Considerações sobre técnicas para a implementação de *skeletal animation* em Jogos 3D

Márcio da Silva Camilo, Bernardo Nogueira S. Hodge, Rodrigo Pereira Martins, Alexandre Sztajnberg
Departamento de Informática e Ciências da Computação
Universidade Estadual do Rio de Janeiro
{pmacstronger, bernardohodge, rodrigomartins, alexszt}@ime.uerj.br

Abstract

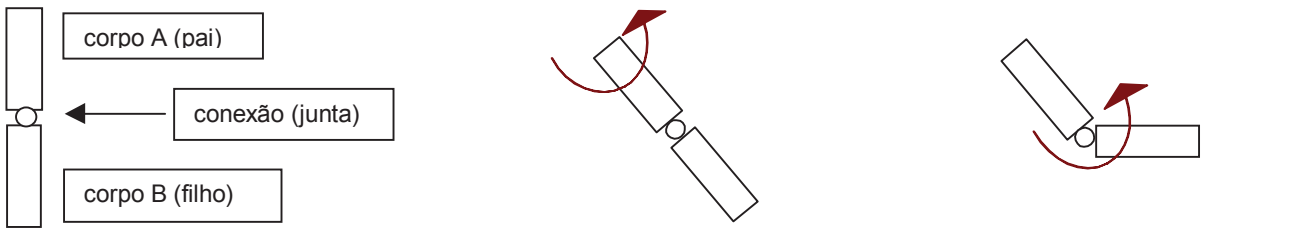
Neste artigo examinamos uma tecnologia de animação conhecida como skeletal animation baseada na junção hierárquica das partes que formam o modelo a ser animado. Esta tecnologia pode ser aplicada a diversos domínios onde a animação de modelos é necessária, como a análise e simulação física de estruturas hierárquicas simples e animações artísticas de modelos e entretenimento (jogos de computador). Para aplicar esta tecnologia desenvolvemos um conjunto de técnicas que permitem a sua implementação de forma simples e eficaz e utilizamos o mesmo na animação de modelos em jogos 3D.

1. Introdução

Ao contrário de jogos 2D, os jogos 3D aproximam-se cada vez mais de verdadeiros ambientes de realidade virtual, pelo realismo em que as ações durante o jogo se processam. Este realismo, com relação aos modelos de personagens é conseguido criando todo o "corpo" do personagem e não apenas um desenho em duas dimensões que mostra o perfil de um personagem como em jogos 2D. A criação do corpo do personagem em jogos 3D permite a realização de interação fisicamente factível do personagem com todos os objetos do cenário e a visualização completa do personagem sob qualquer ângulo de câmera. Para acompanhar este nível de realidade o processo de animação dos personagens em jogos 3D é mais complexo do que animações feitas em jogos 2D, não somente por afetar de forma verossímil todas as partes do corpo do personagem como também porque tenta retratar o mais fielmente possível o movimento de um ser real (humano ou animal, real ou fictício).

A fim de manipular e animar um personagem, uma representação conveniente deve ser modelada. Isto significa construir a definição de uma estrutura esquelética que admita movimento e uma "pele" que descreva a forma externa do ser [5, p23].

Skeletal animation é uma tecnologia para animações de modelos 3D criados em computador que consiste em criar um endoesqueleto que define os movimentos do personagem que, por sua vez, é recoberto por uma malha (vértices e faces), definindo o corpo do personagem. Utilizando o conceito de hierarquia entre as partes do modelo conectadas ao endoesqueleto é possível criar animações de personagens de jogos 3D em tempo real permitindo maior versatilidade e realismo em movimentos [2]. Essa hierarquia se baseia na idéia de que se um corpo A sofre um movimento de rotação ou translação, um corpo B conectado ao corpo A também sofre o mesmo movimento. O corpo B somente tem liberdade para sofrer um movimento de rotação em torno da conexão que o une ao corpo A. Esta regra fundamental define a hierarquia entre o corpo A e o corpo B, como pode ser observado na Figura 1. Podemos dizer que o corpo A é o "pai" do corpo B e analogamente que o corpo B é o "filho" do corpo A.



(a) Hierarquia (b) Movimento de A é "herdado" por B (c) Movimento de B em torno da conexão dele com A

Figura 1. Regra de hierarquia entre corpos

No método de animação convencional mais popular [2], que chamaremos *vertex mesh animation*, cada nova posição do modelo é criada pelo artista durante a fase de *design*. Uma animação simples é composta de vários quadros-chave (*key-frames*), cujos dados são armazenados em estruturas de dados - compostos de várias faces e vértices. Cada quadro-chave representa uma posição intermediária entre a posição inicial e a posição final da animação. Para que uma animação pareça convincente são necessários vários quadros-chave aumentando muito a demanda em armazenamento para cada seqüência de animação [2], uma solução para este problema é a interpolação entre quadros-chave, como por exemplo no Quake-md3 [4]. Nesse método, uma seqüência de animação deve ser criada para cada personagem mesmo que estes personagens tenham estruturas "ósseas" semelhantes. Não é possível nenhum tipo de reutilização entre modelos utilizando o método de animação convencional. Adicionalmente, um modelo não pode ter uma de suas partes modificadas em tempo de execução a não ser que sejam criadas animações *a priori* para estas novas partes. Por exemplo, quando um personagem em um jogo de ação muda de arma, seus movimentos com esta nova arma devem ter sido previamente criados. Esta inflexibilidade aumenta mais ainda a necessidade de quadros-chave do modelo.

