Desenvolvimento de Humanos Virtuais para a Plataforma IViHumans

Ricardo Abreu
Ana Paula Cláudio
Maria Beatriz Carmo

DI–FCUL TR–07–32

Novembro de 2007

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Desenvolvimento de Humanos Virtuais para a Plataforma IViHumans

Ricardo Abreu Ana Paula Cláudio Maria Beatriz Carmo

Novembro de 2007

Conteúdo

1 Introdução

O uso de ambientes virtuais povoados por agentes com uma representação humanóide tem, hoje em dia, diversas aplicações práticas, como o entretenimento, a educação ou a psicoterapia. Estes são exemplos de áreas que podem extrair benefícios de uma plataforma de suporte à geração de cenas animadas com agentes inteligentes, concebida de modo a ser adaptável e proveitosa em diversos contextos.

A plataforma *IViHumans* [1, 2] é a concretização, ainda em desenvolvimento, de um projecto delineado pelo *LabMAg* [3] nesta perspectiva. O projecto foi desenhado de modo a integrar um conjunto de elementos suficientemente vasto para que atinja um grau de completude indispensável à sua aplicabilidade. A plataforma é composta por uma camada gráfica e por uma camada de inteligência artificial. A camada gráfica assenta sobre um motor de *rendering* – o *OGRE 1.4.5* [4] – e a camada de IA é sustentada por uma bancada de agentes – *JADE* [5]. As ferramentas de modelação 3D usadas são o *Poser 7* [6] e o *Blender 2.44* [7].

Alguns dos elementos essenciais da plataforma *IViHumans* prendem-se com a representação gráfica dos agentes e com as animações respectivas. Cada agente deve ter associado um modelo tridimensional, constituído por uma malha poligonal, para o qual estejam definidas animações que veiculem a sua expressão e actuação sobre o mundo, tanto ao nível da face como ao da totalidade do corpo. Idealmente, estas animações são reproduzidas de maneira a espelhar os objectivos, decisões e emoções do agente.

Numa primeira aproximação, decidimos criar apenas um protótipo de humano virtual (uma mulher), focando principalmente os aspectos essenciais à evolução global da plataforma. Por conseguinte, concentrámo-nos especialmente na coerência da geometria do modelo e nas suas animações e relegámos para segundo plano questões como a definição de materiais complexos e realistas ou o carácter deformável de componentes como o cabelo e a roupa. Estas são características que pretendemos aprofundar a curto ou médio prazo.

Nesta primeira fase, criámos uma animação simples para a acção de andar para a frente, pelo facto de ser indispensável à movimentação de um agente pela cena, de forma minimamente credível. De resto, num trabalho anterior, foi modelada uma face com um conjunto de expressões faciais básicas prontas a ser animadas [8]. Este conjunto engloba as expressões definidas por Paul Eckman [9] (alegria, tristeza, raiva, medo, nojo e surpresa) e contém ainda expressões que correspondem a variações localizadas dos traços faciais (por exemplo: elevação do canto direito da boca). Foi implementada ainda uma ferramenta que permite misturar estas expressões, através de uma interface gráfica que possibilita a criação rápida e intuitiva de expressões complexas.

A construção de um modelo humanóide realista e a definição das animações respectivas é, no entanto, uma tarefa morosa, laboriosa e que requer um elevado grau de habilidade artística. Por isso, é desejável encontrar métodos que possibilitem a extensão e reutilização de modelos já feitos. Este relatório pretende expor detalhadamente os processos que seguimos para a criação de um humano virtual, bem como as soluções que encontrámos para abreviar a sua construção. Uma descrição menos pormenorizada deste trabalho originou já um *short paper* apresentado no 15º

Encontro Português de Computação Gráfica [10].

2 Modelação do Corpo

A modelação de humanos virtuais credíveis é uma tarefa demorada. Considerando que a arte da criação, de raiz, de humanos virtuais, não faz parte dos nossos objectivos, decidimos atalhar o processo partindo de um modelo já existente que fomos sucessivamente refinando. Após uma breve análise dos recursos publicamente disponíveis, acabámos por escolher uma ferramenta comercial direccionada para a criação e animação de personagens virtuais – o *Poser*. A versão actual desta ferramenta (a versão 7) fornece, à partida, uma quantidade razoável de conteúdos necessários à tarefa em questão e possibilita a sua parametrização, adaptação e conjugação, ainda que de forma algo limitada. Este software permite, por exemplo, carregar personagens 3D predefinidas, parametrizar o comprimento das pernas ou a largura do tronco, alterar a sua posição, criar roupas e adaptá-las ao modelo e criar animações. Embora o *Poser* disponibilize personagens humanas completas com grande qualidade e detalhe, optámos por adaptar e melhorar um modelo de baixa resolução, devido às restrições inerentes ao processamento gráfico em tempo real, um requisito fundamental à utilização da plataforma *IViHumans*. Alterámos ligeiramente a aparência global do modelo, ainda no *Poser*, através de transformações de escala para diferentes partes do corpo.

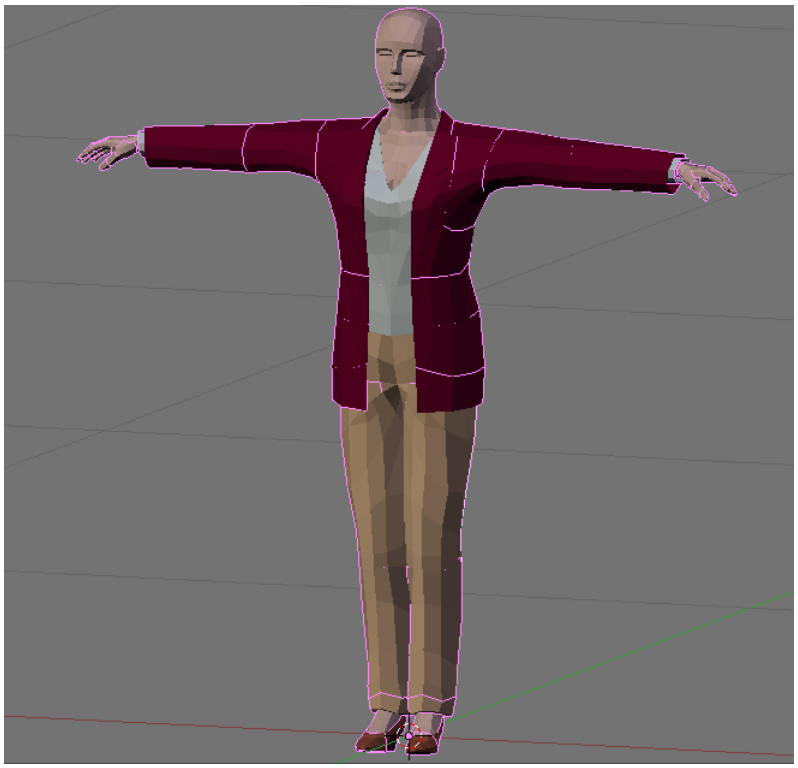

Figura 1: Visualização, no *Blender*, do modelo acabado de importar no *Poser*.

Partimos do modelo apresentado na Figura 1, exportámo-lo para o formato do *Wavefront* (extensão .obj) e importámo-lo no *Blender*. Numa primeira aproximação,

estabelecemos objectivos mínimos para o aspecto do humano e para o realismo do seu movimento. Durante a modelação, procurámos refinar o modelo, dando especial atenção ao melhoramento da geometria e da animação e relegando para segundo plano a definição de materiais e as reacções dinâmicas dos componentes que tencionamos tornar deformáveis. Embora tenhamos já definido, no *Blender*, materiais relativamente complexos para a pele e para os olhos, não existe processo de exportação destes para o *OGRE*. Por agora, reduzimos a definição dos materiais a cores e à aplicação pontual de imagens em texturas.

3 Criação de uma animação para o humanóide

Uma das técnicas mais frequentes para animar modelos tridimensionais, suportada tanto pelo *Poser* como pelo *Blender*, baseia-se na associação de um esqueleto à malha poligonal cujas animações se pretende produzir. De acordo com esta técnica, particularmente apropriada para a animação de personagens vertebradas, a representação abstracta de uma personagem é constituída por duas partes: a malha poligonal e o esqueleto. A malha poligonal é o conjunto estruturado de polígonos que dá forma ao corpo da personagem e que corresponde à imagem visualizada. O esqueleto, não visível no resultado do *rendering*, é constituído por um conjunto de ossos organizados numa ou mais estruturas em árvore.

