Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Aquisição e processamento de modelos tridimensionais faciais

Rubens de Souza Martins

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia de Computação

Orientador
Prof. Dr. Camilo Chang Dorea

Brasília
2014

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Engenharia de Computação

Coordenador: Prof. Dr. André Costa Drummond

Banca examinadora composta por:

Prof. Dr. Camilo Chang Dorea (Orientador) — CIC/UnB
Prof. Dr. Bruno Luiggi Macchiavello Espinoza — CIC/UnB
Prof. Dr. Diogo Caetano Garcia — FGA/UnB

CIP — Catalogação Internacional na Publicação

Martins, Rubens de Souza.

Aquisição e processamento de modelos tridimensionais faciais / Rubens de Souza Martins. Brasília : UnB, 2014.
111 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2014.

1. Kinect, 2. modelagem facial, 3. computação gráfica

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Aquisição e processamento de modelos tridimensionais faciais

Rubens de Souza Martins

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia de Computação

Prof. Dr. Camilo Chang Dorea (Orientador)
CIC/UnB

Prof. Dr. Bruno Luiggi Macchiavello Espinoza
CIC/UnB

Prof. Dr. Diogo Caetano Garcia
FGA/UnB

Prof. Dr. André Costa Drummond
Coordenador do Bacharelado em Engenharia de Computação

Brasília, 6 de fevereiro de 2014

Dedicatória

Dedico este trabalho a todos os que tem me ajudado nessa jornada. A minha família, minha namorada, meus amigos e professores.

Agradecimentos

Agradeço ao Camilo Chang Dorea por ter me auxiliado e direcionado durante a realização desse trabalho.

Resumo

Este trabalho visa a implementação de um sistema de baixo custo que se baseia no uso de um sensor de profundidade (Microsoft Kinect) para aquisição de imagens RGB e de mapas de profundidade para geração de modelos tridimensionais faciais. Modelos tridimensionais da face permitem análises mais complexas de padrões faciais, assim como maior precisão de reconhecimento biométrico do que modelos bidimensionais.

Sua implementação foi dividida em duas etapas principais, aquisição e processamento. Durante a etapa de aquisição, o sistema deve rastrear a face do usuário em tempo real e registrar toda a região do rosto em uma nuvem de pontos. Em seguida, o processamento deve aplicar filtros para melhorar o aspecto visual da nuvem, construir uma malha de polígonos a partir dos pontos e então aplicar texturas sobre o modelo tridimensional.

Palavras-chave: Kinect, modelagem facial, computação gráfica

Abstract

This work aims to implement a low cost system that is based on the use of a depth sensor (Microsoft Kinect) to acquire RGB images and depth maps in order to generate tridimensional facial models. Tridimensional models of the face provide a more complex analysis of facial patterns, as well as more accuracy in biometric recognition than bidimensional models.

Its implementation was divided into two main steps, acquisition and processing. During acquisition the system must track the user's head in real time and register the whole portion of the face in a 3D point cloud. After that, processing applies filters to inhance the visual aspect of the point cloud, build a polygon mesh from the points and then apply textures upon the tridimensional model.

Keywords: Kinect, facial modelling, computer graphics

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação e Justificativa

Características faciais são o conjunto de informações que caracterizam uma face humana. Elas são úteis em diversos tipos de sistemas, como sistemas de segurança, de reconhecimento de expressões, de modelagem 3D, dentre outros [9]. Pesquisas nessas áreas podem fazer uso tanto de informações visuais (por meio de imagens RGB), quanto geométricas, que permitem um conjunto maior de possibilidades de análises. Para que haja a captação de informações de formato geométrico são necessários sensores 3D, como sensores laser Hokuyo [10], ou sensores de profundidade Primesense [11], que atualmente é parte do dispositivo Microsoft Kinect [12]. A vantagem do último é sua viabilidade devido ao baixo custo o que motivou em sua escolha para este trabalho.

Gerar um modelo 3D de uma face a partir da câmera 3D do Microsoft Kinect não é um problema simples pois um único quadro não é suficiente para gerar o modelo completo. Isso ocorre porque os dados do Kinect são de baixa resolução e precisão, o que resulta em informações não confiáveis em superfícies de formato complexo e nas bordas.

Para contornar essa situação é necessário o uso de vários quadros, pois cada quadro é aliado a uma pose que pode fornecer informações novas a respeito do formato do objeto sendo escaneado. Esse processo ocorre de forma iterativa, sempre acumulando mais informações úteis e filtrando ruídos. A soma de vários quadros de baixa resolução aliados a poses diferentes permite aumentar a resolução final do modelo gerado.

Ao final desse processo, o resultado obtido é uma nuvem formada por um conjunto de pontos que serão processados com intuito de melhorar seu aspecto final. Para isso, serão utilizados filtros para remoção de ruídos e suavização de superfície. Em seguida, serão gerados polígonos a partir desses pontos, em um método de triangulação. Esse método une pontos próximos de modo que se tornem vértices de uma malha de polígonos, com cada triângulo possuindo uma face a ser visualizada. Por último serão aplicadas texturas sobre o modelo gerado, sendo essas texturas nada mais que imagens RGB da face escaneada.

Sendo assim, este trabalho será organizado em cinco capítulos, de modo a detalhar o processo teórico e experimental ocorrido ao longo de sua criação:

1. Neste capítulo são mostrados a motivação e os objetivos do trabalho e também a metodogia utilizada;

2. No Capítulo 2 serão explicados os conceitos envolvidos na implementação do programa, bem como especificações dos recursos de hardware e de software;
3. No Capítulo 3 será detalhada a implementação do trabalho, desde o processo de aquisição de nuvens de pontos pelo sensor até a geração de uma malha de polígonos e aplicação de textura;
4. O Capítulo 4 possui uma análise e comparação dos resultados obtidos e outros resultados possíveis;
5. Por último, o Capítulo 5 resume o que foi abordado e conclui sobre os objetivos do trabalho, além de surgerir possibilidades para trabalhos futuros.

1.2 Objetivos gerais e específicos

- Objetivo Geral: Criar um sistema de captação e processamento de imagens RGBD para geração de modelos tridimensionais faciais;
- Objetivo Específico: Otimizar a captura dos dados considerando as limitações do sensor de profundidade;
- Objetivo Específico: Observar as melhores técnicas de pré-processamento de modo a preservar as informações cruciais;
- Objetivo Específico: Verificar a eficácia e viabilidade do sistema proposto em relação às alternativas existentes.

1.3 Metodologia

A metodologia aplicada nesse projeto possui conotação prática em sua maior parte. Foram realizados estágios de desenvolvimento, onde cada estágio teve o objetivo de propor uma melhora ao resultado final. Para cada estágio foi feito:

1. Uma pesquisa bibliográfica sobre as diversas técnicas existentes;
2. Implementação de uma dessas técnicas;
3. Testes objetivos e subjetivos com relação a essa técnica;
4. Incorporação da técnica ao programa caso adequada.

Ao final desse processo é realizado um teste geral do sistema. Nesse teste é usado um mesmo conjunto de amostras de imagens RGBD (imagem RGB e de profundidade) de modo a permitir uma análise mais consistente dos resultados.

Em geral, programas de modelagem tridimensional por meio de sensores 3D usam métricas quantitativas de avaliação do formato geométrico e qualitativas do aspecto visual. Para quantizar a precisão do formato geométrico, são usados mapas de temperatura (*heatmaps*), que indicam regiões quentes onde há grande disparidade entre dois modelos geométricos de um mesmo objeto, como mostrado na Figura 1.1.

Nesse trabalho, entretanto, tal métrica não é utilizada, e os modelos tridimensionais são avaliados apenas qualitativamente tanto nos aspectos geométrico quanto visual.

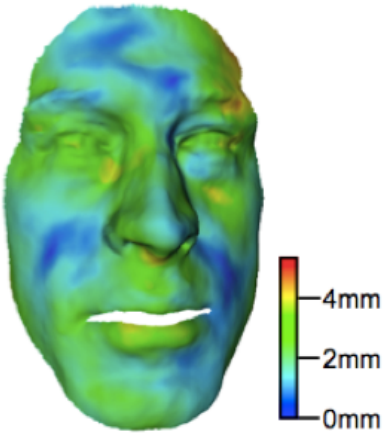


Figura 1.1: Exemplo de mapa de temperatura [1]

Capítulo 2

Revisão Teórica

Este capítulo visa apresentar os conceitos envolvidos na implementação deste trabalho. Serão mostrados termos específicos, os requisitos e especificações de hardware e software utilizados no trabalho.

2.1 Trabalhos relacionados

Sensores 3D são dispositivos que analizam objetos ou ambientes reais de modo a coletar dados sobre o formato e possivelmente sobre a aparência (cor). Os dados coletados podem então serem utilizados para construir modelos digitais tridimensionais. Muitas tecnologias podem ser usadas nesses dispositivos, de acordo com suas limitações e custos associados. Alguns sensores, por exemplo, não funcionam com superfícies brilhantes ou transparentes, e espelhos. Um desses sensores 3D de baixo custo é o Microsoft Kinect.

Desde o seu lançamento, o Kinect vem sendo amplamente utilizado em um grande números de pesquisas com diversos intuitos, possibilitados principalmente pelo seu baixo custo. Assim, muitos temas foram sugeridos, distanciando seu âmbito inicial de jogos eletrônicos para diversas outras aplicações, como a modelagem tridimensional facial. Modelos tridimensionais da face abrem mais possibilidades de análise e comparação de padrões faciais, além de reconhecimento biométrico mais preciso [13].

Um dos métodos usados para essa modelagem é a reconstrução facial baseada em um modelo facial genérico, que sofre deformação de modo a se assemelhar à face escaneada [14] [15]. Zollhöfer constrói um modelo 3D usando o Microsoft Kinect [16]. Ele ajusta o modelo facial genérico aos escaneamentos faciais e faz o refinamento baseado em aspectos da imagem RGB. Todos esses métodos obtém resultados precisos mas o modelo genérico tende a influenciar o resultado final.

Os métodos que atingem resultados de maior precisão geométrica e alta qualidade visual requerem uso de equipamento especial [17], ou um ambiente de estúdio adequado [18]. Neste último, o usuário é escaneado por uma luz com 156 LEDs que capturam a geometria e a reflectância da face. Bradley [19] usa um *setup* de 14 câmeras de vídeo de alta definição para capturar pequenas porções da superfície facial, e depois aplica um método binocular iterativo para reconstruir o modelo.

O KinectFusion [20] usa as informações de profundidade de um Kinect em movimento e cria um modelo 3D de alta qualidade de um ambiente estático, realizando um mapeamento espacial ao calcular de forma precisa o rastreamento e a pose da câmera ao longo do

processo. Todavia, essa abordagem não produz bons resultados para registro de faces ou de determinados objetos.

Em seu trabalho, Hernandez utiliza o Kinect para produzir modelos geométricos faciais sem se basear em qualquer modelo facial genérico, de modo que o escaneamento é feito em tempo real enquanto o usuário é filmado [21]. Para isso, eles usam um quadro de referência para calcular a pose de todos os outros quadros recebidos, em vez de calcular a pose entre quadros consecutivos. O problema dessa abordagem é que o cálculo da pose entre um quadro de referência e os outros quadros é mais demorado que entre quadros consecutivos.

Este trabalho visa implementar um processo de registro de faces em tempo real usando o Kinect como sensor 3D, assim como Hernandez [21]. Porém, o cálculo da pose é feito com quadros consecutivos, em vez de usar um quadro de referência, o que permite maior rapidez. Além disso, também se visa verificar possíveis melhorias nos algoritmos, de modo a obter desempenho e resultados no mínimo semelhantes aos obtidos por ele.

2.2 Nuvem de pontos

Uma nuvem de pontos [22] consiste em um conjunto de dados atribuídos a um sistema de coordenadas. Nuvens de pontos possuem aplicações em diversos tipos de ambientes, como simulação de partículas, mapeamento geográfico [23], modelagem geométrica, imagens médicas [24], entre outras.

Neste trabalho, a nuvem de pontos consiste de uma matriz de pontos com resolução 640x480. Os pontos são uma estrutura de dados com os atributos:

- Cor: um valor inteiro para cada componente R, G e B;
- Localização: um valor em ponto flutuante para cada componente p_x, p_y e p_z de um vetor posição $\vec{\mathbf{p}} = (p_x, p_y, p_z)$;
- Valor da Normal: um valor em ponto flutuante para cada componente n_x, n_y e n_z de um vetor unitário $\vec{\mathbf{n}} = (n_x, n_y, n_z)$. Inicialmente é nulo mas pode ser calculado se necessário.

Os pontos servem unicamente para definir superfícies, de modo que pontos isolados são considerados informações não desejadas.

Nuvens de pontos podem ser geradas por meio de sensores ou diretamente via software. Existem muitos *scanners* 3D de alta precisão que produzem nuvens de pontos de alta fidelidade e precisão, porém ultimamente têm surgido sensores de baixo custo como o Microsoft Kinect que, aliados a programas de grande complexidade, produzem resultados consistentes de acordo com a aplicação alvo.

2.3 Recursos de hardware

2.3.1 Microsoft Kinect

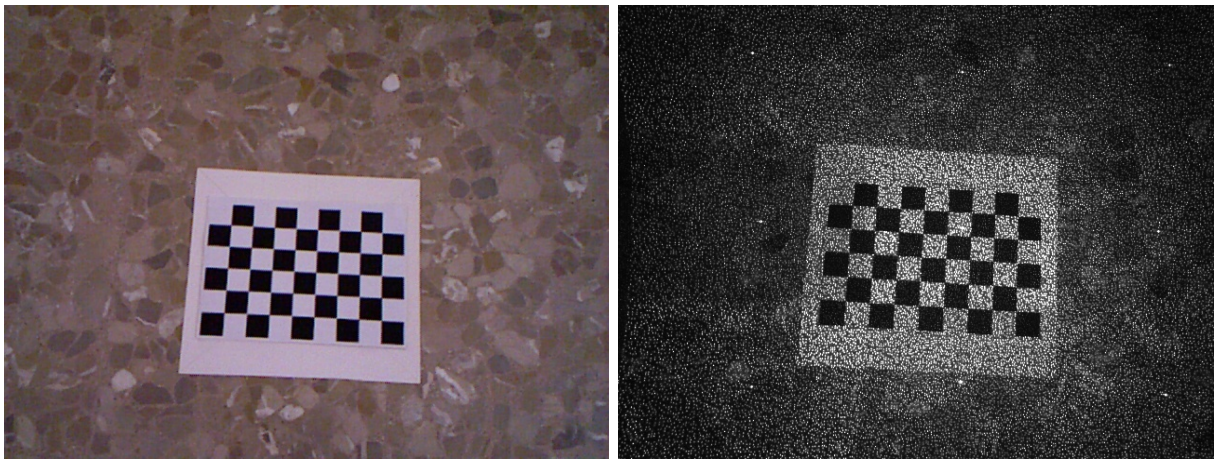
O sensor Microsoft Kinect começou a ser comercializado em Novembro de 2010 como um periférico do console XBOX 360 e, desde os primeiros dias, a comunidade *Open Source* mostrou um grande interesse pelos seus diversos campos de aplicação.

