Algoritmos de eliminação de linhas/superfícies escondidas

Luíza Gomes Accarini

ICMC-USP

28 de outubro de 2018

Conteúdo

1 Introdução

2 Back-Face Detection

3 z-Buffer Method

4 Bibliografia

Introdução

Uma consideração importante na geração de elementos gráficos realistas é o problema de determinar o que é visível dentro de uma cena a partir de uma posição de visualização escolhida.

Várias são as abordagens possíveis para isso e numerosos são os algoritmos desenvolvidos para a identificação eficiente de objetos visíveis. Enquanto alguns métodos requerem mais memória, outros envolvem mais tempo de processamento ou ainda aplicam-se apenas a alguns tipos específicos de objetos, de modo que a escolha do método a ser utilizado depende da finalidade.

Podemos classificar estes algoritmos de acordo com a maneira com que eles lidam com as definições dos objetos ou com as projeções das imagens na cena. As abordagens são conhecidas como:

- *Object-space methods*
  - Comparam objetos e partes de objetos entre si para determinar quais superfícies, como um todo, devem ser consideradas como visíveis.
- *Image-space methods*
  - A visibilidade é decidida ponto a ponto em cada posição de pixel no plano de projeção.

A maioria dos algoritmos utiliza a segunda abordagem, embora a primeira também possa ser utilizada em alguns casos mais específicos, como os algoritmos de exibição de linhas, por exemplo.

Back-Face Detection

Consiste de um simples e rápido *object-space method* para localizar as **faces traseiras** de um poliedro.

- Seja $P = (x', y', z') \in \mathbb{R}^3$ um ponto, e seja também $Ax + By + Cz + D = 0$ a equação explícita do plano definido por uma dada face de um polígono. Sabemos que P reside atrás desta face se vale a relação $Ax' + By' + Cz' + D < 0$.

Assim, quando esta posição está ao longo da linha de visão da superfície, devemos estar olhando para a parte de trás do polígono em questão. Portanto, vejamos como podemos utilizar a posição de visualização para descobrir quais são as faces traseiras:

- Se $\vec{V}$ é um vetor na direção e no sentido de visualização do observador, então a face observada é **traseira** se $\vec{V} \cdot \vec{N} \geq 0$, onde $\vec{N}$ é o vetor normal (externo) à face.

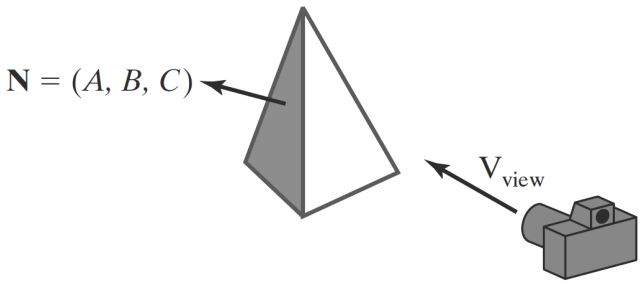

Figura: O vetor $\vec{N}$ normal à face e o vetor $\vec{V}$ de direção de visualização (Retirada de [1]).

Back-Face Detection

Além disso, em um sistema de coordenadas como mostrado na figura a seguir, com direção de visualização ao longo do eixo $-z$, uma face de um polígono é **traseira** se a terceira componente de $\vec{N}$, digamos C, é tal que $C < 0$.

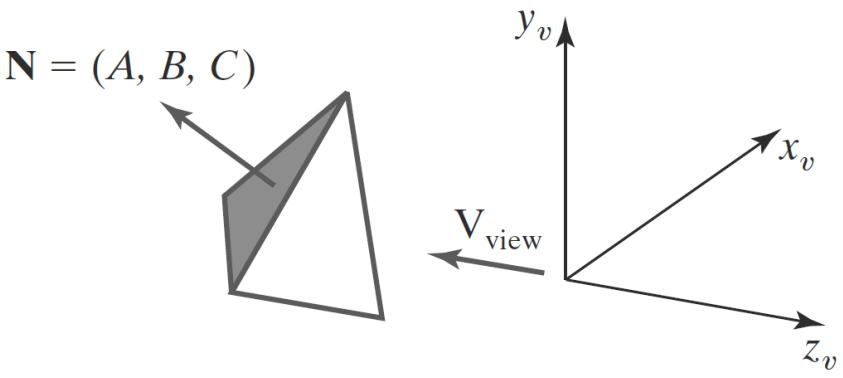

Figura: Retirada de [1].

Adicionalmente, não podemos ver nenhuma face tal que a terceira componente de seu vetor normal seja nula. Portanto, neste caso, podemos rotular uma face como sendo **traseira** se $C \leq 0$.

