Universidade de Aveiro
2005

Departamento de (designação do departamento)

NOME COMPLETO

TÍTULO DA TESE (MÁXIMO 70 CARACTERES)

DOCUMENTO PROVISÓRIO

Universidade de Aveiro
2005

Departamento de Departamento de Electrónica, Telecomunicações e Informática

António Filipe Neves de Lima Abreu

TÍTULO DA TESE (MÁXIMO 70 CARÁCTERES)

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em (designação do mestrado), realizada sob a orientação científica do Dr. (nome do orientador), Professor (categoria do professor) do Departamento de (designação do departamento) da Universidade de Aveiro

texto Apoio financeiro do POCTI no âmbito do III Quadro Comunitário de Apoio.

(se aplicável)

texto Apoio financeiro da FCT e do FSE no âmbito do III Quadro Comunitário de Apoio.

(se aplicável)

texto Dedico este trabalho à minha esposa e filho pelo incansável apoio.

(opcional)

o júri

presidente

Prof. Dr. João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Dr. João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Dr. João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Dr. João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Dr. João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Dr. João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

Prof. Dr. João Antunes da Silva
professor associado da Faculdade de Engenharia da Universidade do Porto

agradecimentos

texto O presente trabalho propõe-se divulgar as mais significativas técnicas de construção existentes em Portugal continental. O livro é composto por uma apresentação dos materiais tradicionais de construção (suas principais características), uma compilação de fichas técnicas (de carácter prático, uma vasta bibliografia comentada e um glossário de termos técnicos.
A importante colaboração de diversas personalidades ligadas à área da História da Arquitectura, bem como o levantamento fotográfico realizado contribuem para o conhecimento e valorização de um saber tradicional.

(opcional)

palavras-chave

texto livro, arquitectura, história, construção, materiais de construção, saber tradicional.

resumo

texto O presente trabalho propõe-se divulgar as mais significativas técnicas de construção existentes em Portugal continental. O livro é composto por uma apresentação dos materiais tradicionais de construção (suas principais características), uma compilação de fichas técnicas (de carácter prático, uma vasta bibliografia comentada e um glossário de termos técnicos.
A importante colaboração de diversas personalidades ligadas à área da História da Arquitectura, bem como o levantamento fotográfico realizado contribuem para o conhecimento e valorização de um saber tradicional.

keywords

texto livro, arquitectura, história, construção, materiais de construção, saber tradicional.

abstract

texto O presente trabalho propõe-se divulgar as mais significativas técnicas de construção existentes em Portugal continental. O livro é composto por uma apresentação dos materiais tradicionais de construção (suas principais características), uma compilação de fichas técnicas (de carácter prático, uma vasta bibliografia comentada e um glossário de termos técnicos.
A importante colaboração de diversas personalidades ligadas à área da História da Arquitectura, bem como o levantamento fotográfico realizado contribuem para o conhecimento e valorização de um saber tradicional.

Índice

Índice 8
I. Introdução 9
II. Métodos de obtenção de um modelo virtual, a partir de um dado espaço real. 10
III. Aquisição de um modelo de pontos virtuais a partir de um modelo real. 11
IV. Limitações de um *scanner* laser. 12
V. Exemplos de *scanners* do tipo tempo de reflexão 13
V.1. 3rdTech DeltaSphere 3000 (http://www.deltasphere.com/) 13
V.2. Cyra Cyrax 2500 scanner (http://www.cyra.com/) 13
V.3. Quantapoint (http://www.quantapoint.com/) 13
V.4. Riegl Laser Scanner LMS-Z390i (http://www.riegl.com/) 13
V.5. Scanner 2D SICK LMS 200 14
VI. Criação do modelo final 15
VI.1. Aquisição da nuvem de pontos 15
VI.2. Leitura do ficheiro da nuvem de pontos 15
VI.3. Algoritmos de triangulação 15
VI.4. Método de triangulação com recurso ao algoritmo Delaunay 16
VI.5. Método de triangulação com detecção de contornos 17
VI.6. Remoção de pontos desnecessários. 18
VI.6.1. Remoção de pontos situados nos limites mínimos e máximos verticais. 18
VI.6.2. Remoção de pontos linhas de pontos com a mesma angular horizontal. 18
VI.6.3. Remoção de pontos com coordenadas fora de um dado volume 19
VI.6.4. Remoção de pontos sobrepostos 20
VI.6.5. Mudança de um modelo de pontos para um modelo triangulado. 20
VI.6.6. Remoção de pontos de modelos que, quando triangulados, dão origem a triângulos com área nula. 21
VI.7. Aproximação de modelos 21
VI.7.1. Aproximação de modelos utilizando três pontos no espaço. 23
VI.7.2. Cálculo da matriz de transformação a partir do algoritmo ICP 26
VI.8. Junção de modelos. 28
VI.8.1. Remoção de pontos não utilizados durante a triangulação / junção de modelos. 30
VI.8.2. Identificação e substituição de superfícies comuns aos dois modelos. 31
VI.9. Redução/simplificação do número de triângulos do modelo 33
VI.9.1. Benefícios e perdas da simplificação do modelo triangulado 33
VI.9.2. Descrição do algoritmo de simplificação do modelo triangulado 34
VII. Aplicação principal 37
VII.1. Introdução 37
VII.2. Descrição geral 37
VII.3. Módulos principais 39
VII.3.1. Objecto: Modelo (`modelManager`) 40
VII.3.2. Objecto: Funções de operação directa sobre modelos (`Functions3D`) 45
VII.3.3. Objecto: Janela de Visualização (`renderWindow`) 46
VII.3.4. Objecto: Janela Real (`windowManager`) 49
VII.3.5. Objecto: Janela da aplicação (`mainWindow`) 51
VII.4. Interface gráfica 56
VII.4.1. Janela principal 56
VII.4.2. Janela real 57
VII.4.3. Janelas secundárias de interacção directa com o utilizador 58
VII.5. Interface do utilizador 60
VII.5.1. Menu principal da aplicação 60
VII.5.2. Opções de processamento da aplicação 62
VII.5.3. Janela de interacção VTK 65
VIII. Bibliografia 67
IX. Índice de Figuras 68

I. Introdução

Mais que nunca, cada vez mais se recorre à realidade virtual como ferramenta de análise, entretenimento, meio de aculturação, planificação e projecção de projectos, suporte ao processo artístico e até como indução por simulação de certos estados mentais.
Neste vasto leque de opções, a utilização da realidade virtual como ferramenta de conhecimento e estudo de um dado espaço real, vai de encontro ao objectivo deste documento - Nomeadamente todo o processo inerente à aquisição, análise, processamento e visualização de um dado espaço virtual obtido a partir de um espaço físico, de forma a obter um modelo virtual que mais se aproxima do modelo real que lhe deu origem.
Devido à diversidade de ferramentas, algoritmos, aplicações e métodos de conseguir esta representação virtual, justifica-se uma análise ao panorama do que já existe e permanece em contínua evolução.
Pelo que após esta análise, uma comparação realista e actualizada poderá mostrar de que forma as ferramentas e métodos desenvolvidos conseguem ser mais eficazes tendo em conta factores de facilidade, potencial, custo e fiabilidade.

II. Métodos de obtenção de um modelo virtual, a partir de um dado espaço real.

Existem variadíssimas formas de obter informação relativa a um dado espaço (real) e transferir a mesma para dados digitais.

Entre esses métodos, destacam-se os seguintes grupos:

1) Captura dos efeitos dos sinais reflectidos a partir dos próprios materiais e projectados sobre uma superfície receptora. (fotografia digital, medidor de velocidade por efeito de Doppler, entre outros).
2) Varrimento do espaço um emissor de energia próprio, a fim de obter um conjunto de pontos com informação relativa à posição das superfícies presentes nesse mesmo espaço devido ao tempo de refracção de certos materiais a estas fontes de energia. (varrimento laser do espaço circundante, eco localização, entre outros)
3) Obtenção relativa à estrutura interna dos materiais através dos efeitos de refracção obtidos a partir da incidência de certas formas de energia capazes de penetrar a estrutura de um dado material. (Rais X, Ultra-sons, entre outros).
4) Outros métodos que são combinações de dois ou mais dos grupos acima (aparelhos de leitura da forma de uma dada superfície, recorrendo a um laser, que ao mesmo tempo vai tirando fotos acerca da mesma, presentes nalguns satélites artificiais).

Nota: Após a obtenção destes valores, é necessário processar os mesmos e até reajustar os aparelhos de leitura a partir da data processada, de forma a conseguir a representação mais coerente possível de certas características de um dado espaço.

III. Aquisição de um modelo de pontos virtuais a partir de um modelo real.

Uma das primeiras etapas na fase de aquisição de aquisição de dados a partir de um espaço real, implica o recurso a um aparelho de aquisição de um conjunto de valores, a partir das características físicas desse mesmo espaço.
Entre os vários métodos de aquisição, aqueles que mais vão de encontro ao trabalho realizado, concentram-se nos métodos de aquisição de pontos a partir de um dado espaço real, recorrendo a um *scanner* de raio laser.
Este *scanner* faz a aquisição da distância de pontos de uma dada superfície até ao emissor do raio, recorrendo ao valor do tempo que demorou a ser recebido um dado raio, desde que este foi emitido, por efeito de reflexão do mesmo. A partir do valor deste tempo, sabendo a velocidade de propagação do laser num dado meio, sabendo os tempos de atraso (sensores, emissores, circuitos) é possível calcular com uma precisão relativamente elevado o valor desta distância.
Dos variadíssimos tipos de *scanner* de raio laser existentes, abaixo é possível observar uma lista de modelos bastante conhecidos, baseados na leitura e aquisição dos dados a partir de um ponto central, baseados no tempo de envio e recepção de um raio laser, após a sua reflexão com uma dada superfície:

IV. Limitações de um *scanner* laser.

- Área do raio laser

Quanto maior for o diâmetro do raio que irá efectuar a medição das distâncias, pior será a resolução do mesmo e mais falível será o seu resultado, quando este incidir sobre uma superfície desnivelada.
Também existe um decréscimo de precisão com o aumento da distância devido ao facto da área do raio laser aumentar com o aumento da distância, devido ao facto do raio (procurar o termo da abertura de um raio laser).

- Reflexões múltiplas

Caso um superfície reflectir o raio de uma forma direccionada para uma segunda superfície, o aparelho irá medir o tempo de percurso do raio entre estas superfícies e não o tempo de percurso atè à zona que se pretende ler. Neste caso ter-se-á um valor errado de profundidade.

- Superfícies não reflectoras

Se uma dada superfície tiver um índice de reflexão demasiado baixo, o sensor não irá conseguir captar qualquer reflexão do raio emitido, pelo que irá considerar que essa mesma superfície corresponde está a uma distância infinita.

- Precisão do tempo de percurso

O tempo de percurso de um raio laser, além de depender do meio em que se encontra, também é extremamente pequeno, pelo que, para o seu cálculo, terá de ser usado um circuito que, apesar da sua elevada precisão, tem sempre um erro associado ao mesmo. A adicionar a este erro também de pode adicionar o tempo de reacção do sensor que irá medir a reflexão do raio. Todos estes factores, em conjunto, irão aumentar bastante o erro associado ao da distância de um dado raio.

- Precisão da direcção do raio laser

Apesar de ser bastante preciso, a direcção do raio tem sempre um erro associado, pelo que o ponto da superfície que se está a pretender medir pode não corresponder exactamente ao ponto em que incide o laser.

V. Exemplos de *scanners* do tipo tempo de reflexão

V.1. 3rdTech DeltaSphere 3000 (http://www.deltasphere.com/)

Trata-se de um modelo que se denota pela possibilidade de, além da profundidade, permitira a captura da cor associada a cada ponto e de uma elevada precisão.
Características gerais:
- Possibilidade de captura angular até 360 graus horizontais, por 290 graus verticais.
- Pode ser ligado a um portátil a correr o sistema operativo Windows™.
- Permite a captura, além da profundidade, da cor de um dado ponto, de forma a criar modelos com a cor real.
- Tem um alcance entre 0,5 a 15 metros com precisão de 7 milímetros, tendo um raio de diâmetro de 2 milímetros.

V.2. Cyra Cyrax 2500 scanner (http://www.cyra.com/)

Trata-se de um *scanner* pensado para captura de estruturas reais para representações em programas tipo CAD (Computer Aided Design – desenho auxiliado por computador).
Tem sido utilizado na reestruturação de plantas de edifícios a fim de se obter modelos tridimensionais actualizados e realistas, dos mesmos.
A precisão deste *scanner* é de 6 milímetros para distâncias entre 1,5 a 50 metros.
Captura volumes até um máximo de 20.000 metros cúbicos.

V.3. Quantapoint (http://www.quantapoint.com/)

Recomendado para capturas rápidas, sem a necessidade de material extra.

Características gerais:

- Tem capacidade de guardar os dados capturados internamente (não necessita de um servidor de dados capturados).
- Possibilidade de especificação de zonas de maior resolução, ou densidade de pontos.
- Possibilidade de gerar de mapas de pontos de tamanho reduzido, inferiores aos formatos usuais.
- Geração de mapas de superfícies, ao contrário de mapas de pontos, de forma a facilitar a análise e integração do modelo.
- Mede 125.000 pontos por segundo e captura até ângulos de 70 por 360 graus (vertical por horizontal).

V.4. Riegl Laser Scanner LMS-Z390i (http://www.riegl.com/)

Trata-se de um *scanner* de elevada qualidade, ideal para múltiplas capturas de elevada precisão, para grandes áreas.

Características gerais:

- Tem uma precisão de 6 milímetros para distâncias entre 1/2 a 140/400 metros (depende do índice de reflexão do alvo).
- Permite capturar ângulos até 360 graus horizontais e 80 verticais.

- Utiliza um conjuntos de sensores de posicionamento, geográficos (GPS), altitude (altímetro), direcção (bússola) e inclinação de forma a posicionar com bastante precisão os modelos de pontos capturados, entre si.

V.5. Scanner 2D SICK LMS 200

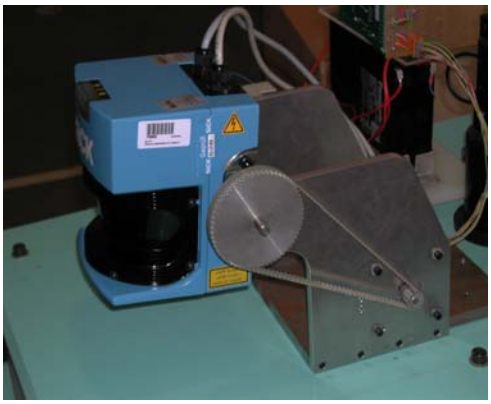


Figura V.1 - Scanner 2D SICK LMS 200

Trata-se de um *scanner* manufacturado na Universidade de Aveiro, com uma precisão e resolução suficientes ao objectivo deste projecto.
O modo de operação deste *scanner*, baseia-se num projector laser, pousado sobre uma superfície estável, no centro do espaço que se pretende obter a malha de pontos que o descreva. Para isso, efectua leituras a partir do tempo de viagem deste que o laser é emitido, até à altura em que este retorna a um receptor junto do emissor, devido ao efeito de reflexão do material em que incide o raio.

VI. Criação do modelo final

VI.1. Aquisição da nuvem de pontos

De todos os métodos existentes, aquele por que se optou para aquisição de dados de uma dado espaço real, foi o *scanner* 2D (SICK LMS 200).
De facto, apesar deste *scanner* não ter nem a precisão, nem a resolução dos modelos acima analisados, consegue efectuar um apanhado da **anta pintada de Antelas** com resolução e precisão suficientes para, no final obter-se um modelo tridimensional realista e fidedigno ao formato da anta, propriamente dita.

Durante o processo de aquisição dos pontos, uma das formas de se conseguir um valor de profundidade com um erro associado menor, é efectuando mais que uma leitura por cada ponto – este método elimina erros do tipo aleatório.

Esta leitura é guardada num ficheiro de texto com linhas com a distancia de cada um dos pontos com a mesma angular vertical, variando do ponto com a menor angular horizontal, até à maior, de forma crescente.

VI.2. Leitura do ficheiro da nuvem de pontos

A fim de efectuar a leitura dos pontos adquiridos a partir do *scanner* 2D (SICK LMS 200), guardando-os num ficheiro de texto, acima descrito, foi criado um objecto capaz de efectuar a leitura de uma ou mais nuvens de pontos para um dado modelo em memória.
Esta leitura cria, em memória, um modelo dos pontos em coordenadas cartesianas, a três dimensões, dispostos no espaço, mas preservando uma organização tal que é possível saber qual posição relativa de cada ponto na matriz de leitura dos pontos. Esta informação é de extrema importância, pois irá ser utilizada nos algoritmos de triangulação desenvolvidos.
Na figura abaixo é possível observar que, recorrendo a um sistema de coordenada esféricas, todos os pontos têm uma variação dos valores de rotação horizontal e vertical igual, dando origem a uma espécie de matriz (ou malha) de pontos. O único valor apenas dependente da superfície a ser amostrada é o da distância do ponto amostrado ao laser:

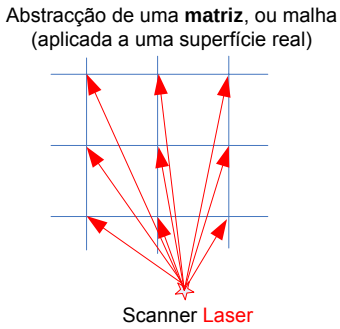


Figura VI.1 - Visualização da nuvem de pontos como uma matriz de pontos

VI.3. Algoritmos de triangulação

A fim de se aplicar uma “pele”, ou superfície (virtual), ao conjunto de pontos referente à malha de pontos acima referida, é necessário unir todos estes pontos recorrendo a triângulos de modo a obter-se uma superfície virtual, o mais similar possível à superfície real (que deu origem aos pontos da malha) no modelo virtual.

A aplicação directa da triangulação a uma nuvem sem quaisquer processos de pré-filtragem, resultaria numa triangulação demasiado complexa, com uma superfície demasiado irregular.
Pelo que, é aplicada uma filtragem que pretende determinar a posição de possíveis vértices ou falhas de continuidade, de forma a não unir superfícies que na realidade não é suposto unir.
Além desta detecção de vértices, também se utilizam valores de *treshold*, ou seja, uma distância mínima e máxima para os pontos adquiridos, relativamente à sua distância ao *scanner* de forma a evitar a utilização de pontos com distâncias fora do intervalo aconselhado para o *scanner* (pontos fora deste intervalo costumam ter um erro associado mais elevado).
Durante o processamento dos pontos (da nuvem de pontos) são removidos conjuntos de linhas verticais com igual angular horizontal. Este procedimento e o porquê da sua existência serão analisados com mais detalhe na secção **remoção de pontos**.

VI.4. Método de triangulação com recurso ao algoritmo Delaunay

Neste método explora-se o facto da superfície que corresponde a todos os pontos ser uma superfície contínua, como se pertencesse a uma espécie de "rede virtual".
O algoritmo Delaunay é um algoritmo de triangulação em que, dada superfície de duas dimensões com um dado conjunto de pontos, tem por objectivo a união desses mesmos pontos criando um conjunto de triângulos com o menor tamanho possível e de forma a não deixar áreas da superfície definida pelo conjunto de pontos sem triângulos, ou com "buracos".
Não sendo possível aplicar este algoritmo directamente sobre o conjunto de pontos a três dimensões, este método converte este conjunto de pontos numa matriz de pontos a duas dimensões, com o mesmos índices que os pontos a três dimensões, ou seja, dispostos entre si no formato original da matriz de aquisição dos pontos.
De seguida é aplicado o algoritmo Delaunay a estes pontos de duas dimensões, a fim de se obter um conjunto de triângulos a unirem todos esses pontos.
De seguida utilizando os índices dos pontos a que se refere cada triângulo, é utilizado o conjunto de pontos a três dimensões com os mesmos índices que antes eram pontos a duas dimensões e agora passam a ser pontos a três dimensões.
Na figura abaixo é possível observar uma descrição gráfica deste processo.

(imagem da transformação entre um sistema 3D e um sistema 2D + 1 (coordenadas esféricas)

A explicação algorítmica pode ser dada da seguinte forma:
- Transforma-se o conjunto de pontos a três dimensoes (x,y,z) para valores de duas dimensões (x',y'), sendo ordenados segundo a matriz de ordenação dos pontos no formato polar, referente ao valor da angular horizontal e vertical, respectivamente.
- Aplica-se a este modelo de matriz uma triangulação do tipo Delaunay de modo a obter uma cobertura para todos os pontos.
- Utilizando os índices dos pontos que formam cada triangulo, volta-se a aplicar essa estrutura aos índices dos pontos 3D para termos um conjunto de triângulos a unir os mesmos, formando uma malha tridimensional.

No entanto este procedimento não tem em conta a eliminação de triângulos presentes em arestas ou zonas em que hajam descontinuidades da superfície.
Este método também não elimina pontos cuja distancia ao emissor laser ou por qualquer outra condição tenham sido descartados como pontos de triangulação.

VI.5. Método de triangulação com detecção de contornos

Um segundo método de triangulação desenvolvido de modo a não criar triângulos entre pontos que:

- Estejam sobre zonas de desníveis abruptos em relação aos pontos vizinhos (detecção de contornos simples).
- Tenham sido considerados inválidos para triangulação por causa da sua distância ao *scanner* estar fora do intervalo de segurança.
- Pertençam aos extremos verticais inferior e superior da nuvens de pontos, visto serem considerados pontos com erro associado elevado.

Este método, tal como o método de triangulação que recorre ao algoritmo Delaunay, explora o facto dos índices dos pontos a três dimensões poderem ser acedidos relativamente à sua posição original, relativamente à matriz de aquisição dos pontos.

