CRISTIANE LUCY RODOLFO

USO DE TÉCNICAS DE DETECÇÃO DE COLISÃO DE AVATARES PARA UM AMBIENTE VIRTUAL

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para a obtenção do título de Mestre em Ciência da Computação.

Orientador:

Prof. Dr. José Remo Ferreira Brega

MARÍLIA
2006

CRISTIANE LUCY RODOLFO

USO DE TÉCNICAS DE DETECÇÃO DE COLISÃO DE AVATARES PARA UM AMBIENTE VIRTUAL

Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM/FEESR, para a obtenção do Título de Mestre em Ciência da Computação: Área de concentração: Realidade Virtual.

Resultado: _____

ORIENTADOR. Prof. Dr. José Remo Ferreira Brega

1º EXAMINADOR_____

2º EXAMINADOR_____

Marília, de de 2006

AGRADECIMENTOS

A Deus que me abençoou com o dom da dedicação aos estudos e pelo sentimento de prazer pela busca constante do conhecimento.

A meus pais Osvaldo e Inês, alicerce de minha vida, que sempre confiaram em meus sonhos, e me ensinaram a acreditar que é possível realizá-los, além de me mostrarem que perseverança e honestidade são as ferramentas ideais para atingirmos nossas metas.

Aos meus irmãos Giovana e Júnior pela compreensão, carinho durante este percurso.

Ao meu namorado Ademir, que sempre está ao meu lado paciente nas horas de angústia, dizendo sempre que devemos viver um dia de cada vez aproveitando o máximo disso.

Aos meus amigos pela força, incentivo, e compreensão pela minha ausência em determinado momento; em especial a Elaine minha companheira de estudo e a Luzia que presente ou virtualmente sempre esteve disposta a me ajudar.

Aos professores, em especial o Sementille pelos ensinamentos transmitidos e dedicação para com seus alunos.

A meus tios e primos Francisco, Lourdes, Willian e Gustavo pelo acolhimento e carinho.

Em especial ao meu orientador José Remo Ferreira Brega, cuja força e paciência dispensadas, foram de grande importância no período em que estive dedicado ao desenvolvimento desse trabalho.

É muito difícil agradecer a todos sem esquecer, inevitavelmente, de alguém, mas aos que esqueci meu pedido de perdão e meus agradecimentos sinceros!

A todos vocês obrigada de coração!

Eu aprendi que o sucesso deve ser medido não tanto pela posição que alguém alcançou na vida e sim pelos obstáculos que teve que ultrapassar enquanto tentava alcançar o sucesso.

Booker T. Washington

RODOLFO, Cristiane Lucy – Uso de Técnicas de Detecção de Colisão de Avatares para um Ambiente Virtual– 2006 106 f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marilia, Fundação de Ensino Eurípides Soares da Rocha, 2006.

RESUMO

A realidade virtual está cada dia mais presente na vida das pessoas, esta presença se dá através da imersão, interação e envolvimento com o Ambiente Virtual. Grande parte dos objetos móveis em cena de Realidade Virtual são Humanos Virtuais, chamados de avatar. O avatar representa o usuário na cena, podendo possuir características físicas e comportamentais. Além dessas características há um problema a ser tratado, a Detecção de Colisão, que ocorre com o movimento dos objetos em ambientes virtuais. A Detecção de Colisão é uma técnica custosa que permite identificar quando objetos virtuais estão se tocando, sendo um fator importante para a obtenção do realismo de um Ambiente Virtual. Assim, este trabalho tem o objetivo de apresentar algumas das principais técnicas que poderiam ser empregadas na detecção de colisão, permitindo fazer uma comparação das técnicas de acordo com o avatar escolhido. No entanto, existem técnicas que são computacionalmente inviáveis por demandar muito processamento, inviabilizando a detecção de colisão em tempo real.

Palavras – chave: Avatar, Ambiente Virtual, Detecção de Colisão.

RODOLFO, Cristiane Lucy – Uso de Técnicas de Detecção de Colisão de Avatares para um Ambiente Virtual– 2006 106 f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marilia, Fundação de Ensino Eurípides Soares da Rocha, 2006.

ABSTRACT

The virtual reality is each more present in people's lives, this presence happens through the immersion, interaction and envolvement with the virtual environment. Most part of mobile objects in scene of virtual reality are human virtual, called avatar. Avatar represents the user in scene, processing, physics and behavior characteristic. Besides these characteristics there is a problem to be treated, the Detention of Collision, that occurs with the movement of objects in virtual environments. The Detention of Collision is one hard technique that allows to identify when virtual objects are each other, being an important factor to obtain a realism of the virtual environment. So, this work has the objective to show some of the main techniques that could be used in the detection of detection, being allowed to more a comparison of the techniques according to the chosen avatar. However, techniques exist that are computational impracticable for demanding much processing, happening the detention of collision in real time.

Keywords: avatar, virtual environment, detention of collision.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELA

LISTA DE ABREVIATURAS

3D – Tridimensional

API – Application Programmer Interface

ASCII – American Standard Code for Information Interchange

AV – Ambiente Virtual

DOF - Degree of Freedom

FPS – Frames por segundo

HV – Humano Virtual

RV – Realidade Virtual

VRML – Virtual Reality Modeling Language

WTK – WorldToolKit

WU – WorldUp Modeler

SUMÁRIO

INTRODUÇÃO

A Realidade Virtual (RV) é uma interface avançada para aplicações computacionais, onde, o usuário pode navegar e interagir em tempo real em um ambiente tridimensional gerado por computador. A RV é uma junção dos conceitos de imersão, interação e envolvimento; onde a imersão é o sentimento que o usuário tem de estar dentro do ambiente; a interação é a capacidade de fazer com que as pessoas vejam as cenas mudarem em resposta aos seus comandos; e o envolvimento, por sua vez, está ligado com o grau de motivação para o engajamento de uma pessoa com determinada atividade. (KIRNER, 2004).

Para que o usuário possa interagir uns com os outros e participar de um mundo sintético gerado pelo computador é necessário inserí-lo em um Ambiente Virtual (AV), que é projetado para simular tanto um ambiente imaginário como um ambiente real.

Com o aumento da popularidade nos últimos anos, ficou evidente o esforço para o desenvolvimento de ferramentas que permitam a criação de AVs com maior grau de realismo. Uma característica comum nesses ambientes é a representação de cada usuário por um objeto, para que ele possa interagir com o mundo no qual está inserido. Este objeto poderá ter as características físicas e movimentos semelhantes à de um humano.

Os programadores têm continuamente buscado caminhos para simular o mundo real com precisão. Para tanto é necessário resolver problemas relacionados com a movimentação dos objetos no Ambiente Virtual. Dessa forma, a necessidade de detectar colisões é imprescindível para dar ao usuário uma maior sensação de interação.

A Detecção de Colisão contribui para a obtenção do realismo de um mundo virtual, determinando a capacidade de interação dos objetos em um dado tempo. O desafio é definir

qual será a técnica ou a combinação de técnicas de Detecção de Colisão ideais para uma aplicação levando em consideração a complexidade do ambiente.

1.1 Objetivos

De acordo com o que foi mencionado, os objetivos desse trabalho são:

- Estudar as técnicas de Detecção de Colisão, observando qual se encaixa melhor no uso de Humano Virtual (HV).
- Elaborar um AV, constituído de modelos diversificados de HV.
- Realizar a implementação de um AV, de acordo com o item anterior, utilizando técnicas de Detecção de Colisão aplicada em cada HV.

1.2 Organização do Trabalho

Para proporcionar uma melhor compreensão, este trabalho está estruturado em seis capítulos.

No Capítulo 1 será apresentado o conceito de Humanos Virtuais, seguido de seu grau de liberdade e a representação que o humano pode ter.

No Capítulo 2 será apresentado a definição de Avatar, abrangendo suas representações e utilizações, seguido da definição de aparência, movimento, interatividade, comunicação e gestos.

No Capítulo 3 será abordada a Detecção de Colisão, com sua definição, características, técnicas, das mais simples até as mais complexas, e estratégias de refinamento de Detecção de Colisão.

No Capítulo 4 será apresentado o WorldToolKit, com suas definições, utilizações, classes e funções, seguido do tratamento que o WTK possui para detectar colisão de avatar.

O Capítulo 5 descreve com detalhes a aplicação implementada, os recursos de software, telas e código fonte.

No Capítulo 6 é apresentada a análise dos protótipos desenvolvidos, comparando o desempenho entre as técnicas adotadas, demonstrando os resultados obtidos.

Em seguida, é apresentada a conclusão desse trabalho mediante os estudos realizados, juntamente com as sugestões para trabalhos futuros.

CAPÍTULO 1 - HUMANOS VIRTUAIS

Na década de setenta, pesquisadores estavam prevendo que no futuro as tarefas humanas iriam ser realizadas por máquinas, desde os jogos até os robôs autônomos. Hoje as pesquisas estão bem próximas daquela realidade, onde, buscando a autonomia das máquinas, fez com que elas auxiliassem as pessoas em tarefas de alto risco, representadas através do Humano Virtual, evitando problemas com presença, segurança e limitações físicas (BADLER, 1997).

1.1 Conceito

Segundo Badler (1997), HV são modelos computacionais de pessoas, que podem ser usados como substitutos de "coisas reais", como por exemplo, em teste ergonômico de projetos computadorizados de veículos, áreas de trabalho e linhas de montagem, entre outros.

Muitas são as razões que conduzem ao desenvolvimento de modelos humanos, sendo necessário levarem em consideração vários aspectos; tais como:

- Análise de fatores humanos (altura, capacidade, comportamento e desempenho);
- Agentes e avatares em tempo real, como serão vistos nos próximos capítulos (pessoas vêm de diferentes culturas e possuem personalidades distintas, sendo necessário refletirem esta diversidade nos HV, desde a influência na aparência, bem como as reações e escolhas dos mesmos);

- Entendimento e geração de instruções (seres humanos se comunicam entre si com uma enorme riqueza de linguagem, sentidos, e experiências; isto precisa também ser estendido para agentes e avatares);
- Simulações biomédicas (a máquina humana é um complexo de estruturas físicas e funções. Para entender o comportamento, respostas fisiológicas e lesões humanas é preciso representar sistemas biológicos);
- Análise de forma e movimento (o que é percebido quando é visto ou sentido deve ser passado para o modelo do mundo físico, deixando mais próximo da realidade, com formas geométricas e deformação do objeto).

Para construir um HV, existem várias noções para o que é chamado de fidelidade virtual, mas ela depende da aplicação. Por exemplo, fidelidade quanto à altura, capacidade, limites de elasticidade das articulações; são essenciais para certos tipos de aplicação, enquanto para jogos e simulação militar, o mais importante é a fidelidade em termos de tempo. Existem, graduações de fidelidade que são inerentes aos modelos.

Alguns modelos podem ser muito avançados em determinada dimensão, mas ineficientes em outras. De uma forma bastante genérica, Badler (1997), caracteriza os modelos de representação de HV em cinco dimensões:

- Aparência: pode ser um desenho animado até um modelo físico exato;
- Funcionalidade: ações de um desenho até as limitações humanas;
- Tempo: geração off-line à produção em tempo real;
- Autonomia: animação direcionada à animação Inteligente;
- Individualidade: especificação pessoal até personalidades variadas.

Humanos Virtuais são diferentes de um simples desenho animado. Qualquer pessoa quando vai ao cinema pode ver personagens sintéticos realísticos (alienígenas, brinquedos, dinossauros, etc.), mas estes personagens são criados tipicamente para uma cena ou um filme

e não deveriam ser reutilizados (exceto eventualmente pelo próprio animador, e certamente não pelo visualizador). A diferença básica consiste, portanto, na interatividade e autonomia dos HV.

O que faz um Humano Virtual ser "humano" não é a sua aparência exterior, mas seus movimentos, reações e tomadas de decisões naturais, apropriadas e sensíveis ao contexto.

Estes Humanos Virtuais são vistos de formas diferenciadas como agente e avatar (será discutido no capítulo 2). O agente é controlado por programas computacionais, e o avatar é controlado pelo usuário (BADLER, 1999). Este trabalho dará ênfase nos avatares, sendo alvo de maior detalhamento.

1.2 Graus de Liberdade (DOF - Degree Of Freedom)

A expressão "graus de liberdade" (DOF) foi usada originalmente para designar a capacidade de movimento de certos sistemas móveis, como robôs mecânicos. Entretanto, neste trabalho, o conceito de DOF é um pouco mais abrangente. Ele serve para designar e descrever cada um dos movimentos individuais presentes em uma articulação (ÇAPIN, 1999).

O número de graus de liberdade define o número de eixos de coordenadas que podem ser manipuladas simultaneamente durante o processo interativo. Cada junta pode ter vários DOFs, como mostra a Figura 1.1, mas cada um deles é um elemento do modelo.

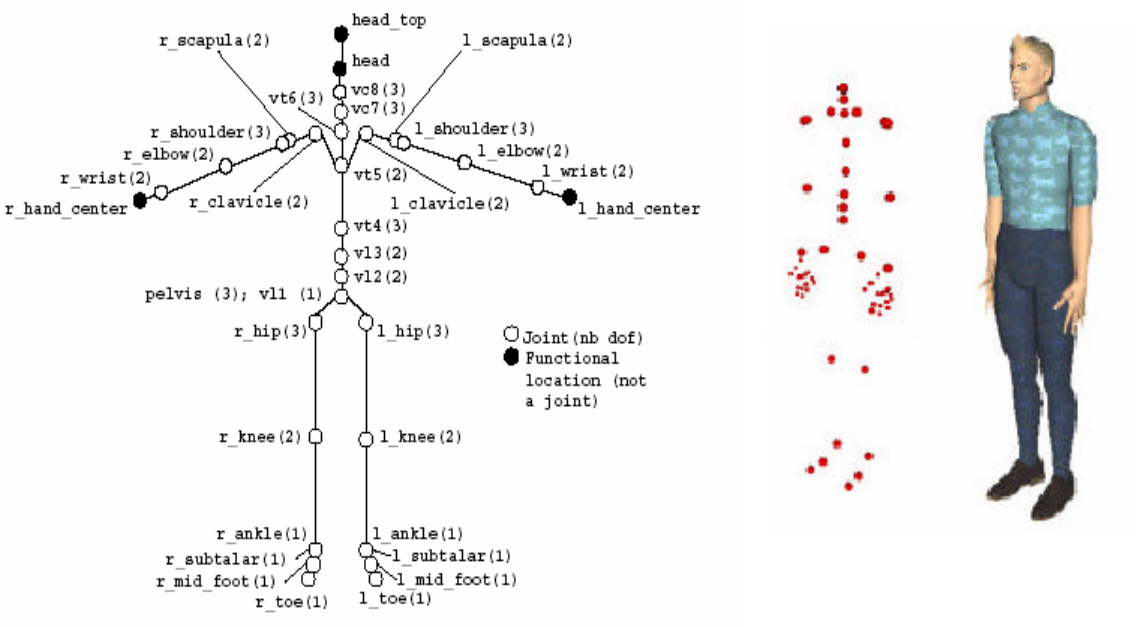

Figura 1.1 Representação do Humano Virtual com os DOF (ÇAPIN, 1999).

Basicamente, a cada DOF correspondem: um eixo de movimento (representado por um vetor), limites angulares máximos e mínimos, limites angulares de conforto, seu estado de repouso, e seu estado atual. Entretanto, para que possam representar diferentes orientações do eixo de movimento em torno de si, cada DOF é contemplado com mais dois eixos (ou vetores), caracterizando um sistema de referência próprio.

De acordo com Maciel (2001), dentre os modelos articulares existentes, se destacam duas abordagens para especificação da posição relativa entre dois segmentos articulados. Uma delas considera apenas juntas com três DOFs de rotação, um em cada eixo coordenado. A outra considera que cada junta tem apenas um DOF; quando se necessita representar uma junta de dois DOFs devem-se usar duas juntas de um DOF dispostas seqüencialmente. Na primeira abordagem, quando deseja fazer rotações em dois eixos, é preciso ter muito cuidado com a ordem em que elas serão feitas. Rotacionar em *x* e depois em *y* é diferente de rotacionar em *y* e depois em *x*. Na segunda abordagem, esse problema fica menor, pois ocorre um prévio conhecimento de qual movimento está subordinado ao outro. Por outro lado, a existência de vários elementos para representar uma única junta torna mais complexo o processo de efetuar

um movimento. A Tabela 1 mostra a distribuição total de graus de liberdade de um corpo segundo Arendi (1996).

Tabela 1 - Lista de graus de liberdade para cada grupo de juntas (ARENDI, 1996).

Nome	Local	Tipo	DOF
Fronto-parietal	Cabeça	Sutura serrátil	-
Interparietal	Cabeça	Sutura serrátil	-
Parieto-temporal	Cabeça	Sutura escamosa	-
Vômer-esfenoidal	Cabeça	Esquindilese	-
Esfeno-etmoidal	Cabeça	Sincondrose	-
Entre os ossos nasais	Cabeça	Sutura homogênea	-
Dento-alveolar	Cabeça	Gonfose	-
Atlanto-ocipital	Pescoço/Cabeça	Trocóide/Pivô	1
Têmporo-mandibular	Cabeça	Condilartrose	2
Escapulo-umeral	Ombro	Esferóide	3
Úmero-ulnar	Cotovelo	Trocleartrose	1
Rádio-ulnar proximal	Antebraço	Trocóide/Pivô	1
Rádio-ulnar distal	Antebraço	Trocóide/Pivô	1
Rádio-cárpica	Punho	Elipsóide	2
Carpo- metacarpiana (polegar)	Mão	Selar	2
Carpo- metacarpiana (outras)	Mão	Artródia	6 - Plana
Metacarpofalangeanas	Mão	Condilartróse (Elipsóide)	2
Interfalangeanas da mão	Mão	Trocleartrose	1
Entre os corpos vertebrais	Coluna	Antiartrose típica	3
Costo-vertebral	Tórax	Artródia	1
Costo-condral	Tórax	Sincondrose	-
Condroesternal	Tórax	Artródia	1
Coxofemoral	Quadril	Esferóide	3
Fêmur-tibial	Joelho	Condilartrose	2
Tíbio-fibular proximal	Canela	Artródia	3 - Plana
Tíbio-fibular distal	Canela	Sindesmose	-
Calcâneo-tibial	Tornozelo	Trocleartrose	1
Tarso-metatarseanas	Pé	Artródia	6 - Plana
Metatarsofalangeanas	Pé	Elipsóide	2
Interfalangeanas do pé	Pé	Trocleartrose	1

1.3 Representação de um Humano Virtual

Para representar graficamente Humanos Virtuais é necessário levar em consideração uma estrutura hierárquica, capaz de representar um conjunto de articulações, e um conjunto de funções utilizadas para gerar movimento sobre estas articulações.

Segundo a especificação do HAnim (2003), o corpo do HV consiste de um número de segmentos (como braço, perna, mão e pé) que estão conectados por juntas (cotovelo, pulso e tornozelo), como representa a Figura 1.2. Cada junta representa um conjunto de possibilidades de movimento, ou seja, um conjunto de DOF.

As juntas são de fundamental importância, pois, representam os relacionamentos matemáticos entre partes do corpo, sendo necessário utilizar outras estruturas para representar ossos, músculos e pele. Desta forma, as estruturas representadas pelos objetos gráficos mudam de posição e/ou orientação de acordo com a posição da junta.

Logo após a definição e implementação de um modelo de representação de corpo, é necessário aplicar movimentos sobre este corpo.

> A movimentação de um membro de um corpo articulado é restringida pelos outros membros em que está conectado através das juntas, as quais podem possuir de um a três DOF. Portanto, o número total de graus de liberdade de um corpo articulado pode ser calculado com a soma entre os 6 graus de liberdade que o corpo possui em relação ao universo e o somatório dos graus de liberdade de todas as juntas do corpo. No caso do corpo humano, pode-se ter até 200 graus de liberdade. No entanto, se os membros forem considerados flexíveis, devem-se somar ao total de graus de liberdade do objeto, os graus correspondentes a cada membro isoladamente. Na Figura 1.2, pode-se ver um exemplo de um corpo rígido articulado e os graus de liberdade de suas articulações (NEDEL, 2000).

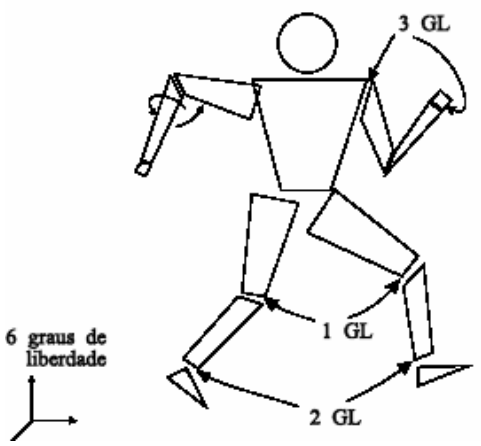

Figura 1.2-Graus de liberdade de movimento para um corpo articulado (NEDEL, 2000).

Nem sempre uma hierarquia detalhada é necessária para toda aplicação, mas o número mínimo de juntas tem que ser considerado para manter a forma de humanóide (pernas e braços estão sempre presentes e não podem ser removidos, sem que percam alguma capacidade de movimento no do humanóide virtual final).

CAPÍLULO 2 - AVATAR

Para tornar o envolvimento do ser humano mais real possível em um Ambiente Virtual, é necessário entre outros requisitos, que ele esteja representado neste mundo. Para isso, serão utilizados os avatares que irão incorporar ações e características do usuário no mundo virtual, textual ou tridimensional, aumentando o grau de envolvimento e imersão do usuário no Ambiente Virtual.

2.1 Conceito

Um avatar, segundo o Dicionário Aurélio (1988) é uma reencarnação de um deus, e especialmente no hinduísmo, a reencarnação do deus Vixnu. Um outro significado seria transformação, transfiguração, metamorfose. No contexto deste trabalho a palavra "avatar" é empregada com a conotação usual na comunidade de Realidade Virtual; que segundo Ventrella, nada mais é do que a representação gráfica do usuário no Ambiente Virtual, junto com seu comportamento. Para Badler (1999), avatar é um Humano Virtual controlado por um participante vivo.

Cada participante de um Ambiente Virtual assume um avatar, que reflete as ações do usuário dentro do ambiente. Podendo ser qualquer representação tridimensional, o avatar não precisa possuir a forma de um corpo humano, ele pode ser um animal, planta, máquina, alienígena, ou outro tipo de Figura (SINGHAL, 1999). Entretanto, aqui neste trabalho, o avatar terá a representação humana.

Cada avatar possui uma representação gráfica de um modelo estrutural de um corpo (presença de braços, pernas, juntas, etc.), modelo de movimento (uma variedade de movimento que as juntas devem ter), modelo físico (peso, altura, etc.).

2.2 Representação do usuário

O usuário possui uma representação humanizada dentro do AV através de um avatar, que pode se movimentar controlado pelo usuário. Além disso, o avatar possui movimentações que não dependem da manipulação do usuário, como: piscar os olhos, mexer a cabeça e braços suavemente, respiração. Isso tudo tem que ser feito cuidadosamente com qualidade e realismo, mas sem prejudicar o envolvimento do usuário (PERUZZA, ZUFFO, 2004).

Um avatar pode ser utilizado basicamente de duas formas dentro de um mundo virtual: como um representante gráfico do usuário; ou como um representante apto a realizar algumas tarefas adicionais (será apresentado na sessão 2.2.2).

2.2.1 Avatares como Representantes Gráficos

Um avatar deveria estar o mais próximo da realidade possível. Seu papel consiste em confrontar o real e o irreal (virtual); nos dois mundos (virtual e real) pessoas disfarçam, vestem máscaras, moldam sua forma e aparência, pretendem ser o que não são. Nos mundos virtuais podem acontecer as mesmas coisas que no mundo real, tendo a vantagem de que neles

tudo é permitido (TEICHRIEB, 1999).

Avatares adotam expressões e posturas normais entre os humanos, criando verdadeiras comunidades virtuais, comuns em locais de encontro como as salas de chat, simulação militar e jogos.

Um exemplo para representação é quando o usuário entra em um jogo de guerra. Na entrada do ambiente ele seleciona o avatar que melhor o representa (forças especiais de elite ou terroristas, por exemplo). Também se identifica com um nome e logo que seu avatar é posicionado no ambiente, visualiza seu nome no avatar, ou próximo do mesmo. Feito isso, pode iniciar uma "guerra" com outros usuários que estão no mesmo ambiente que ele, o que se torna bastante intuitivo, dado o fato de que pode ver no mundo virtual o avatar que o representa e os demais avatares que estão freqüentando o lugar naquele instante. Além disso, na medida em que, novos usuários entram, a presença poderá ser percebida pela chegada de novos avatares.

Estes avatares estão apenas representando graficamente os usuários, pois não possuem a habilidade de realizar qualquer tarefa, como por exemplo, iniciar uma conversa com a "pessoa" em pé ao seu lado, ou decidir qual arma deverá ser utilizada naquela missão. Neste caso pode-se dizer que os avatares são totalmente sem inteligência, ou seja, apesar de representarem os usuários, não realizam nenhuma atividade que estes são capazes de fazer no mundo real.

O objetivo de usá-los é aumentar o grau de realismo do mundo virtual para os visitantes, permitindo que estes se sintam mais presentes (imersos) no ambiente e, consequentemente, menos desorientados.

