Universidade do Estado de Santa Catarina
Bacharelado em Ciência da Computação

Rejane Gomes dos Santos

Detecção de Colisão para um Simulador de Robô Manipulador

Marcelo da Silva Hounsell
Orientador

Joinville – SC

2007

Universidade do Estado de Santa Catarina
Bacharelado em Ciência da Computação

Rejane Gomes dos Santos

Detecção de Colisão para um Simulador de Robô Manipulador

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação.

Marcelo da Silva Hounsell
Orientador

Joinville, novembro de 2007

Detecção de Colisão para um Simulador de Robô Manipulador

Rejane Gomes dos Santos

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação.

Banca Examinadora

_____
Marcelo da Silva Hounsell (orientador)
PhD

_____
Alexandre Gonçalves Silva
Mestre em Eng.

_____
Carlos Norberto Vetorazzi Jr.
Mestre em Eng.

_____
Roberto Silvio Ubertino Rosso Jr.
PhD

_____
Hélio Pedrini
Dr. em Eng. Elétrica e Computação

_____
Tânia Martins Preto
Mestre em Eng.

AGRADECIMENTOS

Agradeço meu orientador Marcelo por toda paciência e por tudo o que ensinou.

Agradeço meus amigos que são minha fortaleza, minha família por opção, que me apoiaram de todas as formas: com amuletos, com uma palavra amiga, auxílios técnicos, administrativos, festivos.

Agradeço aos meus pais pelo apoio, à minha avó pelas orações que com toda certeza trouxeram a energia poderosa para eu sempre continuar. À minha tia madrinha, que um dia não me deixou desistir.

Por fim, vai um agradecimento especial ao chimarrão que não me permitiu dormir.

RESUMO

Foi desenvolvido pelo grupo LARVA (LAboratório de Realidade Virtual Aplicada) um simulador virtual para o robô manipulador articulado Scorbot ER-4PC. Este é um projeto com fins didáticos, que faz a modelagem geométrica pela técnica de representação por fronteiras na linguagem VRML, WEB. Contudo a aplicação atual não realizava diagnósticos de colisão entre os objetos do ambiente virtual.

Com o objetivo de sanar esta necessidade, realizou-se uma ampla investigação sobre as técnicas atuais para detecção de colisão, adequadas às características do estudo de caso. Esta busca resultou em um material sucinto e abrangente acerca das técnicas de detecção de colisão, com base neste material foi feita a montagem de um novo esquema que reúne as diversas práticas apropriadas a cada escopo do algoritmo, definidos com fase abrangente (*broad phase*) e fase específica (*narrow phase*).

Com o estudo da aplicação e literatura definiu-se aqui um novo conceito para detecção de colisão, a Coerência Semântica que, aliada a coerência espacial, proporcionaram um algoritmo para fase abrangente eficiente para o Scorbot Virtual.

Ainda, para promover a eficiência da aplicação foram utilizadas árvores de Volumes Envolventes OBB, como estrutura de repartição espacial dos objetos e representação em envoltório de cálculo menos complexo. O volume se adaptou aos objetos do ambiente virtual de modo a oferecer raras situações de falsas colisões.

A eficiência no uso de arvores de OBB`s e da coerência semântica foi comprovada em todos os testes realizados obtendo-se, inclusive, desempenho superior a 100% se comparado aos testes de força bruta.

Uma nova interface é proposta, com adição da funcionalidade de criação de cenários variantes que proporciona, com ajuda de técnicas de detecção de colisão, uma variedade de cenários com a disposição e formas geométricas randômicas do número de objetos definido pelo usuário. Com a presença destes elementos também é possível agora realizar a pega do objeto mais próximo ao fechamento da garra, proximidade esta definida pela distancia euclidiana entre os centros de suas OBBTrees.

ABSTRACT

It was developed by the group LARVA (LAboratório of Applied Virtual Reality) a virtual simulator for the articulated robot manipulator Scorbot ER-4PC. This is a project with educational purposes, which is the geometric modeling by the technique of boundary representation in the language VRML, WEB. But the current implementation not held diagnoses of a collision between objects in a virtual environment.

In order to remedy this need, there was an extensive research on the current techniques for the detection of collisions, suited to the characteristics of the case study. This search resulted in a concise and comprehensive material on the techniques of the collision detection on the basis of this material was the assembly of a new scheme which brings together the various practices appropriate to each scope of the algorithm, defined with broad phase and a narrow phase.

With the study of literature and application set-up here a new concept to the collision detection, the Coherence Semantics, which, combined with spatial coherence, provided a comprehensive efficient algorithm for broad phase. Still, to promote the efficiency of the implementation were used Tree Volumes Envolventes OBB, as a structure for sharing of objects and spatial representation in wrapper for calculating less complex. The volume has adapted to the virtual objects of the environment in order to provide rare instances of false collisions.

The efficiency in the use of trees, OBB is the semantic consistency has been proven in all the tests are getting even carrying more than 100% when compared to tests of brute force.

A new interface is proposed, with the addition of functionality to create scenarios that provides variants, with help from the collision detection techniques, a variety of scenarios with the provision and geometric shapes random number of objects defined by the user. With the presence of these elements is also possible now to hold the handle of the object closest to the closing of the gripper, proximity defined by the distance euclidiana between the centers of their OBBTrees.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AABB	Axis-Aligned Bounding Box
ATVE	Árvore de Testes de Volumes Envolventes
B-Rep	Boundary Representation
BSP	Binary Space Partition
CHT	Convex Hull Tree
DH	Devanit e Hartenberg
DOF	Deegres Of Freedom
DOP	Discrete Orientable Polytopes
EAI	External Authoring Interface
HVE	Hierarquia de Volumes Envolventes
LARVA	LAboratório de Realidade Virtual Aplicada
OAABB	Overlapping Axis-Aligned Bounding Box
OBB	Oriented Bound Box
RV	Realidade Virtual
SAT	*Separating Axis Theorem*
SDK	*Standart Development Kit*
VE	Volume Envolvente
VRML	Virtual Reality Modeling Language

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos a tecnologia de Realidade Virtual (RV) vem ampliando seu papel na área industrial através, por exemplo, da implementação de projetos na área de simulação robótica. Essa simbiose entre robótica e RV proporciona diversas vantagens para manufatura, como a realização de treinamentos e programação *off-line*. Com uso dessa ferramenta, é possível simular exaustivamente movimentos do manipulador robótico evitando desgaste ao equipamento e sem necessidade de interrupção do funcionamento dos robôs reais na célula de trabalho que pode estar ligada a uma linha de produção (Redel *et al.,*2004:1).

Dentro deste contexto, está sendo desenvolvido pelo grupo LARVA um simulador para o robô manipulador Scorbot ER-4PC, ilustrado na figura 1. Este é um exemplo de robô didático, feito especialmente para o estudo da robótica. Trata-se de uma estrutura vertical articulada permitindo cinco DOF para manipulação. DOF (*Degrees of freedom* ou Graus de Liberdade) são os movimentos elementares, independentes entre si, que podem ser executados pelo robô (Groover, 1989:1).

O SCORBOT virtual foi modelado segundo a técnica de modelagem geométrica Representação por Fronteiras, que construiu na aplicação objetos poliédricos convexos. A linguagem empregada para descrição desses objetos é o VRML (V*irtual Reality Modeling Language*) que foi projetada para ser veiculada via internet sendo suportada para visualização em diversos *browsers* (Carey e Bell, 1997:1). Essa característica da linguagem torna viável a utilização do simulador via WEB. Parte do sistema é desenvolvido na linguagem Java, necessária para implementar funcionalidades que o VRML não possui, neste caso, a interface com o usuário e flexibilidade para manipulação matemática.

Apesar das vantagens da aplicação, o ambiente virtual não é capaz de detectar quando o robô interpenetra geometricamente a mesa de apoio, sua própria base ou quaisquer objetos que se encontre em sua rota, diminuindo assim a sensação de realismo do operador que fará uso da aplicação. Esta deficiência impossibilita também a manipulação de objetos e ainda permite que ocorram acidentes ocasionados por

movimentos que acabam na intersecção dos corpos. Em suma, a carência da detecção de colisão entre os objetos em cena é o principal motivo que limita atualmente uma maior utilidade do Scorbot Virtual.

Figura 1 - SCORBOT ER-4PC (esquerda) e a versão virtual (direita)

Todavia, em um ambiente virtual, onde a interação com o usuário é constante, a detecção de colisão se torna um fator fundamental a ser considerado em virtude do grande número de testes e cálculos necessários para a realização de uma detecção precisa e interativa, que continue a garantir a sensação de imersão e realismo proporcionado pelo ambiente. Em face dessa necessidade, a detecção de colisão se torna um componente estratégico e amplamente estudado para o desenvolvimento de ambientes virtuais interativos como nos campos de robótica, animação, simulação dinâmica e CAD/CAM (Lin, 1993:1). Em um algoritmo de força bruta, onde cada primitiva é testada contra as demais, o gasto computacional para detecção de colisão é dado por uma função quadrática do número de objetos na cena e suas complexidades (proporcional ao número de faces por objeto), dessa forma, quanto maior o número de objetos maior o gasto computacional (Watt, 2000:517).

O ambiente no qual a simulação está inserida é de extrema importância na definição da forma de trabalho do algoritmo de detecção de colisão, pois este deve ser robusto o suficiente para obter um bom nível de desempenho em diversas situações que envolvem, por exemplo: variação do número de objetos pertencentes ao ambiente.

diferentes tipos de movimentos definidos para os objetos, determinação das velocidades finais e iniciais dos objetos, grau de complexidade da cena, grau de interatividade almejado, modelagem geométrica dos objetos, entre outros (Taddeo, 2005:16).

A questão da eficiência em relação à aplicação é um aspecto importante a se considerar devido à publicação do projeto em uma página WEB. Por serem desconhecidos os recursos computacionais de que o usuário dispõe para execução da aplicação, o programa deve ter o melhor desempenho possível até para os requisitos mínimos de operação.

Assim, conhecendo a relevância do tipo de ambiente para o desempenho da simulação, será de grande importância realizar um estudo dos métodos existentes para detecção de colisão, procurando entre as técnicas analisadas as que se mostrarem mais eficientes para o ambiente simulado, considerando estáticos a base do robô, a mesa de operação e os objetos dispostos aleatoriamente, e móvel o braço robótico.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolver no robô manipulador virtual a capacidade de detectar colisões.

1.1.2 Objetivos Específicos

- Investigar métodos de detecção de colisão;
- Avaliar e projetar uma solução eficiente de detecção de colisão para um ambiente robótico simulado;
- Criar uma nova versão do simulador com a capacidade de efetuar a detecção de colisão.

1.1.3 Resultados Esperados

Ao final deste trabalho será obtida uma nova versão do SCORBOT virtual, disponível via WEB com uma interface ligeiramente alterada e com a capacidade de avisar ao usuário quando ocorrer a colisão entre o robô e seu ambiente durante a movimentação. Ainda, o SCORBOT virtual disporá de um conjunto de objetos aleatórios que serão gerados em cena para sua manipulação.

1.2 Estrutura do Trabalho de Conclusão de Curso

Este trabalho está estruturado como segue. O capítulo 1 apresentou a introdução do trabalho. O capítulo 2 é dedicado ao estudo dos fundamentos sobre Robótica e Modelagem Geométrica. Estes conceitos são de suma importância para um claro entendimento dos critérios de escolha do algoritmo empregado para cada tipo de aplicação.

No capítulo 3 são apresentados conceitos sobre o problema da detecção vindos da investigação do problema na literatura. O capítulo 4 apresenta o Estado da Arte, onde são discorridos os motivos pelos quais as soluções existentes não se aplicam na íntegra ao escopo da aplicação desenvolvida nesse trabalho.

No capítulo 5, o método projetado para detecção de colisão no Scorbot Virtual é descrito detalhadamente para a realização da implementação desenvolvida no capítulo 6. O capítulo 7 discorre sobre as conclusões dos resultados obtidos na implementação da técnica proposta, enquanto o capítulo oito traz as consideraçoes finais sobre o trabalho desenvolvido, como a escolha das técnicas para o desenvolvimento da solução para detecção de colisão no SCORBOT virtual e proposta para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão introduzidos os conceitos fundamentais acerca do projeto desenvolvido, como os conceitos de robótica, as peculiaridades da modelagem geométrica empregada no Scorbot virtual, bem como a tecnologia em que o ambiente virtual foi implementado.

2.1 Robótica

Groover(1989:6) define um robô industrial como um manipulador programável, multifuncional, projetado para mover materiais, peças, ferramentas ou dispositivos especiais em movimentos variáveis programados para realização de uma variedade de tarefas.

Além da anatomia, dois aspectos são indispensáveis para automação de robôs industriais: Movimentação e Programação.

2.1.1 Anatomia

A anatomia do robô descreve a construção física do corpo, braço e punho da máquina. O corpo está ligado a base e o braço ao corpo do robô. Na extremidade do braço está o punho. Movimentos relativos entre os diversos componentes do corpo, braço e punho são proporcionados por uma série de juntas que geralmente envolvem movimentos rotativos ou deslizantes, o robô com essa estrutura é chamado por vezes manipulador (Groover, 1989: 24).

Robôs manipuladores industriais são disponibilizados em uma larga variedade de tamanhos, formas e configurações físicas. A vasta maioria dos robôs, disponíveis comercialmente, possui uma das quatro configurações básicas (Groover, 1989):

1. Configuração Polar (Esférico);
2. Configuração Cilíndrica;
3. Configuração de Coordenadas Cartesianas;
4. Braço Articulado (Revoluto)

O robô braço articulado, que é o modelo do Scorbot, possui uma configuração similar ao braço humano. Ele consiste de componentes lineares correspondentes ao braço e antebraço humano, montados num pedestal vertical. Estes componentes são conectados por duas juntas rotacionais correspondestes ao ombro e o cotovelo. Um punho é fixado no final do antebraço, com isto provendo várias juntas adicionais.

No Scorbot, constam dois componentes retos denominados elos, correspondendo ao antebraço e braço humanos, montados verticalmente. Esses membros são conectados por duas juntas rotacionais correspondendo ao ombro e cotovelo. O punho está unido ao antebraço. No caso do SCORBOT 4PC o órgão terminal ligado ao antebraço é uma pinça de dois dedos (Eshed Robotec, 1982: 5). A anatomia do modelo bem como o movimento proporcionado pelas juntas do Scorbot ER 4PC são mostrados na figura 2.

2.1.2 Movimentação

Os movimentos do robô podem ser divididos em duas categorias básicas: movimentos do braço e do corpo e movimentos do punho. Os movimentos são realizados por meio de juntas acionadas.

Os movimentos elementares, independentes entre si, associados às juntas individuais são chamados graus de Liberdade (DOF - *Deegres of freedom*). Para uma movimentação adequada são necessários no mínimo três DOF's (Groover, 1989: 31).

2.1.3 Programação

A programação da trajetória consiste em processar o local geométrico dos pontos no espaço definindo a seqüência de posições através das quais o robô irá movimentar seu punho (Groover, 1989:240). Os métodos de programação de um manipulador podem ser divididos em duas categorias:

a) Método de programação por aprendizagem: Essa técnica exige que o programador desloque o manipulador real através da trajetória desejada, que será memorizada pelo controlador do robô.

b) Linguagens textuais para robôs ou Programação *off-line:* O programador configura a trajetória do robô através de uma linguagem de alto nível, sem auxílio do robô real que normalmente se encontra integrado a uma linha de produção. Tem as seguintes vantagens:

- Permite a utilização de posições determinadas analiticamente;
- Permite integração com sistemas CAD/CAM;
- Facilita manutenção;
- Não há necessidade de parar o processo produtivo para reprogramação;
- Permite predição de eventuais erros.

Atualmente a movimentação do simulador do braço robótico é interativa, realizada através da interface em *applets* - Java ou diretamente com o ponteiro do mouse clicado sobre o robô na interface VRML. Dessa forma, o usuário decide as características do movimento do braço no momento em que interage com o mesmo.

Futuramente, em novos projetos, o simulador pode ser adaptado para receber os testes da programação textual antes que a mesma seja implementada no manipulador real, evitando assim desgastes desnecessários no equipamento e otimizando o tempo de implementação da nova programação. Essa adaptação também tornaria o processo da programação textual do robô uma tarefa mais visual, interativa e agradável.

2.1.4 SCORBOT ER 4PC

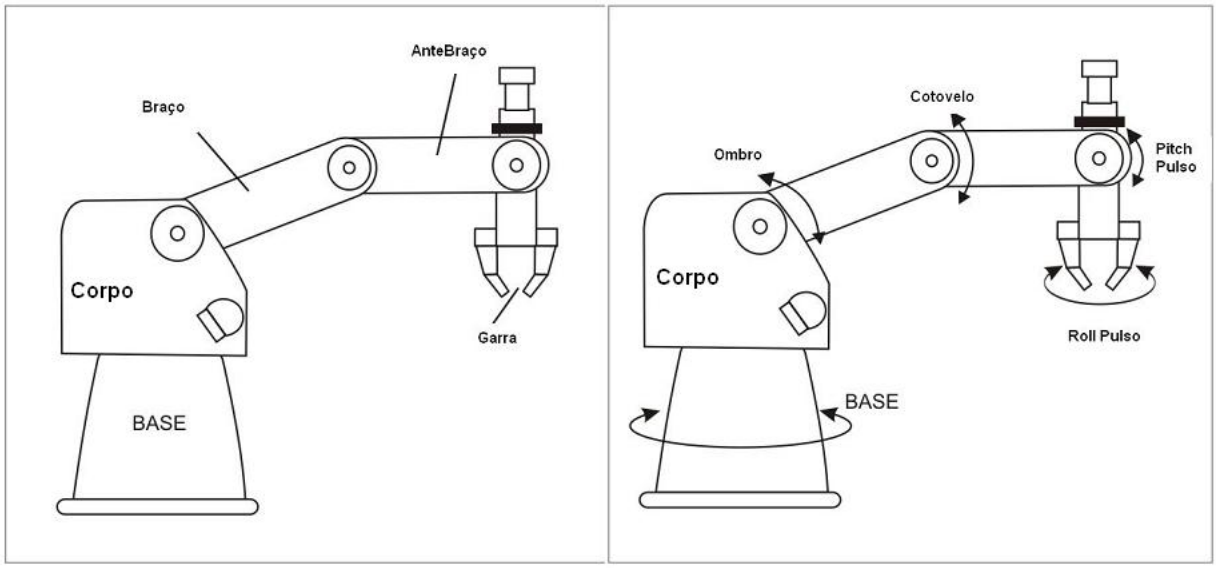


Figura 2 – Anatomia e movimentação do Scorbot ER 4PC.
À esquerda, os membros do robô. À direita, a representação do movimento proporcionado pelas juntas (Eshed Robotec, 1982: 5).

O SCORBOT ER 4PC, ilustrado na figura 1, é um robô manipulador articulado desenvolvido para aplicação em laboratórios e treinamentos. A aplicação didática permite aos estudantes o ganho de experiência prática e teórica em robótica, automação e controle de sistemas (Eshed Robotec, 1982: 5).

No modelo SCORBOT 4PC existem quatro ligações elo-junta que proporcionam cinco DOF's ao manipulador, estes movimentos são ilustrados na figura 2 e descritos na tabela 1. São eles:

1. Movimento de rotação – Base-Corpo
2. Movimento de rotação – Corpo-Braço
3. Movimento de rotação – Braço-Antebraço
4. *Pitch* – Garra-Antebraço (Pulso)
5. *Roll* - Garra-Antebraço (Pulso)

Pitch é o movimento que a garra faz, com sua ponta, de cima para baixo (e vice versa) e *Roll* é o movimento de rotação dela em relação ao seu próprio eixo.

A altura dos elos e o grau de rotação das juntas determinam o alcance do envelope de trabalho do manipulador. Essa região de circulação do braço robótico é ilustrada na figura 3, que mostra o alcance em relação a altura e à área circunvizinha ao robô. A tabela 1 demonstra as características definidas pelo fornecedor relevantes ao escopo desse trabalho.

Especificações do Fabricante	
Estrutura Mecânica	Verticalmente Articulado
DOF	5 eixos rotacionais + Pinça
Alcance	610 mm (24,4")
Abertura da Pinça	75 mm (3")
Rotação dos Eixos	Eixo 1 - Rotação Base: 310º Eixo 2 - Rotação do ombro: +130º/-35º Eixo 3 - Rotação do cotovelo: +/-130º Eixo 4 – *Pitch* do Punho:+/-130º Eixo 5 - *Roll* do Punho: Ilimitado

Tabela 1 – Especificações técnicas do Scorbot ER 4PC
(Eshed Robotec, 1982: 4)

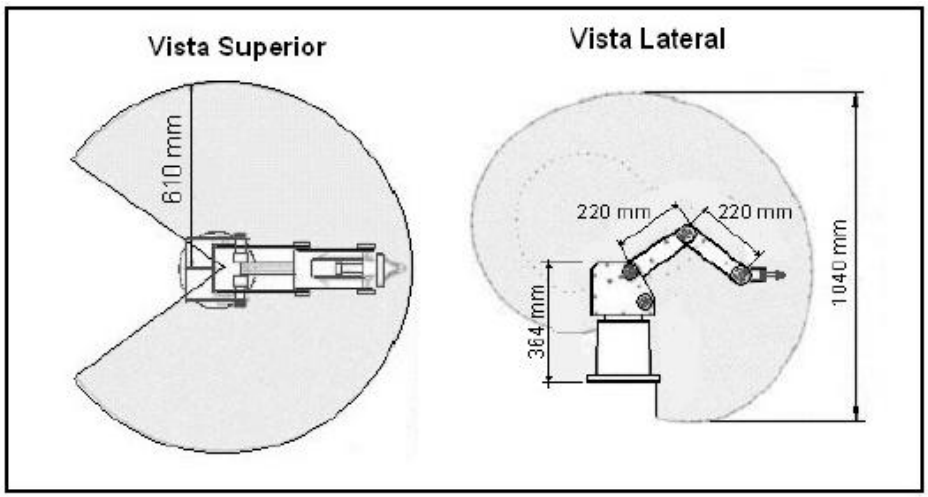


Figura 3 – Alcance do envelope de trabalho do Scorbot ER 5PC
(Eshed Robotec, 1982: 6)

2.2 Modelagem Geométrica

A modelagem de Sólidos é uma forma de se representar um objeto tridimensional com certas propriedades desejadas. É uma representação matemática completa e não ambígua de um objeto físico (Requicha,1980:442)

Segundo Lin e Gottschalk (1998), o modelo de representação geométrica dos objetos em cena influenciará diretamente no desenvolvimento do algoritmo de detecção de colisão projetado para cada aplicação. Figueiredo *et al*.(2002) e Taddeo(2004) inclusive fazem uma classificação dos tipos de abordagens de algoritmos de detecção de colisão utilizando o esquema de modelagem geométrica em que são representados os objetos como fator que categoriza estas abordagens.

Dessa forma, serão apresentadas neste trabalho técnicas eficientes para detecção de colisão em modelos que utilizam a modelagem geométrica empregada no Scorbot Virtual, a Representação por Fronteiras. Esta técnica de modelagem será descrita nos próximos tópicos para maior entendimento dos critérios de projeto do algoritmo de detecção de colisão para o SCORBOT virtual.

2.2.2 Representação por Fronteiras (B-Rep)

A modelagem geométrica de sólidos por representação de fronteira (também designada por b-rep, abreviatura de "*boundary representation*") descreve um objeto por meio das superfícies que o limitam e das arestas e vértices (geometria) que estas superfícies - ou faces – apresentam. Essas superfícies são fechadas, ou seja, são conjuntos de faces que não possuem interrupções, todas as faces são conectadas por arestas, de modo a criar uma "casca" fechada que forma o objeto (Zeid, 1991:370).

As superfícies limitam os objetos mas não indicam de que lado estes se encontram. Para resolver essa situação as faces são orientadas, cada uma possui um vetor normal, que aponta para fora do objeto modelado, permitindo assim obter-se uma clara separação entre interior e exterior do objeto (Zeid, 1991:370). A figura 4 mostra a criação de uma pirâmide de base quadrangular utilizando o esquema de representação

b-rep. A geometria é dada pelos vértices, que associados formam a topologia do objeto, ou seja, as faces limitadas pelas arestas.

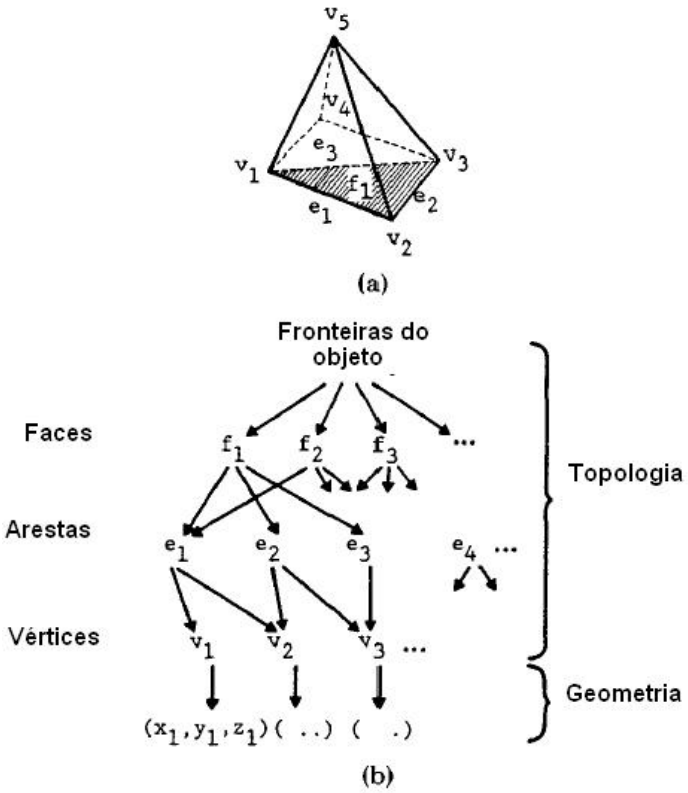


Figura 4 – Modelagem geométrica por fronteiras
(Requicha, 1980: 452)

Em geral, pode-se considerar representações de fronteira em que os elementos das superfícies que as constituem podem assumir qualquer forma geométrica, mas isto significa passar os problemas derivados de formas complexas para cada um dos elementos constituintes da fronteira. Na prática, a maioria das representações b-rep restringe estes elementos geométricos a formas simples como superfícies poligonais planas que formam objetos poliédricos, e este tipo de b-rep é que está sendo utilizado no SCORBOT Virtual.

Restrições ainda mais severas, mas que aumentam a flexibilidade e simplicidade das representações de fronteira, podem obrigar que os polígonos sejam convexos. A precisão da aproximação dependerá do número e dimensão dos polígonos utilizados na sua descrição.

O *b-rep* poliédrico é amplamente utilizado para criação de modelos sólidos de objetos físicos por possuírem uma representação simples, serem versáteis e utilizarem programas de renderização de polígonos implementados em *hardware*.

2.1 VRML – EAI

2.1.1 VRML

VRML (V*irtual Reality Modeling Language*) é a sigla em inglês para a expressão “Linguagem de Modelagem para Realidade Virtual”. Esta linguagem serve para o desenvolvimento de aplicações interativas em três dimensões voltadas para a internet.

Conjuntos de objetos, ou melhor, abstração de objetos e de certas entidades do mundo real, tais como: esferas, cubos, luz, som, dentre outros, são definidos como sendo os "componentes fundamentais " de uma cena em VRML, pois esta é construída a partir da disposição, combinação e interação entre nós.

Um *nó* (ou *node* em inglês) é a construção fundamental de um arquivo VRML. Alguns nós são objetos prontos - Cylinder, Cone, Box, SpotLight. Outros nós são usados como *containers* que armazenam nós com alguma relação lógica. Um nó *Shape*, por exemplo, contém um nó *geometry* que indica a forma do objeto e outro *appearance* que controla a sua aparência. Estes nós podem, por sua vez, conter outros, e assim por diante (CAREY e BELL, 2006).

Além dos nós que têm um efeito visual óbvio, em VRML está definida uma série de nós que oferecem alguns recursos especiais, como ligações com outros arquivos *hyperlinking*, a inclusão de objetos definidos em outros arquivos, e a detecção de colisões entre a câmera e o ambiente. Cada nó contém *campos* que mantêm os dados que o caracterizam como um elemento único dentro do arquivo.

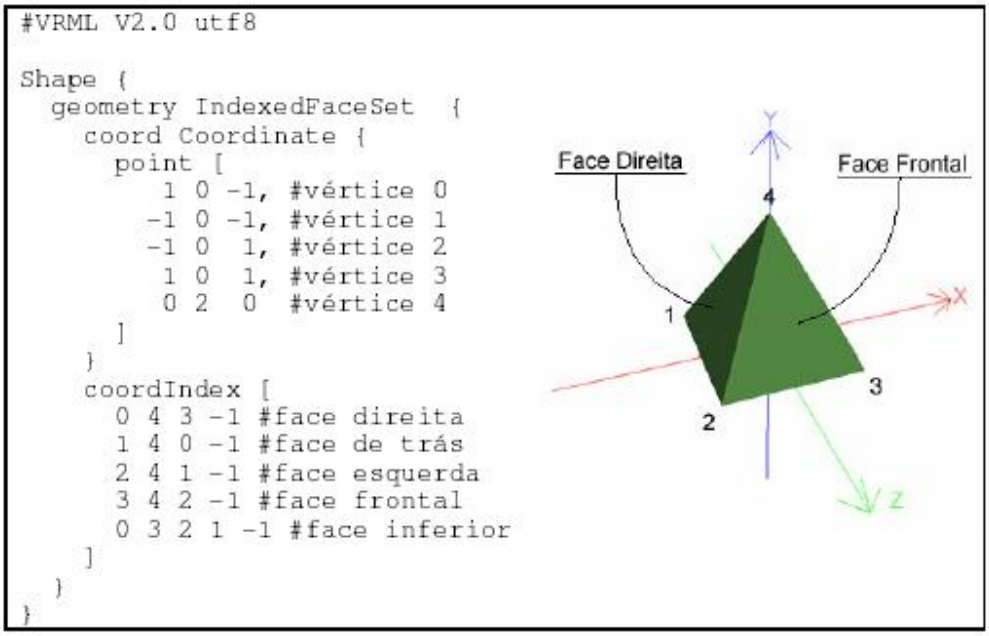


Figura 5 – Implementação de uma Pirâmide b-rep em VRML
(Redel *et al., 2004:3*)

Os membros do robô ER4PC não possuem formatos regulares, o que impossibilita a utilização das formas primitivas prontas do VRML. Então a modelagem b-rep é feita através de um nó chamado *IndexedFaceSet* (Redel *et al.,*2004:3). A partir de uma lista de vértices o *IndexedFaceSet* constrói faces que representam um objeto em 3D. A Figura 5 mostra um exemplo do *IndexedFaceSet* representando uma pirâmide. A regra para o sistema de coordenadas é o da mão direita, o vetor normal da face aponta para a direção dada pelo polegar (Carey e Bell, 1997). A propriedade *coord* contém um nó do tipo *Coordinate* que armazena um conjunto de coordenadas 3D representando os vértices do objeto. O primeiro vértice possui índice zero, o segundo possui índice um e assim sucessivamente. A propriedade *coordIndex* armazena índices com os quais é possível especificar as faces do polígono. A seqüência de vértices finalizada por –1 representa uma face.