Utilizando *skeletal animation*, por outro lado, para o mesmo cenário descrito, é possível criar apenas uma posição básica do modelo e descobrir em tempo real quais novas posições as diversas partes do modelo assumirão baseadas na movimentação hierárquica do "esqueleto". Desta forma, é possível armazenar apenas um modelo para cada personagem.

O modelo pode sofrer modificações em sua estrutura sem que isto resulte em qualquer problema para a animação. Ou seja, no exemplo da troca de arma, não existe problema para a animação uma vez que os movimentos com esta nova arma serão calculados em tempo real assim como o eram com a arma anterior.

De fato ainda é necessária a utilização de quadros-chave, como no *vertex mesh animation*, mas estes são criados apenas para os esqueletos (que contém poucos vértices). Através da interpolação entre dois quadros-chave é possível determinar as posições intermediárias do modelo. Assim, pode-se criar apenas o quadro-chave inicial e final de cada movimento não havendo necessidade dos quadros intermediários.

Finalmente, outro ponto positivo associado à tecnologia de *skeletal animation* é a possibilidade de utilizar os quadros-chave de um movimento para vários personagens que tenham as mesmas características "ósseas". Por exemplo, movimentos de um "esqueleto" bípede servem para descrever animações de humanóides sejam eles gigantes, anões, robôs, etc.

O *skeletal animation* tem contudo, uma desvantagem em relação ao *vertex mesh animation*. A fim de gerar as animações em tempo real cria-se uma grande demanda de processamento a cada *frame* a ser renderizado para descobrir as novas posições de cada vértice do modelo de cada personagem. A velocidade das animações pode ficar comprometida com o aumento de vértices por partes do corpo do modelo e com o aumento de modelos animados em uma cena.

Assim como o *vertex mesh animation*, o *skeletal animation* é dependente do artista que cria os movimentos dos modelos. O artista é o principal responsável por impor restrições de movimentos aos modelos. Estas restrições são necessárias para que a animação não resulte em movimentos absurdos como por exemplo a cabeça de um personagem rotacionando de 360 graus. O artista deve dividir um movimento em movimentos menores para que a seqüência correta de animação seja seguida.

Neste trabalho examinamos os detalhes da tecnologia de *skeletal animation* e apresentamos a proposta de um conjunto de técnicas como um exemplo de aplicação desta tecnologia na construção de jogos 3D. O uso de *skeletal animation* é composto de alguns passos, tais como a construção do esqueleto, a obtenção das transições de animações e composição das animações. Para cada um destes passos foram desenvolvidas técnicas cuja utilização será ilustrada.

O restante do texto esta organizado da seguinte forma. Na Seção 2 é mostrada a criação de um esqueleto hierárquico. Na Seção 3 é ilustrado o processo de obtenção das características de uma transição de animação. Na Seção 4 um algoritmo é descrito para a animação do modelo em tempo real. Na Seção 5 é abordada uma técnica para composição de movimentos para animações complexas. Na Seção 6 será mostrado como o conjunto de técnicas apresentado pode ser empregado no desenvolvimento de jogos 3D. Finalmente, na Seção 7 são apresentados pontos a serem estudados na continuação de nossas pesquisas e as conclusões deste trabalho.

1.1 Criação de um esqueleto hierárquico

A criação de um esqueleto hierárquico (Figura 2(a)) pode ser feita em qualquer programa de animação que permita criar pontos e nomeá-los. Cada ponto do esqueleto irá representar a posição de uma junta. A nomeação de cada ponto é importante, pois a partir do nome é obtida a informação da hierarquia entre as juntas do esqueleto. Uma junta pode ter vários filhos, mas tem apenas um pai.

Um arquivo de pontos contém a descrição dos pontos (juntas) de um esqueleto, com o nome e a posição de cada ponto. Em tempo de execução do programa é realizada a leitura do arquivo de pontos e cada ponto é mapeado em uma estrutura de junta (Figura 2(c)). Utilizando-se a convenção de nomeação, estabelecida quando da criação do arquivo de pontos, uma lista de estrutura de juntas é encadeada, conectando as juntas do esqueleto, formando uma estrutura hierárquica que representa o esqueleto (Figura 2(b)). Esta estrutura é, então, manipulada para gerar as animações.