2.2.2 Avatares com Movimentos Inteligentes

De acordo com Teichrieb (1999), uma das grandes aplicações de avatares é a sua utilização com movimento inteligente. Neste caso, os avatares deixam de ser apenas meros representantes gráficos de usuários no mundo, para se tornarem personagens com comportamento predefinido, ou seja, agentes inteligentes que executam alguma ação depois de receberem uma seqüência de percepções.

Cada tipo possui características e funcionalidades diversas, que são aplicados em AV de acordo com o problema a ser solucionado. Os agentes diferem basicamente com relação ao conhecimento que possuem sobre si mesmo (o agente sabe que está com sede), sobre o ambiente (o agente sabe que é de noite) e sobre outros agentes (a posição dos outros agentes) (TEICHRIEB, 1999).

Estes variados tipos de agentes exigem a utilização de técnicas de Inteligência Artificial para a sua modelagem. A modelagem de agentes é tratada como seres que têm personalidade e que são capazes de sentir emoções, como alegria, tristeza, medo, raiva etc., passando a ilusão de vida.

Investigações utilizando esta modelagem têm sido realizadas, por exemplo, para criação de ambientes de jogos de computador, educacionais, entre outros, modificando as capacidades dos personagens que fazem parte do ambiente, de forma que transmitam ilusão de vida ao usuário, seguindo instruções e improvisando comportamentos.

Nos jogos de guerra, os avatares possuem capacidade de se comportar e expressar emoções segundo definições prévias, pode-se colaborar com eles improvisando um comportamento físico (o usuário quer ir a um determinado lugar e o avatar se posicionará nesse lugar) e verbal (o usuário quer se comunicar com alguém e o avatar inicia um diálogo),

realizando e enriquecendo as intenções dos usuários. Estes também podem direcionar o humor dos seus avatares, o que influencia suas improvisações (o avatar expressa dor, raiva, alívio).

2.3 Exemplos de uso de avatares

Os avatares de domínio público estão geralmente em jogos de videogames ou de computadores. Suas representações variam para cada jogo, podendo constituir-se de formas simples quase geométricas, ou modelagem tridimensional quase humana, inclusive nos movimentos e falas. Tudo depende da complexidade do jogo e ambiente onde acontece a ação. Por exemplo, existem jogos de corrida diversificados, desde os mais simples onde aparece um quadrado e alguns obstáculos, até aqueles onde a interface é um carro com volante, que se mexe com os movimentos do usuário (GOLIN e BUÔGO, 2004).

Um exemplo de avatar bem simplificado é do jogo "Pac Man" (Figura 2.1). O avatar é o Pac Man (come-come), enquanto o usuário o movimenta pela tela a boca abre e fecha (NAMCO,2006). O objetivo do jogo é fazer com que o avatar passe por todos os lugares da tela, "comendo" todas as "pecinhas", e fugindo dos fantasmas, que funcionam como atores virtuais, inseridos neste ambiente de jogo.

Figura 2.1 - Jogo do Pac Man (Atari)

Atualmente os jogos possuem tecnologias mais avançadas, aumentando a complexidade, sendo possível definirem quase tudo em seu avatar, desde a aparência, como sexo, cor dos olhos, do cabelo, tamanho, roupa; até a psicologia de seu personagem, triste ou alegre, extrovertido ou tímido, entre outros. O jogo “The Sims” (Figura 2.2) exemplifica isso, seguido do jogo “Tatical Ops” (Figura 2.3) que é um cenário de guerra onde o usuário escolhe se faz parte das forças especiais ou terroristas.

Figura 2.2 – The Sims

Figura 2.3 - Jogo Tactical Ops

Os projetos de avatares em jogos computacionais e demais sistemas de realidade virtual estão cada vez mais desenvolvidos. Na Realidade Virtual não imersiva, aquela em que os únicos dispositivos são: o monitor, o mouse e o teclado de um computador; o uso de avatares assegura maior imersão do usuário no ambiente, pois o usuário pode ver a sua representação no mesmo.

Outra forma que faz com que o uso de avatares seja satisfatório, são as visões em primeira e terceira pessoa. A visão em primeira pessoa chamada de subjetiva (Figura 2.4) é aquela em que a pessoa vê o ambiente como se estivesse vendo com seus próprios olhos. A visão em terceira pessoa (Figura 2.5) mostra o avatar como um elemento inserido no Ambiente Virtual, o usuário vê a representação do seu avatar. Tanto na primeira como na terceira visão o usuário tem possibilidade de visualizar o avatar dos outros participantes (GOLIN, BUÔGO, 2004).

Figura 2.4 - Jogo Tactical Ops visão em primeira pessoa

Figura 2.5 - Jogo Deadly Dozen visão em terceira pessoa

Assim como qualquer objeto a ser modelado em um mundo virtual, a modelagem de personagens deve ser realística. Os esforços concentram-se tanto nos aspectos visuais (aparência, movimento) como em comportamentos mais próximos ao de um ser humano real (interatividade, comunicação e gestos).

2.4 Aparência

Os avatares podem ser visualmente representados como ícones 2D, desenhos animados, vídeo compostos, formas 3D ou corpos completos (BADLER, 1999).

A aparência do avatar é criada para fazer a identificação do usuário no Ambiente Virtual, dando uma sensação de presença e levando à percepção não intencional dos demais usuários. Quando se fala da "aparência de um avatar", não se está obrigatoriamente exigindo que este exiba uma imagem idêntica à do usuário que este representa. Pode utilizar-se "dicas" que identifiquem os usuários reais como, por exemplo, crachás ou camisetas com textos como "Atendente", "Soldado Bonfeti".

É possível também representar o avatar levando em consideração seu genótipo, que consiste em um vetor de valores que determinam vários atributos. Uma variedade genética sustenta um espaço dentro de algumas possibilidades que pode existir. A Figura 2.6 mostra cinco tipos étnicos.

Figura 2.6 Faces de Avatar mostrando a variedade genética (VENTRELLA)

A forma humana pode ser modelada por malhas de polígonos, com segmentos rígidos representando os membros, movimento controlado ao nível das articulações e limites angulares das articulações suficientemente precisos para aplicações em ergonomia (BADLER, 1997). Para uso de avatares em tempo real, uma geometria simples pode suprir as necessidades da aplicação. Por exemplo, o modelo de um soldado modelado com 110 polígonos texturizados é aceitável se usado suficientemente longe da câmera e permitir que o

usuário o reconheça como um soldado. Por outro lado, o ocupante de um veículo cujo interior está sendo analisado, deve ser exibido com um maior nível de precisão, tanto em nível de geometria como movimento.

2.5 Movimento

Para a criação de avatares é necessário, considerar movimentos que representam os produzidos pelos seres humanos.

O movimento manifestado em um avatar pode ser obtido de diversas fontes: captura de movimento a partir de vídeo em tempo real; captura de movimento por sensores; dados de movimentos previamente armazenados, como transformações 3D globais ou locais (em nível de articulações); síntese de movimento, como interpolação dos ângulos das articulações, cinemática inversa e dinâmica. (BADLER, 1997).

O uso de movimentos sintetizados e pré-armazenados são positivos na implementação de Humanos Virtuais, pois oferecem velocidade de execução ao sistema. No entanto, sua maior vantagem se encontra no reduzido conjunto de parâmetros utilizado e na possibilidade de generalização do movimento (caminhar, procurar, encontrar, etc.) que o modelo oferece.

De acordo com Nedel (2000), a principal desvantagem do uso de modelos previamente armazenados, é que, cada articulação é explicitamente controlada, e quando a distância entre as articulações é alterada, as posições finais das extremidades (pés e mãos) são afetadas, impossibilitando a manutenção do controle de restrições externas e colisões. A desvantagem dos sistemas de síntese reside basicamente na dificuldade em gerar movimentos

que pareçam naturais. A cinemática inversa não é propriamente um método adequado para modelar movimentos humanos.

De fato, a questão da geração de movimentos adequados na síntese de Humanos Virtuais é ainda um tópico de pesquisa complexo e aberto. Considerando que movimentos humanos precisos são difíceis de sintetizar, este método é aplicado na movimentação de personagens em jogos 2D, e recentemente obteve um ganho de popularidade na animação de personagens 3D, avatares e em simulações militares incluindo a síntese de soldados individualizados.

2.6 Interatividade, Comunicação e Gestos

A interatividade ocorre quando o usuário controla o avatar, gerando movimentos. Esses movimentos podem ser simplesmente rastreados por sensores. O avatar poderá ser controlado através de menus, botões de ação, combinação de teclas; com o auxilio de mouse, teclado, luvas e outros dispositivos.

Devido principalmente à sensação de presença que um usuário possui dos demais usuários em um AV, os diversos participantes são estimulados a comunicarem entre si. Essa comunicação pode ocorrer das mais diversas formas, tais como: o *chat* (bate-papo textual); os canais de voz e a comunicação gestual, onde podem ser rastreados através de dispositivos específicos e replicados em seus respectivos avatares em tempo real. Tais dispositivos, porém, não são adequados para a captura de expressões faciais, uma vez que os pontos a serem mapeados encontram-se muito próximos. Experiências nesse sentido têm sido realizadas através da construção do VISTEL (do ATR Research Lab., no Japão), que é um sistema de

videoconferência que utiliza realidade virtual e permite o compartilhamento de objetos virtuais entre dois usuários para a realização de tarefas colaborativas (LEITE, 2000).

Para Badler (1999), os avatares interagem socialmente, estendendo a comunicação vocal aos gestos. A função gestual envolve a comunicação não-verbal. Gestos podem ser entendidos como metáforas vocais, como por exemplo, o ato de apontar para um objeto, um local ou outro ator no espaço virtual pode significar que o participante deseja estabelecer algum tipo de contato.

CAPÍTULO 3 - DETECÇÃO DE COLISÃO

A Detecção de Colisão informa quando objetos do mundo virtual estão passando um dentro do outro, diminuindo o realismo do ambiente. Em algumas aplicações de RV, existem propriedades dos avatares e dos objetos na cena que faz com que não ocorra essa passagem, realçando a sensação de imersão dos usuários. Em outras aplicações de RV onde simulações físicas são executadas, a Detecção de Colisão é imprescindível, e por vezes bastante complexa e robusta para lidar com um grande número de polígonos nas mais diversas posições; devendo ser rápida e precisa em alguns casos.

3.1 Características da Detecção de Colisão

Detectar colisão é determinar quando há uma aproximação entre objetos suficientemente grande para ocorrer sobreposição. Para considerar a possibilidade de colisão entre objetos, é necessário que o ambiente tenha pelo menos dois objetos, e que pelo menos um esteja em movimento.

> De acordo com Nascimento (2003), Detecção de Colisão é um problema muito complexo, visto que os objetos possuem formas variadas e movimentos bem definidos. A colisão entre objetos prescinde de prévia detecção, isto porque, no momento que ela ocorre, o movimento deve cessar imediatamente para que em primeira análise, possa evitar que os corpos se interpenetrem, e em segunda análise, simular o comportamento de um objeto real.

Para diminuir o número de cálculos para os testes de colisão, são utilizados filtros. Estes filtros são aplicados de maneira hierárquica, isto é, dos mais simples e aproximados, aos

mais complexos e exatos. Assim, através de técnicas simples pode eliminar os objetos que não tem a possibilidade de colidirem, e usar técnicas de colisão mais detalhada para objeto que apresentam “perigo” de colisão (NASCIMENTO, 2003).

A técnica simples testa a colisão entre objetos em um determinado instante de tempo, e a técnica mais sofisticada testa a colisão entre objetos em um espaço de tempo.

3.2 Técnicas para Detectar Colisão

Detecção de Colisão é um problema fundamental para muitas aplicações gráficas interativas 3D. Ela necessita ser em tempo real, e para isso é necessária uma análise de técnicas para cada aplicação. O ideal é a criação de técnicas de Detecção de Colisão híbridas, isto é, não é utilizado apenas uma técnica de Detecção de Colisão, mas sim várias técnicas, divididas em passos. Esses passos visam diminuir a quantidade de cálculos, aplicando técnicas mais simples quando os objetos estão distantes e técnicas mais complexas quando os objetos estão mais próximos (PERUZZA, 1997).

Além das técnicas, existem os algoritmos para determinar se ocorre possibilidade de colisão. Estes algoritmos servem para prevenir colisão e não detectar.

A pesquisa referente às técnicas de Detecção de Colisão foi baseada nas técnicas por: esferas, retângulo, pixel, voxel e partículas.

3.2.1 Esferas

Este método segundo Nascimento (2003), consiste em uma verificação simples. É constituído de três fases: Fase de Detecção de Colisão entre envoltórios esféricos, fase de identificação dos prováveis pontos de colisão e fase de validação da colisão. A Figura 3.1 apresenta um diagrama destas etapas.

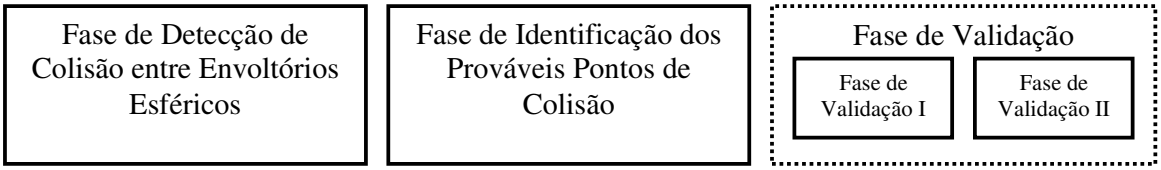

Figura 3.1 - Fases de Detecção de Colisão

Na fase de Detecção de Colisão entre Envoltórios Esféricos, a colisão do par de esferas ocorrerá quando a distância entre os centros das duas esferas for menor que a soma dos respectivos raios dessas esferas. A Figura 3.2 ilustra a situação em 2D, onde os envoltórios esféricos são representados por círculos. A Figura 3.3 mostra um avatar em 3D envolvido por esferas.

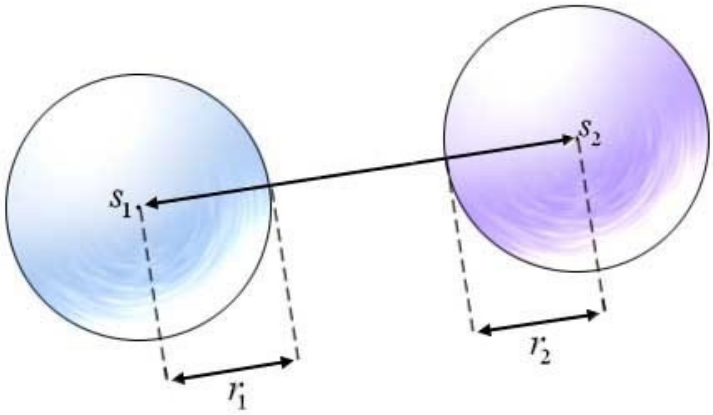

Figura 3.2 – Demonstração do cálculo dos raios dos envoltórios.

Figura 3.3 – Avatar com um envoltório esférico

Considerando que o s1 e s2 são os centros das esferas de raios r1 e r2, o teste de verificação de colisão pode ser dado analisando a seguinte condição: $|s1 - s2| < r1+r2$.

Os prováveis pontos de colisão são identificados somente após a Detecção de Colisão entre os envoltórios.

Na fase de Identificação dos prováveis pontos de colisão, a técnica Octree esféricas (seção 3.3) é aplicada em apenas um dos objetos para tornar mais veloz o processo. Com isso são isoladas as esferas que não interagem com as esferas que envolvem o outro objeto, reduzindo muito os cálculos. Assim só as partes de interesse serão alvo de novas análises.

Provavelmente o ponto de colisão será o ponto central da maior esfera completamente contida dentro do envoltório do outro objeto.

A estratégia é validar cada um dos prováveis pontos de colisão imediatamente após a sua identificação, caso este ponto não seja validado o processo de subdivisões recomeçará a partir da última região analisada.

A Figura 3.4 mostra a identificação de cinco prováveis pontos de colisão representados pelas circunferências preenchidas, obtidos através da aplicação da estratégia Octree do objeto A. Estes cinco pontos correspondem ao ponto central das circunferências totalmente contidas no envoltório do objeto B.

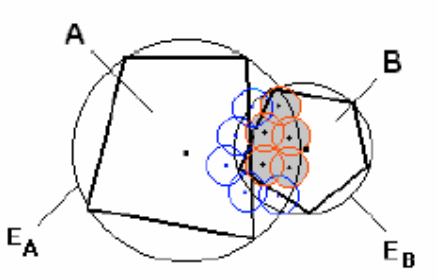

Figura 3.4 – Visualização dos prováveis Pontos de colisão (Nascimento, 2003).

A fase da validação da colisão tem o objetivo de verificar se o provável ponto de colisão obtido na fase anterior pertence ao interior dos dois objetos. Esta estratégia não se aplica aos objetos côncavos. É apresentada em duas fases: validação I e validação II.

Na Validação I são realizadas comparações entre o provável ponto de colisão e o ponto central do envoltório esférico, pertencente ao interior do objeto que sofreu o processo de Octree, em relação à superfície de cada face do objeto.

Aplica a validação no provável ponto de colisão e o ponto do centro da esfera que circunscreve o objeto. Caso alguma dupla de valores seja de sinais opostos e diferentes de zero, pode afirmar que o ponto pertence ao envoltório esférico, mas não está contido no objeto regular, caso contrário pertence alguma face ou está contido no objeto regular. Se o ponto não pertencer ao objeto e o sistema tiver tempo para um novo refinamento então o processo de Octree é reiniciado, caso contrário inicia a validação II.

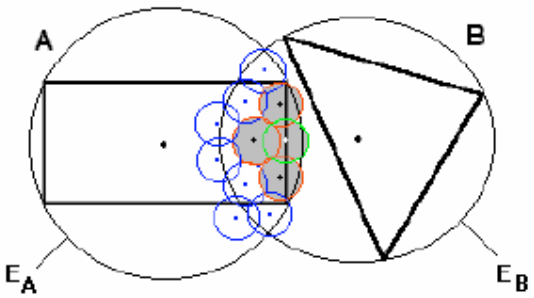

Figura 3.5 - Prováveis pontos de colisão

Através da Figura 3.5 podem ser visualizados os possíveis casos da fase de Validação I, onde os casos de interesse são apresentados pelas circunferências preenchidas.

Na Validação II é feita a comparação com o outro objeto realizando as comparações feitas na fase de Validação I. Na Figura 3.6 a parte verde corresponde à região de intersecção dos dois objetos, e as circunferências vermelhas representam os pontos de colisão.

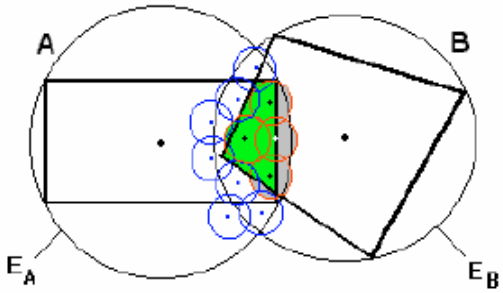

Figura 3.6 - Pontos de colisão (Nascimento, 2003)

3.2.2 Retângulo

Este método consiste em envolver completamente cada objeto com um retângulo. Para saber se dois objetos se colidem, basta testar se os dois retângulos têm algum ponto em comum.

De acordo com Sabino (2004), as vantagens de utilizar esse método é a facilidade de implantação, a velocidade de processamento e precisão quando o movimento do objeto é rápido. A desvantagem é que a precisão depende da forma do objeto. A Figura 3.7 mostra quando ocorre colisão entre dois objetos.

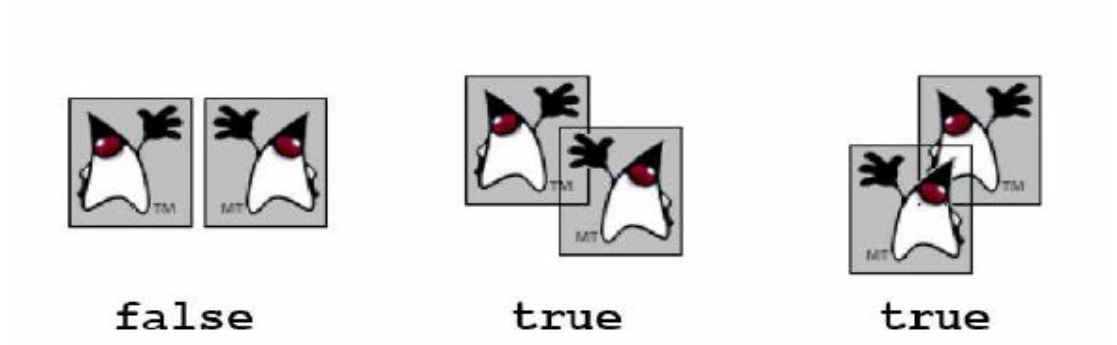

Figura 3.7 Detecção de Colisão usando retângulos (SABINO, 2004).

As vantagens e desvantagens irão depender da aplicação, como por exemplo, em um jogo para celular, geralmente não se precisa de alta precisão de colisão, porque a movimentação rápida do jogo não permite que os usuários possam perceber a imprecisão da colisão.

Na verdade o importante é a percepção do usuário, não a precisão total. É preciso observar se o usuário percebe ou não uma colisão que deveria ter ocorrido.

Para que o teste por retângulo seja melhorado, são colocadas mais figuras geométricas no objeto, como por exemplo, fazer com que o objeto seja composto por três ou quatro retângulos.

Para calcular colisões de forma rápida, os testes de colisão normalmente são realizados entre dois objetos (dois-a-dois). Caso exista n objetos serão realizados n teste de colisão (n) * (n - 1) /2). Em outras palavras, o número de testes é "quase exponencial", ou seja, quanto mais objetos testados, muito mais testes serão necessários, o que pode tornar inviável testes de colisão entre muitos objetos.

A Engine GameBase (KLIEMANN, 2002) reduziu o número de testes quando o número de objetos no jogo é grande. Uma das alternativas utilizadas é "setorizar" o cenário de jogo (a área onde os objetos estão), ou seja, dividir todo o cenário em retângulos, e incluir

os objetos nestes retângulos. Só irá testar a colisão entre os objetos de mesmos retângulos ou de retângulos próximos.

3.2.3 Pixel

Este método verifica se algum ponto da imagem de um objeto está colidindo com a imagem de outro objeto, ou seja, determina quando dois pixels estão sobrepostos (Figura 3.8). Esta verificação é realizada ponto a ponto entre as duas imagens. Se algum ponto de um objeto colidir com algum ponto de outro, a colisão é detectada (SABINO, 2004).

Este método tem o processamento bem lento (por exemplo, se a imagem de cada um dos objetos for pequena, tendo 32 x 32 pontos, serão necessárias até 32*32=1024 comparações) e pode perder a precisão de acordo com o movimento do objeto. A vantagem é que pode detectar colisão entre formas arbitrárias.

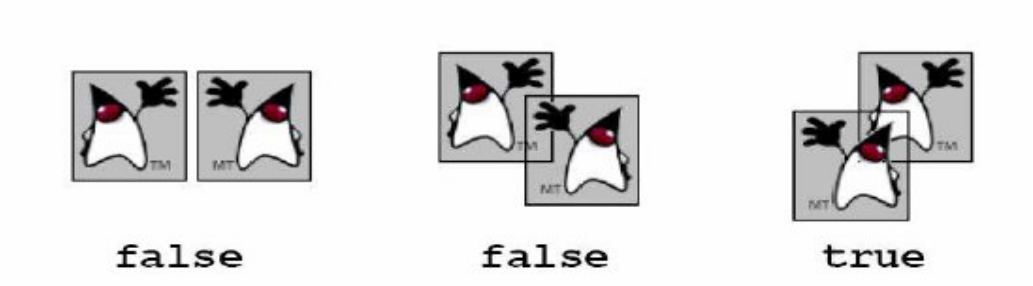

Figura 3.8 - Detecção de Colisão por pixel (SABINO, 2004).

3.2.4 Voxel

Este método consiste em particionar os paralelepípedos, que envolvem completamente os objetos, em cubos de tamanho fixo. É feita uma varredura para determinar os voxels que não pertencem ao objeto. A detecção é feita através da comparação entre os voxels de objetos distinto. Se o voxel do objeto A intercepta o voxel do B, então é detectada a colisão. O voxel difere do pixel pelo fato de ser em 3D.

3.2.5 Partículas

O tempo de processamento para o tratamento de colisões é proporcional ao número de objetos multiplicado pelo número de partículas. Considerando-se que o número de partículas pode estar na faixa de centenas de milhares de partículas, com apenas uma dezena de objetos na cena, o número de cálculos de intersecção fica na faixa de milhões de intersecções entre um sistema de partículas e os objetos, por quadro da animação. Isto é relativamente custoso, especialmente se a animação apresentar um número expressivo de quadros (STEIGLEDER, 1997).

Para acelerar este processo é preciso reduzir o número de partículas que devem ser consideradas no cálculo de intersecção entre o sistema de partículas e os objetos sólidos. Podendo utilizar uma grade pentadimensional, para efetuar uma seleção prévia das partículas que possuem alguma possibilidade de interceptar algum objeto da cena.

O método de partículas é mais utilizado para a modelagem de fenômenos naturais como fogo, fumaça, neblina, entre outros.