A animação referida para a acção de andar foi também criada no *Poser*. Para esse efeito, usámos o esqueleto pré-definido para o modelo escolhido e definimos o seu movimento através da parametrização de um movimento base, recorrendo a um conjunto de funcionalidades que o *Poser* disponibiliza englobadas numa subferramenta designada *Walk Designer*. Os parâmetros manipuláveis através do *Walk Designer* podem ser classificados em dois níveis de abstracção, expostos em seguida.

- Atributos de alto nível que caracterizam o aspecto global do movimento. Estes prendem-se com o carácter que se pretende atribuir ao modelo. Exemplos de atributos representativos desta classe são *Cool*, *Power* ou *Sexy*.

- Características mais concretas, embora ainda de alto nível, que dizem respeito ao movimento de zonas chave do corpo. Estas não estão tão relacionadas com o carácter do modelo a animar, permitindo antes modelar especificidades de um movimento particular. Exemplos dos parâmetros respectivos são a distância dos braços ao tronco ou o comprimento de cada passo.

Ao tentar produzir a animação, constatámos que o esqueleto associado ao modelo do *Poser* não estava perfeito, havendo ossos com posições distorcidas relativamente à malha poligonal. Tentámos colmatar estas falhas, no *Poser*, o que não foi possível devido às limitadas capacidades deste software e a frequentes *bugs* com que nos deparámos no processo. Além disso, os movimentos criados para o esqueleto geravam interpenetrações entre o corpo e a roupa e mesmo entre diferentes partes da anatomia do modelo, como exemplifica a figura 2. Assim, as possibilidades de animações que poderiam ser coerentemente aplicadas ao modelo estavam limitadas à partida e a escolha teve de procurar o equilíbrio entre o realismo do movimento e a minimização de ambiguidades causadas na malha poligonal. Apesar de tudo, algumas

das interpenetrações puderam ser eliminadas procedendo no *Blender* à edição do modelo e à definição de uma ligação adequada entre o esqueleto e a malha. Estes processos são descritos mais à frente, após uma breve contextualização teórica.

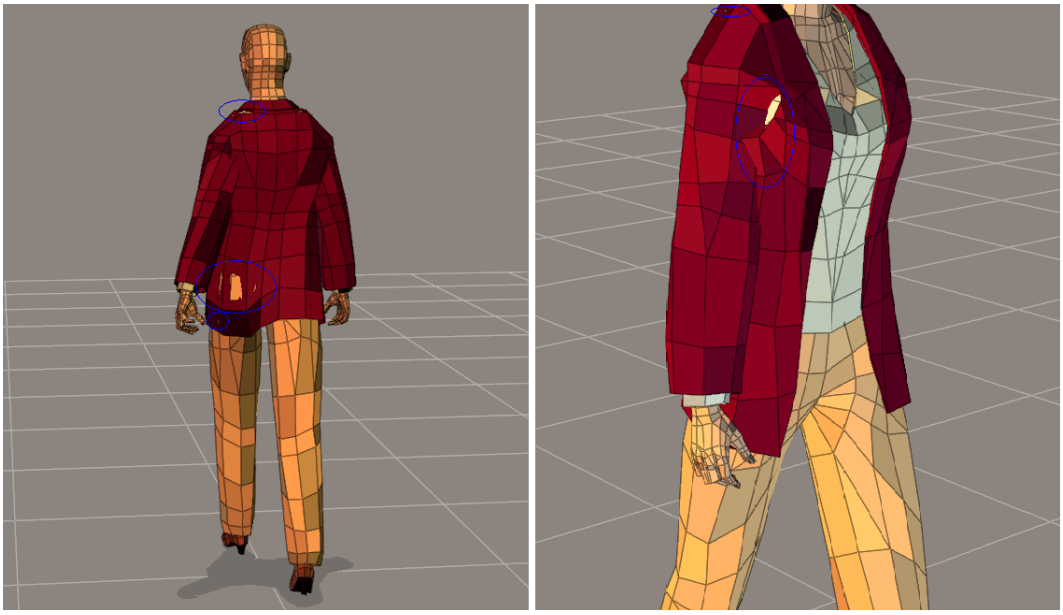

Figura 2: Captura do modelo do humanóide em posições, originadas pela animação de andar, que evidenciam interpenetrações.

3.1 O esqueleto e a sua animação

O sentido da organização hierárquica de um esqueleto é a herança, por parte dos ossos filhos, dos movimentos dos ossos pais. Um ou mais ossos constituem as raízes das árvores que integram o esqueleto e os restantes descendem, directa ou indirectamente deles. Como em qualquer estrutura baseada em árvores, todos os ossos têm um ascendente, excepto as raízes, e todos os ossos tem um ou mais descendentes, excepto as folhas. A animação do esqueleto é criada através da animação individual dos seus ossos, herdando cada osso os movimentos do seu ascendente. Por exemplo, num esqueleto simples cuja única raiz seja o osso da anca e cujos filhos directos sejam os ossos das pernas, estes últimos têm todos os movimentos do primeiro e, eventualmente, outros. Se se aplicar ao osso da anca uma translação de 10 unidades ao longo do eixo Ox e uma rotação de 90º em torno do eixo Oy, também os ossos das pernas sofrerão estes movimentos. Se cada perna tiver ainda um osso para o pé como filho, e se a cada uma for aplicada uma rotação adicional de -60º em torno do eixo Oy, os ossos dos pés herdarão a translação da anca e uma rotação de 30º em torno do eixo Oy. Este tipo de aproximação possibilita uma criação mais rápida de movimentos típicos de seres vertebrados, já que é uma boa concepção do modo como o seu esqueleto se comporta perante a ausência de forças exteriores (em particular, na ausência de gravidade).

Os movimentos de um conjunto de ossos, em si mesmos, isolados, não cumprem qualquer função, desde logo porque um esqueleto não tem uma representação gráfica

disponível na imagem final gerada. O seu propósito é apenas o de ser aplicado a um objecto veiculado por uma malha poligonal. Para que os movimentos de um esqueleto possam ter reflexos na malha é necessário associar o esqueleto e a malha. Isto faz-se associando cada vértice da malha a um ou mais ossos do esqueleto e caracterizando as ligações por um valor real, geralmente compreendido no intervalo $]0;1]$. Este valor pode ser visto como a intensidade ou como o peso com que o movimento do osso afecta o movimento do vértice. O deslocamento de cada vértice da malha é então função do peso com que está associado aos ossos e do movimento destes.

3.2 Conjugação das posturas do esqueleto e da malha poligonal

Como já foi referido, a animação do esqueleto foi construída no *Poser*. De todos os formatos reconhecidos pelo *Blender* para os quais o *Poser* permite exportar os seus conteúdos, nenhum contempla a associação, já existente por omissão no *Poser*, entre a malha poligonal e o esqueleto. Transferimos, portanto, o esqueleto para o *Blender*, separadamente da malha, através de um ficheiro intermédio no formato *motion capture* (extensão .bvh). Depois de alterado e refinado, no *Blender*, o modelo do humanóide, passámos à tarefa de o associar ao esqueleto. Como o resultado das duas exportações, a partir do *Poser*, não é coerente quanto a escalas, orientações e posições dos conteúdos exportados, o primeiro passo para a associação foi o ajuste relativo destas características entre a malha e o esqueleto. Além disso, a postura inicial do esqueleto não era absolutamente idêntica à do modelo, como se pode constatar pela Figura 3 (a diferença mais perceptível é o facto de o modelo ter as pernas juntas enquanto o esqueleto tem as pernas algo afastadas).

