UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE BELAS ARTES
ARTES VISUAIS: HABILITAÇÃO EM CINEMA DE ANIMAÇÃO

Diego de Paula Antunes

BOAS PRÁTICAS PARA A CONSTRUÇÃO DE RIGS PARA PERSONAGENS HUMANÓIDES NA TÉCNICA 3D DIGITAL

Belo Horizonte
2014

Diego de Paula Antunes

BOAS PRÁTICAS PARA A CONSTRUÇÃO DE RIGS PARA PERSONAGENS HUMANÓIDES NA TÉCNICA 3D DIGITAL

Trabalho de conclusão de curso apresentado como requisito parcial para aquisição do grau de Bacharel em Artes Visuais, com habilitação em Cinema de Animação.

Orientador: Prof. Me. Virgilio Carlo de Menezes Vasconcelos

Belo Horizonte
Escola de Belas Artes / UFMG
2014

Resumo

Na técnica de animação 3D digital, a etapa de rigging consiste em criar controles digitais para mover modelos – personagens e objetos – cuja forma foi criada digitalmente a partir de polígonos. Os controles, ou *rig*, são manuseados pelo animador e devem garantir ao modelo a mesma qualidade visual para todas as poses do objeto ou personagem em relação à forma original. Para isso é necessário análise e atenção ao se definir a forma como esse modelo e sua estrutura interna, serão deformados e manipulados. Este texto se propõe a apresentar uma série de técnicas e estratégias para a criação de *rigs* que, além de possibilitar uma forma intuitiva e eficiente de trabalho ao animador, funcionem de maneira previsível, sem distorções indesejadas de geometria.

Palavras-chave:
animação, animação 3D digital, rigging digital, personagens animados, humanoide

Abstract

In the technique of 3D digital animation, the rigging phase consists of creating digital controls to move models – characters and objects – whose shape was created digitally from polygons. The controls, also known as *rig*, are manipulated by the animator and must ensure to the model the same visual quality for all poses of the object or character in relation to its original shape. In order to achieve this, proper analysis and attention is needed to define how this model and its internal structure will be distorted and manipulated. This text aims to present a series of techniques and strategies for creating rigs that, besides enabling an intuitive and efficient way for the animator to work, operate predictably without unwanted distortions of geometry.

Keywords:
animation, 3d digital animation, rigging, animated characters, humanoid

Lista de figuras

Sumário

Introdução

Rigging é a criação dos controles de personagens na técnica 3D digital, parte de seus fundamentos foram inspirados na técnica *Stop-Motion*. Nessa técnica de animação, para que seja possível animar personagens é necessário que, nas etapas de pré-produção, o boneco a ser animado seja fabricado com uma estrutura interna articulável, equivalente a um esqueleto, que dê sustentação e o mantenha imóvel nas diferentes poses impostas pelo animador no momento do registro de cada fotograma.

Na técnica conhecida como 3D digital, existe uma disciplina especializada que lida com questões similares, inerentes ao meio virtual e necessárias para que um modelo tridimensional possa ser controlado pelo animador. No âmbito virtual, os objetos e modelos criados são equivalentes a estruturas físicas imóveis, como esculturas, só que digitais. Um personagem modelado digitalmente é um modelo estático que necessita, além de outras coisas, estruturas internas e controles manipuláveis para que sejam posteriormente usados pelo animador para lhe dar vida e movimento. Essa disciplina é o que se denomina *rigging*, que no inglês significa cordame, um termo originalmente vindo da náutica e que define, *grosso modo*, o conjunto intrincado de cordas e polias que mantém os componentes móveis e estruturais de uma embarcação constantemente ajustados e operantes. O termo, aparentemente distante é válido, pois remete adequadamente à complexidade do trabalho do artista em questão, este denominado casualmente *rigger* ou mais formalmente: Diretor Técnico de Personagem. Usaremos ao longo desses capítulos o termo *rigger* por ser um termo que independe do nível de proficiência do artista, já que o cargo de diretor técnico não é comumente relegado a iniciantes, a quem esse texto foi direcionado. O objeto resultante do trabalho de um *rigger* é popularmente chamado de *rig*, o que designa o sistema de controles do personagem como um todo, já pronto para animação, com todos seus controles e interfaces configurados e acessíveis ao animador.

Em um bom *rig* é de suma importância que a mesma qualidade visual apresentada na pose original do personagem (na qual foi concebido, modelado ou esculpido) seja mantida também em todas as poses que esse personagem venha a assumir. Dessa forma, ele deve ser articulado sem que as deformações impostas, acarretem em distorções incorretas de seu design e sua forma. Para isso é necessário análise e atenção ao se definir a forma como esse modelo e as estruturas criadas pelo *rigger* serão deformados e manipulados. Além disso, outro requisito fundamental é que o *rig* dê ao animador todo o controle necessário para posar e animar o personagem e permita fazê-lo da maneira mais intuitiva e prática o possível, mesmo com toda a complexidade inerente.

É importante ressaltar que, apesar dessa disciplina ser análoga à criação de armações na técnica *Stop-Motion*, diferenças cruciais separam o conceito de *rigging* digital do meio físico. Um delas é que, em uma produção de animação 3D digital que venha a ser realizada por estudantes ou profissionais inciantes sem muitos recursos, é provável que o animador não manipule o boneco com as mãos (como se faz no *Stop-Motion* ou em produções que possuam bonecos articulados ligados ao computador), mas interfira na pose do personagem num ambiente virtual, existente apenas no *software* de animação utilizado. Nesse contexto, o animador precisa trabalhar usando os dispositivos de entrada do computador (mouse, teclado, etc) e, em última instância, a interface gráfica do software em questão, que necessita ser configurada e complementada com controles criados pelo *rigger*. A criação de tais controles é possível utilizando recursos presentes nos pacotes de software de animação, juntamente a ferramentas que devem ser criadas pelo próprio *rigger*, a partir de conhecimentos técnicos específicos.

Outra particularidade do rigging digital se refere à necessidade do *rigger* possuir um certo nível de proficiência em áreas distintas. Mais especificamente, o *rigger* necessita ser um profissional com conhecimentos em três áreas complementares: é vantajoso que ele entenda de animação, do trabalho de um diretor técnico e de um programador.

Primeiramente, entender todas as necessidades do animador só é possível sabendo a fundo como se trabalha com animação. O *rigger* precisa entender todos os princípios de animação, entender a metodologia empregada pelo animador e como ele "constrói" o movimento do personagem passo a passo. Só assim é possível criar, de maneira consciente e inteligente, as interfaces e controles que abarcam todas as necessidades do animador.

Em segundo lugar, existe o papel de diretor técnico propriamente dito. Embora essa menção soe redundante, os conhecimentos técnicos de um rigger são extremamente específicos à sua área. Se embasando nesse corpo de conhecimento, o trabalho do *rigger* envolve compreender a maneira como se torna possível a animação computadorizada de um personagem. E requer o bom entendimento de conceitos como geometria espacial, coordenadas e transformações tridimensionais, cinemática direta e inversa, ordens hierárquicas, *constraints* de animação, deformação de malha geométrica e tantos outros termos que serão abordados adiante.

A proposta deste texto é apresentar uma série de técnicas e estratégias para a criação de *rigs* que — além de possibilitarem uma forma intuitiva e eficiente de trabalho ao animador — funcionem de maneira previsível, sem ser possível fazer o que se chama informalmente de "quebrar" o *rig*, que é quando se manipula o personagem de tal modo que ocorram distorções indesejadas de sua geometria.

Como o objetivo deste texto é servir de referência para estudantes e entusiastas da técnica 3D digital, é fundamental esclarecer que em qualquer projeto nesta técnica faz-se necessário considerar as particularidades da produção. Criar pela primeira vez um *rig* funcional não é algo que se possa alcançar sem o despendimento considerável de tempo, estudo e trabalho. Caso o projeto cinematográfico em questão não disponha de um *rigger* hábil, ou ao menos um membro da equipe que possa dedicar-se às responsabilidades do *rigging*. Pode ser muito vantajoso, e até necessário, optar por utilizar um *rig* "pré-fabricado". Existem na internet, vários sistemas gratuitos ou pagos de criação de *rigs* automatizados. Com eles, é possível se criar um *rig* que obedeça as proporções do personagem em

questão, apenas definindo o posicionamento das articulações e fornecendo parâmetros como: quantidade de dedos nas mãos e pés, número de articulações para o tronco, pescoço e outras opções. Esses sistemas oferecem uma opção viável para uma produção de animação na técnica 3D digital onde a equipe, tempo ou orçamento sejam limitados. Na prática, esse é justamente o caso quando se trata de animações feitas por estudantes ou profissionais com experiência limitada.

Entretanto, para os interessados em conhecer mais sobre as técnicas de rigging digital para personagens humanoides o texto será vantajoso por conseguir solucionar algumas questões problemáticas que confundem e frustram *riggers* em potencial. Devido à natureza intrincada e complexa do *rigging*, há problemas difíceis de serem antecipados e por isso não comumente elucidados. Eles permeiam o trabalho do rigger, revelando erros apenas nas etapas finais do processo, onde se torna muito mais oneroso encontrar soluções e acarretam trabalho e tempo perdido. Ainda assim, acreditamos que os conhecimentos e práticas defendidos nesse texto darão ao leitor uma boa compreensão dos aspectos gerais envolvidos na criação de *rigs* de qualidade, tornado o processo de *rigging* mais fluido e previsível.

1 Conceitos do universo 3D fundamentais ao *rigging*

Para lidar com *rigging* é preciso conhecer os fundamentos que regem o universo da técnica 3D digital. Eles estão sujeitos à forma como computadores funcionam e, mais especificamente, à forma como dados são manipulados dentro dos softwares de animação.

Os procedimentos e metodologia descritos nesta monografia foram inicialmente desenvolvidos através do uso do *software Autodesk Maya*. Entretanto, apesar da maioria das figuras utilizadas mostrarem a interface do *Maya*, esse texto se propõe a fornecer ensinamentos sem se limitar a softwares específicos. Assim, houve um esforço extra para buscar elucidar as técnicas e nomenclaturas equivalentes utilizadas nos softwares *Blender* e *Autodesk 3ds Max*, quando aplicáveis.

Em *rigging*, existem várias maneiras de se alcançar um mesmo resultado visual. Assim, foi abordado um conjunto de estratégias que podem ser combinadas de várias formas para permitir flexibilidade e suprir eventuais limitações dos *softwares*.

No que se refere à técnica de *rigging*, ao se comparar os *softwares,* pode-se afirmar que o acesso ao funcionamento interno do software tem prioridade sobre a simplicidade da interface do mesmo. Isso corrobora com a ideia de que um *rigger* precisa possuir a faceta de programador, pois para se criar as ferramentas necessárias ao animador, deve-se primeiro entender mais sobre funcionamento das ferramentas do *software* utilizado.

1.1 O paradigma da orientação a objetos aplicado à técnica 3D digital

Nos *softwares* de animação 3D os objetos são criados e manipulados no ambiente virtual, agrupados em cenas. Essas cenas consistem em conjuntos de modelos geométricos (cenografia e personagens), luzes e câmeras. Por sua vez, os modelos geométricos possuem materiais e esses materiais geralmente possuem texturas. Essa é uma visão geral e bastante simplificada do que consiste uma cena em *softwares* 3D. Todos os *softwares* analisados são similares nessa organização, pois seguem o paradigma da programação orientada a objetos, uma abstração que trata os dados e procedimentos de um código de programação como objetos do mundo real.

> Na POO o programador é responsável por moldar o mundo dos objetos, e explicar para estes objetos como eles devem interagir entre si. Os objetos 'conversam' uns com os outros através do envio de mensagens, e o papel principal do programador é especificar quais serão as mensagens que cada objeto pode receber, e também qual a ação que aquele objeto deve realizar ao receber aquela mensagem em específico. (DAVID, Marcio Frayze, 2007. **Programação Orientada a Objetos: uma introdução**. Disponível em: <http://www.hardware.com.br/artigos/programacao-orientada-objetos/>. Acesso em: 17 Nov. 2014.)

