UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Visibilidade em cenas dinâmicas com base numa grade regular

Dissertação de Mestrado

Autor: **Harlen Costa Batagelo**

Orientadora: **Profa. Dr-Ing. Wu, Shin-Ting**

Banca Examinadora: **Prof. Dr. Jorge Stolfi (IC/Unicamp)**
Prof. Dr. Clésio Luis Tozzi (FEEC/Unicamp)
Prof. Dr. Léo Pini Magalhães (FEEC/Unicamp)

Dissertação submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Engenharia Elétrica.

Setembro de 2002

Resumo

Nesta dissertação apresentamos uma arquitetura de determinação de visibilidade para a aceleração da visualização de cenas dinâmicas complexas. Nossa proposta consiste em explorar a flexibilidade de uma grade regular como subdivisão espacial da cena para uma avaliação eficiente de múltiplos objetos dinâmicos de movimentação arbitrária. Embora essa escolha implique uma perda de vantagens exclusivas a estruturas de dados hierárquicas, introduzimos diversas otimizações nas etapas de manutenção de objetos dinâmicos escondidos e percurso de células da grade regular de modo a fortalecer o benefício do uso dessa estrutura em cenas dinâmicas. Além disso, utilizamos princípios de rasterização para acelerar a determinação de visibilidade no espaço do objeto usando as regiões opacas da cena como oclusores. Também utilizamos coerência temporal para reduzir a complexidade de determinação de visibilidade em proporção ao número de objetos dinâmicos visíveis e discutimos os resultados de testes de desempenho conduzidos numa implementação da arquitetura.

Abstract

In this dissertation we present an architecture of visibility determination for the acceleration of the visualization of complex dynamic scenes. Our proposal consists of exploiting the flexibility of a regular grid as spatial subdivision of the scene for an efficient evaluation of multiple dynamic objects of arbitrary motion. Although this choice implies a loss of exclusive advantages of hierarchical approaches, we introduce several optimizations in the stages of maintenance of hidden dynamic objects and traversal of cells in the regular grid in order to strengthen the benefits of using this structure in dynamic scenes. In addition, we use raster principles to accelerate the visibility determination in object-space using the opaque regions of the scene as occluders. We also apply temporal coherence to reduce the complexity of visibility determination in proportion to the number of visible dynamic objects and discuss the results of timing tests performed in an implementation of the architecture.

Agradecimentos

Meus sentimentos de gratidão se estendem a todos aqueles que me auxiliaram, ostensivamente ou não, no desenvolvimento deste trabalho. São muitos os nomes, de tal modo que não conseguiria enumerá-los sem correr o risco de esquecer alguém. Essas pessoas incluem todos os professores, inúmeros colegas e amigos que tive a oportunidade de conhecer na Unicamp, na FEEC e mais particularmente no DCA.

Agradeço em especial a meus pais (Rui e Ivonete) e ao meu irmão (Christian), pelo incentivo e suporte sem os quais a realização deste trabalho não teria sido possível. Também sou muito grato à minha orientadora (Profa. Ting) por ter acreditado na proposta desse trabalho, pela atenção e interesse demonstrado em todos os momentos; ao Prof. Ilaim, por ter despertado em mim o interesse pela pesquisa durante o curso de graduação e me incentivado à realização do mestrado; à Banca Examinadora, pelos comentários e sugestões; à CAPES, agência que financiou este trabalho. Por fim, e não poderia deixar de citar alguém tão importante, devo agradecer a Meg, minha *irmã menor*, pela companhia sincera e carinho dispensado tanto nos momentos de trabalho como de descontração. Muito obrigado.

Aos meus pais.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

HOM	Mapa de Oclusão Hierárquico (*Hierarchical Occlusion Map*).
HZB	Z-Buffer Hierárquico (*Hierarchical Z-Buffer*).
PVS	Conjunto Potencialmente Visível (*Potentially Visible Set*).
TBV	Volume Limitante Temporal (*Temporal Bounding Volume*).
H	Matriz de oclusão.
I	Matriz de identificadores.
O	Matriz de oclusores.
O_s	Matriz de oclusores estáticos.
T	Matriz de TBVs otimizados.

Capítulo 1

Introdução

É crescente o número de aplicações em Computação Gráfica envolvendo a exibição de cenas complexas que precisam ser visualizadas e modificadas em tempo real.[1] Nessa cenas, o número de primitivas pode atingir facilmente a ordem de vários milhões de polígonos, ultrapassando a capacidade atual de processamento do *hardware* gráfico para uma visualização em tempo de interatividade (em torno de 10 quadros por segundo). Por outro lado, a necessidade do uso de sombreamentos[2] cada vez mais complexos também tem contribuído para aumentar o custo de renderização dessas cenas. Essa necessidade é principalmente impulsionada pela nova geração de *hardware* gráfico com capacidade de programação de sombreadores [27] que possibilita a síntese de imagens em *hardware* com qualidade antes só possível em aplicativos especializados, como o *Renderman* da *Pixar*.[3] Entretanto, mesmo com o crescimento espantoso da capacidade de processamento dos aceleradores gráficos nos últimos anos,[4] as exigências dos usuários de aplicações em Computação Gráfica têm permanecido invariavelmente além dos recursos disponíveis. Contrastando com estes argumentos, nos ambientes complexos geralmente utilizados nessas aplicações, a fração da geometria visível com relação a qualquer ponto de vista corresponde a apenas uma pequena parte da cena. Assim, mesmo em cenas grandes e complexas, a visualização pode ser feita de forma eficiente se o tempo de renderização não depender do tamanho inteiro da cena, mas apenas do número relativamente pequeno de primitivas atualmente visíveis ao observador, *i.e.*, se a complexidade de renderização for sensível à saída. Cenas com essas características são chamadas de cenas *densamente oclusas* [3, 10] e são encontradas com cada vez mais freqüência em modelos complexos de CAD (*Computer Aided Design*), cenas urbanas, cenas arquitetônicas internas e jogos.

A exibição de cenas densamente oclusas pode ser largamente acelerada por algoritmos que procu-

[1] Consideramos como uma cena um volume no espaço 3D onde se encontram não só os objetos de interesse como também o observador. Portanto a exibição desses cenários subentende uma visualização imersiva.

[2] Sombreamento (*shading*) é o processo de determinar a contribuição da iluminação numa superfície de modo a definir sua cor em *pixels* da imagem.

[3] http://www.pixar.com/renderman

[4] Recentemente Huddy [23] avaliou que o crescimento do desempenho do *hardware* gráfico tem correspondido à aplicação da *Lei de Moore* elevada ao cubo, *i.e.*, a cada seis meses (ao invés dos dezoito meses previstos por Moore) as placas gráficas adquirem o dobro de sua capacidade de processamento.

ram descartar rapidamente os objetos trivialmente escondidos ao observador, evitando assim processá-los desnecessariamente ao longo do fluxo de renderização. De fato, o objetivo dessas abordagens é reduzir o tempo de renderização numa taxa proporcional apenas ao número de primitivas visíveis ao observador, ignorando o processamento do restante da cena. Técnicas baseadas nessa estratégia são conhecidas como técnicas de *descarte de visibilidade* (*visibility culling*) e vem constituindo uma área importante de pesquisa nos últimos anos [8, 9, 30]. Contudo, o tratamento eficiente de visibilidade em cenas *dinâmicas* densamente oclusas tem sido uma questão ainda pouco explorada em Computação Gráfica [9]. Cenas dinâmicas são cenas que, enquanto visualizadas em tempo real por um observador que se movimenta arbitrariamente pelo espaço, permitem que suas primitivas se movimentem também de forma arbitrária e possibilitam inserções e exclusões de primitivas a qualquer momento. Ao contrário, numa cena estática, apenas o observador se move enquanto a cena é exibida em tempo real.

Em geral, algoritmos de descarte de visibilidade envolvem etapas custosas de pré-processamento para construir estruturas de dados de particionamento da cena – subdivisões espaciais – que aumentam a eficiência de operações de consulta de visibilidade em tempo de execução. Tais consultas são normalmente aceleradas por estruturas de dados hierárquicas, uma vez que partes significativas da cena podem ser detectadas como trivialmente escondidas em níveis altos da hierarquia. Entretanto, a atualização de relações de hierarquia em tempo de execução pode não ser trivial e, dependendo do número de atualizações, proibitivamente lenta. Por outro lado, tais alterações são necessárias para refletir mudanças de posição e dimensão de objetos dinâmicos de movimentação arbitrária. Em resumo, quanto mais complexa é a estrutura de dados construída numa etapa de pré-processamento, mais difícil é a sua atualização em tempo de execução. Por outro lado, são muitas as aplicações que, devido a sua natureza altamente interativa, como realidade virtual, jogos, simulações e ambientes virtuais multiusuários, começam a exigir algoritmos de visibilidade dinâmica para tratar cenas cada vez mais complexas.

Embora existam pesquisas devotadas à aceleração da atualização de subdivisões espaciais hierárquicas em cenas dinâmicas (veja o Capítulo 2), nesta dissertação motivamos uma discussão sobre a aplicação de grades regulares em tais cenas, partindo da hipótese de que a flexibilidade dessas estruturas tornam-nas ideais à manipulação de objetos dinâmicos em tempo real. Por outro lado, enquanto os benefícios da maioria das abordagens hierárquicas não parece ultrapassar o custo de manutenção de um grande número de objetos dinâmicos, grades regulares têm a desvantagem de subdividir áreas esparsas e densas da cena de maneira uniforme. Pode haver, dessa forma, um desperdício de células da grade regular, tanto do ponto de vista de memória consumida (desperdício sem grande impacto, já que o custo atual dos módulos de memória tem sido baixo), mas também – e principalmente – de tempo de execução da determinação de visibilidade, já que o algoritmo acaba percorrendo várias células que na verdade estão vazias. Embora esses motivos façam com que grades regulares normalmente não sejam as melhores escolhas para cenas estáticas, em cenas dinâmicas essas desvantagens podem ser contrabalançadas por sua flexibilidade.

O algoritmo de visibilidade proposto para trabalhar na grade regular é baseado em trabalhos anteriores de determinação de visibilidade, especialmente nas abordagens de Schaufler *et al.* [36] e Sudarsky

e Gotsman [40, 39, 41]. Schaufler *et al.* descrevem um algoritmo de visibilidade que utiliza a agregação de regiões opacas da cena como oclusores no espaço do objeto. Os autores mostram que essa forma de identificar oclusores é mais efetiva do que a utilizada por algoritmos que empregam a própria geometria da cena como oclusores no espaço do objeto. Já o trabalho de Sudarsky e Gotsman propõe uma técnica de manutenção de objetos dinâmicos escondidos na subdivisão espacial que reduz o custo de determinação de visibilidade nessas cenas a uma complexidade sensível à saída.

A originalidade do nosso algoritmo reside principalmente nas otimizações utilizadas para adequar eficientemente os trabalhos citados à abordagem baseada na grade regular. Entretanto, embora o algoritmo original proposto por Schaufler *et al.* seja limitado a cenas estáticas e não tenha sido projetado para realizar a determinação de visibilidade em tempo de execução, a maioria dos benefícios que os autores introduziram é herdada pelo nosso, como a habilidade de detectar rapidamente as regiões escondidas da cena através de um processo de *fusão de oclusores* no espaço do objeto e construção de *oclusores virtuais* na subdivisão espacial. Além disso, a técnica introduzida por Sudarsky e Gotsman, utilizada para reduzir o custo de determinação de visibilidade numa complexidade sensível à saída, tem seus benefícios fortalecidos pelas otimizações que utilizamos para adaptá-la à grade regular. Finalmente, nosso algoritmo não depende de qualquer etapa de pré-processamento para seleção prévia de oclusores ou pré-computação de conjunto de objetos potencialmente visíveis, como ocorre especialmente com a técnica original de Schaufler *et al.*

1.1 Contribuições

As principais contribuições que apresentamos nesta dissertação procuram minimizar as dificuldades devidas ao uso de grades regulares em algoritmos de descarte de visibilidade e assim encorajar o uso dessas estruturas como subdivisões espaciais em cenas dinâmicas. Em especial, exploramos uma correspondência natural e intuitiva entre células da grade regular e *pixels* do *frame buffer* (ambos são regulares e uniformes) para:

- desenvolver um procedimento eficiente de percurso de células contidas no volume de visão numa ordem de frente para trás a partir do observador. Desse modo, o descarte de visibilidade pode ser realizado em tempo de execução, de forma incremental, à medida que oclusores mais próximos do observador são encontrados.

- classificar as regiões escondidas da cena através da "rasterização" de volumes de oclusão na grade regular. Assim, o cálculo de oclusão é determinado somente entre as células da grade regular e possibilita explorar a coerência de varredura e coerência de amplitude da rasterização (*scan-line coherence* e *span coherence*, segundo Foley *et al.* [16]) para melhorar ainda mais o desempenho desta etapa. Além disso, essa estratégia isenta o algoritmo da realização de testes geométricos potencialmente custosos de inclusão de células em volumes de oclusão.

- otimizar a discretização de *volumes limitantes temporais* na grade regular para reduzir a sobrecarga da manipulação de objetos dinâmicos escondidos. Em especial, podemos utilizar características de coerência temporal do observador para representar os volumes limitantes temporais de forma simplificada na subdivisão espacial. Além disso, esses volumes podem ser representados em 3D por apenas três matrizes 2D, reduzindo de forma significativa o número de células que precisam ser atualizadas.

1.2 Organização do Trabalho

No Capítulo 2 discutimos o problema da determinação de visibilidade em cenas densamente oclusas e revisamos as técnicas existentes de descarte de visibilidade voltadas a cenas dinâmicas. O caráter ainda incipiente do tratamento dessas cenas se reflete no número reduzido de literatura especialmente devotada a essa área de pesquisa. Desse modo, procuramos revisar as técnicas que, embora inicialmente destinadas a cenas estáticas, podem ser estendidas com relativa facilidade a cenas dinâmicas. Além disso, revisamos a literatura que tem contribuído com ferramentas para o desenvolvimento de algoritmos de visibilidade dinâmica, mesmo que não descrevendo algoritmos completos de visibilidade.

No Capítulo 3 desenvolvemos a idéia que propomos. Inicialmente classificamos o nosso algoritmo segundo a taxonomia sugerida por Cohen-Or *et al.* [9], depois introduzimos as definições das estruturas de dados utilizadas e então apresentamos uma descrição sucinta dos passos do algoritmo. O detalhamento da arquitetura é apresentado inicialmente para cenas 2D e depois estendido para cenas 3D. Essa forma de apresentação facilita o entendimento da arquitetura proposta e explora o fato do uso de grades regulares permitir uma transição de 2D para 3D com relativamente poucas modificações.

No Capítulo 4 discutimos os resultados dos testes de desempenho baseados na implementação da arquitetura no caso tridimensional. Finalmente, no Capítulo 5, concluímos o trabalho com um resumo da arquitetura proposta, suas vantagens e desvantagens, e descrevemos as sugestões que deverão ser seguidas em trabalhos futuros como forma de corrigir limitações ainda existentes.[5]

[5] A versão eletrônica desta dissertação pode ser encontrada em http://www.dca.fee.unicamp.br/projects/prosim/dynvis. Neste sítio também estão disponíveis os códigos-fonte das implementações, demonstrações da arquitetura no caso 2D/3D e artigos relacionados.

Capítulo 2

Trabalhos Correlatos

A palavra visibilidade envolve uma grande variedade de problemas em Computação Gráfica, incluindo desde a tradicional remoção de superfícies escondidas até problemas de geração de sombras, cálculo de fatores de forma para radiosidade, reconhecimento de objetos, planejamento de trajetórias e o chamado problema de *guarda de galerias de arte*.[1] O detalhamento de cada uma dessas questões abrange diversas áreas da Computação Gráfica, tais como Processamento de Imagens e Geometria Computacional e ultrapassa o escopo deste trabalho. Nesta dissertação nos concentramos no problema de descarte de visibilidade, que compreende a questão explorada pela arquitetura que propomos. Para um estudo interdisciplinar dos demais problemas citados sugerimos o trabalho de Durand [12].

Um levantamento abrangente sobre algoritmos de descarte de visibilidade foi feito recentemente por Cohen-Or *et al.* [9] e constitui também numa excelente introdução ao estudo dessas técnicas. Uma discussão sobre o uso de técnicas de descarte de visibilidade em jogos é apresentada por Riegler [35]. Outros levantamentos são encontrados no livro de Möller e Haines [30], no trabalho de Zhang [47] e no trabalho de Aila e Miettinen [2].

Técnicas de descarte de visibilidade têm sido fundamentais na aceleração da visualização de cenas densamente oclusas. Nessas cenas a maioria dos objetos pode ser detectada como trivialmente escondida com um tempo de execução proporcional ao número de primitivas visíveis e sem esforços computacionais de análises mais acuradas. Isso difere dos métodos tradicionais de remoção de superfícies escondidas que procuram identificar os fragmentos exatos das primitivas visíveis com um tempo de processamento pelo menos linear no tamanho da entrada. Segundo Sudarsky *et al.* [40, 39], um algoritmo de visibilidade só pode ser definido como de descarte de visibilidade se o seu tempo de execução por quadro de exibição for sensível à saída, *i.e.*, se o seu tempo de execução for de $O(n + f(N))$, onde N é o número de primitivas no modelo inteiro, n é o número de primitivas visíveis e $f(N) \ll N$. O fator $f(N)$ é uma sobrecarga inevitável

[1] O problema de guarda de galerias de arte pode ser descrito da seguinte forma: Assumindo que uma galeria de arte é descrita por uma planta 2D poligonal e que existem guardas cuja visão tem alcance infinito e campo de visão de 360°, quantos guardas são necessários para guardar (enxergar) completamente a cena e como posicioná-los? Muitas variantes foram estudadas para essa questão, que foi revelada ser NP-difícil. Um tratamento detalhado pode ser encontrado no livro de O'Rourke [32].

imposta pelo modelo e pelo algoritmo de visibilidade. Entretanto, espera-se que f seja uma função sublinear, tal como $f(x) = \log x$ de modo que $f(x) \ll x$, para um x suficientemente grande.

Na estratégia de descarte de visibilidade, o conjunto de objetos detectados como potencialmente visíveis, chamado de PVS (*Potentially Visible Set*), é geralmente uma estimativa *conservadora* dos objetos visíveis da cena. Um PVS é classificado como conservador quando necessariamente contém todos os objetos pelo menos parcialmente visíveis ao observador e possivelmente alguns (espera-se que sejam poucos) escondidos. Para uma renderização correta, apenas este pequeno conjunto é filtrado por um algoritmo de remoção de superfícies escondidas, usualmente o *z-buffer* em *hardware* [6].

A eficácia de um algoritmo de descarte de visibilidade pode ser medida pela quantidade de objetos totalmente escondidos que são erroneamente relatados como potencialmente visíveis. Idealmente, um algoritmo de visibilidade deve enviar ao fluxo de renderização um PVS que contenha apenas os objetos pelo menos parcialmente visíveis ao observador e apenas estes. Tais resultados podem ser alcançados com os chamados algoritmos *analíticos* de visibilidade, como os baseados no *grafo de aspecto*[2] e no *complexo de visibilidade*[3]. Essas técnicas pré-calculam todas as possíveis configurações de visibilidade existentes entre os objetos de uma cena e armazenam essas informações com coerência suficiente para possibilitar a determinação eficiente do conjunto exato de objetos visíveis para qualquer ponto de vista em tempo de execução. Infelizmente, essas técnicas não são praticáveis para cenas complexas devido à elevada complexidade de espaço das estruturas de dados geradas ($O(n^9)$ no grafo de aspecto e $O(n^4)$ no complexo de visibilidade), além da dificuldade para realizar atualizações eficientes das estruturas em tempo de execução. É esperado, entretanto, que os algoritmos de descarte de visibilidade produzam PVSs próximos aos que seriam relatados por um algoritmo de solução ideal e que sejam rápidos o suficiente para permitirem a aceleração da visualização em decorrência da diminuição do número de objetos processados no fluxo de renderização.

2.1 Coerência

Um conceito largamente utilizado para a otimização de algoritmos de visibilidade e sem o qual não seria possível construir técnicas de descarte de visibilidade, é o conceito de *coerência* – ou grau de similaridade local existente entre as partes de um ambiente [16]. Tipicamente a coerência ocorre quando as propriedades de um objeto variam suavemente de um lugar para outro. Sua exploração significa reutilizar um cálculo inicial para regiões próximas, sem precisar alterar o cálculo, ou com alterações incrementais mais eficientes do que seriam obtidas recalculando a informação desde o princípio. Ademais, o cálculo só é refeito nos casos (espera-se que sejam isolados) nos quais a coerência é perdida, o que significa que houve uma mudança brusca, ou *descontinuidade*, na relação de similaridade entre as partes consideradas. Vários

[2] O grafo de aspecto (*aspect graph*) é uma estrutura de dados de descrição de mudanças qualitativas (topológicas) de visibilidade através de uma representação desses eventos visuais num espaço de pontos-de-vista (*viewpoint-space*) [34].

[3] O complexo de visibilidade (*visibility complex*) é uma estrutura de dados de descrição de mudanças qualitativas (topológicas) de visibilidade através de uma representação desses eventos visuais num espaço de linhas (*line-space*) [12, 13]

tipos de coerência foram identificados inicialmente por Sutherland *et al.* [42], embora outras formas também existam e às vezes se confundam entre si. As principais coerências que utilizamos nesta dissertação são as seguintes:

- *Coerência espacial* (*spatial coherence*). Compreende vários tipos de coerência normalmente relacionadas ao espaço do objeto. Por exemplo, se dois objetos são mutuamente visíveis, uma alteração infinitesimal da posição de um objeto não altera a relação de visibilidade. Em outro exemplo, se um objeto A está mais distante do observador do que um objeto B, não é preciso verificar se as faces de B estão sendo obstruídas pelas faces de A. Da mesma forma, se a caixa limitante de um objeto está escondida, então o objeto que está dentro dessa caixa também está e não é preciso verificar separadamente a visibilidade de cada face.

