6
Desenvolvimento do Simulador Virtual

Neste capítulo, descreve-se o desenvolvimento do Simulador Virtual para demostrar a importância das interfaces Hápticas. Além disso, falaremos alguns conceitos utilizados em informática úteis para explicar o funcionamento da biblioteca CHAI3D, e sua programação para este trabalho. O compilador utilizado para desenvolver o Simulador Virtual é o Microsoft Visual Studio 2008. Este ambiente apresenta diversas funcionalidades e facilidades para programar em $C/C++$ e também poder embutir em um mesmo executável arquivos e bibliotecas dll's necessários para compilar o CHAI3D.

6.1
Definições Prévias

6.1.1
Thread

Thread, ou linha de execução em português, é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente [79]. O suporte à thread é fornecido pelo próprio sistema operacional (SO), no caso da Kernel-Level Thread (KLT), ou implementada através de uma biblioteca de uma determinada linguagem, no caso de uma User-Level Thread (ULT). Uma linha de execução permite que o usuário do programa utilize uma funcionalidade do ambiente, enquanto outras linhas de execução realizam outros cálculos e operações.

Os sistemas que suportam apenas uma única linha de execução é chamado de monothread, e aqueles sistemas que suportam múltiplas linhas de execução são chamados de multithread.

6.1.2
Framework

Em desenvolvimento de software, um framework ou arcabouço é uma abstração que une códigos comuns entre vários projetos de software, provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Ao

contrário das bibliotecas, é o framework quem dita o fluxo de controle da aplicação, chamado de Inversão de Controle [80].

6.1.3 API

API, de Application Programming Interface (ou Interface de Programação de Aplicativos), é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por programas aplicativos que não querem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços. De modo geral, a API é composta por uma série de funções acessíveis somente por programação, e que permitem utilizar características do software menos evidentes ao utilizador tradicional [81].

6.1.4 Biblioteca

Na ciência da computação, biblioteca é uma coleção de sub-programas utilizados no desenvolvimento de software. Bibliotecas contêm código e dados auxiliares, que provêem serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular. Alguns executáveis são tanto programas independentes quanto bibliotecas, mas a maioria das bibliotecas não são executáveis. Executáveis e bibliotecas fazem referências mútuas conhecidas como ligações, tarefa tipicamente realizada por um ligador [82].

6.1.5 Motores Gráficos

Motor gráfico (ou motor de renderização, ou motor 3D, ou Engine) é uma ferramenta usada para o desenvolvimento de simulações em tempo real, como os jogos, tanto para videogames ou computadores [79]. Eles possuem um conjunto de bibliotecas específicas para simplificar o desenvolvimento do mesmo. O que caracteriza um motor gráfico é um conjunto de subsistemas como o de renderização, que é usado tanto para gráficos 2D como para 3D, o motor de física ou detecção de colisão, e motores de suporte a sons, animação, inteligência artificial, rede e grafo de cena (scene graph), que é a representação hierárquica de todos os objetos que compõem um mundo virtual.

6.1.6 OpenGL

OpenGL é definido como "um programa de interface para hardware gráfico". Na verdade, OpenGL é uma biblioteca de rotinas gráficas e de modelagem, bi (2D) e tridimensional (3D), extremamente portável e rápida. Usando OpenGL, é possível criar gráficos 3D com uma qualidade visual próxima de um efeitos visuais (*ray tracer*). Entretanto, a maior vantagem na sua utilização é a rapidez, uma vez que usa algoritmos cuidadosamente desenvolvidos e otimizados pela Silicon Graphics, Inc., líder mundial em Computação Gráfica e Animação [83].

OpenGL não é uma linguagem de programação, é uma poderosa e sofisticada API (Application Programming Interface) para criação de aplicações gráficas 2D e 3D. Seu funcionamento é semelhante ao de uma biblioteca C, uma vez que fornece uma série de funcionalidades. Normalmente, se diz que um programa é baseado em OpenGL, ou é uma aplicação OpenGL, o que significa que ele é escrito em alguma linguagem de programação que faz chamadas a uma ou mais bibliotecas OpenGL.

6.1.7 ODE

ODE de Open Dynamic Engine, é uma biblioteca de alto desempenho de código aberto, para simulação dinâmica de corpos rígidos. Seus dois principais componentes são o motor de dinâmicas de corpo rígido e o motor de detecção de colisão. Está repleto de recursos para plataformas independentes, e é fácil de usar em C/C++ API. Essa biblioteca implementa tipos comuns e de detecção de colisão integrada com o atrito. ODE é útil para os veículos que simulam objetos em ambientes de realidade virtual e criaturas virtuais. Atualmente, é usado em muitos jogos de computador, ferramentas de criação 3D e ferramentas de simulação.

