UM ESTÁGIO INTERMEDIÁRIO DE CONFIGURAÇÃO DE ANIMAÇÕES PARA JOGOS E APLICAÇÕES DE SIMULAÇÃO 3D

Márcio da Silva Camilo

Aura Conci

Universidade Federal Fluminense
Rua Passo da Pátria, 156, São Domingos (Sala 452 - 4º andar - Bloco D) - Niterói - Rio de Janeiro
{mcamilo, aconci}@.ic.uff.br

Resumo

A representação de personagens em jogos e aplicações de simulação 3D é baseada em seqüências de animações que representam os movimentos de um personagem. Em geral o processo de representação de personagens é realizado em dois estágios: a criação das seqüências de animações em tempo de *design* (estágio de *design*) e a apresentação das seqüências em tempo de execução do jogo (estágio de execução). Esse artigo propõe a utilização de um estágio intermediário entre os estágios de *design* e execução, no qual os controles de uniformidade, evolução e *timing* são gerenciados através da manipulação do número de *frames* intermediários a cada par de *keyframes* da seqüência de animação.
Palavras-Chaves: Simulação; Jogo; Computação Gráfica; Sistema de Animação

Resumo

The representation of characters in games and 3D simulation application is based in animation sequences that show the movements of this character. Generally, this process is accomplished in two stages: the animation sequences are created in design time (design stage) and animation sequences are shown in the application run-time (run-time estage). This article proposes using an intermediate stage on which animation controls such as: uniformity, evolution and timing can be set by handling the intermediate frames amount for each pair of keyframes of the animation sequence.
Keywords: Simulation; Game; Computer Graphics; Animation System

1. INTRODUÇÃO

Jogos 3D e aplicações de simulação 3D guardam intima relação. De fato, é muito comum se utilizar *engines* ou *frameworks* de jogos 3D para criação de cenários com fins de aprendizado onde o ambiente 3D permite ao usuário um nível maior de imersão. Alguns exemplos podem ser citados como: simuladores de vôo (Flight Simulator), jogos de simulação de administração (SimCity), jogos de estratégia (Comand and Conquer), jogos de situações de combate (Counter Strike), entre outros.

Neste contexto, é importante a criação de animações de personagens que sejam realísticas sem, contudo, comprometer o desempenho da aplicação, cuja ação se processa, muitas vezes, em tempo real. A geração de movimentos em que uma animação se baseia pode ser realizada utilizando técnicas que envolvem esforços complementares de controle humano e

automação [1]. As técnicas mais comumente utilizadas para a geração e representação de movimentos podem ser sintetizadas em: métodos procedurais, *motion capture* e *keyframing* [1].

Métodos procedurais são técnicas em que o movimento é definido automaticamente durante a execução da aplicação [1]. As regras de criação de movimentos podem ser baseadas em leis físicas (como em sistemas de partículas ou superfícies flexíveis), ou em leis de comportamento social (chamado *behavioral animation*) [1]. Essas técnicas apresentam como pontos positivos a coerência de movimentos e a facilidade de animação de sistemas complexos.

A coerência de movimentos permite a representação de objetos semelhantes com movimentos semelhantes. Sistemas complexos que podem ser difíceis de serem criados por um animador como sistemas de partículas, superfícies flexíveis e bandos de animais ficam facilitados pela automação de modelos matemáticos. Como pontos negativos podem ser citados: a dificuldade de geração de cenas complexas e a pouca interação humana.

A dificuldade de geração de cenas complexas ocorre porque pode consumir muitos recursos de processamento causando impacto no desempenho da aplicação. A pouca interação humana no processo de especificação do movimento pode fazer com que as animações sejam sentidas pelo expectador como muito "computacionais", ou seja, pouco naturais.

Motion Capture ou *performance animation* [2] consiste na obtenção de movimentos baseados em mecanismos de captura de movimentos conectados ao corpo de atores ou objetos. Como característica positiva essa técnica apresenta resultados muito realísticos, pois se baseia no movimento real. Como pontos negativos podem ser citados o alto custo de equipamentos de sensores de captura mais sofisticados de movimento, e as limitações que esses sensores podem trazer aos movimentos dos atores durante a gravação de movimentos [3].

Keyframing é uma técnica de animação baseada na antiga técnica de criação de animações feitas a mão em desenhos animados da década de 1930 [4]. Era comum nessa época, dividir o processo de criação de animações em 2D em duas etapas. Na primeira, eram desenhados, a mão, *keyframes* que definiam o movimento de um personagem em uma cena. Em seguida eram desenhados *frames* intermediários entre os *keyframes* para que o movimento parecesse contínuo. Essa técnica é chamada de *inbetweening*.

A técnica de criação de *keyframes* e posterior complementação com *intermeddiate frames* foi transportada para a produção de animações em 3D modernas com as devidas adaptações. Uma importante adaptação diz respeito ao fato de que o processo de *inbetweening* é realizado pelo computador durante a fase de *design* ou posteriormente na fase de execução, em tempo real.

Embora existam esforços no sentido de criar movimentos de personagens proceduralmente, e também a utilização de *motion* capture, a grande maioria dos jogos 3D atualmente ainda baseiam o movimento dos personagens em seqüências de animações *keyframing* [3]. A explicação para a opção da utilização de seqüências *keyframing* é o alto custo de processamento necessário para a criação de movimentos em tempo real, ou captura de movimentos com atores humanos ou objetos reais [3]. Por outro lado, a técnica de *keyframing* demanda o uso intenso de memória uma vez que as informações de cada quadro devem ser armazenadas para que possam ser processadas em tempo real.

