Pontifícia Universidade Católica do Rio de Janeiro
Departamento de Informática

DISSERTAÇÃO DE MESTRADO

Tratamento Eficiente de Visibilidade Através de Árvores de Volumes Envolventes

Aluno: Mauricio Hofmam da Silva
hofmam@tecgraf.puc-rio.br

Orientadores: Prof. Marcelo Gattass gattass@tecgraf.puc-rio.br
Prof. Waldemar Celes celes@tecgraf.puc-rio.br

Rio de Janeiro, Fevereiro de 2002

Agradecimentos

A Marcelo Gattass, orientador, pelo invulgar empenho que dedicou a esta empreitada, pelos conselhos oportunos em momentos decisivos da consecução da pesquisa e pela confiança que depositou em mim desde o início dos trabalhos.

A Waldemar Celes, orientador, pelo apoio e colaboração durante toda a pesquisa e em especial pelas sugestões perspicazes em momentos críticos da implementação.

A meus pais, por tudo que sou hoje.

A minha esposa, pelo seu amor e incentivo, e pela abnegação com que abdicou de muitas comodidades e assumiu tarefas que me cabiam.

Ao meu filho, que, apesar da tenra idade, soube compreender que teria de abrir mão da minha companhia em vários momentos.

A Rodrigo Toledo, pelas sugestões oportunas.

A Paula Friederich, pelo auxílio com as estruturas de dados espaciais.

A Eduardo Gaspar, pelo auxílio com os modelos tridimensionais.

A Ricardo Igreja, pelo auxílio com os gráficos de desempenho.

Ao TecGraf e a todos os seus integrantes, por terem proporcionado não só um ambiente propício para a pesquisa mas também um ambiente de camaradagem.

Aos componentes da banca, por suas sugestões.

Ao Exército Brasileiro e ao Instituto Militar de Engenharia (IME), por terem proporcionado a oportunidade e os meios para realizar esta pesquisa.

Resumo

A restituição de modelos tridimensionais complexos de engenharia tem sido um desafio para a computação gráfica desde seus primórdios, pois modelos detalhados são freqüentemente compostos de milhões de polígonos, enquanto as estações gráficas atuais são capazes de exibir, em taxas interativas, apenas algo da ordem de dezenas ou centenas de milhares de polígonos. Uma das formas de melhorar o desempenho de visualizadores de modelos tridimensionais é reduzir o número de polígonos passados para a cadeia de restituição, eliminando grandes grupos de polígonos determinados como não visíveis por estarem fora do volume de visão ou escondidos por outros polígonos. Neste trabalho, realizamos um estudo do uso de volumes envolventes para determinar os conjuntos de polígonos que são potencialmente visíveis, propomos uma forma de estruturar esses polígonos numa hierarquia de forma a diminuir os cálculos necessários para esse fim e compilamos uma série de resultados que permitem nortear o uso de volumes envolventes e a estruturação de modelos.

Abstract

Rendering complex three-dimensional Engineering models has been a challenge for Computer Graphics ever since its origin, as detailed models are often composed of millions of polygons while current graphic stations are able to display, at interactive rates, only dozens or hundreds of thousands of polygons. A way to increase the performance of viewers of three-dimensional models is to reduce the number of polygons passed to the rendering pipeline by eliminating large groups of polygons classified as non-visible for being out of the viewing frustum or hidden by other polygons. In this work, we study the use of bounding volumes to determine sets of polygons which are potentially visible, propose a way to structure such polygons in a hierarchy so as to restrict the necessary computations for this purpose, and compile a series of results which allow us to take some conclusions on the use of bounding volumes and model structuring.

Índice

Índice de Figuras

Índice de Tabelas

Índice de Gráficos

Glossário

Culling - descarte contra o volume de visão

Oclusion culling - descarte contra oclusores

Rasterization - rastreio

Raytracing - traçado de raios

Rendering - restituição

Rendering engine - motor de restituição

Rendering pipeline - cadeia de restituição

Viewing frustum - volume de visão

1 Introdução

Uma das áreas de pesquisa mais ativas desde os primórdios da computação gráfica é a de visualização interativa de modelos complexos. Tais modelos são produzidos e utilizados na visualização de maquetes arquitetônicas, estruturas mecânicas, simuladores de vôo, planejamentos urbanísticos, etc. Muitas vezes, a complexidade do modelo vai de encontro à capacidade do equipamento de realizar a sua visualização a uma taxa de quadros por segundo realmente interativa. Enquanto muitos modelos chegam a ter malhas compostas por milhões de triângulos, os melhores computadores pessoais conseguem exibir, a uma taxa interativa, modelos de cerca de 40 mil triângulos, e estações gráficas de médio porte cerca de 300 mil triângulos. Portanto, faz-se necessário adotar técnicas para reduzir o número de primitivas (polígonos, superfícies curvas, etc.) submetidas à cadeia de restituição (*rendering pipeline*) para cada quadro. Essa redução, contudo, não deve introduzir muitos defeitos, ou sequer defeito algum. Simplificação adaptativa do modelo, descarte de objetos não visíveis e uso de impostores estão entre as técnicas de redução de primitivas mais usadas. Muitas vezes, essas técnicas são usadas concorrentemente para diminuir o número de primitivas do modelo, de modo que o equipamento possa manipulá-lo em taxas interativas.

Objetos detectados como não visíveis não precisam ser restituídos, evitando cálculos desnecessários de transformação de vértices, transformação ou cálculo de normais, iluminação, aplicação de textura, projeção e rastreio (*rasterization*) e, assim, aumentando o desempenho global da aplicação. Um objeto é determinado como não visível se estiver fora do volume de visão ou se estiver escondido por outro objeto. O descarte de objetos escondidos tem sido usado com sucesso em ambientes virtuais especialmente estruturados, como o interior de edificações; contudo, a maioria das aplicações típicas de engenharia utiliza modelos tridimensionais complexos genéricos, como maquinarias industriais, navios e instalações petrolíferas, que não obedecem a um tipo simples de organização.

Assim sendo, podemos condensar os seguintes requisitos desejáveis para uma técnica de descarte [ZHAN1998]:

- Generalidade - Poder ser aplicada a qualquer tipo de modelo, mesmo criado sem ter em mente a sua visualização interativa ou o uso de uma técnica específica de aceleração.
- Portabilidade - A técnica de remoção não deve ser dependente de algum equipamento ou plataforma específico.
- Melhora significativa do desempenho da aplicação - O custo de verificar se um objeto deve ser desenhado ou não deve ser significativamente menor do que o custo de restituí-lo.

1.2 Objetivos

Determinar se uma primitiva deve ser ou não descartada pode ser mais custoso computacionalmente do que restituí-la. Assim, para decidir se um objeto deve ou não ser descartado, não se pode testar cada parte de sua superfície contra o volume de visão ou superfícies oclusoras. Para efetuar esses testes, são comumente utilizados volumes envolventes mínimos do objeto. Esses volumes são normalmente muito mais simples de serem testados do que o objeto a que se referem, representando um ganho elevado de desempenho. Os tipos de volumes envolventes mais comuns são esfera, cilindro, cápsula, caixa orientada, caixa alinhada com os eixos do sistema de coordenadas global, pastilha e elipsóide. O uso de volumes envolventes para realizar testes de descarte vem sendo feito há vários anos, em aplicações não só de visualização interativa como também de geração de imagens foto-realistas, como, por exemplo, programas de traçado de raios (*raytracing*).

Contudo, esses volumes nem sempre se adaptam bem ao objeto, acabando por determinar que um objeto totalmente escondido ou fora do volume de visão seja enviado para a cadeia de restituição. Um dos volumes mais usados é a esfera, pois seu teste é o mais simples e rápido. Contudo, certos tipo de objetos, como uma haste, por exemplo, têm esferas envolventes com um grande espaço vazio entre a superfície do objeto e a superfície da esfera, elevando a probabilidade de que o objeto seja determinado como potencialmente visível quando na verdade não é. De modo geral, há um tipo de volume que se adapta melhor a cada objeto original. Por outro lado, testar cada objeto da cena pode ainda ser uma tarefa custosa. O número de testes pode ser reduzido agrupando-se objetos e calculando-se o volume envolvente do grupo. Se

o grupo não passar no teste de visibilidade (isto é, se não for visível), todo o grupo é descartado e nenhum outro teste é feito. Se o grupo for determinado como visível, todo o grupo é aceito e também nenhum outro teste é feito. Finalmente, se o grupo não for nem rejeitado nem totalmente aceito, então seus elementos passam a ser individualmente testados. Esta abordagem sugere que se criem grupos de grupos e assim por diante, gerando uma estrutura hierárquica melhor representada por uma árvore de volumes envolventes ajustados, tendo na raiz a cena completa. Uma vez montada a árvore de volumes da cena, devem ser realizados essencialmente dois tipos de testes de visibilidade: contra o volume de visão e contra os oclusores já levantados da cena, que também devem ter passado pelo teste contra o volume de visão.

Transformar uma série de objetos em uma árvore não é uma tarefa simples. Em primeiro lugar, dispomos de vários volumes envolventes para escolher. Os mais complexos se ajustam melhor aos objetos, porém requerem mais cálculos para realizar os testes de visibilidade. Em segundo lugar, a quantidade de árvores possíveis formadas por esses nós tem um grau de complexidade extremamente elevado no número de entradas, e apenas algoritmos heurísticos podem ser usados para buscar a melhor solução. Além disso, em modelos de equipamentos de engenharia, como um robô e seu braço, por exemplo, há algumas hierarquias estruturais que devem ser respeitadas.

Esta dissertação apresenta um estudo sobre os volumes envolventes comumente usados e propõe estratégias para organizar automaticamente objetos de estruturas de engenharia complexas com o objetivo de proporcionar um tipo de visualização caracterizada pelo caminhamento do observador através do modelo (*walk through*).

1.3 Trabalhos Relacionados

Há essencialmente três tipos de trabalhos relacionados com esta dissertação: os que tratam de algoritmos para cálculo de volumes envolventes, de técnicas para organização e indexação espacial e de métodos para tratamento de visibilidade. Deste último tipo daremos uma visão mais detalhada.

1.3.1 Volumes Envolventes

Eberly [EBER1999] compila formas de calcular os principais volumes envolventes a partir dos vértices de malhas de polígonos técnicas que obtêm volumes bem próximos dos mínimos. Welzl [WELZ1985] propõe formas de calcular esferas e elipsóides mínimos. Schömer, Sellen e Teichmann [SCHÜ1999] e O'Rourke [OROU1985] propõem algoritmos para calcular cilindros e caixas orientadas mínimas, respectivamente.

1.3.2 Estruturação de Dados Espaciais

Várias técnicas têm sido propostas para fazer a estruturação de dados espaciais baseada em árvores. O seu principal objetivo tem sido a indexação espacial com a finalidade de acelerar pesquisas em bancos de dados tais como os usados em sistemas de informações geográficas e sistemas de CAD (*computer aided design*). São exemplos dessas estruturas: árvore-kd [BENT1975, BENT1979a], árvore-BSP [FUCH1980], árvore-k-d-b [ROBI1981], árvore-R [GUTT1984], árvore-R compacta [ROUS1984], árvore-skd [OOI1987], árvore-R+ [SELL1987] e árvore-R* [BECK1990], dentre outras. Uma taxionomia abrangente da evolução da indexação espacial pode ser encontrada em [OOI1992]. Uma compilação de métodos de acesso a dados espaciais é apresentada em [GÜNT1995,GÜNT1998].

1.3.3 Determinação de Visibilidade

Os algoritmos de determinação de visibilidade se enquadram em duas categorias principais:

- algoritmos de remoção de superfícies ocultas, para calcular as partes visíveis de um conjunto de superfícies;
- algoritmos de descarte, que detectam e descartam partes não visíveis de um modelo em grandes grupos para acelerar a restituição.

Dentro da primeira categoria, enquadram-se algoritmos que descobrem quais superfícies ou partes de superfícies são visíveis para o observador. Entre eles incluem-se a determinação de linhas visíveis [ROBE1963, APPE1967], o algoritmo de *z-buffer*

[CATM1974], ordenação por profundidade [NEWE1972], algoritmos de linha de varredura [WYLI1967, BOUK1970a, BOUK1970b, WATK1970], algoritmos de subdivisão de área [WARN1969,WEIL1977] e traçado de raios [APPE1968].

A determinação de visibilidade está estreitamente relacionada com a ordenação de primitivas por profundidade ou distância em relação ao observador. Uma vez que as primitivas são ordenadas, elas podem ser restituídas do fundo para a frente. O algoritmo de partição binária do espaço (BSP) [FUCH1980] produz uma organização de polígonos da qual a sua ordenação por profundidade pode ser rapidamente obtida. Duas vulnerabilidades das árvores-BSP são que muitos polígonos têm de ser fracionados quando registrados na árvore e alterações em modelos dinâmicos continuam sendo um desafio para serem tratadas. O principal algoritmo de visibilidade atual é o *z-buffer*, devido a suas implantações em praticamente todas as placas gráficas. Nesta dissertação, consideramos que o computador a ser utilizado possui uma destas placas controlada pelo OpenGL [OPENGL].

Os algoritmos de descarte de objetos não visíveis não substituem algoritmos tradicionais de remoção de superfícies ocultas porque não resolvem a visibilidade no nível mais refinado (o nível de *pixels*). Os dois tipos de algoritmo trabalham em conjunto para restituir um quadro.

Clark [CLAR1976] propôs uma hierarquia de volumes envolventes estruturada em árvore para acelerar a restituição de modelos complexos, que é um dos focos desta dissertação. O conceito de células e portais foi introduzido em um algoritmo de remoção de superfícies ocultas apresentado por Jones [JONE1971] que é extensivamente usado na visualização de interiores de modelos arquitetônicos.

Gottschalk, Lin e Manocha [GOTT1996] apresentam um trabalho sobre detecção de colisões empregando árvores de caixas envolventes orientadas. Para verificar se dois objetos estão causando interferência mútua, seus vértices são sucessivamente compartimentados em uma árvore binária de caixas envolventes orientadas até que as folhas da árvore comportem triângulos individuais. O teste de colisão começa entre as raízes das árvores e o caminhamento prossegue até que a colisão seja rejeitada ou confirmada. Este trabalho é trivialmente estendido para o cálculo de visibilidade se um dos objetos tomados para o teste de colisão for o volume de visão ou o perímetro de um espaço determinado como zona de oclusão.

1.4 Descrição da Dissertação

Esta dissertação apresenta um estudo sobre volumes envolventes e organização de objetos de uma cena industrial complexa com vistas a restituição em tempo real numa trajetória através do modelo.

O Capítulo 2 descreve como o modelo de engenharia, junto com outros tipos de componentes, formam uma cena virtual. O Capítulo 3 trata das formas de se estruturar modelos e cenas visando a sua visualização interativa. O Capítulo 4 lista os principais volumes envolventes e formas de calculá-los. O Capítulo 5 traz algoritmos para verificar se os volumes citados no capítulo anterior interceptam o volume de visão. O Capítulo 6 apresenta as propostas da dissertação. O Capítulo 7 descreve a implementação do programa de testes, os modelos testados e compila os resultados dos testes realizados. Finalmente, o Capítulo 8 conclui sobre os resultados obtidos e sugere trabalhos futuros.

2 Descrição da Cena

A proposta deste estudo é apresentar uma técnica para melhorar o desempenho de aplicações dedicadas à visualização de modelos de engenharia como componentes de cenas virtuais. Portanto, convém neste momento definir claramente uma cena virtual.

Uma cena virtual é um teatro no qual determinados aspectos do mundo real são simulados. Atualmente, os mais diversos tipos de cenários são simulados, apresentando métodos diferentes de prospecção e análise dos dados que representam a simulação. Em alguns deles, é necessário que uma representação visual gráfica seja apresentada ao usuário como resultado da simulação. Encontram-se nesta categoria as cenas virtuais de interesse deste estudo.

Denominam-se *entidades* todos os atores existentes neste teatro virtual. Uma entidade é, portanto, a representação de algum elemento real que esteja sendo simulado. Os principais tipos de entidades que compõem uma cena são os objetos geométricos, luzes, câmeras, auxiliares, componentes mecânicos e sistemas de partículas.

As entidades normalmente não ficam independentemente dispersas na cena, mas organizadas de forma a constituir um *grafo de cena.* Este pode ser montado de mais de uma forma e será estudado detalhadamente no Capítulo 3.

Uma entidade é descrita pelos seus atributos, que podem ser os mais diversos e estar todos sendo avaliados na simulação. Os atributos de maior interesse para este trabalho são aqueles relacionados com o resultado visual a ser apresentado ao usuário. Contudo, nem todos os atributos relacionados com a visualização podem ser contemplados, ou pelo menos não sem uma preparação, pois alguns deles requerem um tratamento cujo custo computacional não permite o seu processamento em tempo real, como, por exemplo, o índice de refração de um objeto transparente ou a radiosidade de um ambiente.

A maioria dos atributos pode ser agrupado em uma destas três categorias: geometria, aparência e comportamento. Um avião, por exemplo, pode ter a sua geometria definida por uma malha de triângulos, sua aparência pela cor ou textura de

suas superfícies e seu comportamento por um plano de vôo que inclui coordenadas e horários em que essas coordenadas devem ser atingidas. Além das entidades, a descrição da cena abrange outros atributos que compõem a caracterização do ambiente virtual.

2.1 Descrição Geométrica

A descrição geométrica é uma maneira de definir como a entidade deve ser representada espacialmente de forma a permitir um processamento que resulte numa imagem da entidade. Existem muitas formas de se fazer essa representação. As mais comuns são através de equações implícitas ou paramétricas de superfícies, conjuntos de polígonos, superfícies de revolução ou mesmo combinações dessas formas. O exemplo mais elementar é um simples ponto.

Essas descrições podem ser ainda estáticas ou animadas, conforme a entidade tenha ou não a sua geometria modificada ao longo do tempo. As representações animadas podem ser descritas como funções do tempo ou como resultado da ação de outras entidades, aplicação de forças ou choques.

O grau de complexidade da geometria de uma entidade é diretamente proporcional à qualidade visual. Contudo, quanto mais detalhes existirem, mais tempo será necessário para restituir a entidade e, como neste trabalho estamos interessados em visualização interativa, a complexidade da cena é uma questão crucial. Não precisamos, entretanto, recorrer ao aumento de complexidade do modelo para melhorar a sua qualidade visual. O uso de texturas é um excelente recurso para isso. Podemos, por exemplo, substituir a geometria dos vincos de uma chapa metálica por um conjunto de texturas que criem o mesmo efeito visual e que podem ser obtidas a partir de desenhos, fotografias de chapas metálicas ou da restituição de um modelo que possua a geometria dos vincos.

A definição da descrição geométrica mais adequada é um problema complexo e foco de extensos estudos, muitas vezes tratada em conjunto com o uso de texturas, sendo que uma entidade muitas vezes dispõe de mais de uma descrição. Pode-se ter, por exemplo, uma entidade definida por um conjunto de equações paramétricas e manipulada desta forma durante toda a simulação, sendo que, no momento da

restituição, essas equações são amostradas de forma a se obter uma malha de polígonos. Dependendo de como a entidade é visualizada ou da capacidade computacional, mais ou menos amostras são calculadas, de forma a se obter uma malha mais ou menos complexa. Esta solução, baseada em múltiplos níveis de detalhes (LOD - *Level of Detail*), pode também ser tratada diretamente a partir da malha mais refinada de uma entidade para se obter representações mais grosseiras. Nos testes apresentados no Capítulo 6, este recurso não foi usado para que não mascarasse os resultados.

2.2 Aparência

O conjunto de atributos que compõem a aparência material da entidade estão, assim como na descrição geométrica, diretamente relacionados com o resultado visual da restituição. A aparência pode ser dada por um conjunto cada vez mais rico de atributos, podendo-se classificá-la nos seguintes níveis:

- Aparência nula. Usada em entidades presentes na cena mas que não se quer visualizar. Pode ser o caso de auxiliares, luzes, câmeras e componentes mecânicos;
- Cores opacas sem cálculo de iluminação, muitas vezes usadas na visualização volumétrica;
- Cores opacas com cálculo de iluminação, em função das componentes de reflexão difusa, especular e ambiente e atributos como reflexividade, nível de especularidade, suavidade, opacidade e radiância, usando, por exemplo, os modelos de iluminação Phong, Strauss, Blinn ou Anisotrópico;
- Uso de diferentes texturas em substituição às componentes difusa especular e ambiente e dos atributos de opacidade, brilho, reflexão, refração, rugosidade (*bump*) e outros, podendo ser dadas por imagens estáticas, animadas ou procedimentais e uni, bi ou tridimensionais.

A aparência, apesar de ser uma característica fundamental das entidades, não tem maiores implicações sobre o tema tratado neste estudo.

2.3 Comportamento

O comportamento, por sua vez, é de grande importância para o estudo do descarte de entidades não visíveis assim como sua geometria. As entidades podem ser totalmente estáticas ou possuírem algum atributo que varie no tempo. Deste modo, o comportamento de uma entidade pode ser definido como o conjunto de mudanças que qualquer um de seus atributos possa sofrer no decorrer do tempo. O contexto do comportamento não está relacionado apenas com mudanças na aparência e na descrição geométrica, mas principalmente com atributos como localização, velocidade e aceleração da entidade no espaço definido pelo ambiente virtual. É, portanto, através do controle do comportamento das entidades de um ambiente virtual que se torna possível a criação de um ambiente virtual animado.

O comportamento pode ser classificado em duas categorias quanto à sua previsibilidade: determinístico e não-determinístico. Os comportamentos determinísticos são todos aqueles cujo estado pode ser definido como uma função do tempo. Eles permitem que se saiba exatamente qual o estado da entidade em cada momento conforme se navegue para a frente ou para trás no tempo. Em contrapartida, o comportamento não-determinístico é imprevisível, sendo geralmente reflexo de ações que não seguem um padrão definido, como por exemplo certas ações humanas ou regidas por variáveis aleatórias. Neste caso, é impossível prever o estado de uma determinada entidade que apresente esse comportamento no futuro e demasiadamente complicado restaurar seu estado no passado.

2.4 Tipos de Entidades

2.4.1 Objetos Geométricos

Representam os objetos concretos do mundo real, como veículos, pessoas, instalações e maquinaria. Sempre têm a descrição geométrica definida, geralmente possuem uma aparência e eventualmente têm um comportamento. Neste estudo vamos nos concentrar em objetos geométricos definidos por malhas de triângulos, pois esse tipo de descrição é o que permite a restituição mais rápida e muitos aplicativos que manipulam entidades com outros tipos de descrição discretizam seus modelos na forma de malhas de triângulos para fazer a sua restituição.

2.4.2 Luzes

O uso de uma iluminação corretamente modelada auxilia na geração de imagens com um maior grau de realismo quando a simulação envolve elementos normalmente presentes no mundo real. Considere, por exemplo, a importância da iluminação na simulação de um grupo de pessoas e objetos dispostos em uma sala de um ambiente doméstico em comparação com a simulação de reações químicas entre moléculas. A primeira tende a exigir um modelo de iluminação mais complexo, que reflita a realidade, enquanto a segunda requer um modelo de iluminação mais simples, uma vez que representa uma abstração científica para fins de visualização sem a necessidade de ser verossímil.

Para que sejam alcançados bons resultados visuais, a descrição da cena virtual deve comportar luzes com atributos que permitam ajustá-las de forma a conseguir reproduzir efeitos observados ao nosso redor. Em particular, as interfaces de programação gráfica de aplicações (API gráfica) mais comuns implementam o modelo de iluminação Gouraud [GOUR1971], que contém os seguintes tipos de fontes de luz:

2.4.2.1 Ambiente: componente de iluminação uniformemente aplicada a todas as primitivas da cena.

2.4.2.2 Onidirecional: fonte de luz pontual com campo de influência em todo o espaço ao redor de sua posição no ambiente virtual, com os raios de luz divergindo a partir da sua posição.

2.4.2.3 Direcional: fonte de luz com raios paralelos. Pode ser modelada como estando posicionada em um ponto no infinito emitindo os raios em uma determinada direção e tendo como campo de influência todo o espaço ou uma região limitada em torno de um eixo que define a direção dos raios.

2.4.2.4 Farolete: caso particular de uma fonte de luz onidirecional cujo campo de influência é limitado a um cone.

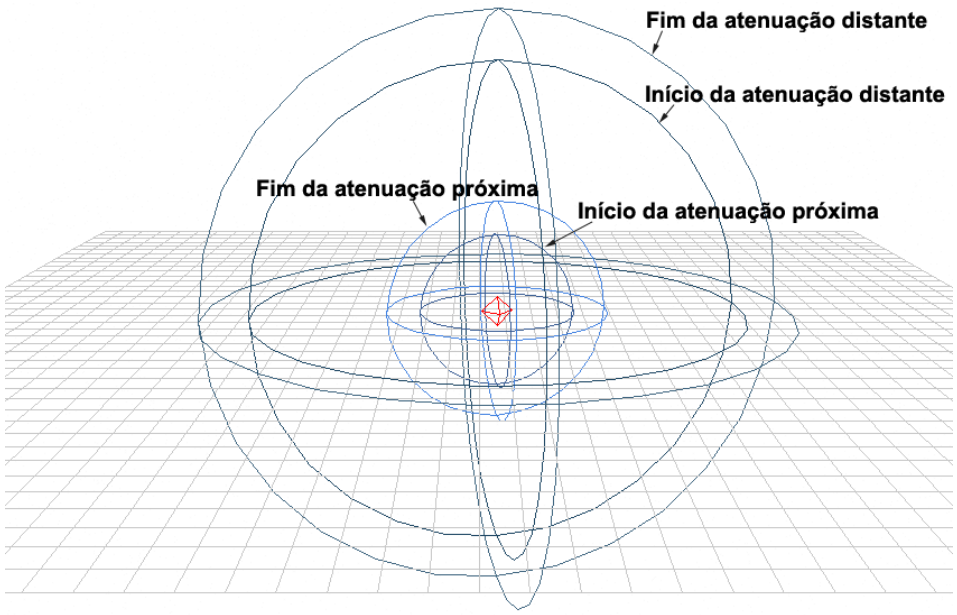

Figura 2.1 - Luz onidirecional

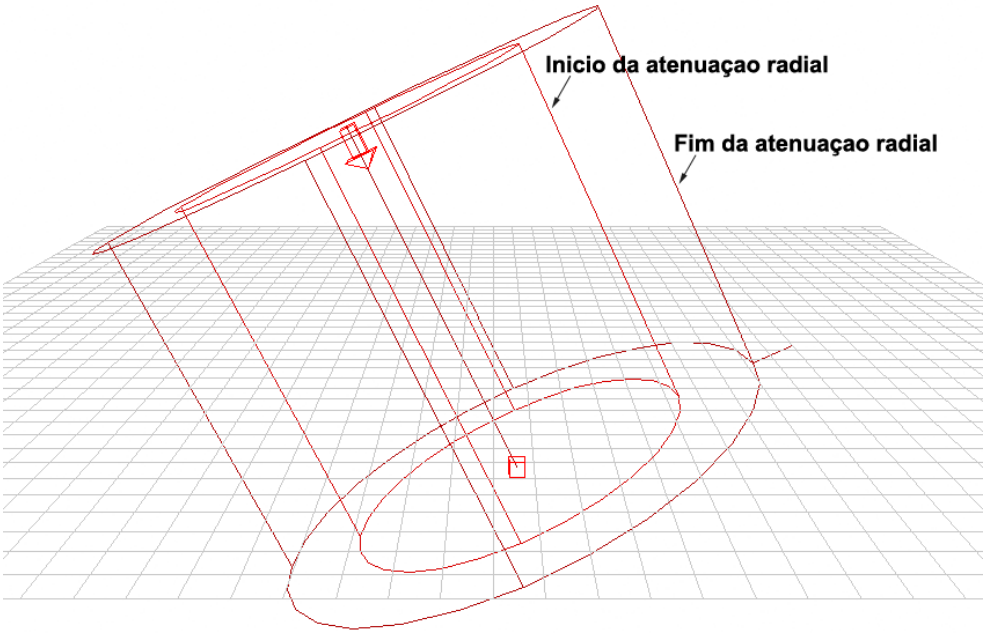

Figura 2.2 - Luz direcional

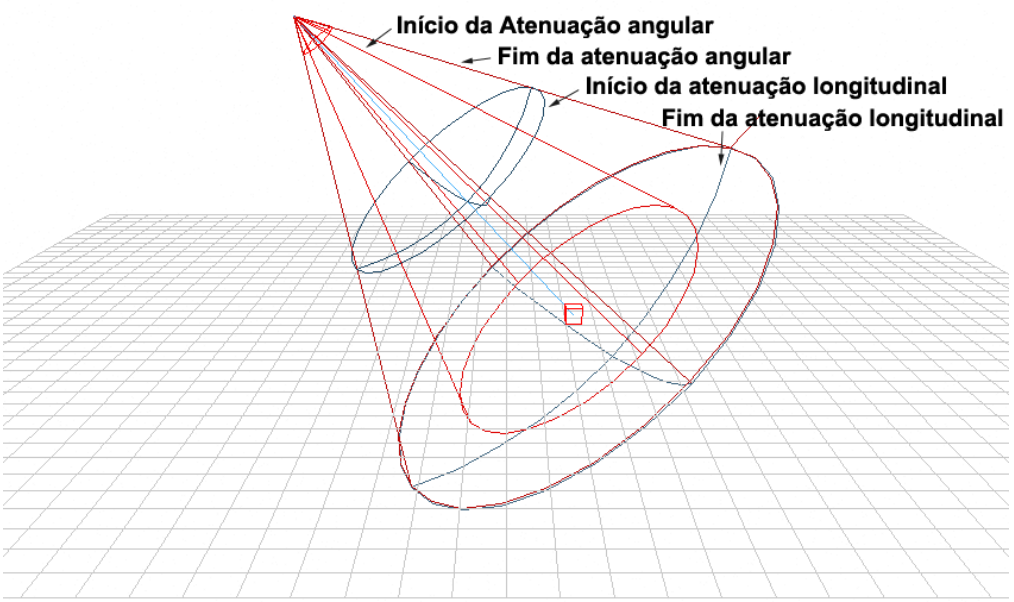

Figura 2.3 - Luz farolete

Dentre os atributos de cada tipo de fonte luminosa, podem ser destacados:

Intensidade da cor difusa: intensidade com que uma determinada fonte influencia a componente difusa da aparência de uma determinada primitiva.

Intensidade da cor especular: influência de uma determinada fonte na componente especular.

Intensidade da cor ambiente: influência de uma determinada fonte na componente ambiente.

Fator de decaimento da cor: modo como decai a influência de uma fonte de luz em função da distância. O decaimento pode ser *longitudinal*, de acordo com a distância do ponto considerado até a posição da luz; pode se *radial*, em função da distância do ponto ao eixo que define uma luz direcional; ou ainda *angular*, medido a partir do eixo do cone de iluminação de um farolete.

A descrição geométrica mínima de uma luz é a sua posição, que pode ser no infinito, no caso de uma luz direcional, ou ter uma descrição mais detalhada, no caso de uma visualização esquemática na qual a luz é representada por um aramado. O seu comportamento pode ser o de seguir outra entidade, como os faróis de um carro.

2.4.3 Câmeras

São entidades que reúnem atributos adicionais para modelar a visão de um espectador da cena virtual. Entre esses atributos, incluem-se a posição *de*; o vetor *para* - direção para onde o espectador está olhando; o vetor *para cima* - vetor que aponta no sentido do topo da câmera; o ângulo que define o campo de visão ou a distância focal; o aspecto ou razão entre altura e largura do campo visual; e o tipo de projeção - *ortográfica* ou *perspectiva*. Este conjunto de parâmetros é usado para alimentar a cadeia de restituição[1] de forma a obter uma imagem retangular da cena conforme vista por um espectador colocado na posição da câmera ou, resumidamente, uma fotografia da cena.

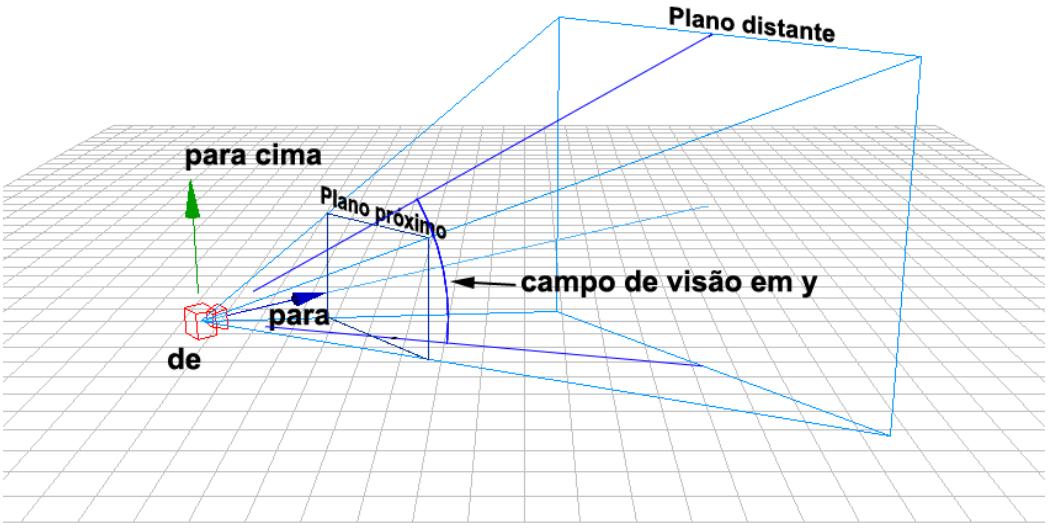

Figura 2.4 - Câmera

Assim como no caso da luz, a geometria da câmera pode se resumir a um ponto ou ser mais complexa, permitindo, por exemplo, realizar testes de colisão ou fazer a sua representação esquemática. Seu comportamento pode ser o de deslocar-se junto com alguma entidade, simulando, por exemplo, a visão do piloto de um avião, ou de observar uma entidade a partir de uma posição da cena.

[1] Veja a Figura 3.4 na Seção 3.4.

2.4.4 Auxiliares

São entidades geralmente utilizadas na modelagem da cena para parametrizar o comportamento de outras entidades. Possuem aparência nula, para que não sejam visíveis durante a simulação. Dentre elas, temos: *alvo* - entidade para a qual ficam apontadas luzes do tipo direcional ou farolete e câmeras; *simulacro* - ocupa uma posição e tem uma funcionalidade normalmente destinada a um objeto geométrico, mas reúne alguns atributos adicionais e não é visível na cena; *esqueleto* - usado para modelar o movimento de pessoas, animais ou objetos articulados; e *deformações espaciais* - para causar deformações em outras entidades.

2.4.5 Componentes Mecânicos

São utilizados para simular a cinemática e a dinâmica da cena. São forças, molas, amortecedores, atrito, vento, gravidade, etc. Possuem atributos como elasticidade, fator de dissipação de energia, coeficiente de atrito, viscosidade, etc.

2.4.6 Sistemas de Partículas

Simulam chuva, neve, poeira, fumaça, espuma, fogo, cachoeiras, névoa, faíscas, etc. As partículas possuem sempre uma geometria bastante simples, geralmente um triângulo, pois para se obter o efeito visual desejado é normalmente necessário ter milhares de partículas presentes na cena. As partículas emanam a partir de um emissor que pode ser um ponto ou um polígono, e se propagam no ambiente segundo uma função de dispersão determinística ou não. Podem ter um tempo de vida limitado, rebrotar após colisões, sofrer efeitos do vento ou gravidade e de deformações espaciais.

3 Organização de Cenas Industriais

As entidades que compõem cenas industriais podem ser organizadas de diversas formas, com base: (a) no projeto de engenharia, (b) nas transformações de instanciação, (c) nos seus atributos de aparência, (d) nas técnicas para a solução de questões de visibilidade, ou (e) numa combinação dessas associações. Essas diferentes organizações podem ser compatíveis ou conflitantes entre si e têm impacto direto no desempenho da visualização.

Normalmente, a relação dos objetos de uma cena de instalação industrial não se modifica, mesmo que os objetos se movam durante a simulação. Isto é importante, uma vez que a definição da melhor relação pode ser muito custosa e deve ser feita em tempo de preparação do modelo, isto é, em tempo de pré-processamento, antes da simulação em tempo real.

3.1 Organização Proveniente do Projeto de Engenharia

Muitas das simulações que envolvem a visualização de objetos tridimensionais são compostas de elementos como edifícios, veículos, máquinas, regiões de terreno, etc. Tais elementos, se estiverem organizados, seguirão uma montagem lógica do projeto de engenharia: engrenagens compondo uma máquina, máquinas e equipamentos compondo uma sala de máquinas, compartimentos formando um convés, conveses formando um navio e assim por diante. Daí nasce uma hierarquia bastante óbvia do ponto de vista funcional. É também comum que as atividades de um projeto complexo, como uma refinaria, por exemplo, sejam divididas entre diversos profissionais - uns definem as tubulações, outros os equipamentos, outros as estruturas, etc.

Essa hierarquia funcional é apropriada para dar suporte geométrico às simulações de engenharia, mas pode ou não satisfazer as necessidades de eficiência da visualização. Por exemplo: suponhamos que temos um navio cujo interior seja atravessado longitudinalmente por longas tubulações. É natural compor o sistema de tubulações por tubos íntegros conduzindo fluídos da sua origem ao destino. Porém, para atender a questões de desempenho, pode ser mais conveniente que a tubulação

seja secionada na altura das paredes divisórias dos compartimentos, de modo a tornar o processo de visualização mais eficiente através da remoção de seções ocultas. Ao passo que a subdivisão de entidades pode ser desejável em alguns casos, a sua aglutinação pode ser necessária em outros. Estes e outros tipos de tratamento se fazem necessários para viabilizar a aplicação de técnicas que permitam a visualização interativa do modelo bruto de engenharia.

3.2 Hierarquia de Instanciação

Um conjunto de objetos pode estar organizado de forma a constituir uma hierarquia através de suas relações de vínculo [GOME1998]. Os objetos com hierarquia podem ser classificados como *articulados* ou *compostos*, conforme haja ou não movimento relativo entre os seus sub-objetos - nos objetos compostos, os vínculos são rígidos e não há movimento relativo entre os sub-objetos e, nos objetos articulados, as suas partes rígidas são conectadas por juntas ou articulações que permitem a movimentação relativa entre elas.

Os vínculos, por sua vez, podem também ser classificados em *geométricos* e *físicos*. Os primeiros decorrem de uma relação geométrica entre dois objetos, como no caso da articulação de um braço de robô, enquanto os segundos se referem a campos de forças de atração ou repulsão aplicados a objetos geométricos, estabelecendo um vínculo dinâmico entre eles, como, por exemplo, um sistema planetário.

Podemos descrever um objeto articulado utilizando um referencial absoluto do espaço, em relação ao qual fornecemos de modo explícito a posição e orientação de cada uma das partes rígidas. Desta forma, é necessário calcular individualmente e de forma independente o posicionamento de cada parte. Essa independência pode dificultar a manutenção da integridade do conjunto, pois pequenas variações no cálculo da posição de uma articulação podem acabar por desconectar partes rígidas da estrutura.

Uma forma mais conveniente de resolver o posicionamento de um objeto articulado é estruturá-lo hierarquicamente e, em vez de usar um referencial único, associar um sistema local (ou relativo) de coordenadas a cada parte rígida do objeto. A orientação e posição de cada parte é dada em seu sistema local de coordenadas e o seu posicionamento global é obtido através da mudança recursiva de sistema de

coordenadas. Assim, temos que somente as articulações de nível hierárquico mais alto (ou a raiz da estrutura) são posicionadas em relação ao referencial global. As demais articulações são ajustadas através de transformações intrínsecas de mudanças de coordenadas. Desta forma, ao aplicarmos uma transformação a uma articulação, todas as articulações subordinadas a ela sofrerão uma transformação idêntica.

A organização hierárquica de um objeto articulado sugere a estrutura de uma árvore na qual os nós representam as partes rígidas e as arestas representam as relações de subordinação através das articulações. Assim, cada nó é posicionado segundo suas transformações particulares em relação ao seu sistema de coordenadas local e segundo as transformações dos nós ancestrais.

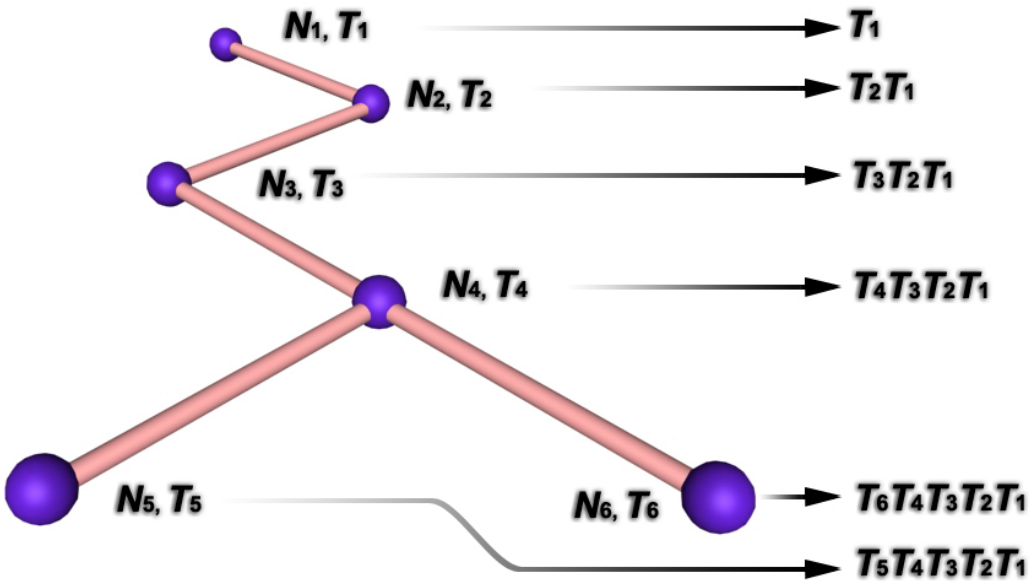

Figura 3.1 - Hierarquia de transformações do braço de robô da Figura 3.2

Para posicionar uma estrutura articulada, faz-se um caminhamento na árvore que descreve a sua topologia passando por todos os nós. Cada nó é posicionado concatenando-se a sua transformação particular com as dos seus ancestrais. Conforme ilustra a Figura 3.1, cada parte rígida possui uma transformação associada T_i, que realiza a mudança entre o seu sistema de coordenadas e o sistema de coordenadas do nó ao qual está subordinada. Assim, cada nó N_i é posicionado no sistema de coordenadas global fazendo o produto das transformações T_j correspondentes aos nós ascendentes até a raiz da árvore. O resultado da concatenação

das transformações para um determinado nó é chamado de *transformação corrente* do nó. Durante o processo de restituição, cada primitiva que descreve os objetos geométricos é desenhada de acordo com seu posicionamento em relação a um referencial absoluto. Isso é obtido aplicando-se a transformação corrente a cada uma das primitivas que compõem as partes rígidas de um objeto geométrico.

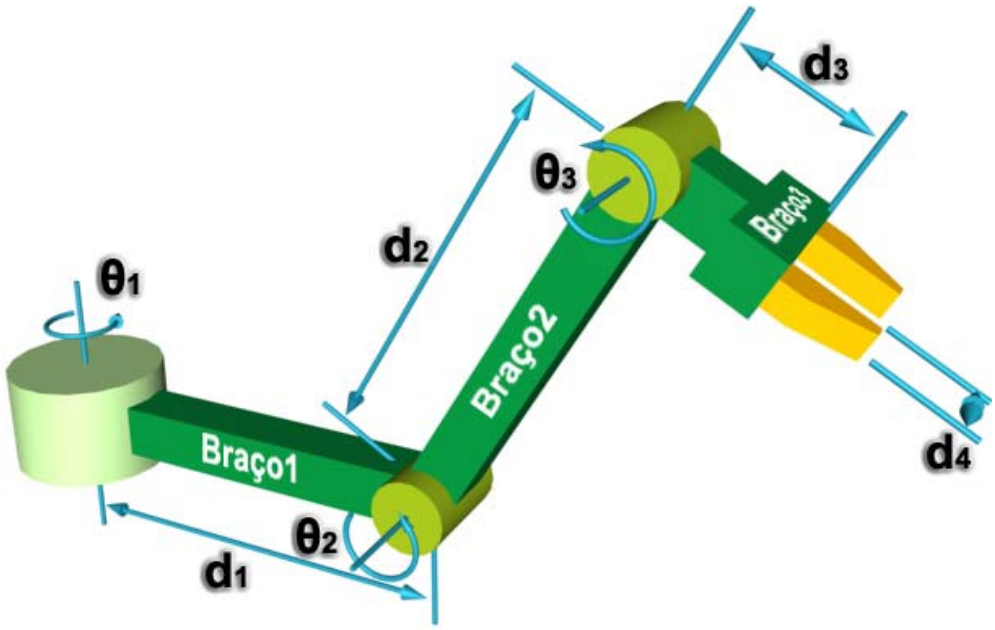

Figura 3.2 - Braço de robô articulado

A Figura 3.2 mostra um braço de robô esquemático que corresponde à hierarquia da Figura 3.1. A Figura 3.3 apresenta os sistemas de coordenadas locais de cada parte rígida. Tomamos como raiz da árvore o *Braço1* e supomos que o seu sistema de coordenadas local coincide com o sistema referencial global, a origem de $\mathbb{R}^3$ ou base canônica. A mudança da base canônica para o referencial do *Braço1* é dada por uma rotação de um ângulo θ_1 em torno do vetor z_1

$$T_1 = \begin{pmatrix} \cos\theta_1 & -\operatorname{sen}\theta_1 & 0 & 0 \\ \operatorname{sen}\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

A mudança do referencial do *Braço1* para o referencial do *Braço2* é feita por uma rotação de um ângulo θ_2 em torno do vetor x_1 seguida de uma translação de comprimento d_1 ao longo do novo eixo y_1. A matriz é dada por

$$T_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_2 & -\operatorname{sen}\theta_2 & 0 \\ 0 & \operatorname{sen}\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_2 & -\operatorname{sen}\theta_2 & d_1 \\ 0 & \operatorname{sen}\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

O posicionamento do braço é finalmente dado por T_2T_1. O posicionamento do *Braço3* e das duas garras pode ser obtido analogamente.