Tudo o que existe numa cena 3D é considerado um objeto do ponto de vista do paradigma da POO. Cada objeto por sua vez, pode ser formado por outros objetos. Um modelo geométrico, por exemplo, é invisível sem que se tenha um material associado a ele, que determine seu comportamento quando iluminado por uma fonte de luz. Nesse respeito, a luz é também outro objeto necessário para visualização de qualquer outro objeto numa cena. Embora o foco deste texto não seja a modelagem, iluminação e texturização de objetos no ambiente 3D digital, é importante ressaltar que objetos em uma cena podem possuir ligações com outros objetos. Essas ligações fazem com que um objeto afete o outro, fornecendo-lhe as propriedades que o definem.

Sendo assim, todo objeto 3D possui atributos associados a ele que definem sua forma geométrica, sua posição, seu tamanho, o material associado a ele, entre outros. Ainda mais importante é entender que esses atributos podem ser conectados uns aos outros, permitindo que o comportamento de um afete o outro. Isso é essencial para o *rigger*, que criará meios para que atributos de um objeto afetem outros objetos e permita com isso facilitar o trabalho do animador em criar a ilusão de movimento de um personagem. Entre os atributos mais importantes para o *rigger* estão a posição, orientação e tamanho de objetos.

1.2 O espaço tridimensional e seus objetos

Para se conceber objetos numa cena 3D, há de se definir o espaço que essas entidades habitam. Este espaço é definido pelos três eixos cartesianos: X, Y e Z. Assim como um plano cartesiano bidimensional tem sua origem na interseção dos eixos X e Y, o ponto de origem da cena 3D é representado pela coordenada 0, 0, 0 na interseção entre os eixos X, Y e Z.

Todo objeto criado dentro da cena possui coordenadas que o localizam dentro do espaço tridimensional. Essas coordenadas expressam sua posição relativa ao ponto de origem. Além da posição (e a transformação equivalente denominada translação), a orientação (e a transformação de rotação) e o tamanho (e a transformação de escalonamento) são os atributos que definem o objeto em 3

dimensões — todos eles expressando valores **relativos** ao ponto de origem, ou ao que chamamos de “espaço global” (mais sobre isso adiante). O conjunto desses atributos é chamado de *transform*, pois representa as transformações geométricas primitivas do objeto[1]. Sabemos que o *transform* por si só não define, por exemplo, se um objeto é um cubo ou uma esfera, ele apenas define as transformações desse objeto como um todo, mais especificamente, ele age sobre o ponto pivô do objeto. O pivô é o ponto axial em torno do qual todas as transformações do objeto são consideradas e realizadas. Por exemplo, para um objeto esférico (como o planeta ou uma bola de futebol) é natural que o ponto pivô fique ao centro. Para objetos que compõem uma dobradiça, já é útil que o pivô fique sobre o eixo de rotação. Sendo assim, o posicionamento e orientação do pivô afetam diretamente as transformações realizadas nos objetos tridimensionais.

1.3 Hierarquia entre objetos, espaço global e espaço local

O conceito de hierarquia, no âmbito do universo 3D, está diretamente ligado ao *transform* dos objetos em cena e à capacidade de se criar hierarquias com esses objetos. Para ilustrar essa ideia, consideremos um espaço imaginário contendo uma pessoa e um automóvel: Quando a pessoa se encontra fora do automóvel podemos dizer que ela e o carro se encontram no espaço global — independentes um do outro — e ao mudarem de posição ou direção, essas transformações serão realizadas no sistema de coordenadas global, referente ao ponto de origem dessa “cena”. Se dissermos agora que a pessoa se encontra dentro do veículo, é certo pensar que sua posição e direção serão afetadas pela posição e direção do veículo. Dessa forma, as transformações da pessoa estarão atreladas às do veículo. Dizemos nesse contexto que o carro e pessoa agora possuem uma relação hierárquica de **pai-filho** (respectivamente). Ademais, o sistema de coordenadas dessa pessoa não é mais relativo à cena, pois agora é o carro que representa o centro do seu sistema de coordenadas. Enquanto a pessoa se mantiver

1 As transformações geométricas primitivas se referem a funções matemáticas usadas internamente pelo *software* de animação para modificar objetos no espaço tridimensional.

imóvel dentro do veículo, suas transformações (os valores de seu *transform*) permanecerão inalteradas, mesmo que o veículo mude várias vezes sua posição e direção. É importante notar que a pessoa ainda mantém a capacidade de se movimentar e mudar de direção dentro do carro. Aqui, as transformações da pessoa ocorrem no que chamamos de “espaço local” ou “espaço do objeto” e são relativas as coordenadas da pessoa dentro do carro, não no contexto da cena como um todo.

A criação de relações pai-filho é possível entre quaisquer objetos na cena 3D, e é um dos fundamentos do trabalho do rigger. Para isso não é necessário que um objeto envolva a geometria do outro ou que estes estejam sequer próximos um do outro. Em relação aos sistemas de coordenadas, é importante saber que os *softwares 3D* permitem manipular objetos tanto no espaço global quanto no espaço local. Basta escolher em qual “modo” ou “espaço” se deseja transformar o objeto. Dessa forma, ao se manipular um objeto que é filho de outro usando o *modo global*, as transformações do objeto filho serão realizadas em referência ao sistema de coordenadas da cena, desconsiderando os valores relativos ao objeto pai.

1.4 Objetos “vazios”: *nulls*, *helpers*, *locators* e *empties*

Os *softwares* 3D também oferecem um tipo especial de objeto que pode ser identificado com diferentes terminologias: *null* significa nulo; *helper,* ajudante; *locator*, localizador e *empty*, vazio, em traduções livres a partir da língua inglesa. Embora a nomenclatura varie, os termos representam objetos com propósitos semelhantes. Eles não possuem geometria — como ocorre com malhas tridimensionais, por exemplo — e podem ser vistos apenas no *viewport*, mas não no *render*[2].

2 De maneira resumida, considera-se *render* o resultado do processamento das informações de luzes, formas, materiais e texturas de uma cena 3D em arquivos digitais de imagem e/ou vídeo. Podemos dizer que esse processo é análogo a capturar "fotos" da cena virtual, gerando imagens que — possivelmente após mais edições e retoques — compõem aquilo que, em última instância, é exibido ao espectador ou consumidor final do trabalho artístico.

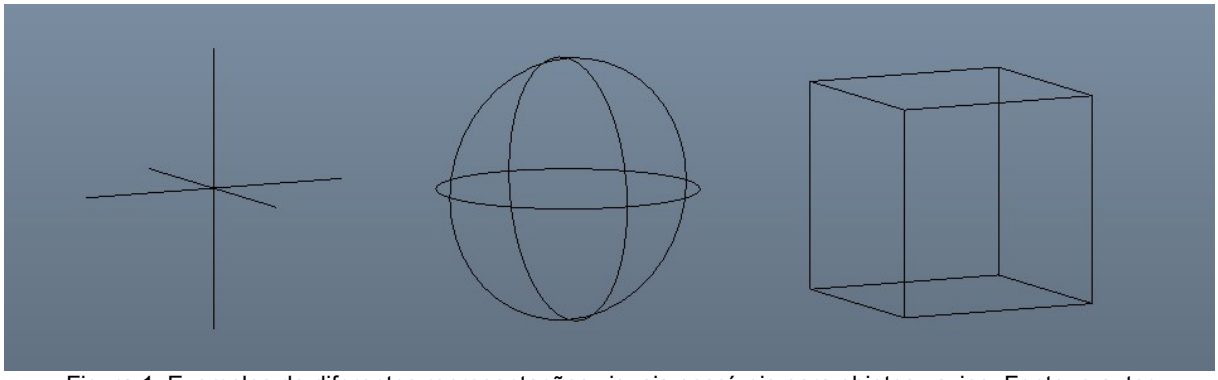

Figura 1: Exemplos de diferentes representações visuais possíveis para objetos vazios. Fonte: o autor

O uso desse tipo de objeto é geralmente facilitar o controle hierárquico de outros objetos. Por exemplo, é possível tornar outros objetos de uma cena hierarquicamente vinculados ao objeto *null*, facilitando a transformação de todos como um grupo. Outro uso comum, especialmente para o rigger, é especificar diferentes pontos pivô de transformação de outros objetos.

1.5 O conceito de *Offset*

Partindo das ideias abordadas sobre hierarquia é possível inferir que, se houver dois objetos com relação hierárquica pai-filho, ao definir como zero as coordenadas do objeto filho este terá as mesmas coordenadas do objeto pai. Ao se fazer isso utilizando um objeto vazio como “pai” — permitindo definir as coordenadas do seu objeto filho (ou objetos filho) como zero sem que este retorne ao ponto central da cena — esse objeto vazio é normalmente chamado *offset*. Existem várias razões para utilizar essa estratégia em *rigging*, uma delas é definir posição e orientação iniciais para os controles do personagem, para que estes tenham os valores de seus *transforms* zerados quando na pose neutra. Dessa forma é possível retornar à pose original do *rig* definindo como zero os valores dos *transforms* dos controles. Além desse motivo, existe o fato de que muitas vezes é necessário automatizar certas ações usadas com frequência ou que envolvam a manipulação de várias partes separadas — como por exemplo o abrir e fechar das mãos do personagem — a fim de acelerar o trabalho do animador. É importante ressaltar que, ainda que estes movimentos sejam automatizados, é preciso dar ao animador a

capacidade de ajustar e refinar manualmente as poses geradas para não limitar sua expressividade. Sendo assim, é possível utilizar *offsets* para receber a animação automatizada, e deixar os objetos filhos livres para receber a animação feita pelo animador, permitindo que ele trabalhe a partir do resultado gerado automaticamente.

1.6 *Constraints*

Constraint pode ser traduzido livremente como restrição e refere-se a métodos aplicados às transformações de um objeto para restringir sua gama de transformações (seu *transform*). Para se utilizar um *constraint* é necessário um objeto que seja responsável por controlar outro. Dessa forma, o *transform* de um objeto controla o *transform* do outro. Com o uso de *constraints*, é possível que o objeto controlador altere diferentes propriedades do objeto controlado sem afetar suas relações pai-filho.

Uma diferença crucial é que na relação pai-filho, o objeto filho ainda retém seu *transform* livre para ser manipulado. Usando *constraints*, os valores dos atributos do *transform* do objeto afetado são fornecidos diretamente pelo objeto que o controla e não podem ser editados pelo usuário — mais um caso em que o uso de *offsets* se torna útil. Outra diferença entre a relação pai-filho, é que os *constraints* não necessitam influir sobre o *transform* do objeto como um todo. É possível criar *constraints* que afetam apenas um atributo, como posição, orientação ou tamanho, individualmente. Além de usar o atributo de um objeto para afetar o mesmo atributo de outro, existem outros tipos de *constraints* que conectam tipos diferentes de transfomações. Um bom exemplo é o *constraint* que faz com que a posição de um objeto afete a orientação do outro. Este é chamado de *track to*, *aim* ou *look at constraint* (com traduções livres do inglês como rastrear, mirar ou olhar para algo, respectivamente) e é normalmente usado, por exemplo, para controlar a direção dos olhos de um personagem. Nesse caso, o objeto restritor é um controle para o qual os olhos ficam sempre apontados, e ao se movimentar esse controle, os olhos se orientam acompanhando-o.

1.7 *Joints* e *Bones*

Joints e *Bones* são termos intercambiáveis que significam respectivamente articulações e ossos. Na anatomia humana, esses termos designam coisas distintas mas no meio digital são sinônimos, pois abrangem tanto a articulação propriamente dita quanto a representação visual do osso que conecta uma articulação à outra.

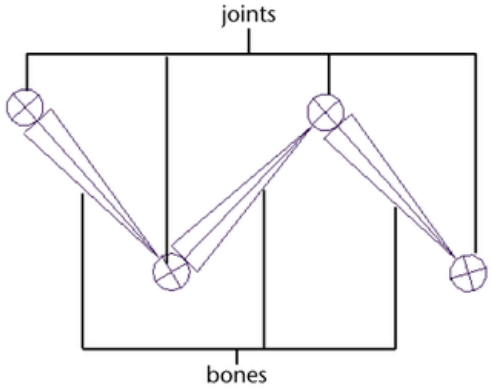

Figura 2: Representação visual de Joints e Bones. Fonte: http://download.autodesk.com/global/docs/maya2014/en_us/

Maya User's Guide: Character Animation > Character Setup > Skeletons > Skeleton Components > Joints and bones.

Acesso em: 20 Set. 2014.

Será adotado termo *joint* por sua função na movimentação de estruturas, já que seu pivô é localizado justamente no ponto de articulação.

