MAC – 499 Trabalho de Formatura Supervisionado
Trabalho tipo Projeto
Orientador: Flávio Soares

SIMULACRO

Monografia

● Fernando Bertolli Petroni ●

Outros integrantes: Gustavo Menezes Ponte Moreira
Rodrigo dos Santos Lima

Introdução

O objetivo do projeto Simulacro é simular a cinética do corpo humano em um computador, de maneira a facilitar simulações mais complexas que usem propriedades como força dos músculos ou restrições de juntas.

Utilizamos no projeto uma biblioteca de simulação física (ODE – Open Dynamics Engine) e um arcabouço gráfico (Ogre3D – Object Oriented graphics engine), ambos *opensource,* como bases do sistema. O arcabouço Ogre3D é todo escrito em C++ e orientado a objetos, a biblioteca física tem um núcleo escrito em C e é acessada através de uma interface OO escrita em C++. Aproveitando essas características, desenvolvemos um sistema orientado a objetos e dividido em camadas que encapsulam os detalhes do ODE e do Ogre3D e facilitam o reuso do código.

Como ponto de partida, foi implementado um sistema de simulação de colisões, conhecido no mundo dos *games* como *Ragdoll Physics*. *Ragdoll* nada mais é do que criar um modelo de corpo tridimensional e agregar a tal modelo sólidos que colidam com o ambiente que o envolve, sem se preocupar com restrições de movimento, apenas considera-se a conexão entre juntas do modelo.

Uma vez que a arquitetura de *Ragdoll Physics* estava pronta, foi feito um estudo baseado em livros de anatomia e de biocinese, sobre o comportamento do corpo humano sobre o efeito do movimento. Com tal estudo foi possível estabelecer restrições de movimentos impostas às juntas, tornando a simulação mais convincente, e trazendo o modelo para mais perto da realidade de um corpo humano.

Para tornar o projeto uma ferramenta aplicável, basta criar sobre o modelo um sistema que simule os músculos de um ser humano. Criar tal sistema não é tarefa das mais simples. Para fazê-lo foi preciso simplificar bastante o funcionamento de um músculo real, de maneira que se mantivessem suas propriedades cinéticas, parte importante para o projeto.

Toda essa base foi estruturada de maneira que seja possível posteriormente fazer com que o modelo ande. Mas andar é potencialmente a ação mais difícil de realizar que o ser humano conhece, afinal somos os únicos bípedes na natureza, mesmo depois de milhões de anos de evolução. A tarefa de andar envolve conceitos de equilíbrio e sinergia entre músculos. Com o projeto Simulacro em mãos, não é preciso se preocupar com fatos como interação do modelo com o ambiente, ação da gravidade sobre o corpo, entre outros, é preciso se focar apenas na coordenação dos músculos e na postura do modelo, ou seja, é dada uma ferramenta para facilitar a simulação de qualquer movimento que o ser humano possa realizar.

Com esse projeto damos um passo no sentido de possibilitar possíveis simulações médicas, educacionais, *games*, interação homem máquina, ou até mesmo situações em que a vida pode ser posta em risco.

Conceitos
e
Tecnologias

Ragdoll – O Ponto de Partida

1 – O Que é *Ragdoll*?

Nos jogos de *vídeo games* mais antigos, eram usadas animações pré-definidas de morte de personagens tridimensionais, isso pela vantagem do baixo uso de CPU, mesmo porque tempos atrás os computadores não permitiam uma quantidade grande de processamento.

Com o passar do tempo, e com a evolução dos computadores, se tornou possível fazer a simulação da física do ambiente em tempo real. Um modelo *Ragdoll* nada mais é do que uma coleção de múltiplos corpos rígidos, organizados por uma hierarquia de ossos a qual é agregada ao modelo tridimensional. Tal coleção de corpos rígidos é exposta à simulação da física em tempo real, adequando-os à forma do ambiente.

O termo *Ragdoll* está associado ao fato de que o sistema de articulações entre os corpos rígidos muitas vezes não possui restrições de movimento levando o modelo a ficar em posições improváveis e muitas vezes cômicas, como se o modelo fosse um boneco de trapos (*Ragdoll* em inglês).

Exemplo de Ragdoll Physics

2 – Vantagens/Desvantagens de usar *Ragdoll*

Devido ao preço computacional alto em executar simulações em tempo real, são feitas aproximações com o intuito de diminuir o uso de CPU, como por exemplo:

- não aplicar física em ossos das extremidades;
- não restringir o movimento de algumas juntas, fugindo da realidade de um corpo humano;
- usar sólidos simplificados no cálculo de colisões, ao invés de adequar perfeitamente o modelo tridimensional ao ambiente.

A grande vantagem que as animações *Ragdoll* oferecem sobre as outras, é a interação muito mais correta com o ambiente que envolve o modelo. Seria extremamente inviável criar animações pré-definidas para cada deformidade de terreno ou qualquer mudança no ambiente.

3 – A parte que interessa ao projeto

Apoiado na vantagem que *Ragdoll* oferece, de se adequar ao ambiente de maneira conveniente, foi desenvolvido um modelo de corpo humano ao qual foram associados sólidos que colidam com o ambiente em que este será colocado. Dessa forma garantimos que a interação do modelo com o ambiente será bem próxima da realidade.

Ogre 3D – Framework Gráfico Orientado a Objetos

1 – Porque usar o Ogre 3D?

O Ogre3D foi escolhido pelo mais simples fato de ser um programa *opensource*, e também propiciar uma série de ferramentas poderosas que facilitariam nosso trabalho na implementação do projeto além de dar suporte a vários recursos 3D interessantes como, por exemplo, sombras, CellShading e anti-aliasing, entre outros.

O único problema do Ogre3D é que ele só funciona em máquinas que possuam placa de vídeo com chipset igual ou superior a GeForce2 ou Radeon 7500.

A biblioteca apresenta uma arquitetura que a princípio é de difícil compreensão, porém quando se entende como ela opera sobre os elementos e age sobre as cenas, torna-se muito amigável ao programador.

Para desenvolver o projeto, foi escolhida a versão para Windows® da biblioteca. Isso devido a sua compatibilidade com OpenGL e principalmente com DirectX 9.0. Assim, toda a parte de manipulação de hardware e de configurações de tela fica por conta do Ogre3D, ajudando na implementação do projeto.

2 – Como o Ogre 3D funciona

O Ogre3D é um arcabouço orientado a objetos implementado em C++ para as bibliotecas DirectX e OpenGL. Foi construído com o objetivo de esconder as peculiaridades de cada hardware e sistema operacional de forma que seu código fonte é compilável para Linux, Windows e Mac OSX.

Os requisitos mínimos para construção de uma aplicação com Ogre3D são bons conhecimentos de C++ e orientação a objetos além dos conceitos básicos de computação gráfica para ambientes tridimensionais.

O loop principal de uma aplicação deve ser da seguinte forma:

1. Define-se a configuração que será utilizada
2. Prepara-se o cenário
3. Lê entrada do teclado, mouse ou joystick
4. Executa rotina pré-renderização
5. Renderiza um frame
6. Executa rotina pós-renderização
7. Volta ao passo 3 ou 2 dependendo da entrada do usuário

3 – O Ogre 3D no projeto

Além dos recursos gráficos, utilizamos algumas classes e operações do Ogre3D que representam conceitos da álgebra linear para ambientes tridimensionais tais como pontos, vetores e matrizes de transformações lineares. Abrimos mão de alguns poderosos recursos gráficos do arcabouço em favor da performance e porque sabíamos que a biblioteca de física deveria consumir boa parte do tempo de processamento da máquina utilizada para a simulação.