Para Steigleder (1997), os passos para definir o método para a detecção de colisões entre um sistema de partícula e os demais objetos da cena são: primeiro, obter o cone de direções das partículas de uma determinada célula a partir do centro da célula, da direção base das partículas e da variação das direções das partículas; segundo, obter os diversos raios de amostragem; terceiro, computar a intersecção entre os raios amostrados e cada objeto da cena e obter a menor intersecção entre os raios e o objeto; quarto, verificar a possibilidade real de colisão entre alguma partícula e os objetos da cena, comparando-se a distância mínima entre as partículas e o objeto com o módulo do vetor velocidade de cada partícula; e por último, caso exista a possibilidade real de colisão, a intersecção da partícula com o objeto deve ser calculada analiticamente.

Para realizar este método deve-se considerar que a direção do movimento de uma partícula possui dois graus de liberdade, e a posição desta partícula possui três graus de liberdade. Deste modo, a trajetória futura de uma partícula em movimento retilíneo, possui somente cinco graus de liberdade.

Um cone, por exemplo, é definido com base nos cinco parâmetros, onde os três parâmetros que definem a posição tridimensional do centro da célula são utilizados para obtenção do vértice do cone, e os dois parâmetros que definem a direção do movimento da partícula são utilizados para se obter o eixo de direção do cone.

Steigleder (1997) trabalha com intervalos, os intervalos dos dois parâmetros que definem a direção são utilizados para definir os ângulos de abertura do cone, obtendo-se desta forma uma primitiva geométrica completa (Figura 3.9).

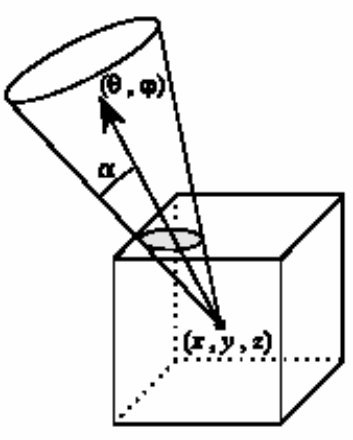

Figura 3.9. Obtenção do cone que define conjunto de movimentos das partículas de uma célula da grade pentadimensional (Steigleder, 1997).

O problema com o uso da posição central da célula como vértice do cone, é que algumas partículas podem estar fora do cone, portanto, podem ser obtidos resultados errôneos. Este problema é solucionado de acordo com Steigleder (1997), movimentando-se o cone na direção oposta à do movimento base das partículas (eixo do cone) de modo que o cone encapsule a célula que define o intervalo das posições tridimensionais (Figura 3.10).

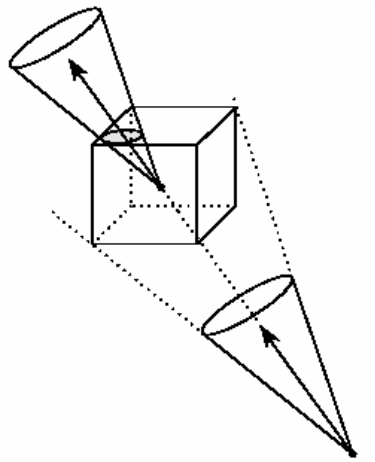

Figura 3.10. Obtenção do cone que encapsula o ortoedro que define as posições. (Steigleder, 1997).

Uma vez obtido o cone que define o conjunto de movimentos das partículas de uma célula, tem que determinar se este cone possui pontos de intersecção com os demais objetos da cena. O procedimento de intersecção entre um cone e um objeto genérico, e a obtenção da

distância mínima do objeto ao centro da célula é custoso e difícil, implicando um cálculo específico para cada primitiva.

A solução é a utilização de procedimentos aproximativos. O método efetua uma amostragem de raios emitidos a partir do vértice da célula e com direções dentro do cone que define o conjunto de direções das partículas. Gerados estes raios, calculam-se as intersecções deles com os objetos da cena. Se algum raio intercepta o objeto, significa que existe pelo menos uma área de colisão entre o objeto e o cone de direções das partículas. Steigleder (1997) cita as vantagens e desvantagens que ocorre neste processo.

As vantagens serão: possibilidade de utilização do mesmo algoritmo para os diversos objetos sólidos; esta abordagem apresenta um custo computacional menor que uma abordagem analítica, exceto se for utilizado um grande número de raios para a amostragem.

A principal desvantagem é que este processo obtém somente uma aproximação do valor ou situação real. O fato de a distância mínima ser somente uma aproximação pode ser parcialmente compensado através do uso de um fator de erro. Como esta distância mínima irá ser utilizada para uma determinação de proximidade, este fator de erro irá diminuir a distância mínima para um valor mais confiável.

Assim, se a distância que a partícula percorrerá é maior que a distância mínima da partícula ao objeto (observando que o cone de direções das partículas intercepta o objeto) existe uma grande possibilidade que a partícula colida com esse objeto. Neste caso, é necessário então calcular a intersecção entre o raio definido pela posição e direção da partícula e o objeto, efetuando a reflexão apropriada se necessário. Outra desvantagem é a quantidade de memória exigida para a alocação da grade pentadimensional.

3.3 Estratégia de Octree

Octree é um estrutura de dados hierárquica simples que consiste na divisão inicialmente do paralelepípedo envoltório (*bounding box*) do sólido a ser modelado em oito octantes , até uma dada precisão (Figura 3.11).

> Os octantes são rotulados como preto branco e cinza. O octante será preto, se estiver totalmente preenchido por uma parte do sólido, ou seja, completamente contido no sólido. O octante será branco, se estiver completamente exterior ao sólido, isto é, não pertence ao objeto. O octante será cinza, se estiver parcialmente preenchido pelo sólido. (Nascimento, 2003).

Enquanto houver nós cinzas serão novamente divididos em mais oito suboctantes, até que todos os nós folha da octante sejam rotulados como preto ou branco. As Figuras 3.11, 3.12 e 3.13 demonstram exemplos da representação do Octree.

As vantagens dessa técnica são determinadas pela facilidade com a qual pode representar objetos sólidos em diversas formas, sem importar se são côncavos ou convexos; e por aumentar o desempenho, pois elimina muitas regiões do objeto que não estão na iminência de colidir.

A desvantagem dessa técnica é que, por mais precisa que seja a representação, ela será sempre uma aproximação do sólido original, nunca uma representação fiel.

Partindo da Octree é encontrada a variação do método chamado Octree esferas, sendo fácil e eficiente na identificação da relação de distância entre os envoltórios esféricos.

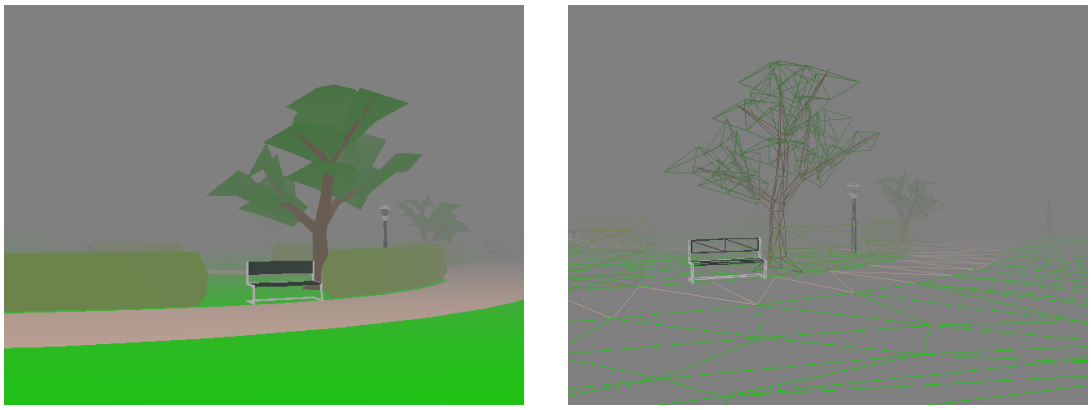

Figura 3.11. Exemplo de um programa para a visualização da estratégia Octree.

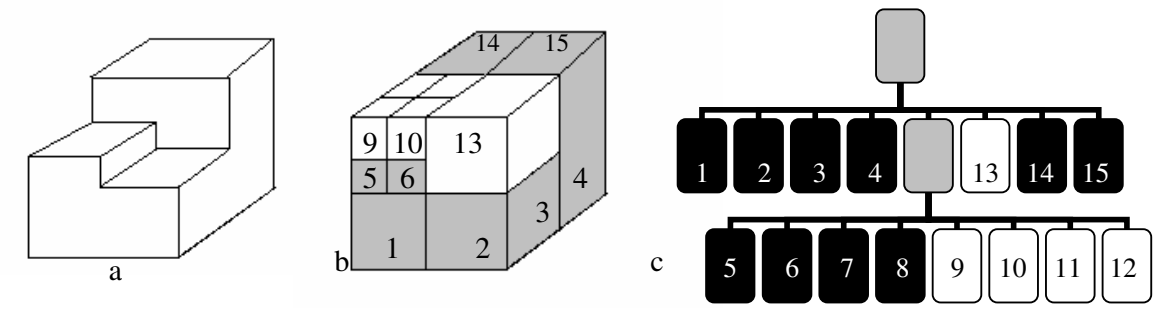

Figura 3.12 - Representação do sólido por Octree.a) sólido b) divisão recursiva do sólido c) Octree representada.

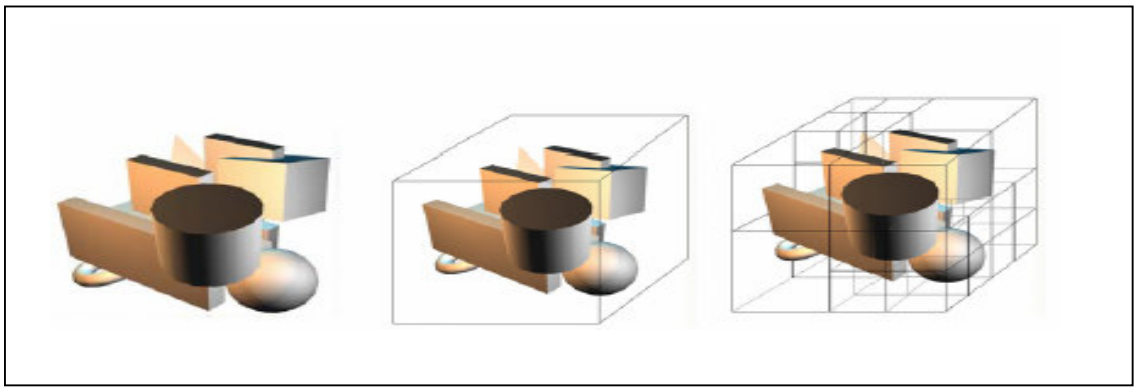

Figura 3.13 - Representação do sólido por Octree com formas diversificadas

CAPÍTULO 4 – WORLDTOOLKIT (WTK)

O objetivo desse capítulo é apresentar algumas considerações sobre o WorldToolKit Release 8, utilizado na implementação das técnicas de Detecção de Colisão. Para isso serão expostas primeiramente origem e conceitos. Em seguida explicações de algumas bibliotecas e funções para o entendimento do trabalho efetuado.

4.1 Considerações para a utilização do WTK

De acordo com Netto, et al. (2002), a Sense8 foi fundada por uma dupla de hackers americanos que pretendia fornecer aos seus companheiros programadores ferramentas de RV de tempo real, independentes de plataforma.

Surgiu então o WorldToolKit para fins científicos e comerciais, um sistema multi-plataforma portátil, para o desenvolvimento de aplicações 3D que oferece vídeo em tempo real de alto desempenho e suporte para som interativo.

Basicamente, o WTK é uma biblioteca em C com mais de mil funções, que permitem interagir e controlar simulações em tempo real. Uma chamada de função pode fazer o trabalho das centenas das linhas do código C, reduzindo drasticamente o tempo de desenvolvimento (SENSE 8 CORPORATION, 1999).

O WTK supre a necessidade de desenvolvimento de filtros de conversão, funcionalidade 3D e outras utilidades, através de APIs (Application Program Interface) de fácil entendimento e utilização. Ele também é estruturado de forma orientada aos objetos,

embora não use herança ou ligações dinâmica. As funções WTK são agrupadas em mais de vinte classes. Entre elas incluem o universo, geometria, nós, ponto de vista, janelas, luzes, sensores, caminhos, e outras.

As funções são incluídas para a configuração de display, detecção de colisão, carregamento e criação dinâmica de geometria, controle de renderização, comportamento de objetos e interfaces gráficas 2D.

Um Ambiente Virtual no WTK pode ser controlado por uma variedade de sensores de entrada, desde um simples mouse até um sensor com 6 graus de liberdade. O usuário pode sentir imerso e interagir com este ambiente usando capacetes de visualização e dispositivos de entrada como mouse ou luva.

No núcleo de sua aplicação encontra-se o loop da simulação, que lê os sensores de entrada, atualiza os objetos, e renderiza um novo frame da simulação. O WTK está projetado para ser usado em aplicações de tempo real, onde a taxa de frames por segundo deve ser mantida numa ordem de zero.

O loop de simulação é iniciado com a chamada da função *WTuniverse_go* e encerrado com a função *WTuniverse_stop.* Alternativamente, pode ser usada a função *WTuniverse_go1* para executar exatamente uma vez e sair automaticamente da simulação. A Figura 4.1 mostra a ordem padrão dos eventos no loop de simulação. A ordem pode ser mudada usando a função WTuniverse_seteventorder (SENSE 8 CORPORATION, 1999).

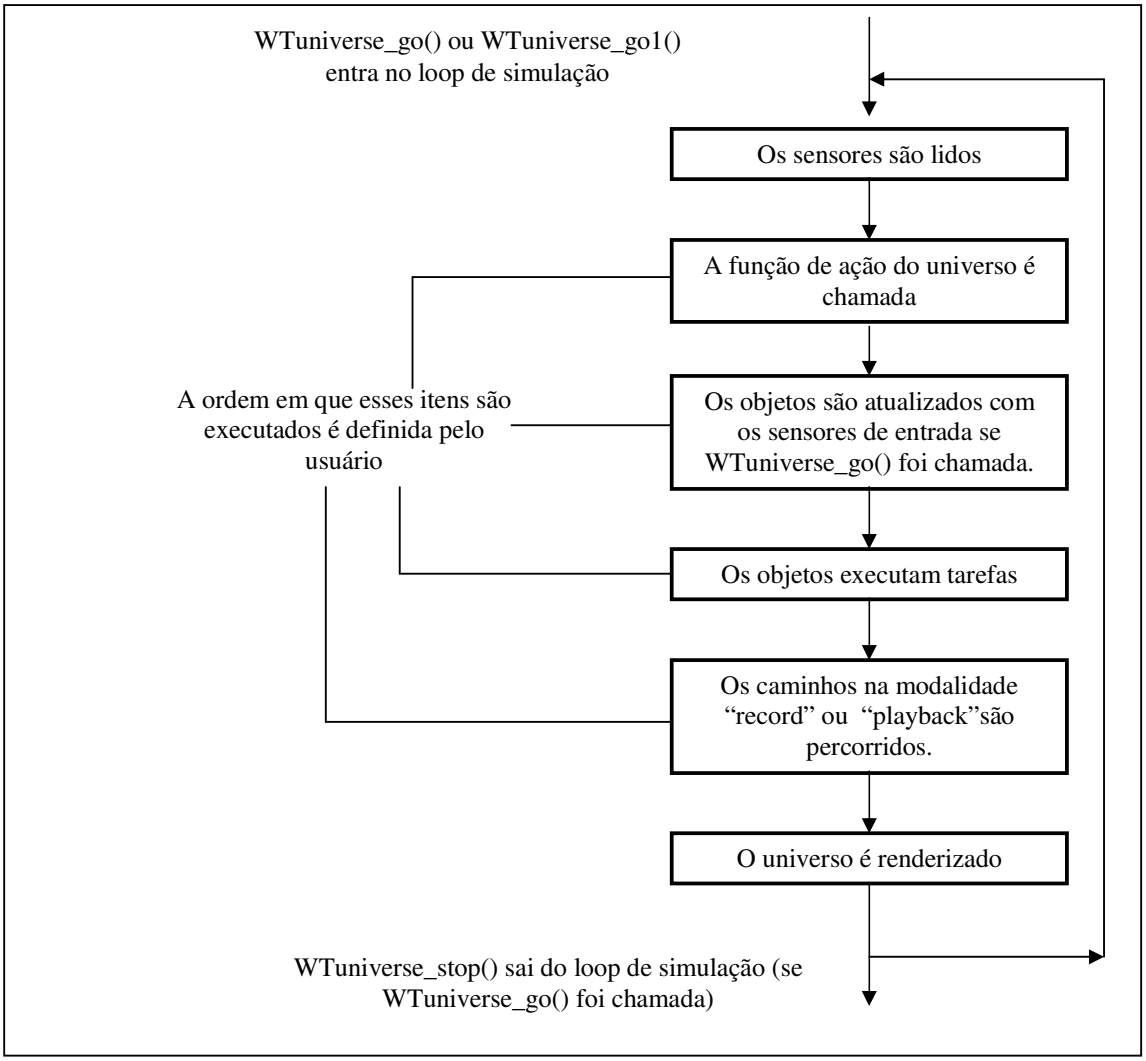

Figura 4.1 – Loop de simulação padrão

4.2 Visão Geral das classes em WTK

De acordo com a Sense 8 Corporation (1999), as funções do WTK são orientadas aos objetos em suas convenções de nomeação, e são agrupadas nas seguintes classes:

Universo: container de todos os objetos WTK, como: geometrias, nós, ponto de visão, sensores, etc. Pode haver vários gráficos de cena em uma simulação, mas somente um universo para cada simulação.

Geometria: objetos gráficos que são visíveis em uma simulação, como: um bloco, uma esfera, um cilindro, e um texto 3D. Pode criar geometrias dinamicamente ou importá-las de outras fontes. Uma vez criada uma geometria, é necessário criar um nó correspondente (da geometria), de modo que possa ser incluído em um gráfico da cena.

Nós: é o bloco construtor do grafo de cena. Como exemplo: geometria, luzes, entre outros.

Polígonos: podem ser criados dinamicamente e mapeados com texturas usando várias fontes de imagem.

Vértices: podem ser criados dinamicamente ou lidos de um arquivo.

Luzes: podem ser criadas dinamicamente ou carregadas de um arquivo.

Pontos de visão: definem a posição e a orientação em um Ambiente Virtual de como as geometrias são projetadas e renderizadas em uma simulação. O WTK suporta um ou mais ponto de vista, podendo controlar a posição e a orientação através de sensores.

Janelas: mostram a cena. Uma aplicação em WTK pode possuir várias janelas no mesmo Ambiente Virtual e/ou várias janelas em diferentes ambientes virtuais.

Sensores: podem ser conectado a nós de transformação, ponto de vista, nós móveis, entre outros, para manipular o movimento dos objetos. São suportados vários sensores.

Caminho: utilizado para mover o ponto de visão ou geometria por um caminho pré definido.

Tarefas: associa comportamentos (como por exemplo, movimento).

Ligações móveis: associa um sensor ou caminho com um alvo.

Som: objetos podem ser carregados, associados com objeto 3D e tocados.

Interface com o usuário: os elementos podem ser criados para ambientes X/Motif e Microsoft Windows.

Rede: as potencialidades permitem construir aplicações que podem comunicar assincronamente sobre placas Ethernet entre diversos PCS e estações de trabalho UNIX. Isto permite que as simulações distribuídas sejam criadas onde uma mistura dos PCs e as estações de trabalho de UNIX suportam uma única simulação.

Portas seriais: simplificam a tarefa de comunicar-se com outras portas seriais.

4.3 Grafo de cena

As simulações em WTK são construídas montando-se nós em uma hierarquia chamada grafo de cena. O grafo de cena permite o uso eficiente da informação, transformação, e a presença de múltiplas janelas, cada qual com o seu próprio grafo de cena. Ele é uma hierarquia baseada em pilha.

4.3.1 Conceito e detalhes da Cena

Uma cena é uma coleção de geometrias e luzes, juntamente com as posições dos elementos. Em WTK, uma cena é simplesmente construída por quatro elementos: geometrias, luzes, informação posicional e fog.

A geometria e as luzes podem ser estáticas, que são carregadas na aplicação, e dinâmicas, que são criadas na aplicação. As luzes servem para iluminar alguma ou todas as geometrias da cena.

A informação posicional inclui as informações de posição associada com as geometrias e luzes, podem ser lidas de um arquivo ou criadas. Descrevem onde as geometrias e luzes podem ser encontradas na cena em relação a algum objeto ou a cena como todo.

O fog é um efeito especial que simula condições do ambiente, como fumaça, neblina, névoa. Pode ser usado em todas as geometrias ou também ser usado em algumas selecionadas.

4.3.2 Tipos de Nós WTK

O nó é o elemento fundamental. É usado para construir o grafo de cena pelo arranjo em uma hierarquia, como mostra a Figura 4.2. Os nós podem ser agrupados em três grupos: geometria, atributos e procedurais.

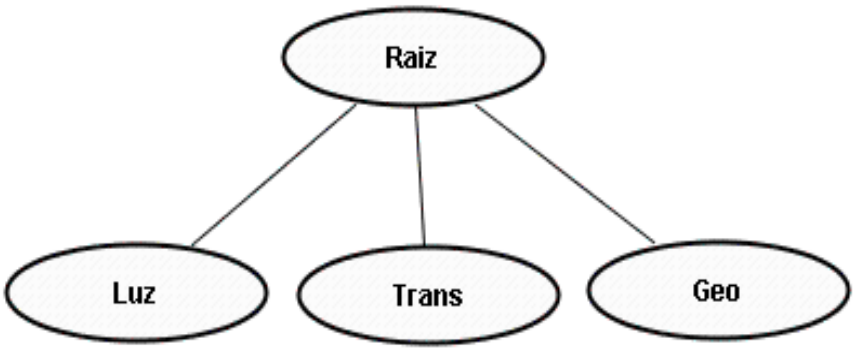

Figura 4.2 – Grafo de cena simples com quatro nós

Os nós geometria contêm a representação das entidades visíveis (nós de geometria (WTgeometry_newcylinder), nós de geometria móveis e texto 3D).

Os Atributos são usados para afetar a forma que os nós de geometria são renderizados (nós de neblina (fog), nós de luz (light), nós de luz móveis e nós de transformação (rotation)).

Os procedimentos são usados para controlar o modo que o grafo de cena será criado e processado (nós de âncora, nós de grupo, nós LOD, nós LOD móvel, nós switch, nós switch móveis, nós separadores, nós separadores móveis e nós separadores de transformação).

4.3.3 Hierarquia do Grafo de Cena

Os nós se mostram em uma forma hierárquica, como uma estrutura de árvore, de cima para baixo. Como mostra a Figura 4.3, o nó “pai” é aquele que tem pelo menos um nó fixado abaixo dele. Aqueles nós unidos debaixo de um outro nó são os "filhos" desse nó. Se dois nós compartilharem do mesmo pai, são considerados "irmãos".

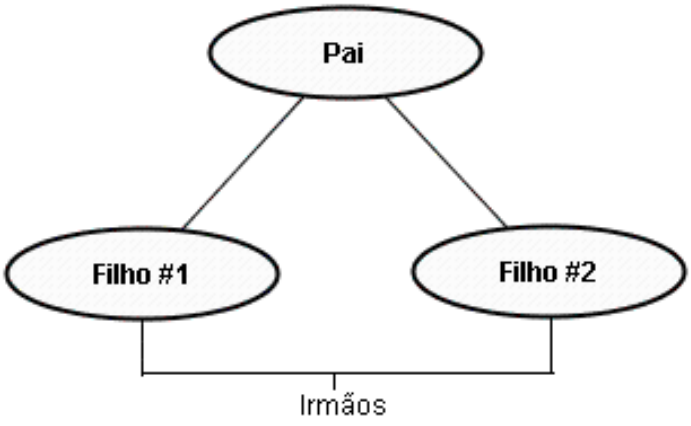

Figura 4.3 – Relacionamento de pai, filhos e irmãos.

Todos os grafos de cena em WTK requerem um ponto inicial. Este ponto é chamado root (raiz). O root fica no topo do grafo de cena, sendo o único nó da raiz, não podendo ser compartilhado com outros grafos da cena. O WTK permite múltiplos grafos de cena, e estes são excepcionalmente identificados por seus nós root individuais.

O nó root é ponto de entrada para o grafo de cena e o ponto onde o WTK inicia a construção da cena. Uma vez no root o WTK começa a percorrer a árvore transversalmente de cima para baixo e da esquerda para direita. O processo é sempre o mesmo. Quando é encontrado um nó com mais de um filho, percorre abaixo do ramo do primeiro filho, atravessando completamente esta parcela da árvore antes de retornar para o ramo acima, para depois processar com o ramo do segundo filho. A Figura 4.4 ilustra o percurso realizado.