As *joints* formam o esqueleto do personagem, definindo todos os seus pontos de articulação. O esqueleto é criado na chamada “pose neutra”, ou seja, a pose original na qual o personagem foi modelado. O esqueleto é a estrutura que modifica interativamente o modelo do personagem, sua malha geométrica[3]. É a partir das *joints* que muitas das deformações necessárias à animação se tornam possíveis. Conceitualmente a *joint* é também um objeto vazio: não possui geometria

3 A malha geométrica de um objeto é o conjunto de vértices, arestas e faces polígonais que definem a forma de um objeto. Os vértices são conectados uns aos outros por arestas e as arestas, por usa vez, enclausuram faces poligonais. A modelagem poligonal é a técnica de utilização desses elementos para criação de formas geométricas em computação gráfica e que recebem o termo malha geométrica.

real, apenas uma representação visual no *viewport*. Possui, entretanto, atributos extras que implementam alguns conceitos abordadas anteriormente como a noção de hierarquia e *offset*.

O esqueleto é formado por cadeias hierárquicas de *joints,* a joint a partir da qual se ramificam todas as cadeias do esqueleto é chamada de *root joint* (articulação raiz) e é a *joint* pai de todas as demais. Cada elo da cadeia (cada *joint)* está um nível abaixo na hierarquia em relação à *joint* anterior. Ou seja, quando se cria uma cadeia de *joints*, cada nova *joint* criada torna-se filha da *joint* anterior e a anterior torna-se pai da recém-criada, assim sucessivamente.

Outra conveniência é a implementação nativa do conceito de *offset.* Ao se criar o esqueleto do personagem e as *joints* que o compõem, é desejável que essas *joints* — por estarem em sua posição neutra — tenham valores zero no atributo rotação de seu *transform* para que seja fácil reposicionar o esqueleto em sua pose original a qualquer momento usando o valor zero. Normalmente isso seria raro já que, naturalmente, muitas *joints* do esqueleto de um personagem humanoide aponta em uma direção diferente da anterior. Por essa razão, o objeto *joint* possui um atributo extra de orientação que guarda sua rotação relativa à *joint* pai e serve como *offset* para que os valores de rotação de seu *transform* sejam iguais a zero em sua pose neutra.

No caso de personagens humanoides, as *joints* costumam ser a base para a criação do *rig.* A partir delas são criados os sistemas mais elaborados, como os de cinemática inversa (a seguir) que permitem, em algumas situações, controlar de forma mais eficiente as poses e o movimento do personagem.

1.8 Cinemática Direta e Inversa, ou *FK e IK*

Nos estudos de mecânica, a cinemática é a disciplina que se ocupa, além de outras coisas, em descrever o movimento e a trajetória de pontos no espaço. Ela é comumente denominada geometria do movimento. A cinemática direta ou *FK* (do inglês: *forward kinematics*) é a forma como as cadeias de *joints* se comportam por padrão, ou seja, respeitando a ordem hierárquica das *joints.* A *joint* mais acima da

hierarquia impõe suas transformações espaciais à *joint* seguinte. Logo, as *joints* abaixo na hierarquia acompanham o movimento da *joint* que a precede. Para definir a pose do personagem no "modo" *FK* é necessário mover ou rotacionar suas articulações, começando com as *joints* mais acima na cadeia, seguindo em direção às extremidades.

Por contraste, a cinemática inversa ou *IK* (do inglês: *inverse kinematics*) é baseada em translação e permite inverter a ordem da hierarquia das cadeias de *joints*, movimentando diretamente a extremidade mais abaixo da cadeia para que as *joints* superiores a acompanhem. No *rig* humanoide, essas extremidades são normalmente as mãos e pés do personagem, de modo a evitar que o animador seja obrigado a rotacionar individualmente todas as *joints* dos braços e pernas respectivamente. Por exemplo, numa situação em que o personagem está gesticulando em pé, é necessário manter seus pés fixos ao chão enquanto seu corpo se movimenta. Usando *FK*, é necessário rotacionar cada *joint* da perna toda vez que houver movimento nos quadris ou tronco, para que o pé mantenha a mesma pose e não atravesse ou flutue sobre o chão. Usando *IK* é possível posicionar o pé diretamente e ter o restante das *joints* da perna articuladas automaticamente para se adequar às alterações na posição do corpo. Esse tipo de configuração é criada usando um algoritmo[4] chamado *IK-Solver* (solucionador de cinemática inversa). O *IK-Solver* funciona de forma similar a um *constraint:* primeiro se define o que se chama de *end effector* (a *extremidade atuante*) que no exemplo anterior é o pé — embora, a rigor, seja a *joint* do tornozelo, já que o pé em si não é uma articulação. Em seguida se define qual a última *joint* para cima na cadeia que será afetada pelo pé — no caso, a *joint* da perna localizada no quadril. Essas *joints* compreendem o segmento da cadeia do esqueleto que será afetado pelo *IK-Solver*. Terminada essa etapa, é criado um *objeto* denominado *IK-handle* (alça de cinemática inversa) ou simplesmente *goal* (do inglês: meta, objetivo; pois é o objeto que o sistema de *IK* tentará seguir). O *IK-handle* — seguindo a lógica dos *constraints* — restringe o

4 De maneira geral um algoritmo se refere ao conjunto de passos necessários à realização de uma tarefa. No âmbito da computação, o algoritmo é o conjunto de instruções para a resolução de problemas baseado em etapas bem definidas, que dependem do uso de operações lógicas e matemáticas.

movimento das *joints* comandadas pelo *IK-Solver*. Assim, movimentando-se o *IK-handle*, a cadeia de *joints* afetadas se articula automaticamente para posicionar a extremidade atuante (a *joint* do pé) na mesma posição do *IK-handle*. Vale notar que o animador, normalmente, não manipula o *IK-handle* em si. No *rig* finalizado, ele controlará o pé através de um controle criado pelo *rigger*, que controlará o *IK-handle* além de outros mecanismos de animação do pé.

É importante ressaltar que o *IK-handle* não fornece todos os parâmetros necessários para descrever o funcionamento completo do IK-Solver. No caso de uma cadeia de *joints* que define o movimento da perna há infinitas direções diferentes nas quais o joelho poderia se dobrar pelos cálculos do *IK-Solver*. Para definir essa direção e oferecer ao animador esse controle, é preciso um outro controlador: o *Pole Vector*.

1.9 O *Pole Vector constraint* ou *IK Target*

No estudo da geometria, é sabido que três pontos distintos definem um plano. Dessa mesma forma, as 3 *joints* da perna controladas pelo *IK-solver* (no quadril, joelho e tornozelo) definem um plano no espaço tridimensional. Outra maneira pela qual a geometria permite definir esse plano é utilizando dois vetores e um ponto.

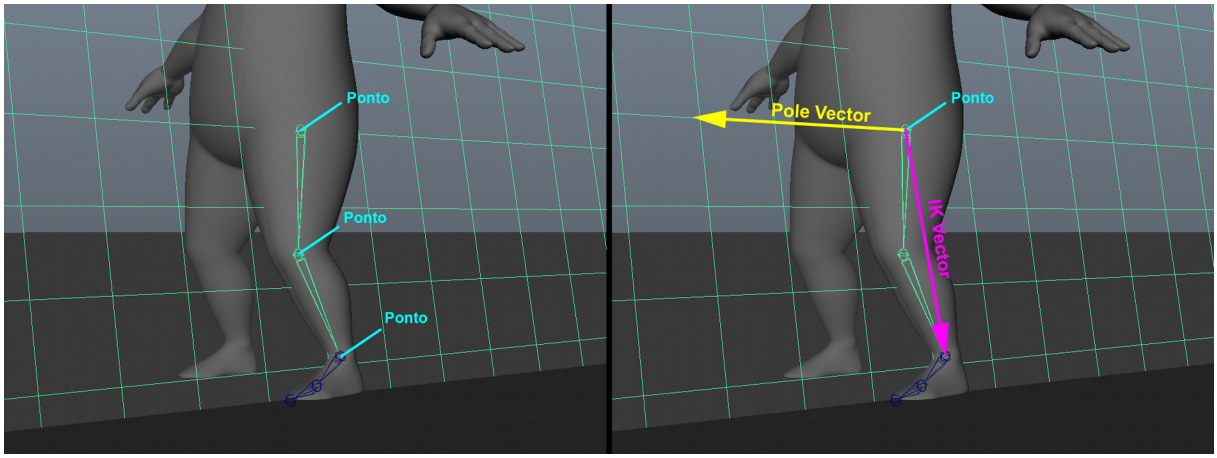

Figura 3: Exemplo de um plano formado por 3 pontos, e outro plano formado por um ponto e dois vetores.

Fonte: o autor

O primeiro vetor é o que aponta da *joint* do quadril em direção ao *IK-handle* (no exemplo, no tornozelo), chamado *ik vector*. O segundo vetor — já que se

trata de um plano — é sempre perpendicular ao primeiro. Esse vetor é chamado *pole vector,* algo como “vetor do mastro”. Pode-se imaginar o *ik* vector como um mastro com uma bandeira rígida e esticada. A direção em que a bandeira aponta (*pole vector*) indica para que lado o mastro (*ik vector*) está virado e representa também a orientação do plano formado por esses dois vetores.

É justamente a orientação desse *plano rotacional* que define para qual direção o joelho do personagem aponta. Para controlar essa direção, existem duas maneiras: a primeira é ajustar um atributo do *IK-Solver* que dependendo do software pode ser chamado de *pole angle* (ângulo do mastro), ou *twist* (torção). Entretanto, esse método não é muito previsível pois ele não define diretamente a orientação do plano, mas acrescenta ou subtrai um valor à rotação do plano, que é calculada internamente pelo *IK-Solver*. Infelizmente, dependendo do quão flexionado o joelho estiver, essa rotação pode variar drasticamente.

A forma mais eficaz para controlar a direção do joelho é usando um objeto para servir de alvo para o *pole vector*. Esse objeto alvo costuma ser um controle para ser usado pelo animador. Vale ressaltar que esse tipo de configuração é usado tanto nas pernas (controlando o joelho) como nos braços, definindo a direção dos cotovelos, sendo que cada braço ou perna possui seu respectivo controle de *pole vector*.

1.10 Objetos controle

Após a apresentação de tipos de objetos como *nulls* ou *joints*, é possível classificar os objetos quanto ao seu propósito na cena 3D. Os objetos de controle, por exemplo, são todos os objetos com os quais o animador pode interagir diretamente. Os outros objetos que compõem um *rig* são a geometria do personagem e os mecanismos internos ao *rig*. Em analogia ao uso de objetos de controle, um automóvel possui muito mais peças internas do que os controles acessíveis ao motorista. Da mesma forma, um *rig* possui um número de mecanismos inacessíveis ao animador *joints*, *offsets*, *IK-Solvers* com seus *IK-handles*, *constraints*, entre outros. O objetivo de um *rigger* é permitir ao animador

todo o controle sobre as poses necessárias à animação correta do personagem e, ao mesmo tempo, poupá-lo da complexidade dos objetos que garantem o funcionamento do mecanismo. Caso um objeto usado como parte do mecanismo seja alterado, por exemplo, pode haver rupturas indesejáveis no modo de funcionamento do *rig*.

Os controles do personagem integram boa parte da interface usada pelo animador. O design dessa interface é tarefa do *rigger* e deve ser pensada cuidadosamente para oferecer a maior ergonomia e eficiência possível, pois ela pode acelerar ou atrasar exponencialmente o trabalho do animador. Essa preocupação se dá porque a animação é um ofício repetitivo, com muitos ajustes e refinamentos e o uso constante desses controles faz com que pequenos instantes de lentidão no manejo da interface sejam multiplicados de acordo com o uso.

Os objetos de controle em si são basicamente alças de manipulação que permitem ao animador interagir com o personagem. São como objetos vazios, visíveis no *viewport* e não pelo espectador da animação final, com a diferença de que sua representação visual é criada manualmente pelo *rigger*, utilizando objetos como curvas *NURBS*[5] (apenas linhas e curvas), ou malhas geométricas. Embora seja possível usar objetos vazios (*locators* e afins) como controles, isso não é recomendado, pois as opções de representação icônica pré-definidas para esses objetos são muito limitadas. Essa representação icônica deve ser criada visando clareza no design, permitindo ao animador entender intuitivamente para que o controle serve e qual parte do corpo do personagem ele afeta.