Aproveitamos algumas características da estrutura do Ogre3D na implementação do Simulacro. Nosso projeto funciona como uma extensão da camada de aplicação (ReferenceApplicationLayer) utilizada nos exemplos que acompanham o código fonte do Ogre3D. Essa camada oculta os detalhes da biblioteca ODE de simulação física utilizada tanto nas aplicações de exemplo do Ogre3D quanto no projeto Simulacro.

4 – Por que usar o ODE?

Desde o principio do projeto sabíamos que para a implementação das funcionalidades desejadas necessitaríamos de uma boa biblioteca de simulação física que implementasse idéias como atrito, torque, colisão e articulações entre sólidos. A principio imaginávamos que tais funcionalidades já existiriam dentro do próprio Ogre3D visto que simulações básicas de física são necessárias à grande maioria dos jogos 3D atuais. Ao analisarmos alguns programas de exemplo que acompanham o código fonte do Ogre3D percebemos que essa funcionalidade não existia dentro do Ogre mas, em alguns exemplos era usada a biblioteca ODE para tais simulações.

Depois de alguns testes, percebemos que o ODE implementava todos os recursos necessários ao nosso projeto e que os exemplos nos serviriam como ponto de partida para sua compreensão.

5 – Como o ODE funciona?

No Windows, o ODE funciona como uma biblioteca de vínculo dinâmico (DLL) que provê algumas classes estruturas e funções para manipulação de modelos físicos tridimensionais.

Numa aplicação típica que utiliza o ODE primeiro são criados os modelos dos sólidos e suas distribuições de massa e são inseridos num ambiente que é chamado de mundo.

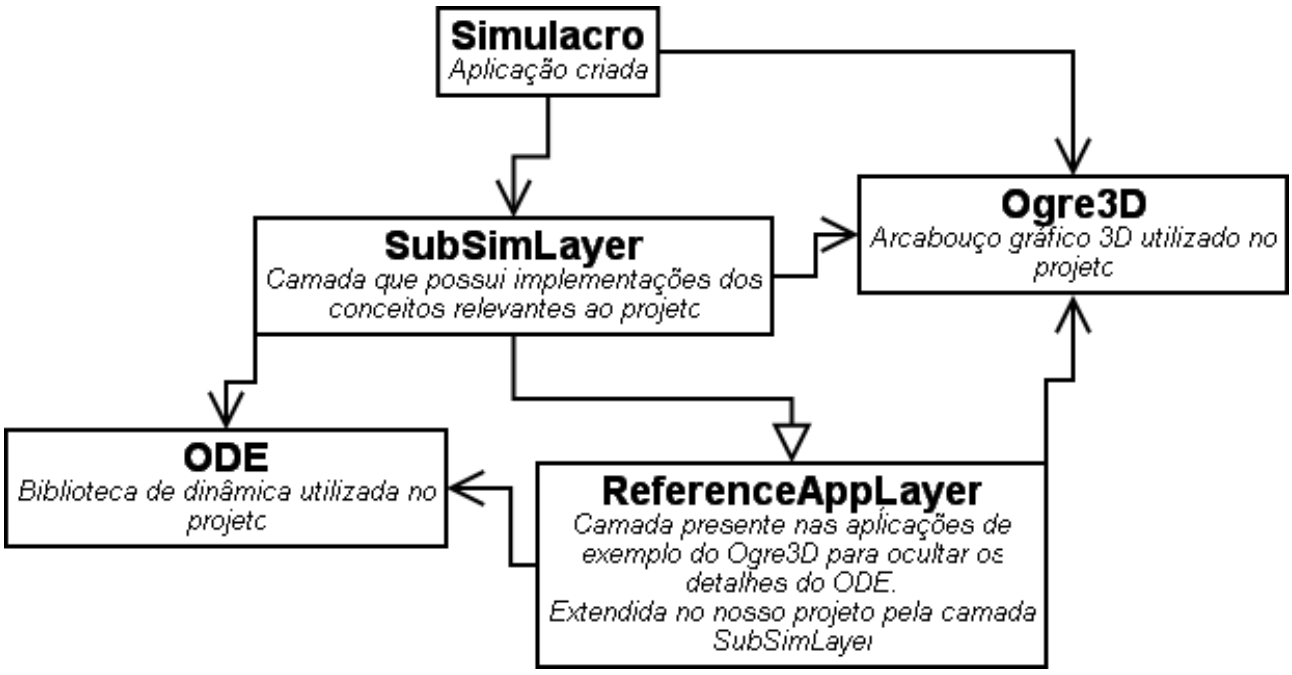
Após a inicialização, são definidas as forças que externas que atuam sobre os sólidos além de alguns parâmetros definidos a cada iteração. É chamada então a função que calculas as novas posições e velocidades dos sólidos do modelo atualiza o ambiente. Esse procedimento é chamado de “passo” da dinâmica e como parâmetro é definido e tempo de duração desse “passo”, o qual é chamado de “simulation step size”.

Internamente, a cada passo é montado e resolvido um grande sistema não-linear e, portanto, dependendo do número de variáveis (sólidos) e restrições (detalhes de geometria usados no cálculo de colisões e articulações), um passo da dinâmica pode levar um tempo consideravelmente grande para ser efetuado.

6 – O ODE no projeto

O ODE é responsável pelo cálculo de grande parte da física do nosso projeto. Não passamos todos os detalhes do modelo para o ODE porque sabemos que isso faria com que a função de atualização da dinâmica levasse um tempo indesejável para ser executada. Portanto, fazemos um pré-processamento das colisões entre os sólidos e dos sólidos com o ambiente, utilizando alguns recursos do Ogre3D como a representação do ambiente como um mapa BSP (Binary Space Partition) que agiliza o cálculo das colisões.

Para conseguirmos boas simulações no ODE, tivemos que nos preocupar com vários detalhes que poderiam levar a um estado inconsistente do ambiente e assim uma má qualidade da simulação e até um *crash* no ODE. Além disso, sabemos que boa parte do ODE é escrita em C puro, o que torna ainda mais difícil sua utilização. Percebendo essas dificuldades no manejo do ODE, resolvemos isolar seus detalhes apenas na camada SubSimLayer conforme mostrado no diagrama abaixo.


Simulacro
Aplicação criada
Ogre3D
Arcabouço gráfico 3D utilizado no projetc
SubSimLayer
Camada que possui implementações dos conceitos relevantes ao projetc
ODE
Biblioteca de dinâmica utilizada no projetc
ReferenceAppLayer
Camada presente nas aplicações de exemplo do Ogre3D para ocultar os detalhes do ODE.
Extendida no nosso projeto pela camada SubSimLayer

Modelagem do Corpo Humano

1 – Blender

Blender é uma ferramenta de modelagem 3D *opensource*.

Com tal ferramenta é possível criar um modelo tridimensional, e agregar ao modelo um conjunto de objetos ligados por meio de articulações, os quais chamamos de *Bones* (ossos em inglês). Tais *bones* são conectados por juntas, de maneira que ao se movimentar um *bone*, este fique com uma de suas extremidades presa ao seu antecessor.

O *blender* também é capaz de agregar aos *bones* um conjunto de vértices do modelo. Desse modo, ao se movimentar um *bone*, movimenta-se também o conjunto de vértices a ele agregado, passando a impressão de que o modelo se deforma junto com o conjunto de *bones*.

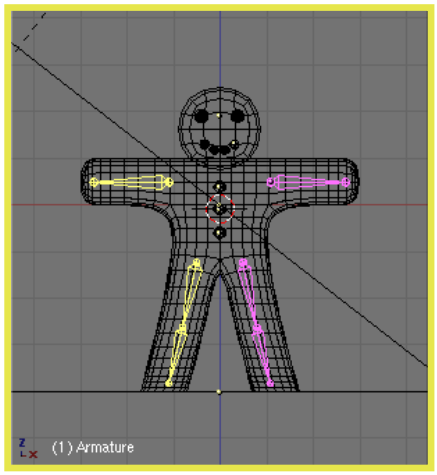
Depois de criado um modelo e seu conjunto de *bones*, podemos exportá-lo em um formato que o Ogre3D possa importá-lo e operar sobre o modelo. Para isso, usa-se um script de exportação para o *blender*, o qual pode ser encontrado no site oficial do Ogre3D. Dessa forma consegue-se estabelecer uma comunicação entre o *Blender* e o *Ogre3D*, facilitando o desenvolvimento do projeto.