Características do sensor

O Microsoft Kinect [2] possui uma câmera RGB, um projetor infravermelho, uma câmera infravermelho, um *array* de microfones, uma base motorizada que permite movimento vertical e acelerômetros que provem informação sobre a orientação do sensor. A Figura 2.1 mostra esse conjunto de sensores que formam o dispositivo.

Figura 2.1: Disposição dos sensores no Microsoft Kinect [2].

A câmera RGB tem um fluxo de dados de 8 bits com resolução VGA (640x480 pixels) a uma frequência de aquisição de quadros de 30Hz. O sensor de profundidade é composto por um projetor infravermelho e um sensor CMOS monocromático em uma câmera infravermelho. O fluxo de dados do sensor de profundidade é de 11 bits com resolução VGA (640x480 pixels) a uma frequência de aquisição de quadros de 30Hz, embora seu campo de visão seja ligeiramente menor que o da câmera RGB pois a distância focal é maior. A Figura 2.2 mostra imagens geradas pelas duas câmeras do Kinect.



(a) Imagem RGB. (b) Imagem Infra-vermelho.

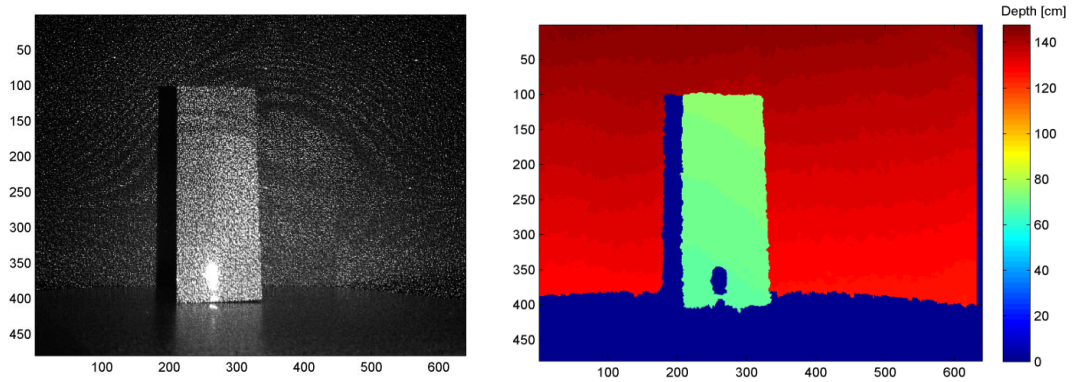
Figura 2.2: Imagens geradas pelo Microsoft Kinect [3].

O conjunto formado pelo projetor infravermelho e pela câmera infra-vermelha proporcionam uma resolução espacial de 3mm de altura e largura e de 1cm de profundidade, embora a gama de operação esteja entre 0,5 e 3,5 metros aproximadamente.

Sensor de profundidade

A resolução da câmera infra-vermelho do Microsoft Kinect determina o espaçamento entre pontos da imagem de profundidade em um plano perpendicular ao eixo da câmera. Sabendo que a imagem de profundidade possui uma resolução de 640 x 480 pixels, quanto maior a distância do objeto para o sensor, menor a densidade de pontos. Considerando a densidade de pontos como o número de pontos por unidade de área, desde que o número de pontos seja constante, a área é proporcional ao quadrado da distância do sensor [4].

O projetor infra-vermelho possui um laser que emite um feixe único que depois é partido em múltiplos feixes por uma grade difratora de modo a criar uma matriz de pontos constante projetada na cena. Essa matriz é capturada pela câmera infra-vermelha e é correlacionada contra uma matriz de referência. A matriz de referência é obtida ao capturar um plano em uma distância do sensor, que é armazenado na memória do Kinect. Quando um ponto é projetado em um objeto cuja distância ao sensor é menor ou maior que a do plano de referência, a posição do ponto na imagem infra-vermelho vai ser deslocada na direção de uma base de referência entre o projetor e o centro da câmera infra-vermelho. Esses deslocamentos são medidos para todos os pontos em um simples procedimento de correlação de imagem, que resulta em uma imagem de disparidade (*disparity image*). Para cada pixel, a distância ao sensor pode ser adquirida por meio da disparidade correspondente. A Figura 2.3 ilustra as medidas de profundidade a partir da matriz de pontos.



(a) Imagem infra-vermelho da matriz de pontos projetada em uma cena

(b) Imagem de profundidade resultante

Figura 2.3: Imagem de profundidade a partir de uma matriz de pontos [4].

A Figura 2.4 ilustra a relação entre a distância de um ponto k de um objeto ao sensor, relativo a um plano de referência XY e a disparidade medida d. Para expressar as coordenadas 3D dos pontos do objeto considera-se um sistemas de coodernadas com sua origem no centro da câmera infra-vermelho. A base de referência entre a câmera e o projetor infra-vermelho é expresso por b.

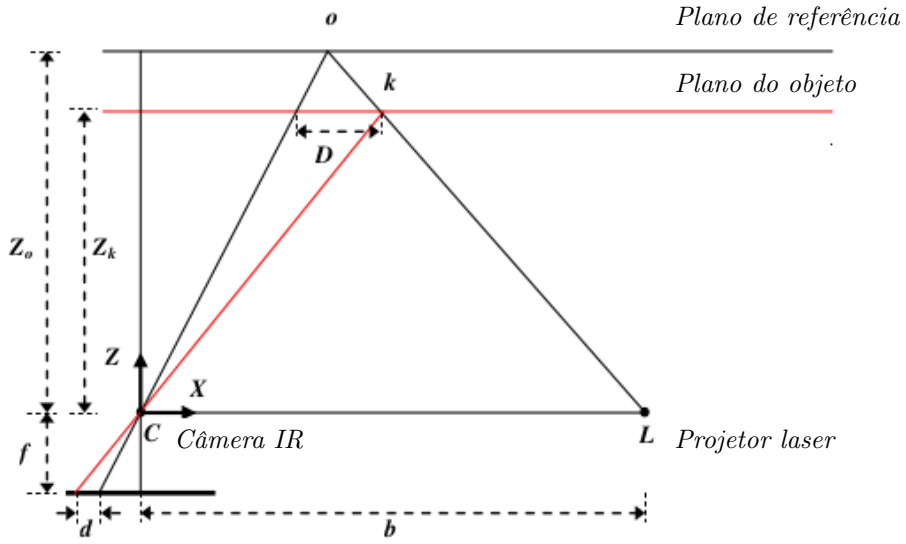


Figura 2.4: Relação entre a distância de profundidade e a disparidade medida [4].

Considere a distância focal da câmera infra-vermelho como f. Como, na prática, as medidas de disparidade são normalizadas e quantizadas entre 0 e 2047 pelo próprio Kinect, d pode ser substituído por $md' + n$, sendo d' a disparidade normalizada, e m, n os parâmetros de uma regressão linear inversa. A Equação (2.1) mostra a relação

linear entre o inverso da distância de profundidade de um ponto e sua correspondente disparidade normalizada:

$$Z_k^{-1} = (\frac{m}{fb})d' + (Z_o^{-1} + \frac{n}{fb}). \tag{2.1}$$

Assumindo *d'* como uma variável aleatória com distribuição normal, é possível propagar a variância das medidas de disparidade de modo a obter a variância das medidas de profundidade:

$$\sigma_{Z_k}^2 = \left(\frac{\partial Z_k}{\partial d}\right)^2 \sigma_{d'}^2. \tag{2.2}$$

Substituindo a Equação (2.2) na Equação (2.1), é possível obter a equação do desvio padrão da profundidade:

$$\sigma_{Z_k} = (\frac{m}{fb}) Z_k^2 \sigma_{d'}, \tag{2.3}$$

sendo σ_d' e σ_Z o desvio padrão da disparidade normalizada medida e o desvio padrão da profundidade calculada, respectivamente. A Equação (2.3) mostra que o erro aleatório de uma medida de profundidade é proporcional ao quadrado da distância do objeto ao sensor.

2.3.2 Computador

O computador utilizado para execução do software foi um *notebook* com as seguintes especificações:

- Processador Intel Core 2 Duo 2,4 GHz (Mobile P8600)
- Memória RAM de 8 GB com frequência de 1066 MHz
- Processador de vídeo NVIDIA GeForce 320M
- Disco rígido de 500 GB Serial ATA (5400 RPM)
- Sistema operacional Linux Ubuntu 12.04 (Precise Pangolin)

2.4 Recursos de software

2.4.1 Bibliotecas utilizadas

As bibliotecas utilizadas possuem desde funções até aplicativos que facilitam a conversão e processamento das informações. São projetos que envolvem processamento e manipulação de imagens, Visão Computacional, nuvens de pontos, entre outros.

MRPT *(Mobile Robot Programming Toolkit)*

Biblioteca escrita em C++, multiplataforma e *open source*, distribuída sob a licença GPL. Inclui módulos com algoritmos de Visão Computacional, SLAM, *motion planning*,

captura de dados de sensores, etc. É utilizada como interface de captura de dados do Kinect por meio da biblioteca **libfreenect**.

OpenCV *(Open Source Computer Vision Library)*

Biblioteca escrita em C/C++, multiplataforma e *open source*, distribuída sob a licença BSD. Inclui algoritmos de processamento de imagens e de Visão Computacional, como empareamento de características, visão estéreo, detecção de objetos, etc.

PCL *(Point Cloud Library)*

Biblioteca escrita em C++, multiplataforma e *open source*, distribuída sob a licença BSD. Inclui algoritmos para processamento de nuvens de pontos n-dimensionais e geometria 3D, bem como métodos específicos para filtragem, segmentação, detecção de características, reconstrução 3D, etc.

Eigen

Biblioteca escrita em C++, multiplataforma e *open source*, distribuída sob a licença LGPL. Inclui métodos de Álgebra Linear, como operações sobre matrizes e vetores, métodos numéricos de resolução de sistemas lineares, etc.

libfreenect

Biblioteca escrita em C, multiplataforma e *open source*, distribuída sob a licença Apache v2 e GPL v2. Inclui uma série de interfaces com os vários sensores e atuadores do dispositivo Microsoft Kinect.

Boost

Biblioteca escrita em C++, multiplataforma e *open source*, distribuída sob a licença Boost Software License. Inclui métodos de Álgebra Linear, *multithreading*, *smart pointers*, processamento de imagens, etc.

VTK

Biblioteca escrita em C++, multiplataforma e *open source*, distribuída sob a licença BSD. Inclui uma ampla variedade de algoritmos para visualização de texturas, volumes, etc.

CUDA

Conjunto de ferramentas criadas pela NVidia que permite codificar algoritmos para GPU mediante uma linguagem parecida com C. A principal utilidade da CUDA consiste em aproveitar o paralelismo de dados para resolver certos problemas de forma mais eficientes que usando a CPU.

2.5 Fluxo de processamento de Visão Computacional

A estrutura de cada um dos elementos que compõem sistemas de Visão Computacional depende muito da aplicação a ser implementada. É possível, no entanto, verificar uma organização geral de tarefas realizadas na maioria das aplicações, como mostrado a seguir:

1. **Aquisição de imagens** a partir de um ou mais sensores que podem ser câmeras sensíveis a um determinado espectro de luz, sensores de profundidade, dispositivos de tomografia, radar, câmeras ultrasônicas, etc. Dependendo do sensor, a imagem pode ser 2D, um volume 3D ou uma sequência deles.
2. **Pré-processamento**. Normalmente, as imagens são processadas antes de aplicar métodos de Visão Computacional sobre elas. Esta etapa é realizada para destacar características da imagem, eliminar ruídos, retificar deformações produzidas pela ótica, etc.
3. **Extração de características**. É possível extrair características das imagens a diferentes níveis de complexidade. Alguns exemplos são linhas, bordas, pontos de interesse, até características mais complexas, como textura, forma, etc.
4. **Detecção/Segmentação**. Em algum instante deve se decidir que pontos ou regiões da imagem são relevantes para as etapas seguintes do processamento. Alguns exemplos são a seleção de um subconjunto de pontos de interesse ou a segmentação de uma ou mais regiões da imagem que contêm um objeto representativo.
5. **Processamento de alto nível**. Neste ponto, os dados de entrada são somente um conjunto de dados reduzido. Tarefas de processamento que costumam aparecer neste nível são, por exemplo:
   (a) Verificação de se os dados satisfazem algum modelo.
   (b) Estimação de parâmetros específicos da aplicação, como a pose de um objeto ou o seu tamanho.
   (c) Reconhecimento de imagens para classificar os objetos detectados em diferentes categorias.
   (d) Registro da imagem para unificar os dados em um mesmo sistema de referência, como por exemplo na construção de mapas 3D.

Capítulo 3

Implementação do Sistema

Neste capítulo será explicado o processo de implementação do sistema e os algoritmos que o constituem. Além disso, será detalhado a sequência de algoritmos desenvolvidos ou utilizados, bem como os ajustes realizados no código e nos parâmetros de entrada.

3.1 Aquisição

O processo de aquisição consiste em receber imagens RGBD como dados de entrada e salvar uma nuvem de pontos representando um ambiente ou objeto como dados de saída. Os dados de entrada são obtidos por meio da captação de imagens com o Microsoft Kinect. Para usar o Microsoft Kinect em tempo real, é necessário a biblioteca libfreenect, que possui o driver de Linux do Microsoft Kinect, e a biblioteca OpenNI, que disponibiliza funções de acesso ao dispositivo.

Como cada imagem RGBD possui uma cena sendo observada, é necessário saber a correlação entre as duas últimas imagens recebidas, de modo que seja possível calcular o deslocamento espacial, seja da câmera ou do ambiente. No caso do deslocamento da câmera, esse problema pode ser resolvido com o uso de sensores adicionais, de modo a informar sua posição no espaço e o vetor de direção da câmera. Sem o uso de sensores extras, surge a necessidade de se estimar a pose de cada novo quadro recebido.

Após a pose do objeto ser atualizada, resta identificar se ela exibe regiões que ainda não foram registradas. O algoritmo fará então o registro dos novos pontos para a nuvem de pontos de saída.