A ODE é uma biblioteca gratuita, utilizada para a simulação articulada de corpos rígidos dinâmicos estruturados. Seu dois principais componentes dessa biblioteca são: o motor de dinâmicas de corpo rígido e o motor de detecção de colisão. Inicialmente implementada para a linguagem C, hoje é disponibilizada para usuários C++ e Delphi. Ela apresenta ótimos recursos para a simulação em tempo real de objetos em ambientes de realidade virtual. Segundo Smith [84], criador da biblioteca, por ser rápida, flexível e robusta, ela possibilita o desenvolvimento de aplicações de extrema complexidade e com alto nível de precisão sem perda de desempenho, tornando possível desenvolver softwares com simulação física muito próxima da realidade.

A biblioteca ODE permite manipular corpos (objetos) considerando forças, torques e energias, além de outros atributos da física, dando aos corpos da cena comportamento e reações físicas realísticas. Hecker [85] destaca que a simulação de fenômenos físicos faz a "solidez"de um jogo.

6.2 CHAI3D

O CHAI3D (Computer Haptics & Active Interface 3D) é um framework open source (código aberto) para a visualização e simulação de interação háptica em tempo real [86]. Verifica-se um crescimento notável, significativo assinalável da sua utilização nos 5 últimos anos desde sua criação, sendo utilizado em vários projetos e em diferentes domínios, tais como: jogos, simuladores, aplicações para o ensino ou aplicações médicas. Esta plataforma está escrita em C++ e foi concebida com o intuito de tornar fácil e intuitiva a criação de novas aplicações. Este framework fornece as estruturas de dados necessárias para que seja possível a implementação de corpos estáticos, dinâmicos e articulados. A componente gráfica do CHAI3D é conseguida através de um motor gráfico leve, baseado em OpenGL. Esta está também apoiada num modelo bem estruturado de classes, que tem como principal objetivo permitir eventuais extensões de forma simples. Está também disponível a importação de modelos 3D desenvolvidos em aplicações como o Autodesk 3D Studio Max e o Alias Wavefront. O CHAI3D foi concebido de forma a suportar módulos de extensão, sendo que atualmente alguns dos módulos mais relevantes são os motores físicos ODE e GEL que permitem simular corpos rígidos e deformáveis em tempo real. Refira-se ainda que este se encontra preparado para a utilização de diferentes dispositivos hápticos, e que é possível a sua utilização em ambiente Windows, Linux e Mac OSX (Multiplataforma).

Apesar das vantagens supra apresentadas, este framework utiliza para a simulação dos comportamentos físicos de objetos o motor de física ODE, sendo que este está atualmente a ser preterido em relação a outros motores físicos mais recentes. O mesmo não apresenta novas funcionalidades há algum tempo e, em consequência desse decréscimo de melhorias e do aparecimento de mais recentes e melhores motores físicos, tem se verificado a redução de utilizadores. Dada a especial relevância do comportamento físico dos objetos neste trabalho, o motor físico utilizado mostra-se como um dos fatores determinantes de sucesso, sendo que o ODE não dá as garantias pretendidas para o sucesso do trabalho que se pretende desenvolver.

O CHAI3D foi desenvolvido por grupo de pesquisadores da Universidade de Stanford USA, Universidade Católica de Brasilia, Universidade de Siena

Itália e o Instituto EPFL no Lausanne, Suiça.

Podem-se destacar algumas características do CHAI3D:

- Renderização gráfica e encapsulamento das funcionalidades da API para o processamento gráfico com OpenGL.
- Renderização gráfica e háptica com detecção de colisões para malhas triangulares e controle de gravidade.
- Importação de modelos com extensão .obj e .eds.
- Interação com superfícies lisas virtuais, através do Proxy háptico.
- Comunicação com muitas placas eletrônicas digitais/análogas de E/S.
- Integração com ActivesX para aplicações portais-embarcados.
- Disponibilizar mecanismos de navegação como translação e rotação de objetos 3D.
- Disponibilizar mecanismos de mapeamento de texturas, câmera, luzes, transparência e uma interface de contato virtual.
- Integração com ODE (Open Dynamics Engine) para interação háptica com corpos rígidos.

