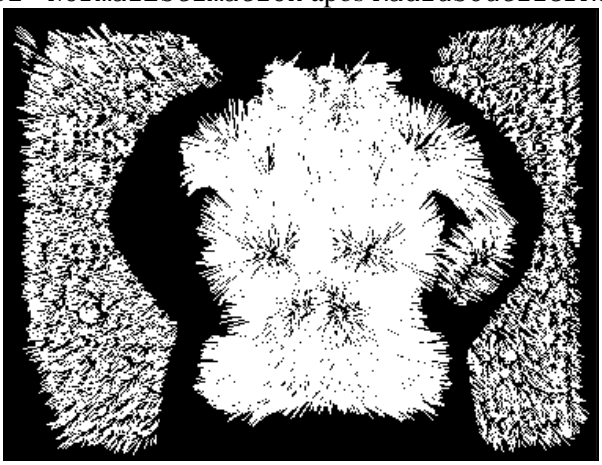
No primeiro teste foi executado o algoritmo de `NormalEstiamtion` após fazer a remoção de ruído utilizando método `StatisticalOutlierRemoval`. A execução demorou 3 minutos e, como pode ser observado na Figura 51, a silhueta da mulher não está definida, existem espaços quando a normal muda de direção.

Figura 51 - `NormalEstimation` após `StatisticalOutlierRemoval`



No segundo teste foi executado o algoritmo de `NormalEstimation` após fazer a remoção de ruído utilizando o método `RadiusOutlierRemoval`. A `NormaEstimation` executou em 9 segundos. O resultado pode ser visto na Figura 52.

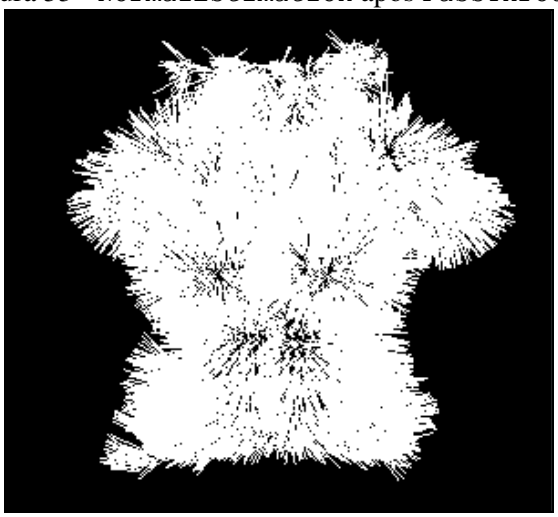
Figura 52 - `NormalEstimation` após `RadiusOutlierRemoval`



Analisando o resultado é possível notar que a silhueta da mulher não está definida e que existem espaços quando as normais mudam de direção.

No terceiro teste foi executado o algoritmo de estimativa de normal `NormalEstimation` após realizar a remoção de ruído através do método `PassThrough`. O tempo de execução da `NormalEstimation` foi de 1 minuto. O resultado é apresentado na Figura 53.

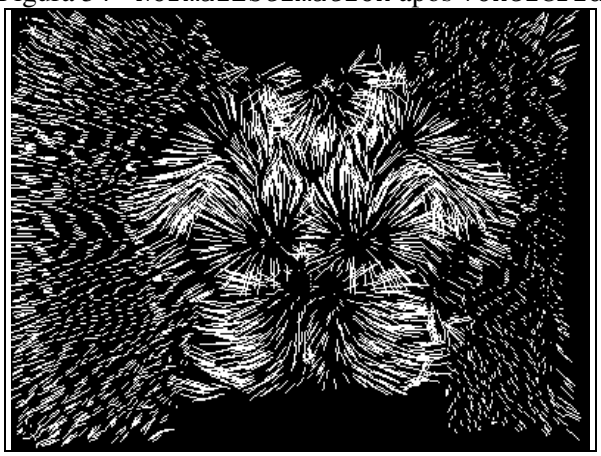
Figura 53 - `NormalEstimation` após `PassThrough`



A partir do resultado apresentado é possível observar que, assim como em testes anteriores, a silhueta não está definida e existem espaços quando as normais mudam de direções.

O algoritmo de estimativa de normal `NormalEstimation` foi executado após fazer a remoção de ruído utilizando `VoxelGrid`. A estimativa de normal demorou 9 segundos. O resultado se encontra na Figura 54.

Figura 54 - `NormalEstimation` após `VoxelGrid`



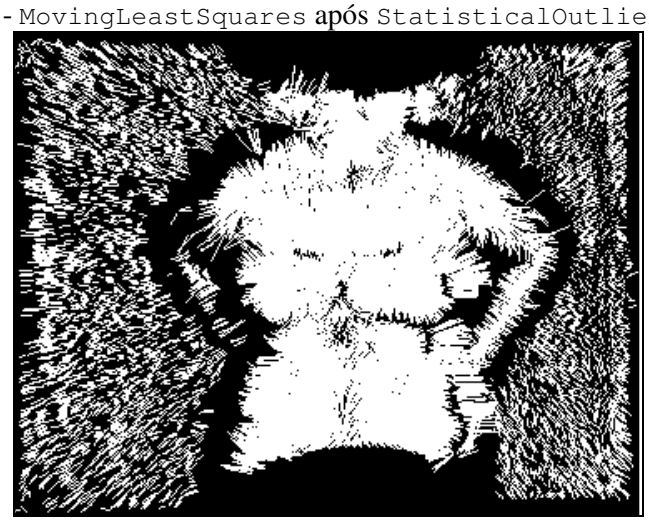
Analisando o resultado, é possível afirmar que a silhueta da mulher está indefinida e existem espaços quando as normais mudam de direção.

