UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Welinton Dias

Ambiente de Desenvolvimento de Manufatura Virtual

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Área de Concentração: Automação e Sistemas Elétricos Industriais.

Orientador: Prof. Leonardo de Mello Honório
Co-orientador: Prof. Luiz Edival de Sousa

Junho de 2006

ITAJUBÁ - MG

Ao meu filho Lucas Henrique Dias.

"Projetistas fazem canais, arqueiros atiram flechas, artífices modelam a madeira e o barro, o homem sábio modela a si mesmo".

Buda Gautama Sakyamuni

COMISSÃO DE AVALIAÇÃO

_____

_____

_____

_____

_____

Agradecimentos

Aos membros e professores que fazem parte do CRTI.

Ao meu Orientador, Leonardo de Mello Honório, meus agradecimentos pela orientação e auxílio na realização deste trabalho.

Ao CNPQ pelo financiamento ao projeto, acreditando nos frutos que este possa vir a proporcionar.

A todos aqueles que direta ou indiretamente colaboraram para que este projeto fosse concluído.

Resumo

A demanda por equipamentos importados em instituições de ensino é um fato conhecido. Este problema se agrava quando a área envolvida é o controle e automação da manufatura, onde os equipamentos utilizados possuem um alto custo. Para que um treinamento de controle destes equipamentos seja tido como satisfatório é necessário que um aluno, além do estudo teórico, faça diversas simulações e observe as respostas dos diversos equipamentos e das formas de programação e ajuste. Para contornar esta situação este projeto propõe a montagem de um ambiente virtual para treinamento em robótica e manufatura integrada. Utilizando técnicas semelhantes às dos tão conhecidos jogos em realidade virtual será construído um ambiente onde será possível a visualização, montagem e programação de quaisquer processos de manufatura, como robôs, mesas pneumáticas, esteiras, etc. A qualidade gráfica da simulação, bem como seu comportamento físico coerente e sua flexibilidade permitem uma imersão realística no ambiente virtual.

Abstract

The demand for foreign equipments in teaching institutions is well-know. To properly train an engineer, a lot of hours in several equipments are required. This is a problem if the equipments are expensive. The problem increases if manufacturing automation and control are focus. The number of equipments needed to simulate real industrial problems is enormous and even if considering the unlikable possibility that founds are not the problem, the physical space would be. To overcome this problem, the present work introduces a robotic and manufacturing framework. Based on the virtual reality present on games this framew^{o}rk enables one to generate any desirable rigid-body scenario raging from manufacturing process to robot, manipulators, pneumatic engines and others. The graphics quality as well its physical behaviors enables an operator to live a realistic immersion in the virtual environment.

Índice

Indice de Figuras

Índice de tabelas

1. Introdução

1.1. Aspectos Gerais

Com o avanço dos hardwares e softwares de computação uma nova modalidade de sistemas de simulação utilizando conceitos de realidade virtual começou a se apresentar no mercado. Esta nova linha busca mostrar uma visão em três dimensões de um sistema, bem próxima à visão em um ambiente real, causando uma imersão do usuário no mundo virtual, de forma que qualquer distúrbio ou interferência de um comando ou equipamento na simulação será representado fielmente, como à ocorrência do fato nos objetos reais. A literatura mostra trabalhos [1, 2] que utilizam este tipo de ambiente para testes e simulações de novos conceitos e idéias. Porém, o grande problema destes sistemas ainda é a baixa qualidade da realidade simulada, devido a simplificações extremas nos modelos de cálculo físico, ou mesmo ineficiência do hardware utilizado para o processamento. Isto ocorre porque a matemática envolvida em uma simulação de realidade virtual é extremamente complexa, necessitando de um tempo relativamente grande para a execução dos cálculos com uma precisão aceitável. A ocorrência destes fatos acarreta alguns problemas: colisões são mal processadas, uma baixa qualidade gráfica é apresentada, ou existe a falta de uma simulação física de qualidade. Nos sistemas que superam alguns destes quesitos componentes pré-concebidos e inflexíveis deixam a experiência de utilizá-los longe da realidade a que eles se propõem.

Para contornar estes problemas o presente trabalho propõe a implementação de um ambiente tridimensional para o planejamento flexível e simulação em tempo real de um processo qualquer de manufatura utilizando avançadas técnicas para o processamento de colisões (*Bounding Volumes*) bem como um modelo físico compatível com a realidade necessária para a construção de um ambiente virtual. Para tanto será necessário criar um

framework (conjunto de classes e objetos) para realidade virtual dedicado ao processamento, design, implementação e reprodução da dinâmica encontrado nos equipamentos da realidade, seja através de máquinas de funcionamento independente ou de uma estrutura colaborativa de manufatura e linhas de produção em série. A classificação das funcionalidades desejadas apresenta uma solução modular composta de três diferentes níveis:

- Módulo compositor da manufatura;
- Módulo de programação;
- Módulo de comunicação e execução em tempo real.

Antes, porém, da utilização de qualquer um dos módulos, é necessário que se tenha a representação em três dimensões das máquinas a serem simuladas, para fim de representação gráfica. Esta representação é gerada em um software de modelagem em três dimensões como o *Maya* ou o *3D Studio* e exportado para um formato que é utilizado dentro do ambiente de realidade virtual.

Com o desenho em três dimensões construído de forma estática, será criado dentro do *Módulo Compositor* um modelo físico que represente a realidade da máquina. Neste módulo será definida a dinâmica da máquina, através de rotações e translações que ocorrem em resposta a comandos pré-estabelecidos, o posicionamento de sensores e resposta que estes irão gerar caso acionados, e também atos de agarrar objetos e quais as partes da máquina podem ser seguradas ou deslocadas de posição. Com a integração de todos os objetos componentes é gerada uma árvore de movimentação estrutural onde o limite de alcance e a ação de cada elemento é determinada.

O *Módulo de Programação* é responsável pelo controle da dinâmica do ambiente, baseando-se no modelo gerado pelo *Módulo Compositor*. O processo é realizado em dois níveis: programação e execução. A programação proporciona acesso à linguagem Ladder, obedecendo à norma IEC61131-3 [3]. Como em um ambiente real, o programa de controle da simulação é desenvolvido em modo *off-line*, isto é, sem a necessidade do ambiente de tempo real estar sendo executado. O nível de execução do Módulo de Programação é responsável pela interação deste com o *Módulo de Comunicação e Execução em Tempo Real*. Esta operação será descrita no capítulo 3.

O *Módulo de Comunicação e Execução em Tempo Real* tem como objetivo, além de fornecer uma biblioteca de comunicação para ambientes distribuídos, de forma a simular programas desenvolvidos em Ladder, fornecer métodos para teste de colisão entre os corpos componentes do ambiente, e a sua respectiva resposta física, alta qualidade de imagem em realidade virtual; e processar, em tempo real, a árvore estrutural de movimentos realizando assim somente movimentos permitidos pela estrutura do próprio modelo.

O mais importante aspecto a ser levado em consideração neste tipo de ambiente, além da qualidade gráfica, é a precisão da física envolvida. Esta precisão está principalmente associada com uma correta identificação e tratamento das colisões entre os elementos, e a qualidade do modelo matemático adotado para a representação de realidade. Sendo assim é necessário identificar, com a maior precisão possível, a posição e estado de cada estrutura presente no ambiente. A literatura mostra uma série de referências [4, 5, 6, 7], a maioria baseada em árvores de volumes de contorno que representam estruturas organizadas em nós. Estas árvores são a representação de um dado elemento, formado por uma estrutura complexa de pontos em três dimensões por um conjunto de volumes de mais fácil representação, como caixas ou esferas. Desta forma uma correta análise da estrutura da árvore e do volume empregado na simplificação é vital para o bom desempenho da simulação.

Testes realizados por este trabalho identificaram que um volume simplificado chamado OBB (Oriented Bounding Box) possuiu um rápido tempo de resposta mesmo se muitos objetos estão presentes. Este OBB é na verdade uma caixa definida por raios em três dimensões e uma orientação qualquer no espaço, que engloba um elemento componente da máquina virtual. Para que o objeto fique eficientemente representado através de uma caixa, seria interessante fazê-lo através da caixa de menor volume que contenha o objeto, assim não haveria uma grande perda de espaço dentro da caixa (a diferença de volume entre o objeto que foi delimitado e a própria caixa). Na questão pertinente ao alinhamento da caixa com objeto foi utilizado um método estatístico que emprega a matriz de covariância para a determinação dos pontos prováveis de alinhamento da figura, sendo que os autovetores desta matriz, quando a mesma é gerada por um problema de origem geométrica, mostram os eixos de maior e menor variação entre as coordenadas dos pontos, e esses tendem ao alinhamento com o objeto. Portanto os

autovetores desta matriz são utilizados como os eixos base para a construção de nossa OBB.

O propósito dos volumes de contorno ser aplicados nos testes de colisão é a velocidade dos testes de colisão entre os mesmos, que sempre são figuras de fácil manipulação, mas durante algum tempo as OBBs não eram utilizadas devido a um grande número de cálculos para determinar sua interpenetração. Até que Gosttchalk [7] propôs a utilização de um teorema (Teorema dos Eixos de Separação) para a determinação de colisão entre duas caixas alinhadas arbitrariamente no espaço. Através deste novo teste as condições para a determinação da colisão entre OBBs que eram de cerca de 50, foram reduzidos para 15, os 15 possíveis eixos de separação entre elas, possibilitando uma resposta satisfatória em tempo de execução, e a aplicação desta técnica em ambientes complexos de simulação em três dimensões.

Mas alguns problemas estão associados a esta metodologia da matriz de covariância: primeiro, uma concentração de pontos em uma parte do objeto pode levar a um eixo desalinhado, aumentando o número de testes de colisões a serem realizados para uma boa precisão. Para contornar este problema foi proposto um algoritmo baseado em busca em árvore para se obter o melhor alinhamento possível.

1.2. Objetivo

O objetivo principal deste trabalho é o desenvolvimento de um conjunto de classes e objetos (*framework)* para simulação de processos de manufatura em realidade virtual, utilizando para isso o ambiente de desenvolvimento *Visual Studio. NET* da *Microsoft*, que disponibiliza uma ferramenta muito importante para a programação voltada à realidade virtual, a API *DIRECTX*.

O *DIRECTX* é uma API (Advanced Programming Interface) que permite interações de alto desempenho em tempo real entre um aplicativo e o hardware gráfico do computador, deixando a cargo do desenvolvedor balancear a qualidade de exibição de objetos na tela e o tempo de execução destas exibições. É o responsável pela apresentação de toda a simulação virtual no computador através de métodos e interfaces expostas para utilização imediata. A linguagem de programação escolhida para o desenvolvimento deste trabalho

foi o c#, que é uma linguagem relativamente nova, mas totalmente orientada a objetos, permitindo uma abordagem mais focada no desenvolvimento das diversas partes que são necessárias para o funcionamento da simulação virtual. Esta linguagem possui também intrínseca as interfaces do DIRECTX, facilitando a utilização deste.

1.3. Organização do Trabalho

O trabalho está organizado da seguinte maneira: o capítulo 2 mostra o *DIRECTX*, que é a ferramenta da *Microsoft* para a exibição de ambientes tridimensionais, discute as suas principais características e propriedades, as matrizes de transformação e como representar os objetos no espaço ditado por ele.

O capítulo 3 fornece informações sobre os módulos que compõem a manufatura virtual, e sobre a montagem de um ambiente de teste: um cenário composto por um braço robótico da Minipa, simulado pelo sistema, e as operações necessárias para a configuração dos sensores necessários para uma correta operação desta simulação. Traz também informações sobre os algoritmos de colisão aplicados e como as suas respostas são tratadas pelos métodos físicos para uma correta simulação da realidade envolvida.

Finalmente o capítulo 4 apresenta os resultados obtidos com a simulação realizada para análise e o capítulo 5 mostra algumas sugestões de desenvolvimentos futuros envolvendo o trabalho, comentando possíveis evoluções tanto do ponto de vista de qualidade quanto da velocidade da simulação, que apesar de não ser um problema ainda pode ser melhorada para permitir um ambiente virtual composto por mais objetos com dinâmicas completamente independentes.

2. Ferramenta Gráfica Utilizada

Este capítulo aborda as principais características do DIRECTX. Serão mostrados os dispositivos (Devices), transformações em três dimensões, conceitos de iluminação, vértices e faces

O *DIRECTX* é uma API desenvolvida pela *Microsoft* para desenvolvimento de aplicativos que necessitem da capacidade total que o hardware gráfico disponibiliza, tanto em qualidade de renderização quanto em quesitos de tempo de execução. Para uma correta utilização deve-se entender corretamente o seu funcionamento, permitindo as abordagens mais eficientes possíveis.

No caso da simulação em tempo real de um processo de manufatura é necessário que as imagens mostradas na tela sejam de elevada qualidade, já que estamos nos utilizando de realidade virtual. Também necessitamos de bastante tempo computacional para que as informações relativas a possíveis colisões sejam tratadas adequadamente, alimentando um modelo físico responsável por gerar as respostas realistas das interações entre os diversos equipamentos que compõem do processo.

Vamos iniciar então mostrando o funcionamento básico do *DIRECTX* e seus recursos, analisando também a representação de objetos em três dimensões, as matrizes de transformação que serão aplicadas a estes descrevendo movimentos absolutos e relativos, quando for necessário exibi-los na tela, e as características de iluminação que regem a simulação virtual.

2.1. Representação do Objeto em 3D

Os objetos em três dimensões que são utilizados em um ambiente de realidade virtual necessitam de uma representação adequada para que sejam utilizados em alto desempenho. Assim se faz necessário apenas armazenar informações estritamente necessárias à apresentação destes. A informação mais básica que um objeto possui são seus vértices, que definem a posição de cada ponto que compõe o desenho, sendo que algumas de suas características são determinadas pelas seguintes grandezas:

- *Coordenada* – Refere-se à posição ocupada pelo vértice no espaço local do modelo, define a geometria do objeto.

- *Normal* – É o vetor que representa qual a direção aponta para fora do objeto que o ponto compõe. É utilizado para cálculo de colisões e iluminação.

- *Cor* – A cor do ponto na tela.

- *Textura* – Define as coordenadas de uma imagem que pode ser sobreposta à face que o vértice compõe no momento da renderização, substituindo a cor própria do vértice.

Como a figura necessita de mais informações para ser renderizada, além de puramente os vértices que a formam, existe a necessidade de conhecer as ligações existentes entre eles, que são representadas como faces (Figura 1). Estas faces descrevem como os vértices componentes da imagem estão interligados, e como devem ser renderizados para a tela.

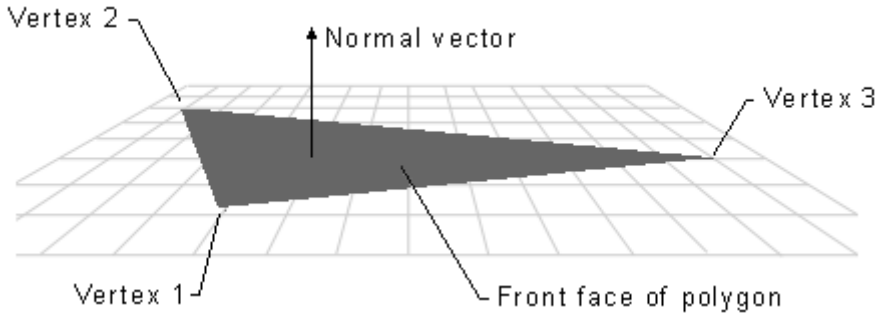

Figura 1 – Vértices, faces e o vetor normal

Para fins de organização e desempenho, os vértices são organizados no *DIRECTX* através de estruturas pré-definidas, para uma operação confiável e robusta, chamadas primitivas, sendo que a mais simples encontrada é uma lista de pontos, mas existem outras mais complexas como leques e listas de triângulos.

Desenhar um modelo em três dimensões na tela requer que este seja decomposto em vértices, e estes então são organizados em faces formando triângulos. Com bases nestas

faces são geradas as primitivas que as representem estas estruturas, que passam então por uma seqüência de cálculos levando em conta a iluminação da cena e a interferência da luz de um objeto em outro, calculando a cor de cada ponto que compõe a tela, e desenhando os objetos presentes na simulação em uma imagem, apresentando então estas figuras ao usuário várias vezes a cada segundo.

O posicionamento dos objetos nos locais desejados é realizado através de uma transformação de posição dos mesmos, que se caracteriza pela aplicação de uma matriz às coordenadas dos vértices, determinando qual o ponto ocupado no espaço por esta imagem. Esta seqüência de transformações de posição e iluminação recebe a denominação de *pipeline* gráfica.

Para que os objetos sejam utilizados em todas estas operações durante a execução da simulação, é necessário que o software tenha uma maneira de representar os objetos componentes da simulação virtual. Mas o software não possui uma interface de desenho, então devemos utilizar um software de modelagem tridimensional para a geração de uma figura que possa ser "entendida" pelo simulador. Este software de modelagem gera então uma representação poligonal dos objetos nele construídos e estas representações serão utilizadas para a execução fiel da simulação gráfica.

Polígono descreve uma figura fechada delimitada por ao menos três vértices, e que pode ser representado através de uma primitiva. O polígono mais simples é um triângulo, e o fato de seus vértices serem todos coplanares facilita muito os cálculos ao longo da pipeline gráfica. Com base nesta simplificação, quaisquer polígonos mais complexos são quebrados e representados através de triângulos na arquitetura do DIRECTX. A Figura 2 mostra a triangulação de um objeto e a Figura 3 um objeto representado de modo sólido.