Esse cenário constitui um dos conceitos de SLAM (*Simultaneous Localization And Mapping*) [25], técnica da área de robótica que consiste de um robô (ou um sistema sensorial) que se encontra em uma localização desconhecida poder construir (ou atualizar) incrementalmente um mapa consistente do seu entorno, e ao mesmo tempo determinar sua posição no espaço. Um dos algoritmos usados nesse processo analisa quadro a quadro qual foi a translação e a rotação do sensor, em um método conhecido como Alinhamento par a par (*Pairwise alignment*), que consiste em um processo de aproximação e refinamento da pose mediante o algoritmo *Iterative Closest Point* (ICP).

Sendo assim, o algoritmo da parte de aquisição é constituído de três etapas:

1. Pré-processamento: a imagem recebida é segmentada de modo que fique apenas a região de interesse;

2. Transformação: a transformação afim (rotação e translação) entre a imagem atual e nova imagem é calculada por meio do ICP e permite que a nova pose seja corretamente estimada;

3. Atualização da nuvem de pontos: caso a nova pose tenha regiões ainda não registradas, novos pontos serão adicionados à nuvem de pontos da imagem atual.

que estão ilustradas na Figura 3.1:

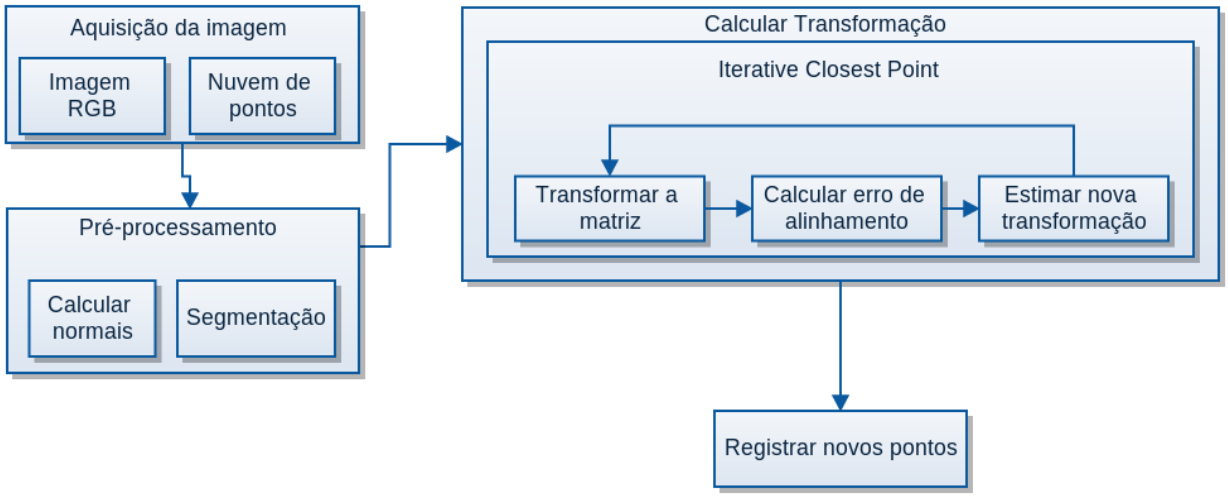


Figura 3.1: Diagrama de blocos constituindo a Aquisição

3.1.1 Pré-processamento

O pré-processamento é uma etapa importante nos algoritmos de processamento de imagens, uma vez que possibilita:

1. A remoção de informações indesejadas, como *outliers* (observações que são numericamente distantes do resto dos dados), ruídos, dados redundantes ou irrelevantes que acrescentam tempo indesejado no fluxo de processamento;

2. Ajuste para um intervalo de frequência específico, quando a essência da informação ou do sensor não corresponde à frequência desejada;

3. Extração de características desejadas, como uma região específica a ser segmentada, linhas, bordas ou formatos específicos.

Neste trabalho, o pré-processamento consiste da extração de características desejadas, sendo elas a segmentação de uma região específica e o cálculo das normais das superfícies formadas pelos pontos. O algoritmo de segmentação é usado durante o pré-processamento pois, se não houver a segmentação da região de interesse, o ICP não consegue calcular a transformação corretamente. As normais de superfície são calculadas no pré-processamento pois são usadas durante o ICP.

Segmentação

Nesta etapa é utilizado um algoritmo de segmentação. Esse algoritmo avalia se cada ponto da nuvem se encontra dentro de uma região retangular, com coordenadas especificadas como parâmetros. Seu uso requer os limites superior e inferior em x, y e z, de modo que quaisquer pontos fora desses limites serão desconsiderados. O código a seguir, escrito em linguagem C, ilustra esse processo.

Neste trabalho, a região segmentada corresponde a um retângulo de dimensões 30cm de largura por 30cm de altura por 22cm de profundidade que se encontra a 50cm de distância do sensor. Esses valores foram determinadas experimentamente.

```
void segmentCloud (PointCloud cloud_in, PointCloud cloud_out)
{
   for (int i=0; i<cloud_in->size(); i++)
   {
      Point point = (*cloud_in)[i];
      if (point.x >= LIMITE_INF_X && point.x <= LIMITE_SUP_X &&
          point.y >= LIMITE_INF_Y && point.y <= LIMITE_SUP_Y &&
          point.z >= LIMITE_INF_Z && point.z <= LIMITE_SUP_Z)
      {
         cloud_out->push_back(point);
      }
   }
}
```

Cálculo das normais de superfícies

Normais de superfície são uma propriedade importante de superfícies geométricas, e são extensivamente utilizadas em muitas áreas de computação gráfica, de modo à aplicar efeitos visuais como iluminação e sombreamento. Dada uma superfície geométrica, geralmente é trivial inferir a direção da normal em um ponto da superfície como sendo o vetor perpendicular à superfície naquele ponto. Entretanto, ao considerar que os *datasets* de nuvens de pontos adquiridos representam uma amostra de pontos de uma superfície real, talvez possa ser necessário usar aproximações para calcular as normais da nuvem de pontos diretamente.

O algoritmo em pseudocódigo usado para calcular as normais de superfície se encontra logo abaixo.

```
Inicio CalcularNormais (nuvemdepontos Nuvem)

Para cada ponto P em Nuvem:
   Buscar os pontos proximos do ponto P;
   Calcular a normal de superficie N do ponto P;
   Verificar se normal esta orientada consistentemente;
   Se normal nao esta orientada consistentemente
      Inventer o sinal da normal;
```

```
    Fim–Se
Fim–Para
Fim–CalcularNormais
```

Para estimar as normais de superfície é necessário uma análise dos autovetores e autovalores por meio da Análise de Componentes Principais (PCA) de uma matriz de covariância criada a partir dos vizinhos próximos de cada ponto. Mais especificamente, para cada ponto $\mathbf{p}_i$, determina-se a matriz de covariância C como:

$$C = \frac{1}{k}\sum_{i=1}^{k}(\mathbf{p}_i - \overline{\mathbf{p}}) \cdot (\mathbf{p}_i - \overline{\mathbf{p}})^T, \tag{3.1}$$

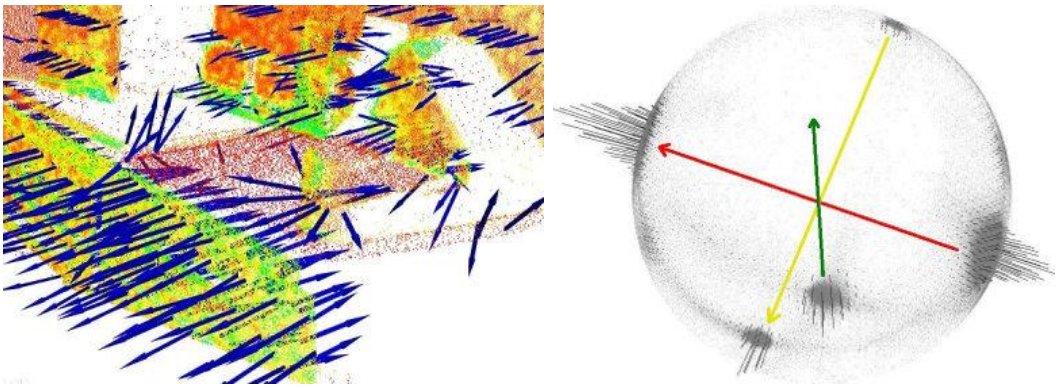
sendo k o número de pontos vizinhos considerados na vizinhança de $\mathbf{p}_i$, $\overline{\mathbf{p}}$ representa a centróide 3D dos vizinhos próximos. Para o PCA, temos:

$$C \cdot \vec{\mathbf{v}_j} = \vec{\mathbf{v}_j} \cdot \lambda_j, j \in \{0, 1, 2\}, \tag{3.2}$$

onde λ_j é o j-ésimo autorvalor da matriz de covariância e $\vec{\mathbf{v}_j}$ o j-ésimo autovetor. A curvatura da superfície σ é estimada pela relação entre os autovalores da matriz de covariância como:

$$\sigma = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}. \tag{3.3}$$

Em geral, como não há um modo de determinar o sinal de uma normal, sua orientação calculada via PCA como mostrado acima é ambígua e acaba não sendo consistentemente orientada ao longo da nuvem de pontos. A Figura 3.2(a) mostra esses efeitos em um conjunto de dados representando um ambiente de uma cozinha. A Figura 3.2(b) apresenta uma Imagem Gaussiana Estendida (EGI), também conhecida como esfera normal, que descreve a orientação de todas as normais da nuvem de pontos. Como a nuvem de pontos foi registrada de um único ponto de vista (*viewpoint*), as normais deveriam estar presentes somente em metade da esfera no EGI. Todavia, devido à inconsistência de orientação, elas estão espalhadas ao redor da esfera.



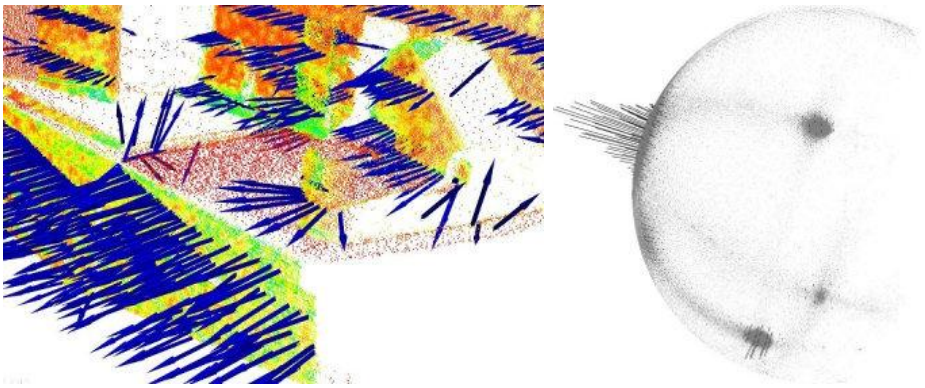
(a) Ambiente com normais de superfície (b) Esfera normal

Figura 3.2: Normais orientadas de forma inconsistente [5].

A solução pra esse problema é trivial se o ponto de vista $\mathbf{v_p}$ for conhecido. Para orientar todas as normais $\vec{n}_i$ de forma consistente em direção ao ponto de vista, elas precisam satisfazer a equação:

$$\vec{n}_i \cdot (\mathbf{v_p} - \mathbf{p}_i) > 0. \tag{3.4}$$

A Figura 3.3 apresenta o resultado depois que as normais foram orientadas em direção ao ponto de vista de forma consistente.



(a) Ambiente com normais de superfície (b) Esfera normal

Figura 3.3: Normais orientadas de forma consistente [5].

3.1.2 Transformação

Assumindo uma situação em que foram registradas duas nuvens de pontos de uma mesma cena:

- A nuvem A, que representa o objeto em sua posição original;
- A nuvem B, que foi transformada afim e sofreu rotação e/ou translação em relação à nuvem A.

O processo de determinar qual foi essa transformação consiste em alinhar a nuvem B à nuvem A até que esse alinhamento atinja um patamar mínimo aceitável. Esse alinhamento é conhecido como alinhamento par a par (*Pairwise Alignment*) e é realizado para que o registro de uma cena possa ser feito corretamente. Para isso, é necessário um algoritmo que efetue esse posicionamento entre as duas nuvens (de modo iterativo, sempre melhorando a estimativa), que é o caso do ICP.

Iterative Closest Point

ICP [26] é um método de registro que usa informações geométricas (em vez de cor ou intensidade) para alinhar duas nuvens de pontos transformadas entre si. O processo de alinhamento ocorre conforme descrito a seguir:

1. Inicialmente a nuvem B está completamente desalinhada à nuvem A.
2. Após n iterações, a nuvem B deve estar alinhada em rotação somente.
3. Após $n + k$ iterações a nuvem B se sobrepõe completamente à nuvem A e está alinhada tanto em rotação quanto em translação.

Para entender como isso ocorre, é necessário entender como funciona a iteração de alinhamento do ICP. Seu princípio consiste em minimizar uma função de erro de alinhamento, cujo resultado é menor à medida em que as nuvens estiverem mais alinhadas.