3.5.3.2 MovingLeastSquares

O algoritmo `MovingLeastSquares` estima as normais calculando a média ponderada dos mínimos quadrados[3]. Este algoritmo também suaviza a nuvem de pontos, preenchendo possíveis buracos e removendo os ruídos.

No primeiro teste foi executado o algoritmo `MovingLeastSquares` após realizar a o filtro `StatisticalOutliersRemoval`. A execução da estimativa de normal demorou 25 minutos e o resultado está na Figura 55.

Figura 55 - `MovingLeastSquares` após `StatisticalOutlierRemoval`

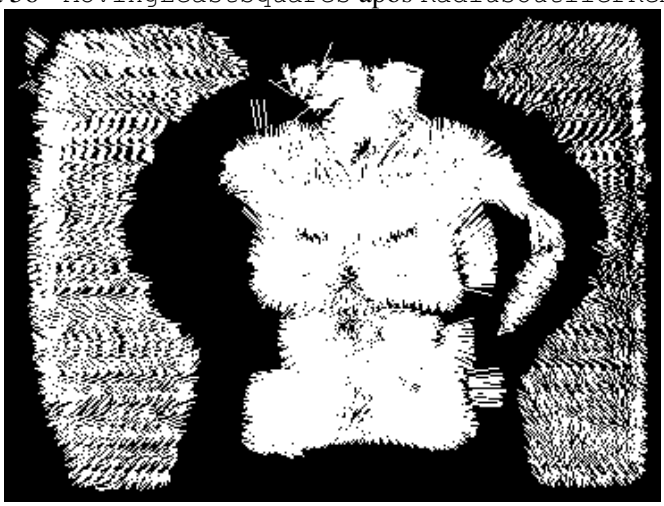


A partir do resultado apresentado é possível ver que a silhueta está definida e não existem espaços vazios quando as normais mudam de direção.

[3] Segundo Least Squares (2014) um mínimo quadrado pode ser definido como, "conjuntos de equações em que há mais equações do que incógnitas".

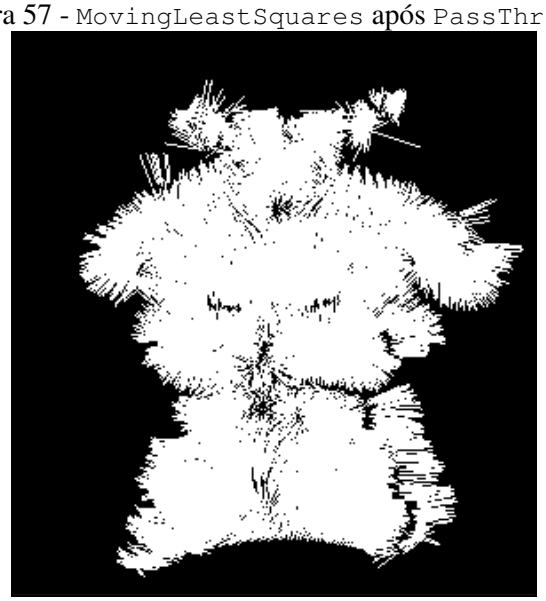
No segundo teste, o algoritmo `MovingLeastSquares` foi executado antes do filtro `RadiusOutlierRemoval`. A execução do algoritmo `MovingLeastSquares` demorou 29 minutos e, como no teste anterior, a silhueta é evidente e também não existem espaços quando as normais trocam de direção.

Figura 56 - `MovingLeastSquares` após `RadiusOutlierRemoval`



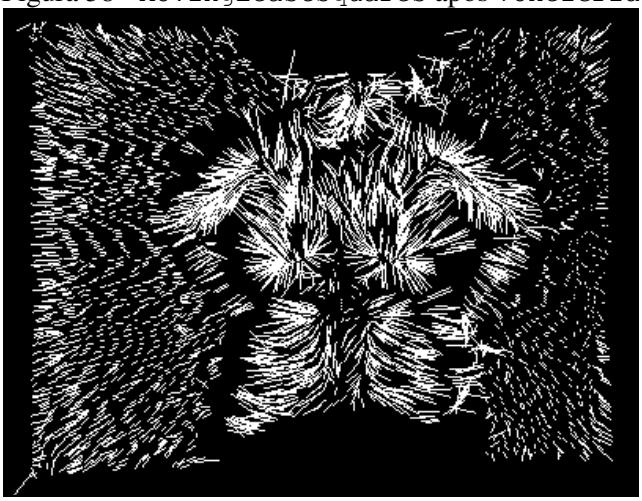
No terceiro teste foi executado o `PassThrough` e depois o `MovingLeastSquares`. O tempo execução foi de 23 minutos. Conforme visto na Figura 57 os resultados são idênticos aos testes anteriores.

Figura 57 - `MovingLeastSquares` após `PassThrough`



No quarto e último teste, foi executado o algoritmo de estimativa de normal `MovingLeastSquares` após fazer a remoção de ruído pelo `VoxelGrid` e, conforme apresentado na Figura 58 o resultado obtido é idêntico aos três testes anteriores. O tempo de execução foi de 9 segundos e a silhueta está bem definida.

