SIKS - *Simple Inverse Kinematics System*

FREDERICO FERRO SCHUH[1], LEANDRO DE LIMA MARTINS[1], ISABEL HARB MANSSOUR[1]

[1]PUCRS– Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática -
Av. Ipiranga 6681, Prédio 30, Bloco 4, 90619-900 Porto Alegre, RS, Brasil
schuh@procergs.rs.gov.br, llmartins@terra.com.br, manssour@inf.pucrs.br

Abstract. The animation of computer games characters is created by artists, who use specific 3D modeling and animation software and then export it to the game. Thus, the game's characters are constrained to the artist's animation definitions, i.e., static animations. This can generate undesired situations during the course of game and a low level of interaction of the character's with the environment. In order to minimize this problem, it is necessary to implement a mechanism that allows the creations of new animations in runtime. This is the motivation to extend the functionalities of the static animation open source library known as Cal3D, which supports animations based on hierarchical structures. The main goal of this work is to present a module, called SIKS, to support the creation of dynamic animations through inverse kinematics, which is a way to create poses on a hierarchical structure. It is only required to specify the point to where a certain part of a chain of the hierarchical structure should be moved, and the positions of the other parts of the chain are automatically calculated, thus allowing the creation of dynamic poses.

1 Introdução

Nos últimos anos houve um crescimento da indústria de jogos e da pesquisa de novas técnicas de entretenimento. Atualmente, a utilização de gráficos tridimensionais é padrão para a grande maioria destas aplicações, pois, com a evolução do hardware, e, conseqüentemente, a redução de custos, o tempo de processamento das imagens está diminuindo e o seu realismo está sendo cada vez mais aperfeiçoado. A animação tridimensional vem sendo usada na indústria de jogos há bastante tempo, mas, foi com a inclusão de figuras articuladas que os jogos começaram a apresentar resultados superiores. Outras técnicas que ajudam a aumentar o grau de realismo, tal como a animação facial, também foram sendo incluídas.

Entretanto, apenas com a popularização das placas gráficas aceleradoras 3D é que os jogos tridimensionais em tempo real começaram a evoluir rapidamente, permitindo que os pesquisadores passassem a se dedicar ao desenvolvimento de técnicas de interação e de realismo. Os jogos lançados nos últimos anos permitem ao jogador interagir com o ambiente à sua volta e com outros personagens. Porém, a maioria das soluções usa animações pré-construídas para representar a interação de um personagem com um elemento.

Por exemplo, o movimento necessário para um personagem humanóide apertar um botão em um painel de controles é, em geral, realizado através da rotação do corpo do personagem para posicioná-lo de frente para o botão, seguido pela execução de uma animação pré-definida que mostra um dos seus braços movendo-se em direção ao botão, até atingi-lo com a mão. Este método é simples, mas, muitas vezes o personagem não consegue atingir o botão corretamente, pois como a animação é construída anteriormente, o botão teria que estar perfeitamente posicionado para que a mão do personagem o atingisse. Com a tecnologia atual, esta limitação pode ser resolvida, sem prender os personagens às mesmas seqüências de movimentos. Fazer com que os personagens movimentem-se de maneira única em determinadas situações é um grande avanço na geração de cenas com mais realismo.

O Cal3D [1] é uma biblioteca de código aberto que contém funções de animação de modelos articulados 3D, e que foi construída tendo como principal objetivo gerenciar toda parte de animação de personagens de um sistema de criação de *Massively Multiplayer Online Role Playing Games* (MMORPG). Assim, os personagens são modelados e animados em uma ferramenta à parte, e o Cal3D reproduz a animação dentro do jogo, utilizando as informações sobre as animações construídas previamente. Também se pode combinar mais de uma animação, resultando em movimentações e poses variadas dos personagens. Porém, uma limitação do Cal3D é não possuir funcionalidades para auxiliar a geração de animações dinâmicas, como no caso da cinemática inversa.

A partir desta deficiência do Cal3D, surgiu a motivação para criar um módulo de cinemática inversa, o SIKS (*Simple Inverse Kinematics System*). O principal objetivo deste trabalho é descrever o desenvolvimento deste novo módulo para o Cal3D, que possibilita criar animações dinâmicas utilizando técnicas de cinemática inversa. Apesar de permitir a criação de animações para esqueletos com qualquer número de articulações, os melhores resultados são

visualizados com estruturas que contenham um pequeno número de articulações, pois, com a utilização de técnicas matemáticas, de cálculos complexos, a implementação em alguns casos não gera uma animação real e consistente.

A Seção 2 apresenta alguns trabalhos relacionados, incluindo a descrição de técnicas de cinemática inversa. A descrição do funcionamento e utilização da biblioteca de animação Cal3D está na Seção 3, e a arquitetura do SIKS, bem como as técnicas e algoritmos que foram implementados, é apresentada na seção Seção 4. A Seção 5 contém as conclusões e trabalhos futuros.

2 Fundamentos e Trabalhos Relacionados

Nesta seção são apresentados alguns conceitos sobre modelos de corpos articulados e algumas técnicas existentes para cinemática inversa. Ferramentas que dão suporte à cinemática inversa também são abordadas.