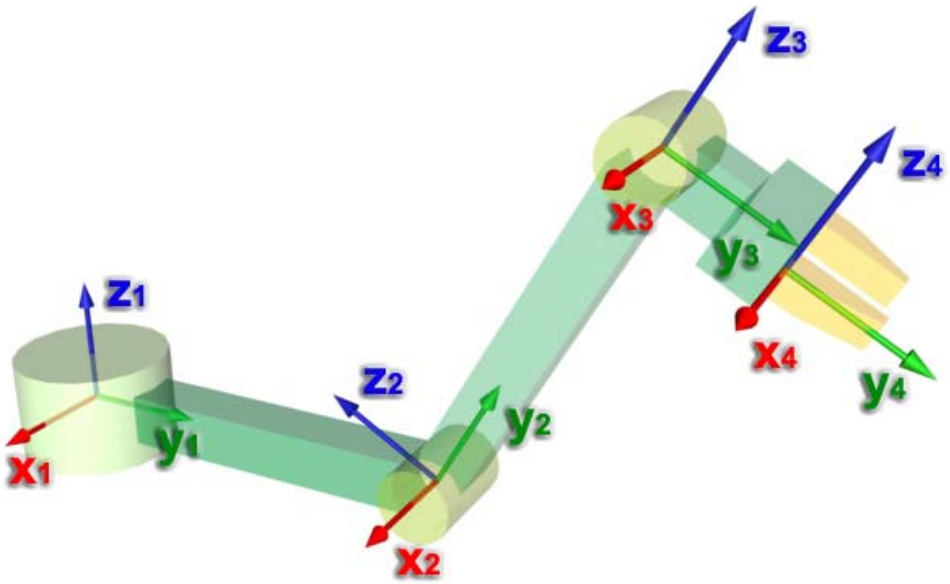

Figura 3.3 - Eixos locais do braço de robô da Figura 3.2

Embora esse processo, chamado de *cinemática direta*, possa deixar transparecer que as extremidades sempre são comandadas por seus ancestrais, isso nem sempre é o que se sucede. Na animação de personagens, por exemplo, a *cinemática inversa* [WATT1992] é empregada extensivamente para modelar o movimento dos membros do personagem. Na cinemática inversa, o animador controla as extremidades e as demais partes rígidas devem responder de acordo com as restrições impostas. Contudo, esta abordagem é baseada na mesma estrutura hierárquica da cinemática direta.

3.3 Aparência Influindo na Organização da Cena

Como vimos na Seção 3.2, as aparências aplicadas aos objetos geométricos podem ter um grau de complexidade bastante variável. Ao se usar aparências mais requintadas, particularmente as compostas por texturas, devemos procurar reduzir a troca de materiais enquanto os objetos são submetidos à cadeia de restituição, visto que as trocas de aparência demandam transferências de dados tanto mais demoradas quanto mais complexa for a aparência [WOOM1999]. Isso pode ser naturalmente obtido ordenando-se os objetos por tipo de aparência e submetendo-os nessa ordem à cadeia de restituição.

3.4 Hierarquia de Volumes Envolventes

A Figura 3.4 mostra uma possível seqüência de operações aplicada aos vértices de um modelo. Observe que somente no cerceamento é que as porções não visíveis do modelo são finalmente eliminadas. Até então, muito esforço foi gasto na transformação de vértices que acabaram não sendo visíveis na cena por estarem fora do volume de visão ou por terem sido encobertos por outros objetos. Várias técnicas têm sido propostas para determinar com antecedência quais objetos são potencialmente visíveis, de modo a minimizar o esforço despendido e aumentar o desempenho global da aplicação.

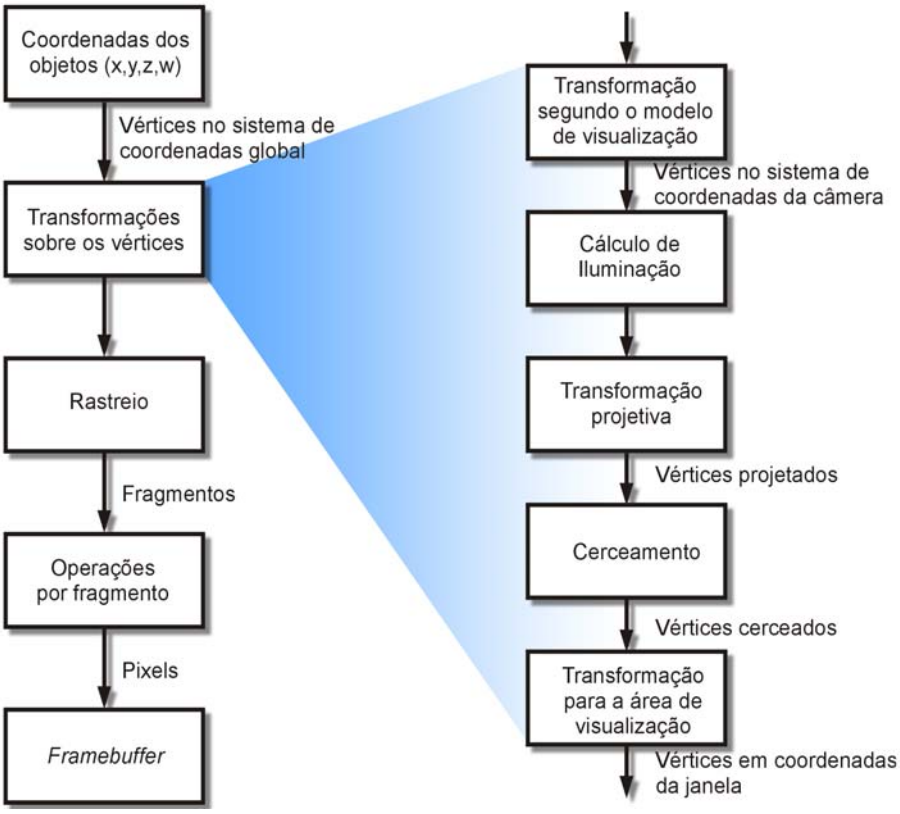

Figura 3.4 - Cadeia de restituição

Neste trabalho, estamos interessados em estudar as técnicas que são baseadas nos objetos envolventes do tipo esferas, caixas, cápsulas e pastilhas. Estas técnicas são bastante semelhantes ao uso da árvore-R e de suas variantes utilizadas na indexação espacial de objetos geográficos. Um estudo abrangente de técnicas de organização espacial de dados pode ser visto em [GÜNT1998].

Um volume envolvente pode encerrar um objeto individual, partições deste objeto ou partições do espaço que abrange vários objetos. Por isso, quando nos referirmos a partições, isso pode dizer respeito a uma região do espaço que contenha tanto objetos quanto partes destes.

A estrutura de dados que melhor representa a hierarquia de volumes envolventes é uma árvore. Num sentido amplo, temos uma árvore de regiões. Na sua raiz a cena toda e, nas folhas, objetos individuais ou partições deles. A construção dessa árvore pode ser manual, como visto na Seção 3.1, montada a partir do projeto de engenharia, ou automática, podendo empregar como restrição o próprio projeto de

engenharia. As técnicas para a montagem das árvores são agrupadas em duas categorias, de acordo com o sentido do processo:

- **De cima para baixo.** O cenário é recursivamente subdividido até que um limite seja atingido. Esse limite pode ser o tamanho da partição ou a quantidade de objetos ou partes de objetos contidos nela. Uma técnica que se enquadra nesta categoria é a árvore-kd.
- **De baixo para cima.** Os elementos atômicos da cena são sucessivamente agrupados até que se chegue a um único grupo abarcando possivelmente toda a cena. Aglomerados e, num sentido estrito, árvores-R encontram-se nesta categoria.

Naturalmente, uma região do espaço pode não estar restrita a fazer parte de uma única partição. A exemplo das variantes da árvore-R, o requisito de regiões distintas pode tornar o processamento mais oneroso.

3.5 Árvore-kd

A árvore-kd [BENT1975, BENT1979a] é uma árvore de busca binária *d*-dimensional que representa uma subdivisão recursiva do universo em subespaços por meio de hiperplanos (*d*-1)-dimensionais. Os hiperplanos são iso-orientados e suas direções se alternam entre *d* possibilidades. Para *d*=3, por exemplo, os hiperplanos separadores são alternadamente perpendiculares aos eixos *x*, *y* e *z*. Cada hiperplano separador deve conter pelo menos um elemento, o qual é usado para sua representação na árvore. Os nós interiores têm um ou dois descendentes cada e funcionam como discriminadores para guiar a busca.

Uma desvantagem da árvore-kd é que sua estrutura depende da ordem em que os pontos são inseridos e os dados ficam espalhados por toda a árvore. A árvore-kd adaptativa [BENT1979b] atenua esse problema particionando o espaço de forma que fiquem aproximadamente o mesmo número de elementos em cada partição. Os hiperplanos continuam paralelos aos eixos, porém não precisam conter um elemento e suas direções não têm de ser estritamente alternadas. Como resultado, os pontos de separação não fazem parte dos dados de entrada e todos os dados são armazenados nas folhas.

Um ponto conveniente para passar o hiperplano de separação é a mediana: dado um conjunto finito e ordenado de pontos do espaço $C = \{c_1 \leq c_2 \leq \cdots \leq c_{n-1} \leq c_n\}$, a mediana m_C desse conjunto é definida pelo elemento central $c_{(n+1)/2}$, se n for ímpar, e pela média dos dois elementos intermediários, se n for par. A mediana é uma medida de localização estatística que divide o conjunto de dados C em duas partes com igual número de elementos. Observe que, ao contrário da média, a mediana não é influenciada pela magnitude dos elementos do conjunto. Um fato importante a ser mencionado é que, no cálculo da mediana, devemos levar em consideração a freqüência de cada elemento c_i do conjunto C.

Para elementos unidimensionais no espaço unidimensional, o uso da mediana para a organização hierárquica do conjunto é imediata: ordenação do conjunto; cálculo da mediana; criação de dois novos conjuntos, o primeiro com os elementos menores ou iguais à mediana e o segundo com os demais; e repetição da operação com os novos conjuntos até que uma condição limite seja atingida. Para qualquer outra dimensão, todo o processo é idêntico, com exceção da ordenação. Tratando-se de espaços n-dimensionais com elementos m-dimensionais, o conceito de ordem é mais difuso pois temos mais de uma grandeza para comparar dois elementos e decidir qual deles é maior. É possível utilizar uma grandeza específica, a maior delas, a sua soma, etc.

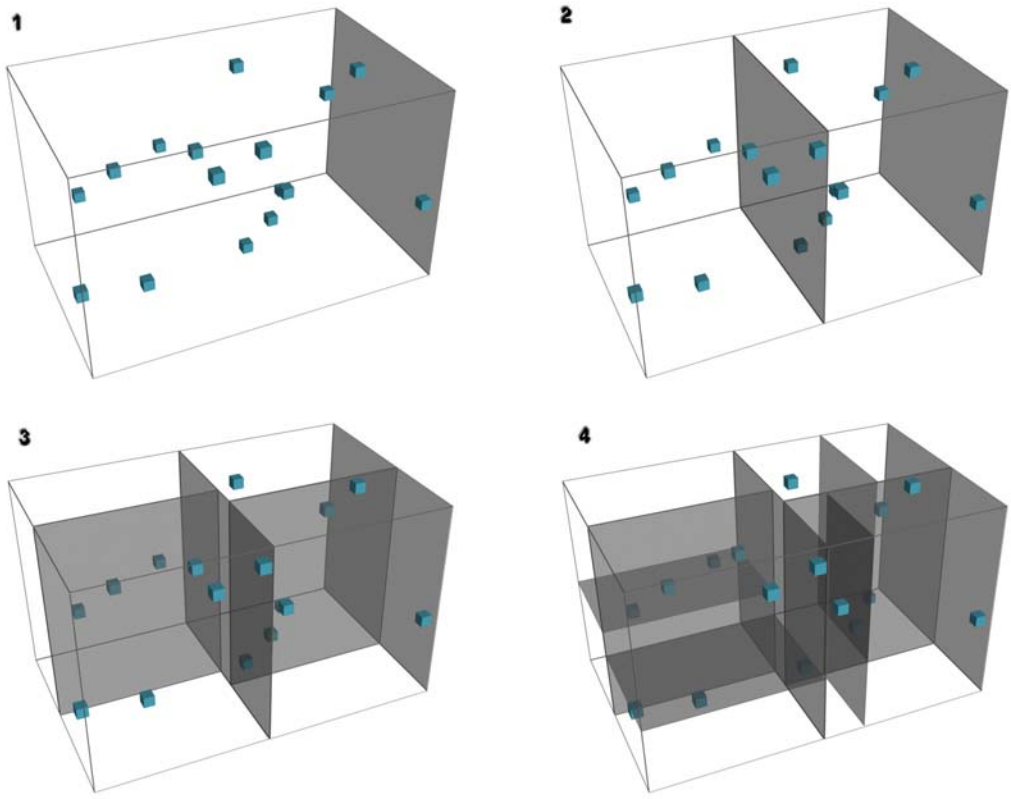

Figura 3.5 - Subdivisões sucessivas de uma cena com 16 objetos utilizando a mediana

No caso particular do espaço $\mathbb{R}^3$, podemos ordenar elementos unidimensionais (pontos) de diversas formas, mas, em se tratando de obter uma mediana, uma forma intuitiva de ordenar esses elementos é de acordo com uma das coordenadas. Toma-se a maior dimensão do conjunto ao longo de um eixo e ordenam-se os pontos segundo a coordenada respectiva. Dada essa ordenação, cria-se a hierarquia conforme descrito no parágrafo anterior.

No tipo de cenário que nos interessa, há elementos tridimensionais num espaço tridimensional. Nesse caso, podemos listar várias formas de ordenar espacialmente os elementos [KAME1993]:

- ordenar os seus centros geométricos como descrito na ordenação de pontos;
- ordenar as coordenadas mínimas da caixa envolvente alinhada com os eixos;

- calcular as caixas envolventes alinhadas com os eixos, designá-las pelas coordenadas mínimas e máximas (min_x, min_y, min_z, max_x, max_y, max_z) e proceder à ordenação como se fossem pontos no espaço $\mathbb{R}^6$.
- ordenar pelo centro da caixa envolvente alinhada com os eixos e extensões medidas na direção dos eixos (c_x, c_y, c_z, e_x, e_y, e_z) no espaço $\mathbb{R}^6$.

A utilização da mediana como separatriz na divisão dos conjuntos conduz à estruturação da cena na topologia de uma árvore binária balanceada, que é a árvore-kd adaptativa convencional. À medida que usarmos outras separatrizes, dividindo a cena em tercis, quartis, quintis, etc., o número de ramos de cada nó aumenta e o número de níveis da árvore diminui. A separatriz apropriada pode ser escolhida com base no tamanho relativo dos elementos, na sua dispersão na cena, na complexidade da cena, em testes de desempenho, etc.

Podemos ainda ter restrições impostas na subdivisão espacial. A classe mais importante de restrições se refere ao *comportamento* de entidades em cenas dinâmicas. Esse assunto será abordado na Seção 3.8.

A seguir, apresentamos o algoritmo para a construção de uma árvore-kd adaptativa para o espaço tridimensional usando os centros das caixas envolventes alinhadas com os eixos de cada objeto geométrico[2].

1. Determine os centros das caixas envolventes alinhadas com os eixos de cada objeto geométrico;
2. Determine as extensões ao longo dos três eixos coordenados do espaço ocupado por esses centros;
3. Tome a maior extensão e ordene os centros segundo ela;
4. Divida os objetos em k grupos segundo a ordenação, onde k é o número de ramos da árvore. Cada grupo será um nó da árvore e os objetos contidos nesse grupo serão descendentes desse nó;
5. Aplique os passos de 1 a 5 para cada novo grupo, até que cada grupo não contenha mais de k objetos gráficos.

3.6 Aglomerados

A organização da cena através de aglomerados parte dos elementos atômicos da hierarquia, objetos ou polígonos, organiza conjuntos de elementos e cria um nó ao qual tais conjuntos são subordinados. Os novos nós, por sua vez, são também organizados em conjuntos e o esquema prossegue até que tenhamos toda a cena subordinada a um único nó. Os pontos centrais desta técnica são os critérios para formar um conjunto e as condições para que um novo nó seja habilitado a ingressar em um conjunto.

Esta discussão pode ser iniciada supondo que a cena seja composta exclusivamente de pontos e que o limite para o tamanho do conjunto seja de dois elementos. No caso particular do tratamento de visibilidade, nos interessa descartar a maior quantidade possível de elementos com o menor número de testes. Portanto, o critério natural para escolher um par de pontos para formar um conjunto é o de proximidade, pois dois elementos próximos têm maior probabilidade de serem descartados em um único teste do que dois elementos distantes um do outro.

Observe que, para encontrar o par mais próximo, *n-1* cálculos de distância têm de ser efetuados. Os dois elementos do primeiro par formado passam a ser representados por um único nó. Este pode ser habilitado a participar de novos conjuntos de imediato, assim que for criado, ou só depois que todos os elementos rotulados como nós folhas já tiverem sido designados para outros conjuntos. No segundo caso, teremos uma árvore binária balanceada com o nível inferior composto apenas de folhas, o nível seguinte formado apenas de nós que possuem folhas como filhos e assim sucessivamente. Se a primeira abordagem for utilizada, a árvore provavelmente não ficará balanceada e surge uma questão: sendo o novo nó um candidato a participar de um conjunto que dará origem a outro nó em função de proximidade, como medir a distância dele aos outros pontos que formam a cena? Uma possibilidade é utilizar um dos seus nós filhos (que é uma folha). As soluções mais comumente aceitas são a de usar o centro geométrico ou a média dos pontos. Observe que esta técnica de organização possui $O(n^2)$ cálculos de distância no número de

[2] O programa de testes apresentado no Capítulo 7 foi implementado usando este algoritmo.

pontos de entrada para o nível inicial. Como a altura da árvore balanceada é $\log_2 n$, teremos uma complexidade final de $O(n^2 \log_2 n)$.

Normalmente, estaremos trabalhando com objetos mais complexos do que pontos. A diferença principal está no critério para escolher os objetos que serão subordinados ao mesmo nó. O critério mais simples é utilizar as distâncias entre os centros geométricos do objetos, ou podemos também usar as menores distâncias entre as superfícies dos objetos, o que constitui um cálculo bem mais complexo. Como estamos tratando de volumes envolventes, outra solução natural seria tomar os pares que possuam o menor volume envolvente. Isso, contudo, implica no cálculo de muitos volumes envolventes para no final aproveitar somente o menor deles, e o custo computacional de se calcular um volume envolvente aliado à complexidade do esquema ($O(n^2 \log_2 n)$) acaba resultando proibitivo. Uma opção é usar heurísticas para reduzir o número de volumes calculados, como uma *octree* para limitar a quantidade de elementos candidatos a serem subordinados a um nó.

A utilização de aglomerados pode ser estendido para a geração de árvores *n*-árias e árvores com um número variável de ramos por nó, em função de restrições como distância máxima aceitável entre elementos e volume interno máximo para um volume envolvente.

O emprego de volumes envolventes como critério para a aglomeração é um tanto delicado, pois a estrutura gerada usando-se esferas envolventes pode ser diferente da obtida com caixas envolventes, por exemplo. Como neste trabalho a proposta é trabalhar com vários tipos de volumes simultaneamente, a escolha do volume como critério pode, ainda que sutilmente, influenciar os resultados, sugerindo que se experimente mais de um volume para calcular os aglomerados.

3.7 Árvores-R

A árvore-R é uma estrutura proposta por [GUTT1984] para tratar dados espaciais, do tipo usado em projetos auxiliados por computador (CAD), geoprocessamento, robótica e visão computacional, cujo objetivo é proporcionar um mecanismo que ajude a recuperar itens de dados com rapidez de acordo com suas localizações espaciais. O sistema de indexação baseia-se em caixas envolventes

alinhadas com os eixos construídas num esquema hierárquico, semelhante ás árvores de volumes envolventes.

A árvore-R original e maioria das suas variantes foram concebidas para ambientes dinâmicos, sendo capazes de tratar inserções e exclusões. Assim como a maioria das estruturas dessa categoria, a árvore-R cumpre melhor sua destinação se estiver convenientemente balanceada. Como o balanceamento é uma operação reconhecidamente custosa, são mantidos espaços livres na árvore para minimizar a necessidade de realizar operações de balanceamento, sacrificando a economia de memória. Operando deste modo, a árvore-R garante que a utilização de espaço seja de pelo menos 50%. Resultados experimentais mostraram que a utilização média é de aproximadamente 70%. Apesar dessa reserva de espaço, não é possível evitar totalmente os balanceamentos, e há certas classes de aplicações interativas que não podem arcar com esses custos sem que se prejudique o curso da simulação.

3.7.1 Árvores-R Estáticas

A árvore-R tem por objetivo manter índices para facilitar a inserção e recuperação de dados espaciais armazenados em bancos de dados. Muitas aplicações operam sobre massas de dados estáticas. Isso permite não só compactar a árvore, eliminado nós vazios, como também obter respostas mais rápidas, pois os nós terão um número maior de ramos ativos e a árvore será possivelmente menor. Kamel e Faloutsos [KAME1993] apresentam um estudo sobre a criação de árvores-R compactas para dados estáticos e propõem métodos que obtêm 100% de utilização do espaço e respostas pelo menos tão rápidas quanto as árvores-R e suas variantes.

O primeiro método proposto para o preenchimento de árvores-R foi provavelmente apresentado por [ROUS1985], que propõe um método para construir árvores-R para dados bidimensionais que atingem quase 100% de utilização do espaço. A lista ordenada de retângulos envolventes é varrida; retângulos sucessivos são designados para o mesmo nó folha da árvore-R, até que o nó esteja completo; então, um novo nó folha é criado e a varredura da lista ordenada continua. Assim, todos os nós da árvore-R resultante serão totalmente preenchidos, com exceção possivelmente do último nó de cada nível, de modo que a utilização será de aproximadamente 100%. Resultados experimentais usando dados compostos por

pontos mostraram que árvores-R preenchidas desta forma apresentam um desempenho superior ao de árvores-R construídas por subdivisão linear e quadrática e árvores-R* para consultas por pontos. Contudo, seu desempenho não é tão bom para consultas por regiões ou por dados retangulares.

Este método também é conhecido como *árvore-R compacta menor-x*, pois uma forma de implementá-lo é ordenar os retângulos de acordo com o valor da coordenada *x* do canto inferior esquerdo (*'menor-x'*). A ordenação por qualquer dos outros três valores leva a resultados semelhantes. Embora este esquema tenha um desempenho muito bom para consultas de dados compostos por pontos, seu desempenho cai para consultas maiores. A Figura 3.6 destaca o problema. O fato de os nós pais cobrirem áreas pequenas explica por que a *árvore-R compacta menor-x* apresenta um desempenho tão bom para consultas por pontos e os seus grandes perímetros justificam a queda do desempenho em pesquisas por região. Intuitivamente, um algoritmo de preenchimento deveria designar pontos próximos para o mesmo nó folha. Ignorando a coordenada *y*, a *árvore-R compacta menor-x* tende a violar essa regra empírica.

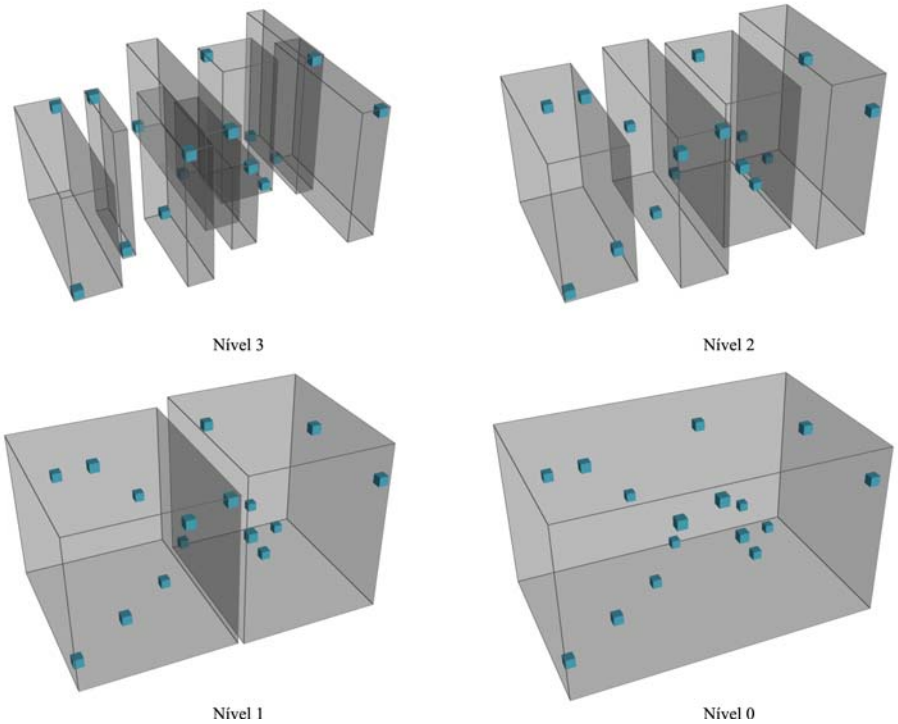

Figura 3.6 - Caixas envolventes alinhadas com os eixos de uma *árvore-R compacta menor-x* (construída de baixo para cima)

Para aglomerar os dados de uma forma melhor, [KAME1993] propõe o uso de curvas de preenchimento do espaço (ou fractais), em especial a curva de Hilbert [HILB1891].

Uma curva de preenchimento de espaço visita todos os pontos em uma grade k-dimensional exatamente sem nenhuma auto-interseção. Ordem-z, curva de Hilbert e curva de código cinza são exemplos de curvas de preenchimento de espaço. Em [FALO1989] é demonstrado experimentalmente que, dentre os três métodos acima, a curva de Hilbert atinge os melhores resultados de aglomeração.

3.7.2 Curva de Hilbert

A curva de Hilbert 2D básica em uma grade 2x2, denotada por H_1, é mostrada na Figura 3.7. Para derivar uma curva de ordem *i*, cada vértice da curva básica é trocado pela curva de ordem *i*-1, a qual pode ser convenientemente rodada ou refletida. A Figura 3.7 também apresenta curvas de Hilbert de ordem 2, 3 e 4. Quando a ordem da curva tende ao infinito, a curva resultante é *fractal*, com dimensão 2. A curva de Hilbert pode ser generalizada para dimensões maiores. Bially [BIAL1969] apresenta um algoritmo para traçar curvas em dimensões maiores. A Figura 3.8 ilustra curvas de Hilbert 3D de ordem 1, 2, 3 e 4.

O caminho de uma curva de preenchimento do espaço impõe uma ordenação linear nos pontos da grade, que pode ser calculada partindo-se de uma extremidade da curva e seguindo o caminho até a outra extremidade. A localização de dados pontuais ao longo da curva é imediata, porém a localização de dados planares ou volumétricos é mais difusa. [KAME1993] propõe três formas de localizar retângulos envolventes alinhados com os eixos em curvas de Hilbert 2D:

Hilbert 4D através dos cantos ("4D-xy"): cada retângulo é mapeado em um ponto no espaço tetradimensional formado pelo canto inferior esquerdo e o canto superior esquerdo, designado por (*infx*, *infy*, *supx*, *supy*).

Hilbert 4D através do centro e extensões ("4D-ce"): cada retângulo é mapeado em um ponto no espaço tetradimensional formado pelo centro e pelas extensões, designado por (*cx*, c*y*, e*x*, e*y*).

Hilbert 2D através do centro apenas ("2D-c"): cada retângulo é mapeado apenas por seu centro no espaço bidimensional. De acordo com os testes realizados, este foi o esquema que obteve os melhores resultados, apesar de ser o mais simples.

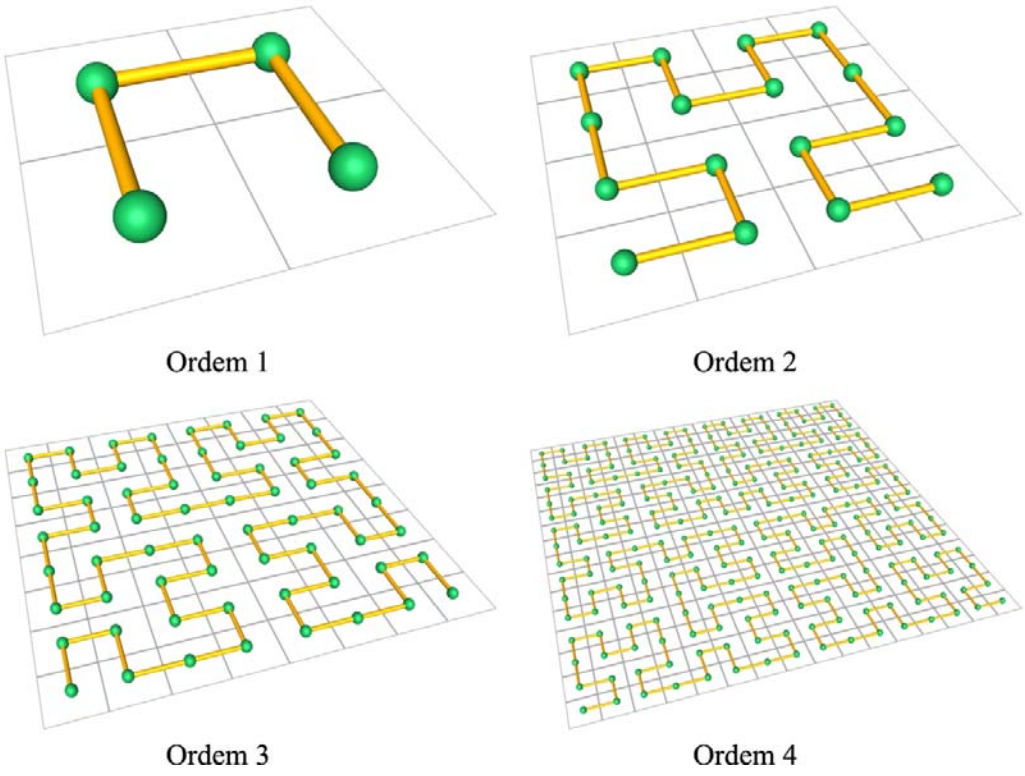

Figura 3.7 - Curvas de Hilbert 2D de ordem 1, 2, 3 e 4

Embora o foco desse trabalho tenha sido a otimização de operações de consulta a bancos de dados espaciais, a sua utilização no tratamento de visibilidade é imediata: basta fazer uma extensão do espaço bidimensional para o tridimensional e a consulta passa a ser pelo volume de visão no lugar de um retângulo.

A estrutura resultante desse trabalho é uma árvore de caixas envolventes alinhadas com os eixos. Uma derivação importante é o aproveitamento dessa árvore para a implementação de outros tipos de volumes envolventes, como os que serão apresentados no Capítulo 4.

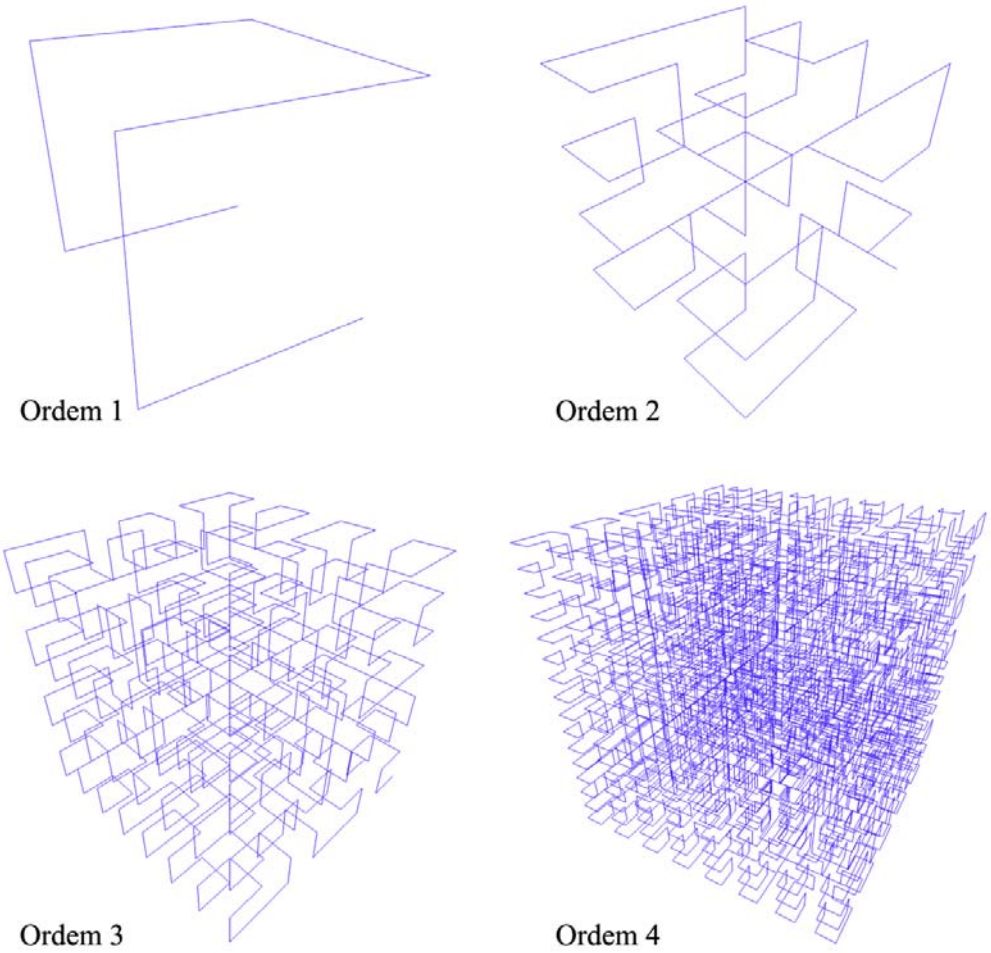

Figura 3.8 - Curvas de Hilbert 3D de ordem 1, 2, 3 e 4

Abaixo é apresentado o algoritmo para a construção de uma árvore-R compacta para o espaço tridimensional usando os centros das caixas envolventes alinhadas com os eixos de cada objeto geométrico através da sua coordenada de Hilbert.

1. Calcule a coordenada de Hilbert para o centro da caixa alinhada com os eixos de cada objeto geométrico;
2. Ordene os objetos geométricos segundo a sua coordenada de Hilbert;
3. Agrupe os objetos geométricos de *k* em *k* objetos, onde *k* é o número de ramos da árvore, seguindo a ordenação acima. Cada grupo é um novo nó da árvore;

4. Repita os passos de 1 a 4 para os grupos criados no passo 3 até que seja criado um último grupo que contenha todos os objetos geométricos.

3.8 Árvores de Volumes Envolventes

3.8.1 Árvores de Volumes Envolventes de Modelos Estáticos

O já citado trabalho de Clark [CLAR1976] propõe o cálculo de volumes envolventes para cada nó da árvore estrutural de um modelo com base nos vértices das folhas para serem usados em testes de visibilidade, como será visto no Capítulo 5. Uma dificuldade deste método é que os cálculos dos volumes são muito caros para serem feitos no decorrer da simulação, tornando seu uso viável apenas para modelos ou partes de modelos cujos componentes não possuam movimento relativo entre si. Se um componente se mover, basta que o seu volume envolvente acompanhe esse movimento, mas os volumes envolventes dos nós superiores da árvore ficarão desatualizados.

3.8.2 Árvores de Volumes Envolventes de Modelos Dinâmicos

Há duas formas imediatas de tratar modelos com componentes móveis. A solução trivial consiste em calcular e usar os volumes evolventes somente para níveis da árvore do modelo a partir dos quais não haja mais movimento relativo entre suas partes. A segunda forma é recalcular grosseiramente os volumes envolventes desatualizados a partir dos volumes envolventes dos nós imediatamente subordinados, em vez de usar os vértices contidos nas folhas subordinadas. Isso, além de resultar em volumes bem maiores do que o necessário, só é viável se o número de volumes recalculados for bastante pequeno.

Propomos, aqui, outras duas formas de tratar esses modelos que podem ser usadas de forma combinada com as anteriores. A primeira é calcular volumes envolventes para uma posição preferencial, se ela existir (como o canhão apontando para frente ilustrado na Figura 3.9), e utilizar um sentinela para assinalar a saída dessa posição. Neste caso, a visibilidade é resolvida utilizando os volumes envolventes dos nós filhos. Esta estratégia tem o uso restrito a modelos ou partes deles que possuam

uma posição preferencial bem evidente. A segunda proposta é usar envoltórias de movimento para calcular os volumes envolventes dos nós superiores.

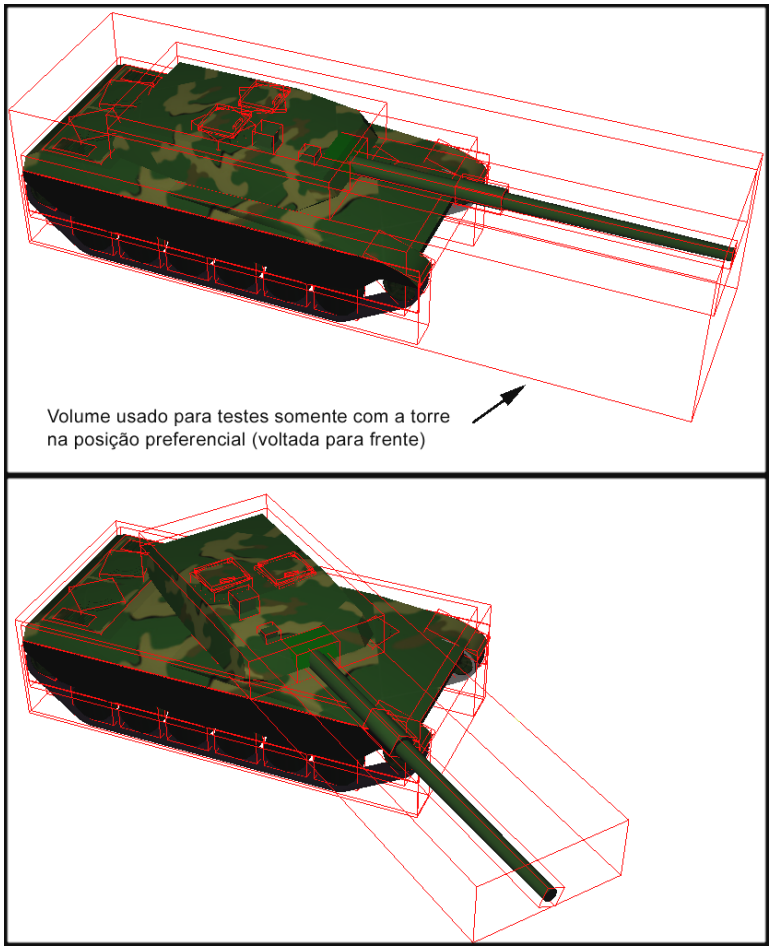

Figura 3.9 - Objeto com posição preferencial: um sentinela avisa quando a torre não está na posição preferencial e a caixa envolvente principal não é mais usada

3.8.2.1 Árvore de volumes envolventes de objetos articulados

Para tirar proveito das envoltórias de movimento na árvore de volumes envolventes de um modelo, como citado acima, podemos usar uma hierarquia combinada de vínculos e composição conforme ilustrado na Figura 3.10. Enquanto temos a liberdade de reorganizar a composição de acordo com os métodos citados nas Seções 3.5, 3.6 e 3.7, devemos manter inalterada a organização de vínculos para preservar a hierarquia de instanciações. Conforme visto na Seção 3.2, temos objetos

articulados construídos a partir de objetos compostos. Para os objetos elementares e para os compostos, calculamos volumes envolventes ajustados à sua geometria, enquanto para objetos articulados calculamos os volumes envolventes a partir das envoltórias de movimento de seus componentes.

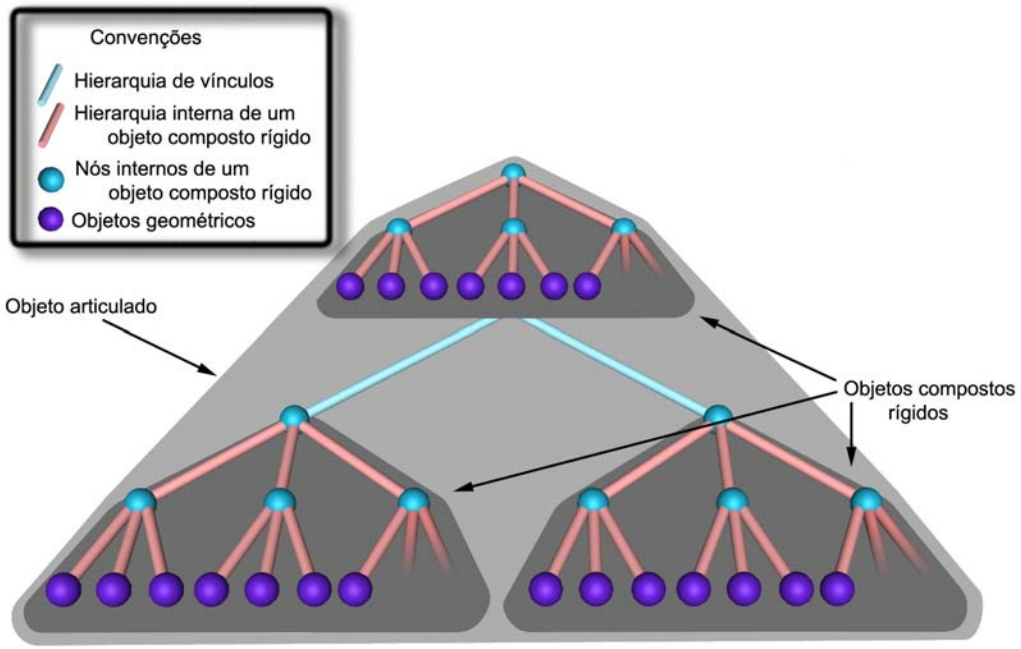

Figura 3.10 - Hierarquia de vínculos e de composição

Suponhamos, por exemplo, um tanque simplificado composto essencialmente pela carcaça e torre do canhão. Como a carcaça e a torre são individualmente íntegras e sem peças com movimentos relativos, elas podem ter volumes envolventes estreitamente ajustados. Já o volume que envolve todo o tanque deve levar em consideração todo o movimento que a torre tem liberdade de fazer em relação à carcaça, como mostra a Figura 3.11. Esse relaxamento no ajuste do volume acarreta naturalmente a desvantagem de o tanque ser aceito quando nenhuma das suas partes for visível, mas que é resolvida ao se processar os nós descendentes.

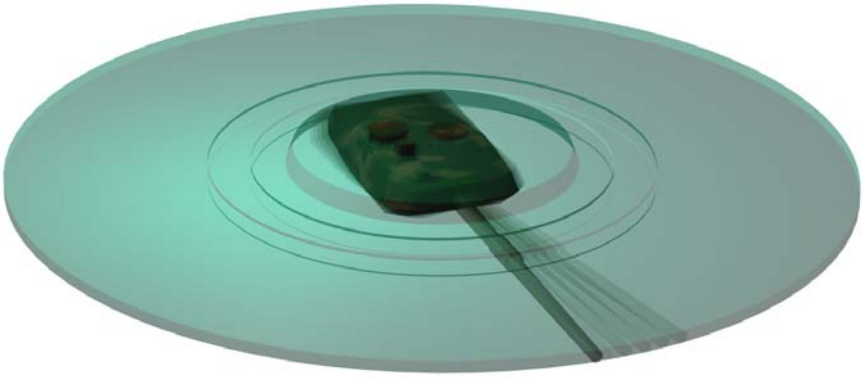

Figura 3.11 - Envoltória de movimento da torre (o movimento vertical do canhão e outros não foram considerados)

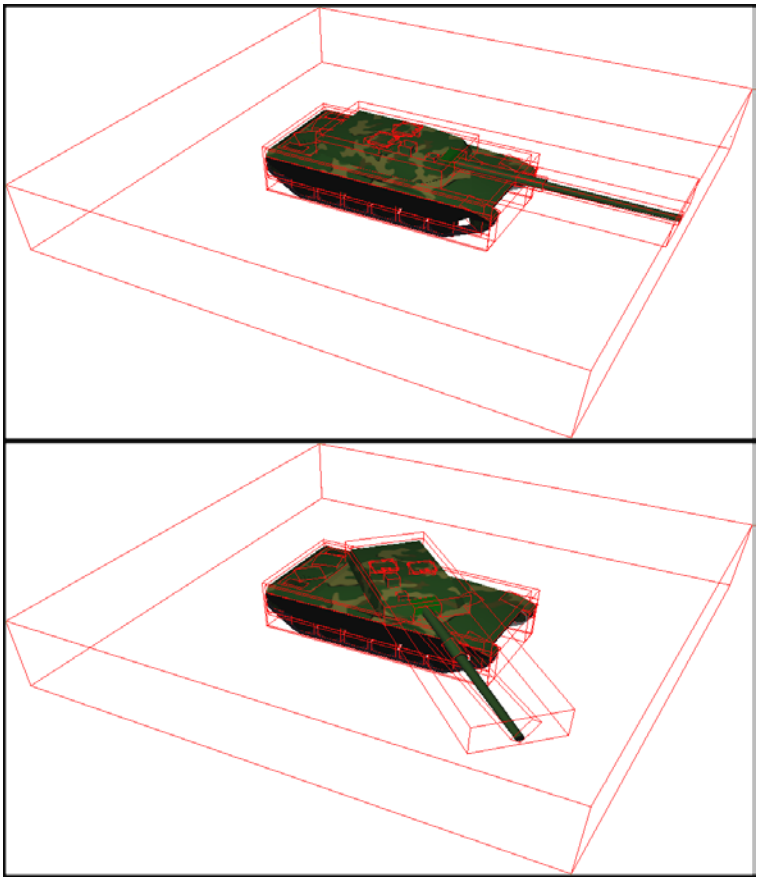

Figura 3.12 - Objeto com comportamento determinístico: a caixa mais externa envolve todas as possíveis posições da torre

A Figura 3.13 mostra a árvore do tanque com a torre subordinada à carcaça de forma que, quando a carcaça se move, a torre acompanha esse movimento. A torre, por sua vez, possui um movimento restrito de rotação em torno de um eixo vertical. A Figura 3.14 apresenta a árvore do mesmo tanque com seus objetos compostos reorganizados.