2.1.2 *External Authoring Interface* - EAI

A interface EAI define um conjunto de funções dos *browsers* VRML que podem ser invocados exteriormente para afetar o mundo VRML, permitindo que qualquer programa Java possa interagir assim de diversas formas com qualquer mundo VRML. Do ponto de vista da linguagem Java, a interface EAI é um conjunto de classes com métodos que podem ser invocados para controlar o mundo VRML. Do ponto de vista do *browser* de VRML, a interface EAI é mais um mecanismo que permite "enviar e receber eventos", tal como o próprio VRML(Marrin, 1997).

A figura 36 ilustra o funcionamento da integração, onde uma *applet Java* interage com o VRML enviando e "ouvindo" eventos do mundo VRML através da interpretação da interface EAI, adicionando assim um meio padronizado e portável para representar cenas tridimensionais dinâmicas e interativas na Internet.

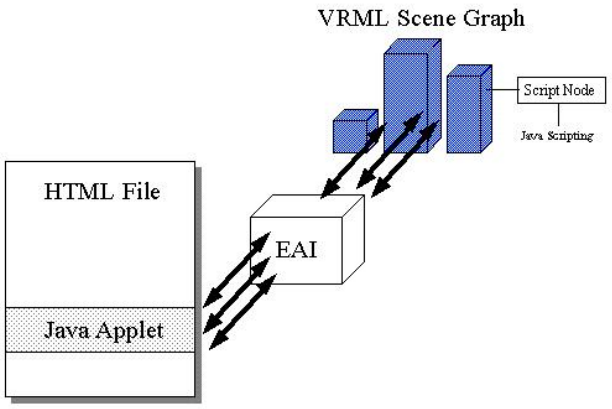


Figura 6 – Integração Java – VRML (Marrin, 1997)

No documento em que Chris Marrin propõe a interface EAI, são identificadas quatro formas de acesso aos mundos VRML através da interface EAI (Marrin, 1997):

1. Acessando às funcionalidades do Script de interface com o Browser;
2. "Enviando eventos" aos *eventIns* (eventos "de chegada") dos nós existentes no mundo;
3. Lendo o último valor "enviado" pelos *eventOuts* (eventos "de partida") dos nós existentes no mundo;

4. Sendo notificado quando do “envio de eventos” pelos *eventOuts* dos nós existentes no mundo.

Para interagir com um determinado nó de um mundo VRML, um programa Java tem de começar por obter uma referência para o *browser* VRML (método *getBrowser*()) e, em seguida, obter uma referência para esse nó (método *getNode*()). No entanto, estão acessíveis os nós que tiverem sido nomeados com o construtor DEF, uma vez que, é o nome associado ao nó quando da construção do mundo VRML que permite obter a referência a este. Uma vez obtida esta referência, torna-se possível aceder aos *eventIns* e *eventOuts* deste nó, recorrendo, respectivamente, aos métodos *getEventIn*() e *getEventOut*().

3.1 Tratamento de Colisões

Uma animação realista requer que os objetos em movimento obedeçam a leis físicas. Para tanto, um importante aspecto deve ser considerado: no mundo real dois objetos não podem ocupar o mesmo lugar no espaço. Isto significa que o momento de uma eventual colisão os objetos podem empurrar outros dependendo de suas massas e velocidades, objetos podem ser empilhados, deformados, etc. Dessa forma o tratamento da colisão acaba se dividindo em duas fases distintas: a detecção de colisão e a resposta à colisão (Atencio, 2005:1).

O principal objetivo dos algoritmos para a detecção de colisão é encontrar eventuais ocorrências de contato entre objetos. Essa tarefa não é trivial e tem se caracterizado como o grande desafio das atuais aplicações que simulam ambientes gráficos interativos. Os tipos de consulta de detecção de colisão podem variar, esta pode requerer a detecção do contato entre dois objetos, a distância euclidiana entre esses corpos ou a distância de interpenetração de dois objetos que se interceptaram. Por exemplo, a informação de distância é útil para calcular interação de forças ou funções de programação de movimento de um robô. As funções de intersecção são importantes para sistemas de animação que precisam conhecer o ponto exato da intersecção para a computação de uma resposta de colisão (Lin e Gottschalk. 1998:4). Assim, embora todos os campos tenham interesses em comum eles possuem, notoriamente, diferentes propósitos baseados no tipo de resposta à colisão.

Os algoritmos de resposta, por sua vez, possibilitam ao ambiente gráfico a modificação do comportamento dos objetos envolvidos na colisão, alterando os diversos parâmetros físicos - tipicamente posição, orientação e velocidade, de forma a garantir que as ações efetuadas pelos objetos colidentes sejam as mais próximas da realidade.

O escopo desse trabalho se limita à realização da detecção de colisão, sendo reportada apenas o momento de ocorrência de contato entre dois objetos. Os algoritmos de resposta à colisão não serão abordados nesse projeto, devido à complexidade envolvida nessas técnicas, que necessitam considerar características físicas do ambiente.

3.2 Coerência Temporal e Espacial

Num ambiente virtual onde os objetos movem-se suavemente, as técnicas de detecção de colisões podem tomar vantagem das pequenas alterações nas posições e orientações desses objetos entre os quadros exibidos. Essa propriedade, conhecida na literatura como coerência temporal, mantém uma estrutura de dados com as interseções anteriormente calculadas com o objetivo de acelerar o cálculo das próximas interseções(Lin *et al.*, 1997).

Os objetos em uma cena encontram-se distribuídos no espaço de acordo com posições, orientações e escalas. As técnicas de detecção de colisão geralmente valem-se da distribuição espacial desses objetos como um artifício para evitar rapidamente objetos que não podem estar colidindo. Essa propriedade, conhecida na literatura como coerência espacial, pode utilizar técnicas de particionamento do espaço para evitar o teste de colisão em pares que se encontram em diferentes regiões do espaço (Taddeo, 2004:33).

3.3 Detecção Discreta X Contínua

Em geral, dependendo de como algoritmos lidam com movimentos dos objetos, a detecção de colisão pode ser categorizada em dois tipos: discreta ou contínua. Em algoritmos de detecção de colisão discreta, as consultas de colisão são realizadas em determinados intervalos de tempo enquanto existe movimento dos objetos. Contudo, o modelo pode mover-se muito rapidamente ou ser muito fino proporcionando facilmente uma falha na detecção de uma colisão, falha causada pelo intervalo de tempo insuficiente na chamada da execução do algoritmo.

Através dessa técnica, uma colisão só pode ser reportada depois que ela já aconteceu (Zhang, 2006:751). Isto porque a análise é feita de forma estática considerando o momento em que o algoritmo é executado, nesse momento ele é capaz de informar se existe colisão, não sendo possível reportar também o momento exato em que o contato ocorreu.

Por outro lado, algoritmos de detecção de colisão contínua consideram o movimento dos objetos e reportam o primeiro instante de contato entre dois modelos se uma colisão vier a acontecer. A grande importância desse tipo de abordagem reside no fato de que o resultado sempre garante a não interpenetração dos objetos. Porém, a maior limitação existente é que são, em geral, mais lentos do que métodos discretos (Taddeo, 2004:32). Existem cinco abordagens utilizadas na literatura para realização da detecção contínua(Zhang, 2006:721):

Resolução por equação algébrica: que procura resolver o problema da detecção contínua pela resolução explicita de uma equação subjacente

Abordagem de varredura: que computa o volume de varredura criado pelo movimento do objeto, testando o volume gerado com o resto do ambiente em busca de colisão.

Estrutura de dados cinemática: é um de esquema de programação baseada no uso de certificados que nos diz quando uma colisão poderá acontecer.

Subdivisão adaptativa: emprega um teste de separação conservativo que assegura separação completa entre determinados intervalos de tempo, e seletivamente subdivide o intervalo de tempo em que falha o teste até que o intervalo subdividido se torne tão pequeno quanto à tolerância ao longo da dimensão de tempo

Abordagem baseada na Soma de Minkowski: calcula o fecho convexo do conjunto de todas as combinações a + b, onde o vetor $a \in$ A e $b \in$ B.

3.4 Níveis de Detecção

Em um ambiente que contém *n* objetos e cada objeto é testado contra cada um dos demais contidos no ambiente, chamado algoritmo de força bruta, a complexidade algorítmica é de $O(n^2)$, assim o gasto computacional para detecção de colisão é dado por uma função quadrática do número de objetos da cena e suas complexidades (proporcional ao número de faces por objeto) (Watt, 2000:517). Conclui-se que, o fator desempenho de um algoritmo depende não apenas do teste de detecção de colisão básico utilizado, mas também do número de vezes que este teste é aplicado. Por isso,

é crucial realizá-los apenas nos momentos e lugares em que realmente a colisão pode acontecer (Jimenez, 2002:269).

Com o objetivo de minimizar o número de testes são implementadas diversas estratégias para redução do número de pares de objetos com potencial colisão e a redução do número de primitivas analisadas(Jimenez, 2002:270). Essas estratégias são executadas dentro de dois níveis de detecção de colisão conhecidos como:

3.4.1 Fase Abragente (*Broad phase* ou *n-body processing)*

Nesta fase, todos os objetos são testados de forma aproximada. Tem como objetivo determinar, através de técnicas eficientes, o maior número possível de pares de objetos que, definitivamente, não colidem. Posteriormente são realizados testes mais precisos nos pares não excluídos para certificar a colisão (Lin e Gottschalk, 1998:11).

A complexidade desta fase, no caso da aplicação de um algoritmo de força bruta, seria de O(n^2), sendo *n* o número de volumes envolventes associados aos objetos. Para melhorar essa complexidade, reduzindo o número de objetos testados, são utilizadas técnicas que visam explorar a coerência espacial através de estruturas de particionamento de espaço (Taddeo, 2004) ou a coerência temporal, aplicando técnicas de varredura e cortes (*Sweep and Prune,* Cohen *et al,1994:9),* que serão melhor detalhadas à frente.

3.4.2 Fase Específica (*Narrow phase* ou *pair processing)*

Por conta do processamento entre os pares de objetos selecionados na primeira etapa, esta fase determina se o par efetivamente colide e qual o ponto exato da colisão (se existir) (Lin e Gottschalk, 1998:11).

Novamente, para detecção de colisão entre os triângulos dos objetos a complexidade do teste de força bruta é de O(m^2), sendo m o número de primitivas testadas.

Para reduzir o número de primitivas testadas, nessa fase tipicamente são utilizadas hierarquias de volumes envolventes - HVE (Bradshaw, 2004), que serão explanadas mais adiante.

Na fase de detecção abrangente, *broad phase,* poderão ser utilizadas técnicas de decomposição espacial que visam explorar a coerência espacial do ambiente elegendo apenas pares de objetos contidos do mesmo subespaço da cena através das estruturas de representação da partição espacial: *grid, r-tree, octree, BSP ou árvores k-d.*

Outra técnica utilizada para essa fase da detecção é a *Sweep and Prune* que explora a característica da posição geométrica dos objetos envoltos por AABB's (a ser detalhada adiante). Ainda é possível realizar um teste entre os Volumes Envolventes (VE) associados aos objetos da cena para verificação de uma possível colisão, sendo este último método chamado força bruta. Alguns dos volumes mais comuns são AABB's (*Axis Aligned Bounding Box*), OBB's (*Oriented Bounding Box*), k-DOP's (*Discrete Orientable Polytopes*), envoltórios convexos.

Para fase específica, *narrow phase*, onde a colisão é de fato certificada, são utilizadas as técnicas de detecção de colisão entre as hierarquias de VE's associados a cada objeto, em uma detecção mais precisa são verificados também o ponto de contato entre os polígonos que compõe os modelos. Cada uma dessas técnicas será abordada, com exceção da intersecção de polígonos. A introdução do esquema visa demonstrar um panorama sobre a organização das técnicas de detecção de colisão para modelos poliedrais.

Com base na investigação de literatura foi estruturado o esquema da figura 7, criado para representar as abordagens existentes mais utilizadas para detecção de colisão em modelos poliédricos. Com base neste esquema é possível visualizar a formação do corpo de um algoritmo para detecção de colisão. Onde, de acordo com as necessidades do ambiente, são escolhidas as práticas mais adequadas para o escopo de cada etapa inferida.

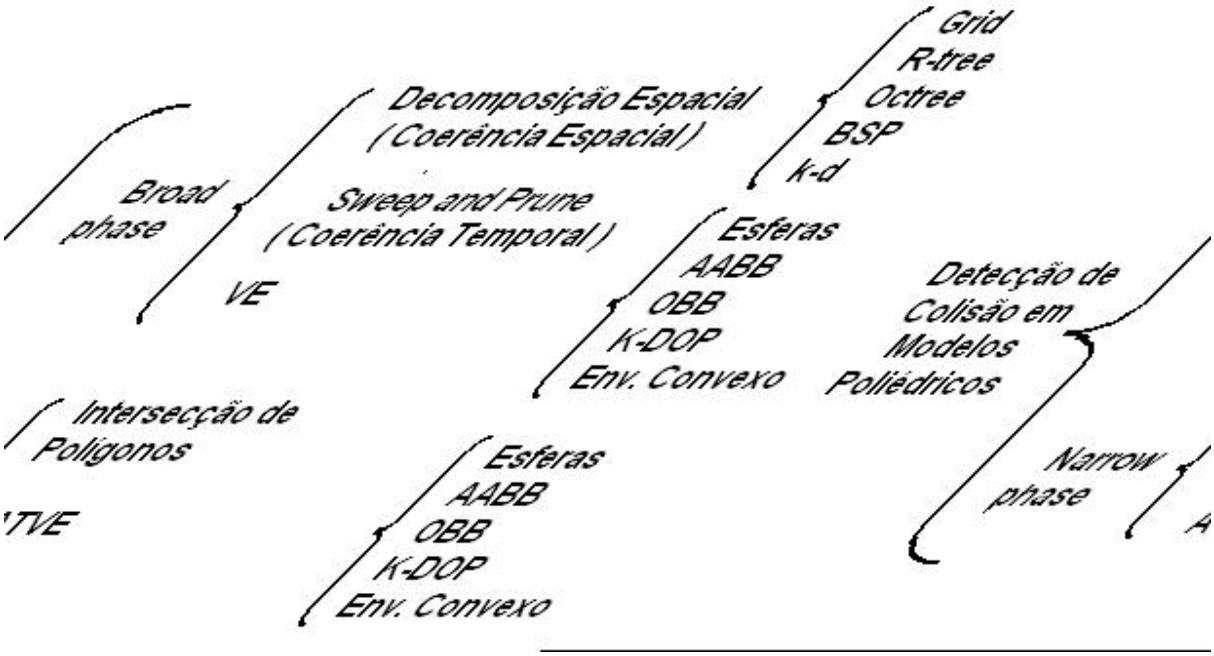


Figura 7 – Esquema dos Métodos de Detecção de Colisão

3.5 Volumes Envolventes

A idéia é envolver cada objeto da cena, ou um conjunto de faces que o compõe, em uma primitiva geométrica para realizar os testes de interseção. Isso, pois, normalmente possuem complexidade inferior às superfícies envolvidas, tornando os cálculos mais rápidos do que o teste de interseção entre polígonos. Ao se constatar que dois volumes envolventes não estão colidindo, tem-se que todos os triângulos que compõem as entidades também não estão (Lin, 2003).

Para o melhor desempenho do algoritmo, a representação por envoltórios deve cumprir as seguintes propriedades da melhor forma possível (Jimenez, 2002: 279):

Possuir um ajuste satisfatório com as primitivas envolvidas, para evitar as falsas colisões;

Possuir um teste de colisão simples entre dois envoltórios;

Deve ocupar pouco espaço de armazenamento;

Possuir rápida atualização da geometria do volume nos movimentos de translação ou rotação dos objetos.

As representações mais utilizadas para detecção de colisão são esferas, AABB's, OBB's (Gottschalk, 2000: 20), mas existem ainda as representações por k-DOP's, envoltórios convexos entre outras. A figura 8 mostra uma escala comparativa em relação ao ajuste, custo de construção e intersecção dos volumes envolventes citados. Nos próximos tópicos, as devidas explicações sobre as características de cada VE (Volume Envolvente) serão discutidas mais detalhadamente.

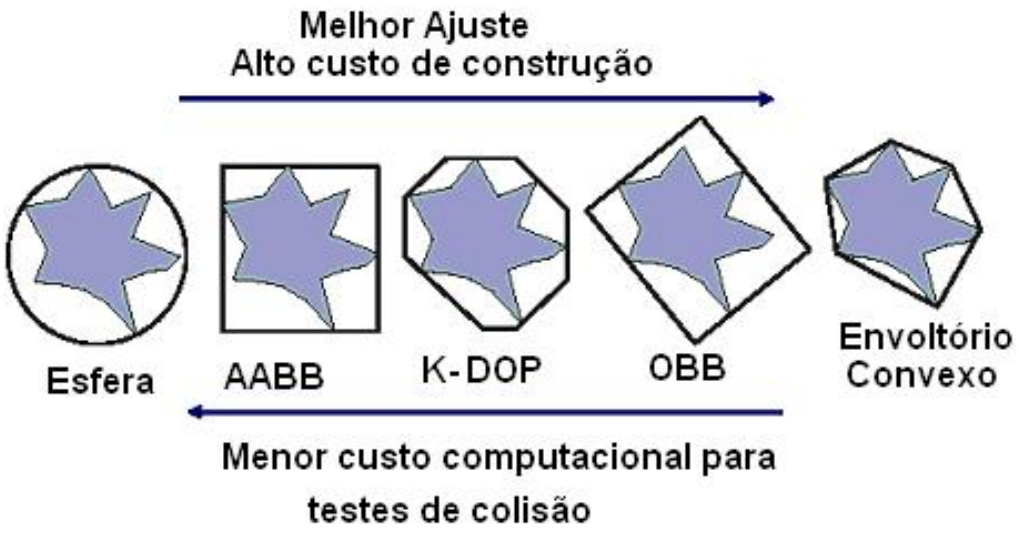


Figura 8 – Comparativo entre VE's
(Kimmerle, 2005:8)

3.5.1 Esferas

Os envoltórios esféricos necessitam somente de dois parâmetros para serem descritos: o ponto onde se localiza o centro da esfera (x, y, z) e o raio da esfera. Elas são rotacionalmente invariantes, assim, a esfera pode ser atualizada simplesmente pela translação de seu ponto central (Bradshaw e O'Sullivan, 2004:2).

Um algoritmo utilizado por Nakamura (2005) para a construção da esfera mínima foi desenvolvido segundo o método de Ritter (1990), e é o que segue:

1. Para todos os vértices da geometria encontrar o mínimo e máximo em cada eixo x,y,z.

2. Para esses três pares de vértices, encontrar o par com a maior distância entre eles.
3. Usar este par para criar uma esfera, sendo o centro (c) igual ao ponto médio do segmento e o raio (r) igual à metade da distância.
4. Para todos os vértices testar se este se encontra dentro ou fora da esfera criada.
5. Se o vértice (v) estiver fora então:
    a. Calcula-se o novo raio para conter o ponto:
       r = (d + r) / 2 onde d é a distância do ponto ao centro da esfera;
    b. Centro_antigo = c;
    c. O novo centro da esfera será: c = (r*centro_antigo + k * Vértice) / d, onde k = v – r.

A colisão de um par de esferas ocorrerá quando a distância entre seus centros for menor que a soma da medida dos seus raios. O teste é realizado de acordo com a seguinte equação, em que duas esferas se interceptam se:

$$(c_{1x}-c_{2x})^2 + (c_{1y}-c_{2y})^2 +(c_{1z}-c_{2z})^2 < (r_1 + r_2)^2 \qquad (3.1)$$

Onde (c_{1x}, c_{1y}, c_{1z}) e (c_{2x}, c_{2y}, c_{2z}) são os centros das esferas, e r_1 e r_2 os respectivos raios.

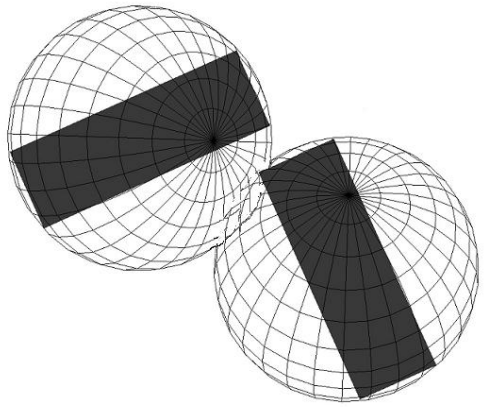


Figura 9 – Volumes Envolventes Esféricos

O teste de colisão possui um ótimo custo para o cálculo de colisão comparado aos outros envoltórios, porém, como se pode perceber na figura 8 esferas não possuem bom ajuste dependendo do objeto envolvido, o que pode ocasionar a detecção de uma falsa colisão quando as esferas se interceptarem em locais das mesmas onde, pelo menos em uma delas, o objeto não está presente.

3.5.2 AABB

Os envoltórios AABB's são paralelepípedos alinhados aos eixos cartesianos (A*xis Aligned Bounding Box* – também chamado apenas *Bounding Box*), ver figura 10. Para construção da AABB mínima é feito o levantamento entre todos os vértices que compõem a geometria do conjunto de superfícies as serem envolvidas o menor e maior valor em cada eixo x, y e z. A combinação desses pontos corresponderá aos vértices da AABB, dessa forma é necessário armazenar apenas os vértices máximos e mínimos.

A intersecção entre duas AABBs pode ser facilmente verificada comparando-se suas dimensões ao longo dos três eixos. O teste pode ser computado com base na separação entre as AABB's em cada eixo (Lin *et al*, 2003: 8). Uma forma de desenvolver esse método é, para cada ponto dos vértices de uma das AABBs, realizar os seguintes testes em relação aos valores máximos e mínimos de cada eixo da segunda AABB, se qualquer um dos testes for verdadeiro, então existe colisão.

1. $(x_{min} < x < x_{max})$
2. $(y_{min} < y < y_{max})$
3. $(z_{min} < z < z_{max})$

O uso desse VE proporciona baixo custo no teste de colisão, porém o ajuste não está entre os melhores se comparado a outros métodos, como pode ser verificado na figura 9.

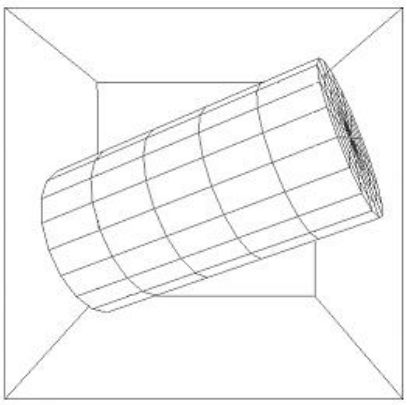


Figura 10 – Volume Envolvente AABB

3.5.3 OBB

OBB's (*Oriented Bound Box*) são paralelepípedos que, diferente das AABB's, são alinhados ao eixo do objeto que representa, como visto na figura 11. Sendo assim, toda a estrutura da OBB rotaciona e translada junto com o objeto envolvido.

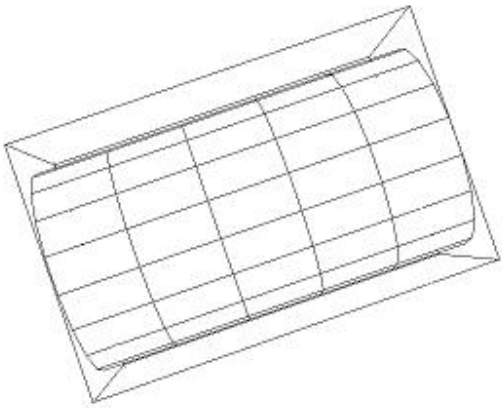


Figura 11 – Volume Envolvente OBB

Uma OBB é construída normalmente por um ponto central C, três vetores ortogonais u^1, u^2 e u^3 que formam o eixo do modelo representado e que compõe as colunas de uma matriz 3X3 de rotação e, finalmente, por três arestas de extensão e_1, e_2 e e_3 correspondentes a cada um dos três eixos (que definiriam altura, largura e comprimento da caixa se ela estivesse orientada ao eixo cartesiano). Dessa forma qualquer ponto normalizado (x_1, y_1, z_1) pertencente à região delimitada pela OBB pode ser definido pela equação 3.3 (Gottschalk,2000:21):

$$R = \{ C + x_1 u^1 e_1 + y_1 u^2 e_2 + z_1 u^3 e_3 \mid x_1, y_1, z_1 \in [-1,1] \} \quad (3.2)$$

Segundo Gotschalk (2000) a forma de uma nuvem de vértices pode ser descrita pela matriz da covariância, M, e sua localização segue o ponto centróide, C. Para pontos no espaço n-dimensional, a forma é dada por uma matriz de simetria nxn e um n-vetor respectivamente. Portanto, a orientação dos eixos do conjunto de primitivas a ser envolvidas pela OBB são calculados com base na distribuição Gaussiana dos vértices V_j que compõe a geometria, desta forma, o centro da caixa é o ponto médio entre os pontos dada pela equação (Gottschalk,2000:21)(Hofman, 2002:51):

$$C = \frac{1}{n} \sum_{j=0}^{n} Vj \quad (3.3)$$

Os eixos do modelo, que são representados na matriz de rotação serão construídos como sendo os auto-vetores da matriz de covariância (Hofman, 2002:52):

$$M = \frac{1}{n} \sum_{j=0}^{n} (Vj - C)(Vj - C)^T \quad (3.4)$$

Se u^j são os auto-vetores unitários, as extensões (arestas) ao longo destes eixos são os extremos das projeções dos pontos sobre eles, verificados por :

$$e_i = \max_j | U_i . V_j - C | \quad (3.5)$$

Um algoritmo eficiente para executar o teste de colisão entre duas OBB's é baseado no Teorema da Separação dos Eixos (*Separating Axis Theorem* – SAT) de Gottschalk *et al.*(1996). O teorema diz que: " Dois poliedros convexos estarão disjuntos se existe um eixo de separação ortogonal a uma das faces de qualquer um dos poliedros ou ortogonal a uma aresta de cada poliedro".

Na figura 12 é possível analisar que os três eixos normais às faces do objeto A e B serão os próprios vetores de orientação de cada objeto. Isso ocorre pelo fato da OBB possuir seus três pares de faces orientados pelo eixo do objeto que envolve, mantendo

sempre o paralelismo entre suas faces. Daí surgem os eixos ortonormais a cada face de cada dos poliedros, somando 6 (3+3) no caso das OBB's (Gottschalk *et al.*,1996).

Da mesma forma, como cada aresta de uma OBB é orientada por algum dos três eixos de orientação do objeto, para definição dos eixos de separação ortonormais a pelo menos uma das arestas de cada poliedro tem-se que o produto vetorial, que estabelece o vetor normal de outros dois vetores, entre os eixos de orientação de cada OBB A^1, A^2, A^3 e B^1, B^2 e B^3 determina os 9 (3 * 3) eixos de separação restantes.

Destas constatações, baseadas no teorema SAT, tem-se ao todo quinze eixos a serem analisados na detecção de colisão entre duas OBB's, demonstrados na seqüência pela tabela 2:

- 3 eixos normais às faces de cada um dos dois poliedros, totalizando 6 vetores;
- 9 eixos resultantes do produto vetorial entre as três possíveis orientações das arestas em cada poliedro, totalizando assim os 15 vetores para análise.

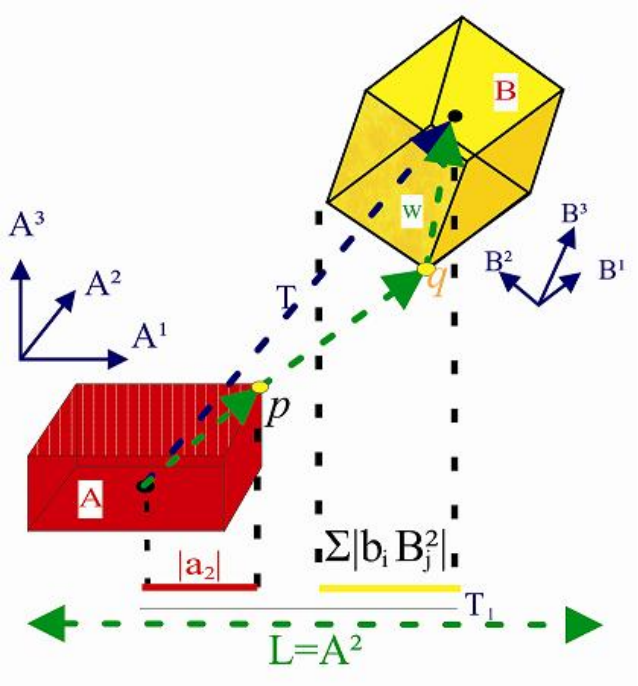


Figura 12 – Projeção de OBB's tridimensionais no eixo A^2
(Gottschalk *et al.*, 1996:9)

Vetores Normais	Produtos Vetoriais
A^1	$A^1X\ B^1$
A^2	$A^1X\ B^2$
A^3	$A^1X\ B^3$
B^1	$A^2X\ B^1$
B^2	$A^2X\ B^2$
B^3	$A^2X\ B^3$
	$A^3X\ B^1$
	$A^3X\ B^2$
	$A^3X\ B^3$

Tabela 2 – Definição dos possíveis eixos de separação entre duas OBB's

Como pode ser observado na figura 13, o ponto central das OBB's são os pontos médios das projeções no eixo testado. Com isso facilmente observa-se que para haver separação entre duas OBB's a soma da largura da projeção dos raios dos VE's no eixo testado deve ser menor do que a largura da projeção do vetor que liga os pontos centrais das OBB's neste mesmo eixo, originando a notação:

$$(VIII)\ |T \bullet L| > r_A + r_B \tag{3.7}$$

Sendo:

r_A e r_B: projeção dos raios das OBB A e B no eixo L

L: Eixo de projeção a ser testado

T: Vetor formado pelos pontos de Centro das OBBs

Por sua vez, o cálculo dessa projeção dos raios das OBB's A e B no eixo L são dadas pelas equações (Gotschalk, 2000:75):

$$r_a = \sum_{i=1}^{3} | aiAigL | \tag{3.8}$$

$$r_b = \sum_{i=1}^{3} | biBigL | \tag{3.9}$$

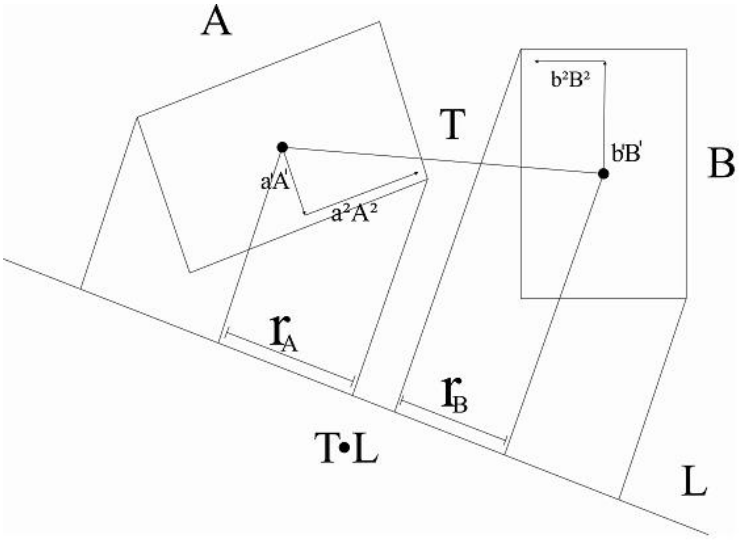


Figura 13 – Eixo de separação entre duas OBBs
(Gotschalk, 2000)

Devido à distribuição Gaussiana utilizada na definição dos parâmetros de tamanho e orientação da OBB o ajuste é o melhor obtido comparando-se às AABB's e esferas, o que diminui a ocorrência de falsas colisões, e conseqüentemente a quantidade de testes realizados posteriormente. Porém, apresenta um custo de armazenamento e de operação computacional no cálculo de colisão mais elevado em relação a esses mesmos envoltórios (Nakamura. 2005:2).