Figura 58 - `MovingLeastSquares` após `VoxelGrid`



3.5.3.3 Discussões dos resultados

Nesta seção são apresentados os resultados obtidos nos algoritmos de estimativa de normal `NormalEstimation` e `MovingLeastSquare`. Os testes foram feitos executando os algoritmos de remoção de ruídos antes das estimativas de normal. Isto se fez necessário para analisar os que se adéquam melhor, independente do algoritmo de remoção de ruídos escolhido na seção 3.5.2.

Os testes executados com a `NormalEstimation` tiveram resultados melhores em relação ao tempo de execução. No qual, o algoritmo mais demorado levou 3 minutos, mas em nenhum dos testes foi possível ver de forma clara a silhueta da mulher por conta dos espaços vazios formados quando as normais mudam de direção.

No algoritmo de `MovingLeastSquare` os algoritmos levaram mais tempo para executar, mas em todas execuções a silhueta é fiel e não existem os espaços encontrados na `NormalEstimation`.

Comparando os dois algoritmos foi possível notar a diferença no tamanho das normais. A `NormalEstimation` tem normais com comprimento maior que as normais do algoritmo `MovingLeastSquare`. Com as comparações concluiu-se que o comprimento também afeta na definição da silhueta.

Com base nos resultados o algoritmo `VoxelGrid` e `MovingLeastSquare` foram escolhidos para atuar em conjunto. O ponto decisivo na escolha foi o aspecto da normal, quanto mais fiel for a silhueta da mulher, melhor será o resultado da reconstrução.

O algoritmo de remoção de ruídos `VoxelGrid` diminui o tempo de execução de ambos os algoritmos de normal, já que ele diminui a densidade da nuvem. Por conta do `VoxelGrid`

não foi necessário se preocupar com o tempo da estimativa de normal. Desta forma, optou-se apenas por escolher o algoritmo com a silhueta mais definida.

3.5.4 Triangulação

O objetivo deste teste é avaliar a melhor forma de realizar a triangulação, quais são os melhores valores para os parâmetros da classe `GreedyProjectionTriangulation` e como os algoritmos de filtro e estimativa de normal impactam na triangulação.

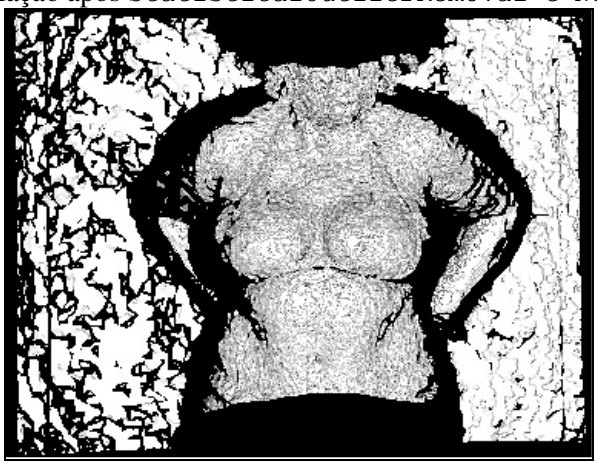
Para realizar os testes foi utilizada a classe `GreedyProjectionTriangulation` fornecida pelo PCL e os valores informados nos métodos foram:

a) `setSearchRadius`: o valor de 0.025 para que o triângulos não percam o formato da superfície;
b) `setMu`: o de valor 2.5, para estreitar o raio de busca;
c) `setMaximumNearestNeighbors`: o de valor 600 para nuvens densas e 190 para nuvens de menor densidade;
d) `setMaximumSurfaceAngle`: o valor utilizado é 45 graus, um ângulo maior que este pode significar que a normal é de um ruído;
e) `setMinimumAngle`: o valor usado é de 10 graus para que os triângulos não percam a profundidade;
f) `setMaximumAngle`: o valor usado é de 125 graus para que os triângulos não percam a profundidade.

Os valores informados acima servem como base para todos os testes realizados a seguir, mas em determinados cenários os valores podem variar. Porém, o objetivo é garantir que a variação seja a menor possível tendo como princípio tornar o algoritmo mais genérico possível.

Os resultados dos algoritmos apresentados nas seções 3.5.2 e 3.5.3 impactam diretamente no resultado da triangulação. Ou seja, quando a remoção de ruídos não consegue lidar com os ruídos da nuvem e/ou diminuir sua densidade (algoritmos `StatisticalOutlierRemoval` e `NormalEstimation`) a triangulação fica conforme apresentado na Figura 59.

Figura 59 - Triangulação após `StatisticalOutlierRemoval e NormalEstimation`

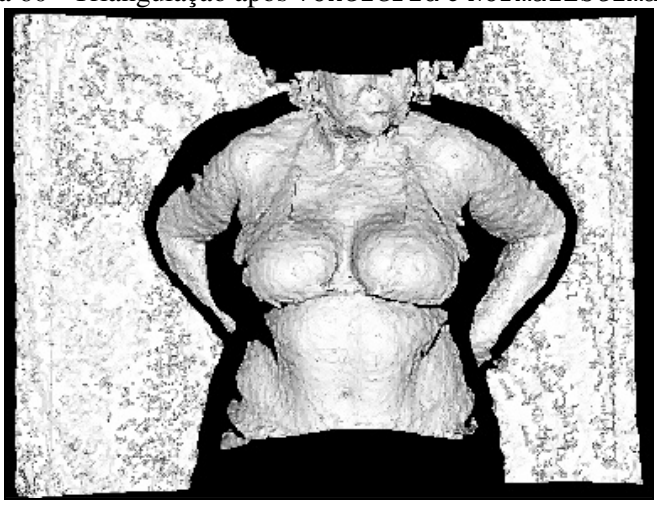


Observa-se que a superfície não é uniforme, a profundidade não está evidente e a execução leva cerca de 35 minutos.

