Capítulo 10

Algoritmos de Visibilidade

Conforme já comentamos na seção 5.1, o sistema de visão humana faz uso de vários recursos para melhorar a percepção de profundidade. Um destes recursos é a **oclusão** de objetos distantes por um objeto opaco $\mathcal{A}$ que os antecede ao longo de uma linha de visão, pois os objetos opacos bloqueiam totalmente os raios luminosos, impedindo que os raios refletidos pelos outros alcancem os olhos. Dizemos então que $\mathcal{A}$ é **visível em relação ao observador**.

Um outro recurso é o sombreamento. Conforme vimos no capítulo 8, a luminância/brilhância que precebemos de um objeto visível depende dos raios que incidem sobre ele. Portanto, é também conveniente distinguir em Sistemas de Informações Gráficas a **visibilidade dos objetos em relação às fontes luminosas**. Os objetos visíveis são mais "claros" e os escondidos são usualmente considerados como estarem "em sombra".

O processo de identificação de partes visíveis, sejam elas em relação a um observador ou a uma fonte luminosa, é denominado **algoritmo de visibilidade**.

O problema de visibilidade pode ser reduzido em três passos:

1. determinação de objetos que intersectam uma linha de visão,
2. ordenação das interseções ao longo do semi-raio de projeção $r(t)$ que parte do observador P_O e passa pelo ponto P_p do plano de projeção, $r(t) = P_0 + t(P_p - P_O)$, e
3. 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 determinação de visibilidade por técnicas baseadas no espaço de imagem é em termos de *pixels* – apropriada para dispositivos de saída *raster* – e a por técnicas baseadas no espaço de objeto é em termos de pontos – adequada para dispositivos de saída *vetorial*.

Antes de apresentarmos alguns algoritmos de visibilidade conhecidos, vamos apresentar de forma geral na seção 10.1 técnicas amplamente utilizadas para aumentar a eficiência no tratamento da visibilidade.

10.1 Técnicas Complementares

Como já mencionado, a essência dos algoritmos de visibilidade é muito simples: remover as partes não visíveis por um ponto tanto para os raios projetores paralelos como para os perspectivos. Entretanto, a sua implementação envolve operações complexas sob o ponto de vista de número de operações. 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 9, 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.

Exercício 10.1 *Quais tipos de coerência que o algoritmo de Bresenham e o algoritmo de varredura por linha utilizam para aumentar a sua eficiência?*

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.)

Exercício 10.2 *Dadas duas facetas traingulares, cujas coordenadas dos vértices são dadas no espaço de dispositivo*

1. $(0, 0, 0.4)$, $(0, 0, 0.4)$ *e* $(3, 8, 0.2)$.

2. $(2, 2, 0.4)$, $(8, 1, 0.5)$ *e* $(5, 7, 0.8)$.

Determine a caixa limitante para cada faceta. É possível ordenar as duas facetas em relação à coordenada z pelas suas caixas limitantes?

A coerência de profundidade é amplamente utilizada na determinação dos objetos visíveis pelos algoritmos com resolução 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.

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.)

10.2 Algoritmo de Visibilidade de Pontos

Essencialmente, consiste em sequenciar os pontos ao longo de um mesmo raio projetivo em termos das suas distâncias em relação ao observador e selecionar o ponto mais próximo do observador. O sequenciamente é trivial se trabalharmos no espaço canônico normalizado com os lados paralelos entre si.

Exercício 10.3 *Dados os pontos* $(4, 12, -16, 1)$, $(2, 6, -8, 1)$ *e* $(1, 3, -2, 1)$ *e o ponto do observador* $(0, 0, 2, 1)$ *no espaço de universo. Qual ponto está na frente dos outros 2 em relação ao observador? Supondo que o plano de projeção tenha VPN=(0,0,1,0) e VRP=(0,0,0,1), qual deve ser a coordenada z dos pontos no espaço normalizado em relação a este plano? Podemos ordenar crescentemente as distâncias dos pontos em relação ao observador pela ordem crescente da coordenada z dos pontos no espaço normalizado?*

10.3 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 face i do poliedro, 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

$$Q(\alpha, t) \cdot \begin{bmatrix} n_{x_1} & n_{x_2} & n_{x_3} & \cdots & n_{x_m} \\ n_{y_1} & n_{y_2} & n_{y_3} & \cdots & n_{y_m} \\ n_{z_1} & n_{z_2} & n_{z_3} & \cdots & n_{z_m} \\ d_1 & d_2 & d_3 & \cdots & d_m \end{bmatrix} < 0 \qquad (10.1)$$

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 a coerência de objetos.

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

Observação 10.1 *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.*

10.4 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 (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.)

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

É possível ainda explorar a coerência de profundidade para determinar recorrentemente os valores de profundidade. Porisso, o algorimto de z-*buffer* é muito utilizado em conjunto com o algoritmo de varredura para determinar corretamente a faceta que deve ser visível em cada *pixel*.

Considere que a equação do plano que contém uma faceta poligonal seja

$$n_x x + n_y y + n_z z + d = 0.$$

Segue-se que a coordenada z de cada amostra (x_k, y_k) é expressa por

$$z_k = \frac{-d - n_x x_k - n_y y_k}{n_z}. \tag{10.2}$$

A coordenada z do seu *pixel* vizinho $(x_k + 1, y_k)$ pode ser obtida de forma recorrente a partir de z_k

$$z_{k+1} = \frac{-d - n_x(x_k + 1) - n_y y_k}{n_z} = z_k - \frac{n_x}{n_z}.$$

E como podemos obter a coordenada z ao passar de uma linha de varredura para a outra? Basta substituir as coordenadas da amosta $(x, y_k + 1)$ na Eq. 10.2.

Exercício 10.4 *Rasterize, com uso do algoritmo de varredura por linha e a tonalização de Gouraud, os dois polígonos especificados pelas coordenadas dos seus vértices no espaço do dispositivo*

1. $(0, 0, 0.4)$, $(6, 0, 0.4)$ *e* $(3, 8, 0.8)$
2. $(2, 2, 0.6)$, $(8, 1, 0.2)$ *e* $(5, 7, 0.2)$.

As cores nos vértices são, respectivamente, $(1, 0, 0)$, $(0, 1, 0)$ *e* $(0, 0, 1)$ *para ambas as facetas.*

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 relação à 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.)

A etapa mais custosa do algoritmo de pintor é o pré-processamento de ordenação. Para reaproveitar os resultados deste pre-processamento em outros contextos, foram propostas várias estruturas de dados para armazenar os polígonos e inferir a sua ordenação espacial. A estrutura mais conhecida é a **árvore de partição espacial binária** (*binary space-partitioning tree*), que apresentamos na seção 4.3.

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