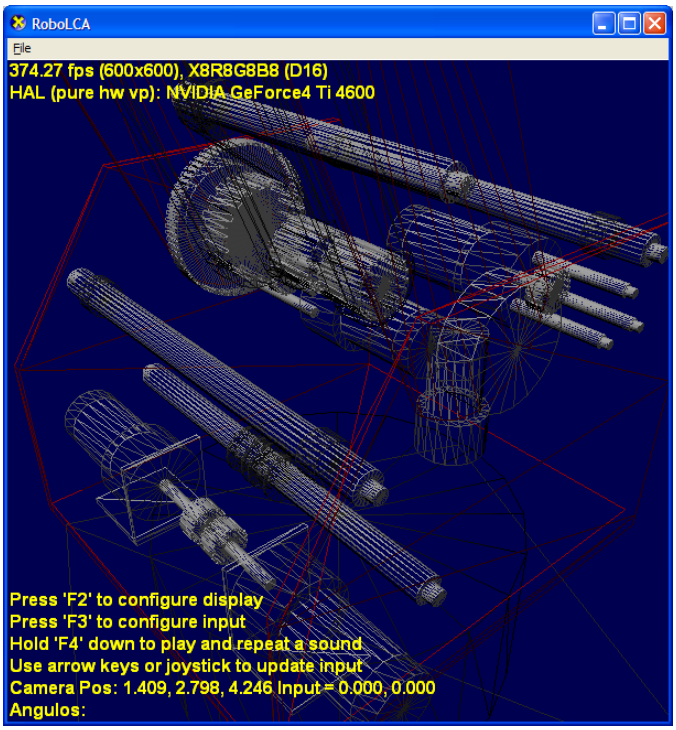

Figura 2 – Triangulação de Polígonos

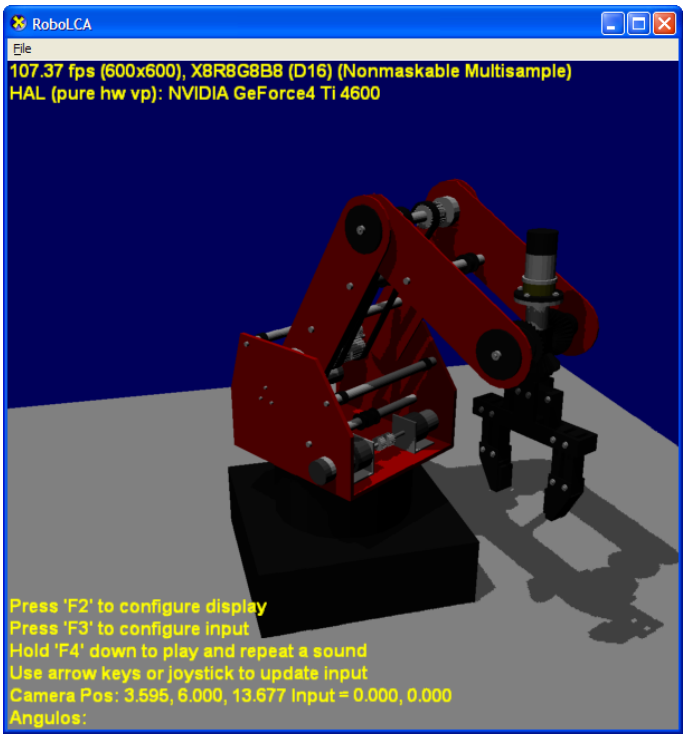

Figura 3 – Objeto renderizado

2.2. Movimentação 3D e Transformações de Espaços

Realizar a correta movimentação de um objeto na tela exige a necessidade de entender o sistema de coordenadas da representação em que ele se encontra. Tipicamente, aplicações gráficas em três dimensões usam dois tipos de sistemas de coordenadas cartesianas: mão-esquerda e mão-direita. Em ambos os sistemas de coordenadas, o eixo x positivo aponta para a direita, e o eixo y positivo aponta pra cima, a variação das representações se encontram no eixo z. Pode-se lembrar em qual direção o eixo z aponta, apontando os dedos da sua mão esquerda ou direita na direção x positiva e rodando os dedos na direção positiva de y. A direção que do polegar aponta, seja aproximando ou distanciando, é a direção que o eixo z positivo apontará para esse sistema de coordenada. A Figura 4 abaixo mostra estes dois sistemas de coordenadas:

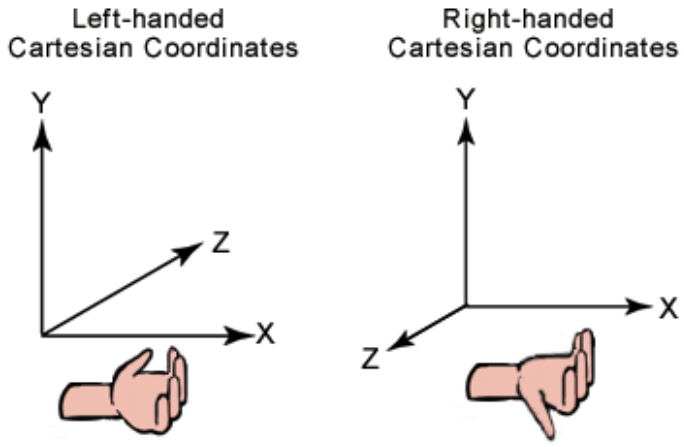

Figura 4 - Eixos de coordenadas da mão esquerda e mão direita

O *DIRECTX* representa os modelos utilizando o sistema de coordenadas de mão esquerda, sendo isso uma convenção.

Independentemente do sistema adotado será sempre necessário realizar transformações nos objetos. Estes, quando lidos do arquivo gerado pelo software de modelagem estão representados em um espaço de coordenadas chamado espaço local, onde somente este

objeto existe e as coordenadas de seus pontos são descritas sem nenhuma translação ou escala. Mas para uma representação em um modelo virtual vários objetos precisam ser escalonados e posicionados com relação a uma origem comum possibilitando a interação entre eles para formar uma cena ou processo. Assim, transformações são usadas para converter a geometria de um objeto de um espaço de coordenadas para outro.

Existem muitas formas de se realizar estas transformações, mas a mais utilizada é através de matrizes, devido à facilidade de manipulação e representação de um espaço vetorial. É possível então aplicar uma matriz para transformar um objeto do espaço vetorial **A** para o espaço vetorial **B**, sendo que a transformação inversa (a própria inversa da matriz) transformará o objeto do espaço **B** para o espaço **A**.

Um espaço vetorial é representado por um conjunto de vetores linearmente independentes, denominado base. Desta forma qualquer vetor dentro deste espaço pode ser obtido através da combinação linear dos vetores da base. Para o caso de um ambiente virtual, cujo espaço de coordenadas é o R3, uma possível, porém não única base é a formada por E = (e1, e2, e3) onde e1 = (1, 0, 0); e2 = (0, 1, 0) e finalmente e3 = (0, 0, 1). Desta forma, qualquer outro vetor em R3 pode ser escrito como combinação linear de E.

Como R3 pode ser formado por infinitas bases, vamos supor agora a existência simultânea de um elemento cujo posicionamento é dado por outra base de R3, denominada W. Um dos problemas nesta situação é a movimentação (rotação, translação ou escala) em uma base de um elemento traçado com referência em outra. Para que isso possa ser realizado é necessária a transformação das coordenadas do elemento de W para E.

Desta forma, qualquer coordenada (x, y, z) pode ser reescrita para (x', y', z') através de uma combinação linear como mostrada na Equação 1, sendo possível expressar a localização de um objeto relativo a outro, rotacionar e escalonar objetos, mudar a posição de visualização, direção e suas perspectivas.

$$
\begin{aligned}
x' &= (x \times M_{11}) + (y \times M_{21}) + (z \times M_{31}) + (1 \times M_{41}) \\
y' &= (x \times M_{12}) + (y \times M_{22}) + (z \times M_{32}) + (1 \times M_{42}) \\
z' &= (x \times M_{13}) + (y \times M_{23}) + (z \times M_{33}) + (1 \times M_{43})
\end{aligned}
$$

Equação 1

Podemos escrever também esta equação na sua forma matricial, o que mostra porque as matrizes são a forma mais utilizada de transformação em três dimensões. A forma matricial da Equação 1 é a Equação 2.

$$[x' \;\; y' \;\; z' \;\; 1\;] = [x \;\; y \;\; z \;\; 1\;] \cdot \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

Equação 2

As transformações mais comuns envolvendo os objetos são translação, rotação e escalonamento, e a matemática associada são mostradas nos itens seguintes.

2.2.1. Translação

A transformação dada pela Equação 3 translada o ponto (x, y, z) para um novo ponto (x', y', z'), que distancia do mesmo (T_x, T_y, T_z).

$$[x' \;\; y' \;\; z' \;\; 1\;] = [x \;\; y \;\; z \;\; 1\;] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Equação 3

2.2.2. Escalonamento

A transformação apresentada na Equação 4 altera a escala do ponto (x, y, z) por valores determinados por S_x, S_y, S_z nas direções x, y e z para o novo ponto (x', y', z').

$$[x' \;\; y' \;\; z' \;\; 1\;] = [x \;\; y \;\; z \;\; 1\;] \cdot \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equação 4

2.2.3. Rotação

As transformações de rotação são dependentes do sistema de coordenadas adotado, sendo que as que são descritas aqui são para sistemas de coordenadas de mão esquerda (o sistema de coordenadas adotado pelo *DIRECTX*), e então podem ser diferentes das matrizes de transformação apresentadas na literatura devido à posição do eixo z. As transformações encontradas na Equação 5, na Equação 6 e na Equação 7 rotacionam o ponto (x, y, z) em volta do eixo 'x', 'y' e 'z' respectivamente produzindo um novo ponto (x', y', z'). Onde teta (θ) representa o ângulo de rotação, em radianos. Os ângulos são medidos em sentido horário olhando ao longo do eixo de rotação em direção à origem do espaço.

$$[x' \;\; y' \;\; z' \;\; 1\;] = [x \;\; y \;\; z \;\; 1\;] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equação 5

$$[x' \;\; y' \;\; z' \;\; 1\;] = [x \;\; y \;\; z \;\; 1\;] \cdot \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equação 6

$$[x' \;\; y' \;\; z' \;\; 1\;] = [x \;\; y \;\; z \;\; 1\;] \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equação 7

2.2.4. Concatenação de Matrizes

Muitas vezes existe mais de uma transformação que deva ser aplicada ao objeto para uma correta representação deste no espaço, então para fins de eficiência podemos concatenar as diversas transformações na ordem em que elas devam ser aplicadas, formando uma única transformação composta que contém os mesmos efeitos de todas as componentes aplicadas isoladamente.

Uma vantagem de usar matrizes é que a concatenação acima descrita é simplesmente uma multiplicação. Isso significa que, para rotacionar e transladar um modelo para alguma posição é necessária a aplicação de duas matrizes, mas podemos multiplicar estas matrizes para produzir uma matriz composta que contém todos os efeitos necessários. Esse processo é mostrado na Equação 8 onde C é a matriz composta que está sendo criada, e M1 até Mn

são as transformações isoladas. Na maioria dos casos, apenas duas ou três matrizes são concatenadas, mas não existem limites matemáticos para a operação.

$$C = M_1 \cdot M_2 \cdots M_{n-1} \cdot M_n$$

Equação 8

A ordem na qual a multiplicação de matrizes é feita é crucial. A Equação 8 reflete a regra "esquerda para direita" da concatenação de matrizes. Isto significa que o efeito visível das matrizes utilizadas para criar uma matriz composta ocorre na ordem esquerda para direita. Uma típica transformação de mundo (que será mostrada na seção seguinte) pode ser ilustrada em uma situação de se movimentar um disco voador, onde se deseja que ele rode em torno do seu centro (o eixo 'y' do espaço do modelo) para depois então transladá-lo para outra localização na cena. Para produzir esse efeito, deve-se primeiro criar uma matriz de rotação com o ângulo desejado para depois multiplicá-lo pela matriz de translação, como mostrado na Equação 9.

$$W = R_y \cdot T_w$$

Equação 9

Caso a ordem da multiplicação seja alterada o efeito visual resultante seria a do objeto, no caso o disco voador, orbitar um centro correspondente com a distância da translação. Matematicamente a explicação da importância da ordem pela qual se multiplica as matrizes é que, ao contrário da multiplicação de dois valores escalares, a multiplicação de matrizes não é comutativa. Multiplicando as matrizes na ordem oposta o resultado é diferente, obviamente afetando o efeito visual de translação do disco voador para sua posição no espaço do mundo, e então rotacionando em volta da origem do mundo.

2.3. Arquitetura Direct 3D

O *Direct 3D* é o componente do *DIRECTX* que trabalha com a visualização de ambientes tridimensionais, permitindo ao desenvolvedor utilizar todos os recursos disponibilizados no hardware gráfico do computador sem a necessidade de programação em linguagem de baixo nível. Este componente irá se comunicar diretamente com a placa de vídeo, tratando das chamadas de rotinas de hardware e permitindo que sejam realizadas as mais diversas operações em alto nível sem prévio conhecimento sobre características da máquina onde o aplicativo deverá funcionar. Isto elimina a necessidade da programação em realidade virtual ser direcionada para apenas um equipamento de hardware isolado, o mesmo código que funciona em uma placa de vídeo produzida pela *Radeon*, também funciona em uma placa da *Nvidia* ou mesmo da *Sis*, que são fabricantes distintos de hardware gráfico para computadores pessoais.

Para um melhor entendimento do ambiente é necessário o conhecimento de como os *Devices* (dispositivos) operam. *Device* é o componente de renderização de realidade virtual. Ele encapsula e armazena os estados de renderização (*Render States*) e executa as transformações e operações de iluminação e rasterização de uma imagem em uma superfície como mostra a Figura 5.

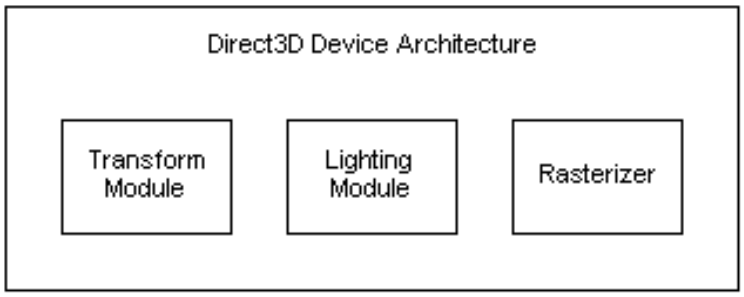

Figura 5 – Arquitetura do Direct 3D

O *device* do *Direct 3D* suporta três modos de operação: um via hardware com rasterização e *shading* acelerados com suporte a processamento diretamente pela placa de vídeo (todas as transformações e cálculos de iluminação são realizados pelo hardware gráfico), um modo de referência, implementado totalmente por software (não há necessidade de hardware gráfico específico instalado no computador) e ainda um dispositivo de software que utiliza as características presentes no hardware da máquina de execução do sistema e emula aquelas que a arquitetura gráfica não suporta.

Pode-se pensar que estes *devices* são como *drivers* separados. Os *devices* acelerados via hardware fornecem muito mais desempenho que os *devices* via software, suportando desta forma mais vantagens, desde que a máquina onde o aplicativo será executado suporte as rotinas necessárias. O código para aplicações desenvolvidas para trabalhar com o *device* via hardware funcionará perfeitamente com os *devices* via software e referencia sem qualquer modificação, independente da quantidade de recursos de hardware disponíveis. Assim os recursos gráficos exigidos pelo aplicativo são todos disponibilizados pela API *Direct 3D*, possibilitando uma alta qualidade tanto em termos de visualização quanto de tempo de execução de softwares que exigem desempenho extremo do computador.

2.4. As Matrizes de Visão, Mundo e Projeção

Como os modelos são todos descritos com relação à sua origem, independentes de outros objetos no mundo, eles estão representados em um sistema de coordenadas conhecido como espaço local, e para representar no mundo virtual estes objetos, necessita-se aplicar transformações nestes para que sejam posicionados em outro espaço, conhecido como espaço global (mundo), onde estão rotacionados, escalonados e transladados para refletir o real estado do processo de manufatura que o objeto representa.

Existem transformações que são pré-definidas na API para executar estas mudanças: as transformações de mundo e as transformações de visão.

Há também a necessidade de corretamente representar o mundo virtual na tela como se ele estivesse sendo filmado por uma câmera, aplicando a perspectiva de distância aos objetos. Para essa finalidade existe a transformação de projeção.

Por questões de desempenho, já sabendo que as matrizes serão aplicadas consecutivamente ao modelo para gerar a renderização, no projeto a matriz de visão já está incluída na matriz de mundo (concatenada).

2.4.1. A Matriz de Mundo (World Transform)

A matriz de mundo (transformação para o mundo) é uma matriz que transforma o objeto do espaço local para o mundo, aplicando as rotações, translações e escalas necessárias para que o modelo seja colocado no local desejado do mundo, formando a cena em três dimensões.

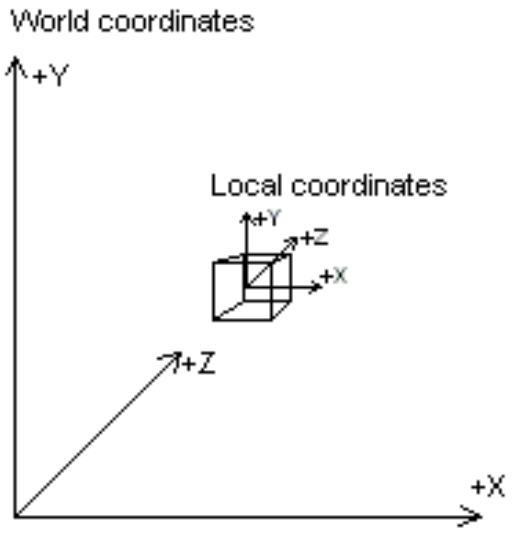

Figura 6- Transformação de Mundo

2.4.2. A Matriz de Visão (View Transform)

A matriz de visão transforma o modelo no mundo para o espaço de visão (o espaço onde a câmera se encontra na origem), possibilitando que o quadro a ser renderizado seja o que a câmera está captando no momento atual.

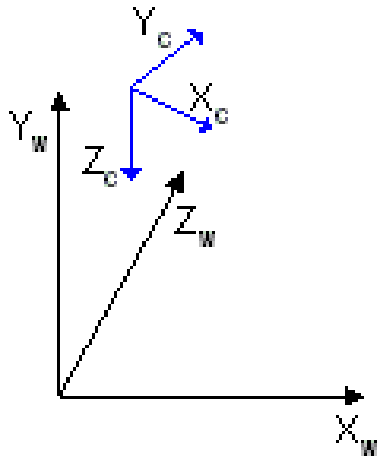

Figura 7 – A Transformação de Visão

2.4.3. A Matriz de Projeção (Projection Transform)

A matriz de projeção opera como a lente de uma câmera, tornando realística a visualização da cena, como se a mesma estivesse sendo filmada. Ela transforma o espaço prismático no mundo (Figura 8), em um espaço cubóide, conforme a necessidade do aplicativo (Figura 9). É a transformação responsável por adicionar a simulação de distância nos objetos, onde um objeto próximo à câmera parecerá maior enquanto um objeto afastado parecerá menor.