2.1 Modelos de Corpos Articulados

Um personagem articulado, humanóide, ou outro objeto ou ser qualquer, é composto de um esqueleto, geralmente dividido em cadeias de *bones* (ossos) e juntas (articulações), que possuem regras que definem seu tipo e graus de liberdade. Portanto, o termo esqueleto é usado para identificar um modelo articulado, que consiste em uma coleção de nodos arranjados em uma hierarquia, ou árvore. Cada nodo representa um *bone* ou uma junta, e pode armazenar, respectivamente, informações como comprimento ou número de graus de liberdade e limitações. Apenas os nodos que representam juntas possuem filhos na árvore de hierarquia, e não podem ser nodos folha [5].

As cadeias de *bones* que formam o esqueleto são usadas para criar animações nas quais apenas alguns são movidos, por exemplo, associados a um membro do personagem articulado. Braços e pernas são exemplos de cadeias de *bones* de um humanóide. Neste caso, a árvore é consultada, e são movidos apenas os *bones* representados no ramo abaixo do nodo que corresponde ao *bone* que está sendo alterado. Assim, não é necessário trabalhar com todos os *bones* ao mesmo tempo, mas é muito importante definir corretamente qual será o nodo raiz da árvore da estrutura do esqueleto, pois uma má escolha pode trazer resultados incorretos, uma vez que ao mover ou girar a raiz, o esqueleto inteiro será afetado.

A Figura 1a ilustra a árvore de hierarquia usada para a modelagem de um personagem humano, na qual os nodos pretos representam as juntas e os cinzas representam os *bones*. O nodo raiz representa a junta da pélvis do modelo humano, de forma que quando a sua posição for alterada, todo o esqueleto também vai mudar de lugar. Um exemplo de esqueleto formado a partir desta estrutura hierárquica aparece na Figura 1b.

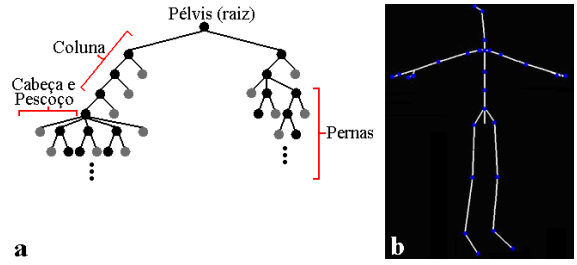

Figura 1: Hierarquia de nodos para um personagem humano (a) e o esqueleto montado a partir desta estrutura (b) [5, 9].

A junta é o componente do corpo responsável pelo movimento, permitindo um deslocamento relativo (e limitado) entre os dois segmentos a ela conectados, que é especificado através de parâmetros. Estes parâmetros são relacionados aos graus de liberdade (DOF - *degrees of freedom*) da junta e aos seus limites de movimento. Os DOFs definem os limites de movimento da junta, que correspondem ao seu movimento em torno (rotação) ou ao longo de um eixo (translação). Existem, basicamente, dois tipos de juntas: rotatórias (Figuras 2a e b), que apenas giram em torno de um ou mais eixos; e deslizantes (Figura 2c), que transladam em um ou mais eixos.

Figura 2: Tipos de juntas: *ball-and-socket* (a), dobradiça (b), deslizante (c) e universal (d) [2].

O tipo de junta rotatória mais simples, a dobradiça (*hinge joint*), possui um DOF e é tipicamente utilizada para exercer o movimento de flexão. Entretanto, pode-se construir diversos tipos de juntas combinando dois DOFs. A mais comum desses tipos de junta, conhecida como junta universal (Figura 2d), permite o movimento de balanço, ou seja, a mudança da direção do segmento para qualquer lado. Como os dois eixos de rotação são independentes neste tipo de junta, os limites podem ser aplicados para cada um separadamente. Apesar de serem muito utilizadas, as juntas universais possuem um problema quando um dos segmentos conectados à junta fica alinhado com o eixo de rotação do outro segmento, pois se perde um grau de liberdade (situação de *gimbal lock* [5]).

Juntas do tipo *ball-and-socket* (Figura 2a) possuem três graus de liberdade, sendo a mais complexa das juntas rotatórias. Permitem o movimento de torção ao redor do eixo alinhado ao segmento, e também o movimento esférico ao

redor dos outros dois eixos. Este movimento esférico, conhecido como balanço, determina a direção na qual o segmento será rotacionado. São usadas para modelar juntas mais complexas do corpo humano, tais como o ombro e o quadril. A parametrização da rotação desta junta não é uma tarefa trivial, pois três graus de liberdade podem gerar problemas na rotação do segmento.

As rotações das juntas podem ser implementadas através de ângulos de *Euler* [3], que representam a orientação absoluta de um objeto através de uma série de três rotações consecutivas, ao redor dos três eixos ortogonais (x, y e z). Porém, este método sofre do problema de *gimbal lock*, quando se perde um grau de liberdade [3, 11]. Uma alternativa é trabalhar com *quaternions*, que representam uma rotação tridimensional através de um valor escalar (ângulo) e um vetor tridimensional (eixo ao redor do qual será feita a rotação) [3, 10]. Operações entre *quaternions* possuem um custo computacional reduzido, pois não sofrem dos problemas de *gimbal lock*, e a interpolação entre *quaternions* é suave.