A técnica de *keyframing*, para jogos 3D, costuma ser divida em dois estágios que são o estágio de *design* e o estágio de execução. O estágio de *design* é quando todas as questões referentes à animação como *timing,* evolução, dentre outras [4] são consideradas na criação de *keyframes* para cada seqüência de animação. O estágio de execução da aplicação é onde as seqüências são processadas e mostradas podendo ou não ser criados *frames* intermediários automaticamente. Os *frames* intermediários significam uma economia de memória uma vez que podem ser obtidos a partir dos *keyframes* por alguma função de interpolação. Dessa forma, uma

grande responsabilidade recai sobre o estágio de *design* enquanto que o estágio de execução tem uma tarefa simplificada de processar os *keyframes* e criar *frames* intermediários.

Em geral a quantidade de *frames* intermediários é globalmente definida para todos segmentos (intervalo entre dois *keyframes*) da seqüência de animação, isto é, um número de *frames* intermediários é definido para todos os segmentos. Sendo assim, não é possível a configuração de segmentos mais longos ou mais curtos para a definição do *timing* e evolução da seqüência de animação. Além disso, é interessante que a aplicação possa controlar de alguma forma o número de *frames* intermediários adaptando-o ao desempenho relativo à sua execução.

Alguns pacotes de criação de animações permitem a configuração de *timing* e evolução de seqüências de animação, chamadas ferramentas de *time-editing* [5][6]. Mas nesse caso, o problema é que *frames* intermediários são geralmente transformados em *keyframes* e armazenados em arquivo como tal. Dessa forma, o problema de armazenamento de *keyframes* em memória durante a execução irá persistir, pois a economia que se podia obter utilizando *frames* intermediários, se perdeu.

Esse artigo propõe não o armazenamento de de *frames* intermediários mas apenas um valor que informa a quantidade de *frames* intermediários a serem criados durante o estágio de execução. Além disso, propõe também a utilização de um estágio intermediário entre os estágios de *design* e execução que atuará como um ajuste fino do estágio de *design* e como configuração de informações a serem utilizadas pelo estágio de execução, como pode ser visto na figura 1.

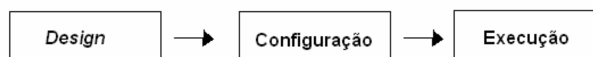

Figura 1. Estágio intermediário de configuração de número de *frames* intermediários

Esse estágio intermediário de configuração utiliza o número de *frames* intermediários de cada segmento da seqüência da animação para esse propósito. Através dele é feita uma configuração híbrida baseada em estratégias de controle: de *timing* e evolução da seqüência de animação, na uniformidade entre os segmentos da seqüência de animação (uniformização de segmentos) e no controle que o estágio de execução pode desempenhar sobre o número de *frames* intermediários durante a execução da aplicação.

O restante deste artigo está organizado como se segue: a seção 2 apresenta as características de animações em jogos 3D como é geralmente feita atualmente. A seção 3 apresenta a idéia do controle de animação pelo número de quadros intermediários e descreve cada uma das estratégias de controle do número de quadros intermediários. A seção 4 mostra as características da implementação em que se baseou este artigo. A seção 5 apresenta alguns resultados já obtidos e algumas considerações sobre eles. Finalmente, a seção 6 tece conclusões e aborda considerações futuras.

2. CARACTERÍSTICA DE ANIMAÇÕES KEYFRAMING PARA JOGOS 3D

Os primeiros jogos 3D utilizaram uma técnica onde cada *keyframe* é especificado como o conjunto de posições de cada vértice do modelo no *frame* [9]. Essa técnica, conhecida como *vertex animation* [9], tem o inconveniente de necessitar de muita memória para armazenar cada *keyframe*[10][11]. Um exemplo de jogo que utiliza essa técnica é o jogo Quake 2 [8], cujo modelo de definição de animação, MD2 (Monster Doom Version 2), utilizava *vertex animation* [8].

Muitos jogos utilizam modelos hierárquicos (esqueletos ou modelos esqueletais) baseados em juntas para a representação dos personagens. Cada junta pode rotacionar em relação a junta de nível superior. Uma rotação é propagada de uma junta para as suas juntas "filhas". O movimento de uma junta é a combinação da sua rotação e das rotações herdadas pelas juntas de nível superior [12][13][14].

Os modelos hierárquicos podem ser utilizados como uma forma mais eficiente de *keyframing* do que o *vertex animation*, uma vez que apenas informações de rotação por juntas precisam ser armazenadas para cada *keyframe*. O processamento em tempo de execução fica um pouco mais complexo já que as rotações precisam ser combinadas e processadas para obtenção das posições reais de cada vértice a partir de uma posição inicial conhecida como *baseframe* [15].

Os modelos hierárquicos foram utilizados em jogos como Half Life [6]. O problema com os modelos puramente hierárquicos é que na região das juntas ocorrem efeitos visuais indesejáveis pois quando vértices de partes do corpo diferentes, como o braço e o ante-braço, se movem podem aparecer áreas de falhas ou interpenetração. Uma solução para esse problema é a utilização de *skinning* [7]. Com o *skinning*, um vértice pode pertencer a mais de uma junta (vértices em regiões limítrofes podem pertencer a duas ou até mais partes do corpo) e a posição desses vértices é determinada pela influência exercida pelos pesos que as associam a cada junta [16].

Mesmo considerando a eficiência proporcionada pelos modelos hierárquicos, a necessidade de guardar as informações de cada quadro principal é ainda um limitador importante para o desempenho do processamento de animações em um jogo. A capacidade de memória pode comprometer não somente o tempo de execução do jogo como também a quantidade de informações por quadros. Por isso formas mais eficientes de definição das estruturas de dados que representam um *keyframe* devem ser otimizadas de forma que apenas as informações vitais sejam armazenadas. É muito comum em jogos de computador que o processamento do jogo fique responsável pelo processo de criação de *frames* intermediários, ou seja, *inbetweening*. Dessa forma é possível criar *keyframes* que definam a animação deixando que os *frames* que sirvam de ligação (para dar continuidade) entre um *keyframe* e o subseqüente sejam criados no estágio de execução.

Se por um lado não é razoável que cada posição de um movimento seja representada por um *keyframe*, uma vez que isso traria um consumo grande e desnecessário de memória, também não se pode exagerar na utilização do *inbetweening* uma vez que para a definição completa do movimento são necessários tantos *keyframes* quantos necessários para descrever todas as variações do movimento.

