Computação Gráfica

Oclusão de Superfícies

Prof. Alaor Cervati Neto

2023/1

Detecção de Superfícies Visíveis

Detecção de Superfícies Visíveis

- Na geração de elementos gráficos realistas, é importante determinar o que será visível dentro de uma cena a partir de uma posição de visualização.
- Há vários algoritmos para a identificação de objetos visíveis, dependendo da abordagem escolhida e a finalidade do programa.
- Por eficiência, é interessante que sejam renderizadas apenas as faces do objeto visíveis para a câmera.

Detecção de Superfícies Visíveis

Estes algoritmos podem ser classificados de acordo com a forma de tratamento das definições dos objetos ou projeções de imagens na cena. Podem ser classificados como:

Métodos de espaço do objeto: Comparam objetos e suas partes entre si para determinar quais superfícies devem ser consideradas visíveis.

Métodos de espaço da imagem: Determina a visibilidade ponto a ponto em cada posição de pixel no plano de projeção.

A maioria dos algoritmos utiliza a segunda abordagem, sendo a primeira utilizada em casos específicos, como a exibição de linhas, por exemplo.

Back-Face Culling

Back-Face Culling

Um método simples de espaço do objeto para localizar as faces traseiras de um poliedro. Tomando a equação explícita de um plano definido pela face de um polígono:

$$Ax + By + Cz + D = 0$$

Back-Face Culling

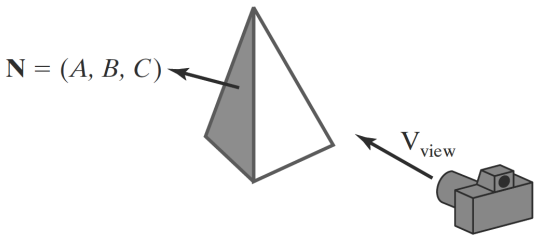
Sabe-se que um ponto $P' = (x', y', z')$ está atrás desta face se:

$$Ax' + By' + Cz' < 0$$

Assim, quando esta posição está ao longo da linha de visão da superfície, vê-se a parte traseira do polígono.

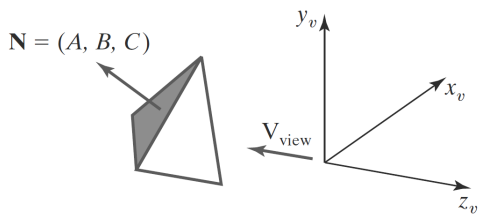
Back-Face Culling

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



Back-Face Culling

Em um sistema de coordenadas com direção de visualização ao longo do eixo $-z$, uma face de um polígono é *traseira* se a terceira componente de N é tal que $C < 0$.



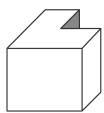
Adicionalmente, não é possível ver nenhuma face cuja terceira componente de seu vetor normal seja nula. Portanto, uma face é *traseira* se $C \leq 0$.

Back-Face Culling

- Pode-se considerar, então, que uma face de um polígono é *traseira* se o ângulo entre o vetor normal N e a direção de observação V for menor que 90°, ou seja, se $V \cdot N \geq 0$.
- Este teste pode ser realizado de forma eficiente utilizando o sistema de coordenadas do Espaço de Visão, no qual o vetor de observação é paralelo ao eixo z. Assim, $V = (0, 0, V_z)$ e $V \cdot N = V_z \cdot N_z$.
- Portanto, para fazer o teste $V \cdot N \geq 0$ basta verificar o sinal da componente z do vetor normal à face.

Back-Face Culling

- Para poliedros convexos, como a pirâmide dos exemplos anteriores, são identificadas todas as superfícies ocultas da cena.
- Para outros objetos, como poliedros côncavos, por exemplo, devem ser realizados testes adicionais para verificar se há faces que sejam total ou parcialmente obscurecidas por outras. Consequentemente, a mesma ideia deve ser aplicada se uma cena contiver objetos sobrepostos ao longo da linha de visão.

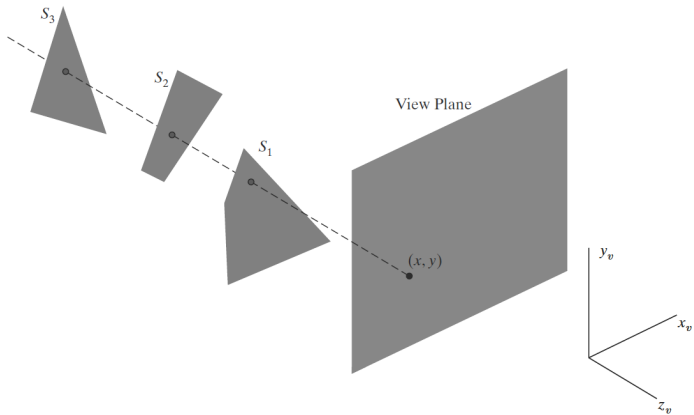


z-Buffer

z-Buffer

- Um método de espaço de imagem muito utilizado, que compara os valores de profundidade de cada superfície para cada 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, pois os valores de profundidade podem ser calculados muito rapidamente e sua implementação é simples.
- O método tem esse nome por usar os valores do eixo *z* do espaço de coordenadas da Visão para calcular as distâncias dos objetos.