O programa é *opensouce* de distribuição gratuita e possui uma versão compilada para Windows, e uma versão compilada para Linux.

2 – O Corpo humano na visão do projeto

O projeto Simulacro considera um corpo humano qualquer conjunto de *bones* que o Ogre3D possa operar. Com isso, abre-se uma infinidade de maneiras de se modelar o corpo humano, aumentando a flexibilidade do projeto e deixando a cargo do programador a escolha do nível de complexidade do modelo.

Com tal modelagem podemos simular não somente a cinética do corpo humano, mas também qualquer corpo móvel cuja réplica tridimensional possa ser reproduzida. Assim, é possível estudar como era a movimentação do ser humano ao longo de sua evolução, ou talvez estudar como uma pessoa ou animal com alguma deficiência, tanto muscular quanto articular, caminharia no plano ou até como faria para subir uma escada.

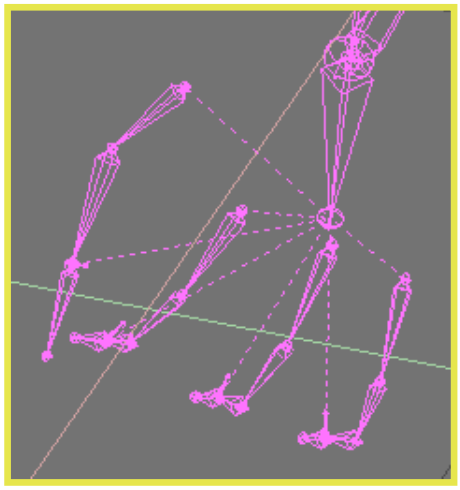


Exemplo de esqueleto não humano

O intuito principal do projeto é oferecer um arcabouço bom em termos de simulação, e que seja de fácil manipulação para facilitar a criação de novos aplicativos de simulação, pois toda simulação depende de um bom embasamento teórico. A princípio, a escolha do modelo para o esqueleto não é importante para o projeto. O que o projeto oferece é a possibilidade de se escolher qualquer modelagem para se executar a simulação, bastando criar um conjunto de *bones* de maneira que o Ogre3D consiga manipular. Mesmo assim, o projeto não garante que uma escolha qualquer obterá bons resultados.

3 – Articulações e Restrições

Uma articulação é um ponto de junção entre dois ossos, os quais não necessariamente estão ligados fisicamente, ou seja, uma articulação é um ponto de referência ao movimento de um osso em relação a outro.



Exemplo de articulações

No ser humano, as articulações só existem entre dois ou mais ossos conectados fisicamente, porém o *blender* possibilita definir tais junções sem conectar os *bones*. Como o objetivo é simular a cinética do corpo humano, o projeto irá tratar apenas articulações cujos ossos associados se ligam fisicamente. Assim, de agora em diante falaremos apenas de articulações que existem nos vertebrados.

Toda articulação possui uma capacidade de movimento bem definida. Por exemplo, o antebraço não ultrapassa os cento e oitenta graus em relação ao braço, já o braço pode ser rotacionado em várias posições em relação ao ombro.

Existem três tipos de articulações: a *sinartose* ou imóvel, a *anfiartose* ou ligeiramente móvel e a *diartose* ou amplamente móvel. Cada articulação específica do corpo possui suas próprias peculiaridades. No nosso caso de estudo iremos considerar principalmente as articulações amplamente móveis ou diartrósicas. Tais articulações possuem uma complexa rede de nervos e tendões para realizar de maneira correta e não destrutiva os movimentos, mas para o projeto iremos supor que todas as articulações são ideais, ou seja, o ossos sempre se manterão unidos e o movimento não desgastará as articulações.

Quanto ao movimento das articulações, sabemos que sua amplitude pode ser limitada por ligamentos, pelo comprimento e extensibilidade dos músculos, pelos tendões, pelo choque de um osso contra outro, dentre outros fatores. Sabendo disso, é preciso estabelecer um conjunto de restrições associados às juntas, para simular essa amplitude de movimentos.

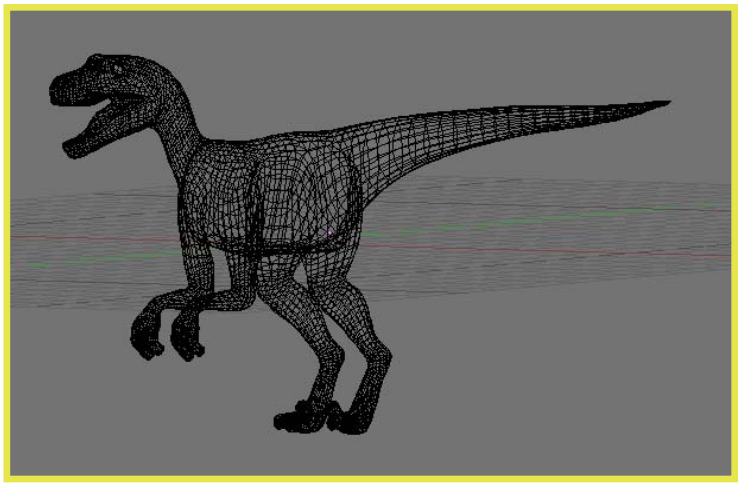
Na fase de criação do esqueleto, consideram-se apenas as restrições geradas basicamente pelo choque entre ossos conectados pela junta. Uma restrição de movimento causada por influência de músculo é tratada posteriormente. Para simular essa restrição, definem-se ângulos de amplitude máxima entre ossos, os quais são associados às articulações, e o programa se encarrega de restringir o movimento, baseado nessa amplitude.

4 – *Mesh*

O *Mesh* é um conjunto de vértices, arestas e faces, algumas vezes agregados a uma textura. Tal *mesh* possui uma organização tridimensional de suas elementos, possibilitando sua visualização espacial em um arcabouço 3D, em nosso caso o Ogre3D.

Os *bones* criados pelo *blender* ou qualquer outra plataforma de desenho 3D servem como sustento para um *mesh*, e portanto não são necessariamente visualizados. Por esse motivo é importante que se escolha um *mesh* apropriado para a simulação, pois este sim será mostrado no aplicativo e portanto deve mostrar de maneira concreta os resultados que estão sendo obtidos.

Novamente a regra de generalização se aplica, ou seja, não é necessário criar um *mesh* que siga perfeitamente as formas de um ser humano pois, uma vez que o esqueleto pode ser definido de qualquer maneira, o resultado obtido através da movimentação dos ossos pode ser mostrado da maneira que parecer mais conveniente. Cabe aqui ao programador ser sensato e seguir os padrões de acordo com o que está para ser simulado, ficando a cargo do mesmo o nível de complexidade do modelo.



Exemplo de *Mesh*

Modelagem Muscular

1 – O Que é mecânica muscular?

Mecânica muscular é o resultado físico da atuação dos músculos sobre o corpo.

O corpo humano, sendo considerado como uma unidade mecânica articular na qual duas ou mais partes estão unidas através de articulações e movidas sob ação da energia mecânica produzida pelos músculos atuando diretamente sobre os ossos que servem de alavanca, tem como objetivo final a produção de um movimento o qual é estudado sob dois aspectos:

a) fisiológico;

b) mecânico: visto no tecido muscular, inervado pelo sistema cérebro-espinhal que lhe proporciona uma contração rápida, voluntária e dependente da nossa vontade.