Uma função de interpolação deve ser utilizada para a criação de *frames* intermediários para que estes sejam coerentes com os *keyframes* que servem de extremos para cada segmento. Segundo [12] há funções de interpolação mais apropriadas para controlar o *inbetweening* bem como existem também funções para definir os trajetos de um movimento, como por exemplo, o trajeto percorrido por um ser humano. Para a técnica de *vertex animation*, a função de interpolação para criação de *frames* intermediários pode ser a interpolação linear que é simples de implementar e consome muito poucos recursos de processamento.

Para representações mais elaboradas como modelos esqueletais, muitos jogos têm adotado a representação de rotações sobre a forma de *quaternions* [7]. A matemática de *quaternions* é muito apropriada para a representação de rotações uma vez que ela resolve problemas existente na representação angular de Euler (interdependência entre eixos e *gimbal lock*) [8][17].

Para criar interpolações entre *quaternions* existe uma função correspondente à linear, a interpolação linear esférica (Slerp[8][17][18]) que se adeqüa perfeitamente ao uso de *quaternions*. Dados dois *quaternions* unitários pode-se encontrar um *quaternion* intermediário

sobre a curva geodésica dentro do espaço geométrico dos pontos definidos pelos *quaternions* unitários, que é uma esfera unitária em 4 dimensões [17]. Dessa maneira, pode-se levar matrizes de rotações para o espaço de *quaternions*, efetuar a interpolação e então voltar com o *quaternion* resultante em uma matriz de rotação.

3. CONFIGURAÇÃO DA SEQÜÊNCIA DE ANIMAÇÃO PELO NÚMERO DE FRAMES INTERMEDIÁRIOS

Um estágio intermediário de configuração entre o estágio de *design* e o estágio de execução pode ser utilizado para a configuração de números de quadros intermediários entre os segmentos da seqüência de animação. Esse estágio pode ser completamente configurado pelo usuário como citado em [19] ou pode ser baseado em estratégias de controle que permitem que os valores dos *frames* intermediários para cada segmento sejam computacionalmente calculados.

Um ajuste de valor por segmento pode ser útil de várias formas. Em primeiro lugar é importante considerar a questão da uniformidade entre *keyframes* (uniformização). Quanto mais equilibrada é a animação maior é a sensação de suavidade sentida pelo observador. Outro ponto importante é que este ajuste possibilita que se associem características de evolução da animação como *slow in* e *slow out*[4]. O terceiro ponto contemplado neste trabalho é a questão do tempo de duração da animação determinado por um fator global (a todos os segmentos da seqüência de animação) configurado no estágio intermediário podendo ser convenientemente manipulado pela aplicação em tempo de execução para que o seu desempenho não seja comprometido.

O valor de controle de distância será multiplicado ao valor de controle de evolução. Esse valor será posteriormente multiplicado por um valor relacionado ao controle de *timing*, assim:

número de quadros intermediários = controle de evolução · controle de distância · controle de *timing*

Segue uma explicação mais aprofundada de como cada uma dessas estratégias foi desenvolvida para uma configuração mais apurada das seqüências de animação.

3.1 CONTROLE BASEADO NA UNIFORMIDADE ENTRE KEYFRAMES

Uma questão relevante na manipulação do número de quadros intermediários é a uniformidade entre *frames*. A idéia aqui é que o estágio intermediário seja capaz de equiparar quadros muito distantes através de um maior ou menor número de *frames* intermediários entre os *keyframes*. Dessa forma, se pode equilibrar a distância entre *keyframes* aumentando ou diminuindo o número de *frames* intermediários e minimizar os efeitos bruscos causados por modificações muito acentuadas de um *keyframe* para outro. Essa mudanças bruscas podem ou não ser pretendidas pelo profissional de animação no estágio de *design*. Caso elas sejam indesejáveis, é possível atenuá-las através desse controle.

A maior dificuldade para a realização desse controle é definir o que é a distância entre dois *keyframes*. Para efeitos práticos e como uma primeira abordagem, na implementação em que se baseou este trabalho, foi utilizada uma definição bastante simples de distância entre os quadros principais baseada na posição final entre os vértices que compõem o corpo (a malha) do personagem. Distância, neste caso, pode ser entendida como diferença, assim, quanto maior a

diferença entre dois *keyframes* contíguos, ou seja, a diferença das posições do personagem nos dois *keyframes*, maior a distância entre eles.

Uma determinada função real e um conjunto domínio, formam um espaço de distância se quatro condições são satisfeitas [20]. As condições para que se possa estabelecer se uma determinada função consiste em um espaço de distância são [20]:

Positividade: se a distância de um ponto a outro dentro do conjunto domínio é sempre um valor positivo e de um ponto para ele mesmo é zero.

Positividade restrita: se a distância entre dois pontos do conjunto domínio é igual a zero então esses pontos são o mesmo ponto.

Simetria: se a distância de um primeiro x a um ponto y é igual à distância do ponto y ao ponto x. Sendo ambos os pontos pertencentes ao conjunto domínio.

Inequação do triângulo: se a distância de um ponto x a um ponto y é menor ou igual à distância do ponto x a um ponto z e do ponto z ao ponto y. Sendo os pontos x, y e z pertencentes ao conjunto domínio.