O CHAI3D suporta os seguintes dispositivos hápticos:

- Delta.x e Omega.x da Force Dimension (`http://www.forcedimension.com/products`).
- PHAToM da SensAble Inc. (`http://www.sensable.com/products-haptic-device.htm`).
- Novint Falcon® Technologics Inc. (`http://www.novint.com/index.php/novintfalcon`).
- Freedom 6S de MPB Technologies Inc Technologies Inc. (`http://www.mpb-technologies.ca/mpbt/mpbt_web_2009/_en/6dof/index.html`).

O CHAI3D suporta os seguintes sistemas de compilação:

- Microsoft Visual Studio 6.
- Microsoft Visual Studio.Net.
- Borland C++ Builder.
- Cygwin - gcc.
- Linux - gcc.

6.2.1 Efeitos Hápticos

A cena gráfica do CHAI3D fornece um conjunto de efeitos hápticos que podem ser atribuídos para superfícies de objeto implícitos. Esses efeitos são computados utilizando o método *computeLocalInteraction* de local interação de cada objeto. A malha, ou qualquer outro objeto complexo que não utilize esse método, não é capaz de aplicar os efeitos hápticos, porque não há forma de calcular um ponto de interação projetada a partir de um algoritmo de objeto genérico. Só o algoritmo para o ponto *Proxy* é usado para esse objetos para calcular forças.

Os efeitos hápticos em base à classe *cGenericEfect* na API são o seguinte:

- Efeito Magnético, *cEffectMagnet* proporciona um campo magnético perto do objeto.
- Efeito Viscosidade, *cEffectViscosity* proporciona um efeito de uma ferramenta movendo-se através de um fluido.
- Efeito de Vibração, *cEffecVibration* proporciona um efeito de vibração com uma especificada frequência e amplitude.
- Efeito de superfície, *cEffectSurface* proporciona um efeito de superfície básica para uma ferramenta que pressiona um objeto.
- Efeito Stick-slip, proporciona um efeito de deslizamento em um objeto sobre outro com aderência produzido por atrito.

6.2.2 Módulo ODE

O CHAI3D não tem implementada sua própria simulação de corpos rígidos. No entanto, existe um módulo que conecta a cena gráfica do CHAI3D com a biblioteca ODE. A comunicação entre CHAI3D e ODE é controlada pelas classes *cODE*, *cODEWorld* e *cODEGenericBody*. O API contém bibliotecas ODE pré-compiladas tanto para ligações estáticas dinâmicas, com dupla precisão.

Cada objeto na simulação com ODE tem que ser adicionado a um específico mundo ODE. Tal objeto é definido como um corpo genérico ODE com propriedades de simulação física e um corpo CHAI3D modela a imagem na cena gráfica.

O módulo ODE habilita a criação de caixas dinâmicas, esferas, cápsulas e malhas. Planos estáticos também estão disponíveis e a gravidade global pode ser definida como um vetor tridimensional que descreve uma força (Figura 6.1).

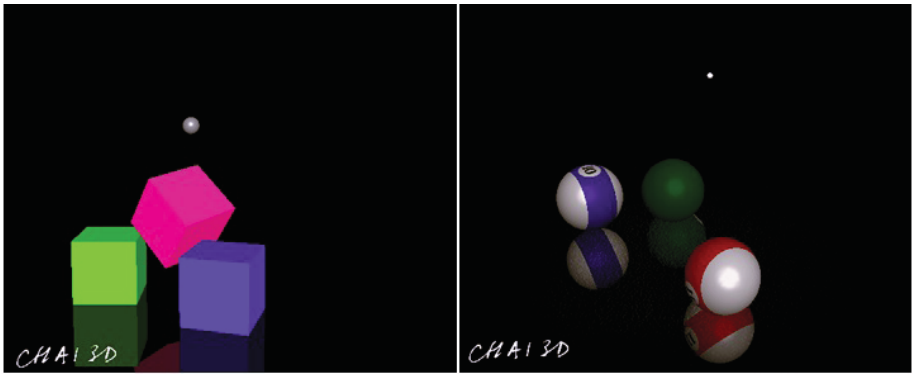

Figura 6.1: Exemplo do módulo ODE em CHAI3D

6.2.3 Módulo GEL

A tecnologia háptica utiliza uma implementação de uma simulação de corpo deformável mais do que qualquer outra tecnologia. O CHAI3D oferece um módulo para criar tais objetos deformáveis na cena gráfica, o qual usa o motor gráfico dinâmico GEL desenvolvido pela Stanford University. Assim como no módulo ODE, o módulo GEL é implementado como um mundo separado (*cGELWord*) de objetos deformáveis. A idéia principal por trás da deformação é uma modelo esqueleto feito de nós (cGELSkeletonNode) e elos (cGELSKeletonLink) entre eles.