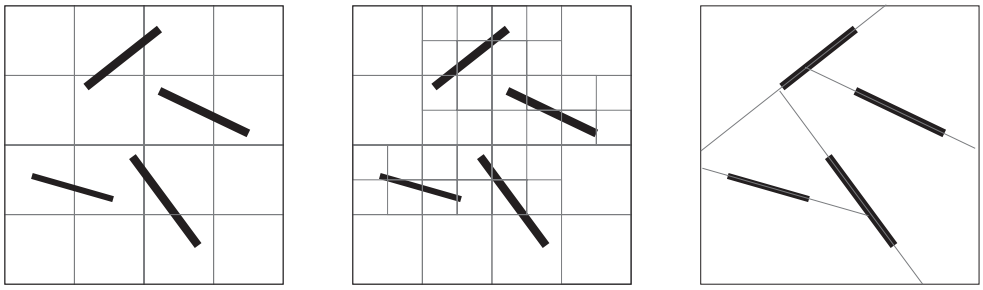
- *Coerência temporal* (*temporal coherence*). Se um objeto de movimentação suave está sendo escondido por alguma parte da cena, esse objeto ainda estará sendo escondido após uma alteração infinitesimal do tempo. Num exemplo, se um carro andando a uma velocidade constante de 5 Km/h acaba de ser escondido por um muro de 100 metros de comprimento, o carro certamente não estará visível pelos próximos 72 segundos. Coerência temporal pode ser utilizada para reduzir o número de vezes que um objeto dinâmico é atualizado numa subdivisão espacial.

- *Coerência de varredura* (*scan-line coherence*). O conjunto de *pixels* de cada objeto visível rasterizado numa linha de varredura de uma imagem difere pouco do conjunto de *pixels* da próxima linha de varredura. Logo, o cálculo realizado em um linha de varredura pode ser reaproveitado para a linha seguinte, com poucas modificações.

- *Coerência de amplitude* (*span coherence*). Um grupo de *pixels* adjacentes é normalmente coberto pela mesma face visível numa linha de varredura. Por exemplo, para determinar os *pixels* cobertos por um polígono convexo numa linha de varredura, basta determinar o *pixel* mais à esquerda e mais à direita da amplitude horizontal; todos os *pixels* intermediários estarão necessariamente cobertos.

2.2 Subdivisões Espaciais

A estratégia mais comum para classificar relações de visibilidade em arquiteturas de descarte de visibilidade é através do particionamento da cena por uma subdivisão espacial. Essa abordagem leva em conta a propriedade de que objetos convexos, alinhados e bem estruturados podem responder a consultas de visibilidade de forma mais eficiente do que em um caso geral, não-estruturado. Por exemplo, um algoritmo de descarte de visibilidade pode utilizar como subdivisão espacial uma estrutura de dados (em geral



Regular, uniforme Regular, não-uniforme Não-regular

Figura 2.1: Subdivisões espaciais.

hierárquica) tal como uma *octree*[4] [18], *kD-tree*[5] [16] ou árvore BSP[6] [17], para organizar a cena de tal modo a permitir a exploração de coerência espacial entre os objetos e assim possibilitar o descarte de partes grandes do ambiente sem a necessidade de testar cada primitiva do modelo. Esse processo de organização da cena é chamado de *discretização* e as regiões discretas do espaço particionado são chamadas de *élulas* ou *voxels.* Uma comparação sobre o comportamento de diferentes subdivisões espaciais em algoritmos de descarte de visibilidade para cenas estáticas foi realizada por Meissner *et al.* [29].

Segundo Yagel e Ray [46], subdivisões espaciais podem ser classificadas de acordo com o modo como as células são organizadas e distribuídas na cena. Essas estruturas podem ser não-regulares e regulares, uniformes ou não-uniformes (Figura 2.1). Uma subdivisão espacial é classificada como regular quando suas células são alinhadas aos eixos coordenados. Por outro lado, é classificada como uniforme se todas as células têm o mesmo tamanho. Exemplos de subdivisões espaciais regulares incluem as grades regulares, *octrees* e *kD-trees*. A grade regular, entretanto, além de ser regular também é uniforme.[7] Entre os exemplos de subdivisões espaciais não-regulares destaca-se a árvore BSP, pois divide a cena segundo os planos definidos pelas superfícies dos objetos, planos esses que geralmente não são alinhados aos eixos.

Cada tipo de particionamento tem seus pontos positivos e negativos que podem variar de acordo com a aplicação desejada. Para os propósitos desse trabalho, podemos comparar as subdivisões espaciais segundo suas características de flexibilidade e adaptação às cenas. De um modo geral, quanto mais flexível é a estrutura de dados, menos adaptativa ela é e vice-versa.

Grades regulares uniformes têm a vantagem de serem bastante flexíveis às atualizações de células em tempo de execução, pois o seu particionamento é fixo e o percurso das regiões discretas não é imposto pelo percurso de uma árvore, como ocorre com outras estruturas, como a *octree* e *kD-tree*. Por outro lado,

[4] *Octree*, ou *octal tree*: decomposição hierárquica do espaço tridimensional de modo que cada região é dividida em oito octantes por três planos alinhados aos eixos coordenados.

[5] *kD-tree*, ou *k-dimensional tree*: decomposição hierárquica de um espaço *k*-dimensional de modo que cada região é dividida em duas por um hiperplano alinhado a um eixo coordenado.

[6] Árvore BSP, ou árvore binária de particionamento espacial (*Bynary Space Partitioning tree*): decomposição hierárquica de um espaço *k*-dimensional de modo que cada região é dividida em duas por um hiperplano.

[7] O termo *grade regular* para designar uma subdivisão espacial regular-uniforme é um abuso de linguagem. Por outro lado, já foi consolidado na literatura e naturalmente subentende que as células também são uniformes.

uma vez que as partições são fixas, as regiões esparsas da cena (com poucos objetos) são particionadas da mesma forma que as regiões densas (com muitos objetos), havendo então um desperdício de células que denota um mau aproveitamento da coerência espacial.

Octrees e *kD-trees* têm a vantagem de particionar a cena de forma adaptativa, *i.e.*, de acordo com a concentração de objetos em diferentes partes da cena e por isso exploram a coerência espacial de forma mais inteligente especialmente em cenas onde os objetos não estão distribuídos uniformemente pelo espaço. Árvores BSP são ainda mais eficazes em questão de adaptação, pois o particionamento da cena é baseado na posição e orientação de cada polígono de tal modo que as células obtidas correspondem sempre a regiões vazias da cena (*octrees* ou *kD-trees* não garantem essa condição, pois para isso podem requerer um número infinito de partições da cena). Além disso, tanto as *octrees*, *kD-trees* como árvores BSP têm o percurso das células organizado de forma hierárquica, o que permite – para o caso do descarte de visibilidade – descartar partes grandes da cena sem a necessidade de comparar todas as células da subdivisão. Por outro lado, a atualização de objetos da cena pode gerar alterações não-triviais dos particionamentos e até mesmo a reconstrução inteira da subdivisão, podendo acarretar uma perda proibitiva de desempenho.

Embora subdivisões espaciais tenham sido largamente exploradas para acelerar o processo de descarte de visibilidade, tais estruturas de dados não permitem a extração de um entendimento global de visibilidade da cena, como seria necessário, por exemplo, para determinar a visibilidade mútua entre todos os pares de polígonos da cena. Infelizmente, esse entendimento só é obtido com estratégias que classificam as relações de visibilidade em espaços diferentes do espaço do objeto, como é o caso das abordagens baseadas no complexo de visibilidade e no grafo de aspecto que, como já apontamos na introdução deste capítulo, têm uma complexidade combinatória elevada. Como resultado, não é possível aplicar um algoritmo de descarte de visibilidade para acelerar, por exemplo, o cálculo de fatores de forma para determinação de radiosidade de uma cena, uma vez que interreflexões entre objetos não são consideradas. Por outro lado, em cenas renderizadas com traçado de raios – caso também dependente de interreflexões entre objetos – as subdivisões espaciais são tradicionalmente utilizadas para acelerar o cálculo de interseção entre os raios e a geometria. Esse processo procura descartar grupos inteiros de objetos contidos em regiões que não intersectam os raios, de sorte que o resultado produzido é semelhante ao resultado desejado por um algoritmo de descarte de visibilidade: redução da complexidade de renderização apenas ao número de objetos que contribuem para a formação da imagem.

2.3 Estratégias de Descarte de Visibilidade

Descarte de visibilidade compreende três estratégias que podem ser utilizadas em conjunto: *descarte de faces invertidas* (*back-face culling*), *descarte por volume de visão* (*view frustum culling*) e *descarte por oclusão* (*occlusion culling*). Cada abordagem procura descartar uma classe específica de geometria considerada como trivialmente escondida. Descarte de faces invertidas [16, 25, 48], descarte por volume de

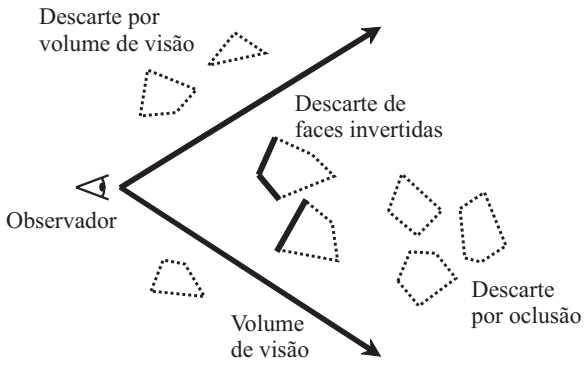


Figura 2.2: Três estratégias de descarte de visibilidade.

visão [4, 16, 37] e descarte por oclusão [11, 14, 24, 28, 36, 45] evitam renderizar, respectivamente, primitivas que não defrontam o observador, primitivas que estão fora do volume de visão e primitivas escondidas por alguma parte da cena. Estas estratégias são extremamente eficazes no aumento do desempenho das técnicas de renderização por traçado de raios. A Figura 2.2 ilustra essas três abordagens.

Em comparação com as estratégias de remoção de faces escondidas e de primitivas fora do volume de visão (cuja implementação, ainda que simplificada, é comumente encontrada em *hardware*), a estratégia de descarte por oclusão compreende um problema especialmente mais desafiador em conseqüência dos complexos relacionamentos globais de eventos visuais entre oclusores e objetos oclusos [12, 34]. Entretanto, seu uso tem sido imprescindível para a visualização eficiente de cenas densamente oclusas.

Cohen-Or *et al.* [9] propuseram uma taxonomia para classificar os diferentes algoritmos de descarte por oclusão existentes na literatura. Entre outros critérios, esses autores classificaram algoritmos de descarte por oclusão segundo a qualidade do PVS gerado (aproximado ou conservador), natureza do ponto de vista considerado (pontual ou regional), computação do PVS (pré-calculado ou em tempo de execução), precisão da determinação de visibilidade (precisão da imagem ou do objeto), maquinaria utilizada (*hardware* ou apenas *software*), dinâmica da cena (estática ou dinâmica) e agregação de oclusores (oclusores individuais ou fusão de oclusores). Cada categoria é apresentada resumidamente a seguir.

- Qualidade do PVS:
  - *Aproximado*: O PVS gerado inclui a maioria das primitivas visíveis e possivelmente algumas escondidas. Pequenos objetos visíveis podem deixar de ser detectados, possibilitando uma visualização mais eficiente da cena, embora com possíveis falhas na imagem.
  - *Conservador*: Garante-se que o PVS gerado contém todas as primitivas visíveis e possivelmente algumas escondidas. Inclui a maioria dos algoritmos de descarte por oclusão.
- Natureza do ponto de vista:
  - *Pontual*: O PVS é computado em relação a um observador descrito por um ponto no espaço.

  - *Regional*: O PVS depende da região onde o observador está, sendo invariante dentro dessa região. O objetivo de um algoritmo de visibilidade regional é amortizar o custo do cálculo de visibilidade para todos os quadros de exibição que contenham pontos de vista dentro da região, uma vez que o PVS é o mesmo em cada um desses locais.

- Computação do PVS:

  - *Pré-calculado*: Compreende os métodos que calculam todos os possíveis PVSs numa etapa de pré-processamento. Em tempo de execução, o algoritmo apenas acessa um banco de dados de PVSs pré-computados, de acordo com a localização do observador na cena.
  - *Em tempo de execução*: Determinam o PVS em tempo de execução.

- Precisão de determinação de visibilidade:

  - *Precisão da imagem*: Utiliza geometria rasterizada no *frame buffer* para determinar a oclusão.
  - *Precisão do objeto*: Utiliza uma representação geométrica ou volumétrica da cena para determinar a oclusão.

- Maquinaria utilizada:

  - *Software somente*: O processamento de visibilidade é independente do *hardware* gráfico.
  - *Hardware*: Tira vantagem do *hardware* gráfico (além do *z-buffer* para a remoção final de superfícies escondidas) para acelerar alguma etapa de pré-processamento ou acelerar a determinação da visibilidade em tempo de execução. É mais comum nos métodos que trabalham no espaço da imagem, integrados em etapas de processamento do *frame buffer* como o *buffer* de estêncil.

- Dinâmica da cena:

  - *Estática*: Calcula o PVS para cenas estáticas, permitindo apenas caminhadas em tempo real.
  - *Dinâmica*: Calcula o PVS para cenas dinâmicas com o observador imerso na cena de interesse, em tempo de execução.

- Agregação dos oclusores:

  - *Oclusores individuais*: A oclusão é computada com relação a apenas um objeto ou primitiva de cada vez. Métodos dessa categoria normalmente utilizam como oclusores polígonos grandes, convexos e mais próximos do observador.
  - *Fusão de oclusores*: A oclusão é computada com relação à combinação de grupos de pequenos oclusores desconexos de modo a formar um oclusor maior com maior capacidade de oclusão. A fusão de oclusores é sempre desejável, pois, em muitas cenas, dadas três primitivas, A, B e C, é possível que C não esteja ocluso nem com relação a A nem com relação a B, mas com

relação a A e B juntos. Essa situação é também cada vez mais comum em cenas densamente oclusas compostas por geometria refinada, uma vez que polígonos grandes não são facilmente encontrados em tal geometria.

2.4 Algoritmos de Descarte por Oclusão em Cenas Dinâmicas

Há relativamente poucos algoritmos de descarte por oclusão especialmente devotados a cenas dinâmicas se comparado com a quantidade de literatura disponível sobre descarte por oclusão em ambientes estáticos. Algumas técnicas destinadas a cenas estáticas podem permitir consultas de visibilidade que respondem se um objeto dinâmico está sendo escondido por alguma parte da cena, mas desse modo consideram como oclusores apenas os objetos estáticos e não conseguem responder se um objeto dinâmico esconde alguma coisa [14, 36]. Por outro lado, em cenas dinâmicas de movimentação arbitrária, qualquer objeto pode se tornar um oclusor em potencial, por exemplo, movendo-se bem à frente do observador e ocupando todo o seu campo de visão, ou então simplesmente aumentando de tamanho. A seguir citamos os algoritmos de descarte por oclusão encontrados na literatura que trabalham em cenas dinâmicas ou que possuem características promissoras para suportar esse tipo de cenas.

Baseado em trabalhos anteriores sobre propagação de visibilidade através de células e portais em cenas arquitetônicas internas [3, 43], Luebke e Georges [28] propuseram um algoritmo de descarte por oclusão no qual os elementos que propagam a visibilidade entre as salas (*e.g.*, portas e janelas) podem ser adicionados, movidos e redimensionados em tempo de execução. Entretanto, o método é restrito a ambientes que podem ser divididos em células disjuntas conectadas por portais de propagação de visibilidade, como é o caso das cenas arquitetônicas internas. Por outro lado, devido a sua simplicidade de implementação e eficiência ainda que numa classe limitada de cenas, essa técnica pode ser utilizada em conjunto com outras estratégias de descarte por oclusão, como ocorre na API dPVS [2].

Wonka e Schmalstieg [45] introduziram um esquema de descarte por oclusão para cenas urbanas 2.5D sem precomputação de visibilidade ou pré-seleção de oclusores, portanto apto a tratar oclusores dinâmicos. A cena é discretizada numa grade regular bidimensional de modo que oclusores próximos do observador podem ser rapidamente detectados em tempo de execução. Cada célula contém um valor que identifica a altura máxima da geometria que o intersecta. Os volumes de oclusão dos oclusores escolhidos (em geral, fachadas de prédios ou paredes) são renderizados num *z-buffer* com a projeção ortográfica da cena vista de cima, de modo que cada *pixel* do *z-buffer* corresponde a uma célula da grade regular e os valores de profundidade correspondem a valores de altura das células (quanto maior a altura, menor o valor de profundidade). Para classificar as regiões oclusas da cena, a altura de cada célula é comparada com o valor de profundidade do *pixel* correspondente no *z-buffer*. Um objeto é declarado ocluso se todas as células que o intersectam possuem alturas que correspondem a valores de profundidade maiores que os correspondentes no *z-buffer*.

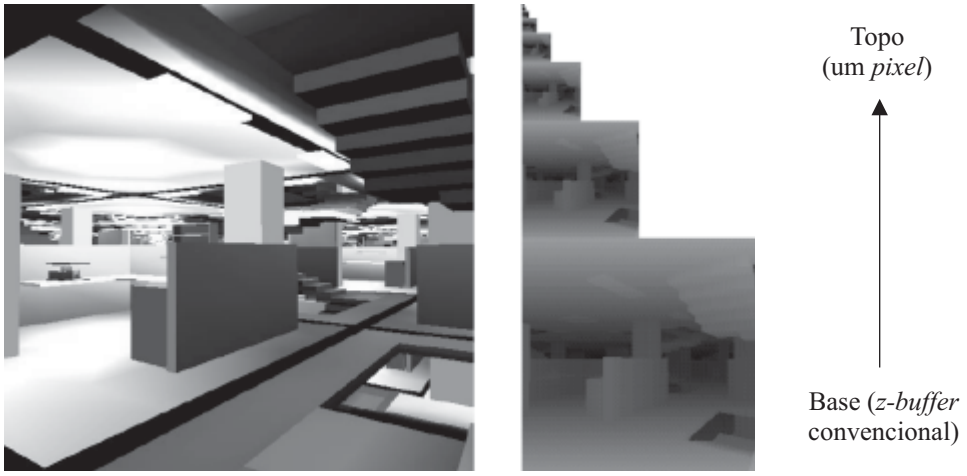


Figura 2.3: Cena renderizada (esquerda) e pirâmide de *z-buffers* correspondente (direita). Adaptado de Greene [21].

Enquanto podemos encontrar técnicas eficientes de descarte por oclusão para cenas 2D e 2.5D, como as expostas anteriormente, o tratamento de cenas dinâmicas tridimensionais tem sido pouco explorado. Para um levantamento de trabalhos anteriores, destacamos a seguir os algoritmos que determinam a visibilidade em tempo de execução, como o *z-buffer hierárquico* [20, 21] e a técnica de *mapas de oclusão hierárquicos* [47, 49]. Tais algoritmos podem ser incorporados numa arquitetura de determinação de visibilidade em cenas dinâmicas, embora atualmente apresentem algumas limitações.

2.4.1 Z-Buffer Hierárquico (HZB)

O *z-buffer* hierárquico (HZB – *Hierarchical Z-Buffer*) [20, 21] é um algoritmo de descarte por oclusão no espaço da imagem que utiliza uma pirâmide de *z-buffers* e uma *octree* como subdivisão espacial para remover partes significativas da cena estruturada em *voxels* com poucas comparações. Os níveis da pirâmide são construídos por um processo iterativo que atribui valores de profundidade mais distantes de janelas de 2x2 *pixels* de um nível, para apenas um *pixel* do nível seguinte, começando pela base da pirâmide que contém um *z-buffer* convencional. Assim, cada nível tem a metade da resolução horizontal e vertical do nível precedente e o topo da pirâmide é apenas um *pixel* contendo o valor de profundidade mais distante da imagem. A Figura 2.3 ilustra uma pirâmide de *z-buffers*, mostrando os valores de profundidade em tons de cinza (quanto mais claro, menor o valor de profundidade) e a renderização da cena correspondente. Em tempo de execução, a *octree* é percorrida em ordem de frente para trás a partir do observador. Durante o percurso, a projeção de cada *voxel* no *frame buffer* é comparada com a pirâmide de valores de profundidade, a partir do nível mais detalhado no qual um *pixel* encobre o tamanho da caixa limitante da projeção do voxel testado. Se a projeção do *voxel* está completamente oclusa segundo essa pirâmide de *z-buffers*, então os sub-*voxels* e objetos contidos em seu interior estão seguramente escondidos e podem ser descartados. Caso

contrário, o teste é repetido recursivamente para os sub-*voxels*. Os objetos associados a *voxels* visíveis, que correspondem a nós-folhas da *octree*, são então renderizados e utilizados para atualizar a pirâmide. O aproveitamente de oclusores é completo, pois todas as primitivas visíveis, por menores que sejam, contribuem para o descarte de partes escondidas da cena.

