Faculdade de Engenharia da Universidade do Porto



Projection Mapping aplicada à indústrial de Fabricação de estruturas para armazenamento alimentar

Carlos Manuel Borges Silva

VERSÃO DE TRABALHO

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Automação

Orientador: Prof. Pedro Luís Cerqueira Gomes da Costa
Co-orientador: Luís André Freitas da Rocha

© Carlos Manuel Borges Silva, 2019

Resumo

Atualmente existe uma forte evolução das tecnologias, nomeadamente em automatizar processos. Este tipo de automatização requer investimento por parte das empresas que retornará em lucro por minimizar os gastos de operacionalidade.

Ao longo desta dissertação pretende-se explicar uma destas automatizações que neste caso se aplica à melhoria de processo de fabricação de estruturas para o armazenamento alimentar.

O projeto incide no desenvolvimento de uma aplicação que visa ajudar a marcação de cortes de tampos para silos alimentares. Esta marcação tem por base um modelo desenhado a computador do tampo final (com todos os cortes e dimensões) contudo é necessário o desenvolvimento de um algoritmo capaz de entender e calcular eficazmente as diferenças, a deformação e a distorção entre o modelo real e virtual.

Posteriormente, essas entidades (cortes e furações) são projetadas com um laser no tampo do silo para que sirvam de guias para um operador efetuar os cortes e/ou as furações.

Os resultados atingidos vão ao encontro dos objetivos, que é a automatização de processos tornando-os mais eficientes e eficazes, tendo sido desenvolvido com sucesso a aquisição das nuvens de pontos, o tratamento desta informação com o modelo virtual, e o cálculo da deformação existente entre os diferentes modelos.

Página em branco

Abstract

Nowadays, there is an huge evolution of technologies, namely in processes automatization, and so that it is necessary an investment of companies that will compensate further to make profit, minimizing the wastes of operability.

Throughout this dissertation it is intended to explain one of these processes whose the goal is improving the process of manufacturing of structures for food storage.

The project consists on the development of an appliance that has the main objective of helping the cutting marks of a food container top. The cutting marks is based in a computer aided design model of the final food container top (with holes and cuts) although it is mandatory the development of an algorithm that is able to understand and compute effectively the differences, the deformation and the distortion between the real model and the virtual model.

Then, those entities (holes and cuts) are projected with a laser on the food container top in order to be guide lines for an operator do the cuts and/or drilling for the holes.

The results were well succeeded when they are compared with the goals, that is the processes automatization becoming in that way those processes more efficient and effective, so that it was developed with an high degree of success the point cloud acquisition, the data analysis of the virtual model and the compute of the deformation that exists between the different models.

Página em branco

Agradecimentos

Aos meus pais e irmãos, pelo o apoio demonstrado em todas as fases da minha vida até aqui e não só esse apoio foi crucial para a conclusão e elaboração desta tese como para as decisões que tomei em toda a minha vida que me permitiram ser a pessoa que sou hoje e me deram bagagem emocional e racional para chegar a esta última etapa da educação superior.

Aos meus amigos, que sempre estiveram no sitio certo à hora certa para dizer as coisas certas e mesmo que se quisesse desistir alguma vez, nunca me deixariam e que sei que sempre se orgulharão de mim pelas minhas conquistas, e esta dissertação é sem dúvida uma delas.

À minha namorada que esteve comigo horas infindáveis, escutando meus problemas que surgiam sobre a tese e mesmo sem saber ajudar-me na prática, teve um impacto enorme na conclusão deste projeto porque sempre me motivou e fez com que as coisas se tornassem mais simples.

Ao meu orientador, Prof. Pedro Luís Cerqueira Gomes da Costa, por em primeira instância aceitar ser meu orientador e pelo apoio e paciência que teve comigo. Agradeço-lhe a sua boa disposição, e disponibilidade que sempre revelou em ajudar-me mesmo que muitas vezes não fosse um assunto fácil.

Aos meus colegas, que partilhavam o dia de trabalho comigo no laboratório do INESC no hipercentro e sempre demonstraram querer ajudar e me ajudaram a integrar no contexto do trabalho. Um especial agradecimento para o Carlos Costa, porque lhe devo todos os minutos que lhe roubei para discutir as melhores formas de abordar os problemas que foram surgindo e por todo o tempo que me aturou e me deu força para continuar, fazendo as coisas mais simples.

Página em branco

Índice

Capítulo 1 1

Introdução 1
1.1 - Contexto/Enquadramento 1
1.1.1 Motivações e Objetivos 2
1.2 - Projeto 3
1.4 - Estrutura 4

Capítulo 2 6

Revisão Bibliográfica 6
2.1 - Introdução 6
2.2 - Aquisição e tratamento de nuvens de pontos 6
2.3 - Estudo geral das tecnologias e métodos existentes 8
2.4 - Resumo e Conclusões 19

Capítulo 3 21

Hardware para a implementação da solução 21
3.1 - *Setup* 21
3.1.1 Características e funcionalidades do *Hardware* 22

Capítulo 4 27

Implementação 27
4.1 - Tratamento de dados da nuvem de pontos e consequente abordagem da deformação existente entre modelo real e modelo CAD 28
4.1.1 Segmentação da imagem 3D capturada 29
4.1.2 Reconstrução da *Point Cloud* 33
4.1.3 Tratamento do modelo CAD 39
4.1.4 Alinhamento do modelo CAD com a imagem 3D segmada 41
4.1.5 Cálculo da deformação existente na peça para as entidades 45
4.2 - Bridge C++/CLI 59
4.3 - Tratamento de informação para interface com o utilizador e com o sistema 63

Capítulo 5 67

Testes e Resultados 67

Capítulo 6 78

Conclusões e Trabalho Futuro 78
5.1 - Satisfação de objetivos 78
5.2 - Trabalho Futuro 79

Lista de figuras

Figura 1 - Esquema representativo do fluxo do processo por tarefas de trabalho 4

Figura 2 - Nuvem de pontos exemplo que representa um objeto pelas suas coordenadas (x, y, z) e pelas suas cores (r, g, b) 7

Figura 3 - Visão global do método para um camelo: (a) - *Template* com 30 pontos para correspondência; (b) - nuvem de pontos com os pontos marcadores; (c) - deformação inicial não ajustado com a nuvem de pontos de *input* o que forma resultados fora de escala; (d) – depois de ser ajustado com a nuvem de pontos; (e), (f) e (g) – primeira, segunda e quarta iterações, respetivamente; (h) – resultado final com a sobreposição da nuvem de pontos para ser percetível a deformação alcançada. Fonte: [4] 10

Figura 4 - Esquerda: Dinossauro com um exemplo de extrema deformação usando o template do camelo da Figura 1(a). (a) - nuvem de pontos; (b) - deformação inicial; (c) - resultado final com pernas mais pequenas assim como cauda e ainda a inserção de orelhas. Direita: Utilização do cavalo em (d) como template para a nuvem de pontos de um camelo definida em Figura 1(b), com o resultado final em (e), Fonte: [4]..... 11

Figura 5 - O dragão representado à esquerda é deformado pela otimização das posições das células, mostrando o seu resultado final à direita, Fonte: [5]..... 12

Figura 6 - Todas as células sofreram rotações aleatórias. A evoluçõ do algoritmo está ilustrada ordenadamente da esquerda para a direita e demonstram as iterações 1, 4, 7 e 25. Fonte: [5] 13

Figura 7 - Vantagens da criação de células dinâmicas num exemplo 2D: (a) - forma original; (b) - células regulares transformadas; (c) - identificação de bordas; (d) - melhoramento de bordas e de erros; (e) - resultado final das células regularizadas. As cores da grelha em (c) e (d) representam o erro, onde o vermelho é visto como um erro elevado e o azul, um erro baixo. O número de células é representado abaixo de cada coluna de modelos e os pontos a vermelho são aqueles que foram determinados como pontos de deformação. Fonte: [5]..... 13

Figura 8 - Cálculo das correspondências, da direita para a esquerda: correspondência pelo ponto mais próximo; correspondências melhoradas a partir de características locais; correspondências filtradas a partir da consistência geodésica; correspondências finais com pesos codificados por cores em que o vermelho representa pouca fiabilidade e o azul máxima fiabilidade, Fonte: [6] 15

Figura 9 - Ilustração de *clusters*, cujo o movimento está representado na ilustração mais à esquerda e como se pode ver as zonas onde têm mais e menores *clusters* é onde de dobra o corpo e que contém por consequência uma maior deformação enquanto os *clusters* com maior área representam zonas de pequena deformação, Fonte: [6]..... 15

Figura 10 - Fluxograma do algoritmo de deformação, Fonte: [6] 16

Figura 11 - Ilustração da evolução do algoritmo de deformação a partir de cálculo de correspondências e de *clusters*, Fonte: [6]..... 16

Figura 12 - Algoritmo de reconstrução, Fonte: [7] 17

Figura 13 - Interpolação das curvas com diferentes métodos de parametrização: (a) - Igual espaçamento; (b) - comprimento de corda; (c) - parametrização centrípeta, Fonte: [7]..... 17

Figura 14 - Algoritmo de reconstrução das faces, Fonte: [7] 18

Figura 15 - Esquerda: modelo CAD da peça final. Direita: peça real no início do processo (sem cortes nem furos)..... 20

Figura 16 - Setup para prototipagem..... 22

Figura 17 - *Setup* multi câmera para aquisição de nuvem de pontos, Fonte: [9] 23

Figura 18 - *Setup* câmera + fonte de luz para aquisição de pontos, Fonte: [10] 23

Figura 19 – *Setup* câmera + luz estrutrada para aquisição de pontos para a representação da esfera, Fonte: [11] 24

Figura 20 - *Setup* usando a técnica tempo de voo para a aquisição de pontos, Fonte:[12] 24

Figura 21 - Campo de visão da câmera, Fonte: [13]..... 25

Figura 22 - Esquerda: projetor laser. Direita: resultados exemplo, Fonte: [14] 26

Figura 23 - Divisão do algoritmo geral do cálculo da deformação existente entre modelos ... 29

Figura 24 - Objeto de análise, protótipo de um tampo de um silo 29

Figura 25 - Ilustração explicativa do filtro passa banda 30

Figura 26 – Esquemático do filtro passa banda 30

Figura 27 - Resultado exemplo da aplicação do filtro passa banda..... 30

Figura 28 - Ilustração representativa de um *voxel* aplicado ao filtro de redução de amostragem 32

Figura 29 - Malha do objeto em análise (à esquerda) e respetiva nuvem de pontos (à direita) 32

Figura 30 - Resultado da aplicação do filtro *downsampling*..... 32

Figura 31 - Esquerda: triangulação *Delaunay* que apresenta muito menos suavidade e polígonos muito mais irregulares; Direita: método de triangulação de *Poisson* que apresenta uma suavidade maior e uma melhor regularidade dos polígonos, Fontes: [17], [18]..... 34

Figura 32 - Extrapolação de vetores normais, Fonte: [20] 35

Figura 33 - À esquerda normas calculadas com sentido das normais ambíguas e à esquerda normais calculadas com sentido consistente, Fonte: [19] 35

Figura 34 - Extração da Isosuperfície, Fonte: [21] 36

Figura 35 - Exemplo de octree, Fonte: [23] 36

Figura 36 - Resultado dos testes de validação do algoritmo de reconstrução, Fonte: [22] 38

Figura 37 - Resultado do alinhamento sem *upsampling* do ficheiro STL 40

Figura 38 - (a): Modelo CAD visualizado como um sólido; (b): Vértices que definem o ficheiro STL representado em (a); (c): Ficheiro STL representado pelos seus triângulos; (d): Resultado do algoritmo de *upsampling* 41

Figura 39 - Pré-alinhamento entre duas superfícies, Fonte: [28] 42

Figura 40 - Pré-alinhamento do objeto e consideração dos primeiros pares de pontos 43

Figura 41 - (a): Nuvem de pontos original (referência) representado preto e nuvem de pontos representada a verde é a nuvem de pontos rotacionada e translacionada; (b): Nuvem de pontos representada a vermelha após uma iteração; (c): Nuvens de pontos alinhadas, Fonte: [31] 45

Figura 42 - Exemplo básico de uma *Kd-Tree* 47

Figura 43 - 1ª etapa: Algoritmo de busca, Fonte: [32] 48

Figura 44 - Resultado prévio da solução do algoritmo, Fonte: [32] 49

Figura 45 - 2ª etapa: Algoritmo de busca, Fonte: [32] 49

Figura 46 - 3ª etapa: Algoritmo de busca, Fonte: [32] 49

Figura 47 - 4ª etapa: Algoritmo de busca, Fonte: [32] 50

Figura 48 - 5ª etada - Algoritmo de busca, Fonte: [32] 50

Figura 49 - Triângulos circundantes a um ponto, Fonte: [33] 51

Figura 50 - Estrutura de dados que relaciona polígono, plano e pontos 51

Figura 51 - Projeção de um ponto no plano com um *ray* de direção equivalente à normal ao plano, Fonte: [34] 53

Figura 52 - (a): Ponto projetado no plano e contido no polígono; (b): Ponto projetado no plano não está contido no polígono o que implica uma projeção para o ponto mais próximo do polígono 53

Figura 53 - Representação gráfica do método para descobrir se um ponto está dentro de um triângulo, Fonte: [35] 53

Figura 54 - S é a projeção de X que representa o ponto mais próximo à reta r, Fonte: [36] 55

Figura 55 - Três casos possiveís para a determinação do ponto mais próximo, Fonte: [36] 56

Figura 56 - Exemplo esquemático do resultado do algoritmo completo sendo que o ponto verde é o ponto eleito para corrigir a deformação porque tem a menor distância relativamente aos pontos vermelhos 57

Figura 57 - Superfície plana com pontos a projetar representados a verde o pontos projetados a azul (correspondentes à deformação), Exemplo 1 57

Figura 58 - Deformação de uma circunferência num cilindo e num cone, Exemplo 2 58

Figura 59 - Deformação de uma circunferência e de uma linha numa semi-esfera, Exemplo 3 58

Figura 60 - Deformação de uma circunferência e de uma linha num aro e num cilindro, Exemplo 4 58

Figura 61 - Deformação de uma linha num cilindro 59

Figura 62 - Estrutura C++ 60

Figura 63 - Estrutura C++/CLI 60

Figura 64 - Funções de implementação da *bridge*, construtor e função de execução, respetivamente 61

Figura 65 - Coversão de *String* para *char** 61

Figura 66 - Conversão *array* de pontos (C#) para vetor (C++) 62

Figura 67 - Conversão de vetor de pontos (C++) para *array* de pontos (C#) 62

Figura 68 - Exemplo de uma entidade original 64

Figura 69 - Exemplo gráfico da projeção laser a partir dos pontos originais das entidades 64

Figura 70 - Exemplo de resultado da função de inserção de pontos 65

Figura 71 – Execução do sistema e resultado gráfico 66

Figura 72 - Transgressão de pontos para linhas 66

Figura 73 - Objeto representativo de um tampo, modelo real em análise 67

Figura 74 - *Point Clouds* do modelo real fragmentado em quatro partes distintas 67

Figura 75 - Nuvens de pontos registadas em uma única, sendo a ilustração da esquerda uma representação das diferentes nuvens de pontos usadas para o registo e a ilustração da direita uma representação cromática real do objeto 68

Figura 76 - *Downsampling* da nuvem de pontos. Origem representado a branco e o resultado representado a verde 68

Figura 77 - Superfíce reconstruída por o método de *Poisson* 69

Figura 78 - (a): Modelo CAD visualizado como um sólido; (b): Vértices que definem o ficheiro STL representado em (a); (c): Ficheiro STL representado pelos seus triângulos; (d): Resultado do algoritmo de upsampling 70

Figura 79 - Primeira iteração do algoritmo *ICP* 71

Figura 80 - Alinhamento completo..... 71

Figura 81 – Alinhamento completo com visualização distancial cromática..... 72

Figura 82 - Modelo CAD à esquerda e à direita uma nuvem de pontos que representa todas as entidades a representar com cores distintas 72

Figura 83 - Projeção final das entidades na superfície do modelo real com as entidades projetadas representadas a azul 73

Figura 84 - *Zoom* da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 1 73

Figura 85 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 2 74

Figura 86 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 3 74

Figura 87 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 4 75

Figura 88 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 5 75

Figura 89 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 6 76

Figura 90 - Consola de comandos do sistema final..... 76

Figura 91 - Paradigma do *setup* de prototipagem..... 80

Figura 92 - Paradigma do *setup* em contexto indústrial 80

Figura 93 - Ilustração do movimento de uma câmara 3D para efetuar capturas para o registro do objeto completo. 81

Lista de tabelas

Tabela 1 - Especificações da câmera 3D 25

Tabela 2 - Tabela de *benchmarking* do algoritmo usado 39

Tabela 3 - *Benchmark* do algoritmo de *downsampling* 68

Tabela 4 - *Benchmark* do algoritmo de reconstrução de *Poisson* 69

Tabela 5 - *Benchmark* do algoritmo de *upsampling* do modelo CAD 70

Tabela 6 - *Benchmark* do algoritmo de alinhamento *ICP* 72

Tabela 7 - *Benchmark* do algoritmo de projeção de pontos 76

Abreviaturas e Símbolos

Lista de abreviaturas

CAD	*Computer Aided Design (texto não português em itálico)*
ICT	*Iterative Closest Point*
PCL	*Point Cloud Library*
STL	*Standard Tessellation Language*
FEM	*Finite Element Method*
BREP	*Boundary representation*
WDE	*Weighted Displacement Estimation*
FEUP	Faculdade de Engenharia da Universidade do Porto

Capítulo 1

Introdução

O paradigma deste projeto de dissertação consiste na melhoria de um sistema de marcação de cortes em tampos de silos, neste caso, aplicados à indústria alimentar. Em todas as indústrias existem soluções que podem ser implementadas, de modo a melhorar os processos dessa indústria tornando-a mais rápida e eficaz e, consequentemente, mais lucrativa. Portanto, a solução implementada neste projeto visa num sistema completamente autónomo e elevadamente mais fiável e eficiente.

Na indústria alimentar são necessários silos para o armazenamento de alimentos ou preparados alimentares para que posteriormente seja processado, de forma a ser comercializado. Pois, os silos têm de ser fabricados de acordo com as necessidades exigidas pelo processo que será submetido e até pelo tipo de alimentos que irá conter. O sistema a implementar focar-se-á apenas nos tampos que são uma parte pertencentes aos silos.

1.1 - Contexto/Enquadramento

A evolução industrial é abismal e todas as políticas de transformação digital das organizações de diferentes setores serão consideradas para que os seus negócios se mantenham competitivos.

Atualmente uma das soluções mais optadas pelas empresas é a automação de processos industriais, ou seja, o desenvolvimento e a implementação de melhorias, visando ganhar agilidade, produtividade e reduzir custos operacionais. A principal premissa baseia-se que na maioria dos casos é possível produzir mais em menos tempo, com um custo associado inferior e com uma qualidade claramente superior.

Este tipo de soluções apresentam vários tipos de benefícios o que levam as organizações a optar pela automação de processos. As principais vantagens são: otimização dos tempos e aumento de produtividade, redução de custos, dados integrados (com a informação totalmente

integrada torna-se fácil criar relatórios em tempo real e detetar eventuais problemas que atrasem os processos, informação e dados dos processos é sempre uma mais valia), integração de sistemas (a possibilidade de diferentes processos comunicarem entre si a partir de aplicações com o objetivo de torná-los mais eficazes) e por fim, maior fiabilidade (garantia de maior fiabilidade ao utilizar as mais recentes tecnologias, o que reduz substancialmente a taxa de erro)[1].

Esta aplicação tem um propósito prático e aplicado à indústria cujo objetivo é tornar sempre o trabalho mais facilitado e para isso foi considerada uma solução para o problema de marcação de estruturas, neste caso tampos de silos para o armazenamento alimentar.

O problema foi encontrado porque um mapa de processos de uma empresa foi abordado tendo em conta as tarefas que constituem as necessidades da empresa e logo, quantos funcionários são necessários e qual o tempo que despendiam cada um desses nas suas tarefas e se esse era o tempo ideal para obter a melhor produtividade.

Posto isto, chegou-se à conclusão que o processo de marcação de silos é demasiado manual o que incorre em erros humanos. Em primeira instância, o desenho 2D do modelo CAD é fornecido com todas as dimensões necessárias ao operador em formato de papel. Para esta tarefa são necessários dois funcionários. Pois, o operador tem que compreender o desenho e com a ajuda de outro colega de trabalho tem que fazer as marcações com régua, esquadro e compasso.

Este trabalho, naturalmente incorre em alguns erros humanos, de medição e de marcação o que provoca defeitos nos tampos. Para além disso e o aspeto mais importante é que cada marcação de um único tampo de um silo demora em média oito horas e é sempre necessário o trabalho de dois recursos humanos.

Dadas as circunstâncias e do ponto de vista de gestão de recursos, sejam materiais ou humanos ou financeiros vale a pena fazer um estudo para perceber se é possível viabilizar este processo e tornar a empresa mais rentável.

1.1.1 Motivações e Objetivos

Como descrito ao longo desta secção, quando se pretende resolver um problema ou otimizar uma solução de caráter industrial ou de gestão, tem que se ter a total noção que no final se resume a obter lucros e rendimentos porque esse é o principal objetivo de qualquer negócio. Nesta aplicação em concreto é fácil de entender que a motivação principal e o objetivo da empresa passam por reduzir custos de operação com a redução de recursos humanos e tempo de operação.

Os objetivos gerais traçados são, ordenadamente:

- Esquematizar o problema, como ilustrado na figura 1;
- Subdividir o problema em partes distintas e independentes (subdividir cada célula representada na figura 1)

- Elaborar um plano que visa desenvolver cada uma das partes metodicamente (cada subdivisão do problema deve ser o mais abstrata possível para que o seu desenvolvimento dependa o menos possível de variáveis externas, como acontece nos temas de cada célula da figura 1 em que se nota que são abordados temas diferentes contudo será necessário a sua articulação posteriormente)
- Testar independentemente essas mesmas partes
- Integração de todas as partes em funcionamento simultaneamente
- Testes finais da solução implementada

1.2 - Projeto

Uma vez identificado o processo a melhorar, encarando-o como um desafio abordando o problema do ponto de vista tecnológico para compreender qual a solução que se poderia adaptar às necessidades da empresa que iria ficar com o produto.

Primeiramente, com o auxílio de uma câmara 3D é capturada uma nuvem de pontos do tampo real. De seguida, é carregado para o programa um modelo CAD do tampo desejado com os cortes e furos.

Depois de uma filtragem e uma segmentação na nuvem de pontos do tampo real para motivos de eficiência e eficácia do programa, o modelo CAD e a nuvem de pontos do tampo real são alinhadas para que possa ser feita uma comparação.

Caso este alinhamento seja bem-sucedido, é calculada a distorção ou deformação entre a nuvem de pontos do tampo real e o modelo CAD para que às entidades dos cortes e furos, sejam calculadas novas entidades, desta vez projetadas na superfície do tampo real. Este algoritmo foi desenvolvido de modo a que as entidades mantenham a mesma forma no tampo real que têm originalmente no modelo CAD.

Por último, foi desenvolvida uma *bridge* entre C# e C++ em C++/CLI de forma a conseguir trocar a informação do modelo CAD presente na interface C# para baixo nível e depois enviar as novas entidades ou as entidades modificadas para a interface de alto nível novamente.

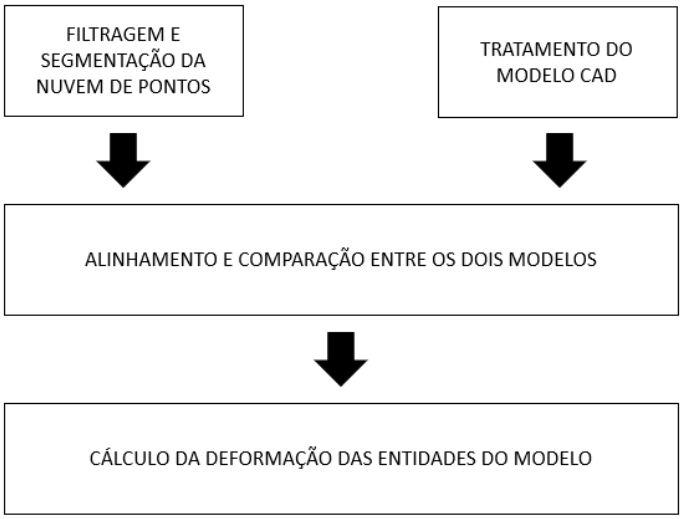


Figura 1 - Esquema representativo do fluxo do processo por tarefas de trabalho

Além do aspeto prático exigido, é necessário estar consciente que estes tampos têm em média quatro metros de diâmetro, este detalhe pode ser determinante para a determinação da solução ótima, contudo para efeitos de prototipagem e de facilidade de testes e afins, foi considerado ao longo deste período de investigação um tampo com sessenta centímetros de diâmetro. O facto de ser usado um tampo de este tamanho é somente para efeitos de investigação e de logística porque se tornaria complicado ter um setup e espaço tão grande em um laboratório, embora a solução que irá ser abordada nesta dissertação incide sobre uma solução pensada para objetos de maior dimensão.

A solução encontrada passa por montar um setup que será constituído por uma câmara 3D sobre uma bancada para a captura de nuvens de pontos onde se encontrará o tampo ou outros objetos para eventuais testes. Este setup também terá um computador que será onde reside o *software* para a implementação da solução assim como uma interface gráfica para o operador conseguir executar as tarefas.

Depois da montagem do *hardware*, é necessário o desenvolvimento do *software* que será onde incidirá este projeto de dissertação.

1.4 - Estrutura

Esta dissertação está estruturada de forma a ser percebido o objetivo do trabalho e ainda mais relevante a maneira como este foi solucionado, portanto, inicialmente o resumo e a introdução (capítulo 1) expõem levemente o objetivo do trabalho e a abordagem à solução implementada. De seguida, é contextualizado o problema com o estado de arte (capítulo 2), abordando assim as soluções existentes no mundo para o mesmo tipo de problemas, nomeadamente formas de aquisição de nuvens de pontos e sua manipulação e o cálculo da deformação entre modelos. Neste ponto já existe um conhecimento suficientemente abrangente para abordar o problema do ponto de vista de *hardware* (capítulo 3) e desse modo, segue-se o capítulo da explicação do *hardware* a ser usado para a implementação elaborada.

Uma vez que, a parte física do sistema está explicada, segue a explicação da implementação de *software* (capítulo 4), ou seja, a abordagem aos algoritmos usados e o porquê da sua utilização e a devida validação dos mesmos, sendo a exposição da implementação dividida e explicada ordenadamente de acordo com ordem da execução do programa, este capítulo determina o *core* do projeto de dissertação, sendo aqui explicado o trabalho na sua essência porque aborda a segmentação das nuvens de pontos, o tratamento do modelo CAD, o cálculo da deformação e o desenvolvimento da *bridge* de informação para os diferentes níveis de programação. Para finalizar, é testado o sistema e os seus resultados (capítulo 5) incidindo estes sobre o paradigma geral da solução. Por último, são abordados nas conclusões (capítulo 6), quais os objetivos atingidos e qual o trabalho futuro para a otimização da solução criada.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é descrito o estado da arte e são apresentados trabalhos relacionados de forma a mostrar o que existe no mesmo domínio e quais os problemas encontrados. Como abordado no capítulo introdutório existe um problema concreto que é solucionar, tecnologicamente o paradigma atual da marcação de entidades em objetos, neste caso aplicado às estruturas da indústria alimentar.

2.1 - Introdução

Para a abordagem do problema reviu-se todas as possíveis soluções existentes e tendo em conta, as especificações presente no caderno de encargos do projeto, investigou-se os temas que são necessários conhecer, tais como, a deformação entre modelos reais e virtuais, entendimento de nuvens de pontos e a sua importância, alinhamento e registo de imagens 3D, reconstrução de superfícies.

2.2 - Aquisição e tratamento de nuvens de pontos

A aquisição de nuvens de pontos pode ser gerada a partir de várias ferramentas e com as mais diversas finalidades, tais como, câmaras 3D, *laser scanning*, *stereo cameras*, etc. Embora existam diversas formas de obter nuvens de pontos, o processamento e segmentação das mesmas visam normalmente os mesmos métodos e o mesmo tipo de *softwares*. Dada esta versatilidade o tratamento deste tipo de estruturas de dados é tão estudado e usado pelas grandes organizações de todo mundo.

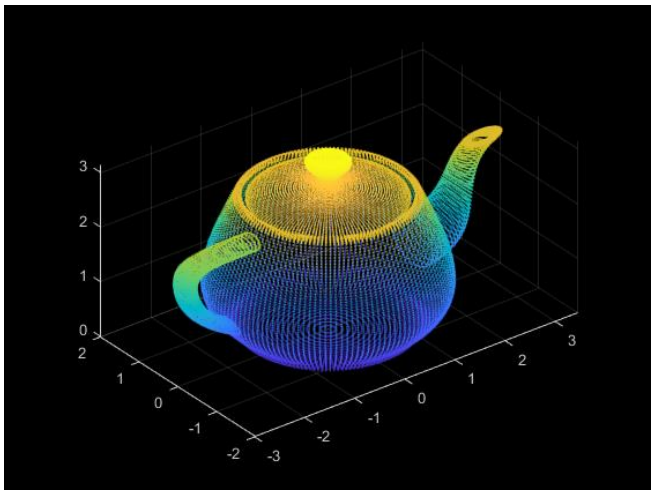


Figura 2 - Nuvem de pontos exemplo que representa um objeto pelas suas coordenadas (x, y, z) e pelas suas cores (r, g, b)

Dependendo das características de cada equipamento para a aquisição das nuvens de pontos, este levantamento pode ser baseado na captura de objetos de todos os tipos de dimensões, simples ou complexos, e com elevado nível de precisão ou não dependendo uma vez mais do tipo de equipamento e das suas especificações.

Uma nuvem de pontos é uma estrutura de dados com a finalidade de representar informação de imagens ou objetos em três dimensões em que cada ponto é representado pelas suas coordenadas cartesianas que tem como referencial de origem o ponto de captura (local da câmera), podendo ainda conter outro tipo de informação, por exemplo, como a cor a partir das três componentes do código de cores RGB como representado na figura 2.

Este tipo de estruturas de dados, usualmente, é utilizado para medição e reconstrução de espaços 3D, visualização tridimensional, criação de modelos geométricos para eventual edição em ferramentas CAD e entre outras.

Para a captura dos dados, é importante ter em conta as dimensões, o relevo e a rugosidade do objeto ou objetos alvos a identificar na imagem 3D e a partir daí selecionar um equipamento que satisfaça as necessidades através das características, podendo ser estas, a densidade de pontos, a precisão espacial, a janela de captura, etc.

Um passo importante é a fase de pré-processamento que consiste no tratamento da nuvem de pontos proveniente diretamente da captura da informação a partir do equipamento. Esta fase tem como objetivo o registo das várias capturas, ou seja, se for o objetivo capturar um objeto de grandes dimensões em que é necessário fazer o levantamento mais de uma captura, é mandatório que se registe todas as cenas antes de segmentar para que não haja perda de informação. O registo consiste em posicionar as várias nuvens de pontos no espaço tridimensional a partir da manipulação de transformações das coordenadas cartesianas de modo a obter uma única nuvem de pontos que partilhem todas o mesmo referencial.

Posteriormente ao registo de nuvens de pontos é muito usual otimizar a nuvem de pontos, diminuindo significativamente a quantidade de pontos da nuvem e consequentemente

o tamanho do ficheiro para que seja exequível o seu processamento gráfico numa máquina standard (computador). Para isso são usadas filtragens (eliminação de pontos indesejados, tais como, ruídos, obstáculos ou objetos fora da área de interesse) e reamostragens (eliminação de pontos redundantes ou sobrepostos).

A fase de processamento vem imediatamente a seguir e tem como objetivo segmentar e modelar o objeto ou objetos alvos e para isso é necessária a correta identificação do objeto e as suas características como cálculo das normais às superfícies, cores do objeto, arestas e outras entidades e geralmente modelar que consiste da passagem de uma nuvem de pontos desorganizada para uma malha poligonal (usualmente triângulos) para que esta estrutura possa ser exportada para ficheiros em que é possível a edição dos mesmos em *softwares* que manipulam ficheiros CAD, como, *SolidWorks, CATIA, AUTOCAD,* etc.