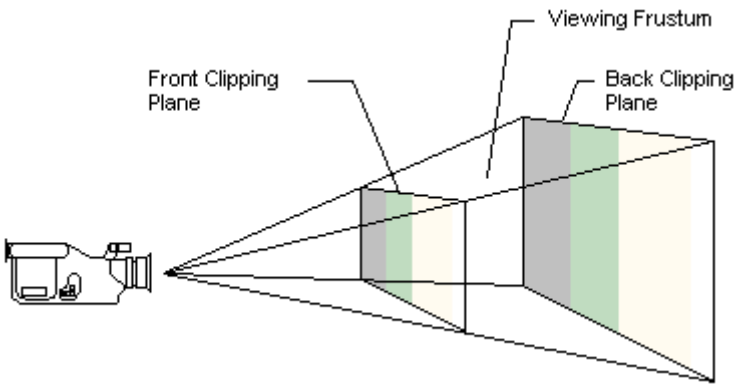

Figura 8 – O prisma que será transformado pela matriz de projeção

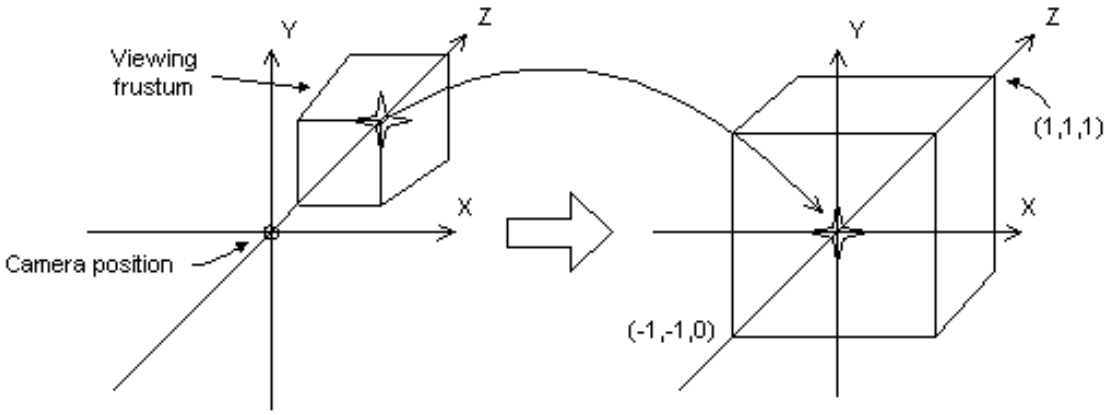

Figura 9 – Transformação do prisma em cubóide

Sabendo que:

- *Front Clipping Plane, Back Clipping Plane* – planos que limitam o espaço de visão da cena que está sendo gerada.
- *Viewing Frustum* – Campo de visão da cena, o local que será mostrado na tela quando ocorrer a renderização.

As fórmulas utilizadas para calcular a matriz de projeção estão descritas abaixo:

$$\mathrm{Pr}oj = \begin{bmatrix} w & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & Q & 1 \\ 0 & 0 & -Q \cdot Z_n & 0 \end{bmatrix}$$

Equação 10

Onde:

$w = \cot\left(\frac{fov_w}{2}\right)$,

$$h = \cot\left(\frac{fov_h}{2}\right), \quad Q = \frac{Z_f}{Z_f - Z_n}$$

Sendo que fov_w é o campo de visão horizontal, fov_h é o campo de visão vertical, Z_n é a coordenada do plano de clipping frontal, e Z_f a coordenada do plano de clipping traseiro.

2.5. Iluminação

Na natureza a luz irradia de uma fonte, é refletida milhares de vezes pelos objetos que se encontram no mundo, alterando sua cor, até chegar aos olhos de um observador. Em um ambiente de tempo real, esta operação consumiria uma quantidade enorme de recursos, e não atenderia ao requisito de tempo. Então no *DIRECTX* a iluminação tem um modelo melhor voltado para o desempenho.

A iluminação no *DIRECTX* é realizada através de luz ambiente e luzes direcionais. A luz ambiente é uma luz que é irradiada na cena como se fosse luz solar (não tem uma fonte nem direção definidas), ela vem de todas as direções e incide sobre todos os objetos presentes no mundo. A luz direcional, como o nome diz, irradia apenas em uma direção, e ilumina faces que estão na frente da fonte de luz. Para determinação da iluminação de cada objeto com relação a uma fonte de luz direcional é utilizado o produto escalar entre o vetor direcional da fonte luminosa em questão, e o vetor normal de cada vértice componente, determinando assim matematicamente se o vértice está na frente ou atrás desta. A cor da luz é combinada algebricamente com a cor do vértice, assim o objeto reflete uma luz com cor diferente da incidente, e proporciona uma riqueza de detalhes sem grande necessidade de consumo recursos. As seções seguintes descrevem os tipos de luz levados em consideração no modelo do *DIRECTX*.

2.5.1. Luz Ambiente

A luz ambiente proporciona iluminação constante na cena, iluminando todos os objetos de maneira igual. É o tipo de luz de cálculo mais rápida, já que não necessita de operações matemáticas complexas, nem dependentes das normais dos vértices, direção da luz, posição ou atenuação. Para seu cálculo apenas multiplicamos a cor ambiente do vértice pela luz ambiente do mundo.

$$Ambiente = C_a \cdot G_a$$

Equação 11

Para a cor do objeto cinza, e uma luz ambiente vermelha (Figura 10), o resultado de aplicação encontra-se na Figura 11.

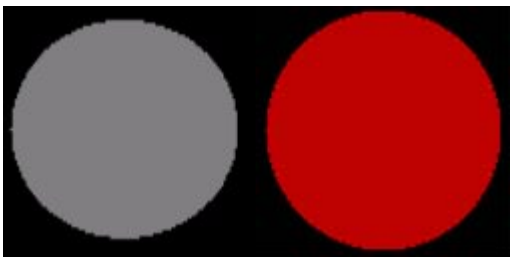

Figura 10 – Cor do Material (vértices) e da luz aplicada

Figura 11 – Resultado da aplicação da Luz ambiente

2.5.2. Luz Difusa

Após o cálculo de atenuação da luz devido à distância e incidência nos demais objetos, o *engine* determina a quantidade de luz que é refletida pelo vértice, dado o ângulo entre o vetor normal do mesmo e a direção da luz incidente conforme a Equação 12.

$$Difusa = \sum \left(C_D \cdot L_D \cdot (N \cdot L) \cdot Atten \cdot Spot \right)$$

Equação 12

Sendo que:

C_D - Cor difusa do vértice.

L_D - Cor difusa da luz.

N - Normal do vértice.

L - Vetor direção do objeto para a luz.

Atten - Fator de atenuação da luz com a distância.

Spot - Fator de spot da luz, o seu cone de iluminação.

Figura 12 – Resultado da Aplicação da Luz Difusa

2.5.3. Luz Especular

Além da luz difusa, é também necessário modelar a luz refletida pelo objeto para uma aparência realística da cena renderizada. A equação utilizada pelo DIRECTX para o cálculo da intensidade da luz especular é o modelo de Phong simplificado e um exemplo está na Figura 13.

$$Especular = C_S \cdot \sum \left(L_S \cdot (N \cdot H)^P \cdot Atten \cdot Spot \right)$$

Equação 13

Sabendo que:

C_S - Luz especular do vértice.

L_S - Componente especular da fonte de luz

N - Vetor normal do vértice.

H - Vetor de meio caminho entre os vetores da luz para o objeto, e do objeto para a câmera

P - Potência especular

Figura 13 – Aplicação de Luz Especular

2.5.4. Luz Emissiva

Além das luzes que provém das fontes luminosas externas, o objeto em si também pode ser uma fonte de luz. Temos a definição da luz emissiva, a luz que é emitida pelo próprio objeto.

A Figura 14 ilustra esta característica.

Figura 14 – Emissão de Luz do Objeto

3. Montagem do Ambiente Virtual

Este capítulo tem por finalidade descrever as etapas necessárias à montagem de um ambiente virtual

Utilizando o conhecimento do funcionamento do DIRECTX e as suas características, foi necessário escolher um equipamento para ser utilizado para o teste do trabalho, já que na verdade o processo de montagem de um ambiente virtual que realmente simule um processo de manufatura começa na escolha e levantamento de cotas dos aparelhos nela encontrados. O equipamento inicialmente selecionado foi um braço robótico da Minipa como mostra a Figura 16. Cada peça que o compõe foi desenhada em três dimensões utilizando-se um software de modelagem em 3D, e depois exportada para o formato de arquivos "***.X***", que é o formato de arquivos intrínseco do Direct 3D, o resultado pode ser visto nas figuras Figura 15 e Figura 16.

Com as peças todas disponíveis são necessárias definir as regras dinâmicas do processo, os motores, acionadores, e sensores para que a simulação ocorra da forma mais correta possível. Estes passos estão descritos nas seções que seguem.

Figura 15 – Vista Lateral do Modelo Virtual

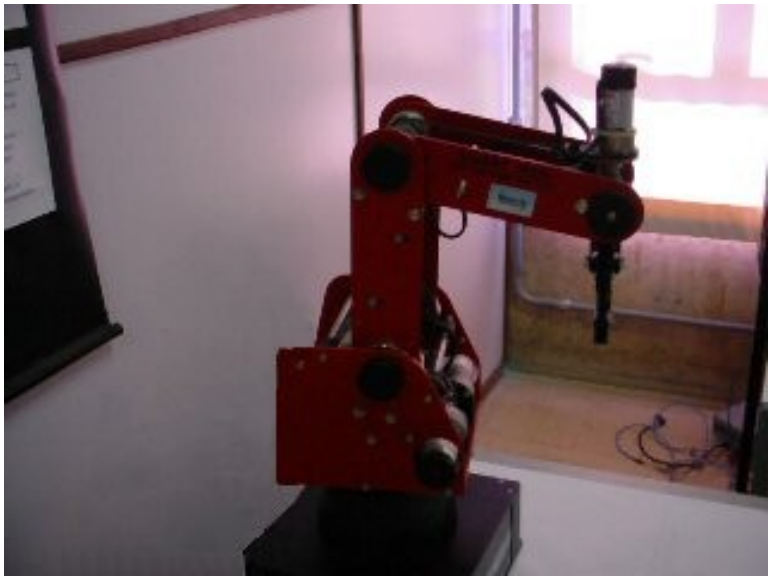

Figura 16 – Vista lateral do Robô Real

3.1. Módulo de Composição

O módulo de composição é responsável pela montagem da linha de manufatura, é o módulo que dita todo o funcionamento dinâmico das máquinas.

Agora além de apenas um desenho, é necessário o estudo das interações entre as peças que compõem a máquina, o sentido de rotação dos motores, os pontos onde estes estão agindo, possíveis pistões que operem os sensores de fim de curso, encoders, e qualquer outra característica que componha o funcionamento do processo. Desta forma para cada equipamento é feita uma análise de movimentação e cada parte identificada como sendo portadora de uma dinâmica independente é exportada em um arquivo separadamente (como objetos em três dimensões).

Estes arquivos são então importados dentro do módulo de composição para que sejam definidos os motores (características dinâmicas da máquina). Cada motor será representado por uma transformação, como mostrado no capítulo 2, representada por uma matriz 4x4 M_{ri}, que será aplicada aos elementos a ele associados na forma de uma árvore de ações. Esta árvore representa uma estrutura hierárquica de movimentação do modelo da simulação. Para entender o porquê de um comportamento hierárquico basta imaginar o modelo de um bloco sendo colocado em uma esteira: quando a esteira se move, ela leva consigo o bloco. Na representação utilizando o nosso modelo em árvore, o bloco estaria representado por um nó que seria o filho do nó que representa a esteira. Desta forma qualquer movimentação que afete a esteira está conseqüentemente afetando o bloco.

Então a árvore é construída de modo que as transformações dos nós que são pais são também aplicadas aos nós filhos, utilizando a propriedade da concatenação das transformações. Esta abordagem nos fornece um tipo de movimentação denominado esqueletal, que é o necessário para descrever praticamente qualquer máquina de manufatura.

Após a construção da árvore dinâmica, cabe também ao módulo de composição a definição dos sensores que serão acoplados ao sistema virtual, estes sensores são responsáveis por determinar quais as colisões que devem ser processadas pela parte física. Esta característica fornece uma economia vital de processamento, já que evita operações desnecessárias ao longo da simulação, poupando um tempo valioso para que o sistema de interação física seja executado livremente.

Com a definição da dinâmica e dos sensores do processo, o módulo compositor irá gerar um arquivo que contém todas estas informações hierárquicas com a extensão “XML” de forma que o mesmo possa ser importado pelo módulo de simulação, onde efetivamente as regras entrarão em operação.

A Figura 17 mostra um resumo do processo descrito, onde após o elemento ser modelado, seus componentes são exportados separadamente e depois importados para o módulo compositor associando-os a motores. A construção da árvore será mais bem descrita a seguir.

Figura 17- Processo de Modelagem da Manufatura.

3.2. Exemplo de Montagem: A montagem do Braço

Após o modelo tridimensional ser construído em um software específico, este é importado para o módulo de composição, de maneira que as operações de construção da árvore de ação possam ser efetuadas.

Logo após o modelo ser importado, a tela exibida é encontrada na Figura 18. Neste momento a árvore do modelo é mostrada no lado esquerdo da tela, onde se encontra apenas o início de qualquer modelo que se queira representar, o nó raiz. No lado direito da tela encontra-se a janela de *preview*, que proporciona uma visão do modelo que está sendo montado já com a dinâmica aplicada, ou seja, com os motores todos em funcionamento. A figura inicial que aparece na janela de *preview* são todos os elementos da máquina a ser montada, que foram carregados na memória.

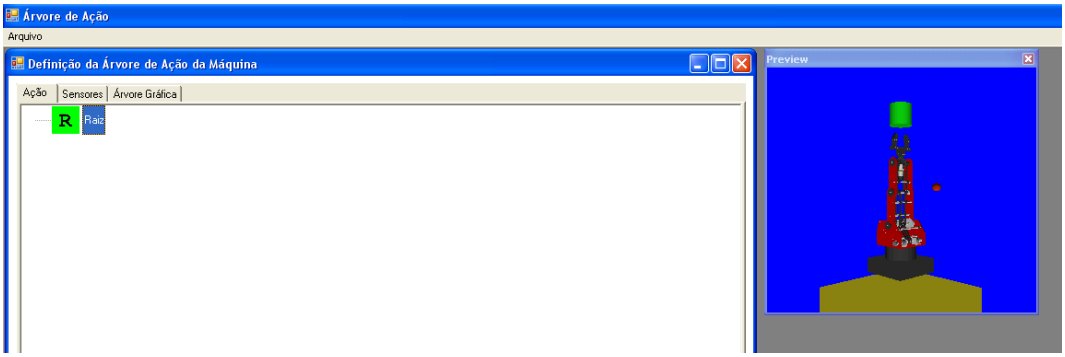

Figura 18 – Primeira tela após a importação do modelo

Neste momento são inseridos os objetos que compõem o modelo, um a um através do comando “adicionar/elementos”.

Inserindo a base, a árvore fica como na Figura 19.

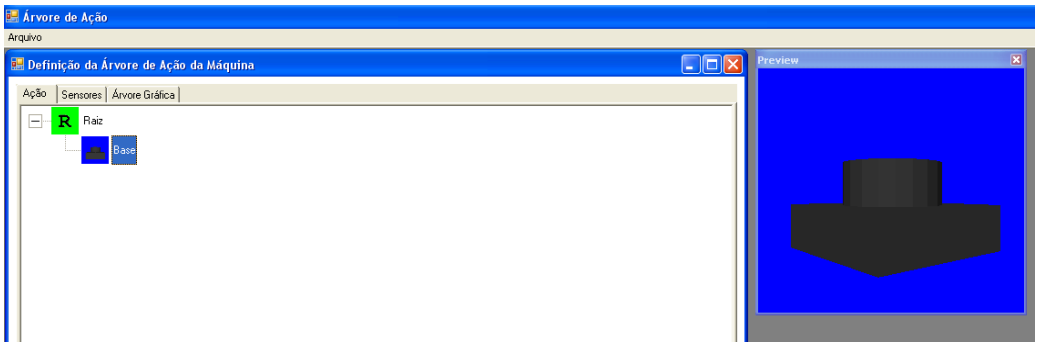

Figura 19 – Adicionando a Base

Com a base adicionada, agora chega a hora de colocarmos o primeiro elemento dinâmico no sistema, o motor que liga a bandeja à base. Assim entra-se com o comando adicionar/motor/rotação, onde se devem colocar os atributos que definem o motor (Figura 21), que são um vetor dado por (angx, angy, angz), sabendo que a rotação do motor será dada utilizando-o como eixo; o nome do motor, o ponto de aplicação do mesmo do objeto, e as teclas que vão comandá-lo no momento da simulação (Figura 20).

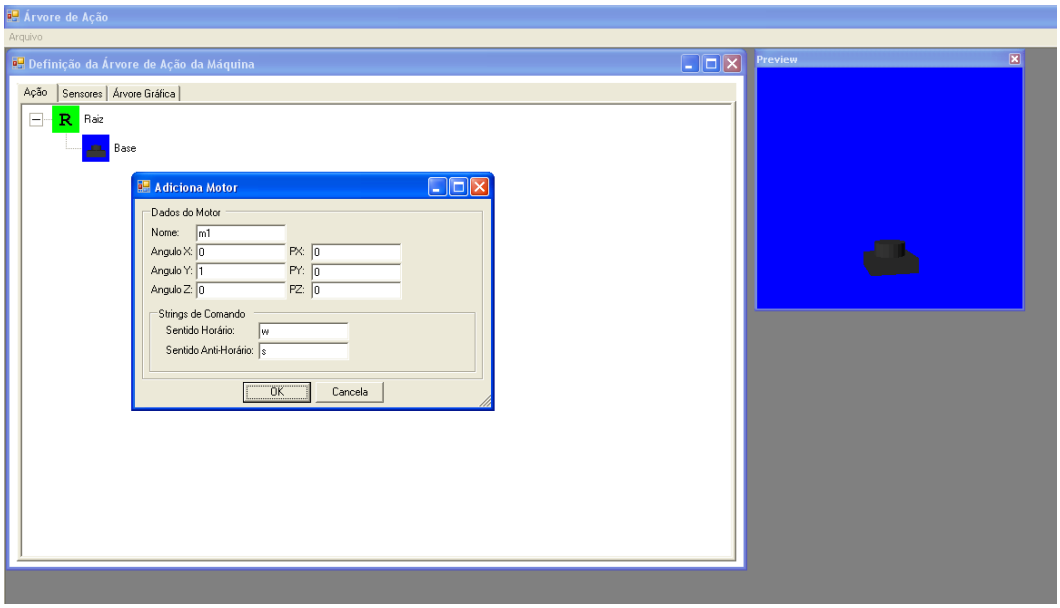

Figura 20 – Inserindo o Motor M1

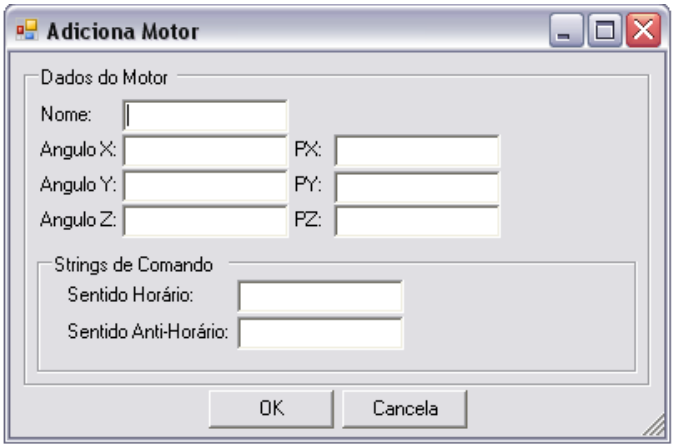

Figura 21 – Detalhe da Tela “Adiciona Motor”

Após a adição do motor podemos colocar os elementos que são superiores a ele na hierarquia, que são os elementos bandeja, bandeja_ba e bandeja_bb. Assim a visualização fica como ilustrado na Figura 22.