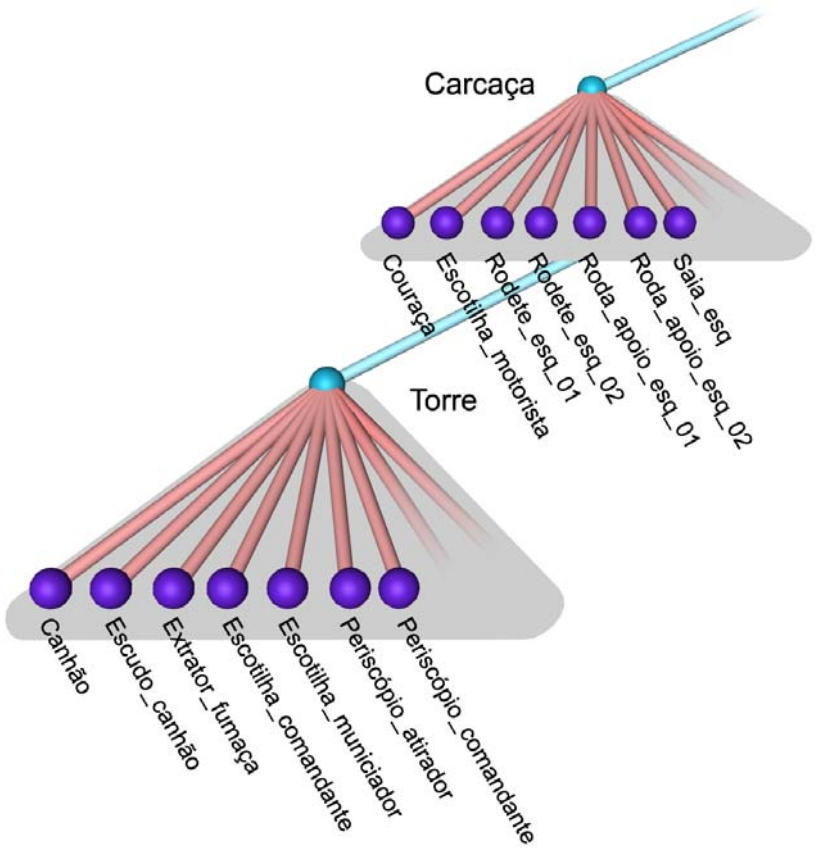

Figura 3.13 - Árvore mostrando a hierarquia combinada de vínculos e de composição para o tanque da Figura 3.9 conforme proveniente do modelador

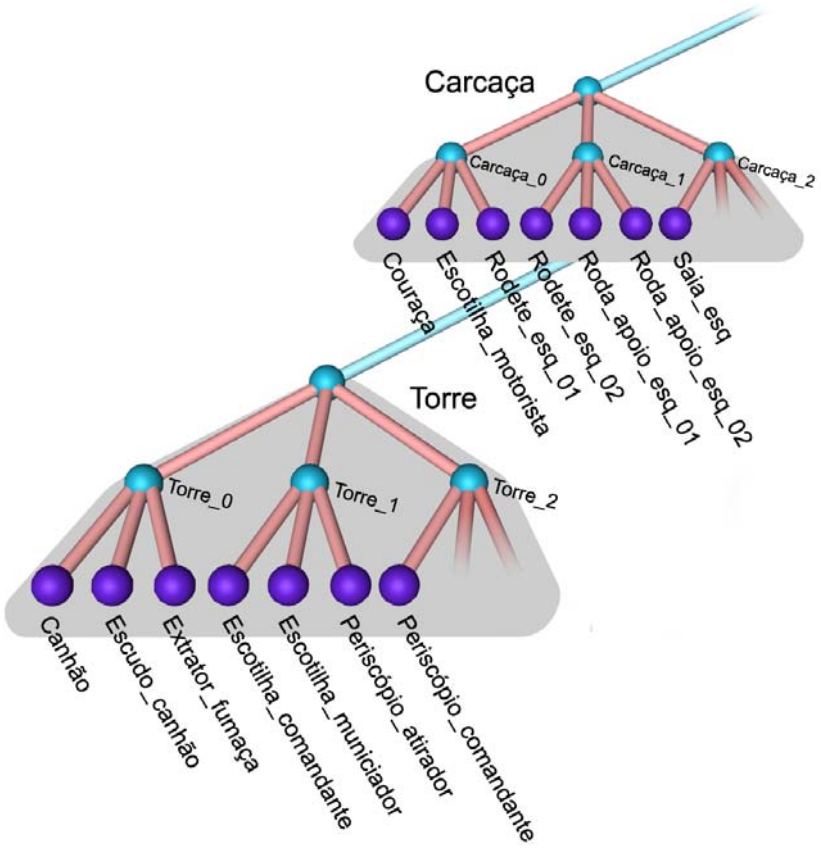

Figura 3.14 - Grafo mostrando a hierarquia combinada de vínculos e de composição para o tanque da Figura 3.12 depois de reorganizada segundo um dos critérios citados nas Seções 3.5, 3.6 e 3.7

3.8.2.2 Árvore de volumes envolventes de objetos com movimento livre

Estendendo o uso de envoltórias de movimento, propomos uma hierarquia de dois níveis, ilustrada na Figura 3.15. Esse esquema pretende atender o requisito de que alguns objetos devem ser agrupados, pois fazem parte de uma composição ou possuem vínculos, enquanto outros são independentes e não podem ser indexados conjuntamente durante toda a simulação.

A hierarquia de mais baixo nível representa índices de macro-objetos, tais como plataformas marítimas, aviões e navios. É uma hierarquia estática construída numa fase de pré-processamento. Desta forma, temos tempo de processamento suficiente para buscar uma forma apropriada de usar os volumes envolventes e uma estratégia apropriada para combiná-los.

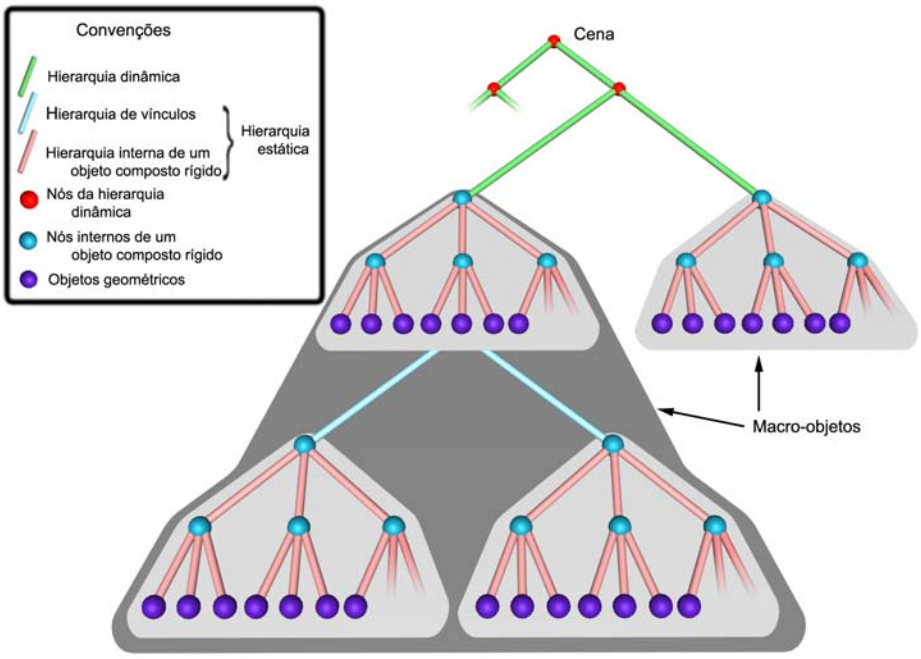

Figura 3.15 - Indexação espacial de dois níveis

Nesta proposta, um macro-objeto é composto de objetos estáticos e objetos vinculados cujos movimentos relativos são conhecidos e de extensão razoavelmente restrita. Para evitar a atualização da hierarquia toda vez que um objeto vinculado se move, calculamos dois volumes envolventes para cada parte móvel: o primeiro se ajusta à sua geometria na posição inicial e o segundo à sua envoltória de movimentos. Este segundo volume é usado somente durante a construção da hierarquia, uma vez que representa uma geometria virtual utilizada para combinar um objeto com outro. Desta forma, se o objeto se mover, o programa de visualização precisa transformar apenas o primeiro volume envolvente. Nenhuma atualização é necessária porque o objeto se move dentro de uma região já considerada durante a fase de pré-processamento.

Em simulações reais, os macro-objetos se movem livremente, e dois objetos inicialmente próximos podem se distanciar um do outro. Por isso, nossa proposta é não apenas atualizar a hierarquia de volumes envolventes mas também reagrupar os nós baseados em coerência espacial. A melhor estratégia para efetivamente manter essa hierarquia dinâmica ainda está sendo investigada. Temos considerado a combinação de hierarquias e técnicas de compartimentação.

4 Volumes Envolventes

4.1 Esfera

Uma esfera é definida pelo conjunto de todos os pontos $\vec{X}$ eqüidistantes de um ponto central com uma distância r > 0. A equação quadrática que define este conjunto é $\left|\vec{X}-\vec{C}\right|^2 = r^2$. Para um objeto geométrico que consiste de uma coleção de pontos $\left\{\vec{V}_i\right\}_{i=0}^{n}$, uma esfera envolvente pode ser calculada de várias formas.

4.1.1 Esfera Contendo a Caixa Alinhada com os Eixos

Uma abordagem simples é calcular a caixa mínima alinhada com os eixos que contenha todos os pontos e então determinar a esfera mínima que contenha os vértices da caixa com centro coincidente com o centro da caixa. Este cálculo é bastante rápido, porém a esfera calculada desta forma não se ajusta tão bem ao objeto quanto poderia.

4.1.2 Esfera Centrada na Média dos Pontos

Esta forma de cálculo gera uma esfera potencialmente mais ajustada aos vértices do objeto, mas implica em um custo maior para ser obtida. Primeiro determina-se o seu centro, que é a média dos valores dos vértices do objeto. Em seguida, o seu raio é obtido a partir da distância do vértice mais afastado ao centro calculado.

4.1.3 Esfera Mínima

Uma forma de calcular a esfera envolvente mínima é através do algoritmo baseado no trabalho de Emo Welzl [WELZ1991]. Esta solução emprega um algoritmo randômico de complexidade *esperada* de ordem linear. No pior caso, o algoritmo é polinomial no número de entradas, mas, como as entradas são aleatoriamente permutadas, o pior caso pode ser desprezado.

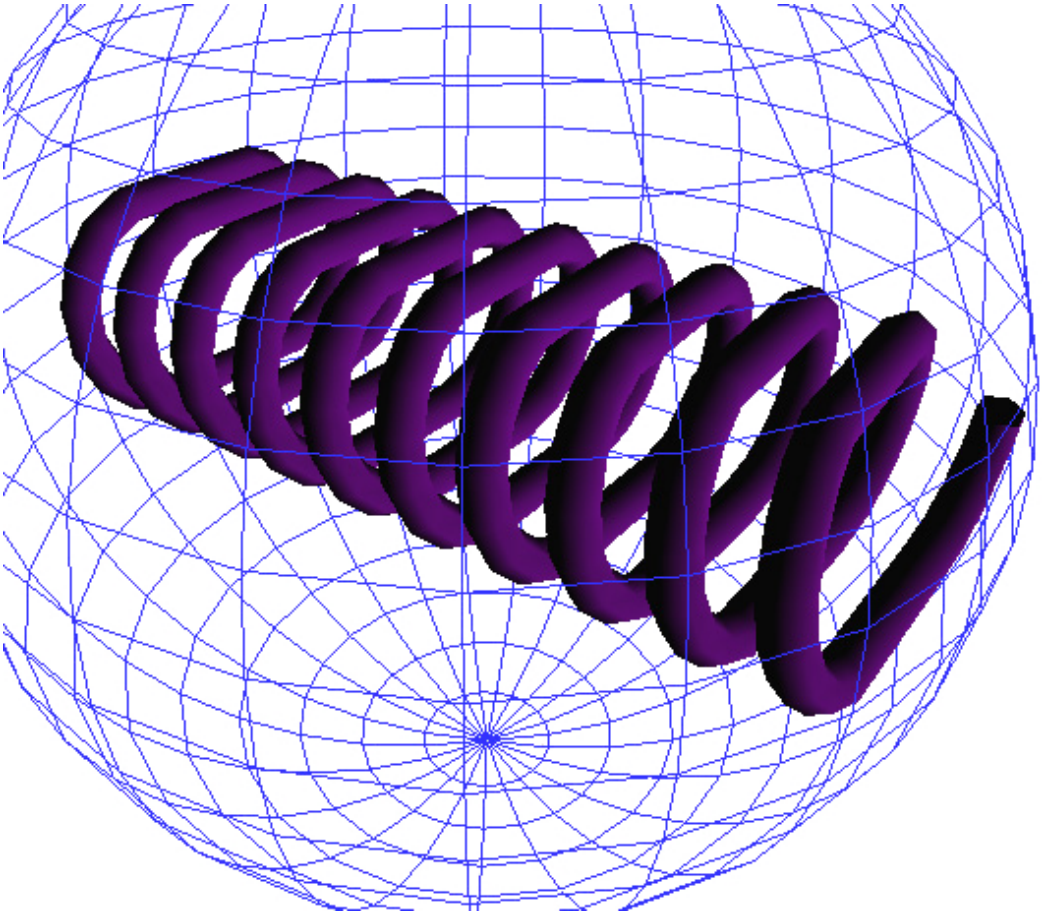

Figura 4.1 - Esfera envolvente

O pseudo-código abaixo calcula a esfera envolvente mínima a partir de n pontos $P[0]$ até $P[n-1]$. O processo consiste em manter um conjunto de pontos de apoio para a esfera enquanto são processados os pontos do conjunto de entrada, um ponto por vez. Os pontos de apoio ficam na superfície da esfera, de modo que não são necessários outros pontos para determinar a esfera.

```
TEsfera CalcularEsferaMinima(int N,Tponto3D p[]){
    permutar aleatoriamente P[0]..P[n-1];
    Tesfera esfera = EsferaExata1(p[0]);
    TconjuntoDePontos apoio = {p[0]};
    i=1;
    enquanto (i<n){
        se (p[I] não pertence a apoio){
            se (p[i] não pertence à esfera){
                adicionar p[i] à apoio e (possivelmente) remover pontos desnecessários;
                calcular a esfera usando o apoio corrente;
                i = 0; //reiniciar quando o apoio muda
```

```
                continuar;
            }
        }
        i++;
    }
}
```

Internamente, o algoritmo requer que as esferas sejam calculadas usando-se exatamente dois pontos, exatamente três pontos ou exatamente quatro pontos. A atualização do apoio pode ser modularizada em uma coleção de funções de atualização, uma para cada número de pontos de apoio.

4.2 Caixas Orientadas

Caixas orientadas geralmente se ajustam melhor a um objeto do que esferas. Uma caixa orientada é definida por um centro $\vec{C}$, três vetores ortonormais $\vec{U}_i$, que formam um sistema de coordenadas obedecendo à regra da mão direita, e três extensões $e_i > 0$ com i = 0, 1, 2. Seja $R = \left[\vec{U}_0 \vec{U}_1 \vec{U}_2\right]$, uma matriz ortonormal com determinante igual a um, qualquer ponto $\vec{X} = \left(x_0\ x_1\ x_2\right)$ dentro da caixa ou na superfície pode ser representado por $\vec{X} = \vec{C} + R\vec{Y}$, onde $\vec{Y} = \left(y_0\ y_1\ y_2\right)$ com $\left|y_i\right| \le e_i$ para todo i.

4.2.1 Ajuste de Pontos Usando uma Distribuição Gaussiana

Uma distribuição Gaussiana é a distribuição de probabilidade com a forma $A\exp((\vec{X} - \vec{C})M^{-1}(\vec{X} - \vec{C}))$, onde A é um fator de escala apropriado, $\vec{C}$ é a média da distribuição e M é a matriz de covariância da distribuição. A distribuição é dita anisotrópica se os autovalores de M não forem todos iguais.

Um método mais sofisticado para construir uma caixa orientada, que normalmente ajusta os pontos melhor do que as caixas alinhadas com os eixos, é baseado no ajuste de pontos com uma distribuição Gaussiana anisotrópica. O centro da caixa é a média dos pontos,

$$\vec{C} = \frac{1}{n}\sum_{j=0}^{n} \vec{V}_j \ .$$

Os eixos da caixa são determinados como sendo os auto-vetores da matriz de covariância

$$M = \frac{1}{n}\sum_{j=0}^{n}(\vec{V}_j - \vec{C})(\vec{V}_j - \vec{C})^T .$$

Se $\vec{U}_j$ são os auto-vetores unitários, as extensões ao longo destes eixos são os extremos das projeções dos pontos sobre esses eixos, $e_i = \max_j \left|\vec{U}_i . (\vec{V}_j - \vec{C})\right|$. O pseudocódigo é

```
// TCaixa é composta por centro, eixo[3] e extensão[3]
TCaixa caixa;

//Calcula a média dos pontos
Tponto3D somatório = v[0];
para (i = 1; i < n; i++)
    somatório += v[i];
caixa.centro = somatório/n;

//Calcula as covariâncias dos pontos
TMatrix3x3 mat=0;
para (i = 0; i < n; i++){
    Tponto3D delta = v[i] - caixa.centro;
    mat += Tensor(delta,delta);
}
TMatrix3x3 covariância = mat/n;

//Os auto-vetores da matriz de covariância são os eixos da //caixa
ExtrairAutoVetores(covariância,caixa.eixos[3]);

//Calcula as extensões como sendo os valores extremos das
//projeções dos pontos sobre os eixos da caixa
caixa.extensão = 0;
para (i = 0; i < n; i++){
    Tponto3D delta = v[i] - caixa.centro;
    para (j = 0; j < 3; j++){
        real escalar = |ProdutoEscalar(caixa.eixo[j],delta)|;
        se (escalar > caixa. extensão[j])
            caixa. extensão[j] = escalar;
    }
}
```

Para um vetor $\vec{W}$, Tensor($\vec{W},\vec{W}$) é a matriz $\vec{W}\vec{W}^T$. O algoritmo requer um código para obter os auto-vetores de uma matriz 3 x 3. Os auto-vetores podem ser calculados utilizando-se uma solução aproximada em vez de um esquema iterativo.

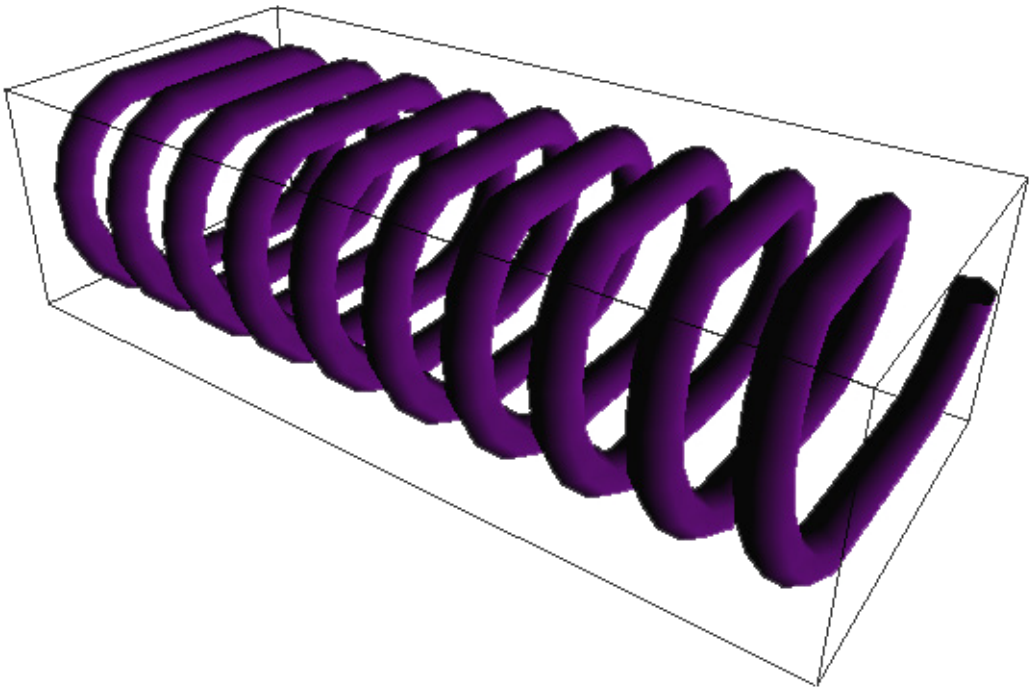

Figura 4.2 - Caixa envolvente orientada

Uma variação do algoritmo consiste em calcular primeiro o fecho convexo dos pontos e então calcular a caixa orientada do fecho. Outra variação é computar os auto-vetores da matriz de covariância, projetar os pontos sobre as retas $\vec{C}+t\vec{U}_i$, $t \in \mathbb{R}$, na direção dos auto-vetores $\vec{U}_i$, e então calcular os intervalos de projeção $[\min_i, \max_i]$. O ponto $\vec{C}$ precisa ser substituído pelo verdadeiro centro da caixa, induzido pelos intervalos projetados

$$\vec{C}' = \vec{C} + \sum_{j=0}^{2} \frac{\min_i + \max_i}{2} \vec{U}_i .$$

4.2.2 Caixa Envolvente Mínima

A caixa melhor ajustada pode ser considerada como a de menor volume que contenha todos os pontos. A construção desta caixa requer um esquema iterativo para resolver um problema de minimização, sendo bem mais custoso do que a baseada na distribuição gaussiana para que seja calculada em tempo real.

Para quaisquer eixos coordenados $\vec{A}_i$ tomados, com i = 0, 1, 2, os pontos são projetados sobre os eixos $\vec{V}_0 + s\vec{A}_i$, $s \in \mathbb{R}$, sendo os valores $\rho_{ij} = \vec{A}_i.(\vec{V}_j - \vec{V}_0)$ para todo j. Define-se $\alpha_i = \min_j(\rho_{ij}), \beta_i = \max_j(\rho_{ij})$, e $\gamma_i = (\alpha_i + \beta_i)/2$. O centro da caixa orientada envolvente mínima com os eixos especificados é

$$\vec{C} = \vec{V}_0 + \sum_{i=0}^{2} \vec{A}_i \gamma_i .$$

As extensões da caixa orientada são $a_i = (\alpha_i + \beta_i)/2$.

Cada conjunto de eixos coordenados pode ser representado por colunas de matrizes de rotação. Cada matriz de rotação é gerada por um vetor unitário $\vec{U}$ e um ângulo $\theta \in [0,2\pi]$. O mapeamento de matrizes de rotação em eixos coordenados não é de um para um, naturalmente. Contudo, o volume das caixas orientadas pode ser visto como uma função $S^2 \times [0,2\pi] \to [0,\infty)$, onde S^2 é a esfera unitária. O volume é $v(\vec{U},\theta) = \prod_{i=0}^{2}(\beta_i - \alpha_i)$. Esta função é contínua no seu domínio compacto de forma que, a partir do cálculo, ela atinja o seu mínimo nesse domínio. Deste modo, existem um eixo $\vec{U}_0$ e um ângulo θ_0 para os quais $v(\vec{U}_0,\theta_0) \le v(\vec{U},\theta)$ para todos os eixos $\vec{U}$ e ângulos θ. A construção de $\vec{U}_0$ e θ_0 pode ser implementada empregando minimização numérica através de técnicas que não requerem derivação. Uma boa escolha é o método de ajuste de direção de Powell [PRES1988]. A razão de convergência para o mínimo depende das suposições iniciais para o eixo e o ângulo.

4.2.3 Ajuste de Triângulos com uma Distribuição Gaussiana

Este método foi apresentado por Gottschalk, Lin e Manocha [GOTT1996]. Se os pontos são vértices de uma malha de triângulos, os próprios triângulos podem ser utilizados para gerar uma caixa orientada contendo os vértices. O ajuste de uma caixa ao fecho convexo dos vértices dados previamente apresenta problemas de amostragem, pois os vértices pertencentes ao fecho convexo podem estar irregularmente distribuídos, de forma que uma concentração pequena e densa de vértices pode afetar incorretamente a orientação da caixa envolvente. Este efeito pode ser minimizado empregando-se uma formulação contínua da matriz de covariância.

Suponha que existam *l* triângulos. Se o *i*-ésimo triângulo tem vértices $\vec{V}_{0,i}$, $\vec{V}_{1,i}$ e $\vec{V}_{2,i}$, então o triângulo e seu interior são representados por $\vec{X}_i(s,t) = \vec{V}_{0,i} + s(\vec{V}_{1,i} - \vec{V}_{0,i}) + t(\vec{V}_{2,i} - \vec{V}_{0,i})$ para $0 \le s \le 1$, $0 \le t \le 1$ e $s + t \le 1$. Seja $m_i = \left| (\vec{V}_{1,i} - \vec{V}_{0,i}) \times (\vec{V}_{2,i} - \vec{V}_{0,i}) \right| / 2$ a área do triângulo. Defina os pesos $w_i = m_i / \sum_{i=1}^{l-1} m_i$. O ponto médio do fecho convexo é

$$\vec{C} = \frac{2}{l} \sum_{i=0}^{l-1} w_i \int_0^1 \int_0^{1-t} \vec{X}_i(s,t) ds\, dt = \frac{1}{3l} \sum_{i=0}^{l-1} w_i \left(\sum_{j=0}^{2} \vec{V}_{j,i} \right)$$

e a matriz de covariância do fecho convexo é

$$M = \frac{2}{l} \sum_{i=0}^{l-1} w_i \int_0^1 \int_0^{1-t} \left(\vec{X}_i(s,t) - \vec{C} \right) \left(\vec{X}_i(s,t) - \vec{C} \right)^T ds\, dt = \frac{1}{12l} \sum_{i=0}^{l-1} w_i \left(\sum_{j=0}^{2} \sum_{k=0}^{2} \left(\vec{V}_{j,i} - \vec{C} \right) \left(\vec{V}_{k,i} - \vec{C} \right)^T \right)$$

Se $\vec{A}_i$ são auto-vetores unitários, as extensões ao longo desses eixos são $a_i = \max_j \left| \vec{A}_i . (\vec{X}_j - \vec{C}) \right|$, onde $\vec{X}_j$ são os vértices. Assim como na Seção 3.2.1, uma variação permite o ajuste de $\vec{C}$ uma vez conhecidos os eixos $\vec{A}_i$.

4.3 Cápsulas

Uma cápsula é uma extensão natural de uma esfera baseada em eqüidistância. É definida como um conjunto de todos os pontos que estão a uma distância $r > 0$ de um segmento de reta com extremidade $\vec{P}$ e direção $\vec{D}$. A outra extremidade é $\vec{P} + \vec{D}$. Uma cápsula é um cilindro com a base e o topo cobertos por dois hemisférios.

A seguir, apresentamos um algoritmo para envolver os pontos $\{\vec{X}_i\}_{i=0}^{n}$ por uma cápsula incluindo ajuste através de mínimos quadrados.

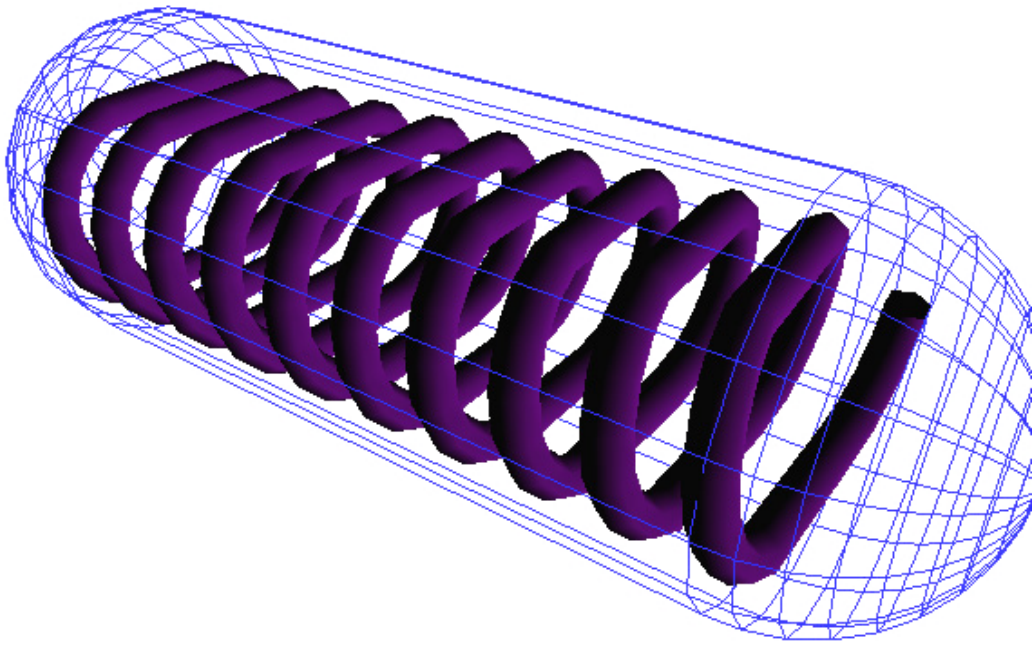

Figura 4.3 - Cápsula envolvente

4.3.1 Ajuste Através de Mínimos Quadrados

Seja a reta $\vec{A}+t\vec{W}$, onde $\vec{W}$ é um vetor unitário e $\vec{A}$ é a média dos pontos dados, a reta conterá o segmento de reta da cápsula. Selecionam-se os vetores $\vec{U}$ e $\vec{V}$ de modo que a matriz $R=\left[\vec{U}\ \vec{V}\ \vec{W}\right]$ seja ortonormal e tenha determinante unitário. Os pontos dados podem ser representados como $\vec{X}_i=\vec{A}+R\vec{Y}_i$, onde $\vec{Y}_i=(u_i,v_i,w_i)$. No sistema de coordenadas (u,v,w), o eixo da cápsula é contido pela reta $t(0,0,1)$. Precisamos calcular o maior ξ_0 de modo que todos os pontos fiquem sobre o hemisfério $u^2+v^2+(w-\xi_0)^2=r^2$ com $w\le\xi_0$. O valor é calculado como sendo

$$\xi_0=\min_i\left\{w_i+\sqrt{r^2-\left(u_i^2+v_i^2\right)}\right\},$$

onde $0\le i\le n$. Similarmente, calculamos o menor valor ξ_1 tal que todos os pontos fiquem abaixo do hemisfério $u^2+v^2+(w-\xi_1)^2=r^2$ com $w\ge\xi_1$. O valor é calculado como sendo

$$\xi_1=\max_i\left\{w_i+\sqrt{r^2-\left(u_i^2+v_i^2\right)}\right\}.$$

As extremidades do segmento de reta da cápsula são $\vec{P}_j = \vec{A} + \xi_j \vec{W}$ para $j = 0,1$. Se, ao contrário, os pontos dados forem ajustados pelos mínimos quadrados a um plano $\vec{W} \cdot \left(\vec{X} - \vec{A}\right) = 0$, o resultado será o mesmo, pois a normal unitária $\vec{W}$ do plano é exatamente a direção da reta.

4.3.2 Mínimo de Círculos Projetados de Área Mínima

Para cada direção unitária $\vec{W}$ tal que $\vec{W} \cdot (0,0,1) \geq 0$ ($\vec{W}$ encontra-se no hemisfério unitário superior), deve-se escolher os vetores unitários $\vec{U}$ e $\vec{V}$ tais que a matriz $R = \left[\vec{U}\ \vec{V}\ \vec{W}\right]$ seja ortonormal e tenha determinante igual a 1. Os pontos dados podem ser representados por $\vec{X}_i = \vec{A} + R\vec{Y}_i$, onde $\vec{Y}_i = (u_i, v_i, w_i)$. As projeções dos pontos sobre o plano $\vec{W} \cdot \vec{X} = 0$ são (u_i, v_i). O círculo de área mínima contendo esses pontos pode ser calculado como tendo raio $r = r\left(\vec{W}\right)$ e centro $\vec{C} = \vec{C}\left(\vec{W}\right)$. Calcula-se então o vetor $\vec{W}'$ que minimiza $r\left(\vec{W}\right)$. O raio da cápsula é $r\left(\vec{W}'\right)$ e $w_{\min}$ e $w_{\max}$ são os valores extremos de w_i. O segmento de reta da cápsula tem como extremidades $\vec{P}_0 = \vec{C}\left(\vec{W}'\right) + w_{\min}\vec{W}'$ e $\vec{P}_0 = \vec{C}\left(\vec{W}'\right) + w_{\max}\vec{W}'$.

4.4 Pastilhas

Uma pastilha também é uma extensão natural de uma esfera baseada em eqüidistância. É definida como o conjunto de todos os pontos que ficam a uma distância $r > 0$ de um retângulo com origem $\vec{P}$ e arestas com direções $\vec{E}_0$ e $\vec{E}_1$, onde $\vec{E}_0 \cdot \vec{E}_1 = 0$. Os quatro vértices do retângulo são $\vec{P}, \vec{P} + \vec{E}_0, \vec{P} + \vec{E}_1$ e $\vec{P} + \vec{E}_0 + \vec{E}_1$. Uma pastilha é uma caixa orientada que possui quatro meios cilindros acoplados às quatro faces menores e quatro quartos de esferas acoplados aos cantos, formando um sólido semelhante a uma almofada.

A seguir, é apresentado um algoritmo para envolver os pontos $\left\{\vec{X}_i\right\}_{i=0}^{n}$ por uma pastilha incluindo um ajuste através de mínimos quadrados.

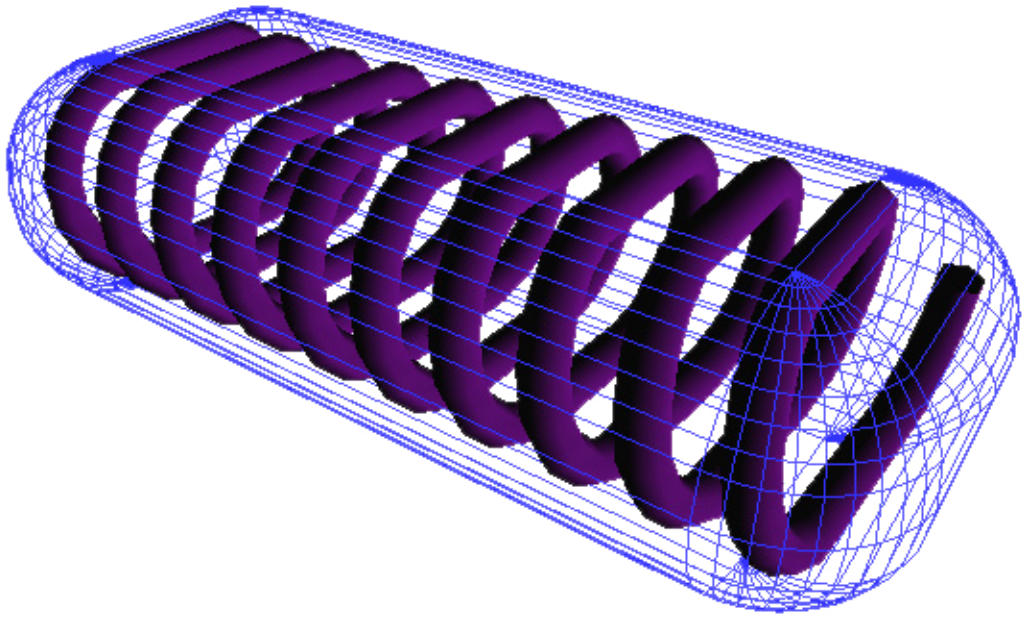

Figura 4.4 - Pastilha envolvente

4.4.1 Ajuste Através de uma Distribuição Gaussiana

É preciso calcular a média $\vec{A}$ dos pontos e a matriz de covariância, tal como no algoritmo de ajuste por uma caixa orientada. Sejam os auto-vetores unitários $\vec{U}$, $\vec{V}$ e $\vec{W}$ da matriz, assumimos que eles sejam rotulados de forma que $\vec{U}$ corresponda ao maior autovalor e $\vec{W}$ corresponda ao menor autovalor. Os pontos dados são representados por $\vec{X}_i = \vec{A} + u_i\vec{U}_i + v_i\vec{V}_i + w_i\vec{W}_i$. Sejam $w_{\min}$ e $w_{\max}$ os valores extremos de w_i, os pontos dados são envolvidos por dois planos $\vec{W}\cdot\left(\vec{X}-\vec{A}\right) = w_{\min}$ e $\vec{W}\cdot\left(\vec{X}-\vec{A}\right) = w_{\max}$. Ajusta-se o raio da pastilha para $r = \left(w_{\max} - w_{\min}\right)/2$ e a média para $\vec{A} \leftarrow \vec{A} + \left(\left(w_{\max} + w_{\min}\right)/2\right)\vec{W}$.

De modo análogo ao ajuste dos pontos por uma cápsula tridimensional, constrói-se uma cápsula bidimensional contendo os pares $\left(v_i, w_i\right)$. Precisamos calcular o maior β_0 possível, mas que seja tal que todos os pontos fiquem acima do semicírculo $w^2 + \left(v - \beta_0\right)^2 = r^2$ com $v \le \alpha_0$. O valor é calculado como sendo

$$\beta_0 = \min_i \left\{ v_i + \sqrt{r^2 - w_i^2} \right\}$$

onde $0 \le i \le n$. Similarmente, existe um valor β_1 que seja o menor possível, mas que ainda assim todos os pontos estejam abaixo do semicírculo $w^2 + (v - \beta_1)^2 = r^2$, com $v \ge \beta_1$. O valor é calculado como sendo

$$\beta_1 = \max_i \left\{ v_i - \sqrt{r^2 - w_i^2} \right\}.$$

Os extremos do segmento de reta da cápsula projetada determinam uma borda da pastilha, $\vec{E}_1 = (\beta_1 - \beta_0)\vec{V}$.

Repete-se o processo para os pares (u_i, w_i) para obter os valores

$$\alpha_0 = \min_i \left\{ u_i + \sqrt{r^2 - w_i^2} \right\}$$

e

$$\alpha_1 = \max_i \left\{ u_i - \sqrt{r^2 - w_i^2} \right\}.$$

Embora pareça que a outra borda da pastilha devesse ser $\vec{E}_0 = (\alpha_1 - \alpha_0)\vec{U}$, na verdade não é. Se os semicilindros fossem afixados dessa forma, os encontros das suas extremidades teriam um aspecto mitriforme e encerrariam mais espaço do que os quartos de esfera. É possível que alguns pontos dados estejam no espaço determinado pela sobreposição dos cilindros. A borda candidata $\vec{E}_0$ pode precisar ser aumentada para encerrar esses pontos.

Seja $\vec{K}_0 = \vec{A} + \alpha_0\vec{U} + \beta_0\vec{V}$ o ponto de um dos cantos do retângulo da pastilha corrente, suponhamos que $\vec{P} = \vec{A} + \alpha_p\vec{U} + \beta_p\vec{V} + \gamma_p\vec{W}$ seja um ponto fora do quarto de esfera com centro em $\vec{K}$. Para que isso seja verdade, $\left|\vec{P} - \vec{K}_0\right| > r$. O canto deve ser ajustado para $\vec{K}_1 = \vec{A} + \alpha_1\vec{U} + \beta_1\vec{V}$ de modo que $\left|\vec{P} - \vec{K}_1\right| = r$. Há dois graus de liberdade para o ajuste. Um grau é eliminado exigindo-se que $(\alpha_1, \beta_1) = t(\alpha_0, \beta_0) + (1 - t)(\alpha_p, \beta_p)$. Substituído na equação anterior de distância, isso nos conduz a uma equação quadrática em *t* que pode ser resolvida por

$$t = \frac{r^2 - \gamma_p^2}{(\alpha_p - \alpha_0)^2 + (\beta_p - \beta_0)^2}.$$

O ajuste no canto do retângulo não afeta as relações anteriores de envolvimento. Assim, podemos fazer uma iteração com a lista de pontos de entrada e ajustar os cantos conforme necessário.

Após os ajustes, o retângulo da pastilha é $[\alpha_0, \alpha_1] \times [\beta_0, \beta_1]$. A origem da pastilha é tomada com sendo $\vec{A} + \alpha_0 \vec{U} + \beta_0 \vec{V}$ e suas bordas como $\vec{E}_0 = (\alpha_1 - \alpha_0)\vec{U}$ e $\vec{E}_1 = (\beta_1 - \beta_0)\vec{V}$.

4.5 Cilindros

Um *cilindro infinito* é o conjunto de todos os pontos que estão a uma distância *r* de uma linha $\vec{P} + t\vec{D}$, onde $t \in \mathbb{R}$ e $\vec{D}$ tem comprimento unitário. Um *cilindro finito* é um subconjunto de um *cilindro infinito*, onde $|t| \leq h/2$ para uma dada altura *h*. Daqui em diante, nos referiremos aos cilindros finitos simplesmente como "cilindros".

A seguir, apresentamos dois algoritmos para envolver os pontos $\{\vec{X}_i\}_{i=0}^{n}$. Os pontos são ajustados por uma linha usando o algoritmo dos mínimos quadrados. Seja a reta $\vec{A} + t\vec{W}$, onde $\vec{W}$ é um vetor unitário e $\vec{A}$ é a média dos pontos dados, selecionam-se os vetores $\vec{U}$ e $\vec{V}$ de modo que a matriz $R = [\vec{U}\ \vec{V}\ \vec{W}]$ seja ortonormal e tenha determinante unitário. Os pontos dados podem ser representados por $\vec{X}_i = \vec{A} + R\vec{Y}_i$, onde $\vec{Y}_i = (u_i, v_i, w_i)$.

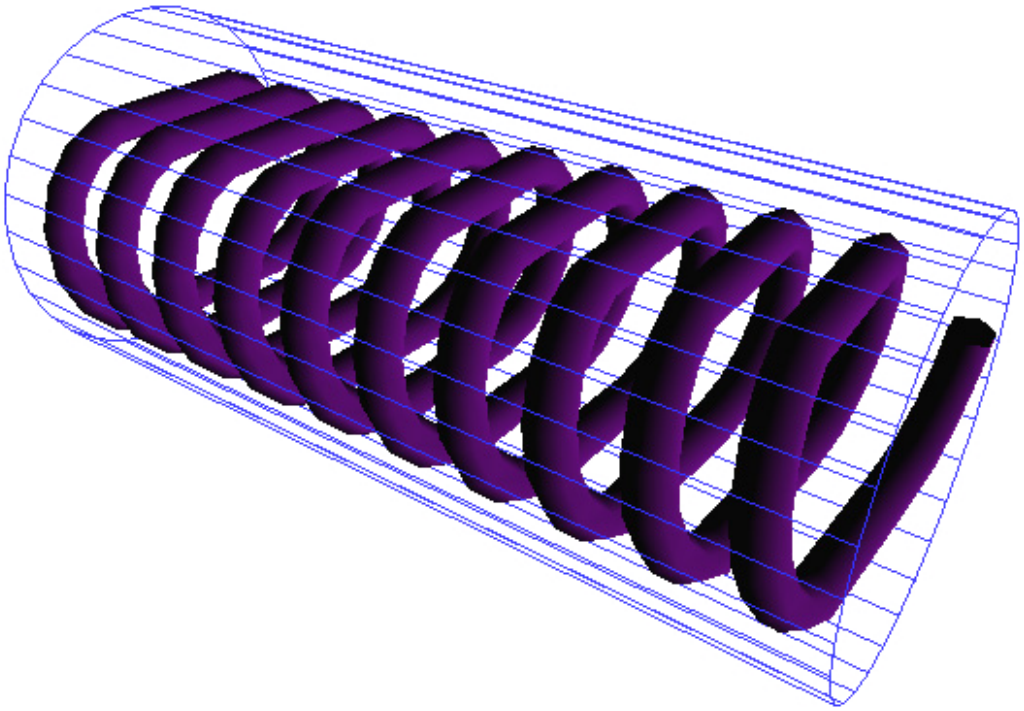

Figura 4.5 - Cilindro envolvente

4.5.1 Reta Ajustada Pelos Mínimos Quadrados Contendo o Eixo

O raio do cilindro é $r = \max_i \left\{ \sqrt{u_i^2 + v_i^2} \right\}$ e sua altura é $h = w_{\max} - w_{\min}$, onde $w_{\min}$ e $w_{\max}$ são os valores extremos de w_i. Para ficar de acordo com a definição de cilindro finito, a reta precisa ter seu vetor de translação ajustado. A nova translação é

$$\vec{A}' = \vec{A} + \frac{w_{\max} - w_{\min}}{2} \vec{W}.$$

A reta é $\vec{A}' + t\vec{W}$ e o cilindro é restringido por $|t| \le h/2$.

4.5.2 Reta Ajustada Pelos Mínimos Quadrados Reposicionada no Centro de Área Mínima

O círculo de área mínima contendo os valores de $(u_i + v_i)$ é calculado e possui centro (u', v') e raio *r*. A reta obtida pelos mínimos quadrados é deslocada para conter o centro do círculo, $\vec{A}' = \vec{A} + u'\vec{U} + v'\vec{V}$.