Partindo deste princípio, a construção dos triângulos pode ser reduzida a um conjunto de grupos de quatro pontos, sendo que cada um desses grupos dará origem a dois triângulos a unir os seus pontos. Na figura abaixo é possível observar um exemplo de como seriam unidos um conjunto de nove pontos dispostos numa relação de três por três:

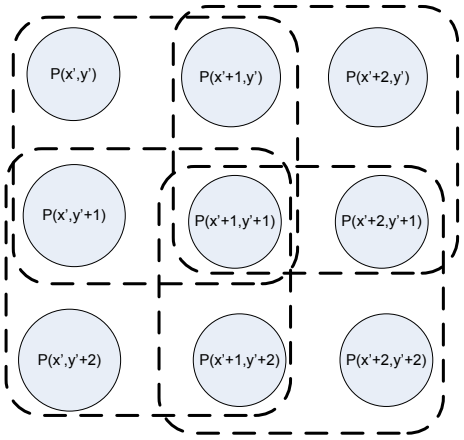


Figura VI.2 - Selecção da matriz de pontos, em quadrados de triangulação

A partir da figura acima, é possível verificar que, cada grupo tem quatro pontos, todos os pontos laterais pertencem a dois grupos, pontos das esquinas pertencem a um grupo e pontos centrais pertencem a quatro grupos.

A diagonal escolhida comum aos dois triângulos é que apresenta uma distância entre pontos inferior, de forma a melhor adaptar a superfície virtual, definida pelos pontos, à disposição dos triângulos. Na figura abaixo é possível verificar as duas opções de criação de triângulos existentes:

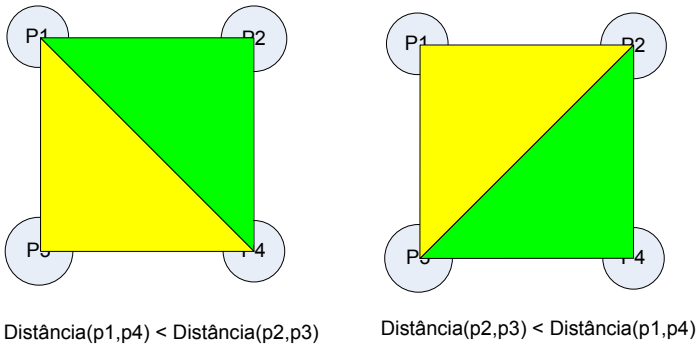


Figura VI.3 - Opções de triangulação de um quadrado baseado nas distâncias dos pontos

A diagonal comum aos dois triângulos seleccionada, será a de distância mais curta. Esta opção deve-se ao facto de, ao seleccionar a diagonal mais curta reduz-se o a possibilidade de descontinuidades e aumenta a probabilidade da triangulação melhor representar a o espaço de onde foi amostrada.

VI.6. Remoção de pontos desnecessários.

Após se ter efectuado a aquisição dos pontos, efectuado a leitura do mesmo e obtido uma nuvem de pontos, é necessário proceder a um conjunto de operações que visam a remoção de pontos desnecessários ou erróneos.
A questão da remoção de pontos, tem um grau de complexidade que implica a sua subdivisão nas etapas seguintes:

- Remoção de pontos situados nos limites mínimos e máximos verticais.
- Remoção de pontos linhas de pontos com a mesma angular horizontal.
- Remoção de pontos com coordenadas fora de um dado volume.
- Remoção de pontos que se sobreponham entre si.
- Remoção de pontos de modelos que, quando triangulados, dão origem a triângulos com área nula.
- Remoção de pontos não utilizados durante a triangulação / junção de modelos.

Embora esta operação seja subdividida em vários tópicos, o processo de remoção de pontos desnecessários é efectuado numa única função.

VI.6.1. Remoção de pontos situados nos limites mínimos e máximos verticais.

Verifica-se que os pontos, em termos de coordenadas esféricas, com uma angular vertical mínima e máxima, apresentam um erro associado bastante elevado, pelo que se opta pela sua não inclusão (ou remoção), durante a fase de leitura do ficheiro da nuvem de pontos.
A figura abaixo exemplifica quais são esses mesmos pontos:

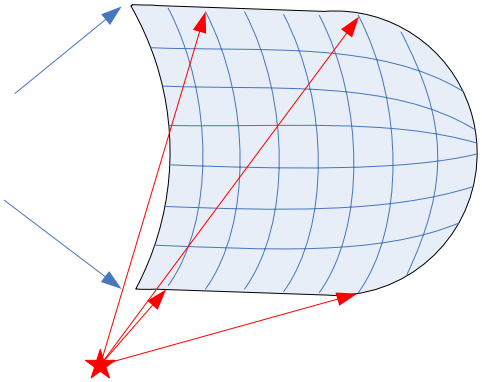


Figura VI.4 - Linhas de pontos a eliminar da matriz de pontos

Nota: esta filtragem de pontos é efectuada aquando da leitura do ficheiro da nuvem de pontos.

VI.6.2. Remoção de pontos linhas de pontos com a mesma angular horizontal.

Embora não seja comum, certos *scanners* laser, por vezes apresentam mais que uma linha com igual valor de angular horizontal. Este evento, teoricamente, nunca deveria acontecer, devido à variação da rotação

horizontal da cabeça de leitura ser crescente ou decrescente, nunca devendo passar mais que uma vês por um dado valor de rotação horizontal.
Esta ocorrência deve-se ao facto deste valor estar sujeito a um dado erro de leitura que por vezes surge aquando da medição, ou até de arredondamento, do valor do ângulo horizontal calculado pelo *scanner*, sendo este igual a um anterior.
A forma encontrada para contornar este problema foi a de utilizar apenas linhas verticais cujo valor da angular horizontal seja superior ao anterior, caso estas se encontrem dispostas de forma crescente, ou vice-versa.
Caso esta operação não ocorra, o processo de triangulação poderá ficar seriamente comprometido, devido ao facto deste explorar o facto do valor da angular horizontal ser crescente, ou decrescente.
Na figura abaixo é possível observar um exemplo de um laser a efectuar um conjunto de leituras de linhas verticais, sendo que cada linha tem um angular maior que a anterior:

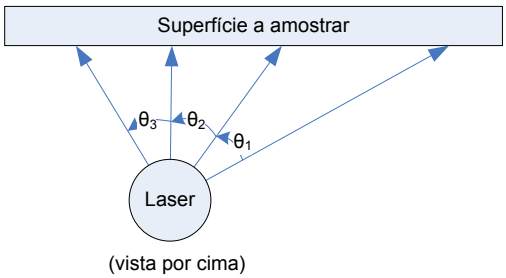


Figura VI.5 - Variação angular horizontal de um *scanner* laser

Conforme se pode observar, a existência de um ângulo $\theta_n \leq 0$ nunca deve acontecer. Apenas poderá ocorrer devido a erros de leitura da angular horizontal do *laser*, efectuado pela ADC (esta ADC tem sempre um dado erro associado que poderá ser decisivo à exclusão uma dada linha de pontos horizontal).

Nota: esta filtragem de pontos é efectuada aquando da leitura do ficheiro da nuvem de pontos.

VI.6.3. Remoção de pontos com coordenadas fora de um dado volume

Funções como a junção de modelos, algoritmos de aproximação por iteração, entre outras, podem ficar comprometidas caso não se efectue uma remoção de pontos que não pertençam a um dado volume definido por duas esferas de raios diferentes.
A esfera de raio inferior permite a remoção de pontos que pertençam ao volume definido pela mesma. Desta forma é possível remover pontos cujo valor da distância ao *scanner* seja inferior à distância mínima, segundo as especificações do próprio *scanner* laser com que foi adquirida a nuvem de pontos.
A esfera de raio superior permite a remoção de pontos que não pertençam à mesma. Deste modo é possível excluir pontos que com o valor da distância superior à distância máxima presente nas especificações do *scanner* laser.

A figura abaixo exemplifica este processo, apenas pontos entre as duas esferas (secção verde) serão aceites.

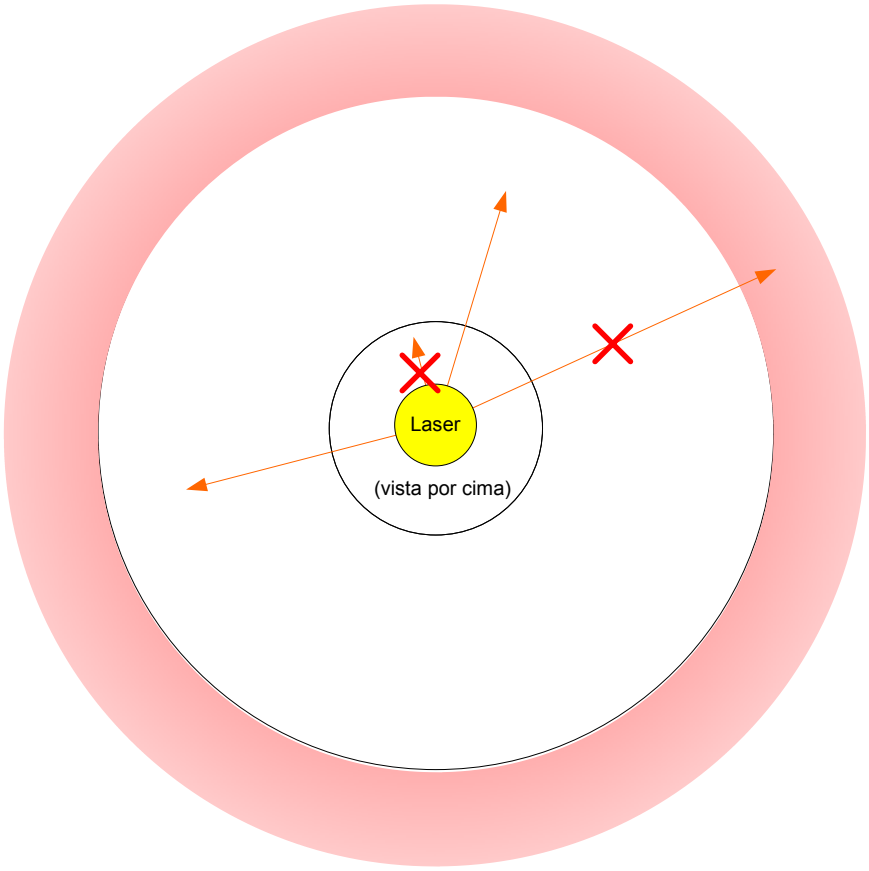


Figura VI.6 - Zonas permitidas e ilegais, baseadas na distância, para validação de pontos

Nota: esta filtragem é efectuada durante a leitura do ficheiro da nuvem de pontos – mas, em vez de eliminar os pontos, perdendo-se a relação matricial entre os mesmos, estes pontos são mantidos, mas são deslocados para a origem (ou centro) da nuvem de pontos de forma a posteriormente identificar e eliminar os mesmos.

VI.6.4. Remoção de pontos sobrepostos

De forma a evitar a presença de mais que um ponto com as mesmas coordenadas, é necessário proceder a uma procura, sobre o modelo triangulado, substituindo conjuntos de pontos com as mesmas coordenadas por um único ponto, actualizando quaisquer triângulos que utilizem os pontos desse grupo pelo ponto que irá substituir esse mesmo grupo de pontos.
Desta forma, além de se simplificar o modelo, também se melhora o resultado das rotinas de aproximação da nuvem de pontos, pois várias pontos nas mesmas coordenadas iriam criar um efeito cumulativo sobre o peso relativo das coordenadas definidas por esses mesmos pontos, o que (como adiante poderá ver-se pela descrição do processo de junção de modelos por aproximações sucessivas).

VI.6.5. Mudança de um modelo de pontos para um modelo triangulado.

Após invocar uma triangulação de um dado modelo de pontos (organizados de forma matricial), pontos que fiquem de fora da triangulação podem ser excluídos, pois, a partir da operação de triangulação, tanto a organização matricial dos pontos, como o conceito de nuvem de pontos muda para um conceito de superfícies de encaixe.

A figura abaixo exemplifica esta alteração:

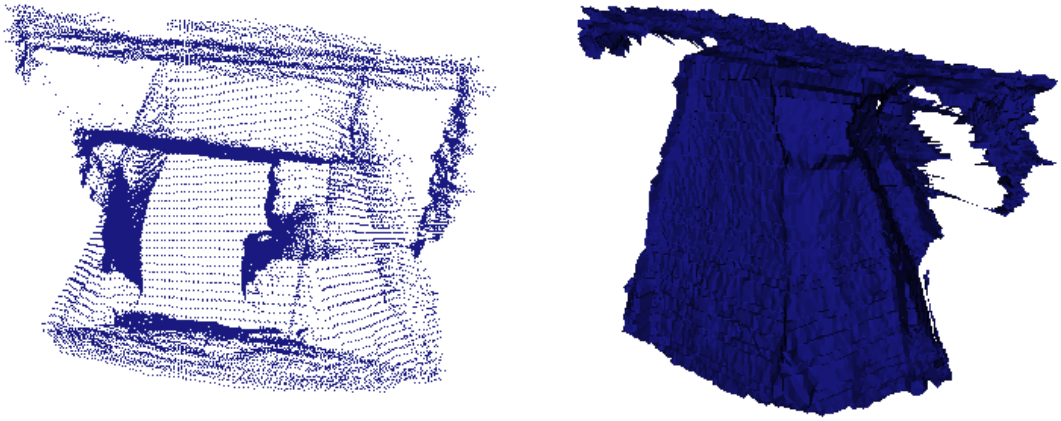


Figura VI.7 - Projecção de pontos vs. projecção de triângulos (após triangulação)

Nota: Pode-se verificar que vários pontos do modelo de pontos são ignorados no processo de triangulação. No entanto, o próprio modelo triangulado continua a ter associado um conjunto de pontos posicionados nos vértices dos triângulos do modelo triangulado.

VI.6.6. Remoção de pontos de modelos que, quando triangulados, dão origem a triângulos com área nula.

Por outro lado, triângulos com área nula podem surgir após proceder-se a uma fusão de dois ou mais modelos (descrita mais à frente).
A operação de remoção dos pontos constituintes de triângulos nulos, antes de ser efectuada, deve-se proceder a uma verificação de que os pontos dos triângulos a remover não são partilhados com nenhum triângulo de área não nula. A forma de efectuar esta operação é a de remover todos os triângulos com área nula ficando os pontos que só estejam associados a estes triângulos soltos.
O modo implementado para encontrar triângulos com área nula consiste na identificação de triângulos em que dois ou mais dos pontos que o constituam ocupem a mesma posição no espaço, ou tenham a uma distância entre si menor que um dado valor mínimo (de *treshold*).
Após ter-se procedido a uma remoção dos triângulos, é efectuada uma remoção de pontos não utilizados por qualquer triângulo, descrito no ponto abaixo.
A remoção destes triângulos e respectivos pontos, além de simplificar o modelo, também melhora a função de aproximação de modelos por aproximações sucessivas (método ICP, descrito mais adiante), pois promove uma distribuição mais uniforme de pontos no espaço, eliminando pontos que estejam muito perto de outros.

VI.7. Aproximação de modelos

De forma a ser possível unir modelos, que partilhem zonas comuns entre si, antes de mais é necessário saber que rotação e/ou translação a aplicar aos modelos que permite a este modelo “deslocar-se” de forma a “encaixar” nos restantes modelos, de forma a todos apresentarem as suas secções comuns em consonância entre si.
Tecnicamente falando, é necessário encontrar a matriz de transformação linear (matriz 4 x 4) capaz de deslocar o modelo adicionado para a posição que este deverá ocupar, relativamente ao modelo base (o utilizador deve seleccionar um dado modelo como base, relativamente ao qual todos os outros serão deslocados de forma a encaixarem entre si).

A fim de mostrar como estas matrizes operam, segue-se a explicação:

Cada ponto (*P*) no espaço tridimensional, pode ser descrito pelas suas componentes cartesianas *x*, *y* e *z*, usando uma matriz 1 x 4:

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

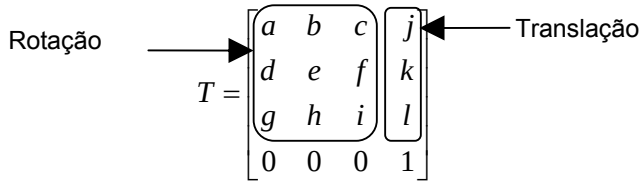
Uma matriz de transformação linear para espaços tridimensionais é do tipo 4 x 4:

$$T = \begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De forma a aplicar uma dada transformação linear a um conjunto de pontos (triângulos também são definidos pelos pontos nos seus vértices), aplica-se esta matriz de transformação linear a todos os pontos desse mesmo modelo, da seguinte forma:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Partindo do princípio que ambos os modelos estão na mesma escala, não se pretende uma alteração à escala dos modelos, logo a transformação linear é do tipo **transformação linear de corpo rígido**. Para o caso específico da transformação linear de corpo rígido (sem alterações de escala) pode-se efectuar uma análise aos elementos deste género de matriz, separarando-os em duas secções:



No caso de se ter as coordenadas e rotação exactas da posição do emissor laser no espaço, aquando da aquisição de cada uma das nuvens de pontos, esta operação não seria necessária, podendo-se calcular a matriz de transformação rígida a partir dos mesmos. No entanto isto muito raramente é possível.

Na figura abaixo é possível observar dois modelos de pontos, em que um é rodado e transladado de modo a ser adicionado ao outro:

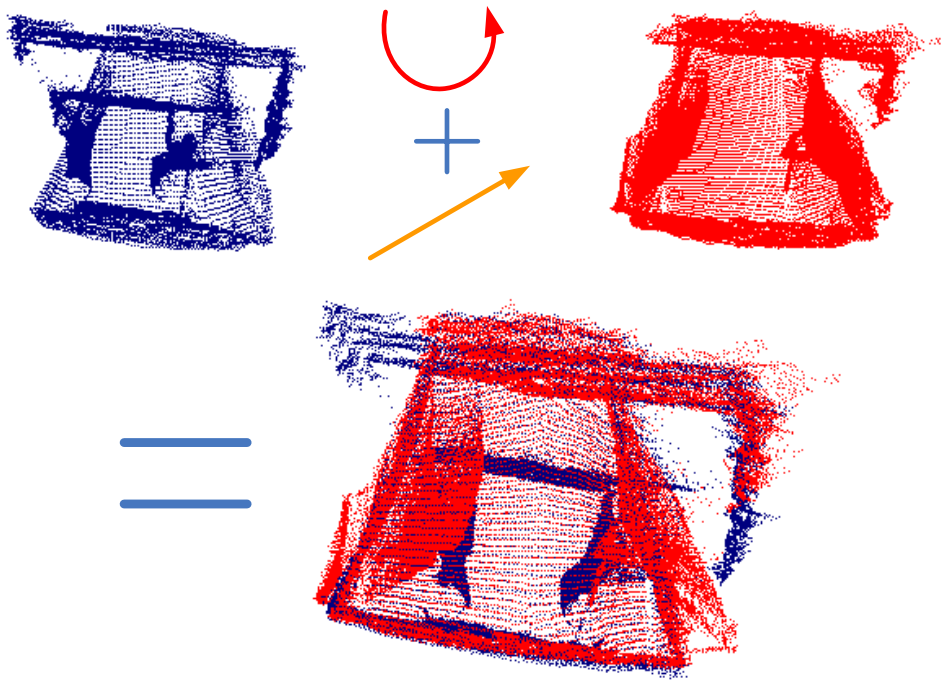


Figura VI.8 - Aproximação de dois modelos VTK por translação e rotação

VI.7.1. Aproximação de modelos utilizando três pontos no espaço.

Esta forma de aproximar dois modelos requer a intervenção do utilizador. É recomendada quando os modelos têm uma separação entre si relativamente elevada (situação em que outros métodos não são eficientes).
Neste método é pedido ao utilizador que indique três pontos, em cada um dos modelos, comuns aos dois, de forma a que a aplicação possa calcular qual a matriz de transformação rígida (acima descrita) que permita efectuar a translação e rotação de um modelo, de forma a aproximá-lo ao outro modelo.

De forma a encontrar a matriz de transformação dos três pontos, a aplicação tem disponíveis duas funções:

- Aproximação dos três pontos tendo em conta a ordem dos mesmos.
- Aproximação dos três pontos tendo em conta a forma geométrica descrita pela união dos mesmos.

Além dos dois modos de calcular, a aplicação também disponibiliza duas formas de projectar os modelos, a fim do utilizador marcar os pontos:

- Projecção dos pontos do modelo.
- Projecção de triangulada do modelo.

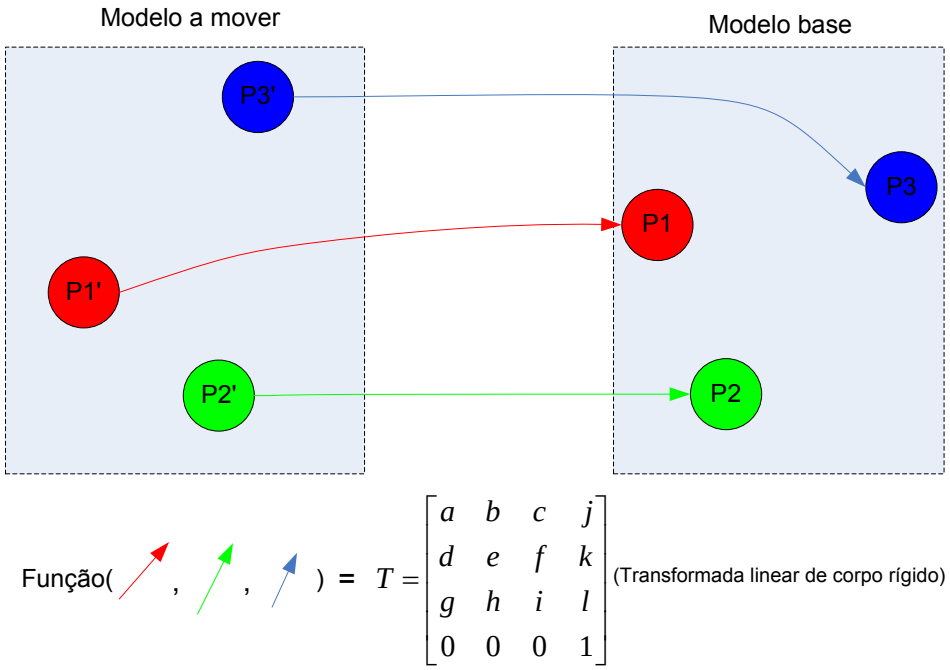
Aproximação dos três pontos tendo em conta a ordem dos mesmos

Esta função, parte dos três pontos comuns aos dois modelos, indicados pelo utilizador, mas tenta encontrar a transformação linear de corpo rígido (acima descrita) que altera a posição dos pontos seleccionada para a posição dos pontos no novo modelo, tendo em atenção a ordem dos pontos, ou seja – cada ponto num dos modelos tem o seu equivalente no outro modelo.