Na Figura 2 podem ser observados um exemplo de esqueleto e a estrutura hierárquica que se deriva dele. A estrutura junta contém informações que a identificam, identificam a sua junta "pai", as suas juntas "filhas" e suas coordenadas reais. Cada junta também é associada a um

conjunto de quadros-chave (`quadro_chave kframes[]` da Figura 2(c)) que guardará as informações de uma transição (ângulo e eixo de rotação, e divisor que determina que fração do ângulo que será interpolada durante a animação). Uma transição neste contexto define, assim, o movimento entre as posições inicial e final de um movimento simples e suas características. Para uma junta particular, várias transições podem estar definidas.

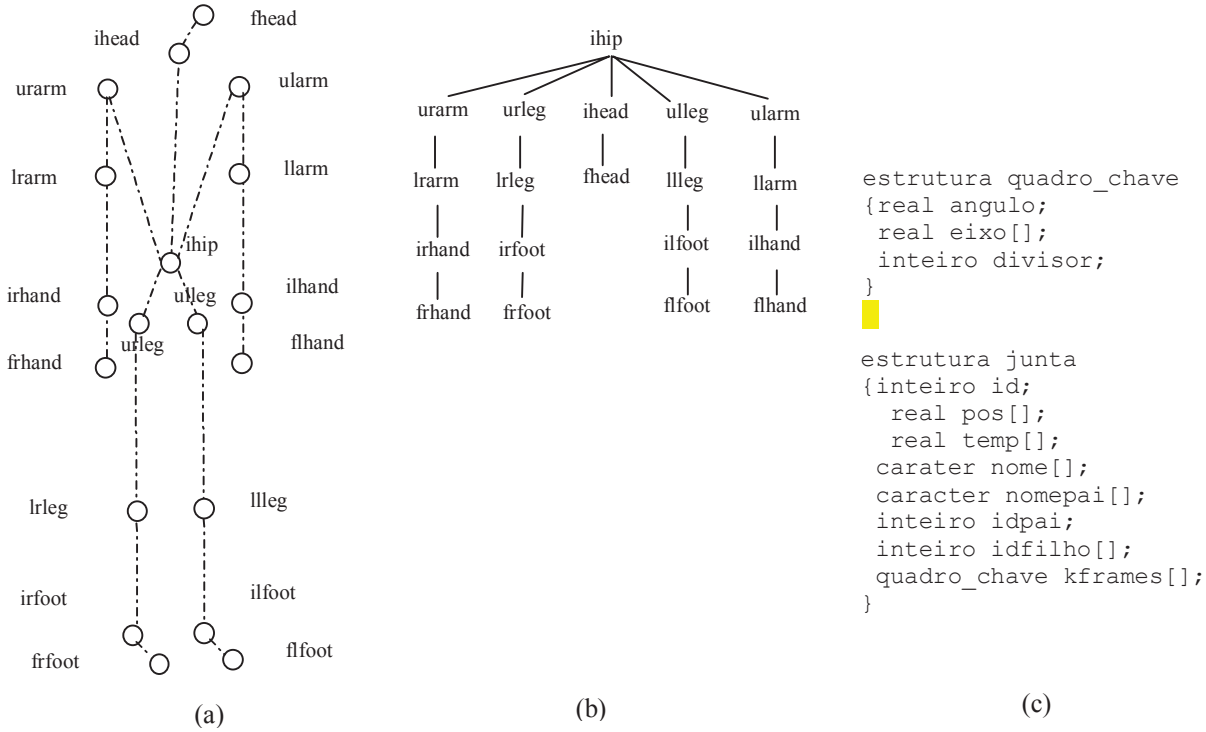


Figura 2. O esqueleto (a), a estrutura de dados hierárquica (b), código da estrutura (c)

Problemas de limitação de movimento do modelo podem ser resolvidos acrescentando mais juntas ao esqueleto. Por exemplo, no esqueleto acima não é possível o movimento quando o personagem rotaciona o torso ao redor da bacia (*ihip*). Neste caso podemos acrescentar duas juntas representando a parte central e periférica do torso de forma que a periférica (*utorso*) gire livre ao redor da central (*ltorso*).