3.5.4 k-DOPs (*Discrete Orientable Polytopes*)

Este tipo de volumes envolventes são uma generalização das AABBs, eles podem envolver um modelo de maneira mais ajustada do que as AABBs usando eixos adicionais e k hiperplanos no total, definidos por normais de orientações discretas, daí o nome popular k-DOP's. A figura 14 mostra o VE com 6,14, 18 e 26 DOPs. Uma AABB, portanto, é equivalente a uma 6-DOP. Os DOPs são definidos por k/2 pares de mínimos e máximos valores em k direções.

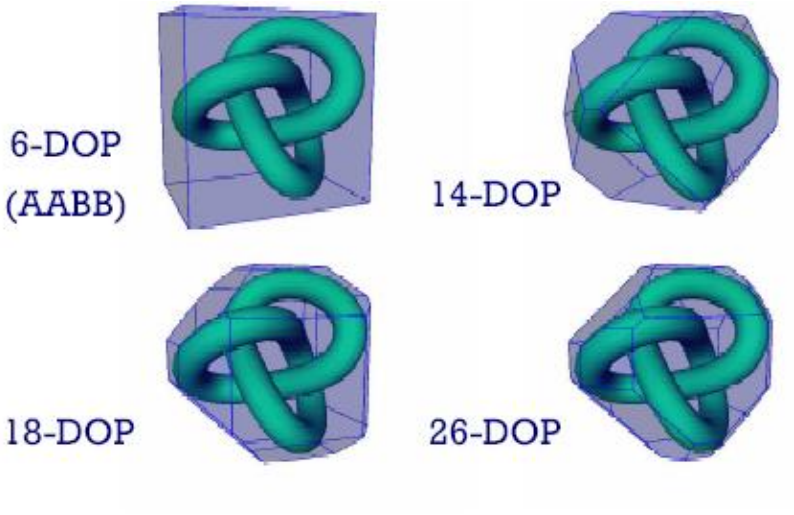


Figura 14 – Diversos tipos de VE's K-DOP's
(Mezger, 2005:17)

Diferentemente das esferas, entretanto, as k-DOPs não são invariantes à rotação, elas devem ser novamente computadas cada vez que o modelo limitado sofre uma rotação, como mostra a figura 15 em três posições de rotação de um objeto. Não obstante, isto pode ser feito de forma razoavelmente rápida, computando simplesmente os intervalos da projeção dos objetos sobre os eixos. Os intervalos de projeção podem ser computados diretamente usando o intervalo de projeção para um eixo v do objeto A é dado pela equação 3.10 (Atencio, 2005:34).

$$[v.\ S_A\ (-v);\ v.\ S_A(v)] \qquad (3.10)$$

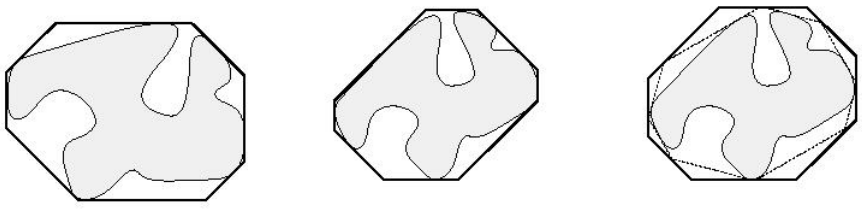


Figura 15 – K-DOP em adaptação a rotação de um objeto
(Klosowski, 1998:10)

O método SAT pode facilmente ser usado para verificação de colisão entre k-DOP's (Lin e Manocha, 2003:8), explanado na seção 3.5.3. A colisão desse sistema é

verificada através da projeção dos envoltórios em k/2 eixos de separação, duas K-DOP's colidem se e somente se suas projeções nos k/2 eixos se interceptam.

Dessa forma, dependendo da magnitude do valor k, o teste de colisão das K-DOP's é normalmente menos custoso que o teste das OBB's, pois o número de eixos testados é menor (Klosowski, 1998:5). Isto acontece porque não existe a necessidade da construção dos vetores normais entre as arestas de duas DOP's de uma aplicação, pois todas elas possuem as mesmas orientações e número de eixos, diferente das OBB's que rotacionam com o objeto envolvido não possuindo ângulos discretos na formação de suas faces e possibilitando a diferente orientação de dois envoltórios na mesma aplicação.

Pelo mesmo motivo as OBB's também apresentam um melhor ajuste ao objetos envolvidos, pois, diferente das DOP's, elas não variam sua estrutura na rotação do objeto. As DOP's apenas se adaptam, podendo não obter um ajuste tão bom para a nova posição do objeto envolvido.

3.5.5 Envoltórios Convexos *(Convex hull)*

A idéia principal na utilização de envoltórios convexos é tornar poliedros não-convexos em modelos convexos para simplificação dos cálculos dos testes de colisão. A figura 16 representa um *convex hull* para um objeto côncavo.

Figura 16 – Envoltório Convexo
(Mezger, 2005:22)

Esta conversão é realizada pela formação do novo poliedro através de algoritmos utilizados em teoria dos grafos. Com base nos vértices que compõem o objeto é calculado o fecho convexo sobre esses pontos, Seidel (1991) mostra que pode ser atingida a complexidade de O(*n*log*n*) com algoritmos de Graham.

Técnicas de programação linear são utilizadas para detecção de colisão destas estruturas, podendo resolver a colisão com complexidade de O(d!*m*), sendo d o número de dimensões e *m* o número de restrições, dadas pelas superfícies convexas (Seidel, 1991).

3.6 Hierarquia de Volumes Envolventes

Uma Hierarquia de Volumes Envolventes (HVE) é a utilização de uma árvore como uma estrutura de dados para representar uma hierarquia de volumes envolventes que compõem um objeto. A HVE é baseada em dois tipos de nós: internos e folhas. Cada um possui um volume envolvente (VE) associado. Nós folhas possuem VE's que envolvem somente a geometria (ou seja, a malha de triângulos) enquanto nós internos possuem volumes VE's que envolvem todos os VE's de seus filhos. A figura 17 demonstra um objeto representado por uma hierarquia de três níveis em uma árvore de esferas.

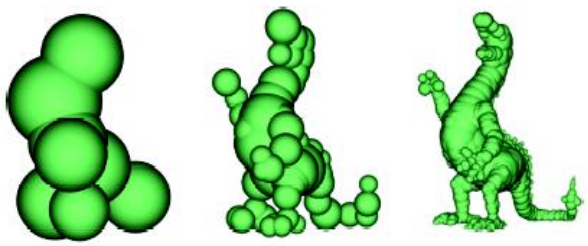


Figura 17 –Dragão em três níveis diferentes da hierarquia de esferas.
(Bradshaw e O'Sullivan, 2004:2)

A utilização da HVE serve para simplificar a topologia do objeto, através da representação aproximada de sua superfície e decomposição do espaço ocupado, com o objetivo de reduzir o número de pares de objetos ou primitivo testadas. A vantagem desse tipo de representação é que em muitos casos a situação de não colisão pode ser facilmente detectada nos primeiros níveis da hierarquia e o refinamento da representação é necessário apenas nas regiões onde a colisão ocorrer (Jimenez.

2002:278). Por outro lado, sua desvantagem reside no consumo extra de construção e de armazenamento.

O teste de colisão é realizado por uma avaliação transversal dos HVE's de dois objetos. Nesta avaliação os modelos são comparados recursivamente e por nível. Se existe colisão entre os VE's raízes que envolvem os objetos, o volume A será comparado com os filhos do nó volume B. Para cada comparação, não havendo colisão, o teste acaba para essa parte da recursão. Senão, um novo teste recursivo deve ser feito desta vez entre o filho de B colidente contra os filhos de A e assim segue recursivamente até que os volumes colidentes sejam folhas das árvores, então deverão ser testadas diretamente as primitivas contidas nesses VE's (Gotschalk, 2000). Dessa seqüência de testes surge uma nova árvore, chamada Árvore de Testes de Volumes Envolventes (ATVE).

As figuras 18 e 19 representam as árvores dos objetos A e B, respectivamente. A divisão de cada uma delas é realizada em três níveis de detalhamento, onde no último nível cada folha representa uma única aresta.

A figura 20 detalha a construção da ATVE a partir dos volumes envolventes que representam as árvores das figuras 18 e 19. Ali é realizada a verificação da colisão entre os dois objetos.

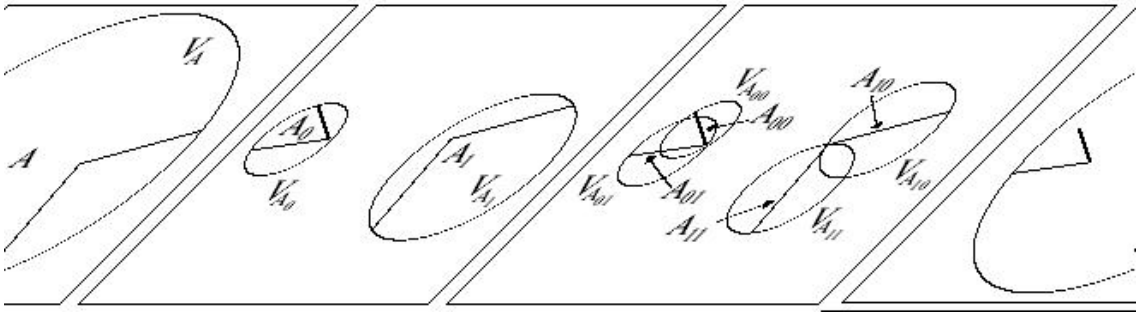


Figura 18 –Representação de três níveis da HVE do objeto A.
Gottschalk (2000:16)

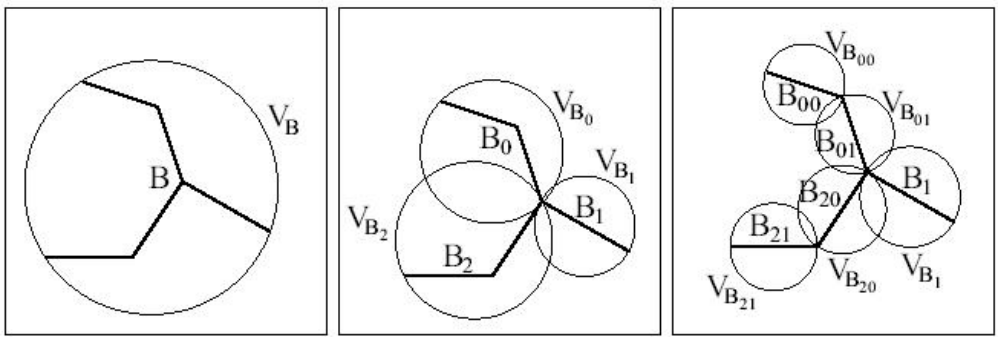


Figura 19 –Representação de três níveis da HVE do objeto B
Gottschalk (2000:16)

Verificada a colisão entre os primeiros níveis da árvore, o teste é agora realizado entre os filhos de A e a raiz de B. Cada VE nível 2 colidente de A é agora verificado contra os filhos nível 2 da raiz de B, assim o teste segue sucessivamente, até que uma não colisão seja encontrada ou quando o algoritmo atingir a detecção de colisão entre folhas, situação ocorrida na problemática abordada pela figura. O pseudo-algoritmo demonstrado no quadro 9 implementa estes testes e transmite uma visão diferente deste procedimento.

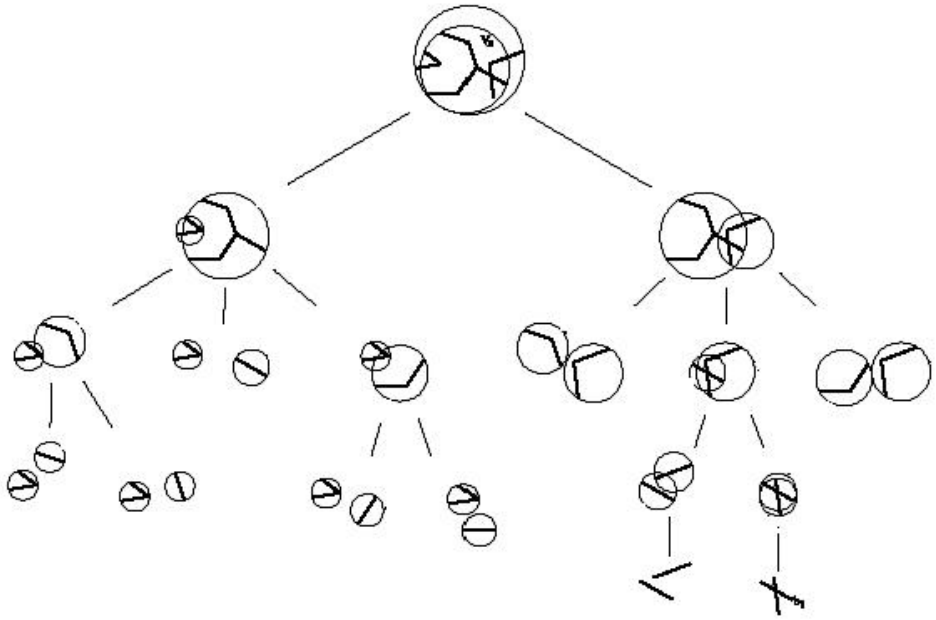


Figura 20 –ATVE para os testes de colisão entre os objetos A e B.
(Gotschalk 2000:25)

A construção da HVE pode ser manual, montada no momento da modelagem do objeto, sendo que a estrutura acompanhará o objeto representado durante as interações. As técnicas para montagem são agrupadas em duas categorias:

- De cima para baixo: Dado um conjunto de polígonos, ajusta-se um volume à coleção e, recursivamente, subdivide o conjunto em dois ou mais grupos, para cada grupo ajustando um novo VE até que um limite definido seja atingido (Gottschalk, 2000: 39). Esse limite pode ser o nível alcançado na árvore ou o número de primitivas que a folha contém. Uma técnica de representação espacial que se enquadra nessa categoria é a árvore K-d, a ser demonstrada nas próximas seções (Hofmam, 2002: 31).
- De baixo para cima: Os conjuntos de polígonos atômicos do objeto são sucessivamente agrupados, criando um pai, até que se chegue a um único grupo que envolve todo o objeto. As R-Tree (discutidas na seção 3.7.2) encontram-se nesta categoria (Hofmam, 2002: 31).

3.7 Estruturas de Decomposição Espacial

Malhas uniformes (*grids*), *octrees*, *R-trees, BSP-trees e* árvores K-d são exemplos de representação de decomposição do espaço. A decomposição espacial é realizada através da divisão do espaço ocupado e, dessa forma é necessária apenas a checagem entre os pares de objetos (ou partes) que estão na mesma célula ou em células próximas (Atencio, 2005:30). Essa estrutura é em geral utilizada como estrutura secundária de representação no processo de detecção de colisão.

Todo o método de decomposição do espaço visa explorar a coerência espacial. Em particular, a decomposição do espaço pode ser utilizada nas duas fases do processo de detecção de colisão (Taddeo, 2004:22), na *broad phase* para realização do teste de colisão a partir das listas de objetos contidos em suas células ou para *narrow phase* realizando a divisão espacial para construção de hierarquias de volumes envolventes.

3.7.1 - Grids

Grids de envoltórios cúbicos decompõem o ambiente 3D (representado como um cubo), em células cúbicas (nxnxn) de igual volume (*voxels*). Cada *voxel* contém uma lista de objetos que pertencem àquela região (estão contidos), ver figura 21, sendo interessante ressaltar que um objeto pode estar contido em diversos *voxels* o que pode acarretar em uma elevada utilização de memória (Requincha, 1980:447). Um algoritmo de força bruta é utilizado para computar o conjunto de pares que se interceptam. A principal vantagem da utilização da decomposição espacial por *grids* é que a estrutura de dados associada é estática, não sendo necessária a constante atualização da mesma.

A variação do número de *voxels* influencia diretamente no grau de precisão e desempenho dos testes de detecção de colisão. Quanto maior o tamanho do voxel, mais objetos ele irá conter, conseqüentemente maior será o número de testes falsos positivos de colisão, o que contradiz o objetivo da utilização dessa representação espacial. Por outro lado, quanto menor o *grid,* menor será a lista de objetos de cada voxel, porém aumenta o número de estruturas que precisarão ser analisadas (Taddeo, 2004:34). Portanto, o balanceamento do tamanho dos *voxels* é fundamental para a eficiência do algoritmo.

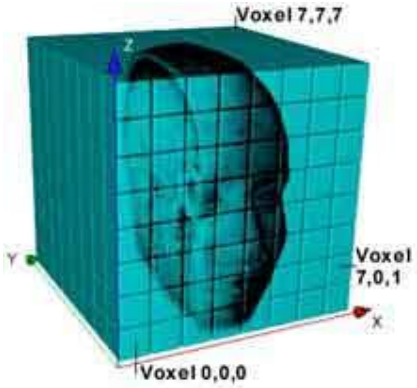


Figura 21 – Exemplo de divisão espacial por grids
(Volume Graphics,2005)

3.7.2 - R-trees

A representação por *R-tree* é uma estrutura de dados hierárquica capaz de representar a decomposição do espaço segundo a localização espacial dos objetos dispostos em cena.

Segundo Figueiredo e Fernando (2003:2), a *R-tree* é utilizada para a construção de um grafo de cena, hierarquizando os objetos contidos no ambiente. A raiz corresponde ao ambiente da cena, e os filhos são as regiões de agrupamentos de objetos que podem se subdividir em novas regiões. Os objetos correspondem às folhas dessa árvore, e têm como pai alguma região do ambiente representada na árvore. A figura 22 demonstra essa partição espacial do ambiente realizada em quatro níveis da árvore.

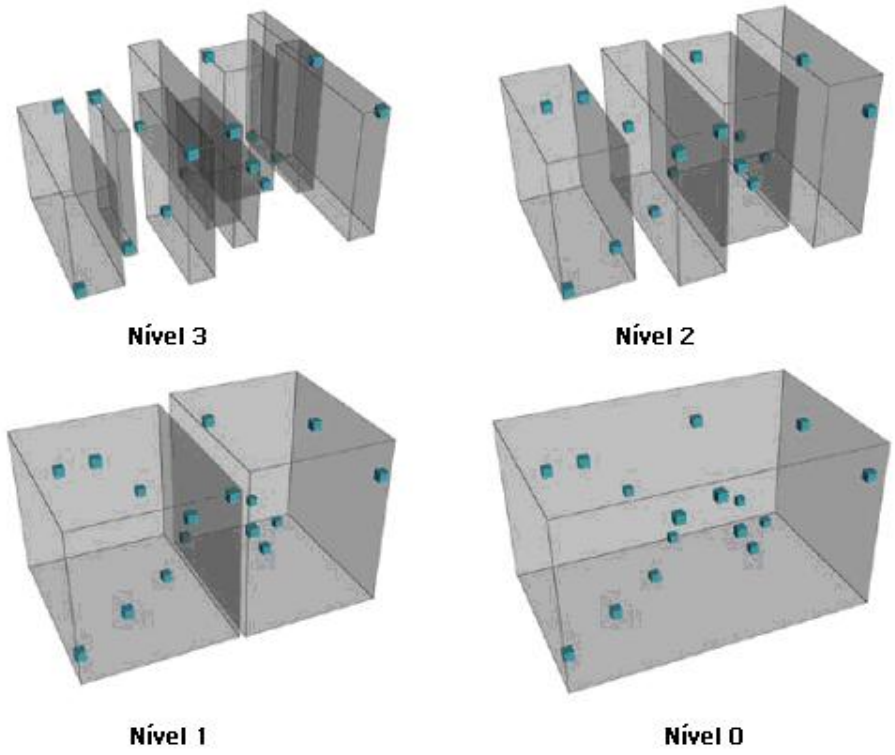


Figura 22 – Decomposição espacial representada pela *R-tree*
(Hofmam, 2002: 38).

A estrutura de dados *R-Tree* é uma arvore n-ária em que os nós não folhas têm entre τ e σ filhos ($\sigma \geq 2\tau$), sendo que esses valores correspondem ao número máximo e

mínimo, respectivamente de filhos por nó. Em uma *R-tree* todas as primitivas aparecem no mesmo nível. A profundidade de uma árvore que armazena *n* primitivas é de O(log*n* /logτ). Sendo assim, esta técnica apresenta um bom desempenho para aplicações que permitem navegação do usuário no ambiente de RV através de dispositivos de entrada e saída - ambientes *walk-through* (Taddeo. 2004:28).

A principal idéia da técnica é explorar a coerência espacial do ambiente para detecção de colisão, realizando testes apenas em objetos vizinhos, filhos do mesmo nó, eliminando comparações entre objetos que se encontram distantes (Figueiredo *et al*., 2003:2). Pode ser utilizada tanto na *broad phase pela* comparação entre os objetos, localizados em sub-regiões do espaço ou na *narrow phase através da* construção da hierarquia de VEs dos objetos.

3.7.3 – Octrees

A octree é uma estrutura de dados utilizada para representar objetos 3D sob a forma de uma árvore que é construída através da subdivisão recursiva em oito partes do espaço. Quando a *octree* é gerada, o espaço do ambiente é subdividido em oito octantes, através da utilização de planos mutuamente perpendiculares ao eixo cartesiano (no caso de utilização de caixas), cada célula pode assumir três estados: cheia, parcialmente cheia e vazia.

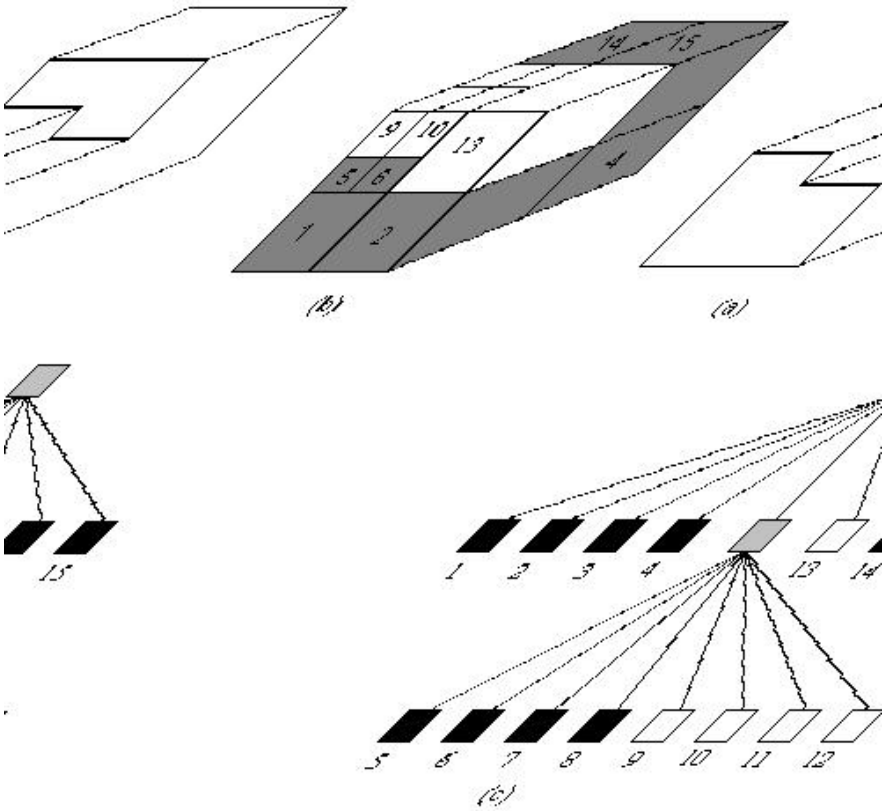


Figura 23 – Representação por Octress de um sólido.
(a) Sólido. (b) Divisão recursiva do sólido. (b)Representação da árvore gerada.
(Nascimento, 2001)

Posteriormente, cada octante misto, é recursivamente subdividido em oito novos octantes e assim, sucessivamente, até que não mais existam octantes mistos ou que se atinja um nível de iteração, resultando, das células mistas o conjunto de pares candidatos à colisão (Taddeo, 2004:40). A figura 23 ilustra a representação da decomposição espacial realizada pelas octrees em um sólido, porém, no exemplo as células mistas são formadas por espaços em branco e alguma parte do objeto.

A construção realizada no nível de cada nó garante uma boa representação do espaço livre, porém por depender indiretamente da posição e orientação dos objetos a octree possui limitação de profundidade. Podendo assim, perder precisão na representação do objeto, bem como demandar de muita memória se não houver a definição do número de iterações, e conseqüentemente, do nível de precisão almejado (Taddeo, 2004:24).

3.7.4 - Árvore BSP

A *BSP-tree* (*Binary Space Partition*) divide o espaço tridimensional recursivamente em pares de subespaços, separados por um plano de posição e orientação arbitrário. Cada nó interno da árvore possui dois nós filhos, um para cada lado do plano. Se o subespaço gerado for homogêneo, seu nó filho será uma folha representando a região que contém certo número de objetos. Uma variante popular da BSP são as árvores K-d, onde os eixos são restritos a orientações perpendiculares aos eixos cartesianos (Atencio, 2005:32). A figura 24 mostra a decomposição espacial no ambiente realizada por uma árvore e a estrutura de dados correspondente, os objetos selecionados para o teste de colisão serão os que se encontrarem na mesma folha da árvore.

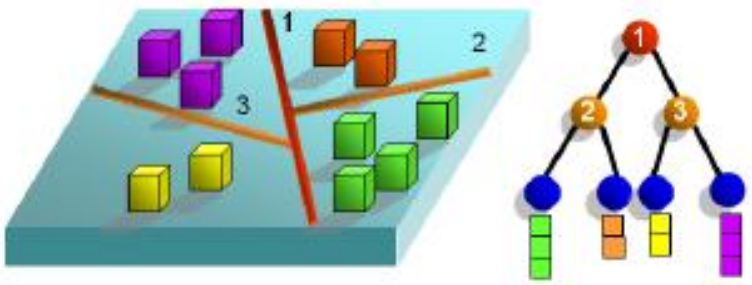


Figura 24 –Decomposição Espacial por uma BSP-tree
(Luque *et al.,* 2006:2)

Na construção da árvore, a melhor partição será a que possuir a melhor reposta para os seguintes parâmetros que avaliam a qualidade de uma partição p (Luque *et al.*, 2006:3).

- População(p): Número de objetos testados contra p;
- Balanceamento(p): taxa de objetos, dada pelo número de elementos do subespaço com menos objetos sobre o número de objetos do subespaço com mais objetos. Representa como os objetos estão distribuídos nas duas sub-árvores do nó;

- Redundância(p): Pode acontecer o plano de separação intersectar objetos. Quando isso ocorre temos que o objeto fica alocado em ambos os lados da partição, provocando redundância.

Uma forma de realizar a partição é pela projeção normal dos objetos do espaço a ser subdividido no eixo normal ao hiperplano que o dividiu mais recentemente. Os valores das projeções são armazenados em uma lista ordenada. Esta lista será usada para selecionar num intervalo apropriado entre todos candidatos, um ponto de referência que maximiza o balanço enquanto minimiza a redundância (Luque *et al.*,2006:3).

Segundo Luque *et al.* (2006), esta é uma boa alternativa para ambientes de larga escala, que contêm um grande número de objetos. Contudo, a realização de partições bem localizadas é um fator fundamental para o bom desempenho de um algoritmo de detecção de colisão.

Entretanto, algumas limitações são consideradas quando da sua utilização. Por exemplo, por envolver a constante reconstrução da árvore possui problemas para trabalhar satisfatoriamente com ambientes dinâmicos, consome considerável memória, não trabalha com objetos deformáveis (pois a deformação gera uma reconstrução da árvore, o que demanda elevado gasto de tempo) e não é uma árvore balanceada (o que pode gerar impacto negativo no desempenho da aplicação) (Figueiredo *et al*, 2002).

3.7.5 - Árvore K-d

A estrutura de particionamento de dados k-d é uma árvore de busca binária *d*-dimensional que representa uma subdivisão recursiva do universo em subespaços por meio de hiperplanos (*d*-1)-dimensionais. Os hiperplanos são sempre orientados de acordo com o eixo cartesiano, assim, em um espaço tridimensional os hiperplanos separadores são alternadamente perpendiculares aos eixos *x*, *y* e *z* (Hofmam, 2002: 32).

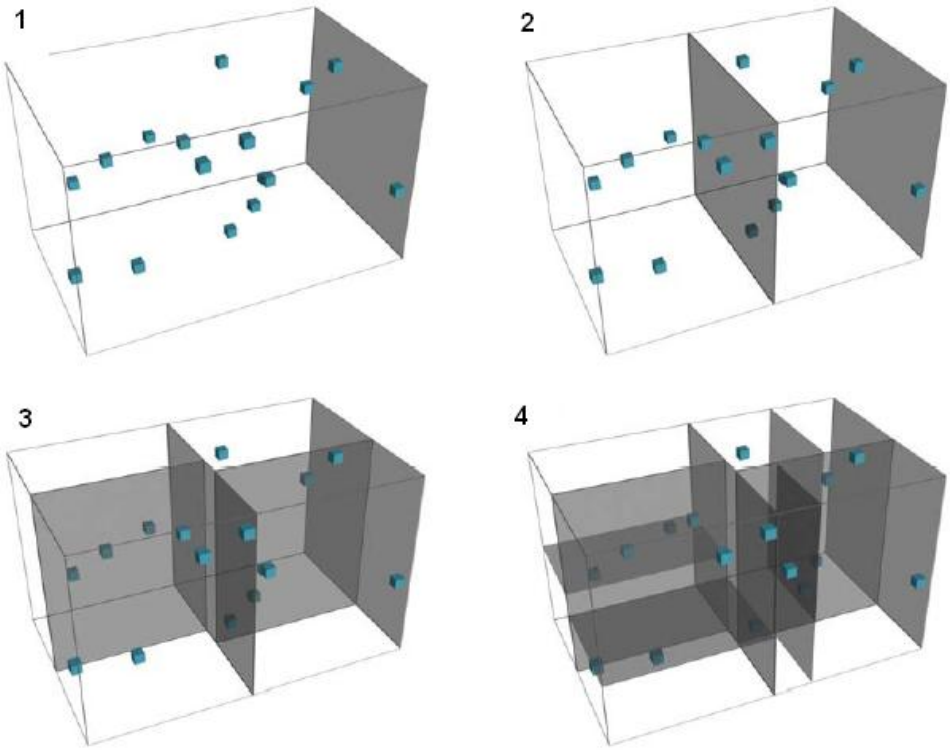


Figura 25 – Decomposição espacial representada pela árvore *K-D* (Hofmam, 2002: 38).