Estes procedimentos são repetidos até termos completa a árvore do modelo, com os objetos e a dinâmica que formam a máquina virtual.

Figura 22 – Inseridos os elementos acima da base

A árvore da máquina completa é mostrada na Figura 23. Esta árvore também pode ser visualizada na forma de árvore gráfica como na Figura 24.

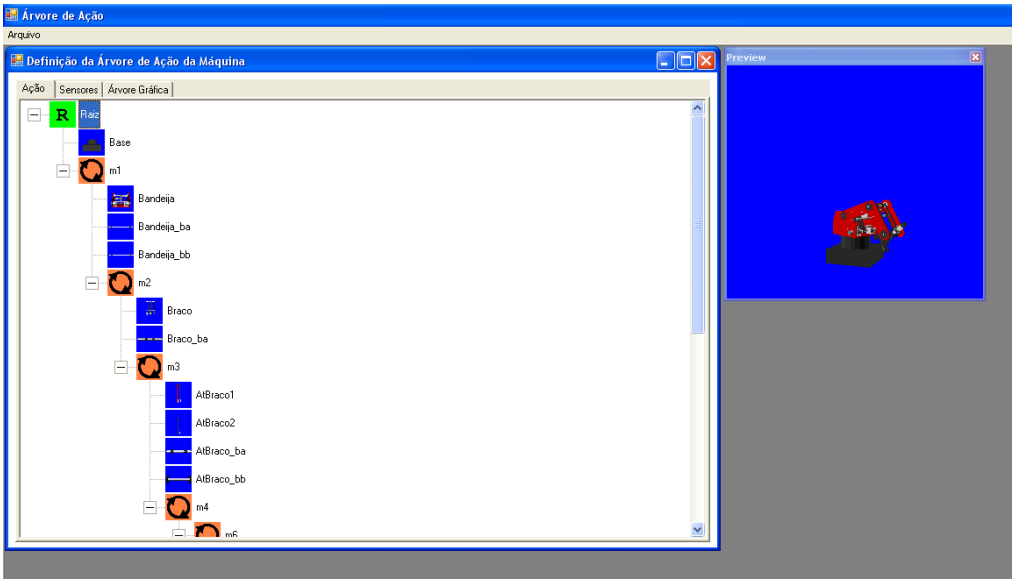

Figura 23 – Árvore Completa do Braço Robótico (visualização normal)

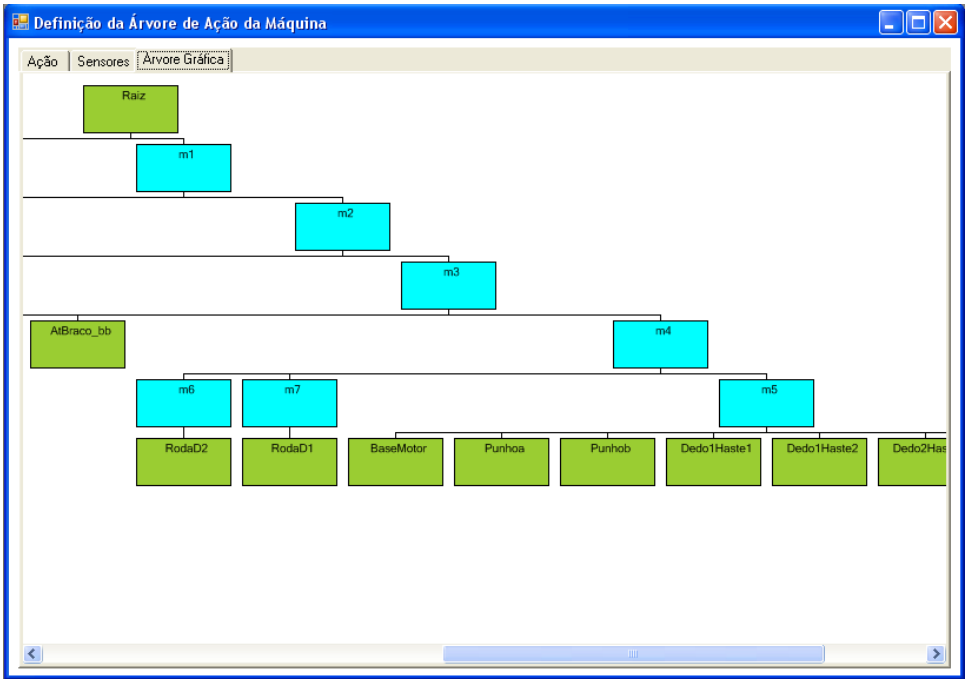

Figura 24 – Visualização Gráfica da árvore de movimentação do braço

Após a composição dinâmica do modelo, agora é necessário inserir os sensores que servirão para a detecção de colisão entre os componentes. Deve-se clicar na aba sensores, e adicioná-los (Figura 26). Os sensores sempre se referem a dois objetos, com regras da forma:

Se **A** toca **B** ou **C** toca **D** então dispare.

De maneira que **A**, **B**, **C** e **D** são quaisquer elementos que foram inseridos na árvore de atuação da máquina. No caso em questão queremos que um sensor nos forneça a informação de quando o robô segurar uma caixa. Inserimos então a regra:

Sensor S1: Se **Dedo1** toca **Box2** ou **Dedo2** toca **Box2**

Sabendo que **Dedo1**, **Dedo2** e **Box2** são elementos que estão representados na árvore de movimentação. No módulo compositor a regra fica como o ilustrado na Figura 25.

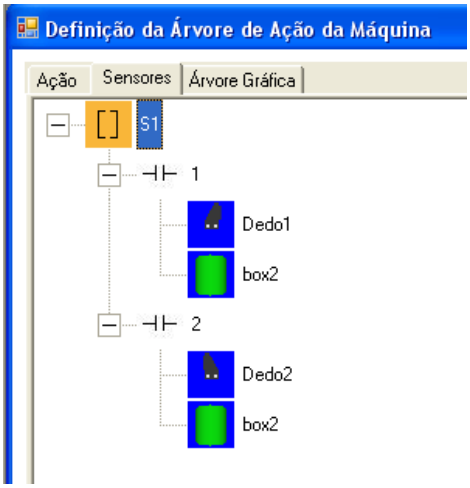

Figura 25 – Definição do Sensor S1

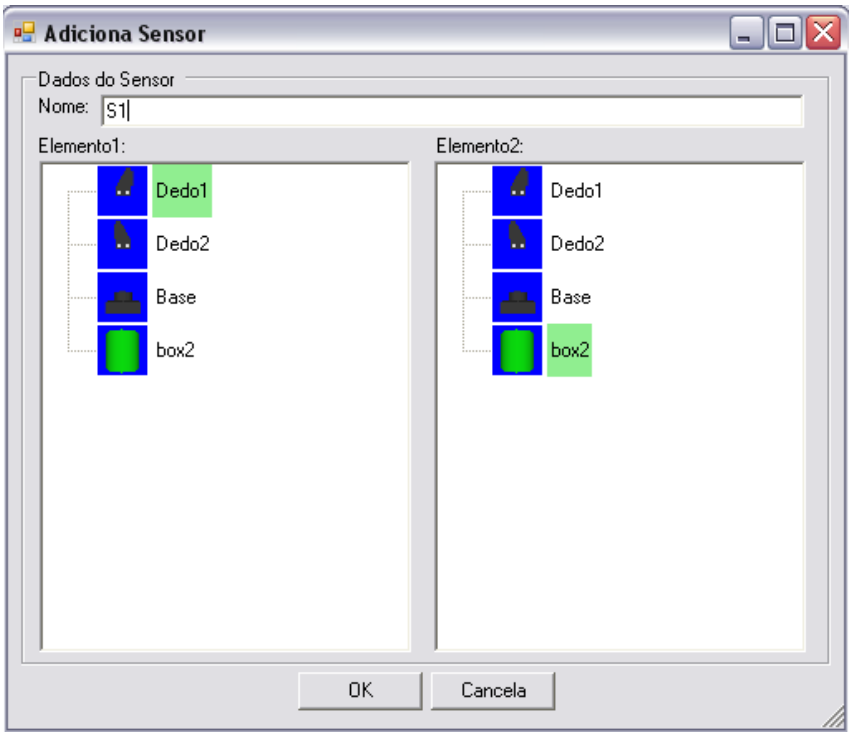

Figura 26 – Detalhe da Tela de Adição de Sensores

3.3. Comunicação com o Módulo de Programação e Execução

O Módulo de Programação permite a elaboração de programas em linguagens padronizadas pela Norma IEC61131-3 [3]. É um suporte para o desenvolvimento de lógicas de controle utilizadas em automação industrial, onde são confeccionados programas para controlar os processos da Manufatura Virtual. O módulo funciona em duas etapas: primeiro o carrega o arquivo de configuração gerado pelo módulo de composição. Este arquivo vai fornecer os motores e sensores aos quais ele terá acesso durante a etapa de execução. Depois de concluída a programação, o sistema entra em modo de execução em tempo real, ou seja, uma comunicação entre os módulos via TCP/IP é estabelecida e os estados das entradas e saídas (E/S) são atualizados bidirecionalmente.

Para que esta comunicação seja realizada, o módulo de simulação em tempo real possui um serviço executado em protocolo de rede que disponibiliza os dados dos sensores para a execução do programa. A arquitetura utilizada é mostrada na Figura 27.

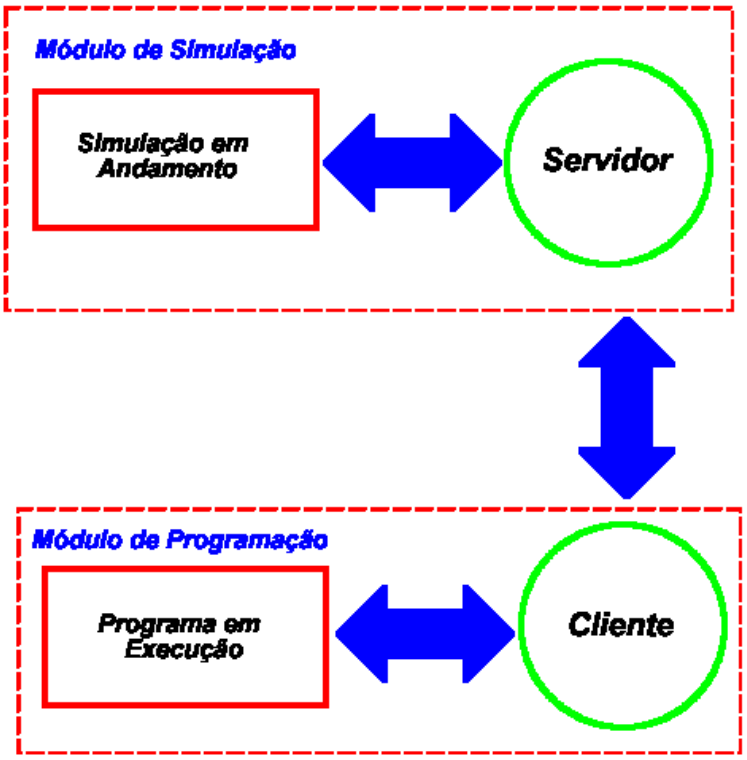

Figura 27 – Comunicação entre os Módulos

Como a comunicação é realizada através de serviços TCP/IP, os módulos de programação e de simulação em tempo real podem ser executados em máquinas diferentes, desde que haja uma ligação em rede entre os mesmos.

Os passos do processo de comunicação em tempo real são os seguintes:

- Ao ser executado, o módulo de simulação carrega a máquina conforme definido pelo módulo compositor. Com a simulação em execução é criado um objeto servidor de sensores que cria na memória uma tabela do estado dos sensores na maquete virtual;
- Com a tabela de sensores criada, o módulo de simulação abre uma conexão como servidor TCP na porta 8080, onde fica constantemente escutando a espera de clientes.
- O módulo de programação então pode se conectar ao módulo de simulação, utilizando o endereço IP do micro computador que o está executando, e é registrado como um cliente da maquete virtual.

A partir desse momento, a comunicação já pode funcionar de maneira bidirecional, o módulo de programação lê os estados dos sensores e encoders da maquete, realiza a sua varredura onde é executado um programa para controle deste, e envia à maquete o resultado das operações, ou seja, os valores de saída para que estes possam ser atualizados em tempo real.

Está construído dessa forma um processo de manufatura independente de execução em um computador local, podendo ser executado até mesmo via internet, completamente virtual.

3.4. Módulo de simulação em Tempo Real

Este módulo é o responsável pela implementação da dinâmica do ambiente através da simulação da realidade. Sua precisão e rapidez nas respostas são vitais para transmitir ao usuário uma boa sensação de realidade. Para tanto, duas características devem ser executadas da forma mais correta possível: os testes de colisão e resposta à colisão envolvendo a física dinâmica que rege os corpos envolvidos. Estas operações são as maiores consumidoras de recursos computacionais em um ambiente de simulação, devendo ser tratadas com o maior desempenho possível, para que reste tempo para que sejam executadas as outras operações da simulação, como processamento de entradas, saídas e renderização.

O correto tratamento das colisões e do processamento físico são a chave para uma simulação virtual de qualidade realística onde o utilizador realmente se sinta imerso no ambiente virtual.

Este módulo opera em ciclos, determinando primeiramente se ocorreram colisões entre dois objetos, a determinação dos pontos envolvidos, bem como o cálculo do eixo de colisão e a energia trocada entre os corpos. Estas informações então são enviadas para o tratamento e tomadas de decisões quanto ao posicionamento futuro dos objetos nos próximos quadros de animação. Todas estas etapas estão descritas nos itens seguintes.

3.4.1. Testes de Colisão

Para uma correta representação do mundo em um ambiente virtual é necessário que se saiba quando ocorrem interações entre os corpos e processá-las para que a resposta física seja a mais correta possível, o mais próximo da realidade que se possa alcançar. Quanto maior for a precisão do processo de identificação das colisões maior é a pertinência da simulação com os acontecimentos no mundo real. Desta forma uma das etapas de maior importância para atingir um grau de realidade convincente são os testes de colisão. Estes testes são responsáveis por informar quando um corpo interage, na forma de penetração ou contato, com outro corpo durante a simulação. A maneira mais óbvia de calculo para tal situação, na tentativa de determinar se houve interação entre dois corpos **A** e **B** seria o exaustivo teste de todos os pontos do corpo **A** contra todos os pontos do corpo **B** para determinar uma possível colisão. Matematicamente este é um procedimento possível, mas do ponto de vista prático, é um processamento extremamente dispendioso em termos de tempo computacional. Na representação em três dimensões um objeto facilmente chega à casa dos milhares de vértices para ser realisticamente representado. Então o processo de vértice contra vértice teria milhões de operações de teste de pontos dentro de polígonos para ser realizados para a renderização de cada quadro de simulação. Uma simulação realística ocorre no mínimo na faixa de 24 a 30 quadros por segundo, o que restringe drasticamente as operações que podem ser realizadas a cada renderização.

Existem dois fatores que mostram a qualidade de um algoritmo de colisão: a precisão do método utilizado e a velocidade de resposta. Apesar de ambos serem importantes o fator determinante da escolha de um entre os diversos métodos é o resultado que se pretende alcançar com o sistema produzido. Geralmente um algoritmo com uma resposta de maior precisão é muito lento para ser executado em softwares de realidade virtual que tenham a necessidade de efetuar muitas outras operações em tempo real. Já em um ambiente estático ou *off-line* pode-se utilizar um algoritmo mais preciso e lento sabendo que não é necessário estar concebendo a resposta do sistema em tempo real, uma vez que o resultado da simulação poderá ser visualizado depois na forma de um filme.

Outro ponto importante que deve ser levado em conta em relação aos algoritmos de detecção de colisão é que estes têm um papel fundamental na simulação física. Uma boa precisão significa uma correta identificação do estado dos objetos no momento da

interação, produzindo assim, uma correta resposta das equações físicas que regem o ambiente, responsáveis por processar as colisões e aumentar o sentimento de realidade.

Por estas razões foram pesquisados e desenvolvidos em diversos trabalhos [8, 4, 9, 5, 6, 7] algoritmos de colisão que evitam cálculos desnecessários, realizando apenas os testes que são estritamente necessários. Estes algoritmos adotam geralmente uma hierarquia de *bounding volumes* (volumes de contorno) que são na verdade figuras mais simples que englobam os objetos, facilitando os cálculos de interpenetração. Os *bounding volumes* mais adotados, por questão de simplicidade são esferas (*bounding sphere*) e caixas (*bounding boxes*).

Esta metodologia propõe a representação de um objeto através de uma árvore hierárquica utilizando volumes geométricos (esferas, paralelepípedos, pirâmides, etc.) bem definidos cuja matemática de cálculo de interatividade seja relativamente fácil de programar. A Figura 28 exemplifica o conceito apresentado através de duas árvores hierárquicas no plano: círculos e retângulos. Em ambos os casos o primeiro nível da hierarquia é composto por apenas um volume ao qual o objeto está inscrito. Em cada novo nível o objeto é dividido em partes que devem ser inscritas por um novo volume. Desta forma, a cada nível da hierarquia, o objeto estará sempre sendo inscrito por um conjunto de volumes mais representativo, ou seja, volumes que englobam melhor a geometria do objeto. Então para que ocorra a colisão entre dois objetos, os volumes de contorno necessitam também estarem colidindo, e como as colisões entre estes é facilmente calculada, existe a simplificação dos cálculos envolvidos na determinação da colisão entre os objetos.

A estrutura de árvore utilizada neste trabalho baseia-se na Figura 29 onde a cada novo nível da hierarquia são adicionados dois filhos. Para esta topologia uma forma simples e eficaz de teste é verificar todos os elementos que estão no mesmo nível e ramificação de uma primeira árvore com todos os elementos na mesma condição de uma segunda. A cada colisão identificada entre dois nós, todos nós filhos destes das duas árvores devem ser testados. Este processo continua até que seja determinada uma colisão entre nós no último nível das duas árvores. Caso uma colisão seja detectada entre eles, os elementos estão em colisão, então nestes níveis é aplicado um teste ponto a ponto apenas com os vértices que estão delimitados pelos nós onde a colisão foi detectada.