Os Nós são representados como esferas com um determinado raio e massa ligados com molas definidos pelas propriedades de deflexão por alongamento e torção (Figura 6.2). Cada nó tem propriedades físicas (amortecimento linear, amortecimento angular, definição de campo de gravidade) e fornece métodos para controle de força e torque. O módulo GEL proporciona uma forma simples de adicionar objetos deformáveis para a cena gráfica.

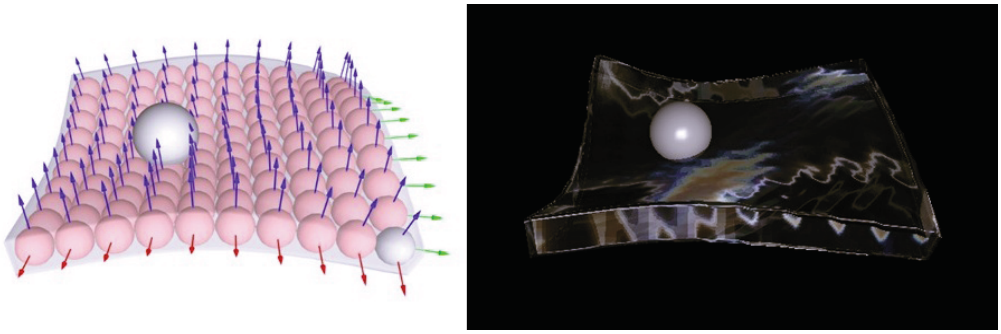

Figura 6.2: Exemplo do módulo GEL em CHAI3D

6.2.4
Arquitetura da Interface Háptica

Os algoritmos de renderização háptica calculam as forças de interação entre a representação da interface tátil dentro do ambiente virtual e os objetos virtuais que povoam o meio. Além disso, esses algoritmos garantem que o dispositivo háptico execute as forças corretamente sobre o operador humano.

Chamamos *avatar* à representação virtual na interface háptica através da qual o usuário interage fisicamente com o ambiente virtual. É evidente que a escolha do *avatar* depende do que está sendo simulado e também da capacidade do dispositivo háptico. O operador controla a posição do *avatar* dentro do ambiente virtual. O contato entre o *avatar* da interface e do ambiente virtual desencadeia forças de ação e reação que dependem também da geometria do *avatar* do tipo contato, que regula essas forças.

Dentro de um determinado aplicativo, o usuário pode escolher entre os diferentes *avatares*. Por exemplo, uma ferramenta cirúrgica pode ser tratada como um objeto volumétrico de troca de forças e posições com o usuário em espaço 6D, ou como um ponto para representação da ferramenta, para a troca de forças e posições em um espaço 3D. Vários componentes compõem tipicamente um algoritmo de renderização háptica, ilustrado na figura 6.3

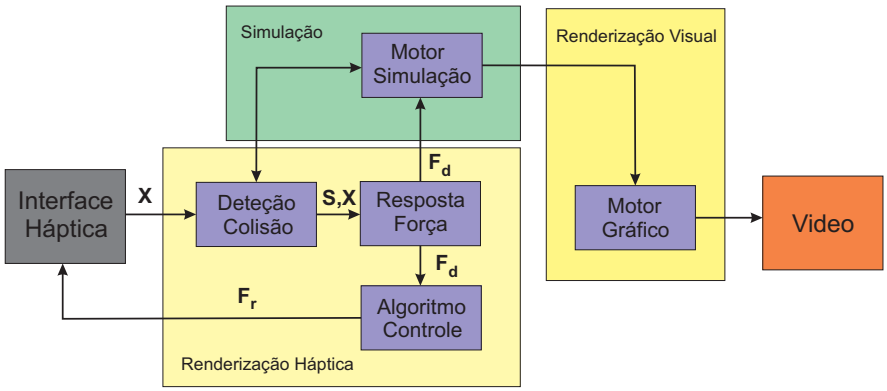

Figura 6.3: Arquitetura da Interface Háptica no CHAI3D

O algoritmo de *Collision-detection* detecta as colisões entre os objetos e *avatares* no ambiente virtual e fornece informações sobre onde, quando e, idealmente para que extensão de colisão (penetração, recortes, área de contato, e assim por diante) tem acontecido.

O algoritmo de *Force-response* calcula a força de interação entre os *avatares* e objetos virtuais, quando uma colisão é detectada. Esta força se aproxima, tanto quanto possível, às forças de contato que normalmente surgem durante o contato entre os objetos reais. Esse algoritmo opera normalmente

nas posições do *avatar*, nas posições dos objetos dentro do ambiente virtual, e no estado de colisão entre os *avatares* e objetos virtuais. Seus valores de retorno são normalmente vetores de força e torque aplicados ao corpo do dispositivo háptico.

