Capítulo 11

Algoritmos de Visibilidade

Para aproximar a imagem gerada sinteticamente das percebidas pela visão humana, deve-se ainda levar em consideração a **opacidade** dos objetos, ou seja a percentagem dos raios luminosos que eles são capazes de bloquear. Objetos de opacidade igual a 1.0, ou seja, objetos que bloqueiam totalmente os raios luminosos, colocados na frente de outros objetos, bloqueiam a visão destes outros objetos. Assim, na geração de uma imagem sintética, objetos que são bloqueados pelos outros objetos opacos ao longo da direção da visão do observador (ou direção de projeção *dop*) devem ser removidos. O processo de remoção de partes não visíveis é conhecido em Computação Gráfica por **algoritmo de visibilidade**.

Observação 11.1 *Vale comentar que os algoritmos de visibilidade se aplicam em outros contextos. Por exemplo, eles podem ser utilizados no contexto de iluminação para determinar as partes que não são visíveis pelas fontes luminosas e, portanto, que devem ficar na sombra, como vimos no Capítulo 6.*

O problema de visibilidade pode ser reduzido em dois passos:

1. determinação de objetos que intersectam uma linha de visão, e
2. ordenação das interseções ao longo do semi-raio de projeção e selecionar o objeto que tem a interseção mais próxima do observador.

Embora seja simples a colocação do problema, a implementação do primeiro passo pode envolver operações de alta complexidade. Pesquisas tem sido conduzidas no sentido de reduzir a complexidade temporal destas operações ou de implementar os algoritmos diretamente em *hardware* ou *firmware*. Em decorrência disso, pode-se encontrar uma grande diversidade de

algoritmos de visibilidade na literatura, que essencialmente podem ser classificados em três grandes grupos: **técnicas baseadas em espaço de imagem**, **técnicas baseadas em espaço de objeto** e **técnicas mistas**. A resolução da determinação de visibilidade das técnicas baseadas no espaço de imagem é em *pixels* – apropriadas para dispositivos de saída *raster* – e a das técnicas baseadas no espaços de objeto é em valores reais – adequada para dispositivos de saída *vetorial*.

Antes de apresentarmos alguns algoritmos de visibilidade mais conhecidos, vamos apresentar de forma geral na Seção 11.1 alguns recursos/propriedades amplamente explorados para aumentar a eficiência no tratamento da visibilidade.

11.1 Recursos Complementares

Como já mencionado, a essência dos algoritmos de visibilidade é muito simples: remover as partes que não são visíveis a partir de um ponto. Para reduzir o tempo de execução destes algoritmos, várias propriedades geométricas e características dos dispositivos de saída tem sido utilizadas para reduzir o tamanho dos objetos sobre os quais são aplicadas efetivamente as operações computacionalmente caras.

Uma propriedade que os algoritmos de visibilidade podem explorar é a **coerência** dos atributos usando o fato de que sempre existe um escopo dentro do qual a variação dos atributos ocorrem de forma bastante suave. Neste caso, ao invés de determinar os atributos "a partir do nada" em cada ponto da cena, podemos utilizar técnicas recorrentes para determiná-los. Os procedimentos recorrentes, como já vimos no Capítulo 8, tem usualmente um desempenho melhor.

Sutherland et al. identificaram uma série de coerências que podem ser exploradas nos algoritmos de visibilidade:

1. Coerência de objetos: dois objetos disjuntos não podem apresentar interseções a nível de faces, arestas ou pontos.
2. Coerência de face: as propriedades gráficas de uma face variam de forma suave. Transições bruscas são normalmente interpretadas como fronteira de duas faces.
3. Coerência de arestas: a transição da parte visível para a parte invisível de uma aresta, ou vice-versa, só pode ocorrer nos pontos de interseção.

4. Coerência geométrica: o segmento de interseção entre duas faces planares pode ser determinado pelos dois pontos de interseção.

5. Coerência das linhas varridas: a variação entre duas linhas de varredura é pequena.

6. Coerência na área de cobertura: um conjunto de *pixels* adjacentes é usualmente coberto por uma face.

7. Coerência na profundidade: os valores de profundidade das amostras adjacentes de uma mesma face variam pouco e os valores de profundidade das amostras de faces distintas usualmente são valores distintos.