Por ser um tipo de árvore BSP, a construção dos planos de particionamento de uma k-d pode ocorrer na mesma metodologia de execução para árvores BSP. Mas também um modo conveniente de realizar essa partição é posicionar o hiperplano sobre a mediana do conjunto objetos contidos no espaço a ser dividido(Hofmam, 2002:33). A mediana é uma medida estatística que divide um conjunto de dados em duas partes com igual número de elementos. Dado um conjunto de objetos ordenados C={c1<c2...<cn}, a mediana m_c desse conjunto é dado pelo elemento central, se n for ímpar e pela média dos pontos intermediários, se n for par. A figura 25 mostra a divisão espacial de um ambiente tridimensional baseado em árvores k-d repartidas em quatro níveis pela mediana.

3.8 Varredura e Corte (*Sweep and Prune*)

Esta é uma abordagem para *broad phase* completamente diferente do particionamento espacial. Esquemas de particionamento de espaço exigem

considerável esforço para encontrar bons tamanhos de partição, porém não será o tamanho da célula o fator que elimina pares de objetos idealmente como no teste de VE's. Além do mais, não trabalham bem para ambientes que possuem uma grande escala na quantidade de objetos (Cohen *et al.,* 1994:9).

Um algoritmo de *Sweep and Prune* realiza a varredura dos pares de volumes envoltórios AABBs que se interceptam formando a lista de pares candidatos a colisão. Para isso é realizada uma redução dimensional, se dois corpos colidem no espaço tridimensional através de suas AABBs então suas projeções ortogonais nos planos xy, yz e xz e nos eixos x, y , z também devem colidir, conforme demonstrado na figura 26. Uma lista é criada para cada eixo, as inserções dos pontos máximos e mínimos de cada envoltórios são inseridos ordenadamente através de um algoritmo de ordenação. Na seqüência, intersecções são buscadas em cada lista na ordem em que se encontra, uma colisão acontecerá se dois objetos se interceptarem nas três listas dos eixos(Cohen *et al.,* 1994:).

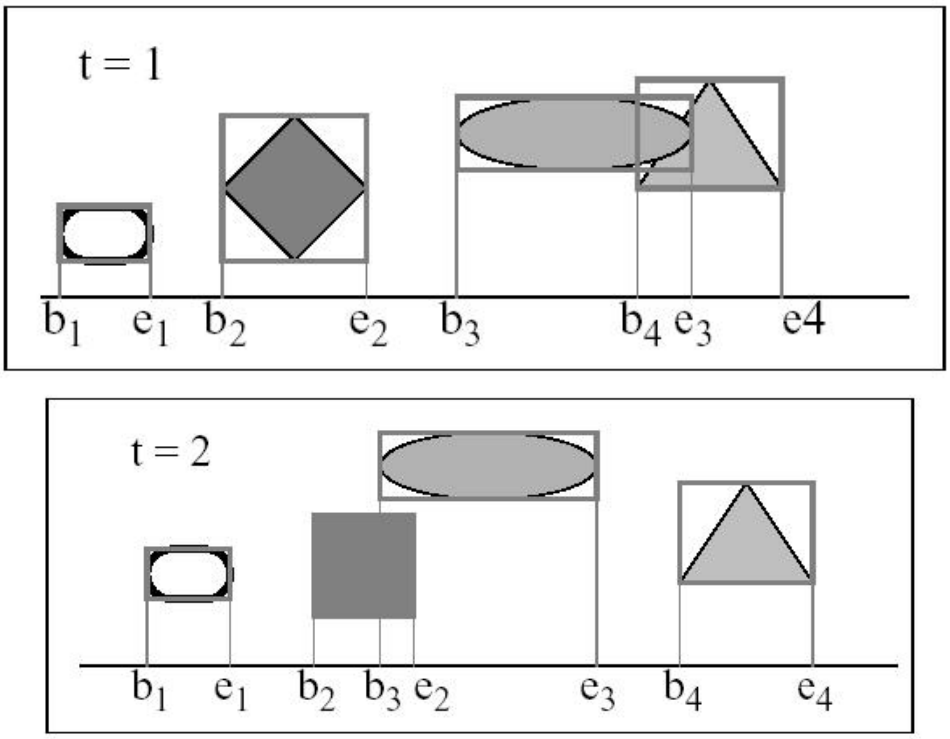


Figura 26– Projeção de AABBs no espaço 1D em dois intervalos de tempo distintos (Cohen e*t al.,* 1994: 13)

4 TRABALHOS RELACIONADOS

Este capítulo mostra que mesmo simuladores de robôs articulados renomados carecem de recursos de detecção de colisão. Este recurso é bastante custoso computacionalmente principalmente no contexto da tecnologia VRML onde se tem os cálculos executados em Java e na máquina do usuário (cuja configuração é difícil de prever). Ainda o capítulo aborda algumas pesquisas recentes sobre detecção de colisão, mas que se mostraram inadequadas às restrições da presente aplicação. Contudo trazem contribuições importantes para esse trabalho.

4.1 – Robô Virtual Java/VRML2.0 sem Detecção de Colisão

Atualmente existem projetos muito reconhecidos, como o VRML 2.0 Robot, que possui características semelhantes ao SCORBOT Virtual. Esse projeto se constitui de um *web-browser* que incorpora um modelo interativo em VRML 2.0 dos robôs MANUTEC, KUKA KR6 ou PUMA 560 (ver figura 27), possui interface implementada em JAVA. O VRML KUKA KR6 atraiu muita atenção e foi condecorado com prêmios como *Pirelli Internacional*, *Cool Robot of the Week* da *Nasa Telerobotics* e *Multimedia Transfer*. Porém, apesar de ser um projeto bem conceituado, o VRML 2.0 Robot não apresenta detecção de colisão.

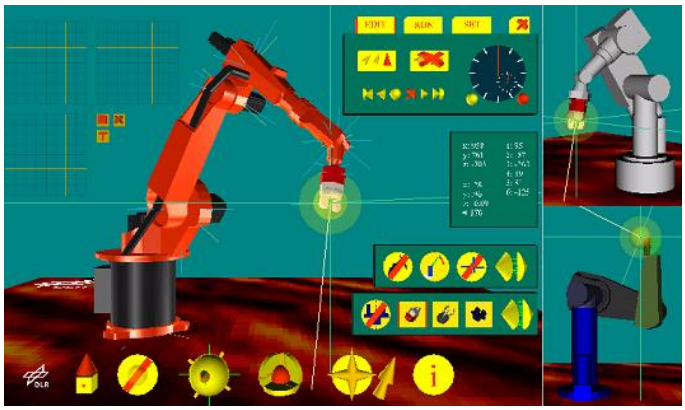


Figura 27 – Robô Manipulador Interativo VRML 2.0
(Rohrmeier, 2000)

4.2 – Detecção Discreta com uso de Octrees Esféricas

Nascimento(2001) desenvolveu um algoritmo que se propõe a realizar uma detecção de colisão entre objetos convexos polimórficos aplicando-se, inclusive, na área de robótica. O sistema é desenvolvido em três fases:

1 – Fase de detecção de colisão entre Envoltórios Esféricos – *Broad Phase*

Utiliza envoltórios esféricos nos objetos para enumerar os pares de objetos colidentes no espaço, encaminhando essa lista para a próxima fase.

2 – Fase de Identificação dos Prováveis Pontos de Colisão – *Narrow Phase*

Esta fase recebe a lista de pares de objetos (A, B) selecionados na fase anterior. Neste ponto da aplicação, para diminuir o esforço computacional, apenas um dos objetos do par possivelmente colidente tem seu envoltório esférico dividido através da aplicação de *octress* esféricas (Figura 28) de modo a eliminar as partes do objeto que não se encontram na rota de colisão e localizar os prováveis pontos de colisão(P_{Pc}). A figura 28 mostra como é feita a partição por meio de octrees esféricas, ali são demonstrados 4 níveis da hierarquia.

As esferas candidatas à colisão subdivididas pela *octree* devem estar integralmente contidas no envoltório objeto subdividido. As esferas que estiverem parcialmente contidas no volume do envoltório de B (E_B) sofrerão novas subdivisões para encontrar novos $P_{Pc,}$ caso os encontrados nessa fase não sejam validados, na fase seguinte as demais serão descartadas por não oferecerem risco de colisão.

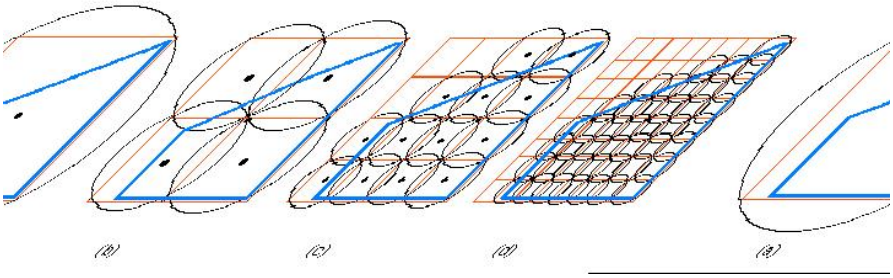


Figura 28 – Visualização em 2D da subdivisão por octrees Esféricas (Nascimento 2001:25)

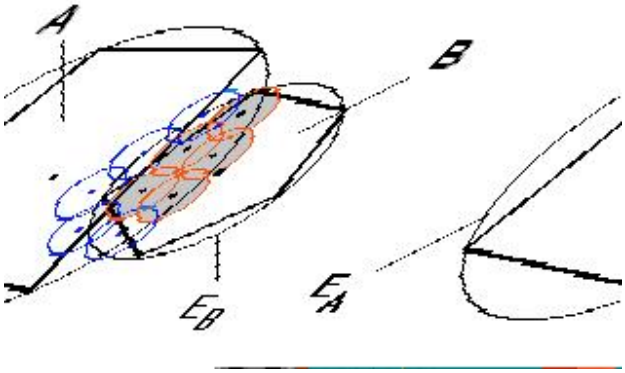


Figura 29 – Visualização em 2D dos prováveis pontos de colisão
(Nascimento 2001:35)

3 – Fase de Validação da Colisão

Consiste em identificar se existe um ponto de colisão real entre os espaços selecionados. Dividida em duas subfases:

a) Abordagem analítica. É realizada uma análise da posição relativa do ponto P_{pc} em relação à superfície de cada face do objeto que sofreu a decomposição por octree. Através do vetor normal e de um ponto contido em cada face de B é avaliado se o ponto central das esferas selecionadas na fase anterior pertence à face, para isso a equação dessa superfície (4.1) é aplicada e validada pela condição $F_{i=0...n}(P_{pc}) \leq 0$, a figura 30 mostra os componentes dessa avaliação.

$$F(x,y,z)=ax+by+cz-(ax_0+by_0+cz_0) \quad (4.1)$$

Sendo:

Vetor normal = Vn(a,b,c)

Ponto pertencente à face = (x_0, y_0, z_0)

Número de faces do objeto A = n

Ponto em possível colisão(centro da esfera) = P_{pc}

Caso $F_i(P_{pc})>0$ então a esfera deve voltar a fase anteriores para uma nova subdivisão caso contrário o ponto P_{pc} passará para a próxima fase de validação.

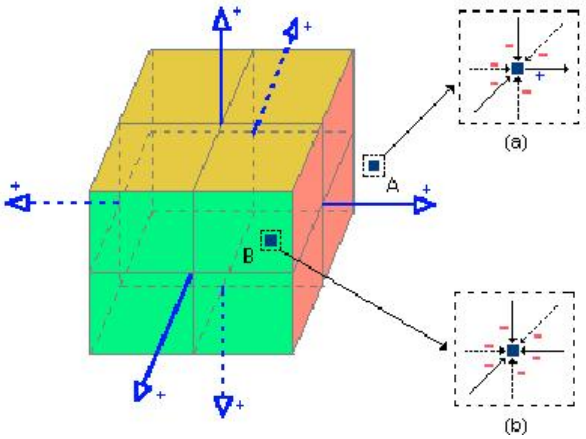


Figura 30 – Validação dos pontos P_{pc},
(a) Pertence ao objeto, validado. (b) Não pertence ao objeto, não validado.
(Nascimento 2001:37)

b) Realizada a análise da posição relativa do P_{pc} com as faces do outro objeto (no caso o objeto A). Se o ponto não pertencer a esse conjunto de faces, então ele volta a fase 1, reiniciando o processo de octree pois houve colisão apenas entre o objeto B e o $E_{A.}$

Este algoritmo recursivo permanecerá detectando colisões até que ocorra uma real colisão e/ou o número de iterações (precisão) seja atingido.

A principal característica desse trabalho é o emprego de octrees para construção de uma HVEs esféricos e a abordagem analítica para validar se um ponto pertence a um objeto ou não. Contudo, conhecendo a característica de ajuste das esferas, abstraí-se que não é o melhor envoltório para aplicação no modelo de robô articulado de membros em forma de barra. Ainda, os resultados da análise do desempenho do algoritmo demonstrado pelo autor mostra que com mais de vinte objetos em cena a sensação de realismo da cena diminui devido ao movimento segmentado dos objetos. Isso se explica pelo grande esforço computacional do algoritmo nesse cenário.

4.3 – Detecção Discreta com uso de OBB-Trees

Nakamura (2005) apresenta uma biblioteca para detecção hierárquica entre objetos de cenas 3D que permite a definição do nível de precisão desejável na

detecção de colisão, fator que facilita o equilíbrio entre o desempenho e precisão desejado por cada aplicação.

O trabalho faz uso de técnicas de OBB-tree, que é uma hierarquia de VE's OBB. Em um primeiro nível, a hierarquia é ligada aos volumes envolventes dos objetos da cena. Esse é o primeiro tipo de consulta possibilitado pela biblioteca. É um teste rápido, porém, impreciso. O segundo nível de testes faz a detecção entre as folhas da OBB-tree associada a cada objeto colidente. A aplicação permite ao usuário a escolha do número de triângulos associados a cada folha, dessa forma pode-se aqui variar a precisão e conseqüentemente o custo computacional do teste, conforme observado na tabela 3, onde foram testadas a precisão na detecção de colisão em um modelo com 276.584 triângulos. Essa tabela mostra o desempenho do algoritmo com as seguintes configurações:

Sem detecção de colisão;

Com detecção de colisão apenas entre os VEs dos objetos;

Detecção de colisão contando com os VE's folhas contendo 1, 5 e 10 primitivas;

Detecção entre os triângulos das folhas colidentes da configuração 3.

O último nível de testes oferecidos tende a ser o mais preciso possível. O algoritmo determina se há colisão entre as hierarquias dos volumes envolventes, testa se os nós da OBB encontram-se em colisão e então faz o teste de intersecção entre os triângulos associados às folhas em colisão das OBB-trees.

Precisão	Quadros por segundo
Sem Colisão	496
Vol. Envolvente. dos Objetos	471
VE das Primitivas (1)	172
VE das Primitivas(5)	450
VE das primitivas(10)	455
Triângulos(1)	105
Triângulos(5)	108
Triângulos(10)	81

Tabela 3 – Resultados do desempenho do algoritmo (Nakamura 2005: 5)

A biblioteca se mostra uma boa ferramenta para ambientes interativos e traz uma contribuição interessante na escolha do número de primitivas que podem

ser associadas aos nós folhas. Essa pode ser uma alternativa interessante para o aumento da velocidade de um algoritmo vinculado a uma aplicação disponível na WEB, porém para a funcionalidade da aplicação em estudo nesse trabalho permite a exploração de técnicas mais eficientes para a *broad phase*.

4.4 – Detecção Discreta com uso de *R-Trees* e AABB

Figueiredo e Fernando(2003) propõem um algoritmo para detecção de colisão em superfícies. Ele realiza as duas fases de diagnóstico de colisão baseadas em estruturas *R-tree* e envoltórios volumétricos AABB. A técnica tira proveito do conceito de coerência espacial proporcionado pela estrutura de *R-trees.* O método é implementado em três fases que seguem:

1 - Filtro para objetos: esta fase resolve a etapa de *broad phase* do problema. O grafo da cena é representado por uma estrutura de dados *R-tree* onde cada folha armazena um objeto de uma vizinhança, aproveitando da propriedade de coerência espacial são testados apenas os objetos que se encontram na mesma subregião (vizinhança) da R-Tree.

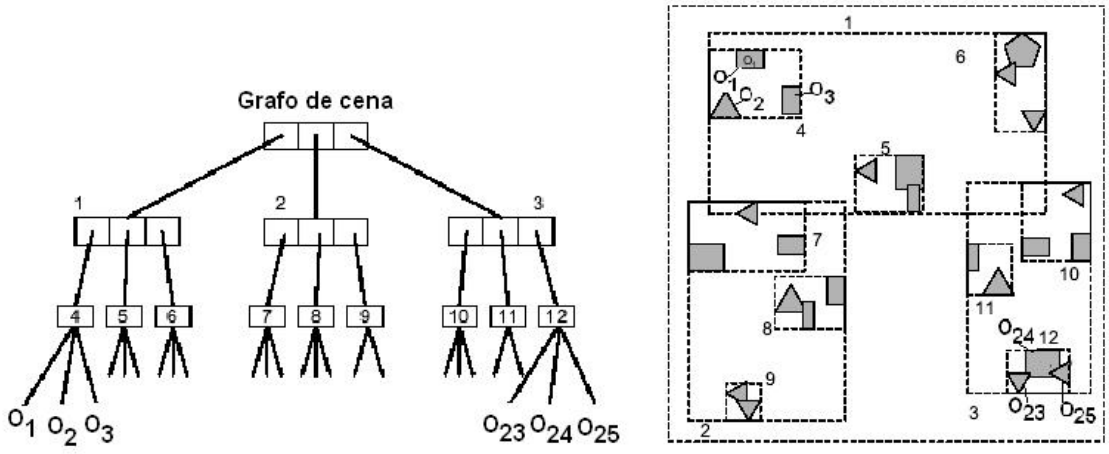


Figura 31 – Grafo de Cena do ambiente 2D particionado por R-Tree.

A figura 30 mostra um particionamento do espaço por *R-tree,* a cena é dividida em três grupos (1-3) que por sua vez se subdividem em novas partições para melhor aproveitamento da coerência espacial entre os objetos. Essas

partições são realizadas segundo as regras para o balanceamento estabelecidas na fundamentação teórica desse trabalho. Serão enumerados para teste de colisão apenas os pares de objetos que possuem o mesmo pai no terceiro nível da árvore(4-12), por exemplo, os objetos O1, O2 e O3 que são filhos do nó 4.

O teste de colisão entre os objetos de uma vizinhança é realizado através de envoltórios AABB's associados aos objetos. Se um par de AABB não colidir, então os dois objetos correspondentes também não se interceptam, portando são filtrados. No caso de existir possibilidade de colisão, o algoritmo determina o volume de sobreposição(OAABB – *Overlapping Axis-Aligned Bounding Box*) entre as duas AABB (Figura 31). Essa estrutura será utilizada no próximo passo.

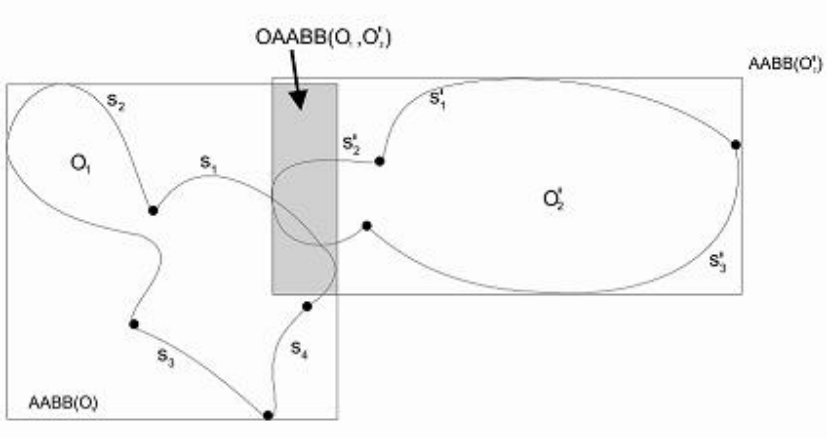


Figura 32 – Conceito de OAABB
(Figueiredo *et al.*,2003: 4)

2 - Filtro de superfícies: esta fase inicia a *narrow phase,* ela recebe a lista de pares de objetos candidatos a colisão e suas respectivas OAABB's. Para atingir um melhor desempenho, o algoritmo utiliza uma AABB para cada superfície s_x,hierarquizando a estrutura. Duas superfícies são candidatas a colisão se suas AABBs correspondentes colidirem com o OAABB relacionada ao par de objetos, situação ilustrada na Figura 32. Esse conjunto de superfícies candidatas a colisão são repassadas ao próximo nível do algoritmo.

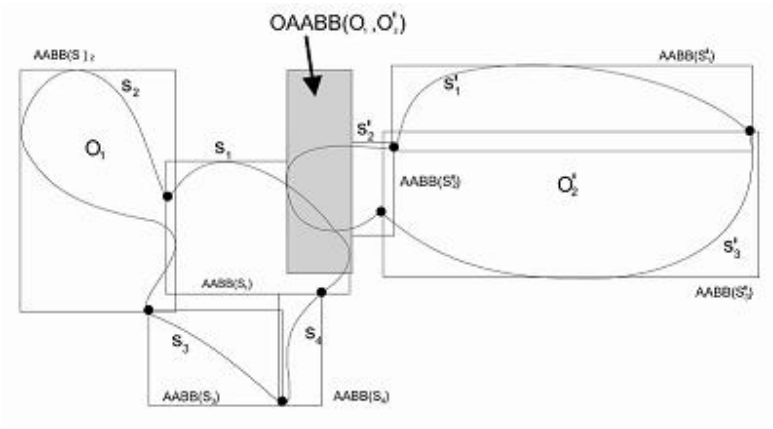


Figura 33 – Processo de filtragem de superfícies
(Figueiredo *et al.*,2003: 4)

3 - Intersecção de polígonos: essa fase faz a determinação exata dos polígonos possivelmente colidentes que compõe a superfície. Nesse passo é calculada uma nova OAABB entre as AABBs das superfícies candidatas, a figura 33 mostra como ocorre essa identificação da nova OAABB. Se as AABBs associadas aos pares de polígonos dos objetos colidirem com a OAABB, então um cálculo de intersecção entre esses polígonos é realizado. O produto dessa fase será o conjunto de pares de polígonos colidentes.

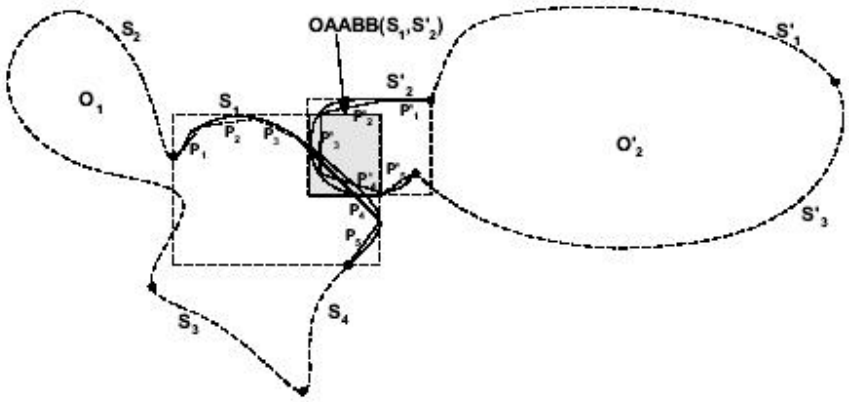


Figura 34 – Processo de filtragem de Polígonos utilizando AABBs e OAABB
(Figueiredo *et al.*,2003: 4)

Esse algoritmo traz como contribuição para a comunidade cientifica a utilização dos OAABBs na realização da *narrow phase* de algoritmos de detecção de colisão de superfícies. O algoritmo utiliza a representação espacial por *R-trees*

que possibilita da exploração da coerência espacial para eliminação de pares colidentes. Ainda, segundo Figueiredo e Fernando(2003), o algoritmo possui um alto grau de paralelismo que pode ser explorado para melhoria do desempenho do método.

A utilização de AABB's não é apropriada ao estudo de caso neste trabalho devido ao mal ajuste que possui para objetos com as características dos encontrado na modelagem do SCORBOT virtual.

5 A SOLUCÃO PROPOSTA

De acordo com o estudo relatado nesse trabalho, uma resposta fornecida por um algoritmo de detecção de colisão contínua fornece a solução ótima por reportar o momento exato em que o contato entre dois corpos ocorre, permitindo assim ao programador evitar a interpenetração após o instante reportado. Contudo, a detecção de colisão contínua também é mais custosa do que a detecção realizada entre espaços de tempo fornecido por métodos discretos. Pois é necessário considerar fatores como informações de velocidade, deslocamento e transformações geométricas ocorridas no caminho para criação de meios que permitam prever a colisão. Isso exige computações mais complexas que em uma consulta estática.

Sendo o ambiente do SCORBOT virtual uma aplicação WEB didática, não é tolerável a perda de tempo para uma resposta exata. Com isso será desenvolvida uma solução construída sob a ótica da abordagem de detecções de colisão discreta, que também possui simples implementação e rápida reposta se comparada aos métodos de detecção contínua.

As consultas de detecção de colisão serão feitas ao final de cada interação com o usuário, quando este solicita que o robô execute alguns movimentos de reposicionamento direto (pelo incremento/decremento de valores de uma determinada junta isoladamente) ou indireto (várias coordenadas de juntas são alteradas devido à cinemática inversa). Ou seja, não há a necessidade a priori de cálculo contínuo, mas, se este se fizer necessário, uma solução discreta poderá ser replicada várias vezes para se obter uma boa solução contínua aproximada.

Será utilizado o VE OBB, pois se conhece *a priori* as geometrias envolvidas (tanto dos objetos da cena quanto dos elos do próprio robô) e pode-se atribuir AABB's aos objetos em suas posições canônicas (originais) e, propagando o posicionamento do objeto na cena à AABB tem-se como resultado a respectiva OBB daquele objeto. Ressalva-se ainda que o VRML permite gerar e manipular uma caixa transparente o que facilita a manutenção deste VE, que não é invariante à rotação.

Com vista à eficiência, vê-se a necessidade da utilização de técnicas que auxiliem na diminuição de testes mais rebuscados. Com esse objetivo, o método de detecção de colisão desenvolvido para o ambiente do SCORBOT Virtual será implementado em dois níveis de complexidade a serem detalhados no tópico seguinte.

5.1 Fase Abrangente (*Broad Phase)*

5.1.1 Coerência Semântica

No estudo das técnicas de *Broad Phase* observou-se que não é considerada como técnica a possibilidade de uma avaliação prévia do ambiente virtual, de suas funcionalidades e significados dos eventos nela contido, como forma de predizer casos certos de não colisão ou situações de maior probabilidade de colisão.

É possível observar no ambiente do SCORBOT virtual que a função desempenhada e limitações inerentes proporcionados pelas características de projeto do modelo fazem com que o contato entre alguns objetos jamais venham a acontecer, ou que, contrariamente, sejam constantemente eminentes.

Assim, essas particularidades levam ao conceito:

"Coerência semântica é o uso do significado relativo
que certos objetos têm em uma cena para auxiliar
a tomada de decisões de otimização".

Portanto coerência semântica é uma técnica de detecção de colisão para fase abrangente que analisa as particularidades de uma cena gráfica com o intuito de prever os pares de objetos que, pelo seu significado e funcionalidades no ambiente, nunca colidirão. Ou, ainda, pares de objetos que devem ser prioritariamente avaliados, pois possuem uma alta probabilidade de colidirem.

Essa coerência no significado dos objetos acaba se mostrando como uma propriedade que pode ser usada como critério de corte no momento da varredura à procura de pares colidentes.

5.1.1.1 - *Coerência Semântica no SCORBOT virtual*

Um estudo detalhado do ambiente robótico traz avaliações decisivas e que influenciam diretamente no desempenho do algoritmo de colisão. No ambiente do SCORBOT virtual podem ser verificadas as seguintes características dos objetos que, combinadas aos significados dos demais corpos contidos no espaço, possuem uma coerência semântica que será útil ao projeto do algoritmo na enumeração de possíveis pares colidentes para a fase da detecção precisa (*Narrow Phase*).

Considera-se, no caso do órgão terminal do robô, a existência da ponta e punho como duas entidades diferenciadas. Quando a garra se encontra livre ela é considerada como a ponta do órgão terminal do robô, porém, ao efetuar a pega de um objeto na cena, este passa a fazer parte da estrutura cinemática sendo então considerado como a ponta do robô virtual para fins de detecção de colisão, na análise da coerência semântica. As seguintes propriedades são consideradas no SCORBOT virtual:

- Mobilidade: Contando com a peculiaridade de imobilidade dos objetos dispostos na cena, se torna desnecessário um teste de colisão entre seus VE's, mesmo que estejam no mesmo subespaço do ambiente. É o caso dos pares (objeto X objeto), (objeto X mesa), (objeto X base) e (mesa X base);
- Alcance: Os limites da articulação do braço manipulador não permitem que determinados membros possam colidir, é o caso dos pares (corpo X mesa), (braço X mesa) e (garra X braço);
- Adjacência: Em um manipulador articulado dois elos adjacentes não se interceptam. É o que acontece com os pares (base X corpo), (corpo X braço), (braço X antebraço) e (antebraço X garra) e (ponta X garra).

A tabela 4 ilustra os pares que realmente devem ser testados no ambiente do SCORBOT virtual para a disposição de *n* objetos sobre a mesa. Essa tabela é montada considerando a realização da *broad phase* baseada apenas nas heurísticas proporcionadas pela coerência semântica do ambiente e aplicadas a cada par de objetos existentes na cena. A tabela têm em suas linhas e colunas os objetos

ordenados daquele que produz ou pode produzir mais deslocamento e velocidade até os objetos fixos.

Percebe-se que a tabela gera uma matriz triangular pois os testes entre dois objetos é uma avaliação comutativa. O símbolo “O” indica que há necessidade de testes mais precisos (*narrow phase*) pois os objetos podem colidir. Abaixo segue o detalhamento dos motivos pelos quais estes pares são eliminados, e a função de exclusão utilizada para mensurar a eficiência do algoritmo para um cena contendo um robô de partes com objeto na ponta e uma cena contendo na mesa vários objetos (totalizando n objetos fixos):

1. Xd: Colisões na Diagonal não existem, pois um objeto não colide consigo mesmo (*n*+5 casos);
2. Xr: Objetos móveis com fixos ou com outros móveis que por características construtivas, não se alcançam (4 casos);
3. Xf : Testes entre objetos fixos entre si mais o teste da base com os demais objetos fixos, ou seja, (n^2-n)/2 acrescido de n. Esta análiseesta relacionada com o critério de mobilidade.;
4. Xa: Eliminados por adjacência(5 casos);

O número total de testes eliminados com a coerência semântica é:

$$(n+5) + 4 + (n^2-n)/2 +5, \text{ ou seja,} \tag{5.1}$$

$$(n^2/2 + 1{,}5n +15) \tag{5.2}$$

Contra detecção de força bruta :

$$(n+6)^2/2 \tag{5.3}$$

Na tabela, grupos de testes são ressaltados. O grupo 1 representa os pares candidatos formados entre a mesa e os membros do robô (objetos móveis). Este grupo foi ressaltado, pois o seu teste pode ser executado com facilidade devido à peculiaridade de a mesa poder ser considerada como plano, sendo mais rápido o teste de colisão, além disso ela se encontra presente nos dois lados da divisão.

O grupo 2 forma os pares entre os próprios membros do robô, pois todos são objetos móveis da cena. O grupo 3 representa os pares candidatos formados entre os membros do robô e os objetos dispostos na mesa.