Além do visual do controle, a posição e orientação de seu pivô é de suma importância. Ao se criar um controle para articular uma parte do corpo, é crucial que o pivô desse controle esteja perfeitamente alinhado com a *joint* da articulação correspondente. Entretanto, nem todo controle é usado para articular diretamente uma *joint* do personagem e pode ter seu pivô posicionado arbitrariamente a critério

5 NURBS é a sigla de Non Uniform Rational Base Spline, traduzido do inglês como: Base Spline Racional e Não Uniforme. Se refere ao modelo matemático — descoberto na indústria automobilística nos anos 1950 — comumente usado em computação gráfica para representar curvas e superfícies geradas parametricamente.

do *rigger*. Sendo assim, os controles de um *rig* podem exercer, além de outras funções, dois papéis importantes ao animador:

- Permitir manipular a parte do corpo correspondente, seja movimentando ou rotacionando-a. Ou seja: manipulando-se o *transform* do controle afeta-se diretamente o *transform* da *joint* que controla a parte do corpo em questão.

- Conter e agrupar atributos especias criados pelo *rigger* — chamados *custom attributes* — servindo a alguma função específica no *rig* ou de determinada parte do corpo, como um atributo especial que permite dobrar todos os dedos de uma mão ao mesmo tempo. Esses atributos especiais podem servir para acionar movimentos mais elaborados, que não consistem apenas na manipulação de um único *transform*. Podem ser criados usando expressões, *driven keys*, ou simplesmente conectando diretamente atributos. Essas técnicas serão abordadas mais adiante no texto.

Alguns controles possuem apenas a função de manipulação direta, para articular ou movimentar membros. Outros apenas contêm atributos especiais, e não são movidos ou rotacionados pelo animador. A terceira opção são controles que contêm ambas funcionalidades; exemplos comuns são os controles dos pés e das mãos, que por se tratarem de partes do corpo com muitas funções, necessitam — além de rotação e translação — atributos extras para as ações especiais que realizam.

Além dos controles usados para animar todas as partes que compõem o personagem, existe um controle principal responsável por carregar consigo toda a estrutura do *rig*. É chamado de controle raiz ou controle do personagem e é ele que que o animador usa para posicionar e orientar o personagem dentro do enquadramento da cena antes começar a animá-lo. Além disso ele é útil quando se usa ciclos de caminhada — nos quais o personagem caminha sem sair do lugar — e serve para deslocá-lo para frente enquanto o ciclo se repete.

1.11 Deformação da malha e o conceito de *Skinning*

O último conceito introdutório abordado neste texto representa a etapa final na cadeia de eventos que regem o processo de animação 3D: a deformação da malha geométrica do personagem — uma vez que o intuito maior do *rig* é manipular um modelo inicialmente estático. O esqueleto é a estrutura responsável por deformar essa geometria, e o processo de ligar um conjunto de *joints* a uma malha geométrica é chamado de *skinning* (que no contexto da animação, significa ligar uma "pele" aos ossos). Isso é possível graças a um tipo de objeto chamado *modificador* ou *deformador*, que é aplicado ao modelo do personagem e deforma sua geometria através da influência de outro objeto.

Existem vários tipos de deformadores e, no caso do *skinning,* emprega-se um deformador específico que age sobre a geometria usando um esqueleto. Em 3D, um modelo geométrico pode ser formado por polígonos: faces poligonais formadas por vértices e arestas; ou através de *NURBS:* superfícies geradas a partir de curvas — as mesmas usadas na criação de controles — que por sua vez, são definidas por pontos que controlam matematicamente sua curvatura. Dessa forma, os vértices das faces de um polígono, ou os pontos das curvas que definem uma superfície *NURBS*, são o que chamamos de componentes dessas geometrias. O deformador atua sobre esses componentes, fazendo com que cada *joint* afete um grupo específico de pontos que compõem cada área distinta da geometria. Na figura 3, indica-se como as *joints* presentes na perna do personagem afetam grupos distintos de componentes (no caso, vértices poligonais).

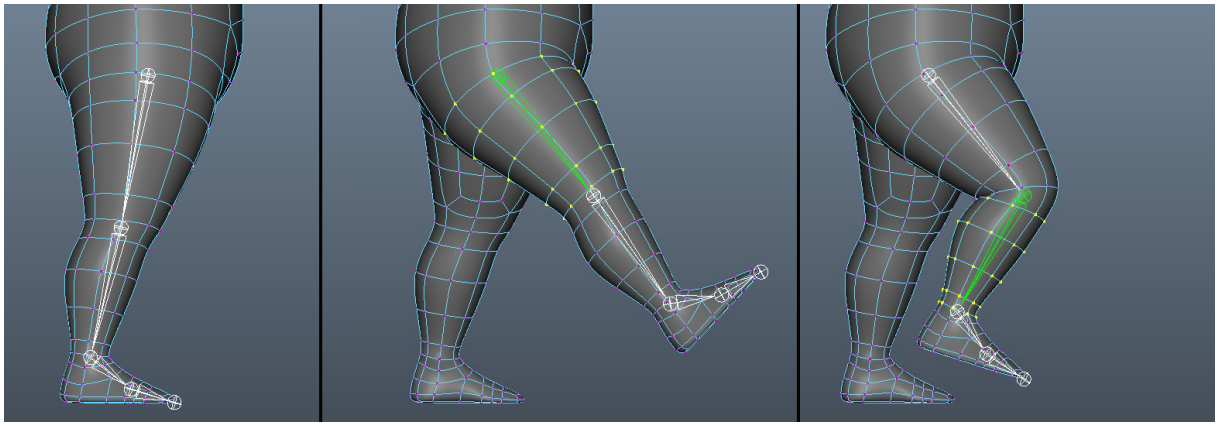

Figura 4: Exemplo da hierarquia de *joints* da perna. Cada *joint* é responsável por deformar um grupo específico de vértices da geometria. Fonte: o autor

É importante que se saiba, entretanto, que muitas vezes o esqueleto não é a única estrutura que afeta o modelo geométrico, pois existem outros tipos de deformadores utilizados na configuração de um personagem que permitem complementar e refinar as deformações obtidas através de *joints*.

2 Estratégias para a produção de um *rig* humanoide

Neste capítulo serão indicadas estratégias para a produção de controles para o corpo em um *rig* humanoide. É necessário ressaltar que as estratégias abordadas neste texto não incluem os controles para movimento da face de personagens, mas apenas seu tronco, cabeça e membros. Isso se dá pelo fato de que um *rig* facial pode ser criado e configurado de várias formas e níveis de complexidade diferentes, podendo ser igualmente boas quando atendem os requerimentos específicos de um projeto. Sendo assim, esse tema pode ser comparável em complexidade ao *rigging* de todo restante do corpo dependendo da ênfase e detalhamento que se busca alcançar nas expressões faciais de um personagem.

2.1 Pré-produção

Talvez a etapa mais comumente negligenciada por amadores durante a criação de um *rig*, a pré-produção, que aborda o planejamento do trabalho, é fundamental para o êxito do trabalho de um *rigger*. Esse planejamento inclui considerações aparentemente óbvias e talvez por isso, deixadas de lado até as

etapas da produção propriamente dita. A pré-produção deve incluir as seguintes tarefas:

- Compreensão sobre a anatomia do personagem, verificar quais são todas as regiões de articulação e definir seus pré-requisitos de animação. Deve-se analisar a fundo os *story-boards* e animáticos para ressaltar quaisquer movimentos ou poses que pareçam requerer controles mais sofisticados ou específicos a uma ação. Assim é possível limitar os recursos do *rig* apenas ao necessário e, ao mesmo tempo, não prejudicar a ação do animador.

- Deve-se também analisar o modelo geométrico do personagem e considerar a necessidade de retoques em sua malha geométrica. Primeiramente é importante pensar sobre a pose original do modelo: A prática comum de muitos modeladores é criar o modelo com braços e pernas estendidos, no que chamamos de "pose T"[6]. A vantagem disso tem a ver com a facilidade em criar as *joints* dos braços e pernas seguindo os planos ortogonais do espaço 3D (os planos perpendiculares entre si, formados pelos eixos X, Y e Z da cena). Entretanto, isso deixa o modelo do personagem em uma pose pouco natural, o que dificulta a configuração de deformações convincentes. Um dos maiores problemas dessa pose são os ombros, que ficam encolhidos devido aos braços estarem abertos, o que os torna mais difíceis de serem configurados do que numa pose mais relaxada, com os braços mais baixos. Sendo assim, é mais interessante ter um personagem numa pose mais natural, com suas articulações nem muito flexionadas nem muito estendidas. Por fim, deve-se considerar junto ao artista responsável pela modelagem que todas as regiões de articulação possuem resolução geométrica (a quantidade de vértices, arestas e faces que compõem uma determinada área da superfície do modelo geométrico) suficiente para abarcar corretamente as deformações necessárias.

6 A pose T é uma pose na qual o personagem se encontra totalmente ereto, com seus membros esticados, seus pés apontando para frente, seus braços abertos e paralelos ao chão, com as palmas das mãos apontando para baixo. Essa pose recebe esse nome simplesmente pois nela a figura humana se assemelha ao formato da letra T.

- Após verificar a gama de movimentos requiridos do personagem e a robustez de sua geometria é extremamente vantajoso buscar referências desses movimentos na vida real. Isso pode ser feito com sucesso através de filmagens de um ator, possivelmente o próprio *rigger*, executando as ações requeridas. Essas filmagens podem ser analisadas quadro a quadro para se compreender a mecânica dos movimentos em questão e se definir exatamente como criar os controles necessários. Por exemplo: o simples ato apoiar os cotovelos numa mesa requer, para o animador, que esses possam se manter fixos enquanto a mão e o ombro conectados se movimentam. Para isso é necessário um sistema de cinemática inversa pivotado no próprio cotovelo. Essa configuração não é comumente antecipada e muitas vezes se mostra necessária ao longo da produção de uma animação.

2.2 Criação consciente do esqueleto

Em animação 3D digital, quase todo *rig* possui um esqueleto, até mesmo se tratando de algo como um automóvel ou uma chaleira falante. Ele é útil para se organizar de forma hierárquica as regiões de deformação de um *rig*. No caso de um *rig* humanoide, o esqueleto criado é bastante similar ao que descreve a anatomia humana. Entretanto, as necessidades de se criar um esqueleto num *rig* diferem das razões da vida real, pois um modelo tridimensional não precisa da sustentação que o esqueleto humano oferece. No processo de *rigging*, "em vez de tentar simular a estrutura do esqueleto humano, o diretor técnico tenta simular a variedade de movimentos." (RITCHIE; CALLERY; BIRI, p. 6, tradução nossa). Sendo assim, a quantidade de articulações é definida durante a pré-produção do *rig e* varia de acordo com as necessidades e design de cada personagem.

De forma geral, em um *rig* humanoide a *root joint* (articulação raiz) é criada na região do pélvis, ou base da espinha dorsal, pois para os fins práticos é vantajoso ter o ponto primordial da hierarquia na base da coluna e próximo ao quadril e ao centro de gravidade do personagem. Isso porque permite definir um ponto inicial para a hierarquia do tronco e das pernas simultaneamente. Além disso facilita, por exemplo, centralizar corretamente a rotação do personagem no ar.

Sendo assim, a partir desse ponto são criadas todas as cadeias de articulações do corpo. A cadeia de *joints* de cada perna é composta de *joints* para o quadril, joelho, tornozelo, planta do pé (região dos metatarsos) e ponta do pé — personagens descalços devem possuir *joints* para os dedos dos pés. No tronco, a cadeia que compõe a coluna tem um número bem reduzido de *joints* em relação à anatomia humana. Algo em torno de 3 a 5 *joints* pode ser suficiente. A última *joint* da coluna corresponde a *joint* do peitoral (ou tórax). A partir dela são criadas 3 cadeias: uma no meio, que segue com *joints* para o pescoço com uma última *joint* para a cabeça (localizada na base do crânio). As outras 2 cadeias laterais que partem do tórax formam os braços e são compostas por *joints* para a clavícula, ombro, cotovelo, punho, metacarpos (geralmente do dedo mínimo e talvez o anelar) e falanges do polegar e dedos. Seguindo essa estrutura geral, o rigger deve se guiar por dois critérios durante a criação das *joints*: o correto posicionamento e a correta orientação das *joints,* para que as deformações ocorram da maneira certa. Uma representação do esqueleto descrito pode ser visto na figura 4.