8. Coerência nos quadros: dois quadros consecutivos de uma animação diferem muito pouco um do outro.

Baseado na coerência de objetos, é comum aplicar a técnica de **caixa limitante** (*bounding box*) para separar dois objetos trivialmente disjuntos. Há várias propostas para determinação de uma caixa limitante justa para um objeto, porém a mais difundida é a menor caixa, com os planos paralelos aos eixos do sistema de referência, capaz de envolver totalmente o objeto.

(Ver Fig. 15.13 – 15.16 do livro-texto de Foley.)

A coerência de profundidade é muito difundida na determinação dos objetos visíveis pelos algoritmos com resolução a nível de *pixels* (de imagem). Nestes algoritmos, o ponto visível é determinado por simples comparações entre os valores de profundidade dos pontos associados a um *pixel*. Na maioria das vezes, estas comparações são válidas porque os valores são distintos. Para determinar estes valores de profundidade, aplica-se a transformação perspectiva/paralela, que vimos no Capítulo 5, em todos os pontos da cena para obter as suas coordenadas no volume de visualização de lados paralelos e normalizado. Particularmente, a coordenada z corresponde à profundidade do ponto em relação ao observador.

Exercício 11.1 *Reveja a matriz de transformação do sistema de referência do mundo (WC) para o sistema de referência normalizado canônico (NC)*

1. *para projeção paralela e*

2. *para projeção perspectiva.*

Outro pre-processamento muito utilizado no processo de remoção de faces não visíveis é conhecido como *back-face cull*. Esta técnica se baseia no fato de que faces com vetores normais com direções opostas à da direção de projeção não podem ser visíveis.

(Ver Fig. 15.17 do livro-texto de Foley.)

Exercício 11.2 *Dados a posição de um observador* $\mathcal{P} = (1.0, 0.0, 0.0)$ *e o ponto de interesse* $\mathcal{I} = (0.0, -1.0, -1.0)$. *Quais das faces com os seguintes vetores normais serão removidas pelo algoritmo back-face cull: (1.0,0.0,0.0), (2.0, 3.0,-1.0), (-2.0, -1.0, 1.0) e (0.0,1.0,0.0)?*

11.2 Algoritmos de Visibilidade de Linhas

Historicamente, estes algoritmos apareceram antes dos algoritmos de visibilidade de superfícies para remover segmentos que não devem aparecer na saída de um dispositivo vetorial.

(Ver Fig. 15.20 do livro-texto de Foley.)

O primeiro algoritmo de visibilidade de linhas foi desenvolvido pelo Roberts que reduziu o problema de visibilidade num problema de programação linear, ao comparar cada segmento P_1P_2 em relação a um volume convexo definido pelas equações das suas faces [V].

A partir de um ponto $P(t) = v$ do segmento P_1P_2 dado por

$$P(t) = P_1 + (P_2 - P_1)t \leftrightarrow v = s + \vec{d}t$$

definem-se segmentos na direção de projeção $\vec{g}$

$$Q(\alpha, t) = u = v + \vec{g}\alpha = s + \vec{d}t + \vec{g}\alpha$$

Como os pontos interiores $P = (x, y, z)$ de um poliedro convexo de m faces planas devem satisfazer a inequação

$$n_{x_i}x + n_{y_i}y + n_{z_i}z + d_i < 0$$

para qualquer uma das suas faces i com o vetor normal $(n_{x_i}, n_{y_i}, n_{z_i})$ orientado para o lado externo do poliedro, os pontos de $Q(\alpha, t)$ que satisfazem

$$Q(\alpha,t) \cdot \begin{bmatrix} n_{x_1} & n_{x_2} & n_{x_3} & \cdots & n_{x_m} \\ n_{y_1} & n_{y_2} & n_{y_3} & \cdots & n_{y_m} \\ n_{z_1} & n_{z_2} & n_{z_3} & \cdots & n_{z_m} \\ d_1 & d_2 & d_3 & \cdots & d_m \end{bmatrix} < 0 \qquad (11.1)$$

estão no interior do poliedro. A solução deste sistema de inequações pode ser obtida com uso de técnicas de programação linear. Se a solução for

$0 \leq t \leq 1$ e $\alpha > 0$, então $P(t)$ é parcialmente ou não é visível, porque o poliedro está entre o observador e o segmento P_1P_2.

Posteriormente, outros algoritmos mais simples e genéricos foram propostos, como o algoritmo de Appel que explora a coerência de aresta e reduz o problema de visibilidade em interseção entre arestas e faces "frontais". Para reduzir o número de operações de interseção, várias estruturas de dados dedicadas foram sugeridas para explorar melhor a coerência de objetos.