Objetos	VE Ponta	VE Garra	VE AnteBraço	VE Braço	VE Corpo	VE Base	VE Mesa	VE Objeto1	...	VE ObjetoN
VE Ponta	Xd	Xa	O	O	O	O	O	O	O	O
VE Garra		Xd	Xá	Xr	O	O	O	O	O	O
VE AnteBraço			Xd	Xá	O	O	O	O	O	O
VE Braço				Xd	Xá	Xr	Xr	O	O	O
VE Corpo					Xd	Xá	Xr	O	O	O
VE Base						Xd	Xf	Xf	Xf	Xf
VE Mesa							Xd	Xf	Xf	Xf
VE Objeto1								Xd	Xf	Xf
...									Xd	Xf
VE ObjetoN										Xd
X	O par não possui possibilidade de colisão									
O	Existe probabilidade de colisão para o par marcado									
	Grupo 1 - Testa colisão entre Objetos Móveis e Mesa									
	Grupo 2 - Testa colisão entre os objetos Móveis									
	Grupo 3 - Testa colisão entre os objetos móveis e objetos inseridos no ambiente									

Tabela 4 - Exploração da coerência semântica para *broad phase*

Concluí-se portanto que, para o estudo de caso, a análise da propriedade de coerência semântica do ambiente diminui significantemente o número de cálculos e comparações necessários, possibilitando a melhora do desempenho do algoritmo desenvolvido de forma rápida e eficiente. Contudo, a técnica se mostra peculiar a cada aplicação, sendo necessária a construção de uma nova tabela para cada caso de uso.

5.1.2 Explorando a Coerência Espacial

A coerência espacial pode ser explorada facilmente e de forma eficiente neste ambiente, isto porque o único fator que modifica o posicionamento dos objetos dispostos na mesa de trabalho é o ato de manipulação desses modelos pelo braço robótico. Neste sentido, uma estrutura de dados para representação do particionamento espacial será utilizada como apoio para enumeração dos objetos candidatos à colisão.

Pela dimensão da mesa de trabalho, uma partição em quadrantes no plano cartesiano xz da mesa de trabalho se mostra suficiente para execução desta tarefa de forma dinâmica, não haverá possibilidade de existirem centenas de objetos por quadrante que justifique novas partições.

A utilização de quatro estruturas de dados pré-computadas e que representem intervalos fixos dos quadrantes para armazenamento dos objetos da mesa segundo sua localização espacial atende satisfatoriamente à necessidade da aplicação. Propõe-se que estas estruturas que representarão os quatro quadrantes sejam compostas de :

1. Extensão e ponto central do quadrante;
2. Uma lista que representará os objetos contidos no quadrante.

A figura 35 mostra a partição espacial do ambiente do SCORBOT virtual. Na figura os objetos dispostos são relacionados à lista do quadrante em que se encontra.

A configuração das estruturas de dados apenas sofrerá alterações quando forem inseridos novos objetos via interface ou quando o robô manipulador movimentá-los para um novo local na mesa no ambiente de trabalho. A montagem das listas será realizada no momento da inserção de cada objeto em uma determinada posição, via interface. Cada quadrante é representado como uma OBBTree, e para o objeto estar inserido em tal quadrante é necessário que ele colida com o mesmo. É preciso atentar ao fato de que um objeto pode se encontrar em mais de uma lista se ele estiver posicionado sobre a região de fronteira entre os quadrantes.

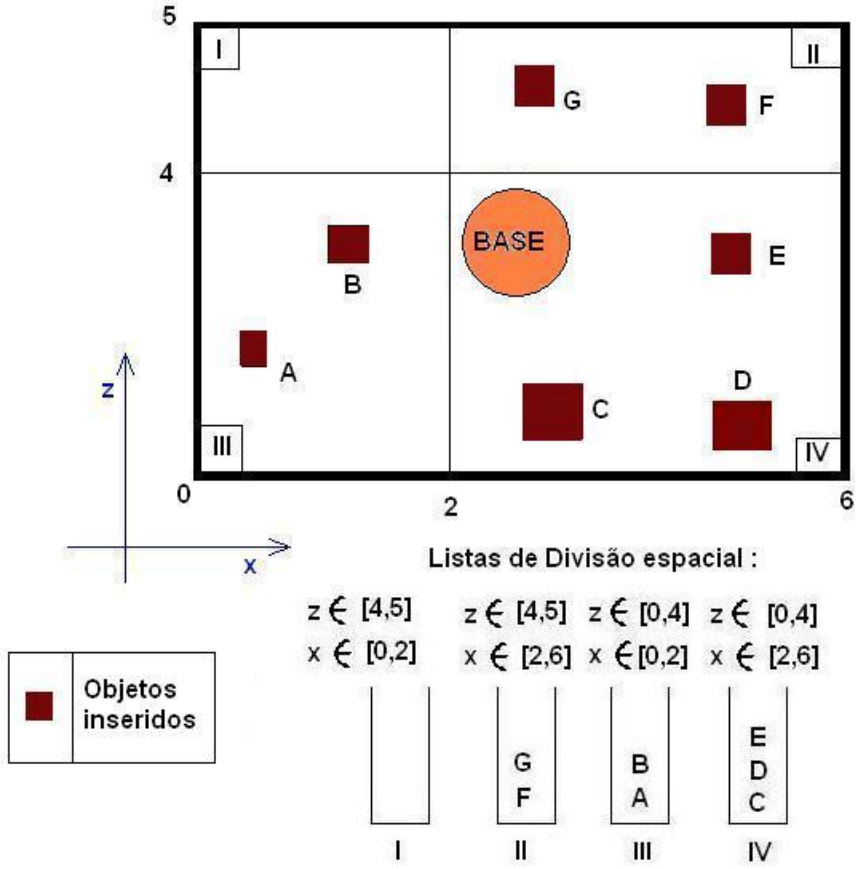


Figura 35 – Representação espacial por listas

A utilização de listas no lugar das árvores apresentadas no capítulo 3, se deve à simplicidade da aplicação que possui um ambiente restrito e com um baixo número de objetos dispostos se comparado a aplicações de larga escala. Aqui, por se tratar de uma estrutura linear, as listas se mostram eficientes para busca, inserção e exclusão para a quantidade de objetos em cena. Dessa forma não compensa a construção de árvores que envolvem problemas de balanceamento além de necessitarem de ferramentas mais sofisticadas de busca, construção ou até de atualização.

Toda esta estrutura foi projetada na primeira parte desde trabalho de conclusão, no capítulo sobre implementação será visto que a melhor repartição para coerência espacial foi a de dois lados simétricos, onde a base se situa entre ambas. Isto ocorreu com a experimentação do ambiente que levou a conclusão de que o tamanho da mesa

de trabalho e dos objetos gerados, que possuem dimensão relativamente pequena, seriam melhor trabalhados pela utilização de apenas dois lados.

5.1.3 O algoritmo para Detecção Abrangente (*Broad Phase*)

A construção da estrutura de dados para partição espacial no item anterior já foi realizada de forma simplificada (apenas quatro listas) vislumbrando a quantidade de simplificações proporcionada pela coerência semântica do ambiente, demonstradas na tabela 4. Unindo os conceitos explanados anteriormente, é possível criar um algoritmo baseado em heurísticas simples que classificarão apenas os pares de objetos que realmente possuem potencial de colisão. Essas regras formam um pseudo algoritmo para a fase da detecção abrangente da aplicação demonstrado no capítulo de implementação. Com a heurística:

No quadrante em que os objetos móveis se encontrarem, enumerar os pares candidatos a colisão segundo o grupo 3 da tabela, considerando apenas os objetos distribuídos sobre a mesa que estiverem localizados na lista do referido quadrante.

Aqui a coerência espacial na utilização das listas e a coerência semântica usada na geração da tabela trabalham juntas para formação de um conjunto mínimo de pares candidatos.

Através dessas simples regras heurísticas define-se ao final, um pequeno conjunto de pares extraídos do universo de objetos possivelmente colidentes do ambiente SCORBOT. Essa lista será verificada por uma segunda etapa do algoritmo que realizará uma detecção de colisão especifica diagnosticando a detecção de colisão, se esta ocorrer.

5.2 Fase Específica (*Narrow Phase)*

Para uma detecção de colisão mais especifica que a realizada na etapa anterior, uma hierarquia de árvore de OBBs estará implícita na estrutura do objeto envolvido, e aparecerão como caixas invisíveis ao usuário. Pela adoção de uma hierarquia pré-

computada que fará parte dos objetos, a construção da HVE será realizada no momento da modelagem dos objetos, sendo que cada árvore será construída manualmente.

Pelo fato de se tratar de um robô virtual didático veiculado via Internet não existe a exigência da precisão exata da resposta do algoritmo de detecção de colisão porém é vital a velocidade de resposta do mesmo algoritmo. Com base nesse fato imprescindível e estratégico para a eficiência da aplicação, não serão realizados testes entre triângulos, ficando a precisão do algoritmo a cargo do número de níveis contidos nas HVEs, nível este a ser inferido na etapa de implementação da solução.

Para detecção de colisão entre as HVEs dos pares de objetos candidatos à colisão será utilizada uma ATVE entre essas estruturas e o teorema de SAT descrito neste trabalho para detecção de colisão entre duas OBBs, que vem sendo bastante utilizado na literatura por apresentar boas respostas para detecção de colisão em ambientes dinâmicos e interativos.

5.3 Geração de objetos

Do ponto de vista educacional, um simulador robótico permite aos alunos vivenciarem experiências que se situam entre a manipulação abstrata de conceitos e estratégias e a manipulação direta dos elementos com que se defrontam na prática, além de auxiliar no desenvolvimento das capacidades de raciocínio lógico e de programação.

Assim, a implementação da funcionalidade de criação de diferentes cenários, com a geração de objetos de formas, tamanhos e posicionamentos aleatórios disponibiliza ao usuário deste sistema uma ferramenta mais completa para o exercício das habilidades de manipulação e programação de um robô. Com essa função o ambiente virtual permite a aplicação de diferentes estratégias em diferentes cenas de dificuldade aleatória. Proporcionando um sistema onde o usuário possa fazer o treinamento diversas vezes, em diferentes ambientes, analisando e observando suas respostas, consolidando assim sua base de conhecimento.

Com a geração aleatória se objetiva proporcionar em cada novo uso uma experiência diferente pela geração de uma quantidade muito grande de situações que poderiam ser difíceis de prever individualmente e, portanto o *layout* da cena não fica restrito a um pequeno conjunto de situações pré-definidas. Como desvantagem pode-se mencionar a dificuldade de controlar “se” e “quando” o usuário encontrará determinada classe de situações que podem ser importantes para seu aprendizado. Entretanto estas classes ainda não estão identificadas na literatura e nem o ambiente virtual controla ainda “quem” é o usuário. Estas funcionalidades podem ser posteriormente incorporadas ao sistema usufruindo assim da solução já implementada.

A função de geração foi projetada para inserir um número de objetos definido pelo usuário via interface, de tamanhos (dentro de faixas definidas) e formas (atualmente cilindro, paralelepípedo e esferas) aleatórios, de diferentes cores, posicionados em qualquer parte da mesa, sob qualquer orientação possível (Santos *et.al,*2007).

Esta geração, entretanto deve obedecer algumas restrições características do sistema implementado e das condições reais de simulação, que são:

1. Um objeto não deve interpenetrar geometricamente a mesa de trabalho, o robô ou qualquer outro objeto já disposto em cena;
2. Cada elemento deverá possuir um volume envolvente (VE) correspondente, ao nível de JAVA, que permita os testes de colisão do sistema proposto;
3. De acordo com a posição do objeto, ele deverá ser incorporado à lista relacionada com o lado da mesa onde estiver, contemplando a coerência espacial.

6 IMPLEMENTAÇÃO

Dividiu-se o problema da implementação da detecção de colisão para o ambiente do Scorbot Virtual em três módulos distintos:

1. Geração de Objetos Variados;
2. Pseudo Pega de Objetos;
3. Detecção de Colisão para o Ambiente Virtual.

6.1 Implementação da Geração de Objetos Variados

O algoritmo descrito exposto aqui foi iproposto no artigo Geração de Cenários não colidentes para um ambiente robótico (Santos *et.al,* 2007)O quadro 1 apresenta o pseudo código da função de geração de objetos. Na linha 1 é gerado o número de objetos a serem inseridos na cena, para cada um é decidido o tipo geométrico do elemento a ser inserido. Na linha 6, a função *objetoRandomico* vai definir os parâmetros do objeto de tipo geométrico selecionado, da seguinte forma:

1. Paralelepípedo: Altura, largura e comprimento entre 0.05m e 0.1m, posicionamento (x, y, z) aleatório dentro dos limites da mesa. Ângulo de rotação no eixo y entre 0.0 e 6.283 radianos.
2. Esfera: raio entre 0.05m e 0.1m, posicionamento (x, y, z) aleatório dentro dos limites da mesa.
3. Cilindro: raio entre 0.05m e 0.1m, altura entre 0.05m, 0.4m. Ângulo no eixo x de 0 ou 180º (em pé ou deitado), se estiver deitado: ângulo em y entre 0 e 360º, para orientação aleatória.

Para cada objeto existe uma classe que representa uma OBBTree (melhor explanada adiante), criada segundo seus parâmetros. Como se trabalha aqui com formas primitivas, que fornecem parâmetros que possibilitam ajuste a uma única OBB, a OBBTree dos objetos possui apenas um nível, ou seja, a própria raiz. O código das linhas 7-19 vai testar se o objeto colide com algum elemento já criado, e em qual lista ele deve ser inserido. É importante salientar que pode ocorrer do elemento estar

posicionado na fronteira das listas de coerência espacial, sendo assim ele deverá ser inserido nas duas. Esse trecho do código trata destas questões. Para isso o *testaObjListaX* , que testa a OBB do novo objeto pode retornar as seguintes situações:

1. Pertencer à lista, não colidindo aos outros elementos nela contidos;
2. Pertencer à lista, colidindo com algum outro elemento nela contido;
3. Não pertence a este lado da mesa, ou seja, não pertence à lista.

Se o retorno indicar colisão, a variável *flagColisao* continuará verdadeira indicando a necessidade da criação de um novo objeto em outra posição aleatória, excluindo o anterior. Caso contrário, o objeto é inserido na lista correspondente e posteriormente no ambiente VRML. Isto acaba garantindo uma certa distribuição uniforme de objetos nos dois lados, pois à medida que um lado fica sobrecarregado isto gera maior probabilidade de colisão de novos elementos naquele lado.

O quadro 1 mostra a implementação da classe OBB, relacionada a cada objeto. O quadro 3 traz a classe do tipo geométrico Paralelepípedo para exemplificar a implementação dos objetos gerados aleatoriamente no algoritmo do quadro 1.

Quadro1.Pseudo Código - Gerador Aleatório de Objetos

```
1 geradorObjetos(RoboComand robo, int numObj)
2 Enquanto (i<numObj)
3     flagColisao = verdadeiro
3     tipoGeometrico = tipoRandomicoEntre(Paralelepípedo, Esfera, Cilindro);
4     Enquanto(flagColisao == verdadeiro)
5           novoObj = objetoRandomico(tipoGeometrico)
7           Se(testaObjLista1(novoObj.getOBBTree)==pertenceLista)
8                 flagColisao = false;
9                 Se(testObjLista2(novoObj.getOBBTree)==colideElementosLista)
10                      flagColisao = verdadeiro;
11                Senão
12                      Se(testaObjLista2(novoObj.getOBBTree)==pertenceLista)
13                            Se (naoColideRobo(robo, novoObj.getOBBTree))
14                                  lista1.insereObj(novoObj)
15                                  lista2.insereObj(novoObj)
16                      Senão
17                            Se (naoColideRobo(robo, novoObj.getOBBTree))
18                                  lista1.insereObj(novoObj)
19          Senão
20                Se(testaObjLista1(novoObj.getOBBTree)==naoPertenceLista)
21                      Se(testaObjLista2(novoObj.getOBBTree)==pertenceLista)
22                            Se (naoColideRobo(robo, novoObj.getOBBTree))
23                                  lista2.insereObj(novoObj)
24    fimEnquanto
25    Se (!flagColisao)
26          InsereObjMundoVRML(novoObj)
27          i++;
28 fimEnquanto
```

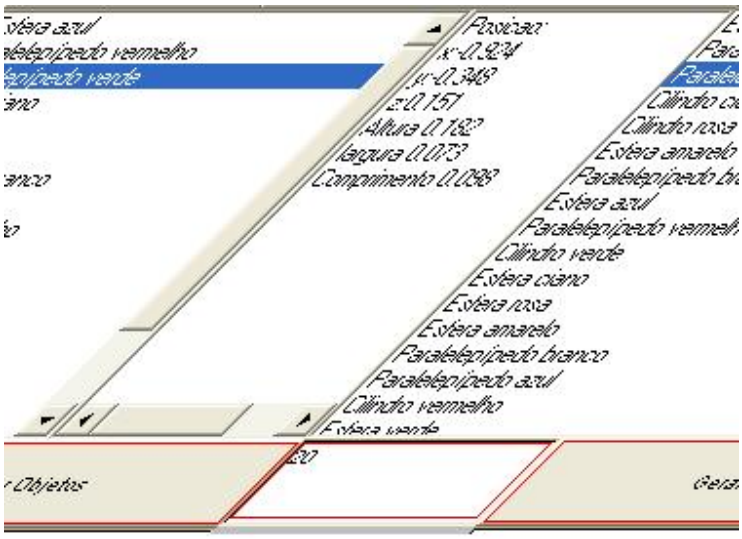


Figura 36. Projeto da nova Interface do Robô Virtual com Geração de Objetos

A figura 36 mostra o projeto da nova interface do ambiente robótico com a possibilidade de geração de cenas variantes não colidentes. Acrescentou-se um botão (“Gerar Objetos”) que ao ser acionado gera a distribuição de diferentes objetos na mesa e uma lista das características de forma e cor relacionadas ao elemento inserido, com dois *cliques* em um objeto da lista, todas as suas características de posicionamento e parâmetros são listadas na caixa de texto ao lado. Essa lista servirá futuramente, como auxilio a um tutor na indicação de objetivos de programação indicados aos aprendizes. Abaixo da lista, encontram-se as informações indicativas a respeito do posicionamento e dimensionamento do objeto selecionado. Ainda, ao lado do botão gerador, se encontra uma caixa de texto que permite ao usuário escolher o número de objetos a ser gerado.

Todas as funcionalidades implementadas estão dentro da classe PartManager.java, apresentada ao decorrer do anexo especificada no quadro 20.

6.2 Pseudo-Pega de Objetos

A simulação de pega dos objetos inseridos visa adicionar ou remover o objeto mais próximo à estrutura da garra (considerando a distância euclidiana entre os centros), o tratamento adequado a respeito dos raciocinios detalhados de pega de objetos para a ferramenta garra “pinça” está sendo desenvolvidos em trabalhos paraleleos.

Assim, aqui, o objetivo desta pseudo-pega é realizar os testes de detecção de colisão considerada ponta da ferramenta manipulando um objeto, pois, os níveis de dificuldade são diferentes de quando a garra se movimenta livremente para quando é realizada a preensão de um objeto qualquer. Este último caso é um fator preponderante para o aumento da dificuldade de manipulação, pois, como a extensão da garra é somada à extensão do elemento pego, o risco de colisão com os demais elementos do ambiente aumenta. Existem duas implicações na construção desta tarefa:

1. Considerar o objeto anexado à extremidade móvel (garra) do robô, tendo, por sua extensão e característica de movimentação, uma grande probabilidade de colidir com os demais elementos da cena.

2. Realizar a atualização dos eixos da OBB do objeto movimentado, segundo a cinemática direta que rege o movimento do braço articulado do Scorbot Virtual.

Conforme pode ser constatado detalhadamente no anexo A, a classe ObjectManager.java gerencia os eventos de preensão de objetos pela garra através dos métodos :

1. graspPart(OBBTree Garra) : Recebe como parâmetros a OBBTree da Garra, verifica o objeto mais próximo dela e adiciona uma Route no VRML com a seguinte estrutura:

browser.addRoute(ROBO,"ponto",filhoObj[idObjPego][0],"set_translation");

- *browser* é um objeto do tipo Browser, classe provida pelo EAI que permite à applet acessar as propriedades do browser VRML, no caso, permite adicionar uma nova Route no mundo VRML mapeado pela página HTML.

Os parâmetros repassados indicam que o evento de alteração do *ponto* que está localizado na ponta da garra do *ROBO* será roteado para os valores de translação do *filhoObj[idObjPego],* que é a representação do objeto pego no VRML.

Dentro da própria classe, o objeto é retirado da lista de coerência espacial a qual pertence e o *flag* objetoPego é ativado, para fins de Detecção de Colisão. O pseudo código do quadro 2 ilustra o procedimento descrito.

Quadro 2 – Pseudo-Código para pega de objetos

```
1 int graspPart(OBBTree garra)
2 Se (!isSegurandoObjeto)
3 {   idObjPego = buscaObjetoProximo(garra)
4     browser.addRoute(ROBO,"ponto",filhoObj[idObjPego][0],
                  "set_translation");
7     segurandoObjt = true;
8     retiraObjQuadrante(idObjPego);
9 }
```

2. releasePart(Obb Garra) : Realiza a operação inversa de graspPart, ela retira do VRML o nó roteador através do comando:

browser.addRoute(ROBO,"ponto",filhoObj[idObjPego][0],"set_translation");

Adiciona o objeto à lista de coerência espacial relativa ao lado, ou aos lados em que estiver contido. E libera o *flag* de objetoPego, o quadro 3 demonstra os passos de seqüência do algoritmo implementado.

Quadro 3 – Pseudo-Código para liberação de objetos

```
1 releasePart(Obb obbGarra, int idObj) {
2 se (isSegurandoObjeto)
3     browser.deleteRoute(ROBO,"ponto",
filhoObj[idObjPego][0],"set_translation");
4     atualizaObbObjetoPego(obbGarra);
5     segurandoObjt = false;
6     insereObjQuadrante(idObjPego);
7 }
```

A maior dificuldade para a implementação desta fase foi encontrar uma forma no VRML que permitisse que o objeto apenas caminhasse com a ponta da garra, sem deslocar-se para o posicionamento da mesma, objetivo não alcançado, ao agarrar um objeto, este se desloca automaticamente para o centro da garra.

6.3 Detecção de Colisão

6.3.1 Definição das OBB's no Scorbot

Pelo fato do modelo do robô virtual ser um braço articulado, as OBBs devem, da mesma forma que a estrutura a que representam, possuir um movimento calculado pela cinemática direta.

O cálculo utilizado por Zwirtes(2004) foi baseado em Devanit e Hartenberg (DH) que propuseram uma notação sistemática para atribuir um sistema de coordenadas ortonormal com a regra da mão direita, um para cada elo numa cadeia cinemática aberta de elos. Uma vez que estes sistemas de coordenadas fixados ao elo são atribuídos, transformações entre sistemas de coordenadas adjacentes podem ser representadas por uma simples matriz de transformação de coordenadas homogêneas 4x4 padronizada, para atribuir o sistema de coordenadas para os elos do robô. Os passos detalhados da construção destes cálculos podem ser vistos em Zwirtes(2004).

Tem-se mostrada a configuração do modelo do robô Scorbot com 5 graus de liberdade na figura 37, após realização do procedimento de DH com todos os parâmetros cinemáticos envolvidos e na tabela 5 são sumarizados os paramêtros cinemáticos do Scorbot Virtual. Estes valores foram encontrados através da medição direta do robô em VRML, esta necessidade foi um obstáculo para o desenvolvimento da aplicação pois os valores atuais na implementação não correspondem às dimensões reais do robô.

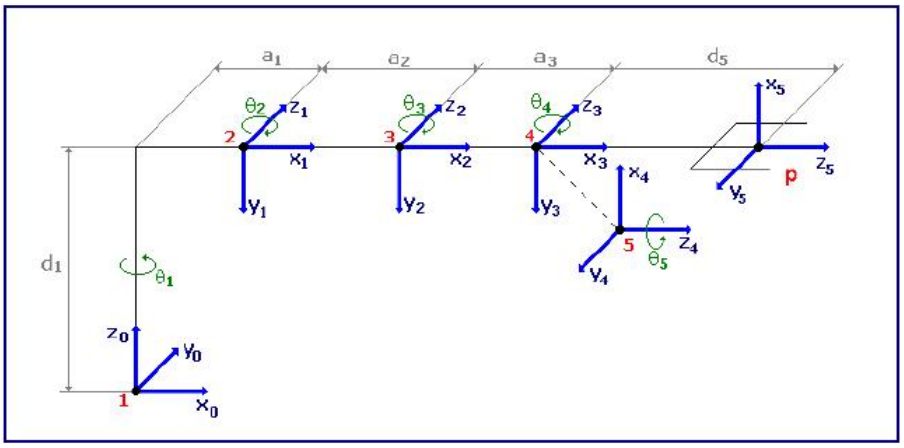


Figura 37 - Esquema Gráfico do procedimento de DH

*Medidas em metros

Junta	ai	di	èi	Nome
1	0,75	6,8	è1	Base
2	9,35	0	è2	Ombro
3	4,45	0	è3	Cotovelo
4	0	0	è4	Punho
5	0	3,5	è5	Rotação

Tabela 5 - Parâmetros cinemáticos do robô Scorbot Virtual

Tendo em mãos os parâmetros cinemáticos de todas as juntas, e utilizando a matriz de transformação de coordenadas homogêneas, transforma-se o sistema de coordenadas de um elo k-1 para o sistema de coordenadas k, substituindo-se para todas transformações os parâmetros cinemáticos de cada junta obtem-se as matrizes de orientação local de cada elo, representadas pelas equações 6.1– 6.5, a figura 38 ilustra o posicionamento dos parâmetros utilizados.

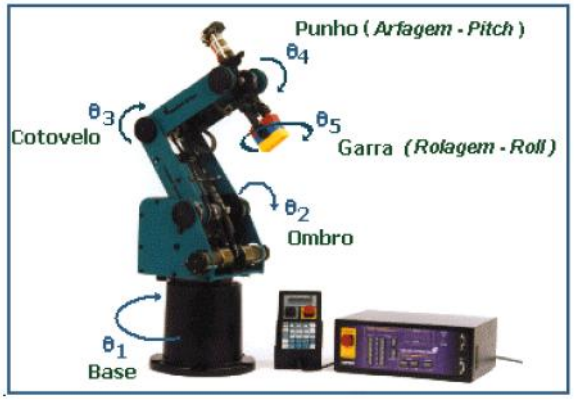


Figura 38 - Robô manipulador SCORBOT-ER4PC (Zwirtes, 2004)

Matriz de coordenada local do Corpo:

$$ {}^{0}T_{1} = \begin{bmatrix} \cos\theta_1 & 0 & -\operatorname{sen}\theta_1 & a_1.\cos\theta_1 \\ \operatorname{sen}\theta_1 & 0 & \cos\theta_1 & a_1.\operatorname{sen}\theta_1 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.1) $$

Matriz de coordenada local do Braço :

$$ {}^{1}T_{2} = \begin{bmatrix} \cos\theta_2 & -\operatorname{sen}\theta_2 & 0 & a_2.\cos\theta_2 \\ \operatorname{sen}\theta_2 & \cos\theta_2 & 0 & a_2.\operatorname{sen}\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.2) $$

Matriz de coordenada local do AnteBraço:

$$ {}^{2}T_{3} = \begin{bmatrix} \cos\theta_3 & -\operatorname{sen}\theta_3 & 0 & a_3.\cos\theta_3 \\ \operatorname{sen}\theta_3 & \cos\theta_3 & 0 & a_3.\operatorname{sen}\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.3) $$

Matriz de coordenada local da rotação da Garra:

$$ {}^{3}T_{4} = \begin{bmatrix} \cos\theta_4 & 0 & -\operatorname{sen}\theta_4 & 0 \\ \operatorname{sen}\theta_4 & 0 & \cos\theta_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4} $$

Matriz de coordenada local da elevação da Garra:

$$ {}^{4}T_{5} = \begin{bmatrix} \cos\theta_5 & -\operatorname{sen}\theta_5 & 0 & 0 \\ \operatorname{sen}\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.5} $$

Onde : As três primeiras colunas representadas por R(q) correspondem respectivamente aos vetores do eixo que orienta o membro em questão e o ponto p(q) representa o ponto de alcance deste mesmo membro. Na implementação, a entidade que representa a árvore é dada no quadro 20 do anexo.

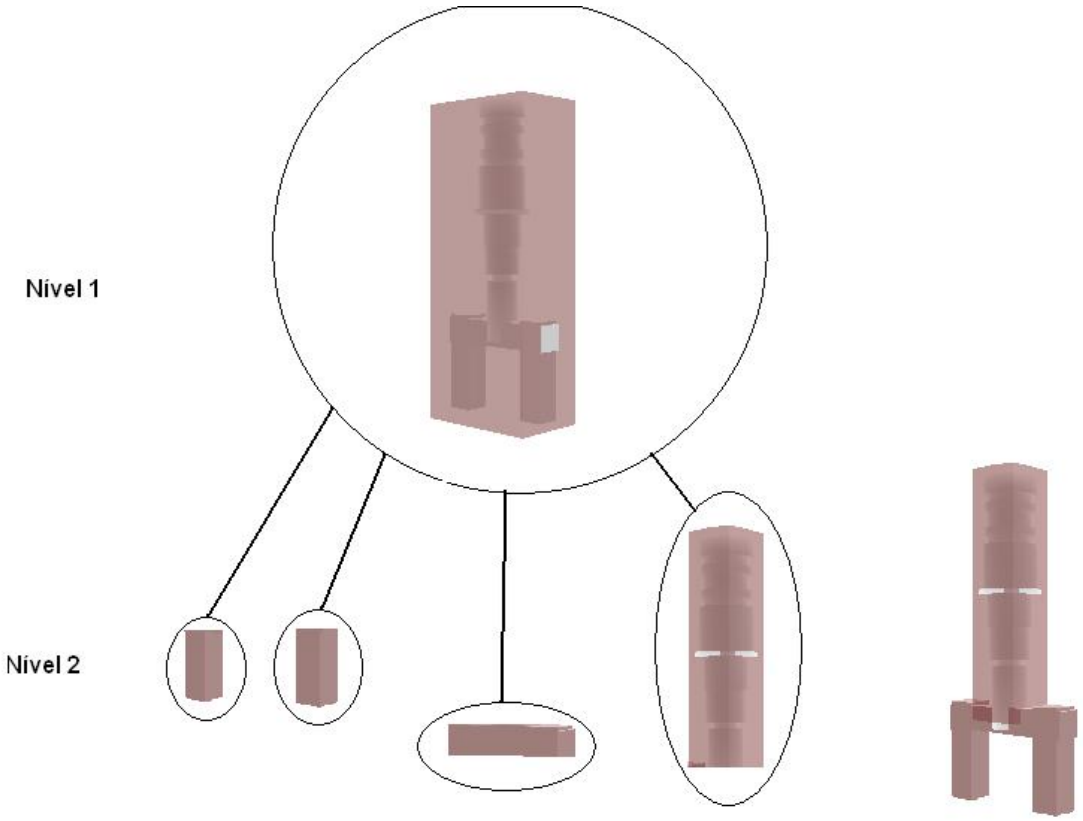


Figura 39 – OBBTree de dois níveis aplicada à Garra

Com todas as matrizes montadas, é realizada a multiplicação sucessiva das matrizes desde a base até o elo k onde se deseja calcular a orientação, assim, define-se a equação que calculará cada eixo de orientação do elo bem como seu ponto central. Este útimo pode ser calculado considerando-se a metade da medida do elo em questão. Para todos os membros foram definidas OBBTrees de um único nível, com exceção da garra, que é modelada com dois níveis da árvore, como descreve a figura 39, onde a garra aparece dividida em dedos, suporte dos dedos e eixo/servomotor da garra. Os centros dos elementros do segundo nível foram obtidos através da translação da mediana no eixo referido.