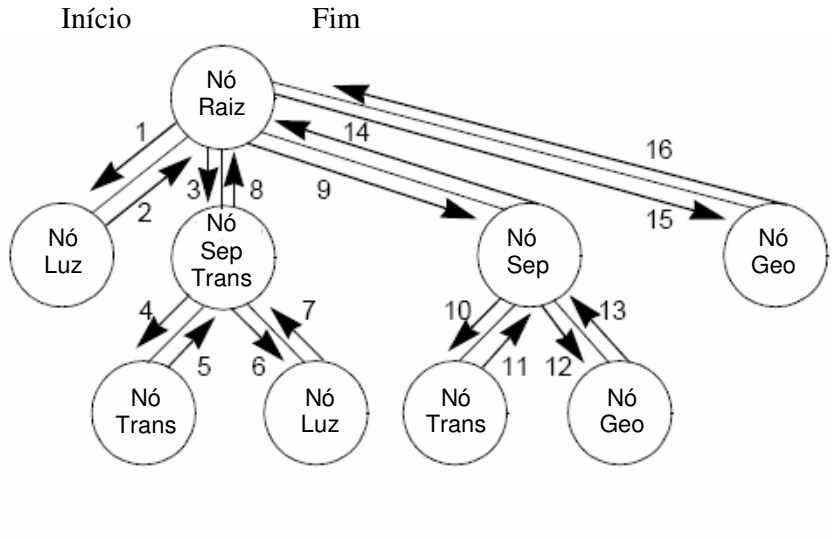

Figura 4.4 – Percurso do grafo de cena

É durante este processo de percorrer a árvore do grafo da cena que a cena é desenhada. Os nós encontrados na cena são avaliados e processados dependendo de seu tipo. Um exemplo disso, se um nó de geometria for encontrado ele é desenhado (na posição e na orientação atual, com luz e fog atuais); quando ele encontra um nó de luz, ele adiciona esta luz ao conjunto de luzes corrente; quando encontra um nó de transformação, modifica a posição e orientação atual; e quando encontra um nó fog, configura o fog corrente.

O WTK executa em um loop de simulação que passa por seis estágios diferentes: leitura de sensores, chamada de função da ação do universo, atualização de objetos, execução de tarefas, percorrer caminhos, e renderização do universo. É neste último estágio que WTK percorre os grafos de cena e desenha em uma janela (Figura 4.1).

4.4 Detecção de Colisão no WorldToolKit

No WorldToolKit, como visto nas seções anteriores, possui seu grafo de cena formado por nós . Os avatares e os objetos do ambiente são formados por nós diversificados. Para que aconteça a Detecção de Colisão entre os objetos, é necessário fazer um teste de intersecção entre os objetos que estão na iminência de colidir.

Os caminhos dos objetos podem ser usados para testar intersecções, sendo necessário primeiramente criar os caminhos de cada objeto com a função *WTnodepath_new*; logo após a criação do caminho, é feito o teste, que poderá ser com uma dessas funções: WTnodepath_intersectbbox, WTnodepath_intersectnode, WTnodepath_intersectpoly. .

A função WTnode_boundingbox coloca um envoltório, limitando a extensão máxima de um objeto, em sua posição e orientação atual. O objeto fica envolvido com uma caixa que pode ser visível ou não dependendo da simulação. A Figura 4.5 ilustra a função e a sintaxe, onde, node é o objeto que será envolvido com a caixa, e no FLAG onoff deverá ser utilizado TRUE, para permitir que a caixa fique visível, ou FALSE, para invisível.

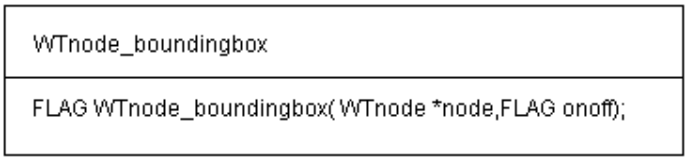

Figura 4.5 – Função Bounding Box

A função WTnodepath_intersectbbox verifica a intersecção de dois node paths, n1 e n2, baseados no Bounding Box. Esta função retorna verdadeiro se uma intersecção for encontrada, caso contrário ela retorna falso. A Figura 4.6 ilustra a função e a sintaxe, onde, n1 e n2 são os trajetos dos objetos para verificação de colisão.

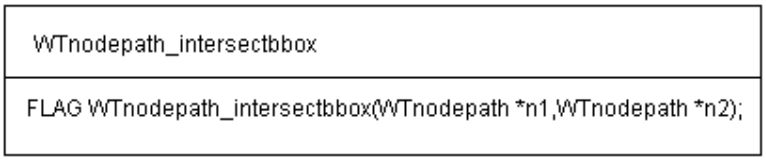

Figura 4.6 – Função WTnodepath_intersectbbox

A função WTnodepath_intersectnode executa um teste da intersecção entre o bounding box do node path especificado e a ocorrência numerada do nó especificado. Se não houver intersecção, retorna NULL, senão, a função busca o nó cujo bounding box possui a menor extensão, e ainda a intersecção do Bounding Box do node path especificado. Então um nodepath para aquele nó é criado e um ponteiro a ele é retornado. A Figura 4.7 ilustra a função e a sintaxe, onde, nodepath é o caminho do objeto, node é o objeto e int which é uma numeração.

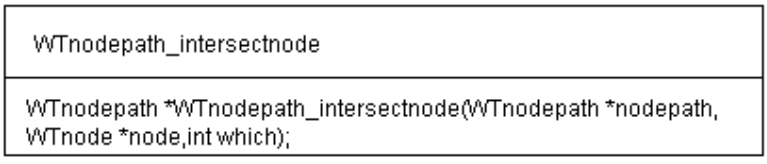

Figura 4.7 – Função WTnodepath_intersectnode

Para a função WTnodepath_intersectpoly não é utilizado o bounding box., é feito à intersecção de alguns polígonos em dois trajetos do nó (nodepath1 e nodepath2), se houver

uma intersecção retorna verdadeiro, se não retorna falso. A Figura 4.8 ilustra a função e a sintaxe, onde, nodepoly1 e nodepath 2 são o caminho dos objetos.

WTnodepath_intersectpoly
FLAG WTnodepath_intersectpoly(WTnodepath *nodepoly1, WTnodepath *nodepath2);

Figura 4.8 – Função WTnodepath_intersectpoly

CAPÍTULO 5 – DESCRIÇÃO DOS PROTÓTIPOS IMPLEMENTADOS

Com base nos técnicas de Detecção de Colisão descritas nos capítulos anteriores, foi elaborada uma aplicação que utiliza algumas técnicas em um Ambiente Virtual. Neste capítulo são apresentadas metodologias, objetivos e detalhes da implementação.

5.1 Aspectos gerais da Aplicação

Para trabalhar com um Ambiente Virtual é necessário torná-lo o mais realista possível, aumentando com isso a sua complexidade e desempenho. Como o Ambiente Virtual pode ser composto de objetos, e o usuário deve interagir com o mesmo, ocorre o problema da colisão gerada quando um avatar, por exemplo, tenta ocupar o espaço de outro objeto qualquer, sendo de fundamental importância para que eles, não ultrapassem uns aos outros como um "fantasma".

Para solucionar os problemas de colisão, foram evidenciadas com esse trabalho, algumas técnicas de Detecção de Colisão acompanhadas de condições, para que um objeto não ocupe o lugar do outro, visto que, as técnicas detectam, mas não impedem a colisão.

A quantidade de testes de colisão é proporcional à quantidade de objetos em um ambiente, levando em consideração a necessidade de o objeto ser testado. Um exemplo disso, é desnecessário testar a colisão do avião com o carro abaixo dele, sendo assim é feito uma análise para verificar a necessidade de cada objeto em cena.

Os testes primeiramente são para detectar a possível colisão, e em segundo passo o que fazer quando a colisão for detectada.

Nesta aplicação o usuário pode escolher o tipo de avatar e a técnica de Detecção de Colisão que ele quer aplicar nos objetos, movimentando um avatar sobre um objeto para verificar o procedimento escolhido.

5.2 Recursos de Software Utilizados na Implementação

Para a implementação das técnicas de Detecção de Colisão para avatares em um Ambiente Virtual, foram utilizados os seguintes recursos de software:

5.2.1 O WorldToolKit

De acordo com Netto(2002), para criar um Ambiente Virtual interativo, é necessário modelar um ambiente composto por múltiplos objetos e habitá-los com características virtuais. O aplicativo de RV é uma simulação animada que permite definir e exibir objetos (mesa, cadeira, parede, chão e avatar), alterar seu ponto de referência, manipulá-los interativamente, e fazer com que esses avatares colidem com os objetos.

Para o desenvolvimento da estrutura do ambiente da sala virtual foi utilizado o WorldToolKit Release 8, descrito com detalhes no Capítulo 4 desse trabalho.

5.2.2 Linguagens e Ferramenta utilizadas

O chão e a parede da aplicação foram feitos utilizando a linguagem VRML (Virtual Reality Modeling Language). Essa linguagem foi escolhida por possuir facilidades de importação, por não necessitar de softwares geradores e principalmente pela aceitação do WTK.

Esta linguagem nasceu em 1994, na primeira conferência anual da World Wide Web, com a necessidade de uma linguagem capaz de representar ambientes 3D interativos na internet. Os arquivos no formato VRML que simulam mundos 3D, são na verdade, uma descrição ASCII, de forma que um programador pode conceber tais mundos utilizando qualquer processador de texto (NETTO, 2002). Nesta aplicação foi utilizado um editor de texto simples para criar o chão e a parede.

A mesa, cadeira e o Avatar Bonfa foram modeladas através do WU (World Up Modeler) da Sense8. Com o WU o usuário cria simulações visuais nas quais os objetos gráficos possuem propriedades e comportamentos reais complexos, combinado várias tecnologias em um ambiente integrado.

O Delphi utiliza a linguagem de programação Object Pascal, possibilitando que se programem sistemas usando o paradigma de Programação Orientado a Objetos (POO), além de possuir tecnologia para implementações usando banco de dados locais, cliente/servidor e aplicações WEB. A tela de apresentação da aplicação foi feita através do Delphi 5.

O Microsoft Visual C++ 6.0 (também conhecido como MSVC) é um Ambiente de Desenvolvimento Integrado para as linguagens C, C++ e C++/CLI, desenvolvido pela Microsoft. Ele contém ferramentas para o desenvolvimento de software escritos especialmente para as APIs do ambiente Windows, para a API DirectX e para o framework Microsoft .NET.

Esta aplicação foi desenvolvida utilizando o WorldToolKit, juntamente com o Microsoft Visual C++. A escolha dessa linguagem ocorreu pelo fato dela ser orientada a objeto e compatível com as funções do WTK.

O Ambiente Virtual aqui descrito consta de um avatar e uma sala com mesa e cadeira. O usuário leva o Avatar ao encontro da mesa ou cadeira para verificar a colisão.

5.3 Demonstração da tela inicial

A tela inicial como ilustra a Figura 5.1, foi elaborada em Delphi, sendo utilizada para fazer um link com as aplicações. O usuário através da escolha do avatar (Bonfa, Max ou Mini) irá entrar em uma janela para escolher o tipo de método de colisão.

O menu Sobre especifica os autores do programa, o menu Sair, encerra a aplicação e o menu ajuda mostra quais as teclas que o usuário deverá utilizar para manipular o avatar.

Figura 5.1 – Tela inicial

A programação é realizada através de procedimentos, como ilustra a Figura 5.2 e a Figura 5.3. Para cada avatar existe uma chamada para uma janela, onde o usuário poderá escolher o método a ser executado pelo avatar. Os procedimentos da Figura 5.2 representam a chamada à janela, os procedimentos da Figura 5.3 representam as informações dos autores, seguido da Ajuda, e por último o procedimento de encerramento do programa.

```
procedure TForm2.SpeedButton1Click(Sender: TObject);
begin
form1.show;
end;

procedure TForm2.SpeedButton3Click(Sender: TObject);
begin
form3.show;
end;

procedure TForm2.SpeedButton2Click(Sender: TObject);
begin
form4.show;
end;
```

Figura 5.2 – Código fonte da Tela inicial

```
procedure TForm2.Sobre1Click(Sender: TObject);
begin
form5.show;
end;

procedure TForm2.Ajuda1Click(Sender: TObject);
begin
form6.show;
end;

procedure TForm2.Sair1Click(Sender: TObject);
begin
application.terminate;
end;
end;
```

Figura 5.3 – Código fonte da Tela inicial para ajuda e encerramento do programa

A Figura 5.4 mostra a escolha pelo avatar Bonfa, onde o usuário tem opção de optar pelo Bounding Box com IntersectBox, Bounding Box com IntersectNode ou pelo Poly intersectpoly, clicando na imagem de porta aberta ao lado, ou ainda a opção retornar, na imagem de porta fechada.

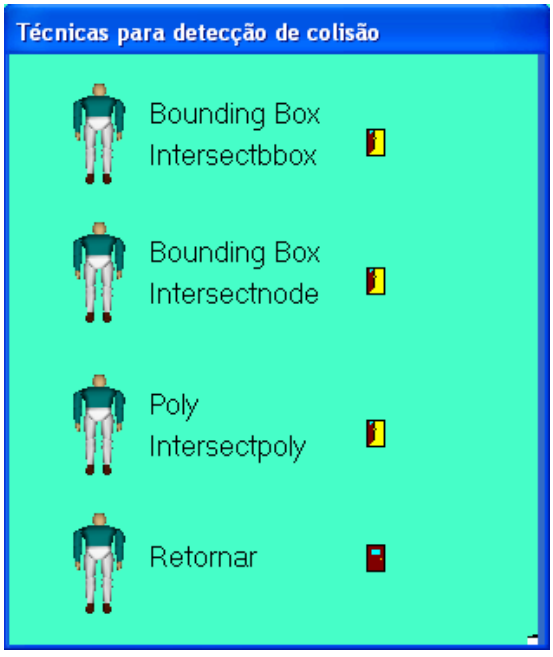

Figura 5.4 – Tela de opções das técnicas para o Avatar Bonfa

As Figuras 5.5 e 5.6 mostram as mesmas opções do Avatar Bonfa, mas para a aplicação com o Avatar Max e Mini respectivamente.

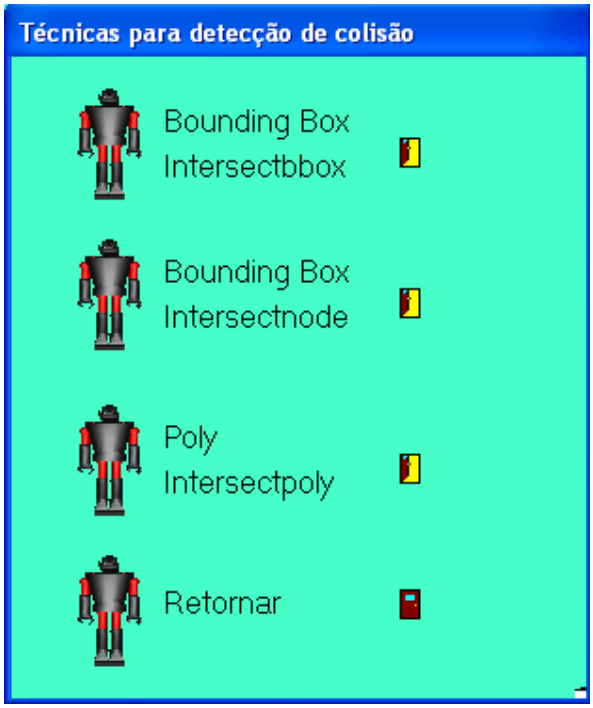

Figura 5.5 – Tela de opções das técnicas para o Avatar Max

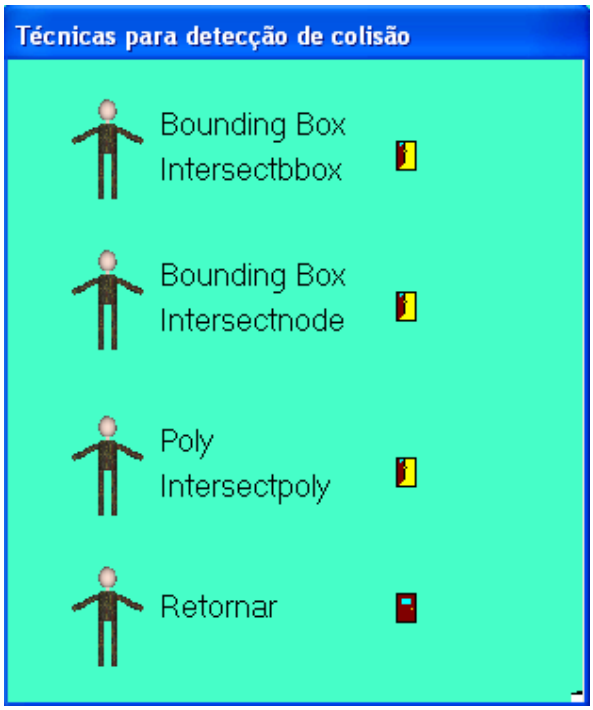

Figura 5.6 – Tela de opções das técnicas para o Avatar Mini

Para que um avatar qualquer possa ter movimento no AV, é necessário manipular o mesmo através de teclas. Para isso a janela Ajuda, como mostra a Figura 5.7, irá apresentar as teclas utilizadas nessa aplicação.

Figura 5.7 – Tela de Ajuda para manipular os Avatares

Para que o usuário possa entrar em contato com os desenvolvedores dessa aplicação foi colocada a tela de Autores, como mostra a Figura 5.8

Figura 5.8 – Tela de informação dos autores

5.4 Formação do Ambiente Virtual

A Figura 5.9 mostra o código fonte do chão e das paredes, criadas com o VRML através do Bloco de Notas, onde, o filename chama uma imagem para servir de textura e o cube é uma geometria que forma através da altura, largura e espessura o objeto chão e parede.

```
Chão
#VRML V1.0 ascii
Texture2 { filename "WOOd.jpeg"
           }
            Cube {
          width 8000
          height 2
          depth 8000
                }
```

```
Parede
#VRML V1.0 ascii
Texture2 { filename "bv4.jpeg"
           }
            Cube {
          width 8000
          height 4500
          depth 1
                }
```

Figura 5.9 – Código fonte do chão e da parede

Para cada Avatar foram desenvolvidas três aplicações diferentes, cada qual com uma técnica de Detecção de Colisão.

O Ambiente Virtual foi construído utilizando um conjunto de geometrias com suas características, como: textura, posição, rotação e translação.

A mesa e cadeira foram modeladas no WTK e depois chamadas para a aplicação. A Figura 5.10 mostra a formação do ambiente juntamente com a escala e posicionamento do mesmo.

```
void cenario()
{
  cena = WTmovnode_load (root,"C:\\Colisão\\sala\\cadeira.nff", 11.0f);
  cena1 = WTmovnode_load (root,"C:\\Colisão\\modelos\\vrml\\chao.wrl", 80.0f);
  cena2 = WTmovnode_load (root,"C:\\Colisão\\modelos\\vrml\\parede.wrl", 80.0f);
  cena3 = WTmovnode_load (root,"C:\\Colisão\\sala\\mesaf.nff", 10.0f);
        poscena[0]=3500.0;
        poscena[1]=700.0;
        poscena[2]=14.0;
        WTnode_settranslation(cena,poscena);
        poscena1[0]=2000.0;
        poscena1[1]=1500.0;
        poscena1[2]=2.0;
        WTnode_settranslation(cena1,poscena1);
        poscena2[0]=2000.0;
        poscena2[1]=80.0;
        poscena2[2]=500.0;
        WTnode_settranslation(cena2,poscena2);
          poscena3[0]=2400.0;
          poscena3[1]=800.0;
          poscena3[2]=14.0;
          WTnode_settranslation(cena3,poscena3);
}
```

Figura 5.10 – Trecho do programa para formação do Ambiente Virtual

5.5 Demonstração dos Avatares

O Avatar Bonfa foi modelado utilizando-se o modelador do WTK; feito por partes e chamado na aplicação por inteiro. A Figura 5.11 mostra a chamada do Avatar, com sua escala, posição e translação para que ele possa encaixar no Ambiente Virtual. A Figura 5.12 mostra o ambiente em execução.

```
void avatar()
{
        //          avatar inteiro
        fig=WTmovnode_load(root,"C:\\Colisão\\Modelos\\cabeca.nff",1.2f);
        posavatar[0]=0;
        posavatar[1]=-150.0;
        posavatar[2]=0.0;
        WTnode_settranslation(fig,posavatar);
        //  ****************
}
```

Figura 5.11 – Formação do Avatar Bonfa

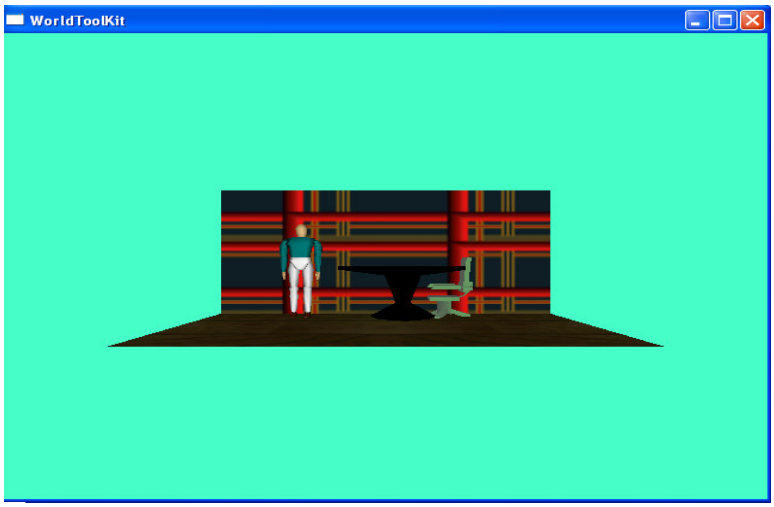

Figura 5.12 – Avatar Bonfa no Ambiente Virtual

O Avatar Max foi modelado utilizando-se o 3D Studio; feito por partes e chamado na aplicação por partes também. A Figura 5.13 mostra a chamada do Avatar, feita através de um procedimento de chamadas para cada membro, montagem da hierarquia e o posicionamento. A Figura 5.14 mostra o ambiente em execução.

```
void carregarRobo()
{
/*cabeca e tronco*/
noBacia  = WTmovnode_load(root,"C:\\colisao\\robo\\Debug\\bacia.3ds", 8.2);
p[0] = 110; p[1] = -1400; p[2] = 0;
WTnode_settranslation(noBacia, p);
. . .
}
void montarHierarquia()
{
        /*criacao da hierarquia*/
        WTmovnode_attach(noBacia,  noTronco,    0);
. . . }
void posicionarMembros()
{
        /*colocando os membros do corpo no lugar correto*/
        /*bacia*/
        p[0] = 110; p[1] = -1400; p[2] = 0;
        WTnode_settranslation(noBacia, p);
. . .
}
void montarRobo()
{
        carregarRobo(parent);
        montarHierarquia();
        posicionarMembros();
}
```

Figura 5.13 – Formação do Avatar Max

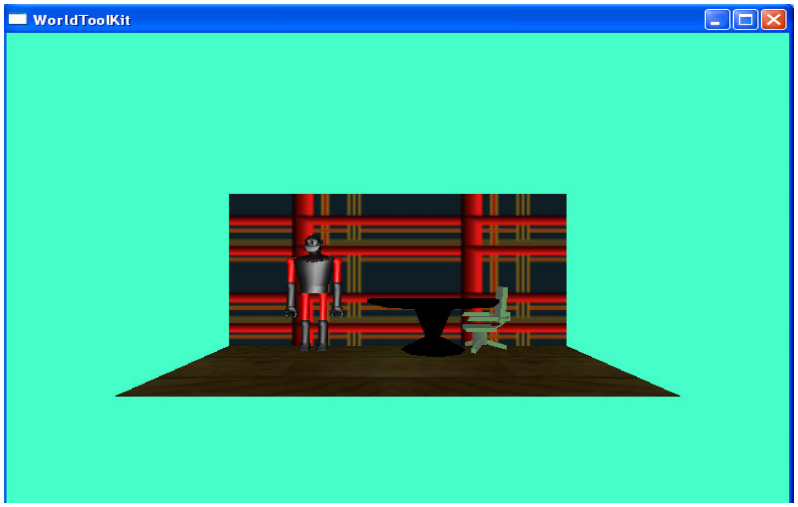

Figura 5.14 – Avatar Max no Ambiente Virtual

O Avatar Mini foi modelado dentro da própria aplicação; feito por partes e agrupado. A Figura 5.15 mostra o procedimento da montagem do Avatar, juntamente com o posicionamento. A Figura 5.16 mostra o ambiente em execução.

```
void avatar()
{
        /*Corpo*/
        corpo= WTgeometry_newcylinder(11, 4, 12, FALSE, TRUE);
        cil=WTmovgeometrynode_new(root,corpo);
        WTgeometry_setrgb(corpo,0,100,255);
        /* Posicionamento do corpo*/
        posc[0]=0.0;
        posc[1]=-5.0;
        posc[2]=0.0;
        WTnode_translate(cil,posc, WTFRAME_PARENT);
        WTnode_settranslation(cil,posc);
        WTgeometry_settexture(corpo, "camo1", TRUE, FALSE);
        WTgeometry_scale(corpo, 70.0,p);
...
}
```

Figura 5.15 – Trecho da Formação do Avatar Mini

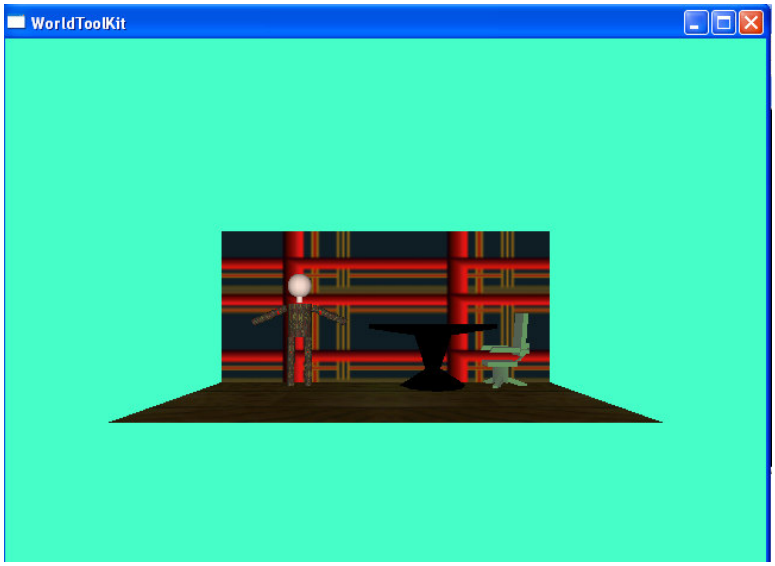

Figura 5.16 – Avatar Mini no Ambiente Virtual

5.6 Descrição das Funções e Técnicas

As técnicas descritas nesta aplicação envolvem três diferentes tipos, onde, um é a interseção de polígonos, e os outros dois, intersecção de envoltórios. Esses últimos envolvem os objetos com uma caixa, e verifica se algum dos lados da caixa de um objeto está encostando-se a um dos lados da caixa do outro objeto. Se esta alternativa for verdadeira ocorre colisão. A técnica sem envoltório é o teste do polígono, que verifica se cada polígono de um objeto encontrou com o polígono de um outro objeto.