O maior obstáculo da utilização do HZB em *hardware* corrente como um substituto mais poderoso do *z-buffer* tradicional é a sua necessidade de ler o conteúdo do *z-buffer* do *hardware* gráfico para a memória do sistema. Infelizmente, a arquitetura da maioria das aceleradoras gráficas não foi projetada para permitir uma transferência eficiente do conteúdo do *z-buffer* à memória do sistema. Uma exceção é encontrada na GPU GeForce3 e chipsets posteriores da nVidia,[8] usando uma técnica chamada de *Z Occlusion Culling* [33]. Entretanto, tal funcionalidade ainda não se tornou padrão na indústria e suas consultas ao *z-buffer* ainda não são suficientemente rápidas para permitir uma implementação satisfatória do *z-buffer* hierárquico.

Recentemente, Greene propôs algumas mudanças na técnica de HZB original para uma implementação eficiente dessa abordagem em *hardware*, respeitando modificações praticáveis ao padrão da indústria [20]. Greene mostra que o tráfego de valores de profundidade na largura de banda do *hardware* pode ser reduzido de forma bastante significativa e, em alguns casos, seu desempenho pode ser mais eficiente do que o obtido em um *z-buffer* tradicional executado sobre a geometria visível conhecida de antemão. Infelizmente, ainda não existe *hardware* gráfico adaptado às mudanças sugeridas por Greene. Há, entretanto, variações simplificadas do HZB já disponíveis em aceleradoras gráficas. Por exemplo, o *hardware* gráfico com tecnologia *Hyper-Z* da ATI [31] implementa uma forma primitiva de *z-buffer* hierárquico com uma pirâmide de apenas dois níveis. Seu funcionamento é transparente ao usuário e tem um desempenho superior ao *z-buffer* tradicional (cerca de 20%), principalmente em cenas renderizadas em ordem de frente para trás a partir do observador. Essa abordagem pode ser utilizada em cenas dinâmicas, mas o algoritmo fica impossibilitado de descartar partes grandes da cena em níveis altos da hierarquia da *octree*, pois o *hardware* gráfico não retorna à CPU nenhuma informação sobre o estado da oclusão da projeção de um *voxel* no *z-buffer*. Desse modo, a complexidade de determinação de visibilidade não é uma função dos objetos visíveis, mas de todas as primitivas que compõem a cena.

2.4.2 Mapa de Oclusão Hierárquico (HOM)

Uma abordagem alternativa ao HZB, que não depende de *hardware* gráfico especial, é a técnica de mapa de oclusão hierárquico (HOM – *Hierarchical Occlusion Map*) [47, 49]. Seu funcionamento é similar ao HZB, mas gera resultados mais conservadores (*i.e.*, não descarta tantos objetos escondidos como o HZB). Enquanto no HZB todas as primitivas são consideradas como oclusores, a técnica de HOM utiliza apenas objetos contidos em um conjunto de oclusores construído durante pré-processamento. Em tempo de execução, oclusores são escolhidos apenas a partir desse conjunto, de acordo com tamanho e distância do observador.

[8] http://www.nvidia.com

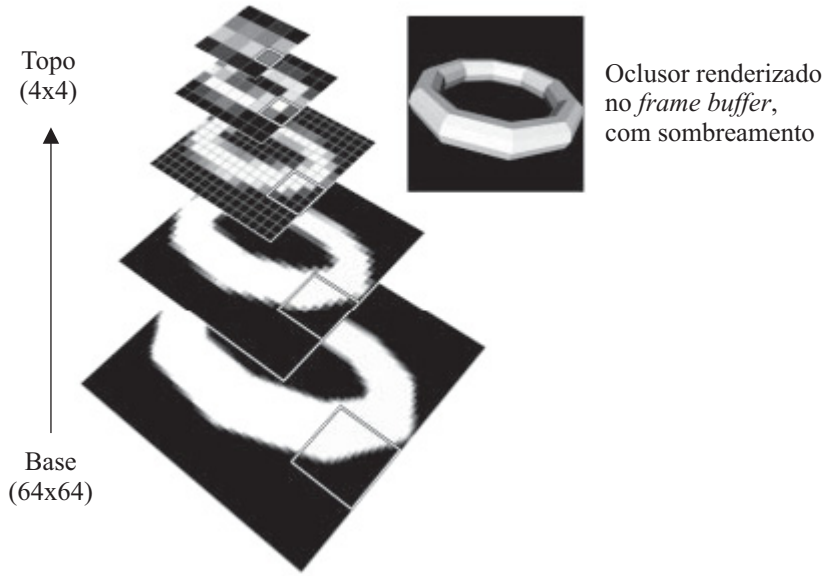


Figura 2.4: Pirâmide de mapas de oclusão. Adaptado de Zhang [47].

O teste de visibilidade utilizando mapas de oclusão hierárquicos é decomposto em um teste de sobreposição e um teste de profundidade. O mapa de oclusão hierárquico é utilizado no primeiro teste e consiste numa pirâmide de mapas semelhante à pirâmide do HZB, mas que contém valores de opacidade ao invés de valores de profundidade. Cada mapa de oclusão é uma imagem em tons de cinza, na qual a intensidade de cada *pixel* corresponde ao grau de cobertura do *pixel* pelos oclusores (quanto mais claro, maior a opacidade). O mapa da base da pirâmide corresponde à renderização dos oclusores na cor branca, sem sombreamento embora com anti-aliás, sobre um fundo preto. Os demais níveis da pirâmide são construídos através de um processo de filtragem da imagem dos níveis de maior resolução e a construção dessa pirâmide pode ser acelerada com auxílio de *hardware* gráfico com suporte à interpolação bilinear de texturas e geração de *mip-maps*. Uma ilustração do HOM gerado segundo a renderização de um oclusor toróide é mostrada na (Figura 2.4). O topo da pirâmide, neste caso, não é um *pixel*, mas um mapa de oclusão de tamanho 4x4.

Para cada quadro, um HOM é construído para um grande grupo de oclusores extraído do banco de oclusores. De forma análoga ao teste realizado no HZB, a subdivisão espacial é percorrida em ordem de frente para trás e a projeção de seus *voxels* é testada para sobreposição contra o HOM. O teste de profundidade é então realizado somente para os *voxels* que sobrepõem totalmente os oclusores discretizados no HOM. Este teste é feito em *software*, através de um *buffer de estimativa de profundidade*, que é um *z-buffer* de baixa resolução dos oclusores. É declarado ocluso o objeto cujo *voxel* sobrepuser apenas *pixels* opacos no HOM e estiver completamente atrás dos oclusores segundo o teste de profundidade.

Em cenas dinâmicas a subdivisão espacial é ignorada devido às dificuldades de atualizá-la em tempo de execução e caixas limitantes são utilizadas em seu lugar. A etapa de pré-cálculo de oclusores também é omitida e os oclusores são escolhidos em tempo de execução segundo tamanho e distância do observador.

Propriedades de coerência entre quadros também são utilizadas para reduzir o custo dessas escolhas ao longo da animação. Entretanto, da mesma forma que no HZB, a complexidade de determinação de visibilidade torna-se linear em relação ao número de objetos. Todos os objetos precisam ser testados contra a pirâmide, mesmo aqueles que não contribuem nenhum *pixel* para a formação da imagem, o que é proibitivo em cenas densamente oclusas de milhões de polígonos. A principal razão disso está na omissão do uso de uma subdivisão espacial, pois somente através dessas estruturas de dados torna-se possível descartar partes grandes da cena sem testar todos os objetos. A utilização de uma subdivisão espacial de atualizações eficientes tornaria a técnica de HOM adequada a cenas dinâmicas.

2.5 Volumes Limitantes Temporais

São duas as principais dificuldades encontradas no tratamento de cenas dinâmicas. A primeira é a dificuldade de atualizar de forma eficiente as subdivisões espaciais hierárquicas que a maioria dos algoritmos de visibilidade utilizam, em geral *octrees* e *kD-trees*. A segunda é o número de alterações necessárias em cada quadro de exibição para manter a subdivisão espacial atualizada. Em especial, se a estrutura de dados é atualizada para todo quadro de exibição e para todos os objetos dinâmicos, a sensibilidade à saída é perdida.

Embora neste trabalho proponhamos a utilização de uma grade regular como forma de diminuir a sobrecarga da atualização de objetos dinâmicos, muitos trabalhos têm sido feitos a respeito da adaptação de *octrees* para cenas dinâmicas. Baseados nos trabalhos de Ahuja, Nash e Weng, [1, 44], Smith *et al.* [38] apresentaram um algoritmo de atualização eficiente de *octrees* para objetos sujeitos a transformações de corpos rígidos (translações e rotações). Libes [26] apresentou uma técnica de representação em *octrees* de modelos que se expandem e se contraem arbitrariamente. Sudarsky e Gotsman [40, 39, 41] utilizaram coerência temporal para atualizar a *octree* somente até o menor voxel que contém a posição anterior e atual do objeto modificado, chamado de *menor ancestral comum*. Eventualmente, o menor ancestral comum pode ser o nível da raiz da hierarquia, mas para objetos que se movem suavemente espera-se que isso não ocorra.

Para diminuir o número de atualizações da estrutura de dados é possível designar para cada objeto dinâmico uma região do espaço que o contém completamente durante toda uma seqüência de animação. Esses volumes limitantes podem ser inseridos na estrutura espacial da cena, de sorte que os objetos dinâmicos correspondentes podem ser ignorados até que o algoritmo de visibilidade classifique esses volumes como visíveis. Tais regiões podem ser determinadas em qualquer lugar onde a movimentação dos objetos dinâmicos é restrita à uma trajetória conhecida.

O desempenho nesses casos depende do quão “compactos” são os volumes limitantes. Em um extremo, nada é conhecido sobre as possíveis localizações dos objetos dinâmicos (os objetos têm movimentação arbitrária), logo os volumes limitantes são iguais ao volume limitante da cena inteira. Como esses volumes são sempre visíveis, cada objeto dinâmico precisa ser verificado pelo algoritmo de visibilidade em todo quadro de exibição e por isso a sensibilidade à saída é perdida. No outro caso extremo, o volume limitante

é compacto o bastante para ser idêntico ao próprio objeto. Na verdade, o objeto é estático, pois não há para onde se mover e o cálculo de visibilidade tem uma complexidade equivalente ao de um algoritmo para cenas estáticas.

Em cenas contendo objetos dinâmicos que se movem de forma arbitrária, não é possível encontrar volumes limitantes para períodos completos da seqüência de animação. De fato, esses períodos podem ser ilimitados e se fossem determinados seriam provavelmente do tamanho da cena inteira.

Ao invés de designar um volume limitante para toda uma seqüência de animação, Sudarsky e Gostsman [40, 39, 41] sugerem calcular volumes limitantes para períodos curtos de tempo, chamados volumes limitantes temporais (TBVs – *Temporal Bounding Volumes*). Por exemplo, se a velocidade máxima de cada objeto dinâmico é conhecida, então, dada a posição do objeto em um certo instante, é possível calcular uma esfera limitante que garante conter o objeto até um tempo futuro dado. TBVs não precisam ser necessariamente esferas e podem assumir outras formas adequadas às características dinâmicas de cada objeto, como alterações de velocidade e direção de movimento.

Supõe-se que todo objeto dinâmico pode ter um TBV designado a partir de qualquer quadro de exibição até um determinado quadro futuro. Esse quadro futuro constitui a "data de expiração" do TBV; o período de tempo até esta data é o período de validade do volume limitante. Um objeto dinâmico escondido só precisa ser considerado para atualização da estrutura de dados se o seu volume limitante se tornar potencialmente visível, ou se a data de expiração for alcançada. Obtém-se a sensibilidade à saída com relação ao número de objetos dinâmicos porque eles são considerados para atualizar a estrutura de dados apenas quando se tornam potencialmente visíveis. Na maioria dos quadros, não se desperdiça tempo atualizando objetos dinâmicos escondidos, nem mesmo determinando se o objeto está escondido, uma vez que eles simplesmente são ignorados durante o percurso realizado na subdivisão espacial.

2.6 Comentários

A API dPVS [2] – uma biblioteca comercial de algoritmos de descarte de visibilidade – manipula objetos dinâmicos através de TBVs e parece ser a única biblioteca de algoritmos práticos de visibilidade em cenas dinâmicas existente atualmente. Nessa arquitetura, a geometria da cena é armazenada em uma árvore BSP de hiperplanos alinhados aos eixos (na verdade, isso reduz a estrutura a uma *kD-tree*) e que, segundo os autores, pode ser atualizada de forma mais eficiente do que uma *octree*. Os autores comentam sobre a possibilidade de usar grades regulares como subdivisão espacial em cenas dinâmicas, mas pelas dificuldades decorrentes do uso dessa estrutura preferem utilizar, à título de hipótese, árvores BSP. Por outro lado, testes comparativos não foram apresentados.

A dPVS é baseada numa abordagem híbrida da técnica de mapas de oclusão hierárquicos, algoritmos de células-e-portais, técnicas de níveis de detalhes, além de um mecanismo de escolha de diferentes

algoritmos de descarte segundo as características da cena tratada e desempenho desejado. Isso resulta numa solução eficiente de descarte de visibilidade para uma grande variedade de cenas complexas.

Capítulo 3

Algoritmo Proposto

Para uma definição precisa do método que propomos e como forma de classificá-lo entre as abordagens existentes, utilizamos a taxonomia sugerida por Cohen-Or *et al.* [9], apresentada na Seção 2.3. O método a ser apresentado neste Capítulo é em relação a:

- qualidade do PVS (*Potentially Visible Set*): conservador. Objetos escondidos podem ser erroneamente classificados como potencialmente visíveis, mas objetos visíveis nunca são classificados como escondidos, mesmo que apenas uma fração de sua geometria esteja visível. Não foi considerado nesta dissertação casos em que modificações na etapa de discretização da cena pudessem gerar PVSs aproximados segundo um limiar de opacidade das células da grade regular.

- natureza do ponto de vista: pontual. Cenas dinâmicas não garantem um PVS invariante dentro de uma célula da subdivisão espacial. Mesmo pequenas alterações da geometria podem modificar o PVS de várias regiões da cena de forma imprevisível, o que impede a utilização de um método de cálculo de PVSs válidos para regiões do espaço.

- computação do PVS: em tempo de execução, devido à impossibilidade de calcular o PVS *a priori* em cenas cujos objetos se movem de forma arbitrária.

- precisão de determinação de visibilidade: espaço do objeto. A escolha de oclusores, sua extensão e classificação da geometria oclusa são realizadas através de uma representação da cena em células opacas e oclusas. Essa representação volumétrica possibilita a geração de *oclusores virtuais* [8, 9] – denominação de oclusores no espaço do objeto que na verdade não fazem parte da cena e podem corresponder à agregação de vários objetos. Essa idéia foi utilizada pela primeira vez por Schaufler *et al.* [36]. Nossa abordagem é a primeira a realizar esse procedimento em cenas dinâmicas.

- maquinaria utilizada: *software* somente.

- dinâmica da cena: dinâmica, permitindo alterações arbitrárias da posição dos objetos, adição de novos objetos e deformação dos objetos existentes.

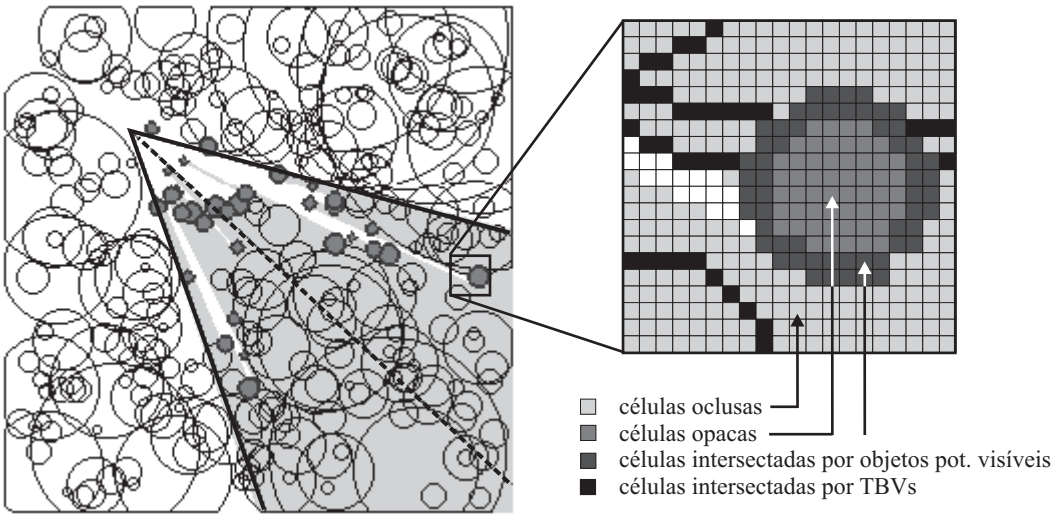


Figura 3.1: Visualização composta dos dados de uma grade regular 2D.

- agregação dos oclusores: fusão de oclusores. Nosso algoritmo realiza fusão de oclusores no espaço do objeto explorando o princípio utilizado por Schaufler *et al.* [36] que considera as regiões opacas e escondidas da cena como oclusores. Essas regiões são representadas por células da grade regular.

3.1 Definições e Terminologia

A grade regular representa uma subdivisão do espaço na qual cada célula identifica características locais da cena como opacidade, oclusão e objectos intersectados. Para cada quadro de exibição, todas as células que intersectam o volume de visão são percorridas em uma ordem de frente para trás a partir do observador, procurando por células opacas que podem ser utilizadas como oclusores. De acordo com a abordagem introduzida por Schaufler *et al.* [36], cada oclusor pode ser estendido agregando células opacas e oclusas na vizinhança de uma célula opaca inicial, gerando, com isso, uma fusão de oclusores. Somente objetos totalmente contidos em células oclusas são considerados escondidos. Desse modo, o conjunto de objetos atribuídos para a renderização é sempre uma superestimativa do número de objetos visíveis da cena.

Para propósitos de otimização, organizamos as informações da grade regular em quatro matrizes:

- *Matriz de oclusores* ($\mathcal{O}$): Classifica cada célula como *opaca* ou *não-opaca*. Uma célula é considerada opaca se está completamente contida no interior de um objeto potencialmente visível.
- *Matriz de oclusão* ($\mathcal{H}$): Classifica cada célula como *oclusa* ou *não-oclusa*. Uma célula é classificada como oclusa se está totalmente escondida por células opacas ou oclusas com relação ao observador.
- *Matriz de identificadores* ($\mathcal{I}$): Associa para cada célula uma lista de identificadores (IDs) de objetos

potencialmente visíveis que intersectam essa região discreta da cena. Todas as células opacas são necessariamente células não-vazias de $\mathcal{I}$.

- *Matriz de TBVs otimizados* ($\mathcal{T}$): Associa para cada célula uma lista de identificadores de TBVs que intersectam essa região.

A Figura 3.1 mostra uma ilustração composta dessas matrizes, correspondendo a uma grade regular de resolução de 256x256 células que representa uma cena 2D composta por 300 objetos dinâmicos (32 potencialmente visíveis) discretizados. Os objetos dinâmicos utilizados foram pequenos círculos de tamanhos variados. Dessa forma, as células intersectadas por objetos potencialmente visíveis formaram círculos preenchidos na ilustração. Cada TBV discretizado tem a forma de um círculo vazio. Por clareza, a linha-de-visão e os limites da região de visão são mostrados sobrepostos, respectivamente, como uma linha preta tracejada e duas linhas pretas sólidas. Como forma de manter a uniformidade dos termos no restante do trabalho, chamaremos de células os elementos da grade regular tanto no caso 2D como no caso 3D.

Supomos que cada objeto possui um identificador único, uma velocidade máxima, além de informações a respeito de seu TBV, como data de expiração, diâmetro (como os objetos têm movimento arbitrário, seus TBVs são sempre círculos em 2D e esferas em 3D), posição e um *flag* indicando se algum TBV foi designado ao objeto. Os identificadores dos TBVs podem ser iguais aos identificadores dos objetos que os possuem. Assumimos também a existência de uma lista de identificadores de objetos classificados como potencialmente visíveis no último quadro de exibição.

3.2 Visão Geral do Algoritmo

O algoritmo de determinação de visibilidade para cenas dinâmicas é composto pelos procedimentos mostrados a seguir, executados para cada quadro.

- **Discretização da cena:** Atualiza as matrizes $\mathcal{O}$, $\mathcal{H}$, $\mathcal{I}$, $\mathcal{T}$ para os objetos contidos no PVS do último quadro de exibição e manipula os objetos escondidos de acordo com o estado de seus TBVs. Em especial, objetos potencialmente visíveis do último quadro são atualizados em $\mathcal{O}$ e $\mathcal{I}$. Os demais objetos são utilizados para atualizar $\mathcal{T}$. Essa etapa é necessária para manter a subdivisão espacial atualizada ao longo da animação. Entretanto, relativamente poucas células são modificadas nas matrizes em cada quadro de exibição. Isso ocorre porque, pela definição de cenas densamente oclusas, o número de objetos potencialmente visíveis em qualquer quadro corresponde a uma pequena parcela do número total de objetos da cena. Desse modo, as matrizes $\mathcal{O}$ e $\mathcal{I}$ são geralmente esparsas. O mesmo não ocorre com a matriz $\mathcal{T}$, mas nesse caso os objetos escondidos só precisam ser atualizados esporadicamente quando a data de seus TBVs expiram. Essa atualização pode ser feita de forma eficiente tanto em 2D como em 3D, segundo as otimizações detalhadas na Seção 3.3.