Considere a matriz de pontos da nuvem A como $\mathbf{S_A}$ e a de B como $\mathbf{S_B}$. O primeiro passo do ICP é encontrar pontos de $\mathbf{S_A}$ e $\mathbf{S_B}$ que são correspondentes. Esse problema é tido como problema de correspondência (*correspondence problem*) e trata do processo de identificar partes correspondentes em duas imagens que foram transformadas entre si. O ICP não trata esse problema, mas em vez disso assume que pontos correspondentes são pontos próximos. Para obter a proximidade entre dois pontos, basta calcular sua distância Euclidiana:

$$d_{Euc}(\mathbf{p^A}, \mathbf{p^B}) = \sqrt{(p_x^\mathbf{B} - p_x^\mathbf{A})^2 + (p_y^\mathbf{B} - p_y^\mathbf{A})^2 + (p_z^\mathbf{B} - p_z^\mathbf{A})^2}, \tag{3.5}$$

onde $\mathbf{p}_i^\mathbf{A} = (p_x^\mathbf{A}, p_y^\mathbf{A}, p_z^\mathbf{A}, 1)^\mathrm{T}$ é um ponto de $\mathbf{S_A}$ e $\mathbf{p}_j^\mathbf{B} = (p_x^\mathbf{B}, p_y^\mathbf{B}, p_z^\mathbf{B}, 1)^\mathrm{T}$ é um ponto de $\mathbf{S_B}$. Apesar dessa aproximação ser pouco precisa, isso torna o problema uma questão de minimização de todas as distâncias dos pontos correspondentes. Para reduzir essas distâncias, é necessário transformar $\mathbf{S_B}$, ou seja, encontrar uma transformação linear $\mathcal{T}$ por meio de uma matriz de transformação $\mathbf{M}$ até que ela fique alinhada com $\mathbf{S_A}$. Essa transformação é descrita por:

$$\mathcal{T}(\mathbf{p}_j^\mathbf{B}) = \mathbf{M}\mathbf{p}_j^\mathbf{B}. \tag{3.6}$$

Para calcular a distância Euclidiana entre o par de pontos correspondentes $\{\mathbf{p}_i^\mathbf{A}, \mathbf{p}_j^\mathbf{B}\}$ faz-se:

$$\mathbf{p}_i^\mathbf{A} - \mathcal{T}(\mathbf{p}_j^\mathbf{B}) = \mathbf{p}_i^\mathbf{A} - \mathbf{M}\mathbf{p}_j^\mathbf{B}. \tag{3.7}$$

Considerando $\mathbf{n}_j = (n_{xj}, n_{yj}, n_{zj}, 0)^{\mathrm{T}}$ como o vetor normal de $\mathbf{p}_j^{\mathbf{B}}$, é possível calcular o erro de alinhamento de todos os pontos do conjunto considerado por meio da função de erro $f(\mathbf{M})$:

$$f(\mathbf{M}) = \frac{1}{N}\sum_{i=1}^{N}((\mathbf{p}_i^{\mathbf{A}} - \mathbf{M}\mathbf{p}_j^{\mathbf{B}}) \cdot \mathbf{n}_j)^2. \tag{3.8}$$

O resultado de $f(\mathbf{M})$ é o que ICP visa minimizar com cada iteração. Quando o erro atingir um valor mínimo aceitável, o ICP considera que o alinhamento foi calculado com sucesso.

O algoritmo utilizado para calcular a matriz de transformação $\mathbf{M}$ é chamado *Point-to-plane ICP Algorithm* [27]. Seu objetivo é fazer uma aproximação linear de modo a encontrar a transformação que resulte no menor erro de alinhamento possível:

$$argmin_{\mathbf{M}} f(\mathbf{M}) = \frac{1}{N}\sum_{i=1}^{N}((\mathbf{p}_i^{\mathbf{A}} - \mathbf{M}\mathbf{p}_j^{\mathbf{B}}) \cdot \mathbf{n}_j)^2, \tag{3.9}$$

e como mostrado no Apêndice A, a matriz $\mathbf{M}$

$$\mathbf{M} = \begin{bmatrix} \cos\beta \cdot \cos\gamma & \cos\beta \cdot \sin\gamma & -\sin\beta & t_x \\ \sin\alpha \cdot \sin\beta \cdot \cos\gamma - \cos\alpha \cdot \sin\gamma & \sin\alpha \cdot \sin\beta \cdot \sin\gamma + \cos\alpha \cdot \cos\gamma & \sin\alpha \cdot \cos\beta & t_y \\ \cos\alpha \cdot \sin\beta \cdot \cos\gamma + \sin\alpha \cdot \sin\gamma & \cos\alpha \cdot \sin\beta \cdot \sin\gamma - \sin\alpha \cdot \cos\gamma & \cos\alpha \cdot \cos\beta & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

é composta de seis parâmetros α, β, γ, t_x, t_y e t_z. Como α, β e γ são argumentos de funções trigonométricas não lineares, não é possível encontrar essa solução por meio da técnica de mínimos quadrados. Sendo assim, é necessário fazer uma aproximação linear desse problema. Se $\theta \approx 0$, é possível aproximar $\sin(\theta) \approx 0$ e $\cos(\theta) \approx 1$. Então, para o caso de $\alpha, \beta, \gamma \approx 0$ temos:

$$\mathbf{M} = \begin{bmatrix} 1 & \gamma & -\beta & t_x \\ \alpha\beta - \gamma & \alpha\beta\gamma + 1 & \alpha & t_y \\ \alpha\gamma + \beta & \beta\gamma - \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.11}$$

o que resulta em $\hat{\mathbf{M}}$:

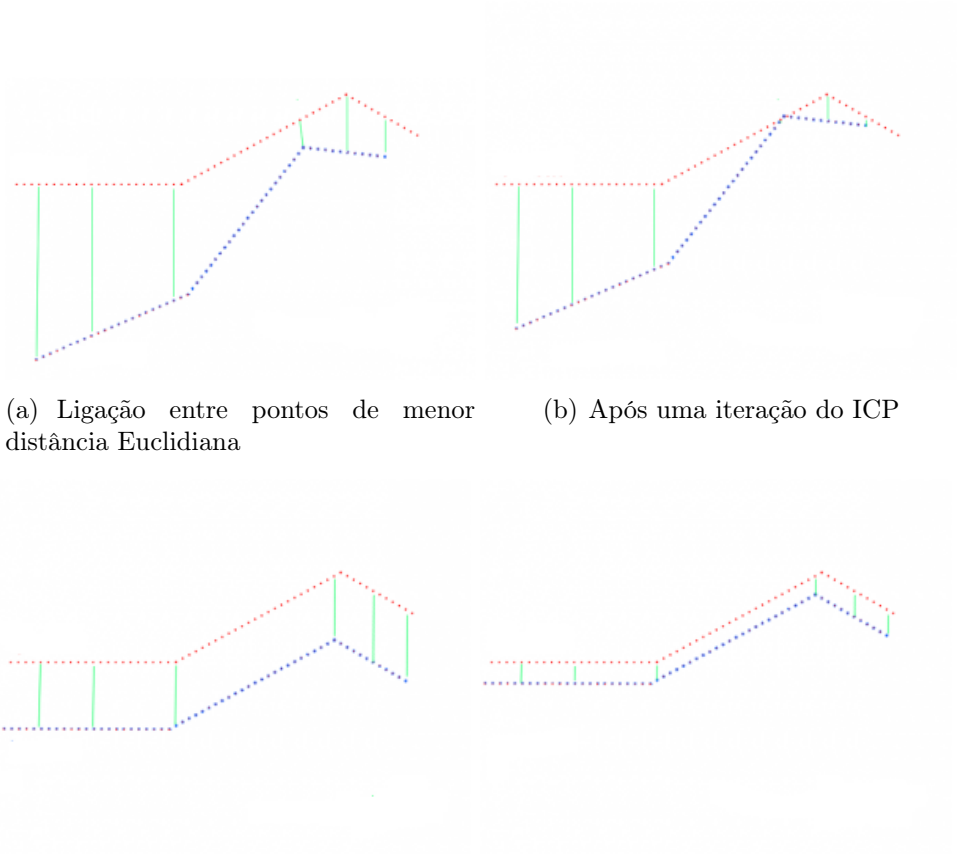
$$\hat{\mathbf{M}} = \begin{bmatrix} 1 & \gamma & -\beta & t_x \\ -\gamma & 1 & \alpha & t_y \\ \beta & -\alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.12}$$

e assim podemos redefinir o erro, que no algoritmo é definido pela constante ***fitness***:

$$\textbf{\textit{fitness}} = argmin_{\hat{\mathbf{M}}} f(\hat{\mathbf{M}}) = \frac{1}{N}\sum_{i=1}^{N}((\mathbf{p}_i^{\mathbf{A}} - \hat{\mathbf{M}}\mathbf{p}_j^{\mathbf{B}}) \cdot \mathbf{n}_j)^2, \tag{3.13}$$

e que agora é um problema de mínimos quadrados padrão, podendo ser solucionado por decomposição em valores singulares (SVD).

Como visto, a intenção do *loop* do ICP é de sempre reduzir as distâncias Euclidianas entre os pontos correspondentes. O resultado final ideal são os pontos correspondentes estarem juntos após a última iteração. A Figura 3.4 mostra um exemplo de como seria esse processo.



(a) Ligação entre pontos de menor distância Euclidiana

(b) Após uma iteração do ICP

(c) Após duas iterações do ICP

(d) Após três iterações do ICP

Figura 3.4: Etapas do alinhamento entre as nuvens feito pelo ICP [6].

O ICP possui diversas implementações disponíveis, inclusive como função da PCL, cuja versão do ICP é a *Point-to-point*. Tendo isso em vista, foi escolhida uma implementação da versão *Point-to-plane*, pois é mais precisa e mais rápida.

Como o algoritmo consome a maior parte do tempo de processamento de cada quadro, é essencial encontrar os valores adequados para os parâmetros do algoritmo, que variam de acordo com o computador utilizado, sendo que computadores de maior custo e consequentemente maior desempenho permitem parâmetros mais precisos. Alguns desses parâmetros são a constante ***max_ iterations***, que determina quantidade máxima de iterações, e a constante ***fitness***, que determina o erro mínimo do cálculo de alinhamento. Esses valores devem permitir que o programa não demore tanto tempo nessa etapa e que também não apresente baixa precisão.

3.1.3 Atualização da nuvem de pontos

Após a transformação entre as duas últimos nuvens de pontos $\mathbf{S_{n-1}}$ e $\mathbf{S_n}$ ter sido calculada, o próximo passo é registrar os pontos não registrados de $\mathbf{S_n}$. Então, considerando p_i como um ponto de $\mathbf{S_n}$, $\mathbf{p}_j$ como um ponto de $\mathbf{S_{n-1}}$, e o par de pontos correspondentes $\{\mathbf{p}_i, \mathbf{p}_j\}$, o conjunto de pontos de $\mathbf{S_n}$ que pode ser registrado é:

$$\sum_{i=1}^{N} \mathbf{p}_i, \nexists\{\mathbf{p}_i, \mathbf{p}_j\}, \tag{3.14}$$

ou seja, todos os pontos de $\mathbf{S_n}$ que não possuem correspondentes em $\mathbf{S_{n-1}}$.

Como o ICP trata pontos correspondentes como pontos de menor distância Euclidiana, o processo de registrar novos pontos funciona de maneira similar. Só que como as nuvens agora estão alinhadas, assumir que pontos próximos são correspondentes é mais válido. Então é necessário determinar uma distância máxima entre os dois pontos correspondentes, distância essa que vai determinar a densidade de pontos adicionados na nuvem. Essa distância é uma constante chamada ***maximum_squared_distance***. Assim, dois pontos serão considerados correspondentes quando:

$$\{\mathbf{p}_i, \mathbf{p}_j\} \rightarrow d_{Euc}(\mathbf{p}_i, \mathbf{p}_j) \leq \textbf{\textit{maximum\_squared\_distance}}, \tag{3.15}$$

de modo que quanto menor o valor de ***maximum_squared_distance*** maior a densidade de pontos da nuvem final. Foi determinado experimentalmente que seu valor ideal é 50.

3.2 Processamento

A etapa de processamento visa melhorar o aspecto final da nuvem de pontos registrada durante a aquisição. São utilizados dois filtros disponibilizados pela PCL, um para filtrar pontos ruidosos e um para suavizar superfícies. Em seguida, é realizada a reconstrução superficial de modo a gerar uma malha de polígonos, e então é feita a aplicação de texturas sobre essa malha. A Figura 3.5 ilustra esse processo.

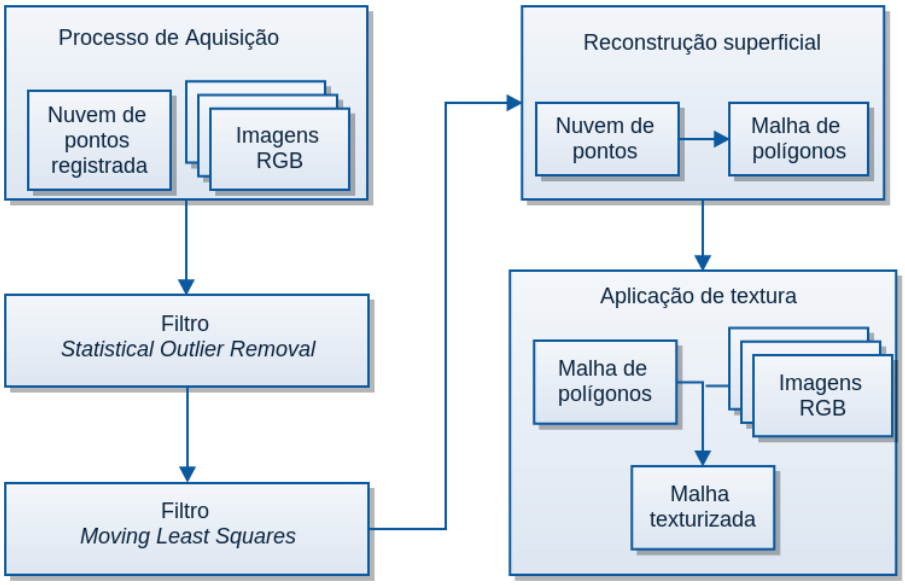


Figura 3.5: Diagrama de blocos.

O filtro de suavização de superfície impede que a malha de polígonos, formada a partir da nuvem de pontos, apresente aspecto ruidoso. Já o filtro de eliminação de pontos ruidosos remove os pontos ruidosos próximos a regiões de formato complexo, como nariz e orelha, além de regiões de borda.

3.2.1 Filtro *Statistical Outlier Removal* (SOR)

Este filtro utiliza estatísticas sobre pontos da vizinhança para filtrar dados *outliers*. Seu algoritmo itera sobre a nuvem de pontos duas vezes. Na primeira iteração, é calculada a distância de cada ponto para os seus K vizinhos próximos, sendo K um valor a ser definido pelo usuário por meio da constante ***max_neighbors***. Considerando $\mathbf{p}_i$ como um ponto da nuvem a ser iterada e $\mathbf{p}_j$ um vizinho próximo, temos:

$$d_{Euc}(\mathbf{p}_i, \mathbf{p}_j), j \in \{0...K\}. \tag{3.16}$$

Em seguida, é calculada a média dessas distâncias $\mu_{\mathbf{p}_i}$:

$$\mu_{\mathbf{p}_i} = \frac{1}{K} \sum_{j=1}^{K} d_{Euc}(\mathbf{p}_i, \mathbf{p}_j), \tag{3.17}$$

e o desvio padrão $\sigma_{\mathbf{p}_i}$:

$$\sigma_{\mathbf{p}_i} = \frac{1}{K} \sqrt{\sum_{j=1}^{K} (d_{Euc}(\mathbf{p}_i, \mathbf{p}_j) - \mu_{\mathbf{p}_i}),} \tag{3.18}$$

para que possa se encontrar o limiar de distância *dist_threshold*:

$$dist\_threshold = \mu_{\mathbf{p}_i} + \sigma_{\mathbf{p}_i} L, \tag{3.19}$$

sendo L também um multiplicador definido pelo usuário.