5.6.1 Protótipo Bonfa com intersectbbox

A Figura 5.17 mostra o grafo de cena do Avatar Bonfa com intersectbbox, seguido do código fonte formado de: variáveis, programa principal, formação de cenário (Figura 5.11), formação do avatar (Figura 5.12), e ação.

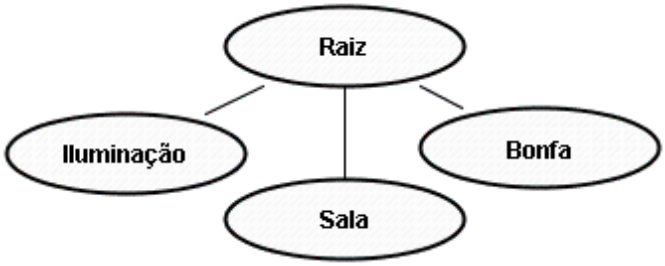

Figura 5.17 – Grafo de Cena do Avatar Bonfa

a) Variáveis

```
#include "wt.h"
static void show_framerate ();
void avatar();
void cenario();
void action();
WTp3 pos,posbola,posesf, poscena,poscena1, poscena2,poscena3,posavatar;
WTnode *child,*root;*bola,*sph, *a,*sept, *sept1; …
```

b) Principal

```
void main()
{
        WTuniverse_new(WTDISPLAY_DEFAULT, WTWINDOW_DEFAULT);
        root = WTuniverse_getrootnodes();
        WTlight_load("lights");
        view = WTuniverse_getviewpoints();
        WTuniverse_setbgrgb(70,255,200);
        sensor = WTmouse_new();
        cenario();  /*Chamada do cenário*/
        avatar();  /*Chamada do avatar*/. . .
 }
```

c) Colocar envoltório

```
if (Key== 'B')
        {
                /*Bounding na Cena*/
                WTnode_boundingbox(cena,TRUE);
                WTnode_boundingbox(cena1,TRUE); . . .
                /*Bounding no humanoide*/
                WTnode_boundingbox(fig,TRUE);
                . . .
        }
```

d) Detectar colisão através de movimentos com as teclas

```
if (Key == 'f')
 {   WTnode_gettranslation(fig, pos);
     /*teste de intersecção*/
   If (WTnodepath_intersectbbox(avt,cen)||WTnodepath_intersectbbox(avt,cen3))
                        {       WTmessage(" Ocorreu Colisão \n");
                                . . .
                        }
```

5.6.2 Protótipo Bonfa com intersectnode

O Avatar Bonfa usando intersectnode possui as mesmas variáveis, programa principal, formação de cenário e formação do avatar Bonfa intersectbbox, diferenciando, portanto a ação (Figura 5.18).

```
if (Key == 'f')
{ WTnode_gettranslation(fig, pos);
  if (WTnodepath_intersectnode(cen,fig,0)||WTnodepath_intersectnode(cen3,fig,0))
                {
                        WTmessage(" Ocorreu Colisão \n");
                        . . .
                }
        }
```

5.18 – Detecção de colisão utilizando a técnica intersectnode

5.6.3 Protótipo Bonfa com intersecpoly

O Avatar Bonfa intersecpoly, possui as mesmas variáveis, programa principal, formação de cenário e formação do avatar Bonfa intersectbbox, diferenciando, portanto a ação, e o fato dele não possuir envoltório (Figura 5.19).

```
if (Key == 'r')
{       WTnode_gettranslation(fig, pos);
        if(WTnodepath_intersectpoly(avt,cen)||WTnodepath_intersectpoly(avt,cen3))
                        {
                                WTmessage(" Ocorreu Colisão \n");
                                . . .
                        }
        `
```

5.19 – Detecção de colisão utilizando a técnica intersectpoly

5.6.4 Protótipo Mini com intersectbbox

A Figura 5.20 apresenta o grafo de cena do Avatar Mini com intersectbbox, seguido do código fonte formado de: variáveis, programa principal, formação de cenário (Figura 5.15), formação do avatar (Figura 5.16), e ação.

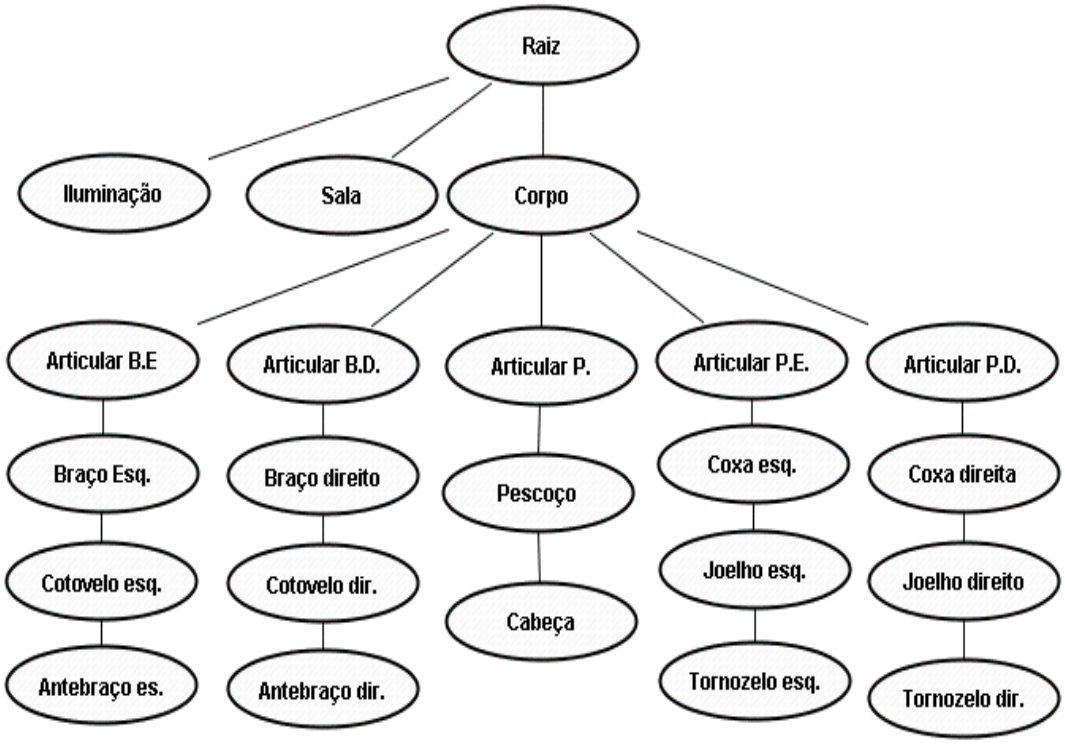

Figura 5.20 – Grafo de Cena do Avatar Mini

a) Variáveis

```
#include "wt.h"

static void show_framerate ();
void avatar();
void cenario();
void action();
WTp3 p,poscena,poscena1, poscena2, poscena3, pos, posmd, posbid, posbie; …
```

b) Principal

```
void main()
{
        WTuniverse_new(WTDISPLAY_DEFAULT, WTWINDOW_DEFAULT);
         root = WTuniverse_getrootnodes();
        WTlight_load("lights");
        view = WTuniverse_getviewpoints();
        WTuniverse_setbgrgb(70,255,200);
        sensor = WTmouse_new();
        cenario();
        avatar();
 }
```

c) Colocando Bounding box

```
if (Key== 'B')
        {
                /*Bounding no corpo do avatar*/
                WTnode_boundingbox(cil,TRUE);
                /*Bounding na cabeca*/
                WTnode_boundingbox(sph,TRUE); …
                /*Bounding na Cena*/
                WTnode_boundingbox(cena,TRUE);
        }
```

d) Detectar colisão através de movimentos com as teclas

```
if (Key == 'r')
{
    WTnode_gettranslation(cil, pos);
    if(WTnodepath_intersectbbox(avt,cen)||WTnodepath_intersectbbox(avt,cen3))
     {
        WTmessage(" Ocorreu Colisão \n");
        WTnode_gettranslation(cil, pos);
        pos[2] = pos[2] +10;
        WTnode_settranslation(cil, pos);
        WTmessage("\nPolygons: %6d, Frame rate: %8.2f fps\n",
        WTwindow_numpolys(WTuniverse_getwindows()), WTuniverse_framerate());
     }
```

5.6.5 Protótipo Mini com intersectnode

O Avatar Mini com intersectnode possui as mesmas variáveis, programa principal, cenário e avatar Mini intersecbbox, diferenciando, portanto a ação (Figura 5.21).

```
if (Key == 'f')
{
  WTnode_gettranslation(sept, pos);
  if(WTnodepath_intersectnode(cen,sept,0)||WTnodepath_intersectnode(cen3,sept,0))
  {
    WTmessage(" Ocorreu Colisão \n");
   }
}
```

Figura 5.21 – Detecção de Colisão utilizando a técnica intersectnode para o Avatar Mini

5.6.6 Protótipo Mini com poly

O Avatar Mini com intersectpoly possui as mesmas variáveis, programa principal, formação de cenário e formação do avatar Mini intersectnode, diferenciando, portanto a ação e o fato de que o intersectpoly não possui envoltório (Figura22).

```
if (Key == 'f')
{
  WTnode_gettranslation(sept, pos);
if(WTnodepath_intersectpoly(avt,cen)||WTnodepath_intersectpoly(avt,cen3))
  {
    WTmessage(" Ocorreu Colisão \n");
   }
}
```

Figura 5.22 – Detecção de Colisão utilizando a técnica intersectpoly para o Avatar Mini

5.6.7 Protótipo Max

O Avatar Max possui as mesmas ações de detecção colisão e cenário do Avatar Mini, diferenciando, portanto, nas variáveis e formação do avatar.

a) Variáveis

```
#include "wt.h"
#include "math.h"
static void show_framerate ();
void cenario();
void action();
void montarRobo();
WTp3 pos,posbola,posesf, poscena,poscena1, poscena2,poscena3,posavatar; . . .
```

b) Formação do Avatar

```
void carregarRobo()
{
/* carregando arquivos*/
noBacia = WTmovnode_load(root,"C:\\Documents and Settings\\Nome\\Desktop\\Mestrado\\robo\\Debug\\bacia.3ds", 8.2);
p[0] = 110; p[1] = -1400; p[2] = 0;
WTnode_settranslation(noBacia, p);
noCabeca = WTmovnode_load(NULL, "C:\\Documents and Settings\\Nome\\Desktop\\Mestrado\\robo\\Debug\\cabeca.3ds", 8.2);
noTronco = WTmovnode_load(NULL, "C:\\Documents and Settings\\Nome\\Desktop\\Mestrado\\robo\\Debug\\tronco.3ds", 8.2);
}
```

c) Hierarquia do avatar

```
void montarHierarquia()
{
        /*criacao da hierarquia*/
        WTmovnode_attach(noBacia,  noTronco,    0);
        WTmovnode_attach(noTronco, noCabeca,    0); . . .
}
```

d) Posicionamento dos membros do avatar

```
void posicionarMembros()
{
        /*colocando os membros do corpo no lugar correto*/
        /*bacia*/
        p[0] = 110; p[1] = -1400; p[2] = 0;
        WTnode_settranslation(noBacia, p);
        /*tronco*/
        p[0] = 0; p[1] = -30; p[2] = 0;
        WTnode_translate(noTronco, p, WTFRAME_LOCAL);
        /*parafusos*/
        p[0] = 0; p[1] = 0; p[2] = 0;
        WTnode_translate(noParafusos, p, WTFRAME_LOCAL);
}
```

e) Montagem geral do avatar

```
void montarRobo()
{
        carregarRobo(parent);
        montarHierarquia();
        posicionarMembros();
}
```

CAPÍTULO 6 – RESULTADOS

Neste capítulo são apresentados os resultados com a efetiva comparação entre as técnicas de Detecção de Colisão, de acordo com os frames por segundo, utilizando para isso três avatares diferentes.

Os Resultados foram obtidos utilizando uma máquina AMD Athlon (tm) XP 2400 com 1,99 GHz e 512 MB de Ram. O Sistema Operacional escolhido foi o Windows XP Professional.

Embora a aplicação esteja em uma chamada dentro do Delphi, não houve diferença nos frames por segundo (FPS).

6.1 Comparações entre os FPS

As comparações foram feitas utilizando três técnicas de Detecção de Colisão para cada Avatar, e entre cada técnica por todos os avatares. Para tanto foram verificadas a média entre 10, 20 e 50 colisões. A Figura 6.1 mostra um trecho do programa onde ocorre a verificação dos FPS no ato da colisão.

```
WTmessage("\nPolygons: %6d, Frame rate: %8.2f fps\n",
WTwindow_numpolys(WTuniverse_getwindows()), WTuniverse_framerate());
```

Figura 6.1 – Verificação dos FPS

6.1.1 Comparações entre os FPS do Protótipo Bonfa

O resultado observado nas Figuras 6.2, 6.3, 6.4 e 6.5 demonstram que o Intersectbox é o mais rápido, seguido do Intersectnode, e o mais demorado é o IntersectPoly, visto nos capítulos anteriores a causa refletida na intersecção entre os polígonos.

Quando o teste é feito utilizando um envoltório, o objeto a ser testado é a caixa, pois o objeto fica dentro da mesma. No caso do IntersectPoly, não há envoltório, fazendo com que o teste seja feito polígono por polígono.

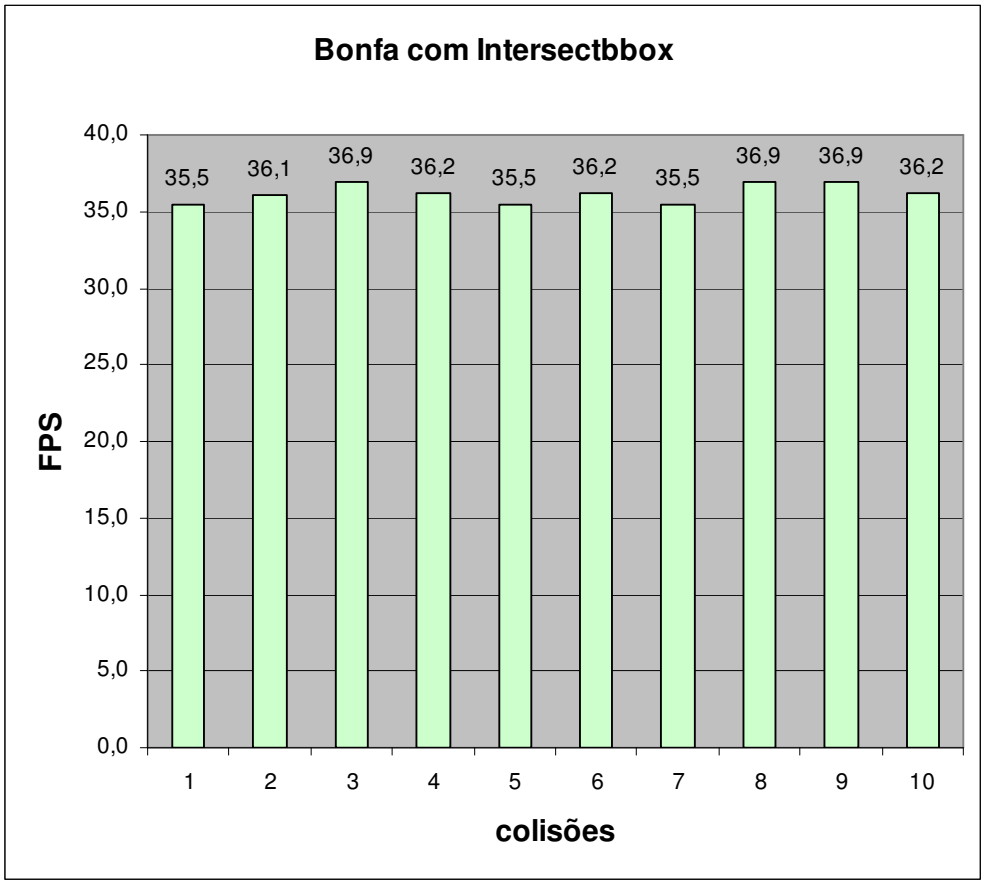

Figura 6.2 – FPS de 10 colisões com Intersectbbox do protótipo Bonfa

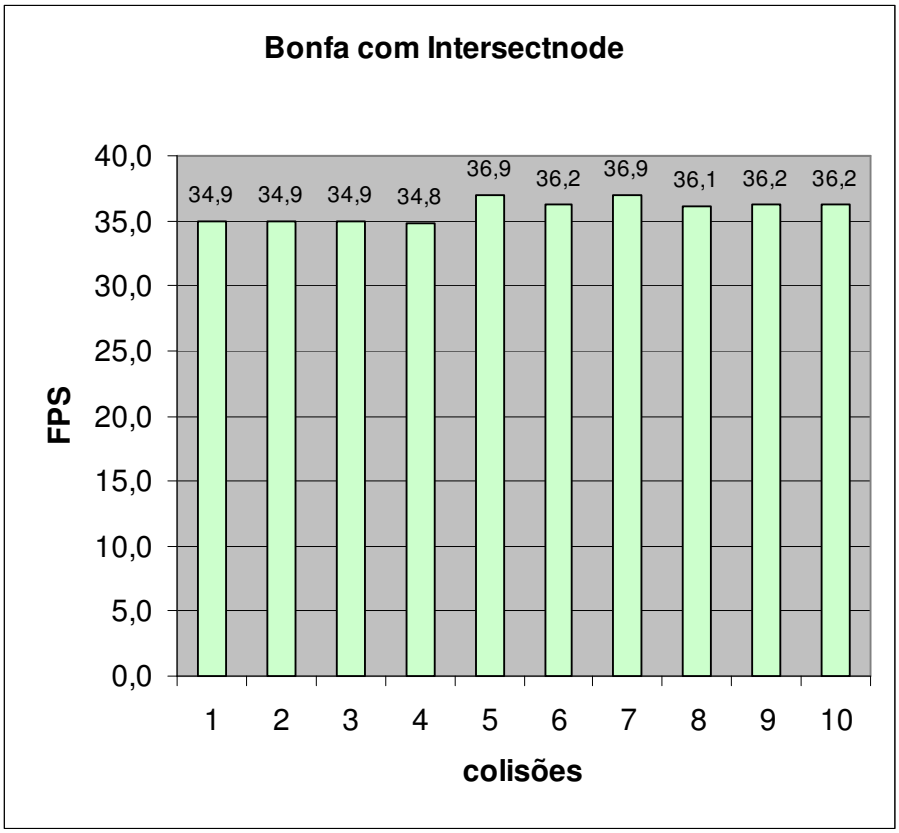

Figura 6.3 – FPS de 10 colisões com Intersectnode do protótipo Bonfa

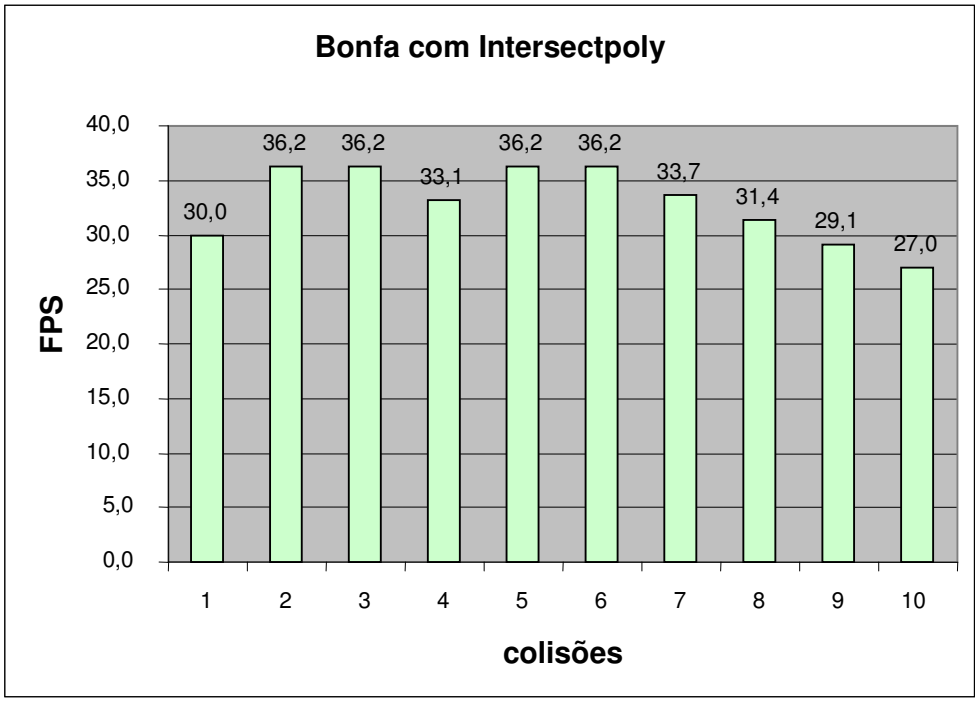

Figura 6.4 – FPS de 10 colisões com Intersectpoly do Protótipo Bonfa

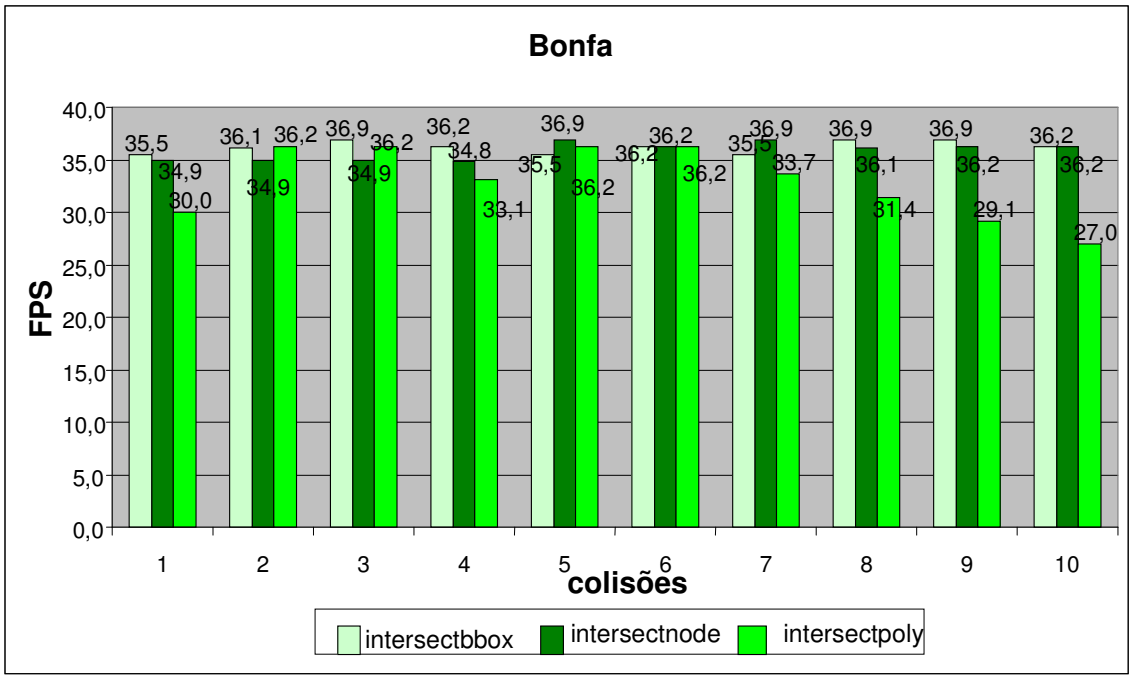

Figura 6.5 – FPS com 10 colisões entre as três técnicas com o Protótipo Bonfa

As Figuras 6.6, 6.7, 6.8 e 6.9 apresentam os FPS entre 20 colisões e as Figuras 6.10, apresenta a média dos FPS entre 50 colisões.