A distância utilizada aqui foi a distância euclidiana[20], que obedece às quatro condições citadas. Em um determinado *keyframe* pode-se dizer que a posição de um vértice v é definida pelo trio de valores (vx_1, vy_1 , vz_1) valores estes associados aos eixos canônicos x, y e z respectivamente. No *keyframe* seguinte a posição do vértice v é dada pelo trio (vx_2, vy_2 , vz_2) nas mesmas condições citadas com relação aos eixos canônicos. É interessante notar que essas são posições finais do vértice v para cada *keyframe*, já considerando a influência da junta (ou juntas) a qual ele está associado. A distância euclidiana entre os valores em cada um dos eixos é dada por:

$$dv_{12} = \left((vx_1 - vx_2)^2 + (vy_1 - vy_2)^2 + (vz_1 - vz_2)^2 \right)^{1/2}$$

Onde dv_{12} é a distância entre o vértice v no *keyframe* 1 e o mesmo vértice v no *keyframe* 2. Ao somar as distâncias de cada vértice entre o *keyframe* 1 e o *keyframe* 2 obtemos a distância entre os dois *keyframes*. A figura 2 mostra a distância entre dois *keyframes*, os pontos azuis são os vértices do personagem.

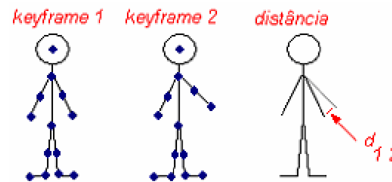

Figura 2. Distância entre dois *keyframes*

A escolha da utilização do somatório das distâncias de todos os vértices foi feita para evitar distorções quando poucos vértices estivessem muito distantes, mas os *keyframes* não fossem muito diferentes entre si. A figura 3 mostra que um vértice não é suficiente para definir a distância entre dois *keyframes*. Pode-se observar que embora o vértice da mão esquerda tenha se movido mais no *keyframe 2*, o *keyframe 2'* é mais distante do *keyframe 1* pois outros vértices contribuíram para o cálculo da distância.

Depois que as distâncias são calculadas para cada segmento (entre os pares de *keyframes* contíguos) pode-se associar um valor normalizado ao maior e menor valores para que todos os valores estejam dentro de uma faixa pré-definida que fará uma correspondência de maiores valores de distância com maiores valores de número de *frames* intermediários.

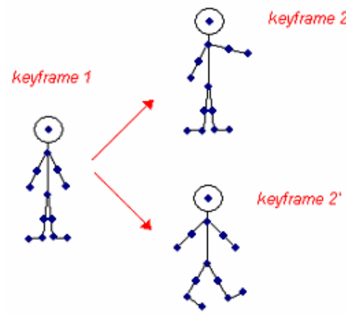

Figura 3. Contribuição de todos vértices para o cálculo da distância

3.2 CONTROLE BASEADO EM EVOLUÇÃO

Existem dentre os princípios da animação aplicados a criação de animações em 3D [4] dois princípios, *timing* e evolução, que podem ser dissociados do estágio de *design* e podem ser tratados no estágio intermediário de configuração para posterior utilização no estágio de execução. Eles estão diretamente relacionados com o número de *frames* intermediários em cada segmento de uma seqüência de animação e por isso, podem ser implementados através da definição correta do número de *frames* intermediários em cada segmento. Assim é possível que o profissional de *design* de animação crie uma animação e depois faça um ajuste fino através da manipulação dos números de *frames* intermediários, como é feito em pacotes de animação dotados de ferramentas de edição de *time-editing* [5] [6][21].

Evolução é a característica de não uniformidade de um movimento causada pela aceleração ou desaceleração do corpo, influência de alguma força externa, ou algum tipo de comportamento que o *designer* deseja enfatizar. O exemplo mais comum de evolução é o chamado *slow in* e *slow out* que é característica de movimento de iniciar lento, acelerar em seu ápice e depois desacelerar novamente[4][5][6][22]. Um exemplo pode ser visto na figura 4. Pode-se observar que a animação foi concebida com 5 *keyframes* e depois com a inserção de *frames* intermediários o efeito de *slow in* e *slow out* foi conseguido, considerando-se tempo constante para representação de cada *frame*. A percepção final é de que o começo e o fim da seqüência de animação demoram mais do que os movimentos intermediários, isto é, parece que leva mais tempo ir de do *keyframe* 1 (*kf 1*) para o *keyframe 2* (*kf 2*) do que ir deste para o *keyframe* 3 (*kf 3*). O mesmo ocorre do meio para o fim.

É importante observar que a idéia da evolução não é, como pode parecer a princípio, contrária a idéia da uniformidade apresentada na seção anterior. A uniformidade corrige erros de continuidade não desejados ao passo que a evolução insere diferenças de continuidade desejáveis para a representação do movimento. As duas idéias são verdade complementares.

Outros exemplos de evolução poderiam ser um comportamento atenuado mostrando cansaço ou algum tipo de comportamento errático como um tique nervoso ou uma aceleração mostrando pressa, medo e fuga.

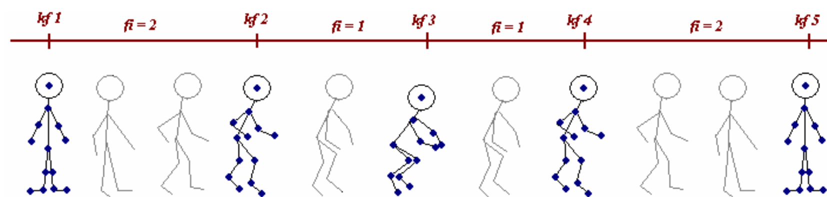

Figura 4. *Slow in* e *Slow out*

Segundo [12] que chama o *slow in* e *slow out* de *easy in* e *ease out*, existem funções de interpolação apropriadas para simular esse tipo de variação no número de quadros intermediários da animação. Algumas curvas podem ser convenientemente modeladas para proporcionar o efeito desejado como curvas senoidas e parabólicas [12]. Também curvas cúbicas (*splines*) definidas como em [8] podem ser utilizadas para a criação do efeito de *slow in* e *slow out* [4]. Com as curvas senoidas, parabólicas e cúbicas, podem ser encontrados valores nos quais se podem basear os números de *frames* intermediários conforme a função definida.

Por exemplo a função senoidal:

`F(x) = 1 - seno (π x)` ,onde x varia de [0 , 1].

A função parabólica

$F(x) = 4x^2 - 4x + 1$,onde x varia de [0 , 1].