Limitações de hardware impedem que os dispositivos hápticos apliquem a força exata calculada pelos algoritmos de força para o usuário. Algoritmos de controle comandam o dispositivo de tal forma que seja minimizado o erro entre as forças ideais e aplicáveis.

A natureza de tempo discreto dos algoritmos de renderização háptica muitas vezes torna isso difícil. Vetores desejados de força e torque calculados pelos algoritmos de força alimentam os algoritmo de controle que vai ser comandado para o dispositivo háptico. Um típico ciclo háptico consiste da seguinte sequência de eventos:

- Algoritmos de controle de baixo nível, que amostram a posição nas articulações pelos sensores do dispositivo háptico.
- Este algoritmo de controle combina a informação recolhida de cada sensor para obter a posição do dispositivo no espaço cartesiano, que seria a posição do *avatar* no ambiente virtual.
- O algoritmo de detecção de colisão usa a informação de posição para encontrar colisões entre objetos e *avatares* e relata o grau resultante da penetração ou recuo.
- O algoritmo de força computa a interação de forças entre *avatares* e objetos virtuais envolvidos na colisão.
- A resposta do algoritmo de força envia as forças de interação para os algoritmos de controle, aplicadas sobre o operador através do dispositivo háptico, enquanto se mantém um comportamento globalmente estável.

O mecanismo de simulação em seguida usa as mesmas técnicas de interação para calcular o seu efeito sobre objetos no ambiente virtual. Embora não existam regras sobre a frequência dos algoritmos hápticos, a taxa comum é de 1KHz. Essa taxa é aceitável, mas para maiores taxas pode-se proporcionar um maior realismo nas sensações de textura, vibração etc.

6.3 Desenvolvimento do Ambiente Virtual

Inicialmente se projetou um ambiente virtual simples, para interagir com o dispositivo Falcon. No ambiente virtual se encontra uma caixa fixa de cor azul, mostrada na figura 6.4, com um orifício na parte central. Além disso, se representa com uma esfera rígida de cor cinza (*Avatar*) o efetuador terminal do Falcon. Esse ambiente será usado posteriormente para avaliar a dificuldade de atravessar o avatar pelo orifício com ou sem realimentação de força.

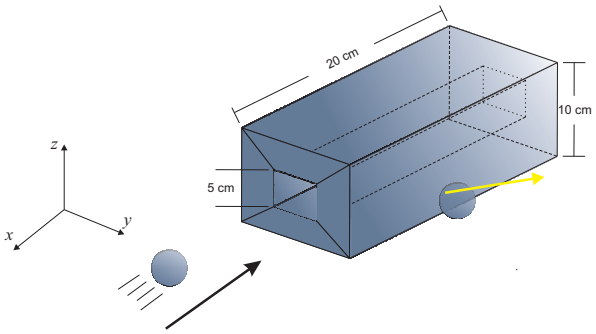

Figura 6.4: Desenho da caixa fixa com orifício na parte central

Esta esfera será aquela que interage com a caixa fixa, enviando uma realimentação de força para o usuário. Também adiciona-se ao programa uma visualização, a posição em tempo real da esfera, e as coordenadas do vetor força criado pela colisão dentro do ambiente virtual.

A figura 6.5 apresenta o ambiente virtual simples, implementado no programa.

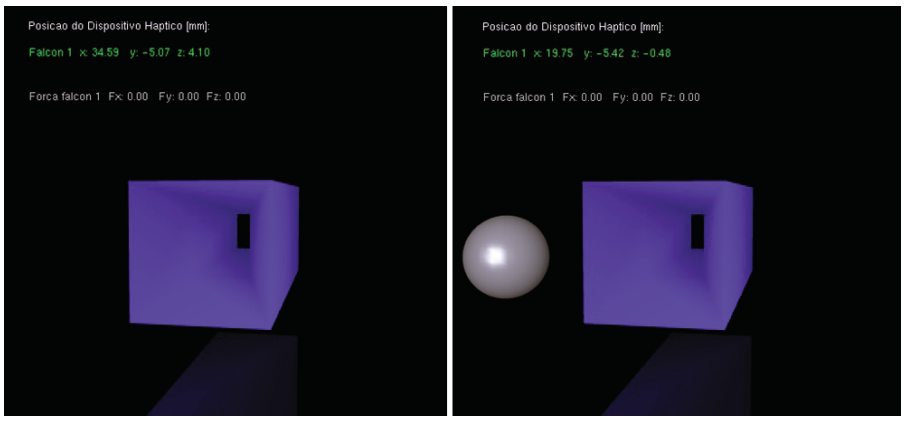

Figura 6.5: Implementação da caixa fixa com orifício na parte central

Para desenhar a caixa, implementou-se uma função *createObjeto()* que utiliza os recursos do Opengl para desenhar superfícies. Com a classe *CMesh* do

CHAI3D podemos computar os vetores normais para a iluminação e também computar a detecção de colisões no objeto.