Em nosso caso, consideraremos apenas o aspecto mecânico do movimento, por questões práticas e pela obtenção de resultados mais concretos e visíveis. Abordaremos apenas o efeito final e visível, traduzido pelo movimento.

É importante considerar a mecânica de funcionamento dos músculos no momento em que será feita a modelagem computacional dos mesmos. Muitos fatores estão envolvidos, desde a contração muscular até o resultado final, no caso o movimento. Logo não se pode ignorar certos fatores biológicos para fazer tal modelagem. Veremos um pouco mais de fisiologia muscular.

2 – Fisiologia dos músculos

Os músculos agem coordenadamente, cooperando entre si, através da sinergia em que um procura auxiliar o outro, quer em ação colateral, secundária ou antagônica.

A maneira de um músculo agir sobre o corpo é através de sua contração. Tal contração pode ser estudada sobre dois aspectos:

a) estática: é aquela que desenvolve tensão insuficiente para mover um segmento do corpo frente a uma dada resistência.

b) dinâmica: é aquela que desenvolve tensão suficiente para mover um segmento do corpo frente a dada resistência. Tal contração pode ser dita *concêntrica*, onde se desenvolve tensão suficiente para vencer uma resistência, ou *excêntrica*, onde não se desenvolve uma tensão suficiente para vencer determinada resistência.

A fim de obter uma simulação correta dos músculos atuando sobre o corpo humano, é preciso considerar algumas funções musculares:

a) agonista: é a que efetua o movimento em contração concêntrica;

b) antagonista: é a que coordena a ação do agonista em contração excêntrica. Existe uma lei que diz: “quando o agonista se contrai, o antagonista se relaxa”. Serve para moderar, frear o movimento e proteger a articulação imobilizada;

Para o projeto estas são as duas funções mais importantes a serem simuladas. Por isso a modelagem computacional dos músculos será baseada principalmente nessas condições de movimento. Com tudo ainda existem duas outras funções:

c) fixadora ou estabilizadora: é aquela que age fixando um apoio, em contração estática;

d) neutralizadora: é a que impede a ação inútil de um dos segmentos imobilizados, com contração concêntrica.

Alguns acreditam que a contração de um músculo biarticular, ou seja, um músculo que está associado a mais de uma articulação, acarreta movimento nas duas articulações, o que seria uma economia e daria maior exatidão ao movimento.

Já outros acreditam que mesmo um músculo monoarticular, ou seja, um músculo associado a uma única articulação, tem ação sobre as duas articulações, logo deverá haver uma cooperação de todos os músculos.

Como pretendemos apenas criar um *framework* para que se possa programar simulações musculares mais facilmente, tal tipo de consideração é deixada totalmente a cargo do desenvolvedor da aplicação.

Consideremos agora as alavancas do corpo humano.

Bem sabemos da integração no nosso organismo entre osso, articulação e músculo, cada um atuando de uma maneira para se integrar no efeito final, o movimento. O corpo humano deve ser considerado como uma unidade mecânica em que os ossos atuarão como uma barra sólida, inflexível, na qual serão aplicados forças; as articulações servirão de fulcro, verdadeira sede do movimento e os músculos agirão como potência na produção do movimento e como resistência quando aliados ao peso dos segmentos ou de qualquer outro objeto externo.

Existem dois tipos de alavancas:

a) alavanca monoarticular: é aquela que possui apenas um osso como barra e um músculo atuando para provocar o movimento na articulação;

b) alavanca poliarticular: é aquela formada por um conjunto de ossos pequenos e movidos por sinergias musculares, atravessando diversas articulações.

Em nosso exemplo de aplicação criaremos um modelo que utiliza apenas alavancas monoarticulares, para facilitar a implementação. Porém o projeto possibilita a criação de alavancas poliarticulares.

3 – Modelagem dos músculos

Para modelar os músculos, respeitando sua fisiologia, nos utilizamos de torque. Aplicamos uma força de rotação em um determinado osso, em uma dada direção.

O controle do torque, ou neste caso o controle do músculo, é baseado na força que será aplicada e no tempo que tal força será aplicada sobre os ossos.

O mais interessante dessa modelagem, é o efeito colateral que é proporcionado no resto do esqueleto. Com aplicação de torque em um determinado osso, o movimento que será gerado provocará uma reação nos ossos adjacentes, e fará com que o esqueleto inteiro sinta a ação do músculo. Isso torna o movimento mais real, pois irá afetar um possível equilíbrio que esta sendo mantido no modelo, e devido às restrições, nenhum movimento absurdo será feito.

Vale ressaltar que o principal motivo da escolha dessa modelagem, é o fato de o modelo continuar respeitando os limites e leis da física. Por mais forte que seja o torque aplicado em um determinado osso, muitas vezes não será possível transpor algumas barreiras do ambiente ou até mesmo do esqueleto. Tudo isso aliado ao fato de não importar a posição do modelo no momento de se realizar o movimento, o que é um problema em animações pré-definidas.

Atividades Realizadas

A Criação do Modelo de Corpo Humano

1 – Modelagem da anatomia

Para mostrar o projeto em funcionamento, precisamos desenvolver um modelo razoavelmente fiel à realidade, de maneira que sejam mostrados todos os conceitos que foram implementados.

Optamos por criar um modelo de corpo humano relativamente simples, onde fique claro as colisões com o ambiente e o funcionamento das restrições sobre o esqueleto.

Criar tal modelo não é tarefa das mais simples. Foi preciso pesquisar em livros de anatomia e biologia o funcionamento do corpo humano. Existe muita informação na parte biológica do processo, porém na parte cinética, nossa parte de interesse, não existe tal diversidade.

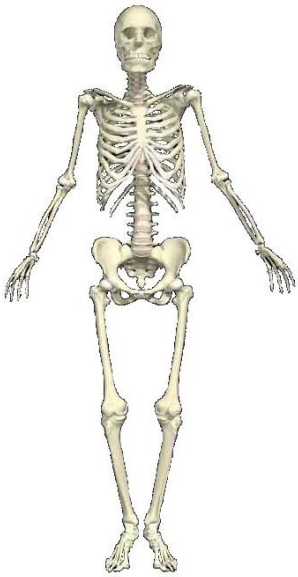
Conseguimos criar uma boa modelagem da anatomia humana baseando-nos em livros de ergonometria. Tiramos nossas próprias medidas de restrições de juntas e proporcionalidades corporais.

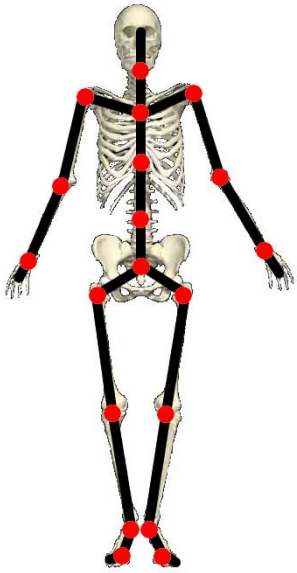
Agora estamos em condições de começar a tornar todos os dados parte do simulacro.

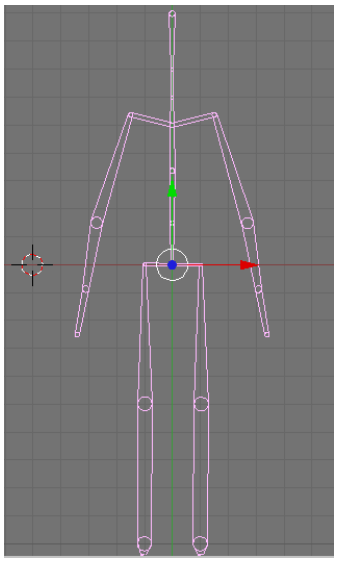
2 – Criação dos ossos e do *Mesh*

O primeiro passo a dar em direção do simulacro é criar o nosso esqueleto de sustentação do modelo.