A fim do utilizador ser informado qual o ponto que acabou de marcar, estes são criados com cores diferentes:

1º ponto: cor vermelha
2º ponto: cor verde
3º ponto: cor azul

A imagem abaixo pretende exemplificar os conceitos acima descritos:



$$\begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura VI.9 - Cálculo da transformada linear capaz de aproximar 2 conjuntos de 3 pontos

A função implementada recorre ao objecto *vtkLandmarkTransform*, disponibilizado pela *framework VTK* a fim de calcular a matriz 'Transformada linear de corpo rígido'.

Aproximação dos três pontos tendo em conta a forma geométrica descrita pela união dos mesmos

Esta função aplica um algoritmo iterativo de aproximações múltiplas. Partindo da forma geométricas de um dado modelo, é aplicado um conjunto de rotações e translações sucessivas, de forma a minimizar a distância entre os pontos dos dois modelos.
Apesar de não ser um algoritmo tão eficiente como o anterior, tem a vantagem de não impor ao utilizador a marcação dos pontos na mesma ordem – para os dois modelos. Ou seja, ganha-se em automatismo e perde-se em eficiência.
Este método impõe uma condição: os tamanhos dos lados do triângulo definido pelos três pontos devem ser todos diferentes, de modo a que o algoritmo devolva a forma mais correcta (e única) de aproximar os dois triângulos, a partir da matriz de transformação linear de corpo rígido.

Com o objectivo de auxiliar o utilizador a verificar, visualmente, se a correspondência dos pontos entre os dois modelos é correcta, conforme no ponto acima, a cor de cada ponto está associada à ordem pela qual estes são inseridos, embora a ordem dos mesmos já não interfira com o resultado final

A fim de melhorar a facilidade de leitura dos dois modelos, a aplicação disponibiliza dois géneros de projecção de modelos, analisados nos pontos seguintes.

Projecção dos pontos do modelo

Esta projecção tem a vantagem de ser computacionalmente bastante rápida, permitindo interacções rápidas sobre um dado modelo, infelizmente, um modelo visto neste formato não é fácil de procurar e encontrar pontos comuns, quando muito complexos.
Na imagem seguinte é possível observar um exemplo desta forma de projecção, já com três pontos marcados:

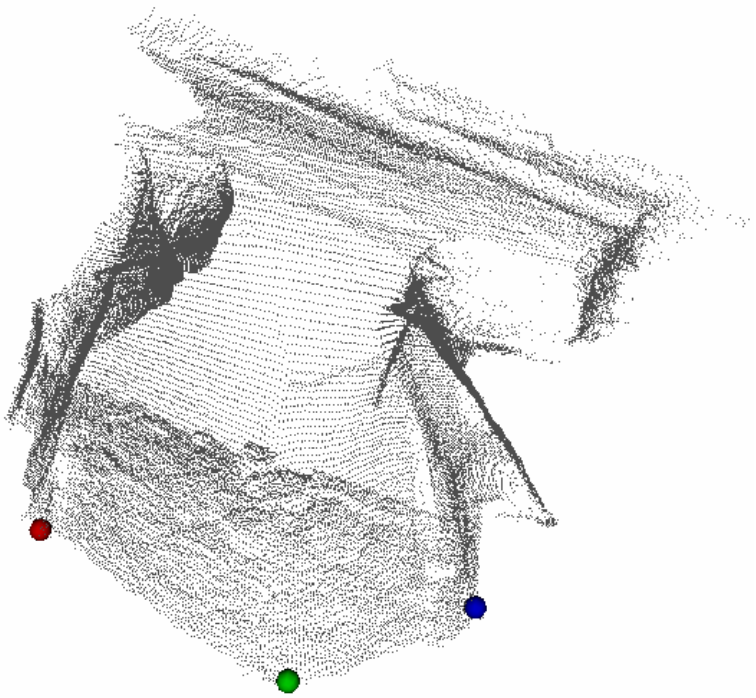


Figura VI.10 - Marcação de 3 pontos num modelo de pontos

Projecção de triângulada do modelo

Este género de projecção, apesar de ser computacionalmente mais exigente, já permite a visualização de vértices, esquinas e superfícies de forma bastante mais fácil-
Contudo, em máquinas menos computacionalmente eficientes, como este género de projecção exige bastante mais poder de processamente, a interacção do utilizador com o modelo torna-se um pouco lenta.

Na imagem seguinte é possível ver o modelo da imagem anterior, com a triangulação aplicada e três pontos marcados.

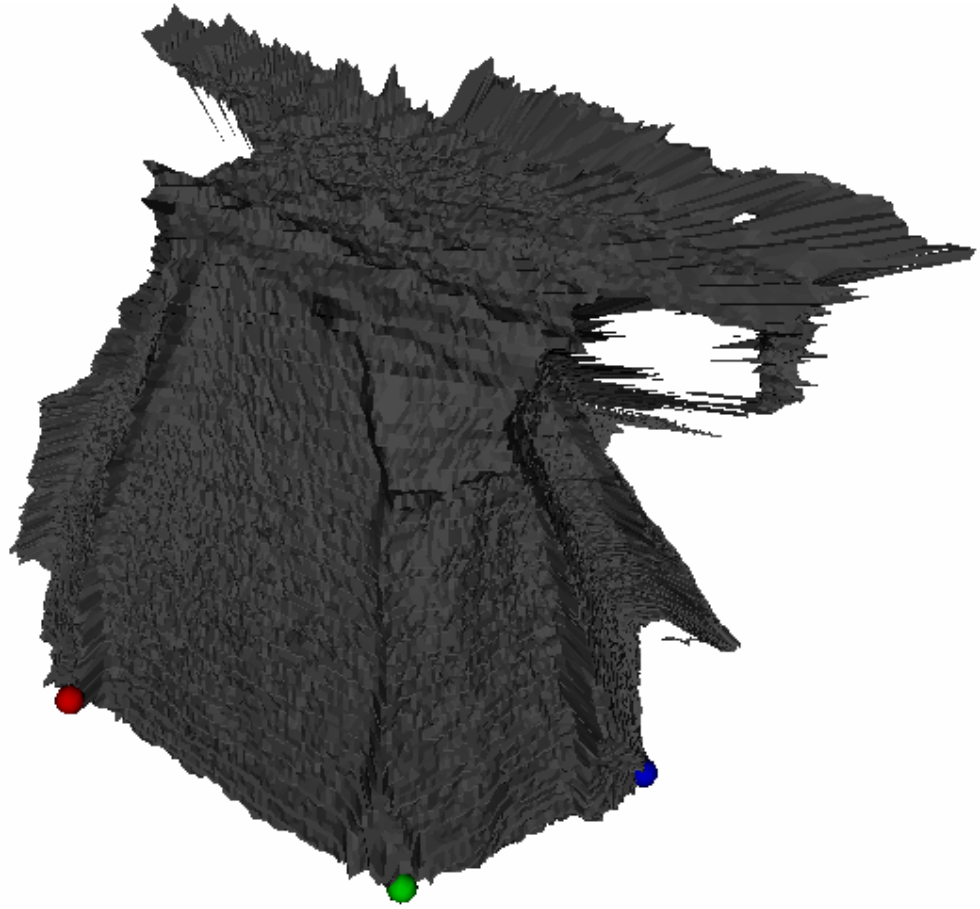


Figura VI.11 - Marcação de 3 pontos num modelo de triângulos

Conforme se pode observar, a presença da triangulação, além de facilitar a percepção do modelo, também permite que o utilizador evite marcação de pontos em áreas de fraca similaridade entre modelos.

VI.7.2. Cálculo da matriz de transformação a partir do algoritmo ICP

Este método deve ser aplicado apenas quando as nuvens dos modelos de pontos já se encontram relativamente próximas da sua posição final, ou seja, após a aplicação de um dos dois métodos de aproximação manual vistos acima.
Denominado pela sigla ICP (Interactive Closest Point) ou método por aproximações sucessivas, trata-se de um método em que é fornecido um conjunto de pontos de origem e um conjunto de pontos de destino.
Este método opera sobre o valor do somatório das distâncias de um dado conjunto de pontos do modelo de origem ao ponto mais próximo do de um conjunto de pontos do modelo de destino. Vai efectuando um conjunto de translações e rotações, de forma a reduzir este valor a um mínimo, de forma a aproximar ao máximo os pontos do modelo de origem para o de destino.
A fim deste método ser computacionalmente expedito, é necessário utilizar um número de pontos limitado, mas suficiente para aproximar os dois modelos de forma correcta.
A forma de seleccionar estes mesmos pontos, baseia-se nas seguintes regras:

- Partindo do pressuposto de que os dois modelos já se encontram relativamente próximos um do outro, apenas se escolhe pontos de uma dado Modelo cuja distância, relativamente a pelo menos um dos pontos do outro modelo, seja inferior a uma dada distância mínima (distância de *treshold*)

O digrama seguinte pretende expor o procedimento descrito:

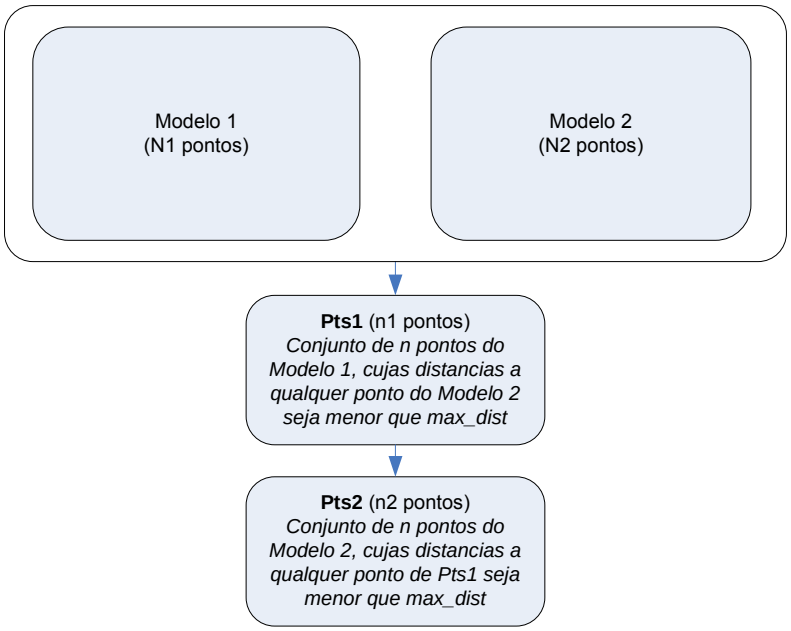


Figura VI.12 - Selecção de dois subconjuntos de pontos de aproximação

- Escolher os pontos da maneira mais aleatória possível, de forma a evitar seleccionar apenas algumas zonas, deixando outras zonas sem pontos de aproximação:

  A forma encontrada para procurar pontos de forma aleatória, está descrita no esquema abaixo:

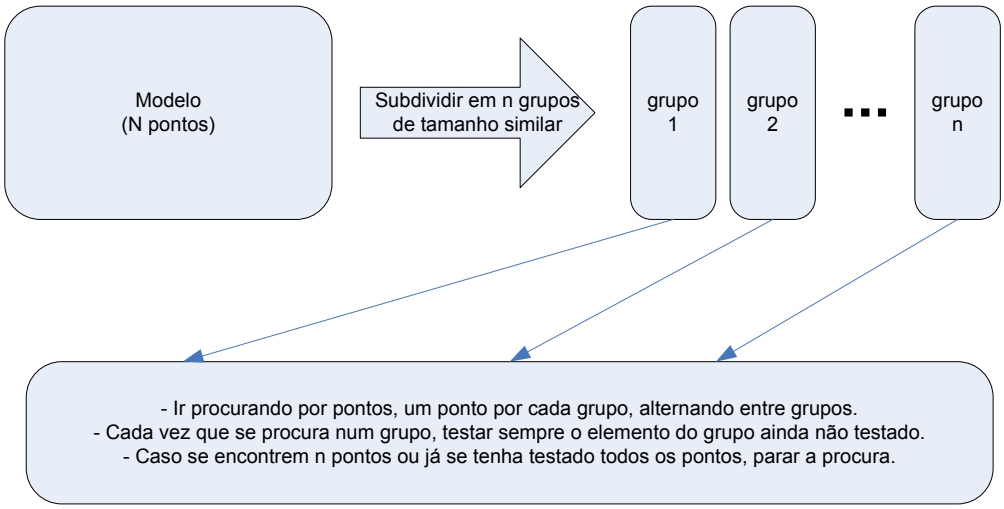


Figura VI.13 - Selecção aleatória dos pontos a fim de todas as zonas terem igual peso

Após ter-se escolhido dois grupos de pontos (*pts1* e *pts2*) é necessário proceder ao cálculo da matriz de transformação capaz de aproximar o conjunto de pontos *pts2* do conjunto de pontos *pt1*, apenas com operações de translação e rotação (operações de escala são desactivadas devido ao facto de se considerar que ambas ambos os conjuntos de pontos se encontram na mesma escala.

Um algoritmo capaz de obter esta matriz, de forma relativamente rápida e eficaz, trata-se do algoritmo de aproximações sucessivas - *Iterative Closest Point (ICP)*[1]. Este algoritmo encontra-se implementado no pacote Visualization ToolKit (VTK) utilizado na criação da aplicação.

De uma forma simplificada, o digrama abaixo expõe o modo de funcionamento deste algoritmo:

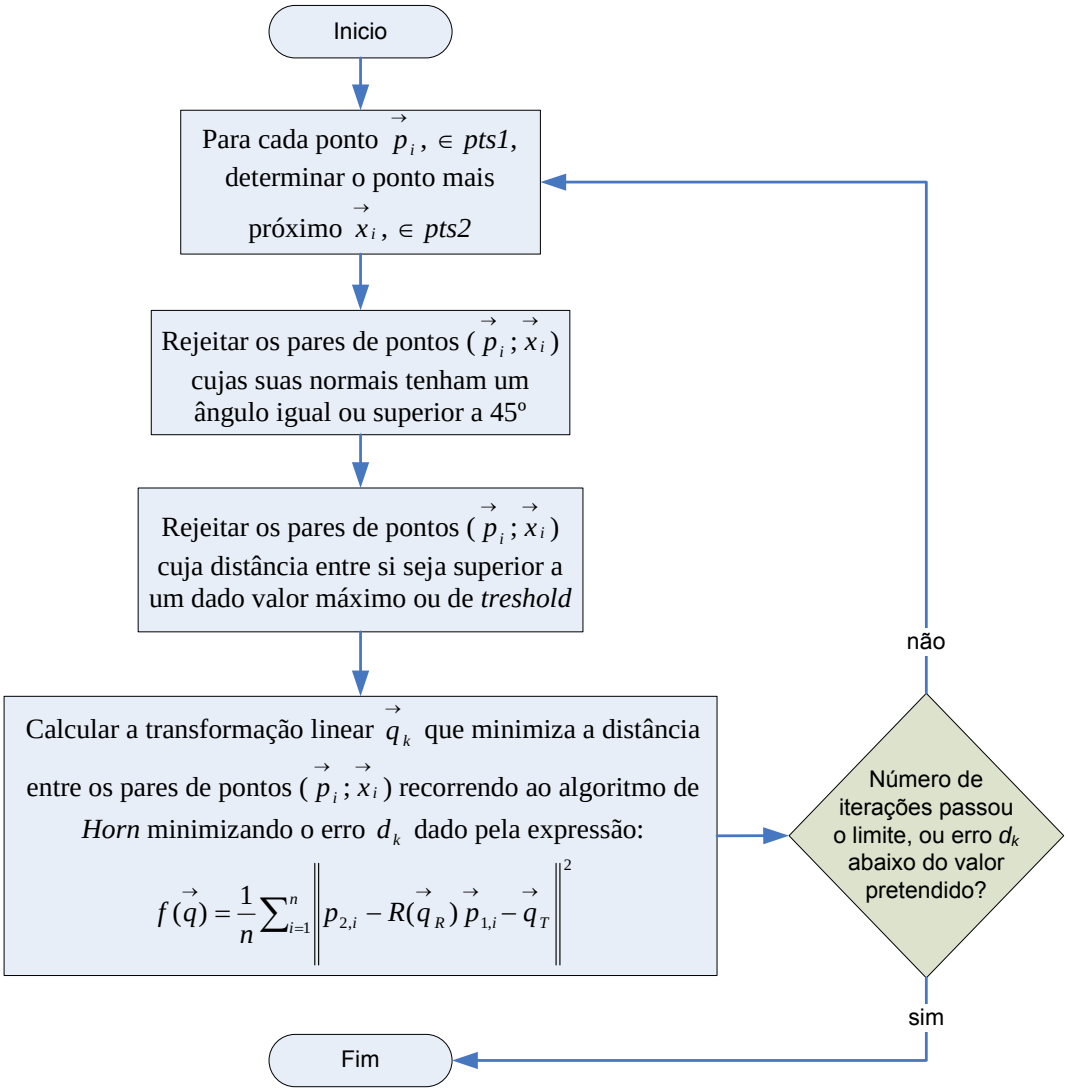


Figura VI.14 - Diagrama simplificado do algoritmo ICP

Este algoritmo, embora consiga aproximar os modelos de forma eficiente, nada garante que só haja uma forma de encaixe dos dois modelos, pelo que pede ao utilizador a validação da operação. Como já foi mencionado, uma primeira aproximação dos dois modelos feita manualmente (via intervenção do utilizador) é altamente aconselhada.

VI.8. Junção de modelos.

De facto, para conseguir a representação tridimensional de um espaço, raramente um único *scan* ou aquisição de pontos consegue guardar todos os dados relativamente a esse mesmo espaço.

A figura seguinte pretende mostrar como ocorre o fenómeno da existência de zonas não capturadas:

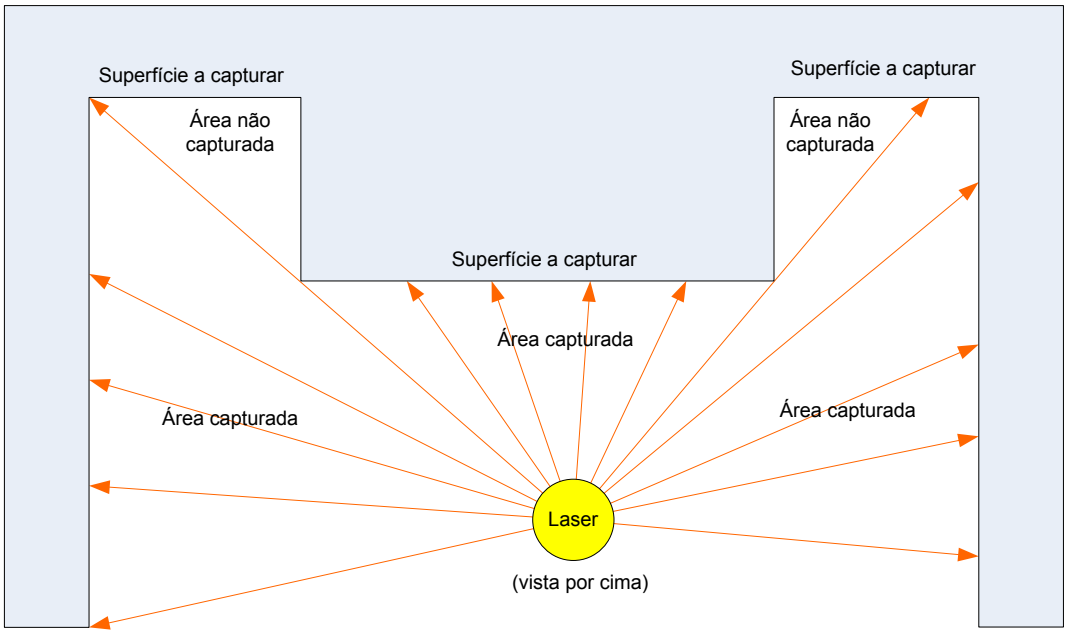


Figura VI.15 - Zonas capturadas vs. Zonas não capturadas

Uma forma de conseguir adicionar mais detalhes a um modelo, é efectuar a aquisição de dados de um dado espaço de outra zona, de forma a poder adicionar esses mesmos dados aos dados originais.