Adicionou-se ao ambiente uma representação gráfica no espaço do vetor força de reação, criado pelas colisões, mediante uma linha branca vide figura 6.6.

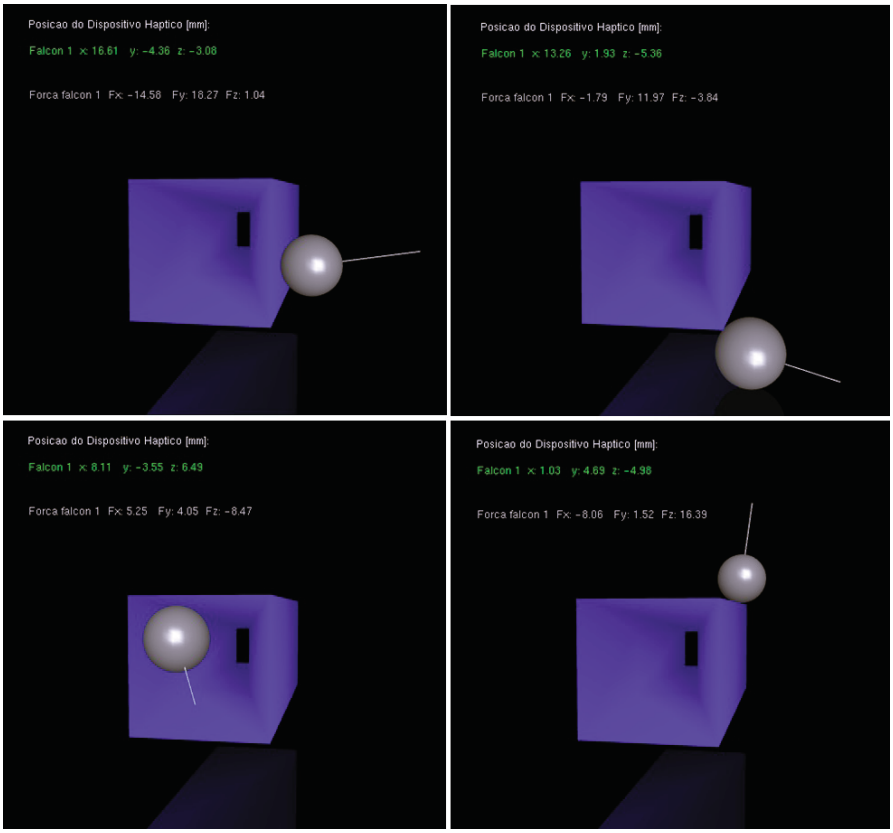

Figura 6.6: Implementação dos vetores de força

6.3.1
Simulador do Robô Manipulador

O algoritmo 1 mostra como foi realizada uma metodologia para desenhar e implementar a modelagem do robô manipulador no ambiente virtual.

Algorithm 1 Desenho e implementação da modelagem do Robô Manipulador

```
1: Desenho do Robô com a função Robot_Schilling
2: Implementação da Cinemática Direta na função CinemaDireS
3: Implementação da Cinemática Inversa na função CinemaInverS
4: Implementação da Dinâmica Direta na função NewtonEulerS
5: Implementação da Dinâmica Inversa na função WalkerOrinS
6: while Encontra-se na área de trabalho do
7:    if dado de entrada é ponto singular then
8:       Computar q → q_d + Δq.
9:       return false
10:   else
11:      computar nova posição q → q_d
12:   end if
13:   Computar a dinâmica do robô.
14: end while
15: return true
```

A figura 6.7 mostra o robô manipulador usado na teleoperação, de 6 graus de liberdade. Cada peça que o compõe foi desenhada em três dimensões utilizando-se as bibliotecas de modelagem em 3D. Após o modelo tridimensional ser construído com as coordenadas e comprimentos de seu projeto, este é importado para o ambiente virtual.