Resultados semelhantes ao apresentado na Figura 59 ocorrem quando é aplicado a `NormalEstimation` juntamente com outros algoritmos ineficientes para diminuir a densidade (`RadiusOutlierRemoval e PassThrough`).

O algoritmo `VoxelGrid` remove a densidade da nuvem mesmo utilizando a estimativa de normal `NormalEstimation`. Neste caso, o aspecto da reconstrução é melhor, conforme pode ser visto na Figura 60.

Figura 60 - Triangulação após `VoxelGrid` e `NormalEstimation`

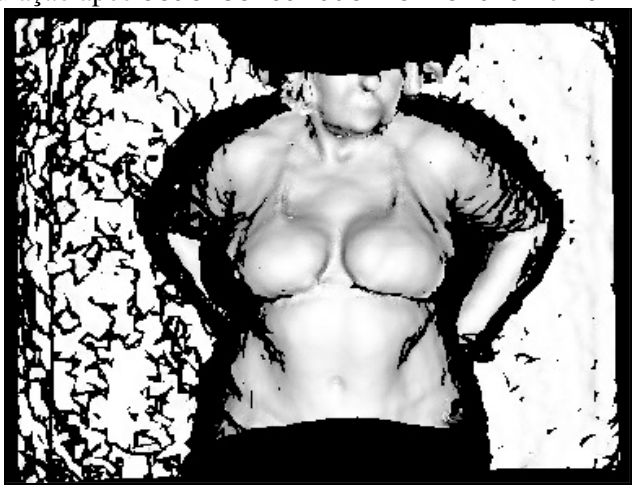


Observando o resultado é possível nota-se que a profundidade e superfície são uniformes.

Quando os algoritmos de filtro não geram bons resultados, o algoritmo de estimativa de normal `MovingLeastSquares` pode ajudar. A Figura 61 mostra o resultado da execução do algoritmo de filtro `StatisticalOutlierRemoval` aplicado antes do `MovingLeastSquares`,

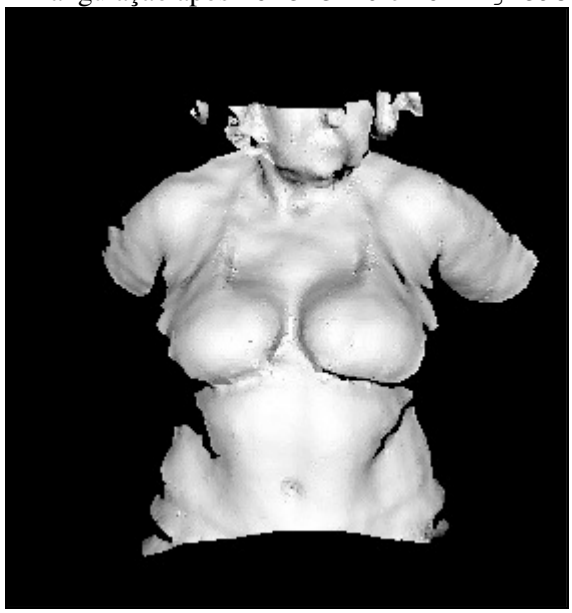
onde é visível a profundidade e superfície uniforme. A execução levou 21 minutos a mais do que a estimativa de normal `NormalEstimation` (56 minutos).

Figura 61 - Triangulação após `StatisticalOutlierRemoval` e `MovingLeastSquares`



Utilizando a estimativa de normal `MovingLeastSquares` com o algoritmo de remoção de ruído `VoxelGrid`, a execução levou 1 minuto, sendo que o resultado pode ser visto na Figura 62. A superfície é homogênea e a profundidade bem definida.

Figura 62 - Triangulação após `VoxelGrid` e `MovingLeastSquares`



3.5.4.1 Discussão dos resultados

Como pode ser notado nos testes, os algoritmos que antecedem o de *surface*, seções 3.5.2 e 3.5.3, afetam diretamente o resultado da triangulação. Os algoritmos escolhidos nestas seções foram o `VoxelGrid` e `MovingLeastSquares` respectivamente, pois são os que melhor preparam a nuvem para a triangulação. Neste caso, o algoritmo utilizado foi o `GreedyProjectionTriangulation` que produz o melhor aspecto em menos tempo.

Os valores de entrada dos métodos da classe `GreedyProjectionTriangulation` são informados no início da seção 3.5.4. A maioria dos valores foram eficientes para a execução do algoritmo de triangulação. Apenas o método `setMaximumNearestNeighbors` teve seus valores alterados conforme a nuvem mudava de densidade. Quando o valor do `setMaximumNearestNeighbors` é igual ou superior a 600, o tempo de execução é impraticável.