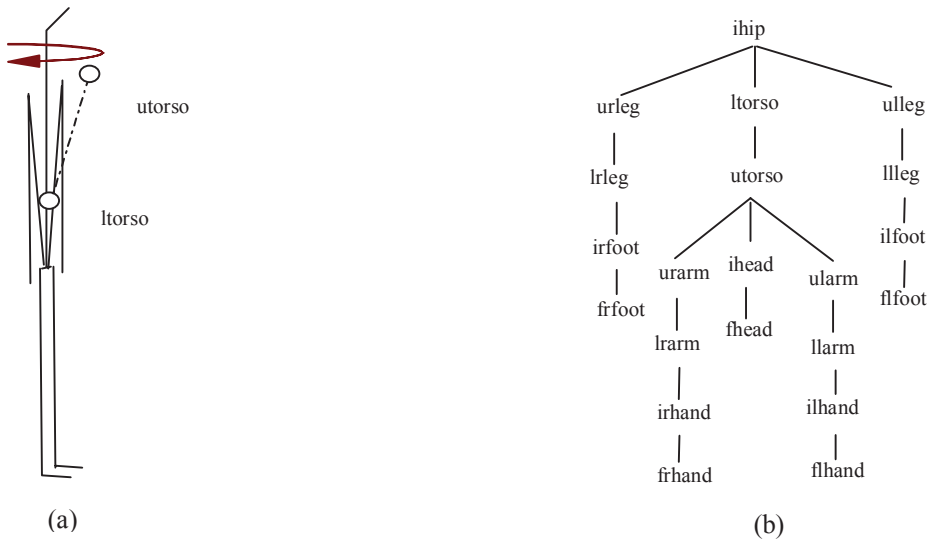


Figura 3. Associação de novas juntas no esqueleto (a) e modificações na estrutura hierárquica (b)

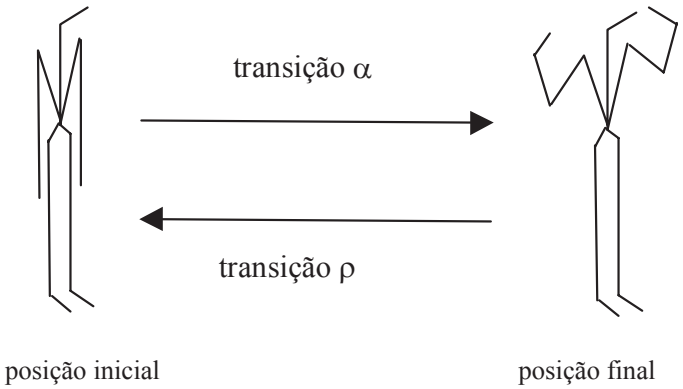


Figura 4. posições extremas de um movimento

Todos as juntas que devem herdar a rotações sofridas por *utorso* passam a ser seus descendentes. O resultado pode ser visto na Figura 3.

Para cada posição extrema que um modelo irá assumir em uma animação um arquivo de pontos deve ser criado. Uma posição extrema é toda posição inicial ou final de um movimento. Um exemplo é mostrado na Figura 4.

As posições intermediárias entre duas posições extremas serão descobertas em tempo real através de interpolação linear dos ângulos de rotação que levam o esqueleto de uma posição a outra.

2. A definição de eixo e ângulos de rotação da animação

A partir dos vários arquivos de pontos definindo as posições extremas que o modelo do personagem irá assumir, o próximo passo na composição do modelo é a definição das características que determinam as suas transições (Seção 2), isto é, que levam o esqueleto de uma posição a outra. Isto pode ser automatizado através da leitura de dois arquivos de pontos, definindo duas posições extremas e, para cada junta, com as informações obtidas destes arquivos, é calculado o par eixo-ângulo de transição (uma rotação) e armazenado na respectiva estrutura de quadro chave (Figura 2(c)). Para cada par de posições extremas este procedimento deve ser repetido. Note-se que isto não é muito, comparado com o volume de informações que se deve armazenar no método *vertex mesh animation*.

O processo de obtenção de par eixo-ângulo que representa a rotação de uma junta em torno de seu pai, é conhecido como *inverse kinematics* [1(p.853)]. A solução utilizada para automatizar este passo, no contexto deste trabalho, é a abordagem recursiva. Começando da junta no topo da hierarquia "visitam-se" as juntas descendentes sempre incorporando a cada uma delas as rotações dos seus ancestrais.Desta forma, faz-se com que o sistema de coordenadas esteja sempre atualizado para a junta atual e a obtenção de sua rotação em torno da junta pai resume-se a um problema simples de álgebra linear. Isso será ilustrado no exemplo a seguir.

Na Figura 5 tem-se uma hierarquia formada por 3 juntas, *j1*, *j2* e *j3*. A letra 'i' foi utilizada para indicar que a junta faz parte da posição 1 e 'f' para indicar que a junta faz parte da posição 2. Sendo *j1* a junta do topo da hierarquia, ela não sofre rotação. Isto significa dizer que as coordenadas de *j1i* são iguais às coordenadas de *j1f*. Passa-se, então, para as outras juntas.

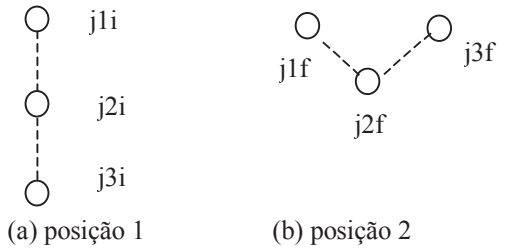


Figura 5. Posicionamento das juntas em duas posições distintas

O ângulo θ entre este dois vetores pode ser determinado como o arco-cosseno do produto escalar entre eles [3]. O vetor v3, perpendicular aos dois vetores, é o eixo de rotação e pode ser determinado calculando-se o produto vetorial entre eles [3]. O eixo v3 também deve ser normalizado. Este processo é mostrado na Figura 6.