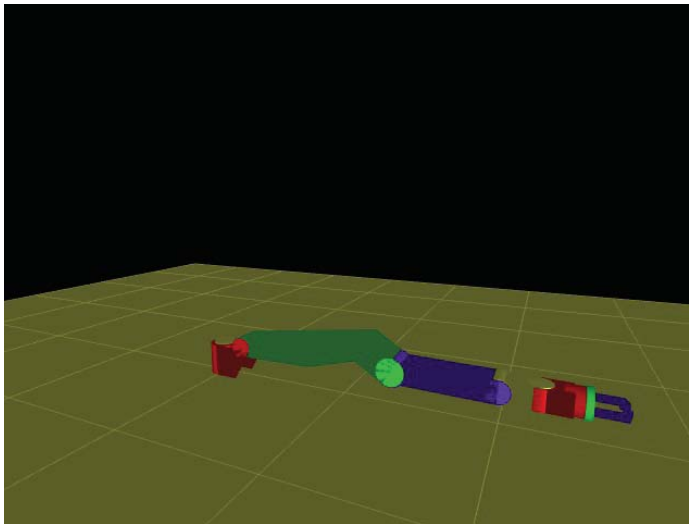

Figura 6.7: Desenho do Robô Manipulador Schilling Titan IV

Com as peças todas disponíveis são necessárias definir as regras dinâmicas do processo, os motores, acionadores, e sensores. Estes passos estão descritos por módulos da seguinte forma:

Para obter uma aproximação real do comportamento de um robô manipulador, inicialmente se definiram as orientações nos eixos de cada atuador de acordo o método de Denavit-Hatenberg.

Pode-se visualizar na figura 6.8 alguns pontos de posicionamento do robô como teste. Aqui, o robô está sendo dirigido para vários pontos coordenados com diversas orientações dentro do ambiente virtual.

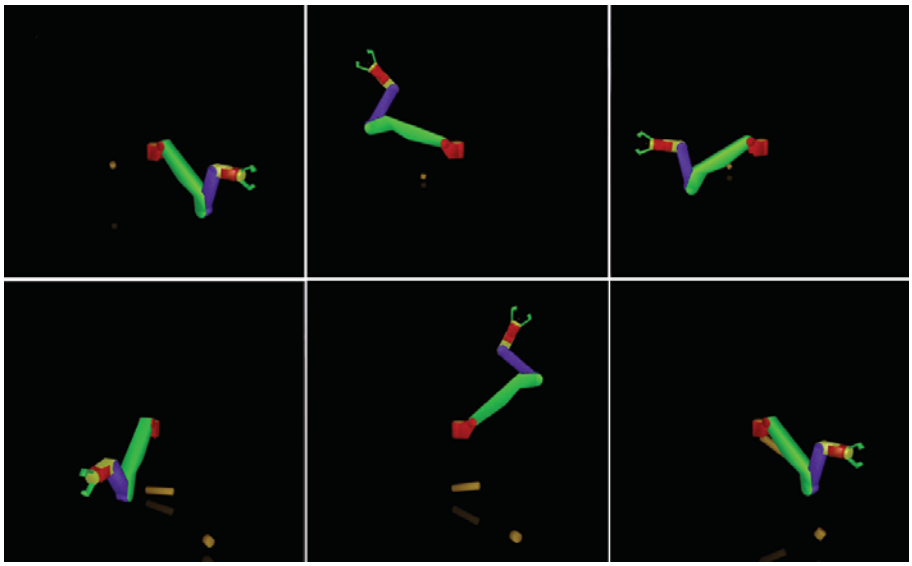

Figura 6.8: Implementação da Cinemática Inversa para o Robô Manipulador

Conforme estudado anteriormente, no Capítulo 3, o conhecimento do modelo dinâmico do robô é de extrema importância para o projeto dos controladores Embora tenha desenvolvido matematicamente o modelo do robô utilizado, para obter um melhor realismo ainda, implementaram-se as funções de detecção de colisões com as funções da biblioteca ODE embarcadas no CHAI3D. Com essas funções, os elos do robô e os objetos que interagem dentro do ambiente virtual não poderão ocupar um mesmo lugar no espaço para cada instante da simulação. A figura 6.9 mostra como o manipulador agarra uma barra e se movimenta junto com ela.

Com as bibliotecas de ODE, é fácil mostrar na simulação trajetórias realistas quando dois objetos interagem, sem precisar implementar outras equações de movimento além das do manipulador.