Para a simulação que está sendo produzida junto com a implementação do projeto, foi utilizado um conjunto de *bones* consideravelmente complexo. O esqueleto (conjunto de *bones*) em questão possui vinte e cinco *bones*, onde um desses *bones* é usado para sustentação de todo esqueleto, ou seja, movendo-o é possível mover todos os outros *bones*. Um modelo comum de animação usado em *vídeo games* possui cerca de dez partes articuladas, muito longe dos mais de duzentos ossos que o esqueleto de um ser humano possui.







Foi escolhida tal modelagem do esqueleto humano baseada em livros de Cinesiologia, e artigos médicos. Com este esqueleto conseguimos simular bem a maioria dos movimentos que um ser humano é capaz de realizar. As partes não modeladas no esqueleto são: os dedos, tanto dos pés quanto das mãos; as costelas, pois estas não possuem relevância na cinética do corpo; e ossos pequenos, como por exemplo ossos da face, ou vértebras da coluna.

Com essa modelagem, conseguimos realizar praticamente todos os movimentos possíveis de um ser humano. Nela ignoramos ossos como costelas e bacia, pois em termos de movimentação entre juntas, estes são irrelevantes. A ausência de tais ossos poderia afetar uma modelagem mais detalhada em termos de peso e colisão com o ambiente, porém tais fatores podem ser corrigidos posteriormente utilizando outros recursos do programa.

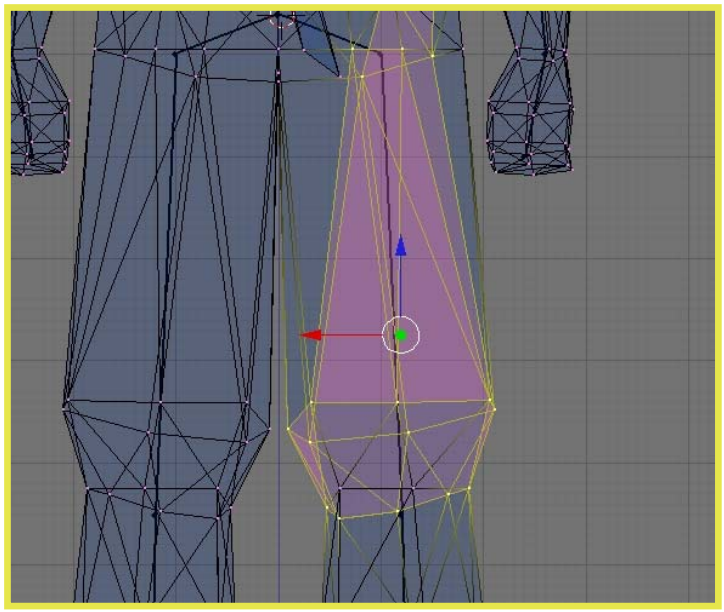
O *Mesh* utilizado em nossa simulação foi o modelo de um ninja que vem incluso no *framework* do Ogre3D. Se trata de um modelo simples, porém muito fiel a algumas características humanas.

Tal *Mesh* foi editado de maneira que se ressaltassem apenas características de um corpo humano comum. Por exemplo, no modelo original o ninja segura a espada em uma das mãos, o que visualmente seria ruim. As texturas foram mantidas.

3 – Agregando movimento ao *Mesh*

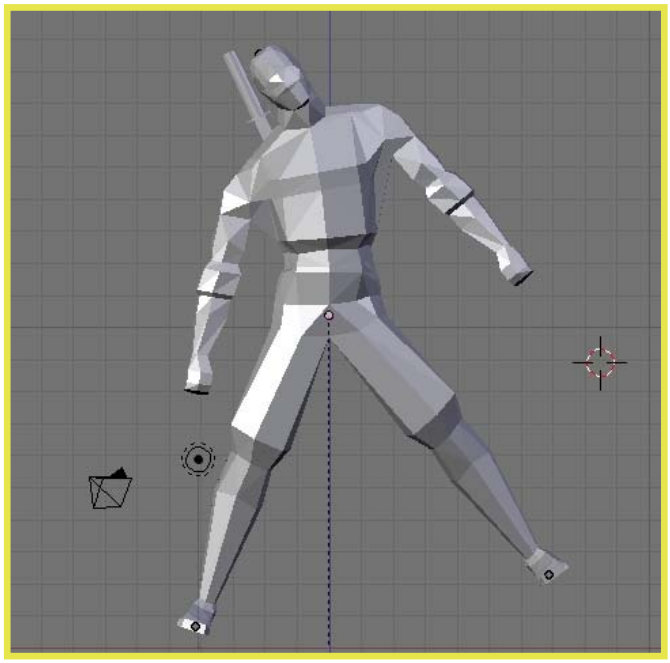
Atualmente possuímos um esqueleto e um *Mesh*, porém ambos são objetos distintos, e na prática um não tem nenhuma relação com o outro, mesmo que tenham sido construídos um pensando no outro.

Agregar o esqueleto ao *Mesh* é simples porém muito trabalhoso e é preciso tomar cuidado com os vértices que serão agregados a cada osso, pois dependendo de como isso é feito, ao se movimentar os ossos podem acabar movendo partes do *Mesh* que não fazem sentido visualmente. Também é complicado reproduzir perfeitamente o movimento das articulações.



Exemplo do fêmur esquerdo do nosso modelo

Nesse momento fica claro porque a escolha do *Mesh* e de um esqueleto que se adeque melhor são importantes. Fisicamente a movimentação não é importante, pois todo seu movimento é baseado no esqueleto, mas visualmente pode se obter resultados bem ruins.



Nosso modelo em movimento

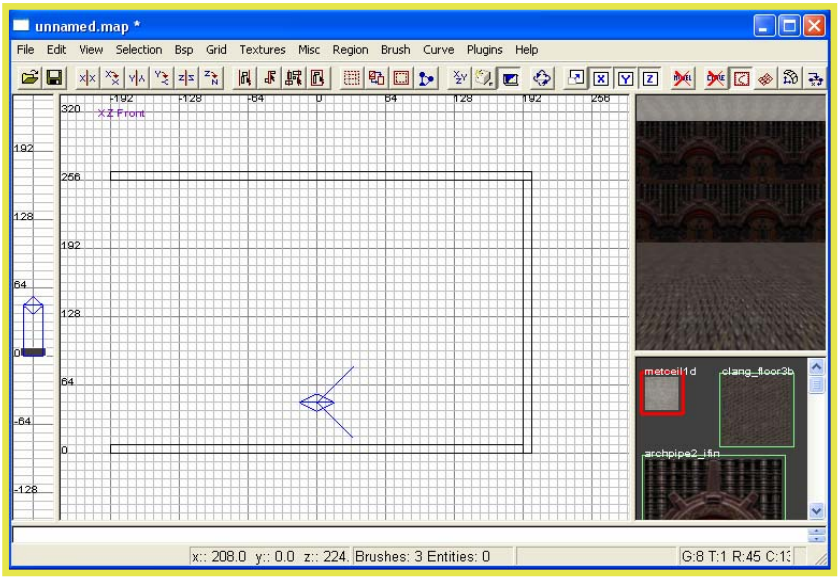
Tomados os devidos cuidados, chegamos em um estado aceitável visualmente.

Criando o Ambiente

1 – Criação do mapa

O mapa é, na verdade, o conjunto de sólidos e superfícies com os quais o modelo poderá interagir, um ambiente onde ele estará inserido. É possível criar tal mapa usando o Ogre3D, mas para isso é necessário declarar cada sólido e superfície, suas posições e dimensões manualmente, o que, para mapas um pouco mais complexos, é uma tarefa monótona. Por isso, o Ogre3D é capaz de importar a geometria de certos tipos de mapas.