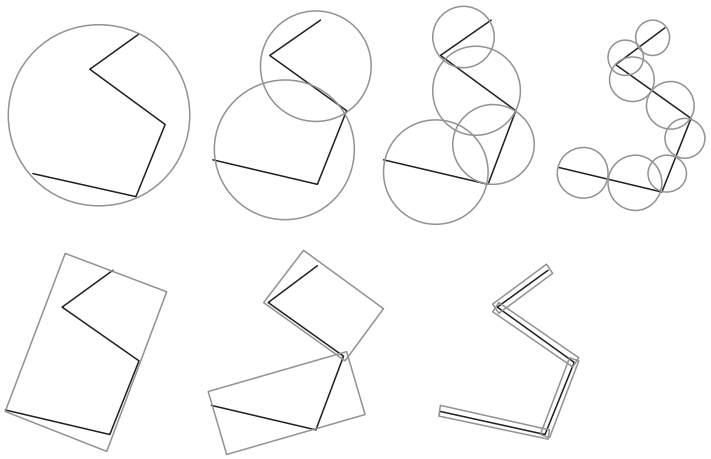

Figura 28 – Formação de uma árvore hierárquica de bounding volumes

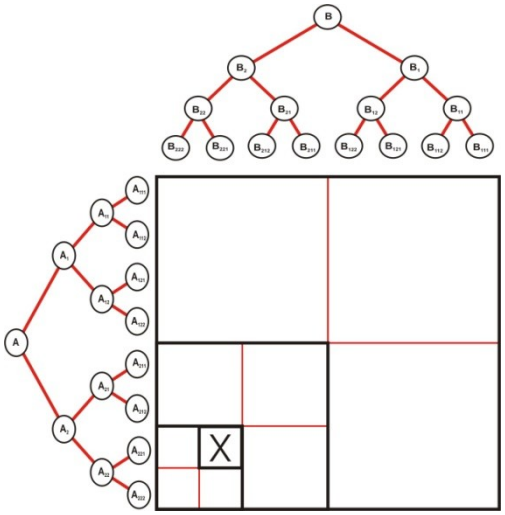

Figura 29 – Processo de Inferência de Colisão

O processo descrito para a determinação da colisão entre duas árvores de volumes de contorno é mais bem visualizado nas figuras Figura 30 e Figura 31. Na primeira figura os volumes de contorno são testados e é determinado que existe uma colisão entre eles. São testados então os filhos dos mesmos na árvore, delimitando que apenas as partes dos objetos delimitadas pelos volumes da segunda figura são passíveis de colisão. Isto reduziu a necessidade de testes mais precisos em praticamente metade dos objetos, salvando assim tempo para outras operações importantes da simulação virtual.

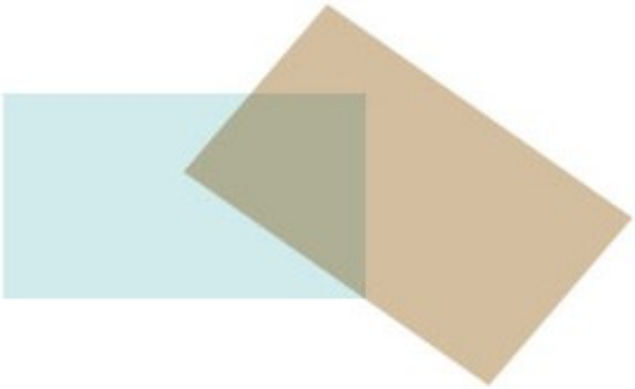

Figura 30 - Princípio de Teste de Colisão

Figura 31 - Segundo Passo no Teste de Colisão

Não existe uma geometria que apresente melhor resultado em todos os casos, o que significa que a escolha de um dado volume para ser utilizado em uma aplicação depende das formas dominantes presentes e da precisão esperada. Os métodos mais utilizados na literatura estão descritos nas seções seguintes.

3.4.2. Bounding Sphere

Este método consiste em inscrever o elemento em uma esfera. Para a construção da árvore vários algoritmos são sugeridos pela literatura [4, 5]. O teste de colisão no caso de esferas é extremamente simples: basta verificar se a distância entre os centros das duas esferas é menor que a soma dos raios de ambas, como mostrado na Equação 14 e na Figura 32.

Cada esfera é definida apenas por um centro e pelo seu raio, sabendo que as transformações que afetam o elemento que possui a esfera também afetam a esfera.

No caso da geometria dos objetos ser esférica, estes testes ficam extremamente precisos, mas no caso de geometrias diferentes, estes testes ficam muito ineficientes, já que muitos pontos serão testados a cada operação. Um ponto forte desse algoritmo é que é muito fácil de calcular uma esfera que contém o objeto, centrada na média dos vértices, e determinando o raio máximo para englobar todos os pontos que o compõem, além da velocidade do cálculo de intersecção.

$$\left\| \vec{C}_A - \vec{C}_B \right\| \leq R_A + R_B$$

Equação 14

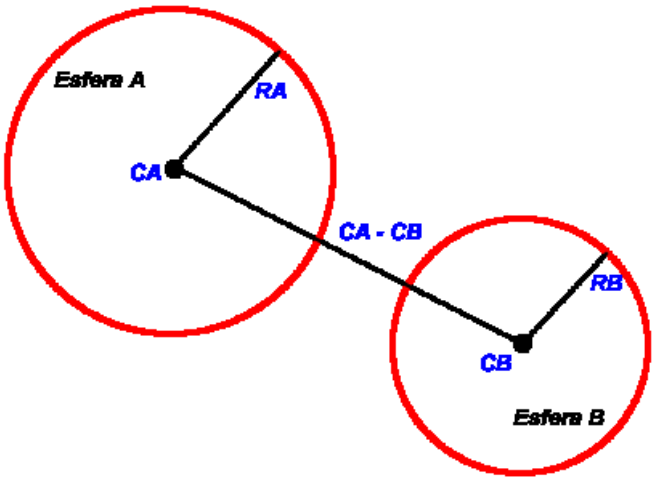

Figura 32 – Bounding Spheres

3.4.3. AABB (Axis-aligned bounding boxes)

No caso deste algoritmo, os volumes escolhidos para encapsular os objetos são caixas alinhadas com os eixos de orientação. É um bounding volume mais eficiente que o anterior no caso de objetos retangulares no modelo, mas precisa ser novamente calculado a cada quadro da imagem por não ter uma rotação específica, o que propicia uma quantidade grande de cálculos que de outra forma seriam evitados. A determinação destas caixas é extremamente fácil, novamente utilizamos a média dos vértices como centro da caixa, então suas dimensões são calculadas com base nas coordenadas mínimas e máximas dos vértices que formam o objeto. O processo de teste de colisão apenas verifica se algum dos vértices de uma das caixas está dentro do espaço delimitado pela outra, como mostrado na Figura 33.

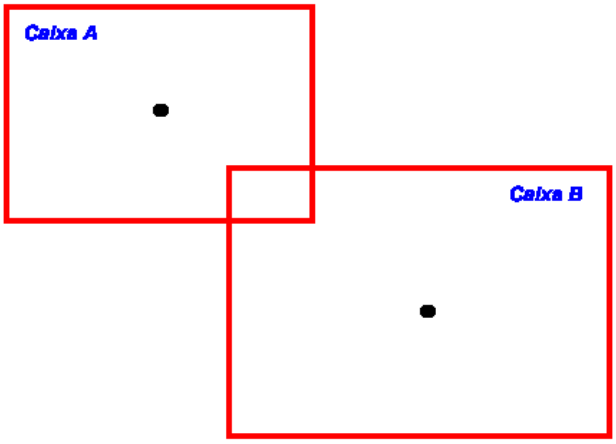

Figura 33 – Caixas Alinhadas com os eixos

3.4.4. SDM (Signed Distance Map)

Este algoritmo não descreve uma simplificação através de bounding volumes dos testes de colisão, mas sim uma otimização dos cálculos ponto a ponto, necessários no último nível das árvores de bounding volumes. Consiste em inscrever o objeto dentro de uma caixa alinhada com os eixos e dividi-la de maneira a formar uma grade em três dimensões. Cada célula desta grade contém um valor relativo à distância entre seu centro e a face do objeto que se encontra mais próximo. Para os testes de colisão, então, calcula-se a distância do ponto colidido em relação à grade. Se esta distancia for negativa, ocorreu uma colisão (o ponto de colisão encontra-se dentro do outro objeto). Como os valores das células podem ser pré-calculados, este processo causa uma grande otimização nos testes de cada vértice dos objetos.

3.4.5. Oriented Bounding Box (OBB)

É também um método que emprega caixas para aumentar o desempenho da colisão, mas neste caso a caixa possui uma orientação que descreve o seu próprio espaço. Isto causa uma melhora significativa em relação às caixas alinhadas com os eixos, nos permitindo construir uma caixa que se alinhe à geometria do objeto de maneira a realizar a menor quantidade de testes necessários para determinar uma colisão. Como mostrado por Gottschalk [7], podemos utilizar propriedades estatísticas para realizar este alinhamento, e esta abordagem foi a adotada no trabalho.

Dado um conjunto de vértices que define um corpo, o OBB é definido como a menor caixa que inscreve todos estes vértices. O cálculo das OBBs é o calculo mais complexo dos volumes de contorno aqui mostrados, consistindo nos passos que seguem.

Primeiramente é calculada a média dos vértices que compõem o objeto, segundo a Equação 15.

$M = \frac{1}{n} \cdot \sum_i v_i$, onde n é o número de vértices e v_i o i-ésimo vértice

Equação 15

A partir da média obtida, é calculada a matriz mostrada através da Equação 16, a matriz de covariância dos pontos que formam o objeto. Os autovetores desta matriz descrevem o alinhamento da geometria representada no espaço pelos vértices que a geraram, nos servindo de base para a construção do espaço da caixa. Um exemplo de caixa encontra-se na Figura 34 – Caixa Alinhada Através de Matriz de Covariância.

$$C = \frac{1}{N}\begin{bmatrix} \sum_{i=1}^{n}(x_i - x_m)^2 & \sum_{i=1}^{n}(x_i - x_m)(y_i - y_m) & \sum_{i=1}^{n}(x_i - x_m)(z_i - z_m) \\ \sum_{i=1}^{n}(x_i - x_m)(y_i - y_m) & \sum_{i=1}^{n}(y_i - y_m)^2 & \sum_{i=1}^{n}(z_i - z_m)(y_i - y_m) \\ \sum_{i=1}^{n}(x_i - x_m)(z_i - z_m) & \sum_{i=1}^{n}(z_i - z_m)(y_i - y_m) & \sum_{i=1}^{n}(z_i - z_m)^2 \end{bmatrix}$$

Equação 16

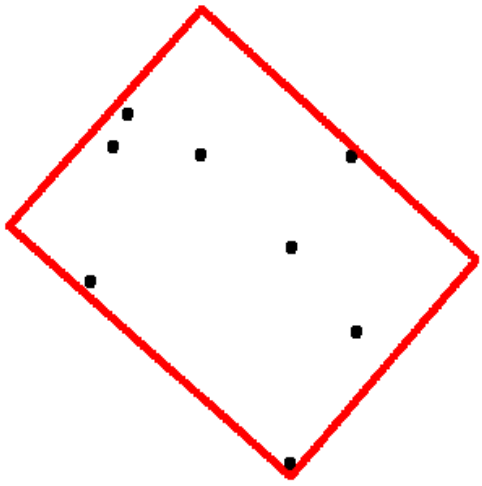

Figura 34 – Caixa Alinhada Através de Matriz de Covariância

Como visto na figura, a caixa alinha-se otimamente no objeto, reduzindo assim significativamente a quantidade de testes a que estes são submetidos durante o processamento.

A correta implementação de uma rotina de alinhamento da caixa com a geometria é o ponto chave deste método, que chama a atenção devido à sua robustez.

Uma comparação entre estes volumes de contorno encontra-se na Figura 35, onde a vantagem da OBB é facilmente visualizada.

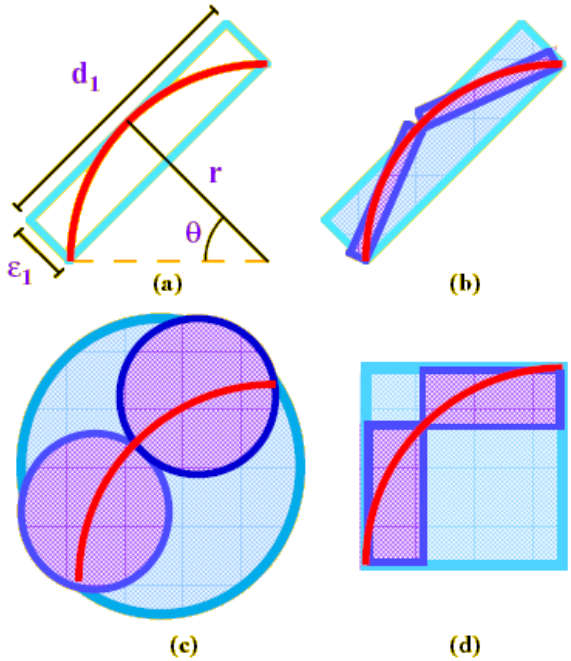

Figura 35 – Comparação entre a aproximação dos volumes de contorno à geometria de um objeto: (a) – OBB; (b) – *OBBTree* com dois níveis; (c) – *Bounding Sphere*; (d) – AABB

3.4.6. Performance dos Métodos

Utilizando agora uma cena de teste, os vários métodos citados nas seções anteriores organizados em árvores hierárquicas foram comparados com relação a seu desempenho em tempo de execução, sendo os resultados apresentados na Tabela 1.

Como a aplicação foca mais a utilização de elementos convexos, as simulações foram realizadas com blocos e retângulos cada um com 48 vértices. O resultado apresentado é referente à média de quadros por segundo após dois minutos de simulação. Para cada caso

a distância máxima entre os objetos de teste foi duas vezes o tamanho do maior objeto da cena. O nível de hierarquia utilizado em cada método está descrito na frente do nome do algoritmo. Esta diferença é necessária para se alcançar um nível desejado de precisão, determinado de forma heurística, durante o processo de simulação. As colisões não foram tratadas, apenas identificadas, sendo assim a resposta consiste apenas na detecção de colisão descartando qualquer comportamento físico de reação.

Tabela 1 - Resultado, em quadros por segundo, de diferentes métodos de teste de colisão.

Algoritmo (níveis)	*Número de Elementos*			
	2	*4*	*8*	*16*
Sphere (5)	120	105	75	30
AABB (4)	115	104	70	52
SDM(3)	113	97	74	38
OBB(2)	115	106	84	77

Levando em consideração a heurística adotada como fator de qualidade o método *OBBtree* obteve o melhor tempo de resposta sem comprometimento da precisão do resultado. Uma alteração, descrita na seção seguinte, foi proposta em relação à bibliografia original.

3.5. Teste de Colisões em Árvores OBB's

A grande vantagem da utilização de Bounding volumes é a redução nos cálculos desnecessários, como mostrado na Figura 36. Podemos através desta verificar que, se estivéssemos testando cada ponto que forma o objeto **A** contra cada ponto que forma o objeto **B** haveria um grande desperdício de tempo computacional, já que na verdade, eles nem chegam a colidir. Cálculos de colisões impossíveis de existir são muitas vezes, a causa de um desempenho extremamente pobre em um software que trabalha em tempo real.

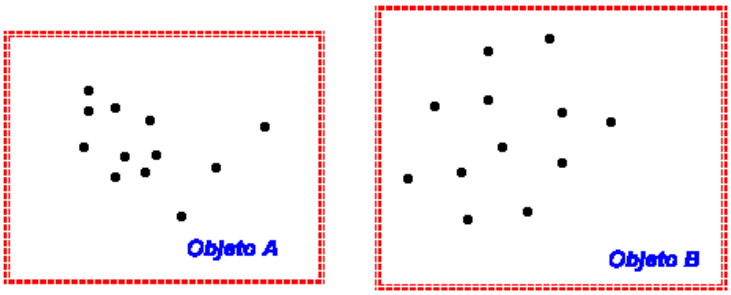

Figura 36 – Não Existência de colisão

Se neste trabalho houvessem sido utilizadas caixas alinhadas com os eixos, como mostrado acima, quando os corpos se aproximassem, inevitavelmente seria necessário o cálculo ponto a ponto para determinar a colisão, já que estes volumes de contorno não levam em consideração o melhor alinhamento com a geometria do objeto. Mas como foram utilizadas caixas alinhadas com os objetos, até mesmo uma situação como a mostrada na Figura 37 seria ignorada, não causando nenhum impacto computacional (em aplicativos gráficos que exigem muito do computador, geralmente quando os corpos se aproximam muito, existe uma sobrecarga de processamento).

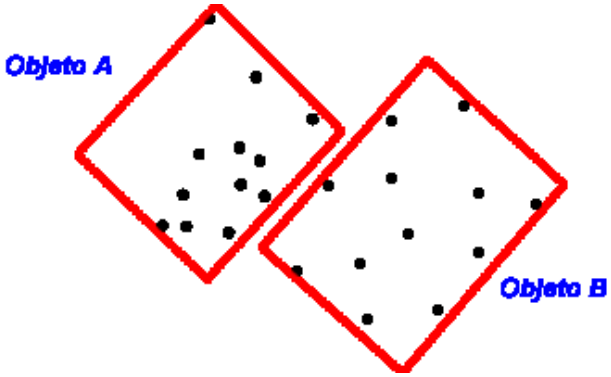

Figura 37 – Economia de cálculos com testes de colisão

Esta característica de economia de cálculo é herdada pelos nós filhos da árvore de colisão de caixas orientadas. O alinhamento geométrico da caixa depende somente dos vértices que ela contém, não tendo nenhuma relação com o nó pai que a gerou. Desta forma o alinhamento da caixa pai não necessariamente é o mesmo de uma caixa filha deste,

o que mais uma vez colabora com a redução dos cálculos relativos à colisão: uma vez que um objeto colidiu com uma caixa não implica necessariamente que ele está colidindo com algum filho desta, o que pode acarretar que, mesmo percorrendo diversos nós das árvores de colisão quando objetos estão muito próximos, podemos ter subárvores inteiras descartadas no processo, poupando recursos do sistema. O processo de inferência nas árvores de colisão ocorre como descrito abaixo.

Para determinar se houve uma colisão entre dois objetos **A** e **B**, necessitamos testar as árvores de ambos, primeiramente através do nó raiz. Se o nó raiz das árvores estiver se interceptando, tem-se que determinar se algum dos filhos do nó raiz de **A** intercepta também algum dos filhos do nó raiz de **B**. Se não houver uma interpenetração entre os filhos destes nós, a possível colisão é descartada. O processo então é repetido até que se encontrem dois nós folhas que colidem (nós que não tem filhos), somente neste caso a colisão é processada ponto a ponto, através do método ***SDM***.

3.5.1. O Teorema do Eixo de Separação(SAT)

Testar a interpenetração entre duas caixas arbitrariamente dispostas no espaço a primeira vista parece uma tarefa extremamente difícil. Mas Gottschalk [7] nos mostra um teste bastante simples para a colisão de duas caixas orientadas: o teorema do eixo de separação (SAT).