Na segunda iteração, serão considerados *outliers* os pontos cuja distância de vizinhança média estão acima do limiar *dist_threshold*. Esses outliers serão removidos no fim da iteração. A Figura 3.6 mostra um exemplo de nuvem de pontos antes e depois da filtragem. Os pontos ruidosos com grande espaçamento próximo são eliminados, e a imagem fica com um aspecto mais limpo.

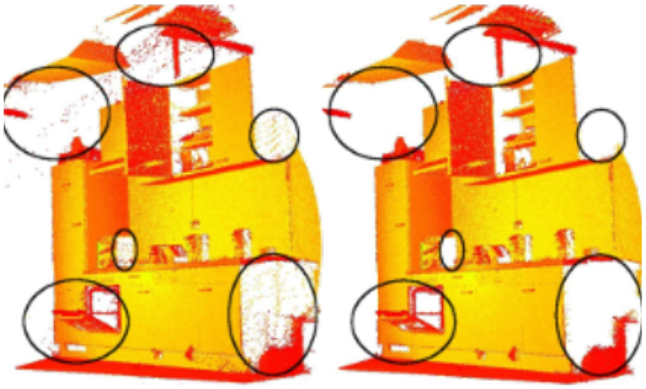


Figura 3.6: Exemplo de cena processada pelo filtro *Statistical Outlier Removal* [7].

3.2.2 Filtro *Moving Least Squares* (MLS)

Sensores são dispositivos que convertem medidas físicas em sinais analógicos ou digitais. Todo sensor possui um erro associado, variando de acordo com sua precisão. No caso do sensor de profundidade do Microsoft Kinect, o caso é o mesmo, e apesar de superfícies serem facilmente vistas por ele, os pontos que às definem possuem a medida de profundidade com um erro associado. Esse erro cria irregularidades na superfície que são muito difíceis de remover por análise estatística. Nesse caso, uma solução é usar algoritmos de reamostragem, que tentam recriar as superfícies por meio de polinômios de interpolação de maior ordem em um conjunto de pontos próximos. Ao efetuar a reamostragem, esses pequenos erros podem ser corrigidos e as superfícies são suavizadas.

A Figura 3.7 mostra um exemplo do uso do filtro MLS. Na imagem da esquerda, é possível ver o resultado do alinhamento entre duas nuvens de pontos juntas. Devido aos erros associados, as normais resultantes apresentam ruídos. A imagem da direita mostra as normais estimadas no mesmo conjunto de dados depois das superfícies serem suavizadas com o algoritmo *Moving Least Squares*.

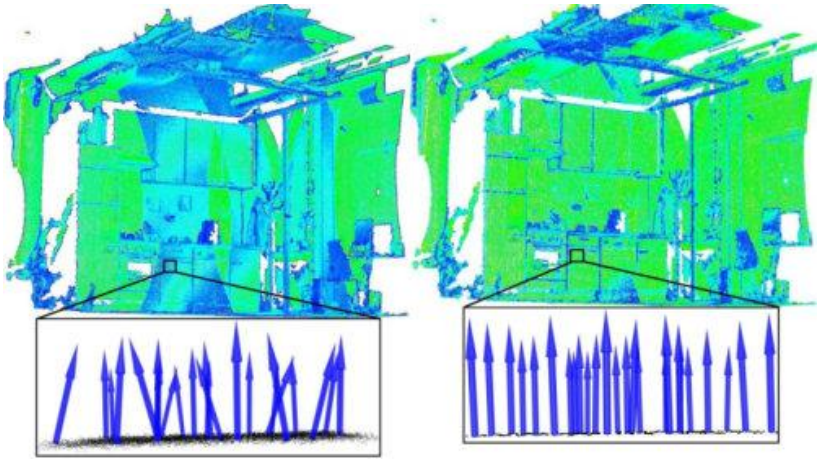


Figura 3.7: Exemplo de cena processada pelo filtro *Moving Least Squares* [7].

Definição

Considere uma função $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ e uma amostra de pontos $S = \{(x_i, f_i)|f(x_i) = f_i)\}$ onde $x_i \in \mathbb{R}^n$ e os f_i são números reais. O objetivo da apromixação de mínimos quadros (*least squares*, LS) é encontrar uma função $p(x)$, definida globalmente, que seja o mais próximo possível de f_i para cada x_i. O erro funcional entre $p(x)$ e f_i deve ser o menor possível:

$$\min_{f \in \Pi_m^d} \sum_i ||p(x) - f_i||^2, \tag{3.20}$$

onde p pertence a Π_m^d, o espaço polinomial de grau m em d dimensões.

Ao considerar os vizinhos x_i de cada ponto x, podemos obter uma aproximação de mínimos quadrados localmente ao considerarmos um peso $\theta(s)$ associado às distâncias euclidianas entre x e seus vizinhos x_i (determinadas pela constante ***search_radius***), de modo que $\theta(s)$ tende a zero à medida que $s \longrightarrow \infty$:

$$\min_{f \in \Pi_m^d} \sum_i \theta(||x - x_i||)||p(x) - f_i||^2, \tag{3.21}$$

que se chama aproximação de mínimos quadrados com peso associado (*weighted least squares*, WLS).

Já o método chamado aproximação movendo mínimos quadrados (*moving least squares*, Figura 3.8) parte de uma aproximação WLS para um ponto fixo, e então move esse ponto para todo o domínio Ω, resultando em uma função MLS. O objetivo é encontrar uma função global $p_x(x)$ obtida a partir de um conjunto de funções locais $p_x(x_i)$:

$$p_x(x) = \min_{f \in \Pi_m^d} \sum_i \theta(||x - x_i||)||p_x(x_i) - f_i||^2. \tag{3.22}$$

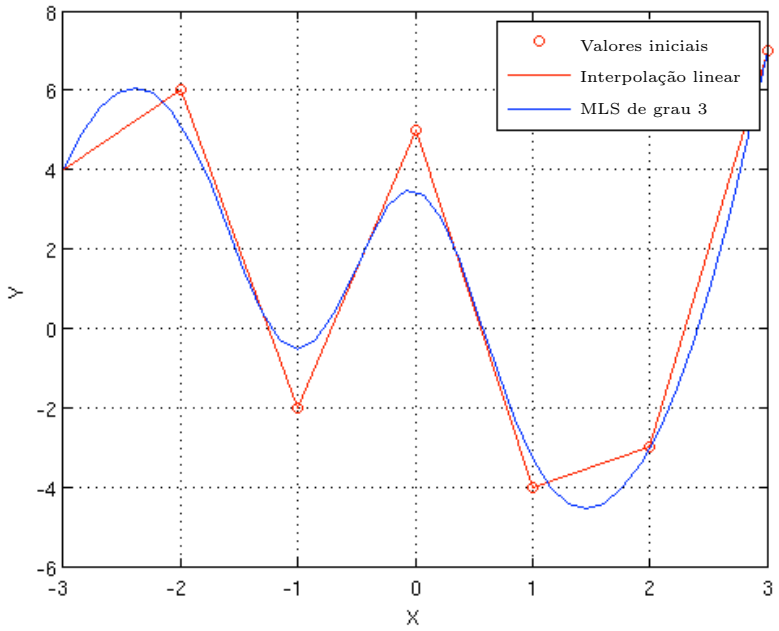


Figura 3.8: Exemplo da aproximação *moving least square* [8].

3.3 Reconstrução superficial

Reconstrução superficial é um método que realiza triangulação entre os pontos de uma nuvem de modo a formar uma malha de polígonos (Figura 3.9). A triangulação é feita localmente por meio da projeção da vizinhança local de um ponto ao longo da sua normal, conectando pontos desconectados. O resultado é melhor se a superfície for suave e houver transições suaves entre áreas com densidade de pontos diferentes.

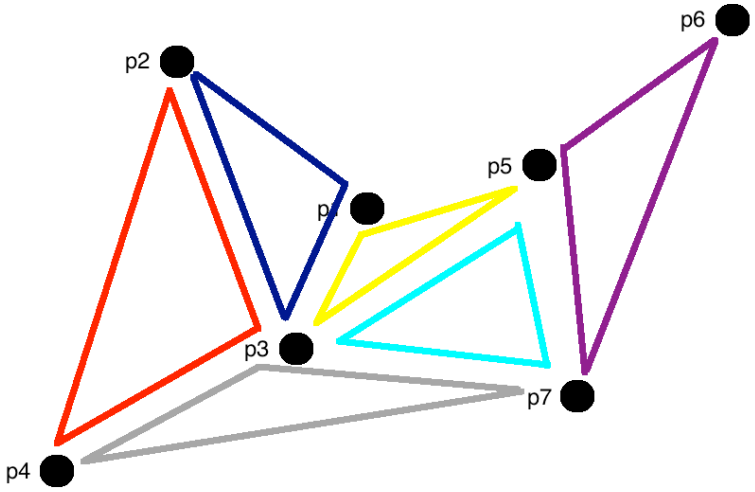
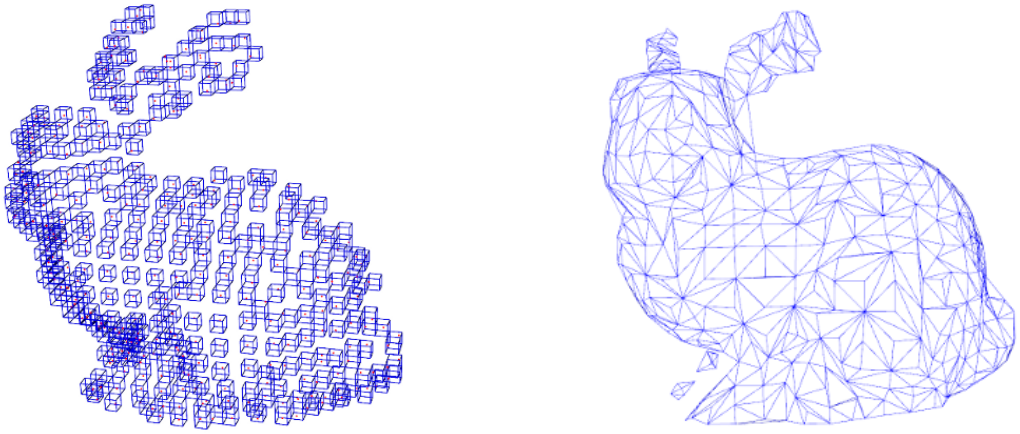


Figura 3.9: Triangulação entre pontos vizinhos

O algoritmo escolhido para a reconstrução superficial foi o *Greedy Projection Triangulation*, pois seu tempo de processamento é consideravelmente baixo e por não alterar a superfície geométrica da nuvem de pontos, apesar de apresentar buracos no resultado final.

Esse algoritmo funciona mantendo uma lista de pontos possíveis de serem conectados, para onde a malha de polígonos crescerá. Esse método estende a malha enquanto houverem pontos disponíveis. O algoritmo pode receber nuvens de pontos não organizadas (nuvens de pontos que sofreram remoção ou adição de pontos) originadas de múltiplos escaneamentos e com várias partes conectadas. Os melhores resultados ocorrem em situações onde a nuvem processada não apresenta grande variação de densidade de pontos. Ele recebe parâmetros relativos ao número de vizinhos de cada ponto, o tamanho máximo possível dos lados dos triângulos, os valores máximo e mínimo dos ângulos dos triângulos, entre outros.

A Figura 3.10 mostra o resultado da reconstrução superficial da nuvem de pontos de um coelho usando esse algoritmo.



(a) Nuvem de pontos envolvidos por voxels (b) Malha de polígonos construída a partir dos pontos

Figura 3.10: Reconstrução superficial de uma nuvem de pontos de um coelho [6].

3.4 Aplicação de textura

Após a construção da malha de polígonos, os pontos se tornam vértices de triângulos. Esse triângulos possuem uma face com cor relativa à cor dos seus vértices, o que resulta em um aspecto visual equivalente ao da nuvem de pontos. Só que, como o rosto deve ser movimentado livremente durante o processo de registro, pontos próximos podem ser registrados com diferentes luminosidades. Isso resulta em áreas com cor não uniforme, e o aspecto do modelo 3D fica prejudicado. Sendo assim, optou-se pela alternativa de aplicar texturas sobre a malha de polígonos.

A textura nada mais é que uma imagem RGB registrada pelo sensor de profundidade. Como toda imagem RGB do sensor é associada à uma nuvem de pontos, a ideia é que todo conjunto de pontos registrados pelo programa sejam associados à sua própria textura. Sendo assim, para n nuvens registradas, devem haver n texturas salvas para que possam ser aplicadas caso necessário.

Apesar do algoritmo salvar uma imagem para cada pose, não chega a ser necessário aplicar todas as texturas salvas durante o processo de aquisição. Uma textura se aplica à todas as faces orientadas para o ponto de vista da câmera, então faces de poses diferentes podem ser preenchidas com uma mesma textura de acordo com as suas normais. O conjunto de faces que não forem orientadas para esse mesmo ponto de vista serão preenchidas com uma outra textura que satisfaça essa condição. Foi determinado experimentalmente que três imagens são suficientes para esse processo.

Capítulo 4

Resultados

Nesta seção serão mostrados os resultados obtidos em cada etapa da implementação, bem como variações das constantes empíricas e seus respectivos resultados. Assim será possível verificar se as escolhas dos algoritmos foram adequadas e se o objetivo do trabalho foi cumprido.

4.1 Uso do programa

O programa inicia com o processo de aquisição de imagens RGBD, mostrando a nuvem registrada em uma janela do Terminal. Nesta etapa, basta que o usuário movimente o rosto livremente em frente à câmera para que o programa possa preencher com pontos toda a região da face. Ao verificar na tela do computador que a nuvem de pontos está adequada, o usuário aperta uma tecla para concluir a aquisição e o programa inicia o pós processamento.