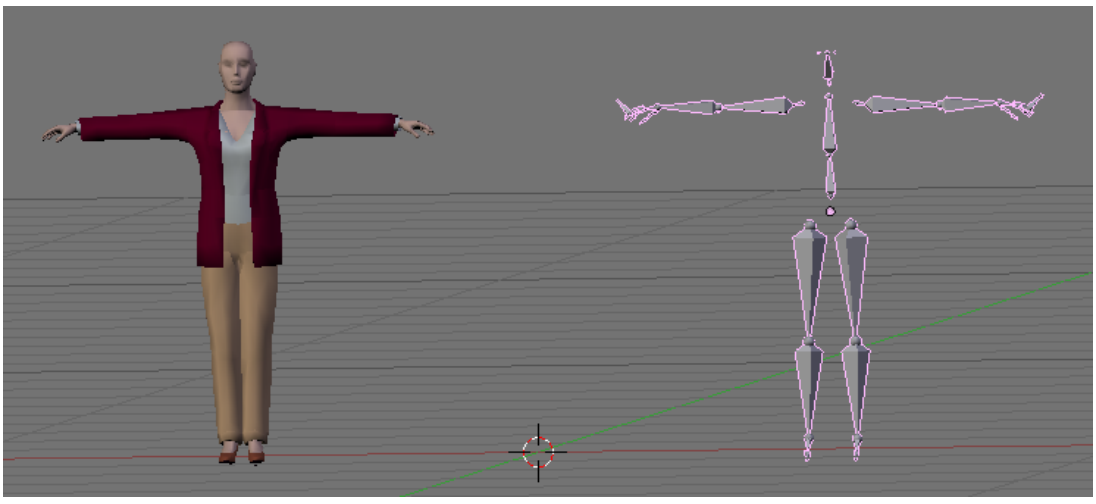

Figura 3: Imagem do modelo do humanóide e do seu esqueleto, no *Blender*, nas suas posturas iniciais, depois de uniformizadas as escalas e as orientações.

Para que os dois objectos estivessem na mesma postura, optámos por corrigir a postura do modelo humanóide. Embora a alternativa de corrigir a postura inicial do esqueleto pudesse aparentar ser mais fácil, julgamos que a opção que tomámos

foi a mais correcta. A justificação da nossa opção prende-se com a maneira como os movimentos dos ossos são registados internamente no *Blender*. Em vez de serem memorizadas as posições, orientações e escalas de cada osso, em cada *frame*, guardam-se, para um conjunto de *keyframes*, a sequência de deslocamentos, rotações e mudanças de escala aplicadas a cada osso, que transformam uma postura do esqueleto noutra postura. Ou seja, os movimentos são registados de um ponto de vista relativo e, quando a posição inicial de um osso varia, também as posições subsequentes da animação se alteram. Assim se percebe que a alteração da postura inicial do esqueleto, para que coincidisse com a do modelo, teria efeitos nefastos sobre a animação já criada.

Criámos então um esqueleto auxiliar, apenas composto por duas árvores, cada uma delas com apenas um osso (simultaneamente raiz e folha). Associámo-los aos vértices das pernas e dos pés, com uma intensidade de 1.0 e rodámo-los de maneira a que a postura do modelo fosse análoga à do esqueleto principal. Uma vez terminado este processo, eliminámos o esqueleto auxiliar e o modelo voltou a não estar associado a nenhum esqueleto.

3.3 Associação do esqueleto e da malha poligonal

Para completar a associação entre o esqueleto e a malha é necessário que cada vértice do modelo fique ligado a pelo menos um osso. No entanto, a associação não tem de ser feita vértice a vértice. Com efeito, é natural que um conjunto de vértices posicionados numa mesma zona fique associado aos mesmos ossos.

No *Blender*, a associação de um esqueleto a uma malha poligonal pode ser feita de várias formas. Aquela que seguimos é relativamente simples e consiste na criação de *vertex groups* que são automaticamente associados aos ossos, através da sua identificação. A título de exemplo, considere-se que o osso que se quer associar à cabeça recebe o nome de "Cabeça". Para que um conjunto de vértices seja automaticamente associado a esse osso basta afectar os seus componentes a um *vertex group* igualmente denominado de "Cabeça". A intensidade/peso das ligações pode ser especificada para o conjunto e/ou individualmente para cada vértice. Se se pretendesse que os vértices da base da cabeça fossem animados também de acordo com os movimentos do osso do tronco, então estes também deveriam ser atribuídos a um outro *vertex group*, que correspondesse a esse osso, e a intensidade da influência de cada um dos dois ossos sobre cada vértice a ambos afecto deveria ser ajustada cautelosamente, até que se tivesse obtido o efeito desejado.

Foi este o método que seguimos para ligar o modelo ao seu esqueleto, tendo jogado conjuntamente com *vertex groups*, com pesos de ligação e com a geometria do modelo por meio a afinar a animação produzida. Uma vez concluída convenientemente esta associação, obtivemos um humano virtual animado (Figura 4).

Resta referir que alguns dos ossos que constituiam originalmente o esqueleto no *Poser* foram eliminados por não serem necessários para os movimentos desejados. Não obstante, os ossos eliminados foram apenas "ossos-folha", pelo que a sua ausência não tem qualquer influência no movimento do que fica do esqueleto.

Figura 4: *Frame* da animação do humano virtual, vista no *Blender*.

4 Expressões Faciais

O relacionamento entre seres humanos assenta, em grande medida, na capacidade de interpretação dos sentimentos e emoções alheias. A interacção entre humanos é moldada pela integração de diversos estímulos no reconhecimento de padrões de comportamento, transmitidos, nomeadamente, por expressões faciais. Ao comunicar, captamos o conteúdo emocional expresso pelo interlocutor através da distinção de traços faciais gerais que nos permitem recolher um padrão semelhante a partir de diferentes caras. Na realidade, somos capazes de reconhecer as mesmas expressões faciais em representações mais ou menos fiéis à realidade, como banda desenhada, ou mundos tridimensionais virtuais.

Na plataforma *IViHumans* pretendemos incluir a expressão facial como forma de ilustração das emoções dos agentes que povoam o ambiente virtual. Com o objectivo de diversificar as emoções que podem ser expressas, criámos uma interface gráfica de auxílio à construção de expressões através da mistura de expressões base. Com este programa, designado *Faces* (Figura 5), podem definir-se novas expressões faciais à custa da variação de parâmetros relativos às expressões básicas, observando-se o efeito em tempo real e enriquecendo-se assim, em qualquer altura, a biblioteca de expressões subjacente.

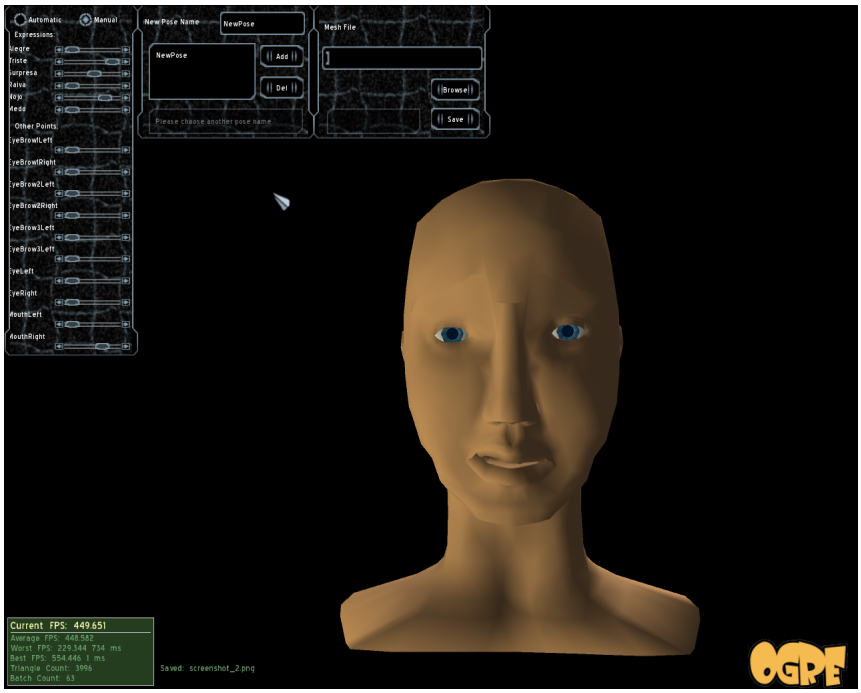

Figura 5: Interface do programa *Faces*.