3.5.5 Região de interesse

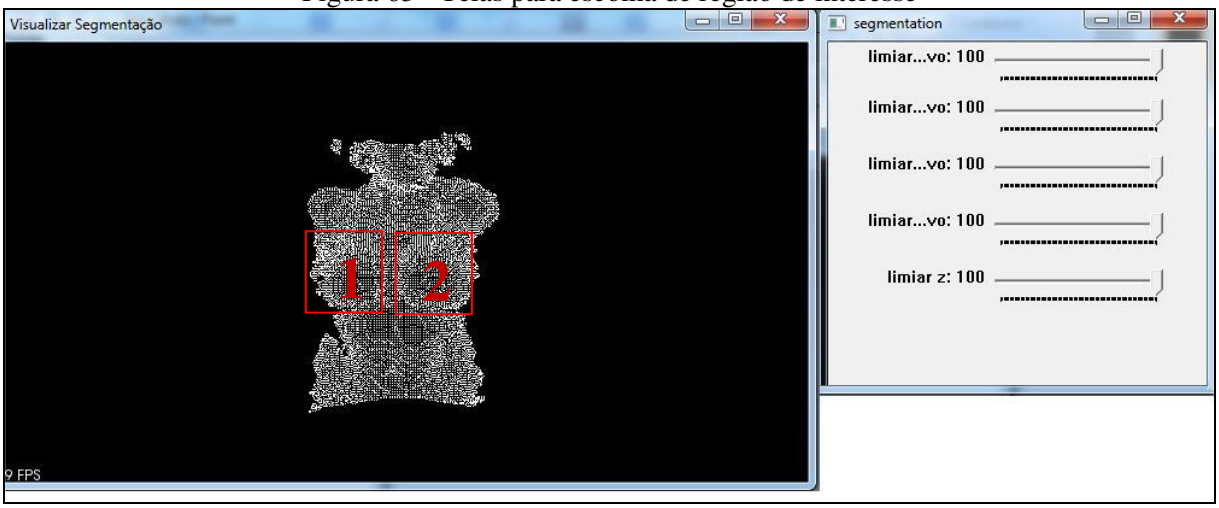
Para que o usuário possa segmentar a região dos seios na nuvem de pontos, o protótipo disponibiliza cinco barras de rolagem, que se encontram na tela *segmentation* (Figura 63), responsáveis pela remoção dos pontos, cujas coordenadas não estão dentro do limite selecionado. As duas primeiras barras excluem pontos com coordenadas y fora dos limites escolhidos, sendo a primeira barra responsável pela coordenada y positivo e a segunda, y negativo. As duas barras seguintes tem a mesma funcionalidade, mas para as coordenadas x. A última barra é para alterar a coordenada z.

Cada posição da barra de rolagem equivale à posição vezes 0.009, desta forma a exclusão dos pontos é mais precisa e se encaixa em amostras menores. Esta multiplicação é feita, pois os pontos não possuem valores superiores a 1.

Para filtrar os pontos foi utilizado o algoritmo de `PassThrough`, passando como limiar, o valor da barra de rolagem vezes 0.009.

Quando o usuário move as barrar da tela *segmentation* a nuvem de pontos da tela visualizar segmentação é alterada, estas telas podem ser vistas na Figura 63.

Figura 63 - Telas para escolha de região de interesse

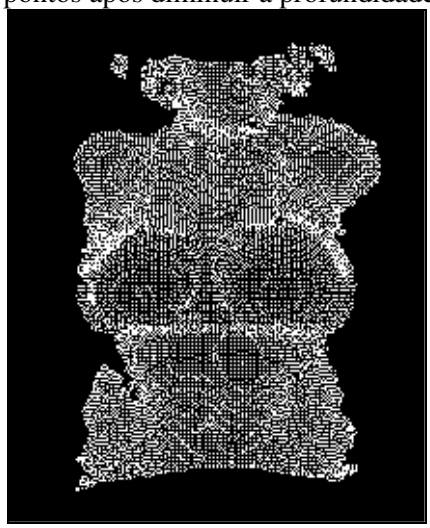


Nesta etapa o usuário deve segmentar uma mama de cada vez, para que no momento de alterar a profundidade os pontos entre os seios não sejam alterados indevidamente. Ao

finalizar a segmentação da primeira mama a tela de visualizar segmentação deve ser fechada. Com isto as telas reapareçam e a segunda mama poderá também ser segmentada. Para dar continuidade à alteração da profundidade dos pontos, a tela deve ser fechada.

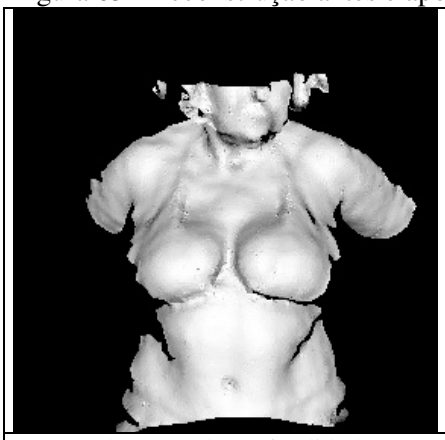
Depois de segmentar o objeto de interesse, o protótipo procura estes pontos na nuvem de pontos original e diminui em um terço a profundidade do ponto. Esta fase tem como objetivo aumentar os seios para simular implantes de mama. Depois de passar por este passo chegasse ao resultado da Figura 64.