2.2 Técnicas e Algoritmos de Cinemática Inversa

Cinemática é o estudo dos movimentos, sem levar em consideração as forças que os causam, ou seja, preocupa-se apenas com a posição, velocidade e aceleração dos corpos. A Cinemática Inversa (IK - *Inverse Kinematics*) tem como objetivo facilitar a criação de animações, não sendo necessário especificar o ângulo de todas as articulações do personagem e trazendo a possibilidade de criar novas animações dinamicamente. Neste caso, basta definir a posição e orientação final de uma parte da hierarquia do esqueleto do personagem, tal como a mão, e as transformações necessárias para todas as articulações envolvidas são automaticamente processadas e aplicadas. Apesar da simplicidade para a elaboração de animações, a parte matemática é bastante complexa, pois o sistema gerenciador de animações fica responsável por descobrir os ângulos que as articulações do modelo precisam ser rotacionados, apenas com as poucas informações fornecidas.

Determina-se, então, a configuração das juntas para que uma tarefa seja cumprida. As tarefas são definidas pelo posicionamento e/ou orientação de *end-effectors*, que consistem em objetos anexados a uma parte de uma cadeia de *bones*. Por exemplo, as configurações das juntas correspondentes ao ombro, cotovelo e pulso devem ser calculadas para que a mão, que é o *end-effector*, chegue a uma determinada posição no espaço. Existem diversos métodos para a resolução deste problema, e alguns deles são brevemente descritos nesta seção.

Métodos analíticos tentam reduzir o problema através da resolução de uma série de equações não-lineares. O resultado é uma solução exata, extremamente rápida de ser calculada e viável para cadeias de *bones* com poucos graus de liberdade. Porém, conforme a complexidade da cadeia aumenta, fica cada vez mais difícil chegar a uma solução fechada. Métodos numéricos tentam resolver o problema através de várias iterações. As soluções são geralmente baseadas em inversão de matrizes, ou alguma forma de otimização, que apesar do alto custo computacional permitem alcançar resultados precisos.

Um método numérico bastante conhecido é o método da Jacobiana Inversa. Neste caso, uma matriz denominada Jacobiana, mapeia mudanças nas variáveis de juntas para mudanças na posição e orientação do *end-effector* [4]. Dois problemas podem ser encontrados na especificação desta matriz: situações de redundância ou peculiares (*singularities*). Uma cadeia é considerada cinematicamente redundante quando possui mais graus de liberdade do que é realmente necessário para se especificar um objetivo para o *end-effector*. Problemas peculiares ocorrem quando se tem uma configuração de juntas na qual o *end-effector* se encontra numa extremidade da cadeia e ao tentar incrementar o ângulo de qualquer uma das juntas ele se move numa mesma direção, sempre de maneira igual. A matriz Jacobiana produzida para esta configuração terá zeros na primeira linha e, portanto, não poderá ser invertida. O método chamado *Singular Value Decomposition* (SVD) é usado para resolver este problema.

O método da Jacobiana Transposta segue a mesma idéia do Método da Jacobiana Inversa, mas difere no fato de utilizar a transposta da matriz Jacobiana. Assim, evitam-se os problemas relacionados com a inversão, além de ser mais simples e menos custoso do que o método que implementa a inversa. As iterações são executadas rapidamente pela ausência das operações de inversão, porém possui uma velocidade de convergência bastante lenta, especialmente perto de uma solução. Por isso, o número de passos necessários para chegar na solução pode ser muito grande em determinados casos, além deste método não ser imune aos problemas de configurações peculiares.

Já o método numérico *Cyclic Coordinate Descent*(CCD) funciona como uma técnica de busca baseada em heurística que tenta minimizar erros de posição e orientação, variando uma junta de cada vez. As juntas são percorridas do final para o início, isto é, iniciam no *end-effector* e terminam na base da cadeia. Em cada iteração, todas as juntas são visitadas e a configuração de cada uma é atualizada de maneira a minimizar o erro. Conforme a solução é obtida para cada junta, a posição e a orientação do *end-effector* são atualizadas no mesmo momento, refletindo a mudança, e o problema de minimização vai sendo modificado. As iterações são repetidas até que o erro fique menor do que um limite mínimo pré-estabelecido, ou então atinja um numero máximo de iterações. Este método não sofre os problemas de configurações peculiares, além

de ser rápido. Porém, a taxa de convergência não é muito estável, pois depende da configuração das juntas da cadeia. Devido à sua heurística, o algoritmo faz com que as juntas mais distantes da base da cadeia se movimentem mais do que as juntas mais próximas. Na área de jogos, atualmente, este método é o mais utilizado.