De posse do eixo e do deve-se achar a nova posição *j2f* a partir de *j2i*. Basta transladar a origem para *j1*, e rotacionar o ponto *j2i* de θ radianos em torno de v3.

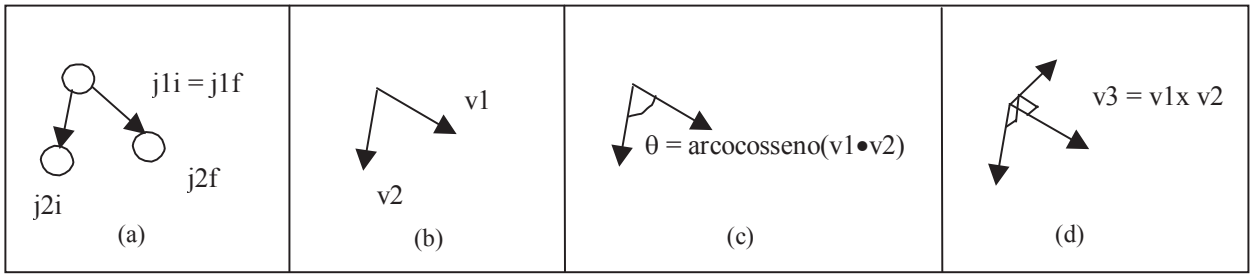


Figura 6. Posição das juntas (a), vetores v1 e v2 (b), obtenção do ângulo θ (c), obtenção do eixo v3 (d)

Observa-se que no caso de *j2* não foi necessário propagar a rotação de seu pai (*j1*), pois este último não sofreu rotação. Para a obtenção do par eixo-ângulo da junta *j3*, por outro lado, será necessário primeiramente rotacionar *j3i* com as características de rotação de *j2*. Assim a nova junta obtida (*j3i'*) estará posicionada corretamente para a obtenção do eixo e do ângulo de rotação, como se pode ver na Figura 7. Em seguida, o mesmo procedimento aplicado a *j2* é também aplicado a *j3*. Este processo é realizado recursivamente até que todas as juntas tenham sido visitadas.

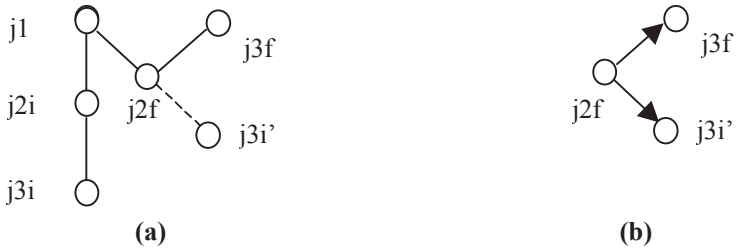


Figura 7. A junta j3i é rotacionada para tornar-se j3i' (a) possibilitando a comparação com j3f (b)

3. A animação do modelo de um personagem

O projeto de um modelo de personagem é iniciado pela criação de um arquivo de modelo, automatizado por um programa de modelagem 3D. Um arquivo de modelo é formado por partes do corpo, que dão a cobertura do personagem, e pontos que conectam as diversas partes do corpo. Cada parte do corpo é representada por um conjunto de vértices e faces, formando objetos nomeados no arquivo. Cada ponto funciona como uma junta do modelo. As partes do corpo do modelo devem ser criadas de forma que cada parte seja subordinada, através do seu nome a um ponto.

A partir de um arquivo de modelo uma estrutura de dados semelhante à estrutura *junta* (do esqueleto) será criada para armazenar as partes e juntas do modelo. Esta estrutura será efetivamente utilizada para renderizar a imagem do modelo em tempo de execução.

Como foi dito anteriormente a partir de um esqueleto é possível animar um modelo de personagem. Assim, na estrutura de dados cada junta do modelo deve receber também as características de rotação (par eixo-ângulo) da junta do esqueleto associada a ela (associação feita pelo nome), como na Figura 8(a) e (b).

O modelo será criado na posição inicial e esta posição será referencial para qualquer outra posição que o modelo tiver que assumir. Em tempo de execução a nova posição do modelo será descoberta utilizando-se *forward kinematics* [1 pg. 853], o que significa determinar as novas coordenadas de cada vértice após as rotações sofridas (ou, rotacionar o modelo). As rotações são obtidas como resultado do cálculo das transições entre posições na estrutura de juntas do esqueleto (Seção 3). Para isso utiliza-se um método recursivo de "visitação" das juntas da estrutura hierárquica do modelo propagando as rotações para as juntas descendentes.