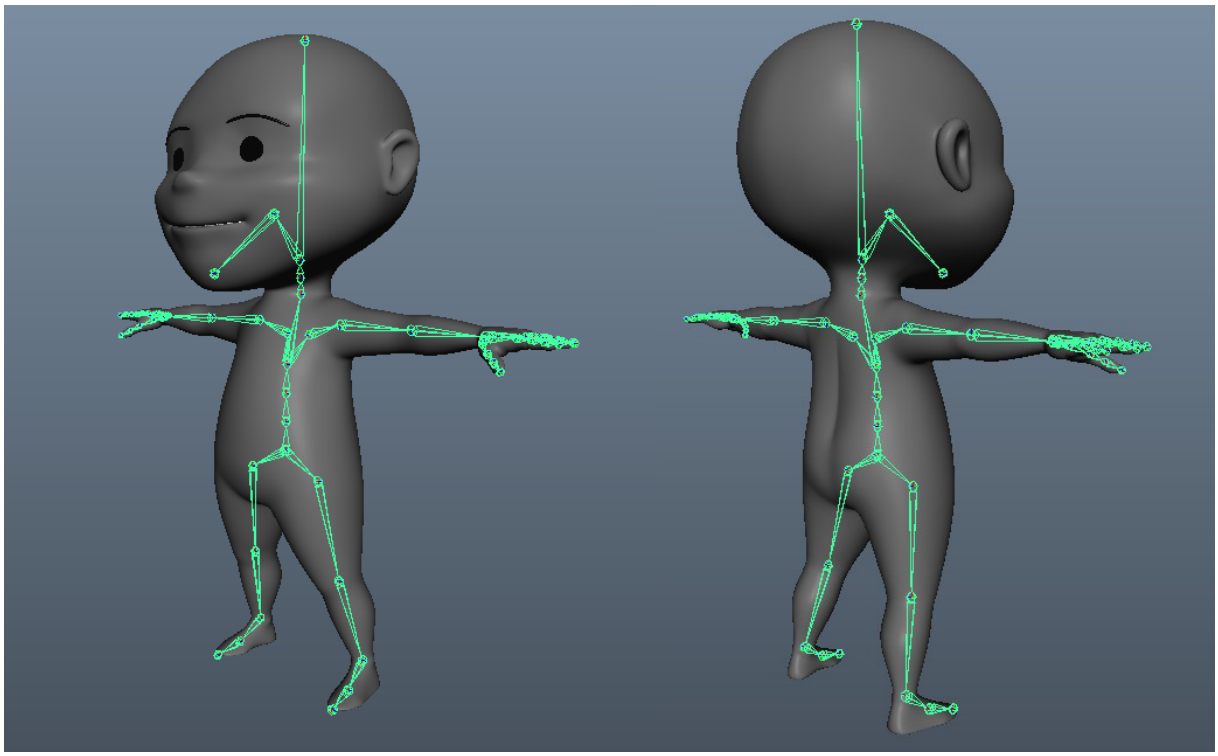

Figura 5: Exemplo da disposição de *joints* no esqueleto completo de um personagem. Fonte: o autor

Posicionar corretamente as *joints* significa inseri-las corretamente nos pontos onde se localizam as articulações no modelo. Mais uma vez, é importante se basear na anatomia humana, mas sem buscar recriá-la em todos os detalhes, pois usa-se apenas um número reduzido de articulações para tentar *simular* o comportamento equivalente no corpo humano.

O posicionamento das *joints* de braços e pernas é bastante descomplicado, pois a geometria desses membros é análoga a tubos cilíndricos — o mesmo é válido para os dedos — assim, as *joints* devem estar centralizadas no interior desses tubos e posicionadas no ponto da geometria que separa um segmento do outro: como a transição do braço para o antebraço (cotovelo), antebraço para mão (punho) etc. Além disso, devem ser criadas *joints* para os metacarpos, necessários para se curvar as palmas das mãos e permitir, além de inúmeras poses, o fechamento do punho de maneira correta.

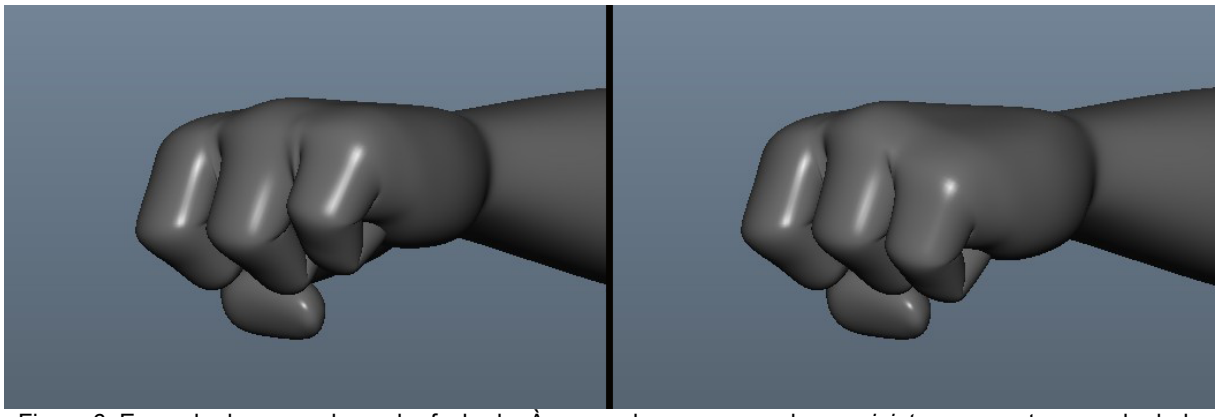

Figura 6: Exemplo de poses do punho fechado. À esquerda: sem o uso de uma *joint* para o metacarpo do dedo mínimo, impossibilitando o curvamento da palma da mão. À direita, uma pose mais expressiva graças à flexão da *joint* extra criada para o metacarpo. Fonte: o autor

O tronco é uma região que merece atenção especial: a cadeia de *joints* mais uma vez deve estar posicionada no centro da geometria, e não muito encostada na parte de trás do tronco (como acontece com as vértebras no corpo humano). Assim como ocorre com os braços e pernas, isso se deve ao fato do esqueleto de um *rig* servir como pivô das deformações da geometria, e não para sustentar o corpo. Caso essas *joints* fiquem muito próximas à superfície das costas,

as deformações comprimirão demasiadamente a região frontal do tronco quando o personagem se curvar para frente, além de expandi-la também excessivamente ao curvá-lo para trás — isso cria uma perda na manutenção de volume, uma distorção indesejada da geometria.

Outra articulação que merece atenção é a da clavícula, cujo correto posicionamento dentro da geometria do personagem não é muito facilmente visualizável pelo *rigger*. A clavícula é o osso que conecta os ombros ao tronco, ela é responsável pelo encolhimento dos ombros e por permitir elevar os braços acima da cabeça. Essa articulação fica próxima ao centro do tronco, numa altura ligeiramente abaixo dos ombros. Caso estejam postas mais acima, ao se acionar o encolhimento dos ombros, eles irão primeiro se afastar do corpo antes de começarem a se aproximar, e isso é incorreto do ponto de vista anatômico.

Após o posicionamento das *joints*, é necessário orientá-las da melhor maneira. Para isso, faz-se necessário apresentar mais informações sobre rotações em 3D. Para descrever a orientação de um objeto em relação à origem da cena ou a seu objeto pai na hierarquia de objetos, a geometria oferece diferentes modelos de representações matemáticas. O modelo mais convencionalmente utilizado, presente em qualquer *software* de animação, é o sistema de ângulos Euler[7], que utiliza 3 ângulos convenientemente nomeados X, Y e Z. Outro modelo importante, disponível em certos softwares é o de *Quatérnios* mas por não ser acessível ao animador em todos os três *softwares* analisados, não será abordado neste texto. Sendo assim, é possível corresponder cada um dos eixos Euler a um movimento específico: por exemplo, usando um eixo para levantar e abaixar os braços, outro para trazê-los para frente e para trás, e o terceiro para a torção longitudinal.

7 Ângulos Euler: Os três ângulos necessários para caracterizar a orientação de um corpo rígido no espaço e que podem ser definidos por três rotações sucessivas. [...] Esses três ângulos caracterizam univocamente a orientação do sistema final em relação ao sistema inicial. (RODITI, p. 13)

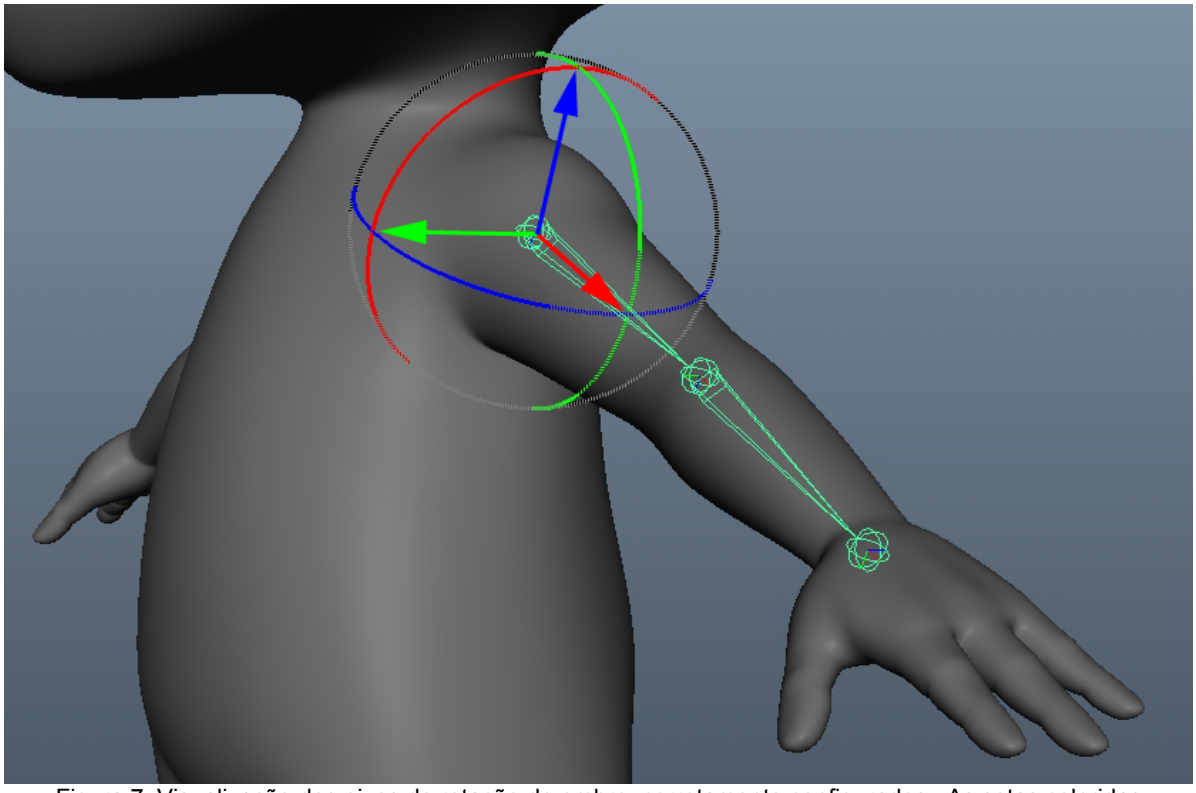

Figura 7: Visualização dos eixos de rotação do ombro, corretamente configurados. As setas coloridas representam os eixos e cada circulo corresponde ao arco de rotação do eixo de mesma cor. Fonte: o autor

Após definir o posicionamento das *joints*, já de início o eixo longitudinal — chamado *eixo primário* — estará obrigatoriamente definido, já que é o eixo que sempre aponta em direção à próxima *joint* na cadeia. Restam então dois eixos a serem orientados, sendo que — como esses eixos são perpendiculares entre si — ao se definir o eixo secundário, o eixo restante é definido automaticamente.

É importante haver consistência na escolha do tipo de movimento que é acionado por cada eixo. Como o eixo primário é obrigatoriamente responsável pela torção, é vantajoso definir o eixo secundário como sendo o da flexão, responsável, por exemplo, por dobrar e esticar braços e pernas. Isso permite ao animador flexionar esses membros rotacionando um único eixo. É importante notar também que diferentes articulações no corpo possuem diferentes *graus de liberdade*: os cotovelos, joelhos e falanges dos dedos, possuem apenas um eixo de rotação, podendo ser chamadas de *articulações de dobradiça*; enquanto ombros, quadris, e articulações da coluna, pescoço e cabeça, possuem três eixos livres — podendo girar para frente, para trás e para os lados, além de torcer longitudinalmente.