2.3 Ferramentas com Suporte à Cinemática Inversa

O *Blender* [6] é uma ferramenta de criação 3D, e não uma biblioteca como o Cal3D (Seção 3), que dá suporte para cinemática inversa. Disponibiliza ao usuário uma área de edição para fazer modelos e esqueletos, e uma grande quantidade de opções para trabalhar sobre estes, permitindo uma estrutura de esqueleto montando as articulações e informando os seus limites. Uma ferramenta superior e com um custo muito elevado é o *3D Studio Max* [7], que possui funcionalidades mais avançadas para trabalhar com cinemática inversa e inclui diferentes tipos de *solvers*, que consistem em mecanismos para resolver as rotações das juntas pertencentes a uma cadeia, conforme a posição de um *end effector*, em tempo real.

Além de estar disponível nestes tipos de ferramentas, a cinemática inversa está sendo muito pesquisada tanto na indústria de jogos como no meio acadêmico. Entre os vários jogos que utilizam cinemática inversa, destacam-se o *Tomb Raider*, o *Hitman* e o *Unreal Tournament 2004*. No meio acadêmico a cinemática inversa é explorada para elaboração de novas ferramentas de uso gratuito que visam aprimorar técnicas antigas e para a criação de novas técnicas de animação. Por exemplo, três ferramentas foram implementadas com estes objetivos:

- *Sequence Editor* [4], que consiste em uma dissertação de mestrado que utiliza técnicas de IK para modelar a animação;
- *Balance* [5], criado em uma tese de doutorado, consiste em uma ferramenta de geração de animações 3D que utiliza a IK no momento da execução da animação;
- IKAN [8] (*Inverse Kinematics using ANalytical Methods*), que é um conjunto completo de algoritmos de cinemática inversa apropriado para pequenas cadeias de bones, tais como braços e pernas, que usa uma combinação de métodos analíticos e numéricos para resolver diversos tipos de problemas de cinemática inversa, incluindo, posição, orientação e limite de juntas.

3 Cal3D

O Cal3D é uma biblioteca de código aberto para animação de personagens tridimensionais baseados em esqueletos. Não é dependente de uma API gráfica, tal como OpenGL, e não é responsável pela geração de imagens, mas apenas por informar o estado dos modelos conforme as transformações vão sendo aplicadas. Para visualizar a cena 3D, a aplicação que estiver utilizando o Cal3D necessita extrair as informações dos vértices e materiais através dos métodos de consulta das instâncias das classes utilizadas. Seu princípio básico consiste em separar os dados que podem ser compartilhados entre objetos, daqueles que se referem especificamente a uma determinada instância de objeto. A biblioteca disponibiliza um conjunto de classes, chamadas *Core Classes*, que representam um tipo de modelo, e que armazenam todos os dados compartilhados entre instâncias daquele modelo. Além disso, existem as *Instances Classes*, responsáveis pelos dados compartilhados entre os objetos instanciados referentes a um mesmo modelo.

Todos os vértices de cada modelo são atribuídos a um ou mais *bones* da estrutura do esqueleto correspondente. Assim, basta mover um *bone* para que os vértices atribuídos a ele sejam alterados, mudando automaticamente a postura do modelo. Para atribuir uma malha de vértices a um esqueleto, existe um processo chamado *skinning*, no qual o animador coloca o esqueleto em uma posição inicial e a malha do modelo em uma posição semelhante, de modo que os *bones* do esqueleto correspondam às partes desejadas. Neste caso, cada vértice também será atribuído a um ou mais *bones*, que terão uma certa influência sobre o vértice, que é representada por um número entre 0 e 1, sendo que a soma de todas as influências em um vértice deve ser igual a 1. Essa atribuição, chamada de ponderação (*weighting*), tem como objetivo evitar que malhas de tecidos macios, tais como a pele, pareçam rígidas, ou então que seus vértices colidam entre si, quando o esqueleto se encontrar em determinadas posições.

As informações sobre os movimentos (animações) são armazenadas dentro do modelo base e contêm uma *track* para cada *bone* que a mesma afeta. Cada *track* contém uma lista de quadros chave para o seu *bone* correspondente, que armazenam suas transformações naquele momento. Assim, é possível combinar diferentes animações, pois é fácil saber quais *bones* serão afetados por determinada animação, que pode ser cíclica ou do tipo ação. Animações cíclicas, que são interessantes para movimentos de caminhar ou correr, ficam em um laço por tempo indeterminado, até que sejam explicitamente paradas. Já as do tipo ação ocorrem apenas uma vez, parando em seguida.

O Cal3D tem um formato próprio para descrever seus objetos que consiste em cinco arquivos com propósitos diferentes: arquivo descritor de esqueleto (.csf) para armazenar a hierarquia do esqueleto; arquivo descritor de animações (.caf) com informações sobre uma animação; arquivo descritor de malha (.cmf) com os dados da malha; e arquivo descritor de materiais (.crf) com as propriedades de materiais, tais como reflexão e especular e o fator de brilho.

Existem dois utilitários exportadores para este formato do Cal3D: um que atua como um *plug-in* para o *3D Studio Max*, e outro para o *MilkShape 3D*.