E a função *spline* cúbica[4]

$$Q_i(x) = \sum_{k=0}^{3} p_{i+k} B_k(x)$$

,onde x é definido no intervalo 0 <= x <= 1 com as funções de base B_k (x) dadas por (k = 0, ...,3):

$$B_0(x) = \frac{(1+x)^3}{6},\ B_1(x) = \frac{3x^3 - 6x^2 + 4}{6},\ B_2(x) = \frac{-3x^3 + 3x^2 + 3x + 1}{6},\ B_3(x) = \frac{x^3}{6}$$

Com os pontos p_1, p_2, p_3 e p_4 da função Q_1, arbritados com os valores 0, 0.5, 05 e 0, tem o comportamento mostrado na figura 10.

Na implementação em que se baseia este trabalho, os segmentos de uma determinada seqüência de animação foram normalizados para serem associados a valores dentro do intervalo [0 , 1]. Assim, se a animação possui 35 segmentos o primeiro deles está associado ao valor 0 e trigésimo quinto ao valor 1. Dessa forma um determinado segmento está associado ao valor:

```
(a - min) / (max - min)
```

Onde a é o valor n-ésimo do segmento e min é valor mínimo do intervalo (0) e max é o valor máximo do intervalo (1).

Outras abordagens como curvas *splines* parametrizadas (*TCB-Splines*) [4][23], possibilitariam ao *designer* a escolha da função que melhor se adapte ao efeito de evolução pretendido.

3.3 CONTROLE ADAPTATIVO BASEADO NA TAXA DE QUADROS POR SEGUNDO DA APLICAÇÃO

Timing é a característica de velocidade de um movimento [4][5][6][24]. Algumas das informações mais importantes para o expectador de uma animação são dadas através da escolha correta do *timing*. Ele pode influenciar a visualização do movimento uma vez que o tempo deve ser suficiente para que a seqüência seja percebida pelo expectador.

Também pode influenciar na noção de peso do objeto uma vez que objetos mais pesados têm maior inércia e costuma executar movimentos mais lentos do que objetos mais leves. Além disso, o *timing* também é capaz de proporcionar a noção de tamanho e escala de um objeto ou

personagem. O estado emocional de um personagem também pode ser bem definido com o ajuste da velocidade de suas ações. Um exemplo da influência do *timing* no estado emocional de um personagem foi dado por [4] e é transcrito a seguir:

Apenas duas poses de uma cabeça, a primeira pose mostrando a cabeça inclinada na direção do ombro direito e a segunda pose com a cabeça sobre o ombro esquerdo e seu queixo levemente levantado, podem ser utilizados para comunicar múltiplas idéias, dependendo inteiramente do timing utilizado. Cada quadro intermediário adicionado entre estes dois "extremos" dá um novo significado ao movimento.

Número de quadros intermediários	*Signficado do movimento*
0	*O personagem foi atingido com tremenda força. Sua cabeça foi quase que arrancada.*
1	*O personagem foi atingido por um tijolo, um rolo de macarrão ou por uma frigideira*
2	*O personagem tem um tique nervoso, um espasmo muscular ou uma contração involuntária*
3	*O personagem está se esquivando de um tijolo, um rolo de macarrão ou de uma frigideira*
4	*O personagem está dando uma ordem seca "Vamos, Mexa-se"*
5	*O personagem é mais amigável "Ei, venha aqui", "Venha para cá"*
6	*O personagem viu uma menina bonita ou o carro sport que ele sempre quis*
7	*O personagem tenta ver melhor alguma coisa*
8	*O personagem está procurando a manteiga de amendoim sobre uma prateleira na cozinha*
9	*O personagem move a cabeça pensativamente*
10	*O personagem estica um músculo dolorido*

Finalmente a questão do *timing* passa pela necessidade de interferência do *designer* para definir o tempo de duração da animação através de um fator global o qual será posteriormente influenciado pelos demais controles.

Como o desempenho computacional (aqui se considerado em função do número de *frames* por segundo) de uma aplicação 3D pode variar muito, se a quantidade de *frames* de uma seqüência de animação for constante, o resultado será que o tempo de duração da animação variará com a taxa de *frames* por segundo. Isso não é o desejável.

Pode-se criar uma via de mão dupla onde sobre a configuração do *designer* seja possível ao estágio de execução interferir aumentando ou diminuindo o fator de *timing* segundo seus requisitos de processamento. Assim, o número de *frames* intermediários em cada segmento é variável com o número de *frames* por segundo, em seu fator de *timing* que é um fator global para toda a seqüência (definido igualmente para todos os segmentos). Os fatores responsáveis por controle de evolução e controle de uniformidade, definidos para cada segmento, se manterão inalterados. O Algoritmo desenvolvido na implementação mostrada na seção 4, é o de controle adaptativo baseado na taxa de *frames* por segundo. Ele incrementa ou decrementa o fator de *timing* segundo a variação da taxa de *frames* por segundo.

4 IMPLEMENTAÇÃO

O projeto em que se baseia o presente trabalho teve como objetivo principal a incorporação de animações de personagens a um *framework* para criação de jogos 3D e aplicações 3D. O *framework* é o Guff (Games Uff) e o modelo de animação utilizado é o MD5.

4.1 O FRAMEWORK GUFF

O *framerwork* GUFF é formado por uma camada de aplicação e um *toolkit* [25]. A camada de aplicação determina a arquitetura das aplicações que serão criadas baseadas no *framework*. Essa camada é modelada como uma máquina de estados que permite criar aplicações decompostas em um conjunto de estados. Estados podem ser simples ou compostos (agrupamento de estados) e cada estado possui um estado pai que por *default* é um estado padrão chamado *MasterState*. Estados possuem métodos e eventos definidos na interface *abstractState*. Os eventos podem ser de três tipos: eventos do sistema, eventos do ciclo principal de execução e eventos de estado. O ciclo principal da aplicação é baseado no modelo *Fixed Frequency Uncoupled Algorithm* [26]. Esse modelo é apropriado para jogos em computadores pessoais uma vez que permite a execução da aplicação de forma determinística, ao mesmo tempo em que permite que o *loop* de renderização ocorra de forma mais veloz possível.