Figura 64 - Nuvem de pontos após diminuir a profundidade da região de interesse

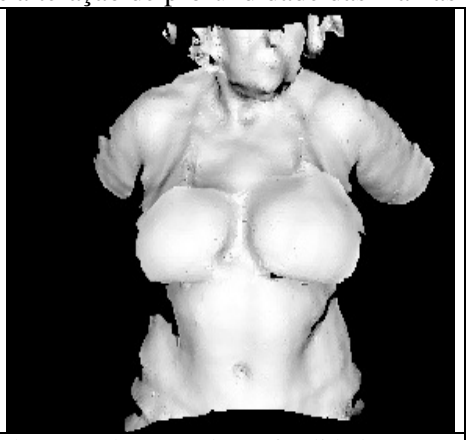


Após reduzir a profundidade, deve-se aplicar os algoritmos `VoxelGrid`, estimativa de normal com `MovingLeastSquares` e, por fim é efetuada a triangulação com `GreedyProjectionTriangulation`. O resultado desta sequência de algoritmos pode ser visto na Figura 65 item b.

Figura 65 - Reconstrução antes e após alteração de profundidade das mamas



(a) Sem alteração da profundidade



(b) Com alteração da profundidade

É possível observar que a região das mamas ficou com uma cor diferente do restante, passando a impressão que a região foi encaixada naquele local.

3.5.5.1 Discussões sobre a região de interesse

Com base nos resultados da triangulação, pode-se notar que a região de interesse não foi eficiente para dar a impressão de implante de mamas. Esta situação acontece, pois foi retirada uma região com pontos e alterada a profundidade sem levar em conta as curvas do corpo.

Por falta de tempo hábil não foi possível alterar o código para que fosse levado em consideração todos os fatores citados acima, com isto o código foi removido da seção 3.3.2, na qual é apresentada a implementação do protótipo.

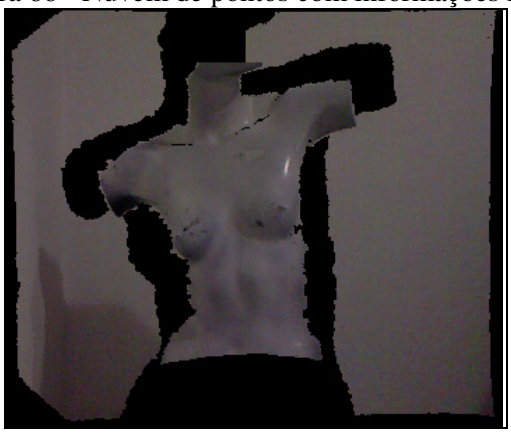
3.5.6 Nuvem de pontos com cores

A PCL suporta a aquisição de pontos com a informação *Red Green Blue* (RGB) provenientes do Kinect. Ela fornece suporte para salvar, manipular e visualizar nuvens de pontos com informações de cor. Em determinados algoritmos, a cor pode ser utilizada para remover ruídos ou suavizar a nuvens de acordo com o padrão de cor da imagem.

Para que a nuvem de pontos seja tratada com as informações RGB é necessário informar `PointXYZRGB` nos métodos que possuem o *template* (recurso da linguagem C++) `PointT`. Foram feitos testes com pontos RGB para que a reconstrução final possuísse além de textura, cores idênticas ao do objeto real.

A primeira diferença encontrada em pontos em cores foi que a iluminação afeta a captura feita pelo Kinect, ou seja, lugares sem iluminação não geram informações, lugares pouco iluminado fornecem imagens com buracos e a iluminações laterais e frontais fornecem boas amostras. A Figura 66 mostra uma nuvem de pontos "colorida".

Figura 66 - Nuvem de pontos com informações RGB



Para visualizar a nuvem de pontos com informações RGB é necessário informar explicitamente no método `addPointCloud`. Esta informação é obtida através da classe `PointCloudColorHandlerRGBField`, que extrai as informações de cor da nuvem.

A biblioteca só suporta visualizações com cores de nuvens de pontos e polígonos isolados, mas para visualizar vários polígonos interligados deve ser utilizado o método `addPolygonsMesh`, que não possui a opção de informar o RGB, ou seja, não é possível visualizar superfícies com as cores originais.

3.5.6.1 Discussões dos resultados

O principal objetivo deste teste foi tentar demonstrar a visualização de uma superfície com as cores reais, porém, isto só é possível a partir de pontos com as informações RGB. Por este motivo, os pontos `PointXYZRGB` não foram aderidos no algoritmo final do protótipo.

3.5.7 Comparação entre os trabalhos correlatos e o trabalho proposto

No Quadro 10 é realizada a comparação entre as principais características dos trabalhos correlatos e o protótipo desenvolvido neste trabalho.

Quadro 10 - Comparação entre os trabalhos correlatos e o trabalho proposto

características/ trabalho relacionado	Protótipo desenvolvido	Hyvarinen (2011)	Izadi et al. (2011)	Yamada et al. (2013)	Mendes (2012)
utiliza MicrosoftKinect como *scanner* 3D	X	X	X	X	X
manipulação de nuvens de ponto	X	X	X	X	X
utiliza PCL	X	X			
utiliza ICP (incremental)			X	X	
utiliza *K-D Tree*	X	X		X	
reconstrução de superfícies	X	X	X		X
reconstrução por triangulação	X	X			X
filtra nuvem de pontos	X	X		X	
interage com mundo real			X		
exporta malha de triângulos para aplicações externas		X			

No quadro acima é possível observar que o protótipo desenvolvido implementa sete das dez características dos trabalhos correlatos. O protótipo desenvolvido, assim como todos os outros utiliza o Microsoft Kinect como *scanner* 3D para obter a nuvem de pontos e, também manipulá-las.

Hyvarinen (2011) e o protótipo são os únicos que utilizam a biblioteca PCL para manipular as nuvens de ponto. O protótipo também filtra a nuvem de pontos, removendo a densidade e eventuais ruídos, assim como Yamada et al. (2013) e Hyvarinen (2011).