- **Percurso do volume de visão:** Percorre as células que intersectam o volume de visão em uma ordem de frente para trás a partir do observador, de modo a detectar objetos potencialmente visíveis e oclusores. Objetos potencialmente visíveis correspondem aos objetos contidos nas células não-vazias de $\mathcal{I}$ e ao mesmo tempo não-oclusas em $\mathcal{H}$. Oclusores são construídos a partir de células opacas e oclusas e são tratados segundo os procedimentos de *extensão de oclusores* e *cálculo de oclusão*, a seguir. O percurso do volume de visão de frente para trás é uma etapa comum à maioria dos algoritmos de descarte por oclusão que determinam a visibilidade em tempo de execução, pois assim evita-se o processamento inútil de oclusores escondidos. Além disso, o descarte de objetos fora do volume de visão é obtido trivialmente com essa estratégia.

- **Extensão de oclusores:** Estende cada célula opaca encontrada durante o percurso do volume de visão às células opacas e oclusas adjacentes, de modo a construir um oclusor com maior capacidade de oclusão. Esse conjunto de células agregadas forma um *oclusor estendido*. Essa etapa, derivada do trabalho de Schaufler *et al.* [36], é aqui utilizada devido a nossa escolha em realizar a determinação de visibilidade no espaço do objeto entre as células da subdivisão espacial. Essa escolha foi influenciada tanto pela facilidade de adaptar os procedimentos adotados por Schaufler *et al.* à grade regular, como pela possibilidade de criar condições para acelerar o cálculo de oclusão através de princípios de rasterização.

- **Cálculo de oclusão:** Constrói um volume de oclusão para cada oclusor estendido e classifica como oclusas as células totalmente contidas dentro desse volume. Embora o cálculo de oclusão seja baseado no mesmo procedimento utilizado na abordagem de Schaufler *et al.* [36], a regularidade e uniformidade da grade regular permitem que princípios de rasterização sejam explorados de modo a acelerar a classificação das células totalmente contidas num volume de oclusão. Como trabalhamos apenas no espaço do objeto, o cálculo de oclusão difere daquele empregado em técnicas como o HZB e HOM, apresentadas no Capítulo 2. Nessas abordagens, o cálculo de oclusão é realizado no espaço da imagem através de um teste de sobreposição/profundidade de projeção de *voxels* da estrutura de dados na imagem, do nível mais alto da hierarquia ao mais baixo. Embora essa estratégia não tenha sido utilizada nesta dissertação, o possível impacto da utilização do cálculo de oclusão no espaço da imagem é discutido no Capítulo 5.

As etapas de extensão de oclusores e cálculo de oclusão podem ser executadas várias vezes dentro do percurso do volume de visão, dependendo do número de oclusores encontrados.

No final de cada quadro de exibição, todas as células de $\mathcal{O}$ são classificados como não-opacas e todas as células de $\mathcal{H}$ são modificadas para o estado de não-oclusas.

3.3 Caso 2D

De modo a facilitar a compreensão do algoritmo proposto, sua apresentação é dada inicialmente em cenas dinâmicas bidimensionais. Devido a nossa escolha em utilizar grades regulares, a extensão do algoritmo para cenas 3D requer poucas modificações. No lugar de volume de visão (*frustum* de visão) e volumes de oclusão, deve-se entender polígono de visão e polígonos de oclusão, respectivamente. Entretanto, mantemos as palavras *volume de visão* e *volumes de oclusão* de modo a conservar a uniformidade dos termos.

3.3.1 Discretização da Cena

No ínicio de cada quadro de exibição, os objetos contidos no PVS do último quadro são discretizados em $\mathcal{O}$ e atualizados em $\mathcal{I}$. No primeiro quadro, todos os objetos são considerados como potencialmente visíveis, já que não possuem TBVs designados e o algoritmo ainda não sabe quais objetos estão escondidos.

No caso bidimensional, a discretização é feita rasterizando a projeção ortográfica de cada objeto visto de cima e associando cada *pixel* do *frame buffer* resultante a uma célula da subdivisão. O objetivo desse procedimento é apenas identificar quais os *pixels* (e por conseqüência quais as células) que estão sendo cobertos, tanto parcialmente como inteiramente, pela projeção dos objetos da cena. Logo, a iluminação, a texturização e o *z-buffer* podem ser desabilitados.

A rasterização deve ser feita de tal forma a produzir resultados conservadores, em que $\mathcal{O}$ subestime as células opacas da cena e $\mathcal{I}$ superestime as células intersectadas pelos objetos. Em outras palavras, devemos garantir que as células opacas estejam inteiramente contidas no interior do objeto e que as células intersectadas pelo objeto estejam pelo menos parcialmente contidas no mesmo.

Uma rasterização ideal, no sentido de ser menos conservadora possível, é obtida com anti-alias de *amostragem por área* (ou *anti-alias analítico*, segundo Glassner [19]). Anti-alias de amostragem por área define a opacidade dos *pixels* de acordo com a porcentagem exata de cobertura da projeção da geometria incidente sobre eles. Assim, as células opacas que correspondem aos *pixels* contidos inteiramente na projeção do objeto são os *pixels* com valor máximo de opacidade. Por exemplo, se um objeto é renderizado em branco sobre um fundo preto, apenas as células correspondentes aos *pixels* brancos são consideradas opacas, pois estão inteiramente cobertas pela projeção da geometria do objeto. Por outro lado, todos os *pixels* diferentes da cor de fundo correspondem a células intersectadas pela geometria e assim o identificador do objeto é adicionado à lista de cada célula correspondente em $\mathcal{I}$. Um exemplo é mostrado na Figura 3.2.

Infelizmente, anti-alias de amostragem por área pode ser muito custoso em *software*, inexato ou não-disponível em *hardware* gráfico. Uma solução mais eficiente, embora mais conservadora, consiste em rasterizar os objetos com um contorno espesso em cor distinta da cor do interior do objeto, de modo que os *pixels* com a cor do contorno correspondem a células não-opacas intersectadas pelo objeto (Figura 3.3). O

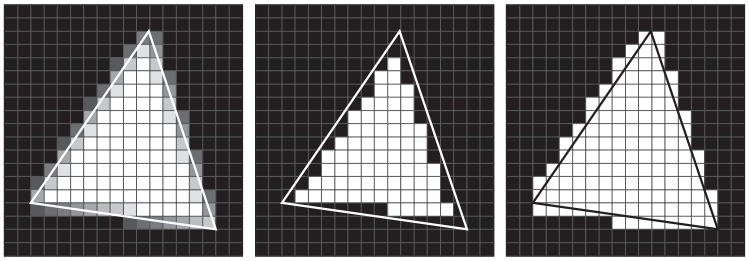


Figura 3.2: Discretização ideal com anti-alias de amostragem por área. Esquerda: rasterização e contorno exato (em linhas cheias). Centro: *pixels* correspondentes a células opacas. Direita: *pixels* correspondentes a células intersectadas pelo objeto.

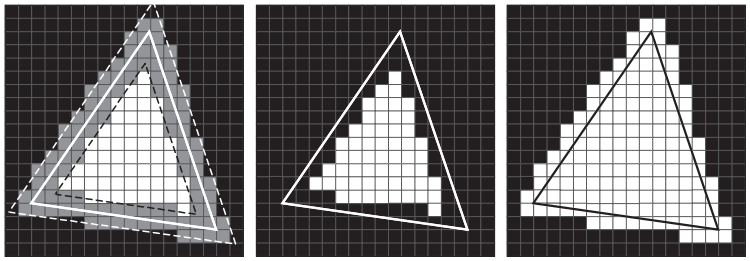


Figura 3.3: Discretização conservadora com contorno espesso. Esquerda: rasterização e ajuste da amostragem para discretização conservadora (em linhas tracejadas). Centro: *pixels* correspondentes a células opacas. Direita: *pixels* correspondentes a células intersectadas pelo objeto.

procedimento de atribuição de *pixels* a células em $\mathcal{O}$ e $\mathcal{I}$ é o mesmo que o anterior. A largura do contorno necessária para uma rasterização conservadora é calculada segundo o procedimento utilizado por Wonka *et al.* [45]: considerando que a amostragem é realizada no centro de cada *pixel*, cada aresta do contorno é contraída e dilatada ao longo do seu vetor normal por um valor suficiente igual à metade da maior diagonal de uma célula, *i.e.*, $\varepsilon = d/\sqrt{2}$, onde d é a largura de uma célula. As rasterizações dos objetos dilatados e contraídos atualizam, respectivamente, $\mathcal{I}$ e $\mathcal{O}$. Esse procedimento pode ser ainda simplificado em OpenGL rasterizando o contorno do objeto como um laço de linhas de largura de dois ou três *pixels*, dependendo da forma como a implementação trata linhas espessas sem anti-alias.

A manutenção de volumes limitantes temporais é baseada no procedimento descrito por Sudarsky e Gotsman [41]. Essa etapa de discretização se aplica apenas a objetos escondidos, de acordo com as seguintes observações: (1) Objetos que não constam no PVS atual e não possuem um TBV são objetos que estavam visíveis no quadro anterior e acabaram de se tornar escondidos. Neste caso, novos TBVs são designados a esses objetos e a matriz $\mathcal{T}$ é atualizada apropriadamente. (2) Objetos que não constam no PVS atual mas possuem um TBV designado são objetos que estavam escondidos e continuam escondidos no quadro atual. Nesse caso, o algoritmo apenas verifica se a data de expiração do TBV foi alcançada. TBVs com validade vencida não garantem mais conter seus objetos, portanto são removidos de $\mathcal{T}$ e substituídos por TBVs com

um novo período de validade.

O tratamento dos objetos que tiveram seus TBVs revelados (*i.e.*, que acabaram de se tornar potencialmente visíveis) é realizado durante o percurso das células do volume de visão, descrito na próxima seção.

Os períodos de validade dos TBVs são escolhidos de forma *adaptativa*, como proposto por Sudarsky *et al.* [41]. Se um objeto estava escondido mas a data de validade do seu TBV expirou, então o período de validade do TBV foi muito curto, pois o objeto poderia ter ficado mais tempo sem ser atualizado se o seu TBV tivesse uma validade maior. Nesse caso, uma data de expiração mais longa é designada para o novo TBV. Por outro lado, se o TBV foi revelado antes da data de expiração ter sido alcançada, então o TBV era grande demais, pois um TBV menor poderia ser ignorado por mais tempo. Assim, um período de validade mais curto é escolhido para o novo TBV – gerando, portanto, um TBV mais compacto. Dessa forma, os períodos de validade se adaptam ao comportamento dos objetos dinâmicos. Objetos que se movem rapidamente e são constantemente visíveis tendem a possuir TBVs de períodos de validade mais curtos ao longo do tempo, enquanto objetos praticamente estáticos tendem a possuir TBVs com períodos longos de validade e são atualizados menos freqüentemente na grade regular. Por outro lado, notamos que a eficiência da atualização de $\mathcal{T}$ decresce à medida que os TBVs aumentam de tamanho, pois o número de células intersectadas pelo TBV é maior. Se o tempo necessário para atualizar $\mathcal{T}$ é maior que o tempo necessário para atualizar o próprio objeto em $\mathcal{I}$, o algoritmo pode ter seu desempenho prejudicado nos quadros onde os TBVs são atualizados e a animação poderá deixar de ser uniforme. Para evitar esse comportamento, limitamos o tamanho máximo dos TBVs adaptativos de acordo com o número de células intersectadas pelo objeto e por sua velocidade máxima. Em especial, o raio máximo de um TBV não ultrapassou $\sqrt{\left(\frac{w}{5}\right)^2 + \left(\frac{h}{5}\right)^2}$, onde w e h correspondiam às dimensões da grade regular. Objetos que intersectam muitas células e movimentam-se rapidamente, toleram a atualização de $\mathcal{T}$ para TBVs grandes, enquanto objetos que intersectam poucas células e se movem lentamente, limitam mais o tamanho de seus TBVs. Esse procedimento responde de forma mais satisfatória a um compromisso entre número e tempo de atualizações de objetos e TBVs e assim mantém taxas de exibição mais uniformes.

3.3.2 Discretização Otimizada de TBVs

Para objetos dinâmicos de movimento arbitrário, isto é, objetos cuja direção de percurso não é conhecida, a representação mais compacta de um TBV bidimensional é um círculo. Como os círculos só variam em posição e diâmetro, sua discretização na grade regular pode ser feita de forma mais simples do que a discretização de um objeto genérico. Considerando a correspondência direta entre *pixels* e células da grade regular, um TBV 2D pode ser discretizado em $\mathcal{T}$ rasterizando um círculo preenchido no *frame buffer* e então associando cada *pixel* da imagem rasterizada a uma célula da grade. Desse modo, o identificador do TBV é adicionado a todas as células que correspondem a *pixels* do círculo rasterizado. Na verdade, podemos simplificar esse procedimento "rasterizando" o círculo diretamente em $\mathcal{T}$ como se este fosse o *frame buffer*.

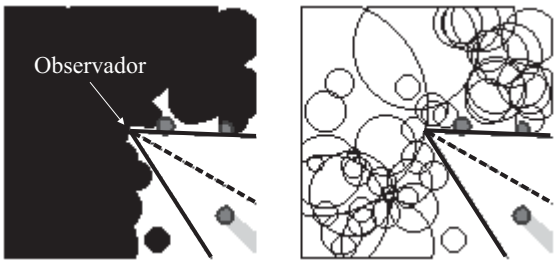


Figura 3.4: Discretização otimizada de TBVs. Esquerda: discretização tradicional. Direita: discretização otimizada.

Sugerimos uma forma ainda mais eficiente de discretizar TBVs para cenas nas quais o observador movimenta-se suavemente pelo espaço. Ao invés de discretizarmos o TBV como um círculo preenchido, podemos atualizar somente as células que intersectam a fronteira desse volume limitante, o que é feito rasterizando o TBV como um círculo vazio, reduzindo de forma significativa o número de células acessadas. De fato, essa técnica pode ser implementada facilmente com uma modificação do algoritmo de Bresenham [16] à "rasterização" de círculos em $\mathcal{T}$. Os TBVs mostrados na Figura 3.1 foram discretizados com essa técnica. Outros algoritmos de desenho de círculos também podem ser utilizados. Na Figura 3.4, duas visualizações do conteúdo da grade regular são mostradas. A imagem à esquerda contrasta com a imagem da direita e as duas correspondem respectivamente à visualização da grade regular sem e com otimização de TBVs. As células não-vazias de $\mathcal{T}$ correspondem a *pixels* pretos e todos os demais elementos são apresentados de uma maneira semelhante à convenção utilizada na Figura 3.1. A redução do número de células não-vazias de $\mathcal{T}$ na discretização otimizada é evidente.

Precisamos garantir que, apesar da redução do número de células discretizadas em $\mathcal{T}$, os TBVs ainda são corretamente detectados. Se um TBV é erroneamente ignorado, é possível que o objeto que o contenha esteja visível, mas como o algoritmo de visibilidade o ignora, pode gerar por isso PVSs que não contenham todos os objetos visíveis.

Em primeiro lugar, consideramos que o observador descreve sempre uma trajetória num caminho 4-conexo ou 8-conexo pela grade regular. Se o observador descreve um caminho 4-conexo, ele não pode entrar em um TBV sem detectá-lo, pois a rasterização por Bresenham forma círculos de fronteiras 8-conexas. Entretanto, se o observador entrar no círculo por um caminho que não seja 4-conexo, o TBV pode ser erroneamente ignorado. Felizmente, este caso é facilmente resolvido modificando o algoritmo de Bresenham para a rasterização de círculos com fronteiras 4-conexas[1] Dessa forma, para um observador localizado fora de um TBV, todos os TBVs serão detectados. Não obstante, para um observador inicialmente dentro de um TBV, pode ocorrer que a discretização do arco de circunferência contido no volume de visão seja escondida por algum oclusor, *i.e.*, a discretização do arco de circunferência pode coincidir apenas com células oclusas.

[1]Nas ilustrações exibidas ao longo deste trabalho, os círculos resultantes da discretização de TBVs otimizados são mostrados como fronteiras 8-conexas. De fato, essa correção não foi implementada no caso bidimensional. Por outro lado, nenhum erro de detecção de TBVs foi notado.

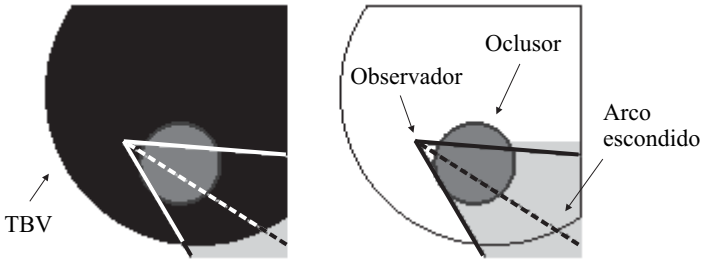


Figura 3.5: Erro de detecção de um TBV otimizado. Esquerda: TBV tradicional revelado durante o percurso das células do volume de visão. Direita: TBV otimizado, erroneamente classificado como escondido porque a discretização do arco de circunferência contido no volume de visão está escondida pelo oclusor.

Desse modo, o TBV é erroneamente ignorado. Um exemplo é mostrado na Figura 3.5, através de uma visualização do conteúdo da grade regular similar à utilizada na Figura 3.1. Na imagem da esquerda, o TBV é corretamente revelado porque há células não-vazias de $\mathcal{T}$ e ao mesmo tempo visíveis no interior do volume de visão. Essa condição necessária à identificação do TBV não ocorre com a imagem da direita, pois as únicas células não-vazias de $\mathcal{T}$ contidas no volume de visão coincidem com células oclusas de $\mathcal{H}$ e por isso estão escondidas. Uma solução também simples para este caso consiste em garantir que o período de validade de todos os TBVs seja escolhido de tal forma que o raio do círculo limitante nunca seja maior que a distância da célula contendo o centro do círculo à célula contendo o observador. Se essa condição não puder ser satisfeita (*e.g.*, quando o raio do TBV precisar ser menor do que o raio do menor círculo limitante do objeto), o TBV não é designado ao objeto e este é discretizado em $\mathcal{O}$ e $\mathcal{I}$ como se o seu TBV houvesse sido revelado (esse procedimento é detalhado na seção 3.3.3). De fato, este último caso ocorre quando o objeto está muito próximo do observador e por isso está prestes a se tornar potencialmente visível. Enfim, essas ponderações garantem que um TBV inserido na grade regular jamais encobre o observador, e como já mostramos que o observador não pode entrar num TBV sem ser detectado, a corretude do algoritmo é garantida.

3.3.3 Percurso do Volume de Visão

A determinação de visibilidade propriamente dita é realizada no percurso do volume de visão. Esse procedimento compreende o percurso de células que intersectam o volume de visão de modo a identificar oclusores e objetos potencialmente visíveis. Ambos são encontrados em células não-oclusas de $\mathcal{H}$. Oclusores correspondem a células opacas de $\mathcal{O}$ e objetos potencialmente visíveis são encontrados em células que contém listas não-vazias em $\mathcal{I}$ ou $\mathcal{T}$.

O percurso das células do volume de visão é realizado em uma ordem de frente para trás a partir do observador. Assim, o algoritmo não perde tempo tratando oclusores escondidos e o PVS pode ser determinado incrementalmente em apenas um percurso.

Posição da célula-semente	Direções iniciais de percurso
$\|x\| > \|y\|$	$+y$ e $-y$
$\|x\| < \|y\|$	$+x$ e $-x$
$(\|x\| = \|y\|) \wedge (x > 0) \wedge (y > 0)$	$-x$ e $-y$
$(\|x\| = \|y\|) \wedge (x < 0) \wedge (y < 0)$	$+x$ e $+y$
$(\|x\| = \|y\|) \wedge (x < 0) \wedge (y > 0)$	$+x$ e $-y$
$(\|x\| = \|y\|) \wedge (x > 0) \wedge (y < 0)$	$-x$ e $+y$

Tabela 3.1: Direções de percurso das células do volume de visão 2D segundo a posição de uma célula-semente (x, y) em coordenadas relativas à célula contendo o observador.

Para calcular de forma eficiente a distância das células ao observador e então realizar um percurso de frente para trás, utilizamos a métrica xadrez.[2] Com isso evitamos operações custosas de raiz quadrada da métrica euclidiana e podemos realizar o percurso sempre em direções alinhadas aos eixos. Visto que a linha de visão está sempre contida no volume de visão, é possível discretizá-la na grade regular, de forma incremental a partir do observador usando o algoritmo de rasterização de linhas de Bresenham [16] e a partir de cada célula que contém a linha de visão discretizada (chamada de *célula-semente*), percorrer as células adjacentes que têm a mesma distância xadrez do observador. A partir de cada célula-semente, o percurso é sempre realizado em duas direções alinhadas aos eixos, decorrentes da métrica xadrez e facilmente determinadas pelo sinal das coordenadas relativas ao observador em que se encontra a célula-semente, conforme ilustra a Figura 3.6. Por exemplo, sejam (x, y) as coordenadas relativas ao observador de uma célula-semente, as direções de percurso serão aquelas mostradas na Tabela 3.1. Além disso, uma direção só é alterada se for alcançada uma célula cujas coordenadas tenham valores absolutos iguais ($|x| = |y|$). Nesse caso, a direção continua sempre perpendicularmente à direção original e o percurso termina quando uma célula fora do volume de visão é atingida, ou quando a célula-semente é novamente alcançada (no caso de um campo de visão de $360°$).