O raio do cilindro é *r* e o algoritmo da Seção anterior é aplicado para calcular *h*. Esse algoritmo também desloca a reta na direção de $\vec{W}$ para $\vec{A}''+t\vec{W}$, onde

$$\vec{A}''=\vec{A}'+\frac{w_{\max}-w_{\min}}{2}\vec{W}.$$

4.6 Elipsóides

Um elipsóide alinhado com os eixos tem a forma padrão

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$$

com centro em (0,0,0) e semi-eixos com comprimentos $a>0$, $b>0$ e $c>0$. As direções dos eixos da elipse são (1,0,0), (0,1,0) e (0,0,1).

Dado um sistema de coordenadas com centro $\vec{C}$ e eixos com direções ortonormais $\vec{U}_i$, com $0\le i\le 2$, o elipsóide com esse centro e eixos é

$$\left(\vec{X}-\vec{C}\right)^T R^T DR\left(\vec{X}-\vec{C}\right)=1,$$

onde $R=\left[\vec{U}_0\,\vec{U}_1\,\vec{U}_2\right]$ é a matriz de rotação, $D=\text{diag}\left\{1/d_0^2,1/d_1^2,1/d_2^2\right\}$ é uma matriz diagonal com elementos positivos que são os quadrados dos comprimentos dos semi-eixos e $\vec{X}$ é a variável algébrica para a equação. Uma equação do tipo $\left(\vec{X}-\vec{C}\right)^T M\left(\vec{X}-\vec{C}\right)=1$, onde *M* é uma matriz positiva definida, também representa um elipsóide. Os eixos e os comprimentos dos semi-eixos são obtidos a partir dos autovetores e autovalores de $M=R^T DR$.

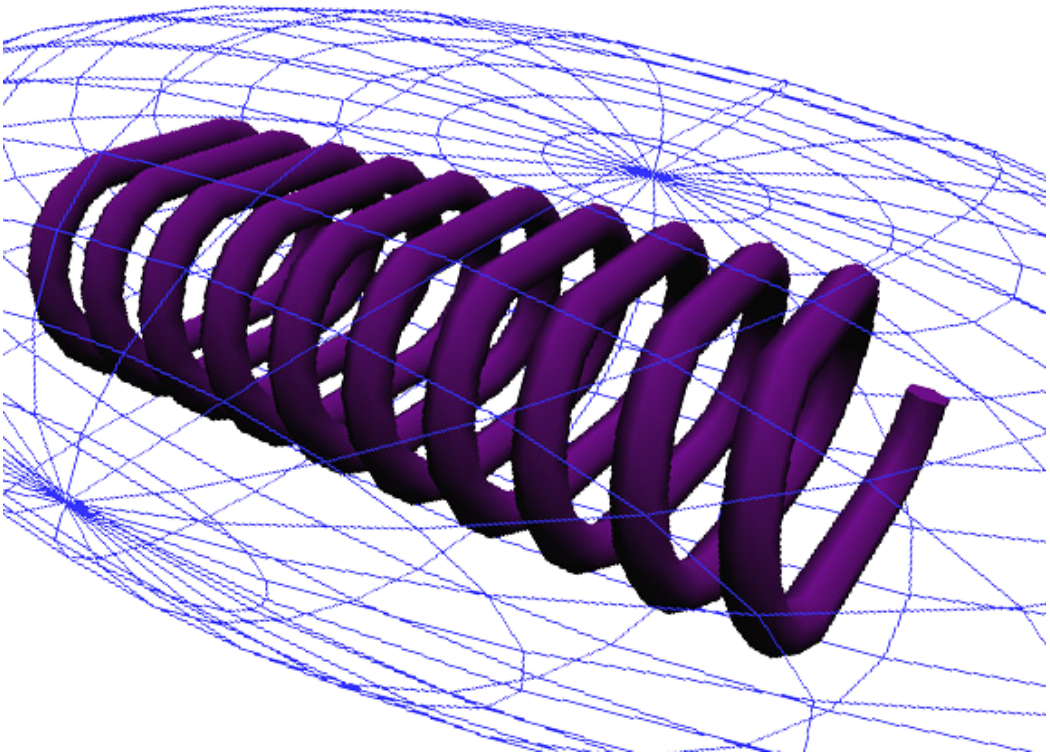

Figura 4.6 - Elipsóide envolvente

4.6.1 Elipsóide Alinhado com os Eixos

Dado um conjunto de pontos $\{\vec{V}_i\}_{i=0}^{n}$, um modo simples de envolvê-los com um elipsóide é primeiro gerar uma caixa alinhada com os eixos contendo os pontos e então estabelecer as razões dos comprimentos do semi-eixos. Sejam $\vec{p}_{\min}$ e $\vec{p}_{\max}$ os vetores que contêm os valores mínimos e máximos dos componentes, o centro do elipsóide é $\vec{C} = (\vec{p}_{\max} + \vec{p}_{\min})/2$. Os comprimentos dos semi-eixos são os componentes de $\lambda(\vec{p}_{\max} + \vec{p}_{\min})/2 = \lambda(\delta_0, \delta_1, \delta_2)$, onde $\lambda > 0$ deve ser determinado. Seja $D = \text{diag}\{1/(\lambda\delta_0)^2, 1/(\lambda\delta_1)^2, 1/(\lambda\delta_2)^2\}$, o elipsóide é $(\vec{X} - \vec{C})^T E(\vec{X} - \vec{C}) = 1$, onde $E = D / \max_i \{(\vec{V} - \vec{C})^T D(\vec{V} - \vec{C})\}$.

4.6.2 Ajuste de Pontos Através da Distribuição Gaussiana

Este método é semelhante ao usado para o ajuste de pontos com uma caixa orientada. A média dos pontos é utilizada como centro do elipsóide e os autovetores

da matriz de covariância são usados como eixos. Os autovalores são empregados da mesma forma que o vetor $(\delta_0, \delta_1, \delta_2)$ no ajuste com o elipsóide alinhado com os eixos. O elipsóide é $\left(\vec{X}-\vec{C}\right)^T E\left(\vec{X}-\vec{C}\right)=1$, onde $E=\left(R^T DR\right)/\max_i\left\{\left(\vec{V}-\vec{C}\right)^T R^T DR\left(\vec{V}-\vec{C}\right)\right\}$.

4.6.3 Elipsóide de Volume Mínimo

Enquanto a teoria para o ajuste de pontos ao elipsóide de volume mínimo foi desenvolvida empregando-se técnicas de distribuição aleatória lineares, uma implementação é extremamente difícil, porque requer o tratamento de casos especiais para conjuntos de até nove pontos (algoritmos para o cálculo de esfera de volume mínimo requerem o tratamento de casos especiais de até quatro pontos). Uma alternativa é o uso de uma minimização numérica restrita, algo que é desafiador, mas não impossível de ser implementado. De qualquer forma, o cálculo rápido de elipsóides de volume mínimo não é possível neste momento para aplicações em tempo real.

5 Descarte Usando Volumes Envolventes

A cadeia de restituição gerencia pelo menos uma câmera cujos parâmetros permitem determinar o volume de visão, que tem a forma de um tronco de pirâmide de base retangular. As entidades potencialmente visíveis estão localizadas no interior desse volume de visão. No instante em que uma fotografia da cena virtual precisa ser gerada, o motor de restituição deve percorrer o grafo da cena, determinar quais entidades estão visíveis e então restituir essas entidades.

O caminhamento do grafo começa pelo nó raiz, que representa a cena como um todo. Se o nó raiz for determinado como visível, então os seus nós filhos são percorridos recursivamente até que sejam descartados ou aceitos. O caminhamento pode ser interrompido se uma destas três condições for atingida: (I) o nó está totalmente fora do volume de visão, caso em que nenhuma das entidades filhas será restituída; (II) o nó está totalmente dentro do volume de visão, sendo aceito pois as entidades filhas são potencialmente visíveis; e (III) uma folha do grafo é atingida. Se um nó não puder ser totalmente aceito ou rejeitado, ele será restituído, pois pelo menos uma parte dele é potencialmente visível. Resumidamente, os filhos de um nó só são percorridos se o nó for cortado por pelo menos um dos planos que definem o volume de visão.

O volume de visão é definido por seis planos: próximo, distante, esquerdo, direito, superior e inferior. Cada nó não precisa ser necessariamente testado contra todos os planos. Suponhamos que um nó tenha sido aceito como visível segundo todos os planos com exceção do esquerdo. Nesse caso, seus filhos também devem ser aceitos segundo os mesmos planos, restando testá-los somente contra o plano esquerdo. Em virtude disso, serão apresentados a seguir dois algoritmos para cada tipo de volume envolvente: o primeiro diz se um volume envolvente está ou não totalmente do lado negativo de um plano dado e o segundo amplia esse resultado dizendo também se o volume está totalmente do lado positivo do plano. Enquanto o primeiro pode realizar menos cálculos, o segundo permite que os nós filhos requeiram uma quantidade menor de testes para determinar se deverão ser aceitos, uma vez que, se o pai foi aceito segundo um plano, o filho também será.

As transformações sofridas pelas entidades devem ser refletidas pelos seus volumes envolventes. Se uma entidade sofrer uma translação ou rotação, o seu volume deve sofrer a mesma translação ou rotação. O mesmo se aplica aos demais nós do grafo de cena que envolvem várias entidades. Se utilizarmos uma matriz para representar essas transformações, as folhas do grafo (que são as entidades) possuirão uma matriz de transformação associada que acumula todas as transformações dos seus nós ancestrais. Se estivermos utilizando uma API gráfica como o OpenGL, ela realizará as atualizações necessárias na sua matriz de transformações. Se a API estiver tirando proveito de recursos de aceleração proporcionados pela placa de vídeo, por exemplo, podem haver diferenças entre os valores da matriz de transformação mantida pela API e a mantida pela aplicação.

A consistência entre as matrizes pode ser obtida forçando-se o uso de apenas uma delas ou fazendo comparações periódicas e compatibilizando-as em função de alguma medida de tolerância. A primeira opção envolve transferir a matriz calculada pela aplicação para a GPU (processador da placa gráfica), ressaltada, no caso do OpenGL, como operação dispendiosa por Woo [WOOM1999], ou copiá-la via API para a aplicação, o que, em certas implementações, tem também custos muito altos pois força a sincronização entre a CPU e a GPU várias vezes durante a restituição de um quadro. Na implementação do programa de testes que será descrito no Capítulo 7, foram mantidas duas matrizes separadas, verificando-se se as diferenças entre elas se mantinham dentro de uma faixa de tolerância.

A partir daí temos duas opções: realizar os testes no espaço de coordenadas global ou no espaço de coordenadas da câmera. Se optarmos pelo espaço de coordenadas global, temos de calcular as equações que definem os planos que constituem o volume de visão a cada quadro para testar os volumes envolventes contra esses planos. Se optarmos pelo espaço de coordenadas da câmera, as equações desses planos não se alteram com o movimento da câmera, porém precisamos transformar os pontos e vetores que definem os volumes envolventes para esse sistema de coordenadas.

Caso esteja sendo usada a matriz mantida pela API para transformar os volumes envolventes e os testes estiverem sendo realizados no sistema de coordenadas global, devemos lembrar que essa matriz acumula as rotações e

translações sofridas pela própria câmera ou a transformação do sistema de coordenadas global para o sistema de coordenadas da câmera, e que deve ser desacumulada.

5.1 Descarte Através de Esferas

Seja a esfera envolvente definida pelo centro $\vec{C}$ e o raio r no sistema de coordenadas da câmera e seja um plano do volume de visão especificado por $\vec{N} \cdot \vec{X} = d$, onde $\vec{N}$ é um vetor unitário que aponta para o interior do volume de visão, a esfera envolvente não intercepta o volume de visão quando a distância de $\vec{C}$ ao plano for maior do que o raio. Um nó do grafo da cena é completamente descartado se sua esfera envolvente satisfizer

$$\vec{N} \cdot \vec{C} - d < -r$$

para cada um dos planos do volume de visão. O lado esquerdo da desigualdade é a distância de $\vec{C}$ ao plano. Se for positiva, significa que $\vec{C}$ está à frente do plano (semi-espaço na direção positiva de $\vec{N}$). Doravante, a expressão "lado positivo do plano" terá essa acepção. O lado direito da inequação é negativo e indica que, para a esfera ser descartada, $\vec{C}$ deve estar fora do volume de visão e deve estar afastado do plano a uma distância de, no mínimo, um raio da esfera. O teste requer 3 multiplicações e 3 adições. O pseudo-código é

```
bool DescartarEsferaContraPlano(TEsfera *esfera, TPlano *plano){
     retorne ProdutoEscalar(plano->n, esfera->c) - plano->d <= -esfera->r;
}
```

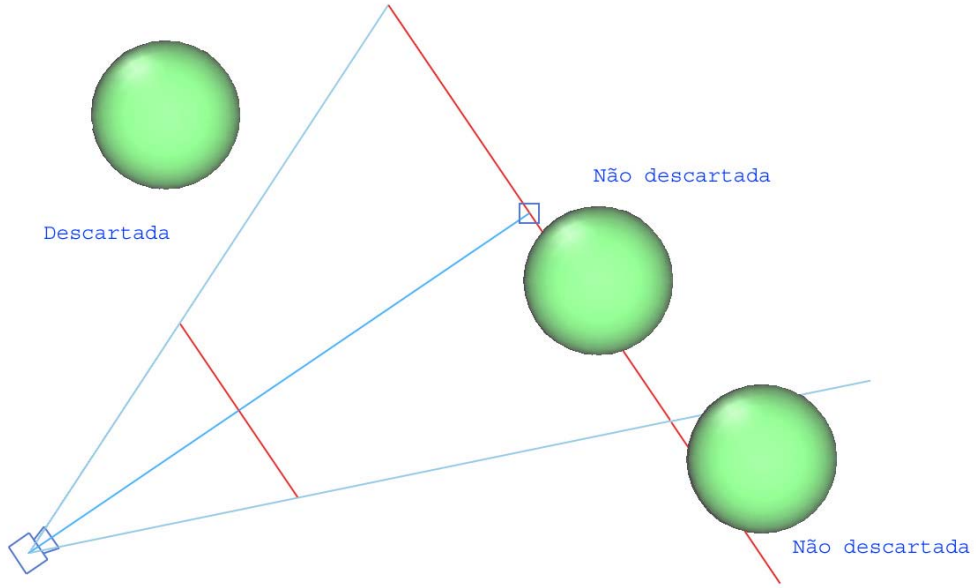

Figura 5.1 - Esferas descartadas e não descartadas

O algoritmo acima retorna verdadeiro ou falso, indicando se uma esfera deve ou não ser descartada por estar totalmente do lado negativo do plano dado. Se desejarmos saber adicionalmente se a esfera está totalmente do lado positivo do plano, o algoritmo precisa ser estendido. Tal informação é útil caso a esfera não possa ser aceita ou descartada contra os seis planos do volume de visão e, assim sendo, os nós filhos necessitem ser testados. Tendo essa informação adicional, os nós filhos só são testados contra os planos que interceptaram nó pai. O pseudo-código para este algoritmo é

```
bool DescartarEsferaContraPlano (TEsfera *esfera,
                                 bool *totDentro,TPlano *plano){

    *totDentro = falso;
    double dist = ProdutoEscalar(plano->n, esfera->c) - plano->d;
    se(dist <= - esfera->raio){
        retorne verdadeiro;
    }
    se(dist > esfera->raio)
        *totDentro = verdadeiro;
    retorne falso;
}
```

onde `totDentro` diz se a esfera está totalmente do lado positivo do plano.

É possível que uma esfera envolvente esteja totalmente fora do volume de visão mesmo que todos os seis testes de descarte falhem. A Figura 5.1 mostra um exemplo de uma entidade que é descartada pelos testes, além de exemplos de entidades que não são descartadas, uma entidade cuja esfera envolvente intercepta o volume de visão e uma entidade cuja esfera envolvente não intercepta o volume de visão. Nos dois últimos casos, as entidades precisarão ser tratadas no processo de cerceamento. Uma alternativa seria efetuar o cálculo exato da distância da esfera envolvente ao volume de visão, a um custo bem maior.

Volumes envolventes melhor ajustados podem conduzir à rejeição da entidade quando uma esfera envolvente não o faz, economizando tempo de processamento. Contudo, a esfera tem a vantagem de não precisar ser rotada para acompanhar os movimentos de corpo rígido da entidade, apenas transladada ou escalada. Isso leva a uma barganha entre o consumo de tempo maior para corrigir a orientação do volume envolvente e a economia de tempo, evitando processar entidades que foram mais precisamente descartadas. Contudo, se estiver sendo utilizada a matriz de transformação de uma API para transformar as entidades, como sugerido no início deste capítulo, esta vantagem torna-se sem efeito.

5.2 Descarte Através de Caixas Orientadas

Uma caixa orientada estará fora do volume de visão se todos os seus vértices estiverem do lado de fora do espaço delimitado pelos planos do volume de visão. O algoritmo óbvio consiste em testar se todos os oito vértices estão do lado negativo desses planos. Tal algoritmo requer oito comparações da forma $\vec{N} \cdot \vec{V} < d$. Os vértices têm a forma

$$\vec{V} = \vec{C} + \sigma_0 a_0 \vec{A}_0 + \sigma_1 a_1 \vec{A}_1 + \sigma_2 a_2 \vec{A}_2 ,$$

onde $|\sigma_i| = 1$ para todo i (oito possibilidades, duas para cada σ_i). Cada teste requer o cálculo das distâncias orientadas

$$\vec{N}\cdot\vec{V}-d=\left(\vec{N}\cdot\vec{C}-d\right)+\sigma_0 a_0\vec{N}\cdot\vec{A}_0+\sigma_1 a_1\vec{N}\cdot\vec{A}_1+\sigma_2 a_2\vec{N}\cdot\vec{A}_2\,.$$

Os quatro produtos escalares são calculados uma vez, sendo que cada produto escalar requer três multiplicações e duas somas. Cada teste exige mais três multiplicações e quatro somas (as multiplicações por σ_i não são levadas em consideração). Assim, os oito testes exigem 36 multiplicações e 40 somas.

Uma forma mais rápida de fazer esse teste é projetar a caixa e o plano contra a qual é testada sobre a reta $\vec{C}+s\vec{N}$, sendo $\vec{C}$ o centro da caixa, $\vec{N}$ a normal ao plano e *s* a variável algébrica. A simetria implícita na definição da caixa leva a um intervalo de projeção $\left[\vec{C}-r\vec{N},\vec{C}-r\vec{N}\right]$. O intervalo é centrado em $\vec{C}$ e tem raio

$$r=a_0\left|\vec{N}\cdot\vec{A}_0\right|+a_1\left|\vec{N}\cdot\vec{A}_1\right|+a_2\left|\vec{N}\cdot\vec{A}_2\right|.$$

O plano do volume de visão se projeta sobre um único ponto

$$\vec{P}=\vec{C}+\left(d-\vec{N}\cdot\vec{C}\right)\vec{N}\,.$$

A caixa estará do lado de fora do plano se o segmento projetado também estiver, caso em que $\vec{N}\cdot\vec{C}-d<-r$. O teste é idêntico ao da esfera, exceto que *r* é conhecido para a esfera, mas precisa ser calculado para cada teste de uma caixa envolvente orientada. O teste requer quatro produtos escalares, três multiplicações e três adições, totalizando 15 multiplicações e 11 adições. O pseudo-código é

```
bool DescartarCaixaOrientadaContraPlano(TCaixa *caixa, TPlano *plano){
    r = caixa.a0*|ProdutoEscalar(plano->N,caixa->A0)| +
        caixa.a1*|ProdutoEscalar(plano->N,caixa->A1)| +
        caixa.a2*|ProdutoEscalar(plano->N,caixa->A2)|;
retorne ProdutoEscalar(plano->N,caixa->C) - plano->d < -caixa->r;
}
```

Analogamente ao descarte da esfera contra o plano, podemos alterar o algoritmo acima para verificar se a caixa orientada está totalmente do lado positivo do plano. O pseudo-código é

```
bool DescartarCaixaOrientadaContraPlano(TCaixa *caixa, bool *totDentro,
                                        TPlano *plano){
     *totDentro = falso;
```

```
     //Cálculo da distância orientada do centro da esfera ao plano
    r = caixa.a0*|ProdutoEscalar(plano->n,caixa->A0)| +
        caixa.a1*|ProdutoEscalar(plano->n,caixa->A1)| +
        caixa.a2*|ProdutoEscalar(plano->n,caixa->A2)|;
    dist = ProdutoEscalar(plano->n, caixa->c) - plano->d;
    se(dist <= -r)
        retorne verdadeiro;
    se(dist > r)
        *totDentro = verdadeiro;
    retorne falso;
}
```

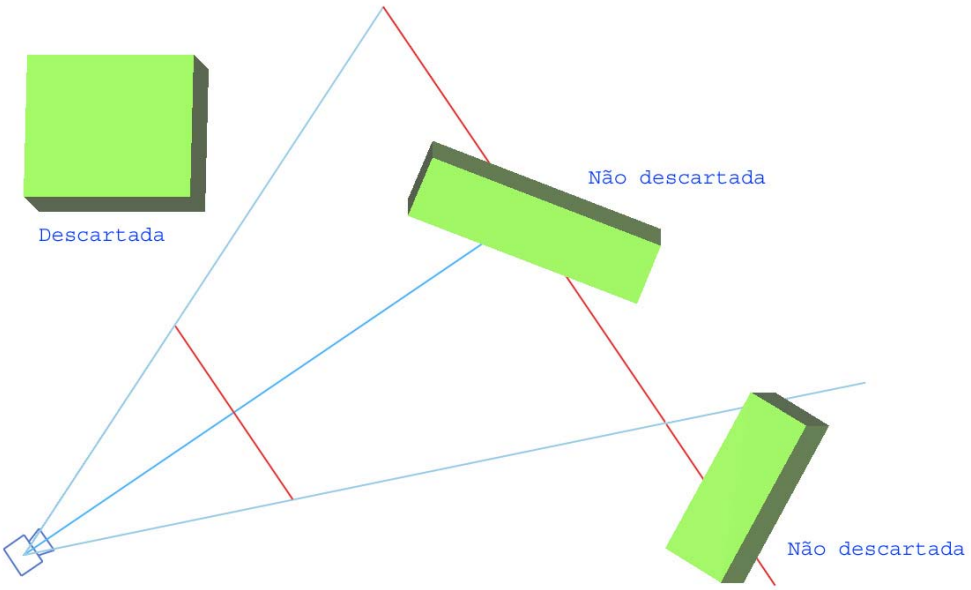

Figura 5.2 - Caixas descartadas e não descartadas

Do mesmo modo que a esfera, a caixa orientada pode não ser descartada quando testada contra cada plano do volume de visão individualmente, mesmo que esteja completamente fora do volume, conforme ilustrado na Figura 5.2.

5.3 Descarte Através de Cápsulas

Uma cápsula é definida por um raio $r > 0$ e um segmento de reta parametrizado $\vec{P} + t\vec{D}$, onde $\vec{D} \neq 0$ e $t \in [0,1]$. As distâncias orientadas do plano às

extremidades são $\delta_0 = \vec{N} \cdot \vec{P} - d$ e $\delta_1 = \vec{N} \cdot \left(\vec{P} + \vec{D}\right) - d$. Se $\delta_0 \geq 0$ ou $\delta_1 \geq 0$, então a cápsula não será descartada, pois estará interceptando o plano do volume de visão ou estará dentro do volume. Caso contrário, ambas as distâncias orientadas serão negativas. Se $\vec{N} \cdot \vec{D} \leq 0$, então a extremidade $\vec{P}$ estará mais próxima, numa distância orientada, do plano considerado do que o ponto $\vec{P} + \vec{D}$. A distância entre $\vec{P}$ e o plano é calculada e comparado comparada com o raio da cápsula. Se $\vec{N} \cdot \vec{P} - d \leq -r$, então a cápsula estará fora do plano do volume de visão e será descartada; caso contrário não será. Além disso, se $\vec{N} \cdot \left(\vec{P} + \vec{D}\right) - d \leq -r$, a cápsula será descartada; caso contrário não será. O pseudecódigo para o algoritmo de descarte é dado abaixo. O resultado booliano será `verdadeiro` se e somente se a cápsula for descartada.

```
bool DescartarCapsulaContraPlano(TCapsula *capsula, TPlano *plano){
   dist0 = ProdutoEscalar(plano->N,capsula->P) - plano->d;
   se (dist0<0){
      dist1 = dist0 + ProdutoEscalar(plano->N,capsula->D);
      se (dist1<0){
         se (dist0 <= dist1)    //P0 está mais próximo do plano
            retorne dist0<= -capsula->r;
         }
         senão{          //P1 está mais próximo do plano
            retorne dist1 <= -capsula->r;
         }
      }
   }
   retorne falso;
}
```

Analogamente ao descarte dos volumes vistos anteriormente, podemos alterar o algoritmo acima para verificar se a cápsula está totalmente do lado positivo do plano. O pseudo-código é

```
bool DescartarCapsulaContraPlano(TCapsula *capsula, bool *totDentro,
                                 TPlano *plano){
    *totDentro = falso;
    double dist0 = ProdutoEscalar(plano->n, origem) - plano->d;
    double dist1 = ProdutoEscalar(plano->n, extremidade) - umPlano->d;
    se(dist0 <= -raio){
        se(dist1 <= -raio)
            retorne verdadeiro;
    }
    se(dist0 > raio && dist1 > raio)
```

```
        *totDentro = verdadeiro;
    retorne falso;
}
```

5.4 Descarte Através de Pastilhas

Uma pastilha é definida por um raio $r > 0$ e por um retângulo parametrizado $\vec{P} + s\vec{E}_0 + t\vec{E}_1$, onde $\vec{E}_0 \neq \vec{0}$, $\vec{E}_1 \neq \vec{0}$, $\vec{E}_0 \cdot \vec{E}_1 = \vec{0}$ e $(s,t) \in [0,1]^2$. Os quatro cantos do retângulo são $\vec{P}_{00} = \vec{P}$, $\vec{P}_{01} = \vec{P} + \vec{E}_0$, $\vec{P}_{10} = \vec{P} + \vec{E}_1$ e $\vec{P}_{11} = \vec{P} + \vec{E}_0 + \vec{E}_1$. As distâncias orientadas são $\delta_{ij} = \vec{N} \cdot \vec{P}_{ij} - d$. Se qualquer das distâncias orientadas for maior do que -*r*, então a cápsula ou intercepta o plano ou está do lado interno do volume de visão e não será descartada. Para reduzir o número de operações, os vértices $\vec{P}_{ij}$, $i = 0,1$ e $j = 0,1$ do retângulo podem ser pré-calculados e armazenados. A distância de cada um dos vértices do retângulo da pastilha ao plano do volume de visão é determinada e comparada com o raio da pastilha para verificar se há interseção. Se pelo menos uma dessas distâncias for maior ou igual a -*r*, então há interseção entre a pastilha e o volume de visão. O pseudo-código para o algoritmo de descarte é

```
bool DescartarPastilhaContraPlano(TPastilha *pastilha, TPlano *plano){
    dist00 = ProdutoEscalar(plano->n, pastilha->p00) - plano->d;
    se (dist00 < - pastilha->raio) {
        dist10 = ProdutoEscalar(plano->n,pastilha->p10) - plano->d;
        se (dist10 < - pastilha->raio) {
            dist01 = ProdutoEscalar(plano->n,pastilha->p01) - plano->d;
            se (dist01 < - pastilha->raio) {
                dist11 = ProdutoEscalar(plano->n,pastilha->p11) - plano->d;
                se (dist11 < - pastilha->raio)
                    retorne verdadeiro;
            }
        }
    }
    retorne falso;
}
```

Analogamente ao descarte dos volumes vistos anteriormente, podemos alterar o algoritmo acima para verificar se a pastilha está totalmente do lado positivo do plano. O pseudo-código é

```
bool DescartarPastilhaContraPlano(TPastilha *pastilha, bool *totDentro,
                                  TPlano *umPlano){
    *umTotDentro = falso;
    double dist0 = ProdutoEscalar(umPlano->n, origemCam) - umPlano->d;
    double dist1 = ProdutoEscalar(umPlano->n, ponto1Cam) - umPlano->d;
    double dist2 = ProdutoEscalar(umPlano->n, ponto2Cam) - umPlano->d;
    double dist3 = ProdutoEscalar(umPlano->n, ponto3Cam) - umPlano->d;
    se ( dist0 <= -pastilha->raio || dist1 <= -pastilha->raio ||
          dist2 <= -pastilha->raio || dist3 <= -pastilha->raio)
        retorne verdadeiro;
    se ( dist0 > pastilha->raio && dist1 > pastilha->raio &&
          dist2 > pastilha->raio && dist3 > pastilha->raio)
        *umTotDentro = verdadeiro;
    retorne falso;
}
```

5.5 Descarte Através de Cilindros

Um cilindro é definido por um raio $r > 0$ e um segmento de reta parametrizado $\vec{P} + t\vec{D}$, onde $\vec{D} \neq 0$, $\left|\vec{D}\right| = 1$ e $t \in [0, h]$. As distâncias orientadas do plano às extremidades são $\delta_0 = \vec{N} \cdot \vec{P} - d$ e $\delta_1 = \vec{N} \cdot \left(\vec{P} + \vec{D}\right) - d$ e r' é o *raio efetivo* do cilindro, definido como metade da projeção da base do cilindro sobre uma reta normal ao plano do volume de visão dado por $r' = r \operatorname{sen} \alpha$, onde α é o ângulo formado entre o eixo do cilindro e o vetor normal ao plano do volume de visão. Lembrando que $\operatorname{sen}^2 \alpha + \cos^2 \alpha = 1$ temos que $r' = r\sqrt{1 - \cos^2 \alpha}$. O $\cos \alpha$ é dado pelo produto escalar dos vetores unitários $\vec{N}$ e $\vec{D}$. Se $\delta_0 \geq -r'$ ou $\delta_1 \geq -r'$, então o cilindro não será descartado, pois estará interceptando o plano do volume de visão ou estará dentro do volume. Se $\vec{N} \cdot \vec{P} - d \leq -r'$ e $\vec{N} \cdot \left(\vec{P} + \vec{D}\right) - d \leq -r'$, o cilindro estará fora do plano do volume de visão e será descartado. O pseudecódigo para o algoritmo de descarte é dado abaixo. O resultado booleano será `verdadeiro` se e somente se a cápsula for descartada.

```
bool DescartarCilindroContraPlano(TCilindro *cilindro, TPlano *plano){
    double cosAlfa = plano->n * cilindro->eixo;
    double raioEfetivo = RaizQuadrada(1-(cosAlfa * cosAlfa))*raio;
    double dist0 = ProdutoEscalar(plano->n,cilindro->p) - plano->d;
     se ( dist0 < -raioEfetivo){
        double dist1 = ProdutoEscalar(plano->n,cilindro->q) - plano->d;
        se ( dist1 < -raioEfetivo)
            retorne verdadeiro;
```

```
    }
    retorne falso;
}
```

Analogamente ao descarte dos volumes vistos anteriormente, podemos alterar o algoritmo acima para verificar se o cilindro está totalmente do lado positivo do plano. O pseudo-código é

```
bool DescartarPlano(TCilindro *cilindro, bool *totDentro,TPlano *umPlano){
    *umTotDentro=falso;
    double cosAlfa = ProdutoEscalar(plano->n, cilindro->eixo);
    double raioEfetivo = RaizQuadrada(1-(cosAlfa * cosAlfa))*raio;
    double dist0 = ProdutoEscalar(plano->n,cilindro->p) - plano->d;
    double dist1 = ProdutoEscalar(plano->n,cilindro->q) - plano->d;
    se ( dist0 <= -raioEfetivo || dist1 <= -raioEfetivo){
        retorne verdadeiro;
    }
    se ( dist0 > -raioEfetivo && dist1 > -raioEfetivo){
        *umTotDentro = verdadeiro;
    }
    retorne falso;
}
```

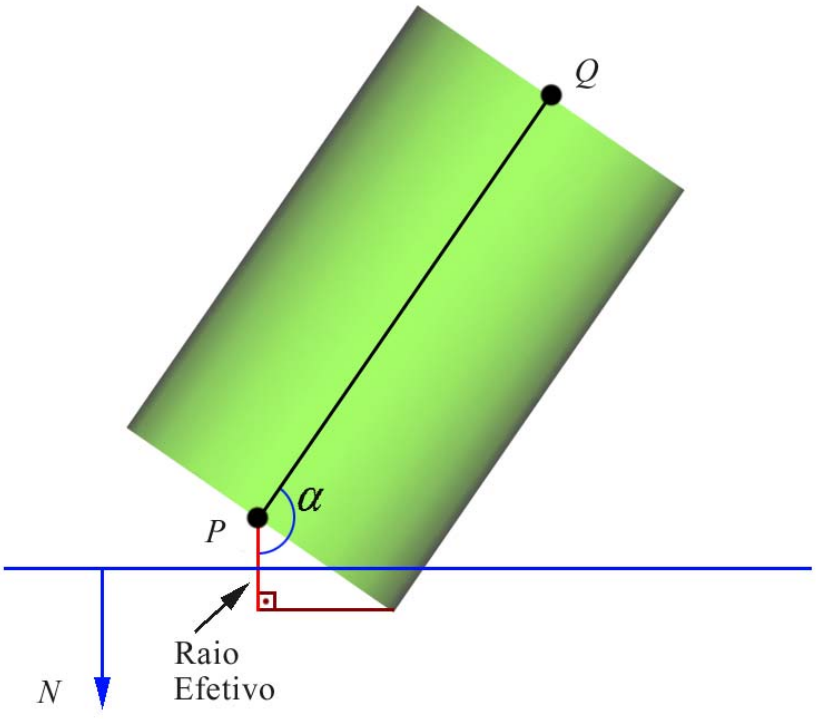

Figura 5.3 - Projeção do cilindro e de um plano do volume de visão

5.6 Descarte Através de Elipsóides

Um elipsóide é representado pela equação quadrática $Q(\vec{X}) = (\vec{X} - \vec{C})^T M(\vec{X} - \vec{C}) = 1$, onde $\vec{C}$ é o centro do elipsóide, M é uma matriz positiva definida e $\vec{X}$ é um ponto qualquer do elipsóide. Um elipsóide estará do lado de fora do volume de visão em relação a um plano sempre que a projeção do elipsóide sobre a reta $\vec{C} + s\vec{N}$ estiver fora do elipsóide em relação ao mesmo plano. O intervalo projetado é $[-r, r]$. O elipsóide é descartado sempre que $\vec{N} \cdot \vec{C} - d \leq -r$.

O cálculo de *r* é feito como se segue. Os pontos $\vec{X}$ que se projetam sobre as extremidades do intervalo devem ocorrer onde as normais ao elipsóide são paralelas a $\vec{N}$. O gradiente de $Q(\vec{X})$ é uma direção normal para o ponto, $\vec{\nabla} Q = 2M(\vec{X} - \vec{C})$. Assim, $\vec{X}$ deve ser uma solução para $M(\vec{X} - \vec{C}) = \lambda \vec{N}$ para algum escalar λ. Invertendo M e multiplicando, obtemos $\vec{X} - \vec{C} = \lambda M^{-1} \vec{N}$. Substituindo esta expressão na equação quadrática, obtemos $1 = \lambda^2 (M^{-1}\vec{N})^T M(M^{-1}\vec{N}) = \lambda^2 \vec{N}^T M^{-1} \vec{N}$. Finalmente, $r = \vec{N} \cdot (\vec{X} - \vec{C}) = \lambda \vec{N}^T M^{-1} \vec{N}$, então $r = \sqrt{\vec{N}^T M^{-1} \vec{N}}$. O pseudocódigo é

```
bool DescartarElipsóideContraPlano(TElipsóide *elipsóide, TPlano *plano){
    dist = ProdutoEscalar(plano->n,elipsóide->c) - plano->d;
    se ( dist < 0 ){
        r2 = ProdutoEscalar(plano->n, elipsóide->mInversa * plano->n);
        retorne dist*dist >= r2;
    }
    retorne falso;
}
```

Analogamente ao descarte dos volumes vistos anteriormente, podemos alterar o algoritmo acima para verificar se o cilindro está totalmente do lado positivo do plano. O pseudo-código é

```
bool DescartarElipsóideContraPlano(TElipsóide *elipsóide, bool *totDentro,
                                   TPlano *plano){
    *umTotDentro=falso;
    dist = ProdutoEscalar(plano->n,elipsóide->c) - plano->d;
```

```
    r2 = ProdutoEscalar(plano->n, elipsóide->mInversa * plano->n);
    se ( dist*dist >= r2 ){
        retorne verdadeiro;
    }
    se ( dist*dist < r2 ){
        *umTotDentro = verdadeiro;
    }
    retorne falso;
}
```

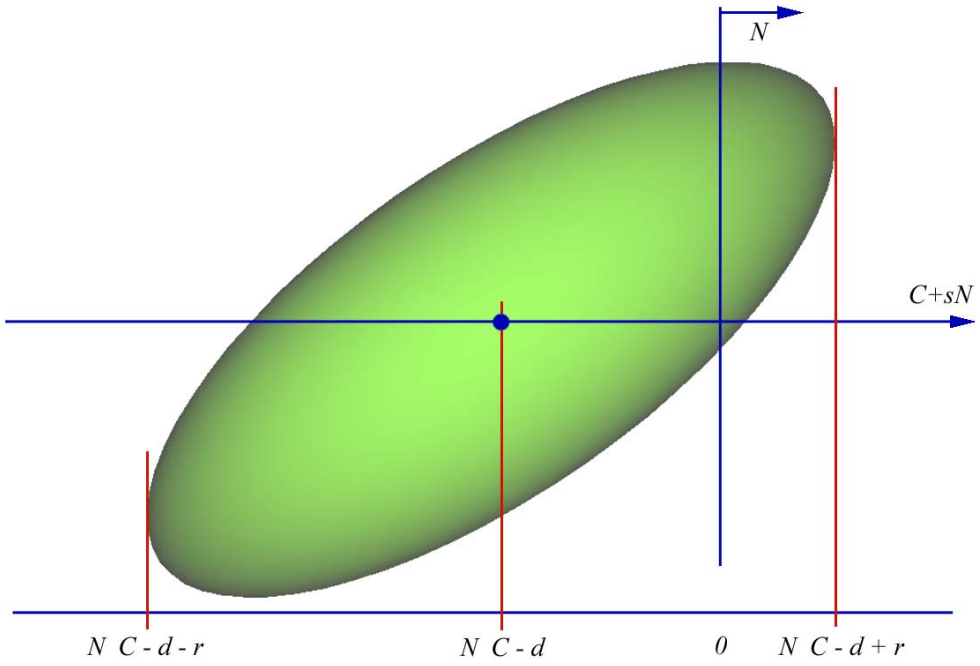

Figura 5.4 - Projeção do elipsóide e de um plano do volume de visão

O elipsóide apresenta uma particularidade importante quando tratamos cenas dinâmicas. Os demais volumes envolventes são definidos por pontos, vetores, raios e distâncias que são diretamente empregados nos cálculos de descarte. Quando o objeto que eles envolvem sofre uma transformação, a mesma transformação é aplicada a esses dados. No caso do elipsóide, o processo é mais complexo. Como foi visto, um dos dados usados no cálculo de descarte é a inversa da matriz que o define. Não podemos cair na armadilha de aplicar a mesma transformação diretamente nessa matriz inversa, pois $T \cdot M^{-1} \neq (T \cdot M)^{-1}$. Em vez disso, somos obrigados a recalcular a matriz *M*, conforme visto na Seção 4.6.1, e efetuar sua inversão a cada transformação sofrida pelo elipsóide, o que geralmente cobra um preço proibitivo para sua execução

a cada quadro gerado. Isso inviabiliza totalmente o uso de elipsóides para a avaliar a visibilidade de objetos geométricos dinâmicos.

5.7 Custos dos Cálculos de Descarte

A Tabela 5.1 compara os custos para os testes de descarte utilizando os algoritmos descritos nas seções anteriores. Vale lembrar que, para cada tipo de volume envolvente apresentado, foram propostos dois algoritmos. Os custos apresentados na tabela correspondem ao primeiro dos algoritmos exclusivamente para os casos em que o volume é descartado ou ao segundo algoritmo em qualquer caso, ou seja, nos casos em que é realizado o número máximo de cálculos. A última coluna da tabela apresenta os custos relativos de cada teste obtidos a partir do número e tipo de operações necessárias para realizá-lo. A Tabela 5.2 mostra os custos relativos dessas operações.

	Produtos Escalares	Multiplicações avulsas	Adições avulsas	Mult. de vetor por matriz	Radiciações	Total de Multiplicações	Total de Somas	Custo relativo parcial teórico
Caixa Orientada	4	3	3	-	-	15	11	32
Cápsula	2	0	2	-	-	6	6	14,4
Cilindro	3	2	3	-	1	9	9	27,1
Elipsóide	2	-	1	1	-	15	11	32
Esfera	1	0	1	-	-	3	3	7,2
Pastilha	4	0	4	-	-	12	12	28,8

Tabela 5.1 - Custos de testes de descarte

	Adição	Multiplicação	Divisão	Radiciação
Custo Relativo	1	1,4	4,9	5,5

Tabela 5.2 - Custos relativos de operações em um computador equipado com processador x86 de 6ª geração

Na implementação descrita nesta dissertação, os testes foram realizados no espaço de coordenadas da câmera, de modo que cada volume envolvente tem de sofrer as mesmas transformações de coordenadas que a entidade a que se refere. Essas transformações eventualmente acumulam rotações e translações sofridas pela entidade ao longo da simulação. Ao se analisar o custo computacional de se fazer um teste de visibilidade utilizando um determinado volume envolvente, devem-se considerar os custos correspondentes a essas transformações de coordenadas. Os algoritmos listados neste capítulo supõem os pontos, centros e vetores que definem os volumes já transformados.

A Tabela 5.3 apresenta os custos relativos às transformações de coordenadas correspondentes a cada volume envolvente considerado nesta dissertação.

	Transformações afins	Custo relativo parcial teórico
Caixa Orientada	4	86,4
Cápsula	2	43,2
Cilindro	2	43,2
Elipsóide*		
Esfera	1	21,6
Pastilha	4	86,4

* Veja a Seção 5.6

Tabela 5.3 - Número de transformações para cada tipo de volume

A Tabela 5.4 consolida os custos acumulados pelas mudanças de base e pelos cálculos de descarte. Cada volume é testado contra um número de planos que varia de 1 a 6 (o volume de visão é definido por 6 planos), resultando em uma média de 3,5 testes por volume[3]. O custo dos cálculos de descarte é obtido multiplicando-se esse valor pelos elementos da última coluna da Tabela 5.1.

[3] Veja o terceiro conjunto de testes na Seção 7.3.

	Custo das transformações afins	Custo dos cálculos de descarte	Custo total teórico	Custo relativo total teórico
Caixa Orientada	86,4	112	198,4	4,239
Cápsula	43,2	50,4	93,6	2
Cilindro	43,2	94,85	138,05	2,949
Elipsóide*		112		
Esfera	21,6	25,2	46,8	1
Pastilha	86,4	100,8	187,2	4

* Veja a Seção 5.6

Tabela 5.4 - Custo médio teórico para os cálculos de teste de descarte

Vale notar que, embora uma transformação em um espaço tridimensional seja dada por uma matriz 4x4 e sugira que sejam necessárias 16 multiplicações e 12 adições para transformar um ponto, como a transformação é afim e a coordenada *w* do ponto é 1, temos na verdade apenas 9 multiplicações e 9 adições.

Observações:

1. Custo de uma transformação afim $\mathbb{R}^3$: 9 multiplicações e 9 somas.
2. Custo de um produto de matrizes 3x3: 27 multiplicações e 18 somas.
3. Custo de uma inversão de matriz 3x3: 30 multiplicações, 11 somas e uma divisão.

5.8 Polígonos Oclusores

Embora a ênfase desta dissertação não seja o tratamento de oclusão, as técnicas de descarte contra o volume de visão usando volumes envolventes podem ser facilmente estendidas para verificar se um objeto é escondido por um polígono convexo. O primeiro passo é calcular o *volume de oclusão* do polígono oclusor. Trata-se de um tronco de pirâmide infinito com o ápice definido pela posição da câmera, os lados contidos por planos definidos por vértices adjacentes do polígono oclusor e a posição da câmera e o topo definidos pelo próprio polígono oclusor. Uma

analogia intuitiva do volume de oclusão é compará-lo ao volume de sombra do polígono relativo a uma luz onidirecional colocada na posição da câmera. Ao contrário dos testes contra o volume de visão, para ser potencialmente visível, um objeto deve estar fora do volume de sombra. No mais, os cálculos são idênticos.

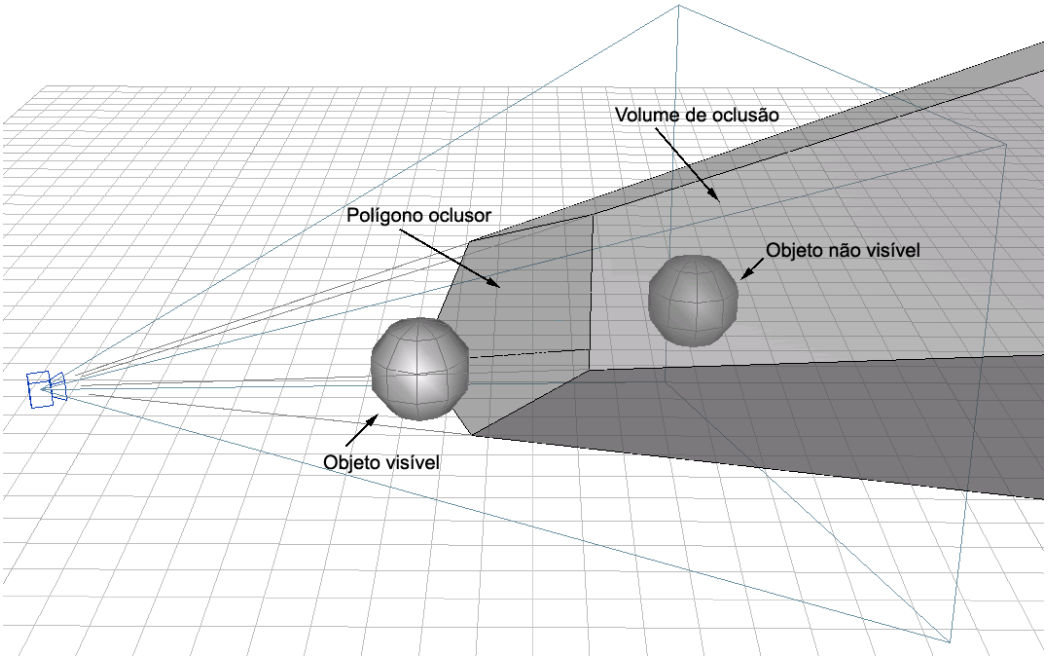

Figura 5.4 – Superfície oclusora

6 Propostas da Dissertação

O primeiro objetivo deste trabalho é propor uma forma de acelerar o processo de visualização diminuindo o número de primitivas submetidas à cadeia de restituição através do descarte de objetos geométricos usando volumes envolventes. O segundo objetivo é apresentar uma forma de otimizar o esquema de descarte através da redução do número de testes contra o volume de visão ou polígonos oclusores utilizando estruturas apropriadas para os modelos. A implementação destas propostas é feita em conjunto e em três etapas sucessivas: reorganização o modelo, cálculo dos volumes envolventes e escolha dos volumes para efetuar o descarte.