Back-Face Detection

Observações

- Observação 1. Para poliedros convexos, como a pirâmide da figura do slide anterior, o *back-face detection* identifica todas as superfícies ocultas na cena.
- Observação 2. Para outros objetos, como poliedros côncavos, por exemplo, mais testes devem ser realizados para determinar se existem faces adicionais que são total ou parcialmente obscurecidos por outras faces. Como consequência, esperando-se que uma cena geral contenha objetos sobrepostos ao longo da linha de visão, a mesma ideia se aplica.

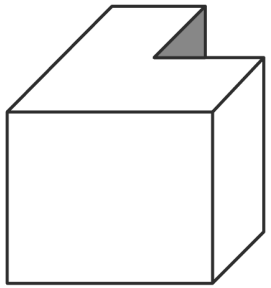

Figura: Poliedro côncavo (Retirado de [1]).

z-Buffer Method

Consiste de um *image-space method* comumente utilizado, o qual compara os valores de profundidade de cada superfície pertencente à cena para cada posição de pixel no plano de projeção.

- Cada superfície de uma cena é processada separadamente, com uma posição de pixel por vez.
- O algoritmo é normalmente aplicado a cenas contendo apenas faces poligonais, porque os valores de profundidade podem ser calculados muito rapidamente e o método é fácil de implementar.
- O método é assim chamado pois a profundidade de um objeto é geralmente medida através do eixo z do sistema de visualização.
- Porém, em vez de usar coordenadas reais dentro da cena, este algoritmo calcula a distância do plano de visão ao longo do eixo z.