A Figura 4.1 mostra o programa em funcionamento. Nela é possível ver o estado atual da nuvem registrada à direita, e à esquerda os tempos de processamento das etapas no computador utilizado para este trabalho:

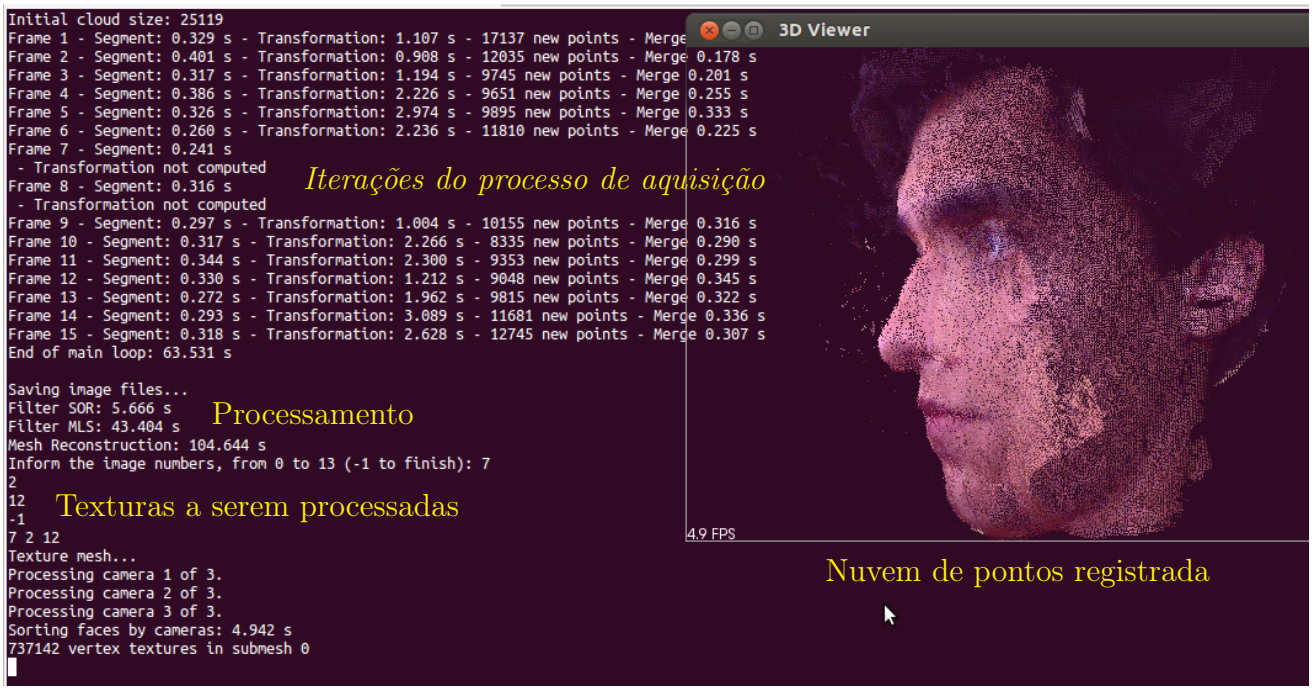


Figura 4.1: Programa em execução.

Podemos verificar também que, durante as iterações do processo de aquisição, o programa não conseguiu calcular a transformação nos quadros 7 e 8. Isso ocorre quando o algoritmo não consegue calcular a transformação corretamente. Quando isso acontece, o usuário deve tentar reestabelecer a última pose válida para que o algoritmo possa continuar o processo. Apesar de perder o rastreamento do objeto, o algoritmo não falha nessa situação.

Após terminado o pós-processamento, o usuário deve informar quantas e quais imagens deseja usar como textura. A ideia é que o usuário faça a avaliação subjetiva das melhores imagens RGB capturadas pelo sensor, e que escolha o menor número possível. Foi determinado experimentalmente que quanto menos imagens, mais homogêneo será o resultado, pois as imagens apresentam variação de luminosidade no rosto.

A Figura 4.1 também mostra os tempos de processamento de cada uma das etapas do algoritmo com o programa sendo executado no computador utilizado neste trabalho. Durante a aquisição, é possível verificar que o cálculo da transformação é a etapa mais longa, assim como a reconstrução superficial é a etapa mais exigente do processamento.

4.2 Aquisição

4.2.1 Pré-processamento

O pré-processamento é constituído de duas etapas: segmentação de uma região de interesse e o cálculo das normais de superfície. A segmentação de uma região de interesse é necessária pois quaisquer pontos fora dessa região irão interferir no ICP. Além disso,

a segmentação diminui a quantidade de pontos a serem processados, o que acelera consideravelmente o processamento. As normais de superfície são utilizadas para verificar qual a direção dos pontos em relação à camera, informação que é útil nos processamentos em sequência.

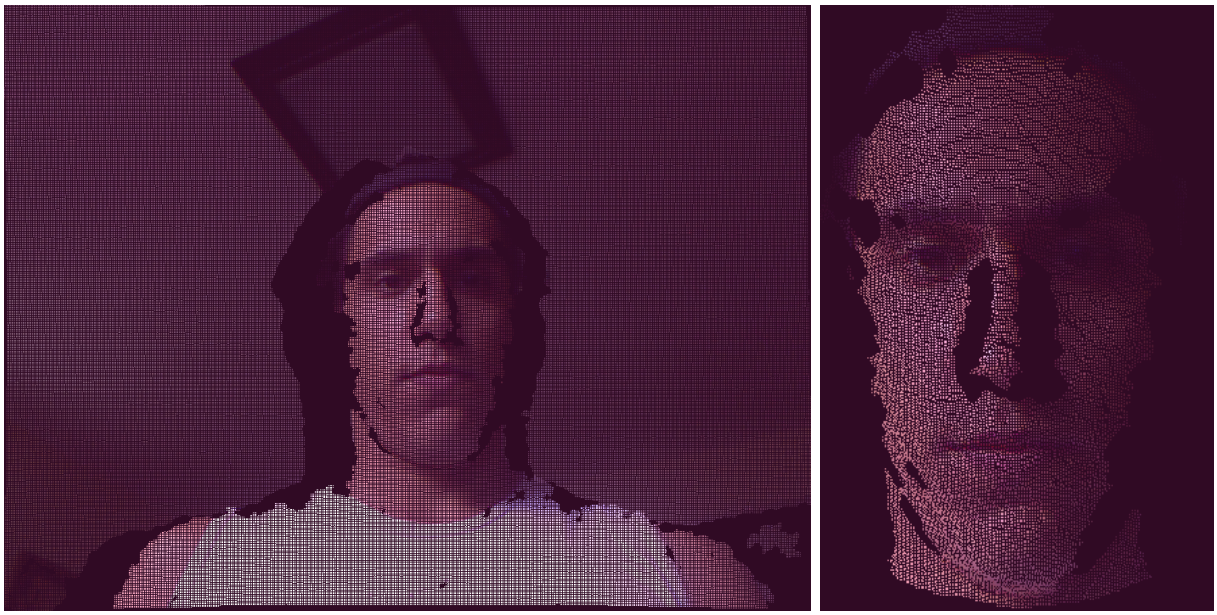
Normais de superfície

As normais de superfície são utilizadas em todas as etapas seguintes portanto são estimadas no início da iteração.

Segmentação

A Figura 4.2 mostra o resultado da segmentação da nuvem na região próxima ao sensor, onde deve se posicionar a face a ser registrada. A segmentação elimina os pontos do ambiente e do corpo da pessoa que não serão registrados, restando apenas os pontos correspondentes à face. Observando a Figura 4.2 (a), é possível ver que:

1. Como o Kinect possui somente um emissor de luz infra-vermelha, superfícies mais próximas do emissor deixarão sombras nas superfícies mais distantes. No caso deste trabalho isso não faz diferença, mas em aplicações que necessitam dessas informações, somente um quadro não seria suficiente.
2. O algoritmo de segmentação remove os pontos cuja normal possui ângulo de inclinação muito grande em relação à câmera. Sabendo que a normal é calculada levando em conta os pontos vizinhos, o resultado disso é que haverão buracos na nuvem do rosto como consequência, principalmente na região do nariz. Isso torna o algoritmo mais robusto aos dados do sensor, uma vez que o Kinect possui baixa precisão.



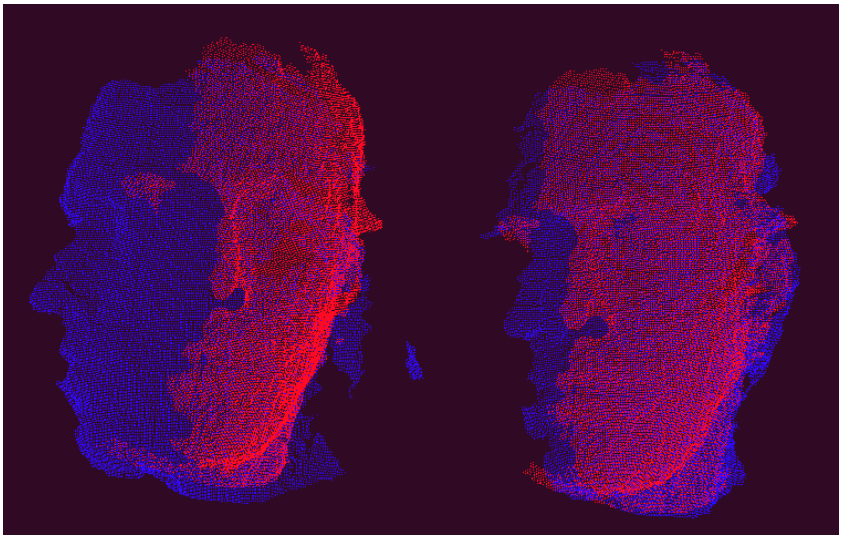
(a) Nuvem de pontos (b) Nuvem de pontos segmentada

Figura 4.2: Resultado da segmentação da região da face

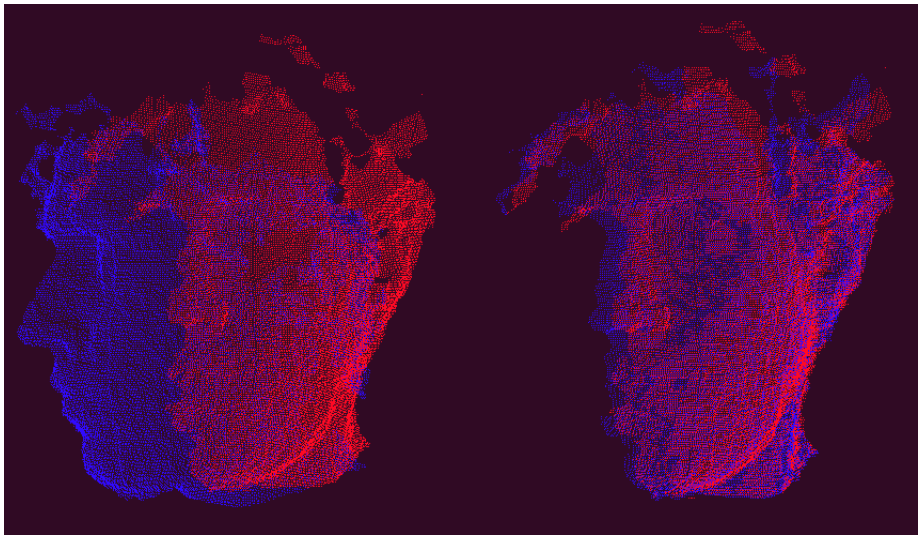
4.2.2 Transformação

O objetivo do ICP é alinhar duas nuvens de pontos desalinhadas (transformadas afim entre si), e o seu resultado é uma matriz de transformação entre elas. Sendo assim, é possível analisar o alinhamento ao transformar uma das nuvens e somá-las, verificando visualmente o resultado e o nível de precisão do alinhamento entre as nuvens. Esse nível de precisão é determinado pela constante ***fitness***, de modo que um valor mais alto permite menor precisão, que resulta em mais pontos ruidosos e maior rapidez, e um valor menor exige maior precisão, com menos ruídos e mais tempo de processamento. No nosso caso, os erros de precisão são compensados pelos filtros utilizados em sequência.

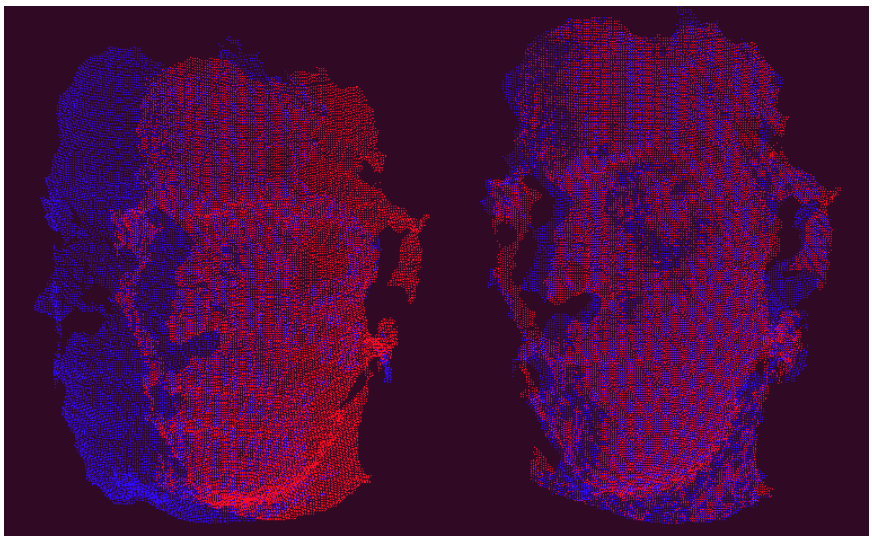
A Figura 4.3 mostra os resultados dos alinhamentos obtidos ao longo do processo de registro facial. As Figuras 4.2 (a), 4.2 (b) e 4.2 (c) possuem valores de ***fitness*** de 10^{-4}, 10^{-5} e 10^{-6} respectivamente. Enquanto a Figura 4.2 (a) mostra que um valor muito alto permite um erro de alinhamento considerável, a Figura 4.2 (c) mostra que um valor muito baixo não produz um resultado significativamente melhor. Sendo assim, o valor que utilizamos para ***fitness*** é de 10^{-5}.