Um desses tipos de mapas é o mesmo usado no jogo Quake3. Assim, o Ogre3D é capaz de receber um mapa compilado (BSP) e seu conjunto de texturas associadas, e montar um ambiente semelhante internamente. Isso torna as coisas mais fáceis, pois há vários editores de mapas de Quake3 disponíveis para uso. Portanto, a tarefa resume-se a criar um mapa em um desses editores, colocar as texturas e iluminação, compilá-lo (gerando o BSP) e usar o recurso de importação do Ogre3D para usá-lo no aplicativo.



Criando o mapa no *Q3Radiant*

2 – Inserindo o mapa no Ogre3D

Após a criação do mapa, deve-se fazer duas coisas.

A primeira é indicar para o Ogre3D onde se encontra o recurso do mapa. Tipicamente aplicativos possuem pastas onde se encontram todos os recursos por ele utilizados, incluindo texturas, meshes, scripts e, no caso, os mapas. O Ogre3D possui um arquivo que indica onde se deve procurar cada um desses tipos de recursos. O que se deve fazer é colocar o mapa e suas texturas em um pacote (tal como .ZIP) e alterar o arquivo de recursos do Ogre3D indicando onde ele pode encontrar esse novo mapa.

A segunda é indicar para o Ogre3D que se deseja usar a geometria desse mapa. Isso pode ser feito com funções que ele fornece. Após esse passo o mapa está pronto para uso.

Mapa pronto no Ogre

Do *Ragdoll* ao Simulacro de Ser Humano

1 – Criando o *ragdoll*

Após terminarmos de construir o mesh e seu esqueleto, é necessário transformar os seus dados de modo que o Ogre3D seja capaz de lê-los e utilizá-los no aplicativo. Para isso, os criadores do Ogre3D inventaram um pequeno *script* que converte um arquivo .BLEND (do *Blender*) para um arquivo XML, que contém todas as informações relevantes, como posicionamento dos *bones*, ângulos entre eles, as malhas de triângulos e os conjuntos de triângulos que cada bone movimenta.

Eles também criaram um aplicativo que tem como entrada esse arquivo XML e origina um arquivo binário, no formato que o Ogre3D reconhece. Então, o programa torna-se capaz de reconstruir o mesh do jeito que estava no *Blender*.

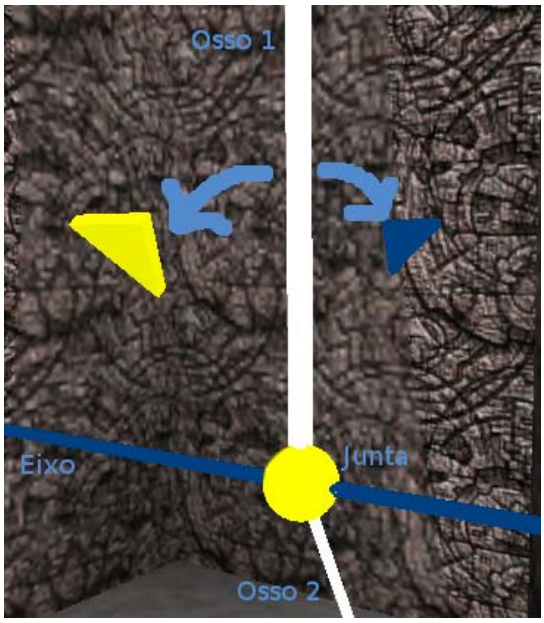
Apesar do XML conter todas essas informações, ainda falta um tipo de dado que não é crucial para o Ogre3D, mas que é importante para nós: o comprimento de cada osso. Montamos os pedaços do modelo baseando-nos nesses comprimentos; ainda que seja possível obter esses dados a partir das posições de um osso e do seu filho, ocorre um problema nos ossos das extremidades, que não têm filhos. Sem essa informação, os pedaços da extremidade teriam tamanhos incorretos.

Pensando nesse problema e tendo em mãos os códigos-fonte do script de exportação do *Blender* e do aplicativo conversor de XML, decidimos fazer uma pequena alteração nos dois de modo que a informação que necessitávamos fosse disponibilizada. Mudamos o *script* de modo que, no arquivo XML, fossem incluídos os comprimentos de cada osso; também fizemos alterações no conversor de XML. Esse caso foi mais complicado, porque não podíamos alterar a maneira que o Ogre3D gera / lê os binários; uma solução, não exatamente a melhor, foi anexar as informações necessárias em um arquivo de texto, que é aberto e analisado em tempo de execução pelo nosso programa.

Portanto, geramos os pedaços com tamanho e massa proporcionais a esses comprimentos, com o formato padrão de cápsula. As juntas são montadas em suas posições, mas qualquer informação de restrição do *Blender* é ignorada e elas são inicializadas com o máximo de restrição possível.

2 – Restrição de movimento

Como as juntas são construídas com o máximo de restrição possível, para que haja movimento é necessário que haja uma forma de definir esses limites depois que as juntas já estão criadas. Isso é feito através do método setLimits() da classe SimJoint que implementa o conceito de juntas. Cada limite é formado por um par de ângulos, máximo e mínimo, definido com relação a um dos possíveis eixos de movimento da junta. Conceitualmente, sabemos que as juntas humanas podem ter entre 1 e 3 eixos de liberdade. No nosso projeto implementamos apenas 2 eixos devido a algumas limitações impostas pela biblioteca de física. Apesar dessa limitação, percebemos que para a grande maioria dos casos no nosso modelo, 2 eixos de movimento eram o suficiente para uma boa simulação. Como exceção temos o caso do pescoço, que pode mover a cabeça para cima e para baixo, girar e pender lateralmente, ou seja, possui 3 eixos de movimento diferentes.



Limites de movimentação de uma junta

3 – Interação com o ambiente

A interação do modelo com o ambiente é feita através dos sólidos de colisão que foram pré-definidos. Quem cuida da identificação de colisões entre sólidos e de sólidos com o ambiente é o *ODE* (Open Dynamics Engine), uma biblioteca de simulação física.

O *ODE* fica encarregado de simular a gravidade e seu efeito sobre os objetos, reações a forças como por exemplo força elástica, e também se encarrega de gerar forças para interação com os objetos, por exemplo o torque que deve ser produzido para realizar movimento, de acordo com nossa modelagem de músculos.

Para produzir um efeito visual mais interessante, o *Mesh* do modelo em si não possui tais sólidos de colisão, apenas o esqueleto carrega os sólidos. Movimentar o esqueleto implica em movimentar o *Mesh*, isso pois cada osso do esqueleto está agregado internamente a um conjunto de vértices do *Mesh*, e nosso programa movimenta tais vértices de acordo com o movimento do esqueleto.

Um sólido de colisão possui propriedades como densidade, massa, elasticidade dentre outras. Cada sólido pode possuir uma forma específica, cada um com suas próprias dimensões, desde que sejam sólidos que o *ODE* reconheça. Não importam o tamanho o peso ou qualquer outro atributo, a única força que age a princípio é a gravidade. Todas as outras forças internas vem em decorrência da gravidade. Ao aplicar torque sobre os ossos, ou seja, simular os músculos, estamos adicionando uma força externa que começará a fazer parte do sistema e só então influenciar na movimentação dos sólidos. Não existe resistência do ar ou atrito, porém graças ao *ODE*, são ambas forças possíveis de serem simuladas.

O interessante em se usar sólidos de colisão pré-definidos é a economia de processamento, pois são sólidos com equações bem definidas facilitando os cálculos de colisão, e também porque facilitam uma adequação ao *Mesh* do modelo que será posto a prova.

O *ODE* é bem fiel à realidade, levando os sólidos e portanto carregando consigo o modelo, em adaptações ao ambiente praticamente perfeitas, fato este que facilita uma aproximação à realidade em termos de interação com o ambiente.