Durante o percurso, se uma célula não-oclusa é encontrada, os objetos contidos em sua lista de identificadores em $\mathcal{I}$ são adicionados ao PVS do quadro atual. Células opacas não-oclusas são consideradas como oclusores, portanto devem determinar quais células estão sendo escondidas com relação ao observador. O detalhamento desta etapa, que compreende a extensão de oclusores e o cálculo de oclusão, é mostrado nas seções seguintes. Células não-oclusas que contém TBVs segundo $\mathcal{T}$, indicam que esses TBVs foram revelados e que os objetos correspondentes podem estar visíveis. Portanto, seus TBVs são removidos de $\mathcal{T}$ e dissociados dos respectivos objetos. Além disso, tais objetos são imediatamente discretizados em $\mathcal{I}$ e $\mathcal{O}$ para que o algoritmo possa determinar, durante a continuação do percurso, se esses objetos são de fato potencialmente visíveis.

Ao terminar o percurso, o PVS foi completamente determinado. Para o início do próximo quadro, todas as células de $\mathcal{O}$ e $\mathcal{H}$ são classificadas como não-opacas e não-oclusas, respectivamente.

[2]Na métrica xadrez, a distância entre dois pontos (x_1, y_1) e (x_2, y_2) é dada na forma $max(|x_1 - x_2|, |y_1 - y_2|)$.

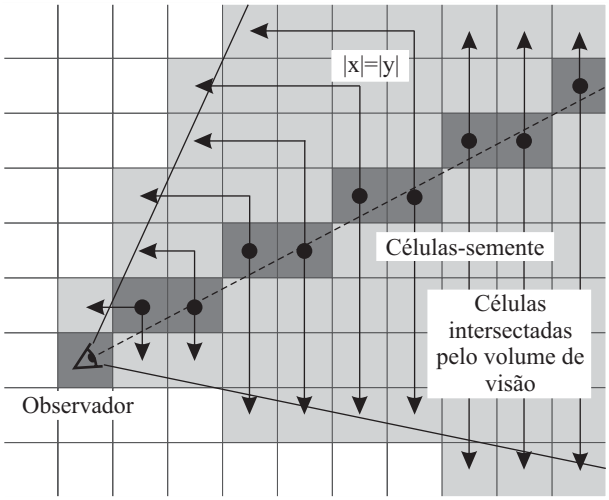


Figura 3.6: Percurso do volume de visão. Células-semente são mostradas em cinza escuro. As setas indicam a direção do percurso a partir de cada célula-semente, de acordo com a métrica xadrez.

O percurso do volume de visão pode ser terminado prematuramente se, durante o percurso nas duas direções determinadas por uma célula-semente, apenas células oclusas forem detectadas. Isso significa que as células restantes da cena estão escondidas e o PVS seguramente não será modificado pelo menos até o próximo quadro.

3.3.4 Extensão de Oclusores

O processo de extensão de oclusores é uma adaptação da técnica de *blocker extension* utilizada por Schaufler *et al.* [36] originalmente para *octrees*. A idéia consiste em agregar células opacas ou oclusas a uma célula opaca inicial encontrada durante o percurso do volume de visão. Esse procedimento procura maximizar o ângulo subentendido entre o observador e o oclusor de modo a aumentar a capacidade de oclusão, além de reduzir o número de cálculos de oclusão necessários para descartar objetos escondidos.

Schaufler *et al.* observam que células oclusas podem ser tratadas como células opacas segundo o argumento de que o observador é incapaz de distinguir se uma célula escondida é opaca ou não. Entretanto, considerando células oclusas como células opacas, é possível estender os oclusores dentro do espaço escondido da cena e aumentar a capacidade de oclusão, efetivando a *fusão de oclusores*. A partir desse conjunto de células agregadas, chamado de *oclusor estendido*, um volume de oclusão é construído de modo a determinar as células oclusas com relação ao observador, que são as células inteiramente contidas nesse volume. A oclusão causada por um volume de oclusão de um oclusor estendido é, em geral, mais eficaz que a união da oclusão causada pelos volumes de oclusão construídos individualmente para cada célula que forma esse mesmo oclusor estendido. Isso ocorre porque há regiões da cena que não estão sendo escondidas inteiramente por apenas uma célula, mas pela combinação da oclusão causada por duas ou mais células.

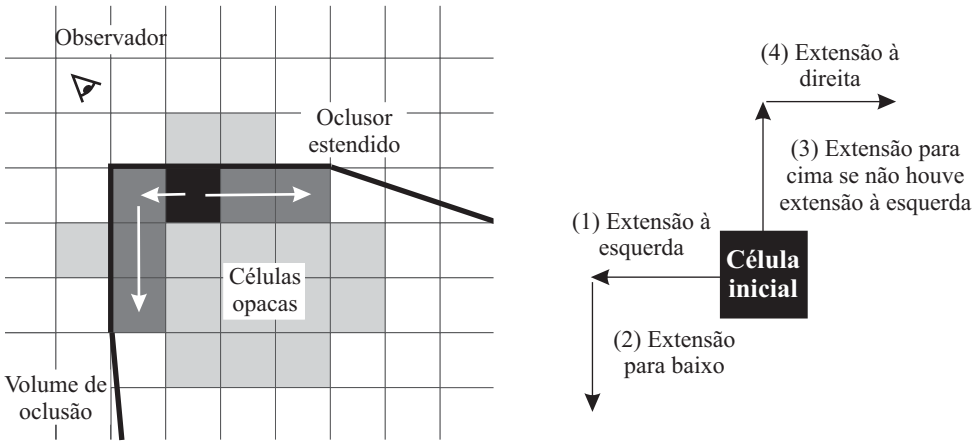


Figura 3.7: Extensão de oclusores. Esquerda: oclusor estendido através da fusão de células opacas em direções alinhadas aos eixos. A célula inicial da extensão é mostrada em preto. Direita: heurística utilizada.

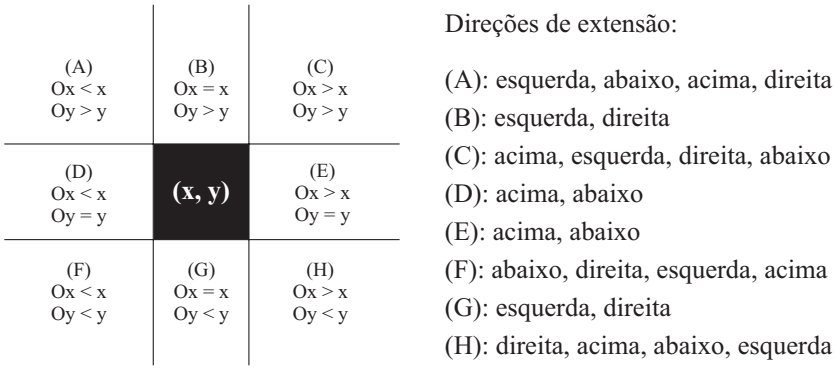


Figura 3.8: Direções dos passos de extensão de oclusores divididos em partições, conforme posição da célula inicial (x, y) em coordenadas relativas à célula (O_x, O_y) contendo o observador.

Oclusores são estendidos através da procura por células opacas ou oclusas em direções alinhadas aos eixos, de acordo com o método também utilizado por Schaufler *et al.* no qual a agregação de células subentende um oclusor convexo em forma de L. A heurística, entretanto, é muito mais simples para grades regulares. A Figura 3.7 mostra um exemplo no qual a heurística utilizada consiste nos seguintes passos: (1) extensão à esquerda a partir da célula inicial; (2) extensão para baixo a partir da célula mais à esquerda determinada por (1); (3) extensão para cima a partir da célula inicial; (4) extensão à direita a partir da célula mais para cima determinado por (3). Para a restrição a oclusores convexos, o passo (3) só é executado quando nenhuma célula é estendida durante o passo (1). No caso geral, as direções de cada passo dependem da posição relativa da célula contendo o observador à célula inicial da extensão do oclusor, conforme mostra a Figura 3.8. As direções de extensão utilizadas na Figura 3.7 correspondem àquelas mostradas na partição (A) da Figura 3.8.

O cálculo de oclusão realizado após a extensão do oclusor (detalhado na seção 3.3.5) pode ser

uma tarefa custosa, pois envolve a atualização de um grande número de células em $\mathcal{H}$. Idealmente, essa etapa deve ser feita para poucos oclusores – grandes e próximos do observador – que possam encobrir a maioria da geometria da cena. Entretanto, essa disponibilidade de oclusores efetivos é dependente do contexto e por isso nem sempre é obtida, ainda que fusão de oclusores seja empregada. Para evitar o cálculo da oclusão de oclusores pouco significativos, como oclusores compostos por apenas uma célula opaca, é possível utilizar um esquema de escolha de bons candidatos a oclusores baseado na heurística de Coorg e Teller [11] que estima o ângulo sólido subentendido entre o observador e o oclusor. Essa estimativa é dada pelo termo: $-a(\vec{n} \cdot \vec{v})/\|\vec{d}\|^2$, onde a é a área do oclusor, $\vec{n}$ a normal do oclusor, $\vec{v}$ a direção de visão do observador e $\vec{d}$ o vetor do observador ao centro do oclusor. Como o oclusor é um conjunto de células que está sempre defrontando o observador, tomamos $(\vec{n} \cdot \vec{v}) = 1$. Apenas oclusores estendidos que ultrapassam um valor mínimo de ângulo sólido são considerados para o cálculo de oclusão. Esse valor é determinado empiricamente e depende do tamanho dos oclusores da cena utilizada.

3.3.5 Cálculo de Oclusão

O cálculo de oclusão consiste em determinar quais células estão sendo escondidas por um oclusor – estendido ou não – com relação ao observador. Após a extensão de um oclusor, uma região de oclusão é construída para esse fim. Em 2D, a região de oclusão é um polígono semi-infinito cujos lados semi-infinitos são colineares às retas de suporte do observador e o oclusor e os lados finitos são os lados do oclusor estendido visíveis ao observador. Visto que essa região de oclusão é sempre um polígono convexo (em decorrência da heurística de extensão de oclusores), é possível discretizá-la eficientemente na grade regular através da adaptação de um algoritmo de rasterização de polígonos convexos a essa estrutura de dados e alterar para o estado de *oclusa* as células coincidentes com a discretização. Por outro lado, como forma de manter a característica conservadora desse procedimento, é preciso garantir que nenhuma célula pelo menos parcialmente visível seja erroneamente classificada como oclusa.

A determinação exata das células contidas na região de oclusão pode ser uma tarefa custosa, ainda que características de coerência de amplitude e coerência de varredura sejam utilizadas para testar poucas células por linha de rasterização. Melhor seria se a rasterização já garantisse que todas as células estão contidas na região de oclusão. De fato, esse resultado pode ser obtido através da utilização do contorno espesso introduzido na seção 3.3.1 para subestimar o número de células oclusas. Entretanto, adaptamos essa idéia para um algoritmo mais simples, que abstrai o uso de vetores normais, mas gera resultados possivelmente mais conservadores. A posição do centro de cada célula é dada em coordenadas inteiras pelo índice da célula na grade regular e assim utilizamos tais índices como vértices da região de oclusão, calculando a declividade das arestas semi-infinitas considerando que o observador está no centro da célula que o contém. Finalmente, discretizamos a região de oclusão com um algoritmo de rasterização de polígonos que toma a amostragem no centro de cada célula. Em resumo, efetuamos todo o cálculo de oclusão em coordenadas inteiras tomadas no centro de cada célula. O efeito obtido é o mesmo de retrair os lados de uma região de oclusão válida para

todos os possíveis pontos de vista da célula contendo o observador, por $(|n_x| + |n_y|)/\|\vec{n}\| \cdot d/2$, onde d é a largura de uma célula e $\vec{n}$ o vetor normal do lado considerado. Esse termo descreve a menor distância de retração necessária para garantir uma rasterização conservadora dessa região de oclusão válida para todos os pontos de vista da célula contendo o observador. Como esse volume retraído está sempre dentro do volume de oclusão válido para todos os pontos de vista da célula contendo o observador, o resultado é sempre conservador.

3.4 Caso 3D

Na seção anterior utilizamos o conceito de uma grade regular 2D para descrever o algoritmo de determinação de visibilidade em cenas bidimensionais. Nesta seção estendemos o algoritmo para cenas tridimensionais utilizando conceitos análogos aos introduzidos até agora.

Devido à utilização da grade regular como subdivisão espacial, o desempenho do nosso algoritmo depende fortemente do número de células da subdivisão que precisam ser percorridas dentro do volume de visão ou modificadas por objetos potencialmente visíveis e TBVs. No caso 3D essa dificuldade é ainda maior. Por exemplo, para uma cena 2D representada por uma grade regular de resolução 100x100 e uma cena 3D equivalente representada por uma grade de resolução 100x100x100, há um aumento de duas ordens de magnitude no número de células. Desse modo, mesmo podendo estender diretamente os algoritmos utilizados no caso 2D para 3D, procuramos introduzir outras técnicas que minimizassem o número de células atingidas. O mesmo cuidado também foi utilizado na extensão das estruturas de dados para o caso 3D. Enquanto as matrizes $\mathcal{O}$, $\mathcal{H}$ e $\mathcal{I}$ tornam-se matrizes tridimensionais, a matriz $\mathcal{T}$ é substituída por apenas três matrizes $\mathcal{T}_{xy}$, $\mathcal{T}_{xz}$, $\mathcal{T}_{yz}$ bidimensionais. Os detalhes dessas modificações são discutidos nas seções a seguir.

3.4.1 Discretização da Cena

Para a discretização da cena na grade regular, utilizamos modelos convexos simplificados dos objetos, tais como esferas e caixas limitantes. O teste de inclusão das células nos modelos limitantes é então realizado no espaço do objeto e a discretização se resume a uma seqüência de testes de interseção entre duas geometrias convexas: a geometria que constitui a fronteira da célula no espaço do objeto e a geometria do modelo limitante. Para que os resultados sejam conservadores, dois tipos de geometria devem ser mantidos: uma que contém totalmente o objeto e outra que está totalmente contida no objeto. A primeira superestima a região da cena ocupada pelo modelo e é utilizada para determinar as células intersectadas pelo objeto; a segunda subestima o interior do objeto e é utilizada na determinação de células opacas. Um exemplo de simplificação de geometria por caixas limitantes é mostrado na Figura 3.9.

Embora a construção de caixas ou esferas limitantes seja trivial, o mesmo não ocorre com a determinação da geometria totalmente contida em um objeto [2]. Para otimizar a renderização dos oclusores na

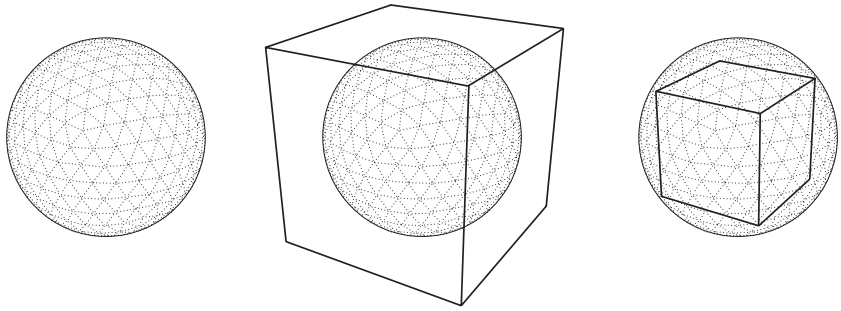


Figura 3.9: Geometria simplificada para discretização de um objeto. (a) Objeto original. (b) Modelo simplificado para determinação de células intersectadas pelo objeto. (c) Modelo simplificado para determinação de células opacas.

técnica de mapas de oclusão hierárquicos (HOM), Zhang [47] sugeriu uma modificação do algoritmo de *envelopes de simplificação* proposto por Cohen *et al.* [7] originalmente destinado à construção de hierarquias de níveis de detalhes. Entretanto, como em outras técnicas semelhantes [15, 22], esse algoritmo não permite restringir a forma do modelo simplificado para um objeto convexo, como é desejado no nosso caso. Em geral, essa geometria deve ser determinada manualmente.

Visto que as limitações apresentadas para o método de simplificação da geometria podem não ser toleráveis, sugerimos outra forma de discretização baseada no algoritmo que utilizamos em cenas 2D. Nessa técnica, os objetos são fatiados em planos sucessivos ao longo de um eixo coordenado e cada seção cruzada resultante é discretizada com o algoritmo utilizado no caso 2D. Esse processo também pode ser feito no espaço da imagem, utilizando a correspondência entre *pixels* do *frame buffer* e células da subdivisão. No OpenGL, as seções cruzadas são determinadas por planos de cerceamento definidos pelo usuário com a função `glClipPlanes()`[3] em um objeto rasterizado com projeção ortográfica. Visto que a rasterização de um objeto compreende somente a rasterização da sua superfície, o objeto cerceado por um plano de corte deve ser convenientemente "tampado" para que o seu interior tenha uma aparência opaca. Blythe e McReynolds [5] propuseram uma técnica simples para tampar sólidos usando o *buffer* de estêncil em OpenGL. Por fim, utilizamos a mesma correspondência entre *pixels* e células utilizadas no caso 2D para atribuir os *pixels* da tampa do objeto fatiado às células intersectadas pelo objeto. Esses mesmos *pixels* também determinam as células opacas, com exceção dos *pixels* da silhueta da tampa de cada seção cruzada. Essa última condição é necessária para manter a discretização conservadora, pois exclui a possibilidade de que células parcialmente contidas no objeto sejam consideradas opacas.

Embora essa última técnica não apresente as limitações do algoritmo de discretização baseado em geometria simplificada, ela é lenta devido à necessidade de carregar várias vezes o *frame buffer* na memória do sistema. Portanto, deve ser utilizada preferencialmente para discretizar objetos estáticos em $\mathcal{I}$ durante

[3] O plano de corte da frente do volume de visão também pode ser utilizado.

Posição da célula-semente	Planos de percurso iniciais
$(\|x\| > \|y\|) \wedge (\|x\| > \|z\|)$	zy
$(\|y\| > \|x\|) \wedge (\|y\| > \|z\|)$	zx
$(\|z\| > \|x\|) \wedge (\|z\| > \|y\|)$	xy
$\|x\| = \|z\|$	zy, xy
$\|y\| = \|z\|$	zx, yx
$\|y\| = \|x\|$	yz, xz

Tabela 3.2: Planos de percurso das células do volume de visão 3D segundo a posição de uma célula-semente (x, y, z) em coordenadas relativas à célula contendo o observador.

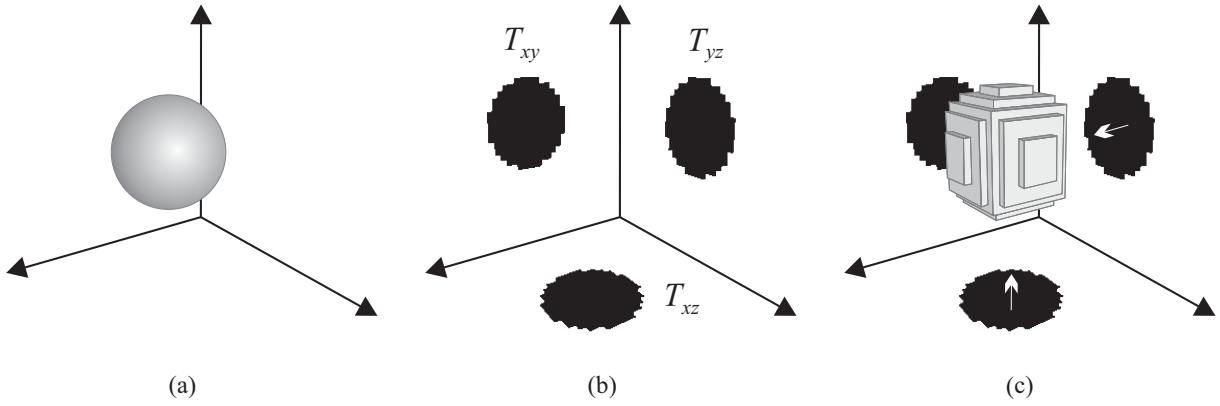


Figura 3.10: Discretização otimizada de TBVs esféricos. (a) TBV original. (b) projeção do TBV em $\mathcal{T}_{xy}$, $\mathcal{T}_{xz}$ e $\mathcal{T}_{yz}$. (c) reconstrução do TBV discretizado.

pré-processamento.

3.4.2 Percurso do Volume de Visão

A estratégia de percurso das células do volume de visão usando a métrica xadrez pode ser estendida de forma direta para o caso 3D. A distância xadrez entre dois pontos (x_1, y_1, z_1) e (x_2, y_2, z_2) do $\mathbb{R}^3$ é dada na forma $max(|x_1 - x_2|, |y_1 - y_2|, |z_1 - z_2|)$. Para a discretização da linha de visão, utilizamos uma extensão do algoritmo de Bresenham para linhas 3D. Entretanto, o percurso das células do volume de visão a partir de cada célula-semente requer mais direções do que apenas as duas suficientes no caso 2D e descreve uma varredura em planos de células ao invés de linhas. Os planos de percurso iniciais a partir de uma célula-semente dependem da posição dessa célula em relação à célula contendo o observador. Um catálogo é mostrado na Tabela 3.2.