Existem diversas ferramentas para a manipulação deste tipo de estruturas de dados (nuvens de pontos), entre as quais *CloudCompare, MeshLab,* entre outros que permitem a realização de diversos tipos de operações: filtragens e reamostragens, medição de distâncias, cálculo de áreas e volumes, criação de malhas poligonais tridimensionais, técnicas de alinhamento, etc.

Embora existam muitas ferramentas disponíveis para a manipulação deste tipo de estruturas é importante realçar que estas ferramentas dispõem de interface gráfica o que têm a vantagem de simplificar a validação e a familiarização com estas técnicas no entanto, o objetivo é criar uma solução autónoma o que implica que seja usado uma biblioteca com estas ferramentas ou então que seja desenvolvida uma no sentido de implementar todas as funções necessárias para o processamento da nuvem de pontos[2].

2.3 - Estudo geral das tecnologias e métodos existentes

A deformação entre modelos de vários tipos já é um caso estudado com alguma relevância em muitas indústrias, e muito usada na área da saúde para a reconstrução de de partes ósseas ou até mesmo para ver deformação existente nos músculos do corpo humano, sendo estas comparações feitas com um *template* da parte que se está a estudar. A ideia neste projeto é entender que tipo de deformação existirá entre a peça real e o modelo CAD, excluindo para esta análise todas as entidades que o modelo CAD apresente porque este representa o resultado final que se deseja projetar no objeto real.

O cálculo desta deformação existente entre os dois modelos pode ser feita através de um procedimento numérico designado por FEM (Método dos Elementos Finitos), cujo o seu propósito é determinar soluções numéricas para problemas sobre o contorno de superfícies modelizadas a partir de equações diferenciais. Numa superfície, este método divide a mesma em partes menores, sendo designada cada parte por elementos finitos.

O uso deste método apresenta um conjunto de vantagens, cuja mais interessante será a representação com uma substancial precisão geometrias complexas. Este método elimina o

uso de equações diferenciais para a representação de superfícies, sendo que os elementos finitos que são constituintes da superfície são criados de modo a não alterarem as propriedades geométricas do modelo, e cada elemento finito é definido por equações lineares o que ajuda em questões de processamento e em eficiência porque modeliza esse elemento matematicamente a partir de uma expressão simples.

Estes elementos finitos são a razão que explica a simplicidade deste método porque estes elementos são articulados globalmente com os vários elementos do sistema sendo ajustados apropriadamente em termos de orientação com base num sistema referencial de coordenadas espaciais. A partir deste método ou das suas variantes é possível a geração de malhas de superfícies ou objetos sólidos e a sua análise mais aprofundada, simplificando a matemática por detrás disso[3].

A reconstrução de superfície em uma malha é uma parte importante para análise de distorção ou de deformação entre dois modelos, isto porque uma nuvem de pontos tem que ter uma relativa qualidade para a sua malha seja fidedigna com a realidade para que o cálculo de deformação seja fiável.

O processo de transformação de uma nuvem de pontos desordenada para um consistente modelo de malha definida por uma superfície, como uma *polygon mesh,* é usualmente designada por reconstrução de superfície, contudo para esta reconstrução existem dois tipos de abordagem: efetivamente a reconstrução de uma malha com base numa nuvem de pontos como *input,* e também a perceção como esta deformação é criada entre a nuvem de pontos e a superfície a ser criada.

Um dos métodos estudados previamente pelo C. Stoll, Z. Karni, C. Rössl, H. Yamauchi e H.-P. Seidel é a utilização de pequeno número de correspondências de pontos marcados interativamente pelo utilizador (o que requer um *know-how* ao operador do sistema sobre mecânica de estruturas e do algoritmo de deformação a ser concebido) para deformar a forma de um *template* em relação à posição de um modelo representado sobre uma nuvem de pontos. Estas deformações são relacionadas com base num critério geométrico. A ideia é reconstruir uma superfície suave e consistente do que é pretendido com o *input* dado no início do processo.

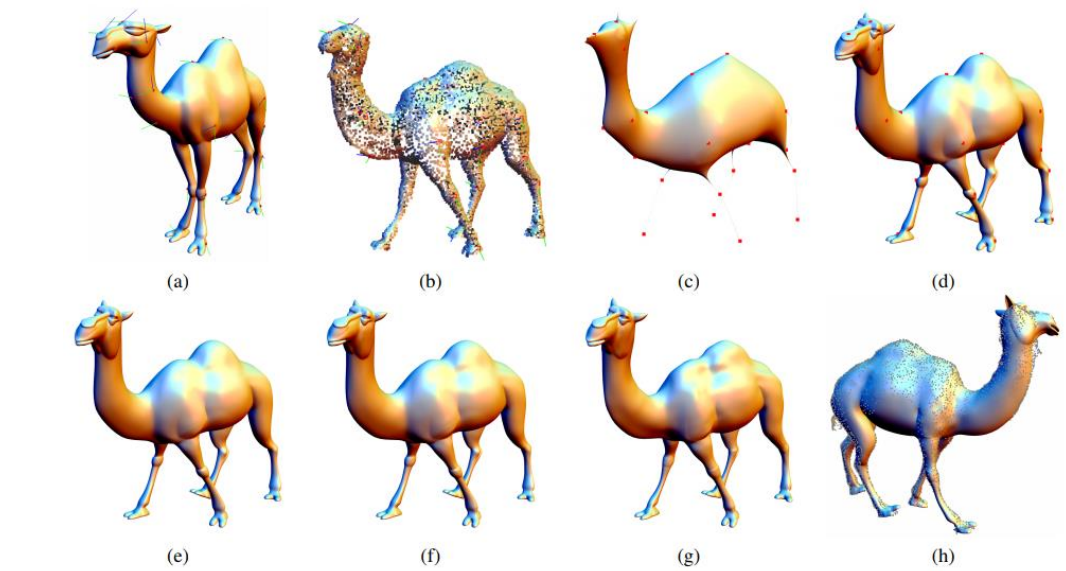


Figura 3 - Visão global do método para um camelo: (a) - *Template* com 30 pontos para correspondência; (b) - nuvem de pontos com os pontos marcadores; (c) - deformação inicial não ajustada com a nuvem de pontos de *input* o que forma resultados fora de escala; (d) - depois de ser ajustado com a nuvem de pontos; (e), (f) e (g) - primeira, segunda e quarta iterações, respetivamente; (h) – resultado final com a sobreposição da nuvem de pontos para ser percetível a deformação alcançada. Fonte: [4]

Uma vez explicadas as funcionalidades e os seus resultados é apresentada uma visão geral do algoritmo para a implementação deste método. Este algoritmo inicia com a escolha de um *template* e de uma nuvem de pontos escolhida pelo utilizador, esta nuvem de pontos considera-se que seja incompleta, com falhas de informação e com ruído associado. Admite-se que a nuvem de pontos seja possível a estimação correta das normais. Posto isto é necessário, uma vez mais o utilizador intervir para definir um conjunto de pares de pontos tanto no *template* como na nuvem de pontos. A escolha destes pares de pontos é um passo fulcral para que o método seja fiável, nesse sentido a escolha dos pontos são tipicamente colocados junto às *features* que representam a forma do objeto, como podemos ver na figura 3, os pontos fulcrais no camelo estão maioritariamente no seu focinho, pescoço, bossas, cauda e patas. Com esta informação pressupõe-se que o utilizador tem que estar consciente da forma do objeto e da forma como o algoritmo é executado.

Depois, desta interatividade para a escolha de pontos e de modelos é iniciada a fase de correspondência e de reconstrução da *mesh* para a deformação detetada, este é um processo iterativo e algumas das interações estão ilustradas na Figuara 3(e) – (g). Para todos os vértices no *template* é procurado por correspondência na informação da *point cloud*. Esta busca de correspondência é realizada a partir de dois parâmetros: um raio máximo de pesquisa e um desvio angular máximo das normais entre o *template* e a nuvem de pontos. As falsas correspondências e/ou a insuficiência de informação nos *inputs* do método são evitadas mascarando essas regiões. Para cada correspondência verdadeira é incluído uma variável de peso posicional ao sistema linear e com base nessas variáveis o *template* é reconstruído. Este

é um processo iterativo sendo sempre alterado o *template* iterativamente até que a constante do peso não seja significativamente alterada ou até ser inferior a um erro previamente definido.

Por último, uma vez que a deformação está calculada e concluída é necessário fazer um escalonamento da malha para que a forma e as proporções do *input* esteja de acordo com o suposto. Em primeira instância é estimada as escalas locais para cada correspondência baseado nos deslocamentos entre o *template* deformado inicialmente e a última deformação. A distância proveniente desses deslocamentos é interpolada sobre o *template* para gerar as posições dos pontos para a reconstrução final. O objetivo desta última etapa é uniformizar a informação de escala por todo o objeto, ou seja, de modo a abranger e a criar um resultado fiável e robusto por todo o objeto, tendo esta etapa especial enfoque nas zonas sem correspondências, quer por falta de pares de pontos ou por falta de informação ou zonas vazias, isto provoca um resultado como demonstrado na figura 3 (h)[4].

Em suma, este algoritmo é bem-sucedido no cálculo da deformação entre dois modelos sendo o desenvolvimento baseado em superfícies *Laplacianas*, usando previamente um algoritmo de cálculo de deformação com correspondências selecionadas por um utilizador de uma interface que interage com os modelos e posteriormente a execução baseia-se na deformação inicial e logo na escalonamento global, passando finalmente por um algoritmo iterativo de melhoria de aproximação.

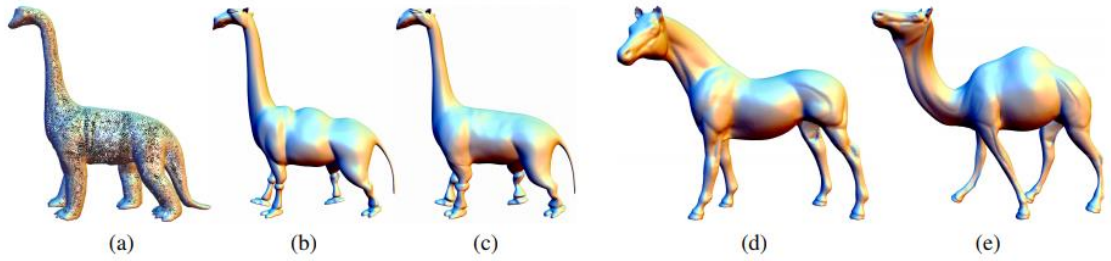


Figura 4 - Esquerda: Dinossauro com um exemplo de extrema deformação usando o template do camelo da figura 3(a). (a) - nuvem de pontos; (b) - deformação inicial; (c) - resultado final com pernas mais pequenas assim como cauda e ainda a inserção de orelhas. Direita: Utilização do cavalo em (d) como template para a nuvem de pontos de um camelo definida em figura 3(b), com o resultado final em (e), Fonte: [4]

Outra forma de abordar o mesmo problema foi desenvolvida por Mario Botsch, Mark Pauly, Martin Wicke e Markus Gross que representam a instituição ETH Zurich. Estes propõem uma visão diferente ou uma variante diferente de encarar a deformação, eles estudaram uma variante mais robusta da técnica de deformação de superfícies para uma abordagem de deformação volumétrica espacialmente baseando-se na energia elástica dos sólidos. Considera-se que a forma do objeto é deformada, não superficialmente mas espacialmente, com o uso que uma *Voxel Grid* que discretiza este sólido em vários sólidos, como demonstra a figura 5.

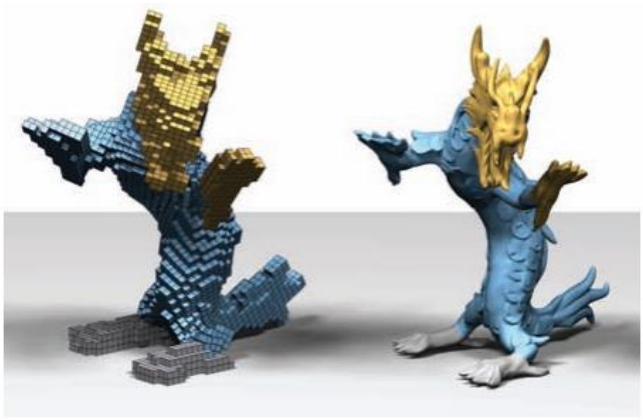


Figura 5 - O dragão representado à esquerda é deformado pela otimização das posições das células, mostrando o seu resultado final à direita, Fonte: [5]

Para a obtenção dos resultados demonstrados na figura 5, o método implementado passa por primeiramente calcular energia elástica não linear de sólidos deformáveis. Este cálculo tem em conta vários conceitos e para isso é usado tipicamente o gradiente de uma função de deslocamento porque a partir desta variável obtida através da função é possível descobrir a energia de tensão elástica para sólidos deformáveis.

Por intuição geométrica, quanto maior for o resultado deste deslocamento (maior será o gradiente por consequência), assim sendo, implica que mais deformação o objeto deve sofrer.

Em oposição ao método dos elementos finitos, onde cada célula é deformada individualmente, este método rege-se por uma abordagem diferente em que mantém as células com um determinado valor de robustez. Adicionalmente a deformação considerada a partir da energia de deformação acopla as células rígidas vizinhas, ainda neste processo é medido a variação local de deformação com base na variável da tensão entre as células.

Uma vez que a energia elástica está calculada e definida nas células do objeto, é importante agora otimizar o método porque novamente o operador seleciona manualmente a zona de análise de deformação. Em modo de contextualização, cada célula ou cubo 3D representa uma parte de um objeto que contém a informação de uma matriz de transformação referentes a movimentos de rotação e de translação, obtendo assim seis graus de liberdade por cada célula. Depois desta explicação, é possível entender que a energia elástica pode ser minimizada iterativamente de modo a obter transformações espaciais de células, tendo em conta as suas vizinhanças, sendo similar a um problema de correspondências de formas de objetos. Contudo, o problema é não linear o que implica uma péssima eficiência processual, então este estudo revela o uso de um iterador que implementa um *Newton solver* e assim linearizam as transformações rígidas das células, este cálculo iterativo é baseado em variáveis como as velocidades lineares e angulares.

Em suma, se os cálculos da aproximação das células pela elasticidade e pela minimização da energia de deformação não-linear, é possível obter resultados extremamente robustos como demonstrado na figura 6, onde as células são compactadas em um único ponto e rodadas aleatoriamente e mesmo assim com a execução do algoritmo para minimizar o valor de energia, o polvo volta ao seu estado inicial[5].

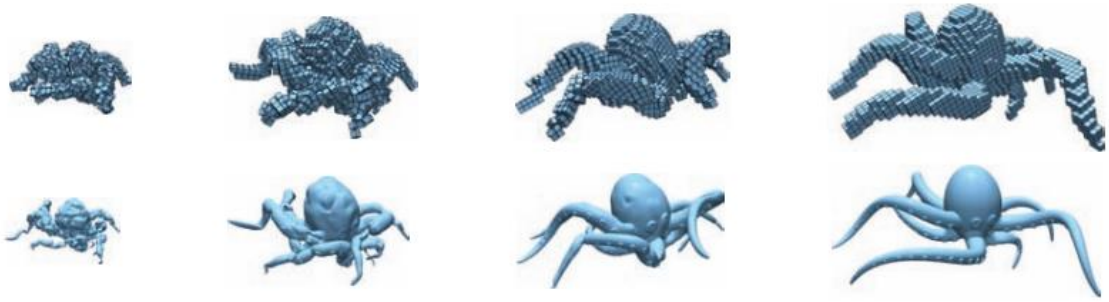


Figura 6 - Todas as células sofreram rotações aleatórias. A evolução do algoritmo está ilustrada ordenadamente da esquerda para a direita e demonstram as iterações 1, 4, 7 e 25. Fonte: [5]

A descrição deste algoritmo é a variante mais simples abordada no artigo referenciado. Pode-se evoluir o nível de complexidade e consequentemente o nível de eficiência e até mesmo de eficácia. Até então, foi considerado que a estrutura de células é regular o que implica uma simples e eficiente implementação do algoritmo, contudo este tipo de critério é bem utilizado para suavidade e deformações de grande escala. Embora para elevar o patamar e a robustez do algoritmo é usado um modo de criação de células dinâmico ou adaptativo, ou seja, em regiões onde a deformação é acentuada existirá um maior número de células com menor volume para representar essa região, se for uma região mais suave pois será o oposto, menor número de células e maior o volume das mesmas. Este método é aplicado usando uma *octree*[5].

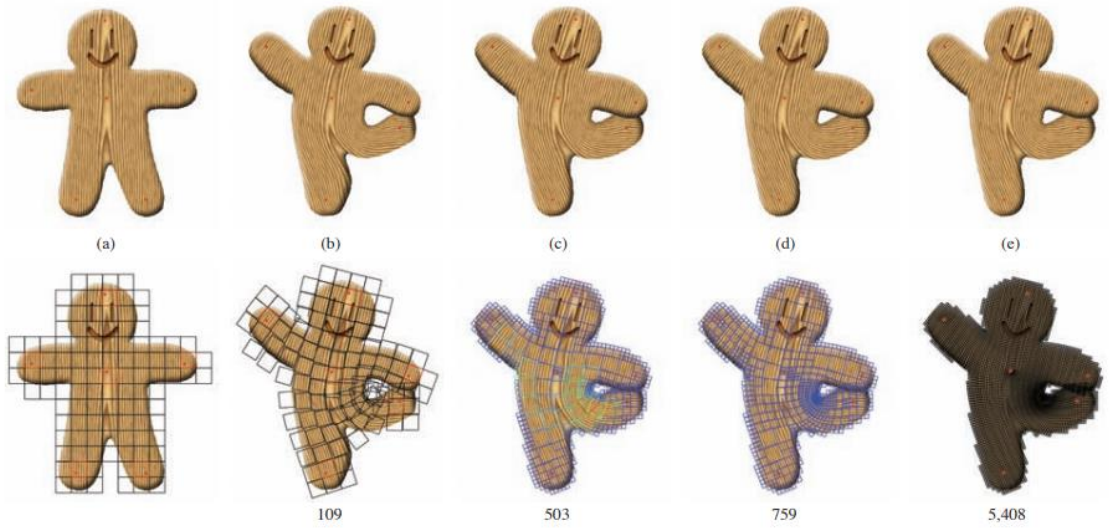


Figura 7 - Vantagens da criação de células dinâmicas num exemplo 2D: (a) - forma original; (b) - células regulares transformadas; (c) - identificação de bordas; (d) - melhoramento de bordas e de erros; (e) - resultado final das células regularizadas. As cores da grelha em (c) e (d) representam o erro, onde o vermelho é visto como um erro elevado e o azul, um erro baixo. O número de células é representado abaixo de cada coluna de modelos e os pontos a vermelho são aqueles que foram determinados como pontos de deformação. Fonte: [5]

Em suma, uma vez contextualizado o algoritmo com base na minimização da energia elástica e no seu erro, é notório que os resultados são satisfatórios para pequenas deformações,

porém para deformações de elevada escala era necessário implementar um método para tornar o robusto a solução, para isso foi desenvolvido um método dinâmico para a criação de uma grelha de células com volumes díspares dependendo da zona do objeto. Sendo os objetivos bem-sucedidos por causa: de um modelo de energia elástica não-linear que permite a deformação de formas de grande escala, um robusta otimização não-linear baseada em correspondência de formas tanto localmente como por todo o objeto, um algoritmo adaptativo de criação de células com base no erro de deformação.

A deformação existente entre dois modelos é um problema que apresenta vários problemas, um deles é o deslocamento espacial que os modelos se encontram um do outro e quando os problemas em análise são sólidos é usual usar-se o algoritmo *ICP* e as suas variantes. Resumidamente este algoritmo calcula as correspondências entre o modelo fonte e o modelo alvo, fazendo os seus cálculos iterativos obtendo uma transformação espacial que minimize as distâncias dos pontos usados para cada correspondência. Para objetos totalmente rígidos, este algoritmo é perfeito, muito utilizado e com ótimas variantes que estão disponibilizadas com várias otimizações nas mais variadas ferramentas de manipulação de imagens 3D, ou *scans,* etc.

Para o paradigma do problema abordado nesta tese, é um facto que os modelos são rígidos porque se trata de aço inox, porém existem erros associados à produção da peça, sejam estes de medidas, de deformação indesejada no material ou outras variáveis que fazem com que se considerem algoritmos para o registo de modelos com pequena ou de grande deformação em materiais não rígidos.

Pois, de acordo com o trabalho relacionado realizado por Qi-Xing Huang, Bart Adams, Martin Wicke e Leonidas J. Guibas consideram dois modelos semelhantes mas que não apresentam a mesma forma e o seu objetivo é regista-los e obter transformações espaciais para combater o problema da deformação.

Primeiramente, é dada uma superfície fonte e outra superfície que será o alvo. Cujo objetivo do registo será encontrar correspondências que relacionem os pontos da nuvem de pontos destas duas superfícies. As correspondências dos pares de pontos das duas superfícies são usadas para obter o alinhamento dos dois modelos. Se as correspondências são válidas então existe um alinhamento aproximado entre os modelos. Se as nuvens de pontos estão incompletas e/ou com ruído, o alinhamento é considerado parcial e aproximado.

Depois de obter o alinhamento, a deformação é corrigida usando a minimização de energia. Este processo é iterativo até haver convergência.

O algoritmo é dividido em duas partes, a primeira trata do cálculo de correspondências e a segunda trata da otimização da deformação.

O cálculo de correspondências foi desenvolvido dinamicamente, ou seja é definido um conjunto de pares de pontos entre o modelo de referência e o modelo fonte, e para cada ponto do par na referência e na fonte são calculadas as suas características, neste caso um vetor e um conjunto de pontos que estão numa vizinhança e calculados a partir da distância

Euclideana. Com estas duas características obtém-se a distância entre as superfícies e a sua orientação. Posto isto, o problema se for abordado da perspetiva de uma grande deformação, as características de orientação geram muito melhores correspondências do que a partir da correspondência do ponto mais próximo. No entanto, a correspondência por o ponto mais próximo é melhor para quando as superfícies estão perto uma da outra.

É muito importante preservar toda a informação sobre as características das superfícies e manter assim os vetores de orientação e as distâncias geodésicas do modelo para que se consiga obter correspondências semelhantes entre os dois modelos.

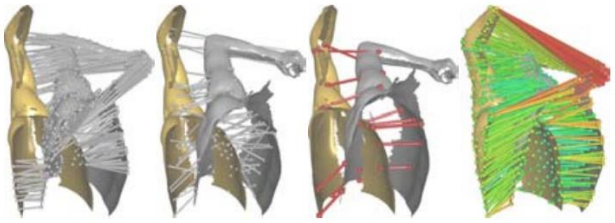


Figura 8 - Cálculo das correspondências, da direita para a esquerda: correspondência pelo ponto mais próximo; correspondências melhoradas a partir de características locais; correspondências filtradas a partir da consistência geodésica; correspondências finais com pesos codificados por cores em que o vermelho representa pouca fiabilidade e o azul máxima fiabilidade, Fonte: [6]

Depois do cálculo de todos os pares correspondentes, executa-se a parte do cálculo da deformação e para isso é utilizado iterativamente o resultado mínimo da energia entre as correspondências. Posteriormente, os pontos do modelo fonte que são transformados segundo uma matriz de transformação e, caso estas sejam similares estes pontos são agrupados em *clusters* e para cada zona de *cluster* é abordado a deformação como uma zona rígida. Para além disto, são agrupados os *clusters* vizinhos para que a deformação seja suave por todo o modelo e não haja discrepâncias onde não é suposto. As deformações das zonas rígidas são calculadas minimizando as somas das energias de correspondências, preservando sempre a rigidez local[6].

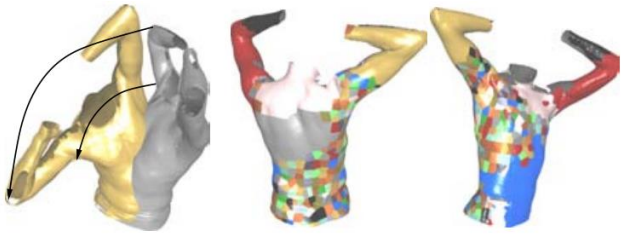


Figura 9 - Ilustração de *clusters*, cujo movimento está representado na ilustração mais à esquerda e como se pode ver as zonas onde têm mais e menores *clusters* é onde de dobra o corpo e que contém por consequência uma maior deformação enquanto os *clusters* com maior área representam zonas de pequena deformação, Fonte: [6]

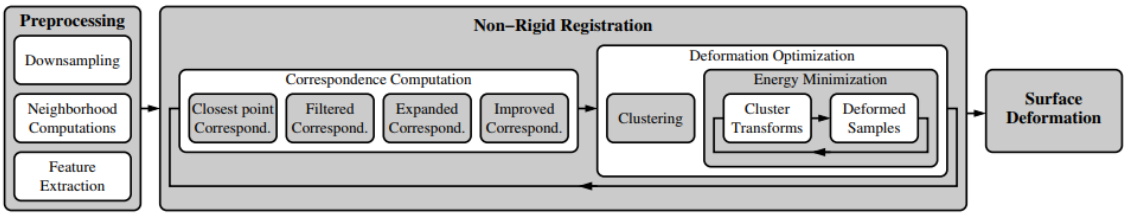


Figura 10 - Fluxograma do algoritmo de deformação, Fonte: [6]

A figura 10 ilustra graficamente o encadeamento do algoritmo explicado acima para obter uma superfície deformada cujos resultados podem ser comprovados com a figura 11:

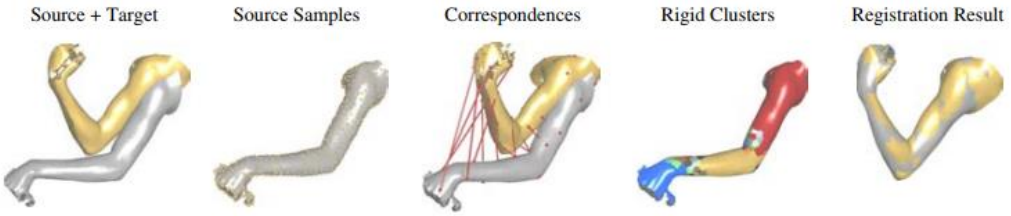


Figura 11 - Ilustração da evolução do algoritmo de deformação a partir do cálculo de correspondências e de *clusters*, Fonte: [6]

Outra abordagem ao problema da deformação é a análise de ficheiros CAD com o auxílio da FEM como explica Borhen Louhichi, Gad N. Abenhaim e Antoine S. Tahan. O seu método permite a reconstrução de um modelo CAD a partir de um modelo BREP (este tipo de modelos contém não só informação geométrica, como superfícies, curvas e pontos, mas também contém informação topológica, como faces, arestas e vértices.

Este método é segmentado em duas grandes partes, a primeira determina a topologia do modelo CAD para se executar a segunda parte que consiste na reconstrução da geometria do modelo.

Como indicado acima, primeiramente, é criado um modelo BREP a partir do modelo CAD. As entidades do modelo BREP são definidas tendo em conta as faces da malha do modelo que está deformado. Depois a reconstrução é obtida através da superfície representada por triângulos que representa a malha do modelo real (modelo deformado quando comparado com o modelo CAD).

O algoritmo de reconstrução que diz respeito à segunda metade do trabalho está subdividido em três partes: identificação da informação da malha para cada entidade do modelo CAD, reconstrução das arestas e vértices, e por último, reconstrução das faces do modelo CAD.

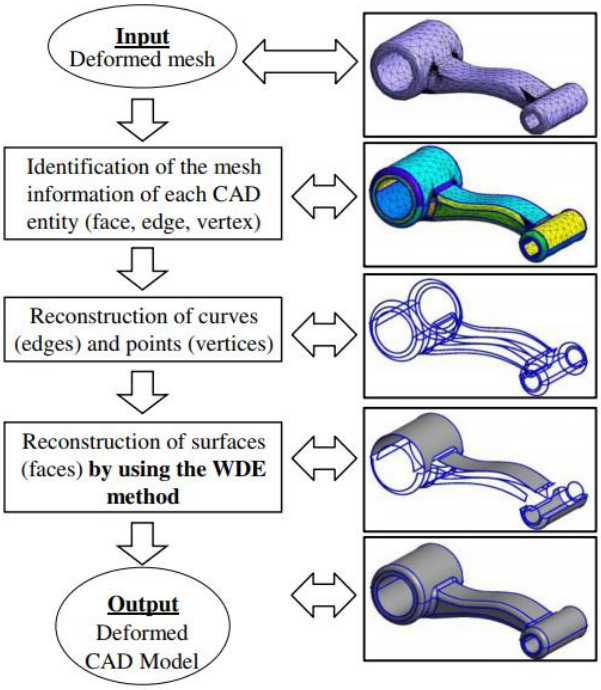


Figura 12 - Algoritmo de reconstrução, Fonte: [7]

O algoritmo começa por identificar cada superfície obtida a partir do modelo da malha de entrada do programa, uma vez feita essa identificação é reconstruído as curvas que representam as arestas e os pontos que representam os vértices, do ponto de vista topológico.

Para esta reconstrução, as arestas são determinadas a partir do cálculo da interpolação dos pontos extraídos dos limites das várias superfícies que definem a malha. Uma nota importante acerca da interpolação advém do facto que as superfícies em CAD são baseadas em superfícies *B-spline* (função que a partir de pontos de controlo determina uma *spline* com uma determinada suavidade e grau). De modo a obter uma boa reconstrução das faces e estas são obtidas a partir das arestas, é importante que as *B-splines* sejam o mais suaves possíveis de modo a obter superfícies que corrijam a deformação existente e não desempenhem o papel contrário.

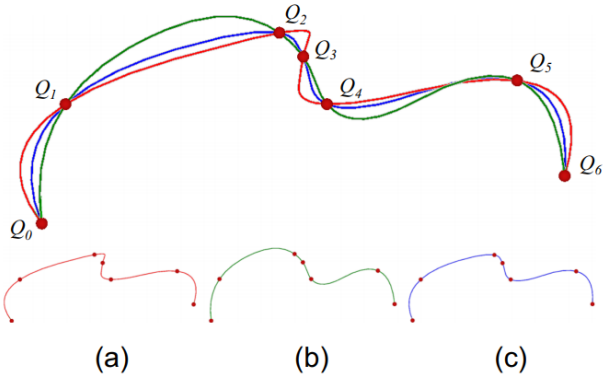


Figura 13 - Interpolação das curvas com diferentes métodos de parametrização: (a) - Igual espaçamento; (b) - comprimento de corda; (c) - parametrização centrípeta, Fonte: [7]

Uma vez reconstruída as arestas ou os limites do modelo CAD com a deformação corrigida, é necessário executar a reconstrução das faces do modelo CAD que representam a parte mais complicada do algoritmo.

Este método será aquele que corrigirá permanentemente a deformação porque num modelo CAD a face que representa um elemento finito quando deformada também deve ser movida e deformada no modelo CAD. Para modelizar as faces a partir das entidades reconstruídas no passo acima é necessário encontrar a função que determina a forma da face, esta é resolúvel a partir do cálculo de dois vetores. Posteriormente, efetua-se a triangulação dos pontos que definem a face depois de criada esta malha, estes pontos devem ser movidos para a face deformada e para isso é usado um algoritmo designado por WDE (*Weighted Differential Algorithm*), cujo o objetivo é projetar os pontos de uma face para a outra, este método estima o deslocamento entre os pontos utilizando o peso do deslocamento médio (obtido a partir da vizinhança de pontos) durante a análise dos elementos finitos (FEM)[7].

A figura 14 esquematiza sucintamente o algoritmo de reconstrução das faces utilizado para uma parte deste método para a correção da deformação.