A imagem abaixo exemplifica como a opção de utilizar duas aquisições de um dado espaço em vez de uma só, pode auxiliar a obter uma aquisição completa do espaço real:

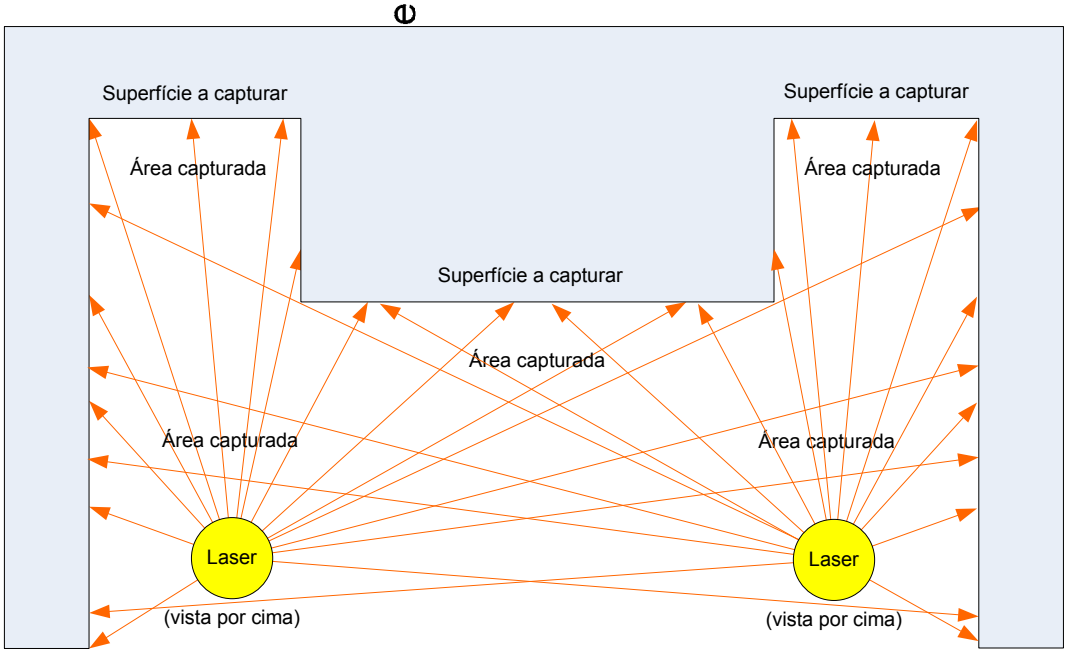


Figura VI.16 - Escolha correcta do *scanner* laser a fim de evitar zonas sem captura

Nota: As zonas a verde-claro são zonas em que a aquisição apenas adquiridas numa das suas aquisições de pontos. Zonas a verde-escuro são zonas adquiridas pelas duas zonas de aquisição – Estas zonas, também conhecidas por zonas de junção (ou encaixe) entre nuvens de pontos, devem ser as mais extensas possíveis, de forma a facilitar o processo de junção de modelos.

No entanto o processo de junção de duas nuvens de pontos não é simples, pois deparam-se vários problemas:

1. É necessário indicar à aplicação qual a posição do modelo a adicionar, relativamente ao modelo base (ou inicial), para que o modelo a adicionar possa ser rodado e transportado para a nova posição, relativamente ao modelo base.

2. Quando duas superfícies dos dois modelos se combinam (em duplicado) numa dada localização, torna-se importante eliminar uma dessas superfícies (ou combinar as duas numa só) de forma a eliminar informação redundante, de modo a tornar a simplificar ao máximo o modelo virtual, sem perda ou corrupção de dados do mesmo. Desta forma o modelo torna-se computacionalmente eficiente e simples de analisar e operar.

VI.8.1. Remoção de pontos não utilizados durante a triangulação / junção de modelos.

A fim de dar o máximo de liberdade ao utilizador, esta rotina pode ser invocada em qualquer fase evolutiva de uma dado modelo, pois analisa o mesmo e age de forma apropriada ao seu estado. No entanto é desaconselhável invocar a mesma antes de triangular um dado modelo, pois irá *destruir* a organização matricial dos pontos, impedindo a triangulação dos mesmos pelos algoritmos presentes nesta aplicação.

Esta rotina deve ser apenas invocada após dois tipos de operações:

1. **Remoção de triângulos com área nula:**

Conforme foi visto no ponto acima, a remoção de triângulos com área nula deixa um conjunto de pontos sem qualquer ligação a um triângulo, que devem e são retirados nesta rotina.

2. **Remoção de pontos que ocupem a mesma posição no espaço, após uma junção de dois modelos:**

Logo após ter sido feita uma união, ou adição, de um modelo de pontos/triângulos (modelo a adicionar) a um segundo modelo de pontos (modelo base), os pontos do modelo a adicionar que se encontram muito próximos de um dos pontos do modelo base, sofrem um deslocamento para a posição exacta do ponto do modelo base.
Desta forma, esta rotina irá remover um de dois pontos que ocupem a mesma posição no espaço, um dos pontos pode ser removido, fazendo com que todos os triângulos com um dos vértices no ponto removido passam a utilizar o outro ponto, que ocupa a mesma posição.

Na figura abaixo é possível observar em que altura, no processo de junção entre dois modelos, a rotina de **Remoção de pontos não utilizados durante a triangulação / junção de modelos** é invocada:

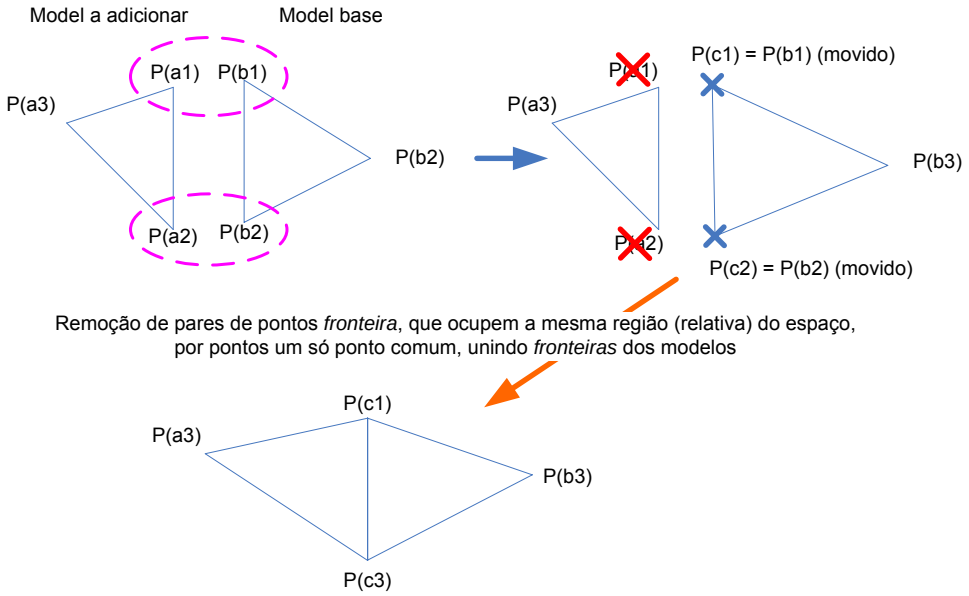


Figura VI.17 Junção de triângulos

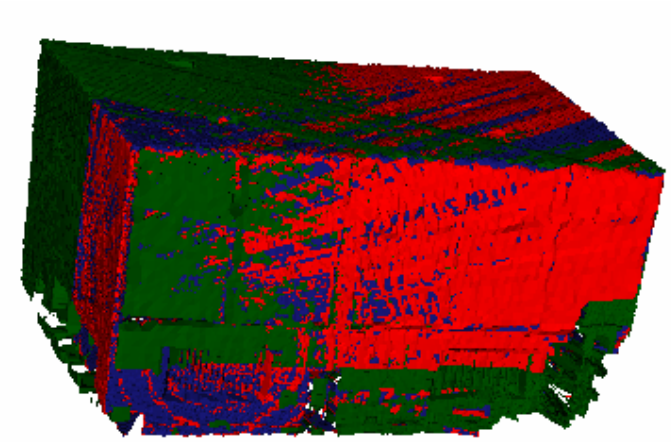
Descrição do processo:

1. Identificação dos pares de pontos do modelo a adicionar que estejam demasiado próximos ao modelo base (até uma distância máxima definida por distância de *treshold*).
2. Mover a localização dos pontos do modelo base para a posição intermédia entre o ponto do modelo base e o ponto do modelo a adicionar acima identificado, passando a identificar-se por pontos de encaixe.
3. Remoção dos pontos do modelo a adicionar acima identificados.
4. Substituição das referências, nos triângulos do modelo a adicionar, referentes aos pontos eliminados pelas referências aos pontos de encaixe.
5. Repete-se o ponto 3 até deixar de se encontrar pontos que tenham pontos vizinhos com uma distância igual ou menor a um dado valor mínimo (de *treshold*).

VI.8.2. Identificação e substituição de superfícies comuns aos dois modelos.

No processo de junção de modelos, devido às múltiplas superfícies comuns aos mesmos, quando se juntam dois modelos, estas superfícies comuns podem ser identificadas e substituídas por uma só superfície, de forma a manter a complexidade do modelo a um nível aceitável.

Na imagem abaixo, pode ser visto o que a acontece a uma dada superfície, quando esta é comum aos mesmos, sendo, a fim de facilitar a leitura da imagem, atribuído a cada um dos modelos uma cor diferente:



Models details

Model	Points	Triangles	Merged	Can angula	hange natri	Original filename
1	92408	179727	no	no	no	11Apr2006_11-26-01vdu.txt
2	92045	179033	no	no	no	11Apr2006_11-26-41vdu.txt
3	90604	176420	no	no	no	11Apr2006_11-30-38vdu.txt

Figura VI.18 Modelos triangulados sobrepostos

De forma a melhor avaliar o algoritmo de identificação e eliminação de superfícies duplicadas, foram utilizadas, no exemplo acima, nuvens de pontos que apanham quase as mesmas secções do espaço.
De facto, pode-se ver que, devido aos erros (relativos) de aquisição dos pontos do modelo e também devido aos erros de rotação e translação de um modelo, em relação ao outro, ambas as superfícies nunca ficam exactamente uma sobre a outra, estando, contudo, a uma distância relativamente próxima uma da outra. Estas superfícies, bastantes próximas, podem ser identificadas substituídas por uma só comum às duas, de forma a optimizar a junção dos dois modelos.
O processo de junção de superfícies é efectuado da forma seguinte:

1. Identificação de triângulos do modelo a adicionar, cujos seus vértices se encontrem próximos de pontos do modelo base.
2. Aproximação dos pontos do modelo base que se encontrem próximos dos pontos dos triângulos acima identificados, movendo estes pontos para uma distância intermédia entre a sua posição e ao ponto do modelo a adicionar que se encontra mais próximo.
3. Remoção dos triângulos do modelo a adicionar, acima mencionado.

Pode-se ver na imagem abaixo uma exemplificação deste processo, em que ambas as superfícies se encontram sobrepostas, mas cada uma com a sua própria triangulação:

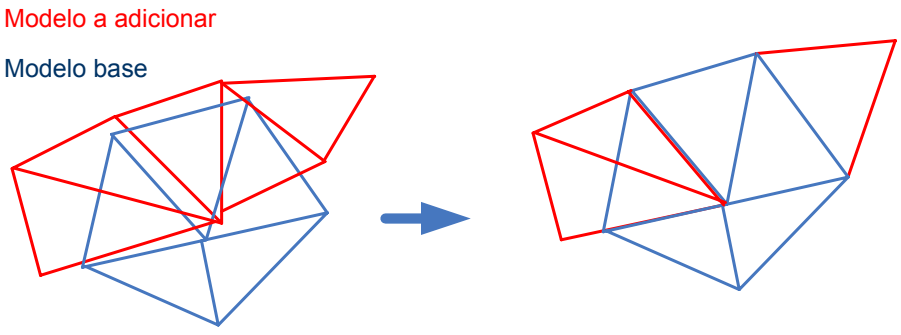
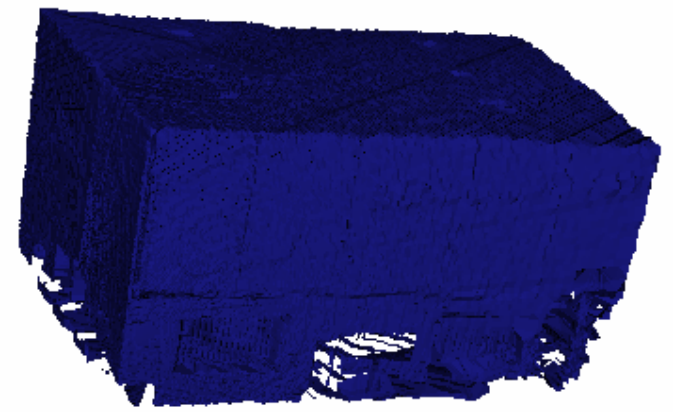


Figura VI.19 - Junção de triângulos de dois modelos

Nota: Conforme se pode observar pela figura acima, este processo elimina todos os triângulos comuns a uma mesma superfície. No entanto, quando estes triângulos se cruzam, numa secção fronteira, podem ocorrer sobreposições parciais dos mesmos (raro, mas possível).
Aconselha-se uma edição posterior do modelo, numa aplicação de edição 3D, a fim de auxiliar a corrigir estas (e outras) ocorrências.

Após a junção de ambos os modelos, obtém o modelo final:



Models details

Model	Points	Triangles	Merged	Can triangulate	Changed matrix	Original filename
1	105575	207002	yes	no	no	11Apr2006...

Figura VI.20 - Modelos triangulados, unidos num modelo

Conforme se pode observar, além da verificação visual a mostrar a correcta junção dos três modelos num só, também se pode verificar que o processo de identificação das superfícies comuns é eficiente, pois o número de triângulos (e pontos) do modelo final ser bastante menor que o da somados dois modelos a unir, caso não se tivesse recorrido a este algoritmo (no entanto este número nunca pode ser menor que qualquer um dos modelos a unir).

VI.9. Redução/simplificação do número de triângulos do modelo

De forma a simplificar o modelo triangulado, é possível substituir um dado conjunto de triângulos por um conjunto de menor número, quando estes definem uma dada superfície fechada, com as faces destes triângulos e terem uma orientação similar (no espaço), definidas por um dado plano.
Os algoritmos de simplificação de modelos triangulados, na sua grande maioria exploram o facto de uma dada superfície relativamente plana, sem falhas (*buracos*) e definida por um dado conjunto de triângulos poder ser representada por um número inferior de triângulos, de forma reduzir os recursos que a mesma requer.

VI.9.1. Benefícios e perdas da simplificação do modelo triangulado

O processo de simplificação de modelos triangulares, implica um conjunto de benefícios e perdas, quando aplicado a um dado modelo triangulado.

Relativamente às suas vantagens, destacam-se:

- Redução dos recursos de processamento em operações sobre o mesmo (mais rápido).
- Redução do consumo de memória e espaço em disco para operação e salvaguarda deste (mais pequeno).
- Aumento da velocidade de projecção (aumento das *frames por segundo – fps)*

Contudo, a simplificação do modelo triangulado, também apresenta as desvantagens seguintes:

- Redução do número de pontos necessários para aproximação de modelos (operações de aproximação de modelos, recorrendo ao algoritmo ICP tornam-se menos eficientes).
- Redução do número de pontos de união entre dois modelos, piorando o resultado de algoritmos de registo e junção de modelos (o reconhecimento de superfícies comuns torna-se menos eficiente, dificultando a junção das mesmas numa só).
- Perda de precisão da superfície definida pelos triângulos, quando se utiliza um valor de oscilação das superfícies definidas pelos triângulos demasiado elevada.

De facto, uma simplificação de um dado modelo de triângulos (ou triangulado), apenas deve ser efectuada quando já se tem um único modelo triangulado final, ou seja: todos os modelos, adquiridos a partir de cada uma das nuvens de pontos, já foram triangulados, movidos (translação e/ou rotação), as secções fronteiras dos vários modelos foram unidas e as superfícies comuns foram substituídas por uma só, ou seja, apenas deve ser aplicado na última fase do processo de criação do modelo virtual (aplicação de textura e correcção manual de incongruências podem ser aplicadas em quase qualquer fase deste processo).

VI.9.2. Descrição do algoritmo de simplificação do modelo triangulado

Basicamente, o algoritmo utilizado procura superfícies fechadas, definida por conjuntos entre 3 (três) a 12 (doze) triângulos unidos e com direcções similares, substituindo-os por um conjuntos com menos 2 triângulos, até deixar de encontrar mais conjuntos de triângulos com estas propriedades.

A forma de procurar um conjunto de triângulos acima descritos é descrita pelo diagrama seguinte:

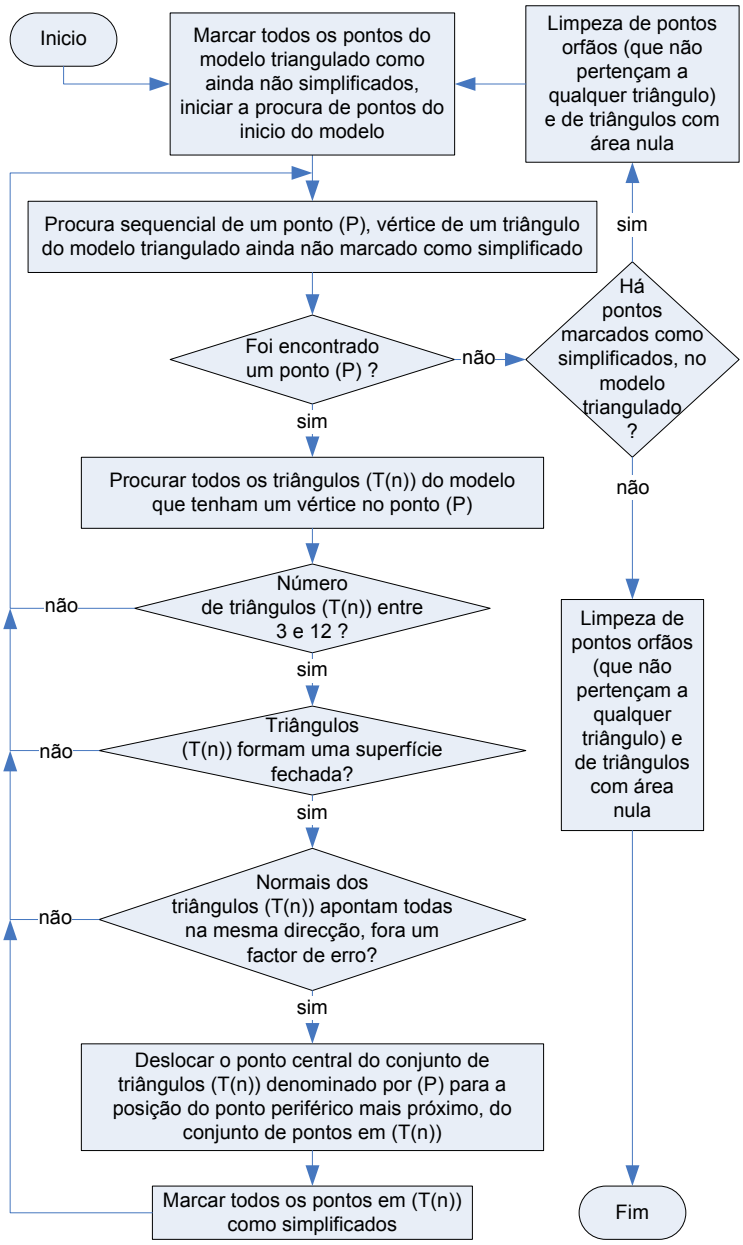


Figura VI.21 - Algoritmo de simplificação de triangulação

Nota: A procura e marcação dos pontos recorre a técnicas de processamento em bloco (dos pontos), de forma a optimizar a velocidade do algoritmo. O tamanho destes blocos é um compromisso entre velocidade e o espaço do bloco em memória.

De forma a melhor compreender como opera o algoritmo de simplificação triangular, a imagem seguinte exemplifica o que acontece a um grupo de triângulos (evidenciados a linhas mais grossas), quando são simplificados:

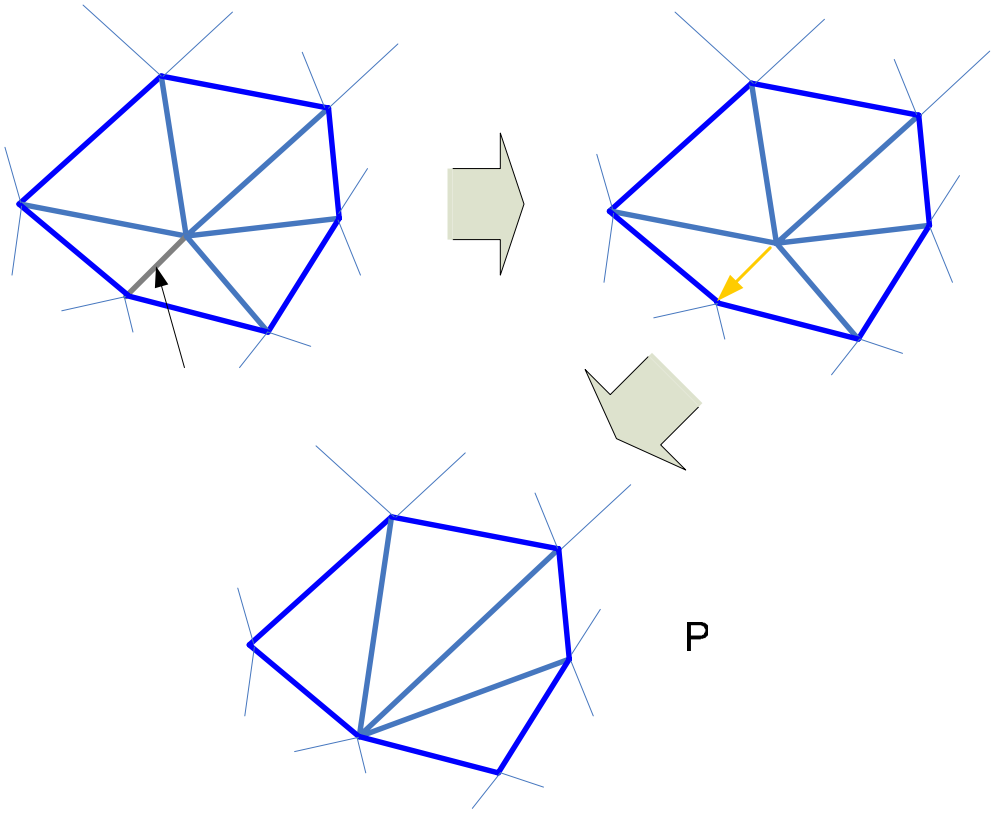


Figura VI.22 - Exemplo de simplificação triangular

Nota: Na imagem acima, tanto a procura do ponto P, como a procura dos triângulos à volta de P, a verificação dos triângulos à volta de P terem a mesma normal (com um factor de variação baixo) e a verificação de que estes triângulos formam uma superfície fechada, não é exemplificado.

Segmento mais curto a partir de P

P

VII. Aplicação principal

VII.1. Introdução

Com o objectivo de conseguir utilizar um dado conjunto de nuvens de pontos adquiridos a partir de um *scanner* laser de distância, juntar e triangular esses mesmos pontos de forma a conseguir um modelo virtual o mais próximo da realidade, procedeu-se à criação de uma aplicação na linguagem C++ totalmente modular, utilizando apenas bibliotecas abertas (QT e VTK), dentro do conceito de desenvolvimento de aplicações académicas.
Embora tenha sido desenvolvido num ambiente Windows, a utilização do VTK com o QT facilita a sua compilação e desenvolvimento em outros sistemas operativos tais como Linux ou OSX.