A estratégia de discretização otimizada de TBVs no caso 2D também pode ser estendida de forma intuitiva para o caso 3D. Enquanto TBVs 2D podem ser discretizados como círculos vazios de modo a reduzir o número de células de $\mathcal{T}$ modificados, podemos discretizar TBVs 3D como esferas ao invés de

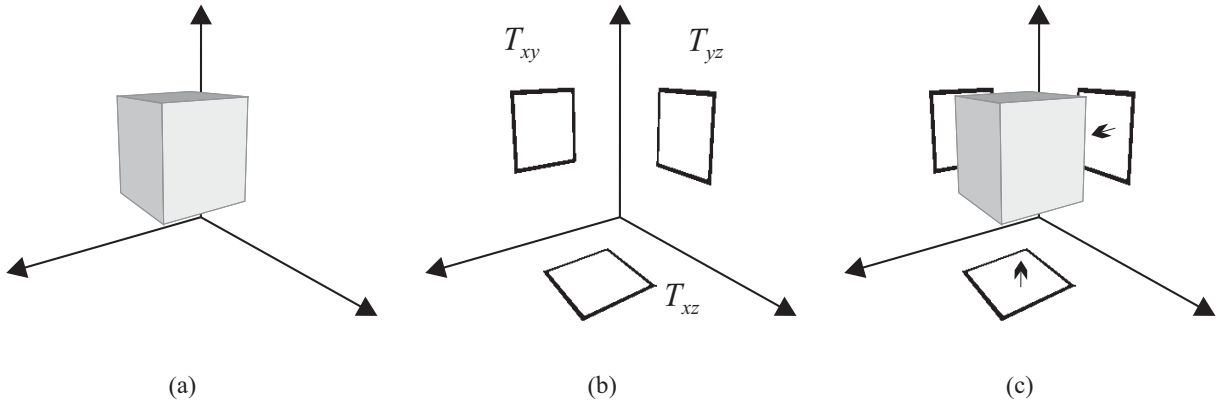


Figura 3.11: Discretização otimizada de TBVs cubóides. (a) TBV original. (b) projeção das arestas do TBV em $\mathcal{T}_{xy}$, $\mathcal{T}_{xz}$ e $\mathcal{T}_{yz}$. (c) reconstrução do TBV discretizado.

bolas. Entretanto, é possível reduzir ainda mais o número de células de $\mathcal{T}$ não-vazias (de $O(n^3)$ para $O(n^2)$ para um TBV dado) e substituir $\mathcal{T}$ por apenas duas matrizes bidimensionais, levando em conta que a geometria de TBVs de objetos de movimentação arbitrária é sempre convexa. Entretanto, para um melhor compromisso entre a discretização e o tempo de reconstrução dos TBVs discretizados, optamos por substituir $\mathcal{T}$ por três matrizes bidimensionais.

Um TBV convexo pode ser representado na grade regular pela sua projeção ortográfica em três matrizes bidimensionais $\mathcal{T}_{xy}$, $\mathcal{T}_{xz}$ e $\mathcal{T}_{yz}$ coincidentes com os planos coordenados, nas quais o identificador do TBV é adicionado às células da matriz atingidas pela projeção. Para identificar posteriormente o TBV durante o percurso do volume de visão, basta verificar se as células coincidentes com a projeção da célula atual nas matrizes $\mathcal{T}_{xy}$, $\mathcal{T}_{xz}$ e $\mathcal{T}_{yz}$ possuem o mesmo identificador do TBV. Um exemplo é ilustrado na Figura 3.10.

Entretanto, notamos que a redução do problema de discretização de TBVs de 3D para 2D introduz uma complicação à utilização de TBVs discretizados como esferas ao invés de bolas. A projeção ortográfica de um TBV esférico numa matriz 2D resulta sempre num círculo preenchido. Dessa forma, a reconstrução do TBV a partir de suas projeções é sempre uma bola (na verdade, uma interseção entre três cilindros) e não uma esfera como desejamos. Felizmente, há um meio de manter a utilização da discretização otimizada de TBVs, sem abrir mão do emprego de apenas três matrizes 2D, utilizando para isso TBVs em forma de cubos ao invés de esferas (para garantir a característica conservadora do algoritmo, o TBV cubóide deve superestimar o TBV esférico). A discretização otimizada de TBVs cubóides consiste em projetar somente as arestas do cubo em cada matriz e adicionar o identificador do TBV somente às células das matrizes bidimensionais que intersectam as arestas projetadas. Dessa forma, esse conjunto de células forma um quadrado vazio. Durante o percurso do volume de visão, os TBVs são encontrados testando se a projeção da célula atual nas matrizes possui o mesmo identificador em $\mathcal{T}_{xy}$, $\mathcal{T}_{xz}$ e $\mathcal{T}_{yz}$. Para um dado identificador, esse teste de existência deve satisfazer a condição $(\mathcal{T}_{xy} \vee \mathcal{T}_{yz}) \wedge (\mathcal{T}_{xy} \vee \mathcal{T}_{xz}) \wedge (\mathcal{T}_{yz} \vee \mathcal{T}_{xz})$. O TBV assim

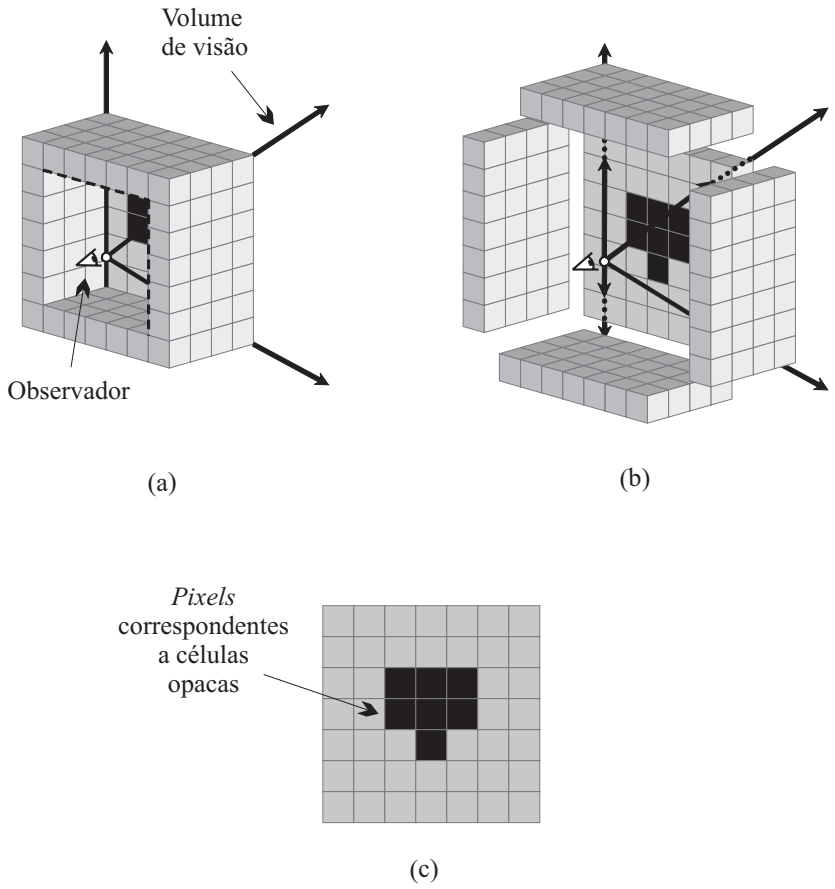


Figura 3.12: Extensão de oclusores em 3D. (a) Subconjunto de células com a mesma distância xadrez ao observador. (b) Decomposição em planos de células. (c) Mapa de bits resultante de um dos planos.

reconstruído será um cubo vazio e o número de células não-vazias de $\mathcal{T}$ neste caso cai de $O(n^3)$ para apenas $O(n)$ para cada TBV. A Figura 3.11 mostra um exemplo.

3.4.3 Extensão de Oclusores e Cálculo de Oclusão

Schaufler *et al.* [36] propuseram uma heurística de extensão de oclusores 3D semelhante à utilizada no caso 2D, que consiste em agregar células opacas ou oclusas adjacentes à célula opaca inicial de modo a maximizar o ângulo sólido subentendido entre o observador e o oclusor. O agregado de células resultante ainda deve implicar um oclusor estendido convexo. Essa restrição é necessária para produzir um volume de oclusão também convexo que facilite o cálculo de oclusão. Entretanto, o acréscimo de uma dimensão adicional de extensão dificulta a determinação de um oclusor estendido eficaz, pois, ao contrário do caso 2D, existem várias soluções possíveis a partir de uma mesma célula opaca inicial.

Por causa da distribuição uniforme das células na grade regular e a conseqüente facilidade de realizar o cálculo de oclusão através da rasterização de volumes de oclusão nessa estrutura, a restrição da

extensão de oclusores para gerar volumes de oclusão convexos não é necessária. Assim, exploramos a métrica xadrez para desenvolver uma heurística mais simples que integra a extensão de oclusores no próprio percurso do volume de visão. A procura por células opacas ou oclusas é restrita ao conjunto de células de distância xadrez igual à distância da célula-semente atual à célula contendo o observador. A Figura 3.12(a) mostra um subconjunto dessas células cuja distância xadrez à célula contendo o observador é igual a quatro. Intuitivamente, esses conjuntos podem ser vistos como "cascas de cebola" de células que envolvem a célula contendo o observador. Essas camadas são cubóides em decorrência da métrica xadrez.

Para cada percurso de planos a partir de uma célula-semente, todas as células opacas ou oclusas que têm a mesma distância xadrez são consideradas como oclusores, mesmo que não sejam ligadas por um caminho 4-conexo (nas Figuras 3.12 e 3.13, essas células são mostradas em preto). Isso significa que vários oclusores desconexos podem ser gerados durante o percurso de um plano de células do volume de visão. Entretanto, devido à métrica xadrez, esse conjunto de células subentende a superfície de um cubo de lados alinhados aos eixos coordenados, de modo que toda célula opaca ou oclusa encontrada estará sempre em um dos lados desse cubo. Utilizando novamente a correspondência natural entre células da grade regular e *pixels* do *frame buffer*, podemos considerar cada lado do cubo (Figura 3.12(b)) como um mapa de bits no qual as células opacas e oclusas são os *pixels* opacos (Figura 3.12(c)). Para realizar o cálculo de oclusão a partir desse conjunto de células opacas e oclusas possivelmente desconexos, vetorizamos cada um dos mapas de bits de modo a construir o que serão os *polígonos-tampa* dos volumes de oclusão (Figura 3.13(a)). Polígonos-tampa são polígonos simples, possivelmente com buracos, que representam os *pixels* opacos do mapa de bits, de forma vetorial. Em seguida, cada vértice dos polígonos-tampa é deslocado ao longo do eixo focal do observador que passa por esse vértice, de modo a formar os volumes de oclusão no espaço do objeto (Figura 3.13(b)). Intuitivamente, essa etapa corresponde a determinar volumes de sombra gerados pelos polígonos-tampa, considerando o observador como uma fonte de luz pontual. Finalmente, cada seção cruzada de cada volume de oclusão é rasterizada em $\mathcal{H}$, modificando o estado de cada célula da rasterização para ocluso.

A etapa de vetorização dos mapas de bits deve ser realizada de modo que os polígonos-tampa gerem volumes de oclusão cujas seções cruzadas possam ser rasterizadas de forma conservadora. Um forma simples de obter resultados conservadores é obtida aplicando a correção de amostragem proposta na Seção 3.3.5 para cada polígono-tampa resultante da vetorização.

3.5 Comentários

Até agora discutimos o tratamento de cenas em que todos os objetos, sem exceção, movem-se arbitrariamente dentro do volume delimitado pela grade regular. Contudo, nas cenas que encontramos comumente na prática, a maioria dos modelos é estática com relação à grade. Por exemplo, numa cena urbana apenas pessoas e carros são objetos dinâmicos; os prédios, muros e demais elementos que compõem o

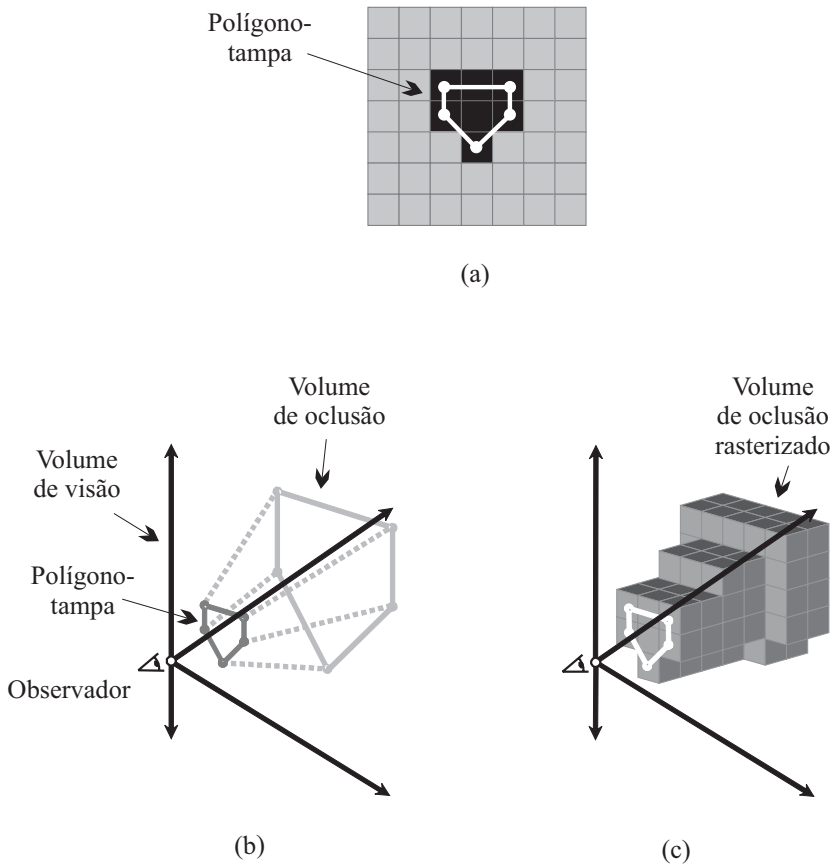


Figura 3.13: Cálculo de oclusão em 3D. (a) Polígono-tampa de um mapa de bits. (b) Volume de oclusão gerado pelo polígono-tampa. (c) Rasterização do volume de oclusão.

cenário principal são todos estáticos.

Embora objetos estáticos possam ser considerados como objetos dinâmicos de velocidade nula, podemos tratar esse caso de forma mais eficiente levando em conta que um objeto estático, quando invisível, não precisa de um TBV. Além disso, um objeto estático, quando visível, não precisa ser discretizado na grade regular para cada quadro de exibição. Dessa forma, introduzimos uma nova matriz de oclusores, a matriz de oclusores estáticos $\mathcal{O}_s$ que contém células opacas apenas de objetos estáticos. Todos os objetos estáticos são discretizados em $\mathcal{O}_s$ e $\mathcal{I}$ numa etapa de pré-processamento. Em tempo de execução, o conteúdo de $\mathcal{O}_s$ é copiado para $\mathcal{O}$ antes do início da determinação de visibilidade para o quadro atual. Esse procedimento é necessário porque o conteúdo de $\mathcal{O}$ é sempre reiniciado no final do percurso do volume de visão. Basta, por fim, não designar nenhum TBV aos objetos estáticos encontrados em $\mathcal{I}$. Observe que essa otimização só é possível quando os objetos considerados são estáticos com relação à grade regular. Tal otimização deixa de ser válida em caso contrário, ainda que os objetos sejam estáticos com relação ao observador, pois isso não isenta modificações do conteúdo da grade regular. Da mesma forma, essa estratégia não pode ser aplicada para otimizar o caso em que o observador é estático.

Capítulo 4

Implementação e Resultados

O algoritmo foi implementado inicialmente para cenas 2D, em Visual C++ usando OpenGL. A Figura 4.1 mostra três *snapshots* do visualizador de cenas 2D com diferentes números de objetos dinâmicos e estáticos, numa grade regular de resolução 256x256. A versão 3D do algoritmo foi construída a partir do código da versão 2D, portanto também implementada em C++/OpenGL. Os testes de desempenho foram conduzidos num PC com processador Athlon XP 1.5GHz e aceleradora gráfica usando *chipset* GeForce2 GTS. As cenas testadas foram constituídas de aglomerados de esferas de tamanhos variados e de movimentação e velocidades arbitrárias. Um número variado de objetos foi utilizado, até um máximo de 5.000 esferas, totalizando 6 milhões de polígonos. Apesar do grande número de primitivas, a qualidade de renderização da cena foi modesta. As cenas não foram texturizadas e apenas 3 fontes de luzes direcionais foram utilizadas para um sombreamento *Gouraud*. Dois *snapshots* do visualizador 3D são mostrados na Figura 4.2. Cada *snapshot* é constituído de uma imagem da cena vista pelo observador e uma visualização do conteúdo da grade regular. A cena visualizada na ilustração de cima da Figura 4.2 contém 7.000 objetos numa grade de resolução 100x100x100 e a imagem de baixo corresponde à visualização de uma cena de 5.000 objetos numa grade de resolução 50x30x50.

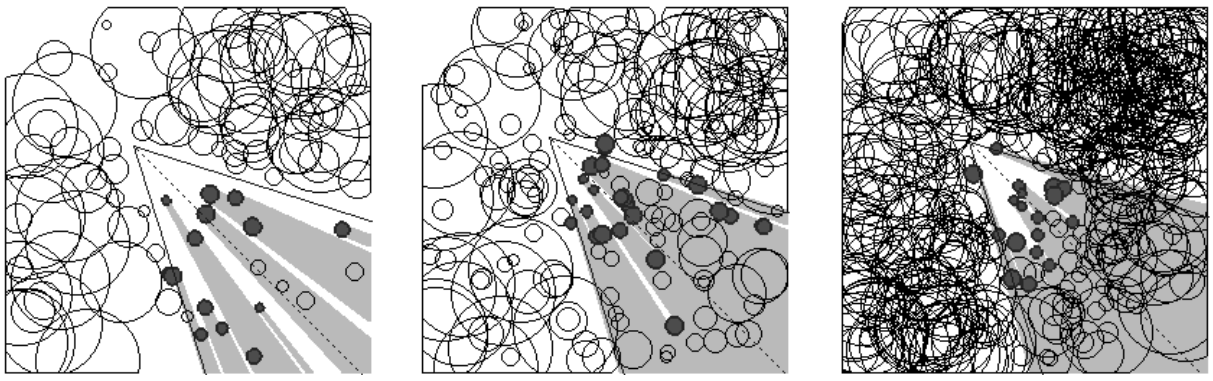


Figura 4.1: *Snapshots* do visualizador para cenas 2D.

Devido principalmente à escassez de algoritmos de descarte de visibilidade em cenas dinâmicas 3D, comparamos os resultados dos testes apenas com *z-buffer* em *hardware*. Como já mostramos no Capítulo 2, trabalhos correlatos como HZB e HOM foram desenvolvidos inicialmente para cenas estáticas, de modo que em cenas dinâmicas não atingem uma complexidade sensível à saída. O HZB, além disso, requer *hardware* gráfico especial. A técnica de HOM pode ser adaptada para a utilização de TBVs, como faz a API dPVS [2], também citada anteriormente. Embora a dPVS tenha semelhanças com a arquitetura que propomos, seus desenvolvedores não adotam uma política de disponibilização de código-fonte (uma licença gratuita para fins de pesquisa pode ser obtida, mas o código é invariavelmente fechado). Esse entrave impossibilitou a realização de testes comparativos importantes, como tempo de manutenção de TBVs e desempenho do percurso do volume de visão.

4.1 Detalhamento da Implementação do Caso 3D

A implementação do algoritmo de determinação de visibilidade compreendeu detalhes que influenciaram de forma significativa os resultados dos testes de desempenho, ainda que não tenham alterado seu comportamento assintótico. Esses detalhes são expostos a seguir, para cada etapa do algoritmo:

- **Discretização da cena:** O conteúdo das matrizes $\mathcal{O}$, $\mathcal{H}$, $\mathcal{I}$, $\mathcal{T}$, foi atribuído segundo o procedimento introduzido na Seção 3.4.1 que explora uma representação simplificada dos objetos por caixas ou esferas limitantes. Essa abordagem foi escolhida por causa da sua eficiência e facilidade de implementação. Por outro lado, deixamos de realizar testes de desempenho em cenas genéricas devido a dificuldade de determinar as geometrias simplificadas manualmente para cenas muito grandes. Nas cenas utilizadas, a determinação das geometrias simplificadas só foi trivial porque todos os objetos eram esferas. Para a organização dos dados das matrizes, utilizamos um misto de código original e código disponível na STL (*Standard Template Library*) do ANSI C++. Para a organização dos identificadores dos objetos em cada célula de $\mathcal{I}$ e $\mathcal{T}$, utilizamos árvores binárias balanceadas implementadas segundo a STL. Desse modo, operações de consulta, inserção e remoção de elementos arbitrários puderam ser realizadas em tempo logarítmico. Por outro lado, as matrizes $\mathcal{O}$, $\mathcal{O}_s$ e $\mathcal{H}$ foram implementadas como seqüências de bits. Embora a STL do Visual C++ fornecesse um formato próprio para representar seqüência de bits, seu desempenho foi muito inferior ao obtido pelo código que produzimos.

- **Percurso do volume de visão:** A discretização da linha de visão foi realizada com o algoritmo de Bresenham de desenho de linhas, adaptado com poucas modificações à grade regular. O percurso das células do volume de visão foi otimizado com laços de iteração como forma de realizar o menor número possível de testes de células dentro do volume de visão. Essa forma de implementação explorou a capacidade de realizar o percurso em direções alinhadas aos eixos segundo a métrica xadrez.