z-Buffer Method

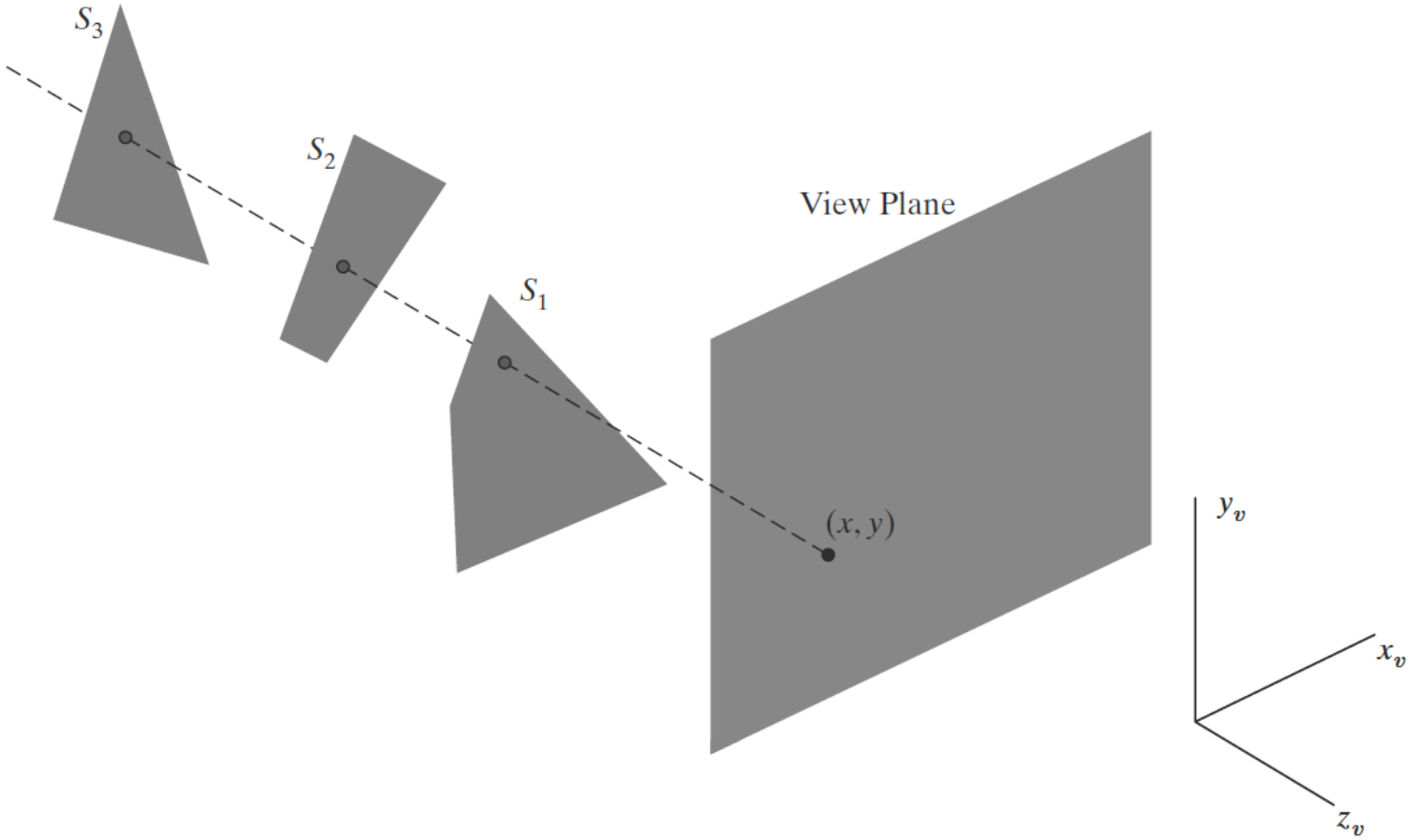

Figura: Três superfícies com distâncias variando distâncias ao longo da linha de projeção ortográfica a partir de uma posição (x, y) em um plano de visão (Retirada de [1]).

z-Buffer Method

- As superfícies representadas na figura podem ser processadas em qualquer ordem.
- Como cada superfície é processada independentemente disso, sua profundidade em relação ao plano de visão é comparada com as profundidades das superfícies processadas anteriormente.
- Deste modo, se uma superfície é **mais próxima** do que qualquer outra que foi previamente processada, sua cor é calculada e salva, junto com sua profundidade.

Veja que as superfícies visíveis em uma cena são representadas pelo conjunto das cores das superfícies que foram salvas depois do processamento completo da mesma.

Apesar de implícito no nome do método, duas áreas de buffer são necessárias:

- Um **buffer de profundidade** é usado para armazenar valores de profundidade para cada posição (x, y) conforme as superfícies são processadas.
- O **buffer de quadros** (frame buffer) armazena os valores de cores de superfície para cada posição de pixel.

z-Buffer Method

- A implementação do *z-buffer* é feita utilizando coordenadas normalizadas.

De maneira geral, podemos descrevê-la da seguinte forma:

1. Inicialmente, todas as posições no buffer de profundidade são definidas como 1.0 (profundidade máxima) e o frame buffer (buffer de atualização) é inicializado para a cor do plano de fundo.
2. Cada superfície listada nas tabelas de polígonos é então processada, com uma linha de varredura de cada vez, calculando o valor de profundidade em cada posição (x, y) de pixel.
3. Esta profundidade calculada é comparada com o valor armazenado anteriormente no buffer de profundidade para esta posição de pixel.
4. Se a profundidade calculada for menor do que o valor armazenado no buffer de profundidade, o novo valor valor de profundidade será armazenado.
5. Em seguida, a cor da superfície nessa posição é computada e colocada na localização do pixel correspondente no frame buffer.

z-Buffer Method

Dados os valores de profundidade para as posições dos vértices de qualquer polígono em uma cena, podemos calcular a profundidade em qualquer outro ponto do plano que contenha o polígono. Na posição de superfície (x, y), a profundidade é calculada a partir da equação do plano

$$z = \frac{-Ax - By - D}{C}$$

Para qualquer linha de varredura (scan line), as posições x horizontais adjacentes na linha diferem em ± 1, e o mesmo ocorre com as posições verticais adjacentes. Se a profundidade da posição (x, y) foi determinada como z, então a profundidade z' da próxima posição $(x + 1, y)$ ao longo da linha de varredura é obtida através de

$$z' = \frac{-A(x+1) - By - D}{C}$$

ou

$$z' = z - \frac{A}{C}$$

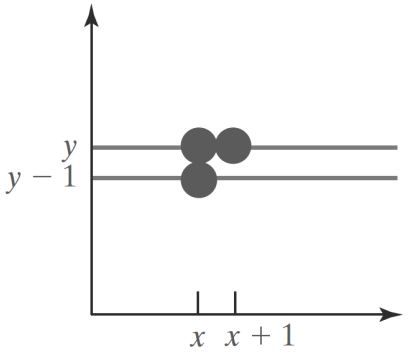

Figura: Retirada de [1].

z-Buffer Method

A razão $-A/C$ é constante para cada superfície, de modo que os valores sucessivos de profundidade através de uma linha de varredura são obtidos de valores anteriores com uma única adição.

Processando posições de pixel da esquerda para a direita em cada linha de varredura, começamos calculando a profundidade na borda esquerda de um polígono que cruza esta linha. Para cada posição sucessiva, calcula-se o valor da profundidade utilizando a equação $z' = z - (A/C)$.