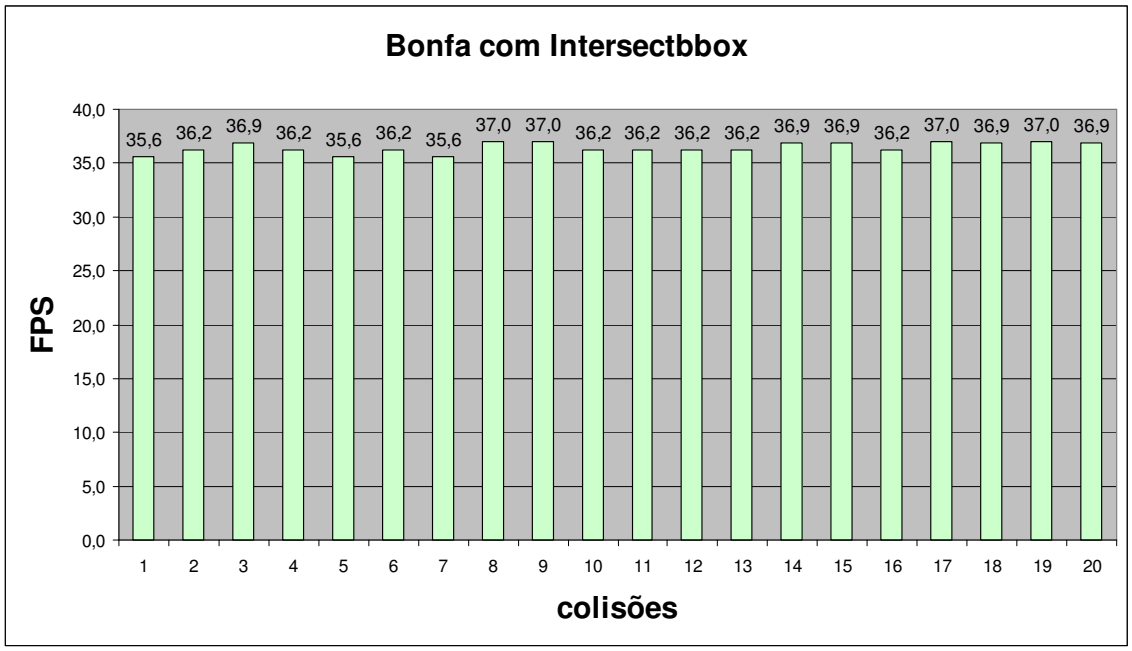

Figura 6.6 – FPS de 20 colisões com Intersectbbox do Protótipo Bonfa

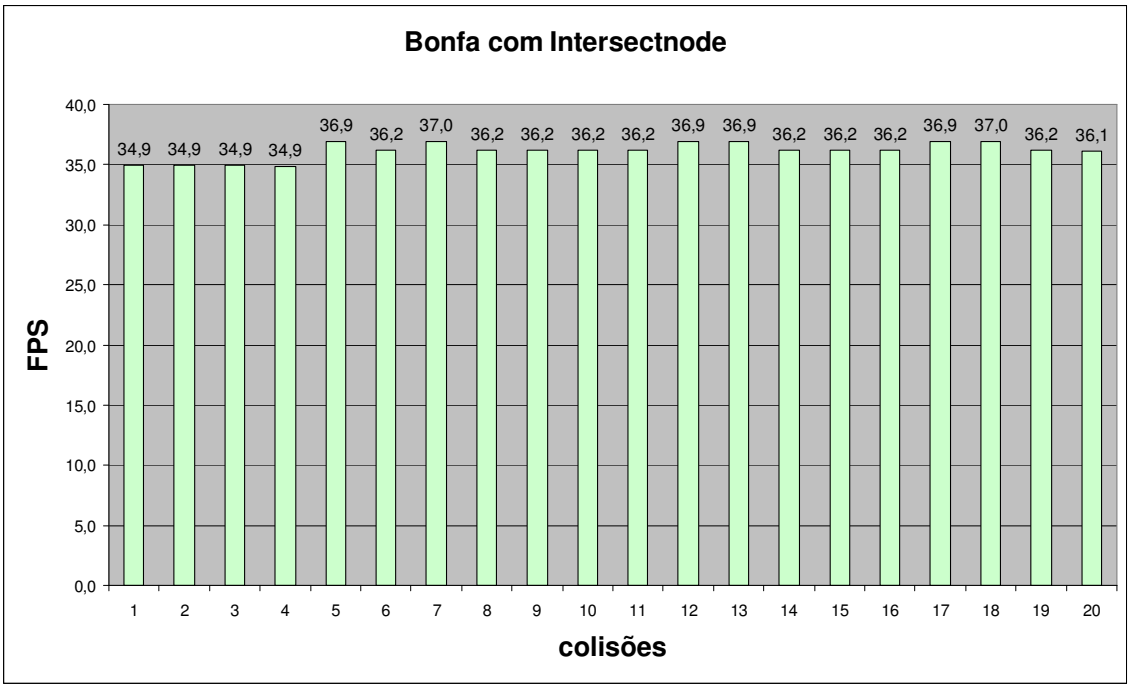

Figura 6.7 – FPS de 20 colisões com Intersectnode do protótipo Bonfa

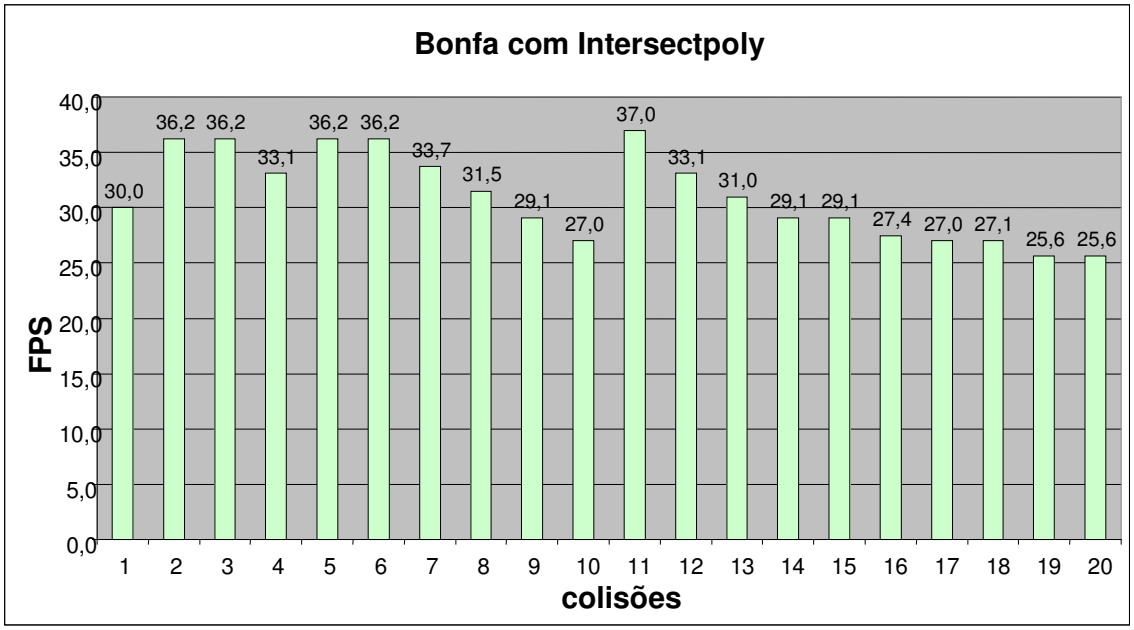

Figura 6.8 – FPS de 20 colisões com Intersectpoly do protótipo Bonfa

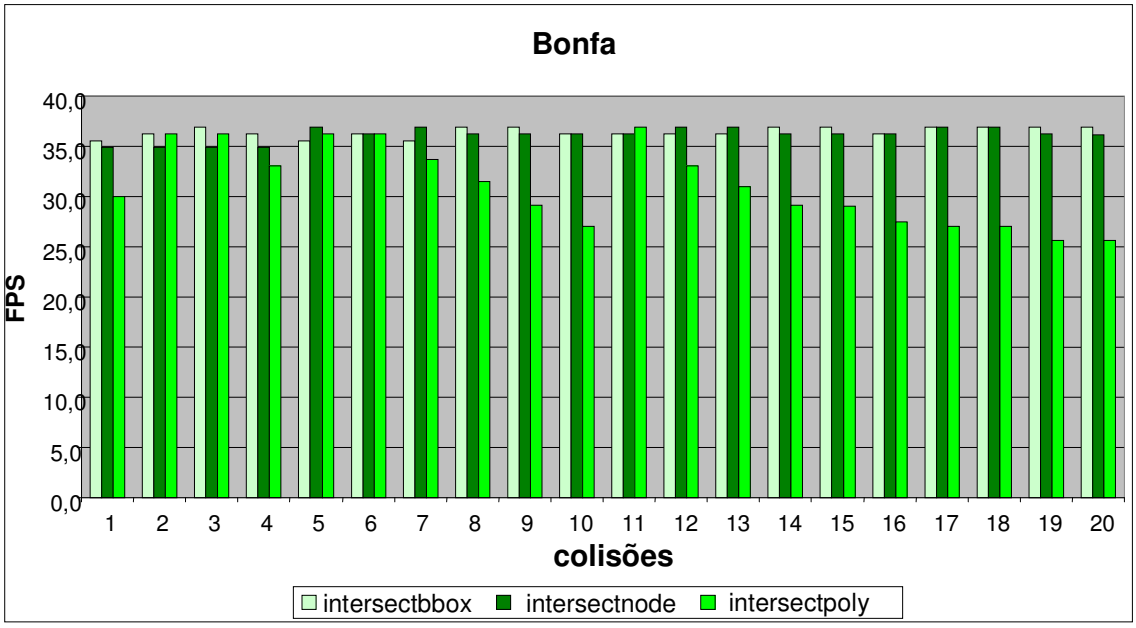

Figura 6.9 – FPS com 20 colisões entre as três técnicas com protótipo Bonfa

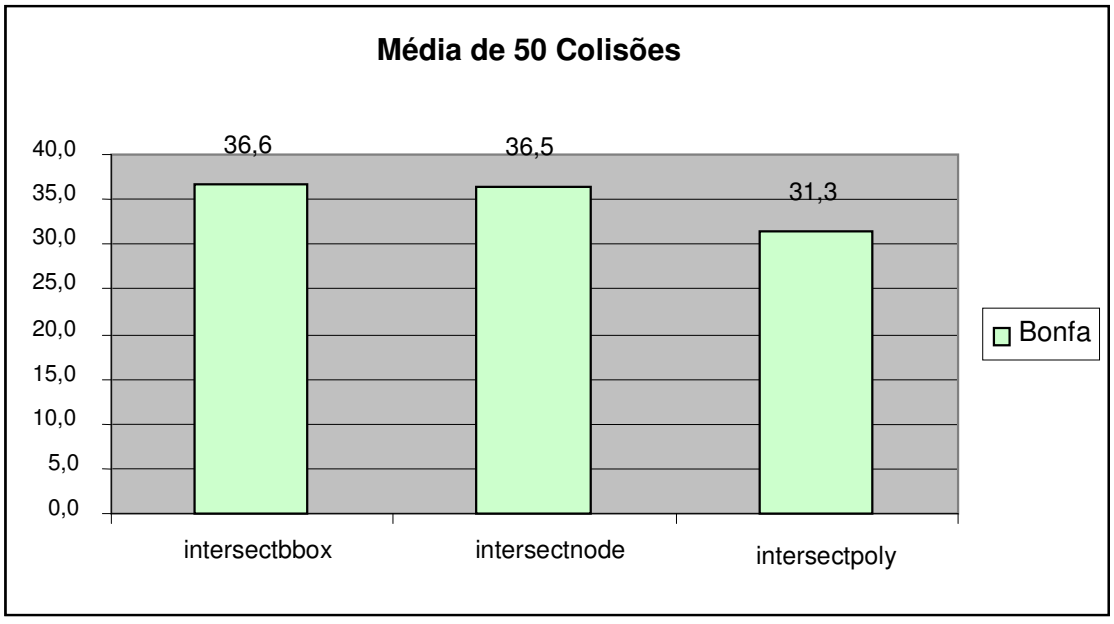

Figura 6.10 – Média de 50 colisões com o protótipo Bonfa

As médias das técnicas entre 50 colisões foram: intersectbbox com 36,5, intersectnode com 36,4, e intersectpoly com 31,3.

6.1.2 Comparações entre os FPS do Protótipo Mini

O resultado observado nas Figuras 6.11, 6.12, 6.13 e 6.14 demonstra que o Intersectnode é o mais rápido, seguido do Intersectbox, e o mais demorado é o IntersectPoly.

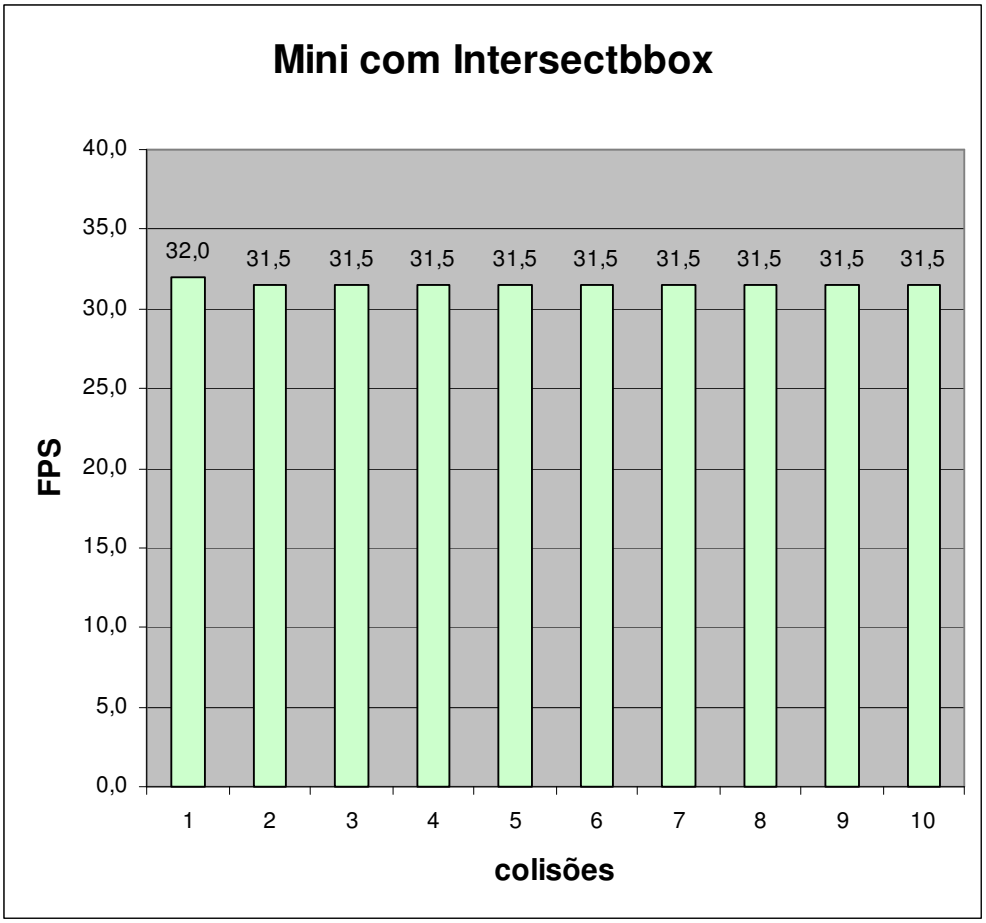

Figura 6.11 – FPS de 10 colisões com Intersectbbox do protótipo Mini

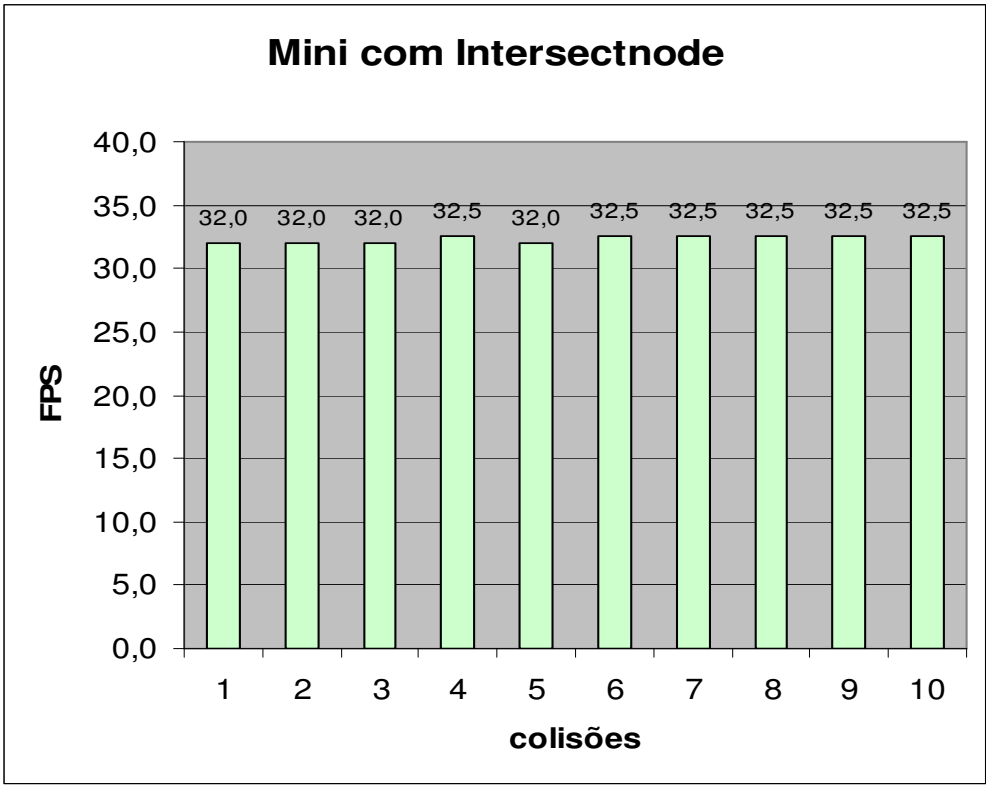

Figura 6.12 – FPS de 10 colisões com Intersectnode do protótipo Mini

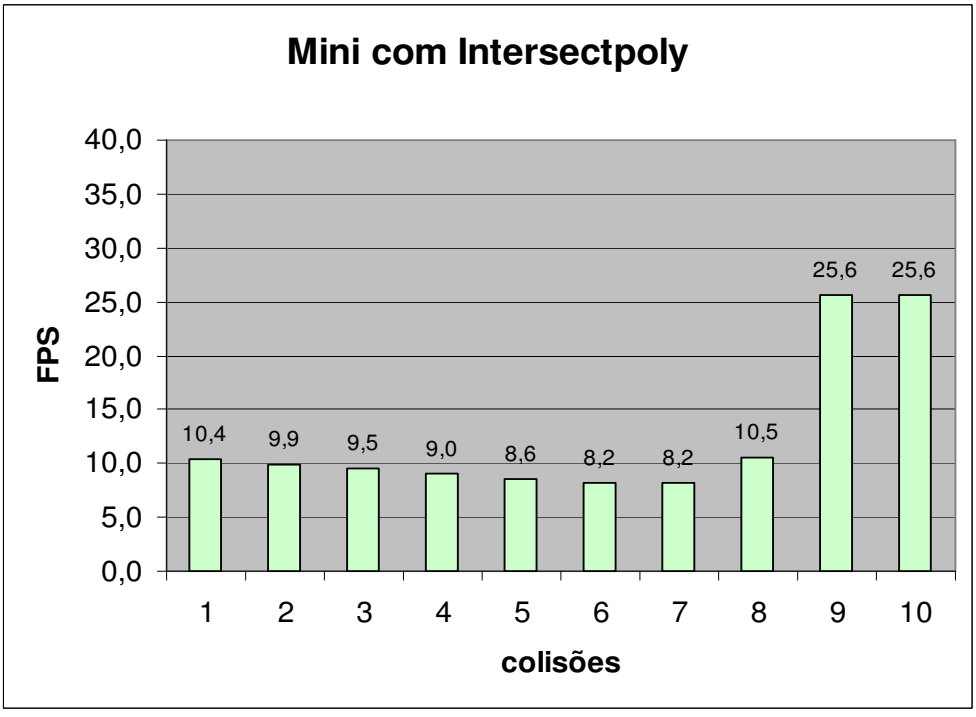

Figura 6.13 – FPS de 10 colisões com Intersectpoly do protótipo Mini

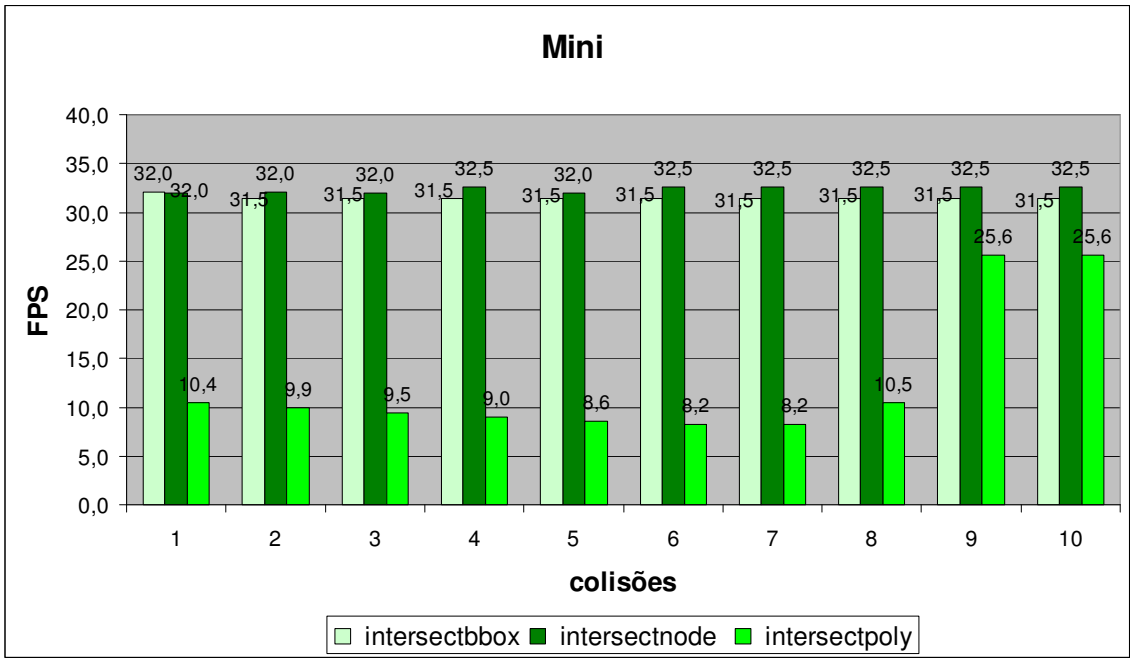

Figura 6.14 – FPS com 10 colisões entre as três técnicas do protótipo Mini

As Figuras 6.15, 6.16, 6.17 e 6.18 apresentam os FPS entre 20 colisões e a Figura 6.19 apresenta a média dos FPS entre 50 colisões.