Definidas as OBB's, o problema da Detecção de colisão no ambiente do Scorbot Virtual segue dividido em duas fases:

1. *Broad Phase*
2. *Narrow Phase*

6.3.2 Fase Abrangente

Esta fase tem por objetivo realizar a enumeração dos possíveis pares colidentes, pois a utilização de OBBTrees se ajusta às características do Scorbot, porém os cálculos de colisão são consideráveis. Sendo assim, quanto maior o número de testes (entre duas OBBs) eliminados nesta etapa, melhor será o desempenho da aplicação. Para isso, esta fase utiliza *Coerência Semântica e* C*oerência Espacial.*

A coerência semântica para o Scorbot Virtual foi definida pela tabela 4, onde a enumeração dos pares de OBBs que oferecem risco de colisão, foram listados em categorias que definem grupos de pares. Estas categorias foram tratadas em métodos que realizam os testes de colisão na ordem de eminência(primeiro ponta) e simplicidade (grupo 1). O critério utilizado para a realização desta ordenaçao, foi a velocidade, assim o objeto que possuir maior amplitude de movimento, em um espaço de tempo, deverá ser testado primeiro, pois, atinge uma gama maior do espaço, no caso, o elemento crítico é a garra ou o objeto pego.

Utilizando-se da coerência semântica, foram definidos os métodos que implementam os grupos semânticos de colisão. O grupo 1 é definido pelo grupo de colisão formado pela ponta do robô (garra livre ou garra realizando preensão) e mesa – quadro 5, seguido pelos testes do grupo 2 listados no quadro 6. Dentro do método que lista os pares do grupo 3 mais uma característica da coerência espacial é utlizada para a deifinição da *broad,* são testados apenas os objetos que estiverem do lado da mesa que a garra estiver presente e os elos móveis do robo, definidos na tabela 4.O quadro 7 e 8 demostram o pseudo-código da aplicação

A Coerência espacial do ambiente foi explorada pela divisão do espaço de abrangência da mesa em duas partes, esquerda e direita com relação ao comprimento. Houve aqui uma alteração no projeto inicial em virtude do tamanho da mesa, a divisão em quadrantes pulverizada em demasia o espaço robótico do Scorbot, não trazendo melhorias.

A evocação dos métodos é centralizada no quadro 4, onde são chamados os métodos citados na respectiva ordem de grupos.

A classe java que realiza este gerenciamento dos objetos no espaço é o ObjectManager.java, que, através de duas listas com referências aos objetos retorna suas respectivas OBBTrees.

Quadro 4. Pseudo-Código *Broad Phase – Coerência Semântica*

```
1 String testeColisaoBroadPhase(RoboComand robô, PartManager Obj)
2            msg = testesColisaoGrupo1(robo);
3            Se (msg != null)
4                  return (msg);
5            msg = testesColisaoGrupo2(robo);
6            Se (msg != null)
7                  return (msg);
8
9            msg = testesColisaoGrupo3(robo, obj);
10           Se (msg != null)
11                 return (msg);
```

A outra forma de explorar a coerência espacial, através da altura dos objetos, em relação à altura da base do Scorbot, também é uma característica gerenciada por ObjectManager.java, que retorna o ponto mais alto do ambiente do ponto de vista da distribuição de objetos. O algoritmo de colisão apenas executa os cálculos com OBBs dos objetos quando a garra se localiza abaixo desta altura, este detalhe pode ser observado no pseudo código do quadro 4 na linha 2 .

Os método “testaColisaoOBBTree3D(OBBTree a, OBBTree b)” se caracteriza como o teste de *narrow phase* da implementação, a ser discutido no próximo tópico.

Quadro 5. Pseudo-Código *Broad Phase – Grupo 1*

```
1 String testesColisaoGrupo1(RoboComand robo)
2     Se (obj.isSegurandoObj())
3           Se (testeColisaoOBBTree3d(obj.getOBBTreeObjPego(),treeMesa))
5                 return ("Objeto Pego e Mesa");
6     Se(colisao.testeColisaoOBBTree3d(robo.getOBBTreeGarra(), treeMesa))
7           return ("Garra e Mesa");
8     Se(colisao.testeColisaoOBBTree3d(robo.getOBBTreeAnteBraço(), treeMesa))
9           return ("AnteBraço e Mesa");
10
11 return null;
```

Quadro 6 Pseudo-codigo *Broad Phase – Grupo 2*

```
1 String testesColisaoGrupo2(PartManager obj, RoboComand robo)
2     Se (obj.isSegurandoObj())
3           Se(testeColisaoOBBTree3d(obj.OBBTreeObjPego(),robo.OBBTreeBase()))
4                 return ("Objeto Pego e Base");
5           Se(testeColisaoOBBTree3d(obj.OBBTreeObjPego(),robo.OBBTreeCorpo()))
6                 return ("Objeto Pego e Corpo");
7           Se(testeColisaoOBBTree3d(obj.OBBTreeObjPego(), robo.OBBTreeBraco()))
8                 return ("Objeto Pego e Braco");
9           Se(testeColisaoOBBTree3d(obj.OBBTreeObjPego(), robo.OBBTreeAnteBraco())
10                return ("Objeto Pego e AnteBraco");
12    Se(testeColisaoOBBTree3d(robo.OBBTreeGarra(), robo.OBBTreeBase()))
13          return ("Garra e Base");
14    Se(testeColisaoOBBTree3d(robo.OBBTreeGarra(), robo.OBBTreeCorpo()))
15          return ("Garra e Corpo");
18    Se(testeColisaoOBBTree3d(robo.OBBTreeAnteBraco(), robo.OBBTreeBase()))
19          return ("AnteBraco e Base");
20    Se(testeColisaoOBBTree3d(robo.OBBTreeBraco(), robo.OBBTreeCorpo()))
21          return ("AnteBraco e Corpo");
26 return null;
```

Quadro 7. Pseudo-codigo *Broad Phase – Grupo 3*

```
String testesColisaoGrupo3(PartManager obj, RoboComand robo)
      Se(obj.getNumeroObjetosGerados() > 0)
            ListaOBBTrees listaOBBTrees;
            String retorno;
            Se (obj.roboLado1(robo.getOBBTreeGarra()))
                  listaOBBTrees = obj.getListaObbsLado1();
                  retorno = testeObjetosRobo(listaOBBTrees, obj, robo);
                  Se(retorno != null)
                        return retorno;
            Se (obj.testaObbLado2(robo.getOBBTreeGarra()))
                  listaOBBTrees = obj.getListaObbsLado2();
                  retorno = testeObjetosRobo(listaOBBTrees, obj, robo);
                  Se(retorno != null)
                        return retorno;
      return null;
```

Quadro 8. Pseudo-codigo *Broad Phase – Grupo 3 – Detalhado*

Vale salientar que mesmo utilizando-se a garra para definir a orientação espacial do lado da mesa em q o robô se encontra, no teste ainda é considerado a garra ou ponta contra os demais objetos fixos.

```
1 String testeObjetosRobo(ListaOBBTrees lista,PartManager obj, RoboCollision
robo)
2 n = tamanho(lista)
3 enquanto(n>0)
4     se (obj.isSegurandoObj())
5           se(testeColisaoOBBTree3d(obj.treeObjPego(),lista.OBBTree(n)))
6                 return "Objeto Objeto ";
7     se (testeColisaoOBBTree3d(robo.OBBTreeGarra(),lista.OBBTree(n)))
8           return "Garra Objeto";
9     se(testeColisaoOBBTree3d(robo.OBBTreeAnteBraco(), lista.OBBTree(n)))
10          return "AnteBraco Objeto";
11    se(testeColisaoOBBTree3d(robo.OBBTreeBraco(), lista.OBBTree(n)))
12          return "Braco Objeto";
13    se(testeColisaoOBBTree3d(robo.OBBTreeCorpo(),lista.OBBTree(n)))
14          return "Corpo Objeto";
15          n--;
16 return null;
```

6.3.3.Fase Específica

Esta fase executa o cálculo de colisão entre duas OBBTrees listadas pela na Fase Abrangente. Esta é a última etapa de testes, considerando o ajuste das OBBs e necessidade de um desempenho razoável da aplicação WEB. O Quadro 9 representa o algoritmo de recursividade implementado para a detecção de colisão entre duas árvores de volumes envolventes, no caso OBBTrees. A resposta à colisão acontece na interface, na caixa de texto de *feedback*, da figura 40, onde, se ocorrer colisão, é mostrado um texto com as entidades colidentes e na própria cena, onde os colidentes mudam de cor, para vermelho. Esta última característica adicionou um efeito visual agradável, aumentando a precisão da simulação. Todo o sistema de testes de colisão criado para o Scorbot se encontra na classe CollisionManager, no anexo, tópico A.2.3.

Quadro 9. Detecção de Colisão entre OBBTrees

```
1 public boolean testeColisaoOBBTree3d(OBBTree a, OBBTree b)
2 {
3     OBBTree childrenA[], childrenB[];
5     boolean flagColisao = false;
6     if (this.testeColisaoOBBTree3d(a.getRoot(), b.getRoot()))
7     {
8           childrenA = a.getChildren();
9           childrenB = b.getChildren();
10
11          if (( childrenA == null) && (childrenB == null)) return true;
12                if(childrenA != null){
13                      for (int i = 0; i< a.getNumChildren(); i++)
14                      {
15                            if
(testeColisaoOBBTree3d(childrenA[i].getRoot(), b.getRoot()))
16                            if (testeColisaoOBBTree3d(b, childrenA[i]))
17                                  return true;
18                      }
19                }else
20                      for (int i = 0; i< b.getNumChildren(); i++)
21                      {
22                            if
(testeColisaoOBBTree3d(childrenB[i].getRoot(), a.getRoot()))
23                            if (testeColisaoOBBTree3d(a, childrenB[i]))
24                                  return true;
25                      }
26
27                }
28    return false;
29}
```

O teste é , realizado recursivamente. Como parâmetros têm-se duas OBBTrees. A raiz das árvores é testada, havendo colisão, o teste é chamado recursivamente desta vez entre cada nó filho da raiz de A contra a própria árvore B. O teste segue até que não ocorra uma colisão, ou até que duas folhas colidam.

7.1 Nova Interface

A nova versão do robô virtual pode ser encontrada em LaRVA (2007), acrescenta a funcionalidade de geração de objetos descrita no tópico anterior. Uma caixa de texto foi adicionada com o objetivo de oferecer a resposta à colisão, indicando os objetos colidentes. Outra caixa de entrada permite que o usuário insira o número de objetos a ser inserido na cena. A figura 40 assinala as novas funções adicionadas.

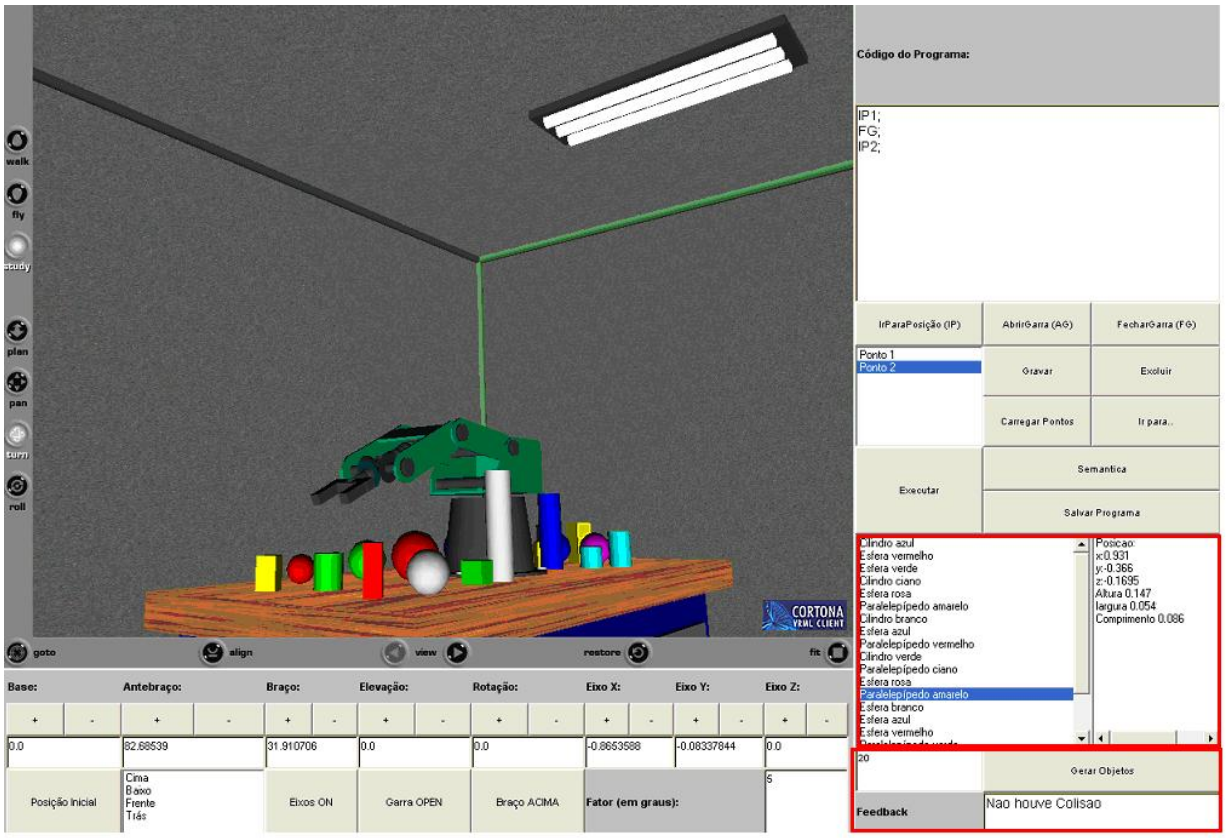


Figura 40. Nova Interface do Ambiente Virtual com Geração de Objetos e detecção de Colisão

As funções anteriores foram mantidas. Na applet horizontal estão as funções de movimentação do robô. As especificações técnicas estão detalhadas no item A.2.1 do anexo. Nestes botões é possível aumentar ou diminuir a angulação de cada elo, abrir ou fechar a garra além de mostrar o posicionamento do ponto do final distal da garra nos eixos x, y e z.

Já a applet horizontal permite a realização da programação do robô virtual, as explicações aprofundadas estão nos itens A.2.2 e A.2.3 do anexo.

7.2 Testes

Para os testes de desempenho dos algoritmos, a aplicação foi hospedada no servidor da UDESC. Para a simulação foi utilizada uma configuração que se espera ser a mínima disponibilizada pelo usuário. A máquina cliente da aplicação dispunha da seguinte configuração:

Memória RAM: 64Mb;

Placa de Vídeo: 8Mb;

CPU: AMD K6, 266MHz;

HD :10Gb.

Sistema Operacional : Windows 98

O novo Scorbot Virtual foi experimentado em três diferentes situações:

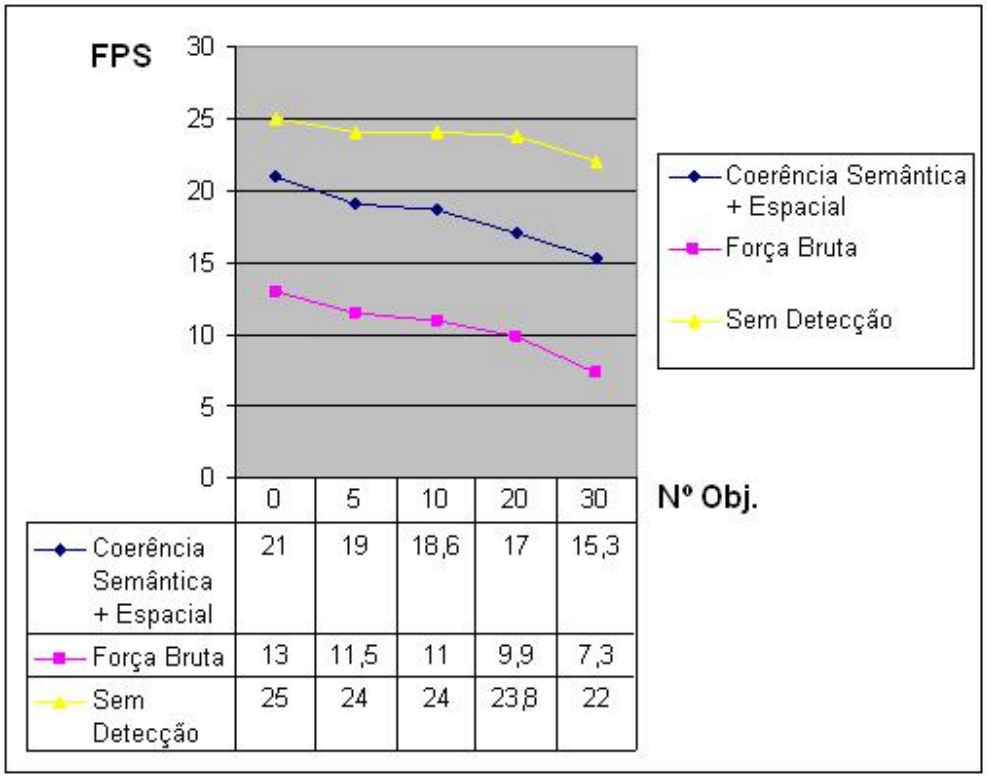
1 – Versão anterior, sem realização dos testes de detecção de colisão;

2 - Utilizando-se um algoritmo de força bruta, onde todos os envoltórios são testados contra todos os outros contidos no ambiente;

3 – Utilizando-se o conceito de coerência semântica e espacial para realização da filtragem inicial (*Broad Phase*).

O comportamento padrão adotado para parametrização dos testes foi de uma volta completa em torno da base do robô e o com um movimento de arco acima da base, com inserção de 0, 5, 10, 20, 30.

A métrica de desempenho analisada está em *frames/*segundo, correspondente a coluna do gráfico de desempenho da figura 41 A discussão a respeito de tal gráfico é realizada no tópico 7.1.



	0	5	10	20	30
Coerência Semântica + Espacial	21	19	18,6	17	15,3
Força Bruta	13	11,5	11	9,9	7,3
Sem Detecção	25	24	24	23,8	22

Figura 41. Gráfico de Desempenho da Aplicação

7.1 Discussão

Avaliando a figura 41 foi possível verificar o bom desempenho do algoritmo, que contribuiu para o sucesso no alcance dos objetivos do trabalho, que se propõem a realizar eficientemente a detecção de colisão no Scorbot virtual não diminuindo de forma significativa o desempenho da simulação, permitindo uma navegação e manipulação satisfatória ao usuário do sistema. Em alguns pontos, inclusive, o desempenho da técnica empregada foi o dobro de uma solução de força bruta, e a tendência desta diferença é aumentar com o número de objetos, pois, superado o *overhead* necessário para o tratamento das coerências a aplicação praticamente se estabiliza, diminuindo cada vez mais o número de testes, se comparado a força bruta.

Como se pode perceber, os testes foram executados em uma máquina de baixo desempenho se comparado às capacidades de memória e processamento disponíveis no mercado hoje. Esta configuração foi utilizada por ser imperceptível a diferença entre os métodos quando aplicados em um computador recente. Por exemplo, um AMD

Athlon 2.2GHz e 256MB de memória RAM manteve o desempenho médio de 61 FPS para todos os métodos. Porém, as diferenças encontradas e demonstradas no gráfico reforçam a característica de acessibilidade do programa, aumentando a abrangência da utilização quando se fala da dependência de recursos computacionais.

Quanto ao Volume Envolvente escolhido, apesar de apresentar cálculos de atualização e parametrização mais complexos se comparados aos outros volumes dissertados, esta diferença não acarretou a perda de desempenho do sistema. Não foi realizada a implementação de outros envoltórios, porém, no contexto alcançado, o envoltório se mostrou suficiente às necessidades da aplicação. Vale ressaltar que o ajuste adquirido com tal envoltório proporcionou pouquíssimas ocasiões para acontecimento de falsas colisões, o que é fator importantíssimo em uma simulação do mundo real. Uma razão que elevou a qualidade foi a utilização de dois níveis na árvore de OBB's da garra (OBBTree) que se mostrou como a representação ideal para tal elemento, conseguindo o detalhamento do envoltório se equivaler à ocupação espacial utilizada pelos próprios componentes da garra. Mas apesar da boa representação, esta continua sendo uma estrutura aproximada que pode causar falsas colisões.

Quanto à estratégia de *Broad Phase*, a implementação da coerência espacial aliada à semântica do ambiente proporcionou um algoritmo inteligente baseado em heurísticas modeladas para o ambiente específico melhorou significativamente o desempenho da aplicação com relação aos testes de força bruta, mesmo quando os recursos disponibilizados pelo usuário são escassos para a natureza da aplicação (Computação Gráfica). Continuando a oferecer acessibilidade salvo às restrições de portabilidade advindas das requisições técnicas do EAI.

Ainda sobre a aplicação, os testes de detecção cessam diante da primeira colisão encontrada. Isto torna a solução menos real, pois, se um objeto móvel colide com 2 objetos ao mesmo tempo, a colisão é reportada para apenas um deles, o primeiro gerado. Porém esta característica tem por objetivo a eficiência do algoritmo, pois seria necessário continuar o teste para todas as instâncias, aumentando assim o custo computacional.

Apesar de todas as facilidades oferecidas pela EAI, a versão livre do software foi descontinuada pela *Parallelal Graphics,* empresa desenvolvedora, deixando a interface

desprovida do acompanhando da evolução das versões Java. Por isso, para o funcionamento da integração na aplicação do Scorbot Virtual, foi necessária a seguinte configuração de software:

1. Máquina Virtual Java: SDK 1.3.1;
2. Editor Java Eclipse: 2.1.1, última versão da ferramenta de desenvolvimento que permite a utilização da versão 1.3.1 plenamente;
3. Utilização do pacote corteai.zip(EAI2) – Cortona (*Parallelal Graphics*)
4. E utilização da Máquina Virtual da *Microsoft* para visualização das *applets* no browser Internet Explorer, amplamente utilizado.

Este último tópico adiciona um novo aspecto negativo à integração, em virtude da desatualização do pacote. Atualmente, a *microsoft* deixou de desenvolver máquina virtual para aplicativos Java, sendo a *Sun* única a disponibilizar da funcionalidade, que não é compatível com a EAI.

O acontecido foi que todas as empresas licenciadas pela *Sun* para usar a tecnologia Java (compiladores, ferramentas, máquinas virtuais, etc.) têm que obrigatoriamente seguir suas definições, para que a portabilidade do Java seja mantida. Condição não atendida pela Microsoft. Seu compilador produzia código baseado em ActiveX (sendo a EAI justamente desenvolvida para tal padrão de código) e só gerava binários para Windows. Por isso, a *Sun* entrou com um processo para que a Microsoft fosse obrigada a retirar do mercado esse produtos ou corrigi-los. Assim, a Microsoft judicialmente teve garantida a possibilidade de continuar a distribuir, mas não desenvolver sua máquina virtual (*Microsoft*,2007).

Ainda, um detalhamento em nível java/applet é que funcionalidades do pacote java.util (Vetores, etc.) devem ser encapsuladas em classes e declaradas como variáveis privadas. Esta necessidade se deve às restrições de segurança de um navegador Web, restringindo a aplicação ao *SandBox* oferecido à applet.

8 CONCLUSÃO

A carência por um sistema de detecção de colisão no ambiente do Scorbot Virtual, desenvolvido pelo grupo LARVA, motivou a realização das pesquisas deste trabalho que proporcionaram um levantamento abrangente sobre técnicas atuais de detecção de colisão que possuem como foco principal modelos poliédricos B-rep, aplicação de simuladores virtuais para robótica, voltados a detecções eficientes para aplicação em uma página na Internet.

Foi realizada uma revisão bibliografica ampla sobre os trabahos relacionados à detecção de colisão que gerou um esquema de classificação didático e abrangente. Este esquema, agora acrescido da técnica de coerência semântica como parte das técnicas de fase abrangente, estrutura os algoritmos de colisão e propõem uma classificação detalhada para técnicas aplicadas a modelos poliédricos. Além disso, é explanada, de forma didática, cada técnica, provendo um conteúdo abrangente e objetivo com intuito de introdução ao leitor, oferecendo ainda acesso à bibliografias atuais acerca das respectivas técnicas. Diante de toda informação encontrada, pôde-se abstrair que a metodologia de funcionamento dos algoritmos para detecção de colisão não diferem muito quanto à essência das abordagens, todas são construídas com o mesmo propósito, obter um equilíbrio de dois critérios geralmente antagônicos: precisão e tempo de processamento.

Com base no estudo da literatura foi proposta uma solução de detecção de colisão para o Scorbot que utilizou-se de uma nova estratégia, até então não abordada em trabalhos relacionados. Com o objetivo de melhorar a eficiencia do método implementado, o trabalho propôs a concepção de uma fase abrangente diferenciada, utilizando do conceito da “coerência semântica”. *Definiu-se aqui a COERÊNCIA SEMÂNTICA como um conjunto de estratégias que incluiu heurísticas oriundas da avaliação e entendimento criterioso da aplicação, de suas probabilidades intrínsecas de funcionamento (ou seja, do significado da aplicação, daí, “coerência semântica”) que leve a identificação rápida de situações definitivas de colisão ou não-colisão.*

Aliada à exploração da coerência espacial, construiu-se o escopo da fase abrangente levando-se em consideração uma análise e entendimento da aplicação da

detecção de colisão. Análise esta que possibilitou elevar significantemente a eficiência dos testes com relação a detecção por força bruta, onde todos os objetos são testados contra os demais. Muitas vezes o desempenho da nova técnica superou em dobro os testes de força bruta.

Por outro lado, pode-se adiantar que a eficiência obtida de uma abordagem *de* fase abrangente utilizando-se coerência semântica tem como contrapartida uma solução que perde em generalidade e que, portanto pouco pode ser aproveitada para aplicações que não sejam semelhantes à aplicação original. Porém, esta constatação vem de encontro às conclusões da pesquisa, de que a simulação de todos os aspectos que envolvem o sistema de tratamento de colisões não possui uma solução proposta genérica e ótima pra todos os casos. A construção manual das heurísticas se deve ao fato de que cada ambiente virtual possui suas próprias características peculiares que podem ou não proporcionar um corte significativo no número de pares a ser testados. Então o que pode ser interessante para um ambiente, para outro a metodologia não possui a mesma relevância.

A fase abrangente foi seguida pela fase específica(*narrow phase*) que utilizou como estrutura de volume envolvente OBBTrees. Este envoltório se mostrou útil às necessidades da aplicação, proporcionando raros casos de falsa colisão. Este acontecimento se deve às características da estrutura geométrica dos componentes da cena que permitiram uma ocupação espacial muito ajustada pelas OBBs. A garra foi detalhada com uma árvore de dois níveis que representou perfeitamente a estrutura envolvida, obtendo-se uma alta precisão no teste de colisão. Ainda vale ressaltar que a eficiência do Teorema da Separação dos Eixos (SAT), onde a não colisão é certificada quando o primeiro dos 15 eixos de separação possíveis entre duas OBBs é encontrado. A implementação se mostrou adequada aos cálculos fornecendo resultados precisos com tempo de resposta baixo.

Além do algoritmo de detecção de colisão, duas novas funcionalidades foram implementadas, a geração e pega de objetos. A implementação da geração de objetos em cenários variantes oferece uma ferramenta para treino e manipulação através da aleatoriedade das situações possíveis. A importância desta funcionalidade se confirmou através da publicação do artigo “Cenários Variantes não Colidentes em um

Ambiente Virtual Robótico" (Santos *et.al,* 2007*)*. Também nesta etapa, a da geração, a detecção de colisão é um problema, pois, um novo objeto gerado não pode interpenetrar geometricamente os gerados anteriormente. Aqui, mais uma vez, a aplicação do SAT foi eficiente para resolução do problema. O sistema oferece a possibilidade de geração do número de objetos escolhido pelo usuário que são gerados aleatoriamente entre os três tipos geométricos primitivos: esfera, cilindro e paralelepípedo. É possível assim simular a pega e manipulação de elementos na cena, restrita aos tipos pré-definidos. Apesar da restrição, o encapsulamento das funções de gerenciamento de objetos em uma classe modularizou como um componente a funcionalidade permitindo em trabalhos futuros a melhoria do sistema.

Somando todas as aplicações, chegou-se a uma nova interface, que conta com a lista dos objetos gerados, a possibilidade de extrair informação de cada objeto ao duplo clique na lista, a escolha do número de elementos criados, além da caixa de texto que possibilita o *feedback* dos eventos sistema para o usuário.

Por fim, toda implementação está devidamente documentada o que é de relevância e empenho considerável diante das dificuldades iniciais para o entendimento de toda arquitetura, viu-se como tarefa complementar e necessária o levantamento dos principais diagramas de seqüência, digramas de classe e descritivos. Com a descrição das classes e do relacionamento entre elas e com o próprio VRML, os futuros trabalhos serão facilitadas.

8.1 Trabalhos Futuros

Abaixo são listadas algumas propostas que se desdobram do trabalho realizado. As técnicas de simulação até agora implementadas (cinemática direta e inversa, detecção de colisão) cobrem uma parte relevante da simulação de um ambiente robótico real, todavia estas ainda não são suficientes para definir todos os detalhes de uma célula real de aprendizagem.

Uma proposta seria a adição de objetos de maior complexidade, inseridos como um arquivo externo que define um novo objeto VRML de complexidade qualquer. A

adaptação do objeto à implementação poderia ser realizada através da criação de uma engenharia de conversão. Esta *engine*, feita em java, construiria automaticamente a árvore de volumes envolventes para qualquer objeto modelado em B-Rep/VRML adicionando a nova estrutura ao sistema. Tal implementação é bastante ousada e exigindo uma reestruturação do sistema de gerenciamento de objetos, e aumentando a abstração do nível de Orientação a Objetos implementado. Este trabalho resolveria a deficiência atual da implementação na geração de objetos complexos, tornando o ambiente flexível e ajustado a qualquer tipo de necessidade.

Juntamente com a melhoria do sistema, sugere-se projetar um algoritmo para previsão de colisões ao decorrer do caminho de trajetórias programadas pelo usuário. Sabendo-se que nestes casos conhece-se o ponto final da ferramenta (garra), a velocidade da movimentação, bem como as angulações do braço robótico, é possível, sem a necessidade da execução da programação, prever se existirá colisão na trajetória selecionada. A atual implementação oferece uma detecção de colisão discreta, que só pode reportar a colisão no momento em que ela já ocorreu. A proposta de trabalho futuro, ao contrário, se oferece a resolver tal problema antes mesmo que ele já tenha ocorrido, este conceito caracteriza-se por detecção contínua, que será útil apenas na analise das programações do Scorbot, não servindo para manipulação direta do robô resolvida pela discreta. Assim, o sistema ofereceria um pacote completo sobre o sistema de colisões.