O Teorema se baseia no fato de que quando dois objetos não estão colidindo, é possível encontrar pelo menos um eixo de separação entre eles, mas não nos diz quais os eixos que devem ser testados. O trabalho citado na referência [7] nos mostra que no caso de caixas, temos 15 possíveis eixos de separação, sabendo que um eixo de separação é um eixo no qual a projeção das caixas não forma uma linha contínua.

O algoritmo apresentado em sugere o teste em 15 eixos de separação (os 3 eixos de cada objeto mais a combinação linear entre estes) e, se algum destes possuir uma descontinuidade na projeção os elementos não estão em colisão. A Figura 38 – Eixo de Separação entre os objetos mostra que quando dois objetos não colidem, é possível encontrar uma descontinuidade em sua projeção. Já quando dois objetos estão sobrepostos, não é possível encontrar um eixo de separação entre os mesmos, como mostrado na Figura 39.

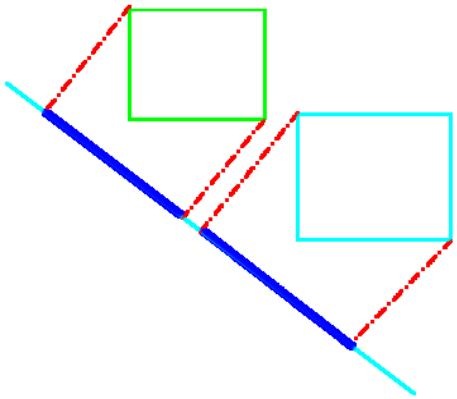

Figura 38 – Eixo de Separação entre os objetos

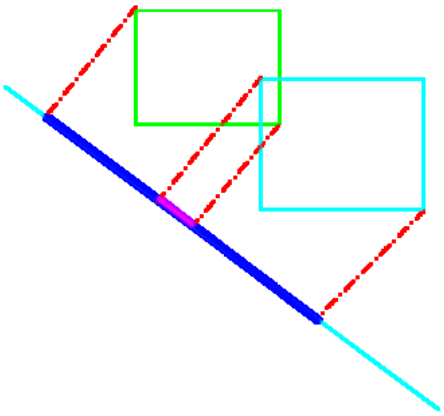

Figura 39 – Não Existência do eixo de separação

A aplicação do Teorema do Eixo de Separação em caixas orientadas consiste no teste de 15 eixos possíveis: os eixos principais de cada uma das caixas e mais os produtos vetoriais entre os eixos de uma caixa e os eixos da outra. Como pode ser visualizado na Figura 40, para dois paralelogramos existe um plano de separação em um eixo analisado se e somente se $S > R_a + R_b$. Como qualquer colisão vai necessariamente envolver uma face (ou um vértice que é o limite de duas faces), a projeção das imagens das caixas A e B sobre o eixo perpendicular à face analisada determinará se há ou não descontinuidade.

Para a execução do teste de colisão, considere dois paralelogramos A e B como da Figura 40. Sejam Ma, Mb, Ca e Cb as matrizes de transformação e os vetores que representam o centro das caixas A e B respectivamente. O algoritmo 1 descreve o procedimento.

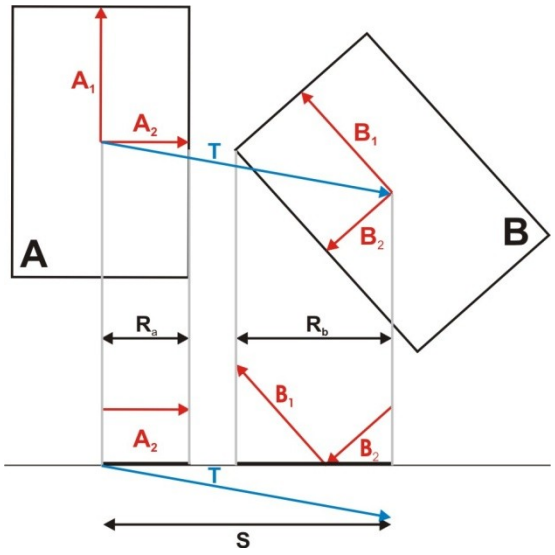

Figura 40 – Exemplo de Aplicação do Teste do Eixo de Separação

.

Algoritmo 1

1– Colocar a caixa B no sistema de coordenadas de A fazendo: Ma' = Ma^{-1}xMa e Mb' = Ma^{-1}xMb.

2– Encontrar T = Mb' x Cb – Ma' x Ca.

3– Verificar a validade do sistema de inequações abaixo

$$\begin{bmatrix} |T_x| \\ |T_y| \\ |T_z| \end{bmatrix} > \begin{bmatrix} R_{ax} \\ R_{ay} \\ R_{az} \end{bmatrix} + \begin{bmatrix} Mb'(1,1) & Mb'(1,2) & Mb'(1,2) \\ Mb'(2,1) & Mb'(2,2) & Mb'(2,3) \\ Mb'(3,1) & Mb'(3,2) & Mb'(3,3) \end{bmatrix} \times \begin{bmatrix} R_{bx} \\ R_{by} \\ R_{bz} \end{bmatrix} \quad (1)$$

4– Gerar outro caso alternando B com A nos passos 1,2 e 3.

5 – Testar o produto vetorial entre os eixos de A e B.

6– Se para os dois casos a inequação 1 for válida e houver separação também nos eixos formados pelos produtos vetoriais, então não ouve colisão.

3.5.2. Ajuste Ótimo de Alinhamento da Caixa

Outra forma de se melhorar o desempenho da simulação é reduzir o número de testes de colisão. Para tanto é necessário ajustar, da melhor forma possível, a caixa que circunscreve o objeto fazendo-a refletir fielmente a geometria associada. Quanto mais justa estiver, mais precisa é sua representação e, por conseqüência, mais eficaz fica o sistema.

O método mais eficaz mostrado pela literatura se baseia no seguinte algoritmo:

Algoritmo 2

1 – A partir dos pontos do objeto, calcular o ponto médio Pc e a matriz de covariância C segundo a equação 2.
2 – Calcular os autovetores de C. Os autovetores representam a orientação do objeto no espaço.
3 – Fazer Vx, Vy e Vz como sendo os autovetores postados em Pc.
4 – Para cada ponto Pi do objeto determinar o vetor Ui = Pi – Pc.
5 – Projetar Ui em cada vetor Vx, Vy e Vz. Os maiores e menores valores de cada projeção determinam os limites da caixa no dado eixo.

$$\frac{1}{N}\begin{bmatrix} \sum_{i=1}^{n}(x_i - x_m)^2 & \sum_{i=1}^{n}(x_i - x_m)(y_i - y_m) & \sum_{i=1}^{n}(x_i - x_m)(z_i - z_m) \\ \sum_{i=1}^{n}(x_i - x_m)(y_i - y_m) & \sum_{i=1}^{n}(y_i - y_m)^2 & \sum_{i=1}^{n}(z_i - z_m)(y_i - y_m) \\ \sum_{i=1}^{n}(x_i - x_m)(z_i - z_m) & \sum_{i=1}^{n}(z_i - z_m)(y_i - y_m) & \sum_{i=1}^{n}(z_i - z_m)^2 \end{bmatrix}$$

Equação 17

Um problema deste método é mostrado na Figura 41, onde uma distribuição maciça de pontos em uma parte do objeto impacta no resultado fornecido pela matriz de covariância e, desta forma, calcula uma caixa com o volume maior do que o necessário.

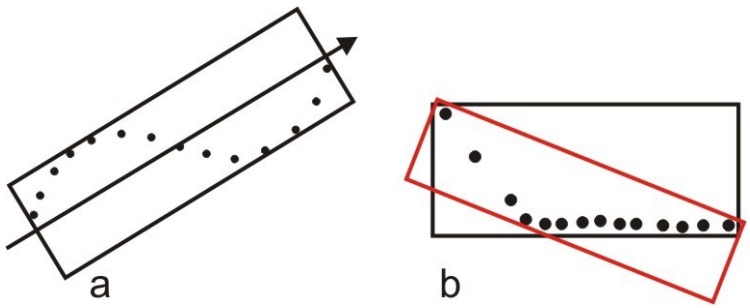

Figura 41 - (a) Alinhamento Ótimo segundo a matriz de covariância. (b) Concentração de pontos em uma parte do objeto gera uma perturbação no resultado obtido através dos autovetores da matriz de covariância

Uma forma de contornar o problema é apresentada no algoritmo 3, onde, a partir de um ponto inicial (dado pela matriz de covariância) um algoritmo de busca em profundidade é utilizado para otimizar o ajuste da caixa no objeto.

Algoritmo 3

1 – Calcular a caixa para o objeto segundo o algoritmo 2.

2 – Gerar uma árvore de busca em profundidade com 6 nós. Cada nó representa uma pequena alteração (tanto positiva quanto negativa) nos ângulos de alinhamento dos autovetores com os eixos do espaço de cerca de 1 grau.

3 – Calcular, para cada nó, a nova caixa através dos passos 3,4 e 5 do algoritmo 2.

4 – Atribuir o peso de cada nó como sendo o novo valor de volume de cada caixa.

5 – Se existir algum volume menor do que o atual, pegar o menor volume e atribuir como estado corrente e voltar ao passo 1. Caso contrário encerrar.

A tabela 2 mostra o resultado da comparação entre o método original, aqui denominado OOBB e o novo NOBB. Os critérios de simulação foram os mesmos utilizados para gerar a tabela 1. Os resultados apresentam uma taxa de desempenho do método aprimorado 27% superior ao original proposto pela literatura.

Tabela 2 - Comparação de Desempenho entre o método original e o modificado.

Algoritmo	***Número de Elementos***			
	2	***4***	***8***	***16***
NOBB(2)	138	129	103	98
OOBB(2)	115	106	84	77
Taxa	1.20	1.21	1.22	1.27

3.6. Resposta Física

Algoritmos de resposta à colisão são fundamentais para expressar de forma correta a realidade, já que objetos dinâmicos são o coração de qualquer processo de manufatura. A habilidade de simular com qualidade esta dinâmica é muito importante, mas também muito complexa. Com base nos requisitos de tempo e robustez do trabalho foi então escolhida uma abordagem que fosse capaz de tratar um grande número de corpos interagindo simultaneamente, através de atrito e empilhamento. Tendo então a resposta da colisão

encontrada, resta saber o que fazer com estes dados. Assim um modelo matemático de comportamento dos corpos mediante contato e colisão foi adotado para reger o comportamento destes ao longo da simulação.

A literatura mostra [6, 7, 10] uma série de pequenas variações na formulação física básica, cada uma enfatizando um tipo de problema. Mas cabe neste ponto ressaltar que a qualidade da resposta física vai depender também da precisão das colisões que serão processadas, já que estas são as entradas no processo de interação física, e estes são os que determinam a simulação virtual, tornando-a realística ou não.

Basicamente, as equações que descrevem o comportamento do corpo rígido (corpo que não se deforma) no tempo são:

$$\dot{x}_t = v, \quad q_t = \frac{1}{2} \cdot \omega \cdot q$$

$$\dot{v}_t = \frac{F}{m}, \; L_t = \tau$$

Equação 18

Sendo que x representa a posição, v a velocidade de deslocamento, q descreve a orientação do corpo no espaço, ω a velocidade angular, F é a força resultante, m a massa, $L = I \cdot \omega$ é o momento angular com o tensor de inércia e τ é o torque resultante. Estas variáveis descrevem o estado do corpo rígido no espaço, e são as características próprias de cada objeto que será simulado, devendo estar armazenadas no mesmo para a aplicação da dinâmica. Para a solução destas equações, devem-se integrar algumas grandezas para que outras quantidades sejam atualizadas, para esta finalidade necessitamos de um componente integrador.

Diversos métodos de integração complexos possuem uma precisão elevada, mas como estes algoritmos não detêm a precisão na presença de corpos em contato e colisão eles foram descartados. Assim, para o trabalho foi utilizado um simples integrador de Euller para encontrar as soluções das equações definidas acima.

A maneira para tratar deste problema físico, como mostrado acima, é integrar as equações no tempo e então tratar das colisões e contatos possíveis entre os corpos. Para

facilitar a abordagem do problema, as duas interações físicas possíveis são interpretadas separadamente: colisão e contato. Colisão descreve um choque entre dois corpos e contato basicamente quando um corpo desliza sobre outro, ou encontra-se em repouso

Geralmente, colisões geram impulsos e descontinuidades para modificar a velocidade, enquanto contatos são associados às forças e acelerações. Mas o uso de forças de atrito requer a utilização de impulsos também no tratamento dos contatos, mesmo que se tenha que conservar as interações impulsivas no mínimo possível. Alguns autores, citados no trabalho encontrado na referência [11] tiveram dificuldades em alcançar o equilíbrio dinâmico por conta dos agentes impulsivos, assim propuseram que as colisões fossem tratadas com suas características impulsivas, mas os contatos fossem substituídos por molas que penalizam o movimento dos corpos. Foi utilizada a magnitude da velocidade relativa para permitir tal aproximação, e diferenciar colisão (Figura 42) de contato (Figura 43).

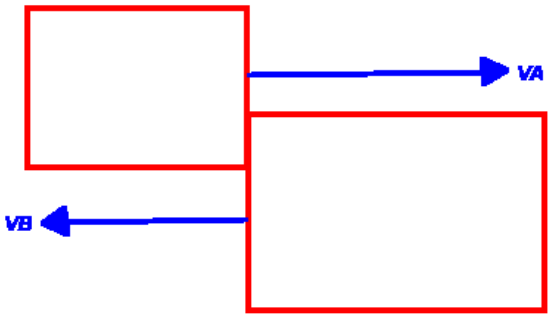

Figura 42 – Colisão entre dois corpos

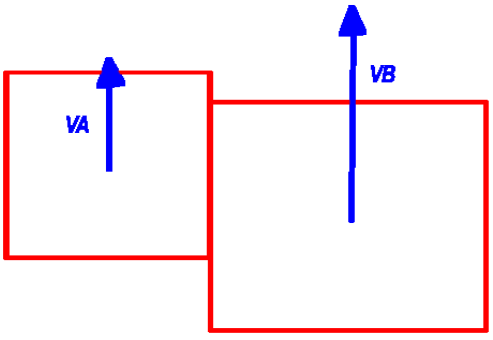

Figura 43 – Contato entre os Corpos

Mas o método adotado neste trabalho realiza a separação das colisões e contatos sem a necessidade da separação através de faixas de velocidade. A abordagem temporal do algoritmo é a seguinte:

- Detecção da colisão entre os corpos;
- Atualização das velocidades utilizando as equações acima;
- Solução das interações de contato.
- Atualização das posições

O algoritmo trata os objetos de forma que em repouso as velocidades de todos é zero. Assim no tratamento da colisão não ocorrerá o processo de deformação elástica. O próximo passo é integrar a gravidade na velocidade, e então a solução dos contatos estabiliza e retorna os objetos ao repouso, de maneira que eles fiquem corretamente parados.

A chave é que a atualização dos contatos ocorre exatamente após a atualização da velocidade com a gravidade. Se isto não ocorresse, objetos que estão em repouso seriam afetados pela resposta impulsiva gerada pelo tratamento das colisões, ou mesmo penetrariam no chão, em caso de empilhamento. Assim a atualização dos contatos é o passo correto a ser utilizado após a atualização das velocidades, sabendo que este é afetado pelas forças, e a atualização das velocidades é o passo onde as forças são incluídas na simulação dinâmica.

Mas um cuidado que se deve tomar é que o mesmo algoritmo de detecção de colisão deve ser utilizado na detecção dos contatos, para garantir que a resposta seja coerente e não haja o surgimento de uma resposta errônea, que causará erros de interpretação no processo físico através de respostas inesperadas.

Foram realizadas diversas simulações para o ajuste dos parâmetros físicos dos objetos, como elasticidade, densidade, número de passos na colisão e no contato, e como resultado, todas as simulações foram bem sucedidas.

Alguns resultados alcançados na simulação encontram-se no capítulo 4, mostrando a qualidade gráfica e a precisão da física aplicada.

4. Resultados

Este capítulo tem a finalidade de demonstrar resultados obtidos com a aplicação dos métodos citados e analisar a qualidade destes

Utilizando os algoritmos citados foi composta uma cena com o braço robótico manipulando uma pilha de cinco blocos, todos independentes. Podem-se segurar blocos, colocá-los em qualquer lugar alcançável pelo braço, interagir com a pilha na forma de colisões do braço na lateral, jogar os blocos, etc.

As figuras Figura 44, Figura 45 e Figura 46 mostram uma garra virtual manipulando um cubo de uma pilha. As figuras Figura 47, Figura 48 e Figura 49 mostram a mesma garra interagindo contra uma pilha de blocos em estado estacionário. Após a colisão os blocos se rearranjam no quadro. A velocidade média de simulação foi de 60 quadros por segundo, o cenário apresentou 28 elementos com mais de 4800 vértices ao todo.