O *toolkit* disponibiliza uma série de funcionalidades que facilitam a criação das aplicações. O *toolkit* está subdividido em *namespaces* que agrupam funcionalidades relacionadas permitindo o acesso e a utilização de cada funcionalidade de maneira mais eficiente. As classes do *toolkit* estão baseadas em bibliotecas de utilização livre, a saber: OpenGL, GLEW, boost, lib3ds, audiere, FTGL, Lua, DevIL e SDL.

4.2 O MODELO MD5

O *framework* Guff contém funcionalidades em seu *toolkit* para a carga, renderização e interação de cenários do jogo Quake3 e de objetos modelos do 3D Studio, mas não possui funcionalidades para exibir animações de personagens[25]. Visando dotar o *framework* Guff da possibilidade de carregar, gerenciar e renderizar animações de personagens, o projeto em que se baseia esse trabalho, seguiu a linha de estudo e implementação de funcionalidades para a utilização do modelo de animação de personagens do jogo DOOM3, chamado MD5[27][28][29].

O modelo de animação MD5 foi escolhido, dentre outros, por sua capacidade de representação de animações baseadas em modelos hierárquicos e controle de pesos de vértices relacionados às juntas do modelo hierárquico (*skinning*) [27]. O modelo de animação MD5 é baseado em dois arquivos principais. O MD5 *mesh* é o arquivo que contém as informações hierárquicas do modelo e as informações de renderização que são as faces, vértices, texturas e coordenadas de mapeamento de texturas. Com esse arquivo é possível renderizar o personagem em uma posição estática padrão[28]. O MD5 *animation* é o arquivo responsável pelas informações da seqüência de animação subdivido em quadros principais onde cada junta pode apresentar variações em sua translação e orientação (rotação representada por *quaternions*) que definem sua nova posição referenciada a partir de uma posição inicial contida no arquivo, chamada *baseframe*. Também nesse arquivo estão armazenadas informações sobre volumes circundantes (*Axis Aligned Bouding Boxes*) tomados em cada quadro principal [28]. Um outro

arquivo chamado MD5 *camera* que reproduz informações de movimento de câmera não foi utilizado nesta implementação[28].

4.3 AS INFORMAÇÕES DO ESTÁGIO DE INTERMEDIÁRIO

O modelo MD5 não possui a informações sobre o número de quadros intermediários [28] uma vez que após a modelagem do movimento, todos os quadros são considerados quadros principais, onerando o tamanho dos arquivos de animação.

Uma vantagem de um estágio intermediário é a economia em armazenamento uma vez que informações de quadro principal são translações e rotações para cada junta do modelo hierárquico, enquanto que o número de quadros intermediários informa com um único número inteiro o número de poses interpoladas que serão criadas em tempo de processamento, para cada segmento.

Para a implementação do projeto em que se baseou este trabalho, foi definido um arquivo com informações de números de *frames* intermediários para cada segmento entre dois *keyframes*. O arquivo também contém a informação de controle de *timing* que é globalmente aplicado ao número de quadros intermediários, em cada segmento. Esse arquivo é configurado no estágio intermediário para que sejam definidos os controles de distância e evolução e o fator de *timing*.

No estágio de execução da aplicação, essas informações são utilizadas para a interpolação do movimento do personagem. Além disso, em tempo real o fator de *timing* é ajustado baseado na taxa de quadros por segundo, conforme foi mostrado na seção 3.3.

5 RESULTADOS

Para verificar a relevância da utilização de um estágio intermediário de configuração entre os estágios de *design* e processamento, alguns testes foram executados e os resultados são mostrados e algumas suposições são feitas sobre esses resultados.

Os testes foram feitos com animações de vários personagens do jogo DOOM3. Aqui os resultados mostrados são baseados no personagem *Archvile*, na seqüência de movimento *attack2*, mostrado na figura 5. Essa seqüência contém 35 quadros principais. As características da máquina onde os testes foram executados são: PentiumIV com 3GHz e 1GBytes de memória RAM e placa de vídeo ATTI-RADEON 9250.

O primeiro teste foi um comparativo das abordagens com e sem a uniformização de número de *frames* intermediários baseada em distâncias dos *keyframes* (conforme mostrado na seção 3.1). Na figura 6, a seguir pode-se observar as distâncias com valores normalizados para cada segmento entre pares de *keyframes* contíguos da animação estudada. A análise visual da animação com e sem o controle de uniformização mostrou que o controle de uniformização funcionou bem mostrando a seqüência animação de forma contínua do início ao fim, ao contrário da seqüência sem o controle de uniformização.

É interessante observar que talvez a não uniformização original da seqüência pode ter sido intencional. É possível que o *designer* estivesse tentando criar algum efeito de evolução deixando a seqüência não uniformizada. Esse efeito foi desfigurado pelo controle de uniformização. Uma conclusão importante a que se pode chegar a partir desse fato é que não é interessante ter evolução e uniformização em estágios diferentes. Isso porque quando o estágio de *design* transforma o efeito de evolução em quadros principais com distâncias variáveis e não em número de quadros intermediários variáveis. Por essa razão esses efeitos são desfigurados pelo

controle de uniformização no estágio intermediário de controle. Por outro lado, é possível ter evolução e uniformização trabalhando conjuntamente em um mesmo estágio. O número de quadros intermediários pode ser submetido aos dois controles e guardar as duas influências se for corretamente configurado para esse fim. De fato esses dois controles devem ser realizados de forma complementares.

A seguir foi feito um teste com uma comparação entre as três abordagens de evolução para o efeito *slow in* e *slow out*: senoidal, parabólica e cúbica. Os resultados foram satisfatórios com relação ao efeito desejado foram satisfatórios para a três abordagens. Contudo, com as abordagens senoidal e parabólica ocorreram momentos de mudança mais abrupta do movimento, o não ocorreu com a abordagem cúbica na qual a seqüência apresentou mudanças mais suaves. Na figura 7, as combinações do controle de uniformização com o controle de evolução (efeito *slow in* e *slow out*), para a seqüência de animação estudada, são mostradas.