VII.2. Descrição geral

A aplicação desenvolvida neste projecto foi escrita de modo a que qualquer operação requerida pelo utilizador esteja directamente acessível, estando tanto os seus parâmetros de funcionamento, como o desenrolar da mesma, directamente acessíveis na janela principal da aplicação.
Todo o processamento e criação de um ambiente virtual, permitidos nesta aplicação, partem de uma, ou mais, nuvens de pontos adquiridas a partir de um *scanner laser* de profundidade.

A partir de uma nuvem de pontos, pode-se efectuar uma das seguintes opções:

1. Limpar os pontos extra:
    a. Não aconselhado pois elimina a sua estrutura matricial, necessária ao processo de triangulação.
2. Junção a outro modelo de pontos ou triangulado:
    a. Antes de se juntar dois modelos, os dois devem ser triangulados, pois os pontos do modelo não triangulado irão perder a sua estrutura e não poderão mais ser triangulados.
    b. Se o modelo base não tiver sido triangulado, o modelo final também não o será.
    c. Se o modelo a juntar não tiver sido triangulado e o modelo base tiver, irão perder-se os pontos do modelo a adicionar durante uma limpeza de pontos, pois a limpeza de pontos, quando detecta um modelo triangulado utiliza apenas os pontos desse modelo triangulado.
    d. Nunca se deve tentar juntar automaticamente dois modelos antes de um deles ser aproximado ao outro, primeiro manualmente (a partir dos 3 pontos) e depois de forma automática (a partir do algoritmo ICP).
3. Triangulação do modelo:
    a. Aconselha-se o modelo de triangulação com detecção de contornos.
    b. Apenas se devem juntar modelos já triangulados.

Depois de se ter aproximado um modelo a outro, deve-se ter em atenção:

- Verificar se está ou não activada a opção de salvaguarda automática da matriz de transformação linear de uma dada nuvem de pontos.
- Em caso de uma aproximação bem sucedida e a salvaguarda automática da matriz de transformação de um modelo não estiver activada, pode-se pedir à aplicação para guardá-la, para não ter de se voltar a efectuar uma salvaguarda da mesma.

É também aconselhável ter em atenção de que o modelo activo e a janela activa são sempre os destinatários dos comandos invocados.

A fim de A aplicação encontra-se dividida em vários objectos, ou módulos, que interagem entre si. A imagem seguinte mostra quais os objectos/módulos principais, a sua dependência e quando existe interacção entre os mesmos:

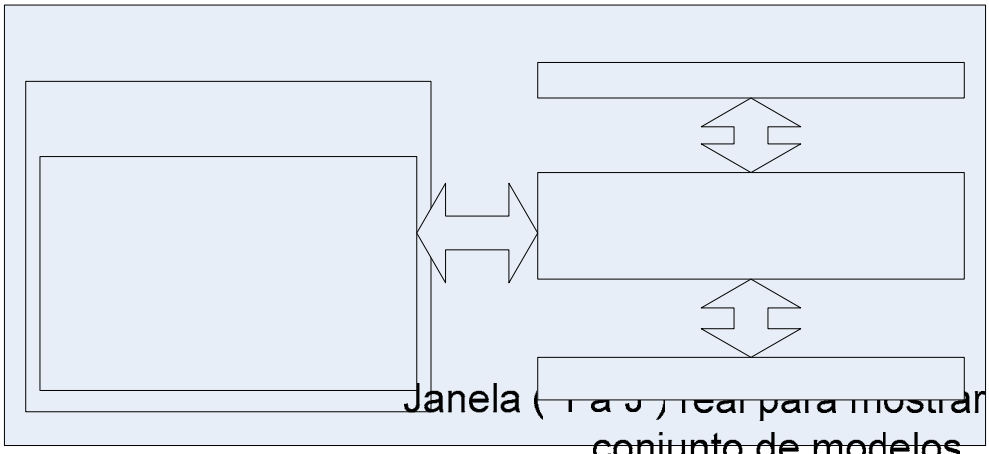


Figura VII.1 - Comunicação entre módulos

Os dados de uma nuvem de pontos são guardados num objecto denominado por modelo. Este modelo é capaz de efectuar um número elevado de operações:

- Manter em memória um modelo de pontos.
- Manter em memória um modelo de triângulos (triangulado).
- Abrir um ficheiro do tipo nuvem de pontos, transformando-o num modelo de pontos, em memória (na disposição matricial, já descrita).
- Duas rotinas de triangulação de um modelo de nuvem de pontos no formato matricial.
- Abrir um ficheiro do tipo nuvem de pontos, transformando-o num modelo de pontos, aplicar uma transformada linear de corpo rígido e adicioná-lo ao modelo de pontos já existente (operação de adição).
- Durante operações de adição, possibilidade de automaticamente triangular o modelo a adicionar, antes de o adicionar ao modelo já existente, apenas quando o modelo existente tem um modelo de triangulado.
- Efectuar operações com matrizes do tipo transformada linear de corpo rígido a qualquer modelo de pontos e/ou triângulos existente.
- Restaurar e guardar a transformada linear de corpo rígido de todas as transformadas lineares de corpo rígido aplicadas num ficheiro associada a uma nuvem de pontos, enquanto o modelo apenas corresponder a uma só nuvem de pontos de forma automática.
- Limpeza de pontos desnecessários/erróneos de modelos de pontos e/ou triangulados.
- Limpeza de triângulos inválidos (área nula, demasiado grandes) de modelos já triangulados.
- Aproximação de dois modelos, de forma automática, a partir do algoritmo ICP.
- Junção de dois modelos de pontos, eliminando pontos demasiado próximos ou sobrepostos.
- Junção de dois modelos triangulados, unindo triângulos dos dois modelos, de forma a criar superfícies o contínuas / unidas.
- Redução do número de triângulos e pontos de um modelo triangulado.
- Relatório automático de todas as operações em curso por eventos QT.

Nota: A fim de facilitar o desenvolvimento deste objecto e também devido à elevada complexidade do mesmo, foi criada uma classe designada por ‘Funções 3D” com algumas das funções utilizadas pelo objecto “Modelo”. Funções estas que apenas necessitam de um modelo VTK de entrada, outro de saída e por vezes um parâmetro extra.
Funções que requeiram uma maior interacção com o objecto “Modelo”, estão escritas no mesmo.
Os modelos de pontos e de triângulos, apesar do segundo ser obtido a partir do primeiro, depois de criados, ficam independentes entre si.

A cada modelo é atribuído um objecto, do tipo janela de interacção (e projecção) do modelo, capaz de efectuar operações sobre esse mesmo modelo:

- Projectar um modelo de pontos ou de triângulos (triangulado), numa janela VTK nova (criada neste objecto), ou numa já existente (criada num objecto similar a este), podendo o utilizador actuar na visualização de múltiplas formas: ampliação, rotação, translação, ver em modo *wireframe*, entre outras.
- Permitir ao utilizador marcar até três pontos, em três cores diferentes, guardando as coordenadas destes pontos.
- Calcular a matriz de transformação rígida dos três pontos desta janela de interacção e uma janela de interacção fornecida.
- Aplicar matrizes de transformação linear a um dado modelo projectado, sem no entanto alterar este modelo.
- Relatório automático de todas as operações em curso por eventos QT.

Por sua vez, a cada dado conjunto janelas de interacção é atribuído um gestor de janela final. Este gestor de janela final permite projectar uma ou mais janelas de interacção, com respectivos modelos e tipos de projecção, numa dada janela de projecção final.
Este objecto tem as seguintes características:

- Contem até um máximo número de janelas de interacção.
- Definir a cor com que cada um dos modelos de cada uma das janelas de interacção é projectado.
- Permite configurar características da janela:
  - Título.
  - Tamanho.
  - Posição.
- Permite abrir / fechar uma qualquer janela.
- Adicionar / remover janelas de interacção de forma instantânea.
- Cada janela criada por este objecto é real e independente das restantes janelas da aplicação.
- Relatório automático de todas as operações em curso por eventos QT.

Todos estes objectos são criados por um objecto denominado de janela principal, que tem como principais objectivos:

- Gerir todos os objectos acima descritos.
- Interagir directamente com o utilizador como interface principal da aplicação.
- Criação do menu de comandos da aplicação.
- Definição da janela real e modelo seleccionados.
- Criação das tabelas de características da janela real e modelo seleccionados, visualizados na janela principal da aplicação.
- Apresentação das mensagens de operação da aplicação (mensagens informativas relativamente ao processamento em curso).
- Apresentação das várias opções definidas para todo o género de comandos disponibilizados.
- Gestão de objectos secundários da aplicação:
  - Acesso a ficheiros do sistema (leitura/escrita).
  - Criação de eventos Qt.
  - Execução de comandos em *threads* secundárias de forma a não bloquear a janela principal da aplicação (em desenvolvimento / não activado).

VII.3. Módulos principais

Toda a aplicação foi criada tendo sempre em mente a modularização e uma facilidade elevada de continuação de desenvolvimento da mesma.

Como uma primeira abordagem à forma como a mesma se encontra organizada, a imagem seguinte mostra os módulos principais da aplicação e como estes são criados e interagem entre si:

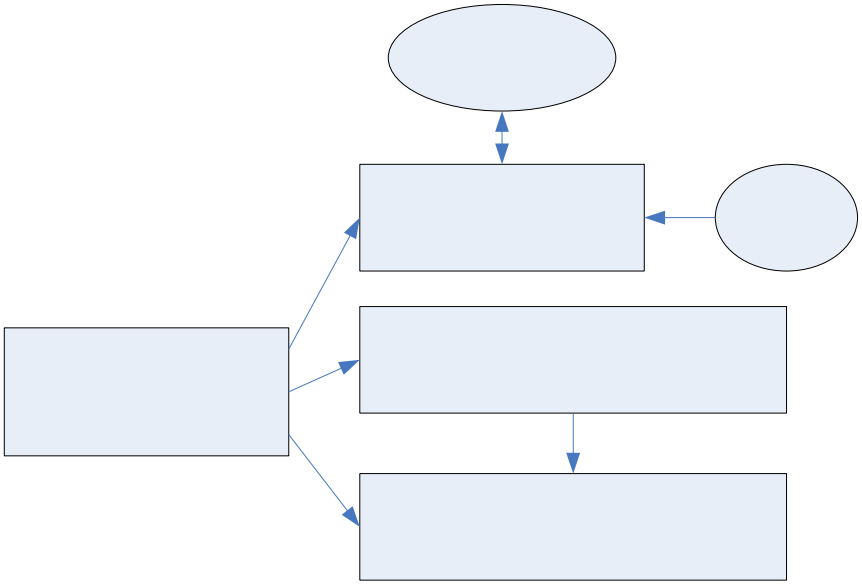


Figura VII.2 - Criação dos módulos principais

VII.3.1. Objecto: Modelo (`modelManager`)

O objecto `modelManager` é composto por múltiplos objectos e características (todos os objectos/variáveis, excepto a aplicação principal, podem ter um valor nulo, a uma dada altura, nos estados de execução da aplicação):

Variáveis (mais significativas)	Aplicação principal (1)

- Nome do ficheiro da nuvem de pontos inicial (`char *m_FileName`)
  Este ponteiro guarda o nome do ficheiro da nuvem atribuída ao objecto.
  Nuvens adicionadas ao modelo não interferem com este nome.

- Altura / largura da matriz de pontos inicial do modelo (`int m_xdim, m_ydim`)
  Estas variáveis têm o valor da altura e largura da matriz de pontos obtidos durante a abertura do ficheiro de nuvem de pontos.
  Qualquer operação de triangulação, limpeza de pontos ou adição de mais modelos torna este valor obsoleto.
  Estes valores são utilizados durante a triangulação do modelo.

- Imagem em tons de cinza obtida a partir da profundidade de cada ponto, a partir do formato matricial, da nuvem de pontos (`vtkImageData *m_Image`)
  Apesar de actualmente não ter qualquer aplicação, esta imagem poderá ser aplicada como um textura ao modelo triangulado.
  A fim de poupar recursos, esta imagem é apenas criada quando o utilizador assim o especifica.

- Matrizes com a posição das zonas dos vértices horizontais, dos verticais ou da matriz de união dos dois (`unsigned char *modelHEdges, *modelVEdges`)
  É apenas criada durante a triangulação com detecção de vértices, quando o utilizador assim o especifica.
  Também é possível especificar a criação de uma imagem em tons de cinza com a informação desta variável, a duas dimensões (formato matricial) a fim de analisar o processo de detecção de vértices. Esta imagem tem o nome (extensão diferente) do ficheiro da nuvem de pontos original.

- Modelo de pontos 3D VTK (`vtkPolyData *m_Model`)
  A primeira operação, após a leitura de uma nuvem de pontos, é transformar esse conjunto de pontos num modelo de pontos a três dimensões que possa ser directamente projectado numa janela de visualização VTK.
  Pode ser lido ou escrito directamente para um modelo 3D (formato WRL).
  A existência deste modelo, após a triangulação já não é obrigatória, pois a aplicação, a partir do momento que efectua a triangulação e respectivos pontos, passa a ignorar este modelo.

- Profundidade de cada ponto, em relação ao emissor laser (`float *rangeVal`)
  Logo após a leitura da nuvem de pontos, a profundidade dos pontos é guardada num array de números de vírgula flutuante, endereçado a partir da expressão:
  $índice = y \times x\_size + x$, em que *x_size* é o número de pontos por linha horizontal.
  Mais tarde este *array* é utilizado durante o processo de triangulação (aumenta o desempenho do algoritmo), para de seguida libertar o espaço ocupado pelo mesmo, em memória.

- Modelo de triângulos 3D VTK (`vtkPolyData *TriangulatedModel`)
  O modelo triângulos, ou triangulado, de um dado conjunto de pontos pode ser obtido após uma operação de triangulação sobre um conjunto de pontos no formato matricial, ou após carregamento directo do mesmo a partir de uma ficheiro.
  Operações de junção de modelos operam de forma mais eficiente se ambos os modelos tiverem o formato triangulado, pois haverá uma operação extra de junção dos limites de ambos os triângulos dos modelos e uma remoção de triângulos duplicados. Usualmente ambos os modelos a unir partilham superfícies em comum, cuja necessidade da existência de duas triangulações da mesma (em ambos os modelos) se torna desnecessária, pelo que apenas a superfície do modelo base é utilizada.
  Note-se que não é possível triangular um modelo de pontos já unido a outro modelo de pontos, pois o processo de união destrói completamente a organização matricial dos pontos. Logo, a operação de triangulação deve ser efectuada antes da junção dos modelos.
  Pode ser lido ou escrito directamente para um modelo 3D (formato WML).

- Matrizes de transformação linear de corpo rígido do modelo (`vtkTransform *transforms,*icpTransform`)
  De facto, todas as operações de transformação linear de corpo rígido (translação e rotação) aplicadas a um dado modelo são guardadas numa matriz de tamanho 4×4, na posição de memória referida pela variável `transforms`. Esta variável pode ser automaticamente guardada e restaurada (opcional) num ficheiro com o nome da nuvem de pontos original e extensão ".matrix".
  Já a variável `icpTransform` refere-se às matriz de transformações de translação e rotação necessárias a aplicar a um dado modelo a fim deste se aproximar a outro. Esta transformação poderá ser (ou não) aplicada ao modelo (a aproximar) após uma apreciação positiva da operação de aproximação de dois modelos, recorrendo ao algoritmo ICP.

- *Flags* (verdadeiro / falso):
  - Foi ou não aplicada uma transformação linear de corpo rígido desde que a matriz de transformação linear foi carregada e/ou guardada num ficheiro (*matrixChanged*).
  - Já não há pontos na origem que podem ser removidos (*complex*).

Funções (mais significativas)

- `bool readWRL(char *Name)`
  Permite a abertura de um modelo tridimensional a partir do ficheiro referenciado pelo ponteiro `Name`, no formato WRL.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool writeWRL(char *Name)`
  Permite a guardar um modelo tridimensional num ficheiro, cujo nome é referenciado pelo ponteiro `Name`, no formato WRL.

Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool readCloud(char *Name, bool showMsg=true, float MIN_DISTANCE=0.0f, float MAX_DISTANCE=7.0f)`
  Permite ler os dados de uma nuvem de pontos guardada num ficheiro, cujo nome é apontado pela variável `Name`.
  A variável `showMsg` define se a função deve ou não operar no modo informativo, (envio de mensagens de texto do evoluir da função), `MIN_DISTANCE` define a distância mínima a que um ponto é considerado quando medida a partir do scanner laser e `MAX_DISTANCE` é a distância máxima.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `vtkImageData *doImage()`
  Retorna um ponteiro do tipo imagem de bits, para uma imagem em tons de cinza, a que cada ponto de cor se refere à profundidade de cada ponto da nuvem de pontos. Esta função requer que os pontos se encontrem no formato matricial, caso contrário devolve um apontador nulo (falha).

- `bool delaunayTriangulation()`
  Efectua a triangulação, a partir do modelo de pontos no formato matricial, recorrendo à triangulação Delauanay do modelo de pontos quando projectados numa matriz bidimensional, ignorando descontinuidades da superfície definida pelos pontos originais.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool doTriangulation(float MAX_DIF, float MAX_DIST, bool createEdges, bool createEdgeImage)`
  Efectua a triangulação do modelo de pontos, quando estes estão no formato matricial, ou seja, quando acabaram de ser lidos de uma nuvem de pontos.
  Utiliza o valor de entrada `MAX_DIF` como o valor máximo que um ponto pode estar afastado do seguinte, antes de se considerar que se trata de uma descontinuidade. O valor `MAX_DIST` define o raio máximo a que a triangulação é efectuado. A variável `createEdges`, quando é true, leva a função a criar um ficheiro de imagem com os contornos identificados – esta imagem pode ser utilizado como verificação da correcta identificação das descontinuidades.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool putModelPoints(renderWindow *rwin, double Cr, double Cg, double Cb)`
  Permite pôr o modelo de pontos, presente neste objecto, numa janela de visualização referenciada pelo ponteiro `rwin` a fim da mesma poder mostrar os pontos, na cor definida pelas variáveis: `Cr`, `Cg`, `Cb` - respectivamente os componentes vermelho, verde e azul, entre zero e um (formato RGB normalizado).
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool putTriangledPoints(renderWindow *rwin, double Cr, double Cg, double Cb)`
  Permite pôr o modelo de triangulado, presente neste objecto, numa janela de visualização referenciada pelo ponteiro `rwin` a fim da mesma poder mostrar os pontos, na cor definida pelas variáveis: `Cr`, `Cg`, `Cb` respectivamente os componentes vermelho, verde e azul, entre zero e um (formato RGB normalizado).
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `void transModel(vtkTransform *trans)`
  Aplica a transformação linear referenciada pelo ponteiro `trans` aos modelos de pontos e/ou triangulado, deste objecto, e actualiza a matriz interna de transformações lineares aplicadas (também deste objecto) com mais esta transformação linear. No caso de ainda não existir nenhum dos modelos acima citados, retorna sem efectuar qualquer operação.

- `vtkTransform* finalICPmove(modelManager *base, double maxDist, int maxpoints, int iterations)`
  Invoca a função ICP de aproximação de pontos, presente no VTK, a fim de aproximar os pontos do modelo a que pertence esta função aos pontos dos modelos do objecto referenciado. Por fim devolve

um apontador para a transformada linear de corpo rígido, capaz de aproxima o modelo da função ao modelo do objecto referenciado. A função de aproximação tenta sempre utilizar os pontos de aproximação dos modelos VTK triangulados. Caso estes não estejam presentes, utiliza os pontos dos modelos VTK de pontos.
A escolha dos pontos de aproximação dos modelos tem em conta que podem ter até uma distância máxima aos pontos escolhidos do outro modelo igual à variáve de entradal `maxDist`.
A função tenta escolher um número de pontos próximos da variável de entrada `maxpoints`.
A função ICP é utilizada com um número máximo de iterações definidos pela variável de entrada `iterations`.
No caso do modelo da função e/ou o modelo referenciado não terem modelos VTK a função sinaliza esta falha com o retorno de um apontador igual a zero.

- `bool saveTrasform2File(char *fileName=0)`
  Permite guardar a matriz de transformação linear de corpo rígido de todas as transformações lineares aplicadas ao objecto num ficheiro com o nome referenciado pelo apontador `fileName`. No deste apontador referenciar para zero, a função 'tenta' utilizar o nome ficheiro da matriz de transformação linear com o nome de omissão (ficheiro que deu origem ao modelo, adicionando-lhe a extensão '.matrix').
  No caso do objecto ainda não ter sofrido mais qualquer transformação linear (flag de transformação linear aplicada a false), desde que o modelo foi carregado, a função aborta.
  No caso de se ter carregado o ficheiro de transformação linear por omissão e ainda não se aplicou mais nenhuma transformação linear, a função aborta.
  No caso de se ter salvo a matriz de transformação linear no ficheiro por omissão e a seguir não se aplicar mais nenhuma transformação linear, a função aborta.
  Devolve true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool applyTrasformInFile(char *fileName=0)`
  Permite carregar e aplicar a matriz de transformação linear de corpo rígido de transformações lineares ao objecto presente num ficheiro com o nome referenciado pelo apontador `fileName`. No caso deste apontador referenciar para zero, a função 'tenta' utilizar o nome ficheiro da matriz de transformação linear com o nome de omissão (ficheiro que deu origem ao modelo, adicionando-lhe a extensão '.matrix').
  No caso de se utilizar o nome por omissão, a flag de transformação linear aplicada é desactivada e activada em caso contrário.
  Devolve true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool mergeModel(modelManager *modelAdded, bool doMergeTriangles)`
  Adiciona o(s) modelo(s) VTK (pontos e triangulado) presete(s) ,no objecto referenciado pelo apontador `modelAdded` ao(s) modelo(s) presente(s) no objecto em que se invoca a função.
  No caso da variável `doMergeTriangles` estiver a true, e existirem m modelos VTK triangulados nos objectos da função e `modelAdded`, a função efectua uma junção dos triângulos, junção de superfícies paralelas e união de limites (fronteiras).
  Antes de se invocar esta função deve-se fornecer ao objecto vários parâmetros de processamento a partir da função `setAddParameters(…)`.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool mergeCloud(char *filename, bool doMergeTriangles)`
  Abre a nuvem de pontos referenciada pelo apontador `filename` aplicando-lhe a transformação linear por omissão deste ficheiro. De seguida, caso o objecto que invoca a função tenha um modelo triangulado, efectua uma triangulação aos pontos da nuvem de pontos.
  Então adiciona o(s) modelo(s) VTK (pontos e/ou triangulado) criados a partir da nuvem de pontos ao(s) modelo(s) presente(s) no objecto em que se invoca a função.
  No caso da variável `doMergeTriangles` estiver a true, e existirem m modelos VTK, a função efectua uma junção dos triângulos, junção de superfícies paralelas e união de limites (fronteiras).
  Antes de se invocar esta função deve-se fornecer ao objecto vários parâmetros de processamento a partir da função `setAddParameters(…)`.

Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `void printMatrixTransform(vtkTransform *trf=0)`
  Mostra uma mensagem com os coeficientes da matriz de transformação linear referenciada pelo apontador `trf`. Caso este apontador esteja a nulo, mostras os coeficientes da matriz de transformação do objecto que invocou a função.

- `bool useTriangulatedModelPoints()`
  Instrui o objecto que invoca a função a utilizar os pontos do seu modelo VTK triangulado para o seu modelo VTK de pontos. Antes disso, esta função elimina quaisquer pontos órfãos do modelo VTK triangulado.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool removeZeroPoints()`
  Limpa os pontos do modelo de pontos do objecto que invoca a função, que foram criados apenas para manter o formato matricial destes pontos, necessário à triangulação. Mesmo após operações de transformação linear, ainda consegue descobrir e eliminar estes pontos.
  Além de limpar estes pontos, também efectua uma limpeza a pontos órfãos do modelo triangulado e elementos usados na triangulação.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool deletePointsModel()`
  Remove o modelo de pontos VTK do objecto que invoca a função.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `bool undoTrasformMatrix(bool show_zero_matrix)`
  Esta função leva a que, os modelos do objecto em que é invocada, seja aplicada uma transformação linear inversa ao produto de todas as transformações lineares aplicadas, desde que este foi carregado a partir de um ficheiro (nuvem de pontos ou modelo WRL) a fim de inverter as mesmas. De seguida reinicializa-se a matriz de transformação linear do objecto.
  O parâmetro `show_zero_matrix` define se é para mostrar ou não (*true, false* - respectivamente) a matriz de transformação linear do objecto.
  Devovle true ou false caso a operação decorra ou não com sucesso, respectivamente.

- `int doSimplifyTriangulation(double vecError,bool recursive)`
  Esta função permite aplicar um algoritmo de simplificação do modelo VTK triangulado do objecto em que se invoca a função.
  A variável `vecError` determina a variação máxima da soma das coordenadas da normal de cada triangulo – utilizada na validação de superfícies (planas).
  A variável booleana `recursive`, quando recebe o valor true, informa a função de que a função deve ser repetida, até deixar de haver mais triângulos simplificados.
  Como retorno, a função devolve -1 em caso de ter ocorrido um problema na sua execução, caso contrário retorna o número de triângulos removidos durante o processo de simplificação, ao modelo triangulado original.

- `void setAddParameters(triangType ttype, float MAX_DIF, float MAX_DIST, float regdist)`
  Visto certas funções, desta classe chamarem outras da mesma, ter de se enviar todos os parâmetros a todas as funções tornaria algumas funções, como a junção de modelos com muitas variáveis de entrada. A fim de contornar este problema, optou-se pelo envio de vários parâmetros de triangulação e junção antes de invocar quaisquer funções que recorram aos mesmos.
  Neste caso `ttype` é uma variável enumerada que determina o tipo de triangulação a ser aplicada; `MAX_DIF` determina o valor máximo de detecção de contornos; `MAX_DIST` a distancia máxima de um ponto; e `regdist` a distancia máxima de união de dois pontos durante uma operação de junção de modelos.

- `int GetNumPoints()`
  Devolve o número de pontos a ser projectados no modelo VTK de pontos do objecto que invoca a função. Caso o objecto não exista devolve -1.

- `int GetNumTriangles()`
  Devolve o número de triângulos a ser projectados no modelo VTK de triângulos do objecto que invoca a função. Caso o objecto não exista devolve -1.

- `bool canTriangulate()`
  Devolve true ou false caso possa, ou não, respectivamente, ser aplicada uma operação de triangulação ao objecto que invoca a função.

- `char *getFilename()`
  Devolve um ponteiro para uma posição de memória com o nome do ficheiro que deu origem aos dados do objecto em questão. No caso de não existir qualquer ficheiro associado, devolve o apontador 0 (zero).

- `void certifyModelsObjects()`
  Verifica e assegura que o modelo de pontos do objecto tem apenas pontos e o modelo de triângulos tem apenas triângulos (função interna).

Além das funções acima descritas, existe um conjunto de funções directamente acedidas por esta classe, numa outra classe, denominada por Functions3D. O porquê deste outra classe deve-se ao facto de assim se conseguir uma separação e simplificação das duas classes.

VII.3.2. Objecto: Funções de operação directa sobre modelos (`Functions3D`)

O acesso a estas funções é directo, pois são completamente independentes de qualquer classe, dependendo apenas das variáveis de entrada de cada função (são funções estáticas) – esta classe não precisa de ser construída, nem tem variáveis internas.

Funções

- `static int simplifyTriangulatedModel(vtkPolyData *model, double VectorErrorLimit, bool recursivo)`
  Testa e simplifica o modelo triangulado `model` com um limite de variação das normais de um triangulo `VectorErrorLimit` e efectuar ou não um algoritmo repetitivo até não existirem mais simplificações, definido pela variável `recursivo`.
  Devolve o número de triângulos eliminados e -1 no caso de ocorrer um erro.

- `static bool cleanOrphanPoints(vtkPolyData *model, int points_per_cell)`
  Elimina todos os objectos não projectáveis (render objects) do modelo VTK `model` que não tenham um número de pontos igual a `points_per_cell` e depois remove todos os pontos que não pertençam aos objectos projectáveis (ponto órfão).
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `static bool mergeTriangles(vtkPolyData *baseModel, vtkPolyData *addModel, float distval)`
  Adiciona os triângulos do modelo VTK `baseModel` os triângulos do modelo VTK `addModel`. Efectua uma junção dos pontos cuja distancia seja inferior a `distval`, eliminando triângulos sobrepostos e com área nula.
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `static bool internTransf(vtkPolyData *&model, vtkTransform *trans, bool invert)`
  Aplica a transformação linear `trans` ao modelo `model`. A variável `invert` define se a transformação linear a inverter é directa (false) ou inversa (true) da fornecida.
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `static bool mergePoints(vtkPolyData *baseModel, vtkPolyData *addModel,float distance)`
  Efectua a adição do modelo VTK de pontos `addModel` ao modelo VTK de pontos `baseModel` juntando os pontos, num só, cuja distância entre si seja inferior a `distance`.
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `static bool createVertsFromModel(vtkPolyData *&destiny, vtkPolyData *source)`
  Cria um modelo VTK de pontos em `destiny` com os pontos do modelo VTK `source`.
  Aceita que um dos apontadores tenha valor nulo.
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `static bool joinModelsBasic(vtkPolyData *&base, vtkPolyData *add)`
  Adiciona todos os objectos do modelo VTK `add` ao modelo VTK `base`. Não efectua qualquer junção de pontos, triângulos que ocupem a mesma posição ou eliminação de triângulos com área nula.
  Aceita que um dos apontadores tenha valor nulo.
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

VII.3.3. Objecto: Janela de Visualização (`renderWindow`)

Tendo em vista que cada modelo VTK a ser visualizado deve ter uma classe capaz de efectuar várias operações sobre o mesmo, nomeadamente:

- Mostrá-lo numa dada janela.
- Adicioná-lo a um conjunto de modelos.
- Marcar pontos de aproximação.
- Cálculos de aproximação.
- Gerir várias características do modelo que representa.

Apesar de ocupar pouco espaço em memória, estes objectos de visualização apenas são criados durante o período de tempo que é necessário mostrar o modelo, sendo imediatamente eliminadas quando já não sejam necessárias.

Na aplicação, esta classe é utilizada de duas formas:

1) Os objectos criados a partir desta classe podem ser directamente utilizado (e são) em vários comandos que requerem alguma interacção entre um modelo modelos e o utilizador, a nível do próprio modelo.

2) Por outro lado foram criadas classes que, trabalhando directamente com esta, podem mostrar um dado conjunto de modelos, de forma interactiva, num única janela de visualização – o que se optou por chamar de janela real em oposto à janela de visualização.

Esta classe pode operar de várias formas distintas, ou combinadas:

1) No modo interactivo em que cada vez que se prime a tecla ‘m’ é marcada uma esfera na posição do ponteiro do rato.
2) No modo de segundo plano, ou passivo em que a janela é quase totalmente independente da aplicação e pode estar aberta durante a execução de comandos da aplicação. Após comandos em que dados a ser visualizados sejam alterados, a partir de um dado modelo, são prontamente propagados à janela, perdendo-se apenas a posição de visualização do modelo (*viewport*).

3) No modo de primeiro plano, ou espera, em que a janela é aberta e só após interacção e fecho da janela, por parte do utilizador, é que a aplicação continua a sua execução.

De seguida, apresenta-se as várias funções e variáveis presentes nesta classe.

Variáveis (mais significativas)

- Transformada linear VTK (`vtkTransform* transform`)
  Guarda a posição de memória do último cálculo da transformada linear calculada pelo objecto, para poder ser eliminada durante a eliminação do objecto.

- Câmara de visualização VTK(`vtkCamera *camera`)
  Este ponteiro guarda o objecto VTK câmara, necessário durante a projecção de um objecto VTK. Para já, não está a ser utilizado.

- Actor de projecção VTK (`vtkActor *actor`)
  Objecto VTK que permite atribuir características (cor, por exemplo) ao modelo VTK a ser projectado.

- Projector VTK (`vtkRenderer *render`)
  Este ponteiro refere-se ao objecto VTK encarregue de "desenhar" o modelo VTK que se pretende projectar.

- Janela de projecção VTK (`vtkRenderWindow *renderWin`)
  Referência de memória para o objecto encarregue de gerir a janela VTK em que o objecto está a ser projectado.

- *Mapeador* de modelos VTK (`vtkPolyDataMapper *modelMapper`)
  Este objecto VTK serve para de fornecer as posições e formas presentes no modelo VTK a serem projectados pelas rotinas OpenGL.

- *Interactor* VTK (`vtkRenderWindowInteractor *interactor`)
  Este objecto permite ao utilizador interagir com a janela de projecção de um dado objecto VTK.

- Selector VTK (`vtkPointPicker *picker`)~
  Este objecto permite seleccionar modelos VTK a partir de um modelo projectado numa janela VTK.

- Apontador para o objecto pai (`renderWindow *rw`)
  Este ponteiro é utilizado para referir qual o objecto, desta classe, de que este objecto é filho. Num sistema um pai, vários filhos, é possível juntar vários modelos VTK, cada um gerido por um objecto desta classe, numa só janela de visualização, em que a janela é gerida pela classe pai.
  Quando este objecto é criado como pai, este apontador aponta para 0 (zero).

- Apontador para o único objecto interactor (`static renderWindow* renderWindowInteracting`)
  Ponteiro para o objecto desta classe que se encontra a interagir com o utilizador a uma dada altura. Apenas um só objecto desta classe pode estar a interagir a uma dada altura. Caso não esteja a ocorrer uma interacção, este objecto tem o valor 0 (zero).

- Conjunto de esferas de selecção (`vtkActorCollection *selectedPoints`)
  Este ponteiro trata-se de um objecto capaz de guardar actores para modelos VTK a serem projectados numa janela. Neste caso guarda os actores das esferas para marcação dos pontos no modelo, pelo utilizador. Mais tarde estes pontos são utilizados para aproximar dois modelos VTK.

Funções (mais significativas)

No momento da sua construção, um objecto desta classe pode ser construído como pertencente a mais objectos desta mesma classe a fim de se poder projectar todos os modelos VTK dos objectos assim unidos numa só janela. A fim de facilitar a compreensão dos mesmos, pode-se considerar o objecto principal como "pai" e o(s) objecto(s) que lhe *pertence(m)* como "filho(s)".
A figura abaixo exemplifica esta forma de organização:

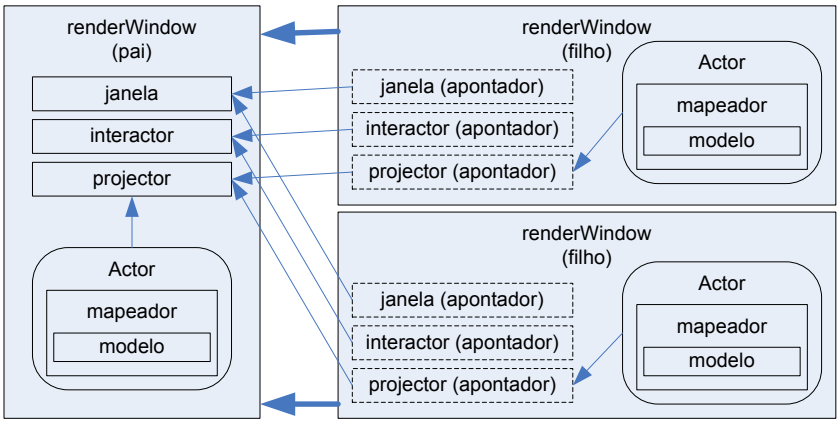


Figura VII.3 - Forma de interligação da classe renderWindow

- `renderWindow(renderWindow *rw_=0)`
  Construtor do objecto desta classe. Pode-se construir a classe como pai (`rw_==0`), ou como filho, fornecendo como parâmetro de entrada a posição de memória do objecto pai.

- `void poeEsfera(double *Coords)`
  Adiciona uma esfera ao conjunto de esferas já existentes, com centro no *array* de 3 (três) coordenadas referenciadas por `Coords`.
  A adição de esferas pode ou não ser efectuada a cores diferentes.
  Pode-se no máximo adicionar até 3 (três) esferas, sendo a mais antiga removida quando se atinge este número e uma nova esfera é adicionada.

- `void actuaPicker()`
  Verifica em que posição do modelo o rato se encontra, e caso seja sobre um ponto válido, invoca a função `poeEsfera(…)` relativamente a essa mesma posição.

- `void createRenderWin(bool enablePicker)`
  Cria os objectos da janela em que irá ser projectado o modelo associado ao objecto e, caso hajam mais objectos associados, os modelos destes objectos.
  Caso o parâmetro `enablePicker` tenha o valor *true*, esta janela irá permitir a marcação de pontos (a partir de esferas).

- `void applyTransform(vtkTransform* trf)`
  Aplica ao modelo do objecto em questão a transformada linear referenciada por `trf`.

- `void startInteraction(bool final=true, char *title=0,int size_x=500, int size_y=500)`
  Depois da função `createRenderWin(…)` ter sido invocada, esta função permite a criação de uma janela capaz de projectar os modelos (já referidos na função `createRenderWin`). Esta janela largura e altura definido pelos parâmetros de entrada `size_x` e `size_y`, respectivamente. O seu titulo será igual ao texto referenciado por `title`, caso este apontador seja 0 (zero), a janela terá um título atribuído pelo próprio VTK.
  Quanto ao parâmetro `final`, caso este seja *true*, ao chamar esta função, a aplicação não retoma o controlo até que o utilizador feche a janela criada pela função.

- `vtkTransform* do3PointICPmove(renderWindow *base, bool moveActor=false)`
  Esta função permite que, a partir de 3 (três) pontos marcados na janela de visualização deste objecto e outros 3 (três) num objecto da mesma classe, referenciado por `base`, esta função retorne um ponteiro para a transformação linear que irá aproximar os primeiros aos segundos grupos de 3 (três) pontos.
  Caso o parâmetro `moveActor` seja *true*, a função também aplica a transformação linear retornada ao modelo do objecto que invoca a função.
  O algoritmo utilizado é o algoritmo ICP, já descrito. Neste algoritmo a ordem dos pontos não interessa, pelo que estes devem formar um triângulo com lados diferentes.

- `vtkTransform* do3PointLandmarkMove(renderWindow *base, bool moveActor=false)`
  Esta função é similar à função anterior (`do3PointICPmove(…)`), excepto que, em vez de recorrer ao algoritmo ICP, utiliza um algoritmo de aproximação directa, que embora seja muito mais preciso que o anterior, requer que a ordem dos pontos, nos dois modelos corresponda, ou seja, o ponto *p_base(n)* de um modelo tem que corresponder ao ponto *p_move(n)* do outro modelo.

- `void remove3points()`
  Esta função elimina todas as esferas, correspondentes aos 3 (três) pontos de marcação desta classe da janela de projecção e dispõe o objecto no modo em que ainda não foi marcado qualquer ponto.

- `bool has3points()`
  Esta função retorna o valor *true* no caso do objecto que a invoca já ter os 3 (três) pontos de marcação com valor não nulo.

- `double *get3Points()`
  Quando os 3 (três) pontos de marcação do objecto que invoca a função tiverem valor não nulo esta função retorna um apontador para a posição em que estão guardadas as coordenadas destes pontos. Caso contrário retorna 0 (zero).

Nota: além desta classe, existe uma classe denominada `vtkMyWindowEvent` que tem como objectivo tratar e lidar com eventos da janela virtual. Apenas se cria e acede a objecto desta classe, em objectos da classe `renderWindow` criados como *pai* (acima descrito).

VII.3.4. Objecto: Janela Real (`windowManager`)

A cada janela de visualização criada na aplicação, encontra-se associado um objecto desta classe.
O objectivo principal desta classe é criar/remover e gerir os diferentes parâmetros que objectos do tipo ‘janelas de visualização’ necessitam a fim de que o modelo VTK a eles associado possa ser projectado numa só janela, com características próprias, atribuídas pelo utilizador, a cada um dos modelos.

A figura seguinte pretende exemplificar a forma como esta classe cria e gere as várias classes a esta associadas:

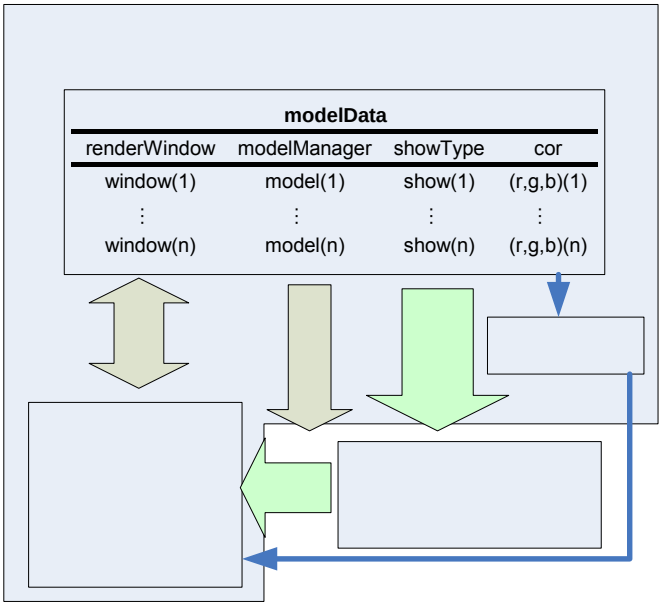


Figura VII.4 - diagrama da classe windowManager

Variáveis (mais significativas)

- Estado de visibilidade da janela (`bool windowCreated`)
  Define o estado da janela real estar ou não visível, com os valores *true* e *false*, respectivamente.

- Número máximo de modelos VTK(`int maxmodels`)
  Esta variável define o número máximo de modelos que uma janela pode conter. É definida na criação do objecto e posteriormente não pode ser alterada.

- Definições da janela (`struct winStats`)
  Esta estrutura de dados guarda o tamanho (horizontal, vertical) e título da janela real.

- Lista dos modelos a projectar (`modelData *models`)
  Este ponteiro define a posição de memória que contém uma estrutura de dados da janela real, relativamente aos modelos a projectar, cor de projecção e pontos ou triângulos.

Funções (mais significativas)

- `windowManager(int maxmodels_=25)`
  Construtor desta classe em que o parâmetro de entrada `maxmodels` define o número máximo de modelos que poderão ser representados numa janela de visualização.

- `bool createWindow(char *title=0, int x = 600,int y=400)`
  Antes de mostrar uma janela de visualização de modelos, é necessário chamar esta função a fim de criar a mesma. Retorna o valor *true* em caso de sucesso.
  Os parâmetros `title`, `x`, e `y` definem o título, largura e altura da janela de visualização dos modelos.

- `bool addModel(modelManager *model, showType show=points, float r=0.6f, float g=0.6f, float b=0.6f)`
  Adiciona um modelo referenciado por `model`, à lista de modelos a projectar na janela de visualização. Os parâmetros seguintes: `show` e `r,g,b` definem se deve-se projectar os pontos ou triângulos em `model` e qual a cor (formato normalizado), respectivamente.

Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `bool removeModel(modelManager *model)`
  Acede à lista de modelos a projectar e remove todas as entradas em que o modelo a projectar pertença ao objecto `model`.
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `void refreshModel(modelManager *model)`
  Informa o objecto que invoca a função de que, caso este tenha na sua lista de modelos a projectar, o objecto `model`, deverá destruir e reconstruir a janela de projecção com novos dados do objecto referenciado por `model`, pois foi alterado.