Abaixo apresentamos um pseudo-código do processo:

```
RenderizaModelo (junta_do_modelo, nkframe)
{
  Para cada filho_de_junta_do_modelo {
    PropagaRotação (filho_de_junta_do_modelo,
        pai_de_junta_do_modelo,
        junta_do_modelo.kframes[nkframes].eixo[],
        junta_do_modelo.kframes[nkframes].angulo)
    RenderizaModelo (filho_de_junta_do_modelo,
                     nkframe)
  }
  Para cada parte do corpo associada a
                          junta_do_modelo {
    Rotaciona (partedocorpo,
               pai_de_junta_do_modelo,
        junta_do_modelo.kframes[nkframes].eixo[],
        junta_do_modelo.kframes[nkframes].angulo)
    Renderiza (partedocorpo)
  }
}

PropagaRotação (junta, ancestral, eixo[], ângulo)
{
  Para cada filho_de_junta_do_modelo {
    PropagaRotação (filho_de_junta_do_modelo,
                    ancestral, eixo[], ângulo)
  }
  Para cada parte do corpo associada a junta
  {
    Rotaciona (partedocorpo, ancestral,
               eixo[], ângulo)
  }
}
```

A função *RenderizaModelo* é inicialmente chamada para a junta do modelo que ocupa a maior posição na estrutura hierárquica (no nosso exemplo a junta da bacia, ou *ihip*). O parâmetro *nkframe* representa o índice do quadro-chave desejado para a transição do movimento. Este índice pode ser obtido através de uma tabela de indexação de transições como será mostrado na Seção 5.

A junta do modelo que está correntemente sendo tratada, tem sua rotação propagada para as suas juntas "filhas". A função *RenderizaModelo* é recursivamente chamada para cada junta "filha" da junta atual. Após percorrer todos os filhos da junta atual, a parte do corpo associada a esta junta é rotacionada e renderizada.

A função *PropagaRotação* é responsável por percorrer recursivamente todas as juntas "filhas" da junta atual, propagando a rotação armazenada na junta atual para todos as suas juntas "filhas". Após "visitar" todas as juntas "filhas" ela trata a junta atual rotacionando a parte do corpo associada a esta junta.

A função *Rotaciona* produz novas coordenadas a partir das coordenadas iniciais dos vértices que formam a parte do corpo que será renderizada. Assim ela rotaciona a parte do corpo utilizando as informações *eixo*, *ângulo* e coordenadas de *ancestral* (uma junta de ordem superior). Para isso, cada vértice da parte do corpo sofre uma translação que equivale a fazer as coordenadas de *ancestral* coincidirem com a origem [3(p.27)]. Então é utilizada a técnica de rotação *rotation about an arbitrary axis* [3(p.41-42)] que rotaciona o vértice ao redor do *eixo* utilizando o *ângulo*. O vértice é então destransladado para reverter o efeito da translação feita inicialmente.

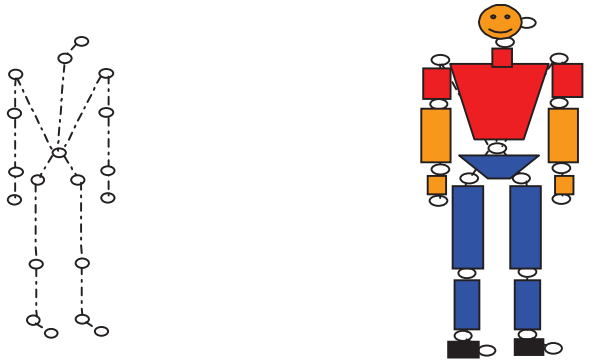
A função *Renderiza* desenha as partes do corpo do modelo cuja implementação é dependente da biblioteca gráfica utilizada.

Observa-se que rotacionando o modelo da posição inicial diretamente para a posição final, os movimentos parecerão bruscos. É possível fazer com que um movimento seja feito de forma suave ao longo do tempo bastando que se controle o crescimento das frações do ângulo até que se atinja o seu valor final. Para cada fração do ângulo de rotação pode-se interpolar posições entre a posição inicial e a posição final. Assim, no código acima por exemplo pode-se substituir `junta_do_modelo.angulo` por `fracaoangulo` onde:

```
fracaoangulo = (contador *
      junta_do_modelo.kframes[nkframes].angulo *
      junta_do_modelo.kframes[nkframes].divisor)
```

sendo `contador` uma variável que assume valores de 0 até o valor de `junta_do_modelo.kframes [nkframes].divisor` fazendo com que o ângulo da rotação (`fracaoangulo`) aumente no tempo. O parâmetro divisor pode ser lido do arquivo do esqueleto uma vez que deve ser definido pelo artista criador dos movimentos do esqueleto do modelo.

Neste exemplo, é importante que cada nova rotação de ângulos interpolados (quando contador é incrementado) seja sempre aplicada às coordenadas iniciais dos vértices das partes do corpo na posição inicial do movimento evitando assim que rotações sejam acumuladas indevidamente.