6.1 Reorganização da Cena

A confecção de um grande projeto dividido por disciplinas de engenharia tende a influir na hierarquia de grupos, de modo que cada setor confecciona sua parte segundo um agrupamento que lhe é conveniente e, no final, as partes são combinadas para compor todo o modelo. Como foi discutido no Capítulo 3, esta organização geralmente não é a melhor para ser submetida à cadeia de restituição.

Propomos, então, reorganizar o modelo de acordo com as técnicas tipicamente usadas em indexação espacial, como as empregadas em cartografia e geoprocessamento listadas no Capítulo 3. A hierarquia de vínculos é preservada e uma nova hierarquia de grupos é construída. As técnicas de geração experimentadas foram a da árvore-kd adaptativa e da árvore-R estática utilizando-se curvas de preenchimento de espaço para selecionar os componentes de cada nó, ambas com números variados de ramos por nó.

6.2 Cálculo de Volumes Envolventes

Após a cena ser reorganizada (ou não, caso se deseje manter a estrutura original), os volumes envolventes de cada nó são calculados. Visto que esses cálculos podem demorar várias horas, como mostra a Tabela 6.1, os volumes são normalmente guardados em um arquivo em disco. Conforme descrevemos no Capítulo 4, os volumes envolventes de um objeto geométrico são calculados a partir dos vértices da

malha de polígonos que descreve o objeto. Os grupos não possuem, eles próprios, vértices, apenas as folhas da sub-árvore que eles encapsulam. Para garantir que os volumes envolventes de um grupo sejam bem ajustados, eles são calculados a partir desses vértices, em vez dos volumes envolventes dos nós filhos.

Tipo de Hierarquia	Tempo
árvore-kd binária	19:26:06
árvore-kd ternária	08:07:56
árvore-kd quaternária	03:42:43

Tabela 6.1 - Tempos gastos para calcular todos os volumes envolventes do primeiro modelo de testes

6.3 Ajuste da Forma de Descarte

Nos Capítulos 4 e 5, foram apresentados os principais tipos de volumes envolventes e como utilizá-los para verificar se um objeto geométrico é potencialmente visível por estar dentro do volume de visão. Nos casos de teste, experimentamos a utilização do mesmo volume para todos os elementos do modelo e propomos três formas de usar esses volumes combinados:

Menor volume - Calculados todos os tipos de volumes envolventes, é tomado o de menor volume interno para ser usado em todos os testes de descarte do objeto ao qual se referem. Em caso de empate (que ocorre tipicamente entre a esfera e a cápsula e entre a cápsula e a pastilha), é tomado o de menor custo no cálculo de descarte.

Menor volume ponderado pelo custo do cálculo de descarte - De forma semelhante ao critério anterior, os valores dos volumes internos são calculados e então multiplicados pelo custo do cálculo de descarte dado pela Tabela 5.4. É tomado, da mesma forma, o de menor volume ponderado.

Aspecto - O volume envolvente para descarte é escolhido a partir do cálculo de três razões de aspecto dadas pela caixa envolvente orientada, conforme ilustrado na Figura 6.1. A *razão 2* é a razão entre a maior extensão da caixa orientada e a segunda maior extensão. A *razão 3* é a razão entre a segunda maior extensão e a menor extensão da caixa. A *razão 1* é a razão entre as *razões 2* e *3*. O volume envolvente é então escolhido a partir de uma combinação desses valores. A Figura 6.2 mostra o

diálogo que permite fazer essa escolha, já com valores iniciais para a seleção dos volumes, além de ilustrar o fluxo de decisões (da esquerda para a direita).

A escolha do volume pode ser realizada em uma, duas ou três etapas. Qualquer uma das três razões pode ser usada em cada etapa. A decisão é tomada em função da comparação entre uma das razões de aspecto do objeto e um valor escolhido. O resultado da comparação leva à escolha de um volume ou a uma nova comparação, até o máximo de três comparações. Esse procedimento tem a estrutura de uma árvore binária de decisões de três níveis. Cada nó da árvore possui uma das configurações da Figura 6.3.

O programa de testes pode ser instruído para coletar dados durante a simulação, de modo a reajustar os critérios de escolha para conseguir melhorar o desempenho da aplicação[4].

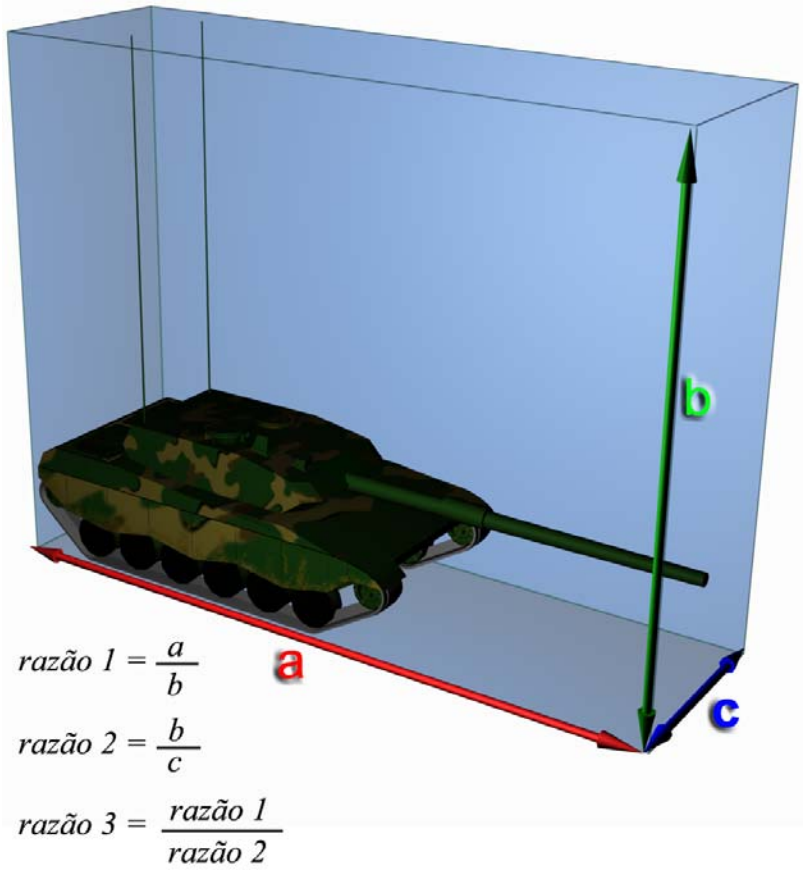

Figura 6.1 - Razões de aspecto

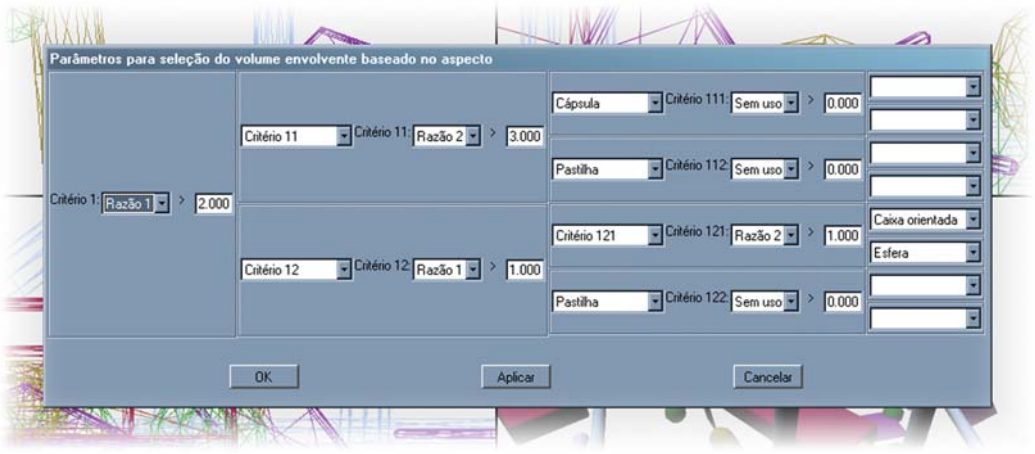

Figura 6.2 - Diálogo de seleção do volume para cálculo de descarte de acordo com as razões de aspecto

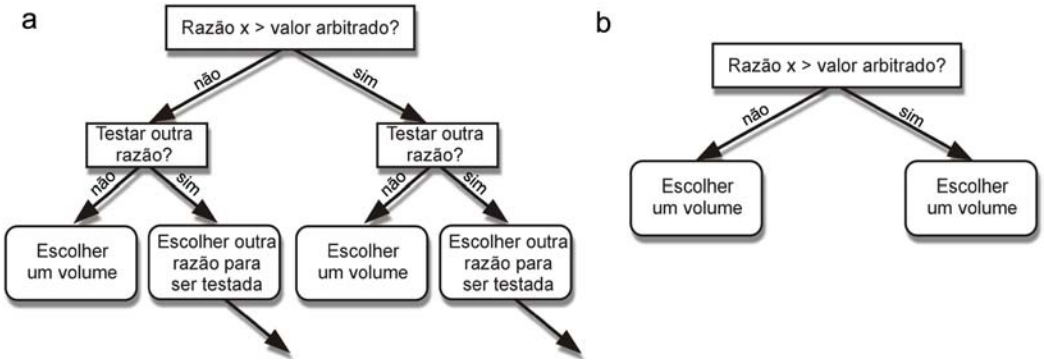

Figura 6.3 - Estrutura dos nós da árvore de decisão que permite a seleção do volume envolvente conforme o aspecto: (a) nós internos; (b) nós folhas

A Figura 6.3 mostra um navio aeródromo com seus componentes envolvidos segundo quatro critérios diferentes: (a) só caixas orientadas; (b) escolha do volume segundo o aspecto; (c) menor volume interno; e (d) menor volume interno ponderado

[4] Veja o oitavo conjunto de testes na Seção 7.3.

pelo custo do cálculo de descarte. As caixas são desenhadas em amarelo, cápsulas em vermelho, cilindros em verde, esferas em azul e pastilhas em lilás.

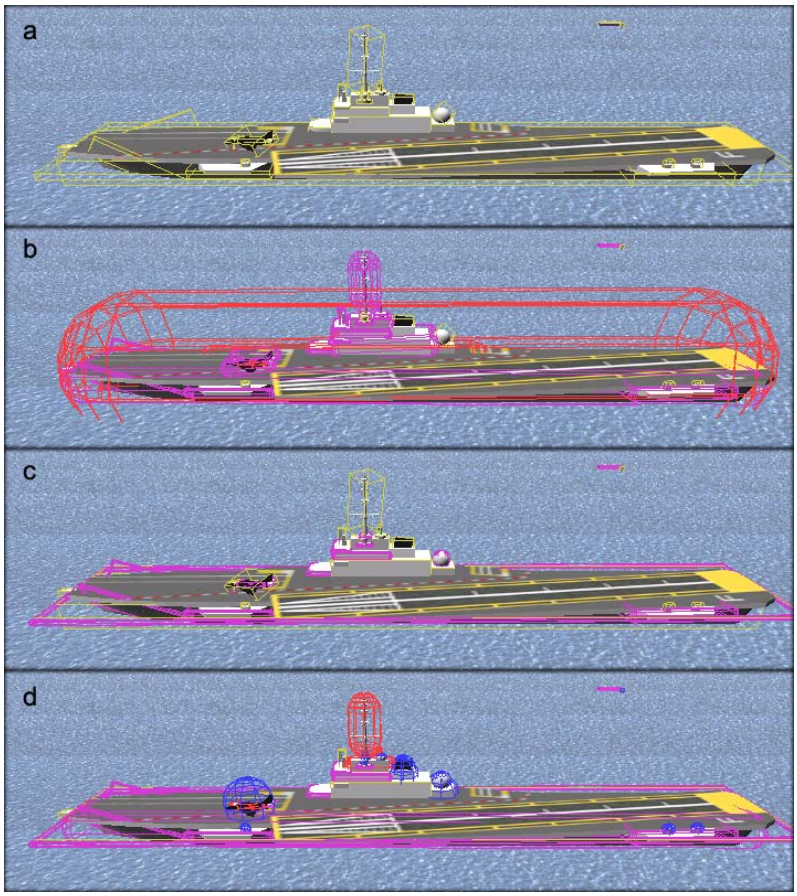

Figura 6.4 - Critérios de seleção de volume envolvente para descarte

7 Implementação e Testes

Como estabelecido no Capítulo 1, a intenção deste trabalho é desenvolver uma pesquisa que forneça subsídios para tentarmos responder às seguintes perguntas:

- Podemos melhorar o desempenho da aplicação com uma escolha apropriada de volumes envolventes para efetuar os cálculos de descarte?
- Como podemos organizar a cena de forma a obter melhores resultados com o descarte de objetos através de seus volumes envolventes?
- Qual o custo relativo dos cálculos de descarte em comparação com o custo de restituir os objetos de uma cena?

Para responder a essas perguntas, foi implementado um programa de testes através do qual foram experimentados modelos com diferentes composições e organizações e várias formas de efetuar o descarte de objetos contra o volume de visão e polígonos oclusores.

7.1 Implementação

O programa de testes, batizado 3DMau[5], foi implementado em C++ e executado em computadores compatíveis com IBM-PC sob o sistema operacional Windows 2000. Contudo, o código é totalmente portável, inclusive a interface, para plataformas Sun, SGI e outras.

Para compor a interface de janelas foi usada a biblioteca IUP [IUP], para a leitura de imagens empregamos a biblioteca IM [IM] e para a área cliente de desenho utilizou-se a biblioteca CD [CD], todas desenvolvidas pelo Tecgraf [TECG]. Para o desenho tridimensional foi usada a API OpenGL [OPENGL], além das bibliotecas GLU e GLUT.

Há que se destacar que o uso do OpenGL, de forma semelhante a outras APIs, combinado com computadores equipados com placas gráficas capazes de acelerar o

[5] "Mau" de Mauricio e "3DM" devido à interface lembrar vagamente o 3DMax.

desenho tridimensional, transfere boa parte dos cálculos do processador principal (CPU) para o processador da placa de vídeo (GPU). Isso é bastante desejável, pois libera o processador principal da carga do tratamento da geometria para que ele possa se ocupar do controle da simulação. Há placas de vídeo robustas que podem arcar com quase todos cálculos desde os primeiros estágios da cadeia de restituição. Outras, mais simples, são capazes de tratar apenas os estágios posteriores. De qualquer forma, é importante ressaltar que os resultados obtidos nos testes não são necessariamente modulados por uma API específica, mas sim pela aceleração do desenho por placas de vídeo que possuem esse recurso e por uma programação capaz de tirar proveito disso.

7.1.1 Funcionamento do Programa de Testes

Para proceder a simulação, uma cena previamente preparada é carregada de um arquivo em disco, reorganizada conforme a necessidade, os volumes envolventes de cada nó do grafo de cena são calculados ou carregados de um arquivo em disco, é ajustada a forma de descarte e a simulação é iniciada.

Carregamento da cena

No que diz respeito ao carregamento de cenas tridimensionais, vários formatos de arquivo foram tratados (PLY, SMF, etc.) e um recebeu atenção especial por conter *a maioria* dos dados necessários à execução deste estudo: o formato de exportação ASE do 3DStudio Max. Este formato, além de incluir as informações básicas a respeito dos objetos geométricos, como faces, vértices, normais e coordenadas de mapeamento de textura, contém chaves de animação para os objetos dinâmicos e as hierarquias de vínculos e de grupos. Outros dados necessários mas ausentes nesses arquivos puderam ser adicionados depois que o módulo complementar de exportação do formato ASE foi modificado, recompilado e reincorporado ao 3DStudio Max. Em virtude disso, o 3DStudio Max tornou-se o principal modelador e conversor de modelos para este trabalho – uma tendência, aliás, seguida por muitos fabricantes de programas deste gênero, como a Id Software [IDSO] no Quake 3 e a Paralelo Computação [PARA] no Fly3D.

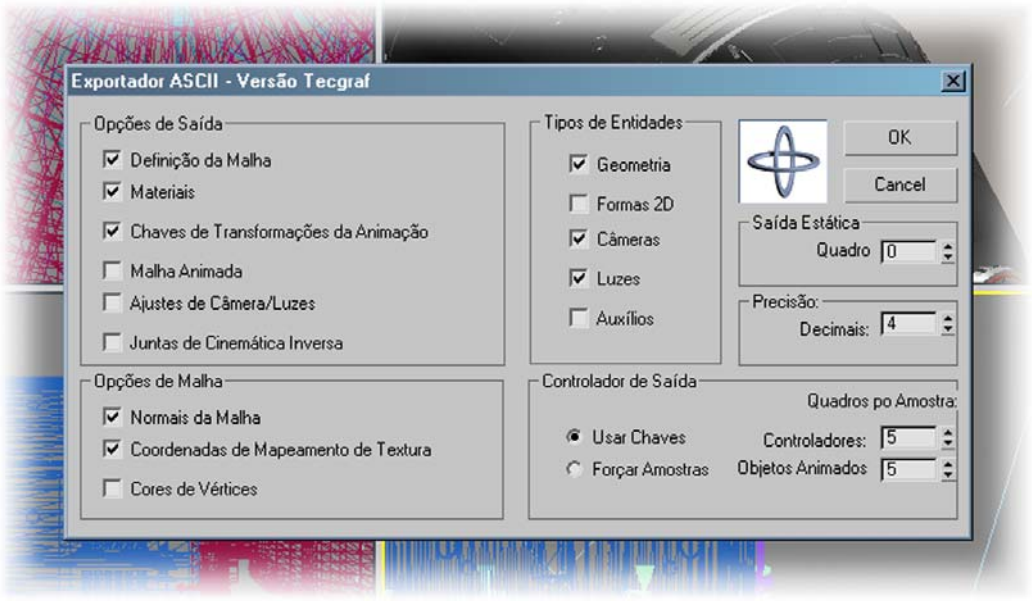

Figura 7.1 - Diálogo do módulo complementar de exportação do formato ASE modificado

A hierarquia de vínculos é a própria hierarquia de transformações. Quando dois corpos possuem movimento relativo, um subordinado ao outro, eles são vinculados através desta hierarquia[6]. A hierarquia de grupos é utilizada no 3Dstudio Max para organizar a modelagem. Um grupo é um conjunto de objetos geométricos ou de outros grupos sem movimento relativo, ou seja, uma vez agrupados, os elementos do grupo ficam congelados uns em relação aos outros. Eles podem, contudo, sofrer transformações de corpo rígido em conjunto, comportando-se como um objeto composto, tal como definido na Seção 3.2. Estas duas hierarquias se ajustam perfeitamente ao modelo hierárquico proposto na Seção 3.8.2.

7.1.2 Organização do Programa de Testes

O 3DMau possui classes dedicadas a carregar cenas tridimensionais, a processar essas cenas, a manter e exibir as entidades da cena e a tratar a organização da cena. A Figura 7.2 mostra um diagrama de classes simplificado contendo as

[6] Veja as seções 3.2 e 3.8.

principais classes do programa. Observe que essa estrutura é semelhante à maioria dos motores de restituição [FERR1999].

Classe TAplicacao - Controla a instância de TImagem, a interface, as telas de desenho e a comunicação da instância de TImagem com o controle da simulação. Existe apenas uma instância desta classe.

Classe TImagem - Comporta atributos que interessam a toda a cena, controla diretamente as entidades que ficam na raiz da cena, controla listas auxiliares de câmeras, luzes etc., coordena o acesso às entidades a partir da interface e possui diversas funções para gerenciar as entidades. Apenas um objeto desta classe é instanciado.

Classe TEntidade - Classe abstrata da qual são derivados todos os tipos de entidades.

Classe TGrupo - Controla conjuntos de entidades, inclusive de outros grupos.

Classe TObjetoGrafico - Classe abstrata da qual derivam todos os tipos de objetos geométricos.

Classe TMalha - Classe abstrata da qual derivam todos os tipos de malhas poligonais carregadas a partir de arquivos.

Classe TObjetoASE - Comporta atributos dos objetos lidos a partir de um arquivo no formato ASE. Possui recursos para visualizá-los e referências para objetos das classes de volumes envolventes.

Classe TArquivo - Classe abstrata da qual derivam todas as classes de leitura e escrita em arquivo e, em especial, as classes para leitura de arquivos de modelos.

Classe TMaxASE - Lê os dados de um arquivo ASE e monta objetos da classe TObjetoASE.

Classe TCamera - Controla uma câmera virtual conforme descrita na Seção 3.4.3. Possui uma instância da classe TTronco que define o volume de visão.

Classe TVista - Controla uma área de exibição da tela onde o modelo é desenhado segundo uma projeção paralela ortográfica. Pode haver mais de uma instância desta classe.

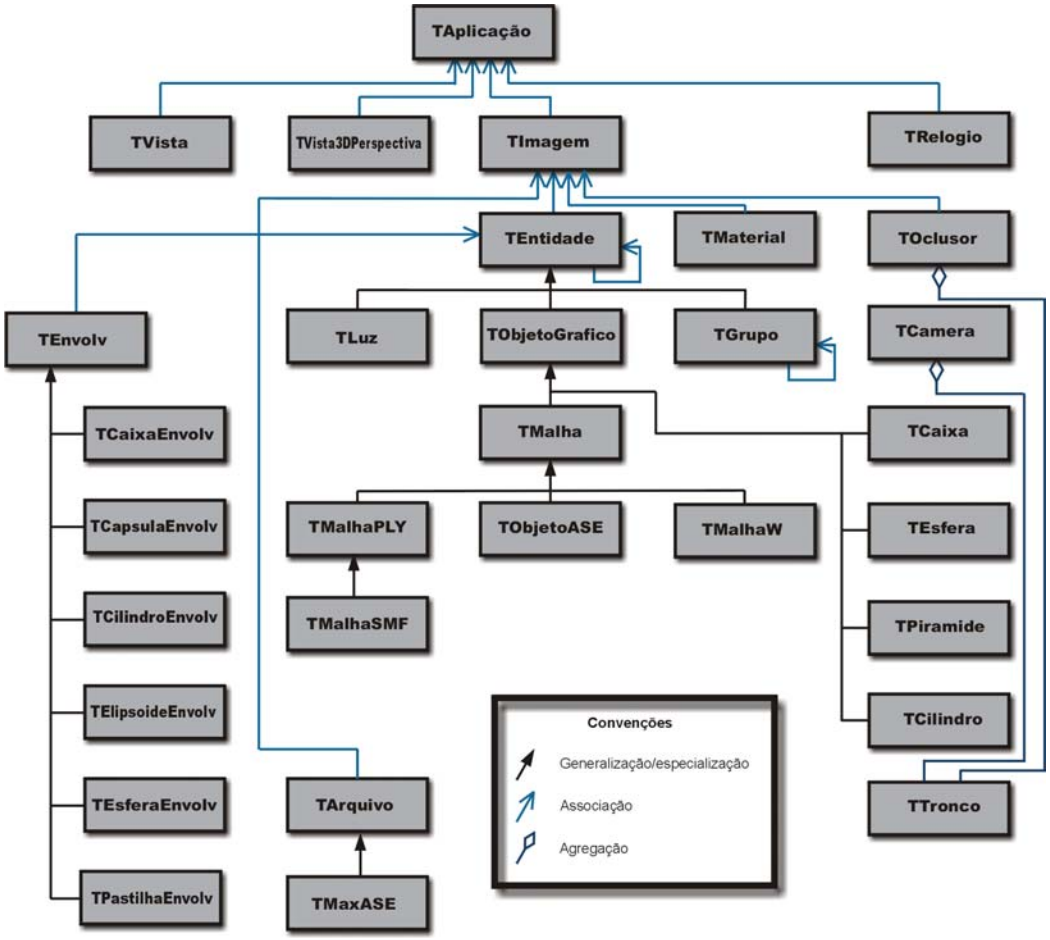

Figura 7.2 - Diagrama de classes simplificado da aplicação

Classe TVista3DPerspectiva - Controla a exibição de uma área da tela onde o modelo é desenhado segundo uma projeção perspectiva momentaneamente associada a uma instância de TCamera. Pode haver mais de uma instância desta classe.

Classe TMaterial - Comporta os atributos de aparência, conforme descritos na Seção 3.2, associados a uma instância da classe TObjetoGrafico.

Classe TOclusor - Comporta um polígono que é explicitamente definido como sendo um oclusor[7] na cena. Possui uma instância da classe TTronco.

Classe TTronco - É um tronco de pirâmide que é associado a uma câmera ou um oclusor. Quando associado a uma câmera, ele é finito e sua base é um quadrilátero, sendo portanto limitado por seis planos, conforme explanado na Seção

[7] Veja a Seção 5.8.

3.4.3. Quando associado a um oclusor, o tronco de pirâmide é infinito e possui i+1 planos, onde i é o número de lados do polígono.

Classe TRelogio - Mantém o relógio da aplicação, que é usado para controlar a simulação em função do tempo.

Classe TEnvolv - Classe abstrata da qual derivam as classes que tratam dos volumes envolventes.

Classes TCaixaEnvolv, TCapsulaEnvolv, TCilindroEnvolv, TElipsoideEnvolv, TEsferaEnvolv e TPastilhaEnvolv - Calculam e gerenciam os volumes envolventes que seus nomes sugerem. Possuem métodos que verificam se o volume está ao menos parcialmente dentro de um tronco (no caso do tronco de visão) ou ao menos parcialmente fora (no caso do tronco de um polígono oclusor).

7.2 Casos de Testes

Dentre os modelos examinados durante o desenvolvimento desta dissertação, dois foram selecionados para serem submetidos à seqüência de testes descrita abaixo por terem sido considerados representativos para o tipo de cenário eleito nesta pesquisa como meta de estudo.

1º modelo - Conjunto de objetos geométricos simples, como cilindros, cubos, cápsulas, etc., cada um com 3.962 a 10.450 faces. A cena toda tem 394.670 faces.

2º modelo - Uma plataforma marinha de produção de petróleo com seu sistema de ancoragem. Cada objeto geométrico possui de 2 a 2.970 faces, totalizando 1.248.679 polígonos. A simulação dura 133 segundos. O cenário é iluminado por duas fontes de luz onidirecionais. Uma particularidade deste modelo é que o mar foi usado como oclusor, ora encobrindo parte do sistema de ancoragem, ora encobrindo parte do corpo da plataforma, conforme a câmera esteja acima ou abaixo da superfície do mar.

Nesses cenários há objetos fixos e móveis, sendo que em ambos a câmera utilizada para captar imagens do mundo virtual realiza um percurso fixo que dura toda a simulação. As Figuras 7.6 a 7.9 exibem vistas panorâmicas dos cenários e os caminhos percorridos pelas câmeras.

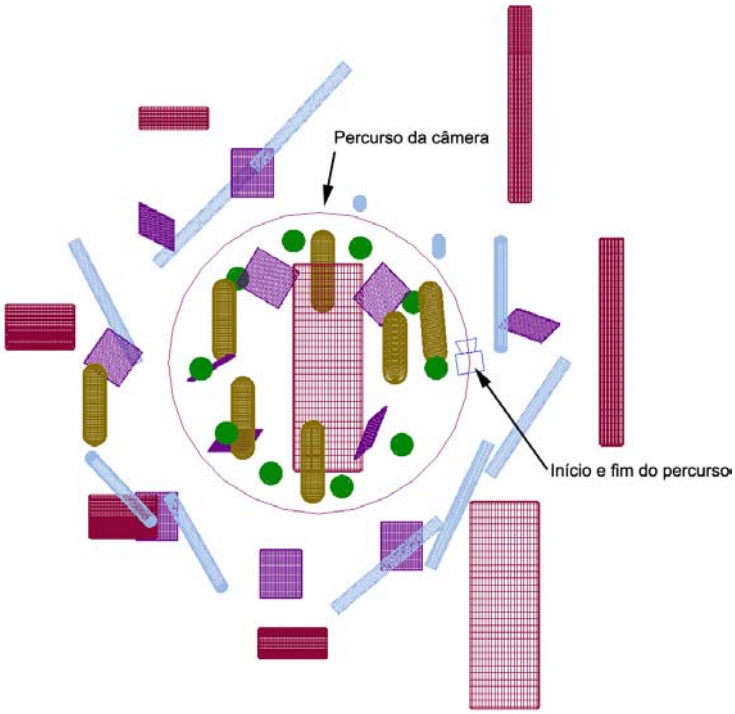

Figura 7.3 - Primeiro modelo: objetos geométricos básicos. O trajeto da câmera é visto de topo

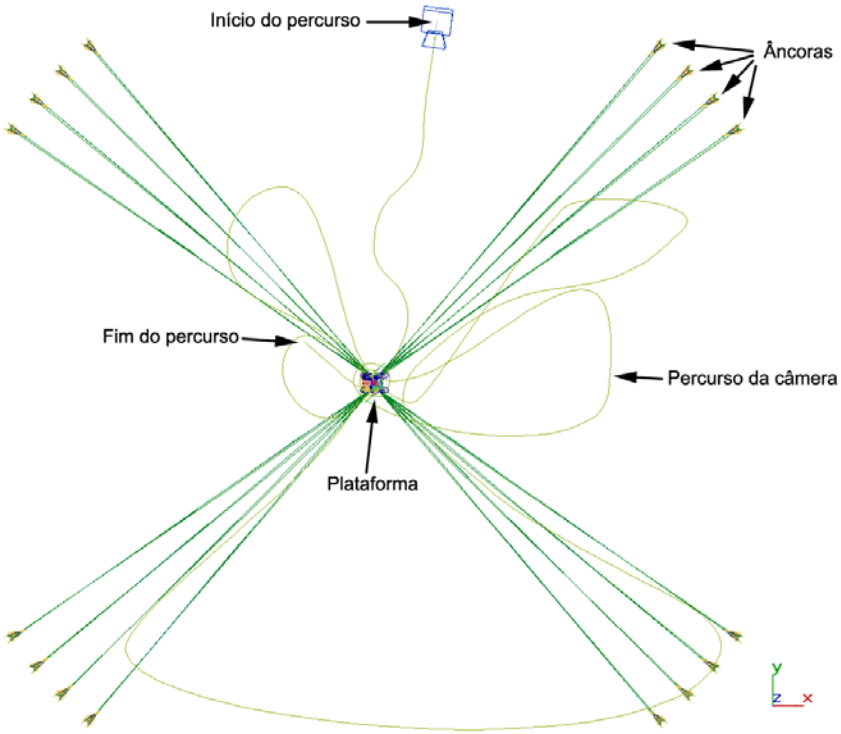

Figura 7.4 - Segundo modelo: plataforma de produção de petróleo vista do topo

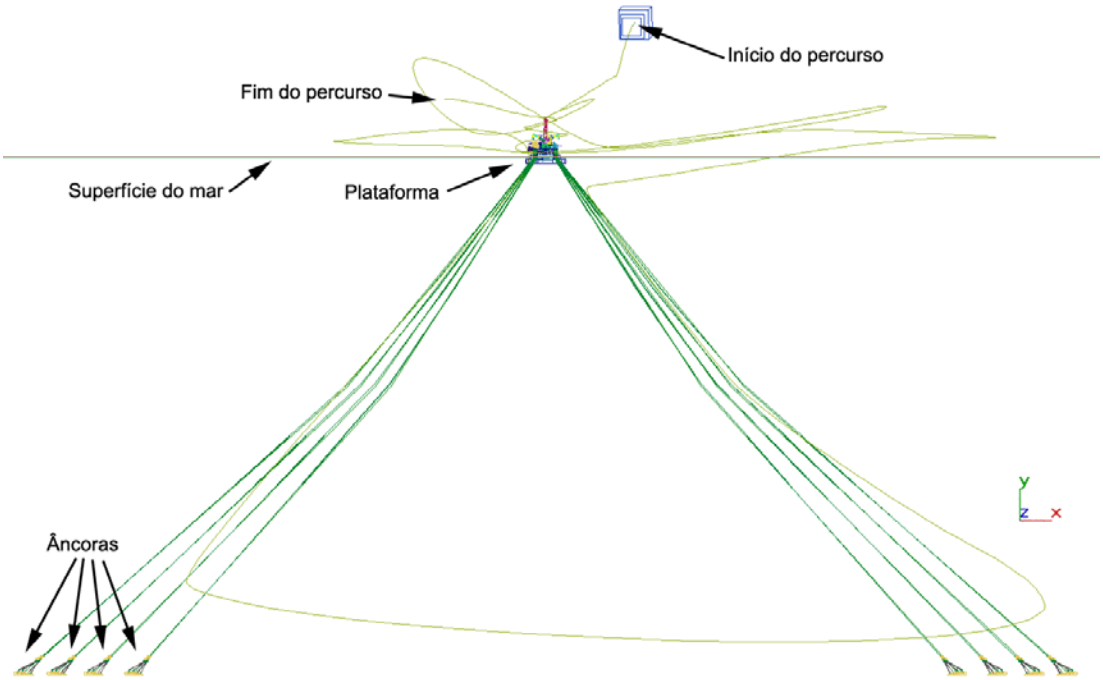

Figura 7.5 - Segundo modelo: vista lateral da plataforma de produção de petróleo

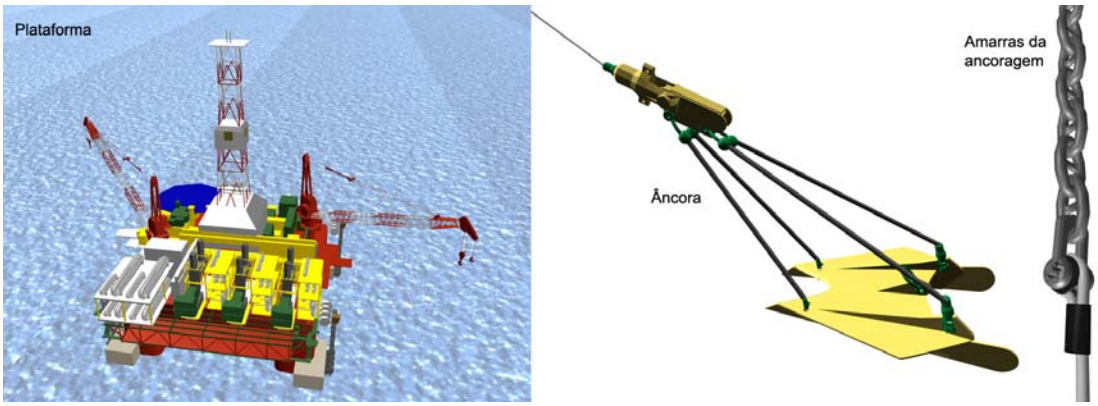

Figura 7.6 - Segundo modelo: detalhes da plataforma marinha de produção de petróleo

7.3 Resultados dos Testes

Os testes foram executados dentro dos intervalos de duração de cada simulação e medidos em tempo de relógio de parede e não em tempo de CPU, pois parte do processamento é realizado pela GPU e o processador principal pode ficar ocioso enquanto aguarda um resultado da GPU. Esta medida tem o inconveniente de que qualquer outro processo presente no computador possa consumir ciclos de CPU, afetando as medidas. Assim, tomou-se o cuidado de manter o mínimo possível de processos paralelos em execução.

Os testes apresentados nesta seção foram executados em dois computadores:

Computador 1 - PC Pentium III de 600 MHz com 256 MB de RAM, placa de vídeo ASUS V6600 De Luxe com processador Nvidia GeForce 256 e 32 MB de memória e sistema operacional Windows 2000 Professional.

Computador 2 - PC Pentium III de 800 MHz com 512 MB de RAM, placa de vídeo ASUS V7700 Pure com processador Nvidia GeForce 2 e 64 MB de memória e sistema operacional Windows 2000 Professional.

Em nenhum dos testes foi usado qualquer tipo de multiresolução, ou seja, uma vez decidido que um objeto geométrico deve ser desenhado, o desenho foi executado no seu nível de detalhe mais refinado.

7.3.1 Primeiro conjunto de testes

O cálculo de volumes *mínimos* não só é mais difícil de ser implementado, como também requer mais tempo de CPU para ser executado. Para avaliar se é compensador calcular volumes mínimos, esta seqüência de testes mostra os resultados obtidos com o uso da esfera mínima e da esfera que envolve a caixa mínima alinhada com os eixos (esfera-cae) de um objeto.

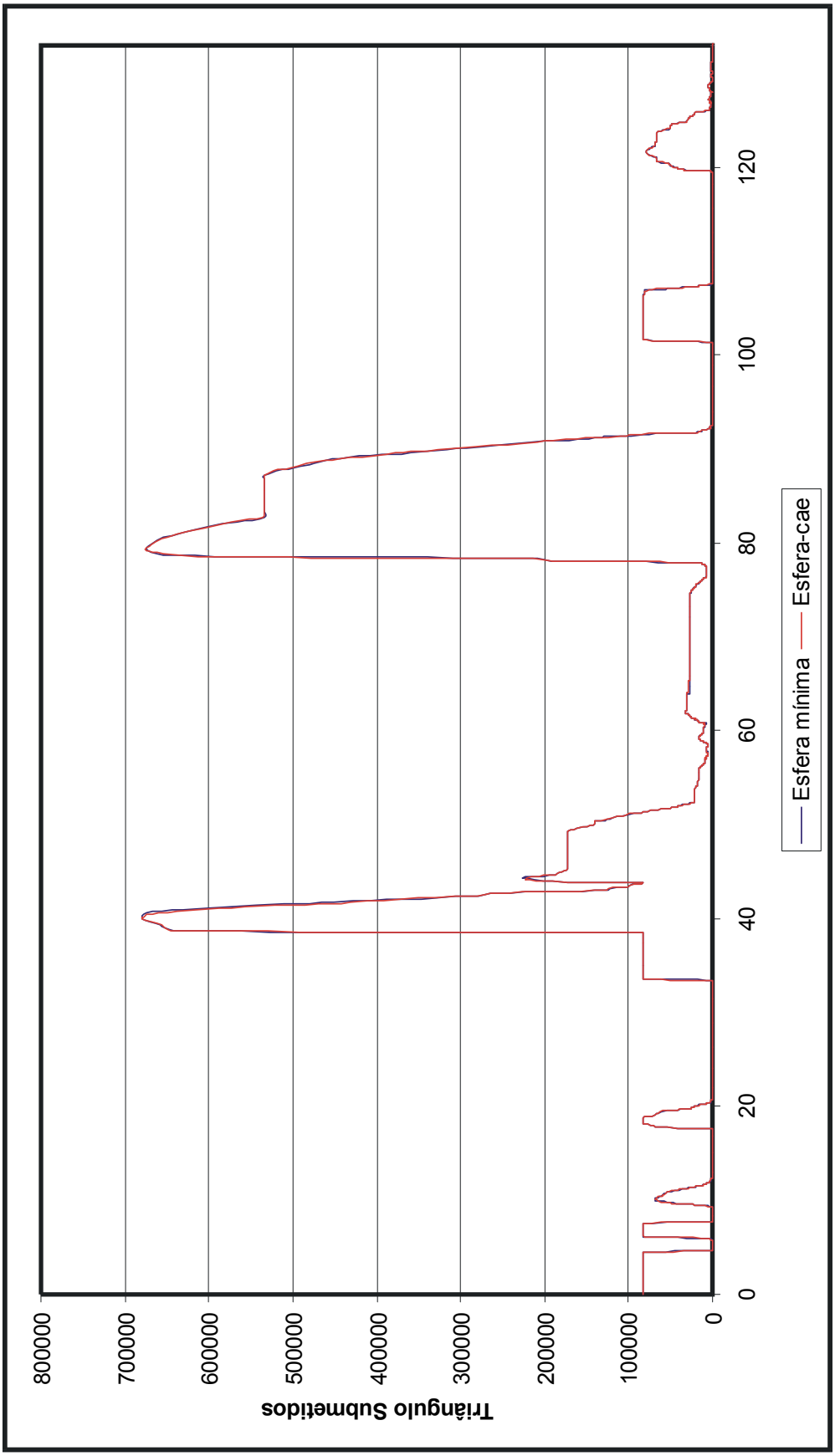

Gráfico 7.1 - Número de triângulos submetidos à cadeia de restituição utilizando esferas envolventes mínimas e esferas-cae

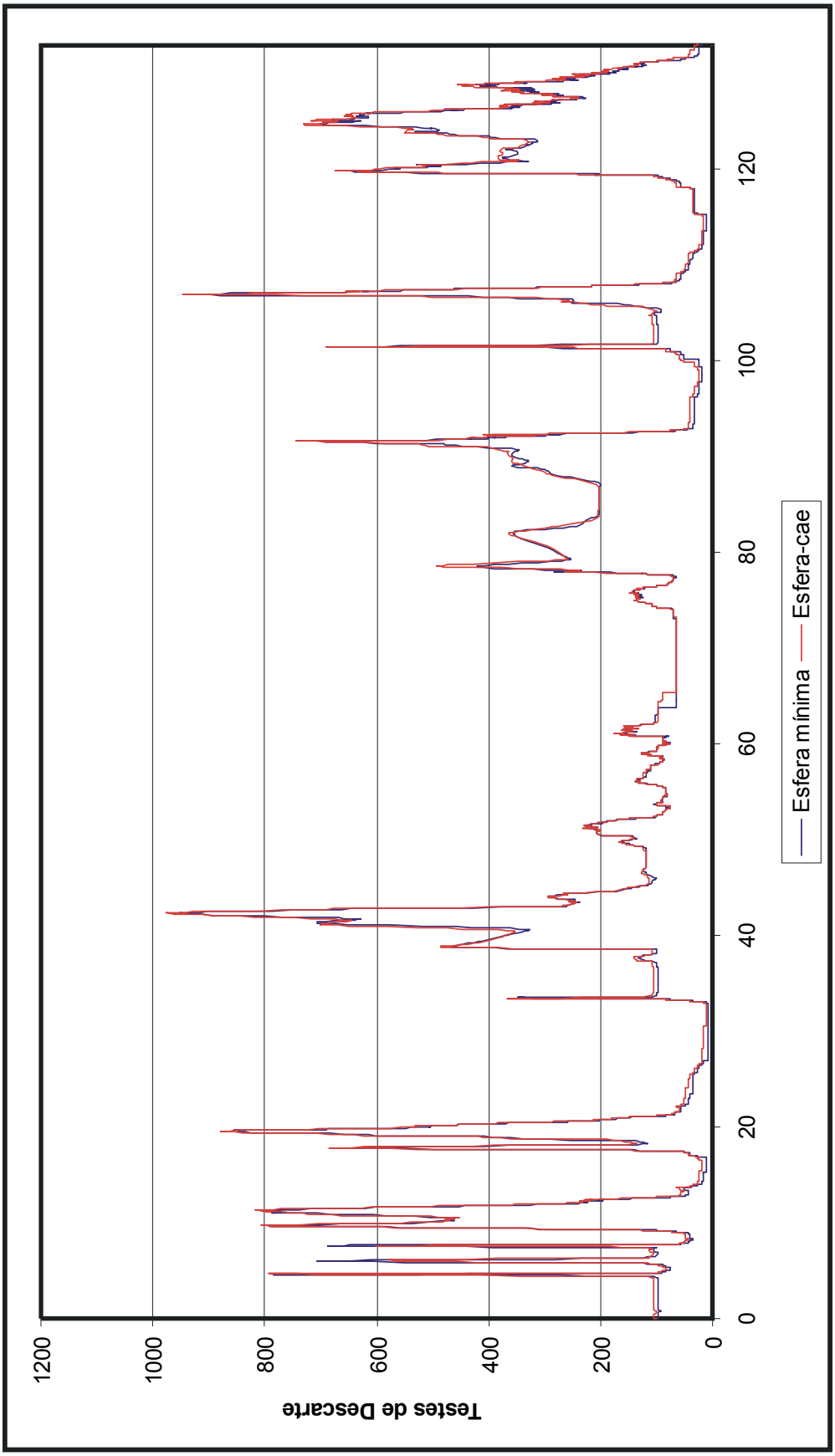

Gráfico 7.2 - Comparação de desempenho em número de testes de descarte usando esferas envolventes mínimas e esferas-cae

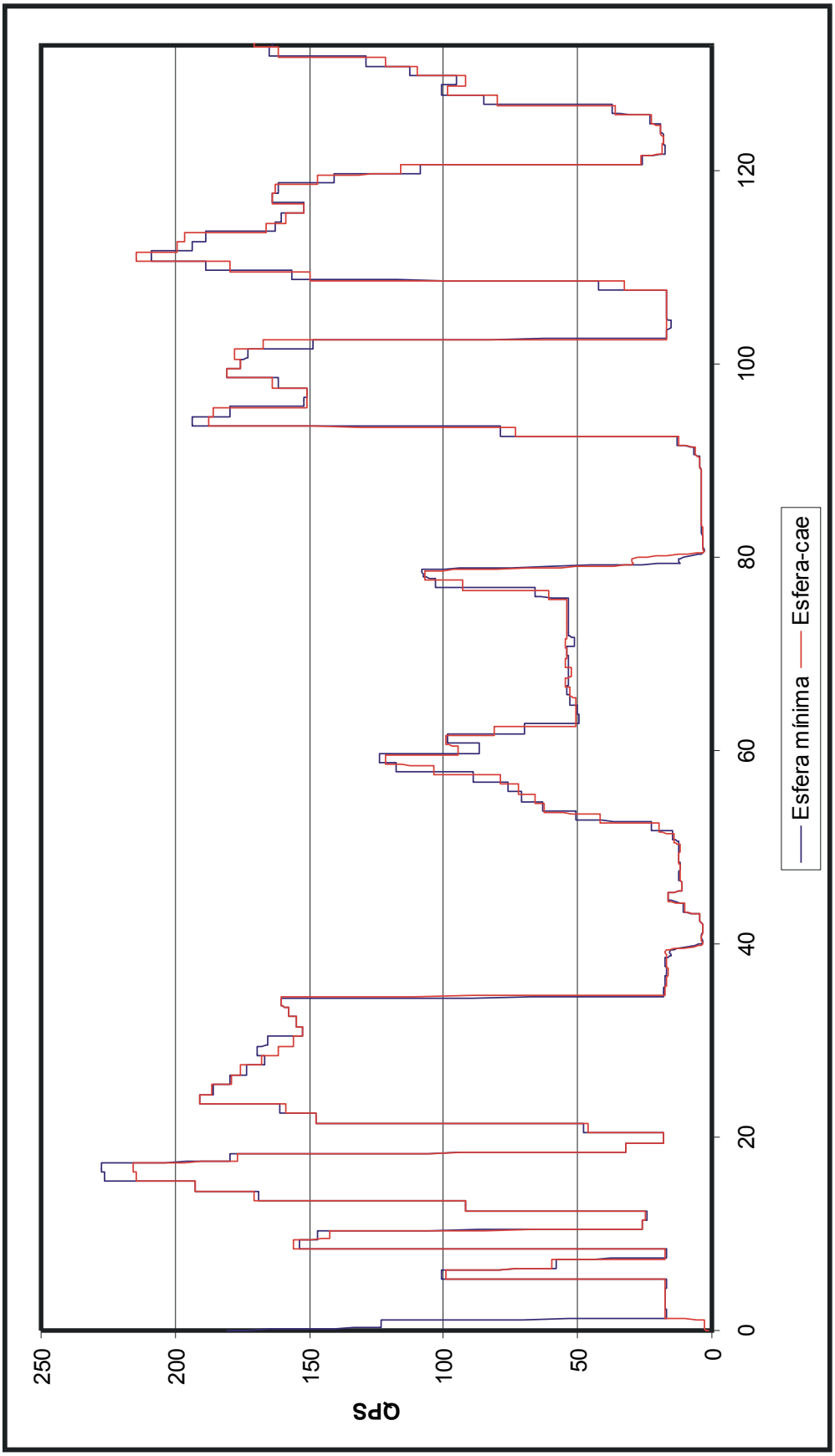

Gráfico 7.3 - Comparação de desempenho em quadros por segundo usando esferas envolventes mínimas e esferas-cae

No Gráfico 7.1, percebe-se que os dois tipos de esferas descartaram aproximadamente o mesmo número de triângulos. O Gráfico 7.2 mostra o número de cálculos de descarte e o Gráfico 7.3 apresenta o desempenho proporcionado por cada tipo de esfera. Para essa comparação, foi empregado o segundo modelo de testes.