(a) ***fitness*** $= 10^{-}4$



(b) ***fitness*** $= 10^{-}5$



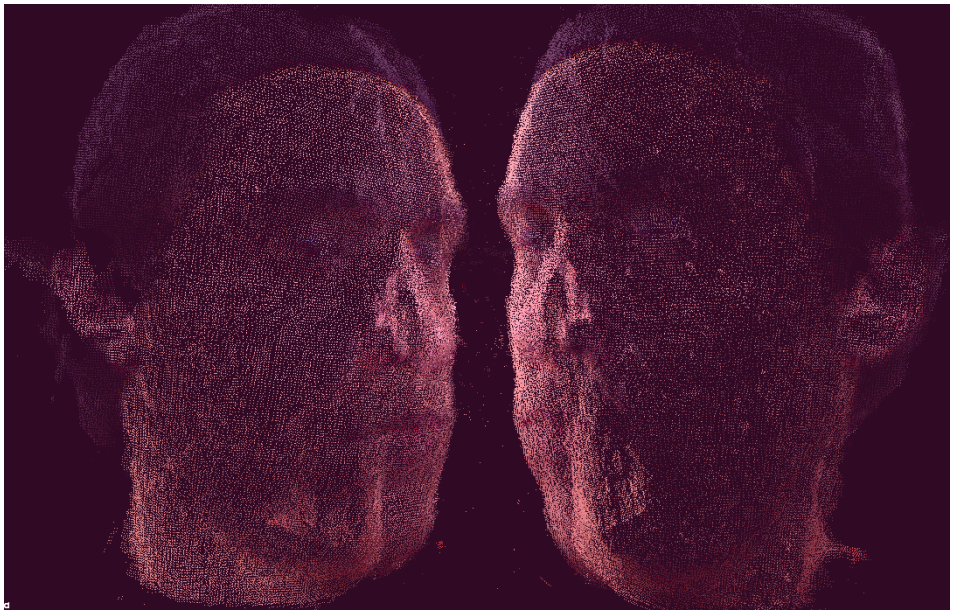
(c) ***fitness*** $= 10^{-}6$

Figura 4.3: Resultados dos alinhamentos do algoritmo ICP

4.2.3 Nuvem de pontos registrada

O resultado final do processo de aquisição é uma nuvem de pontos representando o objeto registrado, que nesse caso é a face humana. O processo de registrar novos pontos, como explicado na Seção 3.1.3, é determinado pela constante de distância máxima entre os pontos ***maximum_squared_distance***. Quanto maior essa constante, mais espaçados os pontos ficarão e menor será a densidade da nuvem final, e a mesma lógica se aplica para o caso contrário. Foi determinado experimentalmente que o melhor caso ocorre quando ***maximum_squared_distance*** possui o valor de 50, pois valores acima disso atrapalham a reconstrução superficial, enquanto valores menores dão resultados visualmente equivalentes mas consomem mais tempo de processamento.

As Figuras 4.4 (a) e (b) mostram dois resultados do processo de aquisição com ***maximum_squared_distance*** igual a 50 e 5 respectivamente. A nuvem de pontos da Figura 4.3 (b) possui o dobro da quantidade de pontos da nuvem de pontos da Figura 4.3 (a), resultando em uma visível diferença de densidade de pontos. Apesar disso, o resultado final que visamos alcançar é baseado em polígonos construídos a partir dos pontos, de modo que polígonos com lados de em média 50 geram resultados satisfatórios. A quantidade de quadros necessários para o registro facial não possui métrica associada, podendo variar de acordo com o usuário em questão.



(a) $\boldsymbol{maximum\_squared\_distance} = 50$

(b) $\boldsymbol{maximum\_squared\_distance} = 5$

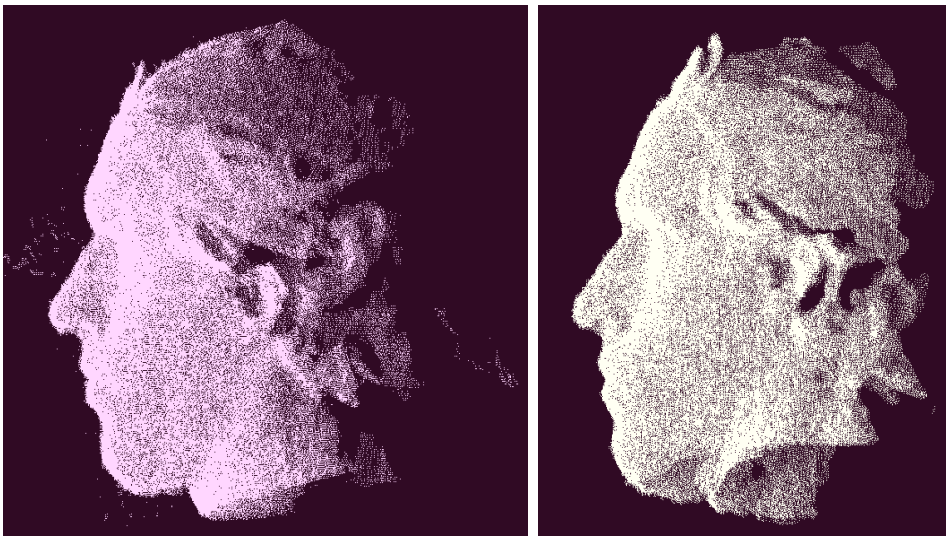
Figura 4.4: Nuvem de pontos representando uma face humana

4.3 Processamento

O processamento permite melhorar o aspecto visual da nuvem de pontos para que a reconstrução superficial atinja um resultado satisfatório. Serão mostradas variações dos parâmetros usados pelos filtros de modo a comparar diferentes resultados e justificar o valor escolhido para esses parâmetros.

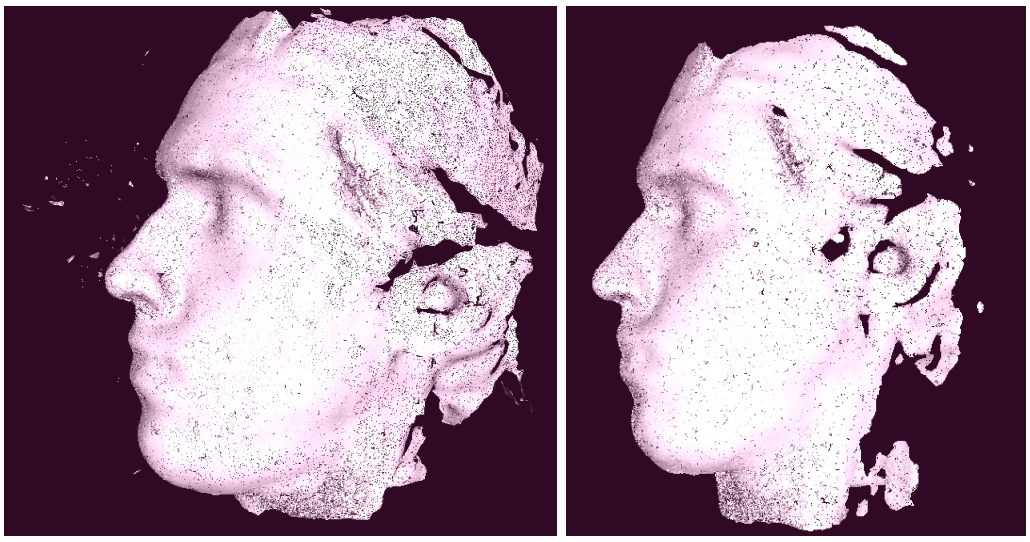
4.3.1 Filtro *Statistical Outlier Removal*

O filtro *Statistical Outlier Removal* se baseia na constante ***max_neighbors***, que determina o números de vizinhos de cada ponto como parâmetro para filtrar os pontos *outliers*, como explicado na Seção 3.2.1.. Seu resultado é eficaz com ***max_neighbors*** = 100 vizinhos. As Figuras 4.6 e 4.7 mostram os resultados com e sem uso desse filtro.



(a) Nuvem de pontos não filtrada (b) Nuvem de pontos filtrada

Figura 4.5: Resultados com e sem o uso do filtro *Statistical Outlier Removal*

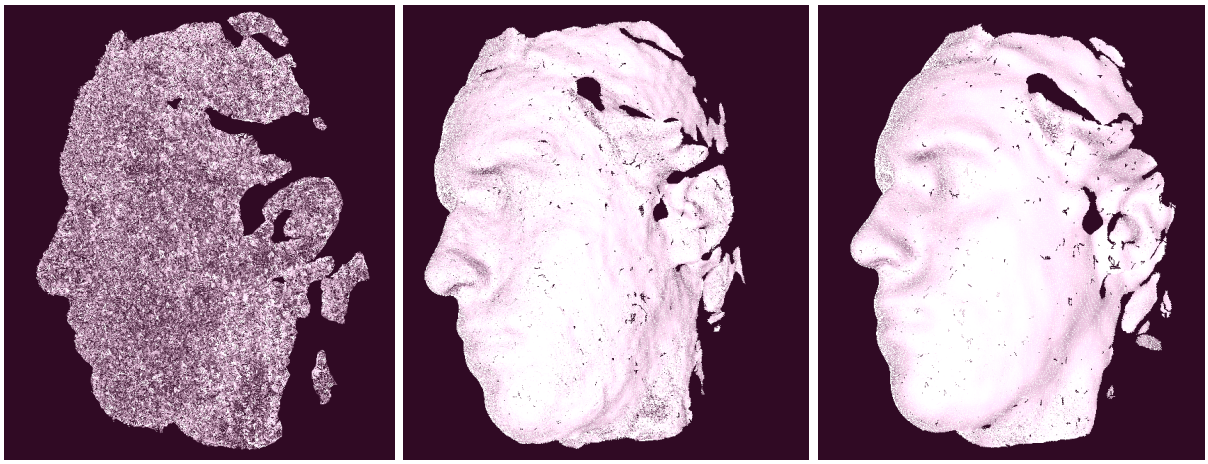


(a) Malha não filtrada (b) Malha filtrada

Figura 4.6: Resultados com e sem o filtro *Statistical Outlier Removal*

4.3.2 Filtro *Moving Least Squares*

O resultado que o filtro *Moving Least Squares* cria na aparência final da malha de polígonos é visivelmente significante, como mostra a Figura 4.8. Sem o seu uso, não seria possível obter resultados satisfatórios com o programa implementado. A constante ***search_radius***, que define o raio de busca do algoritmo, determina o aspecto final da malha de polígonos, de modo que seu valor deve estar de acordo com a escala da dimensão do objeto sendo escaneado. No nosso caso, ***search_radius*** = 1,0 produziu os melhores resultados.

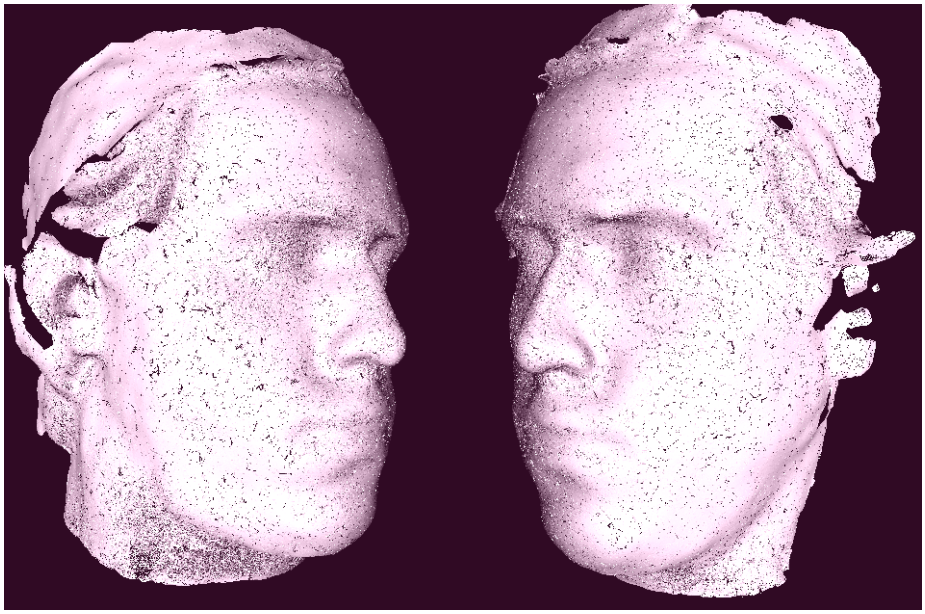


(a) Malha não filtrada (b) Malha filtrada com raio de busca de 0,5 (c) Malha filtrada com raio de busca de 1,0

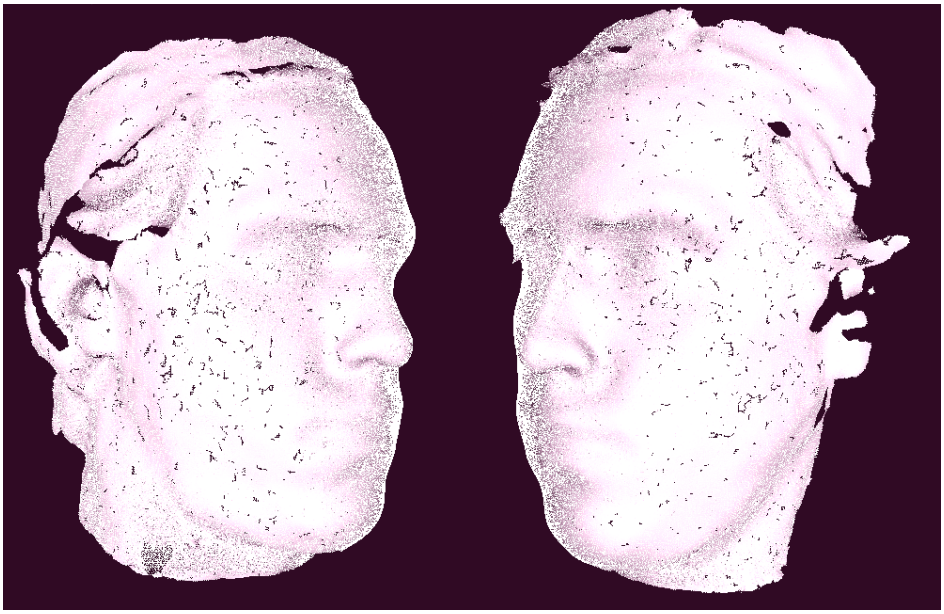
Figura 4.7: Resultados do filtro *Moving Least Squares*

4.4 Reconstrução superficial

A Figura 4.5 mostra o resultado do algoritmo de reconstrução de superfície *Greedy Projection Triangulation*. É possível concluir que ele não produz um resultado ideal, pois deixa pequenos buracos ao longo da malha de polígonos.



(a) Malha com representação em esqueleto, onde somente os lados dos triângulos são vísiveis



(b) Malha com representação em superfície

Figura 4.8: Resultado do algoritmo de reconstrução superficial

4.5 Aplicação de textura

A Figura 4.9 mostra o resultado final obtido com o programa que é uma malha de polígonos representando uma face humana. A Figura 4.9 (a) mostra esse resultado quando se registra a cor de cada ponto adicionado à nuvem. Esse processo resulta em uma textura com regiões de cor não uniforme, pois a iluminação incidente no rosto é inconstante uma vez que o rosto sofre rotação e translação ao longo da filmagem. A Figura 4.9 (b) mostra

o resultado quando se aplica texturas de imagens RGB sobre a malha de polígonos, o que permite uma apresentação visualmente fiel ao rosto escaneado. A Figura 4.10 mostra outros exemplos de registro facial.