Para lidar com essas articulações mais avançadas devemos voltar à teoria por trás das rotações Euler. Primeiramente, vejamos que o modelo Euler é o mesmo usado em um mecanismo gimbal[8]: os três eixos que decompõem a orientação do objeto estão organizados de maneira hierárquica, sendo que o eixo mais externo *carrega* consigo o eixo do meio que, por sua vez, carrega consigo o mais interno.

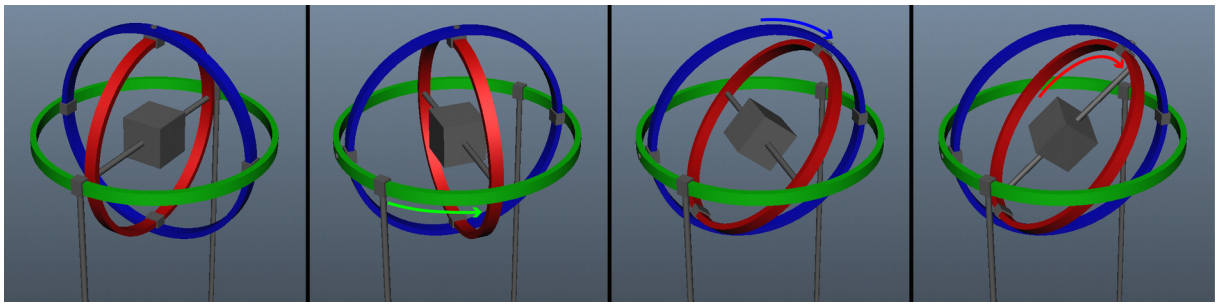

Figura 8: Exemplo de um cubo sendo rotacionado preso a um mecanismo gimbal. A figura ilustra a maneira como os eixos externos controlam a rotação dos eixos internos. Para se alcançar a orientação final do objeto, são rotacionados cada um dos eixo em sequência. Começando pelo eixo Y (em verde), seguido pelo eixo Z (em azul) e por fim o eixo X (em vermelho). Fonte: o autor

Essa hierarquia implica no que se chama de *ordem de rotação*. Significa a ordem na qual os três eixos estão conectados e define qual eixo afeta a orientação do outro. Funciona da seguinte forma: o eixo externo carrega a orientação do eixo do meio, esse por usa vez carrega a orientação do eixo mais interno. A necessidade de entender isso se deve há uma limitação das rotações Euler que causa um fenômeno chamado *gimbal lock* (travamento gimbal). Ele ocorre quando o eixo do meio é rotacionado próximo a 90 graus, isso faz com que o eixo interno fique alinhado com o externo, causando a perda de um dos ângulos de rotação. Observe o exemplo abaixo de um avião e veja que, devido à ordem de rotação, quando se rotaciona o objeto 90 graus no eixo do meio (em azul) torna-se agora impossível apontar o nariz do avião para cima ou para baixo.

8 Gimbal é um mecanismo composto por suportes articulados em eixos ortogonais, que permite, através da força da gravidade, manter um objeto nivelado com o plano horizontal terrestre. É utilizado, por exemplo, em giroscópios, bússolas náuticas e suportes para câmeras de filmagem do tipo steady-cam.

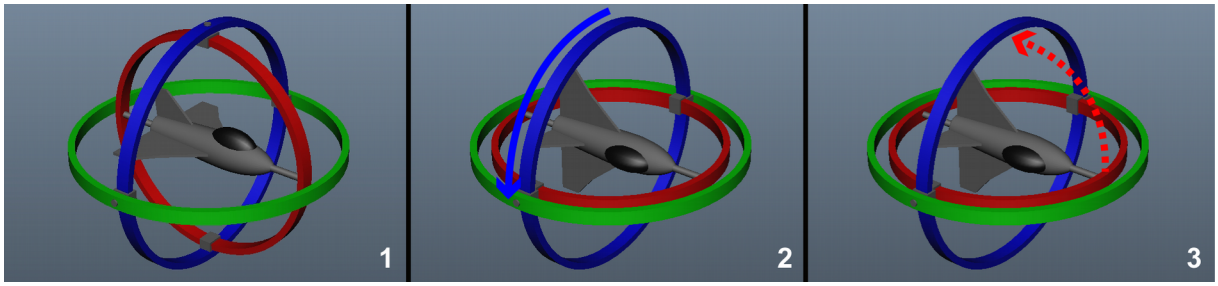

Figura 9: Exemplo de *gimbal lock*. O primeiro quadro mostra a orientação original do avião. O segundo quadro mostra travamento gimbal, resultante de um rotação de 90 graus no eixo do meio (em azul). O terceiro quadro mostra, em tracejado vermelho, o eixo no qual não se pode mais rotacionar o avião. Fonte: o autor

Para minimizar os efeitos desse problema nas articulações que possuem os três eixos livres, deve-se especificar uma ordem de rotação na qual o eixo do meio seja o responsável por controlar a ação que possui a **menor** amplitude de movimento. Um bom exemplo é o da mão, cujos movimentos de maior amplitude são o de torção e o de flexão para frente e para trás. O movimento de rotação lateral (usado para acenar com as mãos) é o que possui a menor amplitude. Assim, deve-se definir o eixo correspondente para que fique na posição do meio na ordem de rotação. Além disso, é recomendado tornar o eixo primário (responsável pela torção) o mais interno na ordem de rotação, para não se perder a capacidade de torção longitudinal sob nenhuma circunstância. Uma última recomendação geral é tornar o eixo mais externo — o que controla a orientação dos eixos mais internos — alinhado ao eixo vertical da cena, pois assim será possível rotacionar o objeto horizontalmente em qualquer direção, já que o eixo mais externo carrega consigo os restantes sem causar alinhamento entre eles.

Terminada a configuração do esqueleto do personagem, deve-se realizar um *skinning* inicial para garantir de que a posição e orientação das *joints* está deformando o modelo da maneira correta.

2.3 A criação de *pole vector constraints* sem desalinhamento do *rig*

Uma frustração que atinge *riggers* iniciantes ocorre quando, ao se definir um objeto como alvo de um *pole vector* (seja para um braço ou uma perna), percebe-se que as *joints* do membro afetado sofrem desalinhamento, mesmo que as vezes ligeiro, mudando assim a direção na qual o joelho (ou cotovelo) aponta. Como

visto anteriormente, as *joints* que formam o segmento controlado por um *IK-Solver* definem um plano no espaço tridimensional. O *pole vector*, junto com o *ik vector* oferecem uma outra forma de se definir um plano no espaço. Quando restringido, o *pole vector* muda de direção para apontar sempre para o objeto alvo. Sendo assim, o objeto alvo muda a orientação do plano definido pelos dois vetores.

Dessa forma, para se evitar o desalinhamento podemos criar o objeto alvo do *pole vector* de forma que ele esteja inserido no mesmo plano definido pelas *joints* que formam o segmento *IK.* Com o objeto alvo inserido nesse plano, o plano definido pelos vetores e o plano definido pelas *joints* estarão alinhados, assim, não haverá mudança de orientação ao se restringir o *pole vector* ao objeto alvo. A questão então é saber como inserir o objeto alvo no plano desejado. Para isso, primeiro deve-se assegurar, durante a criação do esqueleto, de que todas as *joints* do segmento *IK* possuam seus eixos secundários paralelos entre si. Em seguida, basta alinhar a posição e orientação do objeto alvo, com as da primeira *joint* do segmento controlado por *IK* (no caso da perna: a *joint* do quadril). O objeto alvo agora tem seus eixos alinhados com os do quadril. Após alinhado, pode-se mover o objeto alvo para uma posição mais adequada, afastando-a da *joint* — que está dentro da geometria do personagem — e colocando-a um pouco a frente do joelho para facilitar a visualização e acesso do mesmo pelo animador.

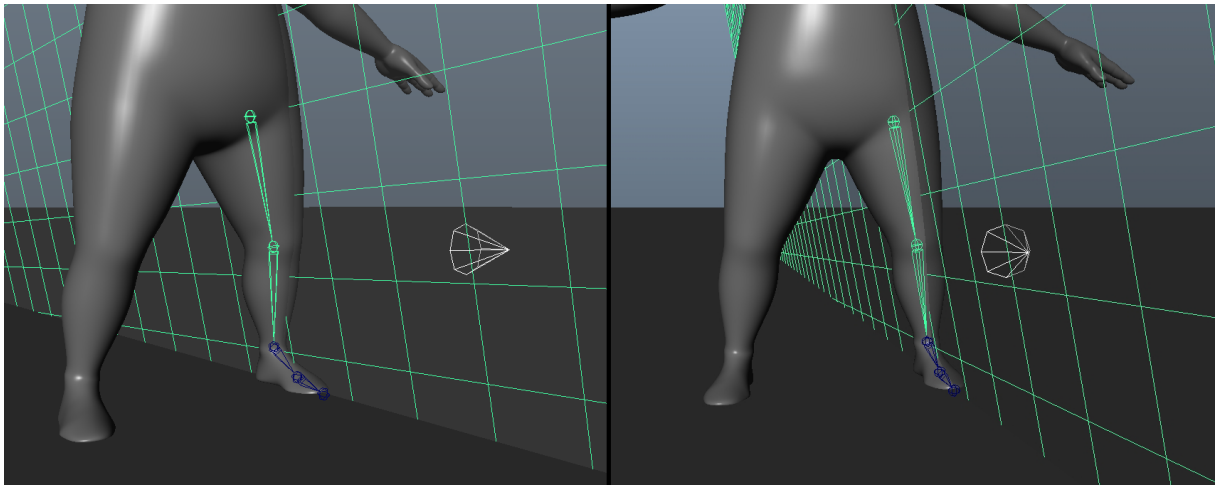

Figura 10: Visualização do plano formado pelas *joints IK* da perna esquerda (em verde), representado por linhas quadriculadas de mesma cor. O alvo do pole vector (o controle branco no formato de cone) está inserido no mesmo plano, para que não ocorra desalinhamento da perna. Fonte: o autor

É crucial nesse momento que o reposicionamento seja feito movendo o objeto alvo no eixo que aponta para *frente* da perna, sem movimentá-lo no eixo secundário, que corresponde ao movimento lateral do joelho, pois isso o tiraria do plano definido pelas *joints* da perna e a faria sair da pose neutra.

2.4 Conexões diretas entre atributos

Para a relação entre atributos, será usado o exemplo de uma técnica de simplificação dos controles dos dedos das mãos. O dedo possui 3 falanges, a falange da base (ligada diretamente ao metacarpo) possui 2 eixos de rotação: pode flexionar o dedo em direção à palma da mão e pode também ser girada lateralmente para aproximar ou afastar os dedos um do outro. As duas falanges restantes, entretanto, são *articulações de dobradiça* e só flexionam em um eixo. Ademais, desde que os dedos não estejam tocando outros objetos, a rotação dessas duas falanges ocorre geralmente em conjunto. Sendo assim, ao definir a pose da mão do personagem, a rotação que houver na falange do meio deverá geralmente ser aplicada à falange da extremidade. Ao reconhecer isso, podemos facilitar o trabalho do animador *conectando* a rotação de falange do meio à rotação da extremidade. Conectar diretamente atributos é a forma mais simples de atrelar dois ou mais objetos e não requer o uso de *constraints*. Dessa forma, o valor do atributo usado como *output* é simplesmente copiado para o atributo usado como *input*, lembrando que esses atributos podem inclusive pertencer ao mesmo objeto.

A princípio, conectar a rotação das *joints* das falanges não é recomendável, pois isso bloquearia o acesso a esse atributo na falange da extremidade e não permitiria rotacioná-la individualmente. É importante lembrar que deve-se oferecer ao animador a possibilidade de rotacionar cada falange separadamente para, por exemplo, posicionar o dedo do personagem sendo pressionado contra objetos. Ao se criar controles para cada uma das falanges o animador poderá posicioná-las individualmente. Entretanto, isso é incompatível com a automatização fornecida pela conexão de atributos. Para obter os dois modos de

comportamento — automatizar a ação e ainda permitir ao animador animar individualmente os movimentos gerados automaticamente — é possível empregar o conceito já familiar de *offset*. Dessa forma, cada falange terá um controle individual, sendo que a falange da extremidade terá, acima do controle, um *offset*. Assim, é possível conectar a rotação da falange do meio à rotação do ***offset*** da falange da extremidade. Isso criará a automação desejada e ainda permitirá manipular livremente o controle de cada falange individualmente.