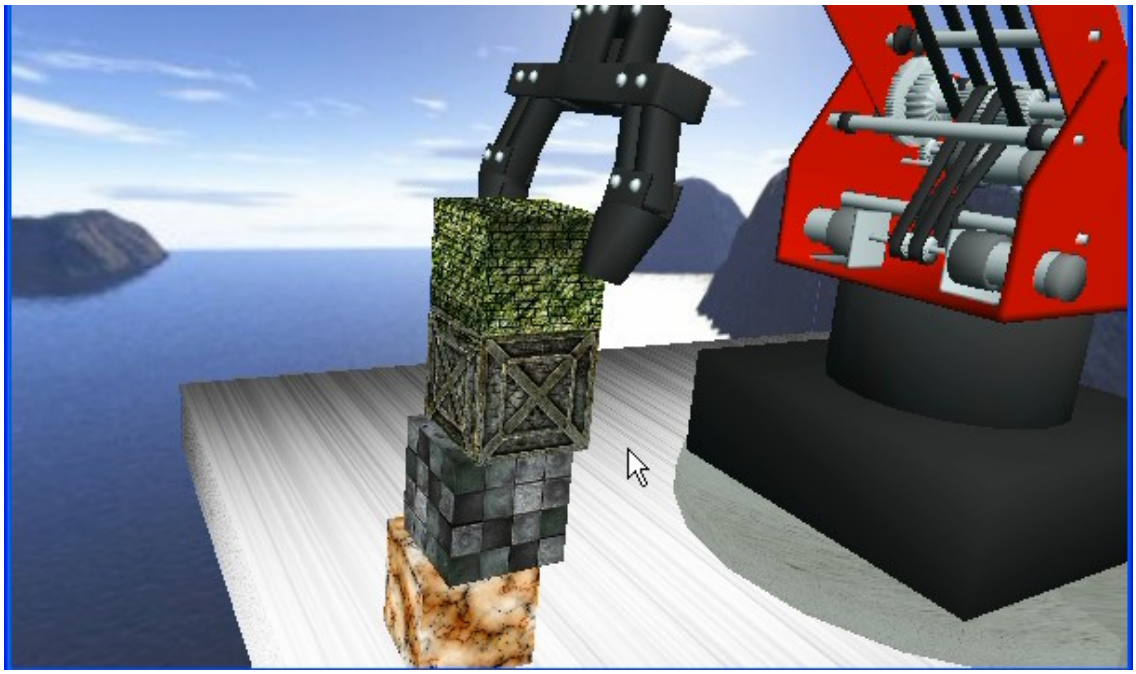

Figura 44 – Pegando o Bloco Superior da Pilha

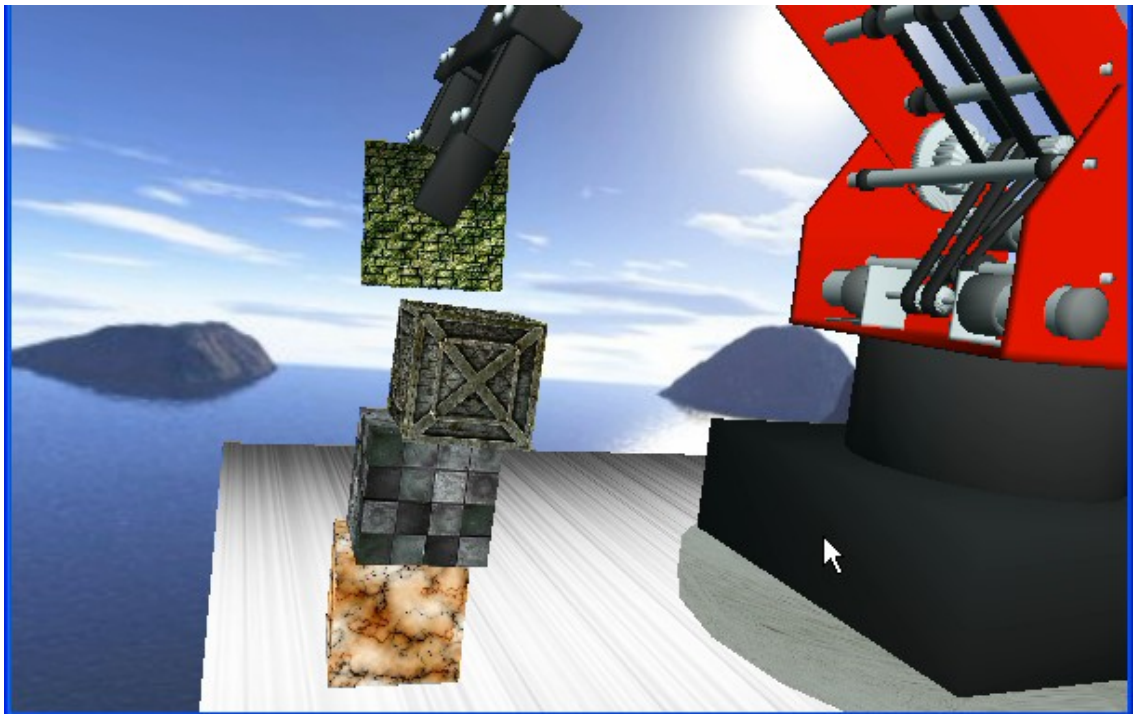

Figura 45 – Segurando o Bloco

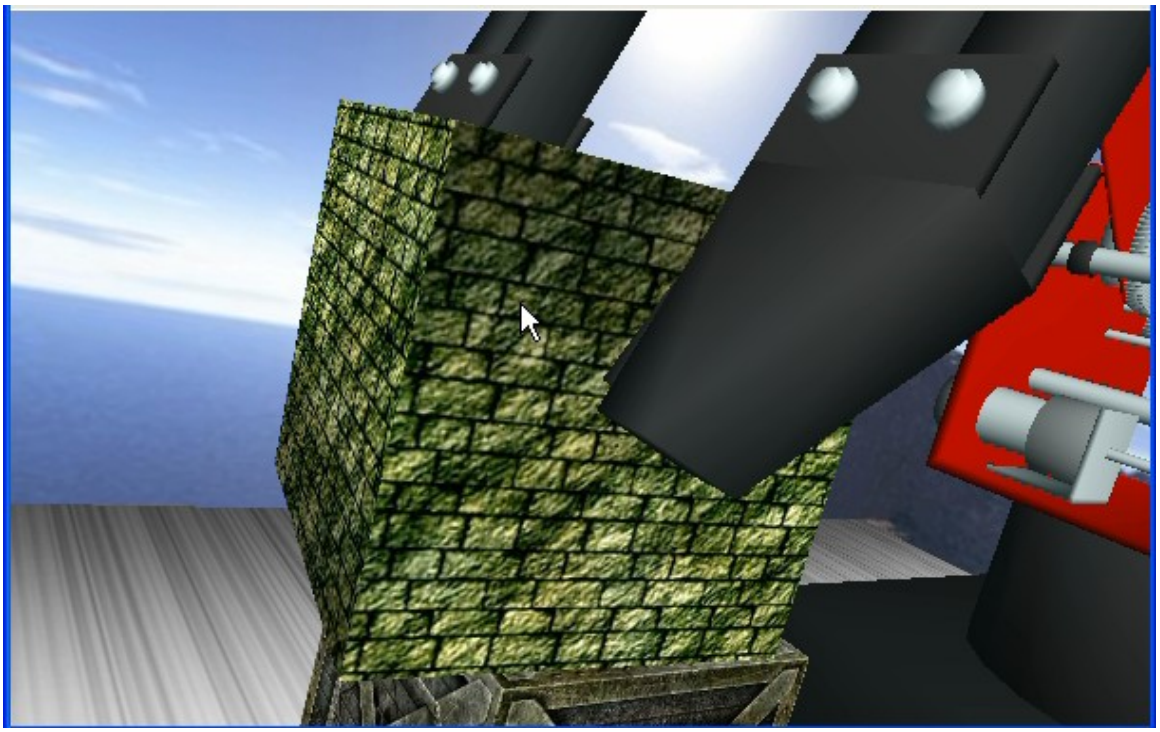

Figura 46 – Detalhe da garra no bloco

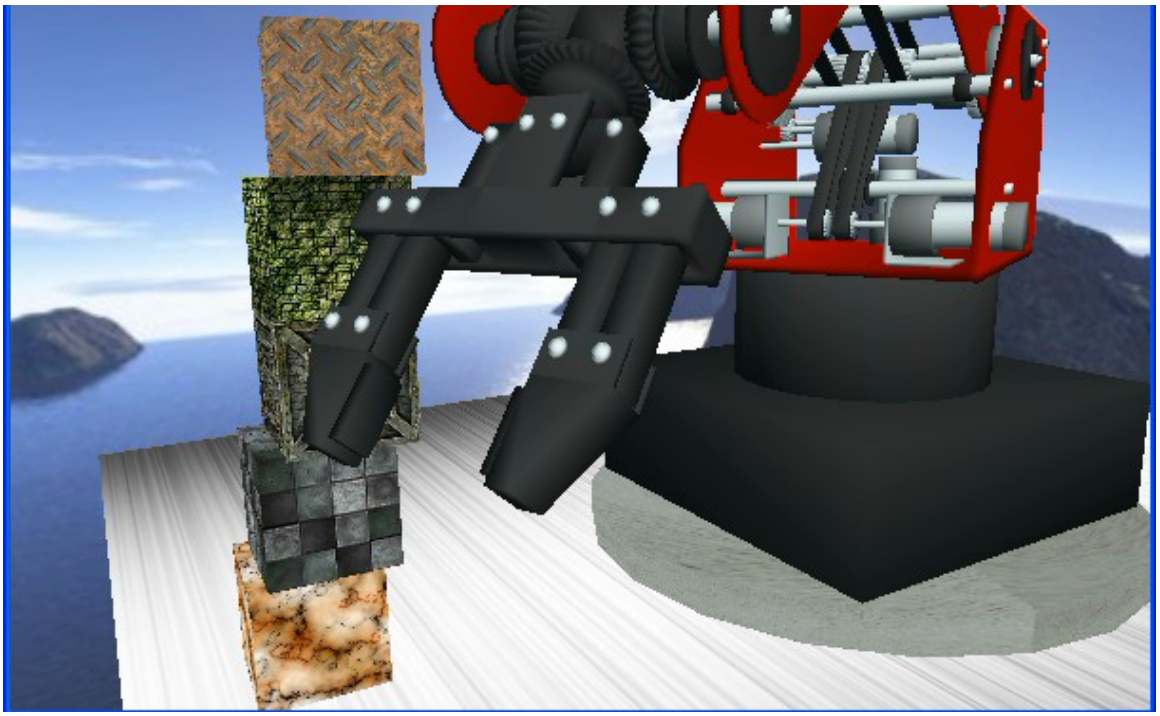

Figura 47 – Imediatamente Antes de uma colisão

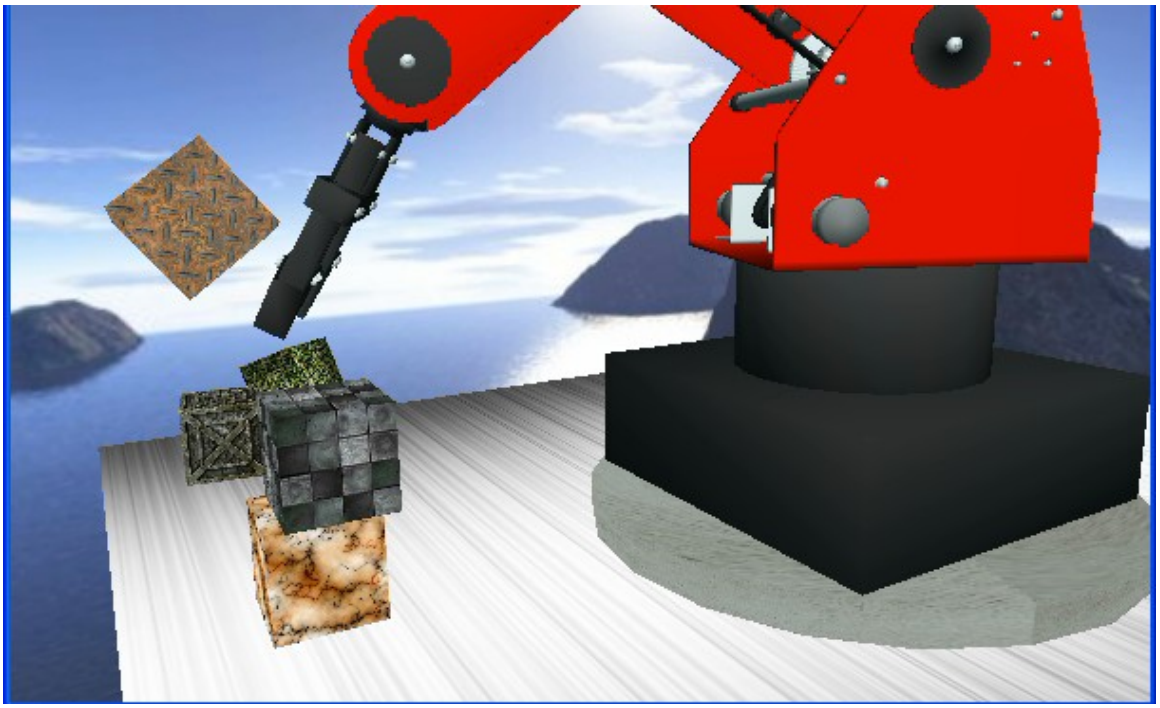

Figura 48 – Colisão do braço com a pilha de blocos

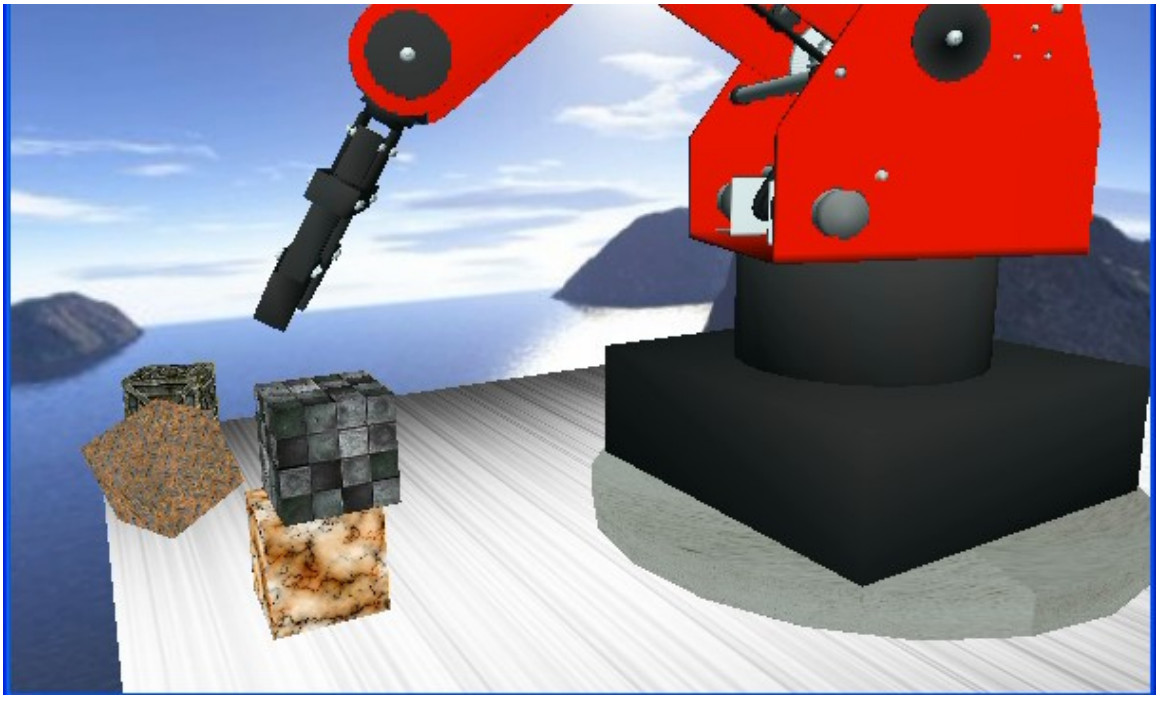

Figura 49 – Após a colisão

Estas figuras apresentam a qualidade gráfica e física da simulação realizada, mostrando a necessidade dos algoritmos aqui apresentados para que não haja a processamento de informações desnecessárias, o que causaria uma perda na velocidade de simulação. Este

mesmo ambiente foi testado também sem a utilização de nenhum método de otimização do processo de colisão mostrando a importância da aplicação deste. Ao invés da velocidade média apresentada utilizando árvores de OBBs, que foi de 60 quadros por segundo de apresentação, sem a utilização das mesmas a simulação apresentou uma taxa de atualização de um quadro a cada 10 segundos de simulação.

Outro ambiente de teste utilizado foi de uma mesa da festo que trabalha furando peças. As figuras Figura 50, Figura 51, Figura 52, Figura 53 e Figura 54 ilustram o resultado conseguido com esta mesa. Uma diferença que este ambiente tem em relação ao braço robótico, é que além de transformações de rotação, agora também são utilizadas transformações de translação para a simulação. Os resultados obtidos são extremamente rápidos (média de 50 quadros por segundo) e de ótima qualidade gráfica.