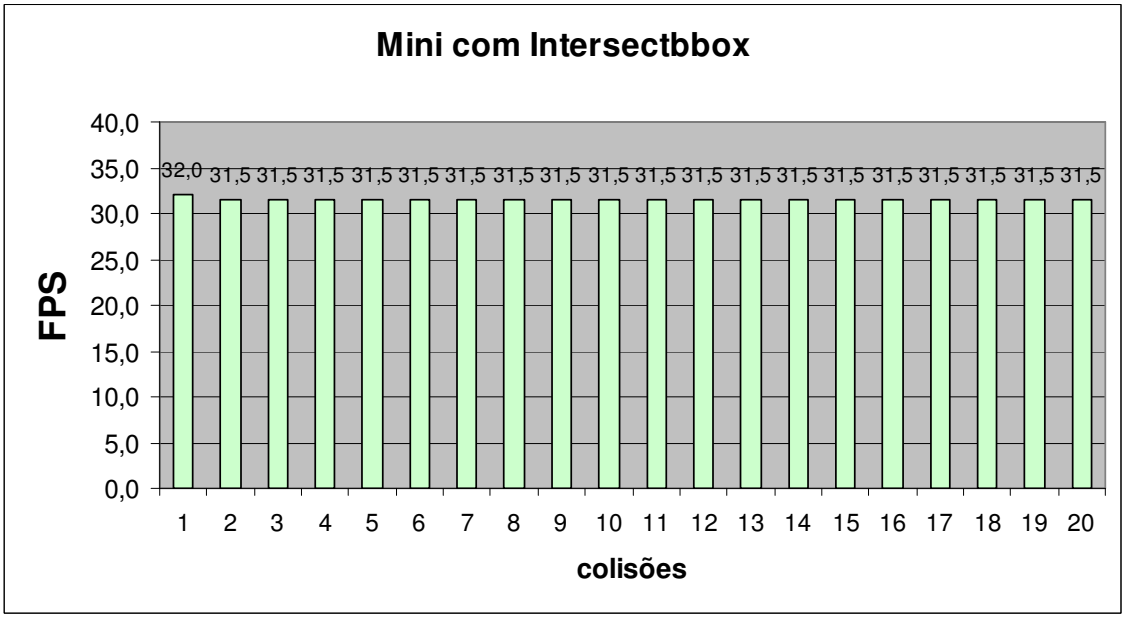

Figura 6.15 – FPS de 20 colisões com Intersectbbox do protótipo Mini

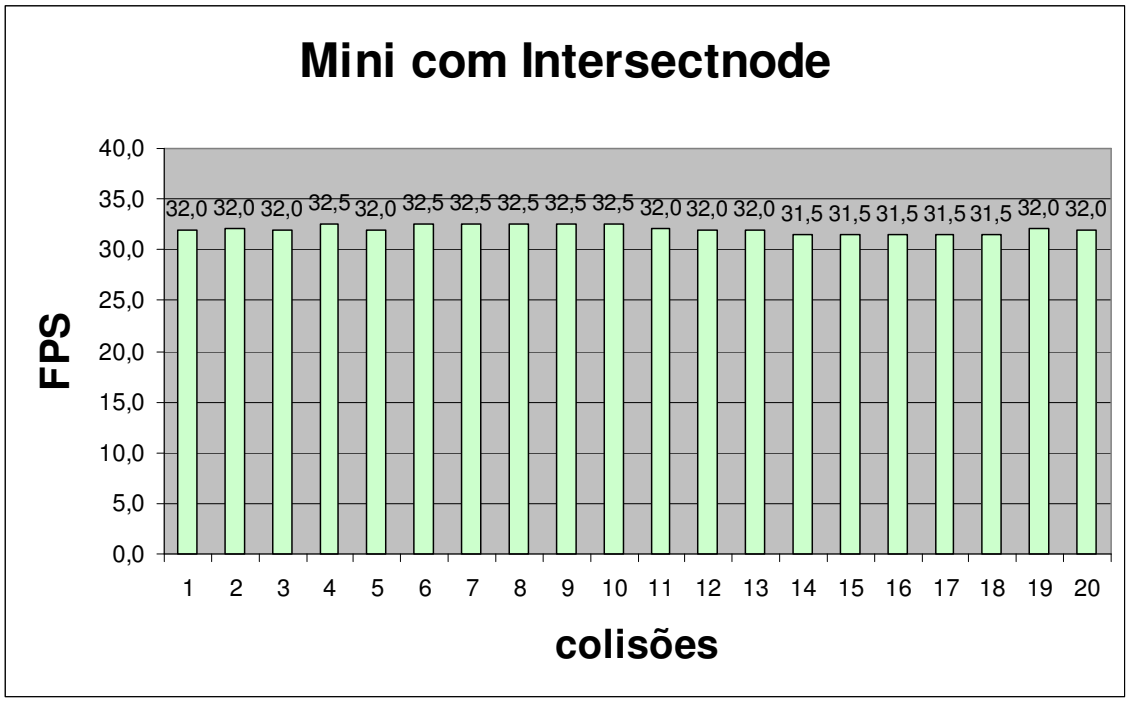

Figura 6.16 – FPS de 20 colisões com Intersectnode do protótipo Mini

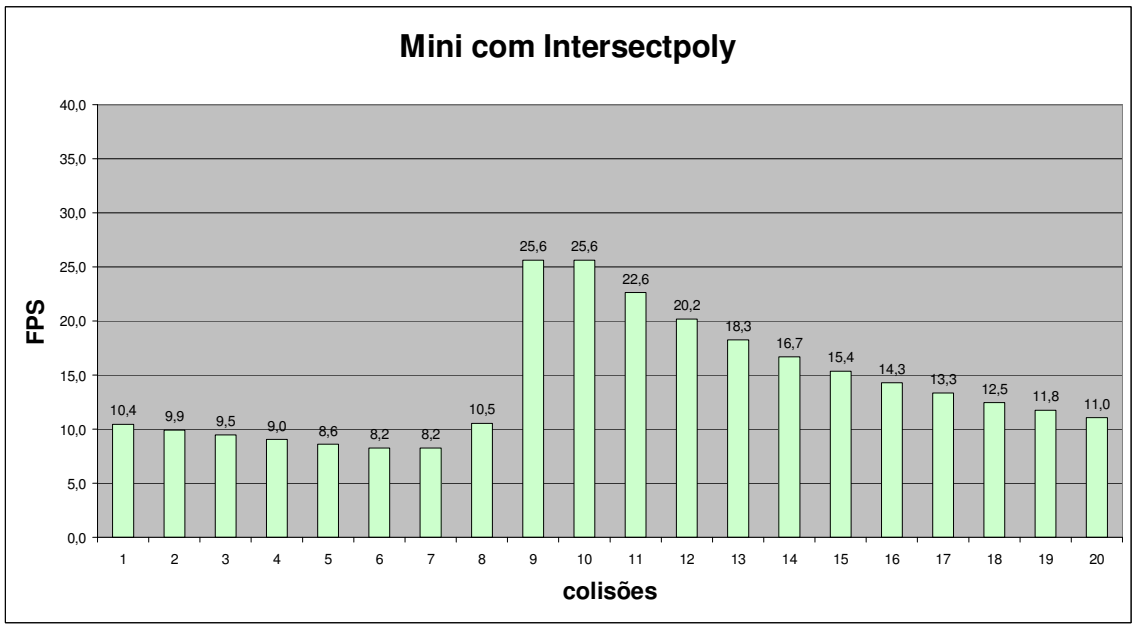

Figura 6.17 – FPS de 20 colisões com Intersectpoly do protótipo Mini

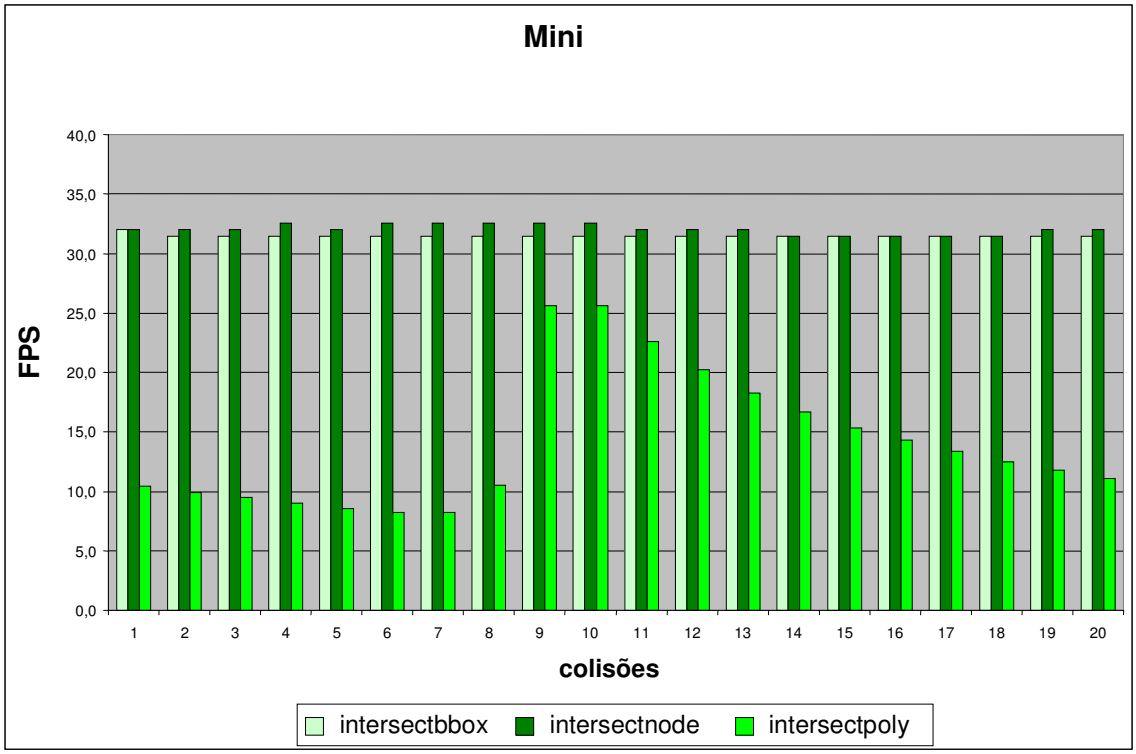

Figura 6.18 – FPS com 20 colisões entre as três técnicas com protótipo Mini

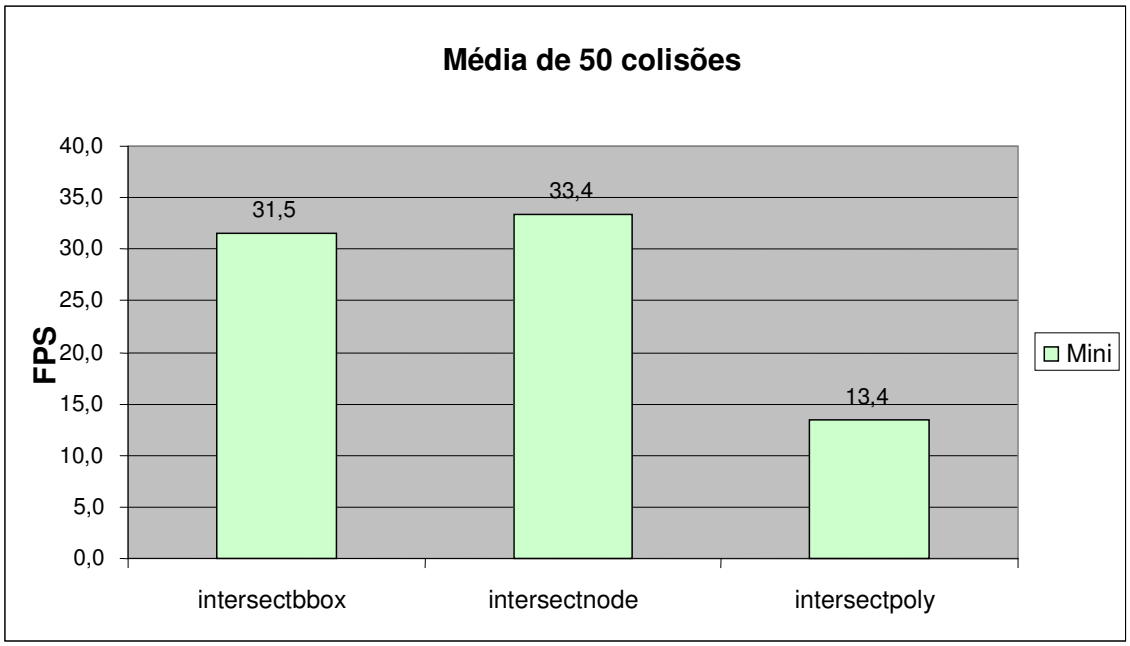

Figura 6.19 – Média de 50 colisões com o protótipo Mini

As médias das técnicas entre 50 colisões foram: intersectbbox com 31,4, intersectnode com 33,4, e intersectpoly com 13,4.

6.1.3 Comparações entre os FPS do Protótipo Max

O resultado observado nas Figuras 6.20, 6.21, 6.22 e 6.23 demonstra que o Intersectnode é o mais rápido, seguido do Intersectbox, e o mais demorado é o IntersectPoly.

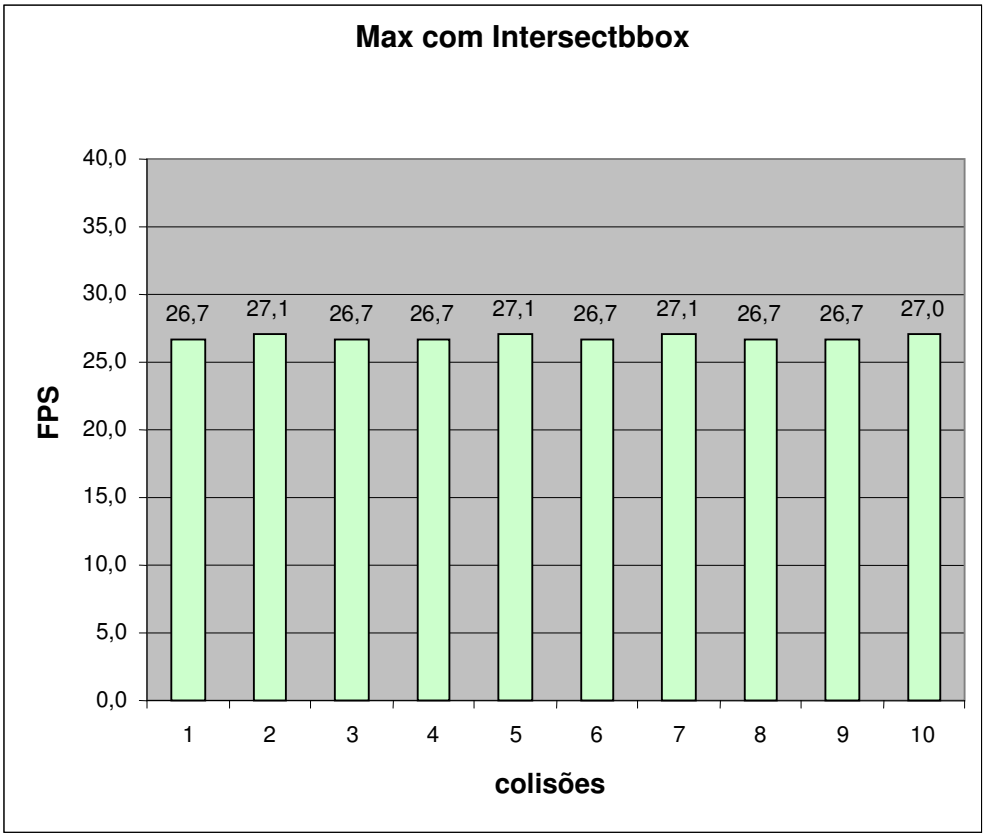

Figura 6.20 – FPS de 10 colisões com Intersectbbox do protótipo Max

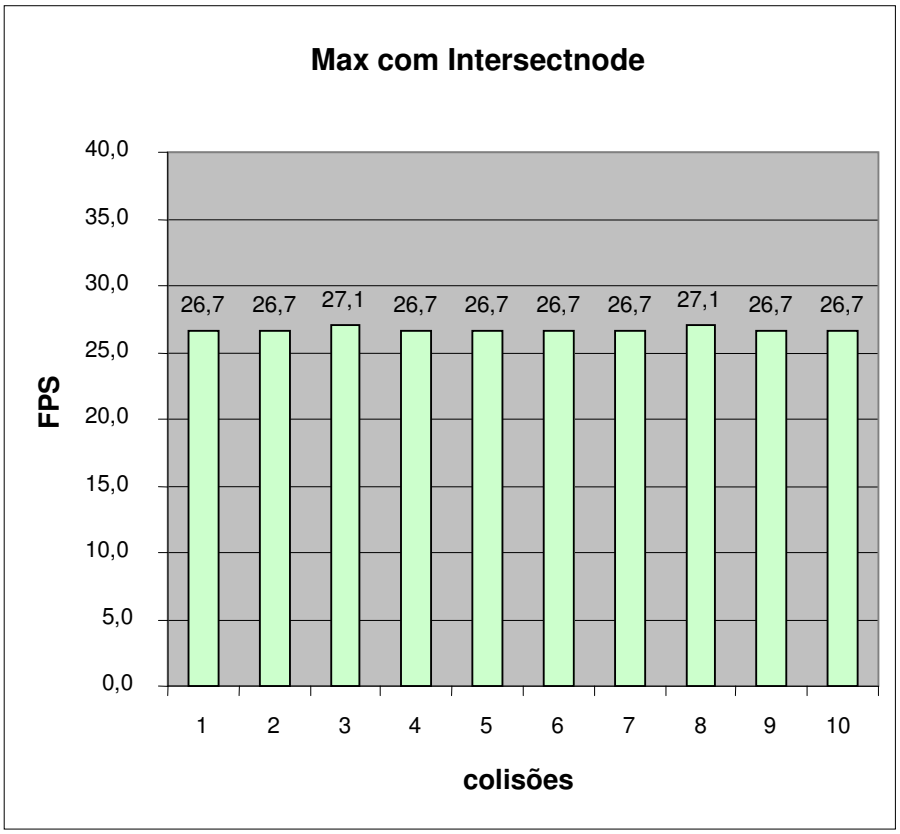

Figura 6.21 - FPS de 10 colisões com Intersectnode do protótipo Max

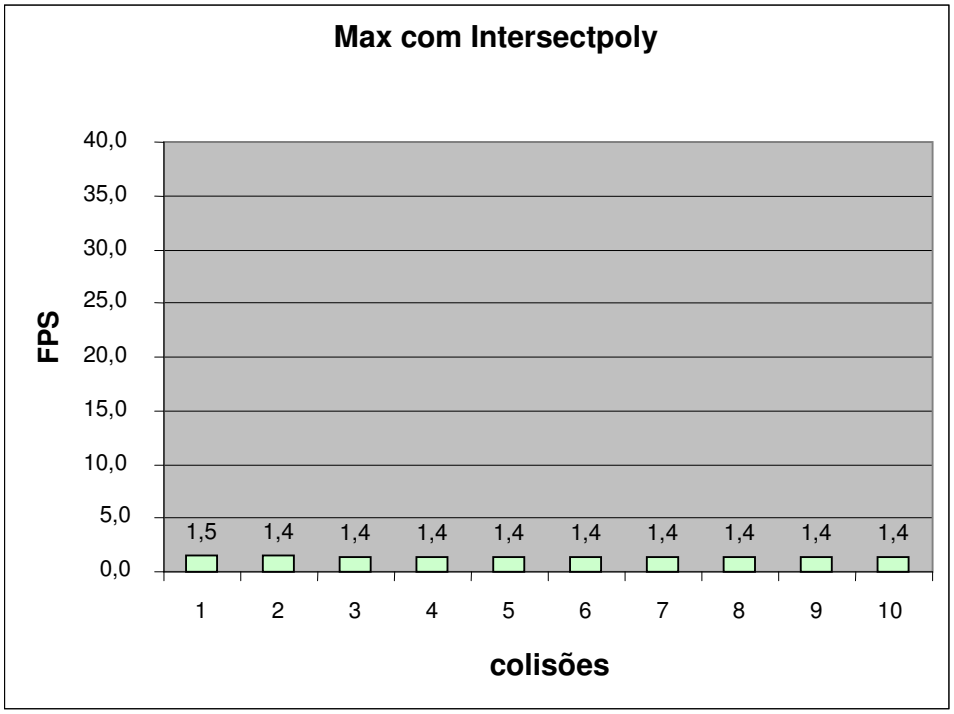

Figura 6.22 – FPS de 10 colisões com Intersectpoly do protótipo Max

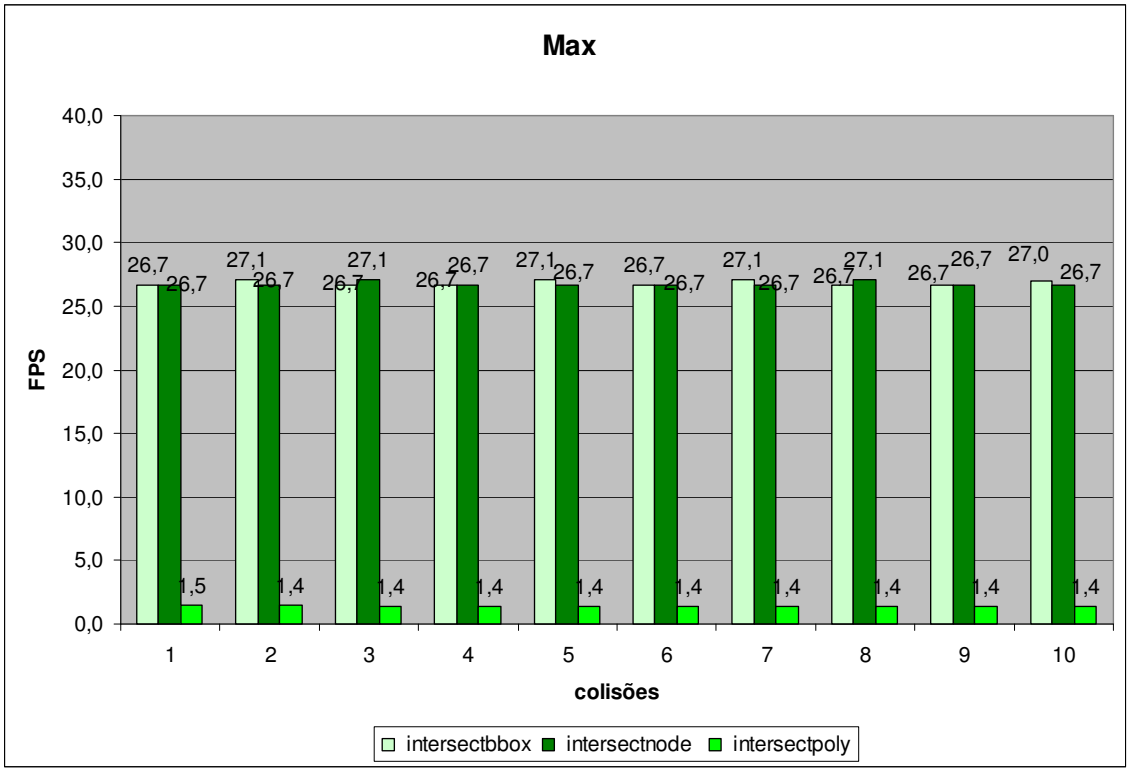

Figura 6.23 – FPS com 10 colisões entre as três técnicas com protótipo Max

As Figuras 6.24, 6.25, 6.26 e 6.27 apresentam os FPS entre 20 colisões e a Figura 6.28 apresenta a média dos FPS entre 50 colisões.