- `bool hideWindow()`
  Comanda o objecto que invoca a função a remover a sua janela de visualização.
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `bool hideWindow(void *ptr)`
  Verifica se o ponteiro `ptr` tem o endereço de um objecto da lista de janelas de visualização do objecto que invoca a função. Caso verifique, instrui o objecto a invocar a remoção da sua janela de visualização.
  Devolve *true* ou *false* caso a operação decorra ou não com sucesso, respectivamente.

- `int getMaxModels()`
  Devolve o número máximo permitido de modelos, da lista de modelos do objecto que invoca a função.

- `modelManager* getModel(int i)`
  Devolve a o endereço do modelo, na posição `i` da lista de modelos a projectar pela janela de projecção.
  Caso `i` esteja fora do intervalo permitido devolve 0 (zero).

- `showType getShowType(int i)`
  Devolve o tipo de modelo a ser projectado (pontos ou triângulos) na posição `i` da lista de modelos a projectar.

- `void getColor(int i, int &r, int &g, int &b)`
  Devolve a cor do modelo a ser projectado na posição `i` da lista de modelos a projectar, nas variáveis de entrada: `r`,`g`,`b` (formato RGB normalizado).
  Caso `i` esteja fora do intervalo permitido devolve a cor preta (0,0,0).

VII.3.5. Objecto: Janela da aplicação (`mainWindow`)

Neste objecto que todos os objectos principais, acima descritos, são criados. Também se cria os objectos gráficos (interface gráfica) e é onde as funções invocadas pelos objectos gráficos estão presentes.

Variáveis (mais significativas)

- Classe criadora dos objectos da janela da aplicação (`Ui_mainWindow *ui`)
  A classe '`Ui_mainWindow`', criada a partir do ficheiro de formulário QT '`forms\mainWindow.ui`', que é compilada, dando origem ao ficheiro '`ui_mainWindow.h`', serve para criar no objecto da classe '`mainWindow`' os objectos gráficos QT nele existentes e aceder aos mesmos.
  O acesso é efectuado a partir do apontador `ui`.

- Apontador para a classe pai (`QWidget *parent`)
  Apesar de não ser o caso, este apontador poderá ser utilizado para "encaixar" a janela QT deste objecto numa janela de nível superior. É utilizado com o valor 0 (zero) a fim de criar o objecto como a janela principal, sem janelas de nível superior.
  Este valor é utilizado na criação do objecto desta classe, como parâmetro de entrada do construtor.

- Objecto de operações de escrita/leitura de ficheiros (`FileIO *fileio`)
  Tendo sido criada para operações de leitura e escrita de ficheiros, a classe `FileIO` actualmente não se encontra a ser utilizada para todas as operações deste género, sendo essas mesmas operações partilhadas com a classe '`modelManager`'.
  Esta variável é um apontador para o objecto da classe `FileIO` da aplicação.

- Tabela de modelos em memória (`modelManager **models`)
  Referência da posição de memória de uma tabela de apontadores para os modelos abertos na aplicação. Os apontadores são todos criados com o valor 0 (zero), sendo-lhes atribuído o valor da posição de memória do objecto '`modelManager`' na criação do mesmo.
  O número máximo de modelos que a aplicação aceita é definido pela constante de compilação '`CONST_MAX_MODELS`'.

- Tabela de janelas reais em memória (`windowManager **windows`)
  Referência da posição de memória de uma tabela de apontadores para as janela reais abertas na aplicação. Os apontadores são todos criados com o valor 0 (zero), sendo-lhes atribuído o valor da posição de memória do objecto '`windowManager`' na criação do mesmo.
  O número máximo de janelas reais que a aplicação aceita é definido pela constante de compilação '`CONST_MAX_WINDOWS`'.

- Janela real seleccionada (`int selWindow`)
  Define qual a janela real se encontra seleccionada, tendo o valor *-1* quando nenhuma se encontra seleccionada.

- Modelo seleccionado (`int selModel`)
  Define qual o modelo se encontra seleccionado, tendo o valor *-1* quando nenhum se encontra seleccionado.

- Cor RGB seleccionada (`QColor *corActiva`)
  Objecto do tipo `QColor` que define qual a cor RGB seleccionada.

- Semáforo QT (`QSemaphore *semaforo`)
  Semáforo QT que permite gerir o funcionamento dos processos da aplicação a fim dos mesmos operarem na ordem correcta
  Não se encontra a ser utilizado.

- Objecto de invocação de processos (`threadRun *thread`)
  Apontador para um objecto da classe `threadRun` que permite criar processos separados para a execução de operações VTK sem estas bloquearem os objectos QT gráficos da aplicação.
  Ainda não se encontra a ser utilizado.

Funções (`slots`)

As funções do tipo '`slots`' são especiais, na medida em que são utilizadas para estabelecer ligações dos objectos QT a partir dos seus sinais ('`signals`') a estas funções, utilizado a função QT '`connect(…)`'.

Atendimento à acção de botões QT:

- `void call_pushButton_ChangeColor()`
  Clicar no botão de mudar a cor activa.

Atendimento às '*pull down boxes*':

- `void call_WinChanged(int)`
  A partir da caixa da janela seleccionada, mudança da mesma.

- `void call_ModelChanged(int)`
  A partir da caixa de modelo seleccionado, mudança do mesmo.

Atendimento ao menu, secção '*Window*':

- `void call_action_WinNew()`
  Acção do menu '*Window→ New*'.

- `void call_actionWinView()`
  Acção do menu '*Window→View*'.

- `void call_actionWinHide()`
  Acção do menu '*Window→Hide*'.

- `void call_actionWinAdd_model()`
  Acção do menu '*Window→Add model*'.

- `void call_WinRemove_model()`
  Acção do menu '*Window→Remove model*'.

Atendimento ao menu, secção '*Model*':

- `bool call_actionModel_New()`
  Acção do menu '*Model→ New model*'.

- `void call_actionModelAdd_cloud()`
  Acção do menu '*Model→Load from cloud file(s)…*'.

- `void call_actionModelMerge_model()`
  Acção do menu '*Model→Merge with model…*'.

- `void call_actionModelMerge_cloud()`
  Acção do menu '*Model→Merge with cloud file…*'.

- `void call_actionModelTriangulate()`
  Acção do menu '*Model→Triangulate*'.

- `void call_actionModelSimplifyTriangulation()`
  Acção do menu '*Model→*'.

- `void call_actionModel3pointApproach()`
  Acção do menu '*Model→→3 Point approach...*'.

- `void call_actionModelFinal_ICP()`
  Acção do menu '*Model→→Final ICP approach…*'.

- `bool call_actionModelClean_extra_points(bool ask=true)`
  Acção do menu '*Model→ Remove zero points (from matrix format)*'.

O parâmetro `ask` define a existência de uma caixa de diálogo a confirmar a operação.

- `void call_actionModelUseTriangulatedModelPoints()`
  Acção do menu '*Model→ Use triangulated model points as points model*'.

- `bool call_actionactionModelLoadMatrix(bool ask=true)`
  Acção do menu '*Model→Linear transformation matrix→Load transformation matrix*'.
  O parâmetro `ask` define a existência de uma caixa de diálogo a confirmar a operação.

- `bool call_actionactionModelUndoMatrix(bool ask=true)`
  Acção do menu '*Model→Linear transformation matrix→Undo transformation matrix*'.
  O parâmetro `ask` define a existência de uma caixa de diálogo a confirmar a operação.

- `bool call_actionactionModelsaveMatrix(bool ask=true)`
  Acção do menu '*Model→Linear transformation matrix→Save transformation matrix*'.
  O parâmetro `ask` define a existência de uma caixa de diálogo a confirmar a operação.

- `void call_actionModelClose(modelManager *mod = 0)`
  Acção do menu '*Model→Close model*'.
  O parâmetro `mod`, quando diferente de 0 (zero) causa o encerramento do modelo referenciado pelo parâmetro, caso contrário a função actua sobre o modelo activo.

Nota: estas funções além de serem invocadas pelos por sinais QT, também podem (e são) invocadas directamente no código da aplicação em caso de necessidade.

Funções (`protected`)

Estas funções são substituições das já existentes na classe QT `QMainWindow`, herdada pela classe em análise (`mainWindow`).
Neste caso tratam-se de funções tratam directamente eventos QT:

- `bool eventFilter(QObject *dest, QEvent *event)`
  Atendimento de eventos QT, definidos pelo utilizador:
  1. Evento de ocultação da janela VTK referenciada pelo evento.
  2. Evento de mensagens de texto enviadas pelas várias classes da aplicação

- `void closeEvent(QCloseEvent *event)`
  Atendimento a um evento de encerramento da aplicação com uma mensagem de confirmação.

- `void resizeEvent(QResizeEvent *event)`
  Atendimento a eventos de redimensionamento da janela principal da aplicação, de forma a redimensionar a largura das colunas das tabelas da janela principal.

Funções (restantes)

- `static void msgShow(const char *msg, mainWindow *_ptr = 0)`
  Função de invocação directa (não necessita a criação do objecto) que permite escrever o texto referenciado por `msg` na caixa de mensagens ('Processing messages'). A primeira vez que se invoca esta função, tem de se referir o endereço da classe principal, a partir do parâmetro `_ptr`.

- `void configureGui()`
  Construtor dos objectos gráficos da janela principal.

- `void connectSlots()`
  Efectua todas ligações QT entre os objectos gráficos da janela principal e as funções to tipo '`slots`' desta classe.

- `void updateWinList()`
  Actualiza a 'p*ull-down box*' da lista das janelas (reais) existentes.

- `void updateModelList()`
  Actualiza a 'p*ull-down box*' da lista dos modelos em memória.

- `void updateWinSpecs()`
  Actualiza os elementos da tabela de modelos da janela activa (*Active window details*).

- `void updateModelsSpecs()`
  Actualiza os elementos da tabela de modelos em memória (*Models details*).

VII.4. Interface gráfica

VII.4.1. Janela principal

Com o objectivo de facilitar o acesso a todas as funções de uma forma directa, com um mínimo de caixas de diálogo, mas sem sobrecarregar o utilizador com informação, optou-se pela utilização de uma janela única com comandos simples, de acesso directo, com as opções directamente acessíveis a partir de um conjunto de abas que as separa em grupos funcionais.
É de salientar a utilização de um modelo de redimensionamento da aplicação parametrizado de acordo com a necessidade de redimensionamento de cada área.
A imagem seguinte dá-nos um exemplo desta janela, com as suas áreas diferentes funcionais e zonas de redimensionamento da janela:

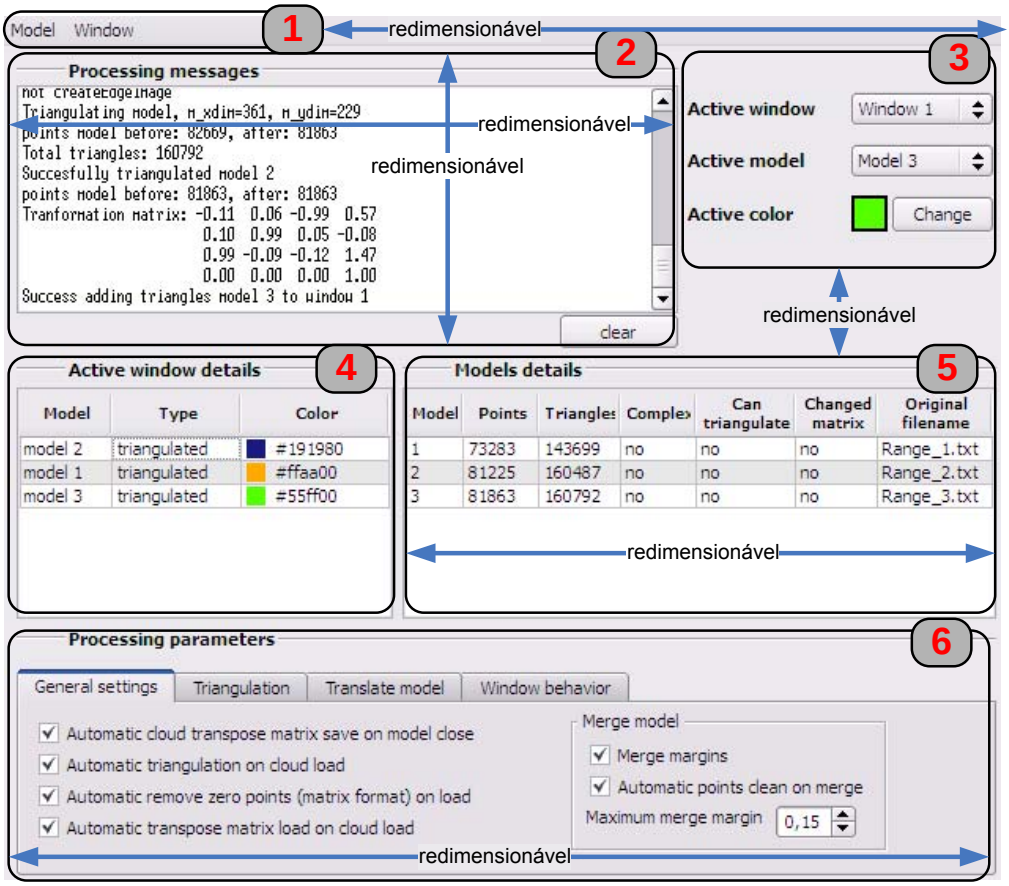


Figura VII.5 - Janela principal da aplicação

Legenda das áreas funcionais:

1. Menu principal.

   É a partir desta área que o utilizador efectua os vários comandos disponibilizados pela aplicação.

2. Área de apresentação das mensagens decorrentes das operações efectuadas.

   Nesta área surgem mensagens, tanto informativas como de notificação do utilizador, relativamente aos que este deve fazer, em determinadas situações.
   Ao clicar no botão '*clear*', desta área o texto presente é limpo.

3. Selectores de: modelo activo, janela activa e cor activa.

A partir desta zona, o utilizador pode seleccionar a janela activa, o modelo activo e a cor activa, relativamente aos comandos chamados a partir do menu principal.

4. Detalhes do modo de apresentação dos modelos presentes na janela activa.

Esta tabela mostra os vários parâmetros com que cada modelo será apresentado, na janela real activa. Pode-se seleccionar a cor, o tipo de apresentação (pontos ou triângulos)
Nota: nem todos os modelos poderão ter o formato de pontos e/ou triangulado.

5. Detalhes dos modelos em memória.

Esta tabela mostra vários dados relativamente aos modelos existentes em memória. Antes do utilizador efectuar algum comando sobre um dado modelo, deve verificar estes dados, a fim de se certificar da operação que pretende efectuar.

6. Opções dos diferentes parâmetros de processamento da aplicação.

A fim do utilizar estar informado, relativamente aos parâmetros do comando que pretende chamar a partir do menu principal, deve seleccionar a aba correspondente ao comando que pretende chamar e confirmar se os parâmetros aí apresentados são os que pretende utilizar.

VII.4.2. Janela real

As janela de apresentação dos vários modelos, disponibilizadas pela aplicação, podem estar permanentemente abertas, a partir do momento que o utilizador lhes atribui (pelo menos) um modelo, adaptando-se aos diferentes comandos efectuados sobre os modelos e/ou janelas.
Estas janelas têm disponível uma interface, disponibilizada pelo próprio VTK, em que é possível mover, ampliar, reduzir, rodar, activar o modo '*wireframe*' entre outros, de um dado conjunto de modelos associados à janela, sem alterar os modelos propriamente ditos.

A imagem abaixo mostra o exemplo de um modelo apresentado por esta janela:

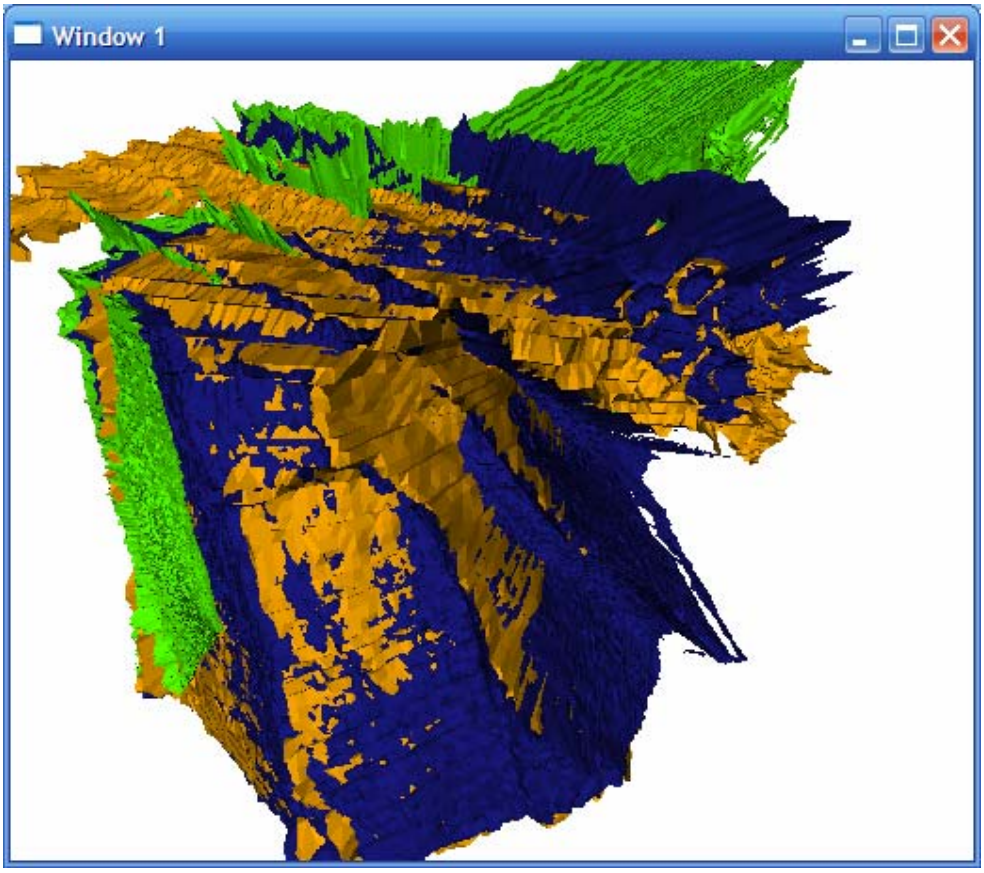


Figura VII.6 - Janela real (para projecção de modelos)

VII.4.3. Janelas secundárias de interacção directa com o utilizador

Além da janela principal da aplicação, existem janelas e/ou caixas de diálogo que informam e, por vezes, requerem alguma interacção por parte do utilizador. Estas podem ser divididas nos seguintes conjuntos:

1. Caixas informativas relativamente ao estado da aplicação:

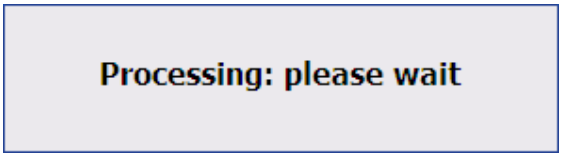


Figura VII.7 – Diálogo de processamento em curso

2. Diálogos(s) de confirmação:

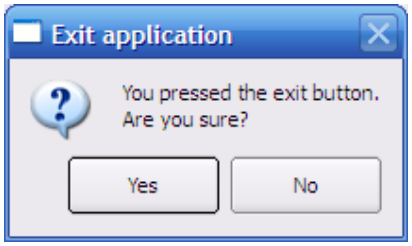


Figura VII.8 - Diálogo de confirmação de saída

3. Diálogo(s) de abertura/gravação de ficheiros:

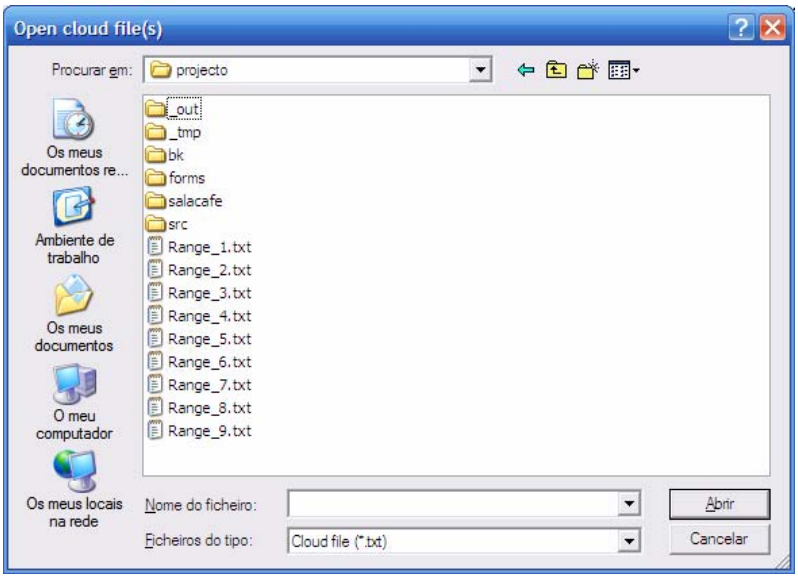


Figura VII.9 - Diálogo de abertura de uma nuvem de pontos

4. Janela de marcação de pontos (após a marcação dos pontos, o utilizador tem de fechar este género de janelas, clicando no botão ou a tecla 'Q')

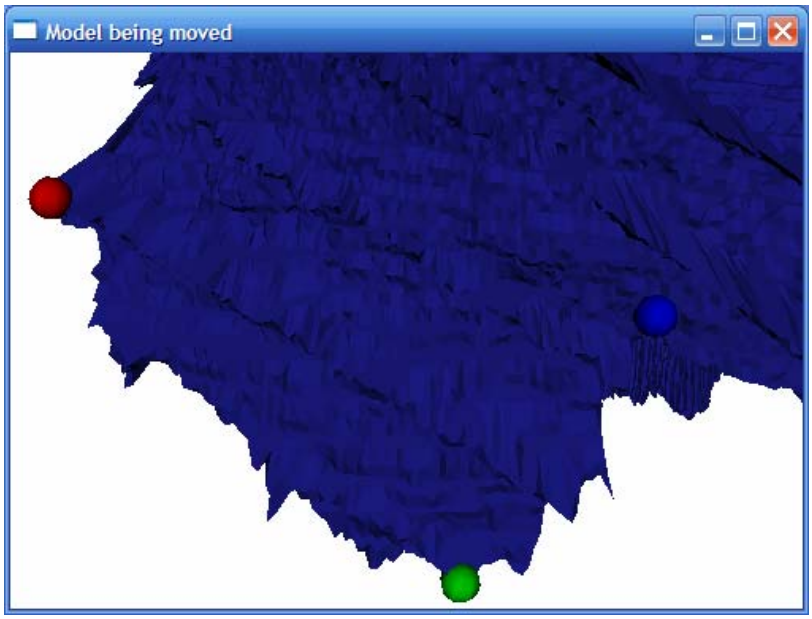


Figura VII.10 - Janela virtual para marcação de 3 pontos de aproximação

5. Selecção de modelo, para uma dada operação:

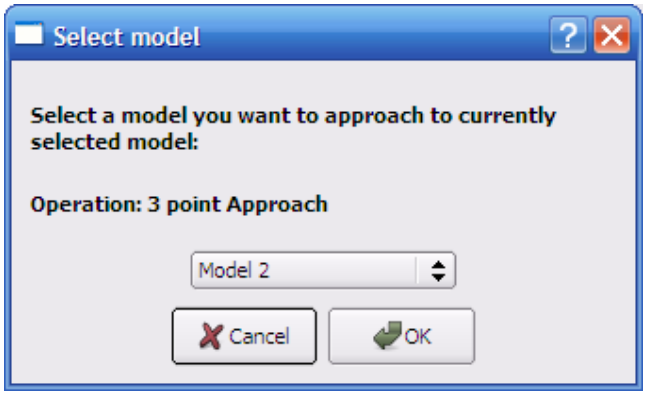


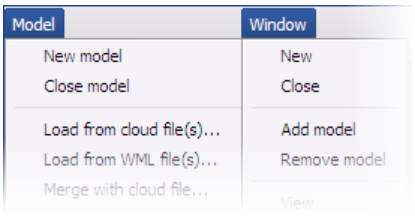
Figura VII.11 - Diálogo de selecção de um modelo para aproximação por 3 pontos comuns

VII.5. Interface do utilizador

Na aplicação desta dissertação, optou-se pela utilização de um conjunto de comandos simples, com um nível de configuração elevado, mas sem ser exagerado e uma interactividade e velocidade de processamento elevadas, relativamente a outras aplicações semelhantes.
Relativamente ao modo de interagir com a aplicação, optou-se por apresentar apenas as secções com um grau de complexidade que justifique a exposição das mesmas.
Logo, as secções aqui apresentadas são as seguintes:

- Menu principal da aplicação.
- Opções de processamento da aplicação.
- Janela de interacção VTK.