Do ponto de vista educacional, uma proposta seria a análise e levantamento de características em cenas geradas (como número de objetos, complexidade dos objetos, disposição, definição do alvo a ser agarrado, etc.) que gerem categorias construídas de acordo com a complexidade na manipulação do Scorbot. Assim seria possível oferecer diferentes estágios de dificuldade selecionáveis para prova da habilidade de manipulação do robô. Tal trabalho implementa a adaptação do ambiente às necessidades de aprendizagem de cada usuário além de melhorar a implementação da geração de cenários variantes.

Como discutido no texto, a tecnologia empregada tende à extinção, sendo assim, um trabalho relevante seria o estudo de novas tecnologias para a migração do robô virtual. Onde todas as características positivas advindas dos trabalhos realizados até o

presente momento possam ser empregadas ou até melhorados pelo uso de novas bibliotecas.

REFERÊNCIAS

ATENCIO, Y. T. P. Esquema De Detecção e Resposta a Colisões Para Animação física Simplificada. Dissertação de Mestrado em Ciências e Engenharia de Sistemas e Computação da Universidade Federal do Rio de Janeiro, p 71, 2005.

EHMANN, S.; LIN, M.C. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In: Computer Graphics Forum, Vol. 20(3), p 500-510,2001.

BRADSHAW , G; O'SULLIVAN, C. Adaptive Medial-Axis Approximation For Sphere-Tree Construction. In: *ACM Transactions, Vol. 23, No.1,* p 1–26, *Janeiro 2004*.

CAREY, R.; BELL, G.. The VRML 2.0 specification. Disponível em http://apia.u-strasbg.fr/vrml/ressources/specs/spec/index.html . Ultimo acesso em 16 nov. 2006, 19:00.

CAREY, R.; BELL, G. The annotated VRML 2.0 reference manual. Addison-Wesley, 1997.

COHEN, J. D.; LIN, M.C; MANOCHA D.; PONAMGI M. K. Interactive And Exact Collision Detection for Large-Scaled Environments. In: Technical report, University of North Carolina at Chapel Hill, p 28, Março 1994.

COHEN, J. D.; LIN, M. C.; MANOCHA, D.; PONAMGI, M. K. I-collide: An interactive and exact collision detection system for large-scale environments. In: Proc. ACM Interactive 3D Graphics Conf., Monterey, California, p. 189-196, 1995.

DIEHL, D. C. Ambiente Virtual para Manipulação de uma Célula Robotizada. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Pontifícia Universidade Católica do Rio Grande do Sul, p 61, 2004.

ESHED ROBOTEC. Scorbot ER-4PC User's Manual. Rosh Ha'ayin Israel, p 38,1982.

FOLEY J.D.; VAN DAM A.; FEINER S.K.; HUGHES J.F. Computer Graphics, principles and practice 2 ed. in C, Addison-Wesley, 1997.

FIGUEIREDO, M.; MARCELINO, L.; TERRENCE, F. A Survey on Collision Detection Techniques for Virtual Environments. In: Proc. of V Symposium in Virtual Reality, Brasil, p 285-307, Outubro 2002.

FIGUEIREDO, M.; FERNANDO, T. An Unified Framework to Solve the Broad and Narrow phases of the Collision Detection Problem in Virtual Prototype Enviroments. In: Proc. of International Conference on Geometric Modeling and Graphics, Salford University, UK, p130-136, 16-18 July 2003.

GOTTSCHALK, S. Collision Queries using Oriented Bounding Boxes, PhD Thesis in the Department of Computer Science, University of North Carolina, Chapel Hill, 2000.

GOTTSCHALK, S., LIN, M AND MANOCHA, D. OBB Tree: A hierarchical structure for rapid interference detection. In: Proc. Siggraph'96, p 171-180, 1996.

GROOVER, M.P. Robótica tecnologia e programação. 1 ed., McGraw-Hill, São Paulo, 1989.

HOFMAM, M.DA S. Tratamento Eficiente de Visibilidade Através de Árvores de Volumes Envolventes. Dissertação de Mestrado do Departamento de Informática Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ), p 148, 2002.

JIMÉNEZ, P.; THOMAS, F.; TORRAS, C. 3D collision detection a survey. In: Computer and Graphics, Volume 25, Number 2, p. 269-285, Abril 2001.

KIMMERLE, S. Tutorial: Real-Time Collision Detection for Dynamic Virtual Environments. In: Proc. of the IEEE Virtual Reality, University of Tübingen, p 18, 2005.

LARVA – Scorbot Virtual Disponível em < www.joinville.udesc.br/~larva/robo > Último acesso 29 set. 2007.

LIN, M.C. Eficient Collision Detection for animation and robotics. Dissertation of Doctor of Philosophy in Electrical Engineering and Computer Sciences of the University of California and Berkeley, p 59 , 1993.

LIN, M.C.; GOTTSCHALK, S. Collision Detection between Geometric Models A survey. In: Proceedings of IMA Conference on Mathematics of Surfaces (San Diego (CA)), vol. 1, pp. 602-608, 1998.

LIN, M.; MANOCHA, D. Collision and proximity queries. In: CRC Handbook of Discrete and Computational Geometry, 2 ed., Boca Raton, FL, 2003.

LIN, M. C.; MANOCHA, D.; COHEN, J.; GOTTSCHALK, S. Collision detection Algorithms and applications. In: Algorithms for Robotic Motion and Manipulation, Wellesley, MA, p. 129-142, 1997.

LUQUE, R. G.; COMBA, J. L.D.; FREITAS, C. M. S. Broad-Phase Collision Detection Using Semi-Adjusting BSP-trees. In: SIBIGRAPI, Brazil, p 8, 2006.

MARRIN, C. Proposal for a VRML 2.0 Informative Annex External Authoring Interface Reference. Silicon Graphics. http://graphcomp.com/info/specs/eai.html, 1997.

MEZGER, J.Tutorial 2 - Collision Handling in Dynamic Simulation Environments. In: Eurographics, p 54, 2005.

MICROSOFT Disponível em <http://www.microsoft.com/mscorp/java/default.mspx> Último acesso 29 set. 2007.

NASCIMENTO, H. P. Detecção de Colisão Entre Objetos Convexos Com Características Geométricas Mutantes Usando a Técnica de Octree Esféricas. Dissertação do curso de Mestrado em Ciência no Curso de Engenharia Eletrônica e Computação do Instituto Tecnológico da Aeronáutica, São José dos Campos, p 93, 2001.

NAKAMURA, F.I.; CELES, W. Detecção hierárquica de colisão em ambientes 3D. In: Workshop of Undergraduate Work, SIBGRAPI, Natal, p 7, 2005.

REDEL R.; HOUNSELL M. DA S. Implementação de Simuladores de Robôs com o Uso da Tecnologia de Realidade Virtual, CBCOMP, Brasil, p 6, 2004.

REDON, S.; KIM, Y.J.; LIN, M.C.; MANOCHA, D. Fast continuous collision detection for articulated models. In: Proc. of ACM Symposium on Solid Modeling and Applications,*p 21, 2004.*

REQUICHA, A.A.G.Representations for rigid solids Theory, methods and systems. In: ACM Computing Surveys, p 437–464, 1980.

RITTER, J. An efficient bounding sphere. In: Graphics Gems, Academic Press, San Diego, CA, p 301-303, 1990.

Rohrmeier, M. VRML 2.0 Robot. Of Institute of Robotics and System Dynamics, 2000. Disponível em <http//www.geocities.com/ResearchTriangle/Lab/8585/robot/robot.html> Último acesso em 27 set. 2006, 19:30.

SANTOS, R.G. HOUNSELL, M.S. DOROW, A. Geração de Cenários Variantes Não Colidentes para um Ambiente Virtual Robótico. Revista Hífen, 2007.

SEIDEL, R. Small Dimensional Linear Programming and Convex Hulls Made Easy. In: Discrete & Computational Geometry Volume 6, Issue 5, p 423-434, 1991.

SOBOTTKA, G.; WEBER, A. Efficient Bounding Volume Hierarchies for Hair Simulation. In: Workshop On Virtual Reality Interaction and Physical Simulation, p.1–10, 2005.

TADDEO, L. DA S. Detecção de colisão utilizando grids e octrees esféricas para ambientes gráficos interativos. Dissertação do Curso de Mestrado em Informática Aplicada da Universidade de Fortaleza, Ceará, p 103, 2005.

VOLUME GRAPHICS. Disponível em <http//www.volumegraphics.com/solutions/> Último acesso 27 out. 2006, 2140.

WATT, A. H. 3D Computer Graphics. 3 ed. Harlow : Addison-Wesley, p 570, 2000

ZEID, I. CAD/CAM Theory and Pratice, McGraw-Hill, United States, p 576,1991.

ZHANG, X.; LEE, M. Y.; KIM, J. Interactive continuous collision detection for Non-Convex Polyhedra. In Visual Comput ., p 749 – 760, 2006.

ZWIRTES, R. A. Cinemática Inversa para Controle da Abordagem de Órgãos Terminais de Robôs Manipuladores. Trabalho de Conclusão de Curso. (Graduação em Bacharelado em Ciência da Computação) - Universidade do Estado de Santa Catarina.

ANEXOS

ANEXO A - Arquitetura do Sistema

Tendo em vista a quantidade de projetos realizados anteriormente sobre o Scorbot Virtual, uma necessidade primordial deste trabalho é a realização do levantamento da arquitetura do sistema existente e estudo do código fonte. Com esta estruturação e documentação a tarefa conseguinte consistiu simplesmente no projeto da adaptação das classes Java necessárias para as novas funcionalidades a serem inseridas no ambiente virtual.

A análise do aplicativo foi feita através da criação de um diagrama de casos de uso que demonstra de forma abrangente as funcionalidades do sistema, com os diagramas de seqüência para cada caso de uso e com o diagrama de classes das principais relações.

Anexo A.1 Diagrama de Caso de Uso

Na figura 42 o ator é definido como sendo o usuário do ambiente virtual, que estará interagindo com o sistema a fim de executar as operações gerais representadas pelos casos de uso do diagrama. Estes elementos seguem detalhados nos próximos tópicos, juntamente com os diagramas de seqüência relacionados.

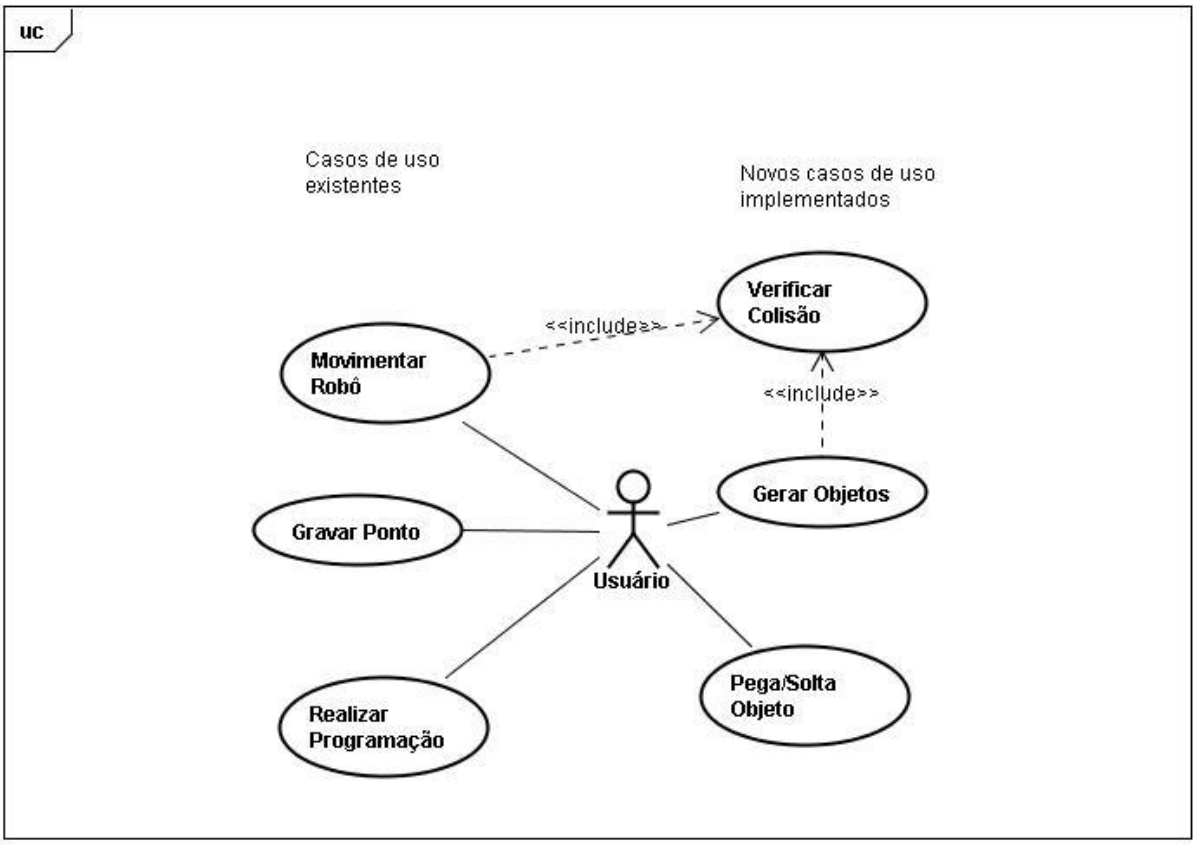


Figura 42 – Diagrama de Caso de Uso do Scorbot Virtual

Anexo A.2 Digramas de Seqüência

O diagrama de seqüência realiza a modelagem comportamental do sistema. Assim, são definidas as relações das comunicações entre classes.

Anexo A.2.1 Diagrama de Seqüência de “Movimentar Robô”

A movimentação do Robô por parte do usuário pode acontecer de duas formas: via applet, através dos botões do painel horizontal (ver figura 43), ou por manipulação direta do modelo virtual. Para os dois casos, a seqüência de operações se difere sutilmente, como pode ser visto nas representações nas figuras 44 e 45, que mostram a seqüência de comunicação entre as classes envolvidas.

Figura 43 – Interface – Manipulação do Braço

Iniciando por quando os eventos iniciam via applet, o método alterarAngulacao() é um nome fantasioso definido aqui para representar uma série de ações que possuem reações similares do sistema, porém em pontos diferente, por exemplo, uma alteração de valores pode ser atribuída ao movimento de elevação da garra ou na rotação da base. O mesmo ocorre com addValor(), que é o evento de comunicação entre a interface PrincipaisControles.java e o objeto de Robô.java, que é a entidade que mapeia e representa os parâmetros do robô.

Ao registrar uma mudança, o objeto da classe Robô.java, através da representação utilizada aqui setEvent(), seta o valor modificado no evento de entrada do nó Javascript de Robô.wrl, este por sua vez realiza todos os cálculos de cinemática direta (Zwiertes, 2004) atualizando a geometria do Scorbot e atualizando os eventos de saida necessários que são ouvidos pelas classes que implementam o VRMLListener (VRMLEventChanged()).

Por serem applets distintas e pelas restrições de segurança inerente a este tipo de aplicação WEB, PrincipaisControles.java e PrincipaisComandos.java instanciam objetos diferentes de Robô.java, sendo assim, é notório que não existe comunicação entres as duas interfaces salvo o mundo VRML.

Quando a transformação geométrica se dá pela manipulação direta do braço via *mouse,* o fluxo é idêntico, desconsiderando os eventos de entrada do VRML via Java, porém, utilizando os procedimentos internos da linguagem de descrição.

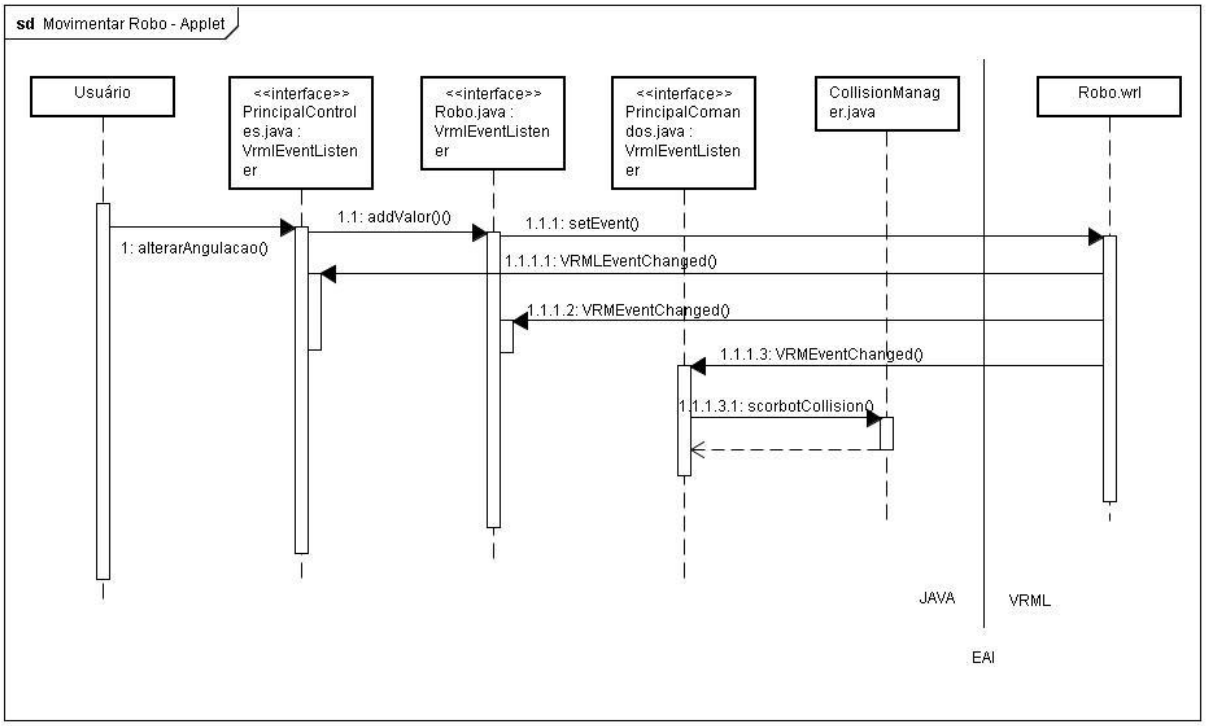


Figura 44 – Diagrama de Seqüência – Movimentar Robô/Applet

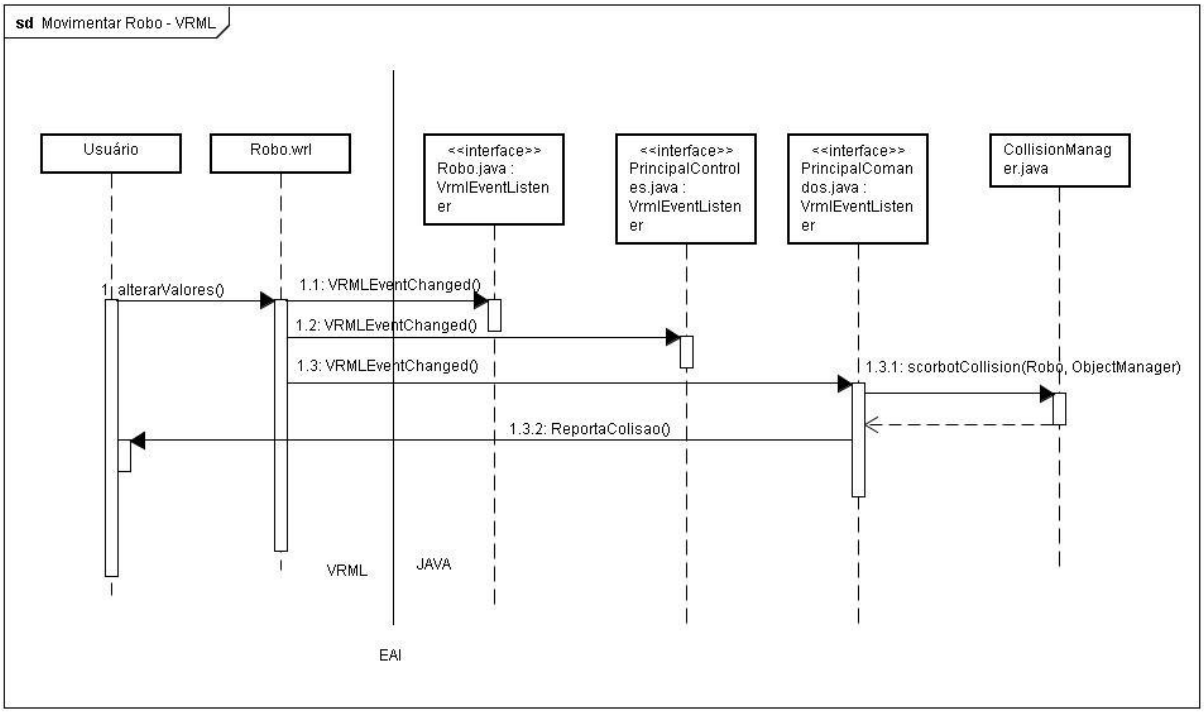


Figura 45 – Diagrama de Seqüência – Movimentar Robô/VRML

Anexo A.2.2 Diagrama de Seqüência de “Gravar Ponto”

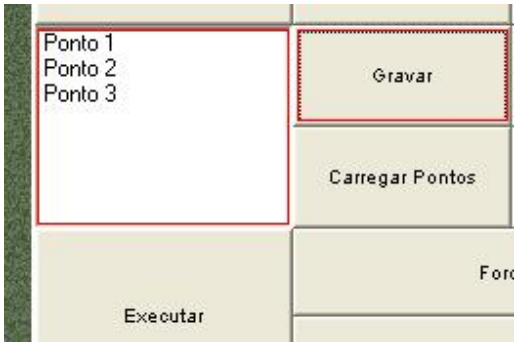


Figura 46 – Interface – Gravar Pontos

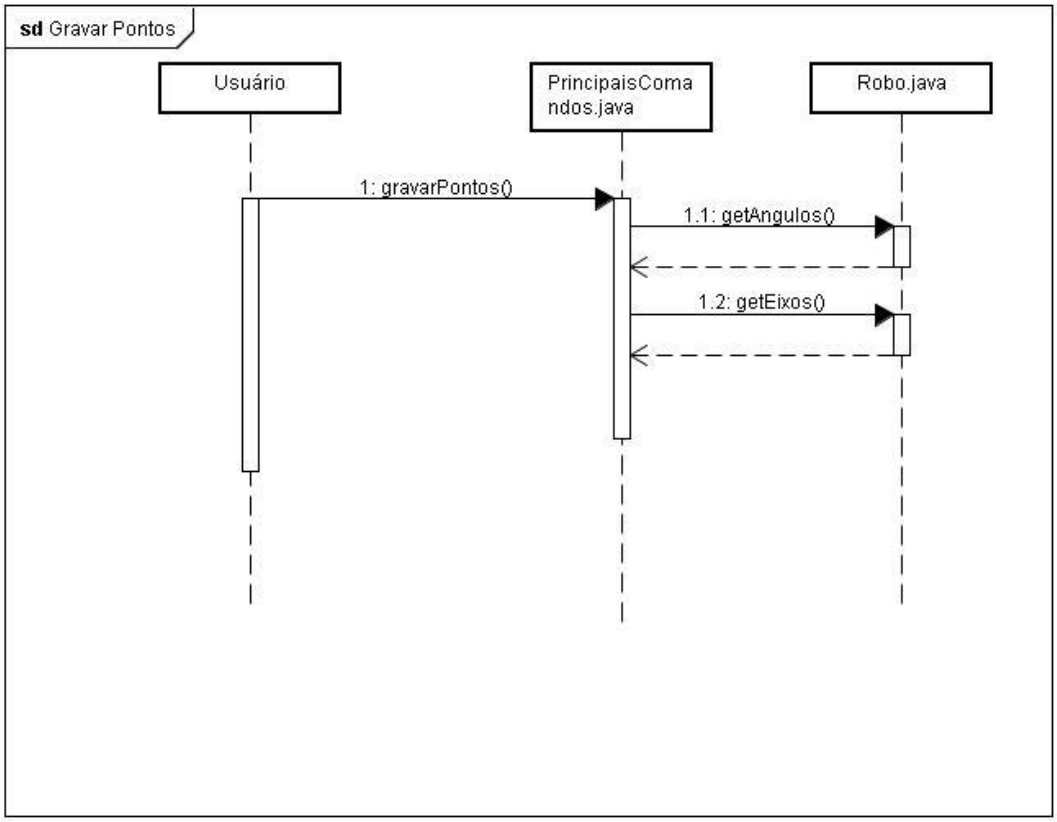


Figura 47 – Diagrama de Seqüência – Gravar Ponto

O objetivo da gravação é oferecer uma lista de pontos de uma trajetória realizada do braço robótico para, pela seleção desejada, criar programações que podem ser executadas pelo Scorbot através da cinemática inversa, implementada por Zwiertes (2004). A operação é simples, ao clique no botão Gravar (assinalado na figura 46), as angulações e eixo reportados por Robô.java são salvas em um objeto da entidade Pontos.java, para então adicionar sua referencia na Lista de pontos do mesmo painel (figura 46). A figura 47 ilustra a seqüência dos eventos e objetos envolvidos.

Anexo A.2.3 Diagrama de Seqüência de “Realizar Programação”

Com uma lista de opções de pontos a alcançar, é possível programar *n* possíveis alternativas de trajetórias para o manipulador. Com esse fim o botão IrParaPosicao() grava na lista de comandos (quadro de texto da figura 48) a seqüência desejada de acordo com os pontos disponíveis previamente gravados. Outra opção a ser adicionada na programação é a pega de objeto, através dos botões abre e fecha garra.

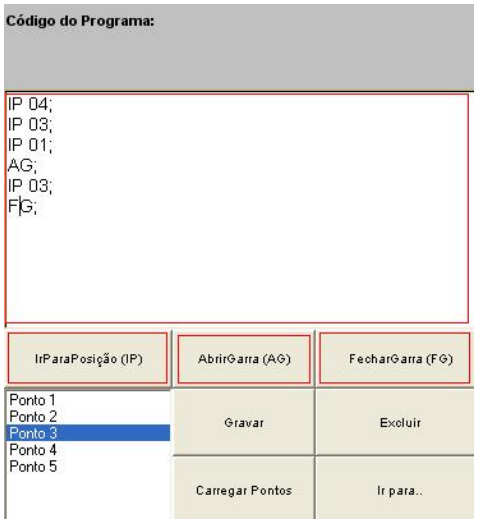


Figura 48 – Interface – Realizar Programação

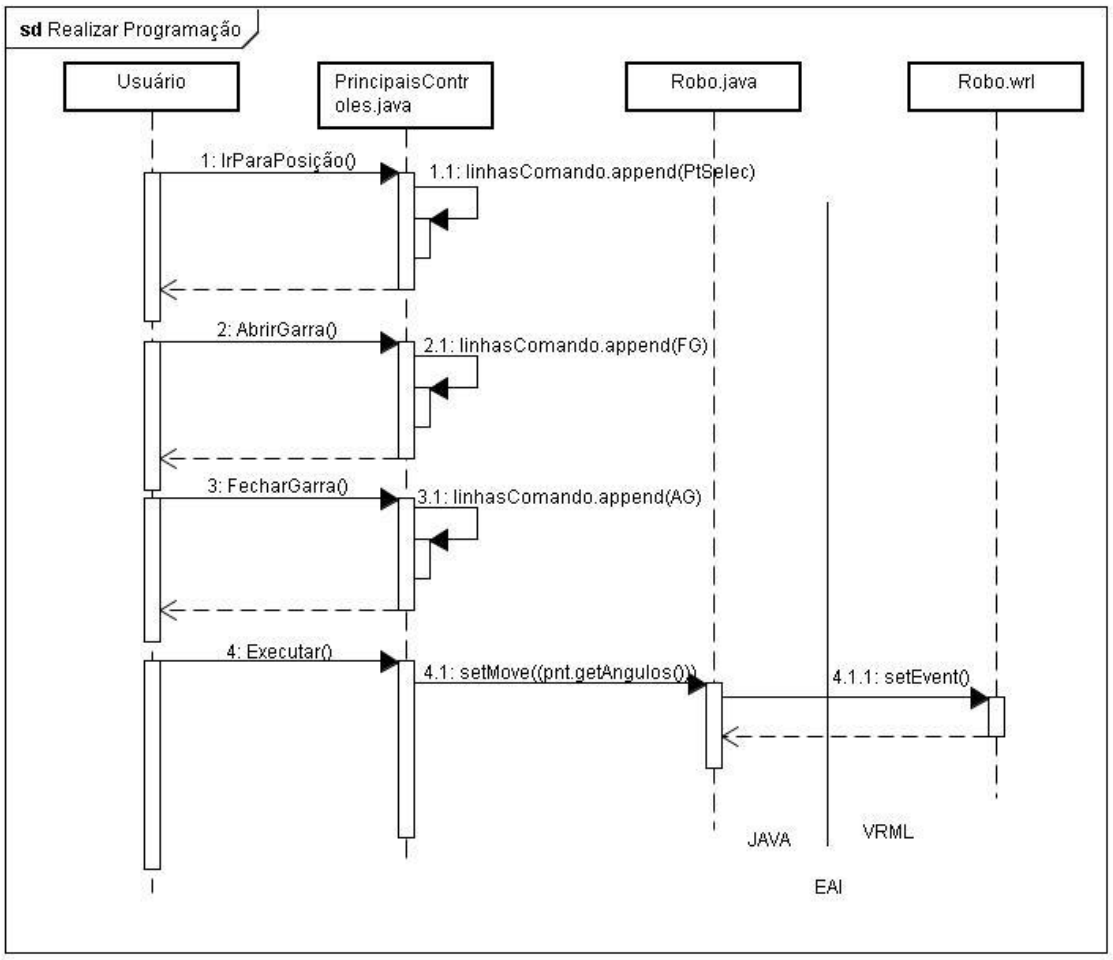


Figura 49 – Diagrama de Seqüência – Realizar Programação

Anexo A.2.4 Diagrama de Seqüência de “Gerar Objetos”

A geração aleatória de objetos é realizada pelo botão “Gerar Objetos” (figura 40) renderizado pela classe PrincipaisComandos.java que envia uma solicitação com o número de objetos para ObjectManager.java que por usa vez realiza todas as operações de criação, validação e inserção dos objetos na cena. Para realizar a validação de um novo objeto criado, conta-se com a classes Collision.java que verifica, através da Obb correspondente, se o novo elemento criado colide com um precursor ou com o próprio robô, e em qual lado da mesa se encontra, isto através de uma Obb atribuída a cada lado da mesa.