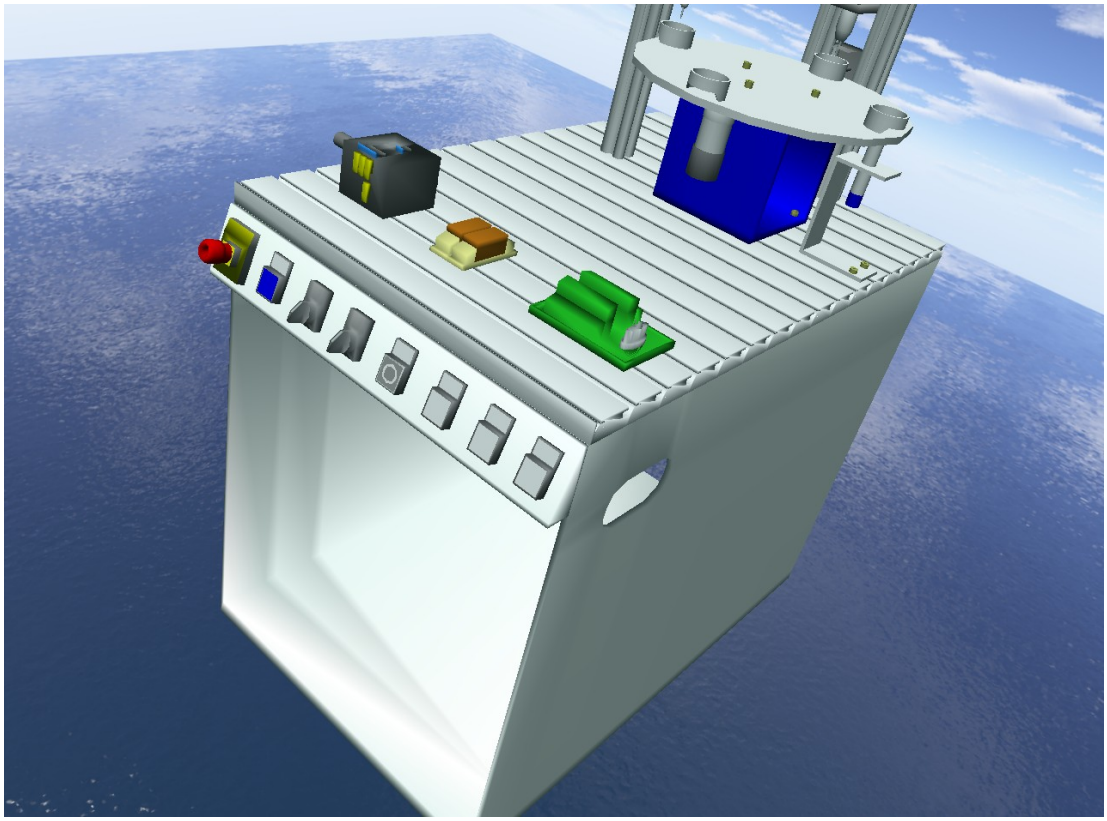

Figura 50 - Mesa Festo Simulada

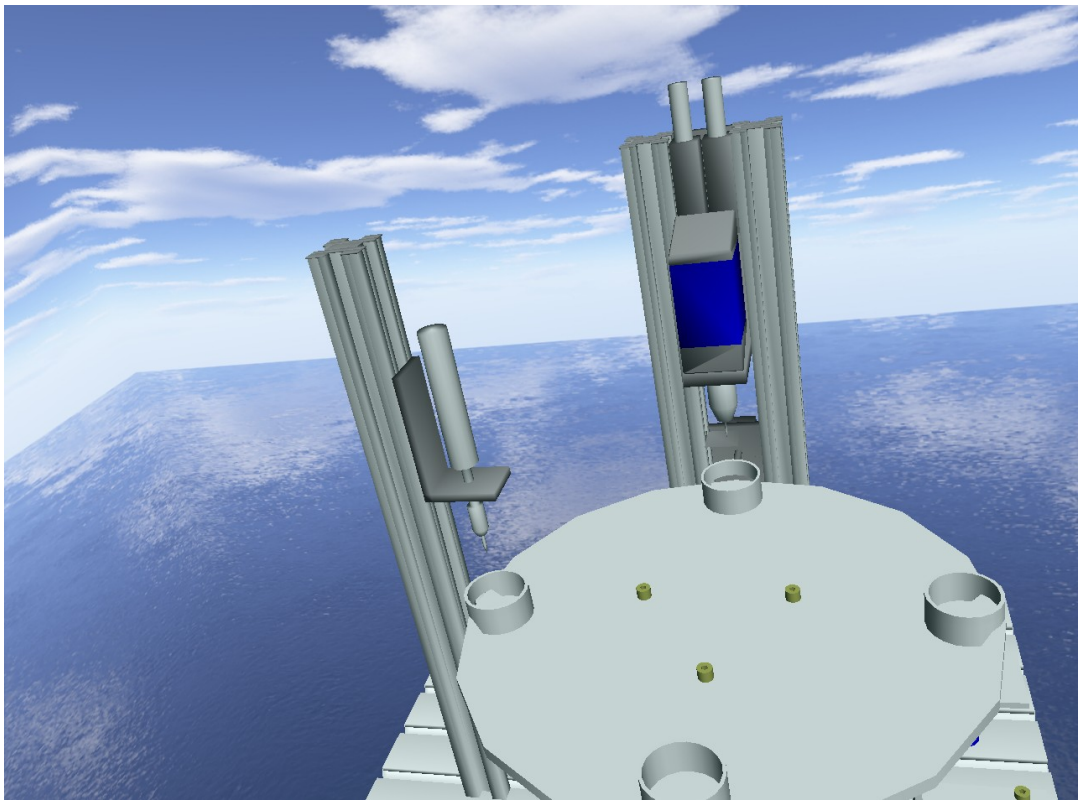

Figura 51- Mesa com o disco, o furador e o testador em posição de repouso

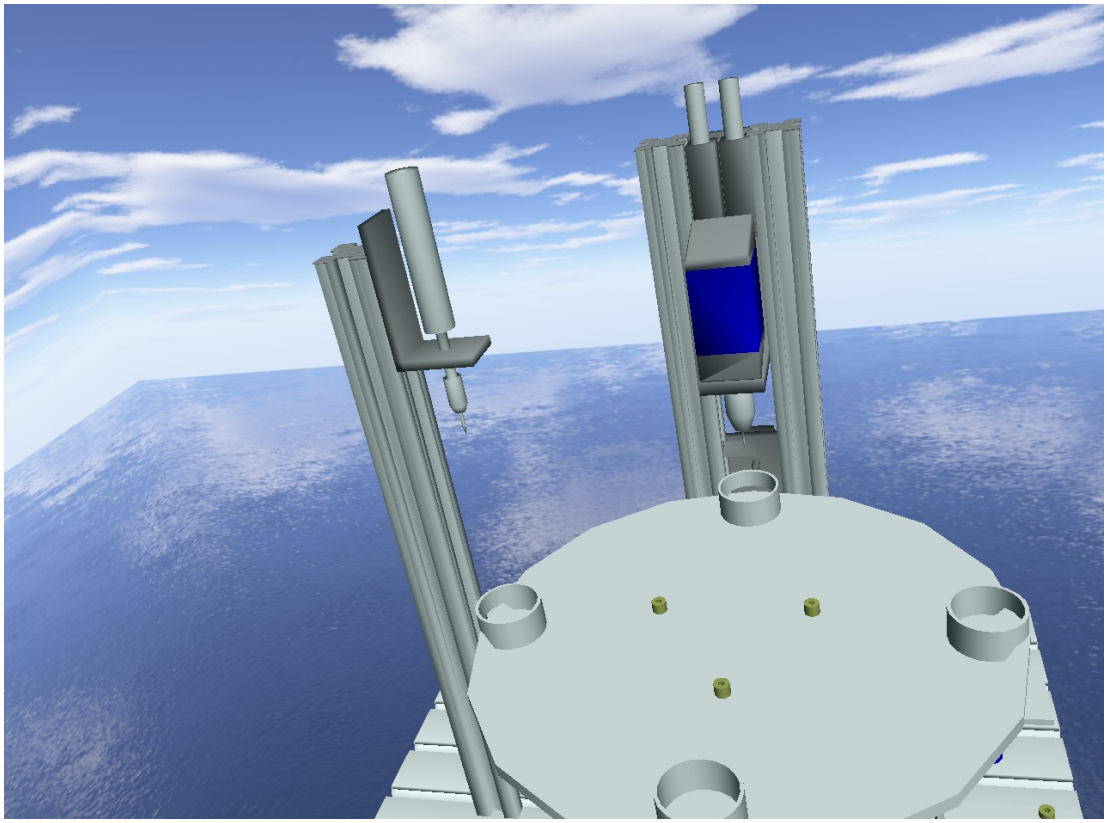

Figura 52 - Testador de furos no fim de curso superior e broca em repouso

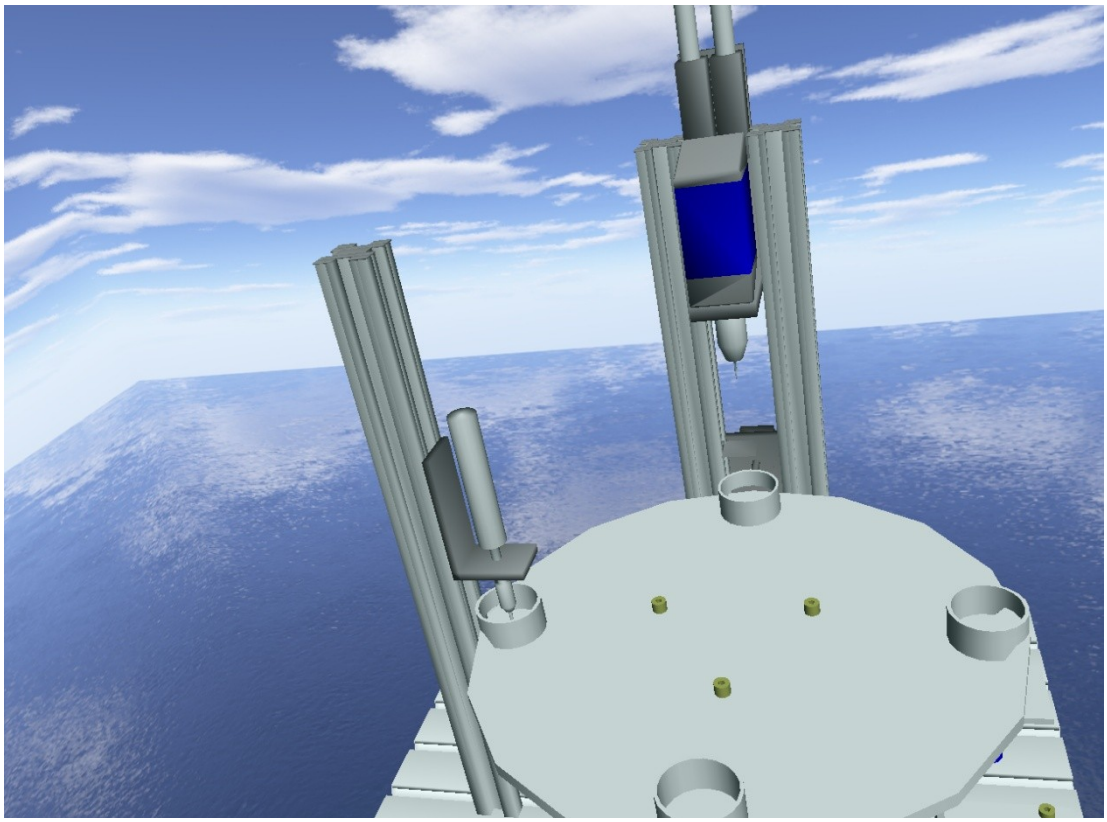

Figura 53 - Testador de Furos em atuação e broca no fim de curso superior

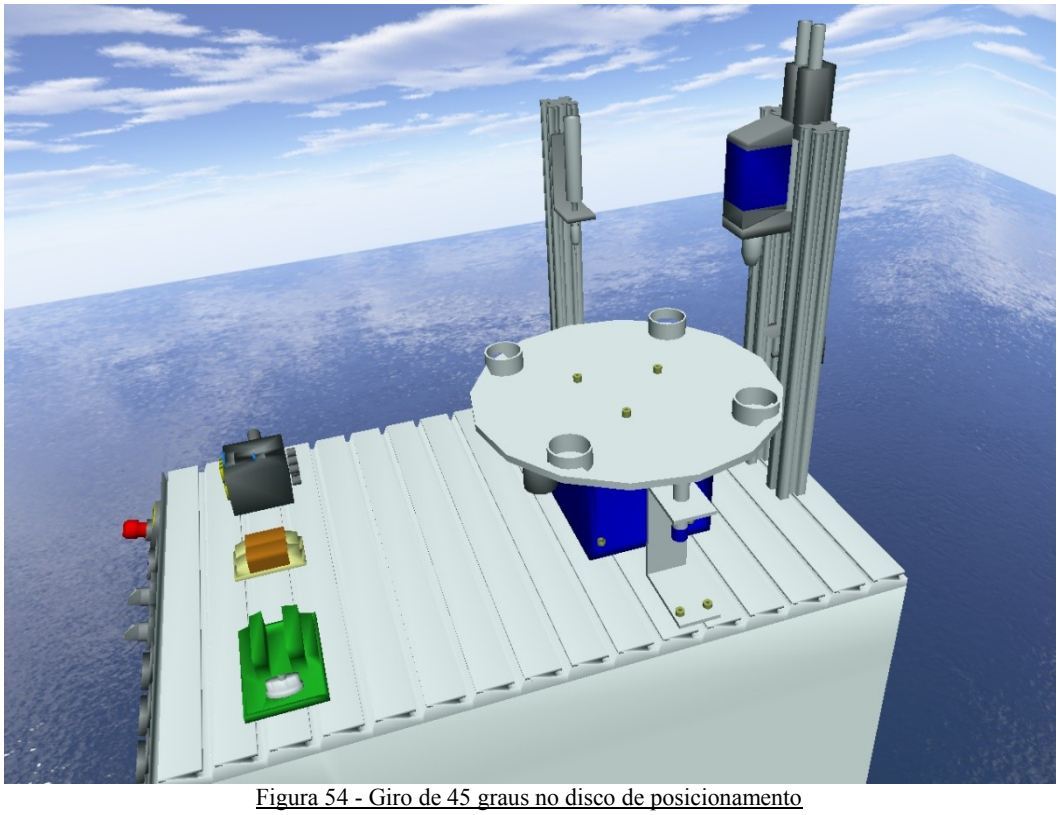

Figura 54 - Giro de 45 graus no disco de posicionamento

5. Conclusões Finais

Este Capítulo fornece as conclusões alcançadas com o trabalho.

A proposta desse trabalho foi a criação de um framework de realidade virtual para a simulação de ambientes de manufatura realísticos, de modo a suprir de uma forma interessante a necessidade de aquisição dos mesmos por parte das instituições de ensino. Foram testados e estudados diversos algoritmos de tratamento de colisões entre corpos e resposta física tratando as interações em questão. Tendo em vista a qualidade alcançada graficamente, bem como a velocidade de renderização das simulações e a possibilidade de comunicação com um módulo de programação, permitem que seja montado com este trabalho um laboratório virtual de automação industrial, de modo que alunos possam aprender o funcionamento e comando através de acionamentos e também o sensoriamento das mais diversas máquinas presentes na manufatura vendo a resposta destas a diversos eventos.

Para a decisão de qual algoritmo de colisão utilizar, todos os métodos conhecidos e de desempenho razoável foram implementados e um teste no ambiente foi realizado como mostrado na seção 3.4, e também com os métodos de tratamento físico a mesma filosofia foi aplicada. Assim a escolha dos métodos adotados para a aproximação da resposta virtual da real foi um processo exaustivo de busca, mas os resultados obtidos apresentaram uma resposta satisfatória em termos de interações entre os corpos e velocidade de simulação.

Assim foi atingida a meta de uma correta simulação virtual de sistemas de manufatura reais, utilizando métodos inovadores de programação em três dimensões, possibilitando a total imersão de um operador no mundo virtualmente simulado.

6. Propostas para Desenvolvimento Futuro

Com o exaustivo estudo realizado para a conclusão deste trabalho, foram descobertas algumas otimizações que podem ser realizadas no ambiente de forma a torná-lo ainda mais eficiente como a utilização de vários tipos de árvores de colisão, assim seria possível utilizar os volumes de contorno que mais se adequassem ao objeto, como OBBs no caso de objetos retangulares, e esferas para o caso de objetos circulares. Desta forma o processo de inferência de colisão efetuaria ainda menos cálculos sabendo que os volumes estariam quase que perfeitamente delimitando os objetos neles inscritos. O protocolo utilizado na comunicação entre os módulos pode também ser expandido, de forma a dar a possibilidade de execução de um ambiente de monitoramento e supervisão da manufatura em tempo real, concluindo dessa forma todo um pacote de simulação de manufatura em três dimensões.

Outra possibilidade é a aplicação da teoria do eixo de separação para o treinamento de redes neurais supervisionadas. Uma vez que cada neurônio representa um hiperplano divisor de um espaço vetorial Rn, é possível aplicar a técnica demonstrada no capítulo 3 para um sistema n-dimensional onde cada plano represente um neurônio. A cada colisão representa a necessidade de se expandir as camadas da rede em desenvolvimento. Desta forma pretende-se que o treinamento em redes supervisionadas seja computacionalmente mais rápido, preciso e que gere de forma automática o número de neurônios e camadas.

7. Referências Bibliográficas

[1] Chuter, C., Jernigan, S., Barber K.: A Virtual Environment Simulator For Reactive Manufacturing Schedules, Proc. of Symposium on Virtual Reality in Manufacturing, Research and Education, UIC The University of Illinois at Chicago, October 7-8, 1996.

[2] Bejczy, A. K.: Virtual reality in manufacturing. In: Re-engineering for Sustainable Industrial Production, Proceedings of the OE/IFIP/IEEE Int. Conf. on Integrated and Sustainable Industrial Production Lisbon, Portugal, May 1997, pp.48-60.

[3] International Electrotechnical Commission :IEC61131-3 Standard Edition 2.0

[4] O'Sullivan, C. and Dingliana, J. 1999. Realtime collision detection and response using spheretrees. In Proceedings of the Spring Conference on Computer Graphics. 83–92.

[5] Palmer, I. and Grimsdale, R. 1995. Collision detection for animation using sphere-trees. Computer Graphics Forum 14, 2, 105–116.

[6] Mirtich, B. 1997. V-Clip: Fast and Robust Polyhedral Collision Detection.

[7] S. Gottschalk M. C. Lin_ D. Manocha. 1996. OBBTree: A Hierarchical Structure for Rapid Interference Detection

[8] Moore, M. and Wilhelms, J. 1988. Collision detection and response for computer animation. Computer Graphics 22(4), 289–298.

[9] O'Sullivan, C. and Dingliana, J. 2001. Collisions and perception. ACM Transactions on Graphics 20, 3, 151–168.

[10] Figueiredo, M., Böhm, K., Teixeira, J.: Advanced Interaction Techniques in Virtual Environments. In: Computers & Graphics, Journal, Vol. 17. No. 6., Pergamon Press Ltd, 1993, pp. 655-661.

[11] Guendelman, E., Bridson, R., Fedkiw, R.. 2003. Nonconvex Rigid Bodies with Stacking SIGGRAPH 2003.

[12] A.S. Carrie, E. Adhami, A. Stephens, I.C. Murdoch: Introducing a flexible manufacturing system. In: Int. J. of Prod. Research, Vol.22, No.6, 1984, pp.907-916.

[13] Barequet, G., Chazelle, B. Guibas, L, Mitchell, J. and Tal. A (1996). Boxtree: A hierarchical representation of surfaces in 3d. in the Proceedings of Eurographics'96.

[14] Bouma, W. and Vanecek, G. (1991). Collision detection and analysis in a physically based simulation, in the Proceedings of Eurographics workshop on animation and simulation, páginas 191-203, 1991.

[15] Cameron, S. (1990). Collision detection by four dimensional intersection testing, in the Proceedings of IEEE International Conference on Robotics and Automation, páginas 291-302, 1990.

[16] Cameron, S. (1991). Approximation hierarchies and s-bounds, in the Proceedings of ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, páginas 129-137, Austin, TX, 1991.

[17] Cameron, S. (1997). A comparison of two fast algorithms for computing the distance between convex polyhedra, IEEE Transactions on Robotics and Automation, 13(6):915-920, December 1997.

[18] Cameron, S. and Culley, R. K. (1986). Determining the minimum translational distance between two convex polyhedra, in the Proceedings of IEEE International Conference on Robotics and Automation, páginas 591-596, 1986.

[19] Canny, J. F. (1986) Collision detection for moving polyhedra, IEEE Transactions on PAMI, 8:200-209, 1986.

[20] Cohen, J., Lin, M., Manocha, D. and Ponamgi, M. (1995).
I-Collide: An interactive and exact collision Detection system for large-scale environments, in the Proceedings of ACM Interactive 3D Graphics Conference, páginas 189-196.

[21] Cremer, J. F. and Stewart, A. J. (1994). The Architecture of Newton, A General Purposed Dynamic Simulator, Proceedings of the IEEE International Conference on Robotics and Automation, May 1994.

[22] Dobkin, D. P. and Kirkpatrick, D. G. (1990). Determining the separation of preprocessed polyhedra - A unified approach, in the Proceedings of the 17th Internat. Colloq. Automata Lang. Program., V. 443 of Lecture Notes in Computer Sciences, páginas 400-413. Springer- Verlag.

[23] Garcia-Alonso, A., Serrano, N., and Flaquer, J. (1994). Solving the Collision Detection Problem, IEEE Computer Graphics and Applications, 13 (3): pp. 36-43, 1994.

[24] Gilbert, E. G., Johnson, D. W. and Keerthi, S. S. (1988). A fast procedure for computing the distance between objects in three-dimensional space, IEEE J. Robotics and Automation, Vol RA-4:193-203.

[25] Gregory, A. Lin, M. Gottschalk, S. and Taylor, R. (1999). H-COLLIDE: A Framework for Fast and Accurate Collision Detection for Haptic Interaction, the Proceedings of IEEE VR'99, pp. 38-45, March 1999.