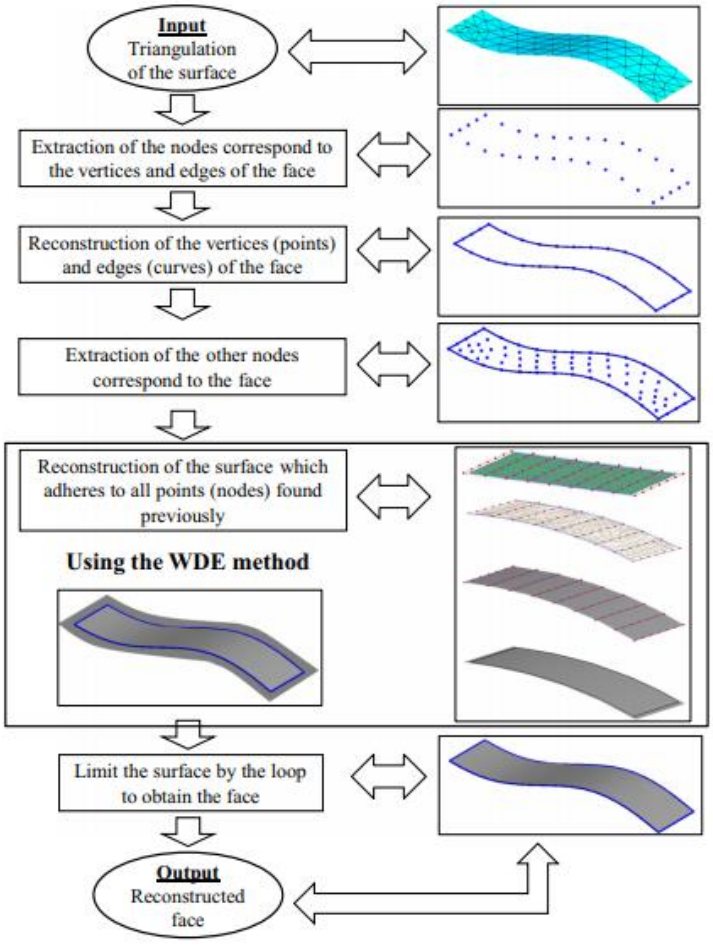


Figura 14 - Algoritmo de reconstrução das faces, Fonte: [7]

2.4 - Resumo e Conclusões

A investigação teve como objetivo principal corrigir a deformação existente entre dois modelos, sendo que um se trata de um modelo CAD que é desenhado previamente como o desejo de um resultado final e traduzido num objeto físico aplicado ao armazenamento da indústria alimentar.

Os diversos métodos estudados foram estudados de modo a perceber o que poderia ser melhorado e usado e o que não seria usado de todo contudo seria importante abordar para obter conhecimento prévio acerca deste tipo de soluções.

Primeiramente, é necessário entender qual as limitações de cada método e projetar o sistema deste de modo a ser o mais eficiente possível. Como, o paradigma do problema abordado nesta dissertação é um pouco ímpar foi pensado um algoritmo que implementará de uma perspetiva diferente e peculiar uma alternativa que visa corrigir a deformação entre modelos focando-se apenas na deformação local e não na deformação integral do objeto como é abordado em todos os casos estudados, embora seja abordado o problema de forma diferente é usada algumas *nuances* de cada projeto estudado.

A "pré solução" discutida e abordada entre o autor da dissertação e os orientadores foram arranjar uma forma de solucionar o problema mantendo em conta o sistema em aplicação, ou seja, por exemplo não é necessário uma capacidade de processamento tão grande como para o cálculo da deformação de estruturas não rígidas, contudo a sua abordagem pode ter nuances importantes para o algoritmo a implementar na solução deste trabalho.

Posto isto, foi projetado o sistema por partes:

1. Capturar o objeto que considera-se estar deformado e que será o nosso modelo real, o retorno deste passo será uma nuvem de pontos.
2. Segmentar o modelo real de modo a poder ser comparado posteriormente com o modelo CAD com um elevado nível de fiabilidade
3. Compreensão dos modelos CAD e fazer a sua análise de modo a ser possível a comparação entre os dois modelos como é abordado num dos casos estudados
4. Alinhar espacialmente os dois modelos para perceber as distâncias e pesos e para que se compreenda que tipo de deformação existe
5. Implementar o algoritmo de deformação (este ponto é o *core* do trabalho de dissertação). O pensamento exercido nesta parte foi muito focado na eficiência e eficácia do sistema visto que, tem aplicação na indústria e tempo significa lucro ou prejuízo do ponto de vista de gestão. Ao avaliar o propósito do sistema chegou-se à conclusão que as únicas partes que são essenciais para corrigir a deformação são as entidades que formam os buracos e furos na peça, e estas entidades são constituídas por pontos, o que leva a pensar que ao abordar a deformação como uma projeção de pontos numa superfície, obtém-se uma solução que é inovadora e que permitirá poupar recursos de processamento e diminuir consequentemente o tempo o que leva ao aumento de benefícios

financeiros, não sendo necessário recursos de processamento de topo como pela própria diminuição do tempo de execução do algoritmo. Esta é a premissa adotada para a realização deste projeto.

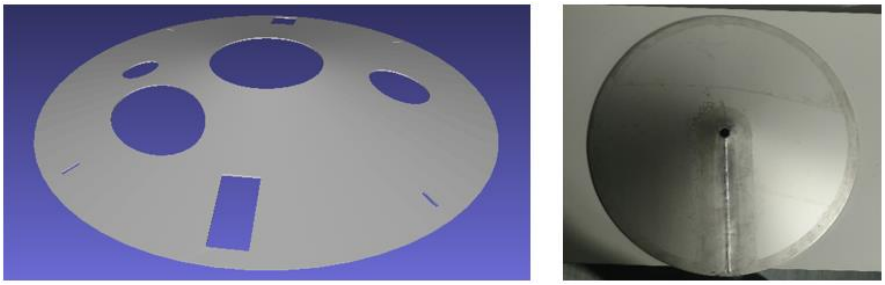


Figura 15 - Esquerda: modelo CAD da peça final. Direita: peça real no início do processo (sem cortes nem furos)

a. Considera-se cada ponto a projetar, sabendo previamente que os dois modelos se encontram alinhados e apresentam pequenas deformações.
b. O modelo real é estruturado por pontos e faces.
c. O ponto a projetar é analisado e obtém-se o ponto mais próximo desse na superfície do modelo real.
d. Depois analisa-se todas as faces em torno do ponto obtido.
e. Determina-se as menores distâncias entre o ponto a projetar e o ponto mais próximo das diferentes faces.
f. O ponto que representa a menor distância é o ponto que é eleito para corrigir a deformação.
g. Este processo é iterativo para todos os pontos a projetar.

6. Uma vez obtidos todos os pontos com a deformação corrigida, estes serão projetados na superfície por meio de uma interface com um feixe laser.

Esta é a solução encontrada para inovar e simplificar esta área de investigação para a abordagem às deformações entre modelos reais e virtuais.

O trabalho será desenvolvido em torno desta linha de raciocínio e todos os pormenores e algoritmos utilizados para a implementação do mesmo serão abordados aquando da explicação de cada fase do algoritmo para melhor entendimento e seguimento deste documento. As possibilidades dos resultados serem satisfatórios é muito alta, sendo esta análise baseada na pesquisa feita e no trabalho desenvolvido até então por outras entidades.

Capítulo 3

Hardware para a implementação da solução

Este capítulo explica todas as ferramentas e materiais utilizados e necessários para a implementação da solução em questão de *hardware*.

Este capítulo será importante para consciencializar, e consequentemente, tornar a compreensão da solução mais clara e objetiva aquando da explicação da implementação do ponto de vista de *software*, fazendo com que o leitor se inteire das limitações e das características do *hardware*.

Por último, é importante salientar que este *setup* incide sobre a fase de prototipagem devido ao seu tamanho, contudo será abordado a forma como se adapta o problema para material com portes substancialmente maiores.

3.1 - *Setup*

A figura 16 representa o *Setup* que é usado para a implementação prática da solução do sistema sendo formado por um *laser projector*, uma câmera 3D que não está representada na figura aquando da captura da imagem, contudo pode estar localizada no poste direito do pórtico similar à posição do projetor laser ou na barra horizontal do pórtico, simulando assim a posição que terá provavelmente em contexto industrial. Para efeitos de simplificação, admite-se que ambos os equipamentos de projeção e captura estão devidamente calibrados para a aplicação do sistema. Uma vez explicado a posição dos equipamentos de projeção e captura, que são os mais relevantes, ainda é necessário ter em conta a robustez do resto dos equipamentos usados como o computador que executará a solução projetada e que também não está representada na figura, a robustez e consistência da bancada e do seu pórtico para que não afete os resultados e as projeções laser. De modo a simplificar a segmentação da nuvem de pontos pode-se considerar ainda que o objeto é sempre colocado no centro da

bancada e assim reduzir posteriormente a área de interesse tendo em conta o tamanho do objeto. Por último é importante salientar que este *Setup* foi desenvolvido somente com o objetivo de validar o sistema de prototipagem e que o *Setup* a ser desenvolvido posteriormente deve ser projetado e pensado para o contexto de indústria ainda que possa ser baseado neste *Setup* de prototipagem mas com uma seleção de materiais e dimensões totalmente diferentes.



Figura 16 - Setup para prototipagem

3.1.1 Características e funcionalidades do *Hardware*

A seguinte lista de *hardware* visa mostrar as características das ferramentas físicas para a implementação da solução e para além disso mostrar qual o propósito da sua utilização.

- Câmera 3D – é um tipo de câmera de alcance (*range camera*), cuja sua designação advém de um leque de técnicas que são usadas para produzir uma imagem 2D com a distância de todos os pontos a partir de um ponto específico, ponto este, associado a partir do sensor até aos diferentes pontos. A imagem resultante tem diferentes distâncias associadas a cada ponto como indicado acima e esta distância é correspondida em valor de *pixels*. Contudo os diferentes sensores que existem no mercado podem ser propriamente calibrados de modo a transformar estes valores de *pixels* em unidades como o metro ou polegada ou outra coisa qualquer[8].

  O sensor usado para produzir estes resultados pode usar diferentes técnicas, como:

  - Triangulação *Stereo* – este método é implementado a partir da profundidade dos *pixels* onde esta profundidade é determinada a partir de um *setup* duas ou mais câmeras. Este método não necessita de condições

especiais de iluminação e a obtenção da nuvem de pontos pode ser adquirida usualmente com duas câmeras ou com um equipamento que já dispõe de duas câmeras integradas.

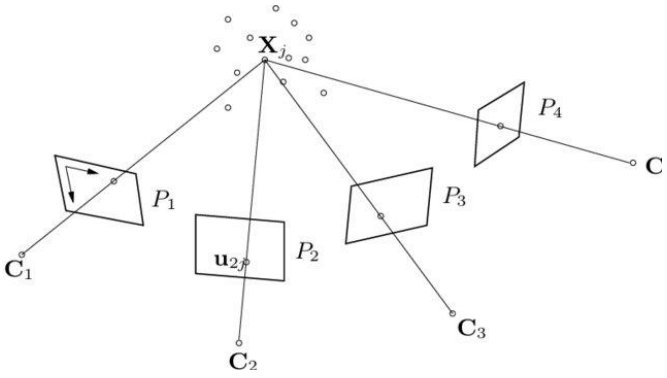


Figura 17 - *Setup* multi câmera para aquisição de nuvens de pontos, Fonte: [9]

- Triangulação de feixe de luz – a cena é iluminada com feixes de luz que criam linhas na cena e estas são refletidas. Sendo assim, sabendo as posições e orientações da câmera e da fonte de produz os feixes de luz é possível calcular a distância entre os pontos refletidos e a câmera ou a fonte de luz.

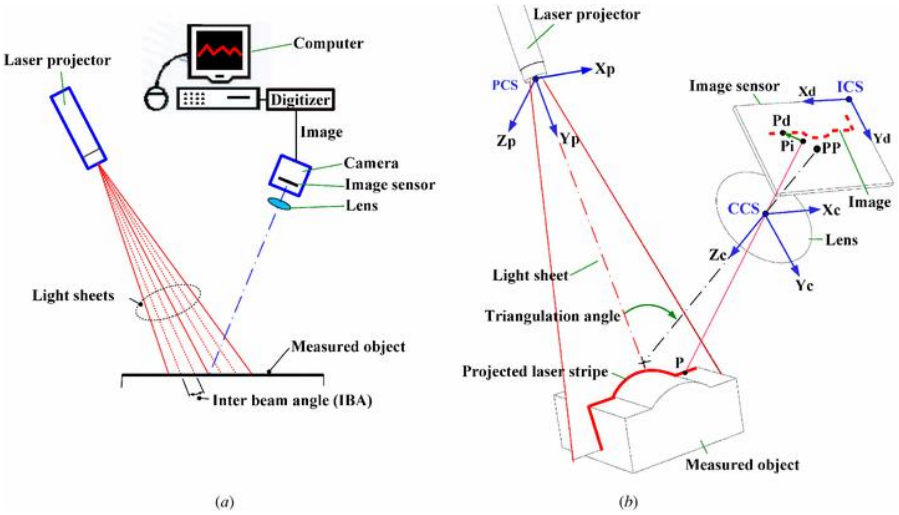


Figura 18 - *Setup* câmera + fonte de luz para aquisição de pontos, Fonte: [10]

- Luz estruturada – A distância dos diferentes pontos à câmera é calculada iluminando a cena com um padrão de luz especial.

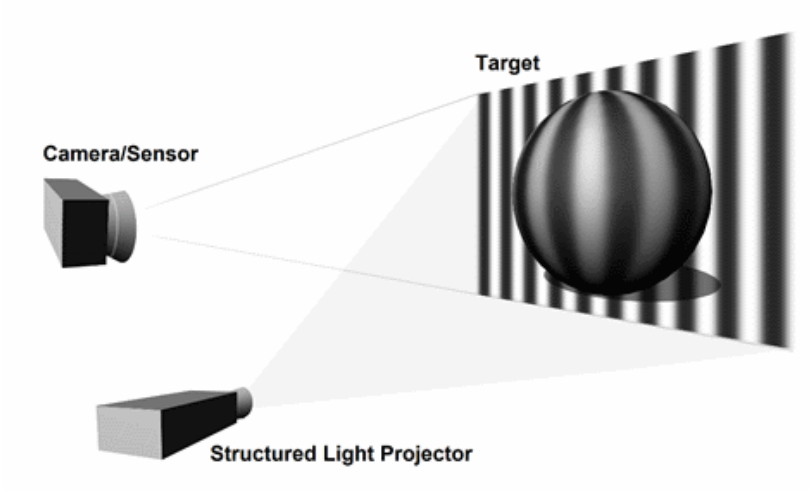


Figura 19 – *Setup* câmera + luz estrutrada para aquisição de pontos para a representação da esfera, Fonte: [11]

- Tempo de voo – A profundidade é calculada da câmera a cada *pixel* com o tempo de voo, similar a um radar mas ao invés de um pulso de radiofrequência, é usado um pulso de luz.

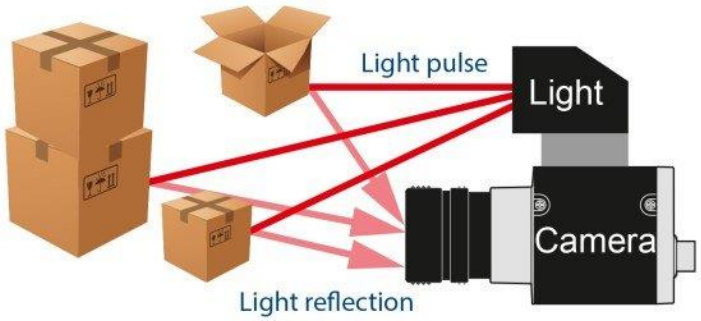


Figura 20 - *Setup* usando a técnica tempo de voo para a aquisição de pontos, Fonte:[12]

Com base nas técnicas mais usadas para a construção de câmeras 3D, a seleção do equipamento foi uma câmera com bastante aplicação no mercado que oferece uma elevada precisão de captura, designada por *Zivid One Plus L*.

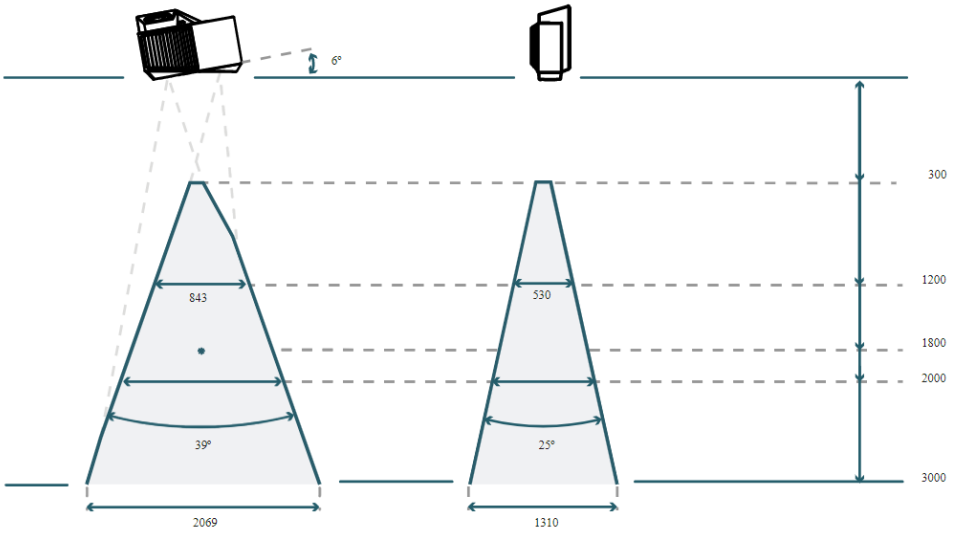


Figura 21 - Campo de visão da câmera, Fonte: [13]

ESPECIFICAÇÕES DA CÂMERA 3D

ALCANCE	1.2 – 3.0 m
ALINHAMENTO FINAL CONVERGIU	0.3 - <2.0 mm
CAMPO DE VISÃO	843 x 530 mm 2069 x 1310 mm
PRECISÃO PONTUAL	225 µm
PRECISÃO DIMENSIONAL	100 µm

Tabela 1 - Especificações da câmera 3D

As únicas limitações que este equipamento tem são: não capta objetos em movimento, objetos transparentes e superfícies altamente refletoras.

- Projetor Laser – o projetor laser selecionado para o desenvolvimento do trabalho é o *Medialas ILP 622* é uma solução de projeção laser universal para integrar em soluções industriais em que a de projeção é muito rápida e tem uma baixa cintilação e alta flexibilidade. Embora a potência do laser seja baixa, este projeta imagens de alto contraste mesmo em condições de luz ambiente como é o caso usual da indústria.

  A característica diferenciadora destes projetores é o suporte direto e transferência de arquivos de programas CAD e ainda pode projetar números, letras, carateres, textos, linhas vetores e até figuras ou desenhos[14].

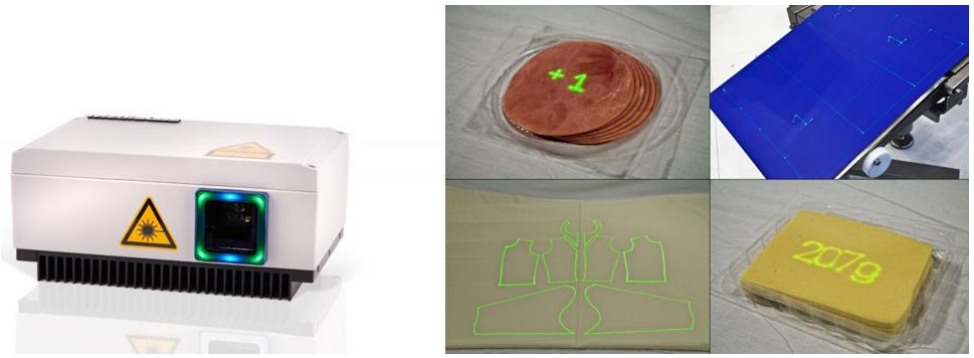


Figura 22 - Esquerda: projetor laser. Direita: resultados exemplo, Fonte: [14]

- Computador – Máquina standard capaz apenas de processar os dados necessários para a execução do sistema. Nota: é importante ter em conta que terá que ter uma capacidade de cálculo elevada e que lida muitas vezes com operações complexas dada a natureza do problema (imagens 3D, muitos pontos para analisar, etc.).

Capítulo 4

Implementação

Como descrito no capítulo introdutório, depois de pesquisa e estudo do problema foi desenhada uma possível solução para o projeto proposto e a partir daí foi concebido e implementado em código essa solução que será validada e testada de modo a conferir se serve para o propósito da aplicação.

Como todos os grandes problemas (até mesmo os pequenos) para serem solucionados é necessário ou é aconselhável recorrer à premissa muito aplicada à informática: "Para solucionar um problema complexo basta subdividir em vários pequenos e simples problemas".

Em concordância com o parágrafo anterior essa premissa foi aplicada e este projeto que contém três grandes partes:

1. Tratamento de dados da nuvem de pontos e cálculo da deformação existente entre a peça real e o modelo CAD.
2. "*Bridge C++/CLI*" para possibilitar a transferência de informação entre níveis de programação.
3. Tratamento e criação de dados artificiais dos ficheiros CAD de modo a melhorar a adaptação e fiabilidade do sistema.

Estas três partes do trabalho realizado foram desenvolvidas independentemente e ligadas entre si passando naturalmente primeiro por testes de validação e só depois dessa validação foram interligadas entre si de modo a fornecer a solução completa.

A primeira parte do projeto aborda o tratamento de dados da *point cloud* e da projeção das entidades tendo em conta a deformação existente entre a peça real e o modelo CAD. Esta é a parte em que se investiu mais tempo a desenvolver, e consequentemente é a parte com mais enfoque neste documento. Para além do projeto estar dividido em três distintas secções, esta secção também será subdividida em várias pequenas partes e serão devidamente explicadas todas as etapas posteriormente. Ainda antes desta descrição do algoritmo desenvolvido, é necessário introduzir uma breve explicação à principal ferramenta usada para

a conceber este trecho do trabalho. A ferramenta usada e desenvolvida em linguagem C++ é uma biblioteca *open-source* utilizada e desenvolvida por as mais variadas empresas e instituições reconhecidas internacionalmente, tais como, *Toyota, Leica, nVidia, intel, MIT,* entre outras, e ainda por utilizadores comuns ou simplesmente por entusiastas, designada por *PCL – Point Cloud Library*.

Esta biblioteca abrange um leque enorme de ferramentas para processamento de imagens e nuvens de pontos, seja para 2D ou 3D. A estrutura desta ferramenta dispõe de inúmeros algoritmos eficientes, testados e validados para filtragem, estimação de características, reconstrução de superfícies, registo de *point clouds*, *model fitting* e segmentação de imagem. Estes algoritmos podem ser usados, por exemplo, para filtrar *outliers* de imagens com ruído, para unir duas ou mais *point clouds*, para segmentar partes importantes de uma imagem 3D, para a extração de pontos chaves, para a criação de *descriptors* para reconhecer objetos baseados na sua geometria e para criar superfícies com base em *point clouds* e visualizá-las.[15]

A segunda parte consiste na transferência de informação entre a camada inferior que está descrita no ponto 1. e a camada superior que está descrita no ponto 3. Esta secção é um ponto essencial no trabalho porque permite articular a simplicidade da programação em C# (aplicações gráficas, uso de software de alto nível) com a eficiência da programação em C++ onde está desenvolvida a maior quantidade de trabalho para a solução.

A terceira parte foi desenvolvida com a ideia de testar o sistema a funcionar por completo, ou seja, previamente na apresentação do projeto é anunciado que a solução tem de passar por uma interface com o operador que trabalhará com este sistema e como se sabe a priori que a aplicação em questão é desenvolvida usando a linguagem de programação em C#, pois foi criada uma aplicação similar para testar e simular o sistema a funcionar na sua totalidade e para posteriormente será mais fácil e rápido a transgressão para a interface real, adaptando-a com ligeiras alterações para o sistema desenvolvido.

4.1 - Tratamento de dados da nuvem de pontos e consequente abordagem da deformação existente entre modelo real e modelo CAD

Uma vez que o problema abordado neste subcapítulo está subdividido em primeira instância. Esta secção servirá somente para a análise e descrição do ponto 1. enunciado no início do capítulo 4.

Para que o tratamento de dados da *point cloud* e o cálculo da sua deformação fosse realizado com eficiência e eficácia este problema foi fragmentado da seguinte maneira:

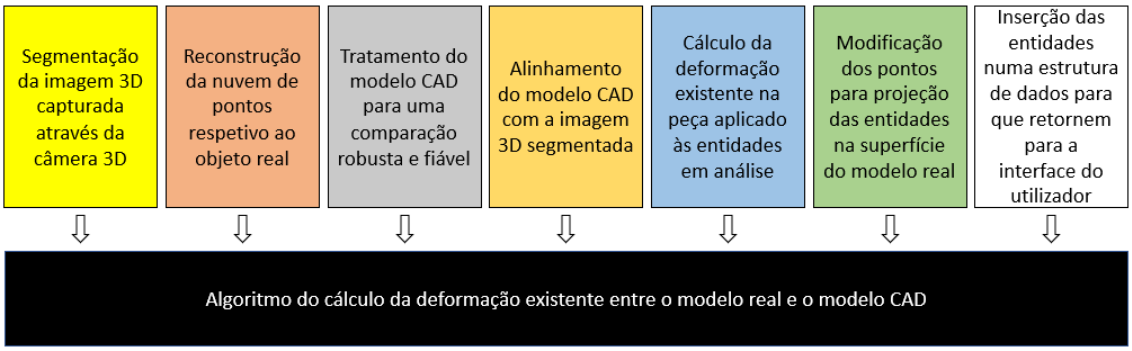


Figura 23 - Divisão do algoritmo geral do cálculo da deformação existente entre modelos

Nesta abordagem, os problemas tornam-se mais objetivos e mais fáceis de resolver e ainda se pode tirar partido da fragmentação porque cada etapa pode ser perfeitamente desenvolvida e testada independentemente com uma enorme abstração. O processo para a solução de cada objetivo da totalidade dos sete enunciados será descrito imediatamente abaixo deste parágrafo.

4.1.1 Segmentação da imagem 3D capturada

Primeiramente, para a correta análise desta parte do trabalho é necessário entender o Setup real e as suas limitações, como é demonstrado no capítulo 3.

O objeto (tampo de um silo, figura 24) irá estar no topo de uma bancada com a câmara 3D precisamente em cima desta apoiada num pequeno "pórtico" (como demonstrado na figura 16 do capítulo 3) e para simplificação do algoritmo de segmentação, a câmara será calibrada tendo em conta a bancada de trabalho considerando, por fim o topo da bancada como a origem do referencial cartesiano tridimensional.



Figura 24 - Objeto de análise, protótipo de um tampo de um silo

De modo a segmentar o ruído basta utilizar um filtro passa banda com o limite mínimo na origem do referencial e com o limite máximo pouco superior à altura da peça, como mostra a figura 25.

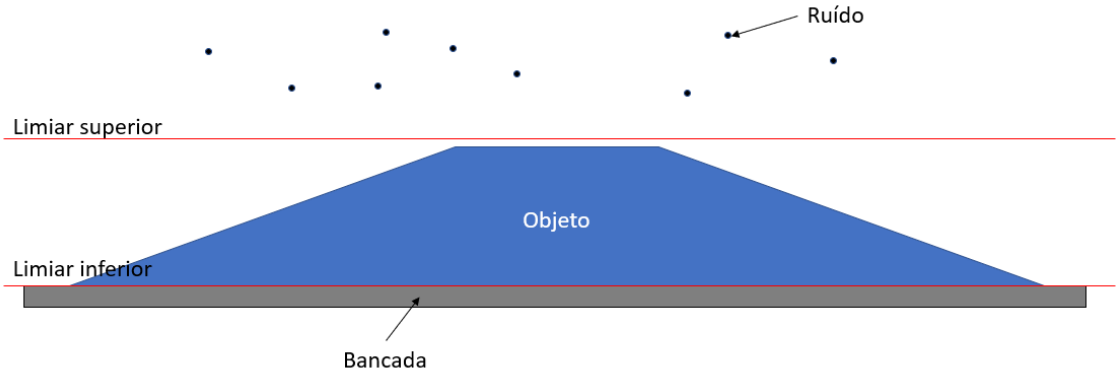


Figura 25 - Ilustração explicativa do filtro passa banda

Para obter o tipo de resultado da figura 25 foram utilizadas as seguintes instruções:

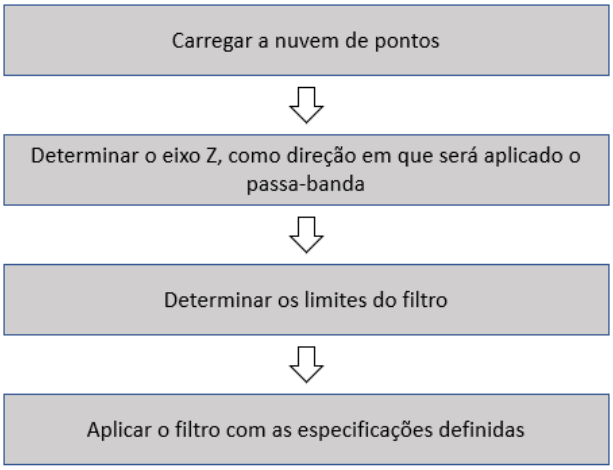


Figura 26 – Esquemático do filtro passa banda

Primeiramente, foi inicializado o tipo de filtro a utilizar, de seguida definiu-se a *point cloud* para a filtragem, depois foi dada a instrução de aplicar os limites do filtro ao eixo Z do referencial predefinido anteriormente, logo declara-se os limites e por fim, aplica-se o filtro e essa filtragem será carregada no argumento da função, que representa uma nuvem de pontos.

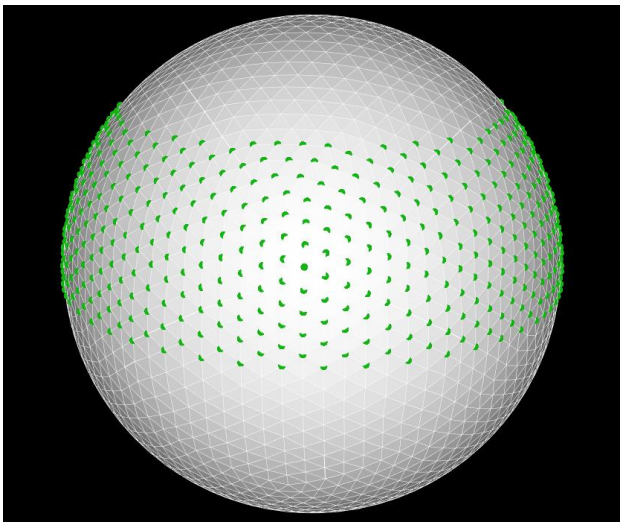


Figura 27 - Resultado exemplo da aplicação do filtro passa banda

Os pontos a verde representam o resultado do filtro passa banda aplicado à *point cloud* da esfera representada na figura 27.

Posteriormente e com o objeto em análise totalmente segmentado porque este filtro é suficiente porque assume-se que as condições do setup e as imagens capturadas pela câmara 3D para os limites do filtro são claras e sem ruído envolvente, ou seja, a altura em que o objeto se encontra não tem nenhum ruído, de qualquer das formas e por precaução pode ser utilizado um filtro SOR, contudo e para a aplicação em causa vai ser utilizado ainda um filtro de downsampling.

"SOR (Statistical Outlier Removal)" é um filtro que pode ser usado em PCL. Esta ferramenta consiste no cálculo da distância média de cada ponto aos seus vizinhos (considerando o parâmetro k, sendo k o número de vizinhos mais próximos do ponto em questão). Então, são rejeitados todos os pontos acima da soma entre a média e um segundo parâmetro, que representa um certo número de vezes o desvio padrão[16].

Considerando as variantes do sistema, foi concluído que o filtro para a redução de amostra é imprescindível, *"downsampling filter"*. A explicação para a utilização deste filtro ser absolutamente necessário baseia-se em limitações de *hardware*, ou seja, a nuvem de pontos capturada diretamente pela câmara tem uma imensidão de pontos que não permite à máquina processar a imagem eficientemente, fazendo com que o sistema se torne impossível, dado o elevado tempo de processamento. Para corrigir isso tem que se retirar qualidade à imagem tanto quanto possível, e essa redução deve ser feita de modo a não pôr em causa a fiabilidade do sistema na solução final mas também deve ser o mais pequena possível para eficiência e redução de tempos do programa, sendo assim necessário encontrar um valor ótimo.

Posto isto, a implementação deste filtro baseia-se num algoritmo em que se começa por inicializar uma *voxel grid*, esta *voxel* grid são como um conjunto de pequenos cubos 3D no espaço tridimensional e os pontos vão estar dentro dos *voxels* (cubos 3D) e para o *downsampling* ser aplicado é calculado todos os pontos dentro de um *voxel* e são substituídos pela sua *centroid* (centro da densidade de pontos presentes no voxel). Naturalmente, este processo é um pouco mais lento do que arrastar todos os pontos para o centro de cada *voxel* contudo para a representação de superfícies é mais preciso o primeiro método e evita rugosidades que não se verificam na realidade.