Desta seqüência de testes podemos tirar a seguinte conclusão parcial: as esferas mínimas proporcionaram uma quantidade de testes ligeiramente menor que as esferas-cae, porém descartaram aproximadamente as mesmas quantidades de triângulos, resultando disso uma diferença de desempenho muito pequena (80,5 contra 80,2 qps). Embora as esferas mínimas sejam normalmente menores que as esferas-cae, elas ainda tendem a ter muito espaço vazio em seu interior, dependendo da geometria do objeto.

7.3.2 Segundo conjunto de testes

Esta seqüência permite verificar a influência de otimizações no código do programa de simulação sobre o desempenho global da aplicação e fornece dados para tentarmos localizar o gargalo da aplicação ou o conjunto de rotinas que demanda mais ciclos de CPU. A Tabela 7.1 mostra os resultados obtidos com o primeiro modelo organizado segundo uma árvore-kd adaptativa com 3 ramos por nó interno. As células em negrito destacam a alteração feita em relação aos testes anteriores.

	Tipo de ajuste nos procedimentos de teste							Tipo de descarte								
Teste	Computador	Debug/Release	Normais Pré-calculadas	Matriz OpenGL/Aplicação	Passagem dos vértices para a API	Com/Sem iluminação	Tamanho da área de desenho	Sem descarte	Caixas	Cápsulas	Cilindros	Esferas	Pastilhas	Aspecto	Menor volume	Menor volume ponderado
1	2	Dbg	Não	Aplic.	Vert	Com	1/1	1,783	11	11	10	9	11	11	10	11
2	2	Rel	Não	Aplic.	Vert	Com	1/1	2,8	16	15	16	14	17	16	17	17
3	2	Rel	Sim	Aplic.	Vert	Com	1/1	4,183	25	24	24	22	26	25	27	26
4	2	Rel	Sim	Aplic.	**Arr**	Com	1/1	5,75	24	24	24	22	26	25	27	26
5	2	Rel	Sim	Aplic.	Arr	**Sem**	1/1	6,867	31	33	33	31	34	34	34	34

Tabela 7.1 - Resultados das otimizações de código do programa de testes em número de quadros por segundo gerados para o primeiro modelo de testes

Comentários sobre a Tabela 7.1:

1. A primeira alteração foi compilar uma versão *release* da aplicação onde todas as otimizações de compilação e *linkedição* foram ativadas. O desempenho melhorou em todos os casos em torno de 30%. Como mostra a comparação entre a primeira e a segunda linha, não houve mudança na relação de eficiência entre os volumes ou combinações deles.

2. A linha 3 mostra o uso de normais pré-calculadas em vez de calculadas por demanda. Como esse cálculo era realizado pela aplicação, a CPU foi aliviada, permitindo uma melhora global de desempenho.

3. Na linha 4 experimentou-se passar valores de coordenadas de vértices, normais e mapeamento de textura através da desreferenciação de vetores em vez de passar os valores individualmente. Dessa forma, a quantidade de dados transferida da aplicação para o OpenGL e da CPU para a GPU é menor. Se o gargalo da aplicação fosse a transferência de dados, o desempenho global deveria ter melhorado, o que não aconteceu.

4. Para o teste referente à linha 5, a iluminação do modelo foi desativada. A melhora no desempenho significa que o cálculo de iluminação, realizado pela GPU, é um dos gargalos da aplicação para a visualização deste modelo[8]. A melhora no desempenho indica que o módulo de cálculo de iluminação está ligeiramente sobrecarregado.

Obs.: veja as conclusões parciais do quinto conjunto de testes.

7.3.3 Terceiro conjunto de testes

Nesta seqüência, o programa de testes foi monitorado ao longo da simulação para compilar dados que permitam avaliações instantâneas da eficiência das técnicas de descarte e das rotinas de descarte. Foram acompanhados o número de cálculos de descarte contra o volume de visão, o número de triângulos submetidos à cadeia de restituição e o número de quadros gerados por segundo. Estes testes foram feitos no

computador 2 usando o primeiro modelo organizado segundo a árvore-kd adaptativa com três ramos por nó.

O Gráfico 7.4 apresenta o número de triângulos submetidos à cadeia de restituição. Observe as grandes diferenças de eficiência apresentadas por cada método. O Gráfico 7.5 representa a mesma simulação, mostrando, para melhor clareza, apenas os resultados das esferas e do menor volume envolvente, que obtiveram respectivamente o pior e o melhor desempenho global.

Os gráficos 7.6 e 7.7 mostram as quantidades de testes de descarte contra o volume de visão realizados durante a simulação. As diferenças de eficiência no número de descartes entre os métodos mostrados nesse Gráfico são moderadas.

Os gráficos 7.8 e 7.9 apresentam o número de quadros gerados por segundo pelo programa de testes destacando a relação direta entre o número de triângulos descartados e o desempenho da aplicação.

O Gráfico 7.10 exibe a média de triângulos submetidos por quadro, de quadros por segundo gerados e de descartes por quadro contra o volume de visão efetuada por cada método.

[8] A iluminação, contudo, mostrou-se não ser um gargalo no quinto conjunto de testes.

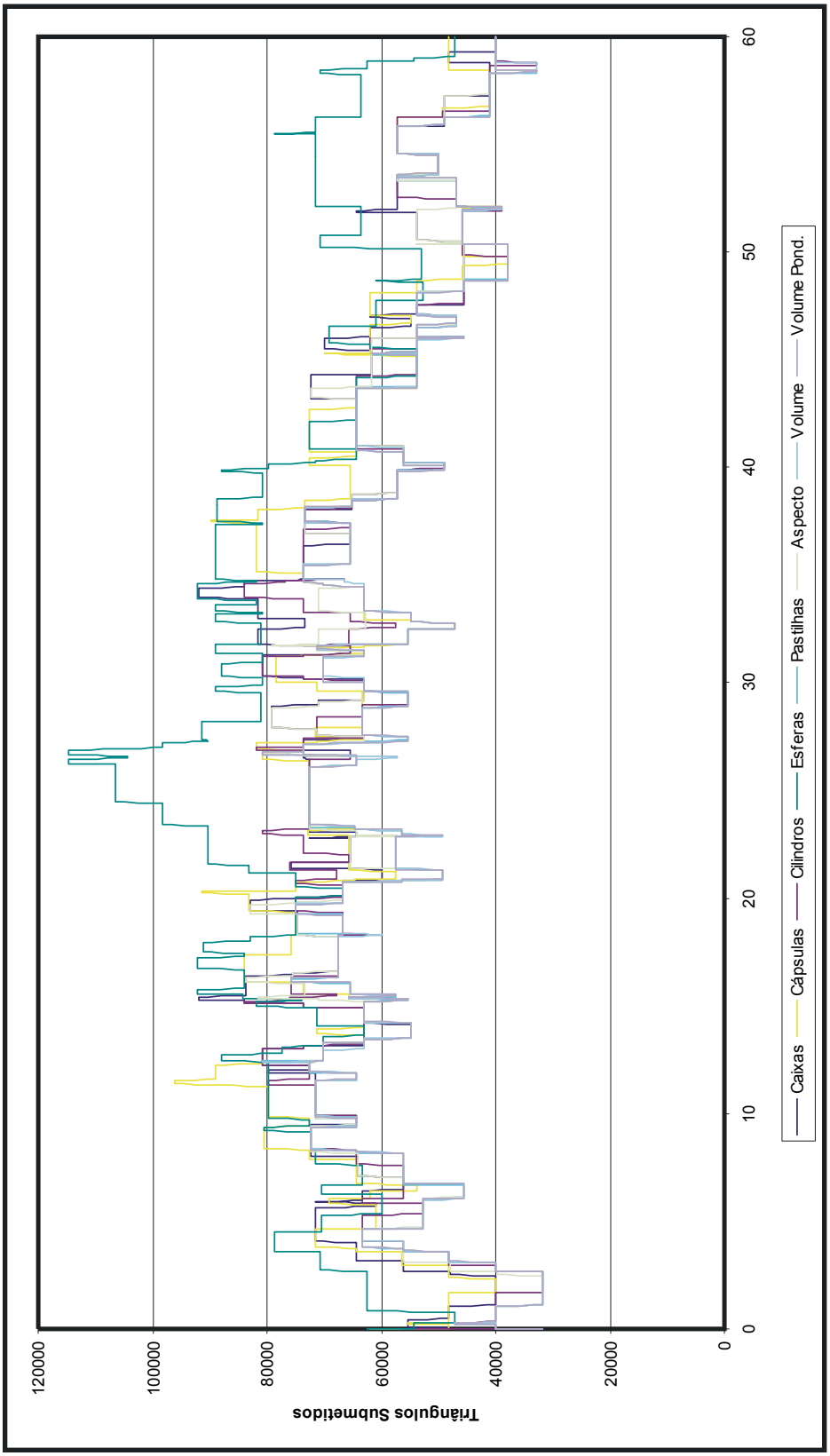

Gráfico 7.4 - Número de triângulos submetidos à cadeia de restituição no primeiro modelo de testes - todos os métodos

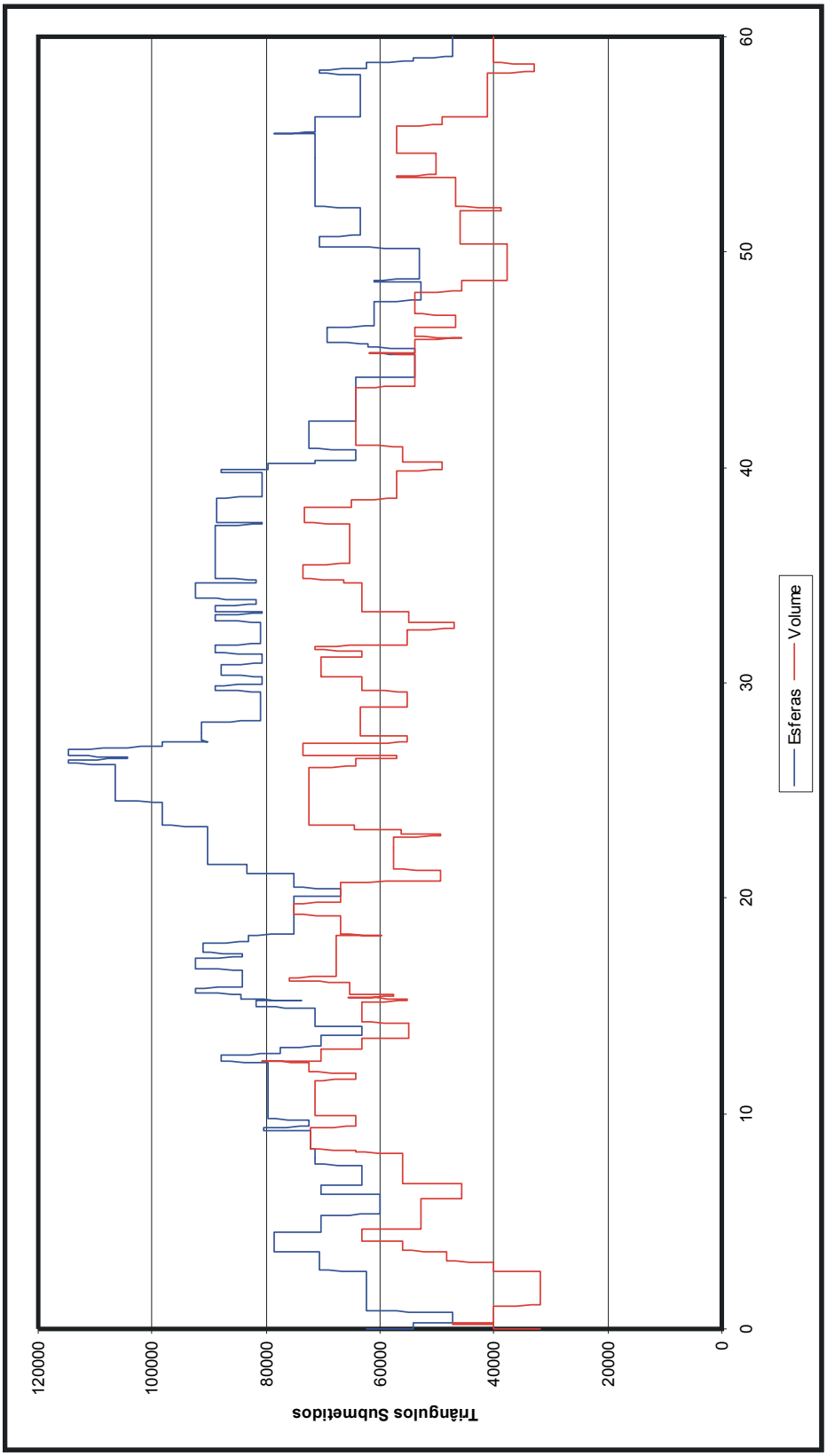

Gráfico 7.5 - Número de triângulos submetidos à cadeia de restituição no primeiro modelo de testes - esferas e menor volume

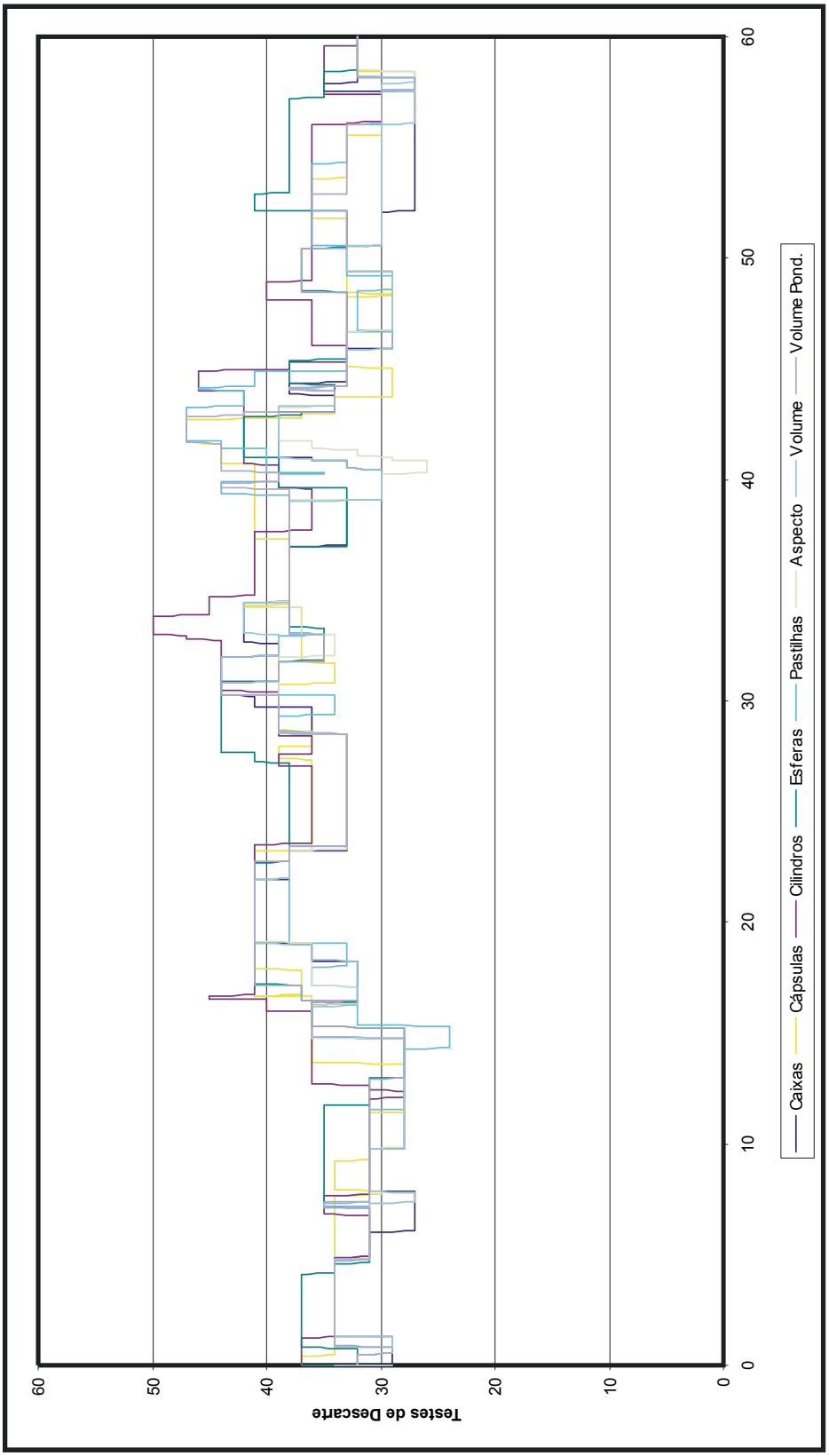

Gráfico 7.6 - Número de testes de descarte contra o volume de visão no primeiro modelo de testes - todos os métodos

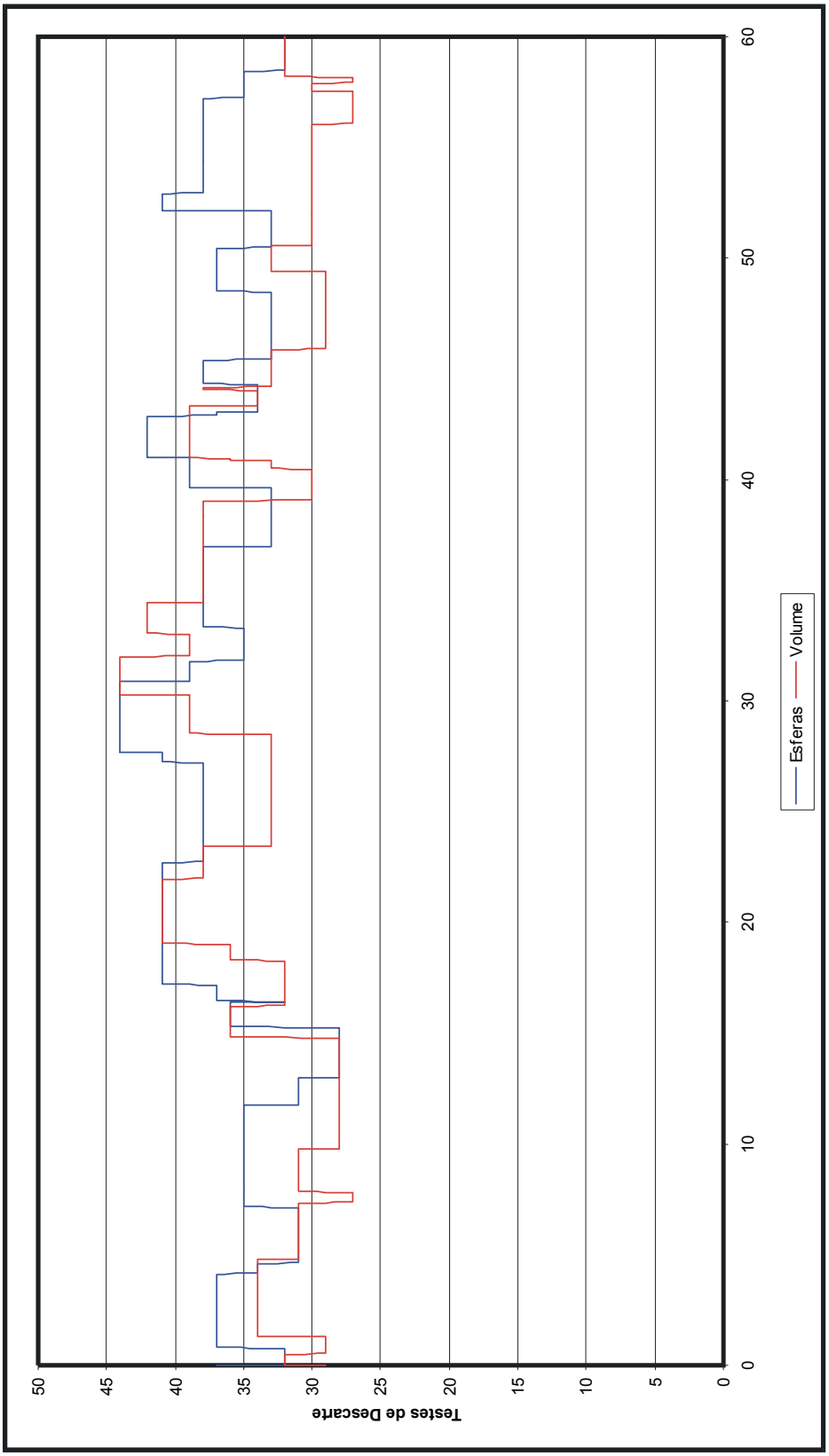

Gráfico 7.7 - Número de testes de descarte contra o volume de visão no primeiro modelo de testes - esfera e menor volume

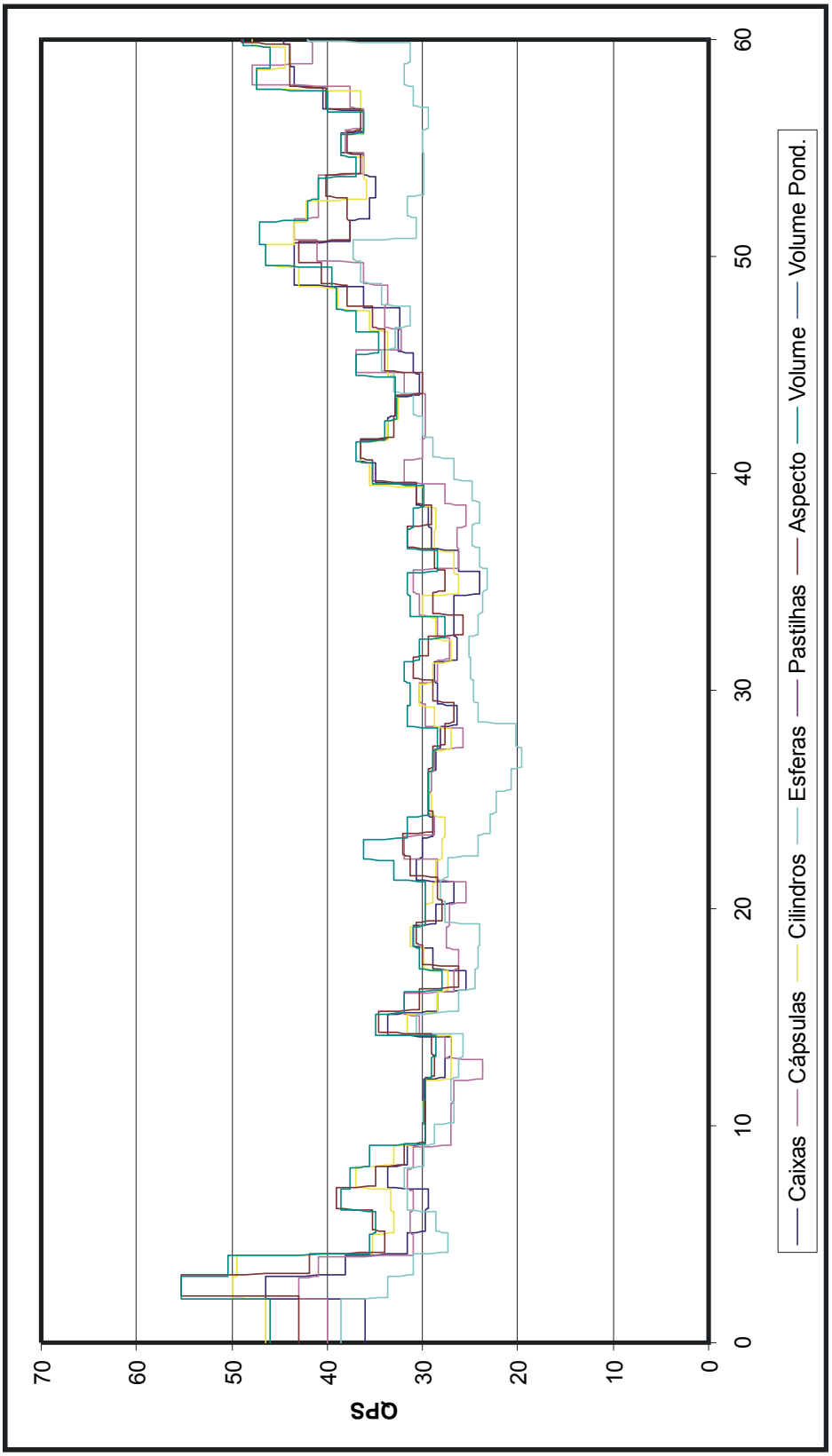

Gráfico 7.8 - Número quadros por segundo restituídos no primeiro modelo de testes - todos os métodos

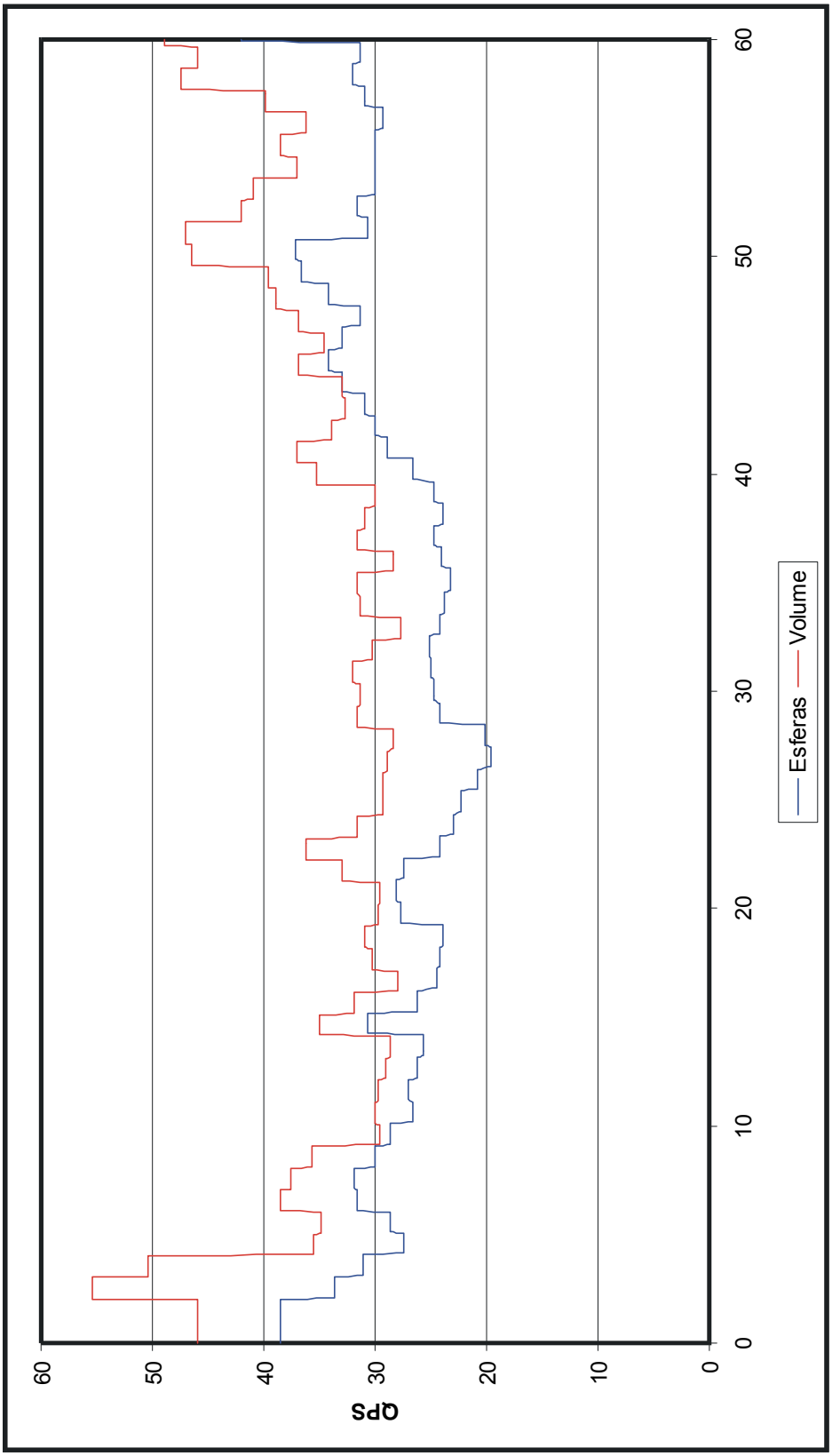

Gráfico 7.9 - Número quadros por segundo restituídos no primeiro modelo de testes - esferas e menor volume

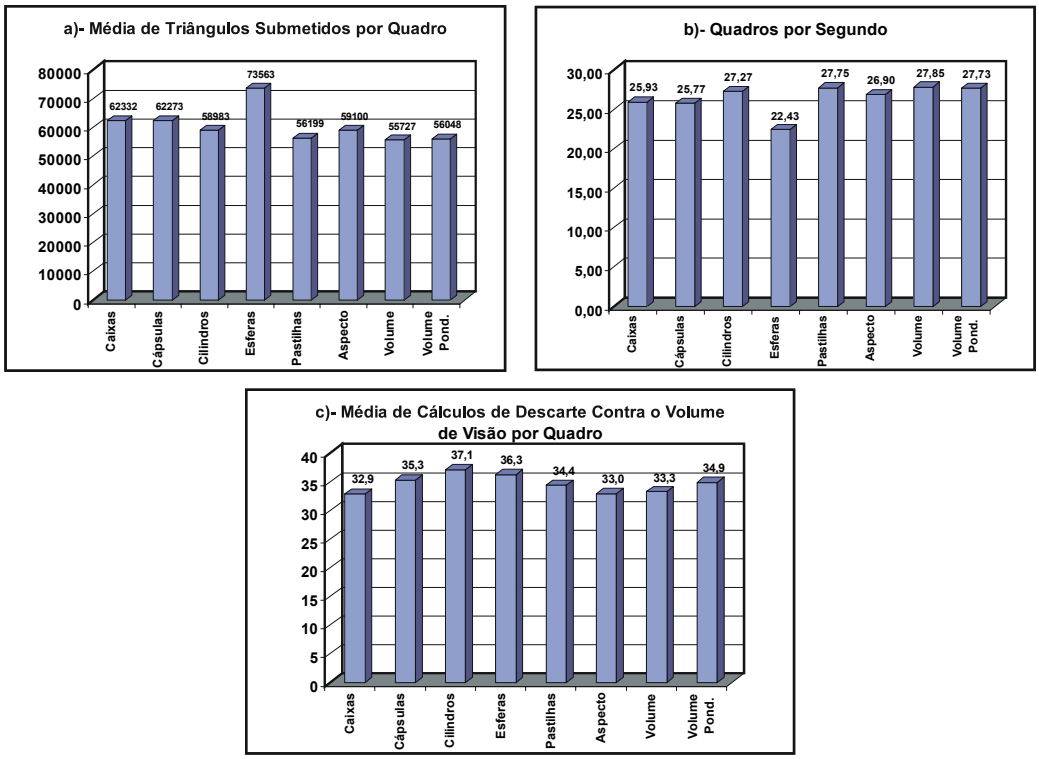

Gráfico 7.10 - Valores acumulados durante os 60 segundos da simulação do primeiro modelo de testes de acordo com cada tipo de volume envolvente

Durante este conjunto de testes, foi analisado o consumo de ciclos de CPU por módulo interno e externo ao programa de testes. A Tabela 7.3 mostra, na coluna laranja, o uso de CPU pelo módulo executável. As colunas amarelas são referentes aos módulos do OpenGL e gerenciadores da placa de vídeo. As colunas azuis se referem ao tempo de CPU consumido pelas chamadas ao sistema operacional. Esta tabela revela que a maior parte do trabalho de restituição de uma cena foi passada ao OpenGL e que uma pequena parcela de tempo de CPU é gasta com os cálculos de descarte.

CPU%	3Dmau.exe	Other32B	nvoglnt.dll	nv4_disp.dll	nv4_mini.sys	opengl32.dll	Σ	gdi32.dll	ntoskrnl.exe	win32k.sys	hal.dll	ntdll.dll	kernel32.dll	videoptr.sys	Σ	Outros
Sem descarte	1	53,3	41,2	0,1	0,3	0,1	95	0,2	2,1	0,3	0,3	0,6	0,1	0,2	3,6	0,2
Caixas	2,8	37,9	47,6	0,3	0,8	0,1	86,7	0,2	6,6	1,3	0,4	0,7	0,3	0,5	9,8	0,5
Cápsulas	2,6	37,5	48	0,2	0,8	0,1	86,6	0,2	6,5	1,5	0,5	0,7	0,3	0,5	10	0,6
Cilindros	2,9	37,6	46,9	0,3	0,9	0,1	85,8	0,2	7	1,5	0,5	0,7	0,4	0,6	10,7	0,4
Esferas	2,4	38,7	48,2	0,2	0,7	0,1	87,9	0,1	5,9	1,1	0,4	0,6	0,3	0,5	8,8	0,8
Pastilhas	2,9	36,7	48	0,2	0,9	0,1	85,9	0,2	6,9	1,4	0,4	0,7	0,3	0,6	10,3	0,7
Aspecto	2,8	37,2	47,6	0,3	0,9	0,1	86,1	0,2	6,9	1,4	0,4	0,7	0,4	0,6	10,4	0,5
Volume	2,9	36,1	48,3	0,2	0,9	0,2	85,7	0,2	7,1	1,4	0,5	0,7	0,3	0,6	10,6	0,6
Volume Pond	2,9	36,4	48,2	0,3	0,8	0,1	85,8	0,2	7,1	1,4	0,5	0,7	0,4	0,6	10,7	0,4

Tabela 7.2 - Uso de tempo de CPU por módulo externo ao programa de testes para o primeiro modelo

Desta seqüência de testes podemos tirar duas conclusões parciais:

- O volume escolhido para fazer o descarte teve grande influência no número de triângulos que foram descartados graças a uma boa adaptação (ou não) do volume envolvente ao objeto geométrico (ou conjunto de objetos). Observe que os métodos que utilizam apenas pastilhas ou o menor volume obtiveram resultados muito próximos. A Figura 7.7 mostra uma comparação da adaptabilidade dos dois métodos (apenas aos objetos geométricos, por motivo de clareza).
- De todo o tempo de processamento, somente 3% é usado pelo próprio programa de testes. Deste tempo, apenas uma parcela é dedicada aos cálculos de descarte. Assim, a influência desses cálculos no desempenho registrado em quadros por segundo pode ser praticamente desprezado devido ao seu baixo consumo de tempo de CPU.

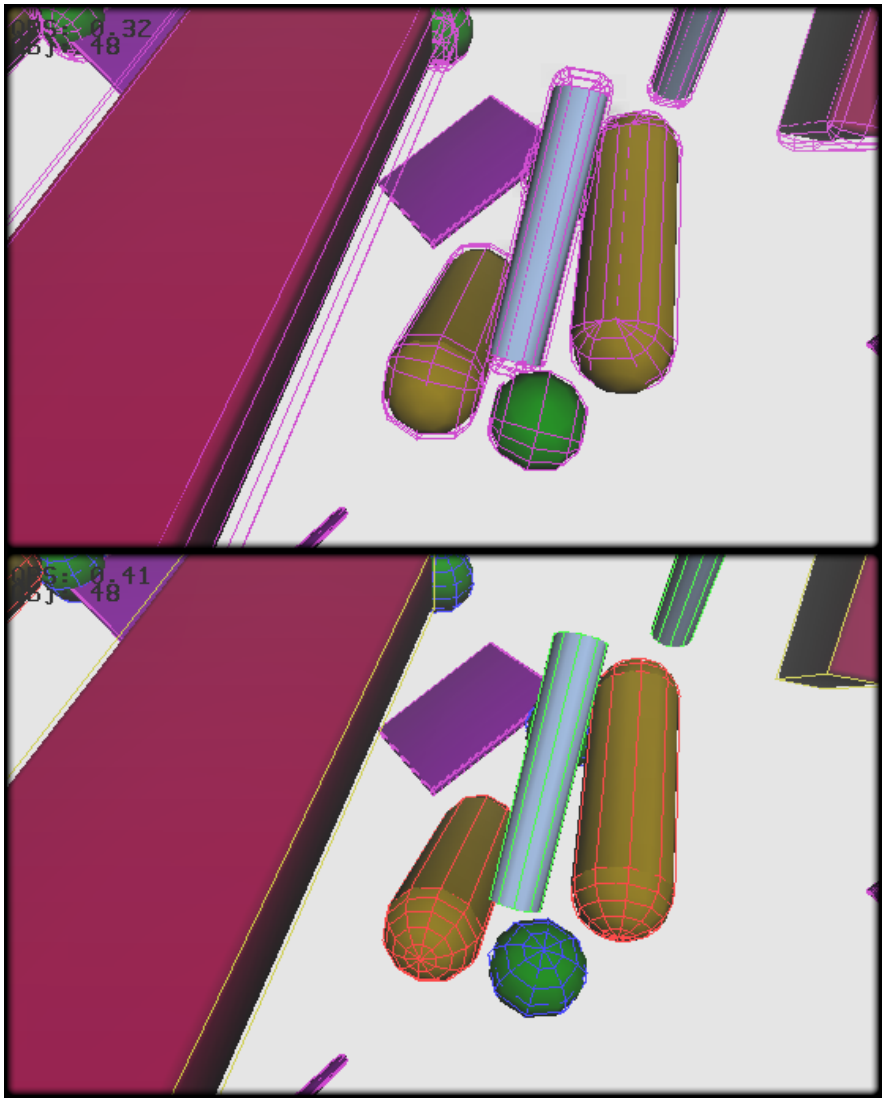

Figura 7.7 - Comparação da adaptabilidade dos volumes envolventes no primeiro modelo de testes. Na imagem de cima foram utilizadas pastilhas e na de baixo os menores volumes

7.3.4 Quarto conjunto de testes

Esta seqüência de testes foi efetuada para verificar a influência do tipo de organização da cena e do tipo de volume utilizado nos cálculos de descarte sobre o desempenho da aplicação. A Tabela 7.3 mostra os resultados obtidos com o segundo modelo, medidos em taxa média de quadros restituídos por segundo durante os 133 segundos de duração da simulação, executada no computador 2. Quanto maior este valor, melhor o desempenho. A linha *Engenharia* se refere ao modelo com a organização funcional do projeto de engenharia. As linhas *kd* indicam o modelo

reorganizado segundo uma árvore-kd adaptativa, com 2, 4, 8 e 16 ramos por nó. As colunas *H* referem-se ao modelo reorganizado segundo uma árvore-R estática com nós adicionados através do caminhamento do espaço usando a curva de Hilbert 3D através dos centros[9].

Comentários sobre a Tabela 7.1:

1. Na execução desta simulação, sem nenhum tipo de descarte, foram gerados em média apenas 1,75 quadros por segundo. Mesmo o pior resultado com o uso de descarte, em uma cena organizada hierarquicamente, é, para este modelo, 37 vezes mais rápido do que se não fosse feito nenhum descarte.
2. Todos os casos em que o modelo foi reorganizado tiveram um desempenho melhor do que os casos em que o modelo foi mantido com a organização original, embora as diferenças entre eles tenham sido modestas.
3. Os modelos organizados segundo a árvore-kd adaptativa obtiveram resultados ligeiramente melhores que os obtidos pelos modelos organizados segundo árvores-R estáticas.
4. O desempenho obtido pelas esferas foi 10% pior do que o obtido pelos volumes que alcançaram os melhores resultados.

	Caixas Orientadas	Cápsulas	Cilindros	Esferas	Pastilhas	Aspecto	Menor Volume	Menor Vol. Pond.
Sem hierarquia	16	19	17	**19**	18	18	18	18
Engenharia	66	79	76	65	**79**	78	76	77
kd2	85	82	83	79	86	84	86	84
kd4	86	84	84	80	**88**	86	**88**	85
kd8	85	83	82	80	86	86	86	85
kd16	84	82	81	79	85	84	84	83
H2	85	84	82	80	85	85	85	85
H4	85	83	82	80	85	85	85	85
H8	84	84	81	79	85	85	86	85

Tabela 7.3 - Resultados de testes com vários tipos de volumes envolventes e organizações da cena

[9] Veja a Seção 6.10.2.

7.3.5 Quinto conjunto de testes

Esta seqüência foi realizada para medir a influência de otimizações no código do programa de testes sobre o desempenho global da aplicação usando, desta vez, o segundo modelo. Por um lado, esta seqüência corrobora as constatações do segundo conjunto de testes no que diz respeito aos efeitos das otimizações. Por outro, ela exibe uma menor diferença de desempenho proporcionada pelos diferentes métodos de descarte. Neste modelo, as diferenças do melhor para o pior método ficaram em torno de 10% apenas, contra 20% do primeiro modelo. A Tabela 7.4 mostra os resultados obtidos com o segundo modelo organizado através uma árvore-kd adaptativa com 4 ramos por nó interno, por ter sido a organização que conduziu ao melhor desempenho, conforme a Tabela 7.3. As células em negrito destacam a alteração feita em relação aos testes anteriores.

Comentários sobre a Tabela 7.4:

1. A primeira alteração, mostrada na linha 2, foi compilar uma versão *release* da aplicação onde todas as otimizações de compilação e *linkedição* foram ativadas. O desempenho novamente melhorou em todos os casos em aproximadamente 30%, porém não houve mudança na relação de eficiência entre os volumes ou combinação deles.

2. A linha 3 mostra os resultados do uso de um computador mais veloz. Mais uma vez houve melhora de desempenho (cerca de 40%), mas o desempenho relativo permaneceu praticamente inalterado.

3. A linha 4 apresenta o resultado da utilização de normais pré-calculadas em vez de calculadas por demanda, aliviando a CPU e permitindo uma melhora global de desempenho.

4. Para o teste da linha 5, o tamanho da área de desenho foi reduzido para um quarto do tamanho original. Como isso só afeta a resolução da janela, a única etapa da seqüência de restituição que é afetada por esse ajuste é o rastreio. As demais permanecem com a mesma carga de processamento: descarte, transformações de vértices e cálculos de iluminação. O rastreio foi aliviado por ter uma área bem menor da tela

para preencher. Como não houve alteração nas medidas, este módulo da cadeia de restituição tem trabalhado com folga.

5. A partir da linha 6 passou-se a usar uma matriz de transformação interna à aplicação para realizar as transformações de corpo rígido dos volumes envolventes de forma a refletir as transformações sofridas pelos objetos geométricos a que se referem. Até então, era usada a matriz mantida pelo OpenGL. O uso dessa segunda matriz nos obriga a forçar a sincronização de processamento entre a CPU e a GPU toda vez que for necessário recuperá-la, gerando, em certos casos, grandes quedas de desempenho[10]. Contudo, o desempenho piorou pois, por um lado, essa sincronização não estava afetando severamente a simulação e, por outro, veio a sobrecarregar a CPU com os cálculos de atualização dessa matriz. O uso de matrizes mantidas por processadores diferentes para controlar itens mutuamente vinculados não afetou os resultados da simulação, pois as diferenças entre os elementos das matrizes começam a aparecer depois da 6ª a 8ª casa decimal, podendo geralmente ser desprezadas.