Para criar uma biblioteca de expressões faciais foi necessário construir primeiro uma face, concebida de acordo com o que mais se aproxima de uma expressão neutra. Cada expressão básica foi então concebida no *Blender* por meio da alteração da malha original. Para cada expressão distinta são guardados os vectores de deslocamento, um para cada vértice cuja posição tenha sido alterada. O conjunto de vectores deslocamento que usamos para produzir uma expressão é frequentemente

denominado de pose. Aplicando os deslocamentos registados aos vértices correspondentes obtém-se a expressão básica respectiva. Este método pode ainda ser alargado de maneira a albergar a especificação de uma intensidade para cada expressão e, uma vez registadas as poses que codificam as expressões, podem assim gerar-se expressões intermédias. Seguindo este caminho, se se tiver definido, por exemplo, uma pose que codifique a expressão de surpresa, pode-se produzir muito facilmente uma expressão de "80% de surpresa" sem qualquer esforço adicional de modelação. Para isso, basta aplicar os deslocamentos resultantes da redução, em um quinto, dos vectores registados pela pose em questão.

No programa *Faces* este conceito é concretizado com recurso à API do *OGRE*. Através do simples manejamento de botões de *scroll*, um para cada expressão base definida, o utilizador instrui a aplicação para que sejam geradas expressões, com intensidades entre 0 e 1 correspondentes às posições dos *scrolls*. Quando se faz variar a posição de mais do que um botão de *scroll*, o resultado é a mistura das expressões deste modo activadas. A mistura de expressões resulta apenas da soma algébrica dos vectores subjacentes às poses correspondentes, ou seja, em rigor, o deslocamento aplicado a um vértice é o consequente da soma dos vectores de deslocamento, para cada pose activa, que lhe estão afectos, multiplicados pela intensidade especificada pela posição dos botões de *scroll*.

À medida que o utilizador faz variar as posições dos *scrolls*, os efeitos são gerados imediatamente, o que lhe confere a possibilidade de experimentar e aperfeiçoar as novas expressões que deseje criar.

5 Exportação e Integração do Humano Virtual na Plataforma *IViHumans*

Antes de associar a malha poligonal do humano virtual completo ao esqueleto, foi necessário concluí-la. Houve, nomeadamente, que proceder à adaptação mútua da face, que já tinha sido concebida, e do corpo, resultante das alterações efectuadas sobre o modelo original do *Poser*. Foi nesta etapa que surgiu um dos maiores contratempos com que nos defrontámos e que consiste na indisponibilidade, no *Blender*, de um método de junção de objectos em que pelo menos um deles tenha poses definidas. Esta funcionalidade seria da maior importância para a conclusão do modelo do humano virtual e a sua inexistência representou a necessidade de despender um tempo adicional para contornar o problema.

Uma vez que o *Blender* dispõe de uma interface de *scripting*, através de uma API em *Python*, pesquisámos por *scripts* que alguém com um problema análogo tivesse já desenvolvido e que nos pudessem ajudar a ultrapassar este obstáculo. Não tendo encontrado nenhum que nos pudesse auxiliar nesta questão, considerámos ainda a hipótese de o criar, mas acabámos por a deixar de lado, por constituir um desvio excessivo face ao caminho inicialmente pensado, devido à complexidade inerente e por oferecer menos garantias de sucesso.

Para superar este impedimento dividimos a resolução do problema em três passos de complexidade crescente:

1. Posicionar e ajustar a face e o corpo de modo a juntá-los num único objecto.

Como já foi referido, neste caso não se juntam as poses e, por isso, a face tem apenas a expressão base, mantendo-se a animação do esqueleto.

2. Posicionar e ajustar a face e o corpo de modo a que pudessem ser vistos como apenas um objecto, sendo embora objectos distintos, alcançando assim uma solução de compromisso com algumas imperfeições ao nível do realismo. Deste modo mantêm-se as expressões faciais mas é necessário encapsular os dois objectos numa entidade abstracta, aquando da sua integração na plataforma *IViHumans*.

3. Realizar um procedimento que junte de uma forma coerente, num único objecto, a face e o corpo, mantendo as poses.

A execução dos passos anteriores foi conduzida pela preocupação de produzir modelos que possam vir a ser usados no *OGRE*. Ao longo daqueles procedimentos fomos desenvolvendo pequenos programas com recurso à API do *OGRE*, de modo a testar os resultados e a aperfeiçoá-los de acordo com o que então se revelasse conveniente. A solução do passo 2 seria adequada, ao ponto de possibilitar a criação de novas personagens apenas pela troca de caras (num estilo *plugin*) não fora o facto de o algoritmo de *shading* usar a direcção das normais às faces adjacentes para realizar uma interpolação e colorir cada *pixel* da face considerada. Devido à estratégia seguida pelo algoritmo de *shading* que possibilita um efeito de *smoothing*, num humano virtual construído com recurso a malhas poligonais distintas, são visíveis descontinuidades nas zonas de contacto entre a cara e o corpo. Embora a falha entre os dois sub-objectos obtidos por este processo não seja excessivamente vincada, considerámos que os resultados limitariam o realismo que poderia, de outro modo, ser alcançado e decidimos, portanto, procurar um processo alternativo para atingir o nosso objectivo (que corresponde ao passo 3).

Para os nossos propósitos, a criação de humanos virtuais tem por fim a sua integração na plataforma *IViHumans*, construída sobre a API do *OGRE*. A transferência de modelos do *Blender* para o *OGRE* implica a realização de várias operações. Há um conjunto de operações básicas que são complementadas com operações adicionais para atingir o resultado final desejado.

5.1 Processo geral de transferência do modelo do *Blender* para o *OGRE*

Para que as animações associadas ao esqueleto estejam disponíveis, é necessário que este seja exportado do *Blender* conjuntamente com o modelo a que se aplica. Tanto para a malha poligonal do humanóide como para o seu esqueleto é assim produzido um ficheiro XML que deve ainda ser processado previamente por uma ferramenta distribuída em conjunto com o *OGRE* – o *OGREXMLConverter*. Esta ferramenta, por seu turno, gera novos ficheiros num formato binário, rapidamente processáveis pela plataforma (através da API do *OGRE*), em função do conteúdo dos ficheiros XML fornecidos como input.

Os ficheiros XML resultam da execução de um *script* de exportação que comunica com o *Blender* através da sua interface *Python* e que obtém a informação

necessária. No processamento efectuado por este *script* inclui-se um conjunto de transformações indispensáveis que lidam com a diferença de referenciais usados pelo Blender e pelo OGRE. A Figura 6 mostra um esquema que ilustra o processamento base de exportação.

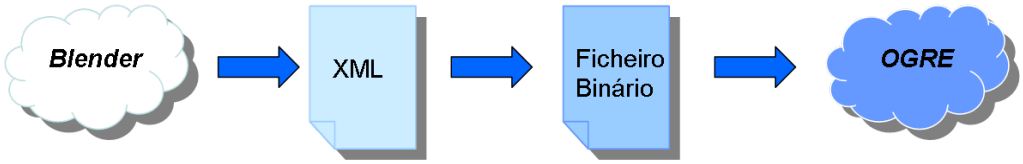

Figura 6: Processo de transferência de um modelo do *Blender* para o *OGRE*.

5.1.1 Conteúdo dos ficheiros XML

O conteúdo de um ficheiro XML é variado e depende do objecto que codifica. No caso de uma malha poligonal (*mesh*), a geometria, as poses e as animações são os principais atributos registados no ficheiro XML. O ficheiro é estruturado através da subdivisão da *mesh* global em *submeshes*, de acordo com os materiais aplicados. Assim sendo, cada zona da malha que tem um material diferente é registada no ficheiro XML como uma sub-malha separada das restantes. Para cada sub-malha são primeiro definidas as faces através de listas de identificadores dos vértices que as compõem. A informação que define completamente estes vértices surge num bloco posterior do ficheiro. O número de ordem da declaração de cada vértice neste bloco é o seu identificador. Embora no *Blender* a malha possa ser definida à custa de quadriláteros, o *OGRE* apenas aceita malhas triangulares. Por esta razão, cada face que no *Blender* corresponda a um quadrilátero é dividida em dois triângulos no processo de geração do XML, pelo que a definição do modelo no ficheiro contém mais faces do que no *Blender*.