(a) juntas do esqueleto (b) juntas do modelo

Figura 8. Esqueleto e Modelo

A hierarquia entre os objetos (partes do corpo) formadores dos modelos possibilita que se façam substituições em partes do corpo a qualquer momento. Assim o torso de um personagem pode se modificar quando ele troca de armadura ou uma arma pode ser substituída por outra. Basta que os objetos (`partedocorpo`) sejam retirados ou anexados à estrutura hierárquica de juntas do corpo do modelo. Da mesma forma novos objetos podem ser agregados a partes do corpo bem como outros podem ser desconectados quando necessário.

É importante observar que as medidas de distâncias entre as juntas no esqueleto e as juntas no modelo não precisam ser as mesmas. Isto possibilita a utilização, como já foi dito, de um único esqueleto para várias estruturas similares. Um esqueleto bípede pode servir para qualquer modelo de ser bípede.

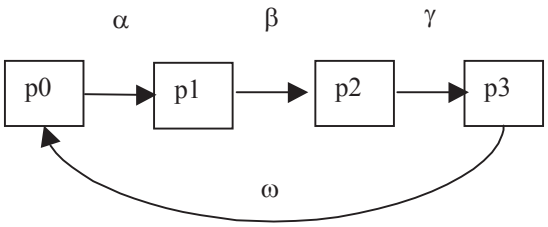
4. Composição de animações

Até este ponto, a animação de um modelo requer que este inicie sua movimentação partindo sempre de uma posição referencial, representada pelo arquivo do modelo. É necessário, portanto, criar a possibilidade de movimentos que não partam da posição referencial. Para isso utiliza-se um mecanismo simples de armazenamento da última malha (conjunto de vértices e faces do modelo). Assim, para cada transição de posições, além de guardar as características necessárias para a execução do movimento, devemos também guardar o estado da malha após cada movimento para possibilitar que o próximo movimento parta do ponto atual e não da posição referencial.

Adicionalmente, como os personagens em geral são implementados através de máquinas de estado bem definidas é possível mapear-se exatamente todas as transições de movimento que podem acontecer. Assim é possível criar ciclos de movimentos que tenham a posição referencial como base (iniciem e terminem nela).

Por exemplo, na Figura 9(a), no final da transição α, de p0 a p1, guardamos a malha transformada e então aplicamos sobre ela a transição β, de p1 a p2, guardando ao final dela a malha transformada e aplicamos sobre ela a transição γ, de p2 a p3, guardando ao final dela a malha transformada e aplicamos sobre ela a transição ω, de p3 a p1.

Na prática pode ser criada uma tabela para determinar que índice numérico da estrutura quadros-chave associa-se a que transição entre duas posições (tabela de indexação de transições na Figura 9(b)). Note que as letras gregas no diagrama da Figura 9(a) servem como identificadores das transições. Note também que outras transições seriam possíveis.



(a)

pos inicial	pos final	índice
p0	p1	1 (α)
...	...	...
p1	p2	5 (β)
...	...	...
p2	p3	8 (γ)
p3	p0	9 (ω)
...	...	...

(b)

Figura 9. Seqüência de animação formada de transições (a) e tabela de indexação de transições (b)

5. *Skeletal animation* em um jogo 3D

Nesta seção será discutido como o conjunto de técnicas apresentadas de implementação do *skeletal animation* se inserem na arquitetura de um jogo 3D.

Na Figura 10 pode-se observar um com as principais atividades realizadas durante a execução de um jogo 3D. No sub-passo de *leitura de arquivos* são lidos os arquivos de esqueleto para cada posição e de modelos dos personagens. Após esta leitura serão criadas as estruturas hierárquicas das juntas para esqueletos e de juntas do modelo contendo partes do corpo para os personagens. No sub-passo de *criação de estruturas*, as estruturas de juntas de esqueleto originárias de cada arquivo são comparadas de forma que para cada par de arquivos se crie uma estrutura de quadro-chave refletindo a transição do esqueleto de uma posição à outra, através da descoberta das características de transição (eixo e ângulo) da animação. As informações das juntas de esqueleto são passadas às juntas de modelo dos personagens.

No sub-passo *IA dos personagens* será determinada qual a nova animação que o modelo deverá seguir, de acordo com a interação do personagem com o ambiente e o seu mecanismo de inteligência artificial. Essa animação será determinada pesquisando a tabela de indexação de transições (Figura 9(b)), sendo o índice resultante utilizado para indicar que posição no *array* de quadro-chaves será utilizado. No sub-passo *animação dos modelos* essa

animação (transição entre a posição corrente e a nova posição) será realizada utilizando o algoritmo de animação em tempo real sobre a posição corrente que criará uma posição intermediária interpolada entre a posição corrente e a nova posição. Em intervalos de tempo subseqüentes o processo de interpolação é continuado a fim de realizar plenamente a animação. Este processo termina quando a animação é completada.