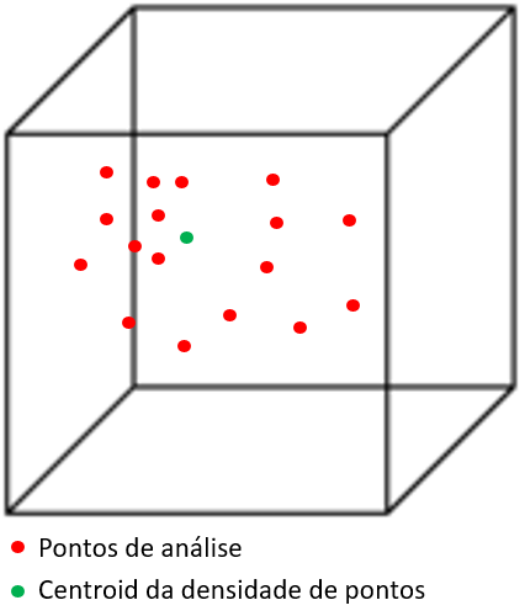


Figura 28 - Ilustração representativa de um *voxel* aplicado ao filtro de redução de amostragem

Para efeitos de validação deste filtro foram feitos testes como demonstram as figuras 29 e 30:

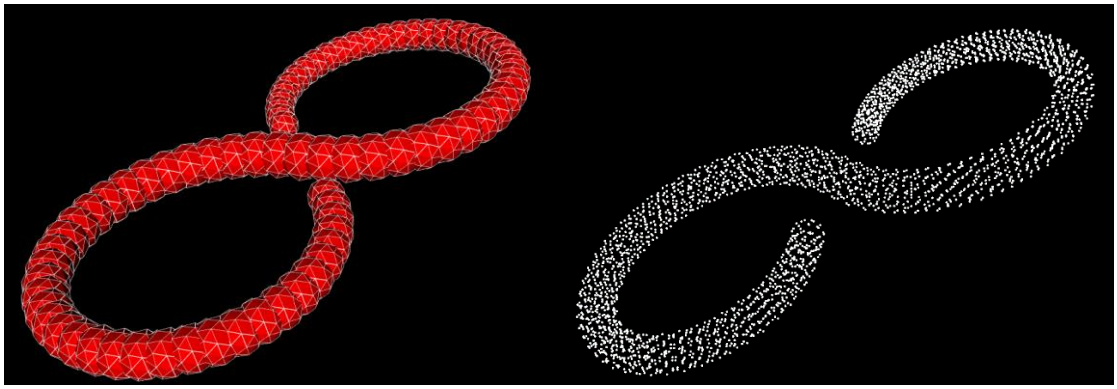


Figura 29 - Malha do objeto em análise (à esquerda) e respetiva nuvem de pontos (à direita)



Figura 30 - Resultado da aplicação do filtro *downsampling*

Aplicando o filtro discutido nesta secção em PCL com parâmetros de redução substancialmente grandes e por efeitos de demonstração apenas, é possível observar que sobre

a *point cloud* original existem uma point cloud com os pontos representados a verde que representam o resultado do filtro e este tem visivelmente menos pontos que a nuvem de pontos original, como seria expectável.

4.1.2 Reconstrução da *Point Cloud*

Com a segmentação da peça finalizada e com características que permitem à máquina a capacidade de processamento necessária para avançar. O próximo passo é criar uma malha a partir dos pontos obtidos. Existem diferentes abordagens para este problema como por exemplo, triangulação *Delaunay* ou *Poisson surface reconstruction*.

Para a seleção do melhor método foi usado *software* externo para ser mais eficiente na leitura dos resultados, *software* esse designado por *Meshlab* e *CloudCompare*. Os testes foram efetuados para os dois métodos anunciados acima porque segundo estudo prévio são estes dois que mais se apropriam à aplicação do sistema sendo os mais estudados e utilizados.

Enquanto a triangulação Delaunay não pode ser usada na maioria dos casos porque é muito sensível ao ruído e é necessário uma nuvem de pontos muito limpa para usar este método, o que na prática é complicado de encontrar este tipo de *point clouds*, no entanto, neste caso pode ser usada e os resultados não são péssimos. Contudo, a superfície fica muito rugosa por causa do registo das diferentes partes do tampo o que pode provocar depois uma má projeção dos pontos na superfície. Ainda assim foi considerado os resultados deste método à aplicação em estudo e comparado com o método de *Poisson*.

Discutindo neste parágrafo a reconstrução de superfícies de *Poisson*, esta faz com que a malha seja criada com um grau de suavidade (*"smoothness"*) elevado dado o seu parâmetro *"reconstruction depth"*. Outra vantagem que não é aplicada para este projeto (porque o objeto em questão não contém estas características mas que poderá ser útil em algum momento da aplicação bastando o objeto de análise ser diferente) é o facto de ser possível ser usado para superfícies com furos ou buracos. O grande desafio deste método é obter um bom conjunto de normais e caso isso não aconteça os resultados serão desastrosos.

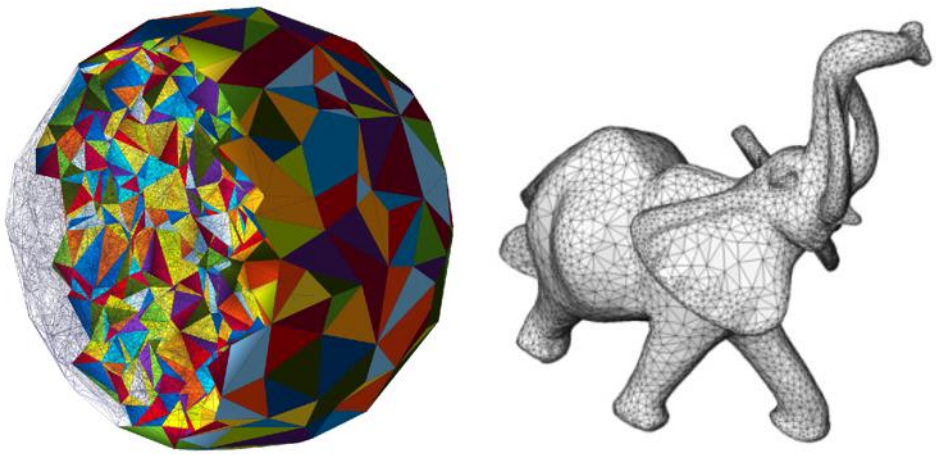


Figura 31 - Esquerda: triangulação *Delaunay* que apresenta muito menos suavidade e polígonos muito mais irregulares; Direita: método de triangulação de *Poisson* que apresenta uma suavidade maior e uma melhor regularidade dos polígonos, Fontes: [17], [18]

Comparando os dois resultados a escolha mais sensata por tudo o que foi descrito acima é o método de *Poisson*, embora seja possível o uso da triangulação de *Delaunay*. O método de *Poisson* também se adequa mais até pela forma como os triângulos são gerados, porque são mais regulares e facilitará no algoritmo de projeção por causa do número de triângulos a compartilhar num vértice. Em Poisson geralmente o máximo de triângulos a partilhar o mesmo vértice é seis enquanto em Delaunay pode ser vários porque a forma de gerar os polígonos é diferente e pode apresentar triângulos completamente irregulares (o que vai afetar substancialmente o eficiência do algoritmo mais a diante).

Em primeira instância, para a implementação do algoritmo de reconstrução de Poisson, este necessita das normais orientadas para todos os pontos da nuvem de pontos em análise. A extrapolação das normais é feita automaticamente por ferramentas da biblioteca utilizada que visa determinar uma normal por um plano tangente à superfície, o que se torna num problema de estimação de ajuste do plano pelos mínimos quadrados. Contudo, por uma questão de eficiência, a solução que é implementada pelas funções do PCL para a estimação das normais às superfícies é reduzida a uma análise de valores e vetores próprios de uma matriz de covariância criada a partir dos vizinhos mais próximos do ponto em questão.[19] Mais especificamente, para cada ponto, é criada uma matriz de covariância como sugere a seguinte equação (1):

$$C = \frac{1}{k} \cdot \sum_{i=1}^{k} (p_i - \bar{p}) \cdot (p_i - \bar{p})^T, \qquad C \cdot \vec{v_j} = \lambda_j \cdot \vec{v_j}, \qquad j \in \{1, 2, 3\} \tag{1}$$

Onde k é o número de pontos vizinhos considerando a vizinhança de p_i, $\bar{p}$ representa a *centroid* de todos os vizinhos próximos e λ e $\vec{v}$ representam os valores e vetores próprios, respetivamente[19].

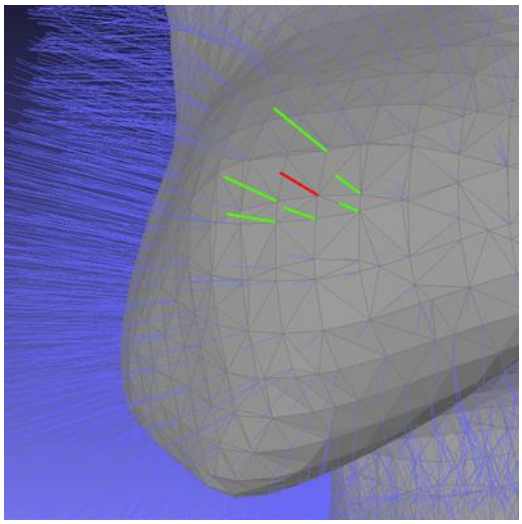


Figura 32 - Extrapolação de vetores normais, Fonte: [20]

O algoritmo calcula as normais a partir da matriz e covariância, como um produto ponderado a partir de k vizinhos. Considere-se para efeitos de exemplares a figura 32, a normal representada a vermelho é o vetor que está a ser calculado com base em seis vizinhos representados a verde.

Com esta abordagem as normais são calculadas considerando só a sua norma porque a orientação calculada com este método é ambígua e não é orientada de forma consistente porque não existe forma matemática para resolver este problema e a maneira encontrada pelos desenvolvedores do PCL baseia-se no simples facto de considerar o ponto de vista da imagem, ou seja, como o ponto de vista é conhecido por causa da câmara (é conhecido o ponto espacial da câmara) e neste caso é estático (a câmara não se move).[19] Para orientar as normais $\overrightarrow{n_i}$ consistentemente em direção ao ponto de vista, basta se reger por esta condição:

$$\overrightarrow{n_i} \cdot (v_p - p_i) > 0 \tag{2}$$

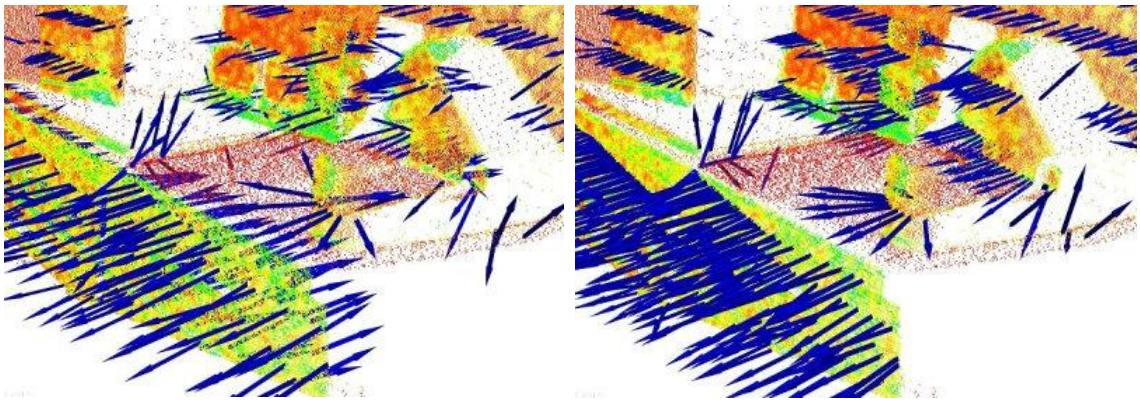


Figura 33 - À esquerda normas calculadas com sentido das normais ambíguas e à esquerda normais calculadas com sentido consistente, Fonte: [19]

Como demonstra a figura 33, a abordagem para solucionar o problema do sentido dos vetores normais é eficaz e simples. Basta considerar o ponto no referencial onde foi capturada a imagem e depois o cálculo do sentido das normais torna-se consistente como demonstrado.

Depois desta extrapolação das normais, é calculado o gradiente e de seguida a função indicadora é aproximada ao campo normal. A função indicadora é zero em todo lado exceto perto da superfície, é derivada a partir do gradiente e deste modo a superfície do objeto pode ser extraída.[21], [22] A figura 34 demonstra a explicação acima:

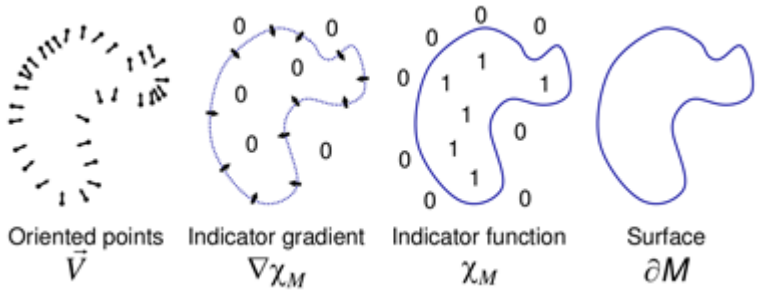


Figura 34 - Extração da Isosuperfície, Fonte: [21]

Para a extração da superfície é usado uma variante do algoritmo *"marching cubes algorithm"* que é usado para criar uma estrutura octree ((Oct + Tree ou árvore de oito) é uma árvore, onde cada nó que não seja folha possui interligação com mais outros oito nós da estrutura de dados[23]) para a representação da superfície.

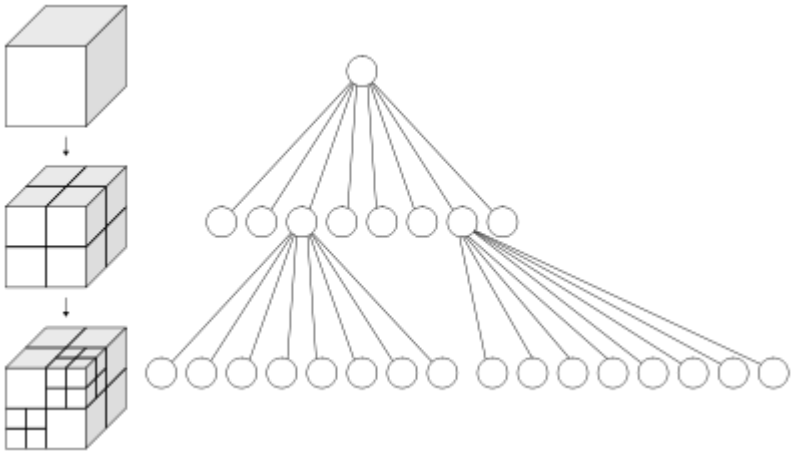


Figura 35 - Exemplo de octree, Fonte: [23]

O algoritmo *"marching cubes"* foi concebido para finalidade na área da saúde, para visualização médica, como tomografia computacional axial e leitura de imagens por ressonância magnética[24]. A implementação do algoritmo começa por dividir o espaço tridimensional em um determinado número de cubos (*voxels*). Depois todos os cubos são testados para ver se estão dentro ou fora dos limites do objeto, este teste é feito com base na análise de cada vértice do *voxel*. Os cubos que terão importância para determinar a isosuperfície são aqueles em que alguns dos vertices estão dentro dos limites e outros se encontram fora dos limites, pois isso dá a informação que a superfície está algures dentro desse cubo e que atravessa as arestas do *voxel*. Finalmente, é descoberta a interceção da superfície com as arestas dos cubos formando um conjunto de pontos. Este algoritmo é a versão clássica porém a reconstrução de *Poisson* como foi descrito acima usa uma variante deste algoritmo com octrees aquando da descoberta das arestas que têm a interceção da superfície porque

assim vão comprimindo a informação 3D tornando-a mais eficiente para a manipular para a triangulação[25].

Em suma, este algoritmo divide a *point cloud* numa *Voxel Grid* e é feita uma análise de quais pontos definem a isosuperfície do objeto. Posto isto, a partir dos pontos calculados para a definição da isosuperfície são criados os triângulos da malha, obtendo deste modo a reconstrução total da superfície em análise.

Dada a explicação acima, é possível entender como este algoritmo de reconstrução funciona e de forma a pormenorizar e a aprofundar conhecimento em relação aos seus resultados abaixo será enunciada os seus principais parâmetros e suas consequências e propósitos.

1. **Profundidade (“*Depth*”):** A profundidade da árvore usada para a reconstrução da superfície. O valor standardizado pelo *PCL* é oito, isto significa que a árvore contém oito níveis. Quanto mais níveis a árvore tiver, mais detalhados vão ser os resultados mas agregado a isto está mais capacidade de processamento, mais tempo investido nesta operação. A razão pela qual isto acontece é, quando é formada a *Voxel Grid* a partir do algoritmo dos “*marching cubes*” este parâmetro quanto maior for maior será o número de *voxels*, o que implica automaticamente que maior será o detalhe do modelo. Por outro lado, se este parâmetro apresenta um valor baixo, inerente ao raciocínio anterior, a *Voxel Grid* vai ter um número mais reduzido de *voxels* o que irá dar mais suavidade à superfície, contudo esta perderá detalhe (rugosidade)[22].
2. **Amostras por nó (“*Samples per Node*”):** Este parâmetro determina o número mínimo de pontos que devem estar dentro dos diferentes nós da o*ctree*, pois esta depende da amostragem. Um valor alto de pontos de amostragem fornece uma suavidade mas está diretamente associada a uma perda de detalhe porque quando se atribuí muitos pontos de amostra a um único nó, necessariamente irá haver perda de detalhe, isto é, um número elevado reduz a contagem resultante de vértices e quando existe essa redução, a perda de detalhe irá estar na consequência desta medida. Enquanto um número baixo de pontos de amostras o detalhe vai ser realçado. Mais importante ainda é entender quando se deve ter pouco ou muito detalhe pois isto pode vir a tornar-se um problema futuramente, quer por tempos de análise de *point clouds*, quer por falta de detalhe para a resolução do problema final, por isso é importante encontrar um meio-termo para que a eficiência e eficácia do sistema não seja comprometida[22].

Existem mais parâmetros que é possível determinar contudo não têm um impacto muito grande no resultado da malha final. Os dois parâmetros enunciados acima apresentam um grande impacto no resultado final quando alterados.

Para efeitos de validação desta parte do sistema foi revisto vários exemplos em que é testado o algoritmo e a implicação das suas variáveis no resultado final. A figura 36 demonstra exatamente isso e demonstra também a qualidade dos resultados revelados que sustentam a escolha do algoritmo para a reconstrução na aplicação em causa.

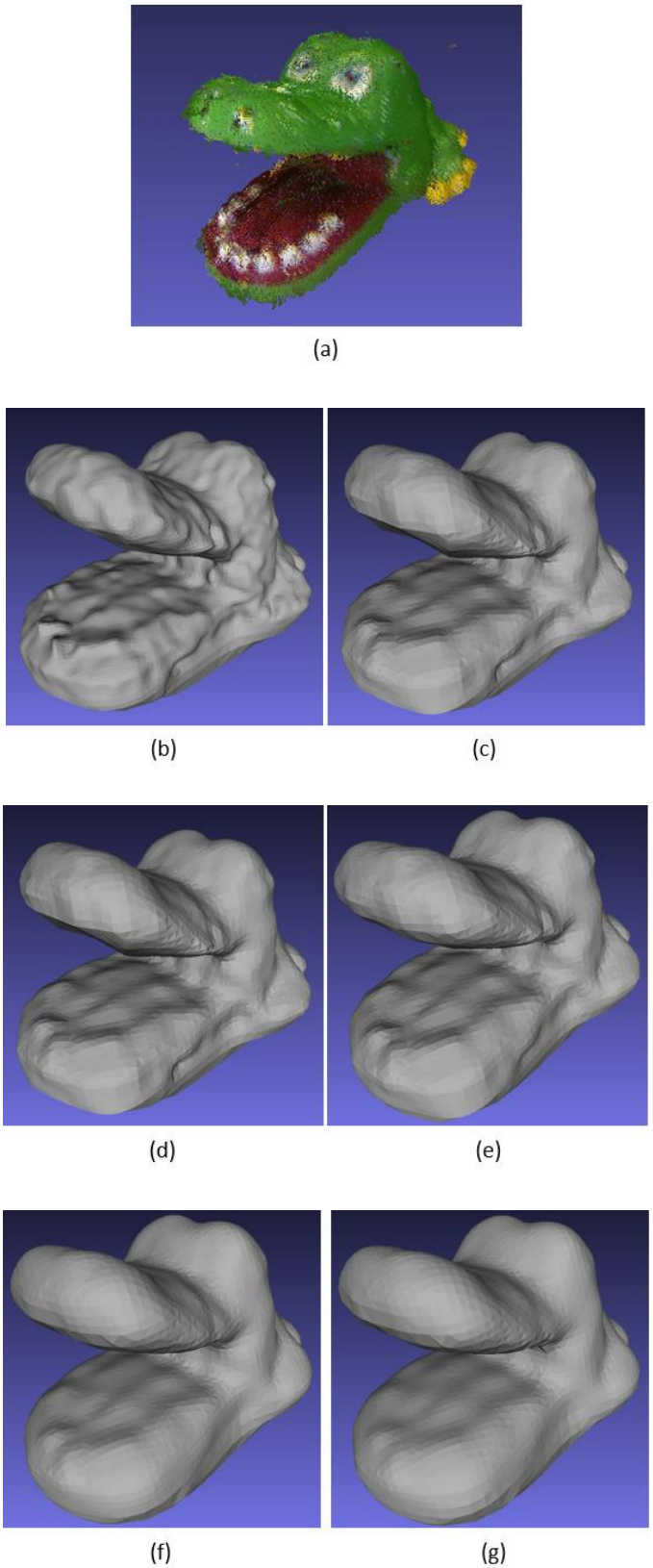


Figura 36 - Resultado dos testes de validação do algoritmo de reconstrução, Fonte: [22]

	Profundidade	Amostras por nó	Faces	Vértices
Figura 36 (b)	8	1	49794	24899
Figura 36 (c)	6	1	12546	6275
Figura 36 (d)				
Figura 36 (e)	6	5	12640	6322
Figura 36 (f)	6	15	11662	5833
Figura 36 (g)	6	20	10512	5258

Tabela 2 - Tabela de *benchmarking* do algoritmo usado

A figura 36. (a) representa uma nuvem de pontos de uma imagem 3D com algum ruído capturada a partir de um *kinect*. Esta nuvem de pontos contém 89326 pontos e será usada para *benchmark* do algoritmo.

Reduzindo o valor da profundidade de oito para seis como acontece na transição da figura 36 (b) para a figura 36 (c) nota-se que existe uma elevada perda de detalhe mas associado a isso obtém-se uma superfície mais suave que é uma vantagem quando se tem imagens ruidosas como é esta nuvem de pontos.

Uma vez que a compreensão do parâmetro de profundidade está percebido resta testar o efeito ao alterar a outra variável que representa o número de amostras por nó.

Agora, ao aumentar gradualmente o número de amostras por nó como acontece nas seguintes figuras: figura 36 (d), figura 36 (e), figura 36 (f) e figura 36 (g). A consequência deste aumento como era expectável com a explicação teórica do algoritmo é a obtenção de uma superfície ainda mais suave e com menos detalhe, o que implica a diminuição gradual do número de vértices obtidos a partir do algoritmo de *Marching Cubes* e consequente diminuição do número de triângulos.

4.1.3 Tratamento do modelo CAD

Uma vez segmentada e reconstruída a imagem 3D proveniente do objeto real é necessário analisar o modelo CAD e identificar mudanças para que facilitem o alinhamento e a comparação entre a *point cloud* do modelo real e do modelo CAD.

O modelo CAD é selecionado na interface do utilizador e este faz o carregamento em ficheiro. STL (é uma abreviatura para estereolitografia), este tipo de ficheiros é muito usado para prototipagem, impressão 3D, *CAM ("computer-aided manufacturing")*. Ainda, este tipo de ficheiros descreve o modelo CAD como um sólido constituído por triângulos que são formados por vértices que têm como atributos coordenadas do sistema referencial cartesiano (dado importante porque usa o mesmo sistema de coordenadas da *point cloud* o que faz com que não seja necessário nenhum tipo de conversões ou transformações de sistemas referenciais)[26].

O problema encontrado quando se observou e estudou os modelos CAD foi o facto de estes apenas considerar um número reduzido de vértices para delimitações e representação de

entidades do modelo. De modo a obter uma solução para este problema porque os algoritmos de alinhamento entre as duas nuvens de pontos não funcionariam porque não existiam semelhanças suficientes, ou pontos suficientes para a execução do algoritmo e incorreria num péssimo alinhamento. Identificado o problema, concluiu-se que seria necessário fazer um aumento de vértices no modelo CAD de maneira a obter um número, com a ordem de grandeza similar ao modelo real a ser comparado posteriormente.

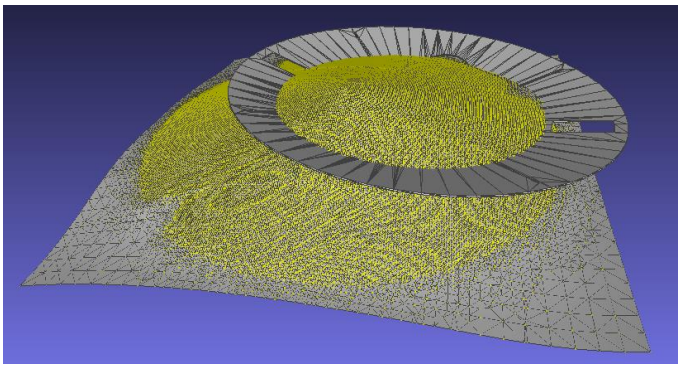


Figura 37 - Resultado do alinhamento sem *upsampling* do ficheiro STL

A biblioteca *VTK (Visualization Toolkit)* é uma ferramenta que complementa as funcionalidades do PCL e para este algoritmo foi expressamente necessário porque simplifica contendo funções e classes predefinidas para o fim que se pretende.

De seguida, aplicou-se uma classe para filtrar a informação poligonal de modo a gerar os triângulos (*"vtkTriangleFilter"*) do modelo CAD e de seguida mapeou-se esta informação com o uso da classe *"vtkPolyDataMapper"*, com o objetivo de facilitar o uso desta estrutura. Com a preparação da informação inicial foi chamada a função principal deste algoritmo que visa aumentar os vértices do modelo CAD para um número desejável dada a aplicação do sistema. Os próximos parágrafos explicarão detalhadamente os passos para a obtenção de resultados.

A função principal, tem como argumentos de entrada, a informação poligonal descrita acima, o número de pontos que se pretende criar e a *point cloud* que será o resultado final. Primeiramente, é construído um vetor que contém todas as áreas dos diferentes triângulos sendo o seu índice o identificador de célula do triângulo, isto é feito através de um varrimento a todas as células do polígono e calculando todas as áreas dos triângulos escrevendo nesse vetor iterativamente. Para além disso ainda é calculado a área total de triângulos com uma soma cumulativa ao longo do ciclo.

Finalmente, depois de obtido todas as informações, é programado um ciclo que será executado um número de vezes igual ao número de amostras que se pretende inserir na *point cloud* todavia para isso é necessário passar por um último algoritmo que calcula as coordenadas exatas para a adição do ponto. Uma vez mais, os argumentos desta função são a informação poligonal, as áreas dos triângulos, a soma das áreas dos triângulos e o ponto para inserir na *point cloud* final. De seguida é calculada, de maneira relativamente aleatória, as coordenadas do novo ponto a partir de números dinâmicos obtidos com o produto entre valores das áreas e números aleatórios. Embora exista aleatoriedade ao gerar os pontos, este processo

implementado de maneira iterativa fará com o que o objeto seja uniformemente preenchido pelo número de pontos que se definiu a priori.

Posteriormente, é criada uma *Voxel Grid* com a nuvem de pontos obtida do aumento de amostras e sobre essa nuvem de pontos é aplicada uma função que define o tamanho da V*oxel Grid* com um tamanho ajustável. A finalidade desta parte do algoritmo visa a obtenção de resultados mais regulares entre pontos da nuvem de modo a não haver disparidades entre superfícies por causa da aleatoriedade de inicial de adição de pontos.

Por último esta *Voxel Grid* criada anteriormente é transformada numa *point cloud* final de modo a deixar o resultado obtido numa estrutura preparada para ser implementada em passos seguintes. Como forma de demonstração gráfica, as figuras abaixo mostram o algoritmo em funcionamento para a criação de 100 000 pontos de amostragem e para um *"leaf size"* da *Voxel Grid* de 0.3 para as diferentes dimensões (X, Y, Z).

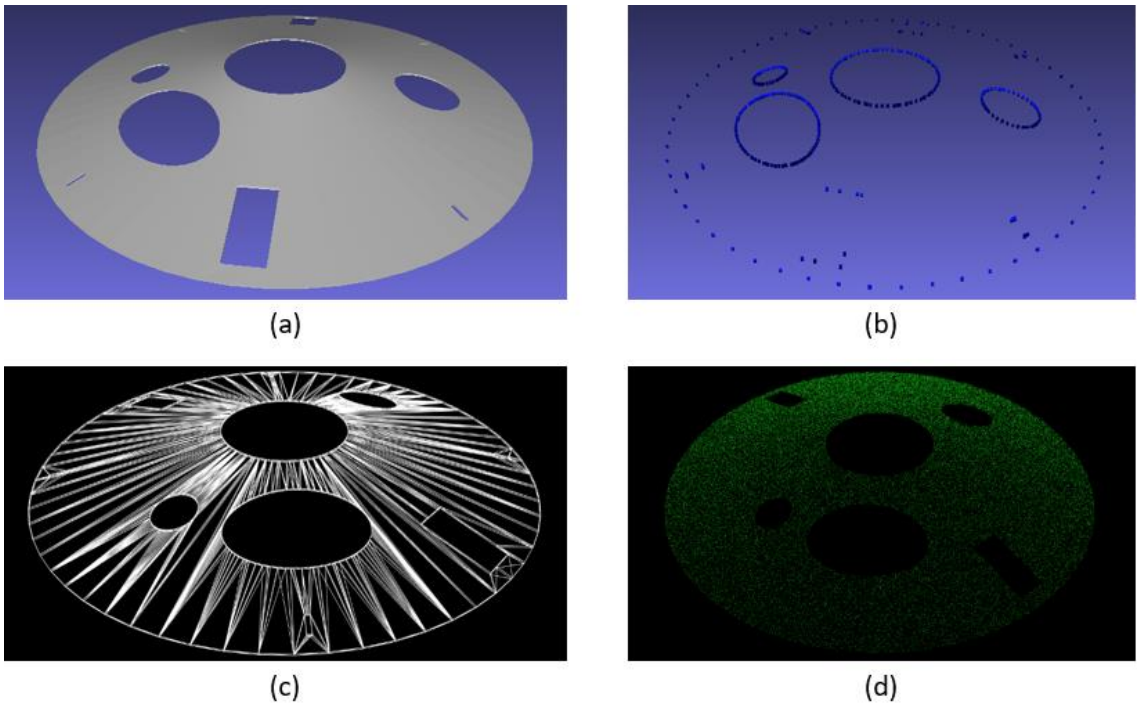


Figura 38 - (a): Modelo CAD visualizado como um sólido; (b): Vértices que definem o ficheiro STL representado em (a); (c): Ficheiro STL representado pelos seus triângulos; (d): Resultado do algoritmo de *upsampling*

4.1.4 Alinhamento do modelo CAD com a imagem 3D segmentada

Esta quarta parte do projeto incide sobre o alinhamento de nuvens de pontos e esta parte é essencial porque vai ser a partir das transformações que se vai conseguir projetar as entidades de pontos sobre a superfície do modelo real. Simplificando e resumindo, as duas nuvens de pontos vão estar aleatoriamente pelo espaço tridimensional e resumidamente uma das nuvens vai ter de se mover segundo uma matriz de transformação de modo a estar perfeitamente alinhada com a outra nuvem.

O algoritmo usado para este fim é um algoritmo já muito estudado e validado com aplicação prática e implementação em PCL designado por *ICP ("Iterative Closest Point")*. O seu objetivo é minimizar a diferença distancial entre duas nuvens de pontos. Usualmente este algoritmo é usado para a reconstrução de superfícies 2D e 3D a partir de diferentes imagens, para a localização de robôs para encontrar o melhor caminho para robôs percorrem uma distância do ponto A ao ponto B, para o registo de modelos ósseos, entre outros[27].