À declaração das faces segue-se a declaração dos vértices que as compõem. Cada vértice é definido através das coordenadas da sua posição, relativamente a um referencial global, e das coordenadas de um vector normalizado que corresponde à sua normal. Outras características menos importantes são ainda registadas para cada vértice.

Um atributo relevante é especificado de seguida e corresponde à associação entre os vértices da sub-malha em questão e os ossos a que estão ligados. Os ossos estão definidos num ficheiro XML próprio e são aqui apenas referidos através do seu identificador (mais uma vez, correspondente à posição da sua declaração).

As poses são definidas no ficheiro XML de acordo com a definição antes avançada, ou seja, como conjuntos de vectores deslocamento. A declaração de cada deslocamento é constituída pelas coordenadas do vector correspondente e pelo identificador do vértice a que se aplica. Por seu lado, a pose, na sua globalidade, é também indexada à *submesh* que modifica. As animações das poses são definidas à custa de *keyframes*. A cada *keyframe* é atribuído um instante temporal e uma intensidade para cada pose associada.

As animações construídas com recurso ao esqueleto (*skeletal animations*) são definidas no ficheiro XML que contém toda a definição do esqueleto. Neste ficheiro

é primeiro especificado o conjunto de ossos que definem o esqueleto. Em seguida vem especificada a hierarquia dos ossos, através de relações binárias do tipo "x é filho de y", e finalmente aparece o registo das animações, construído de forma análoga ao que acima se descreve para as animações das poses, mas com um conjunto de *keyframes* a ser definido para cada osso e com o papel das poses substituido por transformações de translação, de rotação e de escala.

A especificação completa dos ficheiros XML da malha poligonal e do esqueleto é anexada a este relatório na forma de documentos *Document Type Definition – DTD*.

5.2 Processamento complementar do modelo exportado - passos 1 e 2

Durante os processos envolvidos na modelação, brevemente descritos anteriormente, não nos apercebemos de uma questão de extrema importância: a informação espacial a que o *script* de exportação acede não contempla transformações de escala, rotações ou translações que tenham sido aplicadas ao modelo como um todo, sem que explicitamente se tenha dado a instrução de alargar os seus efeitos directamente às primitivas que o compõem. Na realidade, o *Blender* separa a informação das características da malha poligonal, como sejam as posições dos seus vértices ou os deslocamentos que codificam as suas poses, da informação que é efectivamente usada para o *rendering*. Com efeito, uma qualquer transformação aplicada à totalidade de um objecto não tem influência sobre, por exemplo, as posições registadas para cada um dos vértices. Em vez disso, estas transformações são tidas como ulteriores à modelação *per se* e guardadas independentemente. Os seus resultados são só calculados durante a geração da imagem (correspondente ao ambiente do *Blender*), sendo então guardados em *buffers* cuja persistência é volátil. Por conseguinte, é natural que o resultado da exportação de um modelo apresente uma posição, uma dimensão e uma orientação incongruentes com o que é visualizado no *Blender*. Este problema fez-se sentir, naturalmente, também no caso do nosso modelo, a cuja totalidade aplicámos transformações diversas, nomeadamente aquando do ajuste necessário à associação com o esqueleto.

Apesar de o *Blender* disponibilizar um meio de estender os efeitos das referidas transformações directamente à definição da malha poligonal, esta funcionalidade nem sempre opera correctamente. Em particular, no caso do nosso modelo, o seu uso tem efeitos nefastos sobre o sentido das normais a uma grande parte dos milhares de faces que o constituem. Por esta razão, e porque a alteração manual de milhares de normais é impraticável, fomos obrigados a abandonar este método.

Mais uma vez, teria sido possível resolver este problema através de um outro *script* que modificasse correctamente todos os aspectos que compõem a malha poligonal. Não encontrámos, no entanto, nenhum *script* já desenvolvido que executasse eficazmente esta operação e optámos, mais uma vez, por não o criar, pela mesma razão que acima referimos. Além disso, as modificações que tínhamos, de qualquer modo, de levar a cabo sobre a malha podiam ser realizadas ao nível do XML.

Criámos então um programa *Java* que processa os ficheiros XML produzidos e que os altera em função de parâmetros que permitem especificar as transformações geométricas a realizar. Este programa contempla ainda as consequências da operação

sobre um referencial diferente do usado no *Blender* e realiza as transformações adicionais que este facto acarreta, para que o modelo processado tenha uma aparência análoga à que é observada no *Blender*. O programa representa pontos e vectores com coordenadas homogéneas e as transformações 3D através de matrizes 4×4. A sequência de transformações desejada, introduzida como input, é processada através da construção de uma matriz para cada transformação pretendida. As matrizes individuais são então concatenadas numa única matriz que representa a transformação global. Deste modo, basta multiplicar cada ponto ou vector por uma matriz que condensa a sequência de transformações, em vez de ter de se efectuar, para cada ponto ou vector, tantas multiplicações quanto o número de transformações individuais [11]. Sendo n o número de vértices a transformar e m o número de transformações a aplicar, a complexidade temporal deste método fica assim reduzida de $O(nm)$ para $O(n+m)$. O programa prossegue processando elemento a elemento o ficheiro XML, aplicando a transformação global em todos os casos pertinentes (vértices, normais, deslocamentos, ...) e escrevendo simultaneamente um novo ficheiro resultante das transformações aplicadas sobre o original.

O programa funciona quer o input seja o ficheiro que contém a definição da *mesh*, quer seja o que contém a definição do esqueleto. Quando é executado sobre os ficheiros XML criados imediatamente após a exportação, com parâmetros correspondentes às transformações aplicadas no *Blender* sobre os objectos respectivos, conseguimos gerar novos ficheiros que correspondem ao que é visualizado no programa de modelação, a menos de pequenas imperfeições que surgem como consequência de propagação dos erros resultantes da baixa precisão com que o *Blender* disponibiliza determinados dados numéricos – apenas 4 dígitos significativos. No entanto, estes detalhes são imperceptíveis para qualquer *zoom* razoável.

Através deste programa, obtivemos um modelo plenamente coerente e pronto a ser integrado na plataforma *IViHumans*. Este modelo perdeu, no entanto, as expressões faciais. O passo 1 apresentado na secção anterior ficou assim completo.

Para realizar o passo 2, ajustámos as duas malhas no *Blender* mas não as juntámos numa única malha, o que nos possibilitou manter as poses associadas à cara. Pelo mesmo processo de exportação são produzidos dois ficheiros XML, um para a face e outro para o corpo (além do ficheiro do esqueleto). Ambos os ficheiros sofreram os processamentos complementares descritos anteriormente. Ao nível do OGRE, as duas malhas são integradas numa única entidade abstracta. O seu encapsulamento garante que as transformações aplicadas ao objecto sejam reflectidas nas duas malhas poligonais que, estando ajustadas e alinhadas à partida, são vistas como uma unidade, excepto no que concerne à descontinuidade induzida pelo algoritmo de *shading*, tal como foi antes explicado.

5.3 Processamento complementar do modelo exportado - Passo 3

Para alcançar os objectivos do passo 3, o processamento envolvido ao nível do tratamento dos ficheiros XML foi mais complexo. A partir dos ficheiros XML obtidos com o passo 2 para a face e para o corpo é criado um único ficheiro, através de uma sequência de operações que se expõe a seguir.

Tal como foi exposto no ponto 5.1.1, um ficheiro XML que conserve a definição de uma malha poligonal tem uma estrutura que compreende a divisão da malha em sub-malhas, de acordo com os materiais associados a cada região. As várias sub-malhas são referenciadas em diferentes zonas do ficheiro através de índices que se prendem com a ordem da sua declaração, o que se passa também para grande parte das restantes propriedades. Foi portanto fácil adicionar as sub-malhas dos olhos, dos lábios, dos dentes e do cabelo à malha do corpo. Dado que os seus materiais são diversos de qualquer dos materiais do corpo, bastou copiar as secções correspondentes entre os dois ficheiros XML (do da face para o do corpo), corrigir as referências às sub-malhas e acrescentar as declarações necessárias para associar os novos vértices ao osso da cabeça, tendo cada uma das novas associações um peso de 100%. Como, nestas declarações, a única coisa que varia é o índice do vértice referido, a sua produção (através de outro programa *Java*) e a sua adição ao ficheiro foram operações triviais.