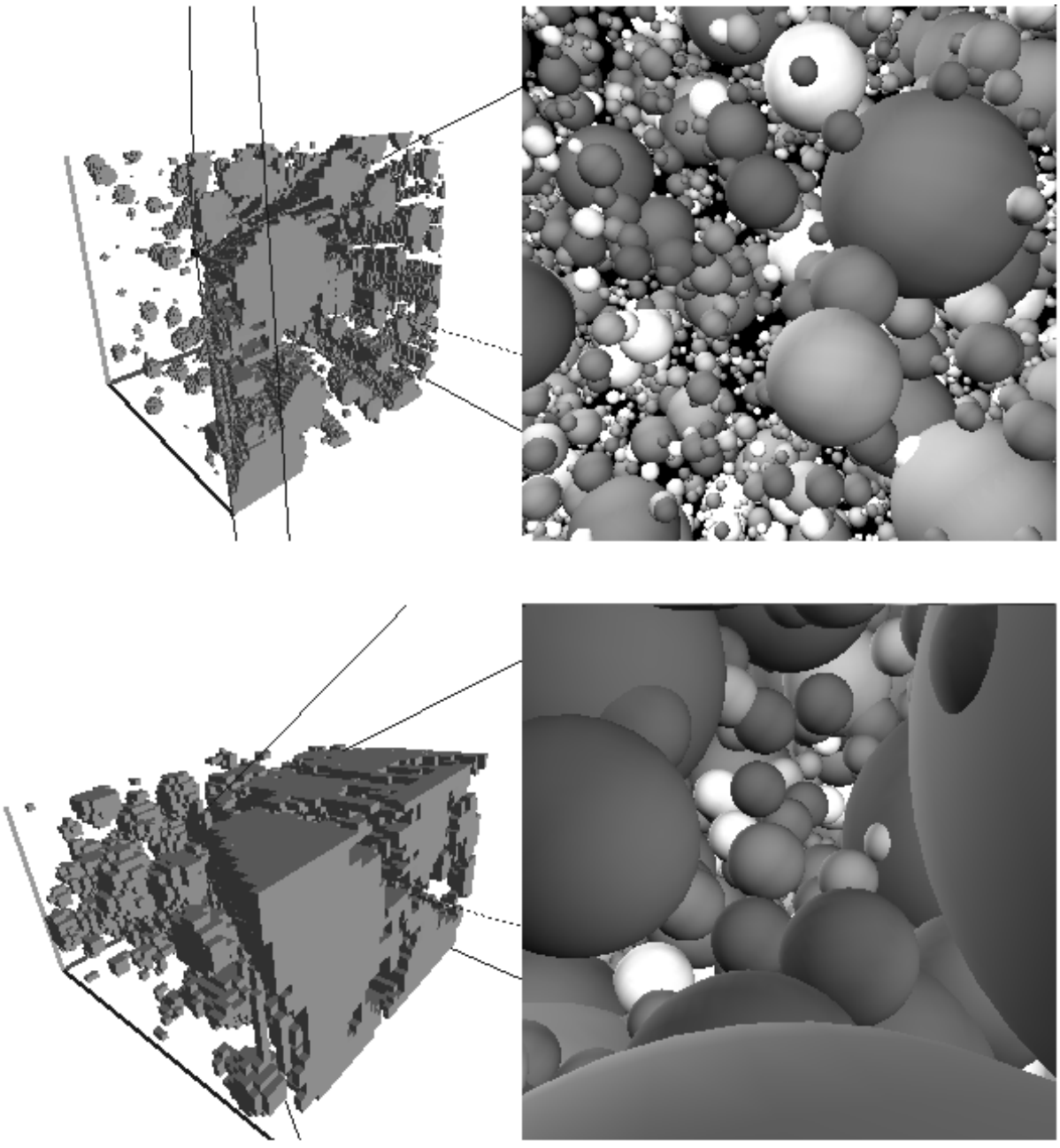


Figura 4.2: *Snapshots* do visualizador para cenas 3D.

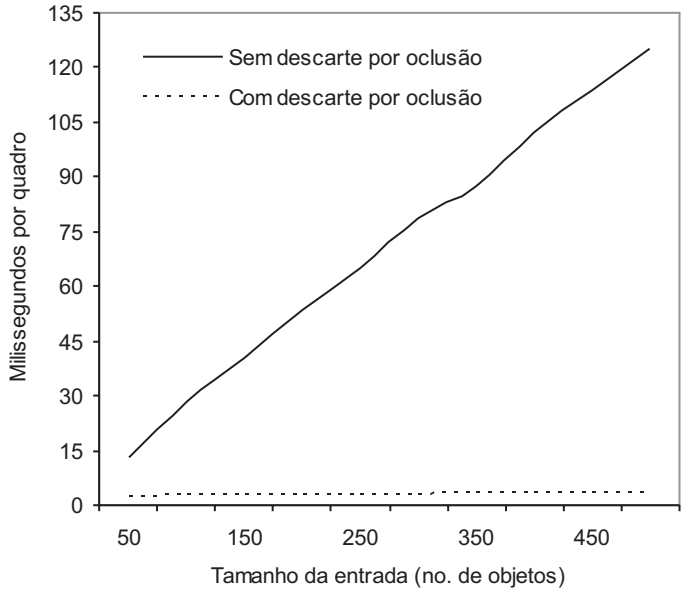


Figura 4.3: Desempenho da renderização para um número crescente de objetos dinâmicos escondidos.

- **Extensão de oclusores e cálculo de oclusão:** Construímos um algoritmo original para realizar a vetorização dos mapas de bits construídos durante o percurso do volume de visão. Ao invés de produzir um polígono orientado a partir do mapa de bits, nossa técnica utilizou uma representação intermediária suficiente para os nossos propósitos, ainda que sendo redundante em comparação com um algoritmo ótimo de vetorização. A construção dessa representação intermediária constituiu apenas numa varredura vertical e horizontal do mapa de bits, armazenando as coordenadas de início e término das linhas e colunas de *pixels* correspondentes a *voxels* opacos ou oclusos. Tais coordenadas definiram os vértices utilizados na geração do volume de oclusão e a rasterização para cada seção cruzada do volume de oclusão foi então realizada com base numa varredura dessa representação. A heurística de determinação de bons candidatos a oclusores proposta por Coorg e Teller [11] não foi utilizada nessa implementação do algoritmo no caso 3D e assim todos os oclusores estendidos foram utilizados para o cálculo de oclusão.

4.2 Desempenho da Manutenção de Objetos Dinâmicos Escondidos

O primeiro teste de desempenho foi destinado a medir a eficiência da manutenção de objetos dinâmicos escondidos na grade regular, para um número crescente de objetos localizados fora do volume de visão. Os resultados são mostrados na Figura 4.3. O desempenho foi medido com relação ao tempo de renderização obtido em cenas com e sem o auxílio do descarte por oclusão proporcionado pelo nosso algoritmo. Nas cenas que não utilizaram o nosso algoritmo, foi utilizado apenas o descarte de faces invertidas e

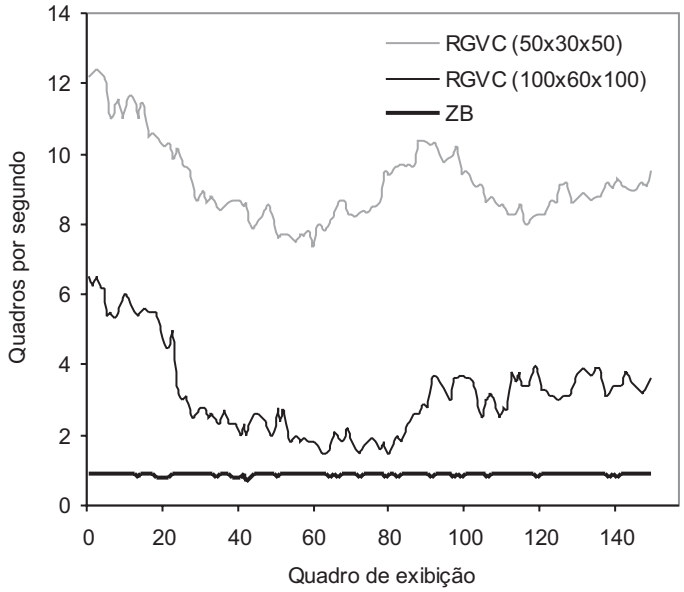


Figura 4.4: Desempenho da renderização numa caminhada pela cena.

volume de visão implementados em *hardware*. Os resultados mostraram que o desempenho da renderização sem o descarte por oclusão teve um crescimento linear, como já era esperado numa abordagem que verifica todos os objetos. Por outro lado, o desempenho obtido pelo nosso algoritmo resultou num comportamento constante, em acordo com a complexidade sensível à saída desejada, uma vez que o observador não estava enxergando nada (na verdade, nenhuma célula foi acessada no percurso do volume de visão). Além disso, a sobrecarga de manutenção dos TBVs foi praticamente desprezível (3,4ms para 500 objetos dinâmicos escondidos). De fato, conseguimos obter taxas de exibição em torno de 100 quadros por segundo para cenas contendo mais de 10.000 objetos dinâmicos escondidos. Esse resultado favorável foi devido principalmente à flexibilidade de manutenção dos objetos dinâmicos em grades regulares, à otimização de discretização de TBVs e à redução da grade regular 3D para apenas três matrizes bidimensionais. Ainda que o teste de desempenho tenha sido conduzido sobre uma grade regular de baixa resolução (50x30x50), a utilização de uma resolução duplicada em cada eixo coordenado (100x60x100) adicionou uma média de apenas 0,20ms para esse tempo. Desse modo, embora não tenhamos realizado testes de comparação com outras estruturas de dados, os resultados exibidos servem como argumento para acreditarmos que o desempenho da manutenção de TBVs em grades regulares é de fato superior ao obtido em estruturas de dados hierárquicas como *octrees* e árvores BSP.

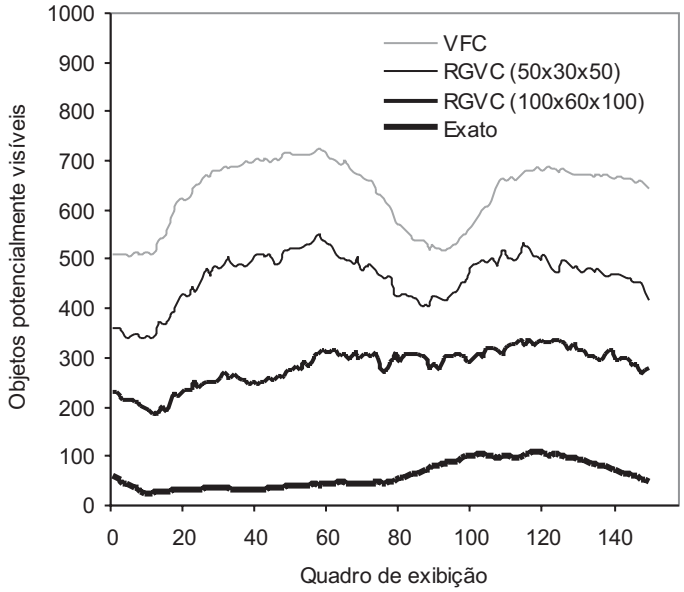


Figura 4.5: Aproximação do PVS ao conjunto exato de objetos visíveis.

4.3 Desempenho da Renderização numa Caminhada

O segundo teste de desempenho foi medido para uma caminhada do observador imerso num cenário composto de 200 objetos dinâmicos e 4.800 objetos estáticos. A caminhada foi constituída de 150 quadros de exibição. O resultado é mostrado na Figura 4.4. O desempenho da renderização, em quadros por segundo, foi comparado com *z-buffer* em *hardware*. Além disso, o desempenho do nosso algoritmo (denotado por RGVC) foi medido segundo a utilização de duas resoluções da grade regular. O gráfico mostra que o tempo de renderização da abordagem usando somente *z-buffer* é constante para todos os quadros. Esse comportamente já era esperado, pois todos os 5.000 objetos são enviados indistintamente ao fluxo de renderização. Por outro lado, o desempenho devido ao nosso algoritmo apresenta um comportamento irregular, que pode ser explicado por sua sensibilidade à saída: quanto menor o número de objetos potencialmente visíveis, maior o desempenho (compare também com o gráfico da Figura 4.5). A utilização de uma grade regular de baixa resolução (50x30x50) implicou um desempenho muito superior à grade de alta resolução (100x60x100). Embora esse comportamento corrobore com nossa hipótese de que o maior gargalo da utilização de uma grade regular está no elevado número de células visitadas, a verdadeira causa desse resultado não parece estar atrelada ao uso de grades regulares, como discutimos na seção 4.4.

4.4 Aproximação do PVS à Solução Exata

Recentemente, Cohen-Or *et al.* [9] sugeriram como procedimento satisfatório para se medir a eficácia entre algoritmos conservadores de visibilidade a comparação do grau de conservação dos PVSs gerados com relação a uma solução ideal. Num algoritmo ideal de descarte de visibilidade o PVS deve ser igual ao conjunto exato de objetos visíveis, *i.e.*, deve conter todos os objetos pelo menos parcialmente visíveis ao observador e apenas estes. No outro extremo, se o algoritmo determinar todos os objetos da cena como potencialmente visíveis, o resultado terá sido o mesmo que o obtido sem qualquer algoritmo de descarte de visibilidade. Infelizmente, a maioria da literatura existente sobre algoritmos de descarte por oclusão não contém esse tipo de medição. Em contrapartida, é mais comum encontrarmos medições de porcentagem de geometria descartada como trivialmente escondida em uma cena dada – algo que não possibilita uma comparação justa entre os algoritmos, pois é um critério muito dependente do contexto (*e.g.*, basta que o observador não olhe para nenhum objeto e 100% da geometria será descartada). Em trabalhos mais recentes, como o de Aila e Miettinen [2], os autores sugerem que a razão entre o número de objetos potencialmente visíveis e o número exato de objetos visíveis deve ser pelo menos igual a dois para que o algoritmo de visibilidade seja considerado satisfatório. Aparentemente, esse é o resultado obtido na API dPVS dos mesmos autores. Da mesma forma, nos testes conduzidos na tese de Zhang [47] sobre a técnica de mapa de oclusão hierárquico, essa razão ficou numa média entre dois e quatro.

O gráfico mostrado na Figura 4.5 contém a quantidade de objetos potencialmente visíveis gerados pelo nosso algoritmo durante a caminhada realizada na cena utilizada no teste anterior. Os resultados são comparados com o número de objetos potencialmente visíveis gerados apenas com descarte por volume de visão (denotado VFC) e o número exato de objetos visíveis (denotado Exato). Os resultados gerados pelo nosso algoritmo de visibilidade (RGVC) estiveram necessariamente limitados a essas duas curvas e foram obtidos para duas resoluções da grade regular. Para a grade regular de 50x30x50 células, observamos que houve uma superestimativa excessiva do número de objetos visíveis. Os resultados se aproximaram em torno de duas vezes do conjunto exato quando utilizamos a grade de 100x60x100 células, mas como já mostramos no teste anterior, o desempenho foi reduzido em mais da metade. Deve-se notar, porém, que o grau de aproximação do PVS ao conjunto exato não dependeu exclusivamente da resolução da grade regular, mas da quantidade de células opacas da cena tratada e também do algoritmo de visibilidade utilizado, pois este empregou etapas que subestimaram o número de oclusores da cena (como as etapas de discretização da cena e cálculo de oclusão). Entretanto, acreditamos que esses resultados foram devidos principalmente ao uso exclusivo de fusão e determinação de oclusores no espaço do objeto, pois mesmo em casos extremos em que apenas um objeto encobria todo o campo de visão do observador, o algoritmo ainda considerava vários objetos como potencialmente visíveis – algo que não ocorreria se a oclusão fosse realizada também na precisão da imagem, pois as partes dos oclusores perdidas durante a discretização de $\mathcal{O}$ seriam aproveitadas. De fato, Aila e Miettinen [2] já haviam presumido a necessidade da utilização de fusão de oclusores na precisão da imagem para obtenção de resultados precisos. Portanto, a baixa precisão obtida durante o uso

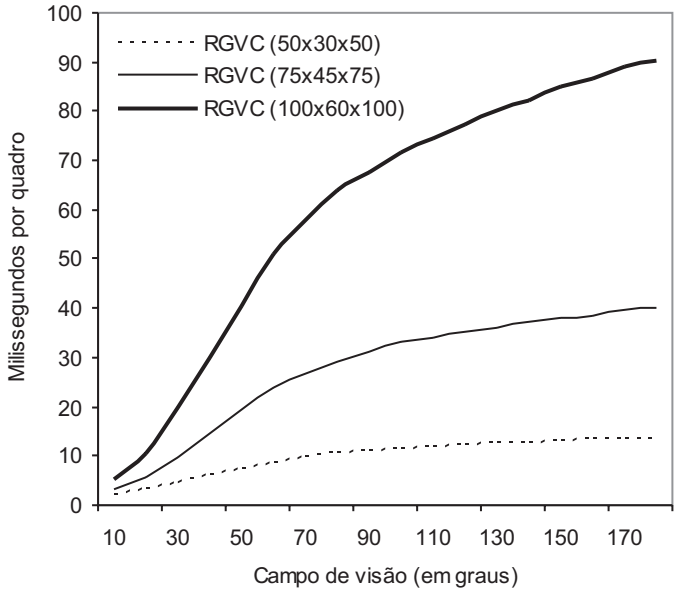


Figura 4.6: Desempenho do percurso do volume de visão.

de grades regulares de baixa resolução não constituiu uma limitação dessa estrutura de dados. Foi o fato do algoritmo de visibilidade relatar vários objetos escondidos como potencialmente visíveis que desencadeou um percurso de volume de visão mais custoso, pois um número maior de extensão de oclusores e cálculo de oclusão foi, por conseqüência, realizado na grade regular.

4.5 Tempo de Percurso do Volume de Visão

Para verificar o impacto do percurso das células do volume de visão sobre o desempenho da arquitetura, medimos o tempo de percurso dessas células numa cena vazia para diferentes ângulos de abertura da câmera, até um máximo de 180° que então correspondeu ao percurso de todas as células da grade regular. Os resultados são mostrados na Figura 4.6. A relação entre o ângulo de abertura da câmera e o tempo de percurso tem um aumento acentuado até aproximadamente o ângulo de 80° e torna-se menos acentuada a partir desse ponto, quando então a maior parte das células da grade regular já está dentro do volume de visão.

A sobrecarga do percurso cresce rapidamente com o aumento da resolução da grade regular. De fato, deve ser considerado que o número de células de uma grade regular de 50x30x50 para 100x60x100 aumenta oito vezes. Esse custo é ainda maior para cenas contendo oclusores, devido ao tempo adicional da extensão de oclusores e rasterização dos volumes de oclusão.

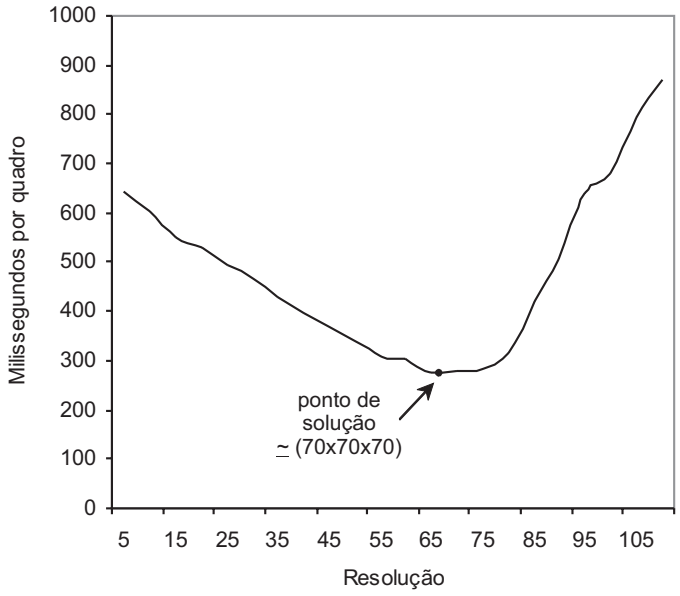


Figura 4.7: Desempenho em função da resolução da grade regular.

4.6 Desempenho em Função da Resolução

Os testes anteriores indicam que a resolução da grade regular influencia ambos o desempenho do descarte de visibilidade e a precisão dos resultados gerados. Por um lado, a diminuição da resolução implica uma diminuição do número de células que precisam ser acessadas durante o percurso do volume de visão. Por outro lado, essa diminuição da resolução gera resultados menos precisos, pois o aproveitamento da oclusão gerada pela cena é menor. Da mesma forma, o aumento da resolução implica um aumento de células que precisam ser percorridas, mas gera PVSs mais precisos.