Em síntese, existem duas *point clouds*, uma delas é a referência (esta é como se estivesse fixa no espaço tridimensional) e a outra é a fonte (esta *point cloud* vai sofrer translações e rotações de modo a alinhar-se nas respetivas coordenadas da referência). Isto acaba por ser um pouco indiferente determinar a referência e a fonte porque é possível obter sempre o mesmo resultado mesmo que se troque a fonte pela referência e para isso basta calcular a matriz inversa da respetiva transformação.

Primeiramente, será implementada uma correspondência ICP para que se obtenha um conjunto de pontos relacionais entre as duas nuvens de pontos. Para isso, a *point cloud* tem que sofrer uma transformação rotacional e/ou translacional. Estes pontos relacionais ou pontos correspondentes são aqueles que estão mais perto entre a fonte e a referência depois de correr uma iteração do algoritmo ICP. Esta correspondência de pares de pontos é, numa primeira instância, estimada a partir das normais de cada superfície e depois de calculadas é feito um varrimento por ambas as superfícies de modo a comparar cada normal. Deste modo, é feita um designado pré alinhamento, como mostra a figura 39.

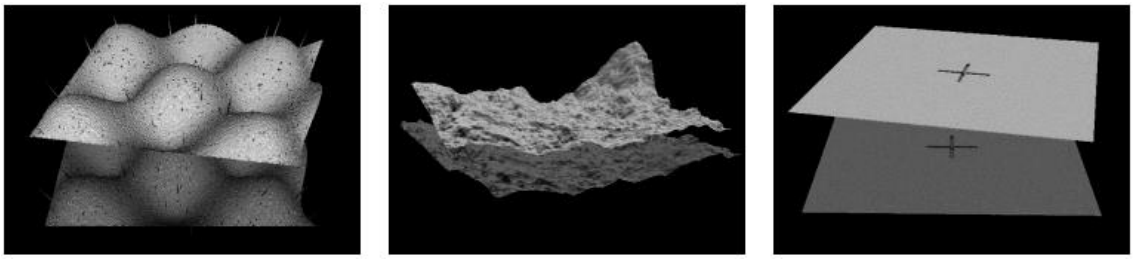


Figura 39 - Pré-alinhamento entre duas superfícies, Fonte: [28]

Logo, o algoritmo ICP calcula a menor distância euclidiana desde o ponto na referência até ao ponto na fonte (pares de pontos obtidos através da estimação das normais). A distância euclidiana entre um ponto da referência, p, e um ponto da fonte q, em que o número em índice na equação representa a coordenada espacial para dimensão n. Neste caso só será usado três índices correspondentes às coordenadas X, Y e Z do espaço tridimensional. A equação abaixo representa matematicamente o que foi descrito no presente parágrafo[29].

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \qquad \textbf{(3)}$$

Depois deste primeiro alinhamento como mostra a figura 40, o problema é encarado como um problema de redução de erro até que este seja notoriamente pequeno para que a projeção de pontos seja fiável posteriormente. Neste ponto do algoritmo já foi completado a

primeira parte atribuindo um conjunto de pares de pontos entre a referência e a fonte, com esta sendo transformada para a mínima distância euclidiana dos pares de pontos.

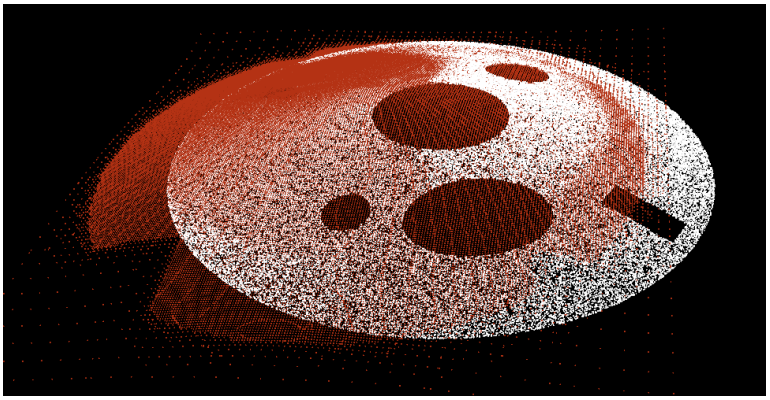


Figura 40 – Pré alinhamento do objeto e consideração dos primeiros pares de pontos

A próxima etapa, visa estimar uma transformação rotacional e/ou translacional para a distância euclidiana calculada para cada pares de pontos, o que para otimização do algoritmo este irá ponderar a correspondência de pontos a partir do seu peso do par e depois será feita uma análise para entender uma eventual rejeição de pares de pontos.

O peso dos pares é um importante dado para a análise do alinhamento porque o resultado deste dado permitirá perceber quais pares de pontos são mais fiáveis e eventualmente possa ser reduzido esses pares rejeitando-os, permitindo um alinhamento perfeito porque não existirá contradições. O objetivo deste dado é caracterizar cada par de pontos e dar um peso que será levado em conta como um par mais relevante ou não.

Existem dois tipos de pesos que devem ser levados em conta, o primeiro é referente à distância ponto a ponto entre pares que diz que quanto mais pequeno for o resultado do peso maior será a distância entre os dois pontos o que pode levar à rejeição deste par se ultrapassar o limiar predefinido na parte da rejeição de pares.

$$Peso_{distancial} = 1 - \frac{dist(p_1, p_2)}{dist_{máx}} \quad \textbf{(4)}$$

O mesmo se aplica às normais à superfície em cada ponto, com a equação abaixo a atribuir este segundo peso ao produto interno dos vetores do par de pontos. Tendo em conta que os cálculos dos produtos internos vão considerar os vetores normais devidamente normalizados, sendo que a única influência será a direção dos diferentes vetores, e o resultado do peso será tanto maior quanto menor for o ângulo entre os dois vetores.

$$Peso_{normal} = \overrightarrow{n_1} \cdot \overrightarrow{n_2} \quad \textbf{(5)}$$

Posto isto, o resultado final do peso é ponderado tendo em conta o peso distancial e normal e depois de calculado passará por uma fase de testes para considerar se os pares existentes são fiáveis e fidedignos e caso sejam permanecem, caso não sejam são rejeitados como explicará os próximos parágrafos.

O propósito desta parte do algoritmo passa por ignorar os pontos dispersos que não interessam para o propósito do algoritmo, neste caso alinhar duas *point clouds*, o que terá um grande impacto de performance. A posteriori irá ser enumerado e explicado todas as condições

de rejeição. A estratégia para esta parte do algoritmo passa primeiramente, por rejeitar os pares de pontos que contém o mesmo ponto da referência mais do que uma única vez, isto fará com que exista apenas o número necessário de pares de pontos e estes pares são atribuídos para cada ponto da referência que apresentar um melhor peso em relação às suas características (peso distancial e normal). De seguida, a rejeição de uma percentagem de amostra baseada na ambiguidade distancial do objeto, por exemplo, é possível ter uma superfície formada por vários pontos e também é possível obter a mesma superfície formando-a com menos pontos, ou seja, a ideia é reduzir os pontos e os seus pares de modo a ter uma amostra mais clara e com menos pontos para processar, para isso exclui-se uma percentagem de pontos e mais uma vez os pesos dos pares são relevantes para esta escolha assim como a distância entre os diferentes pares, para que não haja uma parte do modelo com menos pontos e uma outra parte com uma densidade de pontos muito superior. Com isto é importante ter em conta que o peso dos pares de pontos é importante mas a uniformidade da amostra também o é para que não se perca qualidade e rigor no alinhamento. Outra premissa de rejeição, é eliminar os pares em que os seus pontos estão a uma distância superior a um múltiplo do desvio padrão das distâncias, considerando este desvio padrão a amostra dos pares de pontos sem qualquer rejeição até então. Isto permite que os pontos que estão espacialmente distantes em relação aos outros pontos não afetem a transformação final. Uma das premissas mais importantes para o bom funcionamento do algoritmo é a rejeição de pares de pontos que não são consistentes com a sua vizinhança. Ou seja, se a distância de um par é muito discrepante dos seus pares vizinhos, pois esse par é rejeitado, deve-se perceber qual o número de vizinhos a analisar e qual o limiar a usar. Considerando (p_1, p_2) um par e (q_1, q_2) outro par, o par em análise deve ser rejeitado se for maior que um limiar a parametrizar, como clarifica a inequação (6). A parametrização desta variável deve ser dinâmica de modo a adaptar-se a todos os modelos e a todos os tamanhos de nuvens de pontos porque é possível tratar-se de nuvens de pontos de ordens de grandeza completamente distintas e naturalmente esses parâmetros teriam de ser ajustados, deste modo, é possível considera-los standardizados e dinâmicos para todo o tipo de ordens de grandeza.

$$|dist(p_1, p_2) - dist(q_1, q_2)| < 0.1 \cdot máx(dist(p_1, p_2), dist(q_1, q_2)) \quad \textbf{(6)}$$

O uso desta inequação será aplicada para a análise de um único ponto em N pontos vizinhos e se a inequação for falsa com mais de metade dos pontos vizinhos que se está analisando, pois esse par é rejeitado. Finalmente por último, rejeitar os pares de pontos que contêm os limites da nuvem de pontos da referência é importante porque estes não contém informação fidedigna das normais e para além disso a reconstrução de Poisson efetuada na nuvem de pontos da fonte contém erro em relação ao modelo original porque este algoritmo suaviza a superfície dando continuidade para além dos limites reais da peça o que provoca o erróneo cálculo das normais. Posto isto, a melhor solução passa sempre por eliminar estes pares de pontos que não fazem diferença alguma na análise porque neste caso a amostra é expressiva e retira pontos para processar o que é determinante para a eficiência do algoritmo[28].

No seguimento a esta etapa surge a medição do erro e a respetiva minimização, este processo é o ponto final do algoritmo ICP e a biblioteca PCL utilizada para a realização deste projeto usa o SVD (*Single Value Decomposition*) para a estimação da transformação final. A decomposição em valores singulares é uma fatorização de uma matriz real (neste caso mas pode ser complexa também) na forma $M = U\Sigma V^*$, em que M representa uma matriz quadrada $m \times m$, então U, Σ, V^* representam matrizes $m \times m$ de rotação, de escala e de translação, respetivamente[30].

Uma vez que o algoritmo é executado iterativamente até encontrar um erro mínimo desejável e para isso é repetidamente gerado o conjunto de pares de pontos usando a atual transformação e percorrendo todos os passos anteriores até encontrar uma nova transformação que minimize o erro.

Na implementação prática, a biblioteca PCL faz tudo pelo programador e é só necessário definir a nuvem de pontos de referência e a nuvem de pontos respetivo à fonte. Uma vez isso feito, foi necessário fazer o pré alinhamento porque as duas nuvens estavam dispersas no espaço tridimensional e essa primeira iteração permitiu que se juntassem na direção e sentido corretos como demonstra a figura 40. Depois, foi executado um novo ICP agora com as nuvens de pontos referenciadas espacialmente como estão na figura 40 e foi ainda definido um número de 200 iterações e ao fim dessas iterações é gerado a matriz de transformação e logo a duas *point clouds* ficam alinhadas.

A figura 41, abaixo, demonstram na prática o algoritmo utilizado em PCL para duas nuvens de pontos iguais mas que estão espacialmente afastadas entre si, quer por rotação quer por translação

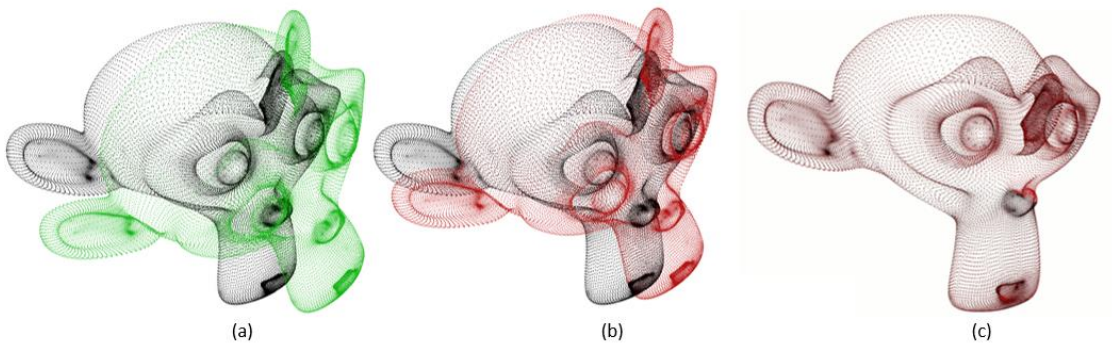


Figura 41 - (a): Nuvem de pontos original (referência) representado preto e nuvem de pontos representada a verde é a nuvem de pontos rotacionada e translacionada; (b): Nuvem de pontos representada a vermelha após uma iteração; (c): Nuvens de pontos alinhadas, Fonte: [31]

4.1.5 Cálculo da deformação existente na peça para as entidades

Esta secção consistirá na explicação da abordagem usada para o problema da deformação que é o tema com mais enfoque nesta dissertação. O próximo parágrafo explicará, em modo síntese, o algoritmo usado e posteriormente será abordado todos os métodos e algoritmos adicionais que complementaram este algoritmo principal.

Resumidamente, o algoritmo recebe como argumentos os pontos para projetar e a nuvem de pontos onde será feita essa mesma projeção e retornará uma nuvem de pontos referentes aos pontos projetados na superfície considerando a deformação existente entre o modelo real e o modelo CAD. Primeiramente, é criada uma *Kd-Tree* da nuvem de pontos do modelo real com o intuito de encontrar, eficientemente, o ponto mais próximo que os pontos das entidades estão da nuvem de pontos. Posteriormente, é efetuado um varrimento pelos polígonos, que devido à reconstrução de *Poisson* abordada anteriormente na secção 4.1.3 são exclusivamente triângulos, da malha do modelo real com a finalidade de encontrar todos os polígonos que partilham do ponto mais próximo descoberto anteriormente. Depois de efetuado esta etapa, é calculado os planos formados por cada polígono e feita uma projeção do ponto original em análise em cada plano dos diferentes polígonos, sendo esta projeção feita a partir da normal ao plano, por fim resta ter conhecimento se o ponto projetado está dentro do polígono ou não. Se não estiver, é feita uma segunda projeção para que o ponto projetado em primeira instância seja movido para o ponto mais próximo do polígono, podendo ser um dos vértices ou uma das arestas. Depois deste processo ser realizado para todos os polígonos, é calculado a distância entre o ponto original e os diferentes pontos obtidos a partir da projeção, a menor distância é elegida como o ponto mais fiável para corrigir a deformação. Ainda é importante relevar que este algoritmo é processado para todos os pontos recebidos como argumento pertencentes às entidades originais do modelo CAD.

Como explicado no parágrafo acima é possível entender que se usa diversos mecanismos e algoritmos complementares para a realização deste algoritmo principal. A primeira estrutura usada é uma *Kd-Tree* e esta escolha foi baseada na rapidez de processamento que advém com esta estrutura de dados para algoritmos de pesquisa, neste caso pesquisa do ponto ou pontos mais próximos.

Kd-tree é uma estrutura de dados muito usada para a partição de dados com especial enfoque para a organização de pontos porque dispõe de uma operacionalidade multidimensional podendo ser aplicada, construída e usada as suas aplicações em infinitas dimensões.

Aprofundando a questão para facilitar o entendimento destas estruturas, estas árvores funcionam como árvores binárias em que cada nó é um ponto com k dimensões. E cada um desses nós podem ser tomados como um hiperplano (conceito de plano em diferentes dimensões, por exemplo, no espaço unidimensional um hiperplano é representado por um ponto, no espaço bidimensional é representado por uma reta, no espaço tridimensional é representado por um plano e assim sucessivamente) que tem como objetivo dividir esse espaço em duas partes, daí funcionar como uma árvore binária. Como neste caso será usado uma *kd-tree* para o espaço tridimensional podendo alterar este nome para *3d-tree*, irá considerar-se a imagem abaixo que visa simplificar com base num exemplo o modo de construção desta árvore.

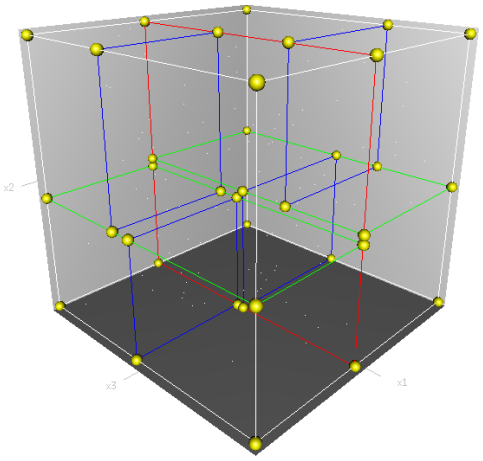


Figura 42 - Exemplo básico de uma *Kd-Tree*, Fonte: [32]

Como mostra a figura 42, para efeitos de demonstração, o cubo formado por arestas brancas representam a célula de raiz e esta é subdividida por um hiperplano do espaço tridimensional representado por um plano, definido por quatro pontos e para efeitos de visualização com os limites a vermelho. De seguida, cada subespaço, dividido anteriormente pelo plano vermelho, é subdividido de novo por os planos verdes criando assim mais subespaços e finalmente cada um desses subespaços são divididos novamente pelo corte dos planos representados a azul. Como a profundidade (quatro) desejada para este exemplo termina com as divisões de subespaços criados pelos planos azuis a estas oito células finais designa-se células folha[32].

A ideia base para a construção destas árvores baseia-se na escolha dos planos de divisão para a criação de subespaços e a biblioteca PCL usa o método canónico que apresentam as seguintes características:

- Os planos de divisão são sempre alinhados com o eixo para evitar que haja uma complexidade matemática muito grande que teria impacto a nível de tempos de processamento. Tendo em conta que o algoritmo já é complexo o suficiente para estruturas muito grandes.
- À medida que se percorre os diferentes níveis da árvore, estes vão iterando os planos de divisão, para o melhor entendimento siga-se este exemplo, numa árvore tridimensional como o caso abordado neste projeto, o plano de divisão do nó de raiz estará alinhado com o eixo X, sendo que os filhos deste nó teriam os planos alinhados com o eixo Y, os netos com o eixo Z, os bisnetos com o eixo X novamente, os trisnetos com o eixo Y, e assim sucessivamente.
- Como em quase na maioria das aplicações, o uso destas árvores é tanto ou mais eficiente quanto mais balanceado esta árvore esteja (a árvore é totalmente balanceada se a partir do seu nó de raiz a sub-árvore da esquerda tiver o mesmo número de níveis da sub-árvore da direita). Esta propriedade é essencial para a aplicação em causa porque a finalidade da construção da mesma é para a

realização de pesquisas de vizinhos mais próximos e se a árvore estiver balanceada terá de percorrer menos níveis para encontrar esse ponto, caso não esteja terá obrigatoriamente de percorrer mais níveis no seu pior caso. O pior exemplo de árvore desbalanceada é o nó de raiz ter uma sub-árvore gigante de um dos lados e do outro lado apenas ter um único nó, o seu outro filho, perdendo deste modo eficiência e também a vantagem da utilização deste algoritmo. Para a criação dos planos de corte será utilizado a mediana dos pontos colocados na árvore tendo em conta o eixo que está sendo utilizado.

Uma vez construída a *kd-tree*, terá de se proceder à pesquisa do vizinho mais próximo com o ponto da entidade original que está em análise. Esta pesquisa é feita de forma muito eficiente devido ao facto de considerar-se que a árvore encontra-se balanceada, o que faz com que a sua complexidade seja $O(\log n)$ porque o algoritmo vai descartando múltiplas sub-árvores que contém uma imensidão de pontos para o caso a estudar.

Observe-se as figuras seguintes (43 – 48) como exemplo de aplicação do algoritmo de pesquisa do vizinho mais mais próximo numa *Kd-Tree* (para facilitar a explicação é usado um exemplo no espaço bidimensional pelo que os hiperplanos serão representados por retas):

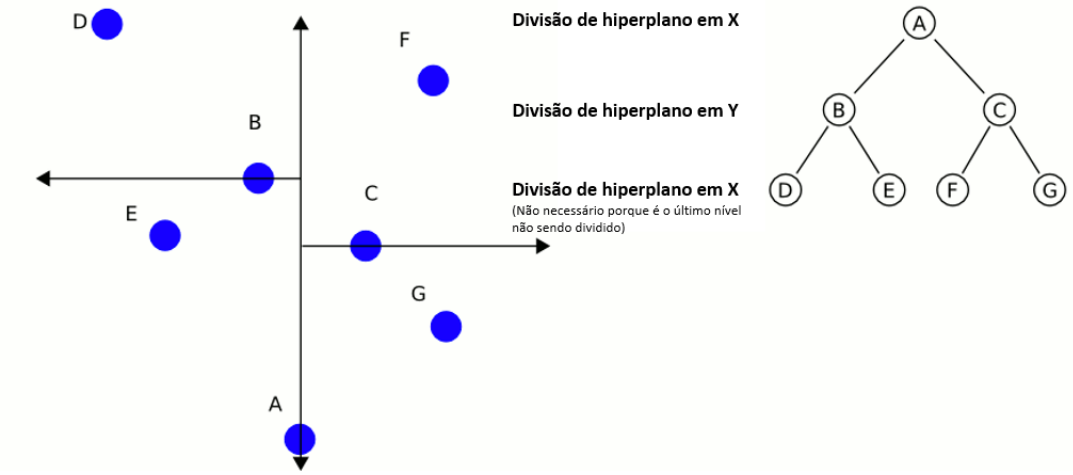


Figura 43 - 1ª etapa: Algoritmo de busca, Fonte: [32]

Considerando a *Kd-tree* do exemplo desta figura 43 com a representação gráfica da árvore binária em que os hiperplanos (neste exemplo são retas porque está-se no espaço bidimensional) começam por ser divididos segundo o eixo X e no nível seguinte (filhos) os hiperplanos são criados de modo a serem alinhados com o eixo Y e assim sucessivamente.

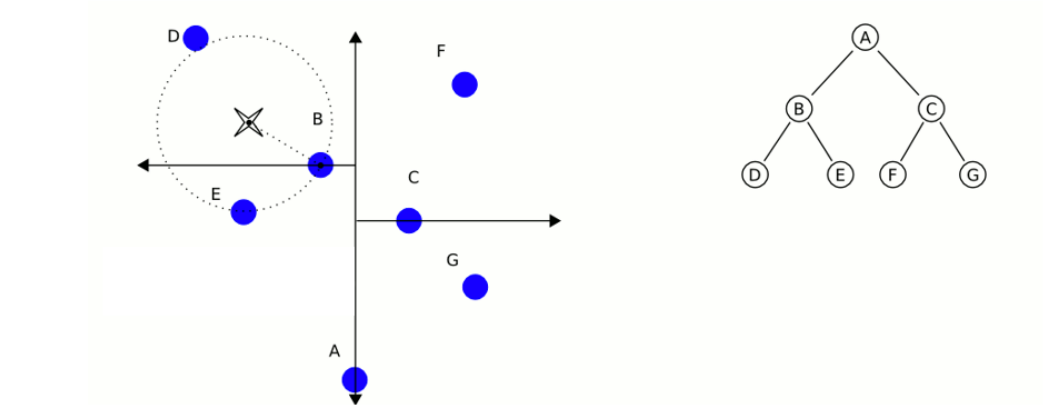


Figura 44 - Resultado prévio da solução do algoritmo, Fonte: [32]

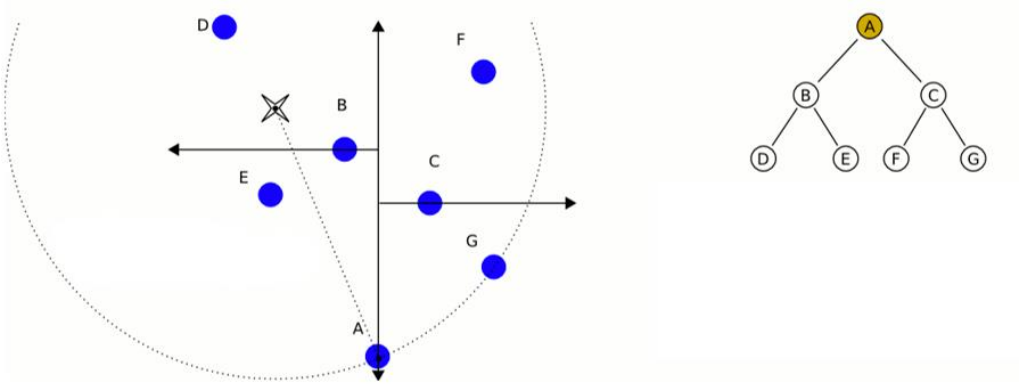


Figura 45 - 2ª etapa: Algoritmo de busca, Fonte: [32]

O algoritmo começa pelo nó de raiz e determina a distância ao ponto A, representado graficamente no espaço bidimensional onde se encontram os pontos a azul. Um dado importante a realçar neste ponto é o facto de o algoritmo de busca se basear em *"depth-first search"*, ou seja, o algoritmo faz a sua análise sempre em profundidade, o que significa que quando analisa uma sub-árvore, mantém-se nela até não ser possível obter um resultado melhor, retrocedendo posteriormente para outro ramo não visitado.

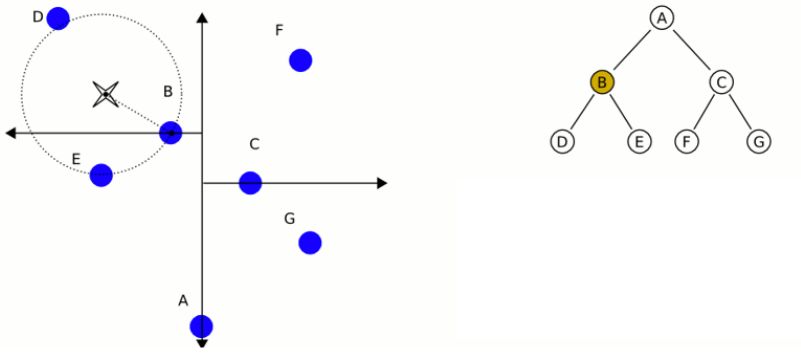


Figura 46 - 3ª etapa: Algoritmo de busca, Fonte: [32]

Uma vez completo o cálculo a distância entre o ponto em análise e o ponto A, avança-se para a sub-árvore esquerda o que significa que se determina a distância entre o ponto dado e o ponto B (que é o nó imediatamente a seguir). Nota-se que a distância é menor entre o

ponto em análise e o ponto B do que entre o ponto e o ponto A, desse modo atualiza-se a melhor estimação.

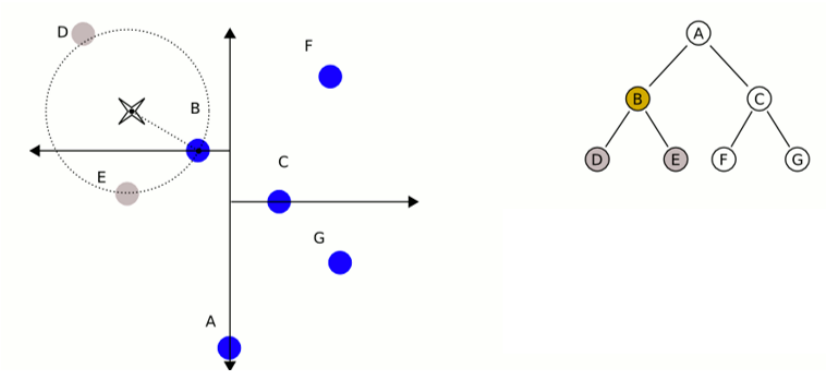


Figura 47 - 4ª etapa: Algoritmo de busca, Fonte: [32]

O próximo passo é continuar a percorrer a árvore em profundidade e determinando as distâncias, primeiro, entre o ponto dado e o ponto D e compara-se os valores da melhor estimação calculada até então com a distância determinada em último momento e percebe-se que é superior o que faz com que se rejeite este dado e se passe para o outro nó, o próximo nó é o ponto E e faz-se os mesmos passos e conclui-se que a distância continua a ser superior, logo a melhor estimação para a sub-árvore da esquerda é a distância entre o ponto dado e o ponto B.

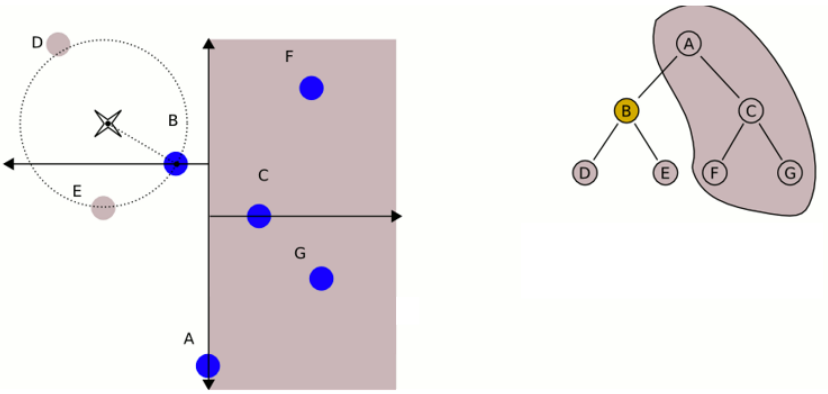


Figura 48 - 5ª etada - Algoritmo de busca, Fonte: [32]

Por fim, nota-se que a região não interseta a circunferência com centro no ponto dado e com o raio da melhor estimativa para o vizinho mais próximo, calculando a distância ao ponto C, o que consequentemente obriga a que seja rejeitada toda a sub-árvore da direita e deste modo todos os nós ficam “visitados” ou analisados direta ou indiretamente. Em suma, o ponto mais próximo do ponto dado é o ponto B.

Em PCL, este algoritmo já está inserido na sua biblioteca o que simplifica a sua utilização e tempo de programação e a função que implementa este algoritmo tem quatro argumentos de entrada, o ponto de referência, um inteiro referente ao número de vizinhos que se quer encontrar, um vetor de inteiros respeitante aos índices dos pontos vizinhos e um vetor de floats que correspondem às distâncias do ponto de referência aos pontos vizinhos. No caso deste projeto, apenas será necessário o vizinho mais próximo pelo que o argumento do número

de vizinhos a descobrir deve ser um e os vetores resultantes devem ser dimensionados à priori com dimensão 1 também porque só haverá um índice e uma distância, obviamente, poderá ter uma dimensão superior ao necessário contudo por rigor e brio só se deve despender os recursos necessários para o efeito de cada função.

Posteriormente à obtenção do vértice ou vizinho mais próximo a partir do algoritmo e estrutura de dados explicada acima é necessário adquirir os planos formados pelos polígonos ao redor desse vértice. Para isso, na algoritmia do programa foi usado um varrimento pelos polígonos da malha do modelo real e caso esse vértice pertença ao polígono em análise, existe uma estrutura de dados que grava o valor do índice do triângulo. Este varrimento termina quando for varrido todos os polígonos da malha ou então, mais usualmente quando for atingido seis polígonos circundantes ao vértice em análise, isto porque na reconstrução de superfície de *Poisson*, a geração da malha só é constituída por triângulos e existem no máximo seis polígonos a partilharem o mesmo vértice. Este algoritmo torna-se ineficiente mas é absolutamente necessário, contudo seria mais prudente e ousado utilizar um circulador que baseia-se em encontrar o primeiro triângulo e depois selecionar todos os polígonos circulando-os em torno do vértice em análise, isso evitava que fosse efetuado seis varrimentos à mesma malha.