A etapa que levantou mais problemas foi a da adição, ao modelo do corpo, da sub-malha da face, coberta pelo material concebido para a pele. Sendo este o mesmo material já usado por uma das sub-malhas do corpo, e uma vez que o algoritmo de *shading* deveria encarar a ligação dos polígonos adjacentes da face e do corpo como contínua, não seria suficiente seguir o mesmo procedimento que para as restantes submalhas da face. Foi, antes, imprescindível fundir as sub-malhas da pele, dos lados do corpo e da cara, numa única. Além da substituição de referências e da associação dos vértices da face ao esqueleto, tivemos de conferir um carácter de continuidade aos limites adjacentes da face e do corpo, através da "fusão" dos vértices fronteiriços, de forma a que a imagem gerada não evidenciasse uma falha entre os polígonos. A Figura 7 ilustra esta situação através de um caso análogo. Nela estão assinalados os pares de vértices que deveriam ser unificados.

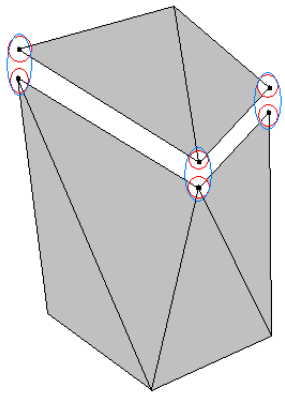

Figura 7: Ilustração de duas submalhas simples antes de serem integradas numa única. Os vértices limite estão assinalados a vermelho e os pares de vértices a unificar estão assinalados a azul. A figura não pretende ser rigorosa e o facto de os vértices não estarem co-posicionados justifica-se com uma mais fácil percepção do que se pretende transmitir.

Para atingir este propósito era forçoso identificar os vértices limite. Como o *Blender* não permite aplicar materiais a vértices de modo independente das faces que os compõem, nem exportar para XML malhas poligonais que contenham apenas vértices e arestas mas que não englobem nenhum polígono, tivemos de arranjar

uma alternativa à simples exportação de uma malha em que os vértices a identificar fossem "coloridos" com um material diferente, ou em que fossem isolados na constituição da malha, através da exclusão dos restantes vértices. Decidimos então criar um *script* elementar que tem como única função imprimir para um ficheiro as coordenadas dos vértices que compõem um modelo. Pensamos que esta opção se justificou devido à sua simplicidade, que contrasta com a dificuldade das hipóteses de desenvolvimento de *scripts* anteriormente apresentadas. Eliminando, das duas malhas, todos os vértices excepto os que queríamos identificar nos ficheiros XML e executando o *script*, alcança-se agora a informação pretendida.

Dispondo das coordenadas dos vértices particulares que queríamos identificar, entre milhares de outros vértices, pudemos criar outro programa *Java* para efectuar esta identificação. A execução deste programa passa por percorrer uma sub-malha, num ficheiro XML, calculando a distância entre cada vértice aí definido e cada um dos vértices constantes de um outro ficheiro, fornecido como input, que contenha as coordenadas dos vértices que se pretenda identificar. Consideram-se quaisquer dois vértices como idênticos quando a distância euclidiana entre eles é inferior a um dado limiar. O programa mantém um contador que indica a posição da declaração de cada vértice no ficheiro XML. Assim, quando um vértice é encontrado, sabe-se o seu identificador numérico (que não é mais do que a posição em que ocorre).

Não obstante, quando se executa o programa verifica-se que, normalmente, são identificados vários vértices no ficheiro XML, para cada vértice impresso a partir do *Blender*, independentemente do limiar de distância escolhido. De facto, geralmente, o *script* de exportação regista vários vértices duplicados no lugar de cada vértice existente no modelo a exportar. Estes vértices duplicados têm a mesma posição que o vértice original mas as suas normais variam. Embora a razão intrínseca deste funcionamento por parte do *script* nos escape, julgamos que corresponde, provavelmente, a uma necessidade causada pela diferença da representação das normais às faces no *Blender* e no *OGRE* ou pela diferença de funcionamento dos algoritmos de *shading*. Possivelmente, os algoritmos de *shading* do *Blender* usam explicitamente as normais às faces enquanto os do *OGRE* usam as normais aos vértices que as compõem, sendo necessário considerar diferentes normais para cada vértice, consoante a face processada. Apesar disto, tentámos contornar este obstáculo partindo do princípio de que o uso de cerca de três dezenas de vértices com normais comuns, em detrimento da inclusão dos seus duplicados, não teria consequências perceptíveis na visualização de um modelo constituido por milhares de vértices. Uma vez concluída a fusão, viemos efectivamente a verificar que esta suposição estava correcta e que a continuidade do modelo era indiscutível.

Para cada vértice nos limites entre o corpo e a cara (cujas coordenadas imprimimos através do referido *script*), escolhemos um dos duplicados existentes no ficheiro XML e substituímos todas as referências aos restantes duplicados por referências ao vértice escolhido. Por exemplo, suponha-se que a um dos vértices impressos a partir do modelo correspondiam 3 vértices coincidentes (a, b e c) no ficheiro XML e que cada um dos duplicados era usado na especificação de um polígono diferente da malha (polígonos A, B e C). De acordo com o processamento descrito, um qualquer destes vértices seria escolhido para substituir os outros. Suponha-se que o vértice escolhido era o a. As especificações das faces A, B e C passariam a referir todas o

Figura 8: *Screenshots* do humano virtual com animação de expressões, no ambiente *OGRE*.

vértice a. As declarações dos vértices b e c seriam mantidas mas estes tornar-se-iam vértices *dummy*, ou seja, vértices cuja existência em nada afectaria o *rendering* do modelo, uma vez que já não seriam usados na definição de qualquer face. Esta tarefa foi novamente executada por um programa *Java* que desenvolvemos e, no fim da sua execução sobrou um número igual de vértices limite do lado da cara e do corpo. Estes vértices puderam então ser emparelhados univocamente.

Conhecendo o emparelhamento próprio entre os vértices que definiam a fronteira entre a cara e o corpo, modificámos as faces do lado do corpo, de maneira a que fossem construídas com base nos vértices limite da face. Pudemos então adicionar as expressões da cara ao ficheiro XML, corrigindo os índices das *submeshes* referidas e, para a região da pele, alterando também os identificadores dos vértices a deslocar. Esta última edição resumiu-se à soma do número de vértices da pele, do lado corpo, ao índice de cada vértice da face usado para a definição das poses. Isto foi mais uma vez feito através de um pequeno programa *Java*. A Figura 8 mostra *screenshots* do humano virtual com animação de expressões, no ambiente *OGRE*.

6 Conclusões e Trabalho Futuro

Este relatório pretende descrever o método seguido para a concretização de um subobjectivo da criação da plataforma *IViHumans*. Através dos procedimentos acima descritos, cumprimos a finalidade de incluir o modelo gráfico de um humano virtual na biblioteca que estamos a desenvolver. Para que tal fosse possível, foi necessário obter uma representação coerente do modelo construído no *Poser* e no *Blender*. Além da tradução do modelo de um formato para o outro, a sua transferência foi enriquecida com os processamentos complementares que possibilitaram obter um humanóide com as características desejadas, designadamente com expressões faciais e com um movimento para andar, algo que não tinha sido alcançado com o *Blender*.

Exceptuando o *Poser*, os *softwares* usados para a concretização da plataforma *IViHumans* foram o resultado de uma decisão elaborada aquando da concepção abstracta da plataforma. A sua escolha obedeceu a um conjunto de critérios bem definidos – como quais as capacidades oferecidas, o custo ou a existência e dimensão de uma comunidade activa de utilizadores – e está documentada em [12].

No caso particular do *Blender*, apesar de ser uma ferramenta bastante completa e eficaz, padece da falta de um método de junção de dois objectos, quando estão definidas poses para pelo menos um deles, o que originou dificuldades acrescidas. Sendo certo que o problema poderia ser contornado pelo uso de um software alternativo para a modelação gráfica, essa aproximação exigiria muito tempo de adaptação e seria uma potencial fonte de novos problemas.