6. Na linha 7 experimentou-se passar valores de coordenadas de vértices, normais e mapeamento de textura através da desreferenciação de vetores em vez de passar os valores individualmente. Desse modo, a quantidade de dados transferida da aplicação para o OpenGL e da CPU para a GPU é menor. Não ter havido melhora de desempenho significa que a transferência de dados não é um gargalo.

7. Para o teste referente à linha 8, a iluminação do modelo foi desativada. A melhora no desempenho foi sutil, indicando que, na visualização deste modelo, o módulo de cálculo de iluminação não fica parte do tempo ocioso durante a restituição, mas também não está sobrecarregado.

[10] Em um computador de configuração semelhante ao computador 2, o uso da rotina de obtenção da matriz mantida pela API OpenGL tornou o programa de testes aproximadamente oito vezes mais lento.

	Tipo de ajuste nos procedimentos de teste							Tipo de descarte								
Teste	Computador	Debug/Release	Normais Pré-calculadas	Matriz OpenGL/Aplicação	Passagem dos vértices para a API	Com/Sem iluminação	Tamanho da área de desenho	Sem descarte	Caixas	Cápsulas	Cilindros	Esferas	Pastilhas	Aspecto	Menor volume	Menor volume ponderado
1	1	Dbg	Não	OGL	Vert	Com	1/1	0,44	24	22	24	22	24	24	24	24
2	1	**Rel**	Não	OGL	Vert	Com	1/1	0,56	31	29	31	28	31	31	31	31
3	**2**	Rel	Não	OGL	Vert	Com	1/1	0,79	43	39	42	39	43	43	43	43
4	2	Rel	**Sim**	OGL	Vert	Com	1/1	1,56	47	42	45	42	46	46	46	46
5	2	Rel	Sim	OGL	Vert	Com	**1/4**	1,56	47	42	46	42	47	47	47	47
6	2	Rel	Sim	**Aplic**.	Vert	Com	1/1	1,55	46	43	45	42	46	46	46	46
7	2	Rel	Sim	Aplic	**Arr**	Com	1/1	1,56	46	43	45	42	46	46	46	46
8	2	Rel	Sim	Aplic	Arr	**Sem**	1/1	2,21	48	45	48	45	48	48	48	48

Tabela 7.4 - Resultados das otimizações de código do programa de testes para o segundo modelo de testes (veja as convenções da Tabela 7.1)

A análise conjunta desta e da segunda seqüência de testes nos permite tirar três conclusões parciais:

- Os esforços para melhorar o desempenho do programa não favoreceram nenhum tipo de volume usado no descarte.
- A maior parte do consumo de tempo no processamento da simulação fica por conta dos cálculos pertinentes à GPU ou a cargo de módulos relacionados ao OpenGL, especialmente nas operações sobre os vértices e cálculos de iluminação.
- Tomando qualquer das linhas da tabela, vemos que o uso de esferas envolventes para o cálculo de descarte tem se mostrado o método menos eficiente, enquanto os demais apresentam resultados muito semelhantes.

7.3.6 Sexto conjunto de testes

Esta seqüência de testes é análoga à terceira seqüência, porém, desta vez, aplicada ao segundo modelo. Estes testes foram realizados no computador 2 com o modelo organizado segundo a árvore-kd adaptativa com 4 ramos por nó. Adicionalmente foi acompanhado, neste caso, o número de cálculos de descarte

contra o oclusor. A seguir, traçamos um paralelo entre os resultados obtidos por esses dois conjuntos de testes.

Os Gráficos 7.11 e 7.12 mostram o número de triângulos submetidos à cadeia de restituição. Observe que todos os tipos de volumes acabaram por descartar aproximadamente o mesmo número de triângulos. Estes gráficos apresentam um patamar notável na altura dos 90.000 triângulos. Ele representa, de certa forma, a superfície do mar, sendo atingido nos instantes em que a câmera está acima da superfície e a maior parte da plataforma fica visível, enquanto a maior parte da ancoragem fica encoberta pelo mar.

Os Gráficos 7.13 e 7.14 exibem as quantidades de testes de descarte contra o volume de visão realizados durante a simulação. As diferenças de eficiência no número de testes de descarte entre os métodos mostrados nesse gráfico são bem claras. Os Gráficos 7.15 e 7.16 ilustram o desempenho dos volumes no descarte contra o oclusor presente na cena. A diferença de eficiência é ainda maior. Os Gráficos 7.17 e 7.18 mostram o número de quadros gerados por segundo pelo programa de testes.

O Gráfico 7.19 apresenta a média de triângulos submetidos por quadro, de quadros gerados por segundo, de descartes por quadro contra o volume de visão e de descartes por quadro contra o oclusor efetuada por cada método.

Alguns dos gráficos a seguir apresentam marcas associadas a pontos notáveis da simulação, cujos quadros correspondentes são exibidos na Figura 7.8. Observe que, nos instantes 6 e 11 segundos, a plataforma é cortada por planos do volume de visão, forçando um caminhamento na sua árvore para verificar se os nós cortados estão dentro ou fora do volume de visão. Os instantes 41 e 80 segundos são exemplos de momentos em que grandes quantidades de triângulos são submetidas à cadeia de restituição, pois quase todo o sistema de ancoragem está visível e os métodos de descarte nada podem fazer para acelerar o processo de restituição. Como a ancoragem é vista de uma distância relativamente grande, o uso de impostores ou de algum tipo de multiresolução poderia compensar a queda de desempenho. As maiores taxas de quadros por segundo são obtidas em momentos como o observado aos 30 segundos, quando apenas o mar é visível.

Durante este conjunto de testes também foi verificado contra quantos planos do volume de visão um nó da árvore da cena não visível teve que ser testado antes de ser rejeitado. No programa de testes, cada nó é sucessivamente testado contra os planos próximo, distante, esquerdo, direito, superior e inferior, sempre nesta mesma ordem. O número de testes contra esses planos foi respectivamente de 50.936, 0, 430.977, 105.940, 366.840 e 20.760. Desta forma, cada nó foi testado em média contra 3,27 planos antes de ser descartado, um número bastante próximo do suposto na Seção 5.7.

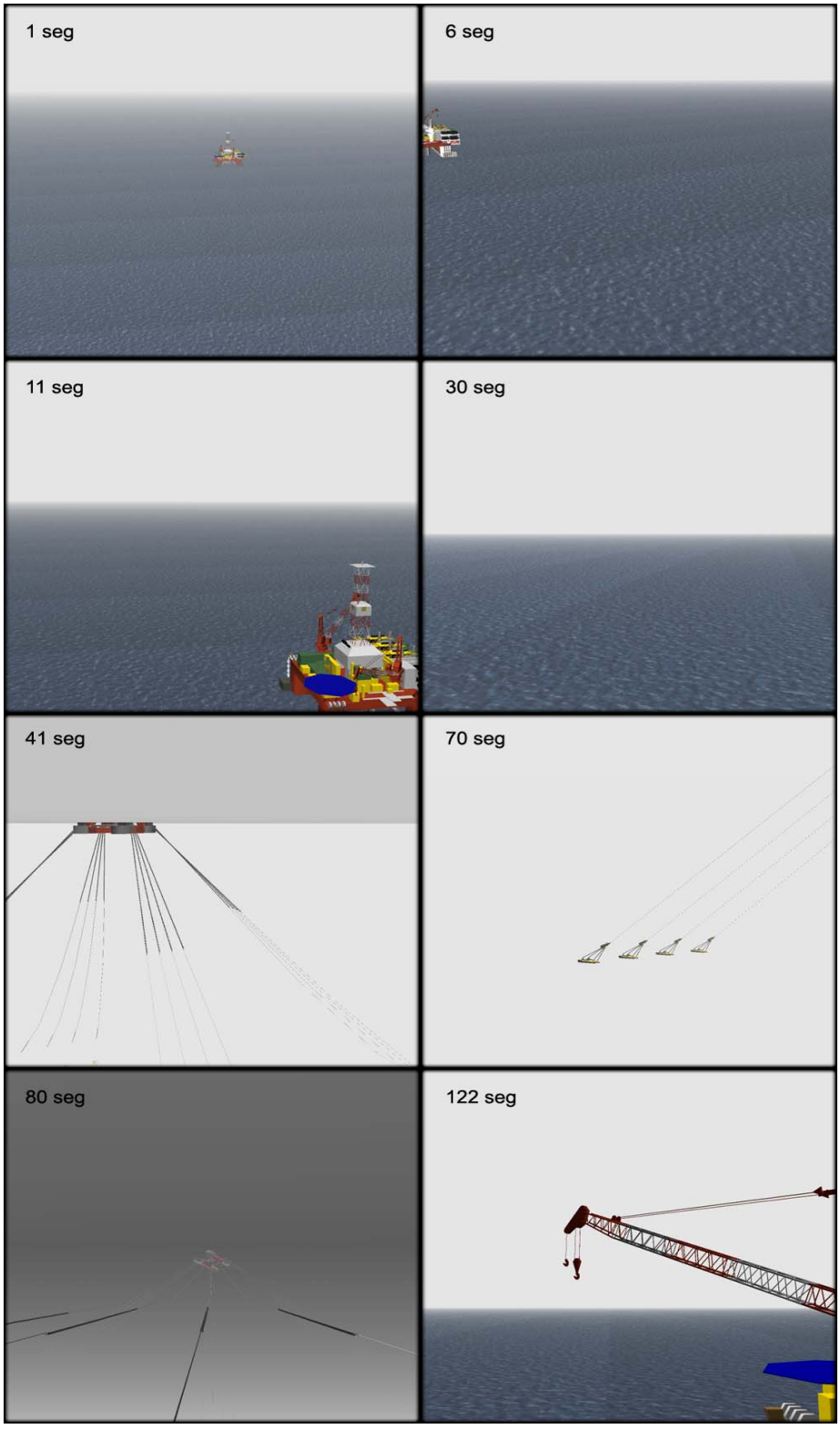

Figura 7.8 - Imagens restituídas durante a simulação

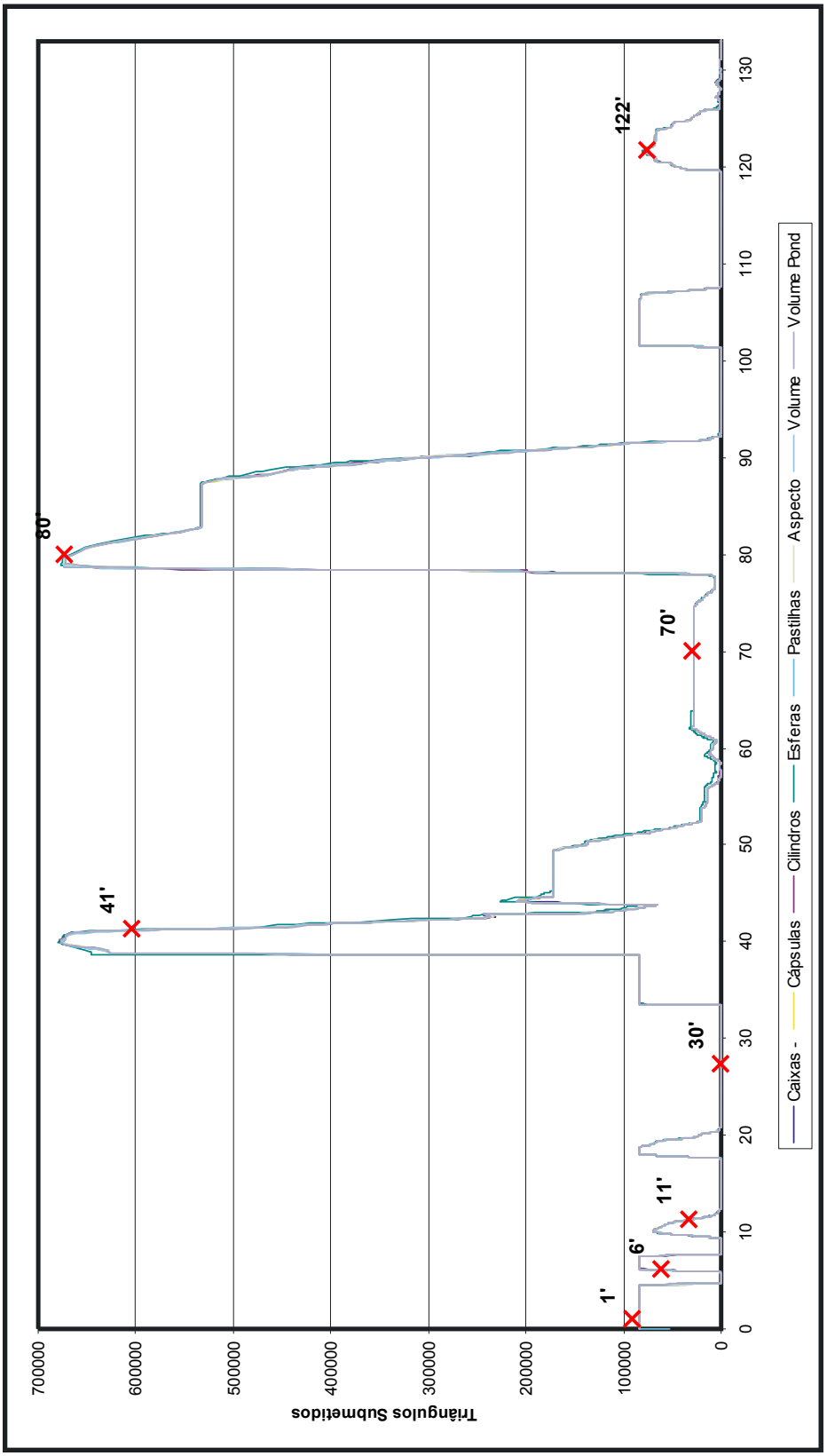

Gráfico 7.11 - Número de triângulos submetidos à cadeia de restituição no segundo modelo de testes - todos os

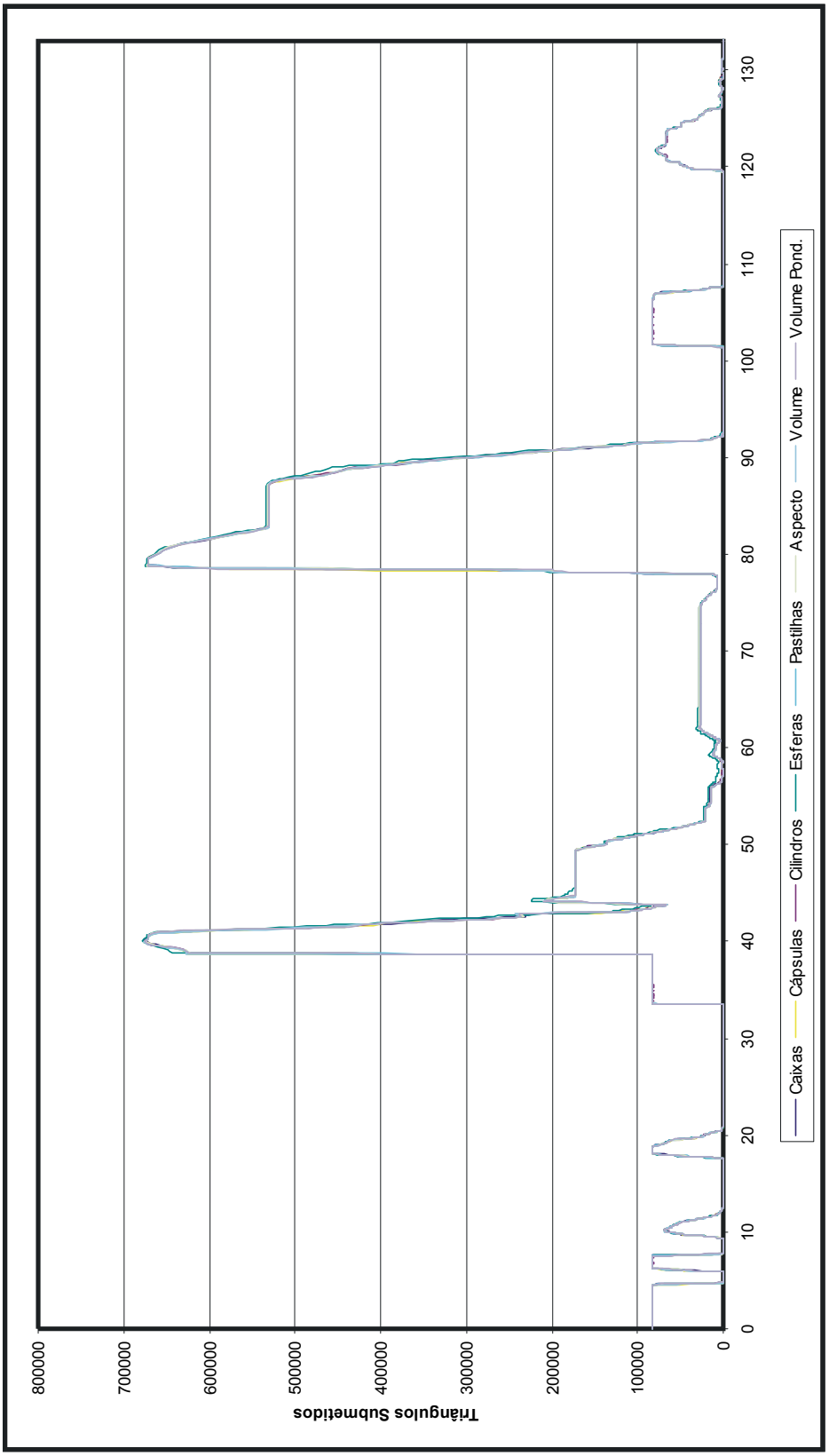

Gráfico 7.12- Número de triângulos submetidos à cadeia de restituição no segundo modelo de testes - esferas e menor volume

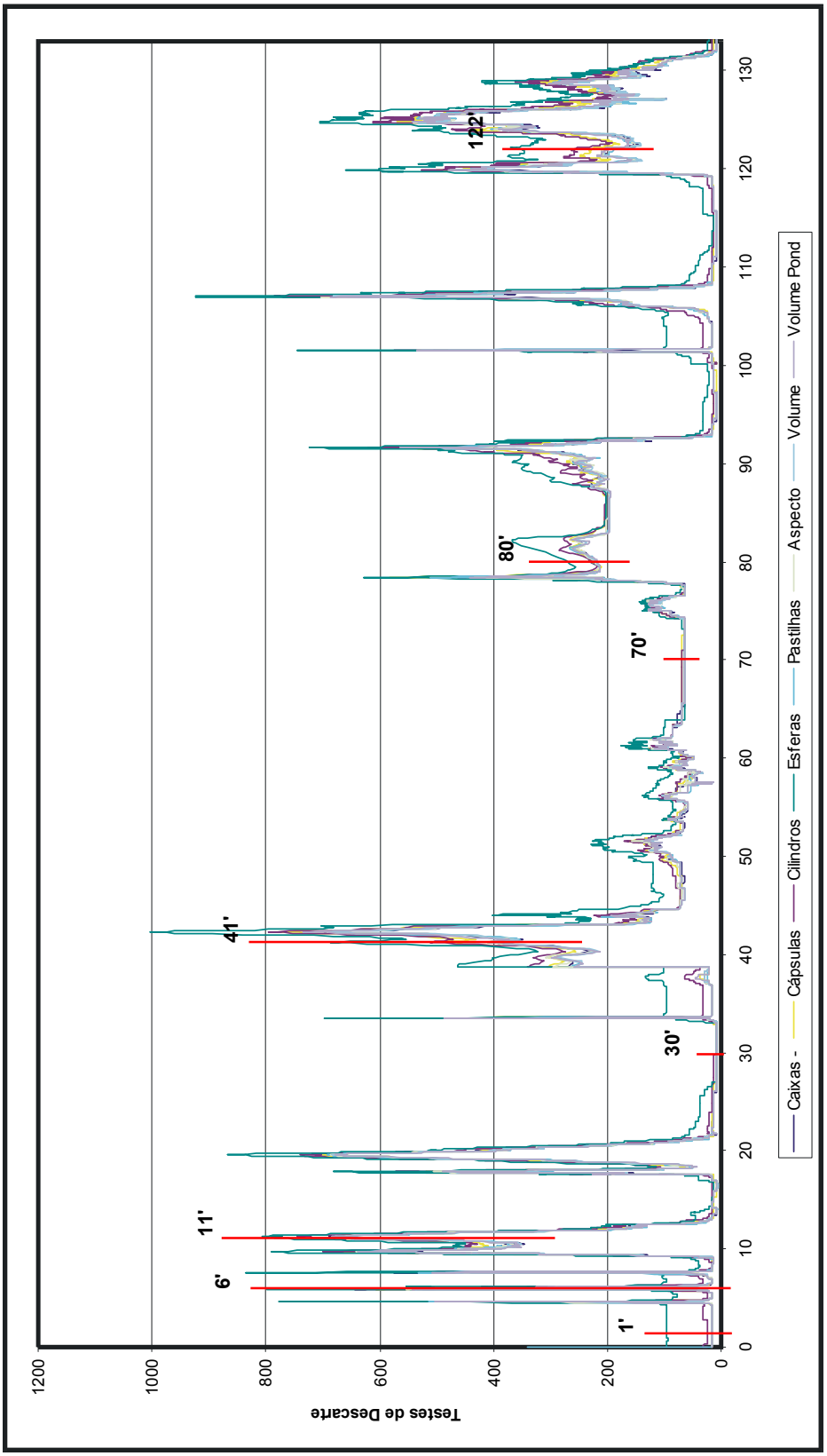

Gráfico 7.13 - Número de testes de descarte contra o volume de visão no segundo modelo de testes - todos os métodos

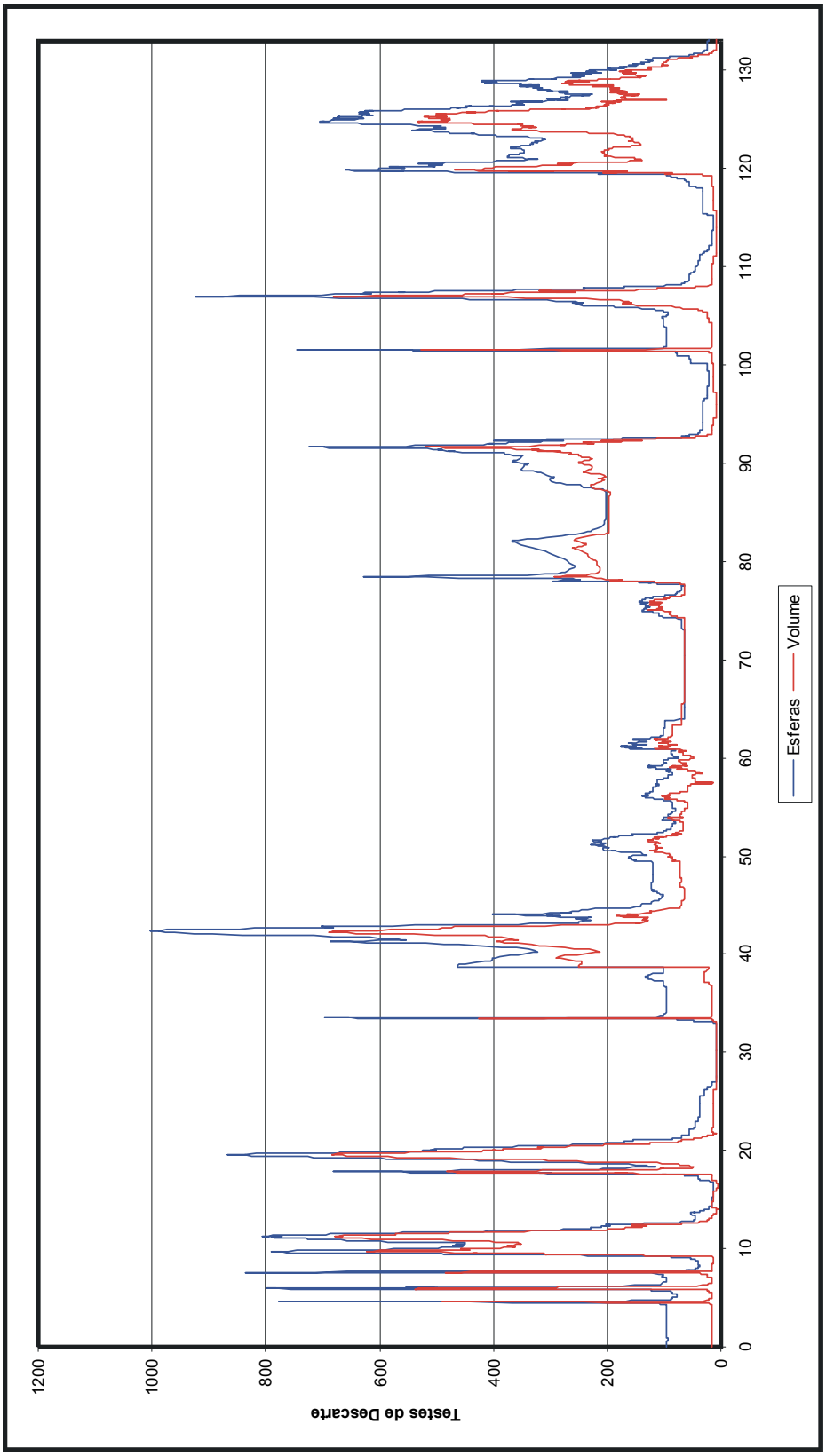

Gráfico 7.14 - Número de testes de descarte contra o volume de visão no segundo modelo de testes - esfera e menor volume

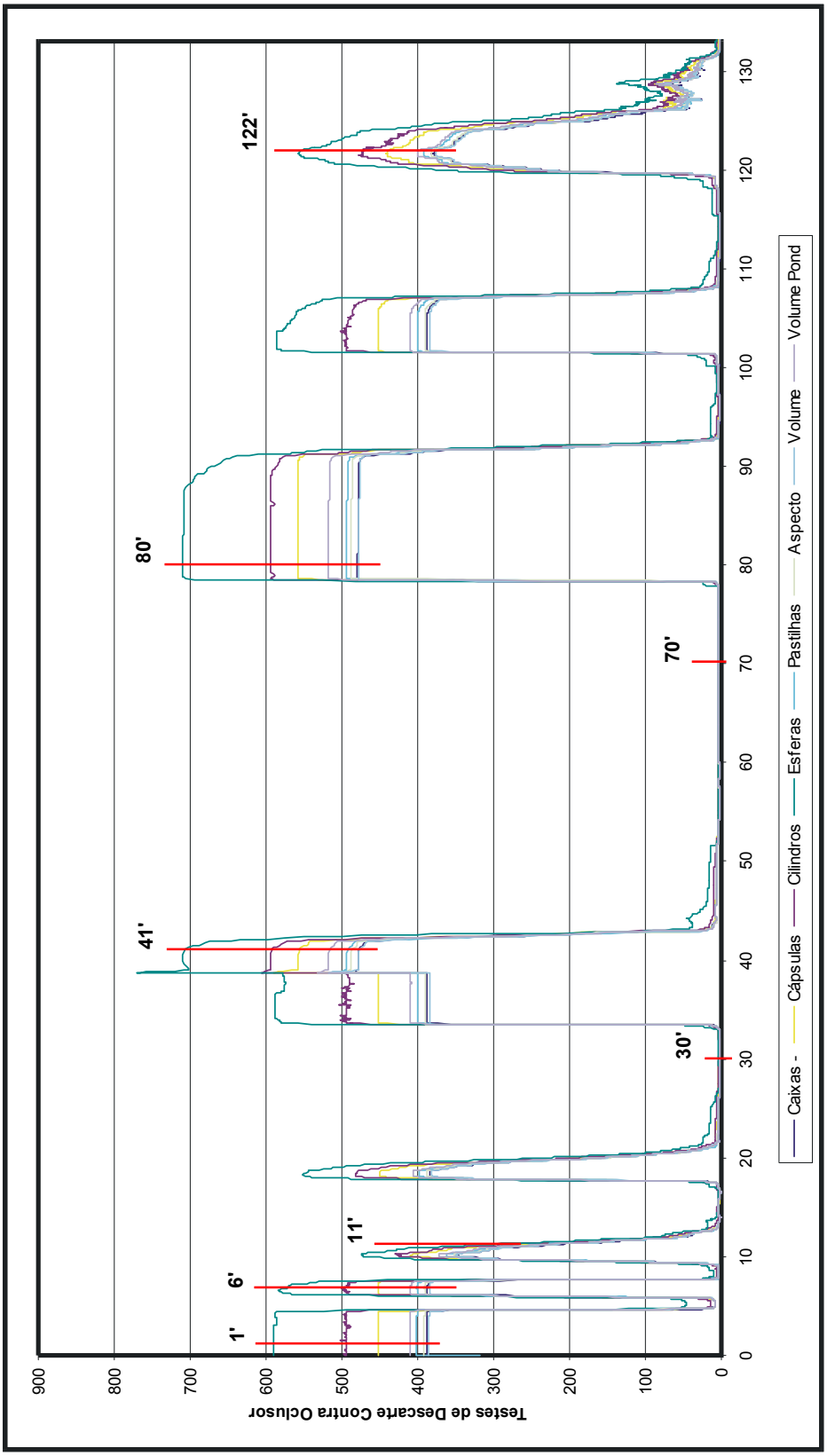

Gráfico 7.15 - Número de testes de descarte contra o oclusor no segundo modelo de testes - todos os métodos

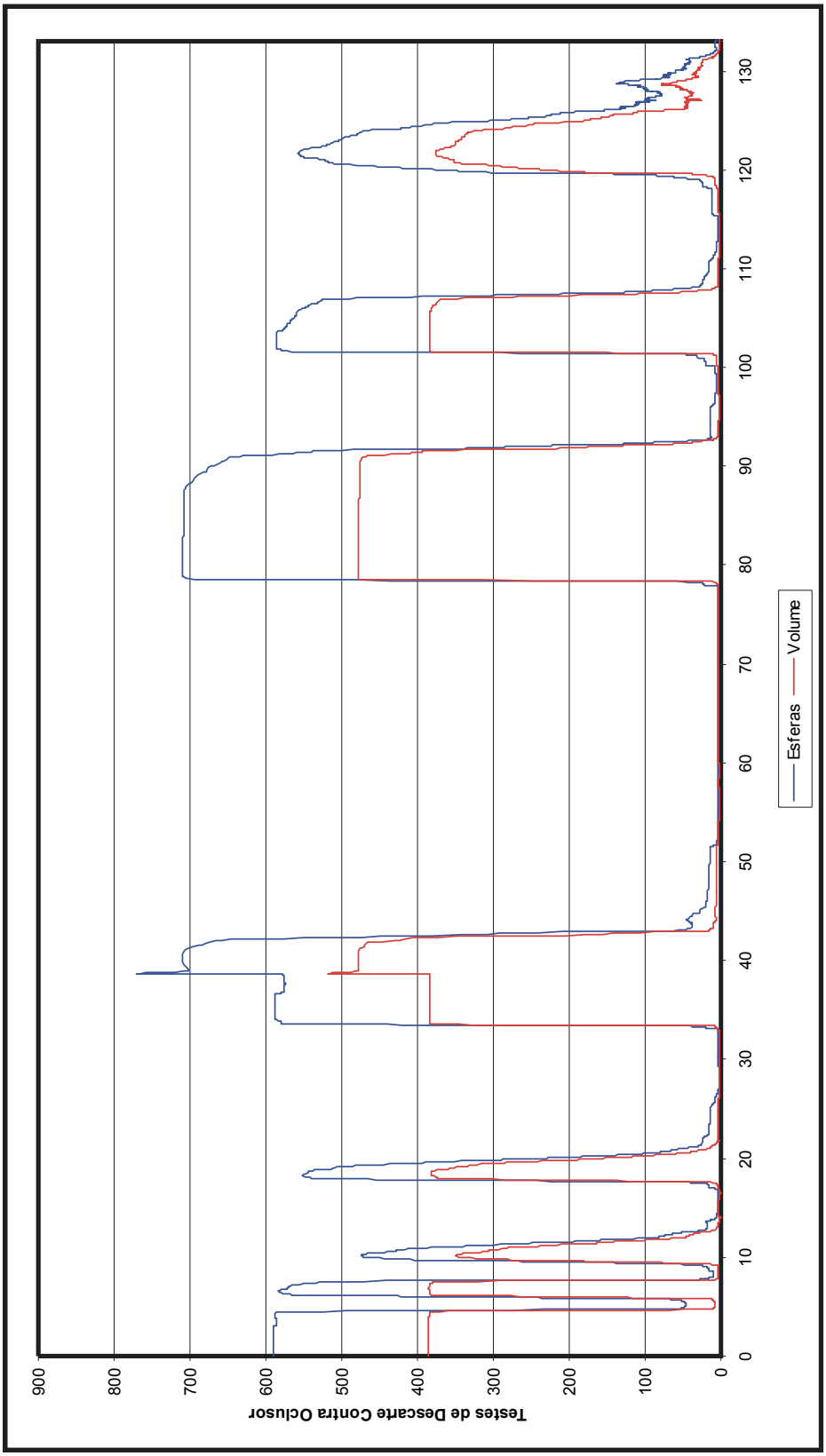

Gráfico 7.16 - Número de testes de descarte contra o oclusor no segundo modelo de testes - esfera e menor volume

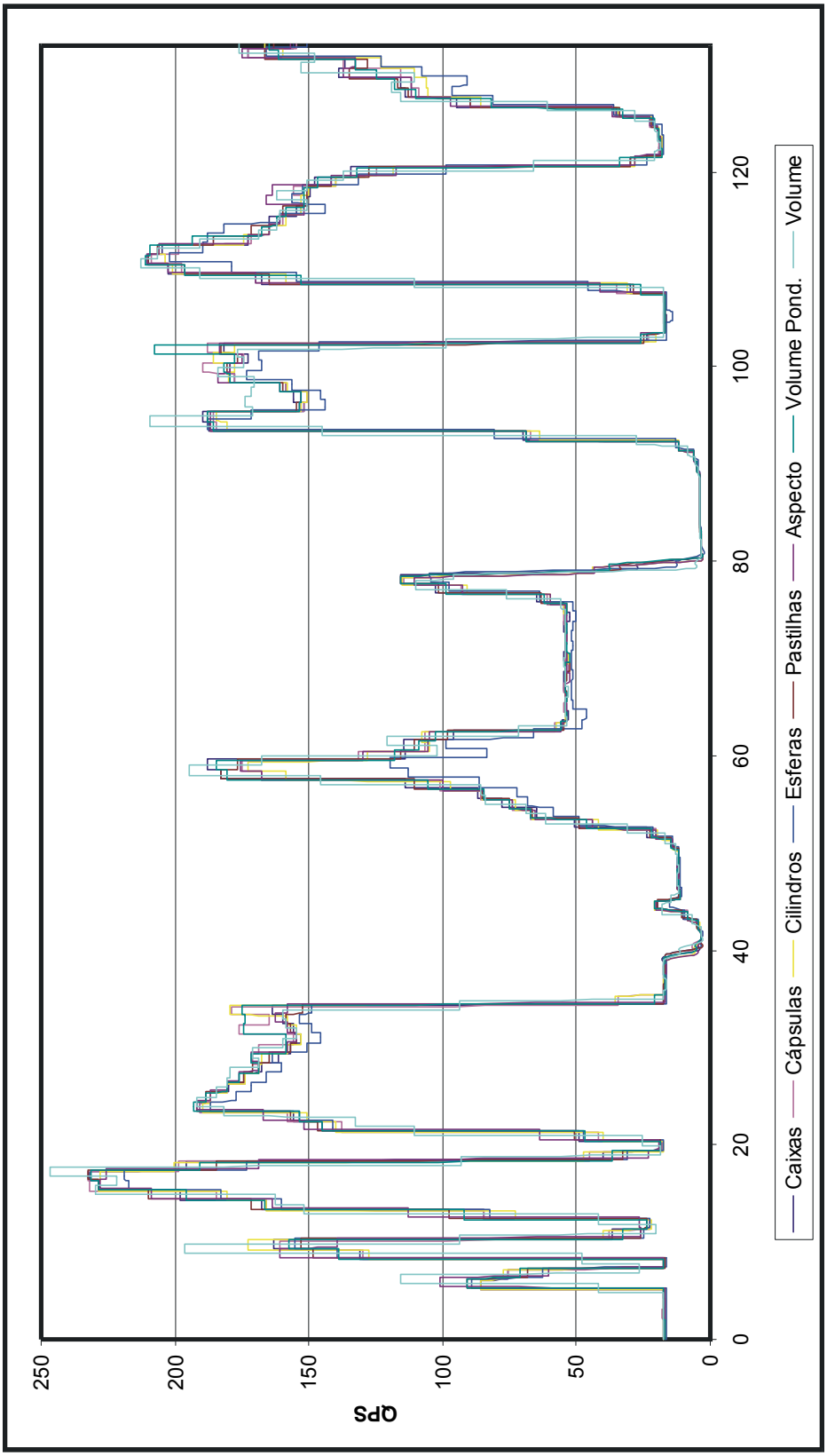

Gráfico 7.17 - Número quadros por segundo restituídos para no segundo modelo de testes - todos os métodos

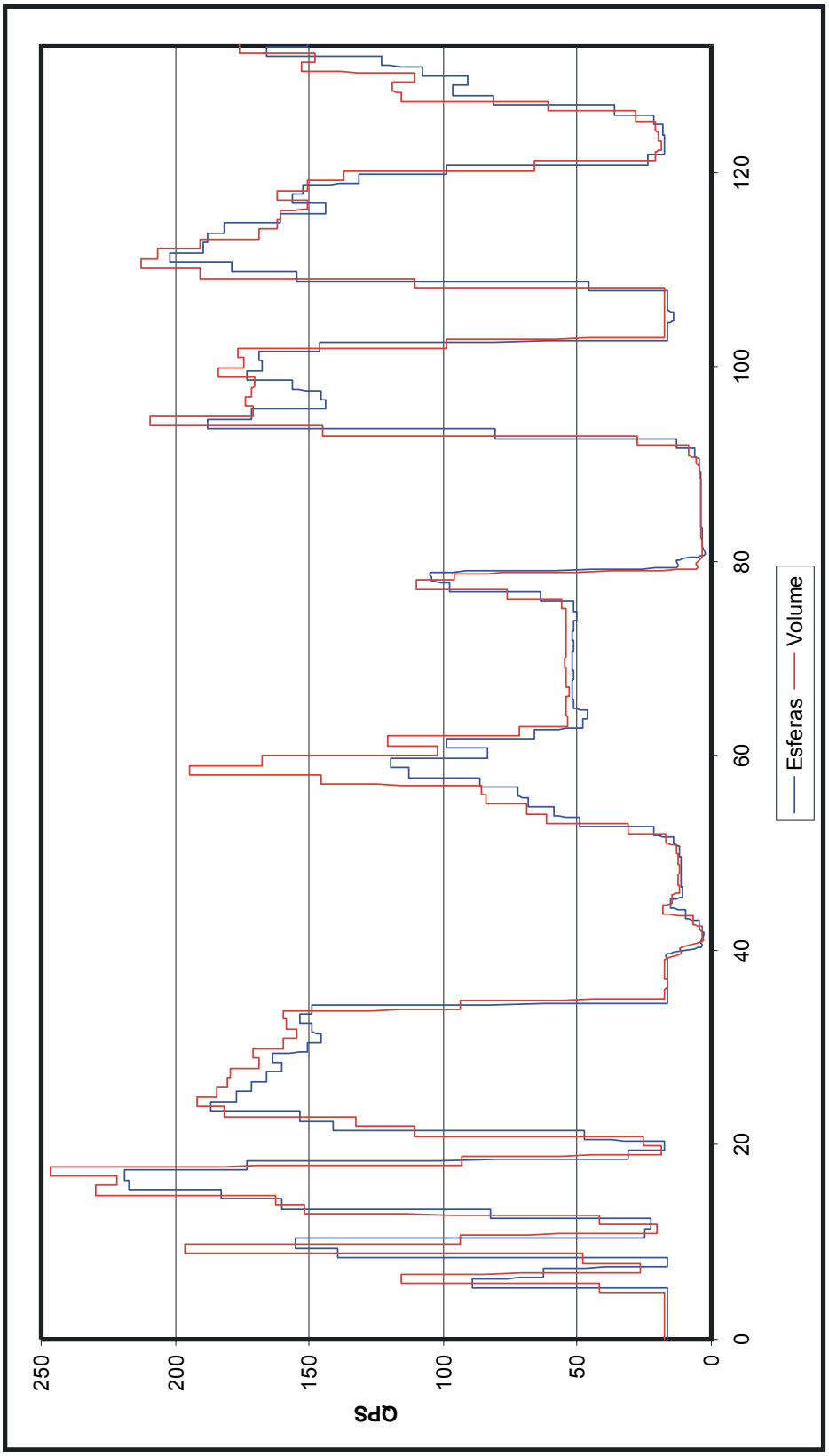

Gráfico 7.18 - Número quadros por segundo restituídos no segundo modelo de testes - esferas e menor volume

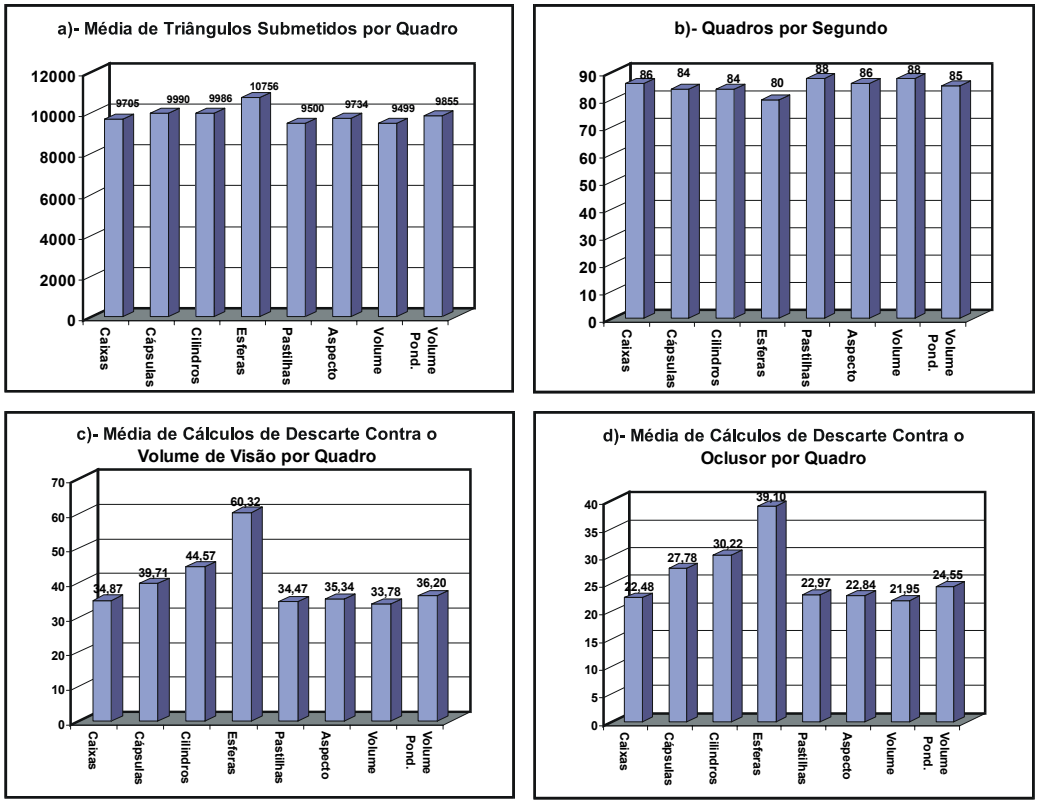

Gráfico 7.19 - Valores acumulados durante os 133 segundos da simulação do segundo modelo de testes de acordo com cada tipo de volume envolvente

CPU%	3Dmau.exe	Other32B	nvoglnt.dll	nv4_disp.dll	nv4_mini.sys	opengl32.dll	Σ	gdi32.dll	ntoskrnl.exe	win32k.sys	hal.dll	ntdll.dll	kernel32.dll	videoptr.sys	Σ	Outros
Sem descarte	4,1	52,5	39,4	0	0,1	0,4	92,4	0	0,8	0,1	0,2	1,4	0,6	0,1	3,2	0,3
Caixas	10,2	23	48	0,4	1	0,4	72,8	0,3	9,2	2,3	0,6	2,5	0,7	0,7	16	0,7
Cápsulas	11,1	22,8	47,9	0,4	1	0,4	72,5	0,3	8,7	2,2	0,6	2,4	0,7	0,7	15,3	0,8
Cilindros	10,9	23,4	46,8	0,4	1,1	0,4	72,1	0,3	9,1	2,1	0,6	2,6	0,7	0,7	15,8	0,9
Esferas	10,2	22,6	48,4	0,4	1,1	0,4	72,9	0,3	9,3	2,2	0,6	2,3	0,7	0,7	15,8	0,8
Pastilhas	10,3	22,7	48,9	0,4	1,1	0,4	73,5	0,3	8,8	2,1	0,5	2,3	0,7	0,7	15,1	0,8
Aspecto	10,6	22,5	48,8	0,4	1	0,4	73,1	0,3	8,7	2,2	0,6	2,3	0,7	0,7	15,2	0,8
Volume	10,4	22,8	49,4	0,4	1	0,4	74	0,3	8,5	2,1	0,5	2,2	0,7	0,7	14,7	0,6
Volume Pond	10,6	22,6	48,9	0,4	1	0,4	73,3	0,3	8,7	2,2	0,5	2,3	0,7	0,7	15,1	0,7

Tabela 7.5 - Uso de tempo de CPU por módulo externo ao programa de testes para o segundo modelo

Durante este conjunto de testes, foi analisado o consumo de ciclos de CPU por módulo interno e externo ao programa de testes. Os resultados são mostrados na Tabela 7.5, que segue a mesma convenção de cores da Tabela 7.2.