VII.5.1. Menu principal da aplicação



Model →

New model
Cria um novo modelo em memória, sem pontos ou triângulos.

Close model
Remove o modelo activo da memória da aplicação.

Load from cloud file(s)…
Carrega, a partir de um ficheiro do tipo nuvem de pontos, os seus pontos para o modelo activo. Em caso de não existir qualquer modelo activo, cria um novo.
No caso de se seleccionar mais que um ficheiro, a aplicação carrega cada um dos modelos de pontos num modelo independente, em memória (atenção ao tamanho e número de ficheiros abertos).

Load from WML file(s)…
Carrega, a partir de um ficheiro do tipo modelo WML, os seus pontos ou triângulos para o modelo activo. Em caso de não existir qualquer modelo activo, cria um novo.
No caso de se seleccionar mais que um ficheiro, a aplicação carrega cada um dos modelos WML num modelo independente, em memória (atenção ao tamanho e número de ficheiros abertos).

Merge with cloud file…
Junta o modelo activo a um modelo aberto a partir de um ficheiro do tipo nuvem de pontos. Caso exista, carrega e aplica automaticamente a matriz de transformação linear da nuvem de pontos. Caso o modelo activo tenha triângulos, triangula o modelo de pontos de forma automática e, caso a opção esteja activada, faz uma junção dos triângulos dos dois modelos, quando detectar superfícies paralelas a uma distância não superior a um valor configurável.
Caso o modelo activo tenha pontos adiciona os pontos.

Merge with model…
Adiciona ao modelo activo os pontos do modelo a adicionar.
Adiciona, ou combina os triângulos do modelo a adicionar ao modelo activo, dependendo da opção de unir modelos triangulados.

Save to WML file…
Guarda o modelo de pontos ou o modelo de triângulos do modelo activo num dado ficheiro do tipo WML.

Clear points model
Remove/limpa o modelo de pontos do modelo activo (para libertar memória).

Clear triangle model
Remove/limpa o modelo de triângulos do modelo activo (para libertar memória).

Triangulate
Efectua a triangulação do modelo activo.

Simplify triangulation
Simplifica o modelo de triângulos do modelo activo.

Approach and register a model to this using... →

3 Point approach…
Mostra duas janelas, uma do modelo activo e outra de um modelo a aproximar, escolhido pelo utilizador, de forma a que este marque 3 pontos comuns aos dois modelos, de forma a que a aplicação efectue uma aproximação do modelo seleccionado ao modelo activo, confirmando a mesma, ao apresentar os dois modelos aproximados, antes de aplicar a transformação propriamente dita.

Final ICP approach…
Aplica o algoritmo de aproximação de modelos, de forma a aproximar um modelo seleccionado pelo utilizador ao modelo activo. Confirma a operação de aproximação, ao apresentar os dois modelos aproximados, antes de aplicar a transformação propriamente dita.

Remove zero points (from matrix format)
Identifica e remove os pontos do modelo activo, quando estes ainda se encontram no modelo de matriz (necessário à triangulação, logo este modelo ainda não foi triangulado) marcados como inválidos. Convém salientar que após esta operação a triangulação do modelo activo torna-se impossível.

Use triangulated model points as points model
Cria/modifica o modelo de pontos do modelo activo de forma a este utilizar os pontos do seu modelo triangulado.

Linear transformation matrix →

Undo transformation matrix
Relativamente ao modelo activo, desfaz todas as suas transformações lineares aplicadas e inicializa a variável das transformações lineares.

Load transformation matrix
Relativamente ao modelo activo, caso existam transformações lineares aplicadas, inverte-as. De seguida inicializa a variável das transformações lineares e carrega o ficheiro de transformações lineares associado ao ficheiro que deu origem ao modelo em questão.

Save transformation matrix
Relativamente ao modelo activo, caso a sua a variável de transformações lineares aplicadas não seja nula, guarda-a no ficheiro de transformações lineares associado ao ficheiro que deu origem ao modelo.

Nota: os ficheiro de transformação linear associados aos ficheiros que deram origem a um dado modelo, têm o mesmo nome, mas adiciona-se o sufixo ‘,matrix’. Estes ficheiros são do tipo texto, podendo ser editados a partir de um editor de texto comum.

Window →

New
Cria uma nova janela real (de visualização) sem modelos associados.

Close
Remove da memória a janela activa, sem apagar os seus modelos associados.

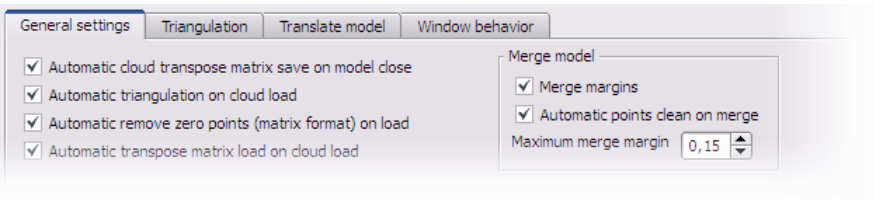
Add model
Adiciona o modelo activo à janela activa. Caso não haja janelas em memória, cria uma e adiciona-lhe o modelo activo.

Remove model
Caso exista, remove todas as entradas do modelo activo na janela activa.

View
Mostra a janela activa (reinicializa o campo de visão da mesma).

Hide
Esconde a janela activa (é similar a clicar no botão de fechar da própria janela:).

VII.5.2. Opções de processamento da aplicação



General settings:
Parâmetros gerais da aplicação.

Automatic cloud transpose matrix save on model close
Quando activado, a aplicação guarda a matriz de transformação de qualquer modelo, quando este é fechado, caso esta não seja nula.

Automatic triangulation on cloud load
Quando activado, efectua uma triangulação automática após carregamento com sucesso de um ficheiro do tipo nuvem de pontos.

Automatic remove zero points (matrix format) on cloud load
Quando activado, efectua uma limpeza automática dos pontos inválidos, após o carregamento e triangulação automática de uma nuvem de pontos.

Automatic matrix load and apply on model load
Quando activado, leva a que a matriz de transformações lineares associada à nuvem ou ficheiro WML seja carregada e aplicada ao modelo acabado de carregar.

Triangulation:
Parâmetros das operações de triangulação de modelos.

Type
Tipo de triangulação a ser efectuada pela aplicação, por omissão.

Edge detection (recommended)
Sempre que for requerida a operação de triangulação de um dado modelo, é invocado o algoritmo de triangulação com detecção de descontinuidades.

Delawanay
Sempre que for requerida a operação de triangulação de um dado modelo, é invocado o algoritmo de triangulação com formação de triângulos baseado no algoritmo Delawanay.

Triangulation - edge detection settings
Parâmetros da triangulação baseada na detecção de descontinuidades.

Maximum difference
Valor de distância máxima para um desnível de uma superfície ser considerada uma descontinuidade.

Maximum Distance
Distância máxima a que um dado ponto pode estar do emissor laser para ser permitida a sua triangulação.

Create edge image (not recomended)
Quando activada, esta opção leva a que seja criada uma imagem, num ficheiro com o nome '`contour.jpg`', com a marcação das descontinuidades detectadas na nuvem de pontos durante o processo de triangulação.

Triangulated model simplification
Parâmetros da operação de simplificação de um modelo triangulado.

Full simplification (recursive)
Quando activada, esta opção indica que a operação de simplificação de um modelo triangulado deve ser efectuada de forma repetitiva até não serem encontrados mais triângulos a simplificar.

Maximum normal vector coordinate variation
Valor máximo da soma dos erros das coordenadas que a normal de um dado triângulo aceita para considerar esse triângulo paralelo a uma dada superfície.

Approach model:
Parâmetros para aproximação de um modelo a outro.

3 Point approach by
Definição da operação de aproximação a partir de 3 pontos, marcados pelo utilizador, em ambos os modelos, que devem definir pontos comuns a ambos os modelos no espaço real.

Order of points
O cálculo da transformada linear, de aproximação de um modelo ao outro, tem em conta a ordem pela qual os pontos são dispostos no espaço (para ver a ordem dos pontos, ter em atenção a sua cor).

Triangle shape
O cálculo da transformada linear, de aproximação de um modelo ao outro, tem em conta a forma do triangulo definido pelos pontos dispostos no espaço (convém criar triângulos com lados de tamanhos diferentes de forma a não haver mais que uma possibilidade de encaixe dos mesmos).

ICP approach
Parâmetros do algoritmo de aproximação de dois modelos relativamente próximos, a partir dos seus pontos, de forma automática.

Points used
Número de pontos utilizados para aproximação dos dois modelos, pelo algoritmo ICP. Quanto maior for este valor, melhores resultados deverá ter o algoritmo, mas também mais tempo de processamento irá requerer. Também se pode designar estes pontos por pontos de encaixe.

Iterations
Número máximo de iterações que o algoritmo irá efectuar a fim de optimizar o encaixe dos dois modelos, a partir dos seus pontos de encaixe.

Join points with distance less than
A escolha dos pontos baseia-se no facto destes se encontrarem a uma distância máxima definida por este valor, do outro modelo. Este valor deve ser sempre maior que a distância relativa entre os modelos. Valores demasiado elevados levarão à escolha de pontos de encaixe pouco eficientes.

On approach, try project models like
A projecção dos dois modelos, tanto antes de aproximar, a fim de marcar os pontos (se for o algoritmo dos 3 pontos) como após a aproximação dos dois modelos, para verificar o encaixe correcto dos dois modelos, entre si, pode ser efectuada de duas formas.

Triangulated model
São utilizados os modelos de triângulos, para dar ao utilizador uma melhor percepção das superfícies (apenas quando os dois modelos a aproximar têm modelos de triângulos, caso contrário tenta-se a projecção do modelo de pontos).

Points model
São utilizados os modelos de pontos, para dar ao utilizador uma melhor percepção da profundidade dos modelos (apenas quando os dois modelos a aproximar têm modelos de pontos, caso contrário tenta-se a projecção do modelo de triângulos).

Nota: no caso de nenhuma das opções ser possível a operação é cancelada.

Merge model
Parâmetros de junção/união de modelos.

Merge triangulated models margins
Quando activado, instrui a aplicação a fazer uma junção de triângulos e superfícies sobrepostas durante a união de dois modelos com modelos de triângulos.

Maximum merge margin
Distancia máxima para que uma margem (modelo triangulado), ou um ponto (modelo de pontos), sejam unidos, durante a junção de modelos.

Window behaviour:
Parâmetros das operações do menu principal, submenu '*Window*'.

Try add models as
Ao adicionar um dado modelo activo a uma janela de visualização:

Points
Tentar adicionar o modelo de pontos, mas em caso deste não existir, ou não ter pontos, tentar adicionar o modelo de triângulos.

Triangles
Tentar adicionar o modelo de triângulos, mas em caso deste não existir, ou não ter triângulos, tentar adicionar o modelo de pontos.

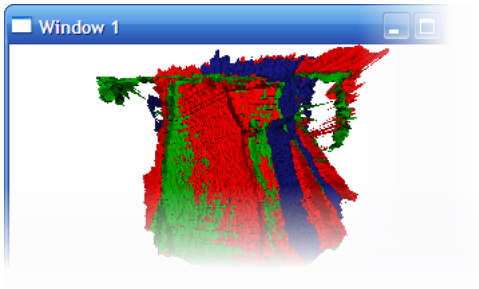
Nota: no caso de nenhuma das opções ser possível a operação é cancelada.

Default window size
Define o tamanho com que uma janela de visualização é criada.

X
Definição da largura da janela.

Y
Definição da altura da janela.

VII.5.3. Janela de interacção VTK



A projecção dos modelos é efectuada por uma janela de visualização, que por sua vez pode ou não estar a ser gerida por uma janela de visualização real.
Estas janelas de visualização recorrem ao VTK para criar janelas de visualização, comuns a esta *Framework*, com enormes capacidades de visualização e interacção com o utilizador, a seguir segue uma breve descrição dos vários comandos disponibilizados por estas janelas e adicionadas pela aplicação.

Alteração da posição da câmara, relativamente ao modelo

Devido à enorme quantidade de modos de interacção, disponível numa janela de visualização de um modelo, as operações de rotação, recorrendo ao rato e/ou teclado, optou-se pela utilização de figuras exemplificativas, de modo a uma fácil compreensão:

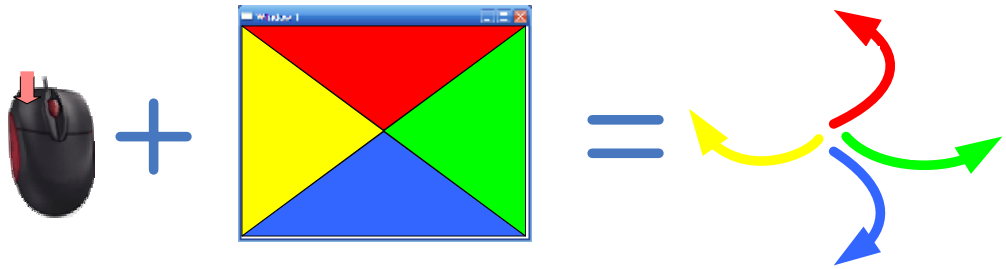


Figura VII.12 - Rotação básica

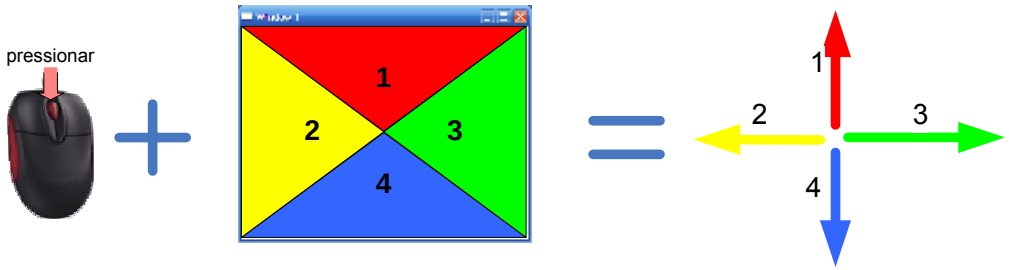


Figura VII.13 - Movimentação básica

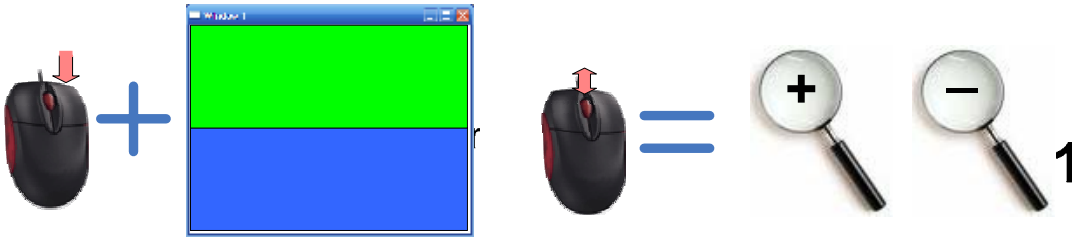


Figura VII.14 - Ampliar ou reduzir o ângulo de visão

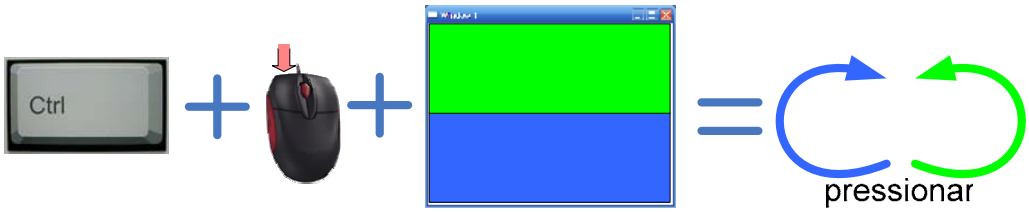


Figura VII.15 - Rotação no sentido horário ou contra-horário

Teclas de acção directa

F	Aproxima a posição da câmara para a zona do modelos onde se encontra o ponteiro do rato.
W	Mostra o modelo no formato de linhas, ou *wireframe*.
S	Mostra o modelo no formato sólido, ou normal.
	Repõe a câmara à distância em que esta se encontrava, quando a janela de visualização foi criada, com o modelo no centro.
	Marca o ponto em que se encontra o rato com uma esfera (apenas activo durante a marcação dos 3 pontos).
D	Remove o último ponto marcado com uma esfera - ver tecla 'M' (apenas activo durante a marcação dos 3 pontos).
P	Selecciona o modelo sobre o qual está o rato (actualmente a aplicação encontra-se a ignorar esta selecção).
Q	Fecha/esconde a janela.

R

M

VIII. Bibliografia

1. *(Wikipedia) Iterative Closest Point (ICP).* [cited; Available from: http://en.wikipedia.org/wiki/Iterative_Closest_Point.

IX. Índice de Figuras

Figura V.1 - Scanner 2D SICK LMS 200....14
Figura VI.1 - Visualização da nuvem de pontos como uma matriz de pontos....15
Figura VI.2 - Selecção da matriz de pontos, em quadrados de triangulação....17
Figura VI.3 - Opções de triangulação de um quadrado baseado nas distâncias dos pontos....17
Figura VI.4 - Linhas de pontos a eliminar da matriz de pontos....18
Figura VI.5 - Variação angular horizontal de um *scanner* laser....19
Figura VI.6 - Zonas permitidas e ilegais, baseadas na distância, para validação de pontos....20
Figura VI.7 - Projecção de pontos vs. projecção de triângulos (após triangulação)....21
Figura VI.8 - Aproximação de dois modelos VTK por translação e rotação....23
Figura VI.9 - Cálculo da transformada linear capaz de aproximar 2 conjuntos de 3 pontos....24
Figura VI.10 - Marcação de 3 pontos num modelo de pontos....25
Figura VI.11 - Marcação de 3 pontos num modelo de triângulos....26
Figura VI.12 - Selecção de dois subconjuntos de pontos de aproximação....27
Figura VI.13 - Selecção aleatória dos pontos a fim de todas as zonas terem igual peso....27
Figura VI.14 - Diagrama simplificado do algoritmo ICP....28
Figura VI.15 - Zonas capturadas vs. Zonas não capturadas....29
Figura VI.16 - Escolha correcta do *scanner* laser a fim de evitar zonas sem captura....29
Figura VI.17 Junção de triângulos....31
Figura VI.18 Modelos triangulados sobrepostos....32
Figura VI.19 - Junção de triângulos de dois modelos....32
Figura VI.20 - Modelos triangulados, unidos num modelo....33
Figura VI.21 - Algoritmo de simplificação de triangulação....35
Figura VI.22 - Exemplo de simplificação triangular....36
Figura VII.1 - Comunicação entre módulos....38
Figura VII.2 - Criação dos módulos principais....40
Figura VII.3 - Forma de interligação da classe renderWindow....48
Figura VII.4 - diagrama da classe windowManager....50
Figura VII.5 - Janela principal da aplicação....56
Figura VII.6 - Janela real (para projecção de modelos)....58
Figura VII.7 – Diálogo de processamento em curso....58
Figura VII.8 - Diálogo de confirmação de saída....58
Figura VII.9 - Diálogo de abertura de uma nuvem de pontos....59
Figura VII.10 - Janela virtual para marcação de 3 pontos de aproximação....59
Figura VII.11 - Diálogo de selecção de um modelo para aproximação por 3 pontos comuns....59
Figura VII.12 - Rotação básica....65
Figura VII.13 - Movimentação básica....65
Figura VII.14 - Ampliar ou reduzir o ângulo de visão....65
Figura VII.15 - Rotação no sentido horário ou contra-horário....66