4 – Controle dos músculos

No nosso projeto, os músculos foram modelados através da classe SimMuscle. Cada músculo está ligado a um par de ossos e um eixo de movimento que será usado para aplicação de torque nos ossos. Para aplicar uma força, devemos chamar o método applyForce() da classe SimMuscle passando como parâmetros a quantidade de força que deverá ser aplicada e o tempo de duração da aplicação dessa força. Internamente, é aplicado um torque a um dos ossos e um torque inverso ao outro, de forma a termos um par ação-reação.

Resultados e Produtos Obtidos

Um Princípio no Estudo da Cinética do Corpo Humano

Após implementarmos o projeto, e criarmos o nosso próprio simulacro, pudemos observar o quão difícil é manter um corpo em equilíbrio e mais difícil ainda é fazer com que tal corpo ande.

Chegamos em um estado de simulação física bem condizente com a realidade, onde percebemos não ser necessário modelar um sistema de colisões perfeitamente adequado ao *Mesh* do modelo. Com sólidos simples foi possível criar um modelo que visualmente colide perfeitamente com o ambiente o qual este está inserido.

Ainda com relação aos sólidos, percebemos que quanto mais simples estes fossem, mais flexibilidade o nosso programa teria. Por isso fica mais fácil e mais rápido ajustar os sólidos ao *Mesh*, independentemente da complexidade do modelo. Para atingir um nível de detalhamento maior por parte das colisões é necessário dividir um dado osso em vários outros, onde a restrição de movimentos entre eles é máxima, e para cada osso definir um sólido de colisão próprio. Computacionalmente esta é uma alternativa muito pesada, porém bastante genérica.

Percebemos que o número de sólidos no simulacro, e o formato de tais sólidos não influem muito na velocidade de execução do programa. Também notamos que as colisões com o ambiente são praticamente perfeitas, e variam apenas se forem alterados valores como por exemplo, o *stepsize* ou a taxa de erro de arredondamento dos números (valores ajustáveis no programa).

Esses fatores justificam o que ocorre com os modelos de *Ragdoll* que existem hoje em dia. Nos *vídeo games* não existe uma capacidade de processamento muito grande, por isso são usados modelos relativamente simples em termos de movimentação, porém que colidem de maneira quase perfeita com o ambiente.

Quanto à movimentação do esqueleto percebemos que este é um ponto importante para se considerar no momento de se criar o simulacro. Deve-se levar em conta o tipo de movimentação que o simulacro vai ser capaz de realizar. Por exemplo, quando está se modelando as juntas do joelho, não queremos que a canela dobre para frente em relação ao fêmur.

Tal controle de movimento pode ser feito de duas formas: apenas especificando o tipo de junta entre os ossos, ou além disso especificar também os ângulos de movimentação máxima e mínima. A primeira maneira é a mais utilizada no mundo dos *vídeo games*, porém é muito pouco fiel a realidade. A segunda maneira seria o mais correto, mas é computacionalmente muito pesada. Com os teste que o grupo realizou, notamos que ao adicionar as juntas com restrição de movimento, obtivemos uma perda de cerca de cem vezes de performance, ou seja, torna-se inviável usar tais restrições se o intuito da aplicação não é simular perfeitamente a física da realidade. O projeto é capaz de lidar com os dois tipos de juntas, basta ligar ou desligar o *Ragdoll mode*, o que será explicado melhor posteriormente.

Conseguimos avançar mais um passo em direção a modelagem ideal do ser humano. Obtivemos um modelo com um esqueleto bem definido, com sólidos que acompanham bem o formato de seu *Mesh*, e também com os movimentos dentro das restrições da realidade.

A partir desse momento nos dedicamos à implementação dos músculos que atuariam no modelo.

Com nossa modelagem de músculos, percebemos um fato interessante que não existe nas animações atualmente. Quando movimentamos um ou mais ossos, a simulação física se encarrega automaticamente de passar para os demais ossos a reação que tal movimento proporcionou. Por exemplo, ao fazermos com que o modelo levantasse um dos braços, ficou visível o quanto o movimento altera o estado do esqueleto em relação aos ombros, a coluna e assim por diante. Mesmo sendo um movimento muito simples, visualmente produz uma animação muito mais real do que simplesmente uma rotação entre os ossos.

Conseguimos assim uma base bastante fiel em relação às leis da física, para um possível estudo da cinética do corpo humano. Fazer com que o modelo ande é talvez a tarefa mais difícil, afinal esta é provavelmente uma das tarefas mais difíceis para nós seres humanos.

O Projeto Simulacro:
Passo a Passo

Preparativos: (Os passos de 1 a 5 podem ser ignorados se não se deseja alterar o mesh ou o mapa.)

1- Criar o mesh no Blender, e utilizar nosso script de exportação para gerar os arquivos necessários ao Ogre3D (XMLs do mesh e do esqueleto).

2- Utilizar o OgreXMLConverter por nós modificado em ambos XMLs para gerar os arquivos binários. Ao gerar o binário do esqueleto, o programa deverá gerar também um arquivo chamado “skeletonLengthInfo.txt”. Copie esse arquivo para a pasta do executável Simulacro.exe. (/bin)

3- Colocar os binários do mesh e do esqueleto na pasta bin/media/models, SOBRESCREVENDO os arquivos do ninja (ninja.mesh e ninja.skeleton). Lembre-se de fazer uma cópia dos originais antes.

4- Criar um mapa de Quake3 em um editor de mapas (por exemplo, Q3Radiant) e gerar o mapa compilado (BSP).

5- Use um aplicativo de compressão para colocar o BSP e as texturas em um pacote “teste.zip”, e coloque esse pacote em bin/media/packs, SOBRESCREVENDO o existente (copie o original antes).

6- Execute o programa Simulacro.exe.

Explicação da interface e dos comandos:

- Há três tipos de teclas de comando em nosso programa: navegação, edição de parâmetros e controle de estado.

- Teclas de navegação: controle da câmera (WASD). As setas esquerda / direita alternam entre as partes do modelo, e as setas cima / baixo alternam entre os modos de edição de matéria, juntas e músculos, quando em modo de edição. A tecla TAB alterna entre os subparâmetros de um parâmetro.

- Os parâmetros são editados de duas formas. Os que não envolvem quantidades numéricas são diretamente alterados ao pressionar sua tecla correspondente, como a visibilidade. Pressionar a tecla irá alternar entre todos os possíveis valores do parâmetro. Aqueles que envolvem números funcionam de um jeito diferente: pressionar sua tecla selecionará o atributo, e as teclas - / + do teclado numérico decrementam / incrementam o valor correspondente.

- Teclas de controle de estado: A tecla ENTER alterna entre o modo de simulação e o modo de edição. A tecla SHIFT direita retorna o modelo à sua posição inicial. As teclas F2 e F4 salvam e restauram um estado do simulacro, respectivamente.

- O mouse controla a orientação da câmera, e pode-se lançar uma cabeça de Ogre com o botão direito do mouse.

Considerações quanto ao estado inicial: Os tamanhos e massas de cada parte do corpo serão pré-definidos em função dos tamanhos dos ossos, podendo posteriormente ser editados. Todas as juntas começam altamente restritivas, também podendo ser alteradas.

Conclusões

Apesar de não termos implementado todas as funcionalidades as quais gostaríamos que o projeto possuísse, o resultado final foi satisfatório porque serve como uma boa base para implementação de projetos futuros. Ficamos muito contentes especialmente com o resultado gráfico obtido e todo o conhecimento que obtivemos no processo.

Uma das preocupações desde o início do projeto, foi com que produzíssemos uma aplicação que fosse de fácil manuseio e extensão para estudos posteriores. Acreditamos que conseguimos isto através de uma boa estrutura orientada a objetos que isola detalhes de implementação das bibliotecas utilizadas.