As mesmas dificuldades poderiam ter sido superadas eliminando as poses, antes da junção, e redefinindo-as, quando toda a geometria estivesse completa. Uma tal redefinição poderia ser, em princípio, manual ou automática. A redefinição manual das expressões foi posta de lado, uma vez que implicaria a repetição de um esforço já despendido e que envolveria um gasto excessivo de tempo. Quanto à alternativa da redefinição automática, esta poderia ser efectuada com recurso a algum procedimento já experimentado ou através de um método novo. Depois de um período de investigação e procura por métodos já concebidos para resolver este problema,

concluímos que a única alternativa documentada consistia no uso de um *script* embebido no *Blender* que permite a criação de uma pose para um objecto, com base na deformação de um duplicado desse mesmo objecto. Contudo, o funcionamento deste *script* não é perfeito, sendo o alcance do seu domínio limitado, pelo que pudemos verificar, a objectos simples, constituídos por um pequeno número de polígonos. De facto, a sua aplicação ao nosso problema concreto revelou-se totalmente infrutífera, dando origem unicamente a desfigurações claras que inviabilizaram o seu uso. Por conseguinte, vimo-nos forçados a criar e seguir um caminho alternativo, correspondente ao que aqui é exposto.

Os procedimentos envolvidos possibilitaram-nos cumprir os objectivos autopropostos e deram origem a programas auxiliares que podemos usar se nos voltarmos a deparar com este problema, ficando o tempo despendido na sua resolução largamente reduzido. Julgamos que a contribuição deste trabalho passa pela divulgação de uma experiência pessoal e pela descrição de um processo para a resolução de um problema que, embora particular, poderá ocorrer com alguma frequência.

Com o modelo do nosso humanóide completo, de forma coerente e funcional, perspectivamos, para um futuro próximo, associar à sua representação gráfica a capacidade de percepção do mundo. A percepção do ambiente será realizada através de um algoritmo de visão sintética. Este algoritmo permite detectar quais os objectos visíveis por cada agente, num determinado instante ou durante um curto intervalo de tempo. Através da camada de inteligência artificial, concebida de um ponto de vista abrangente, cada agente será capaz de planear trajectos e de exprimir emoções adequadas às circunstâncias percepcionadas.

Uma vez concluída esta primeira iteração, apresentam-se-nos várias ideias para um aprofundamento posterior. Destas, destacamos como incontornáveis o refinamento de materiais e a adição de texturas, assim como a concepção de mais movimentos pré-definidos, de modo a construir um leque suficientemente abrangente para que os agentes possam operar realisticamente sobre o mundo virtual através de um conjunto adequado de acções compostas. Consideramos também imprescindível a introdução de novas personagens e contamos com uma maior rapidez na sua criação, já que os métodos serão análogos aos já conhecidos.

O esquema de prioridades para as tarefas que desejamos levar a cabo é determinado por uma ideia central para a nossa aproximação ao desenvolvimento da plataforma *IViHumans*: o ideal é estabelecer um fio condutor que una a camada de computação gráfica à camada de inteligência artificial, sendo este fio posteriormente alargado e consolidado, de modo iterativo, até se tornar numa ponte plenamente consistente.

7 ANEXO

7.1 *Document Type Definition* para uma malha poligonal

```
<!ELEMENT mesh (sharedgeometry?, submeshes,
    skeletonlink?, boneassignments?, levelofdetail?, submeshnames?,
    poses?, animations?, extremes?)>
<!ELEMENT sharedgeometry (vertexbuffer+)>
<!ATTLIST sharedgeometry
    vertexcount CDATA #REQUIRED>
<!ELEMENT submeshes (submesh+)>
<!ELEMENT submesh (textures?,faces,geometry?,boneassignments?)>
<!ATTLIST submesh
    material            CDATA           #REQUIRED
    usesharedvertices   (true|false)    "true"
    use32bitindexes     (true|false)    "false"
    operationtype       (triangle_list|triangle_strip|triangle_fan)
    "triangle_list">
<!ELEMENT textures (texture+)>
<!ELEMENT texture EMPTY>
<!ATTLIST texture
    alias   CDATA   #REQUIRED
    name    CDATA   #REQUIRED>
<!ELEMENT faces (face+)>
<!ATTLIST faces
    count       CDATA   #IMPLIED>
<!ELEMENT face EMPTY>
<!-- Do not need all 3 vertex indexes if triangle_strip or
    triangle_fan since every face after the first one is defined by
    a single vertex -->
<!ATTLIST face
    v1      CDATA   #REQUIRED
    v2      CDATA   #IMPLIED
    v3      CDATA   #IMPLIED>
<!ELEMENT geometry (vertexbuffer+)>
<!ATTLIST geometry
    vertexcount     CDATA       #IMPLIED >
<!ELEMENT skeletonlink EMPTY>
<!ATTLIST skeletonlink
    name    CDATA   #REQUIRED>
<!ELEMENT boneassignments (vertexboneassignment+)>
<!ELEMENT vertexboneassignment EMPTY>
<!ATTLIST vertexboneassignment
    vertexindex     CDATA   #REQUIRED
    boneindex       CDATA   #REQUIRED
    weight          CDATA   "1.0">
```

```
<!ELEMENT levelofdetail ( (lodmanual|lodgenerated)+ )>
<!ATTLIST levelofdetail
    numlevels       CDATA           #REQUIRED
    manual          (true|false)    "false">
<!ELEMENT lodmanual EMPTY>
<!ATTLIST lodmanual
    fromdepthsquared    CDATA   #REQUIRED
    meshname            CDATA   #REQUIRED>
<!ELEMENT lodgenerated (lodfacelist)>
<!ATTLIST lodgenerated
    fromdepthsquared    CDATA   #REQUIRED
    meshname            CDATA   #REQUIRED>
<!ELEMENT lodfacelist (face)+>
<!ATTLIST lodfacelist
    submeshindex        CDATA   #REQUIRED
    numfaces            CDATA   #REQUIRED>

<!ELEMENT vertexbuffer (vertex+)>
<!ATTLIST vertexbuffer
    positions                   (true|false)    "false"
    normals                     (true|false)    "false"
    colours_diffuse             (true|false)    "false"
    colours_specular            (true|false)    "false"
    texture_coords              (0|1|2|3|4|5|6|7|8)     "0"
    texture_coord_dimensions_0  (0|1|2|3)           "2"
    texture_coord_dimensions_1  (0|1|2|3)           "2"
    texture_coord_dimensions_2  (0|1|2|3)           "2"
    texture_coord_dimensions_3  (0|1|2|3)           "2"
    texture_coord_dimensions_4  (0|1|2|3)           "2"
    texture_coord_dimensions_5  (0|1|2|3)           "2"
    texture_coord_dimensions_6  (0|1|2|3)           "2"
    texture_coord_dimensions_7  (0|1|2|3)           "2"
    tangents                    (true|false)    "false"
    binormals                   (true|false)    "false">
<!ELEMENT vertex (position, normal?, tangent?, binormal?,
 colour_diffuse?, colour_specular?, texcoord*)>
<!ELEMENT position EMPTY>
<!ATTLIST position
    x   CDATA   #REQUIRED
    y   CDATA   #REQUIRED
    z   CDATA   #REQUIRED >
<!ELEMENT normal EMPTY>
<!ATTLIST normal
    x   CDATA   #REQUIRED
    y   CDATA   #REQUIRED
    z   CDATA   #REQUIRED >
```

```
<!ELEMENT tangent EMPTY>
<!ATTLIST tangent
    x   CDATA   #REQUIRED
    y   CDATA   #REQUIRED
    z   CDATA   #REQUIRED >
<!ELEMENT binormal EMPTY>
<!ATTLIST binormal
    x   CDATA   #REQUIRED
    y   CDATA   #REQUIRED
    z   CDATA   #REQUIRED >
<!ELEMENT colour_diffuse EMPTY>
<!-- 'value' is a space-separated string containing r,g,b and
    optionally alpha for example value="1.0 0.0 0.0 0.5" or
    value="0.7 0.5 0.2" -->
<!ATTLIST colour_diffuse
    value   CDATA   #REQUIRED>
<!ELEMENT colour_specular EMPTY>
<!-- 'value' is a space-separated string containing r,g,b and
    optionally alpha for example value="1.0 0.0 0.0 0.5" or
    value="0.7 0.5 0.2" -->
<!ATTLIST colour_specular
    value   CDATA   #REQUIRED>
<!ELEMENT texcoord EMPTY>
<!ATTLIST texcoord
    u   CDATA   #REQUIRED
    v   CDATA   "0"
    w   CDATA   "0" >
<!ELEMENT submeshnames (submeshname+)>
<!ELEMENT submeshname EMPTY>
<!ATTLIST submeshname
    name    CDATA   #REQUIRED
    index   CDATA   #REQUIRED >
<!ELEMENT poses (pose+)>
<!-- A single pose references a single set of geometry data with
    a set of offsets.   If target is 'mesh', targets the shared
    geometry, if target is submesh, targets the submesh identified
    by 'index'. -->
<!ELEMENT pose (poseoffset+) >
<!ATTLIST pose
    target  (mesh|submesh) #REQUIRED
    index   CDATA   "0"
    name    CDATA   "">
<!-- poseoffset lists the vertices that change position, and by
    how much -->
<!ELEMENT poseoffset EMPTY>
<!ATTLIST poseoffset
```

```
    index   CDATA   #REQUIRED
    x       CDATA   #REQUIRED
    y       CDATA   #REQUIRED
    z       CDATA   #REQUIRED >
<!ELEMENT animations (animation+)>
<!ELEMENT animation (tracks)>
<!ATTLIST animation
    name    CDATA   #REQUIRED
    length  CDATA   #REQUIRED >
<!ELEMENT tracks (track+)>
<!ELEMENT track (keyframes)>
<!-- Morph animation is a keyframed set of absolute vertex
    positions. Cannot be blended with other morph animations or
    pose animation. Pose animation is a set of keyframes
    referencing poses and a weight, with one track per set of
    vertex data.
     Can be blended with other poses but not with morph animation.
    If target is 'mesh', targets the shared geometry, if target is
    submesh, targets the submesh identified by 'index'.
-->
<!ATTLIST track
    target  (mesh|submesh) #REQUIRED
    index   CDATA   "0"
    type    (morph|pose) #REQUIRED>
<!-- keyframes are applicable for all tracks, but for morph tracks
     they contain positions, and for pose tracks they contain pose
     references -->
<!ELEMENT keyframes (keyframe*)>
<!ELEMENT keyframe (position*, poseref*)>
<!ATTLIST keyframe
    time    CDATA   #REQUIRED >
<!-- Pose reference, links to pose via numeric index. target of
    parent track must agree with target of referenced pose. For a
    single track, each keyframe can reference multiple poses at
    different weights. -->
<!ELEMENT poseref EMPTY>
<!ATTLIST poseref
    poseindex   CDATA   #REQUIRED
    influence   CDATA   "1.0">
<!-- Optional extremity points on submeshes for sub-object
    transparency sorting -->
<!ELEMENT extremes (submesh+)>
<!ELEMENT submesh_extremes (position+)>
<!ATTLIST submesh_extremes
    index   CDATA   #REQUIRED>
```