(Ver Fig. 15.19 do livro-texto de Foley.)

Exercício 11.3 *Considere um cubo unitário centrado na origem e um segmento definido pelos pontos* $P_1 = (1,0,2,1)$ *e* $P_2 = (-1,0,-2,1)$. *Considere ainda que a direção dos raios projetored é* $(0,0,-1,0)$. *Utilize o algoritmo de Roberts para determinar a visibilidade do segmento.*

Exercício 11.4 *Podemos obter o efeito de um algoritmo de visiblidade de linhas com uso de algoritmos de visibilidade de superfícies. A idéia consiste em "pintar" os polígonos com a cor do fundo e traçar as linhas e as arestas com uma outra cor distinta. Devido à discretização, as linhas podem aparecer "interrompidas". Como se pode evitar isso?*

11.3 Algoritmos de Visibilidade de Superfícies

O algoritmo de visibilidade de superfícies mais difundido nas placas gráficas é o algoritmo de *z-buffer*, que consiste armazenar, além dos atributos gráficos (por exemplo, a cor) do ponto projetado no *pixel*, o valor da sua profundidade z. Com isso, ao rasterizar um polígono, verifica-se para cada *pixel* se o valor de profundidade do ponto é menor que o valor existente no z- buffer para então substituir o conteúdo da posição do *frame buffer* correspondente pelos atributos gráficos do novo ponto.

(Ver Fig. 15.22 do livro-texto de Foley.)

É possível ainda explorar a coerência de profundidade para determinar recorrentemente os valores de profundidade quando se trata de faces planas.

Exercício 11.5 *Seja* z *o valor de profundidade do ponto* (x,y) *de um polígono cujo plano suporte é dado pela expressão* $Ax+By+Cz+D=0$. *Determine a expressão recorrente dos valores de* z *deste polígono para* $\Delta x = 1$ *e para* $\Delta y = 1$.

A grande vantagem do algoritmo de *z-buffer* é que, a custo de uso de uma memória maior, consegue-se evitar determinação de interseção e ordenação de interseções ao longo do raio de projeção.

Exercício 11.6 *Dados três polígonos:* {(3,6,1),(2,1,0),(7,4,4)}, {(2,4,3), (2,2,1),(4,2,1),(4,4,3)} *e* {(1,6,2),(1,1,2),(3,1,2),(3,6,2)}. *Utilize o algoritmo de z-buffer para determinar a visibilidade destes polígonos em relação a raios projetores* $(0,0,1,0)$.

Um outro algoritmo muito conhecido é o **algoritmo de pintor** ou **algoritmo de lista de prioridades**. Este algoritmo pré-processa os polígonos da cena, ordenando-os de forma crescente em termos da sua distância em relação ao observador. Esta ordem é utilizada pelo algoritmo de rasterização para estabelecer a prioridade de rasterização dos polígonos – o que tiver mais distante é rasterizado primeiro e o mais próximo por último de forma similar a um pintor pintando o seu quadro. Quando tiver ambiguidade na ordenação de um polígono, o polígono é subdividido até que todas as ambiguidades sejam resolvidas.

(Ver Fig. 15.24–15.27 do livro-texto de Foley.)

Exercício 11.7 *Ordene os três polígonos do Exercício 11.6 em relação a suas distâncias em relação ao observador.*

A etapa mais custosa do algoritmo de pintor é o pré-processamento – a ordenação. Para reaproveitar os resultados deste pre-processamento em outros contextos, foram propostas várias estruturas de dados para armazenar os polígonos de forma a permitir a inferência da ordenação espacial destes polígonos em tempo real. A estrutura mais conhecida é a **árvore de partição espacial binária** (*binary space-partitioning tree*). Esta árvore organiza os polígonos de tal forma que, enquanto a cena não for alterada, consegue-se obter a ordenação correta dos polígonos para qualquer posição do observador e permitir uma renderização em tempo real (ver `http://symbolcraft.com/graphics/bsp/index.html`)

(Ver Fig. 15.28 e 15.31 do livro-texto de Foley.)

Exercício 11.8 *Crie uma árvore de partição binária para a cena composta de três polígonos do Exercício 11.6. A partir da árvore, infere a ordem dos polígonos para a direção de projeção* $(-1,0,0,0)$ *e a direção* $(0,1,0,0)$.