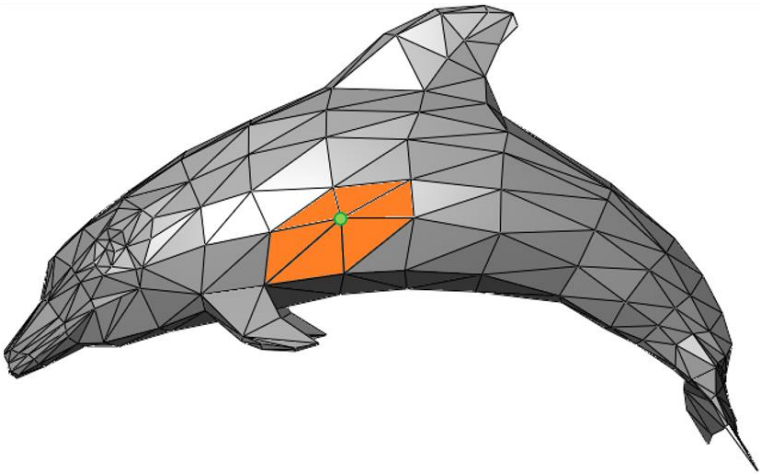


Figura 49 - Triângulos circundantes a um ponto, Fonte: [33]

Daqui em diante, enfrentou-se um problema de abstração de dados que obrigou de modo a facilitar a utilização de uma estrutura de dados que fosse capaz de ter toda a informação referente aos dados necessários para efetuar a devida projeção do ponto de referência para a superfície considerando a deformação existente. A figura 50 mostra um trecho de código usado para a definição da estrutura de dados:

```
struct VertexTrianglePlanePoint
{
    pcl::PointXYZ v0;
    pcl::PointXYZ v1;
    pcl::PointXYZ v2;
    std::vector<float> plane;
    pcl::PointXYZ original_point;
    pcl::PointXYZ projected_point_triangle;
}typedef VTPP;
```

Figura 50 - Estrutura de dados que relaciona polígono, plano e pontos

Sendo que v0, v1 e v2 representam os vértices dos triângulos descobertos a partir dos índices obtidos nos varrimentos da malha descritos no parágrafo anterior, o vetor de floats representam os coeficientes do plano formado por os três vértices, e por fim a estrutura contém o ponto que está em análise e um outro ponto sendo o resultado final do algoritmo para obtenção da projeção da deformação entre os dois modelos.

Para o preenchimento dos dados iniciais da estrutura de dados foi necessário um algoritmo para o cálculo dos planos, sendo os vértices obtidos de uma maneira direta a partir do índice do polígono. Pois uma vez tendo os vértices pode-se demonstrar abaixo o algoritmo usado para a obtenção da equação do plano. Considere-se o seguinte exemplo que demonstra o algoritmo para a obtenção do plano a partir de três pontos genéricos P, Q, R. A partir destes pontos são formados dois vetores, como PQ e PR, podendo ser usado qualquer par de vetores possíveis com pontos do mesmo plano. Agora é conhecido da geometria que o produto externo entre dois vetores será ortogonal a esses mesmos vetores. Uma vez calculado o produto externo significa que o vetor normal do plano está descoberto. Posto isto, pela equação do plano $ax + by + cz + d = 0$, e sendo a normal $\vec{n} = (a, b, c)$, o que significa que só resta descobrir a variável d e para obter esta variável basta calcular uma simples equação com os dados que se tem até então. Pela substituição de x, y e z por um ponto do plano na equação que define o mesmo, formando a seguinte expressão para a obtênção da única variável que resta descobrir: $d = -ax - by - cz$. Dada a obtenção de todos dados do plano, resta introduzir isso na estrutura de dados, que para o caso é um simples vetor com 4 elementos em que os seus elementos são ordenados segundo esta disposição $[a, b, c, d]$.

De seguida, é calculado o ponto mais próximo entre o ponto dado e o plano calculado a partir do polígono e para isso basta fazer uma projeção em que o ponto mais próximo ao plano é aquele que segue a direção do vetor normal. Sendo assim é utilizado um algoritmo de *ray-tracing* em que o *ray* é o vetor normal ao plano com a origem no ponto dado para a análise. Primeiramente, o plano é normalizado de modo a que o vetor normal tenha norma unitária e depois é calculado a distância entre o ponto e o plano, sendo considerado a equação do plano como $ax + by + cz + d = 0$ e o ponto em análise como $P(x_0, y_0, z_0)$. Uma vez que os dados estão completos, agora, para o cálculo da distância é aplicada esta expressão $dist = a \cdot x_0 + b \cdot y_0 + c \cdot z_0 + d$. Por fim o ponto contido no plano e mais próximo do ponto dado é, admitindo que a normal unitaria é $\vec{n} = (n_x, n_y, n_z)$, $Ponto_{plano}(x_0 - n_x \cdot dist, y_0 - n_y \cdot dist, z_0 - n_z \cdot dist)$.

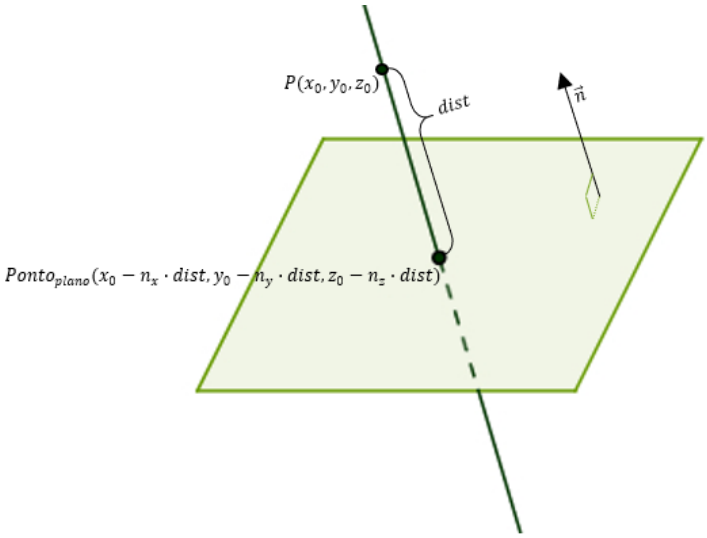


Figura 51 - Projeção de um ponto no plano com um *ray* de direção equivalente à normal ao plano, Fonte: [34]

Depois da obtenção do ponto no plano resta saber se essa projeção está dentro do polígono ou triângulo que formam esse plano. Para isso a figura 52 tem o objetivo de clarificar objetivamente o algoritmo usado para este efeito e será explicado a posteriori.



Figura 52 - (a): Ponto projetado no plano e contido no polígono; (b): Ponto projetado no plano não está contido no polígono o que implica uma projeção para o ponto mais próximo do polígono

A figura 53, abaixo, relaciona a posição de um ponto e de um triângulo, relativamente à sua geometria e lugar espacial.

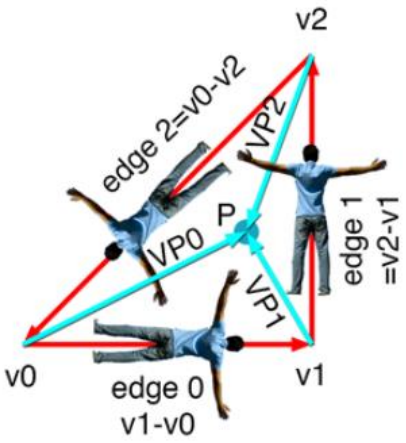


Figura 53 - Representação gráfica do método para descobrir se um ponto está dentro de um triângulo, Fonte: [35]

Para descobrir se o ponto P está dentro do triângulo, começa-se por representar os três vetores que delimitam o triângulo, representados em vermelho na figura 53 e depois é considerado os outros três vetores representados a azul na figura 53, sendo estes vetores formados com a origem nos diferentes vértices e o destino no ponto em análise pertencente ao plano. Pois de seguida é calculado o produto externo entre os pares de vetores vermelhos e azuis, sendo que este resultado forma outro vetor com uma determinada direção e um determinado sentido, ortogonal aos mesmos. Por fim, o ponto está dentro do triângulo se os produtos internos entre a normal ao plano e os resultados dos produtos externos dos diferentes pares de vetores sejam positivos. Basta que um dos produtos internos sejam negativos para que o ponto esteja fora do triângulo, por exemplo, supondo que o produto interno entre a normal ao plano e o resultado do produto externo (VP0 e edge0) seja negativo, que fará com que o ponto P esteja do lado direito do vetor edge0 ao invés do lado esquerdo, que representa o interior do triângulo. O trecho seguinte representa o pseudocódigo do algoritmo descrito acima:

```
Vec3f edge0 = v1 - v0;
Vec3f edge1 = v2 - v1;
Vec3f edge2 = v0 - v2;
Vec3f C0 = P - v0;
Vec3f C1 = P - v1;
Vec3f C2 = P - v2;
if (dotProduct(N, crossProduct(edge0, C0)) > 0 &&
    dotProduct(N, crossProduct(edge1, C1)) > 0 &&
    dotProduct(N, crossProduct(edge2, C2)) > 0) return true; // P is
inside the triangle
```
[35]

Caso o ponto esteja dentro do triângulo, pois este fica assume a qualidade de ponto projetado porque significa que está na superfície da malha.

Caso se verifique o contrário será necessário recorrer a um outro algoritmo que irá projetar o ponto contido no plano no ponto mais próximo do polígono pertencente a esse plano e esse ponto irá pertencer a uma das arestas ou a um dos vértices do polígono. O algoritmo usado é implementado três vezes para analisar a distância entre o ponto e as três arestas do polígono, a menor distância resultante fica como o ponto projetado na malha.

A figura 55 esquematiza as diferentes situações possíveis de localização do ponto em relação a uma das arestas do triângulo e posteriormente ao cálculo da sua distância.

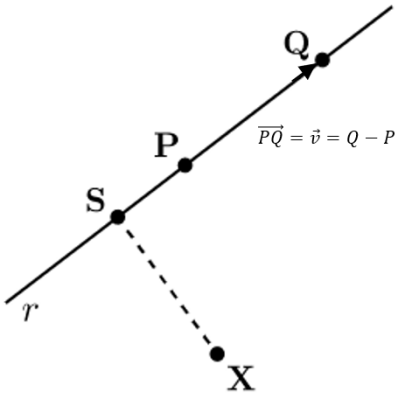


Figura 54 - S é a projeção de X que representa o ponto mais próximo à reta r, Fonte: [36]

Sendo a reta r definida por P e Q, esta pode ser parametrizada da seguinte forma:

$$r \coloneqq \{R(\lambda) = P + \lambda \cdot \vec{v} = P + \lambda \cdot (Q - P), \forall \lambda \in \mathbb{R}\} \quad \textbf{(7)}$$

Considerando a distância entre X e S, a mínima distância entre X a reta r, obtém-se:

$$d(X,S) = \min_{\lambda \in \mathbb{R}} d\big(X, R(\lambda)\big) \quad \textbf{(8)}$$

Sendo $d(\cdot,\cdot)$ a distância euclidiana então, para o cálculo do mínimo da distância descrita na equação acima considera-se que λ mínimo seja $\lambda_{mín}$, o que equivale ao mesmo que está presente na equação abaixo porque o mínimo é igual à derivada como demonstra a equação abaixo:

$$\left.\frac{d\big(X, R(\lambda)\big)}{d\lambda}\right|_{\lambda=\lambda_{mín}} = 0 \quad \textbf{(9)}$$

Considerando agora, a fórmula da distância Euclidiana obtém-se a seguinte expressão:

$$\left.\frac{d}{d\lambda}\sqrt{(R(\lambda) - X)^2}\right|_{\lambda=\lambda_{mín}} = \frac{2(R(\lambda_{mín}) - X) \cdot R'(\lambda_{mín})}{2\sqrt{(R(\lambda_{mín}) - X)^2}} = 0 \quad com \quad R'(\lambda_{mín}) = Q - P \quad \textbf{(10)}$$

O que simplificando um pouco mais a esquação, obtém-se:

$$\left.\frac{d}{d\lambda}\sqrt{(R(\lambda) - X)^2}\right|_{\lambda=\lambda_{mín}} = \frac{(R(\lambda_{mín}) - X) \cdot (Q - P)}{d\big(X, R(\lambda)\big)} = 0 \quad \textbf{(11)}$$

Por fim, considerando a equação (12) e a equação parametrizada da reta, obtém-se a seguinte equação que permitirá descobrir-se diretamente $\lambda_{mín}$ para a encontrar o ponto desejado na reta:

$$(P + \lambda_{mín}(Q - P) - X) \cdot (Q - P) = 0 \quad \textbf{(12)}$$

O que implica que:

$$\lambda_{mín} = \frac{(X - P) \cdot (Q - P)}{(Q - P) \cdot (Q - P)} \quad \textbf{(13)}$$

Sendo que as coordenadas do ponto S são obtidas a partir:

$$S = R(\lambda_{mín}) = P + \lambda_{mín} \cdot \vec{v} = P + \lambda_{mín}(Q - P) \quad \textbf{(14)}$$

Pois sabe-se agora que existem três casos possíveis para a resolução deste problema ao admitir-se um segmento ao invés de uma reta. De modo a facilitar o entendimento deste ponto, descreve-se S como uma soma de um ponto com um vetor como representado na equação (14).

Onde $\vec{v} = Q - P$, ou seja S é a soma de P com o vetor $\vec{v}$ multiplicado com o escalar $\lambda_{mín}$, sendo o escalar $\lambda_{mín}$ a única variável que depende de uma variável externa, o ponto X,

pois com $\lambda_{mín}$ entre 0 e 1 exclusive, o ponto S estará entre P e Q (ou seja, na aresta do triângulo), se $\lambda_{mín}$ for maior ou igual a 1 pois S será coincidente com Q e por fim de $\lambda_{mín}$ for menor ou igual a 0, S será coincidente com P, como demonstra a figura 55.

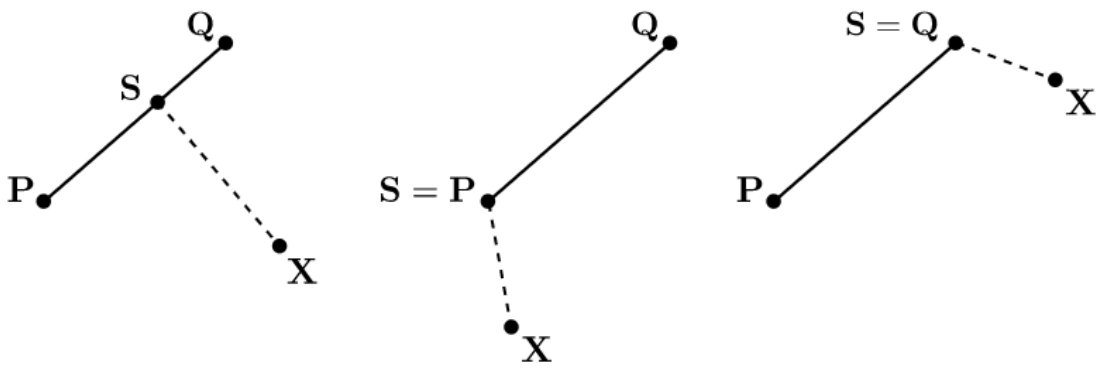


Figura 55 - Três casos possiveís para a determinação do ponto mais próximo, Fonte: [36]

A figura 55 representa os três casos possíveis da "projeção" de um ponto X num segmento considerando a distância mínima entre as duas entidades (ponto-aresta). Tendo em conta a equação (13), é relativamente fácil entender o intervalo de $\lambda_{mín}$ que determina qual o ponto mais próximo do segmento. Para as duas últimas figuras da figura 55 a descoberta do ponto é de fácil determinação porque são coincidentes com os vértices conhecidos. Para o primeiro caso, também é fácil no entanto é necessário fazer mais um cálculo depois de ter sido determinado $\lambda_{mín}$ e o cálculo é simplesmente usar a equação (14) de determinação do ponto S para o caso da reta (porque no fundo é o mesmo caso com a particularidade do ponto estar entre os dois pontos que definem essa mesma reta), .

Em modo de síntese o algoritmo usado para a determinação da mínima distância entre um ponto e um polígono que partilham o mesmo plano é:

1. Calcular $\lambda_{mín}$ usando a equação (13);
2. Se $\lambda_{mín} \leq 0, então\ S = P$;
3. Se $\lambda_{mín} \geq 1, então\ S = Q$;
4. $0 < \lambda_{mín} < 1, então\ S = P + +\lambda_{mín}(Q - P)$;

Depois deste algoritmo de uma maneira ou de outra já existe a capacidade de conhecer o ponto contido em cada triângulo, quer no seu interior ou na sua fronteira. Uma vez feito este processo para todos os triângulos ao redor do ponto em análise, é feito um cálculo da distância entre o ponto em análise e os diferentes pontos dos polígonos calculados previamente. A menor distância é a melhor estimação para o novo ponto pertencente à entidade corrigida ou atualizada considerando o problema principal da dissertação, a deformação existente entre os modelos reais e os modelos virtuais[36].

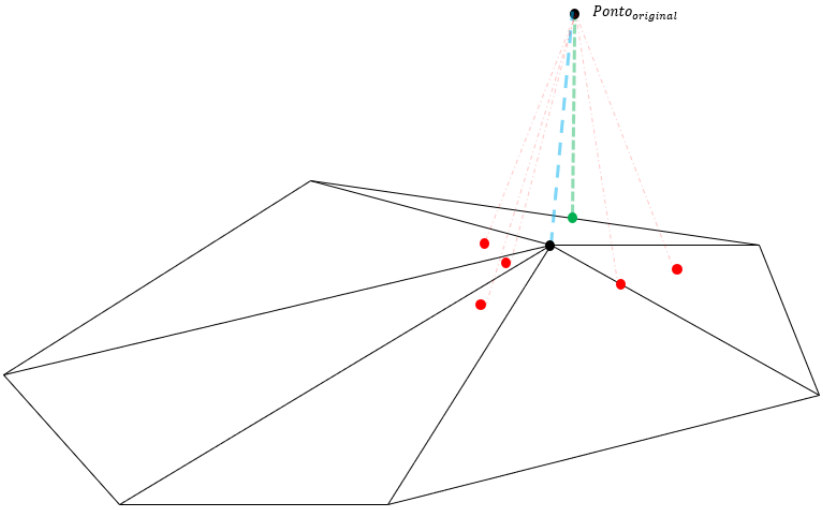


Figura 56 - Exemplo esquemático do resultado do algoritmo completo sendo que o ponto verde é o ponto eleito para corrigir a deformação porque tem a menor distância relativamente aos pontos vermelhos

Este processo que foi descrito neste capítulo é feito iterativamente para todos os pontos que se desejam projetar. E esta estrutura (vetor de pontos a projetar) como enunciado de forma introdutória é um dos argumentos de entrada desta função, do ponto de vista programático.

Para além dos resultados finais é importante a validação do sistema para comprovar que o algoritmo funciona de acordo as necessidades, ou seja de prever a deformação e projetá-la devidamente na superfície real. De modo a validar o sistema, utilizou-se exemplos muito simples e de fácil análise para provar o sistema. As próximas figuras (57-61) demonstram exatamente essas validações para posteriormente ser possível aplicar à prática com a certeza que o sistema é fiável e prático. Outro aspeto importante a ter em mente é o facto da distorção nos exemplos de validação serem grotescamente exageradas o que tal não acontecerá na prática porque é preciso ter consciência que a diferença ou deformação entre os dois modelos (virtual e real) tem que ser residual quando comparado com estes exemplos, uma vez que o modelo real é baseado nas dimensões modelo CAD. Posto isto, o algoritmo caso apresente provas de funcionamento para estes exemplos, certamente corresponderá às expectativas na aplicação real do sistema.

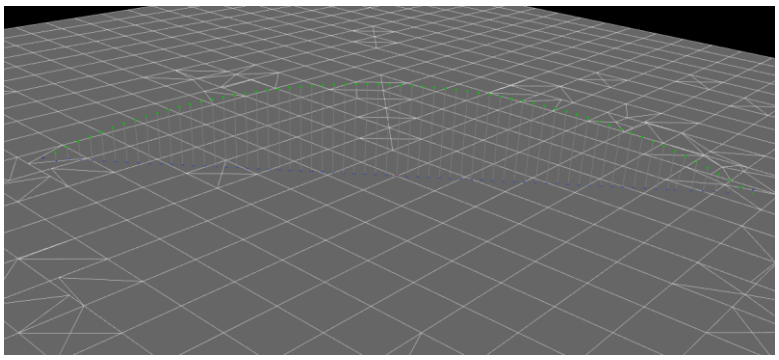


Figura 57 - Superfície plana com pontos a projetar representados a verde o pontos projetados a azul (correspondentes à deformação), Exemplo 1

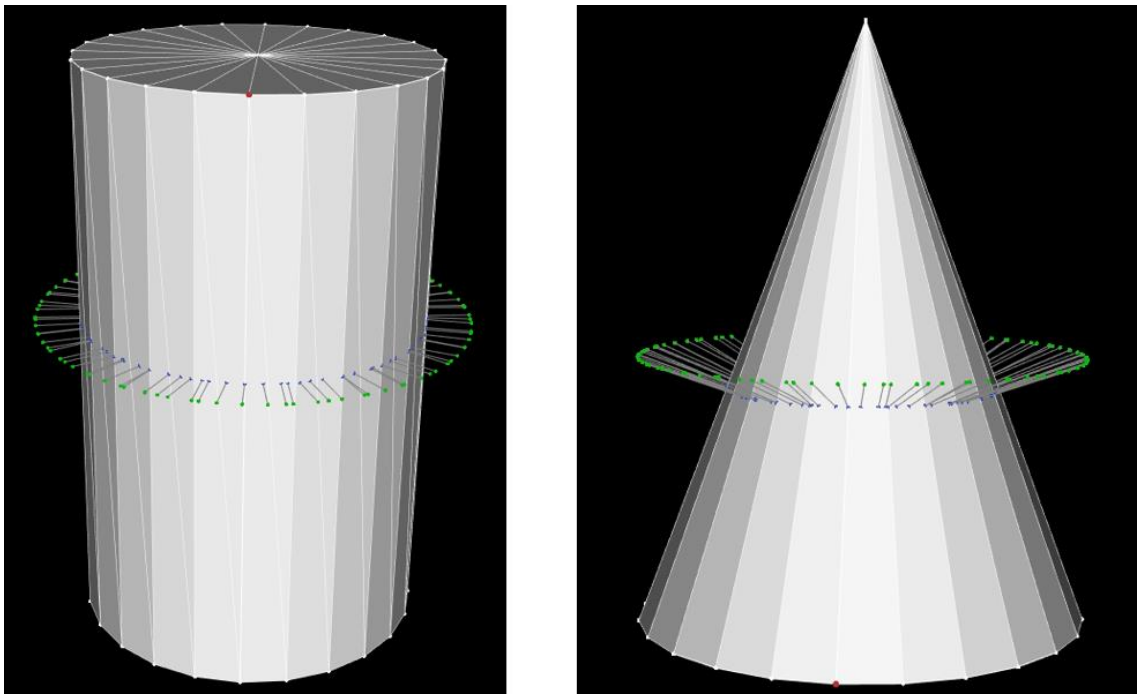


Figura 58 - Deformação de uma circunferência num cilindo e num cone, Exemplo 2

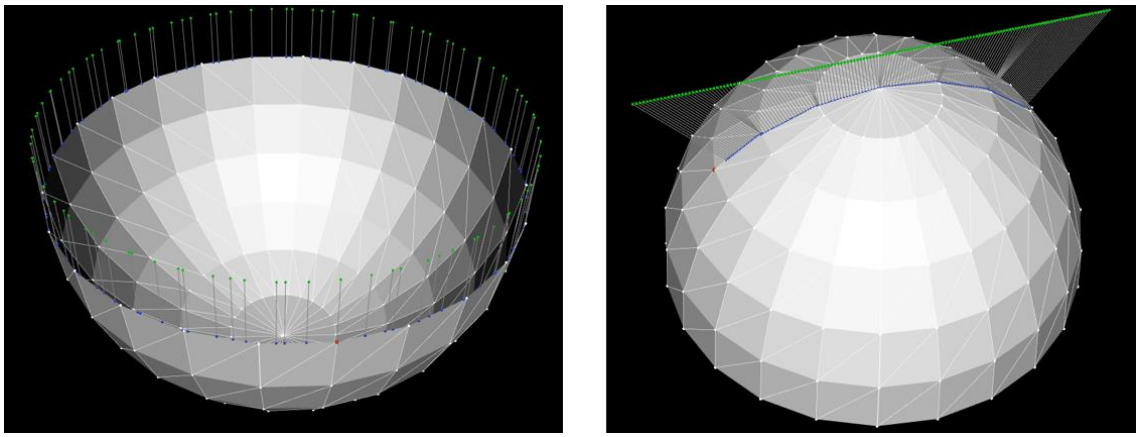


Figura 59 - Deformação de uma circunferência e de uma linha numa semi-esfera, Exemplo 3

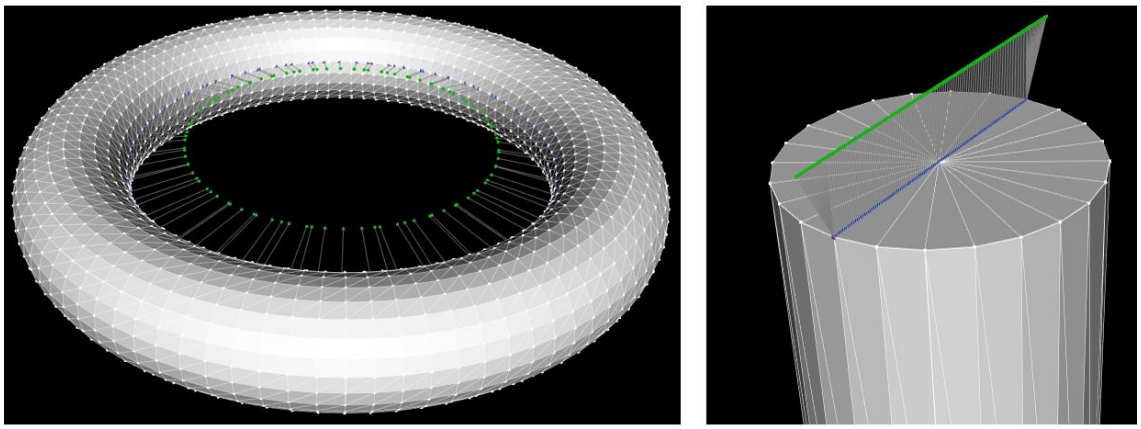


Figura 60 - Deformação de uma circunferência e de uma linha num aro e num cilindro, Exemplo 4

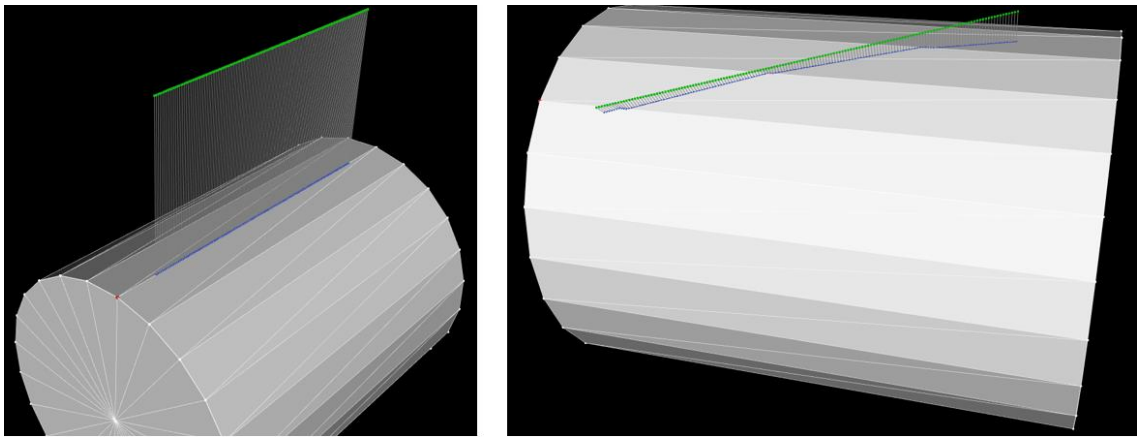


Figura 61 - Deformação de uma linha num cilindro

4.2 - Bridge C++/CLI

Uma vez que o algoritmo base está totalmente desenvolvido, testado e validado ainda sem algumas otimizações que poderão ser desenvolvidas. Partiu-se para o desenvolvimento de outro ponto *core* para o sucesso deste trabalho. Como foi descrito em capítulos superiores, a interação com o utilizador da solução desenvolvida e descrita nesta dissertação acontece com uma aplicação em C#, contudo este nível de programação é demasiado alto para a solução completa ser programada nesta linguagem, tendo sido programado em C++, o que dificulta a transferência de informação entre a interface e o algoritmo *core* (secção 4.1) do sistema. Para colmatar este problema foi considerado a utilização de uma ponte que articula estas duas linguagens que é programada numa linguagem criada para este problema designada C++/CLI.

C++/CLI é uma linguagem de programação independente mas que pode ser vista como uma extensão à linguagem C++, por outro lado pertence à família de linguagens .NET, assim como C# e Visual Basic. Foi projetado e desenvolvido pela Microsoft com o intuito de criar uma possibilidade de misturar linguagens .NET com código nativo em C++. Existem várias razões para o uso desta ferramenta, tais como, o desenvolvimento de *software* que necessita de bibliotecas nativas a C++ e a .NET, no desenvolvimento de uma biblioteca que interliga código nativo com .NET ou quando no desenvolvimento de um programa é necessário a eficiência de processamento de dados, feita preferencialmente em C++ nativo, além disso também é necessário ter uma complexa interface gráfica, que idealmente é utilizada aplicações programadas .NET[37].

```
class Entity
{
public:
    const char* m_Name;
private:
    std::vector<std::vector<float>> m_points_cpp2cs;
    std::vector<std::vector<float>> m_points_cs2cpp;
public:
    Entity(const char* name, std::vector<std::vector<float>> points_cs2cpp);

    void Move();
    inline std::vector<std::vector<float>> GetPoints() const { return m_points_cpp2cs; };
};
```

Figura 62 - Estrutura C++

O trecho de código imediatamente acima representa a estrutura de dados definida em código nativo, C++, em que *m_Name* é a *string* que contém a o *"file path"* do modelo CAD, *m_points_cs2cpp* representa o vetor de pontos das entidades no modelo CAD (o nome da variável é sugestivo porque considera em que sentido segue a informação, *cs2cpp*, que significa de C# para C++), *m_points_cpp2cs* representa o vetor de pontos das entidades projetadas na superfície, designa-se pelo resultado final e uma vez mais o nome da variável é sugestivo o que facilita o entendimento. A função *Entity* representa o construtor da classe, que necessita do ficheiro CAD identificado pelo seu *"file path"* e necessita do vetor de pontos das entidades a projetar. A função *move()*, é a função que executa todo o algoritmo em C++ abordados nas secções anteriores que engloba a segmentação, alinhamento, projeção de pontos, etc. Por fim a função *GetPoints()* é simplesmente uma função que retorna o vetor de pontos transformados, de modo a possibilitar o acesso a partir do programa em C#.