4 Módulo SIKS

O Cal3D foi desenvolvido para facilitar a criação de aplicações que utilizam animações pré-construídas, ou seja, estáticas. Porém, é interessante poder movimentar um modelo de maneira diferente do que foi definido previamente pelo animador, permitindo que o mesmo reaja de acordo com diferentes situações, e aumentando a interação entre os modelos e o mundo no qual se encontram. Além disso, uma vantagem na utilização de um sistema de animação baseado em esqueletos é a flexibilidade, uma vez que o modelo não fica limitado apenas às animações que foram criadas em um certo momento. O objetivo do SIKS, então, é estender a biblioteca Cal3D para que seja possível aplicar o uso da cinemática inversa em qualquer tipo de esqueleto, de uma forma simples e intuitiva. A Seção 4.1 apresenta uma breve descrição do SIKS e sua arquitetura. Um exemplo da sua utilização é mostrado na Seção 4.2 e a Seção 4.3 contém uma análise do SIKS, comparando-o a algumas soluções existentes.

4.1 Descrição e Arquitetura

O SIKS consiste em uma arquitetura aberta e expansível para *solvers* de cinemática inversa, sendo um *solver* o controlador que resolve o problema da cinemática inversa para uma ou mais cadeias de *bones*. A superclasse *IKSolver* (Figura 3) o representa no SIKS.

Uma deficiência do Cal3D é não possuir o conceito de DOF, isto é, não conter informações sobre a configuração de limites impostos aos movimentos dos *bones*. Na biblioteca, todas as juntas são tratadas como *ball-and-socket*, deixando muita liberdade de movimentação a um esqueleto, que pode ocasionar a geração de posturas não adequadas. Para sanar esta deficiência, foi implementado o conceito de restrições nas juntas, que no caso da animação baseada em esqueletos, impede que o esqueleto execute movimentos indesejados que podem culminar em posturas inadequadas. Neste caso, os movimentos de torção e de balanço foram separados para poder limitar cada um destes movimentos de forma independente. Portanto, é necessário quebrar a rotação do *bone* em duas rotações distintas, pois assim fica mais simples limitar a rotação da junta. Como estas rotações são representadas por *quaternions*, basta fazer os testes em cada uma delas separadamente.

A biblioteca de classes projetada para o SIKS está ilustrada na Figura 3. A classe mais importante, e considerada o ponto de partida para quem for utilizar o SIKS, é a classe *IKSolver*. Esta classe define uma interface para implementações de algoritmos de cinemática inversa, e, além disso, realiza todas as comunicações necessárias com as outras classes do SIKS para gerar as animações correspondentes a cada pose. Inicialmente, apenas o algoritmo *Cyclic Coordinate Descent* foi implementado (Seção 2.2), e se encontra na classe *CCDSolver*, que é subclasse de *IKSolver*.

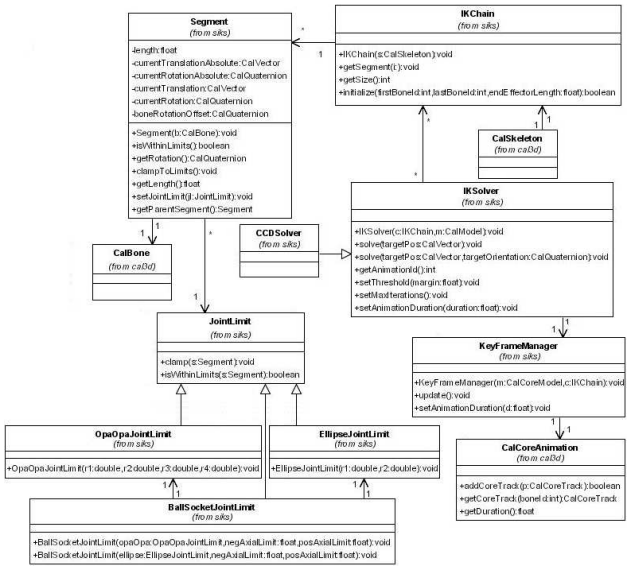

Figura 3: Diagrama de Classes do SIKS.

O SIKS trabalha com o conceito de cadeias de IK, que são representadas pela classe *IKChain*. Uma cadeia de IK é um conjunto de *bones* interconectados que serão considerados no cálculo das transformações necessárias a serem aplicadas em suas respectivas juntas, para alcançar um determinado objetivo. Para calcular as transformações é obrigatório criar uma cadeia de IK que inclua todos os *bones* a serem envolvidos. Assim, evita-se que todos os *bones* do esqueleto sejam afetados por qualquer animação gerada pelo sistema. O ideal é criar diversas cadeias de IK, uma em cada parte do esqueleto, para não afetar *bones* indesejados.

Enquanto a classe *IKSolver* trabalha sobre uma cadeia de *bones*, a classe *IKChain* trabalha com objetos do tipo *Segment* para representá-los. Basicamente, um *bone* é uma linha que parte de um ponto inicial e que possui um tamanho e uma rotação relativa à rotação do *bone* "pai". Esta representação é diferente do Cal3D, onde os *bones* são apenas pontos e não possuem tamanho. Como o *solver* necessita de um *end-effector* para trabalhar, e este normalmente se encontra na ponta do último *bone* de uma cadeia e os *bones* do Cal3D não possuem tamanho, não há como saber onde o *end-effector* se localiza. Portanto, no SIKS, quando um objeto *IKChain* é criado e inicializado, é feita uma busca pelos *bones*que fazem parte da cadeia, e os tamanhos de cada um são calculados, tomando como base as distâncias entre um e outro. Além disso, deve-se pas-

sar informações sobre o *end-effector* na inicialização de um *IKChain*.