(a) Resultado final com textura original da nuvem de pontos

(b) Resultado final com textura de imagens RGB

Figura 4.9: Resultado final do registro facial

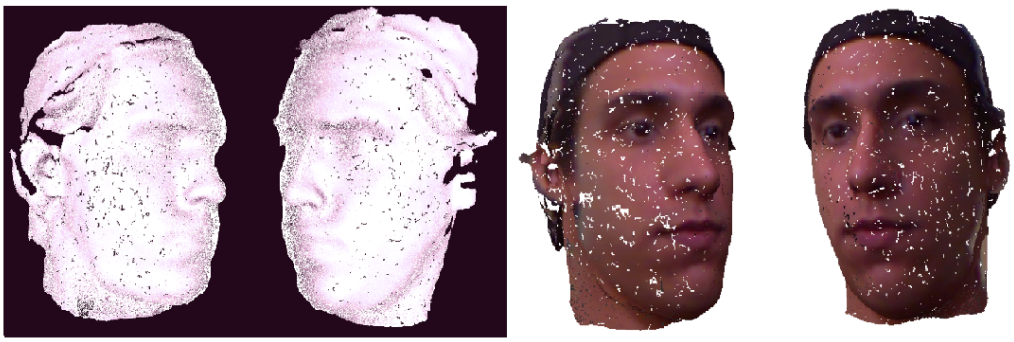
4.5.1 Comparação de resultados

A Figura 4.10 mostra os resultados obtidos por Hernandez, Zollhofer e os obtidos nesse trabalho. O aspectos geométrico e visual se assemelham aos dos rostos escaneados em todos os casos, sendo possível afirmar que todos conseguiram gerar modelos

tridimensionais da face com qualidade aceitável. Todavia, o resultado desse trabalho apresenta buracos, o que não ocorreu nos outros casos,

(a) Resultado obtido por Hernandez [21]

(b) Resultado obtido por Zollhofer [1]

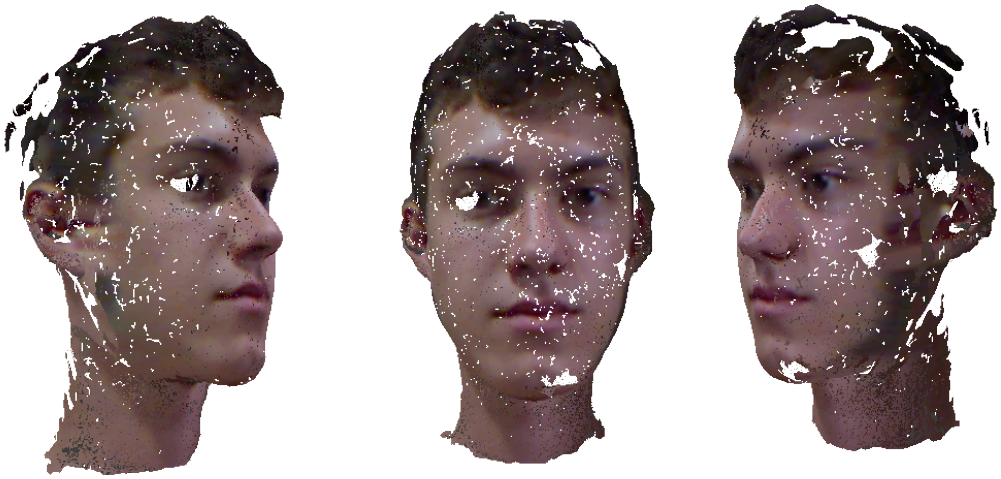


(c) Resultado obtido nesse trabalho

Figura 4.10: Resultado final do registro facial

4.6 Outros resultados

(a)



(b)

Figura 4.11: Outros resultados de registro facial

(a)

(b)

Figura 4.12: Outros resultados de registro facial

(a)

Figura 4.13: Outros resultados de registro facial

Capítulo 5

Conclusão

Por meio deste trabalho, foi possível criar modelos 3D representativos de faces humanas a partir do sensor de baixo custo Microsoft Kinect. Os modelos apresentaram aspectos geométricos e visuais próximos aos rostos humanos representados, e o software, apesar de ser computacionalmente exigente, é utilizável em tempo real em computadores modernos.

Quanto ao processo de aquisição, este apresenta falha ao permitir o registro de pontos outliers à nuvem representativa. Mesmo esse erro sendo corrigido em sequência, não deveria ocorrer. Apesar disso, alguns métodos foram testados para impedir que isso ocorresse mas prejudicaram a velocidade de processamento mesmo apresentando resultados semelhantes.

A reconstrução superficial também deixou a desejar pois deixou buracos na malha de polígonos mesmo no melhor dos resultados. Contudo, muitos programas que manipulam esses tipos de dados possuem algoritmos próprios para resolver esse e outros problemas eventuais.

Finalmente, foi possível implementar com sucesso um algoritmo de escaneamento facial alternativo aos referidos na Seção 2.1, com resultados de qualidade semelhante. Isso possibilita consequentes estudos em reconhecimento facial para modelos tridimensionais. Contudo, é importante notar que as pequenas imperfeições mostram que ainda há o que melhorar nos trabalhos futuros.

5.1 Trabalhos Futuros

Como já mostrado anteriormente, existem algumas falhas no algoritmo que interferem na qualidade do resultado. Algumas delas surgem pois demandam mais processamento, um recurso escasso pois já é muito exigido pelo software. Outras são problemas que não puderam ser resolvidos no escopo desse trabalho. Sendo assim, as possíveis melhorias são:

1. Melhorar o processo de registro de novos pontos de modo a uniformizar melhor a cor e impedir outliers;
2. Sugerir melhoras no algoritmo Greedy Projection Triangulation para evitar buracos na malha;
3. Aumentar a velocidade do software ao implementar alguns dos algoritmos para processamento em GPU.

Referências

[1] M. Zollhofer, M. Martinek, and G. Greiner, "Automatic reconstruction of personalized avatars from 3d face scans," *Computer Animation and Virtual Worlds*, March 2011.

[2] Wikipedia, "Microsoft kinect," October 2013.

[3] M. A. Borrego, "Desarrollo e implementacion de un metodo de generacion de mapas 3d usando el sensor kinect," February 2012.

[4] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, February 2012.

[5] PCL, "Pcl normal estimation," November 2013.

[6] PCL, "Pcl documentation," October 2013.

[7] PCL, "Pcl filters," October 2013.

[8] Wikipedia, October 2013.

[9] G. M. Araujo, "Algoritmo para reconhecimento de caracteristicas faciais baseado em filtros de correlacao," February 2010.

[10] Hokuyo, "Hokuyo," October 2013.

[11] PrimeSense, "Primesense natural interaction," October 2013.

[12] Microsoft, "Microsoft kinect," October 2013.

[13] N. s. A. Berk Gokberk, Albert Ali Salah and L. Akarun, "3d face recognition,"

[14] L. Tang and T. Huang, "Automatic construction of 3d human face models based on 2d images," *ICIP*, pp. 467–470, 1996.

[15] B. Amberg, A. Blake, A. Fitzgibbon, S. Romdhani, and T. Vetter, "Reconstructing high quality face-surfaces using model based stereo," *ICCV*, 2007.

[16] M. Zollhöfer, M. Martinek, G. Greiner, M. Stamminger, and J. Süßmuth, "Automatic reconstruction of personalized avatars from 3d face scans," *Computer Animation and Virtual Worlds*, vol. 22, 2011.

[17] D. Solutions, 2011.

[18] O. Alexander, M. Rogers, W. Lamberth, J.-Y. Chiang, W.-C. Ma, C.-C. Wand, and P. Debevec, "The digital emily project: achieving a photorealistic digital actor," *IEEE Computer Graphics and Applications*, vol. 30, 2010.

[19] D. Bradley, W. Heidrich, T. Popa, and A. Sheffer, "High resolution passive facial performance capture," *IEEE Computer Graphics and Applications*, vol. 30, 2010.

[20] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, J. S. P. Kohli, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," *ISMAR*, 2011.

[21] M. Hernandez, J. Choi, and G. Medioni, "Laser scan quality 3-d face modelling using a low-cost depth camera," *20th European Signal Processing Conference*, 2012.

[22] Wikipedia, October 2013.

[23] DroneMapper, October 2013.

[24] A. Sitek, R. Huesman, and G. Gullberg, "Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud," *Medical Imaging, IEEE Transactions*, vol. 25, pp. 1172–1179, 2006.

[25] J. Leonard and H. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," *Intelligent Robots and Systems '91*, vol. 25, pp. 1442–1447, 1991.

[26] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *International Journal of Computer Vision*, pp. 119—-152, 1994.

[27] K.-L. Low, "Linear least-squares optimization for point-to-plane icp surface registration," *Technical Report TR04-004*, February 2004.

Apêndice A

Matriz de transformação

Em algebra linear, transformações lineares podem ser representadas por matrizes. Se $\mathcal{T}$ é uma transformação linear que mapeia $\Re^n$ para $\Re^m$ e $\mathbf{x}$ é um vetor coluna de n valores, então

$$\mathcal{T}(\mathbf{x}) = \mathbf{A}\mathbf{x} \tag{A.1}$$

para uma matriz de transformação $\mathbf{A}$, mxn. A matriz de transformação $\mathbf{A}$ é uma matriz da forma:

$$\mathbf{A} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}, det(\mathbf{A}) = 1 \tag{A.2}$$

onde $\mathbf{R}$ é uma matriz de rotação e $\mathbf{p}$ é um vetor de translação. Ela é uma matriz $4x4$ que descreve tanto rotação quanto translação. O vetor que descreve uma translação no espaço é da forma:

$$\mathbf{p} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{A.3}$$

enquanto, no sistema de coordenadas $\Re^3$, rotações nos eixos x, y e z são feitos por meio das matrizes:

$$\mathbf{R_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \quad \mathbf{R_y} = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}, \quad \text{e} \quad \mathbf{R_z} = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

.

só que como a rotação pode ser feita nos três eixos ao mesmo tempo, é necessário multiplicar as matrizes de rotação $\mathbf{R_x} \cdot \mathbf{R_y} \cdot \mathbf{R_z}$, sendo o seu produto a matriz de rotação

R:

$$\mathbf{R} = \begin{bmatrix} \cos\beta \cdot \cos\gamma & \cos\beta \cdot \sin\gamma & -\sin\beta \\ \sin\alpha \cdot \sin\beta \cdot \cos\gamma - \cos\alpha \cdot \sin\gamma & \sin\alpha \cdot \sin\beta \cdot \sin\gamma + \cos\alpha \cdot \cos\gamma & \sin\alpha \cdot \cos\beta \\ \cos\alpha \cdot \sin\beta \cdot \cos\gamma + \sin\alpha \cdot \sin\gamma & \cos\alpha \cdot \sin\beta \cdot \sin\gamma - \sin\alpha \cdot \cos\gamma & \cos\alpha \cdot \cos\beta \end{bmatrix} \tag{A.5}$$

.

e, finalmente, a matriz de transformação **A**:

$$\mathbf{A} = \begin{bmatrix} \cos\beta \cdot \cos\gamma & \cos\beta \cdot \sin\gamma & -\sin\beta & t_x \\ \sin\alpha \cdot \sin\beta \cdot \cos\gamma - \cos\alpha \cdot \sin\gamma & \sin\alpha \cdot \sin\beta \cdot \sin\gamma + \cos\alpha \cdot \cos\gamma & \sin\alpha \cdot \cos\beta & t_y \\ \cos\alpha \cdot \sin\beta \cdot \cos\gamma + \sin\alpha \cdot \sin\gamma & \cos\alpha \cdot \sin\beta \cdot \sin\gamma - \sin\alpha \cdot \cos\gamma & \cos\alpha \cdot \cos\beta & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.6}$$

A matriz de transformação é associada à uma matriz de pontos para descrever sua posição no espaço 3D. Havendo somente uma matriz de pontos, sua matriz de transformação será a matriz identidade. Quando há mais de uma matriz de pontos, faz-se necessário que cada uma tenha sua pose precisamente descrita por uma matriz de transformação, para que o posicionamento e o alinhamento entre diferentes nuvens de pontos seja exato.

Apêndice B

Orientações práticas

B.1 Ambiente de desenvolvimento

Após instalar o Ubuntu 12.04, o primeiro passo é instalar as bibliotecas necessárias para compilar o código. As bibliotecas Eigen, libfreenect, Boost, VTK e CUDA são instaladas primeiramente pois são menores e algumas já estão compiladas. Já as bibliotecas PCL, MRPT, e OpenCV tem que ser compiladas e instaladas manualmente, e demoram algumas horas. Além disso, é comum haver erros de compilação, sendo necessário editar o código fonte de alguns arquivos.

O ambiente de desenvolvimento utilizado é a IDE Eclipse CDT. Como são muitas bibliotecas, o processo é automatizado por meio do programa CMake, que gera e exporta um projeto para Eclipse ou CodeBlocks baseado em um arquivo de diretivas chamado CMakeLists. Nesse arquivo é especificado as pastas de origem, de destino, o nome do projeto, subdiretórios, outros diretórios para compilação e linkagem, bibliotecas e executáveis.

B.2 Uso do programa

É importante que o ambiente esteja bem iluminado para que as imagens tenham um aspecto visual nítido. Além disso, não é aconselhável que haja uma fonte luminosa no campo de visão do sensor, uma vez que isso resulta em escurecimento do resto da cena nas imagens. Sendo assim, não é aconselhável que o sensor observe diretamente a luz solar do ambiente. O programa pode ser executado à noite desde que haja iluminação artificial adequada.

Após o programa ser executado, o usuário deve se posicionar à 50cm de distância do sensor, pois é a distância usada pelo algoritmo de segmentação. Em seguida, o software dará início ao processo de aquisição, e o usuário deve movimentar seu rosto sem movimentos bruscos, enquanto o programa registra as diferentes regiões da face. Ao fim desse processo, o usuário deve apertar a tecla "a"para que o algoritmo finalize a aquisição.

Em seguida, será iniciada a etapa de processamento, que durará alguns minutos. Por último, o programa fará a requisição de quais imagens deve usar como textura, e o usuário deve informar as imagens que acha adequado. É aconselhável escolher uma imagem para

cada faceta do rosto, ou seja, uma de vista frontal e duas de vista lateral. Após isso, o programa deve finalizar gerando o modelo 3D na mesma pasta em que foi executado. Para visualizar o modelo, é necessário um software de visualização de malhas de polígonos, como por exemplo o *freeware* Meshlab.