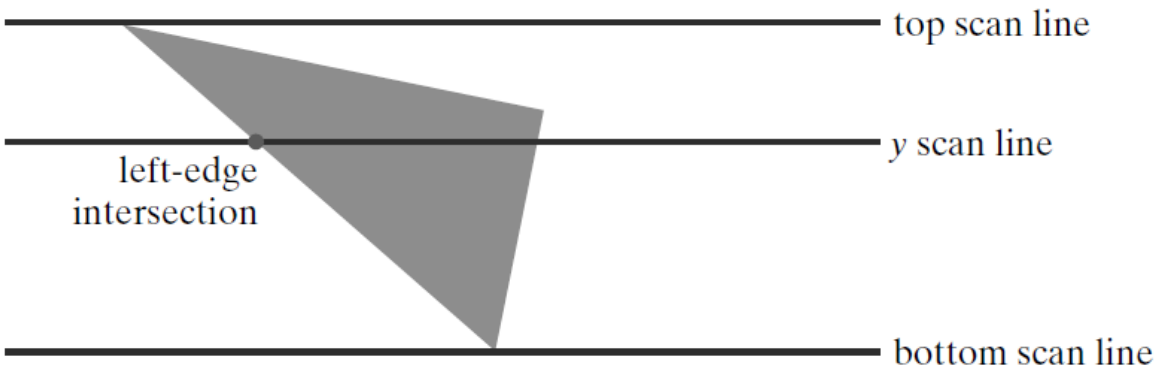

Figura: Retirada de [1].

z-Buffer Method

Podemos implementar o algoritmo z-buffer iniciando em um vértice superior do polígono. Com isso, é possível calcular recursivamente os valores da coordenada x abaixo da borda esquerda do polígono. O valor x para a posição inicial em cada linha de varredura pode ser calculado desde o início com

$$x' = x - \frac{1}{m}$$

onde m é a inclinação da borda.

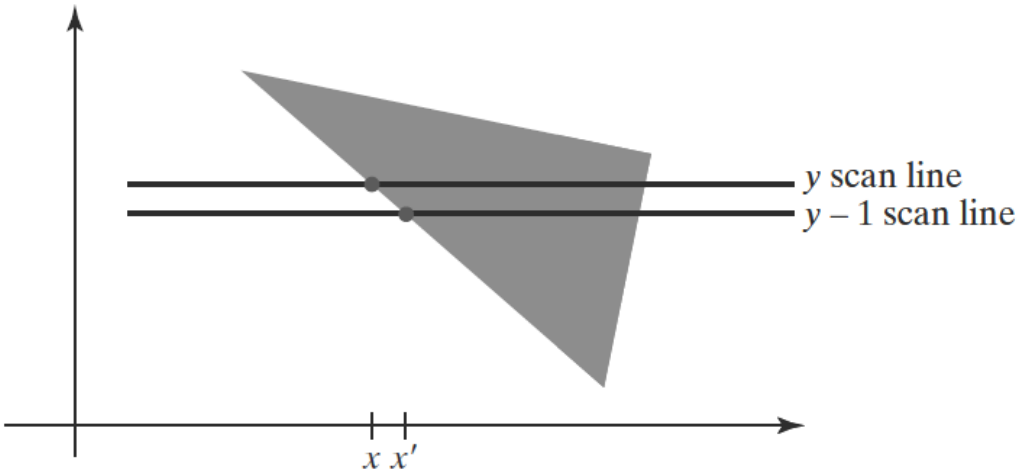

Figura: Retirada de [1].

z-Buffer Method

Observações

Os valores de profundidade abaixo desta borda são obtidos recursivamente como

$$z' = z + \frac{(A/m) + B}{C}$$

Se, por outro lado, estamos processando de cima para baixo uma borda vertical, a inclinação é infinita e os cálculos recursivos reduzem-se a

$$z' = z + \frac{B}{C}$$

- Observação 1. Uma complicação desta abordagem é que, enquanto as posições de pixel estão em coordenadas inteiras (x, y), o ponto real de intersecção de uma linha de varredura com a borda de um polígono pode não ser inteiro. Como resultado, pode ser necessário ajustar o ponto de intersecção arredondando a parte fracionária para cima ou para baixo.
- Observação 2. Uma abordagem alternativa é usar um algoritmo do tipo *Bresenham* para determinar os valores x iniciais ao longo das arestas para cada linha de varredura.

z-Buffer Method

Observações

- Observação 3. O *algoritmo de Bresenham* é um algoritmo criado para desenho de linhas em dispositivos matriciais que permite determinar quais os pontos numa matriz de base quadriculada que devem ser destacados para atender o grau de inclinação de um ângulo.