A classe *IKChain* também é responsável pela definição dos limites impostos às juntas, que evitam que o esqueleto se posicione de forma incorreta. Estes limites de movimento são especificados nos objetos *Segment*, sendo que um *IKChain* trabalha com uma lista de objetos do tipo *Segment*, os quais representam um bone cada um. A classe *JointLimit* especifica a interface necessária para limitar o movimento. Com a arquitetura aberta do SIKS, a maneira como são definidos os limites para as juntas fica a cargo de quem for desenvolver o *solver*, não sendo obrigatória a adoção de uma estratégia. Por exemplo, se diferentes tipos de limites precisam ser desenvolvidos, basta criar uma subclasse de *JointLimit* e implementar o método *clamp()*, que tem como função alterar a orientação da junta, para que ela respeite os limites.

No SIKS, algumas definições de limites de juntas estão disponíveis. Para juntas de revolução, é suficiente definir um ângulo mínimo e máximo. Para juntas com dois graus de liberdade, duas maneiras diferentes de limite, definidas nas classes *EllipseJointLimit* e *OpaOpaJointLimit*, são implementadas. As duas seguem basicamente a mesma idéia: limitar um movimento do tipo balanço, tendo como área permitida de movimento uma elipse. Um *EllipseJointLimit* é criado a partir da informação de seus dois raios. A Figura 4a mostra uma representação gráfica deste tipo de limite em um plano 2D, na qual o ponto I viola as bordas da elipse, e, portanto, é corrigido, sendo o ponto II a nova posição correta. Um *OpaOpaJointLimit* é equivalente à união de quatro quartos de elipses, para gerar uma forma diferente. Isto significa que é necessário informar quatro raios diferentes, e cada quarto de elipse usa dois destes raios. A Figura 4b mostra a sua representação gráfica. Na figura, pode-se notar que a curva existente em cada quadrante é equivalente a um quarto de uma elipse.

Para juntas do tipo *ball-and-socket*, o SIKS implementa um limite composto, sendo a parametrização realizada através da decomposição de sua rotação em dois movimentos distintos: balanço e torção. A classe *BallSocketJointLimit* incorpora um limite para o movimento de balanço, e outro separado para o movimento de torção. No momento de criação da instância da classe, é fornecida a opção de se escolher um *EllipseJointLimit* ou um *OpaOpaJointLimit* para ser o responsável pelo limite do movimento de balanço. Para limitar a torção, informa-se apenas o ângulo mínimo e o ângulo máximo.

É possível também combinar as animações geradas pelo SIKS com as animações pré-construídas que o Cal3D já era apto a gerenciar. Isto é possível graças à classe *KeyFrameManager*, que é gerenciada automaticamente pela classe *IKSolver*. O papel desta classe é transformar as poses geradas pelo *solver* em animações na estrutura utilizada pelo Cal3D, de forma que elas possam ser executadas transparentemente, como se fossem animações estáticas. O SIKS cria *tracks* e quadros chave em tempo de execução, sendo que cada animação gerada possui sempre dois quadros chave: um especifica a posição da cadeia de *bones* antes de executar o movimento; e o outro contém as informações da cadeia de *bones* em sua posição final. Esta animação também tem associado um número identificador, podendo ser executada como se fosse uma animação normal através da classe *CalMixer*. O tempo de duração da animação também deve ser definido através de um objeto *IKSolver*. Caso isto não seja feito, a duração padrão das animações geradas é de 1 segundo.

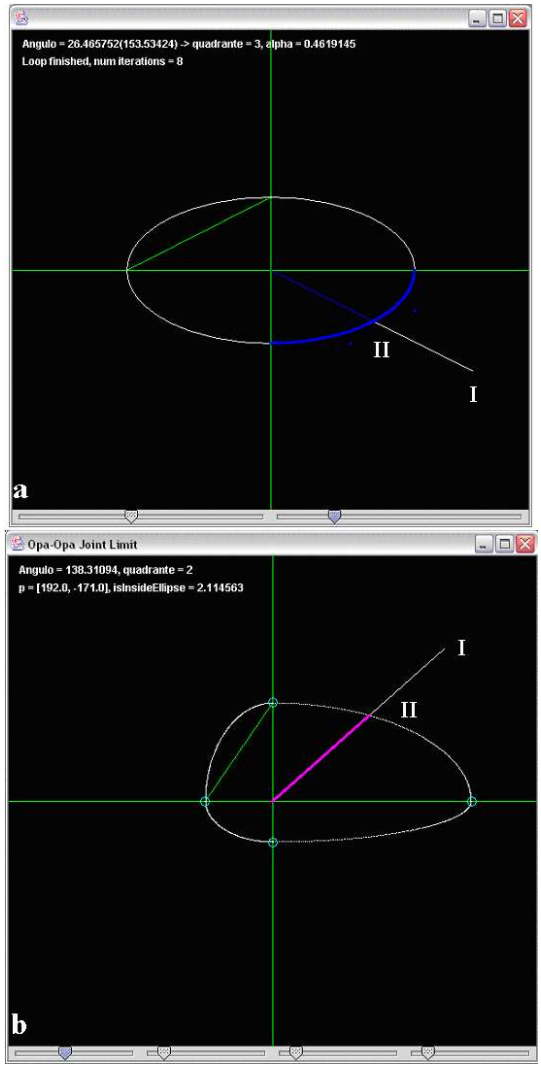

Figura 4: *EllipseJointLimit*(a) e *OpaOpaJointLimit* (b).