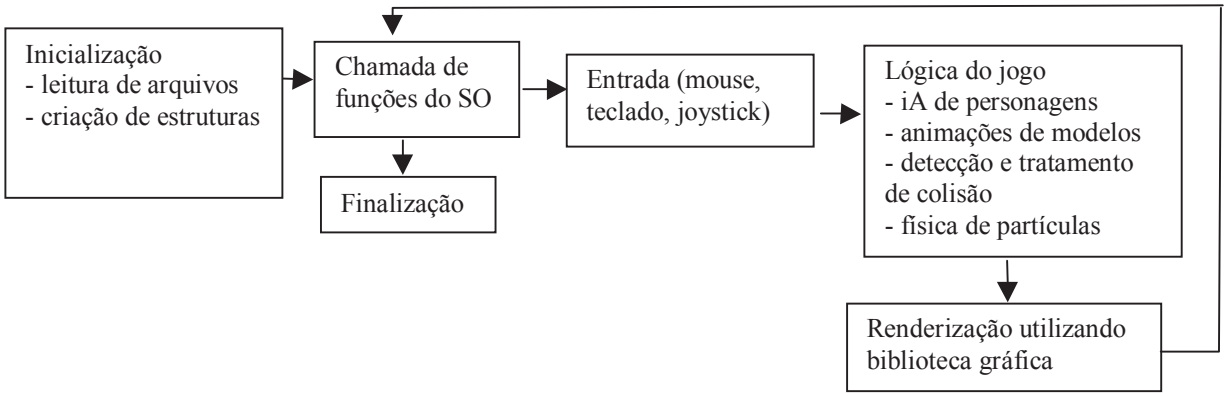


Figura 10. O diagrama esquemático simplificado de um jogo 3D.

5. Conclusões

Neste texto foi apresentada a tecnologia conhecida como *skeletal animation*, suas vantagens sobre o método de *vertex mesh animation* (como a reutilização de estruturas, economia em armazenamento de modelos); e desvantagens em relação ao custo da animação em tempo real, que pode ser muito alto no que tange ao processamento. Foram introduzidas também técnicas desenvolvidas que possibilitam uma implementação simples e eficaz de *skeletal animation*.

Algumas tentativas de utilização de *skeletal animation* em jogos 3D podem ser consideradas muito bem sucedidas como é o caso do modelo Quake-md3 da Id Software [4]. Este modelo utiliza, porém, uma implementação mista de *skeletal animation* e o *vertex mesh animation* na tentativa de balancear as características das duas tecnologias.

Atualmente estamos estudando otimizações no conjunto de técnicas proposto para o processo de animação do modelo. Uma possibilidade é a utilização da matemática de *quaternions* que viria minimizar o custo dos cálculos de rotação uma vez que podem representar uma rotação sem a necessidade de calcular senos e cossenos como é feito com a técnica *axis-angle* [3(p.41-42)]. *Quaternions* também representam uma forma versátil de interpolar animações entre estados intermediários sem a necessidade de armazenamento das características de rotação entre as posições não iniciais, através de SLERP (*Spherical Linear InteRPolation*) [3(p.48)].

A possibilidade de utilização de multiplicação de matrizes de rotação via hardware também está sendo estudada. Esta alternativa é altamente dependente do hardware, mas facilidades contidas em bibliotecas como o OpenGL e DirectX podem vir a serem utilizadas no sentido de armazenar matrizes de rotação em tempo real de renderização.

Como proposta de continuação de nossas pesquisas, investigaremos a possibilidade de incorporar pesos às partes do corpo do modelo a fim de permitir simular deformações de partes do corpo do modelo do personagem (*soft skinned model*) [2].

Mesmo considerando as limitações das técnicas discutidas neste trabalho, sobretudo com relação ao *overhead* do processamento, o *skeletal animation* mostra-se uma tecnologia promissora de simulação de modelos hierárquicos em computadores. Com o aumento do poder de processamento e da transferência de certas operações do *pipeline* de renderização para o *hardware* das placas de vídeo, em um futuro próximo, a tecnologia de *skeletal animation* passará a ser o padrão para criação de animações em jogos 3D.

Agradecimento. O trabalho é parcialmente financiado pelo CNPq (PDPG-TI processo 552192/2002-3).

6. Bibliografia

[1] Lamothe, Andre, "Trick of the Windows Game Programming Gurus", Sams, 1999.

[2] Anderson, Eike, "Real-Time Character Animation for Computer Games", National Centre for Computer Animation, Bournemouth University, 2001.

[3] Möller, T. and Haines, E., "Real Time Rendering", A. K. Peters, 1999.

[4] Humphrey,Benjamin."MD3Animation_OGL", http://www.gametutorials.com, 2003.

[5] Badler, Norman; Phillips, Cary and Webber, Bonnie, "Simulating Humans: computers graphics, animation and control", Oxford University Press, 1999.