Por exemplo:

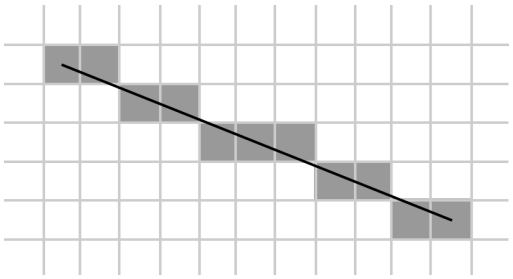

Figura: Algoritmo de Bresenham. Disponível em: <https://www.wikiwand.com/pt/Algoritmo_de_Bresenham>. Acesso em: 27 out. 2018.

- Observação 4. O método (z-buffer) pode também ser aplicado a superfícies curvas determinando os valores de profundidade e cor em cada ponto de projeção da superfície.

z-Buffer Method

Observações

- Observação 5. Para superfícies poligonais, o método z-buffer é fácil de ser implementado e não requer ordenação das superfícies em uma cena. Isto, porém, requer a disponibilidade de um segundo buffer, além do buffer de atualização (refresh buffer). Um sistema com uma resolução de 1280×1024, por exemplo, exigiria mais de $1,3$ milhões de posições no frame de profundidade, com cada posição contendo bits suficientes para representar o número de incrementos de profundidade necessários.
- Observação 6. Uma maneira de reduzir os resquícios de armazenamento é processar uma seção de cena por vez, usando um buffer de profundidade menor. Depois que cada seção de exibição é processada, o buffer é reutilizado na próxima seção.

z-Buffer Method

Um exemplo de mapa de profundidade gerado:

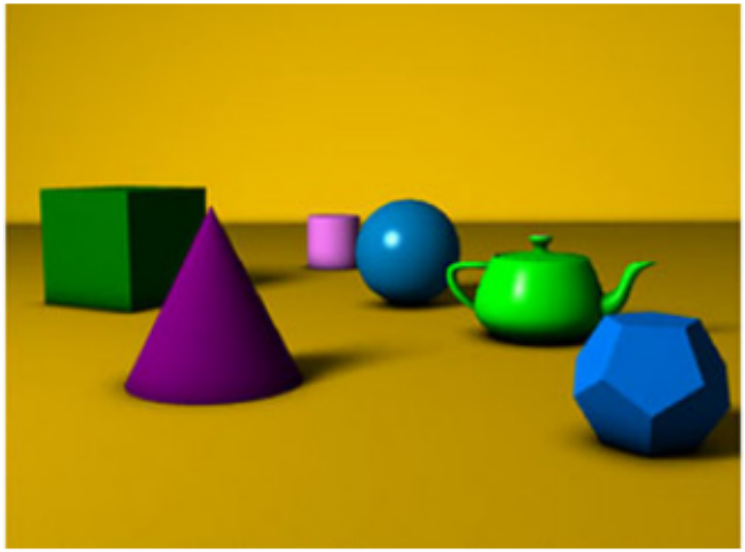

Figura: Cor mais clara → Mais longe; Cor mais escura → Mais perto (Retirada de [4]).

Bibliografia

- [1] HEARN, Donald D.; BAKER, M. Pauline; CARITHERS, Warren. Visible-Surface Detection Methods. In: HEARN, Donald D.; BAKER, M. Pauline; CARITHERS, Warren. *Computer Graphics with OpenGL*. 4. ed. Essex, England: Pearson Education Limited, 2014. cap. 14, p. 465-491.

- [2] SHIRLEY, Peter et al. The Graphics Pipeline: Operations Before and After Rasterization. In: SHIRLEY, Peter et al. *Fundamentals of Computer Graphics*. 3. ed. Natick, Massachusetts: CRC Press, 2009. cap. 8, p. 173-180.

- [3] VAN WIJK, Jack. *2IV60 Computer graphics set 11: Hidden Surfaces*. Eindhoven, Netherlands, 2016. 31 slides, color. Disponível em: <https://www.win.tue.nl/~vanwijk/2IV60/2IV60_11_hidden_surfaces.pdf>. Acesso em: 17 out. 2018.

- [4] VIDALON, José E. Y.; NAKASHIMA, Natasha S. D. *Visibilidade em Computação Gráfica:* Principais algoritmos de visibilidade. Disponível em: <http://www.dca.fee.unicamp.br/courses/IA725/1s2011/projetos/vidalon-nakashima/Algoritmos.html>. Acesso em: 18 out. 2018.