O SIKS gera quadros chave em tempo de execução reutilizando os mesmos objetos, ou seja, estes são mutáveis.

Isto é o que diferencia uma animação estática de uma animação dinâmica. Conforme o *solver* gera novas poses para a cadeia de *bones*, os quadros chave são atualizados, e, quando a animação for executada novamente, o resultado será percebido instantaneamente. Como o SIKS utiliza informações inexistentes no Cal3D, é necessário utilizar a API da biblioteca para fazer algumas configurações. Entretanto, para facilitar este processo, é interessante que o usuário armazene as informações em um arquivo, que não precisa ter um formato obrigatório.

4.2 Utilização

O SIKS deve ser utilizado em conjunto com o Cal3D, pois é necessário carregar os modelos através da API do Cal3D, para depois utilizá-los com o SIKS. Esta seção apresenta um exemplo simples da sua utilização, no qual é criada uma cadeia de *bones* e um *solver*, para mostrar o seu uso para gerar uma pose e uma animação. Considerando que um objeto *CalModel* já foi carregado com os dados de um modelo e inicializado, a primeira tarefa consiste em criar um objeto *IKChain*, passando o esqueleto por parâmetro, e informar os *bones* desejados e o tamanho do *end-effector*. Isto é feito da seguinte maneira:

```
CalSkeleton *p_skeleton = getCalModel()->getSkeleton();
IKChain *chain = new IKChain(p_skeleton);
bool success = chain->initialize(3, 5, 10);
```

Se *success* receber verdadeiro, significa que a cadeia foi criada com sucesso. Neste exemplo, foi criada uma cadeia do *bone* identificado pelo número 3 até o número 5, com o tamanho 10 para o *end-effector*. Se necessário, deve-se definir os limites de junta da seguinte maneira:

```
EllipseJointLimit *limit = new EllipseJointLimit(10, 15);
Segment *seg = chain.getSegment(2);//retorna o 3o. bone
seg->setJointLimit(limit);
```

Neste caso, foi definido um limite do tipo *EllipseJointLimit* apenas para o último segmento da cadeia, com o primeiro raio igual a 10, e o segundo igual a 15. O próximo passo é definir qual *solver* utilizar. Atualmente, só existe a opção do *CCDSolver* no SIKS, o qual implementa o método de *Cyclic Coordinate Descent* (Seção 2.2). No exemplo abaixo, foi definido que o tempo de duração das animações geradas pelo *solver* é 2 segundos. Este tempo de duração pode ser trocado a qualquer momento durante a execução do programa.

```
CCDSolver solver = CCDSolver(chain, getCalModel());
solver.setAnimationDuration(2);
```

Para colocar o *solver* em execução é necessário, por exemplo, executar os passos a seguir.

```
CalVector target = CalVector(10, 10, 10);
solver.solve(target);
```

Aqui foi definido como alvo o ponto (10, 10, 10), e o estado interno da cadeia foi atualizado pelo método *solve()*. A seguir os quadros chave são atualizados para refletirem a transição da pose atual da cadeia até a nova pose gerada pelo *solver*. Para executar esta animação, basta fazer o seguinte:

```
int id = solver.getAnimationId();
getCalModel()->getMixer()->executeAction(id, 0.0, 0.1);
```

A chamada para o método *executeAction()* indica que a animação deve ser executada, e os dois últimos parâmetros passados são, respectivamente, o tempo de transição entre o estado atual do esqueleto até o primeiro quadro chave da nova animação a ser executada, e o tempo de transição entre o último quadro chave até o estado atual da animação que estava sendo executada anteriormente, para que a mudança não seja brusca. O manual do usuário contém uma descrição mais detalhada sobre a API do SIKS.

4.3 Comentários Finais

O método analítico de animar é mais preciso e mais rápido do que o numérico, porém, quando a cadeia de *bones* cresce, o método analítico se torna mais complexo, exigindo novas fórmulas para a resolução dos casos. Os métodos numéricos são mais lentos e na grande maioria dos casos alcançam bons resultados, mas não são exatos. Por outro lado, podem ser utilizados em cadeias grandes, pois as fórmulas de cálculo das rotações não aumentam na medida em que a cadeia vai aumentando, o que aumenta, apenas, é o número médio de iterações que o método irá executar. Entre os métodos numéricos, os métodos da Jacobiana Inversa e da Jacobiana Transposta possuem custo computacional mais elevado, mas apresentam resultados melhores. O CCD tem o custo relativamente mais baixo, porém os seus resultados não são tão precisos, pois os *bones* da base da cadeia se movimentam menos que os da ponta. Como um dos objetivos do SIKS é ser utilizado no desenvolvimento de jogos, que necessitam de resultados rápidos, optou-se, inicialmente, pela implementação do método do CCD. Entretanto, a sua arquitetura permite que novos métodos, tal como a Jacobiana Transposta, sejam implementados.