O protótipo utiliza a biblioteca PCL então, implicitamente utiliza-se a estrutura *K-D Tree*, assim como Hyvarinen (2011) e Yamada et al. (2013). Ele também realiza a reconstrução 3D por triangulação, método utilizado por Mendes (2012) e Hyvarinen (2011).

O protótipo não suporta: algoritmo de ICP implementado por Yamada et al. (2013), interação com o mundo real desenvolvido por Izadi et al. (2011) e não exportar malhas de polígonos para outros editores como desenvolveu Hyvarinen (2011).

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um protótipo que permite realizar a reconstrução tridimensional de uma nuvem de pontos obtida pelo *scanner* 3D Kinect. Também foram apresentados alguns conceitos/algoritmos de reconstrução 3D e a biblioteca PCL. Além de vários testes com algoritmos fornecidos pelo PCL para saber qual deles se adequará melhor ao objetivo proposto.

O primeiro teste foi realizado para ver em qual posição a mulher deve estar para que nuvem de pontos seja ideal para a reconstrução. Neste teste viu-se que quanto mais próximo o Kinect pior é a amostra e quanto mais afastado mais informações não indesejadas aparecem, com este cenário decidiu-se por um meio termo de 50 a 57 centímetros de distância entre a mulher e o Kinect.

No segundo teste foram obtidas várias amostras de silhuetas para testar os filtros implementados. Os testes realizados em amostras com posições mais próximas ao sensor obtiveram os mesmos resultados de amostras na posição correta. Não importa se existem buracos na nuvem, pois os algoritmos de filtro tratarão a nuvem da mesma forma que uma que não possuem buracos, mesmo que alguns valores sejam alterados para se adaptar a nuvem.

Com base no segundo teste foi constatado que somente um algoritmo de filtro não é o suficiente para o cenário, para remover o fundo e possíveis ruídos ao redor da silhueta deve ser utilizado o `PassThrough` e para diminuir a densidade o `VoxelGrid`.

No terceiro teste foi realizada a comparação entre os algoritmos `NormalEstimation` e `MovingLeastSquares`, para verificar qual deles gera as melhores normais.

Observando os resultados do terceiro teste pode-se ver que ambos os algoritmos não tiveram bons resultados com nuvens densas, por este motivo a comparação ficou por conta de nuvens preparadas anteriormente com o algoritmo de `VoxelGrid`. Neste caso, o `MovingLeastSquares` foi o que teve o melhor resultado,

Para que a superfície seja reconstruída utilizou-se a classe `GreedyProjectionTriangulation` que é responsável pela triangulação. Os testes realizados com essa classe evidenciaram os melhores valores de entrada para os métodos que auxiliam o algoritmo de triangulação. O valor de entrada mais influente na triangulação foi o do método `setMaximumNearestNeighbor` que quanto maior o valor, mais demorada fica a triangulação e, quanto menor mais pontos serão descartados afetando a superfície, sendo que o melhor valor encontrado foi 190, pois ele não deixa nenhum ponto de fora e não faz com que a execução seja demorada.

Os testes com ROI e pontos coloridos foram realizados para talvez incluir no protótipo final, mas não obtiveram bons resultados, por isto foram descartados, mas abriram espaços para que trabalhos futuros tentem melhorar estas idéias e incorporem ao protótipo.

Por fim, a partir dos resultados alcançados conclui-se que o protótipo criado consegue a partir de um ambiente controlado capturar uma nuvem de pontos, salvá-la e reconstruí-la de forma homogênea e realística em menos de três minutos.

4.1 EXTENSÕES

Como sugestão de extensão para o protótipo propõe-se:

a) utilizar pontos coloridos, para que o resultado final tenha textura igual ao objeto original;
b) realizar a segmentação das mamas e incluir uma interface para que o usuário possa alterar o tamanho delas;
c) criar uma interface amigável para o usuário de modo que ele consiga salvar a nuvem e reconstruí-la de maneira intuitiva;
d) incluir o algoritmo de ICP, para que seja possível visualizar as laterais e a parte de trás do corpo.

REFERÊNCIAS

BORGES, Daniela A. E. G. **Reconstrução de Objectos 3D Usando Kinect**. 2013. Dissertação (Mestrado em Engenharia Informática e de Computadores) - Técnico Lisboa, Lisboa.

CASTRO, André. **Kinect SDK 1.5 parte** 2: câmera de profundidade (depth). [S. l.], 2013. Disponível em: <http://www.100loop.com/destaque/kinect-sdk-1-5-parte-2-camera-de-profundidade-depth/>. Acesso em: 15 abr. 2013.

COUSINS, Steve; RASU, Radu B. **3D is here:** Point Cloud Library (PCL). Xangai, maio 2011. Disponível em: <http://pointclouds.org/assets/pdf/pcl_icra2011.pdf>. Acesso em : 19 maio 2014.

CRAWFORD, S. **How Microsoft Kinect Works**. [S. l], 2010, Disponível em: <http://electronics.howstuffworks.com/microsoft-kinect2.htm>. Acesso em: 30 jun. 2014.

CRMMEDIC. **Dúvidas mais frequentes sobre implantes mamários.** [S. l.], [2010?]. Disponível em: < http://www.crmmedic.com.br/guia-da-paciente.php>. Acesso em: 16 jul. 2014.

GOIS, João P. **Reconstrução de superfícies a partir de nuvens de pontos**. 2008. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, São Carlos.