Percebemos também que a simulação física no computador, com uma boa precisão, é algo extremamente custoso em termos de processamento. Mas apesar disso sabemos que em um futuro próximo será possível tornar cada vez mais complexas e precisas tais simulações físicas, aumentando a viabilidade do uso de programas como o nosso para aplicações reais.

Após termos o projeto em mãos, vimos o quanto é difícil fazer com que o modelo ande somente pela ação de seus próprios músculos. Pudemos ver o quão complexo é o sistema de coordenação motora do ser humano e o quão poderoso nosso cérebro é.

Bibliografia

RASCH, PHILIP J; BURKE, ROGER K, colab (1977) *Cinesiologia e anatomia aplicada : a ciência do movimento humano,* Rio de Janeiro: Guanabara-koogan.

RASCH, PHILIP J. (1989) *Kinesiology and Applied Anatomy*, Philadelphia: Lea & Febiger.

FRACCAROLLI, JOSE LUIZ (1981) *Biomecânica: análise dos movimentos*. 2. ed. Rio de Janeiro: Cultura Médica.

FELIX E. ZAJAC, RICHARD R. NEPTUNE, STEVEN A. KAUTZ (2002) *Biomechanics and muscle coordination of human walking*. www.elsevier.com/locate/gaitpost.

Parte Subjetiva

Foi uma boa experiência o desenvolvimento deste projeto de formatura. Além de exigir conhecimentos em linguagens de programação e UML, o projeto exige vários outros tipos de conhecimento, que involveram desde a criação do modelo no *Blender* até o desenho do mapa usado no aplicativo, ou seja, coisas que não precisam ser feitas por um computólogo em particular, mas que ainda são essenciais. Isso demonstra o caráter interdisciplinar de alguns projetos; poderíamos ter sido auxiliados por um fisiólogo, no que diz respeito à complexidade do modelo humano e as proporções corporais.

Agora discorrerei sobre alguns itens relacionados à minha experiência.

desafios e frustrações encontrados:

Um dos desafios foi acostumar-me com a ferramenta de desenvolvimento, no caso, o programa *Visual Studio 2005*. Durante a maioria do curso fizemos apenas projetos pequenos, que não precisavam de bibliotecas externas, cujos aplicativos não usavam DLLs externas e os programas podiam ser feitos em um editor de texto qualquer. Somente em Engenharia de Software que tivemos a experiência de mexer com um projeto razoavelmente grande, e em uma ferramenta de desenvolvimento, no caso o Eclipse. Tais ferramentas automatizam uma série de processos que eram feitos manualmente por nós, o que nos confundiu um pouco... no fim, foi necessário um tempo razoável até que conseguíssemos configurar o nosso projeto corretamente. O *Visual Studio* também tem o hábito de, por padrão, gerar arquivos gigantescos de depuramento, recurso que não utilizávamos. Era comum uma pasta que somente continha código-fonte, após compilação e geração do executável, ficar com mais de 100 megabytes de dados...

Outro desafio foi conseguir balancear performance e realismo no projeto. Como já mencionado, simular física e restrições em todas as partes de um corpo é extremamente custoso em termos de processamento, portanto é necessário reduzir a velocidade de simulação para que máquinas menos potentes possam rodar o projeto. Mas uma velocidade muito lenta é prejudicial na hora de fazer os testes, quando é necessária agilidade. Encontrar esse equilíbrio exigiu muitas tentativas.

O maior problema foi a falta de sincronia entre os horários dos membros do grupo. Cada membro estava cursando disciplinas diferentes, e não houve muitos momentos que podíamos nos reunir para discutir sobre o projeto. Isso também causou atraso na implementação, e acabamos fazendo bem menos do que o planejado.

lista das disciplinas cursadas no BCC mais relevantes para o trabalho:

- Introdução à Computação

  Ensinou os conceitos básicos, sem os quais é impossível programar.

- Engenharia de Software

  Nos deu a primeira experiência de um projeto grande e de todos os passos necessários para sua realização, do planejamento à implementação.

- Computação Gráfica

  Apesar de a biblioteca gráfica já ter sido implementada, foram necessários conceitos dessa disciplina para entender como usá-la.

- Programação Orientada a Objetos

  Especialmente útil para compreender o funcionamento do Ogre3D e projetar a modularização do programa.

- Álgebra Linear

  Usamos essa disciplina para efetuar operações no espaço tridimensional.

- Física I

  Aplicamos física mecânica o tempo todo em nosso projeto.

interação com membros da equipe que tenham agido como mentores do trabalho:

Entre nós não houve exatamente mentores; cada um tinha habilidades e disponibilidades específicas para cada tarefa, e distribuímos os afazeres de modo a explorar isso o melhor possível. O fato de nos conhecermos há um bom tempo ajudou no relacionamento e nas discussões, e saber as condições às quais cada membro estava sujeito foi importante.

diferenças notadas entre a forma de cooperação com colegas do BCC nas tarefas em grupo das disciplinas e a forma de trabalho conjunto na empresa:

Eu nunca trabalhei em nenhuma empresa, mas de acordo com alguns amigos que já tiveram essa experiência, o que pode-se observar é um ambiente mais caótico, devido aos curtos prazos de entrega. As equipes na empresa não tem muitas oportunidades para discutir e explicar uns aos outros coisas sobre o trabalho, nem de escrever documentação suficiente para que os outros entendam, pois precisam implementar as coisas o mais rápido possível.
Outro fator importante é a responsabilidade em cada um dos ambientes; no BCC, somos todos alunos e estamos aprendendo, o que nos permite errar muito mais do que na empresa, onde cada erro prejudicará a empresa e pesará contra o profissional.

observações sobre a aplicação de conceitos estudados nos cursos no contexto prático de aplicações reais:

Algumas vezes durante o decorrer do curso é normal perguntarmo-nos sobre a utilidade de algumas disciplinas, principalmente as mais teóricas. Então, ao desenvolver um sistema grande e fazer uma sub-parte desse sistema, deparamo-nos com um problema que nos parece familiar... desde ordenação até busca de menor caminho.

É assim que usamos os conceitos estudados no curso. Eventualmente , não são problemas explícitos; eles não dão dica alguma sobre como resolvê-los, diferentemente dos problemas resolvidos no curso, onde, por exemplo, na disciplina de grafos, sabemos que devemos usar grafos... Ou seja, há mais etapas envolvidas no processo. É preciso encontrar o problema, identificá-lo, descobrir o que pode ser usado para resolvê-lo e, enfim, implementar uma solução.

É nesse momento que se percebe a importância de uma formação sólida em ciências da computação. Sem ela, ou desanimaríamos perante dificuldades encontradas ou deixaríamos alguém mais capacitado enfrentá-las, o que nem sempre é possível...

se o aluno fosse continuar atuando na área em que exerceu o estágio, que passos tomaria para aprimorar os conhecimentos técnicos/metodológicos/comerciais/científicos/etc relevantes para esta atividade?

Se continuasse a trabalhar no projeto, eu provavelmente procuraria pesquisar mais a fundo o modelo humano e modificar a nossa abordagem para deixá-la ainda mais realista. Também procuraria por uma biblioteca física que se adequasse melhor ao nosso contexto, reduzindo a sobrecarga de operações no processador e possibilitando melhora na complexidade do modelo.

Também seguiria em frente com o plano original do projeto, implementando uma “linguagem de músculos” que possibilitaria passar comandos a cada um dos músculos. A coordenação de vários desses comandos poderia ser vista como uma ação complexa por nós realizada, tal como caminhar, correr ou sentar. Ou poderia-se inserir um sistema de inteligência artificial cujo objetivo é manter o equilíbrio do modelo, nas mais variadas situações. Assim, o sistema poderia evoluir para ter aplicação real.