São poucas as implementações de bibliotecas de cinemática inversa existentes hoje em dia. Uma de fácil acesso, a IKAN [8], resolve este problema através de um método analítico, porém, é limitada e só funciona em membros sim-

ples, como um braço ou uma perna. Ferramentas de modelagem tridimensional comerciais, como o 3D Studio MAX, possuem *solvers* próprios mais complexos, porém não são de código aberto. Esta é uma das grandes vantagens do SIKS, além de ser uma biblioteca com suporte a cinemática inversa, é gratuito e pode ser facilmente estendido para incluir novas funcionalidades.

5 Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado o SIKS (*Simple Inverse Kinematics System*), que consiste em um módulo criado para o Cal3D para dar suporte à criação de animações dinâmicas utilizando técnicas de cinemática inversa. Já foi implementada a técnica de CCD, que apresentou resultados bastante satisfatórios, quando comparado à alta complexidade dos algoritmos que utilizam a matriz Jacobiana. Atualmente começamos a trabalhar na extensão do SIKS para incluir o método da Jacobiana Transposta.

Para trabalhos futuros, seria interessante permitir o acréscimo de mais de um *end-effector* em diferentes partes da cadeia. Assim, seria possível especificar mais de uma tarefa de uma só vez e, por exemplo, mover *bones* localizados no meio de uma cadeia, ao invés de apenas os *bones* que se encontram no final da mesma. Isto aumentaria o nível de controle sobre o resultado final. Além disso, também seria interessante criar um *solver* que fizesse uso do IKAN internamente. Este *solver* aceitaria cadeias de no máximo dois *bones*, e atuaria como um *wrapper* sobre a API do IKAN, simplificando o seu uso, e diminuindo ao máximo possível as configurações necessárias para fazê-lo funcionar. Também é desejável que novas técnicas de limites de juntas sejam implementadas, pois as técnicas disponíveis no SIKS não possibilitam simular, por exemplo, os limites de um ombro humano.

Apesar de todo avanço, hoje em dia os jogos ainda apresentam soluções incompletas de cinemática inversa, devido ao alto custo computacional de *solvers* mais complexos. Por isso, apesar de estar ficando cada vez mais comum o uso de animações dinâmicas em jogos, estas ainda são muito simples, pois implementam apenas soluções para movimentos triviais dos personagens articulados. O SIKS constitui um primeiro esforço de extensão do Cal3D para incluir cinemática inversa, auxiliando seus usuários a implementarem programas que se beneficiam da geração de animações dinâmicas. Os resultados alcançados nesta primeira versão do módulo de IK apresentam algumas limitações, mas o projeto está disponível para o acréscimo de novas funcionalidades para aprimorar os resultados alcançados, tanto no desenvolvimento de novos algoritmos de IK, como na implementação de novas técnicas para limitar o movimento das juntas.

Referências

[1] B. Heidelberger. *Cal3D - 3D Character animation library*. Disponível em http://cal3d.sourceforge.net. Acesso: julho, 2004.

[2] R. Smith. *ODE - Open Dynamics Engine User Guide*. Disponível em http://ode.org/. Acesso: julho, 2004.

[3] R. Parent. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann., 2001, 560 p.

[4] C. Welman. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation*. 1993. Master Thesis - Simon Fraser University, Canada. Disponível em ftp://fas.sfu.ca/pub/cs/theses/1993/ ChrisWelmanMSc.ps.gz. Acesso: julho, 2004.

[5] P. Baerlocher. *Inverse Kinematics Techniques for the Interactive Posture of Articulated Figures*. 2001. PhD Thesis - Ecole Polytechnique Federale de Lauseanne, Lauseanne. Disponível em http://ligwww.epfl.ch/ baerloch/papers/thesis.pdf . Acesso: julho, 2004.

[6] *Blender*. Disponível em http://www.blender3d.org. Acesso: julho, 2004.

[7] *3D Studio Max*. Disponível em http://www.3dmax.com. Acesso: julho, 2004.

[8] *IKAN - Inverse Kinematics using ANalytical Methods*. Center for Human Modeling and Simulation, University of Pennsylvania.Disponível em http://hms.upenn.edu/software/ik/ik.html. Acesso: julho, 2004.

[9] *The Cally Demo*. Disponível em http://cal3d.sourceforge.net. Acesso: julho, 2004.

[10] M. DeLoura. *Game Programming Gems*. Charles River Media, 2000. 600 p.

[11] J. Lander. Better 3D: The Writing Is on the Wall. *Game Developer Magazine*, march, 1998. Disponível em http://www.darwin3d.com/gamedev/articles/ col0398.pdf. Acesso: julho, 2004.