GUEDES, Andre L. P. **Triangulação**. Paraná, 1996. Disponível em: <http://www.inf.ufpr.br/andre/geom/node8.html>. Acesso em: 30 maio 2014.

HYVARINEN, Juha. **Surface reconstruction of point cloured with microsoft kinect**. [s. l], 2012. Disponível em: <http://publications.theseus.fi/bitstream/handle/10024/42161/Hyvarinen_Juha.pdf?sequence=1>. Acesso em: 15 maio 2014.

IZADI, Shahram et al. **KinectFusion**: real-time 3D reconstruction and interaction using a moving depth camera. United Kingdom, 2011. Disponível em: <http://research.microsoft.com/pubs/155416/kinectfusion-uist-comp.pdf>. Acesso em: 05 abr. 2013.

JORGE, Lúcio A. C.; REIS, Vitor A. **Reconstrução digital de objetos em 3D por nuvem de pontos utilizando o Kinect**. São Paulo, 2013. Disponível em: <http://uniseb.com.br/presencial/revistacientifica/arquivos/jul-15.pdf>. Acesso em: 20 jun. 2014.

JOSÉ, Marcelo A. **Sistema de reconstrução 3D de baixo custo**. 2008. Dissertação (Mestrado em Sistemas Eletrônicos) - Escola Politécnica, São Paulo.

JOSÉ, Marcelo A.; LOPES, Roseli D. **Sistema de reconstrução 3D de baixo custo**. São Paulo, 2007. Disponível em: <http://www.degraf.ufpr.br/artigos_graphica/SISTEMA%20DE%20RECONSTRUCAO.pdf> Acesso em 20 jul. 2014.

Least Squares. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <<http://en.wikipedia.org/wiki/Least_squares>. Acesso em: 12 maio 2014.

LOWENSOHN, John. **Timeline:** a look back at Kinect's history. CNET gives a rundown of the history of Microsoft's Kinect, from the first rumors of a motion controller all the way to the technology becoming a smash hit. [S. l.], 2011. Disponível em: <http://news.cnet.com/8301-10805_3-20035039-75.html>. Acesso em: 22 abr. 2014.

MARTINS, Rubens S. **Aquisição e processamento de modelos tridimensionais faciais**. 2014. 48 f. Monografia (Bacharelado em Engenharia de Computação) - Universidade de Brasília, Brasília.

MENDES, Daniel U.. **Modelagem tridimensional de ambiente utilizando Kinect**. 2012. Monografia (Bacharelado em Ciência da Computação) - Universidade Regional de Blumenau, Blumenau.

MILES, Rob. **Using Kinect for Windows with XNA.** Kinect for Windows SDK. [S. l.], 2013. Disponível em: <http://www.docstoc.com/docs/147085351/Using-Kinect-for-Windows-with-XNA>. Acesso em: 22 abr. 2013.

MINATO, Mouren de V.; TRAESEL, Elise S. **Corpo e contemporaneidade:** paradigmas estéticos e envelhecimento na atualidade. [Santa Maria], [2009?]. Disponível em: <http://www.abrapso.org.br/siteprincipal/images/Anais_XVENABRAPSO/296.%20corpo%20e%20contemporaneidade.pdf>. Acesso em: 23 maio 20013.

PIZO, Gerardo A. I. **Sistema computacional para tratamento de nuvens de pontos e reconstrução tridimensional de superfícies baseada em modelos deformáveis**. 2009. 126 f. Dissertação (Mestrado em Sistemas Mecatrônicos) - Universidade de Brasília, Brasília, 2009.

POINT CLOUD LIBRARY. Documentation. [S. l.], [S. n.]. Disponível em: <http://www.pointclouds.org/documentation >. Acesso em: 19 maio 2014a.

POINT CLOUD LIBRARY. Tutorial. [S. l.], [S. n.]. Disponível em: <http://www.pointclouds.org/documentation/tutorials>. Acesso em: 19 maio 2014b.

POLIZELLI Junior, Valdecir. **Métodos implícitos para a reconstrução de superfícies a partir de nuvens de pontos**. 2008. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, São Carlos.

SÁ, José G. P. **Construindo uma DSL para reconhecimento de gestos utilizando Kinect**. 2011. 76 f. Trabalho de graduação (Graduação em Ciência Da Computação) - Centro De Informática, Departamento de Informática, Universidade Federal de Pernambuco, Pernambuco.

SILVA, João L. S. **O uso do Kinect além dos games: OpenKinect e Kinect SDK**. Vitória, 2014. Disponível em: <http://jlsouzasilva.wordpress.com/2014/02/18/dica-o-uso-do-kinect-alem-dos-games-openkinect-e-kinect-sdk>. Acesso em: 20 jul. 2014.

VALDIVA, Paola T. **Correção de normais para suavização de nuvens de pontos**. 2013. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, São Carlos.

Voxel. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <http://en.wikipedia.org/wiki/Voxel>. Acesso em: 12 maio 2014.

YAMADA, Fernando A. A. et al. **Reconstrução de objetos 3D utilizando estruturas de indexação espacial com o Microsoft Kinect**. Juiz de Fora, 2013. Disponível em: <http://www.ufjf.br/getcomp/files/2013/03/Reconstru%C3%A7%C3%A3o-de-Objetos-3D-utilizando-Estruturas-de-Indexa%C3%A7%C3%A3o-Espacial-com-o-Microsoft-Kinect.pdf>. Acesso em: 25 maio 2014.