Aparentemente, os resultados mostrados na Figura 4.4 fazem crer que a relação entre resolução e desempenho é inversamente proporcional (quanto menor a resolução, maior o desempenho). Entretanto, esse raciocínio contradiz com o fato do resultado ser cada vez menos preciso à medida que uma menor resolução é utilizada (ver Figura 4.5), pois um caso extremo de uma grade de resolução 1x1x1 deve classificar todos os objetos como potencialmente visíveis, o que é o mesmo que não utilizar qualquer estratégia de descarte por oclusão ou por volume de visão. Logo, para qualquer cena, deve existir um *ponto de solução* no qual a resolução da grade regular tem o melhor desempenho da visualização em comparação com outras resoluções. Para verificar a existência desse ponto, medimos o desempenho da visualização de uma cena composta por 4.800 objetos estáticos para diferentes resoluções da grade regular. Foi considerado um ponto de vista fixo e um ângulo de abertura da câmera de 55 graus. O resultado é mostrado graficamente na Figura 4.7, onde o eixo horizontal é um N que identifica a resolução da grade regular em NxNxN. O ponto de solução para este caso corresponde a uma grade regular de aproximadamente 70x70x70 células. O aumento da resolução a partir deste ponto diminui o desempenho de forma mais acentuada do que em relação à diminuição da

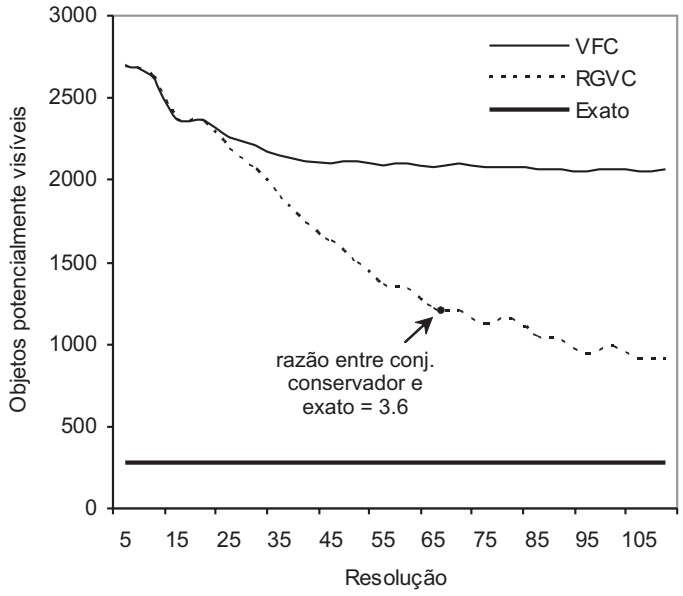


Figura 4.8: Precisão dos resultados em função da resolução da grade regular.

resolução, o que evidencia o impacto significativo do percurso das células do volume de visão, como já foi mostrado na Figura 4.6.

A precisão dos resultados em função da resolução da grade regular para a mesma cena do teste da Figura 4.7 é mostrada na Figura 4.8. Na legenda, VFC corresponde aos resultados obtidos apenas com descarte por volume de visão, RGVC corresponde aos resultados obtidos utilizando nossa estratégia e Exato corresponde ao número exato de objetos visíveis. Abaixo da resolução de 25x25x25 células, a precisão do resultado obtido é a mesma que a obtida apenas com descarte por volume de visão. Isso significa que, para resoluções menores que 25x25x25 células, nenhuma célula está sendo classificada como opaca dentro do volume de visão, ou o número de células opacas é insuficiente para esconder algum objeto.

No ponto de solução, a razão entre o conjunto conservador e o conjunto exato é de 3.6, o que ainda é superior à razão sugerida por Aila e Miettinen [2]. Entretanto, o aumento da resolução para até 100x100x100 células não gera resultados mais precisos na mesma proporção que o aumento no número de células, pois neste caso a razão entre os dois conjuntos cai apenas para 3.2.

4.7 Comentários

De modo a obter mais informações a respeito do desempenho da implementação do nosso algoritmo, executamos uma análise estatística de tempo de execução através do recurso de *profiling* disponível no compilador do Visual C++. Essa análise permitiu que observássemos quais etapas do algoritmo de visibilidade consumiram mais tempo, e por conseqüência, onde os esforços devem ser concentrados para melhorar

seu desempenho, ainda que na forma de ajustes finos da implementação. A cena testada para essa análise foi composta de 4.800 objetos estáticos e 200 dinâmicos, representados numa grade regular de 70x70x70 células. O comportamento das funções foi testado sobre cerca de 100 quadros de exibição. A seguir, comentamos o desempenho de cada etapa do algoritmo, em ordem decrescente de consumo de tempo:

1. A renderização dos objetos no *frame buffer* correspondeu a 5,6% do tempo total de execução do programa, com 63.218 chamadas de funções. Esse resultado correspondeu a cerca de 500 esferas renderizadas por quadro, de um total de 5.000 objetos da cena, quando apenas uma média de 100 esferas eram de fato visíveis. De acordo com os resultados da Figura 4.5 sobre a aproximação do PVS ao conjunto exato de objetos visíveis, seria possível reduzir o tempo de renderização pela metade se a razão entre o número de objetos relatados no PVS e o número de objetos do conjunto exato fosse, por exemplo, igual a dois – valor satisfatório sugerido por Aila e Miettinen [2].

2. A atualização da grade regular correspondeu a cerca de 55% do tempo total de execução. Entretanto, mais de 40% do tempo total de execução correspondeu somente a chamadas da STL (*Standard Template Library*) para atualizar a matriz $\mathcal{I}$. Devido a melhora expressiva de desempenho que obtivemos com seqüências de bits implementadas sem STL para a manutenção de $\mathcal{O}$, $\mathcal{O}_s$ e $\mathcal{H}$ (seção 4.1), é possível que a porcentagem de tempo consumida na atualização de $\mathcal{I}$ também possa ser reduzida com código otimizado. O elevado número de objetos relatados como potencialmente visíveis também contribuiu para a degradação de tempo nesta etapa, pois mais objetos precisaram ser atualizados em $\mathcal{O}$ e $\mathcal{I}$.

3. O percurso do volume de visão, excluindo as etapas de extensão de oclusores e cálculo de oclusão, correspondeu a 20,4% do tempo total do algoritmo. Entretanto, somente a verificação do conteúdo das células dentro dessa etapa correspondeu a 11,3% do tempo total do algoritmo. Esse tempo poderia ser reduzido se mais células fossem classificadas como escondidas. Para isso, a determinação de visibilidade teria de ser mais precisa, mas a mesma resolução da grade regular poderia ser mantida.

4. O cálculo de oclusão, que compreende a rasterização dos volumes de oclusão, correspondeu a 13,3% do tempo total. Por outro lado, a construção dos mapas de bits e posterior vetorização correspondeu a apenas 2,4% do tempo total de execução. O código dessa etapa já havia sido otimizado. Como o cálculo de oclusão é um dos principais responsáveis pela determinação da precisão dos PVSs gerados, seria mesmo tolerável permitir um aumento do consumo de tempo desta etapa, desde que os resultados alcançados fossem mais exatos.

A porcentagem restante foi distribuída em frações menores do que 1% para funções diversas do algoritmo. As funções utilizadas para a manutenção de objetos dinâmicos escondidos (incluídos nos 55% de tempo da manutenção da grade) somaram cerca de 5% com a STL sendo utilizada para a manutenção de $\mathcal{T}$.

Capítulo 5

Conclusão e Trabalhos Futuros

Apresentamos uma arquitetura de descarte de visibilidade para cenas dinâmicas densamente oclusas baseada numa grade regular como subdivisão espacial. Além da eficiência para representar objetos dinâmicos visíveis e volumes limitantes temporais (TBVs), os benefícios do uso de grades regulares foram reforçados por métodos originais de percurso do volume de visão e cálculo de oclusão utilizando princípios de rasterização. De modo a acelerar a classificação de partes escondidas da cena, utilizamos um algoritmo de visibilidade que explorou a capacidade de fundir oclusores no espaço do objeto e construir oclusores virtuais através da utilização de regiões opacas da cena como oclusores. Além disso, o custo de determinação de visibilidade obtido é proporcional ao número de objetos visíveis – sejam eles estáticos ou dinâmicos – e não depende do número de polígonos nos modelos. Desse modo, nossa abordagem pode ser empregada em cenas de geometria refinada e mesmo em cenas não-poligonais.

A arquitetura foi implementada inicialmente para cenas 2D e depois estendida para o caso 3D. Essa estratégia de implementar o algoritmo inicialmente em 2D foi útil para facilitar o entendimento da nova arquitetura e estimar os pontos críticos de desempenho do caso 3D, uma vez que relativamente poucas modificações foram necessárias para essa transição.

Testes de desempenho foram conduzidos com o algoritmo para cenas 3D executado num computador pessoal equipado com *hardware* gráfico dedicado. De acordo com os resultados dos testes, a sobrecarga devido à manipulação de objetos dinâmicos escondidos através de volumes limitantes temporais foi desprezível (menor que 5ms para uma cena de 500 objetos dinâmicos) mesmo com o aumento da resolução da grade regular de 50x30x50 células para 100x60x100 células. Esse resultado foi possível devido às otimizações que propusemos para a discretização de TBVs na grade regular (ver Seção 3.4.2), em especial com o uso de TBVs "vazios" para reduzir o número de células acessadas e a utilização de apenas três matrizes bidimensionais de TBVs ao invés de uma matriz tridimensional. Por outro lado, o desempenho das etapas de percurso de células do volume de visão e cálculo de oclusão é excessivamente dependente da resolução da grade regular, de tal modo que os ganhos de tempo de renderização só foram significativos quando da utilização de uma grade de baixa resolução (50x30x50). Em tal situação, conseguimos acelerar

a visualização das cenas testadas numa média de dez vezes em comparação com a utilização exclusiva de *z-buffering* em *hardware*. Esse valor caiu para apenas quatro com uma grade de resolução duplicada em cada eixo coordenado e a precisão dos resultados obtidos apenas duplicou.

Enfim, bons desempenhos só foram obtidos para uma grade regular de baixa resolução, mesmo que dessa forma os resultados fossem mais imprecisos, *i.e.*, mesmo que o conjunto de objetos potencialmente visíveis relatasse um número elevado de objetos escondidos.

Entre as limitações do algoritmo que apresentamos, destacamos o alto grau de conservação na estimativa de objetos potencialmente visíveis. Nos resultados mais satisfatórios obtidos nos testes de desempenho (resultados em termos de aceleração da visualização da cena testada), o conjunto de objetos potencialmente visíveis foi cerca de seis vezes maior que a solução ideal. Embora esse valor tenha sido obtido com a utilização da grade regular de baixa resolução, o alto grau de conservação não parece estar apenas relacionado ao uso de grades regulares de baixa resolução, mas possivelmente ao fato de termos utilizado um algoritmo de descarte por oclusão que realizou a determinação e fusão de oclusores apenas no espaço do objeto. Esse argumento foi influenciado pela observação de que o algoritmo relatou vários objetos como potencialmente visíveis ainda que um único objeto visível estivesse encobrindo toda a imagem – algo que não aconteceria se fosse considerada a oclusão extraída da imagem. De fato, a utilização de oclusores apenas como células contidas totalmente na geometria dos objetos da cena tende a desperdiçar a oclusão causada por partes de objetos que estão apenas parcialmente contidas numa célula. Essa mesma oclusão não é desperdiçada no espaço da imagem, já que a cobertura da projeção dos objetos na imagem se soma *pixel* a *pixel* para formar oclusores maiores. Com base nessas observações, podemos afirmar que o baixo desempenho do algoritmo de visibilidade em grades regulares de alta resolução não foi devido a uma limitação da utilização de grades regulares como subdivisão espacial, mas em especial ao algoritmo de visibilidade que desenvolvemos para trabalhar nessa estrutura. Uma vez que o algoritmo descarta poucos objetos escondidos durante o percurso do volume de visão, um maior número de células da grade regular precisa ser percorrido. Ao mesmo tempo ocorrem mais detecções de oclusores que na verdade estão escondidos, o que acrescenta uma sobrecarga ainda maior de extensão de oclusores e cálculo de oclusão. A utilização de oclusores determinados apenas por células opacas e oclusas no espaço do objeto também limitou o algoritmo ao tratamento de cenas que pudessem ser discretizadas em volumes opacos. Nossa abordagem só utilizou como oclusores objetos fechados (poliédricos) e os casos de objetos compostos por polígonos desconexos não foram tratados (*e.g.*, como a oclusão causada por folhas de uma árvore na cena de uma floresta). Essa limitação não esteve relacionada ao uso de grades regulares.

Como trabalho futuro, pretendemos explorar a fusão de oclusores não só no espaço do objeto como também no espaço da imagem. Essa extensão deverá possibilitar que as principais limitações do algoritmo de visibilidade sejam corrigidas, por exemplo, realizando a determinação de células opacas (atualização da matriz de oclusores) através de testes de sobreposição da projeção das células na imagem ao invés dos testes geométricos de inclusão das células nos objetos. Por outro lado, sugerimos que as etapas de extensão de oclusores e cálculo de oclusão continuem sendo consideradas para realizar a determinação de células

escondidas no espaço do objeto, pois tais etapas podem classificar as células escondidas de maneira mais eficiente do que a obtida fazendo apenas testes de sobreposição da projeção de cada célula na imagem. Pode ser ainda possível diminuir o número de testes de visibilidade se considerarmos a utilização de grades regulares recursivas, isto é, uma hierarquia de grades regulares que na subdivisão se assemelhe a uma *octree*. O desempenho da atualização de objetos dinâmicos não deve ser prejudicado com essa estrutura de dados mais intrincada, uma vez que os objetos podem continuar sendo atualizados apenas no último nível da hierarquia que é a grade regular como apresentada neste trabalho. Com esses melhoramentos, acreditamos ser possível obter PVSs mais próximos de uma solução ideal sem abrir mão da utilização de grades regulares de baixa resolução. Tão importante quanto isto, a utilização do espaço da imagem para a determinação de células opacas na matriz de oclusores derrubaria a limitação do tratamento de oclusores a cenas de objetos fechados, abrindo espaço para a manipulação de cenas mais genéricas.

A extensão do algoritmo proposto para a utilização adicional de fusão de oclusores no espaço da imagem pode ser feita através de sua adaptação à técnicas como *z-buffer* hierárquico ou mapas de oclusão hierárquico. No manual da API dPVS de visibilidade, Aila e Miettinen [2] destacam os requisitos necessários para que um algoritmo de visibilidade possa ser adaptado a tais técnicas, entre eles a capacidade de realizar um percurso da subdivisão espacial em ordem aproximada de frente para trás a partir do observador e a capacidade de poder modificar a solução de visibilidade ao longo desse percurso em decorrência da adição de novos objetos dinâmicos. Nosso algoritmo preenche todos esses requisitos.

A área de pesquisa relacionada ao tratamento de visibilidade em cenas dinâmicas tem sido pouco explorada até este momento. Pretendemos preencher em parte esta lacuna através da realização de testes comparativos adicionais da nossa abordagem de grades regulares com subdivisões espaciais hierárquicas, tais como *octrees* e árvores BSP. Em especial, pretendemos comparar o tempo de manutenção de TBVs nessas diferentes estruturas de dados e o desempenho geral de cada uma num mesmo algoritmo de visibilidade (se possível) e em cenas comumente encontradas na prática. Também pretendemos implementar a arquitetura proposta numa API em OpenGL, nos mesmos moldes da dPVS, mas mantendo uma política de código aberto.

Referências Bibliográficas

[1] Narendra Ahuja and Charles Nash. Octree representations of moving objects. *Computer Vision, Graphics, and Image Processing*, 26:207–216, May 1984.

[2] Timo Aila and Ville Miettinen. *dPVS Reference Manual Version 2.10*. Hybrid Holding, Ltd., Helsinki, Finland, October 2001.

[3] John M. Airey, John H. Rohlf, and Jr. Frederick P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. In Rich Riesenfeld and Carlo Sèquin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 41–50, March 1990.

[4] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools*, 5(1), 2000.

[5] David Blythe and Tom McReynolds. Capping clipped solids with the stencil buffer. *SIGGRAPH '97 Course Notes (Programming with OpenGL: Advanced Rendering)*, 1997. http://www.sgi.com/software/opengl/advanced97/notes/notes.html .

[6] Edwin E. Catmull. Computer display of curved surfaces. In *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure*, pages 11–17, May 1975.

[7] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *SIGGRAPH '96*, pages 119–128, August 1996. http://www.cs.unc.edu/~geom/envelope.html.

[8] Daniel Cohen-Or, Yiorgos Chrysanthou, Cláudio T. Silva, and George Drettakis. Visibility, problems, techniques and applications. *SIGGRAPH 2000 Course Notes*, July 2000.

[9] Daniel Cohen-Or, Yiorgos Chrysanthou, Cláudio T. Silva, and Frédo Durand. A survey of visibility for walkthrough applications. *SIGGRAPH 2001 Course Notes*, August 2001.

[10] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998.

[11] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *ACM Symposium on Interactive 3D Graphics*, pages 83–90, April 1997.

[12] Frédo Durand. *3D Visibility: Analytical Study and Applications*. PhD thesis, Université Joseph Fourier, Grenoble, France, July 1999.

[13] Frédo Durand, George Drettakis, and Claude Puech. The 3d visibility complex. *ACM Transactions on Graphics*, 2002. http://graphics.lcs.mit.edu/~fredo/PUBLI/cplx3d.pdf.

[14] Frédo Durand, George Drettakis, Joelle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. In *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000.

[15] Carl Erikson and Dinesh Manocha. Gaps: General and automatic polygonal simplification. In *ACM Symposium on Interactive 3D Graphics*, pages 78–99, 1999.

[16] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Co., Reading, MA, 2nd edition, 1990.

[17] H. Fuchs, Z.M. Kedem, and B.F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH'80, Computer Graphics*, volume 14, pages 124–133, 1980.

[18] Andrew S. Glassner. Space sub-division for fast ray tracing. *IEEE Computer Graphics and Applications*, 4:15–22, October 1984.

[19] Andrew S. Glassner. *An Introduction to Ray Tracing*. Academic Press, San Diego, 1989.

[20] Ned Greene. Occlusion culling with optimized hierarchical z-buffering. *SIGGRAPH 2001 Course Notes*, August 2001.

[21] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical z-buffer visibility. In *Proceedings of SIGGRAPH '93*, pages 231–238, July 1993.

[22] Xianfeng Gu, Steven Gortler, Hugues Hoppe, Leonard McMillan, B. Brown, and A. Stone. Silhouette mapping. Technical Report TR-1-99, Department of Computer Science, Harvard University, March 1999.

[23] Richard Huddy. What comes after 4? Slides presentation. nVidia Corporation. *Game Developers Conference 2002*, March 2002.

[24] Tom Hudson, Dinesh Manocha, Jonathan Cohen, Ming C. Lin, Kenneth E. Hoff III, and Hansong Zhang. Accelerated occlusion culling using shadow frusta. In *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, pages 1–9, July 1997.

[25] Subodh Kumar, Dinesh Manocha, Bill Garrett, and Ming Lin. Hierarchical back-face computation. In *7th Eurographics Workshop on Rendering*, pages 235–244, 1996.

[26] Don Libes. Modeling dynamic surfaces with octrees. *Computers & Graphics*, 15(3), 1991. Pergamon Press, New York, NY.

[27] Erik Lindholm, Mark J. Kilgard, and Henry Moreton. A user programmable vertex engine. In *Proceedings of SIGGRAPH 2001*, August 2001.

[28] David Luebke and Chris Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In Pat Hanrahan and Jim Winget, editors, *1995 Symposyum on Interactive 3D Graphics*, pages 105–106, April 1995. ISBN 0-89791-736-7.

[29] Michael Meissner, Dirk Bartz, Tobias Hüttner, Gordon Müller, and Jens Einighammer. Generation of subdivision hierarchies for efficient occlusion culling of large polygonal models. Technical Report TUBSCG-1999-6-1, Institute of Computer Graphics, Technical University of Braunschweig, 1999.

[30] Tomas Möller and Eric Haines. *Real-Time Rendering*. A.K. Peters Ltd., 2^{nd} edition, 2002.

[31] S. Morein. ATI Radeon Hyper-Z Technology. In *Hot3D Proceedings. Graphics Hardware Workshop*, 2000.

[32] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, NY, 1987.

[33] Thomas Pabst. High-tech and vertex juggling - nVidia's new GeForce 3 GPU. *Toms Hardware Guide*, February 2001.

[34] Harry Plantinga and Charles R. Dyer. Visibility, occlusion and the aspect graph. *International Journal of Computer Vision*, 5(2):137–160, 1990.

[35] Harald Riegler. Point-visibility in computer games. *Computer Graphics Seminar. Computer Game Technology*, June 2001.

[36] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François Sillion. Conservative volumetric visibility with occluder fusion. In *Proceedings of SIGGRAPH 2000*, pages 229–238, 2000.

[37] Mel Slater and Yiorgos Chrysanthou. View volume culling using a probabilistic caching scheme. In M. Bergamasco S. Wilbur, editor, *Proceedings of Framework for Immersive Virtual Environments FIVE*, December 1996.

[38] Andrew Smith, Yoshifumi Kitamura, and Fumio Kishino. Efficient algorithms for octree motion. In *IAPR Workshop on Machine Vision Applications*, pages 172–177, 1994.

[39] Oded Sudarsky. *Dynamic Scene Occlusion Culling*. PhD thesis, Technion–Israel Institute of Technology, January 1998.

[40] Oded Sudarsky and Craig Gostsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. In *Proceedings of Eurographics '96*, volume 15, Poitiers, France, August 1996.

[41] Oded Sudarsky and Craig Gotsman. Dynamic scene occlusion culling. *IEEE Transactions on Visualization & Computer Graphics*, 5(1):217–223, 1999.

[42] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys*, 6(1):1–55, 1974.

[43] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Proceedings of SIGGRAPH '91*, volume 25, pages 61–69, July 1991.

[44] Juyang Weng and Narendra Ahuja. Octrees of objects in arbitrary motion: representation and efficiency. *Computer Vision, Graphics, and Image Processing*, 39(2):167–185, 1987.

[45] Peter Wonka and Dieter Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. In *Proceedings of Eurographics '99*, August 1999.

[46] Roni Yagel and William Ray. Visibility computation for efficient walkthrough complex enclosed environments. *PRESENCE*, 5(1):1–16, 1996.

[47] Hansong Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. PhD thesis, Department of Computer Science, Uiversity of North Carolina at Chapel Hill, 1998.

[48] Hansong Zhang and Kenneth E. Hoff III. Fast backface culling using normal masks. In *1997 Symposium on Interactive 3D Graphics*, pages 103–106, April 1997.

[49] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH '97*, volume 31, pages 77–88, August 1997.