Depois de criada a estrutura em código nativo para possibilitar a utilização da bridge, foi necessário cumprir tudo para o seu funcionamento, o que passa por imediatamente a seguir ser criada a mesma classe mas adaptada a linguagem C++/CLI, para que seja possível uma conversão. O seguinte trecho de código representa essa estrutura.

```
Entity(String^ name, array<float, 2>^ points);
void Move();
property array<float, 2>^ Points
{
    public:
        array<float, 2>^ get()
        {
            return GetVecPoints(m_Instance->GetPoints());
        }
    private:
        void set(array<float, 2>^ value)
        {
        }
}
```

Figura 63 - Estrutura C++/CLI

É possível notar ligeiras alterações no ponto de vista de sintaxe, o que é normal visto que é uma linguagem *.NET*. Posto isto, o construtor passa a ter como argumentos *String* e *array* que são estruturas das linguagens *.NET*, pelo que as suas conversões serão relatadas e explicadas nos parágrafos seguintes. Contém igualmente a função *move()*, porque esta necessita de ser chamada no projeto de C#, visto que os outros projetos (C++/CLI e C++) vão

funcionar em *"background"*. Como variáveis, é necessário o uso de um *array* de *arrays* que contém a informação relativa aos pontos e pode-se notar que a função *get()* retorna o *array* que a função *GetVecPoints()* tem como resultado, esta função é um conversor de vetor nativo para *array .NET* em que o seu argumento de entrada é o vetor nativo retornado pela função *GetPoints()*, em C++, inserida na semelhante estrutura do código nativo.

```
namespace CLI
{
    Entity::Entity(String^ name, array<float, 2>^ points)
        : ManagedObject(new Core::Entity(string_to_char_array(name), float_array_conversion2(points)))
    {
        Console::WriteLine("Creating a new Entity-wrapper object!");
    }
    void Entity::Move()
    {
        Console::WriteLine("The Move method from the Wrapper was called!");
        m_Instance->Move();
    }
}
```

Figura 64 - Funções de implementação da *bridge*, construtor e função de execução, respetivamente

Este trecho de código é onde as funcionalidades da bridge são feitas na sua essência, uma vez que as estruturas estão projetadas, agora resta implementá-las para servir o propósito da sua aplicação principal. O código representado acima demonstra as duas funções base para o seu uso na programação da interface em C#. Primeiramente, foi desenvolvido o construtor da classe criada previamente, tendo como argumentos a String e o array de pontos, uma vez que é necessário transmitir esses dados para a estrutura de dados em linguagem nativa, é criado uma conversão para que esta informação seja transmitida corretamente de modo a existir a construção da estrutura em C++, ou seja, quando se cria uma estrutura deste tipo em C++/CLI, acontecerá que irá ser criado a mesma estrutura em C++. A linha de código que representa o construtor significa que quando esta função é chamada, será criado um objeto que criará uma estrutura de dados em que os seus argumentos são o *char array* que representa a *String* em *.NET* e um vetor de pontos que representa o *array* desses mesmos pontos em *.NET*, estas funções designam-se *string_to_char_array()* e *float_array_conversion2()*, respetivamente. A função *Move()* só contém a informação para chamar a instância que lida com o objeto de C++ e chama a função *Move()* de modo a ser executada na parte nativa do projeto.

```
static const char* string_to_char_array(String^ string)
{
    const char* str = (const char*)(Marshal::StringToHGlobalAnsi(string)).ToPointer();
    return str;
}
```

Figura 65 - Coversão de *String* para *char

O código acima representa a função de conversão de uma *String* do tipo *.NET* para um apontador de caracteres, que será chamado em código nativo. Esta conversão é muito utilizada pelo que é fornecido os seus exemplos pela Microsoft (empresa desenvolvedora).

```
static std::vector<std::vector<float>> float_array_conversion2(array<float, 2>^ lb)
{
    std::vector<std::vector<float>> board;
    for (int i = 0; i < lb->Length / 3; i++) {
        std::vector<float> temp;
        for (int j = 0; j < 3; j++) {
            float num = lb[i, j];
            temp.push_back(num);
        }
        board.push_back(temp);
    }
    return board;
}
```

Figura 66 - Conversão *array* de pontos (C#) para vetor (C++)

O código acima representa a conversão da estrutura de dados que transporta as entidades a projetar, uma vez que é um caso muito específico, não existem bibliotecas que façam esta transformação automaticamente como aconteceu no caso da *String*, portanto a função foi criada com base na lógica do autor da tese. A função recebe a estrutura *.NET* e devolve um vetor da linguagem nativa e a sua implementação é baseada no varrimento da estrutura *.NET* e copiada a informação devidamente para estar contida dentro do vetor que a função retornará com a mesma ordem de pontos e todas as características a ela associadas.

```
array<float, 2>^ GetVecPoints(std::vector<std::vector<float>> temp)
{
    std::vector<std::vector<float>> tempVec = temp;
    //array<array<float>> temp1 = gcnew array<array<float>>;
    const int SIZE = tempVec.size();
    //array<array<float>^>^ tempArr = gcnew array<array<float>^>(SIZE);
    array<float, 2>^ tempArr = gcnew array<float, 2>(SIZE, 3);

    for (int i = 0; i < SIZE; i++)
    {
        for (size_t j = 0; j < 3; j++)
        {
            tempArr[i, j] = tempVec[i][j];
        }
    }
    return tempArr;
}
```

Figura 67 - Conversão de vetor de pontos (C++) para *array* de pontos (C#)

Resta explicar a função acima descrita, que tem como objetivo a conversão oposta, ou seja, de nativo para *.NET*, porque é necessário retornar a resultante das projeções dos pontos para a interface de modo a possibilitar a finalização do processo em que o operador terá as entidades projetadas e corrigidas no modelo real. O processo desta função é rigorosamente igual aos anteriores, porém a diferença está em o seu argumento de entrada ser um vetor de pontos nativo, este é varrido através de um ciclo *for* e posta essa informação dentro de um *array* compatível para linguagens .NET, sendo esta estrutura o resultado da função.

4.3 - Tratamento de informação para interface com o utilizador e com o sistema

A última parte da explicação da implementação do processo baseia-se na aplicação em C# e alguns tratamentos de dados e mecanismos que foram necessários para o bom funcionamento do programa.

A interface gráfica não foi desenvolvida porque esses requisitos não constavam no caderno de encargos, uma vez que a entidade que necessita deste produto tem uma aplicação já criada que pode ser facilmente adaptável. Em suma, o objetivo da aplicação em C# criada nesta dissertação é tornar esta adaptação possível e para isso é usado o *software EyeShot*, desenvolvido pela *devDept Software*.

O *Eyeshot* é um *software* que lida com CAD, ou seja desenho assistido por computador, sendo este software baseado em *.NET framework*. O que permite aos desenvolvedores criarem interfaces gráficas e/ou em simultâneo processar e modificar de alguma forma os desenhos CAD. Este *software* é muito completo e contém uma variedade enorme de todos os tipos de entidades como, curvas, superfícies, nuvens de pontos, malhas, etc. Para além disso contém poderosas ferramentas fundamentais para a aplicação em questão e outras que nem tanto, o que faz com que seja uma poderosa e versátil ferramenta para a programação CAD. Estas ferramentas incluem triangulação de malhas, operações booleanas, ajustes de curvas, reconstrução de superfícies, interceções e ainda é possível exportar e importar ficheiros que são descritos geometricamente, o que pode ser útil para a análise do objeto.[38]

Primeiramente, é criada uma *String* que identificará o ficheiro CAD a ser analisado para a extração das entidades que se deseja projetar, na interface gráfica esta *String* será encontrada pelo utilizador a partir de uma janela de *"Browse..."* pela máquina. Uma vez que essas entidades estejam contidas num *array* de *floats* estas são analisadas a posteriori, na interface gráfica estas entidades vão ser selecionadas graficamente pelo utilizador e automaticamente serão armazenadas numa estrutura adequada as coordenadas dos seus pontos.

Esta análise consiste em preparar as entidades, pensando logo no resultado final, com esse objetivo percebe-se que as entidades são constituídas por um conjunto de pontos, sejam estas entidades, circunferências, linhas retas, curvas, quadrados, etc. Este conjunto de pontos servirá para constituir linhas, de modo a dar instruções ao laser para projetar as entidades no modelo real, isto porque a programação para o projetor laser não está preparada para lidar com circunferências e outras geometrias que não sejam linhas, ou melhor, que não sejam dois pontos, por consequência a isto para representar uma circunferência por exemplo é necessário ligar todos os pontos constituintes desta geometria por interpolação de modo a formar retas entre os pontos.

Com base na informação adquirida no parágrafo anterior, pode-se concluir que uma entidade qualquer, considere-se um círculo de pontos para efeitos de exemplo e para facilitar

a explicação do algoritmo usado. Esta explicação será detalhada por figuras e textos ordenadamente.

1. Considere-se o círculo da figura 68 que simula a entidade extraída do ficheiro CAD

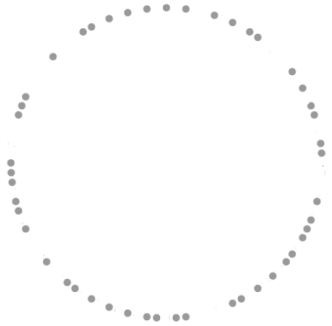


Figura 68 - Exemplo de uma entidade original

Estes pontos são a resultante de uma extração de uma entidade e admitindo que este conjunto de pontos não é alterado, é feito o cálculo da deformação que mantém exatamente a mesma forma da entidade original (que é o objetivo desse cálculo, não existir diferenças de formas de entidades mesmo que a superfície seja ligeiramente diferente), desse modo, será usado a mesma figura para o passo seguinte. Depois de considerada a deformação é necessário para o resultado final, uma projeção laser no modelo real sendo assim criada uma lista de elementos de uma classe, designada por *ScreenLine,* disponível em *EyeShot*.

A *ScreenLine* é uma linha formada por dois vértices cujo seu principal objetivo é a sua projeção gráfica e esta será usada na projeção laser. A lista de *ScreenLines* irá ser a estrutura responsável por conter todas as linhas que formam a entidade que posteriormente será usada na projeção real como foi mencionado acima.

Considere-se a figura 69 para efeitos de análise:

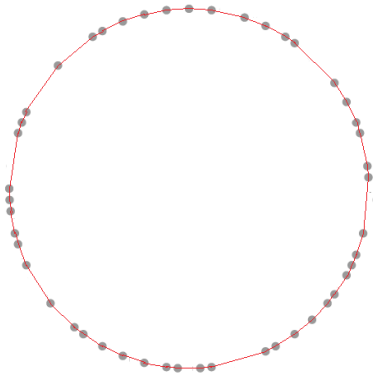


Figura 69 - Exemplo gráfico da projeção laser a partir dos pontos originais das entidades

As linhas representadas a vermelho e unidas a partir de dois pontos representam as *ScreenLines* que seriam o resultado da projeção laser. Como é de fácil análise a projeção destas linhas dariam um resultado defeituoso porque o circulo passaria a não ser um círculo por causa das discrepâncias distanciais entre pontos o que

poderia comprometer a fiabilidade do sistema. Para corrigir este problema foi criado um mecanismo que será implementado no ponto seguinte.

2. Uma vez extraído a entidade a projetar, neste caso um círculo, e analisados os problemas a resolver é implementada uma solução que visa povoar com pontos os espaços que apresentam uma falha de pontos e onde estará possíveis erros. O *software* usado é uma ferramenta muito útil porque extrai informação sobre a entidade que tem a forma de um círculo. Uma vez identificada a forma é usada uma função designada por *GetPointsByLenght(lenght)* que subdivide a curva com a inserção de pontos por a distância fornecida como argumento de entrada e devolve um *array* de pontos com os pontos espaçados uniformemente entre si. A distância fornecida como argumento deve ser determinada dinamicamente porque se existem uma imensidão de pontos o tempo de processamento vai ser muito grande no cálculo da deformação assim como a projeção laser vai ter muitas linhas para emitir, por outro lado se a distância fornecida for muito grande fará com que os tempos de processamento e projeção sejam mais rápidos embora o resultado final projetado no modelo real seja defeituoso e não apresente a forma desejada. Posto isto deve ser encontrado um valor ótimo, valor este que é extraído a partir do comprimento total da entidade (perímetro), uma vez determinado este perímetro é só extrair um valor com base numa densidade desejada e pôr o produto entre o perímetro e esta densidade no argumento da função. Este valor será o argumento de entrada da função. A figura 70 representa esquematicamente o resultado proveniente deste passo:

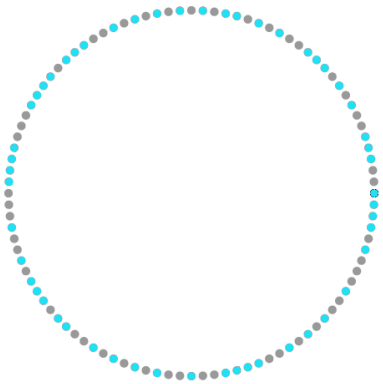


Figura 70 - Exemplo de resultado da função de inserção de pontos

Como o programa tem conhecimento prévio da forma da entidade a distribuição uniforme implementada pela função rege-se pelas equações das entidades, neste caso um circunferência, mas pode ser qualquer entidade, como, elipses, linhas, quadrados, etc.

3. Neste passo todos os pontos (criados e já existentes) são inseridos num *array* de pontos em que cada posição desse *array* contém um *array* de três *floats* que representam as coordenadas x, y e z, respetivamente. Posteriormente é criada uma

classe que será a classe responsável pela transferência de dados para código nativo como demonstra a figura 71:



Figura 71 – Execução do sistema e resultado gráfico

4. Depois de executada a função é gravado num *array*, os pontos com a deformação e logo é criada uma lista de *Screenlines* para passar a informação para a estrutura final como demonstra a figura 72:

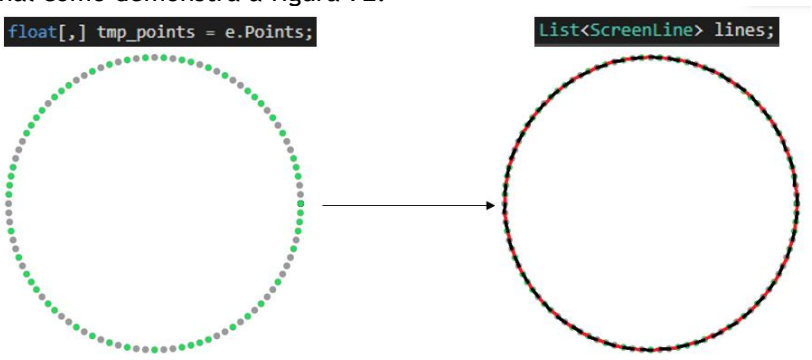


Figura 72 - Transgressão de pontos para linhas

Agora a lista de *ScreenLines* são constituídas por as linhas pretas e vermelhas que constituem um círculo perfeito e faz com que o resultado seja ótimo para a projeção laser. Uma vez que esta lista esteja criada o programa está completo para ser adaptado às necessidades seguintes contudo estas adaptações que não constarão nesta dissertação porque não fazem parte do plano de trabalho.

Em suma, este capítulo sintetiza de forma eficaz as diferentes entidades para que seja adquirido bons resultados de projeção aquando da transferência dos dados para o projetor laser. A forma eficaz encontrada para solucionar o problema parte do princípio de conhecimento prévio a partir das ferramentas utilizadas da forma da entidade e depois povoar a mesma com pontos segundo uma densidade dinâmica e ajustável.

Capítulo 5

Testes e Resultados

Será enunciada toda a fase de testes e resultados ordenadamente por critério processual, ou seja, os testes e resultados serão enunciados pela a ordem que ocorrem na realidade com a execução do algoritmo.

Primeiramente foi capturada com o auxílio de uma câmara 3D várias capturas do objeto em análise representado ilustrativamente na figura 73.



Figura 73 - Objeto representativo de um tampo, modelo real em análise

Para facilitar o desenvolvimento da componente de *software* procedeu-se à captura de algumas nuvens de pontos do objeto, tal como descrito acima. No entanto, a câmara utilizada não foi a mesma que será usada na solução do projeto e foi necessário capturar quatro nuvens de pontos, tal como representado na figura 74.

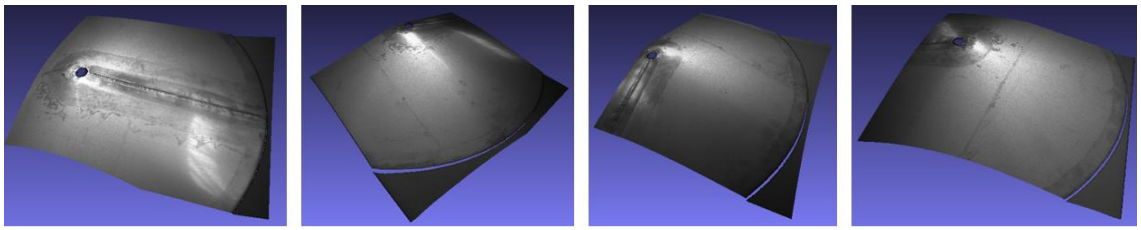


Figura 74 - *Point Clouds* do modelo real fragmentado em quatro partes distintas

Como na prática não será implementada desta maneira e a nuvem de pontos do objeto será capturada numa só imagem, foi necessário recorrer a *softwares* externos como o caso do *MeshLab* ou *CloudCompare* para obter a nuvem de pontos registada de modo a formar o tampo do silo de uma forma completa como demonstra a figura 75.

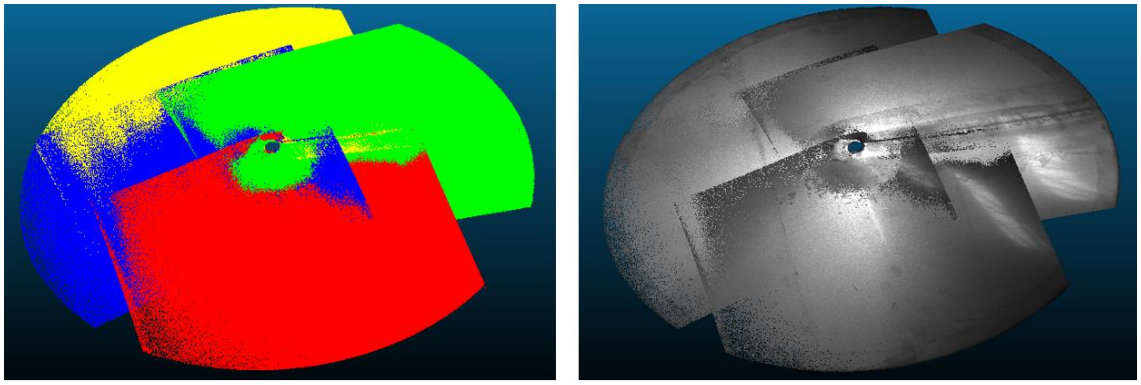


Figura 75 - Nuvens de pontos registadas em uma única, sendo a ilustração da esquerda uma representação das diferentes nuvens de pontos usadas para o registo e a ilustração da direita uma representação cromática real do objeto

A partir da *point cloud* representada na figura 75, foi iniciada a execução do algoritmo, sendo a primeira fase a segmentação da imagem.

Esta parte passa por fazer um *downsampling* à nuvem de pontos em análise.

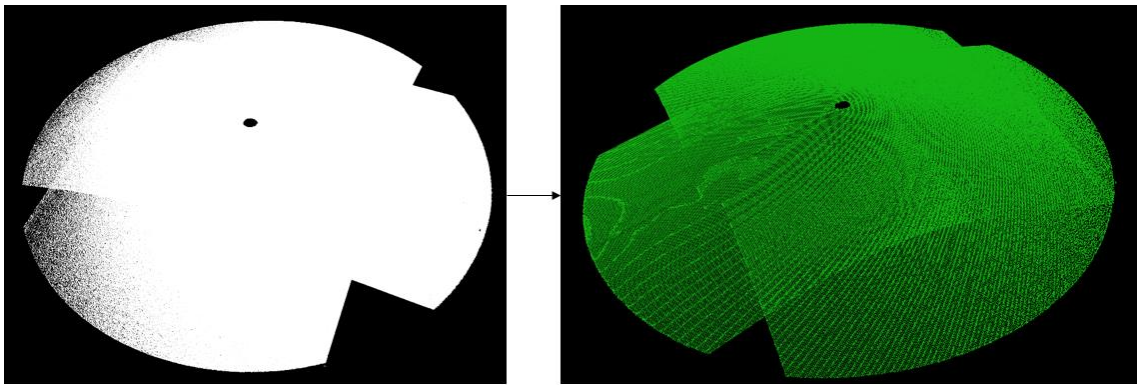


Figura 76 - *Downsampling* da nuvem de pontos. Origem representado a branco e o resultado representado a verde

BENCHMARK DOWNSAMPLING	
NÚMERO DE PONTOS ORIGINAIS	6 654 200
NÚMERO DE PONTOS DA NÚVEM DE PONTOS FILTRADA	181 338
***LEAF SIZE* (*SIZE* X, *SIZE* Y, *SIZE* Z)**	(1.51, 1.51, 1.51)
TEMPO DE EXECUÇÃO	0.690127 segundos

Tabela 3 - *Benchmark* do algoritmo de *downsampling*

Depois de segmentada a imagem 3D foi aplicada a reconstrução da superfície com o método de *Poisson*.

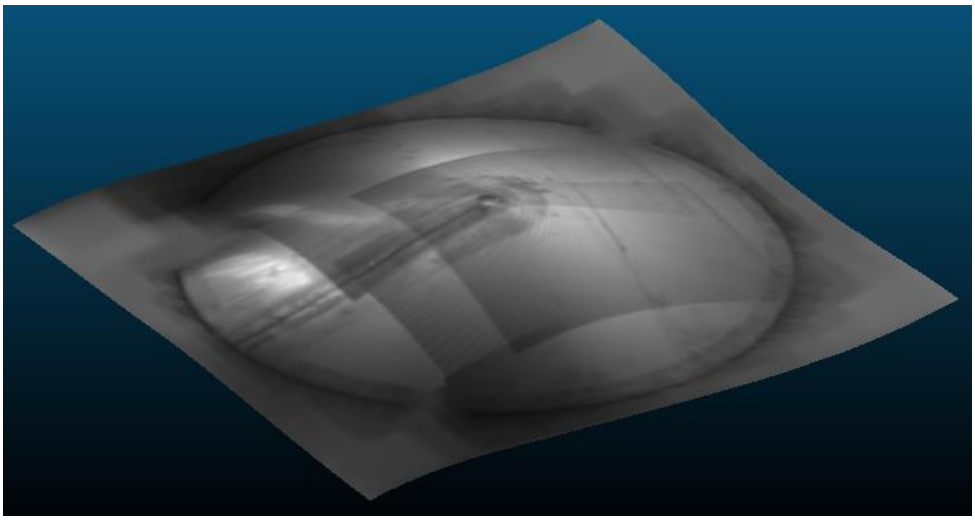


Figura 77 - Superfície reconstruída por o método de *Poisson*

BENCHMARK* DO ALGORITMO DE RECONSTRUÇÃO DE *POISSON	
NÚMERO DE *THREADS*	8
RAIO DE PESQUISA DAS NORMAIS	2.01
PORFUNDIDADE DA *OCTREE* (*DEPTH*)	8
AMOSTRAS POR NÓ	1
VÉRTICES	62 997
FACES	125 804
TEMPO DE EXECUÇÃO	6.34412 segundos

Tabela 4 - *Benchmark* do algoritmo de reconstrução de *Poisson*

O número de *threads* a ser definido foram com o valor de oito porque na estimação das normais irá ser usado um sistema com oito *cores*, o que fará com que a velocidade deste processo se torne seis a oito vezes mais rápido que se fosse usado só uma *thread*.

Uma vez tratado o modelo real, é necessário desenvolver os testes e os resultados para o tratamento do modelo CAD e a sua consequente incrementação de amostras como explicado na secção 4.1.3.

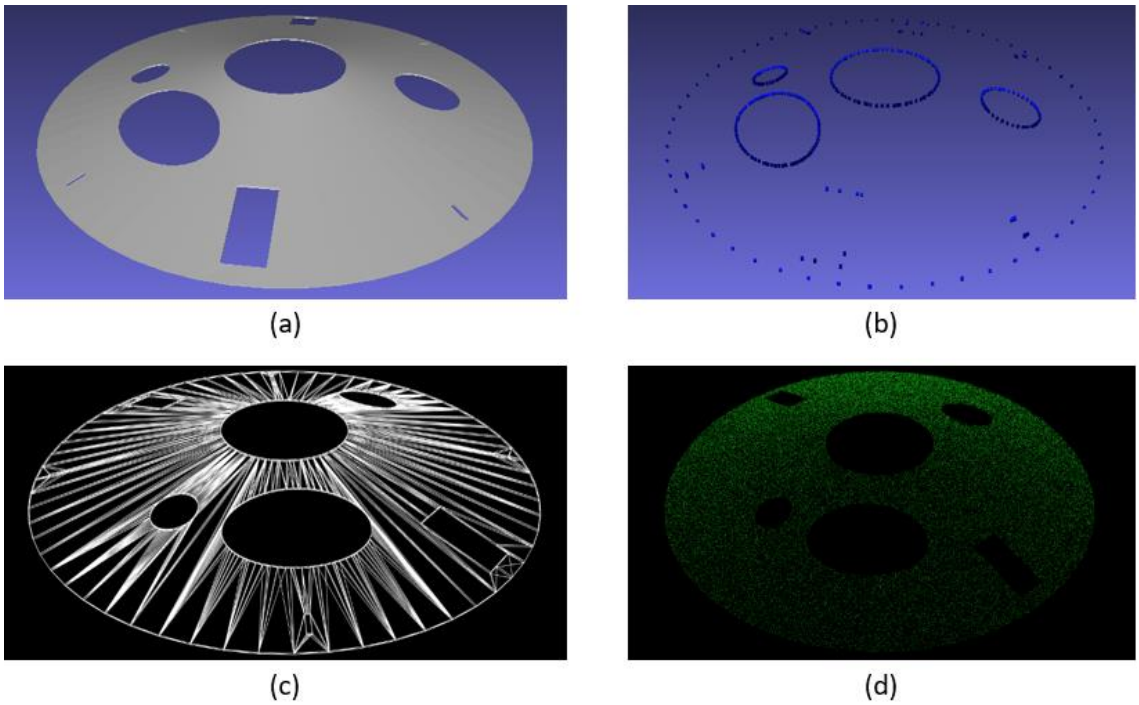


Figura 78 - (a): Modelo CAD visualizado como um sólido; (b): Vértices que definem o ficheiro STL representado em (a); (c): Ficheiro STL representado pelos seus triângulos; (d): Resultado do algoritmo de upsampling

BENCHMARK* DO ALGORITMO DE *UPSAMPLING

NÚMERO DE AMOSTRAS	100 000
FILTRO DE *LEAF SIZE*	0.3
NÚMERO DE PONTOS ANTES DA EXECUÇÃO	802
NÚMERO DE PONTOS DEPOIS DA EXECUÇÃO	99089
TEMPO DE EXECUÇÃO	0.0837211 segundos

Tabela 5 - *Benchmark* do algoritmo de *upsampling* do modelo CAD

O resultado mais curioso e que deve ser discutido é o facto de o número de amostras ser definido em cem mil pontos e o resultado final são de noventa e nove mil e oitenta e nove pontos. Esta discrepância deve-se ao facto de ser usado um leve filtro de *downsampling* para obter resultados mais precisos e consistentes com o modelo CAD.

Uma vez que os dois modelos estão tratados e analisados é necessário executar o algoritmo de alinhamento e obter as respetivas transformações para posterior análise de *benchmark*.

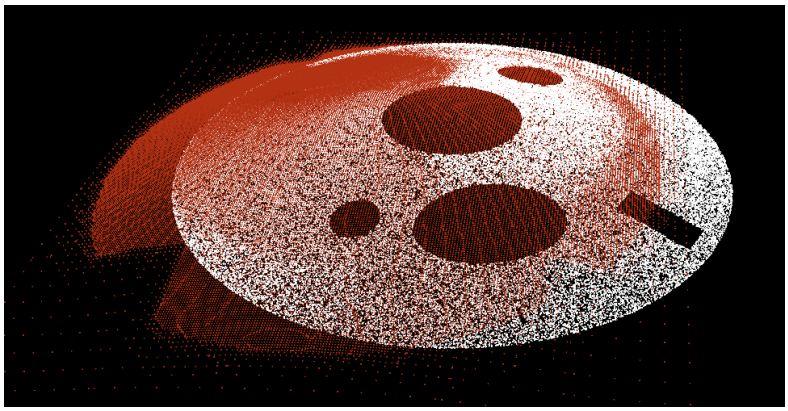


Figura 79 - Primeira iteração do algoritmo *ICP*

A figura 79 representa a primeira iteração do algoritmo de *ICP* para o eventual pré-alinhamento que existe caso as nuvens de pontos estivessem afastadas espacialmente.

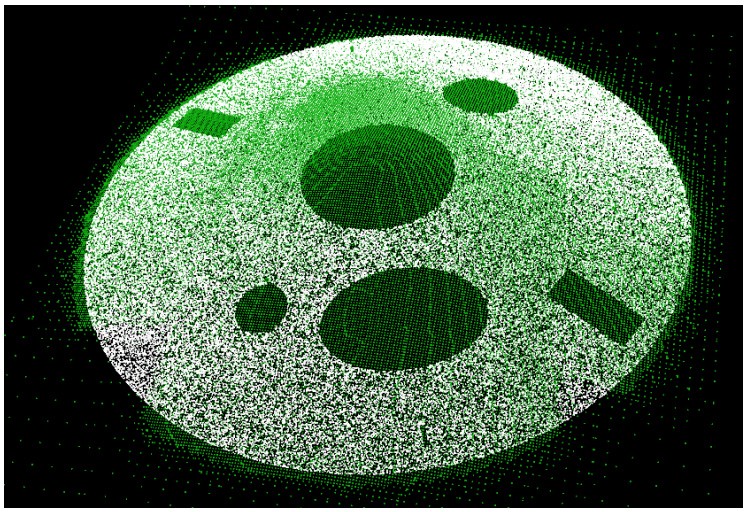


Figura 80 - Alinhamento completo

A figura 80 representa o resultado visual das nuvens de pontos alinhadas uma em relação à outra que se regem pelas seguintes transformações, uma relativa ao pré-alinhamento e outra relativa ao alinhamento executado como o resultado da figura 80, respetivamente pelas equações (15) e (16).

$$H_{pré-alinhamento} = \begin{bmatrix} 0.904977 & -0.379792 & -0.191770 & 315.227 \\ -0.253630 & -0.119686 & -0.959869 & 558.844 \\ 0.341598 & 0.917297 & 0.204640 & 449.914 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(15)}$$

$$H_{alinhamento\ final} = \begin{bmatrix} 0.999399 & 0.021316 & 0.028340 & 20.2123 \\ -0.023493 & 0.996647 & 0.078858 & -5.94571 \\ -0.026493 & -0.079473 & 0.996501 & -76.11300 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(16)}$$

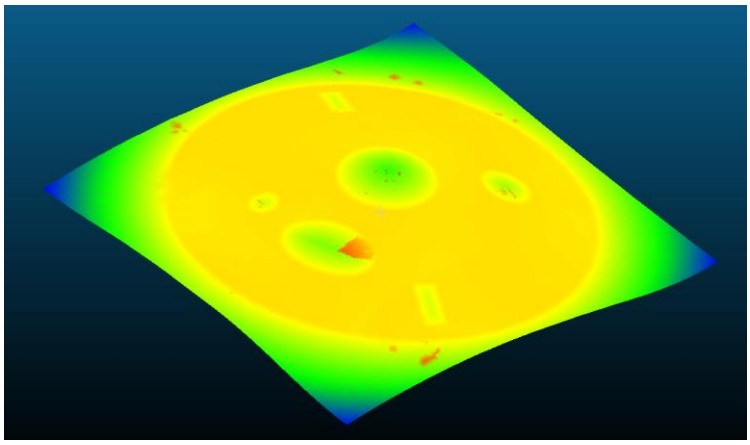


Figura 81 – Alinhamento completo com visualização distancial cromática

A figura 81 representa o mesmo que se pode ver na figura 80 mas com uma noção de distância espacial baseado nas cores sendo que o azul é quando a comparação entre as duas nuvens de pontos estão espacialmente distantes uma da outra e o amarelo alaranjado que se vê na figura é quando praticamente estão coincidentes. Deste modo, é possível ter a certeza que o alinhamento foi bem sucedido.

BENCHMARK* DO ALGORITMO DE *UPSAMPLING

PRÉ-ALIINHAMENTO CONVERGIU	Sim
ALINHAMENTO FINAL CONVERGIU	Sim
TEMPO DE EXECUÇÃO	24.9933 segundos

Tabela 6 - *Benchmark* do algoritmo de alinhamento *ICP*

Aquando da execução do alinhamento, segue-se a execução das projeções das entidades na superfície do modelo real.

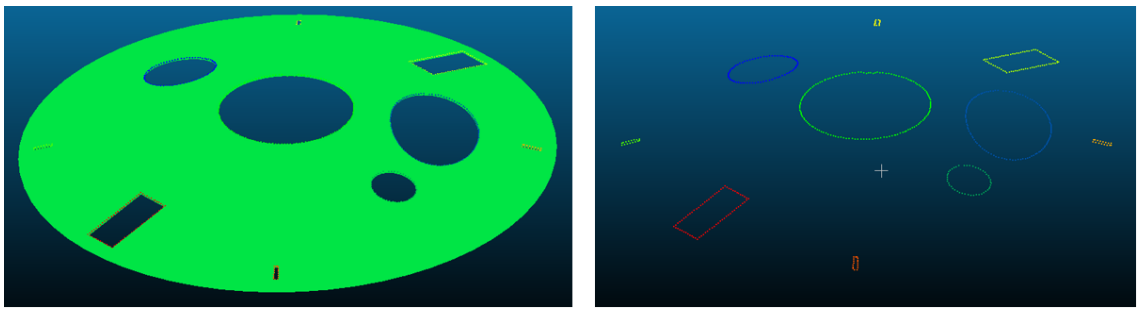


Figura 82 - Modelo CAD à esquerda e à direita uma nuvem de pontos que representa todas as entidades a representar com cores distintas

Realizada o tratamento das entidades, estas passam a ser um dos argumentos da função que executa a projeção dos pontos das entidades na superfície real do tampo. Como é demonstrado nas figuras 83 - 89.