Desta seqüência de testes podemos tirar quatro conclusões parciais:

- O volume escolhido para fazer o descarte não teve influência notável no número de triângulos que foram descartados. Isso porque os objetos geométricos têm dimensões pequenas comparadas com o espaço ocupado pela cena toda e são compostos por um número pequeno de polígonos (os maiores comportam cerca de 0,1% do total de polígonos da cena). O resultado é que, com pequenas variações, os triângulos que estavam fora do volume de visão ou oclusos acabaram descartados com alguma precisão, qualquer que tenha sido o método utilizado.
- Embora todos os métodos tenham submetido aproximadamente o mesmo número de triângulos à cadeia de restituição, alguns deles conseguiram dar à aplicação um desempenho moderadamente melhor por terem precisado fazer menos testes de visibilidade. Essa eficiência, contudo, foi parcialmente contrabalançada pelo maior custo de fazer cálculos mais precisos.
- Como aproximadamente 10% do tempo de CPU é consumido pelo programa de testes e na restituição de um quadro e, deste tempo, apenas uma parcela é usada no cálculo de descarte (contra o volume de visão ou contra oclusores), há indícios[11] de que o custo dos cálculos de descarte não são tão relevantes, mas sim a sua eficiência na eliminação de triângulos não visíveis.
- Uma comparação entre as Tabelas 7.2 e 7.5 revela que o programa de testes consumiu mais ciclos de CPU em relação aos módulos externos no segundo modelo por ter de tratar uma quantidade maior de objetos geométricos e, logo, precisar realizar mais testes de visibilidade.

[11] Veja as conclusões parciais do sétimo conjunto de testes.

7.3.7 Sétimo conjunto de testes

O objetivo deste conjunto de testes é avaliar a importância da organização da cena na eficiência dos métodos de descarte. Para isso foi utilizado o segundo modelo de testes, desta vez com uma organização bem mais simples. O primeiro nível da hierarquia possui três entidades: o mar e dois grupos - um que contém os objetos geométricos que formam a plataforma e outro que contém os objetos que formam o sistema de ancoragem. O grafo da cena fica, então, com três níveis apenas, como mostra a Figura 7.9.

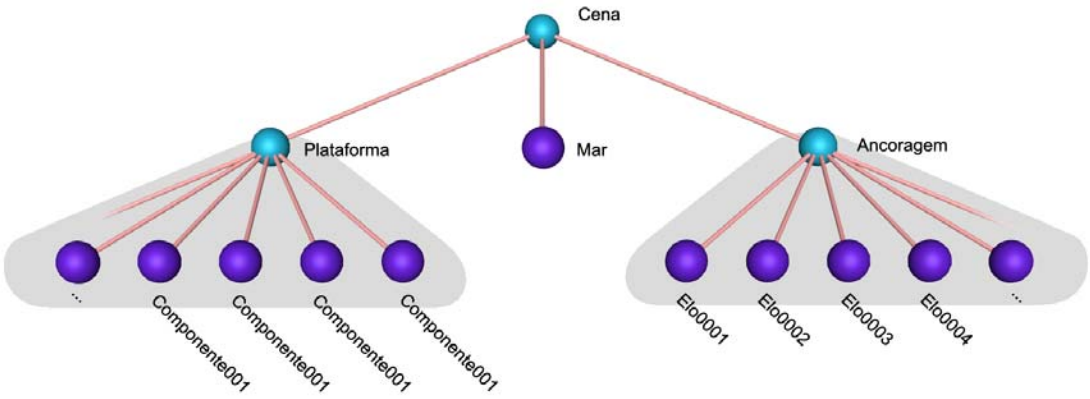

Figura 7.9 - Grafo do segundo modelo de testes sem hierarquia

O Gráfico 7.20 exibe as quantidades de cálculos de descarte contra o volume de visão ao longo da simulação. Caso um dos dois grupos (plataforma ou ancoragem) não possa ser inteiramente rejeitado, os seus elementos devem, então, ser todos testados. Esse fato é responsável pelos patamares do gráfico.

A comparação dos Gráficos 7.20 e 7.11 mostra, como era de se esperar, que esta organização da cena proporciona o descarte da mesma quantidade de triângulos que a organização baseada na árvore-kd. Este paralelo pode ser mais claramente visto no Gráfico 7.24.

Apesar de novamente todos os métodos terem descartado o mesmo número de triângulos, o método que desta vez conseguiu proporcionar o melhor desempenho à aplicação foi o que usa apenas esferas envolventes, como mostram os gráficos 7.22 e 7.23.

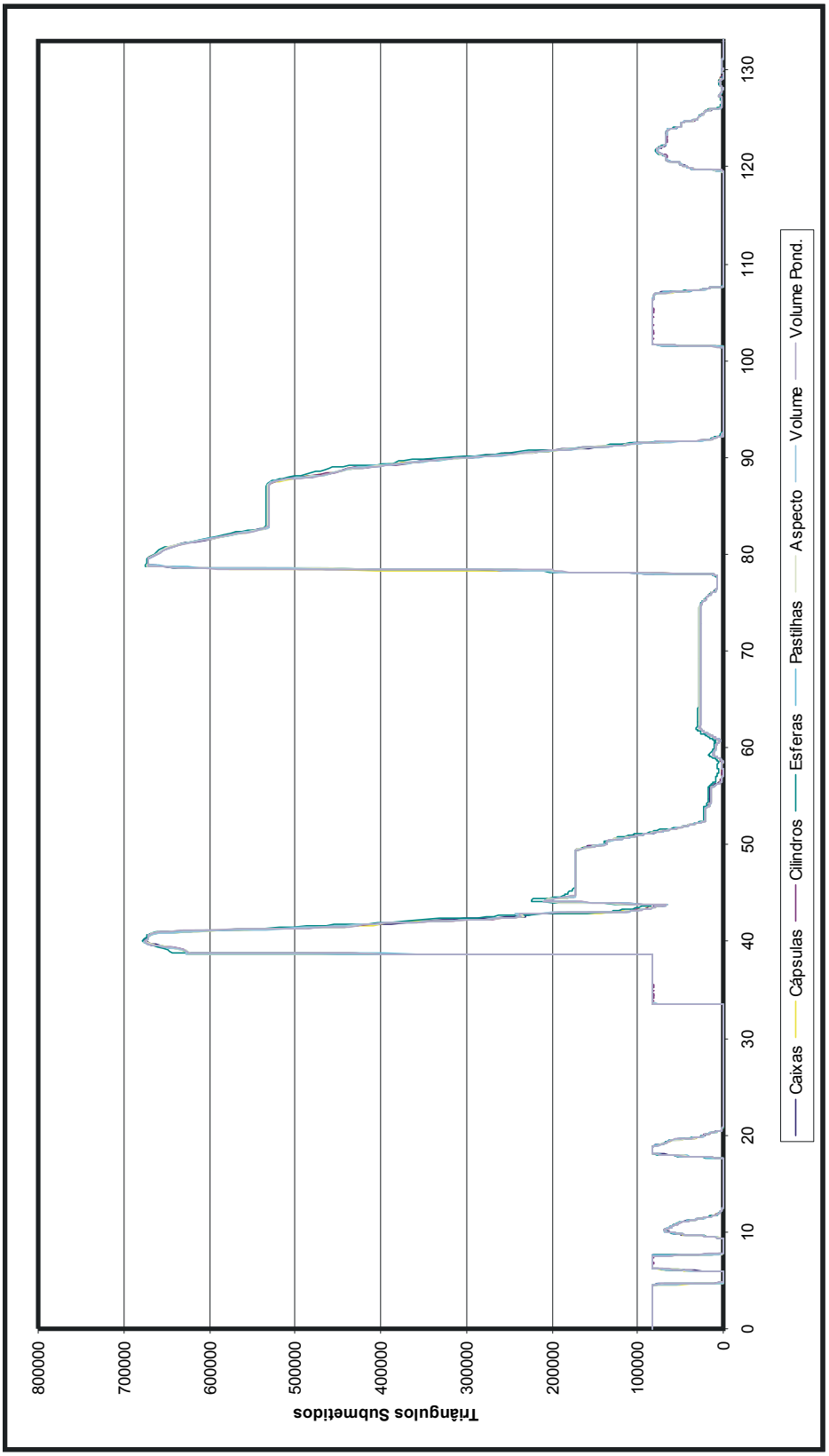

Gráfico 7.20 - Número de triângulos submetidos à cadeia de restituição no segundo modelo de testes sem hierarquia - todos os métodos

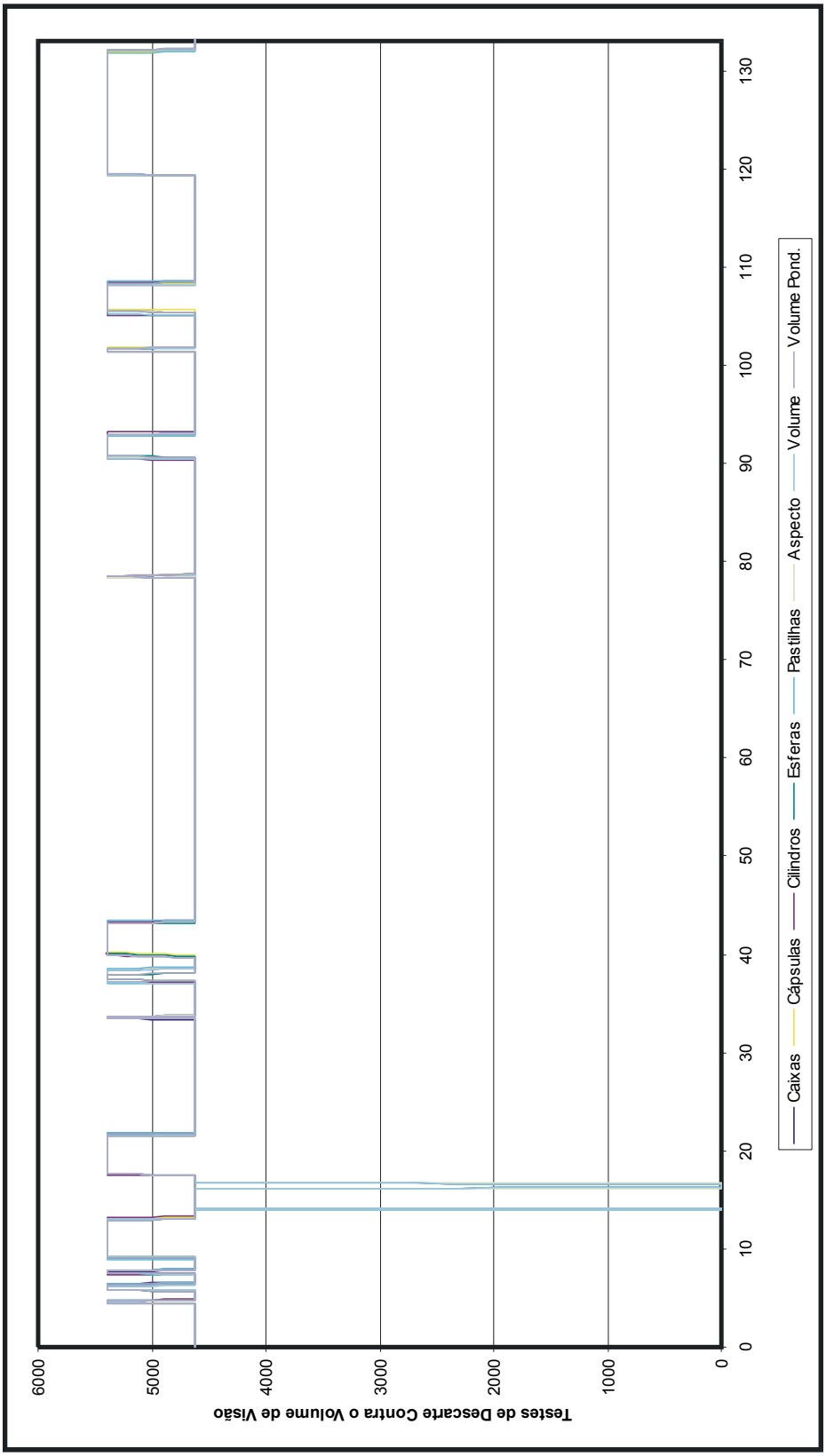

Gráfico 7.21 - Número de testes de descarte contra o volume de visão no segundo modelo de testes sem hierarquia - todos os métodos

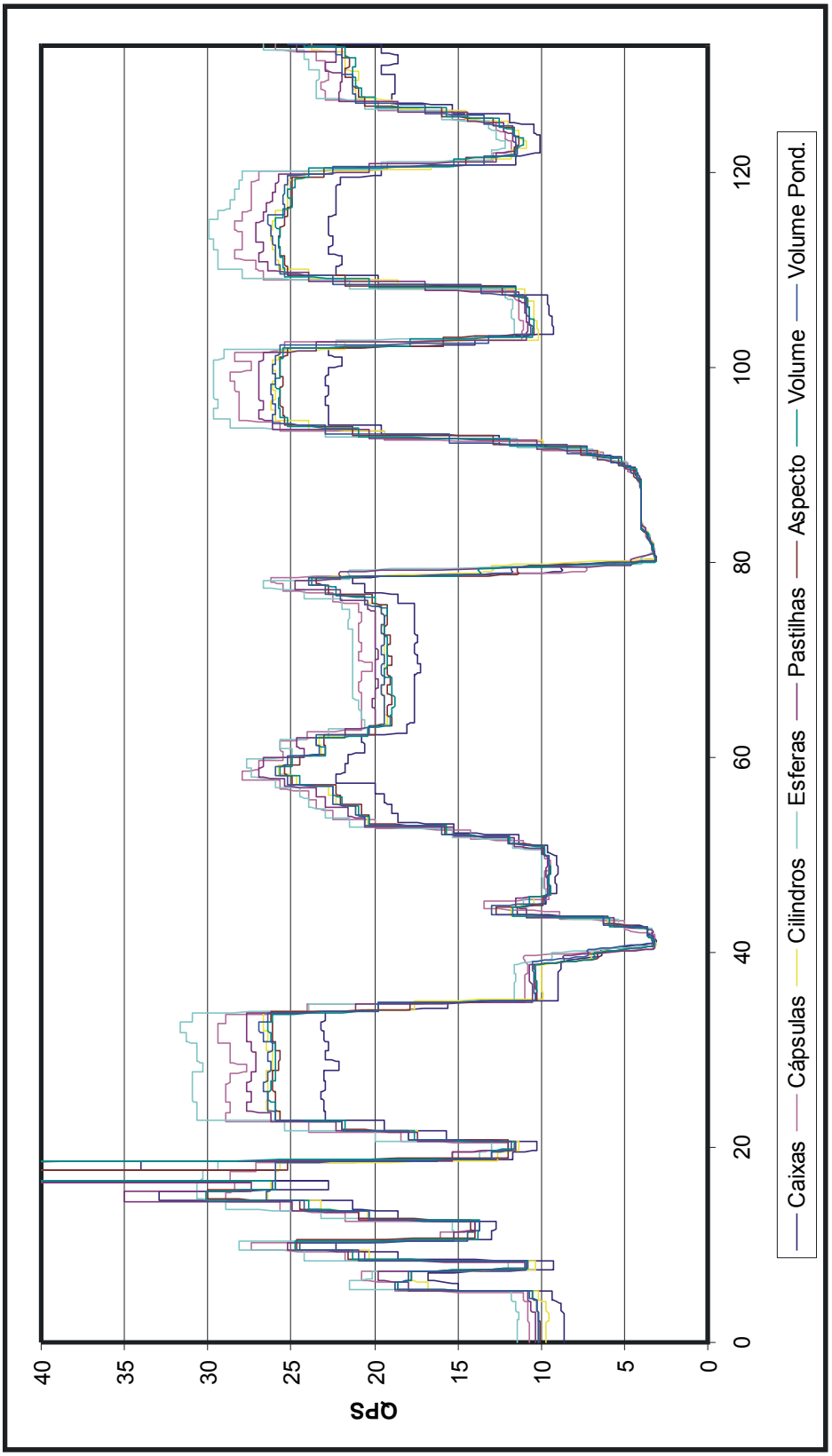

Gráfico 7.22 - Número de quadros por segundo restituídos no segundo modelo de testes sem hierarquia - todos os métodos

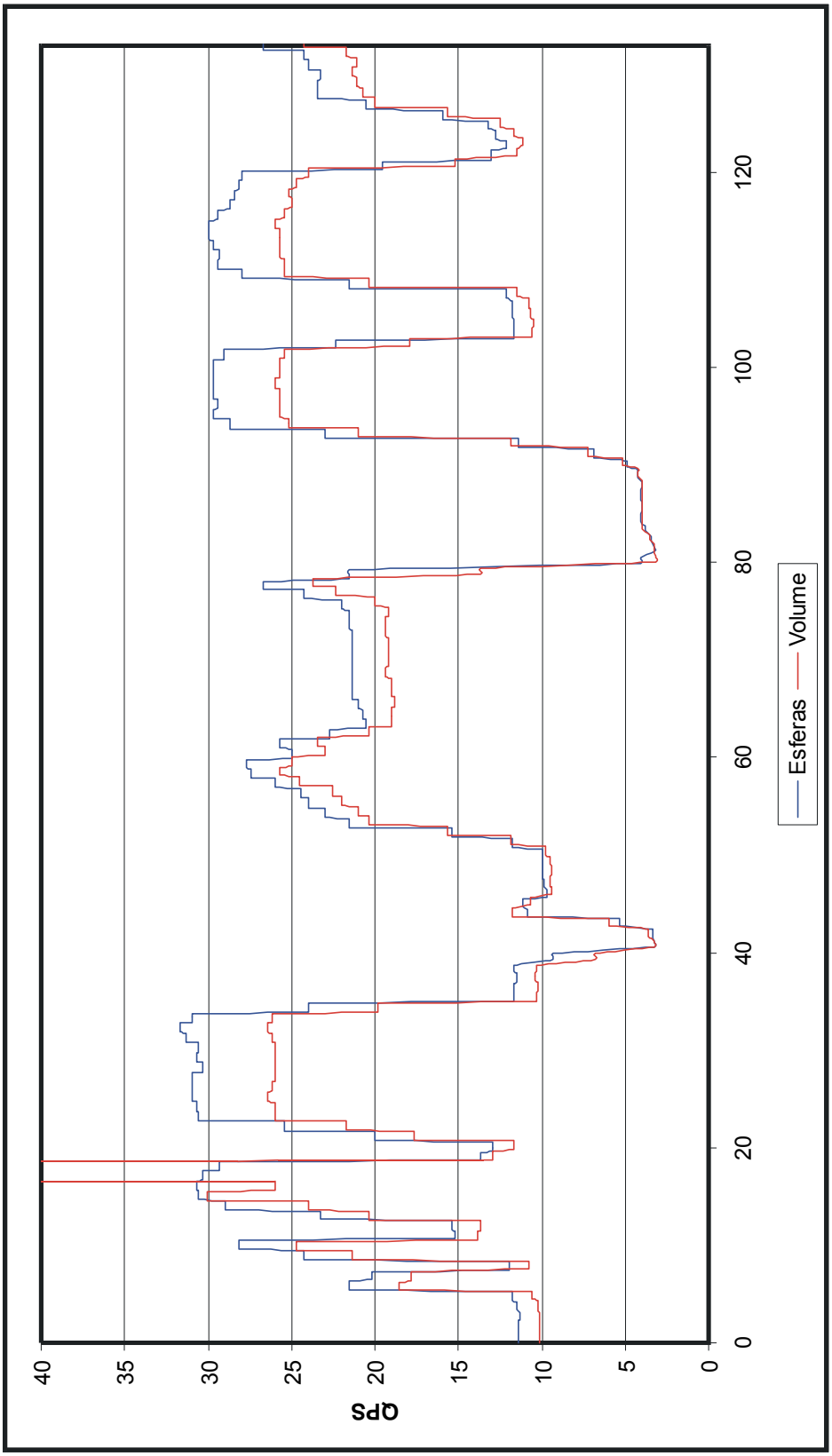

Gráfico 7.23 - Número de quadros por segundo restituídos no segundo modelo de testes sem hierarquia - esferas e menor volume

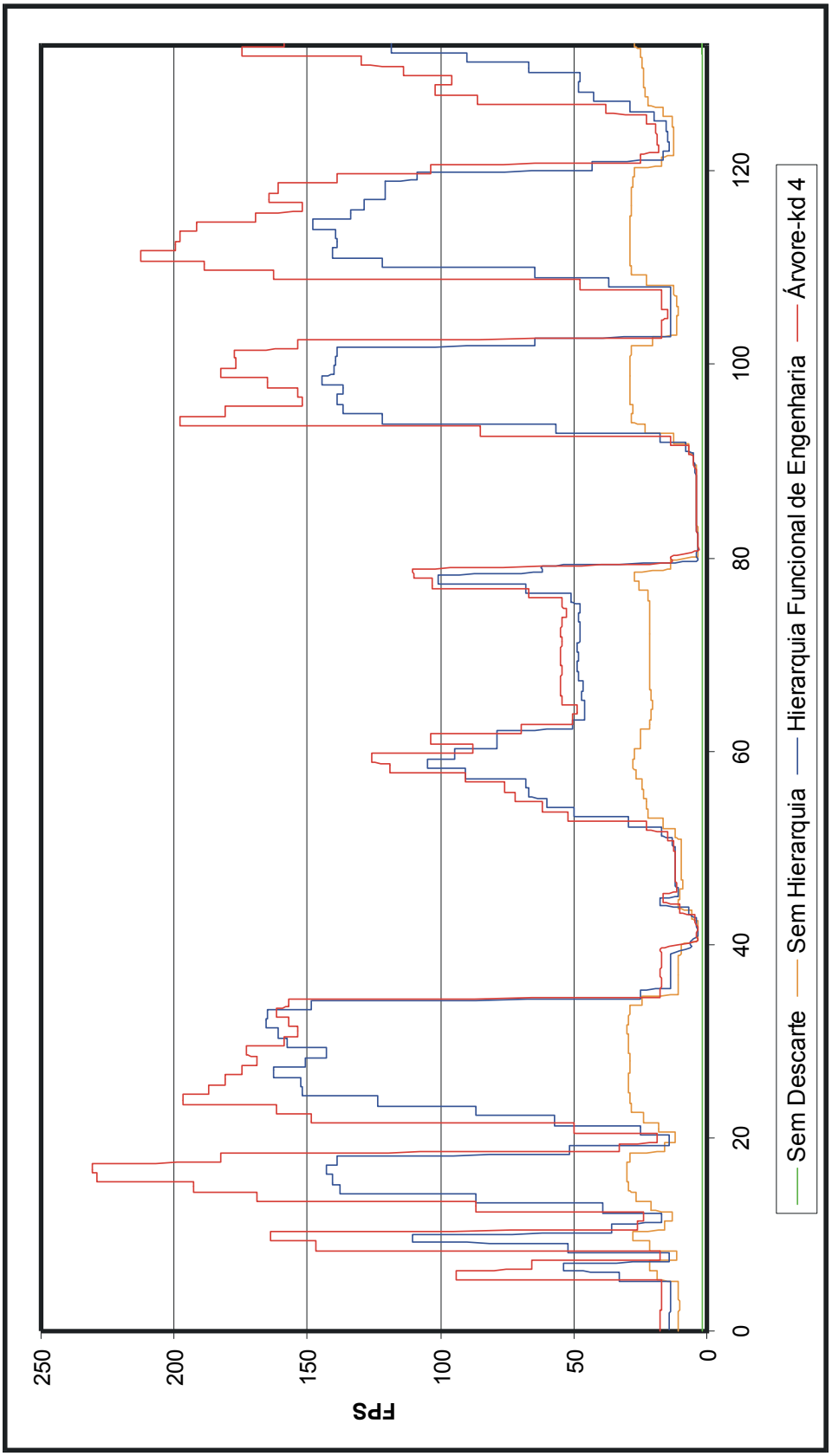

Gráfico 7.24 - Número de quadros por segundo restituídos no segundo modelo de testes com e sem hierarquia usando menor volume para ambos

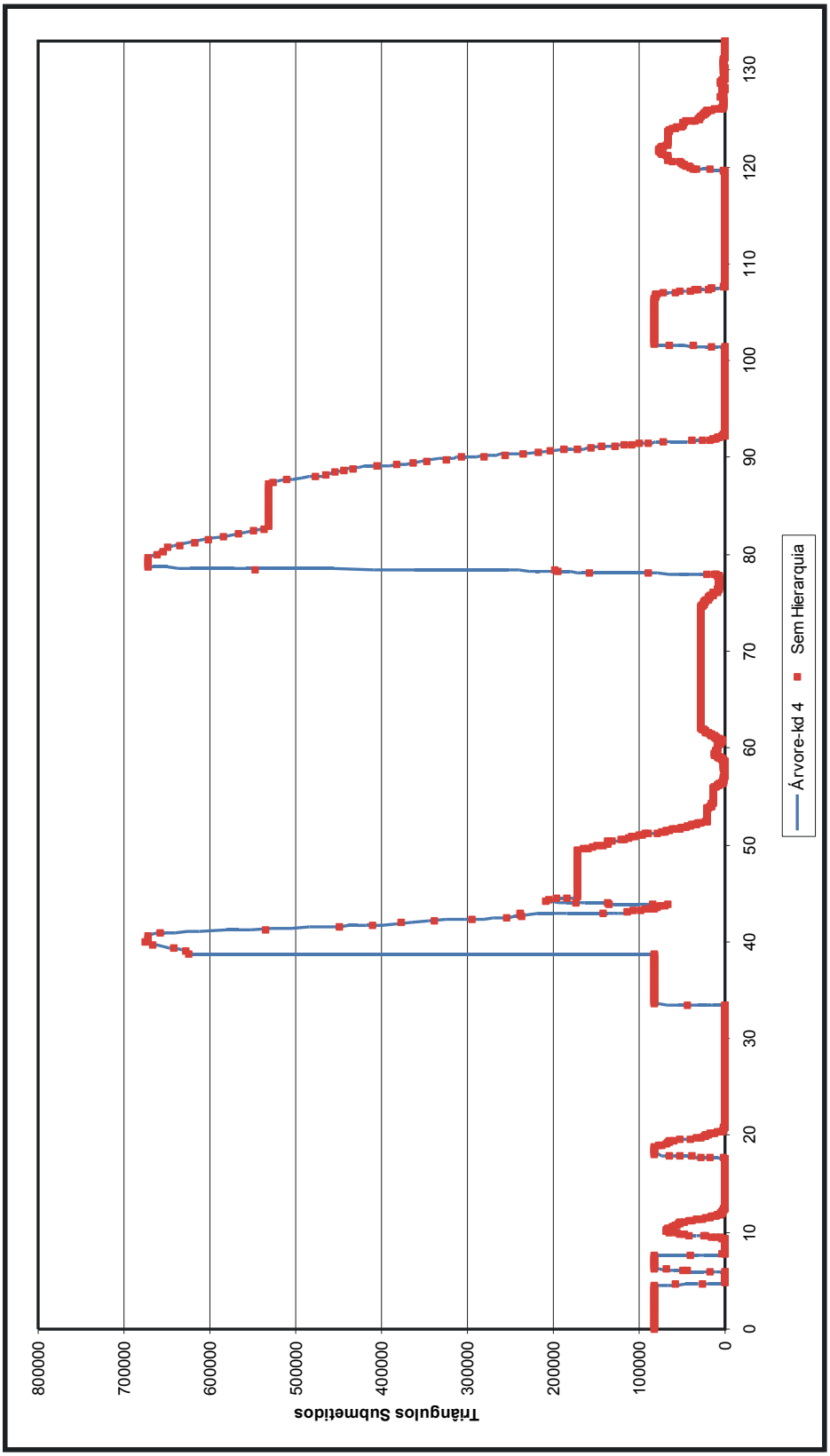

Gráfico 7.25 - Número de triângulos submetidos no segundo modelo de testes com e sem hierarquia usando menor volume em ambos casos

Dado um tipo de volume envolvente, a existência ou não de uma hierarquia pode ter grande influência no número de testes necessários para aceitar ou rejeitar um objeto geométrico. O Gráfico 7.24 mostra essas diferenças de eficiência quando são utilizadas esferas envolventes.

Durante este conjunto de testes, foi analisado o consumo de ciclos de CPU por módulo interno e externo ao programa de testes. A Tabela 7.6 mostra o consumo de ciclos de CPU por módulo externo ao programa de testes seguindo a mesma convenção de cores da Tabela 7.3.

CPU%	3Dmau.exe	Other32B	nvoglnt.dll	nv4_disp.dll	nv4_mini.sys	opengl32.dll	Σ	gdi32.dll	ntoskrnl.exe	win32k.sys	hal.dll	ntdll.dll	kernel32.dll	videoptr.sys	Σ	Outros
Caixas	46,2	15,2	23,6	0,1	0,5	0,5	39,9	0,1	3,3	0,8	0,3	6,8	2	0,3	13,5	0,3
Cápsulas	37,6	16,9	27,7	0,2	0,5	0,5	45,8	0,1	4,3	1,1	0,4	7,5	2,3	0,3	15,9	0,6
Cilindros	42,9	15,6	25,3	0,2	0,5	0,5	42,1	0,1	3,7	1,1	0,3	6,9	2,1	0,3	14,4	0,5
Esferas	36,9	17,2	27,8	0,2	0,5	0,6	46,3	0,1	4,2	1,1	0,4	7,8	2,3	0,3	16,1	0,6
Pastilhas	40,2	16,3	26,7	0,2	0,5	0,5	44,2	0,1	4	1,1	0,3	7	2,2	0,3	14,9	0,6
Aspecto	41,4	15,6	26,4	0,2	0,5	0,5	43,2	0,1	4	1,1	0,3	6,9	2,1	0,3	14,7	0,6
Volume	41,4	15,8	26,2	0,3	0,5	0,5	43,3	0,1	3,9	1,1	0,4	6,9	2,2	0,3	14,8	0,4
Volume Pond	41	16,1	26,2	0,3	0,5	0,5	43,6	0,1	3,8	1,1	0,4	7,1	2,2	0,3	14,9	0,4

Tabela 7.6 - Uso de tempo de CPU por módulo externo ao programa de testes para o segundo modelo sem hierarquia

Desta seqüência de testes, podemos tirar a seguinte conclusão parcial: os conjuntos de testes anteriores podem ter dado a falsa impressão de que os cálculos de descarte têm um custo desprezível. O exemplo acima, contudo, deixa claro que esforço deve ser feito no sentido reduzir o número desses testes. Uma das estratégias para isso é a montagem de uma hierarquia apropriada que permita remover grandes blocos de objetos gráficos o mais cedo possível no caminhamento da árvore do modelo.

7.3.8 Oitavo conjunto de testes

Nesta seqüência, o programa de testes foi instruído a repetir continuamente a simulação baseada no segundo modelo e a fazer ajustes nos critérios de seleção de

volume para descarte com base no aspecto após cada simulação. Outros ajustes foram feitos manualmente. Depois de diversas execuções, observou-se que o desempenho era tanto melhor quanto mais semelhantes eram os volumes selecionados com base no aspecto e os de menor volume interno. Ao passo que é bem mais simples selecionar os volumes de acordo com o seu tamanho, esse tamanho precisa ser calculado. A seleção baseada no aspecto não traz vantagens em tempo de execução, como mostraram os testes anteriores, mas permite economizar tempo de pré-processamento calculando apenas os volumes envolventes mais apropriados.

8 Conclusões e Proposta de Trabalhos Futuros

8.1 Conclusões

Este trabalho apresentou os principais volumes envolventes usados na determinação de objetos potencialmente visíveis, juntamente com algumas formas de calculá-los e de utilizá-los para eliminar objetos que estão fora do volume de visão ou escondidos por um polígono oclusor. Além disso, foi avaliada a eficiência de cada um dos volumes envolventes em cálculos de descarte.

Foram propostas formas de re-estruturar modelos de engenharia com o objetivo de tirar máximo proveito do uso de volumes envolventes no descarte de objetos geométricos não visíveis. As re-estruturações não são restringidas por características peculiares a certos tipos de modelos, como ocorre com árvores-BSP e portais, podendo ser usadas virtualmente em qualquer tipo de modelo.

Os estudos realizados e o desenvolvimento do trabalho apresentaram as seguintes conclusões:

- Para modelos convenientemente organizados, as esferas envolventes apresentaram o pior desempenho no processo de descarte contra o volume de visão e contra polígonos oclusores.
- Pelo menos no caso das esferas, não é compensador o esforço de calcular o volume mínimo. O volume envolvente bem ajustado tem resultados quase tão satisfatórios quanto o mínimo.
- O uso de pastilhas obteve um resultado muito bom. Embora o seu cálculo de descarte implique em um custo quatro vezes maior que o da esfera, elas se adaptam bem a formas bastante variadas, de modo que conseguem eliminar com boa precisão os objetos não visíveis.
- Os resultados obtidos com o uso dos diversos volumes envolventes dependem da complexidade dos objetos gráficos. Quanto mais complexos os objetos, melhores os resultados para os volumes que se adaptam de uma forma mais ajustada.

- A eficiência de um tipo de volume envolvente depende em primeiro lugar da sua capacidade de eliminar triângulos não visíveis e em segundo lugar do seu custo em termos de tempo de CPU. A capacidade de eliminar objetos não visíveis é proporcional à exatidão com que os volumes se adaptam aos objetos. Em virtude disso, o uso do menor volume envolvente obteve em média os melhores resultados por ser o que melhor se ajusta.
- Os cálculos de descarte demonstraram que consomem tempo de processamento bastante baixo mas, apesar disso, é necessário esforço para minimizar o número desses cálculos através da montagem de uma hierarquia apropriada. Tanto a árvore-kd adaptativa quanto a árvore-R estática obtiveram bons resultados nessa organização.
- A árvore-kd e a árvore-R estática, tradicionalmente usadas em indexação espacial, se mostraram boas estruturas para a organização de modelos tridimensionais para visualização interativa.
- O critério de agrupamento funcional do projeto de engenharia provou ser pouco adequado para proceder testes de descarte. Para a visualização, grandes modelos de engenharia devem ser submetidos a um algoritmo de reagrupamento automático.

As principais contribuições desta dissertação são:

- A proposta do uso de dois volumes envolventes para os objetos geométricos (ou grupos destes) que sofrem algum tipo de movimento limitado: um que envolve estreitamente a sua geometria e outro que envolve todas as suas possíveis posições no espaço, o primeiro para testes de visibilidade do próprio objeto e o segundo para calcular os volumes envolventes da hierarquia à qual ele está subordinado.
- A proposta de organização do grafo de cena em uma hierarquia de dois níveis, de forma que tanto objetos estáticos (ou vinculados) como objetos em movimento possam ser acomodados em uma estrutura hierárquica eficiente.
- O estudo comparativo de volumes envolventes para o descarte de objetos não visíveis.

- A proposta de uso de técnicas de indexação espacial para a organização de modelos tridimensionais para visualização interativa.
- A avaliação da eficácia de se calcular volumes mínimos em vez de volumes apenas bem ajustados.

8.2 Trabalhos Futuros

A ênfase desta pesquisa foi o uso de volumes envolventes e a estruturação de modelos para determinar entidades potencialmente visíveis em relação ao volume de visão. No que diz respeito à oclusão, nos limitamos a mostrar como estender a técnica de forma trivial para descartar objetos contra polígonos oclusores convexos. Muito ainda pode ser pesquisado quanto à oclusão, tanto no que diz respeito à determinação dos oclusores compensadores quanto à forma de combiná-los e de proceder os testes.

Para o caso de modelos com componentes dinâmicos com comportamento não determinístico, podem-se estudar os custos de se atualizar a estrutura desses modelos e avaliar em que condições vale a pena recalcular volumes durante o curso da simulação.

Podem-se experimentar outras formas de organização de cena para compará-las às usadas nesta dissertação. Para as árvore-R estáticas pode-se estudar, por exemplo, outras curvas de preenchimento do espaço.

No caso de modelos com objetos complexos, pode ser experimentada a subdivisão desses objetos e o uso de volumes envolventes para testes de visibilidade com essas frações. Indo mais longe, pode-se procurar determinar a quantidade ótima de polígonos por fração elementar do modelo.

Para efetuar os testes de descarte, cada volume foi confrontado contra os planos do volume de visão na seguinte ordem: próximo, distante, esquerdo, direito, superior e inferior. A média suposta de 3,5 planos testados para cada descarte foi comprovada nos testes, uma vez que essa ordem foi sempre mantida. Esse valor pode ser reduzido se for explorada a coerência temporal da simulação: dado que um objeto foi descartado contra um certo plano em um quadro, ele provavelmente será descartado contra o mesmo plano no próximo quadro. Tal plano deve, portanto, ser o

primeiro a ser usado nos testes dos quadros posteriores. Outras otimizações deste tipo podem ser pesquisadas para buscar uma melhora do desempenho do programa de visualização.

Bibliografia

[APPE1967] Appel, A. The Notion of Quantitative Invisibility and The Machine Rendering of Solids. Minuta do ACM Natlional Meeting., 1967, p. 387.

[APPE1967] Appel, A. Some Techniques for Shading Machine Rendering of Solids. Minuta do ACM Natl. Mtg., 1968, p. 387.

[BECK1990] Beckmann, N.; Kriegel, H. e Seeger, B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. Minuta do ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322-331.

[BENT1975] Bentley, J. L. Multidimensional Binary Search Trees Used for Associative Searching. Comm. ACM 18,9, 1975, pp. 509-517.

[BENT1979a] Bentley, J. L. Multidimensional Binary Search in Database Applications. IEEE trans. Software Eng., (4) 1979, pp. 333-340.

[BENT1979b] Bentley, J. L. e Friedman, J. H. Data Structures for Range Searching. ACM Comp Surv., 11(4) 1979, pp. 397-409.

[BERG1997] de Berg, M.; van Kreveld, M.; Overmars, M. e Schwarzkopf, O. Computational Geometry. Springer-Verlag, 1997.

[BIAL1969] Bially, T. Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction. IEEE Transactions on Information Theory, 1969, IT-15(6), pp. 658-664.

[BISH1998] Bishop, L.; Eberly, D.; Whitted, T.; Finch, M. e Shantz, M. Designing a PC Game Engine. IEEE Computer Graphics and Applications, 1998.

[BOUK1970b] Bouknight, W. J. A Procedure For Generation of Three-Dimensional Half-Toned Computer Graphics Presentations. Communications of The ACM, 1970.

[BOUK1970a] Bouknight, W. J. e Kelly, K. C. An Algorithm For producing Half-Tone Computer Graphics Presentations With Shadows and Movable Light Sources. Minuta do AFIPS JSCC, 1970, v. 36, pp. 1-10.

[CATM1974] Catmull, E. E. A Subdivision Algorithm for Computer Display of Curved Surfaces. Tese de Doutorado, Dept. of CS, U. Of Utah, 1974.

[CD] Canvas Draw. www.tecgraf.puc-rio.br/cd

[CLAR1976] Clark, J. H. Hierarchical Geometric Models for Visible Surface Algorithms. Communications on the ACM, 1976, 19(10), pp. 547-554.

[DELO2000] DeLoura, M. (ed.). Game Programming Gems. Charles River Media, 2000.

[DURA1998] Durand, F.; Drettakis, G. e Puech, C. The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. IMAGIS - GRAVIR/IMAG - INRIA, 1998.

[EBER1999] Eberly, D. H. 3D Game Engine Design. Morgan Kaufmann, 1999.

[FALO1989] Faloutsos, C. E.; Roseman, S. Fractals for Secondary Key Retrieval. 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1989, pp. 247-252.

[FERR1999] Ferreira, A. G. Uma Arquitetura Para a Visualização Distribuída de Ambientes Virtuais. Dissertação de Mestrado. Dept. de Informática, PUC-Rio, 1999.

[FERR2000] Ferreira, A.; Cerqueira, R.; Celes, W. e Gattass, M. Multiple Display Viewing Architecture for Virtual Environments over Heterogeneous Network. XII Brazilian Symposium on Computer Graphics and Image Processing, 2000 pp. 83 a 92.

[FUCH1980] Fuchs, H. On Visible Surface Generation by *a Priori* Tree Structures. Computer Graphics, 14, 1980, pp. 124-33.

[GOME1998] Gomes, J. e Velho, L. Computação Gráfica, 1998, v. 1, IMPA.

[GOTT1996] Gottschalk, S.; Lin, M.C. e Manocha, D. OBBTree: A Hierarquical Structure for Rapid Interference Detection. Minuta do ACM Siggraph'96, 1996.

[GOUR1971] Gouraud, H. Continuous Shading of Curved Surfaces. IEEE Trans. on Computer,. 1971. C-20(6), pp. 623-629.

[GÜNT1995] Günther, O. e Gaede, V. Multidimensional Access Methods. Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin, 1995, ISS-16.

[GÜNT1998] Günther, O. Environmental Information Systems. Springer-Verlag, 1998.

[GUTT1984] Guttman, A. R-Trees: A Dynamic Index Structure For Spatial Searching. University of California, Berkeley, 1984.

[HILB1891] Hilbert, D. Uber Die Steitige Abbilding Einer Linie Auf Ein Flachenstuck. Math. Ann., 1891.

[IDSO] Id Software. www.idsoftware.com.

[IM] Image Manager. www.tecgraf.puc-rio.br/im

[IUP] Interface do Usuário Portátil. www.tecgraf.puc-rio.br/iup

[JONE1971] Jones, C. B. A New Approch To The "Hiden Line" Problem. Computer Journal, 1971. 14(3), pp. 232-237.

[KAME1993] Kamel, I. e Faloutsos, C. On Packing R-Trees. Minuta do 2nd International Conference on Information and Knowledge Management, 1993, pp. 490-499.

[NEWE1972] Newell, M. E.; Newell, R. G. e Sancha, T. L. A Solution to The Hidden Surface Problem. Minuta do ACM Nat. Mtg, 1972.

[OOI1987] Ooi, B. C.; McDonnell, K. L. e Sacks-Davis, R. Spatial kd-tree: An Indexing Mechanism for Spatial Databases. Minuta do IEEE Int. Comp. Software & Applicatios Conf. Japan, 1987.

[OOI1992] Ooi, B. C.; Sacks-Davis, R. e Han, J. Indexing in Spatial Databases. Dept. of Inf. Sys. and Comp. Sci. National U. of Singapore, Kent Ridge Singapore 0511, 1987.

[OPENGL] OpenGL. www.opengl.org

[OROU1985] O'Rourke, J. Finding Minimal Enclosing Boxes. International Journal on Computer Information Science, 1985, v. 14, pp. 183-199.

[PARA] Paralelo Computação. www.paralelo.com.br.

[PRES1988] Press, W. H.; Flanery, S.A. e Vetterling, W.T. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, 1988.

[BUIT1975] Bui-Tuong, P. Illumination for Computer Generated Pictures. CACM, 1975, 18(6), pp. 311-317.

[ROBE1963] Roberts, L. G. Machine Perception of Three-Dimensional Solids. TR 315, Lincoln Lab, 1963.

[ROBI1981] Robinson, J. T. The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. Minuta do ACM SIGMOD Int. Conf. on Management of Data, 1981, pp. 10-18.

[ROUS1984] Roussopoulos, N. e Leifker, D. An Introduction to PSQL: A Pictorial Structured Query Language. Minuta do IEEE Workshop on Visual Language, 1985, pp. 17-31.

[ROUS1985] Roussopoulos, N. e Leifker, D. Direct Spatial Search on Pictorial Databases Using Packed R-Trees. Minuta do ACM SIGMOD, 1985.

[SCHÜ1999] Schömer, E.; Sellen, J.; Teichmann, M. e Yap, C. Smallest Enclosing Cylinders. 12th Annual ACM Symposium on Computational Geometry, 1999, S. C13-C14.

[SELL1987] Sellis, T.; Roussopoulos, N. e Faloutsos, C. The R+-tree: A Dynamic Index For Multidimensional Objects. Minuta do 13th Int. Conf. Very Large Data Bases, 1987, pp. 507-518.

[TECG] Tecgraf - Tecnologia em Computação Gráfica. www.tecgraf.puc-rio.br.

[WARN1969] Warnock, J. A Hidden-Surface Algorithm for Computer Generated Half-Tone Picture. Technical Report TR 4-15, NTIS AD-733 671, Comp. Sc. Dept., University of Utah, 1969.

[WATK1970] Watkins, G. S. A Real Time Visible Surface Algorithm. Technical Report UTEC-CSc-70-101, NTIS AD-762 004, Comp. Sc. Dept., University of Utah, 1970.

[WATT1992] Watt, A. e Watt, M. Advanced Animation and Rendering Techniques, Addison-Wesley, 1992.

[WATT2001] Watt, A. e Policarpo, F. 3D Games, Real-time Rendering and Software Technology. Addison Wesley, 2001.

[WEIL1977] Weiler, K. e Atherton, K. Hidden Surface Removal Using Polygon Area Sorting. Computer Graphics. Minuta do SIGGRAPH'77, 1977, 11(2), pp. 214-222.

[WELZ1991] Welzl, E. Smallest Enclosing Discs (Balls and Ellipsoids). In Maurer, H., ed., New Results and New Trends in Computer Science. Lecture Notes in Computer Science, Springer-Verlag, 1991, v. 555, pp. 359-370.

[WOOM1999] Woo, M.; Neider, J.; Davis, T. e Shreiner, D. OpenGL Programming Guide. Terceira edição, Addison-Wesley, 1999.

[WYLI1967] Wylie, C.; Romney, G. W.; Evans, D. C. e Erdahl, A. C. halftone Perspective Drawing by Computer. FJCC, 1967, pp. 49-58.

[ZHAN1998] Zhang, H. Efctive Occlusion Culling for the Interactive Display of Arbitrary Models. Tese de doutorado, Chapel Hill, 1998.