7.2 *Document Type Definition* para um esqueleto

```
<!ELEMENT skeleton (bones, bonehierarchy, animations?,
    animationlinks?) >
<!ELEMENT bones (bone+) >
<!ELEMENT bone (position, rotation, scale?) >
<!ATTLIST bone
    id      CDATA   #REQUIRED
    name    CDATA   #REQUIRED>
<!ELEMENT position EMPTY>
<!ATTLIST position
    x       CDATA   #REQUIRED
    y       CDATA   #REQUIRED
    z       CDATA   #REQUIRED>
<!ELEMENT rotation (axis)>
<!ATTLIST rotation
    angle   CDATA   #REQUIRED>
<!ELEMENT axis EMPTY >
<!ATTLIST axis
    x       CDATA   #REQUIRED
    y       CDATA   #REQUIRED
    z       CDATA   #REQUIRED>
<!ELEMENT bonehierarchy (boneparent*)>
<!ELEMENT boneparent EMPTY>
<!-- NB both the below are bone names, not ids -->
<!ATTLIST boneparent
    bone    CDATA   #REQUIRED
    parent  CDATA   #REQUIRED>
<!ELEMENT animations (animation+)>
<!ELEMENT animation (tracks)>
<!ATTLIST animation
    name    CDATA   #REQUIRED
    length  CDATA   #REQUIRED>
<!ELEMENT tracks (track+)>
<!ELEMENT track (keyframes)>
<!ATTLIST track
    bone    CDATA   #REQUIRED>
<!ELEMENT keyframes (keyframe+)>
<!ELEMENT keyframe (translate?, rotate?, scale?)>
<!ATTLIST keyframe
    time    CDATA   #REQUIRED>
<!ELEMENT translate EMPTY>
<!ATTLIST translate
    x       CDATA   #REQUIRED
    y       CDATA   #REQUIRED
    z       CDATA   #REQUIRED>
<!ELEMENT rotate (axis)>
```

```
<!ATTLIST rotate
    angle   CDATA   #REQUIRED>
<!ELEMENT scale EMPTY>
<!-- UNIFORM SCALE is 'factor', or you can use per-axis (not both)
    -->
<!ATTLIST scale
    factor  CDATA   #IMPLIED
    x       CDATA   #IMPLIED
    y       CDATA   #IMPLIED
    z       CDATA   #IMPLIED>
<!ELEMENT animationlinks (animationlink+)>
<!ELEMENT animationlink EMPTY>
<!ATTLIST animationlink
    skeletonName    CDATA   #REQUIRED
    scale           CDATA   "1.0">
```

Referências

[1] M. B. Carmo, A. P. Cláudio, J. D. Cunha, H. Coelho, M. Silvestre, and M. Pinto-Albuquerque. Plataforma de Suporte à Geração de Cenas Animadas com Agentes Inteligentes. In *13º Encontro Português de Computação Gráfica*, pages 79–84, 2005.

[2] IViHumans. **I**ntelligent **Vi**rtual **Humans**. `http://labmag.di.fc.ul.pt/virtual/`. [Online; accessed 12-October-2007].

[3] LabMAg. Laboratório de Modelação de Agentes. `http://labmag.di.fc.ul.pt/`. [Online; accessed 12-October-2007].

[4] OGRE. Object-Oriented Graphics Rendering Engine. `http://www.ogre3d.org`. [Online; accessed 12-October-2007].

[5] JADE. Java Agent DEvelopment Framework. `http://jade.tilab.com`. [Online; accessed 12-October-2007].

[6] Poser 7. `http://www.e-frontier.com/go/poser`. [Online; accessed 12-October-2007].

[7] Blender. `http://www.blender.org/`. [Online; accessed 12-October-2007].

[8] Janete Faustino, Ana Paula Cláudio, and Maria Beatriz Carmo. Faces – Biblioteca de Expressões Faciais. In *Actas da 2ª Conferência Nacional em Interacção Pessoa-Máquina, Interacção 2006*, pages 139–142, Outubro 2006.

[9] G. Mandler. The Psychology of Facial Expressions. *Cambridge University Press*, 1997.

[10] Ricardo Abreu, Ana Paula Cláudio, and Maria Beatriz Carmo. Humanos Virtuais na Plataforma IViHumans – a Odisseia da Integração de Ferramentas. In *Actas do 15º Encontro Português de Computação Gráfica, 15º EPCG*, pages 217–222, Outubro 2007.

[11] João D. Cunha. Guião de computação gráfica – aulas teóricas. `http://mc.di.fc.ul.pt/cg/CG06_07/acetatosTeoricas0607/CG_0607_01_2pg.pdf`, 2006.

[12] Miguel Silvestre, Maria Pinto-Albuquerque, Maria Beatriz Carmo, Ana Paula Cláudio, João Duarte Cunha, and Helder Coelho. Arquitectura de Suporte à Geração de Cenas Animadas com Agentes Inteligentes. Technical Report DI-FCUL TR-04-7, Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa, 2004.