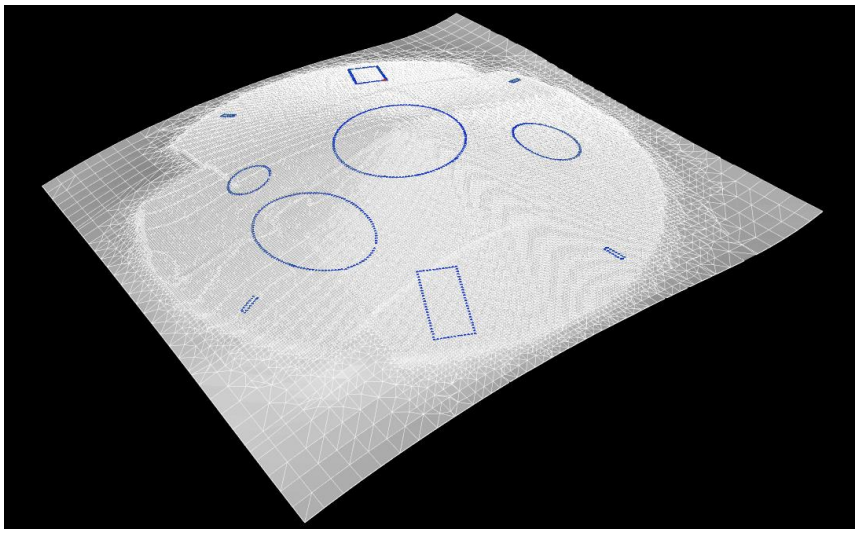


Figura 83 - Projeção final das entidades na superfície do modelo real com as entidades projetadas representadas a azul

Nas figuras abaixo (84-89) vão ser apresentados imagens mais nítidas do efeito da projeção para que se possa perceber que os pontos são projetados com base na deformação, representando as diferentes entidades.

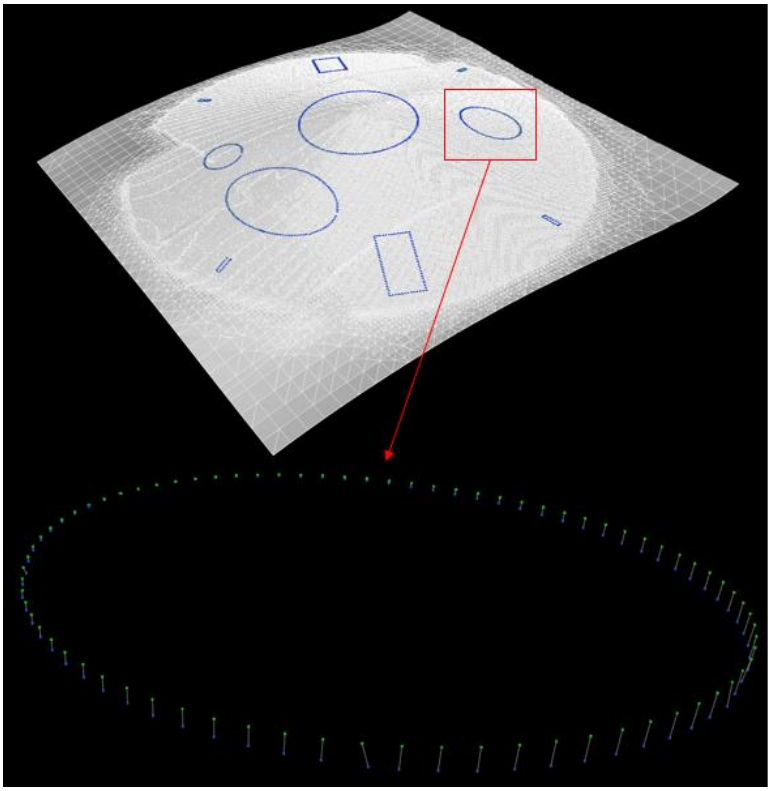


Figura 84 - *Zoom* da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 1

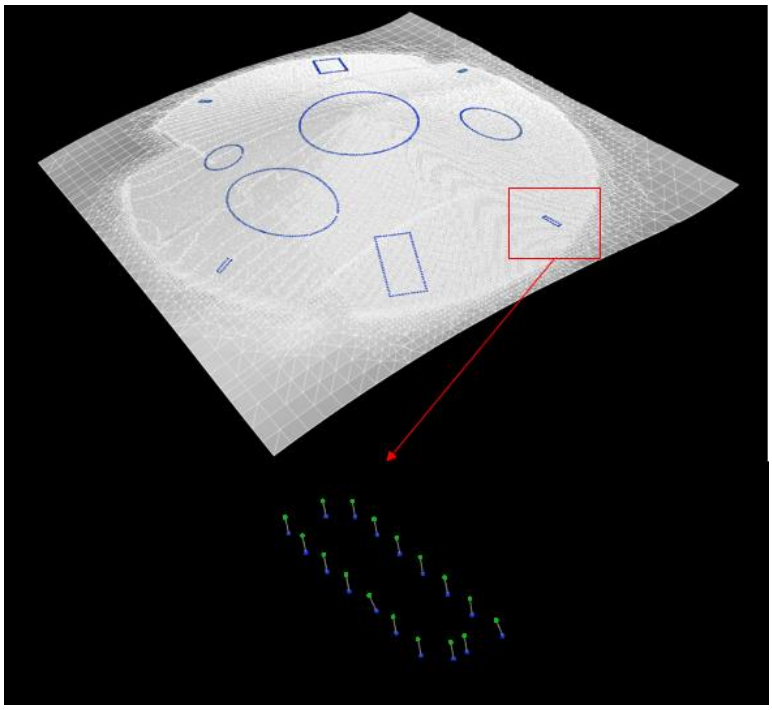


Figura 85 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 2

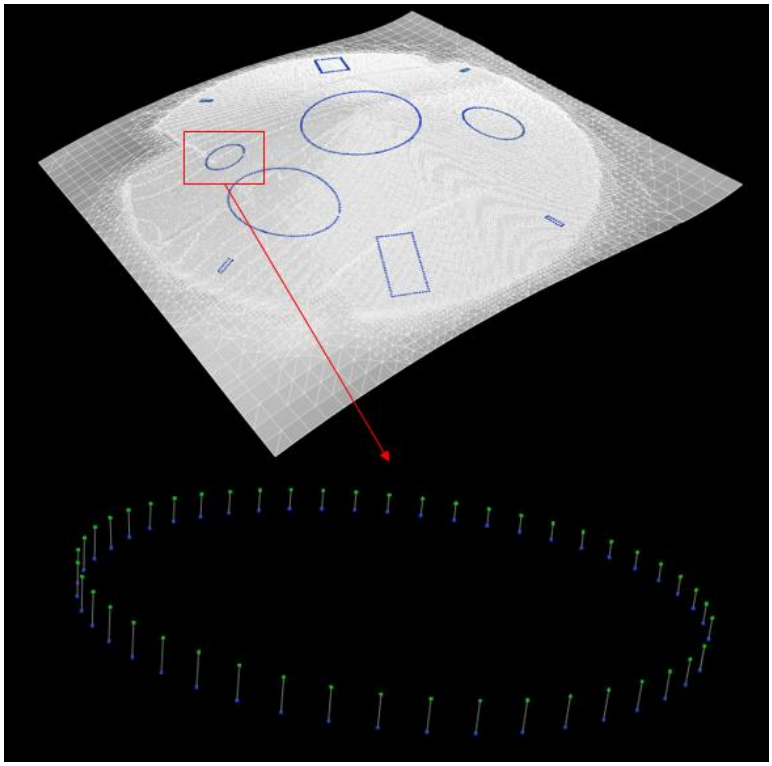


Figura 86 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 3

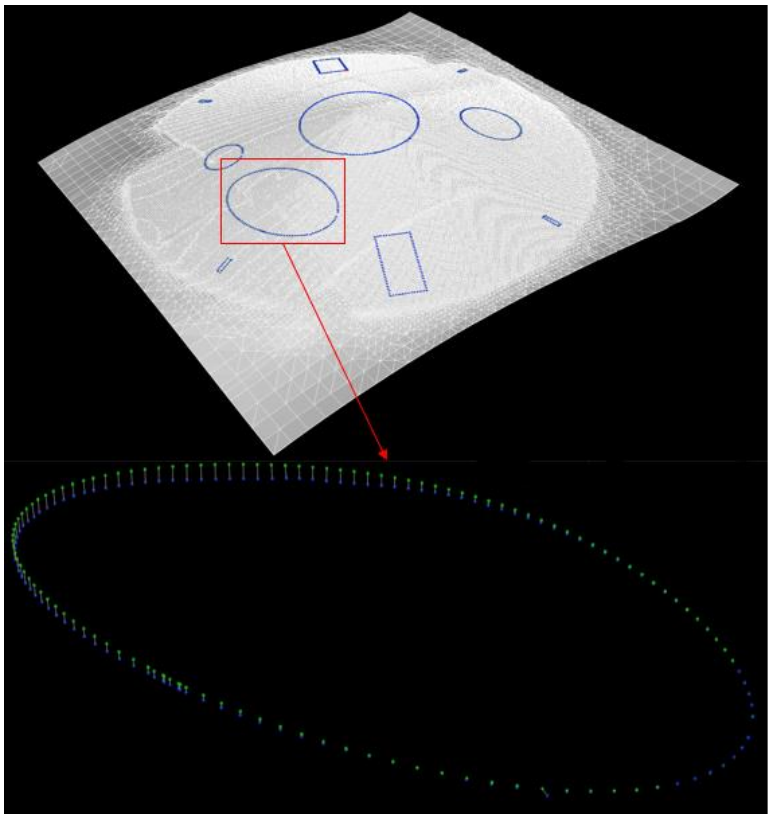


Figura 87 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 4

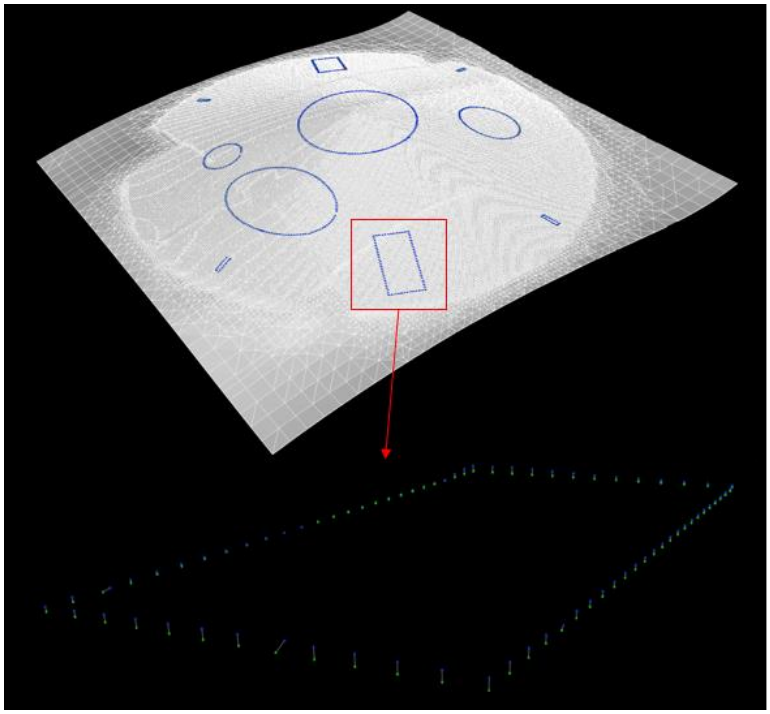


Figura 88 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 5

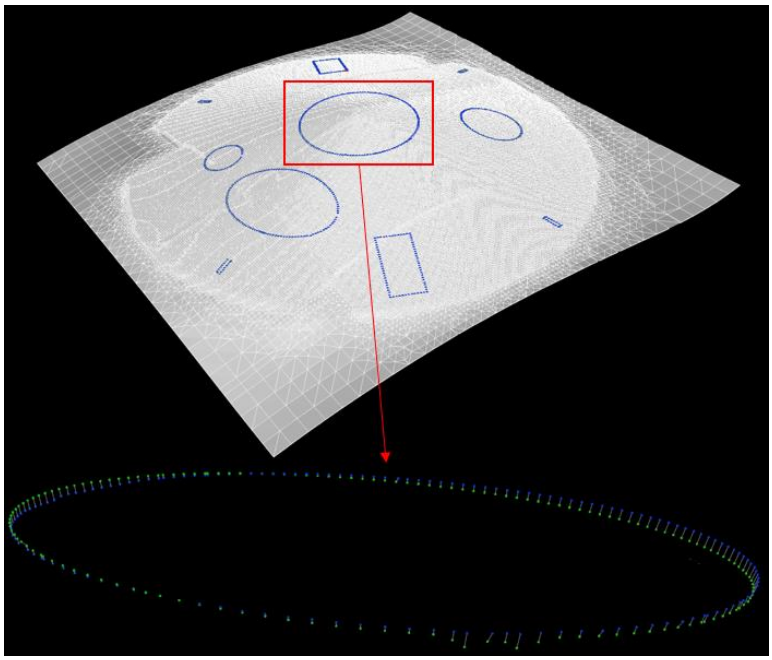


Figura 89 - Zoom da deformação na entidade selecionada com os pontos a verde corresponderem às entidades retiradas do modelo CAD e os pontos azuis corresponderem aos pontos projetados na superfície do modelo real, entidade 6

***BENCHMARK* DO ALGORITMO DE PROJEÇÃO DE PONTOS**

NÚMERO DE PONTOS A PROJETAR	609
NÚMERO DE FACES PARA O VARRIMENTO	125 804
NÚMERO DE VERTICES DA MALHA REAL	62 997
TEMPO DE EXECUÇÃO	13.1609 segundos

Tabela 7 - *Benchmark* do algoritmo de projeção de pontos

Uma vez todo o algoritmo em código nativo executado e testado foi testada a leitura e escrita final dos pontos que correspondem às entidades originais e projetadas, para efeito de exemplo a figura 90 representa a linha de comandos de dois pontos sendo que a primeira e terceira linhas correspondem ao sentido da informação na bridge e as segunda e quarta linhas correspondem a dois pontos das entidades originais provenientes do ficheiro CAD e aos pontos projetados e transformados na superfície do modelo real, respetivamente.

```
(C#  -> CLI ->  C++)  points
{ { -318.637   -223.769   628.873 } { -352.596   -206.842   636.875 }
(C++  -> CLI ->  C#) points
{ { -233.958  -35.64014  297.0887  }{ -268.3445  -46.39282  283.5354  }
```

Figura 90 - Consola de comandos do sistema final

Uma vez todos os resultados analisados, é possível concluir que o comportamento do sistema é satisfatório e cumpre requisitos temporais e de precisão com alguma eficiência e eficácia embora seja visível que existe trabalho futuro a desenvolver de modo a tornar o sistema ainda mais rápido.

No entanto, quando comparado com o paradigma atual do desempenho da mesma função é possível entender que é despendido em dois operadores oito horas de trabalho e se analisarmos as *benchmarks* do algoritmo é fácil de perceber que são projetadas todas as entidades com um rigor muito superior em menos de cinquenta segundos, o que representa um aumento de eficiência superior a 115000%, porém a eficácia é complicada de ser mensurável nesta fase do projeto porque não foi avaliada a qualidade das marcações no paradigma atual do desempenho desta função.

Em suma, é provado que a automação de processos industriais é uma mais-valia porque reduz o tempo de operação e consequentemente, a redução de custos mesmo que seja necessário um investimento inicial elevado porque este vai ser abatido ao longo do tempo, ainda para mais se se tratar de um processo *core* do negócio em questão.

Capítulo 6

Conclusões e Trabalho Futuro

Este capítulo finda a explicação da solução de um problema prático com aplicação na indústria alimentar para a marcação de cortes nos tampos dos silos de armazenamento de qualquer tipo de alimentos.

Todos os capítulos acima revelam e explicam detalhadamente de forma ordeira o processo de trabalho para a implementação e projeto do sistema completo.

É ainda de relevar, em última instância, que a conceção da tese foi criada para provar que pode ser desenvolvida um sistema fiável para solucionar o paradigma da marcação de superfícies sem que exista erros de medição e humanos e ainda ser possível adaptar esta projeção aos diferentes modelos que podem estar deformados, visto que o modelo virtual nunca é igual ao modelo real, existe sempre erro, nem que seja residual.

5.1 - Satisfação de objetivos

Aquando da apresentação desta tese, os objetivos traçados eram:

1. A implementação de *projection mapping* em objetos
2. A implementação de perceção 3D
3. A conceção de um sistema que articule os dois primeiros pontos, de modo a criar uma solução para o problema.
4. Testar e validar o sistema para que possa ser, eventualmente, uma solução concreta para a(s) empresa(s) que necessitam deste tipo de sistema.

Começando pela análise do ponto 1., a implementação de *projection mapping* em objetos é concluída com sucesso com a criação de algoritmos competentes que articulados entre si formaram de forma competente uma estrutura que permite projetar os pontos na superfície real do objeto com base em transformações, alinhamentos e tratamento de pontos

de um modelo para o outro. Em suma, este primeiro ponto pode ser considerado bem sucedido podendo ainda ser considerado um algoritmo versátil e com diversas aplicações, não só nesta indústria.

A implementação de perceção 3D, foi realizada usando a ferramenta *PCL* o que ajudou de forma notória à rapidez de processos e rapidez de implementação porque esta como foi descrita acima dispõe de algoritmos e ferramentas que simplificou o trabalho e permitiu analisar todas as nuvens de pontos dos objetos em análise em praticamente tempo real. Senão fosse por esta ferramenta o trabalho realizado teria de ser encurtado substancialmente por limitações temporais, desse modo, podemos concluir que esta perceção foi conseguida com sucesso a partir do estudo intensivo desta biblioteca.

Dadas todas as ferramentas que estavam ao dispor, foi criada uma solução fiável e robusta que executa de forma exemplar as tarefas necessárias para a implementação da solução, como demonstrado em testes e resultados e até mesmo nas validações dos próprios algoritmos.

Em suma, o trabalho árduo e o estudo e compreensão do problema, a seleção e uso das ferramentas necessárias e os testes e resultados obtidos para análise o que inúmeras vezes, a posteriori era necessário alterar e melhorar as abordagens ao problema, encarando o problema como uma malha fechada (sendo sempre possível melhorar, contudo a saída é satisfatória quando passa numa fase de testes, isto é, supera o erro esperado). A soma de todos estes fatores determinou essencialmente o sucesso de todos os objetivos impostos previamente nesta dissertação.

5.2 - Trabalho Futuro

Embora todos os objetivos desta dissertação fossem atingidos com sucesso, isto não implica que não haja trabalho a desenvolver futuramente, o que não é verdade.

Existe muito trabalho a desenvolver, como por exemplo, os dois pontos seguintes:

- Adaptar o sistema para objetos de grande dimensão
- Otimizar o algoritmo de modo a tornar-se ainda mais eficiente

Como citado ao longo deste documento esta dissertação serviu como prova e validação de uma solução para um sistema industrial, contudo sempre foi usado material expressamente necessário para a validação e consequente prototipagem do sistema, e este foi projetado nesse âmbito. É importante enfatizar que todos os algoritmos dispõem de capacidade de processamento e de abstração suficiente para lidar com objetos de qualquer dimensão ainda que possam ter que ser ajustados algumas variáveis e parâmetros funcionais. O problema de adaptação de *software* não é o mais difícil, aliás, é simples e rápido a transgressão do *software* para as diferentes aplicações.

Uma vez abordado o a parte do *software* é mais importante relevar a solução encontrada para as limitações de *hardware*. A solução desenhada para este tipo de sistema não

torna o sistema mais complicado embora terá que ser feito uns pequenos ajustes. Considere-se a imagem abaixo que caracteriza a seguinte situação no *setup* atual de prototipagem.

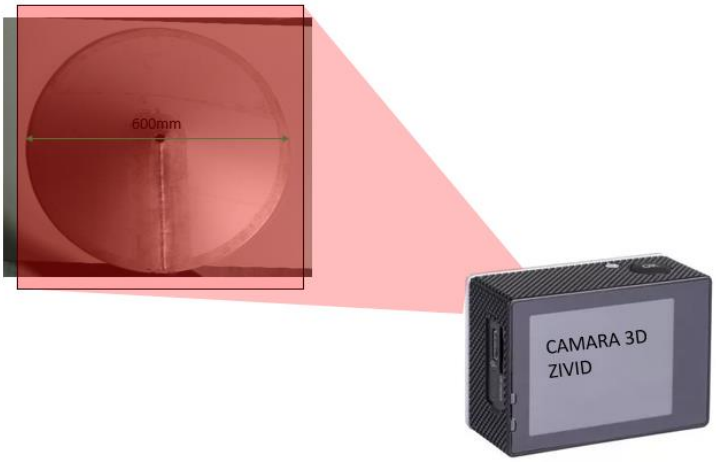


Figura 91 - Paradigma do *setup* de prototipagem

Como descrito na figura 91, o *range* da câmara é capaz de cobrir toda a área útil do objeto para a análise e posterior execução do algoritmo que contempla a solução porém trata-se de um tampo com sessenta centímetros de diâmetros e que foi fornecido apenas para aplicações experimentais. Considere-se agora o paradigma para o sistema real em contexto industrial.

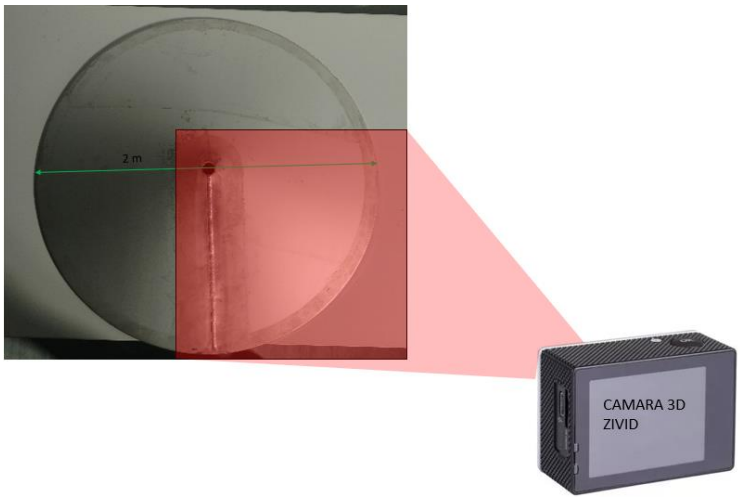


Figura 92 - Paradigma do *setup* em contexto indústrial

Uma vez aplicado o sistema à prática, ou seja, em contexto industrial para fazer jus ao propósito do seu desenvolvimento o sistema exatamente como está é falível porque não se preocupa com os sistemas de grandes dimensões, visto que a câmara 3D tem limitações distanciais para efetuar as capturas, como se pode ver na figura 92, para obter uma nuvem de pontos fidedigna do tampo em análise que tem um diâmetro de dois metros são necessárias quatro capturas. Para isso foi pensado um sistema que mova a câmara paralelamente à bancada de trabalho com a ajuda de motores de passo com uma relativa precisão para que seja medida a translação da câmara para depois ser efetuada um registo ponto a ponto, que significa simplesmente juntar as diferentes nuvens de pontos com base nas coordenadas onde foram

efetuadas as capturas e deste modo, obter uma única nuvem de pontos que considere todo o objeto para prosseguir para a restante execução do sistema. Ainda outro aspeto a considerar caso o diâmetro dos objetos varie muito é ler o ficheiro CAD e perceber o diâmetro de cada tampo e depois considerando o *range* da câmara basta mover esta um número mínimo de vezes baseado no arredondamento da divisão entre estas duas varáveis, respetivamente. Por exemplo, considera-se um objeto no plano horizontal com diâmetro de três metros e uma câmara 3D que tem um *range* de um metro, calcula-se $\frac{diametro\ do\ objeto}{range\ da\ câmara} = \frac{3}{1} = 3$, o que significa que tem que tirar nove capturas, $3_{eixo\ x} \times 3_{eixo\ y}$, como demonstra a figura 93.

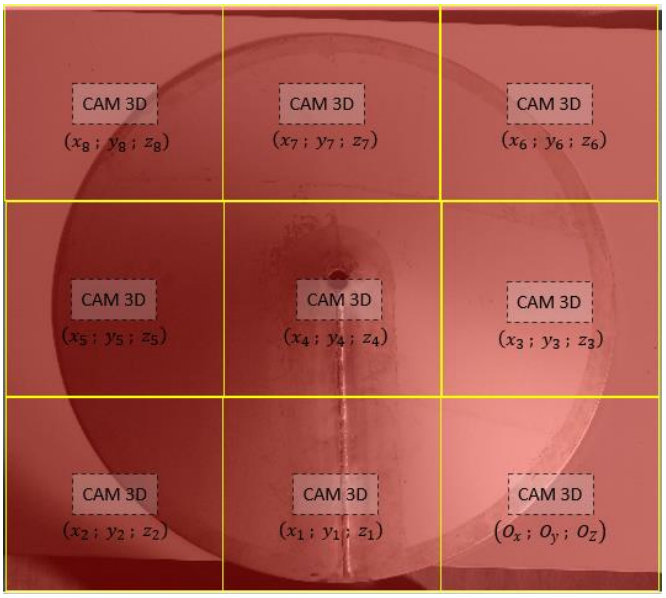


Figura 93 - Ilustração do movimento de uma câmara 3D para efetuar capturas para o registro do objeto completo.

A última parte do trabalho a desenvolver futuramente é otimizar os diversos algoritmos sendo que o mais importante de otimizar será o que lida com os cálculos da deformação, isto porque o algoritmo num certo ponto faz um varrimento pela malha do modelo para encontrar todos os polígonos circundantes a um vértice e corre a malha até seis vezes, uma alternativa de otimização é usar um algoritmo que use um circulador em torno do vértice. O algoritmo de alinhamento também deve ser revisto de modo a implementar mais restrições para rejeitar pares de pontos porque este algoritmo despende muito tempo no global do sistema.

Em suma, existem diversas melhorias para implementar, contudo o desenvolvido até agora tem uma eficiência e eficácia bastante satisfatória quando comparado com o trabalho convencional atual para a resolução do mesmo problema.

Referências

[1] "Automação de processos - Transformação Digital." [Online]. Available: https://www.jornaldenegocios.pt/transformacao-digital/detalhe/automacao-de-processos-o-que-tem-a-sua-empresa-a-ganhar. [Accessed: 20-Jun-2019].

[2] N. J. Groetelaars, "Tecnologia 3D Laser Scanning : características , processos e ferramentas para manipulação de nuvens de pontos," *XV Congr. SIGRADI*, pp. 1–5, 2011.

[3] "FEM - Finite Element Method - Wiki." [Online]. Available: https://pt.wikipedia.org/wiki/Método_dos_elem.

[4] C. Stoll, Z. Karni, C. Rössl, H. Yamauchi, and H. P. Seidel, "Template deformation for point cloud fitting," *Proc. 3rd Eurographics / IEEE VGTC Conf. Point-Based Graph.*, pp. 27–35, 2006.

[5] M. Botsch and M. Gross, "Adaptive Space Deformations Based on Rigid Cells botsch_2007_ASD.pdf," vol. 26, no. 3, 2007.

[6] Q. X. Huang, B. Adams, M. Wicke, and L. J. Guibas, "Non-rigid registration under isometric deformations," *Eurographics Symp. Geom. Process.*, vol. 27, no. 5, pp. 1449–1457, 2008.

[7] B. Louhichi, G. N. Abenhaim, and A. S. Tahan, "CAD/CAE integration: updating the CAD model after a FEM analysis," *Int. J. Adv. Manuf. Technol.*, vol. 76, no. 1–4, pp. 391–400, 2014.

[8] "Range imaging - Wiki." [Online]. Available: https://en.wikipedia.org/wiki/Range_imaging.

[9] T. Svoboda, H. Hug, and L. Van Gool, "ViRoom — Low Cost Synchronized Multicamera System and Its Self-calibration," 2002, pp. 515–522.

[10] D. Bračun, M. Jezeršek, and J. Diaci, "Triangulation model taking into account light sheet curvature," *Meas. Sci. Technol.*, vol. 17, no. 8, pp. 2191–2196, Aug. 2006.

[11] "What is Structured Light Imaging? | RoboticsTomorrow." [Online]. Available: https://www.roboticstomorrow.com/article/2018/04/what-is-structured-light-imaging/11821. [Accessed: 20-Jun-2019].

[12] "Basler Time-of-Flight Camera | Basler." [Online]. Available: https://www.baslerweb.com/en/products/cameras/3d-cameras/time-of-flight-camera/. [Accessed: 20-Jun-2019].

[13] "Large Zivid One Plus 3D camera." [Online]. Available: http://www.zivid.com/zivid-one-plus-large-3d-camera. [Accessed: 20-Jun-2019].

[14] "Projetor Laser LAN ILP 722 com Controle de Rede - Lasershop.de." [Online]. Available: https://www.lasershop.de/de/laserprojektor-ilp722-572.html. [Accessed: 20-Jun-2019].

[15] R. Bogdan Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," 1995.

[16] "SOR filter - CloudCompareWiki." [Online]. Available: https://www.cloudcompare.org/doc/wiki/index.php?title=SOR_filter. [Accessed: 28-May-2019].

[17] "3D Delaunay Triangulation on the GPU." [Online]. Available:

https://www.comp.nus.edu.sg/~tants/gdel3d.html. [Accessed: 20-Jun-2019].
[18] "CGAL 4.14 - Poisson Surface Reconstruction: User Manual." [Online]. Available: https://doc.cgal.org/latest/Poisson_surface_reconstruction_3/index.html. [Accessed: 20-Jun-2019].
[19] "Estimating Surface Normals in a PointCloud." [Online]. Available: http://pointclouds.org/documentation/tutorials/normal_estimation.php. [Accessed: 28-May-2019].
[20] "Poisson Surface Reconstruction | Lib4U." [Online]. Available: https://globlib4u.wordpress.com/2013/05/06/poisson-surface-reconstruction/. [Accessed: 28-May-2019].
[21] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson Surface Reconstruction," 2006.
[22] "» Poisson Surface Reconstruction - 3D Scan 2.0." [Online]. Available: http://vr.tu-freiberg.de/scivi/?page_id=365. [Accessed: 28-May-2019].
[23] "Octree – Wikipédia, a enciclopédia livre." [Online]. Available: https://pt.wikipedia.org/wiki/Octree. [Accessed: 28-May-2019].
[24] "Marching cubes – Wikipédia, a enciclopédia livre." [Online]. Available: https://pt.wikipedia.org/wiki/Marching_cubes. [Accessed: 28-May-2019].
[25] W. E. Lorensen and H. E. Cline, "Computer Graphics, Volume 21, Number 4, July 1987, MARCHING CUBES: A HIGH RESOLUTION 3D SURFACE CONSTRUCTION ALGORITHM," in *ACM siggraph computer graphics*, vol. 21, no. 4, 1987, pp. 163–169.
[26] "STL format." [Online]. Available: https://en.wikipedia.org/wiki/STL_(file_format).
[27] "Iterative Closest Point - Wiki." [Online]. Available: https://en.wikipedia.org/wiki/Iterative_closest_point.
[28] M. Levoy and S. Rusinkiewicz, "Efficient Variants of the ICP Algorithm-2001-Cit=1307.pdf."
[29] "Euclidean distance - Wiki." [Online]. Available: https://en.wikipedia.org/wiki/Euclidean_distance#equation_1.
[30] "Decomposição em valores singulares - Wiki." [Online]. Available: https://pt.wikipedia.org/wiki/Decomposição_em_valores_singulares.
[31] "Interactive Iterative Closest Point." [Online]. Available: http://pointclouds.org/documentation/tutorials/interactive_icp.php. [Accessed: 29-May-2019].
[32] "Kd-Tree - Wiki." [Online]. Available: https://pt.wikipedia.org/wiki/Árvore_k-d.
[33] "Polygon mesh - Wiki."
[34] "Posição relativa entre reta e plano - Brasil Escola." [Online]. Available: https://brasilescola.uol.com.br/matematica/posicao-relativa-entre-reta-plano.htm. [Accessed: 30-May-2019].
[35] "Ray Tracing: Rendering a Triangle (Ray-Triangle Intersection: Geometric Solution)." [Online]. Available: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/ray-triangle-intersection-geometric-solution. [Accessed: 30-May-2019].
[36] "Computing the closest point on a segment to a point - Diego Assencio." [Online]. Available: https://diego.assencio.com/?index=ec3d5dfdfc0b6a0d147a656f0af332bd. [Accessed: 30-May-2019].
[37] A. Sawicki *et al.*, "C + + / C L I T u t o r i a l," no. December, pp. 1–30, 2011.
[38] "Eyeshot, CAD and 3D Graphics for WinForms and WPF - devDept Software." [Online]. Available: https://www.devdept.com/Products/Eyeshot. [Accessed: 31-May-2019].