Figura 5. *Keyframes* da seqüência de animação *attack2* do personagem Archvile

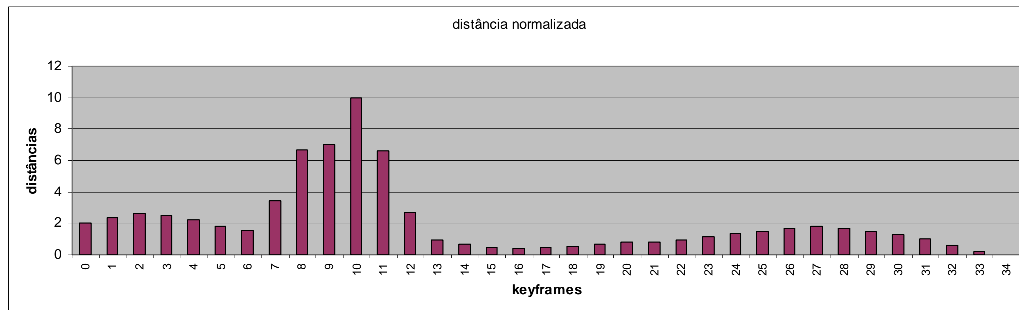

Figura 6. Distâncias de segmentos da animação *attack2* do personagem *archvile* do jogo DOOM3.

Um terceiro teste foi realizado a fim de observar a atuação do estágio de execução sobre o fator de *timing* conforme a variação da taxa de quadros por segundo da aplicação.

Utilizou-se um fator de corte de 60 *frames* por segundo, o que significa que acima desse valor o fator de *timing* não foi afetado. Abaixo desse valor, o fator de *timing* foi incrementado ou diminuído seguindo as variações da taxa de quadros por segundo da aplicação. Os gráficos da figuras 8 (a e b), mostram as variações para 7 e 10 cópias, respectivamente, do personagem se movendo sendo animados simultaneamente (para valores menores como 1, 2 e 5 cópias a taxa de *frames* por segundo se manteve superior a 60 *frames* por segundo).

Seguindo as variações de *frames* por segundo o fator de *timing* fez com que o tempo de execução se mantivesse mais estável. Contudo, não se pode garantir que esse tempo será constante uma vez que os fatores de evolução e de uniformização não variam com a taxa de

frames por segundo. Assim o número de *frames* intermediários varia com a taxa de *frames* por segundo da aplicação, mas mantém os comportamentos de uniformização e evolução durante o processo.

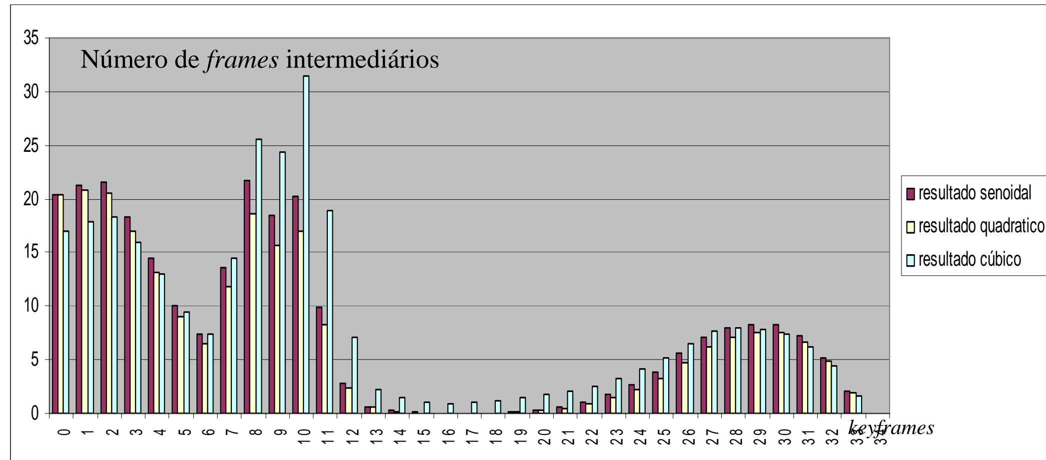

Figura 7. Combinação dos controles de uniformização e evolução

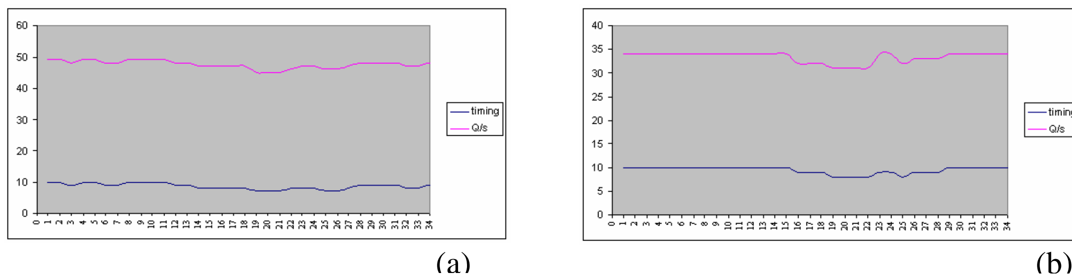

Figura 8. Variação de Quadros por segundo (Q/s) e Fator de *timing* (*timing*)

6 CONCLUSÕES

O presente artigo mostrou que pode ser interessante a utilização de um estágio intermediário, de configuração de número de *frames* intermediários, entre o estágio de *design* e o estágio de execução para a criação e execução de seqüências de animações de personagens para aplicações 3D. Esse estágio pode complementar o estágio de *design* contemplando controles relacionados uniformização, evolução e *timing*. Também pode servir para suprir o estágio de execução com informações necessárias para a interpolação de *frames* no processamento da animação e no controle do número de *frames* intermediários, baseado na taxa de *frames* por segundo da aplicação.