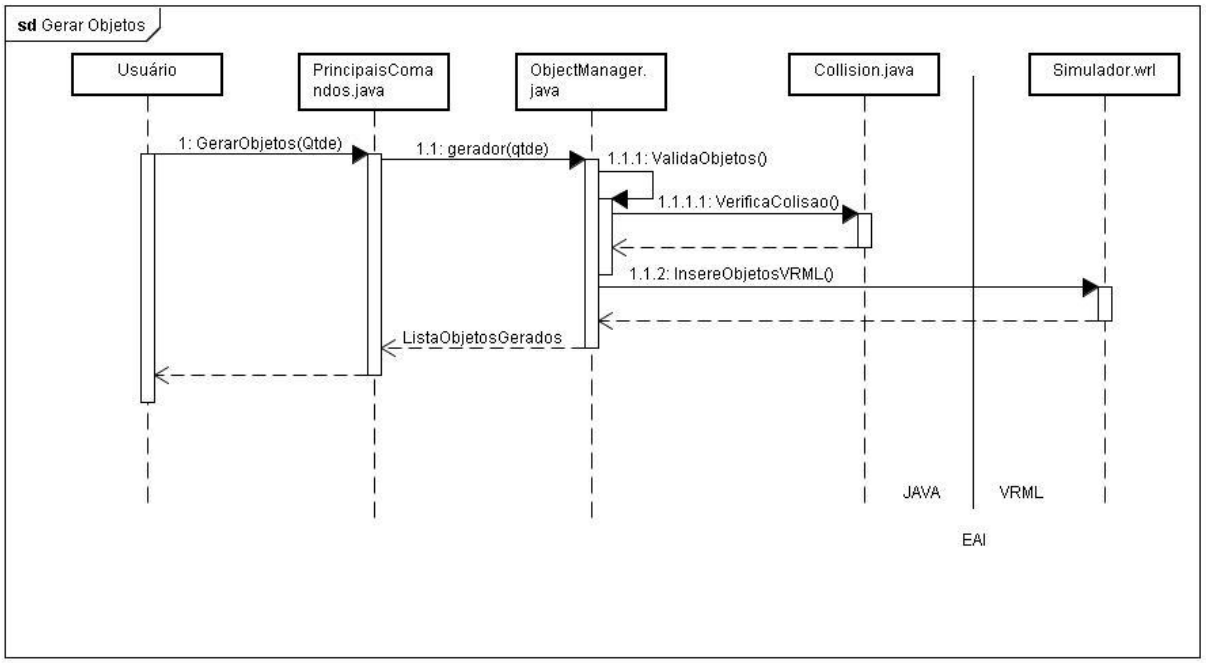


Figura 50 – Diagrama de Seqüência – Gerar Objetos

Anexo A.2.5 Diagrama de Seqüência de “Pega/Solta Objetos”

Por meio de qualquer um dos botões de fecha-abre garra existentes na interface, uma solicitação é enviada a classe Objectmanager.java. Que trata a solicitação adcionando ao elemento alvo uma Route via EAI que adiciona nele os movimentos de translação da garra. Na operação de soltura, esse Route é excluído e passa a o objeto volta à lista de lado correspondente ao seu posicionamento. O nome das funções na classe são respectivamente graspPart() e releasePart().

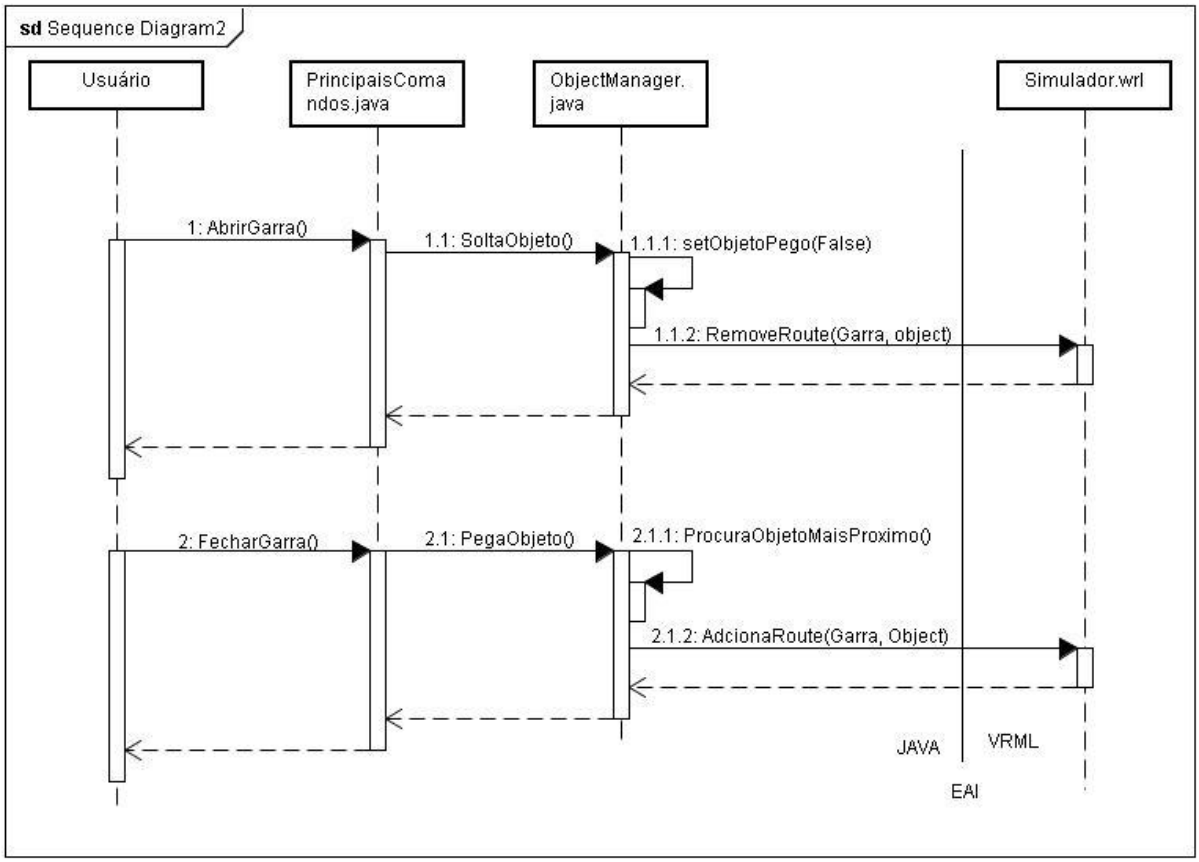


Figura 51 – Diagrama de Seqüência – Pegar/Soltar Objetos

Anexo A.3 Diagrama de Classes

A figura 52 demonstra a relação entre as principais classes do ponto de vista da interface com o VRML. Nela os objetos PrincipaisComandos e PrincipaisControles utilizam a classe Browser, provida pelo EAI, para buscar todas as referencias ao mundo VRML instanciadas em seus escopos.

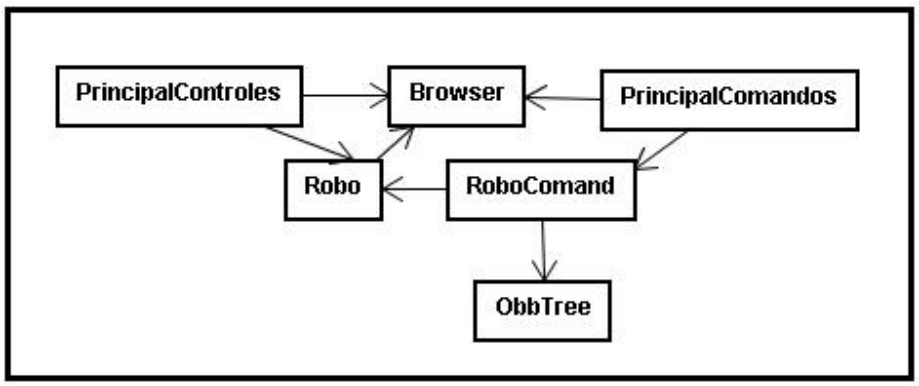


Figura 52 – Diagrama de Classes Simplificado Acerca Interface

A figura 53 mostra a relação da classe gerenciadora de todos os eventos e parâmetros dos objetos, ObjectManager, com as demais classes de entidade dos objetos. Aqui uma característica importante da Orientação a Objetos permite que novos elementos sejam incluídos, bastando ser criada sua nova classe com a OBBTree correspondente.

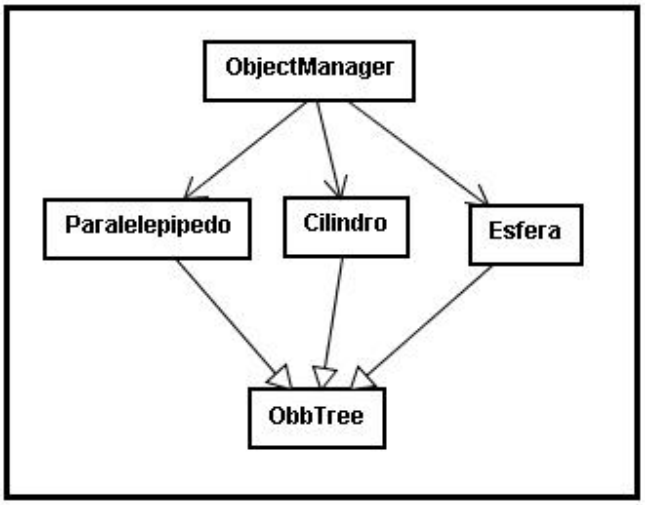


Figura 53 – Diagrama de Classes Simplificado Acerca Gerenciador de Objetos

Usando como ponto principal para representação a detecção de colisão, a figura 54 mostra o diagrama de classes onde as ações ocorrem em Principais comando que inicia o teste de colisão projetado pelo Scorbot e encapsulado por CollisionManager passando como parâmetros os objetos que referencia de ObjectManager, que possui todas as informações sobre os objetos disposto e agarrados, e o RobôComand.java com todas as parametrizações do robô Virtual. A classe Collision manager por sua vez, toma as OBBtress necessárias destas ultimas classes citadas e realiza os testes projetados.

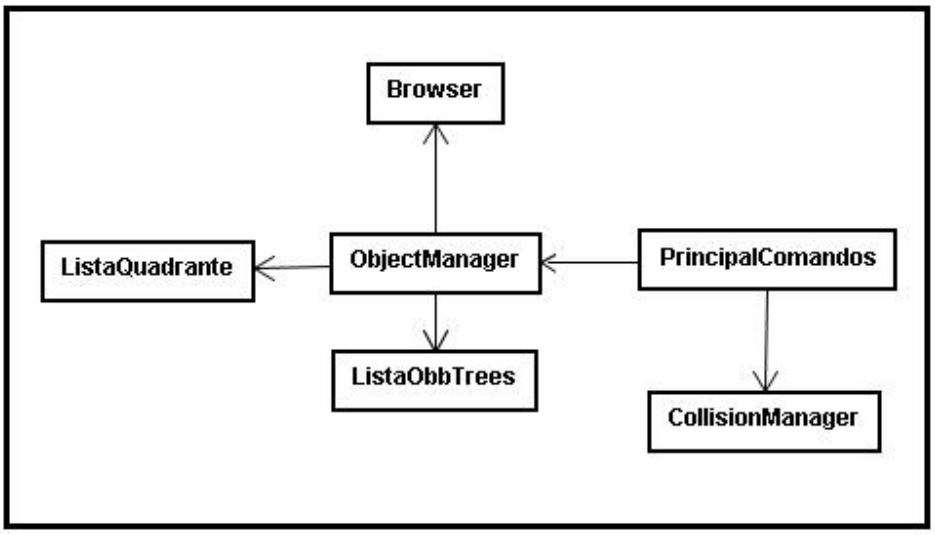


Figura 54 – Diagrama de Classes Simplificado Acerca Detecção de Colisão

Classes Precursoras:

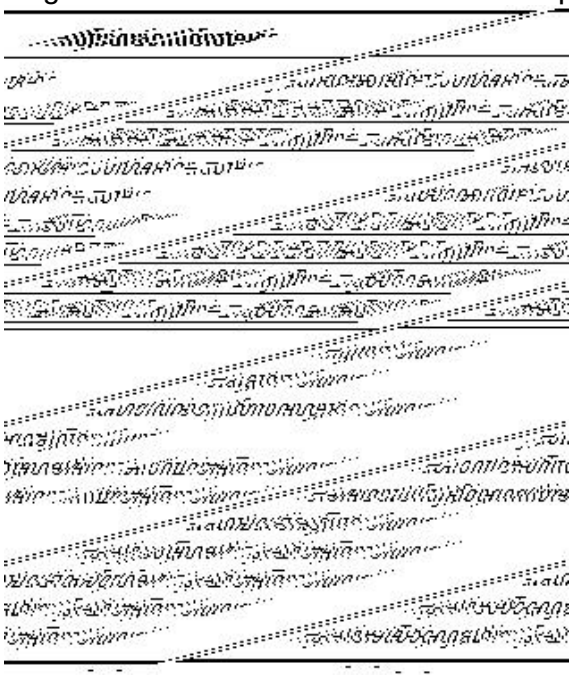
PrincipaisControles.java

Classe de interface que renderiza a applet horizontal do ambiente robótico (figura 43). Por esta interface são inseridos e visualizados os valores das transformações geométricas do braço robótico.

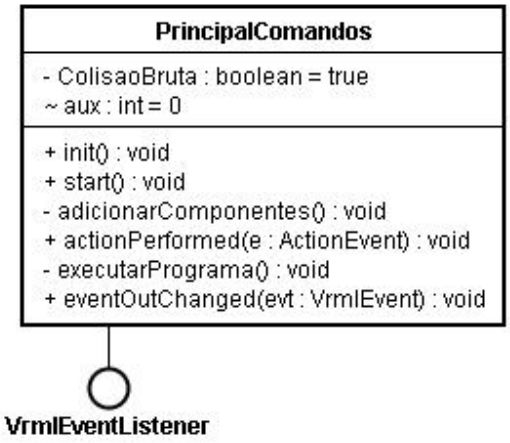
PrincipaisComandos.java

Realiza a interface de conexão com todas as propriedades de programação do ambiente virtual, proporcionando gravação de pontos de localizaçao da garra, programação de trajetórias, dispõe da geração de objetos, bem como traz um *feedback* ao usuário quanto as suas ações que afetam o ambiente robótico. Por ser apenas uma classe de interface, todos os métodos citados são chamados ao clicar dos botões, no método ouvinte *actionPerformed*. O método atualizaTexto atualiza a informação da interface mediante a alteração de qualquer evento no mundo virtual.

Quadro 10 – Diagrama de Classes Detalhado - PrincipaisControles.



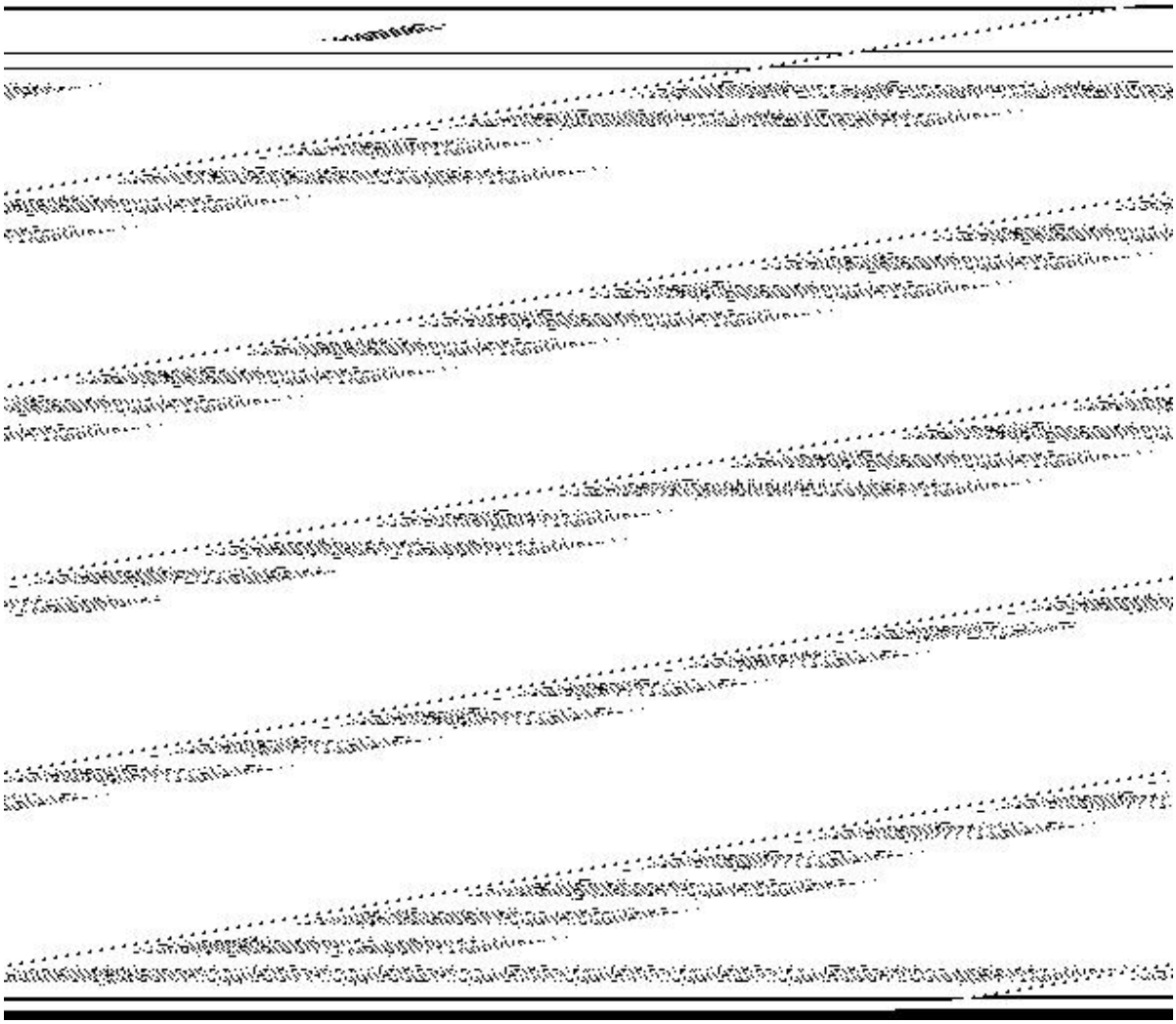
Quadro 11 – Diagrama de Classes Detalhado - PrincipaisComandos.



robo.java

Entidade que realiza toda comunicação com os eventos ocorridos com o manipulador robótico no ambiente virtual VRML.

Quadro 12 – Diagrama de Classes Detalhado - Robô.java.



Classes adcionadas:

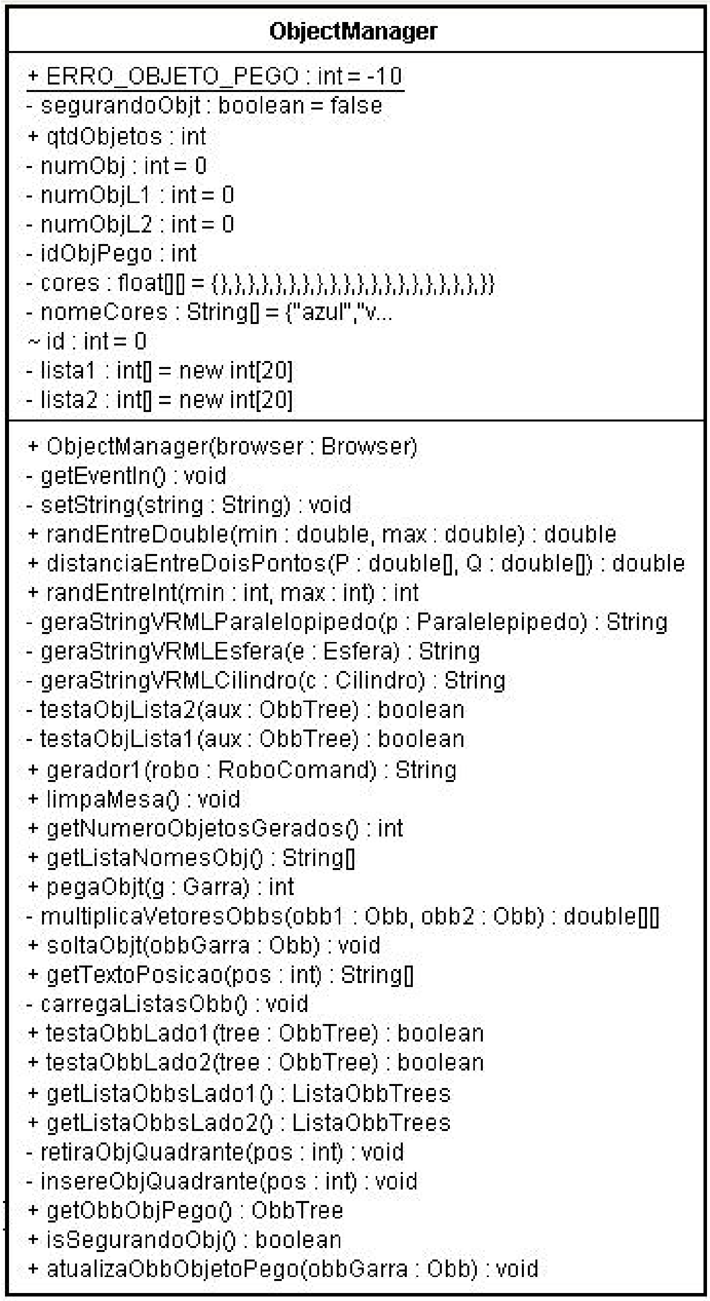
roboComand.java

Classe que extende as propriedades de robo.java, acrescentando métodos que permitem a esta “ouvir” os eventos referentes às alterações de parâmetros das OBB's dos elos do robô, calculadas no nó javascritpt de robo.wrl. Como a *applet* de PrincipaisControles.java também utiliza a mesma classe pai, houve a necessidade desta herança para que não fosse redundante a realização das atualizações dos valores de OBBs, melhorando assim desempenho da aplicação.

ObjectManager.java

Esta classe realiza todo o gerenciamento dos comportamentos relacionados aos objetos inseridos no ambiente, além de fazer a interação desses com o VRML. Entre as funções mais importantes estão os métodos de geração aleatória de elementos na mesa além da preensão e solta destes.

Quadro 13 – Diagrama de Classes Detalhado - PartManager.java.



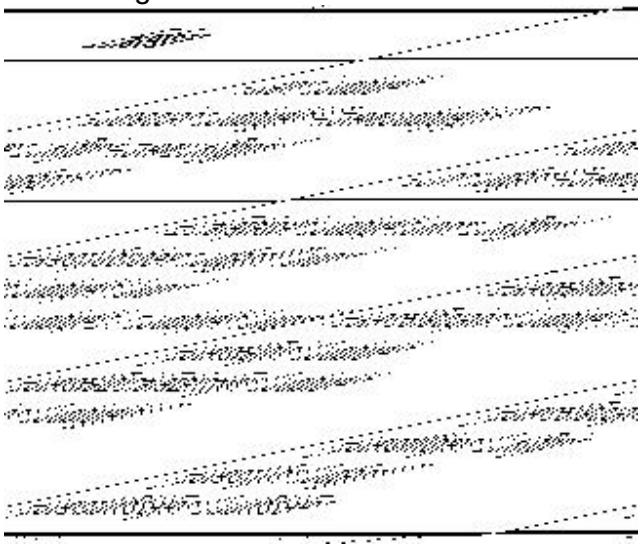
Paralelepípedo.java

Entidade que representa todos os parâmetros de um paralelepípedo, inclusive a OBBTree que o representa.

Esfera.java

Entidade que representa todos os parâmetros de uma esfera, inclusive a OBBTree que a representa.

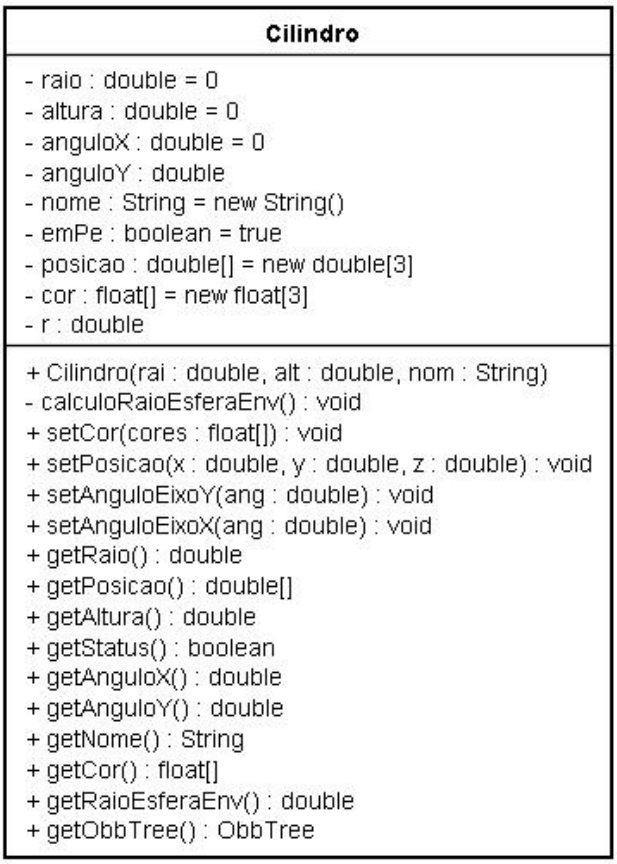
Quadro 14 – Diagrama de Classes Detalhado - Esfera.java.



Cilindro.java

Entidade que representa todos os parametros de um cilindro, inclusive a OBBTree que o representa.

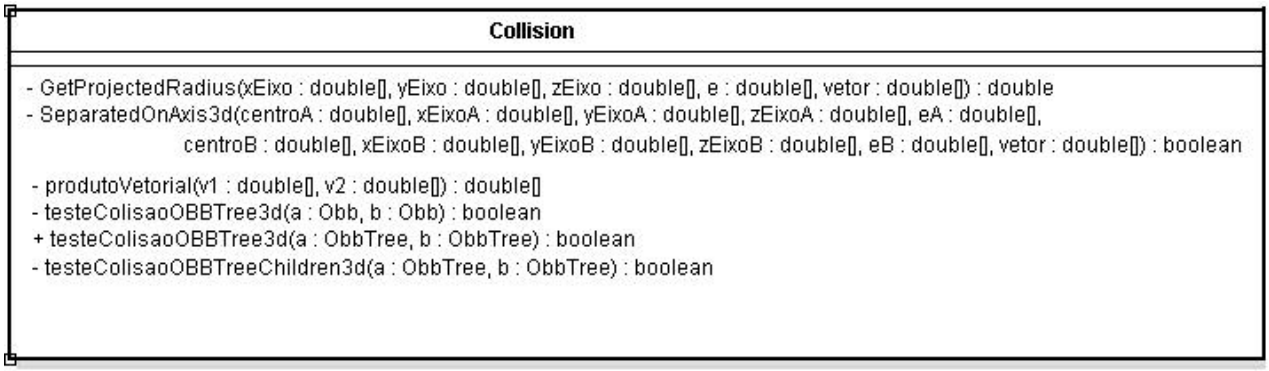
Quadro 15 – Diagrama de Classes Detalhado - Cilindro.java.



Collision.java

Aqui são realizados todos os tratamentos de colisão não específicos ao Scorbot Virtual. Esta classe se dispõe como uma biblioteca de colisão que atualmente oferece a capacidade de realizar cálculos de colisão entre OBBTrees, que é a estrutura de Volumes Envolvente escolhido para o robô.

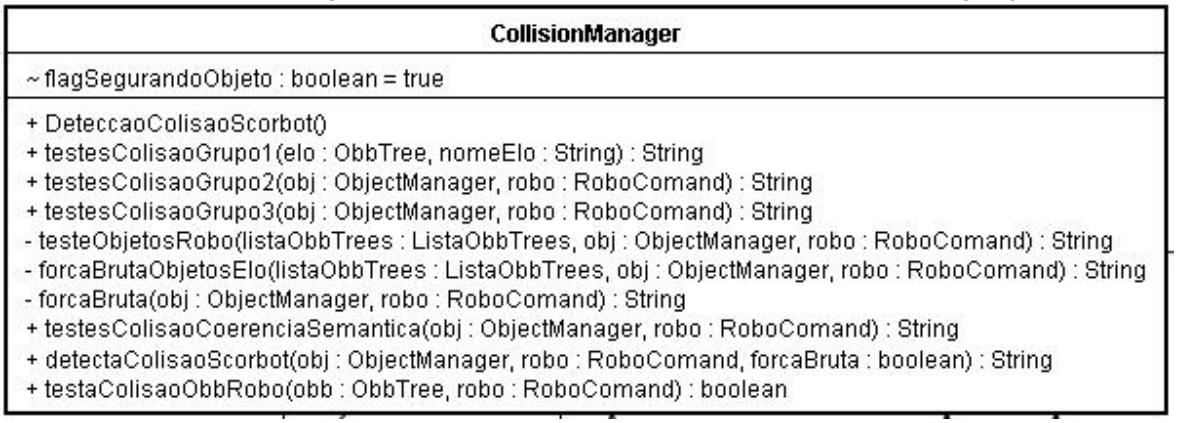
Quadro 16 – Diagrama de Classes Detalhado - Collision.java.



CollisionManager.java

Implementa a estratégia de detecção de colisão a ser utilizada para o ambiente Virtual em questão. Através desta entidade é possível encapsular a funcionalidade, recebendo como parâmetros para instanciação apenas os objetos que representam os objetos gerados dinamicamente (*ObjectManager.java*) e o comportamento e parâmetros do robô (*RoboComand.java*).

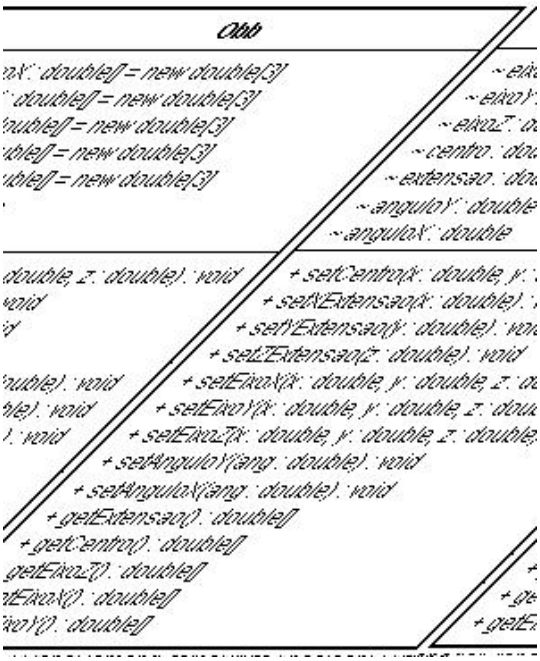
Quadro 17 – Diagrama de Classes Detalhado - CollisionManager.java.



Obb.java

Entidade que representa as OBB's do sistema. Trazendo como atributos os parâmetros deste VE, tais como extensões da caixa, eixos de orientação e ponto central.

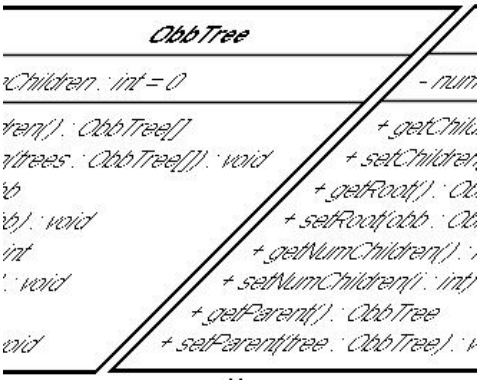
Quadro 18 – Diagrama de Classes Detalhado - Obb.java.



OBBTree.java

Entidade que organiza as Obb's em estrutura de árvores que representam um objeto.

Quadro 19 – Diagrama de Classes Detalhado - Obb.java.



ListaObjetos.java

Utilizada por ObjectManager.java. Armazena todos os objetos de uma geração.

ListaOBBTrees.java

Utilizada por ObjectManager.java. Armazena todas as estruturas de VE's de todos os objetos de um lado da mesa.