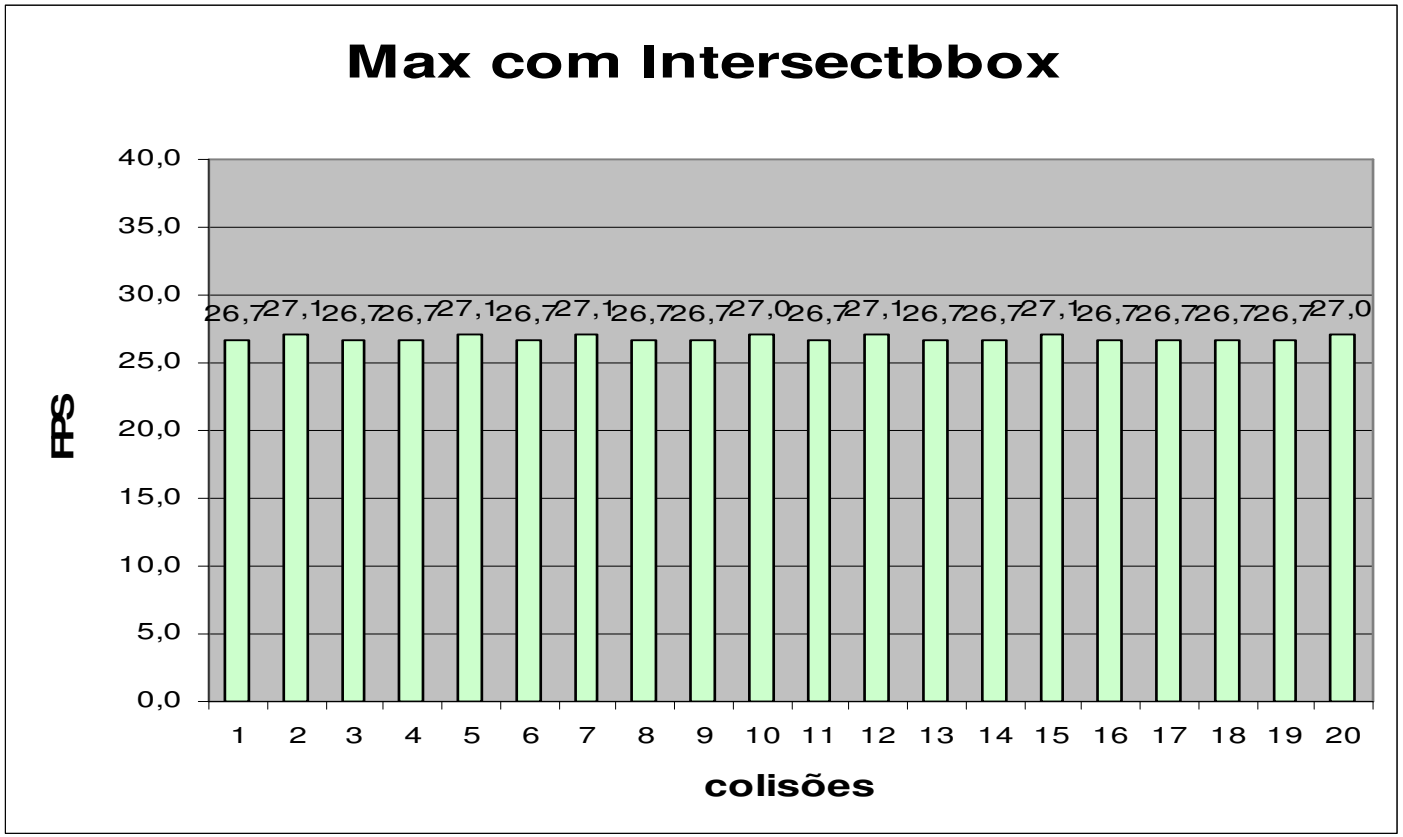

Figura 6.24 – FPS de 20 colisões com Intersectbbox do protótipo Max

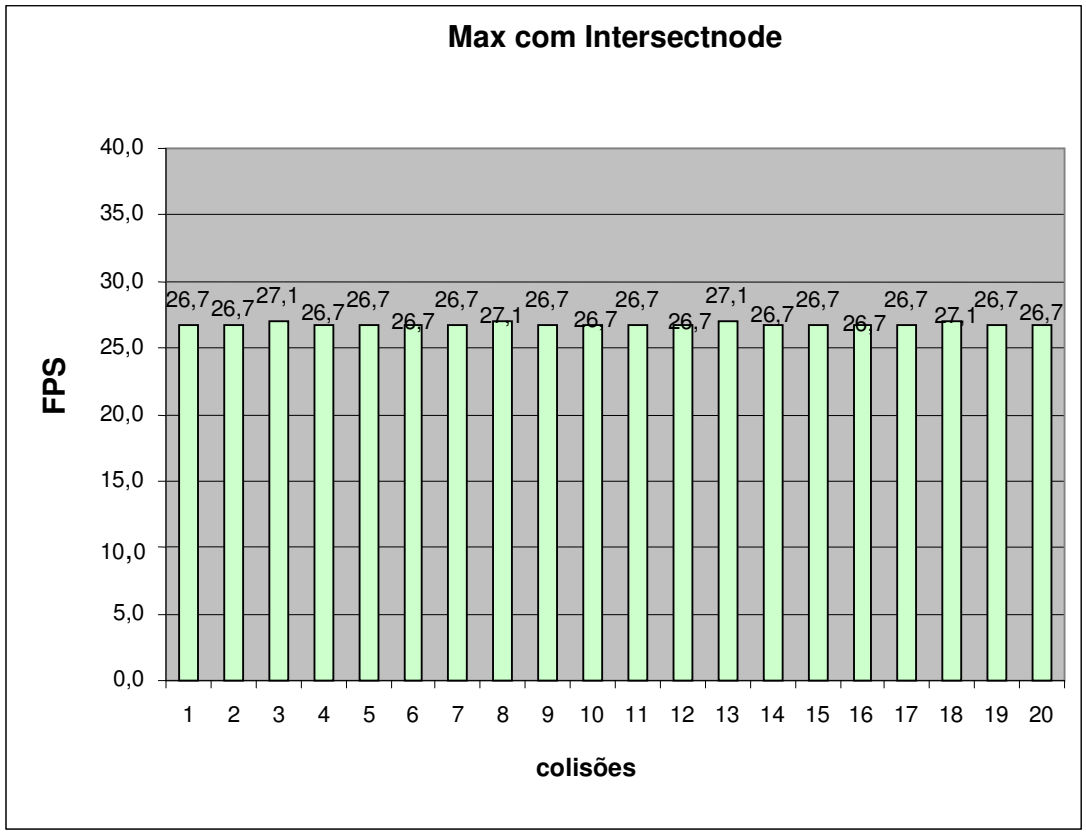

Figura 6.25 – FPS de 20 colisões com Intersectnode do protótipo Max

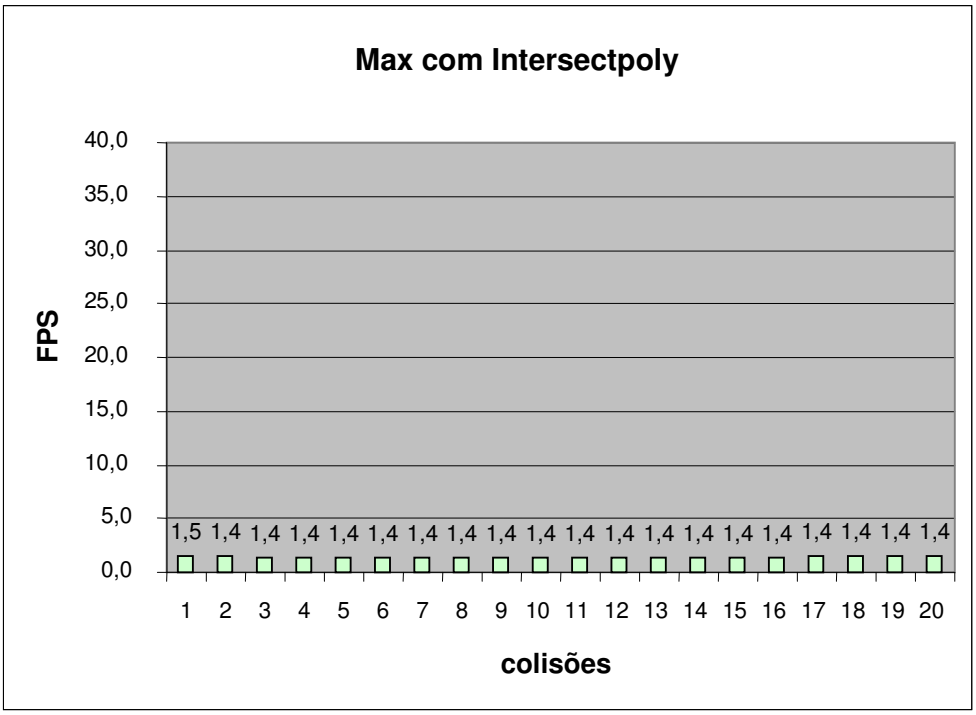

Figura 6.26 – FPS de 20 colisões com Intersectpoly do protótipo Max

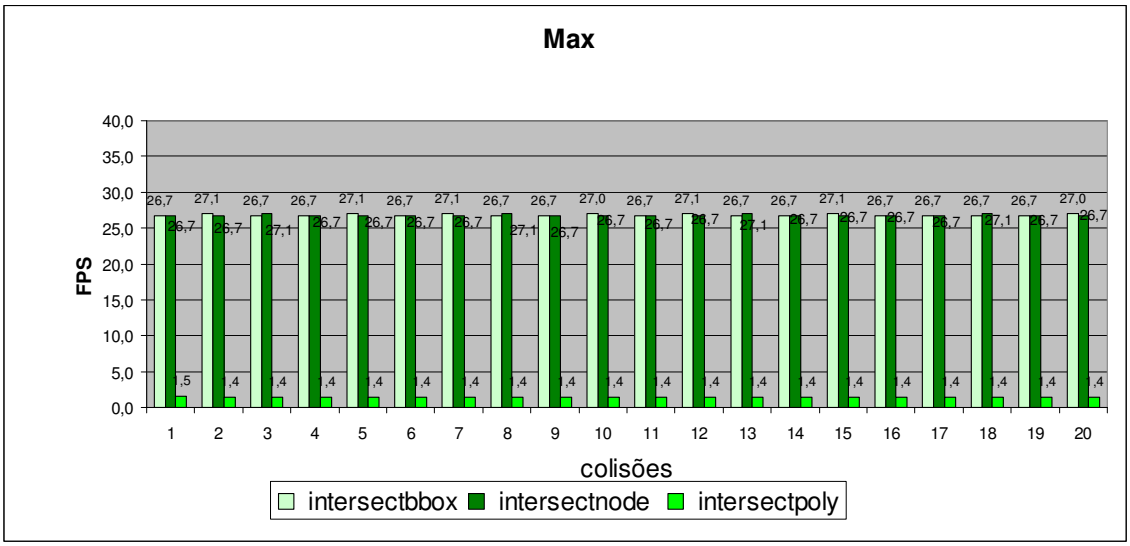

Figura 6.27 – FPS com 20 colisões entre as três técnicas do protótipo Max

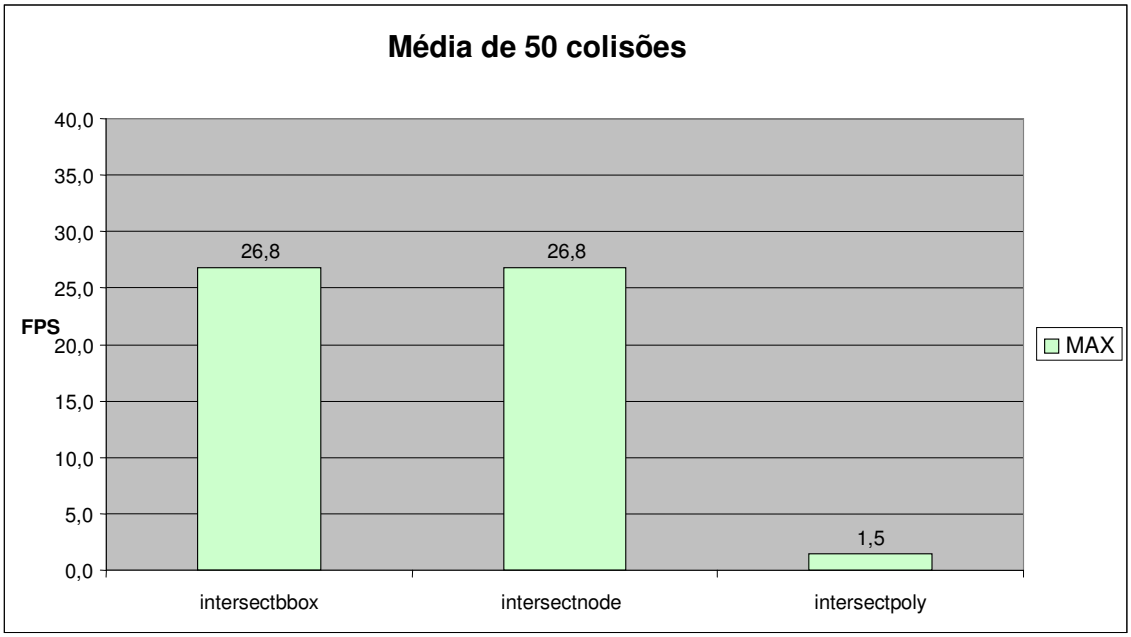

Figura 6.28 – Média de 50 colisões com o protótipo Max

6.1.4 Comparações das médias dos FPS dos Protótipos andando e colidindo.

O resultado observado na Figura 6.30 demonstra que o Bonfa é mais rápido, seguido com uma diferença mínima o Mini e o mais demorado o Max.

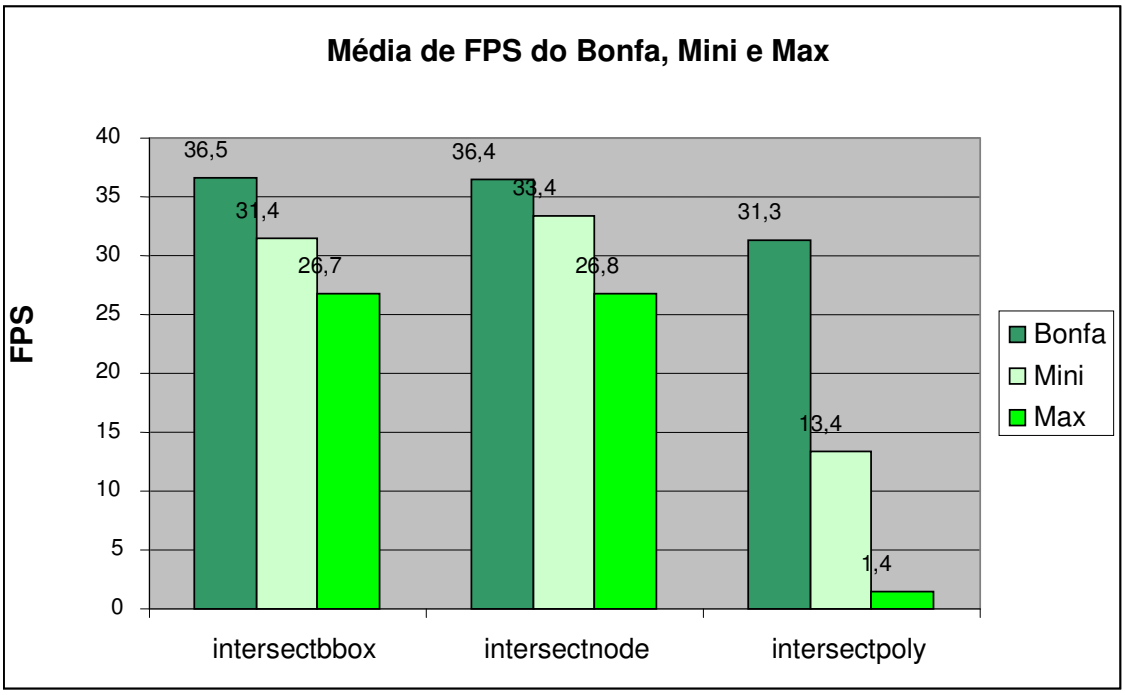

Figura 6.29 - Média geral dos FPS dos três avatares com 50 colisões

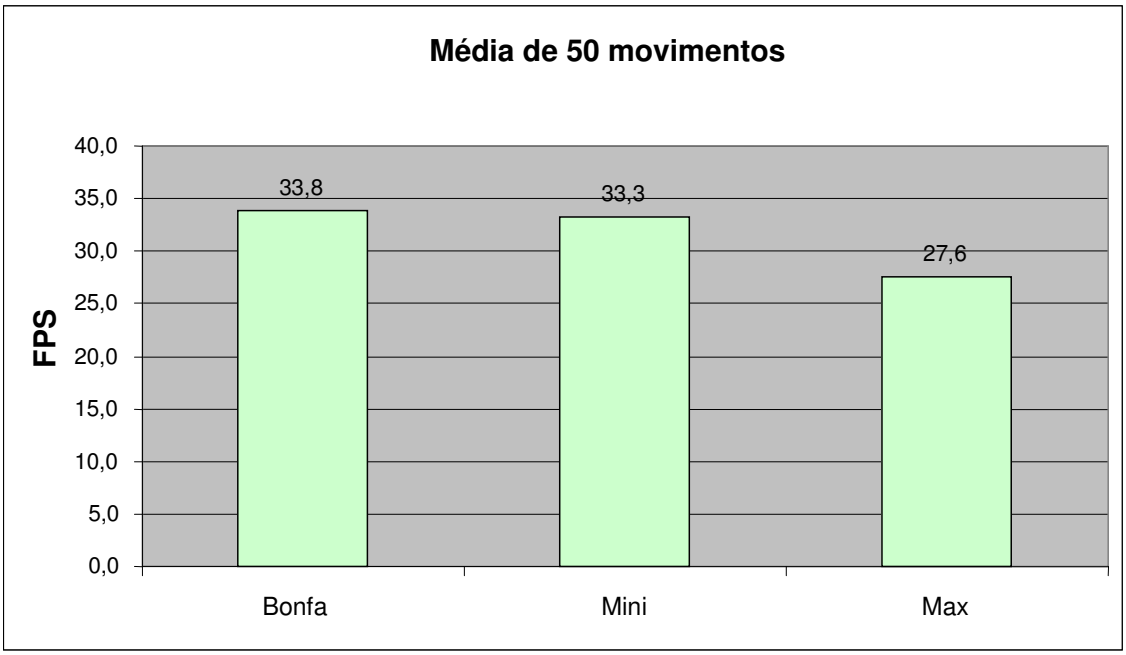

Figura 6.30 – Média geral dos FPS dos três avatares com 50 movimentações

Como visto no Capitulo 5, os avatares foram desenvolvidos utilizando softwares diversificados, onde cada um possui uma quantidade de polígonos (Bonfa com 772 , Mini com 1868 e o Max com 4294). Com isso surgiram as diferenças entre as médias das técnicas de Detecção de Colisão. O avatar Max possui maior quantidade de polígonos, sendo o mais demorado, seguindo do Mini e por último o Bonfa.

Ficou claro através das Figuras que a quantidade de FPS na técnica Intersectnode foi a mais rápida para o Max e o Mini, seguida da técnica Intersectbbox. Com o Bonfa a mais rápida foi a técnica Intersectbbox, seguida da técnica Intersectnode. A mais demorada foi a Intersectpoly. Esse resultado ocorreu igualmente para todos os avatares, diferenciando, portanto na quantidade de FPS. A Figura 6.29 apresenta a média geral das técnicas de Detecção de Colisão e a Figura 6.30 apresenta a média geral dos avatares em movimento.

As aplicações em RV possuem algumas regras para não perder o realismo. Neste contexto, 10 FPS para o avatar colidir é aceitável, mas o ideal seria 20 FPS. Através das Figuras 6.29 e 6.30 foi observado que os avatares em movimento estão com a quantidade de

FPS ideal. Aplicando as técnicas de detecção de colisão intersectbbox e intersectnode os avatares também estão com a quantidade de FPS ideal. O problema esta na técnica de intersectpoly, o Mini está aceitável, mas o Max está com a quantidade de FPS baixa demais.

CONCLUSÕES

A Realidade Virtual vem experimentando um desenvolvimento acelerado nos últimos anos, indicando perspectivas bastante promissora para diversos segmentos, encontrando grandes aplicações em projetos, jogos e simulações.

A interface com RV envolve um controle tridimensional altamente interativo de processos computacionais. O usuário entra no espaço virtual das aplicações e visualiza, manipula e explora os dados da aplicação em tempo real.

Para a realização de um Ambiente Virtual composto de objetos, agentes e avatares é de fundamental importância fazer a Detecção de Colisão entre os mesmo para tornar o ambiente real.

Para solucionar os problemas de colisão foram evidenciadas com esse trabalho algumas técnicas de Detecção de Colisão mais apropriadas para Avatares.

Para a comprovação desse propósito, foram efetuados estudos das técnicas, e elaborou-se uma metodologia para o desenvolvimento de aplicações, descrita no Capítulo 5. Com a finalidade de verificar as técnicas de Detecção de Colisão, foram criados três protótipos, onde cada um possui três aplicações, uma para cada técnica.

Os testes descritos no Capítulo 6 revelaram que, quando o avatar possui um envoltório em seu corpo, os FPS são mais rápido, comparado a técnica de intersecção de polígonos, indiferente da quantidade de polígonos utilizada para a modelagem do mesmo. O teste com o envoltório, verifica se algumas das faces em torno do objeto esbarraram com o envoltório de um outro objeto. O avatar é envolvido em uma caixa e o teste ocorre entre as caixas. O segundo método é o teste do polígono, que verifica cada polígono de um objeto de encontro a cada polígono de um outro objeto.

Cabe ainda ressaltar que o primeiro método é razoavelmente rápido, mas não é exato em relação ao avatar humano; o segundo método é muito exato, mas pode ser extremamente lento dependendo da aplicação.

Para este trabalho foi utilizado o Delphi para as telas iniciais, visto que não influenciou no tempo em que ocorreu a colisão.

O avatar pode colidir com o mesmo objeto várias vezes e obter resultados dos frames por segundos diferenciados. Esta taxa oscila em cada colisão, indiferente do método que esteja utilizando.

A utilização de mais de um avatar foi necessária para a verificação dos frames por segundo. Isso resultou nas diferenças entre as técnicas Intersectbbox, intersectnode e intersectpoly. As primeiras possuem diferenças mínimas, visto que ambas possuem envoltório. A técnica intersectpoly como verifica polígono a polígono é bem diferenciada, sendo mais demorada que as demais.

A seguir, são listados alguns itens que ao serem pesquisados poderão dar continuidade a esse trabalho:

- Teste dos FPS, levando em consideração a Detecção de Colisão de Avatar com Graus de liberdade de movimento para um corpo articulado.
- Aplicação da estratégia Octree.
- Mudança na ordem dos eventos do loop de simulação do WTK.
- Testes com técnicas de detecção de colisão hibridas.

BIBLIOGRAFIA

ARENDI, C. F.; DUARTE, T. P. Princípios de Anatomia Humana. São Paulo: CI, 1996 112p.

AVATAR. In Novo dicionário básico da língua portuguesa Folha/Aurélio. Rio de Janeiro, 1988, p 75.

BADLER, Norman; PHILLIPS, Cary B; WEBBER, Bonnie L. Simulating Humans: Computer graphics, animation, and control, 1999

BADLER, Norman Virtual Humans for Animation, Ergonomics, and Simulation, University of Pennsylvania, Philadelphia, 1997.

ÇAPIN, T. K.; PANDZIC, Igor S.; THALMANN, Nádia M.; THALMANN, Daniel. Avatars in Networked Virtual Environments. New York, John Wiley and Sons, 1999.

ÇAPIN, T. K; ESMERADO, Joaquim; THALMANN, Daniel. A Dead-Reckoning Technique for Streaming Virtual Human Animation disponível em http://ligwww.epfl.ch/~thalmann/papers.dir/ieeetr_csvt.PDF. Acesso em 10 de fevereiro de 2005

KLIEMANN, Giovani, Equipe Engine GameBase. Detecção de Colisão disponível em http://www.gamebase.hpg.com.br/, 2002. Acesso em outubro de 2004.

GOLIN, Geisa; BUÔGO Mariana. Atores Virtuais e Avatares disponível em www.lrv.ufsc.br/ddrv/2004.2/Artigos.html. Acesso em 2004

H-ANIM. Humanoid Animation Working Group. Web3d Consortium, 2003 disponível em http://www.h-anim.org/. Acesso em 29 de outubro de 2004

KIRNER, Cláudio. Sistemas de Realidade Virtual disponível em http://www.dc.ufscar.br/~grv/tutrv/tutrv.htm#sumario4.3. Acesso em 2004

LEITE, Antônio José Melo Junior, Ataxia: uma arquitetura para a viabilização de NET-VE'S voltados para a educação a distância através da internet, dissertação apresentada ao mestrado de Ciência da Computação da Universidade Federal do Ceará ,2000.

MACIEL, Anderson, Modelagem de Articulações para Humanos Virtuais Baseada em Anatomia, dissertação para o mestrado da Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.

NAMCO – Jogo do Pac Man disponível em http://pt.wikipedia.org/wiki/Pac-Man. Acesso em outubro de 2006

NASCIMENTO, Haroldo P.; Pellegrino, Sérgio R. M. – Detecção de Colisões entre objetos convexos com características geométricas mutantes – Proceedings of SRV 2003 – Symposium on Virtual Reality – Ribeirão Preto - SP-2003.

NEDEL, Luciana P. Animação de personagens para uso de jogos disponível em http://www.inf.ufrgs.br/%7Enedel/talks/character-animation.pdf. Acesso em novembro de 2004.

NEDEL, Luciana P. Animação por Computador: Evolução e Tendências disponível em www.inf.ufrgs.br/cg/publications/ n**ed**el ,2000. Acesso em setembro de 2004

NETTO, Antonio Valério; MACHADO, Liliane dos Santos; OLIVEIRA, Maria Cristina F. Realidade Virtual Fundamentos e Aplicações - Visual Books – 2002.

PERUZZA, Ana Paula P. M; Kirner, Cláudio; Nakamura, Eliana Y. - Solução do Problema da Detecção de Colisão em Ambientes de Realidade Virtual, São Carlos – SP, Anais do 1º Workshop de Realidade Virtual - WRV'97, São Carlos - SP, 1997.

PERUZZA, Ana Paula P. M, ZUFFO, Marcelo Knörich, Análise da Contribuição da Realidade Virtual para a Educação através do Sistema ConstruiRV , São Paulo VII Symposium on Virtual Reality, 2004.

PINHO, Márcio Serolli; Interação em Ambientes Tridimensionais disponível em http://www.inf.pucrs.br/~pinho/3DInteraction/#_Toc493643480. Acesso em 25 de agosto de 2004

PINHO, Márcio Serolli - Manipulação Simultânea de Objetos em Ambientes Virtuais Imersivos disponível em http://www.inf.ufrgs.br/cg/publications/pinho/tese-inf-0404102.pdf Acesso em setembro de 2004.

PINHO, Márcio Serolli; KIRNER, Cláudio – Uma introdução à realidade virtual, 1997 disponível em http://grv.inf.pucrs.br/Pagina/TutRV/tutrv.htm Acesso em setembro de 2004

SABINO, Vanessa Cristina, Game API Simplicidade e poder nos jogos para celulares, 2004, disponível em http://www.linuxchix.org.br/artigos/GameAPI-unificado.pdf acesso em 28 de setembro de 2004.

SINGHAL, Sandeep e ZYDA, Michael. Networked Virtual Environments: Design and Implementation, (Siggraph Series), ACM Press, New York, 1999.

STEIGLEDER, Mauro; LASCHUK, Anatólio. Detecção de Colisões Entre Sistemas de Partículas e Objetos Sólidos Através de Grades Pentadimensionais, 1997 disponível em http://mirror.impa.br/sibgrapi97/anais/ART42/ acesso em 22 de novembro de 2004.

TEICHRIEB, Verônica. Avatares como Guias Interativos para Auxílio na Navegação em Ambientes Virtuais Tridimensionais, 1999 disponível em http://www.cin.ufpe.br/~if124/dissertacoes/VeronicaT/ Acesso em 30 de dezembro de 2005

THE SIMS, Jogo disponível em http://www.thesims.com/. Acesso em: 29 de setembro de 2004

VENTRELLA, JJ. Avatar Physics and Genetics**,** Virtual Worlds (ed. Heudin, J.C.), Springer-Verlag Berlin/Heidelberg.