Uma implementação sobre a qual se baseou esse artigo foi desenvolvida permitindo a reprodução de seqüências de animações de personagens em formato MD5 pudessem ser utilizadas no *framework* Guff de criação de uma aplicação 3D. Além disso, foi desenvolvida uma ferramenta onde o *designer* pode controlar a uniformidade, criar efeitos de evolução baseados em funções paramétricas e pode manipular o número de *frames* intermediários em cada segmento definindo o *timing* da seqüência de animação. Essa ferramenta é o próprio estágio intermediário de configuração.

Testes realizados mostraram a eficácia dos controles de uniformização, evolução e *timing*. Mostraram também que os estágios de uniformização e evolução devem ser efetuados de forma complementar e que o estágio de evolução apresenta melhores resultados com funções

splines cúbicas. Finalmente os teste mostraram que o controle adaptativo da seqüência de animação no estágio de execução, baseado em *frames* por segundo, permite um controle adaptativo do tempo de duração da seqüência de animação.

Um ponto importante a ser estudado é a possibilidade de o controle sobre o número de *frames* intermediários estar relacionado ao sistema de tomada de decisão e resposta a estímulos do próprio personagem.

7 REFERÊNCIAS BIBLIOGRÁFICAS

[1] Hodgins, J.; O´Brien, J. F.; Bodenheimer, R. E. Computer Animation. Atlanta. Georgia Institute of Technology. 1999

[2] Thalmann, M. N.; Thalmann, D. Computer Animation in Future Technologies. Switzerland. Miralab. 1998

[3] Azevedo, E. Conci, A. Computação Gráfica. Teoria e Prática. Rio de Janeiro. Editora Campus. 2003

[4] Lasseter, J. Principles of Traditional Animation Applied to 3D Computer Animation. Pixar. San Rafael, California. ACM. 1987

[5] GDC 2003: The 12 Principles of Animation Applied to 3D Animation. GamaSutra. Disponível em (01/2006): *http://www.gamasutra.com/gdc2003/*

[6] Cook, S. Adding a New MilkShape3D Model to Half-Life. 2000. Disponível em (01/2006): *http://www.planethalflife.com/hlsdk2/sdk/MS3d_to_HL.htm*

[7] Watt A.; Policarpo F. 3D Games: Real-Time Rendering and Software Technology. Vol.1. New York. Addison-Wesley 2001

[8] Watt, A.; Watt M. Advanced Animation and Rendering Techniques: Theory and Practice. New York. Acm Press. Addison Wesley.

[9] Lamothe, A. Tricks of 3D Game Programming Gurus. Indiana. Sams. 2003

[10] Polack, T. Trend Polack´s OpenGL Game Programming Tutorial: Model Mania. . 2001. Disponível em (02/2006): *http://www.ic.uff.br/~mcamilo/gprogramming/tutorialdemd2.pdf*

[11] Taylor, P. Tweening Three-Way or Using Vertex Shaders. Microsoft Corporation. MSDN. 2001

[12] Parent, R. Computer Animation Algorithms and Techniques. San Francisco. Morgan-Kaufmann. 2001

[13] Camilo, M.; Martins, R; Hodge B.; Sztajnberg, A. Considerações sobre Técnicas para Implementação de Skeletal Animation em Jogos 3D. Rio de Janeiro. Cadernos do IME - UERJ. 2003. Disponível em (02/2006): *http://www.ic.uff.br/~mcamilo/gprogramming/skeletalanimcarcara.pdf*

[14] Monster, M. Sketal Animation. Disponível em (07/2005): *http://home.planet.nl/~monstrous*

[15] Anderson, E. F. Real-Time Character Animation for Computer Games. Bournemouth University. 2001. Disponível em (02/2006): *http://ncca.bournemouth.ac.uk/newhome/alumni/docs/CharacterAnimation.pdf*

[16] Lander, J. "On Creating Cool Real-Time 3D". GamaSutra. 1997. Disponível em (07/2005): *http://www.gamasutra.com*

[17] Möller, T.; Haines, E. "Real Time Rendering". Massachussets. A. K. Peters. 1999

[18] Humphrey, B. MD3 loader and MD3 animation. Disponível em (07/2005): *http://www.gametutorials.com*

[19] Perez, A. Synthetica 2.0: Software for the Synthesis of Constrained Serial Chains. Utah. Procedings of Detc'04. 2004

[20] Naylor, A. W.; Sell, G. R. Linear Operator Theory in Engineering and Science. Holt, Einenart and Winston. New York.

[21] Cantor, J. 19 Common CG Animation Pitfalls. Sonic Pictures ImageWork. 2002

[22] Lam, D. Animating for Convincing Human Motion. New School University. 2004

[23] Pipenbrick, N. Hermite Curve Interpolation. 1998. Disponível em (01/2006): *http://www.cubic.org/docs/hermite.htm*

[24] Kerlin, I. Applying the Twelve Principles to 3D Computer Animation. 2004. Disponível em (01/2006): *http://www.artof3d.com/feature.htm*

[25] Valente, L. GUFF: Um *frame*work para desenvolvimento de jogos. Niterói. UFF. 2005. Disponível em (02/2006): *http://www.ic.uff.br/~lvalente/en/projects.html*

[26] Valente, L; Conci A.; Feijó B. Real Time Game Loop Models for Single-Player Computer Games. São Paulo. Anais do Wjogos 2005 (pp. 89 a 99). SBGames. 2005

[27] Monster, M. Doom 3 Models. [S.l.:S.n.]. 2004. Disponível em (01/2006): *http://home.planet.nl/~monstrous*

[28] Md5.mesh and Md5.Anim format files. 2004. Disponível em (01/2006): *http://www.modwiki.net/wiki/Main_Page*

[29] Making Doom3 Mods: Introduction. 2004. Disponível em (01/2006): *http://www.iddevnet.com/doom3/*