2.5 Os Custom Attributes

Existem ações mais complexas que serão abordadas mais adiante, necessárias a um *rig* eficiente, que não podem ser criadas apenas conectando-se atributos diretamente. Além disso, essas ações não se relacionam conceitualmente apenas com valores de rotação ou translação, e sim com funções da mecânica do movimento humano ou específicas ao tipo de *rig* que se quer criar.

Suponha-se que, para sofisticar ainda mais o controle dos dedos da mão, deseja-se criar uma maneira de dobrar todos os dedos ao mesmo tempo, numa pose de punho fechado. Para o animador é ideal que essa ação seja acessível através do controle da mão do personagem. Sendo assim, o controle da mão receberá um *custom attribute* (do inglês: atributo personalizado), que será identificado como "fechar punho". Para criar um desses atributos, seleciona-se o objeto que o receberá (no caso, o controle da mão) e ativa-se a opção para adicionar um *custom attribute*.

É importante nomear o atributo de forma que sua função seja clara ao animador. Junto a isso, é preciso definir o tipo de valor desse atributo. Os atributos do *transform* (rotação, translação e escala) armazenam valores numéricos decimais em cada um dos seus 3 componentes (X, Y e Z). *Custom attributes*, por sua vez, podem armazenar vários tipos diferentes de valores. As opções para os tipos de valores são: *integer:* números inteiros; *float:* números decimais; *vector:* um conjunto de três números decimais; *string:* nomes ou palavras; *enum:* (enumerador) uma lista de opções (*strings)*; e *boolean:* apenas dois valores binários de 0 ou 1, verdadeiro

ou falso, ligado ou desligado. A escolha do tipo de valor depende do tipo de funcionalidade que terá o atributo criado. Além do tipo de valor, é geralmente uma boa prática definir os limites correspondentes ao valor máximo e mínimo que o atributo deverá possuir dentro do contexto de sua utilização. Lembrando que isso só se aplica a valores escalares como *integers*, *floats* e *vectors*.

No caso do fechar da mão, é ideal que se tenha um atributo que possa variar suavemente ao longo de dois limites: uma pose com dedos totalmente abertos e outra com eles fechados ao máximo contra a palma da mão. Para isso deve-se usar valores decimais. Os limites podem ser simplesmente números fáceis de lembrar, como 0 e 100. Dessa forma podemos usar esse atributo para posar a mão totalmente aberta com o valor 0 e totalmente fechada com o valor 100.

A simples criação do atributo não tem nenhuma consequência. Para que sejam de fato úteis, os *custom attributes* devem ser usados como *output* para acionar mudanças em outros mecanismos criados pelo *rigger*. Ademais, os valores do *custom attribute* muitas vezes precisam ser processados antes de serem repassados como *input* para outros objetos. No exemplo do atributo para fechar o punho, é necessário que os valores de 0 a 100 sejam modificados para corresponder aos valores exatos de rotação das falanges necessários às poses aberta e fechada.

Entretanto, antes que possamos avançar e explicitar ao leitor como realizar conexões e manipulações de atributos — necessárias às tarefas mais intrincadas que abordaremos — será interessante ressaltar algumas diferenças nas interfaces do *Maya, Blender* e *3ds Max* no que se refere à forma como estes *softwares* nos permitem conectar atributos entre objetos, a fim de esclarecer usuários de ambos sobre a localização das ferramentas utilizadas.

2.5.1 Diferenças na interface de conexões de atributos

Como já foi sugerido no texto, o *software Autodesk Maya* permite ao usuário um nível considerável de acesso a seu funcionamento interno. Isso porque sua interface e sua arquitetura giram em torno dos (já famosos) *nodes*. A exemplo disso, no *Maya* quaisquer tipos de conexões entre objetos podem ser realizadas através do *Node Editor*. Entretanto o *Node Editor* disponível no Blender e no 3ds Max — apesar de visualmente muito semelhante — permite apenas trabalhar com materiais, texturas e outros elementos de composição, mas não permite, por exemplo, acessar diretamente objetos e seus *transforms.*

Para se realizar conexões diretas entre atributos, como as do exemplo do penúltimo subcapítulo, o Maya possui uma interface própria chamada *Conection Editor* (do inglês: editor de conexão), que permite visualizar atributos de dois objetos quaisquer e conectá-los entre si. As outras opções são os *nodes* utilitários e as expressões. *Nodes* utilitários são, na verdade, um conjunto de diferentes *nodes*, cada um feito para de realizar um tipo de operação matemática. Para utilizar um desses *nodes* basta conectar o *output* de um objeto ao *input* do *node* utilitário, em seguida conecta-se o *output* do *node* — agora com o valor resultante da operação — ao *input* do atributo do objeto que se quer afetar.

As *expressões* são a segunda opção para conectar atributos de objetos. Elas permitem usar uma linguagem de programação para criar pequenos *scripts* com variáveis referentes a atributos de objetos, e executar nelas operações matemáticas da mesma forma que com os *nodes* utilitários. No *Maya*, criamos expressões através da interface chamada *Expression Editor* (do inglês: editor de expressões). Sendo assim, expressões não dependem do arranjamento de cadeias de *nodes*, e são — para um *rigger* com alguma fluência em programação — uma forma talvez mais direta de definir operações em cima do *output* de um atributo.

O *software Blender*, por sua vez, combina essas funcionalidades através de um objeto chamado de *driver*. Os *drivers* reúnem a interface usada para conectar atributos de objetos, realizar algumas operações pré-definidas (como fazem os

nodes utilitários) e permitir a criação de expressões utilizando variáveis criadas a partir de atributos de objetos.

No caso do *3ds Max*, para se conectar atributos usa-se uma interface chamada *Parameter Wiring* (do inglês: cabeamento de parâmetros) que basicamente funciona como uma combinação do *Connection Editor* e o *Expression Editor*, presentes no *Maya*. Na parte de cima existem duas janelas lado a lado onde se pode visualizar e conectar atributos de dois objetos. Na parte inferior estão os campos para a criação de expressões, que utilizam uma linguagem própria do *3ds Max* chamada *MaxScript.*

2.6 O *IK-FK switch*

Na técnica 3D digital, tanto os modos de controle *IK* quanto *FK* são necessários para abranger uma vasta gama de movimentos de personagens. Embora antagônicos em seu modo de funcionamento, ambos devem estar disponíveis para uso pelo animador. Já foram abordadas as vantagens de se animar em *IK*, mas o modo *FK* é igualmente vantajoso para animar o personagem com seus membros livres (sem ter as extremidades fixas em outro objeto). Isso permite um controle muito melhor da decomposição do movimento e da aplicação do princípio de *overlapping action*[9] nas situações necessárias.

Sendo assim, um *rig* deve permitir ao animador trocar o modo de animação (*IK* / *FK*) usado em cada membro individual. Além disso, deve permitir essa troca a qualquer momento durante uma cena, de um quadro para o outro. Isso é feito através da criação de um *IK-FK switch* (do inglês: *interruptor* de *IK-FK*). Trata-se de um *custom attribute* que permite alternar entre o modo *IK* e *FK*, através de valores que correspondam respectivamente a um dos dois modos.

9 Overlapping action (do inglês: ação sobreposta) se refere a um princípio de animação que dita que o movimento das partes de um corpo não ocorre em total sincronia. Ao iniciar ou interromper uma ação, a energia mecânica é transferida ao longo do corpo em etapas, havendo um certo atraso entre as partes. Isso resulta numa sobreposição de ações, na medida em que o início e fim da ação de cada parte não ocorre simultaneamente, e sim num efeito cascata, com o final da ação de uma parte sobrepondo o início de outra.

Apesar de ser possível realizar transições suaves entre o modo *IK* e *FK*, a opinião defendida nesse texto é que a alternância entre os modos deve ser feita de um quadro de animação para o outro, não havendo tempo de interpolação na transição para que não haja movimentos imprevisíveis para o animador. Dessa forma recomenda-se que o *IK-FK switch* seja um atributo com valores do tipo *integer* (de 0 a 1), *boolean* (também 0 ou 1), ou *enum* (com duas opções: *IK* e *FK*). Outra consideração é a escolha de qual controle usar para conter esse atributo, muitas vezes é interessante criar um controle exclusivo para o *IK-FK switch*, que esteja isolado das transformações dos membros do personagem.

Para que se possa afetar uma cadeia de *joints* em dois modos diferentes existem algumas opções. A primeira é utilizar uma única cadeia que possua um sistema de *IK*. Isso já é suficiente pois o *IK-solver* possui um atributo que permitir controlar sua influência sobre as *joints* ou simplesmente desligá-la por completo. Isso faz com que as *joints* voltem a obedecer a orientação definida por seus pais dentro da hierarquia, retomando o comportamento *FK*. Apesar de suficiente, esse método não é indicado, pois dificulta a implementação de outras funcionalidades por depender de uma única cadeia de *joints*, além de razões mais complexas devido a limitações do *IK-Solver* que vão além do escopo deste texto.

A metodologia mais segura para se criar o comportamento esperado do *IK-FK switch* envolve a criação de duas cadeias de *joints* extras, além das *joints* do esqueleto ligado à geometria do personagem. Essas cadeias extras possuem somente as *joints* correspondentes ao segmento que forma cada um dos membros que funcionarão nos modos *IK* e *FK*, ou seja, os braços e pernas. A primeira dessas cadeias extras será responsável pela animação em *IK*, com controles para o *IK-handle*, para o *pole vector* e todo restante que acompanha a configuração desse sistema já previamente abordado. A outra será responsável pela animação no modo *FK* e deverá possuir controles individuais para a manipulação de cada articulação.

Após criar essas duas cadeias, serão usados *constraints* para que se possa controlar as *joints* do segmento correspondente associada à geometria do personagem. Todos os *constraints* possuem um atributo que controla sua influência

sobre os objetos controlados e, além disso, é possível utilizar mais de um objeto para controlar outro. Assim, definem-se as duas cadeias extras de *joints* para controlar a cadeia associada à geometria através de *constraints*. Após isso, usa-se o valor do *IK-FK Switch* para controlar as influências desses *constraints.*

Sendo assim, cada um dos valores binários do *IK-FK Switch* (0 ou 1) deverá simultaneamente definir o valor da influência de uma das cadeias extras como 1 (ligado) e a influência da outra como 0 (desligado). Para isso, é possível conectar diretamente o valor fornecido pelo *IK-FK Switch à* influência de um dos *constraints*. Entretanto, esse valor deverá ser invertido antes que possa ser conectado à influência do *constraint* oposto.

Além disso, é importante reforçar que os sistemas *IK* e *FK* são acompanhados de controles exclusivos para cada modo. Sendo assim, deve-se torná-los disponíveis apenas quando o modo correspondente estiver ativado. Isso é feito de maneira simples, conectando-se os mesmos *outputs* — usados para controlar a influência dos *constraints* de cada modo — ao atributo de *visibilidade* dos controles. Esse atributo está presente em todo objeto 3D (ou seja, que habite o *viewport* da cena). Sendo assim o *output* ligado aos *constraints* do modo *IK*, será também ligado à visibilidade dos controles *IK*, e o equivalente vale para o modo *FK*.

2.7 A necessidade de isolar a rotação de partes do corpo

A cinemática direta — maneira como as *joints* se comportam por natureza — faz com que rotações no tronco do personagem sejam propagadas para seus apêndices (cabeça, braços e pernas), mas muitas vezes esse comportamento é indesejado: caso o animador queira animar o personagem arremessando um objeto com a mão; a pose final dessa ação terá o braço do personagem apontando na direção em que o objeto foi atirado. Se o animador ajustar a curvatura da coluna para enfatizar a pose, a direção do braço mudará também. Dessa forma o animador será obrigado a *contra animar*[10] a orientação do braço para compensar os ajustes

10 O termo *contra-animar* se refere ao trabalho adicional de ajustar a pose do personagem, seja articulando ou reposicionando partes do corpo, a fim de compensar ou neutralizar movimentos indesejados, gerados involuntariamente pela maneira como foi configurado o funcionamento do *rig*.

feitos no tronco. O mesmo princípio se aplica ao curvar ou girar o personagem e manter sua cabeça apontada numa direção fixa, mirando algum ponto de interesse. Para solucionar situações desse tipo é interessante oferecer ao animador a opção de *isolar* a rotação da cabeça, clavícula ou quadris. Isso impedirá que essas partes acompanhem a rotação do corpo nas situações que ele desejar.