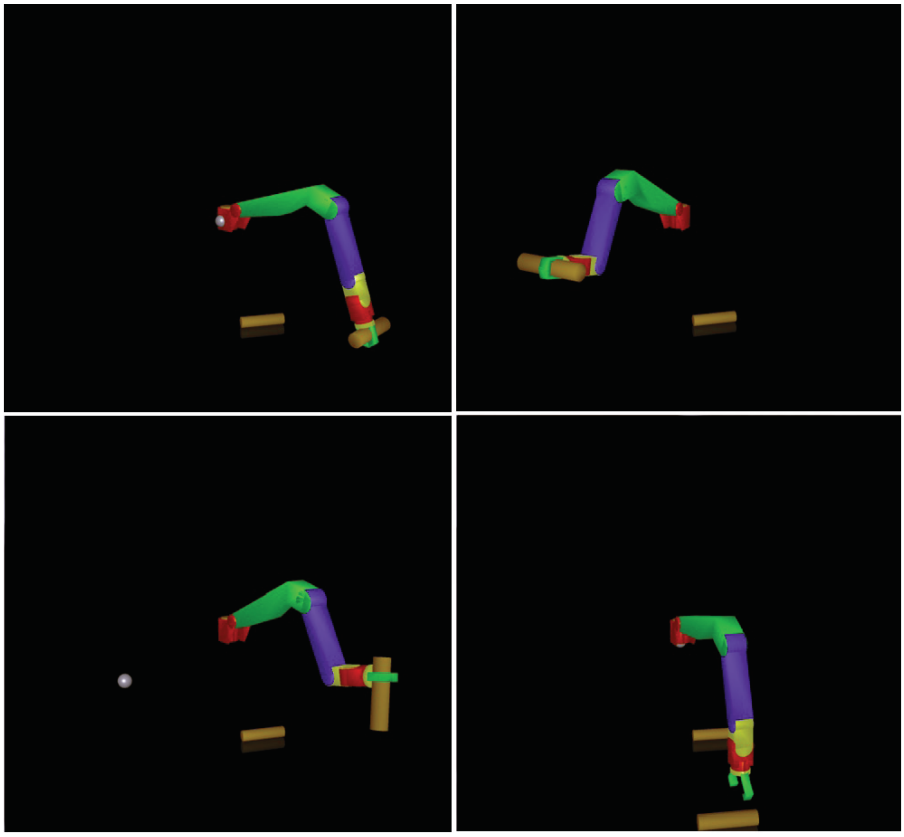

Figura 6.9: Implementação do algoritmo de colisões

6.3.2 Integração da Interface Háptica com o Ambiente Virtual

Depois de ter implementado os comportamentos dinâmicos e cinemáticos para o robô manipulador, com a modelagem matemática dentro do ambiente virtual, o passo seguinte é a integração com o dispositivo háptico, que no inicio será feita com só um dos dois Falcons. O movimento de um Falcon é somente em 3 graus de liberdade (posição x,y,z). O algoritmo 2 mostra como foi realizada uma metodologia para juntar o robô manipulador com a interface háptica.

O controle de forças foi feito com a função da classe *cGenericHapticDevice*, que envia as forças de reação ao dispositivo háptico por meio da porta USB.

Por fim, para o projeto da interfaces hápticas para teleoperar o robô com uma posição e orientação desejada, implementou-se o algoritmo 3 abaixo

A integração do robô manipulador com o dispositivo háptico possibilita enviar forças para gerar torques e perceber uma sensação de movimento em 5 graus de liberdade com o algoritmo desenvolvido no capítulo 5.

Algorithm 2 Robô Manipulador comandado com o Falcon

```
1: Incluir Algoritmo 1
2: while Encontra-se na área de trabalho do
3:    Ler posição do Falcon
4:    if dado de entrada é ponto singular then
5:       Computar q → q_d + Δq.
6:       return false
7:    else
8:       computar nova posição q → q_d
9:    end if
10:   Computar a dinâmica do robô.
11:   Aplicar o algoritmo de controle para a teleoperação
12:   Computar colisões
13:   Enviar forças de reação da colisão
14: end while
15: return true
```

Algorithm 3 Integração da Interface Háptica com o Ambiente Virtual

```
1: Incluir Algoritmo 1
2: while Encontra-se na área de trabalho do
3:    Ler as posições dos Falcon's
4:    Transformar as posições do Falcon ao sistema de coordenadas global
5:    Sincronizar As áreas de trabalho
6:    if dado de entrada é ponto singular then
7:       Computar q → q_d + Δq.
8:       return false
9:    else
10:      computar nova posição q → q_d
11:   end if
12:   Computar a dinâmica do robô.
13:   Aplicar o algoritmo de controle para a teleoperação
14:   Computar colisões
15:   Enviar forças de reação da colisão
16: end while
17: return true
```