O método para se criar esse tipo de configuração é idêntico tanto para a cabeça, braços ou pernas, e requer o uso de *offsets*, *constraints* e *custom attributes*. Será usado o controle da cabeça para ilustrar o processo: primeiramente, é necessário dar a opção de usar ou não esse sistema, para isso será criado um *custom attribute* no controle de animação da cabeça que permita ligar e desligar esse comportamento. Em seguida cria-se um *offset* do controle da cabeça que servirá para receber as futuras conexões sem afetar o *transform* do controle em si. Para definir os dois modos de orientação da cabeça — acompanhando ou não a orientação do corpo — cria-se dois *constraints* de orientação. Um dos *constraints* restringirá a orientação do *offset* do controle da cabeça à orientação pescoço. O segundo restringirá o mesmo *offset* à orientação do controle raiz (de forma que a cabeça não acompanhe a orientação do tronco), isso é feito pois, mesmo que se queira isolar a rotação do cabeça em relação ao tronco, ela deve acompanhar o controle principal do personagem, caso contrário, a cabeça ficaria torcida ao se orientar o personagem dentro da cena antes de animá-lo. Tendo criado esses dois *constraints*, basta conectar o *custom attribute* de isolamento de rotação ao valor de *influência* de cada *constraint*, isso deve ser feito de forma que o valor do atributo de isolamento ligue a influência de um *constraint*, ao mesmo tempo que desliga a influência do outro, e vice-versa. É uma lógica idêntica à usada na configuração do *IK-FK switch*.

2.8 O conceito de *Driven Keys* e o exemplo de sua aplicação

Os *driven keys* (traduzido livremente do inglês como *chaves acionadas*) são ainda mais uma maneira de se automatizar ações de um personagem, utilizando atributos de um objeto para modificar a animação de outros. A título de desambiguação, os *driven keys* no *Maya* são equivalentes aos *action constraints* do

Blender e aos *reaction controllers* do *3ds Max*. Esse método se baseia na utilização de quadros chave e curvas de animação, em vez de expressões ou operações matemáticas com atributos.

Em animação 3D digital o animador cria quadros chave para causar mudanças nas transformações do personagem ao longo da duração da cena. Cada quadro chave define valores diferentes de acordo com o valor correspondente ao tempo (em quadros) do instante atual na trilha de animação. Uma curva de animação é formada por quadros chave e pelo trajeto (a curva propriamente dita) que delineia a interpolação entre os pontos definidos por esses quadros. Por padrão, essas curvas de animação usam o valor associado ao tempo da cena como *input.*

Os *driven keys*, por outro lado, definem quadros de animação baseados não no tempo, mas sim no valor de um atributo usado como *input*. Dessa forma é possível usar *driven keys* para controlar a transição entre quadros chave de atributos de um objeto, através dos valores do atributo de outro. É possível então, utilizar valores de um *custom attribute* para percorrer através de uma animação criada a partir de *driven keys*. O objeto que possui as *driven keys* conectadas a seus atributos, será animado quando o *custom attribute* controlando a transição desses quadros chave tiver seu valor alterado. Um exemplo comum da aplicação de *driven keys* na criação de controles mais sofisticados para um personagem é a criação de um sistema de *Foot-Roll* (traduzido do inglês de maneira literal como: rolar do pé). Esse termo se refere ao movimento (semelhante ao de um mata-borrão) que o pé realiza ao pisar no chão durante um passo de uma caminhada. Começa com o calcanhar tocando o chão, segue fixando toda a sola do pé, depois começa a levantar o calcanhar dobrando a base dos dedos e termina deixando o chão, tocando-o apenas com a ponta do pé.

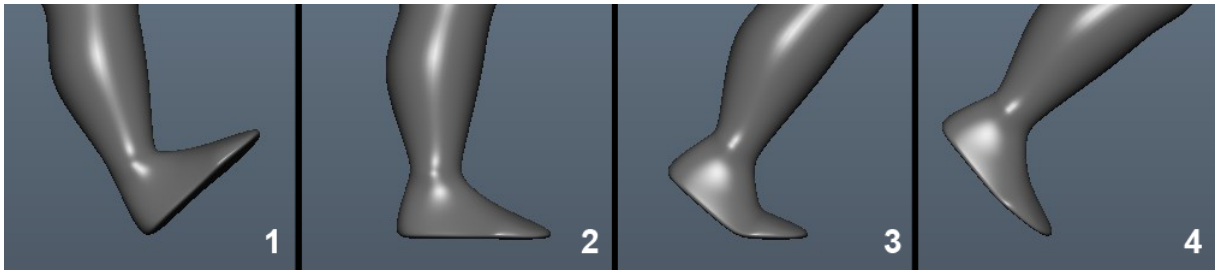

Figura 11: Etapas de uma pisada completa do pé durante uma caminhada. Fonte: o autor

Esse movimento de pisada é realizado quando se anima a perna no modo *IK*. Além disso, antes de se falar na utilização dos *drivens keys* será necessário realizar mais algumas configurações no *rig* do pé em IK. Primeiramente, deve-se criar *IK-handles* para as *joints* da base dos dedos e da ponta do pé. Sendo assim, haverá um *IK-Solver* submetendo a *joint* do tornozelo à *joint* da base dos dedos; e outro submetendo a *joint* da base dos dedos à *joint* da ponta do pé. Só assim é possível realizar o movimento que vemos no terceiro quadro da figura: onde o pé dobra na articulação da base dos dedos e com isso levanta o calcanhar (uma *joint* que está acima na hierarquia). Além disso, evita-se que a ponta do pé seja rotacionada junto com a base dos dedos, e mantém a região entre a base dos dedos e a ponta plantada no chão. As etapas dessa ação são compostas por 3 rotações seguidas, com pivôs em pontos diferentes: primeiro no calcanhar, em seguida na base dos dedos e por último na ponta dos dedos. As *joints* da base e ponta dos dedos já existem e já permitem a rotação nesses pontos. É preciso criar, então, o ponto de rotação a partir do calcanhar. Ademais, é importante buscar um sistema favorável à realização dessas rotações na ordem em que acontecem. Uma estratégia eficaz para isso é criar uma cadeia de *joints* que seguirá a ordem inversa das *joints* do pé, e começará a partir do calcanhar, essa técnica é apelidada *reverse foot rig* (*rig* inverso do pé). Assim, cria-se uma cadeia de *joints* começando por uma *joint* para o calcanhar, seguida da *joint* da ponta dos dedos e, por último, a *joint* da base dos dedos. É importante lembrar que as *joints* da ponta e da base dos dedos são simplesmente duplicatas das *joints* já presentes no pé, com sua posição e orientação idêntica às das originais, apenas com a ordem hierárquica inversa.

Em seguida essas *joints reversas* serão usadas para controlar os *IK-handles* das *joints* originais do pé. Isso é feito a partir do uso de *constraints* de transformação (que copiam posição e orientação). Sendo assim o *IK-handle* da ponta do pé será *restringido* pela *joint* reversa da ponta. Restam então o *IK-handle* criado na base dos dedos e também o *IK-handle* do tornozelo, criado anteriormente para o sistema de *IK* da perna. Esses dois *IK-handles* serão restringidos pela *joint* reversa da base dos dedos. Dessa forma, será possível levantar o calcanhar ao rotacionar essa *joint*. Tendo feito isso, estarão prontos os mecanismos necessários para facilitar a realização, numa ordem conveniente, dos movimentos que compõem o pisar.

Em seguida inicia-se a criação e configuração dos *driven keys*. Primeiramente usa-se o controle do pé para conter um novo *custom attribute*, nomeado *Foot Roll*. Esse atributo terá valores máximo e mínimo correspondentes à primeira e última pose correspondentes aos quadros 1 e 4 da figura 11. Ou seja, a pose com o pé levantado para trás, apoiado apenas no calcanhar, e a pose com o pé para frente, apoiado apenas na ponta. É desejável também poder retornar o pé à sua pose neutra sempre que necessário. Dessa forma, usa-se um valor negativo para o limite mínimo (pose do contato com o calcanhar); o valor 0 para a pose neutra (com o pé fixo ao chão); e um valor positivo para o limite máximo (pose de desprendimento do chão pela ponta do pé). Podemos usar, por exemplo, os valores -10 e 10, para os limites mínimo e máximo respectivamente. Esse atributo será usado para criar os *driven keys*.

No *Maya* isso é feito através de uma janela chamada *Set Driven Key* (do inglês: definir *driven key*), essa janela é divida em dois painéis: um chamado *Driver*, outro chamado *Driven*, e representam respectivamente o objeto que aciona a ação em questão e os objetos que são afetados por essa ação. No *Blender* o comportamento idêntico é obtido a partir dos chamados *Action Constraints*, em que as poses de uma animação pré-definida (ou *action*) são controladas por um objeto ou *joint*. No *3ds Max*, usa-se a interface denominada *Reaction Manager* (do inglês: Gerenciador de reações). Esse método, independente do *software*, permite a

simplificação no processo de automatização de movimentos mais complexos, evitando o uso de códigos de programação ou operações matemáticas.

Considerações finais

Como apresentado ao longo do texto, existe uma quantidade, talvez até intimidadora, de conhecimentos necessários a criação de bons *rigs*. Assim, buscou-se aqui oferecer ao leitor não familiarizado com a disciplina de *rigging*, uma introdução ao tema de uma maneira didática, abordando o máximo de conceitos pertinentes a técnica dentro das limitações de espaço inerentes ao formato do texto. Houve a preocupação com justificativas teóricas das razões de ser dos métodos e estratégias aqui propostos para fornecer ao leitor um bom senso crítico durante a aplicação desses ensinamentos em seu próprio trabalho. Nem sempre as soluções para problemas são muito óbvias e muitas vezes requerem um certo nível de experiência e olhar clínico para se identificar as formas de combinar técnicas a fim de se alcançar os efeitos e comportamentos desejados.

A evolução tecnológica na área de computação gráfica propicia o surgimento de novas ferramentas e novas formas de trabalhar a todo momento. Seja graças a novos *plugins* ou mesmo novos *softwares* e *hardware*. Tendo isso em vista, este texto buscou tratar dos conceitos e práticas que se fazem necessários ao entendimento do processo de *rigging* de maneira geral, sem discriminação de plataforma de trabalho. Também acreditamos que atualmente essa disciplina possui uma grande escassez de informação disponível em nossa língua materna. Em conclusão, esperamos ter servido para introduzir de maneira clara e abrangente a alunos e profissionais interessados em animação 3D digital os assuntos maiores que definem a técnica do *rigging* para personagens humanoides.

Referências Bibliográficas

RITCHIE, Kiaran; ALEXANDER, Oleg; BIRI, Karim. The Art of Rigging. California, EUA: CG Toolkit, 2006

BECK, Jerry. Animation Art: From Pencil to Pixel, the World of Cartoon, Anime, and CGI. Nova Iorque, EUA: Harper Design, 2004.

ALLEN, Erick; MURDOCK, Kelly. Body Language: Advanced 3D Character Rigging. Indianapolis, EUA: Wiley Pub., 2008.

CABRERA, Cheryl. An Essential Introduction to Maya Character Rigging. Boston, EUA: Focal Press, 2008.

O'HAILEY, Tina. Rig it Right! Maya Animation Rigging Concepts. Burlington, MA, EUA: Focal Press, 2013.

RODITI, Itzhak. Dicionário Houaiss de Física. Rio de Janeiro: Editora Objetiva, 2005.

Recursos on line

Blender User Manual. http://wiki.blender.org/index.php/Doc:2.6/Manual (acesso em: 11 Set. 2004)

Autodesk Maya 2014 User's Guide. http://download.autodesk.com/global/docs/maya2014/en_us/ (acesso em: 11 Set. 2004)

Autodesk 3D Studio Max 2015 Help. http://help.autodesk.com/view/3DSMAX/2015/ENU/ (acesso em: 11 Set. 2004)

Blender Quick Tip: How to set up IK pole vectors without breaking the bind pose. http://cgfromspace.blogspot.com.br/2013/11/blender-quick-tip-how-set-up-ik-pole.html (acesso em: 20 Set. 2004)