z-Buffer



z-Buffer

- As superfícies representadas na figura podem ser processadas em qualquer ordem.
- Como cada superfície é processada independentemente, 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 guardada, junto com sua profundidade.
- As superfícies visíveis em uma cena são representadas pelo conjunto das cores das superfícies armazenadas após seu processamento completo.

z-Buffer

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.

A implementação do *z-Buffer* é feita utilizando coordenadas normalizadas. De maneira geral, pode ser descrita como:

z-Buffer

1. Inicialmente, todas as posições no *buffer* de profundidade são definidas como 1 (profundidade máxima) e o *frame buffer* é inicializado para a cor do plano de fundo.
2. Cada superfície 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 de profundidade será armazenado.
5. Em seguida, a cor da superfície nessa posição é calculada e colocada na localização do pixel correspondente no *frame buffer*.

z-Buffer

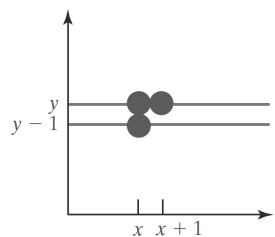
Dados os valores de profundidade para as posições dos vértices de qualquer polígono em uma cena, pode-se calcular a profundidade em qualquer outro ponto do plano que o contenha. Na posição de superfície (x, y), a profundidade é calculada a partir da equação do plano:

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

z-Buffer

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 por:

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

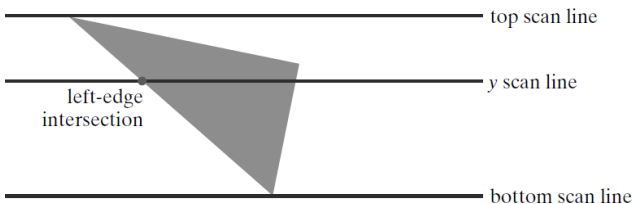


z-Buffer

A razão $-\frac{A}{C}$ 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.

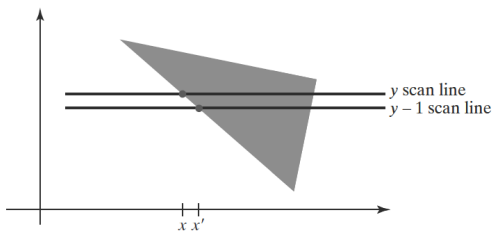
z-Buffer

Processando posições de pixel da esquerda para a direita em cada linha de varredura, inicialmente é calculada 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 $z' = z - \frac{A}{C}$.



z-Buffer

O algoritmo *z-Buffer* pode ser iniciado 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 por $x' = x - \frac{1}{m}$, onde *m* é a inclinação da borda.



z-Buffer

Os valores de profundidade abaixo desta borda são obtidos recursivamente como:

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

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

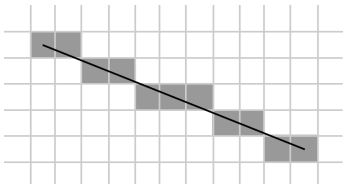
$$z' = z + \frac{B}{C}$$

z-Buffer

- 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.
- 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

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



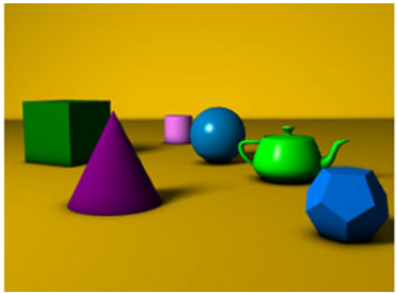
- O método (*z-Buffer*) pode também ser aplicado a superfícies curvas determinando os valores de profundidade e cor em cada ponto de projeção da superfície.

z-Buffer

- 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 *buffer* de profundidade, com cada posição contendo bits suficientes para representar o número de incrementos de profundidade necessários.
- Uma maneira de reduzir os requisitos 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

Um exemplo de mapa de profundidade gerado:



Material de base para a aula I

- 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.
- 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.

Material de base para a aula II

- 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.
- 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.
- Algoritmos de eliminação de linhas/superfícies escondidas. Slides de Luíza Gomes Accarini. Disciplina SCC0250/0650, ICMC/USP, 2018.

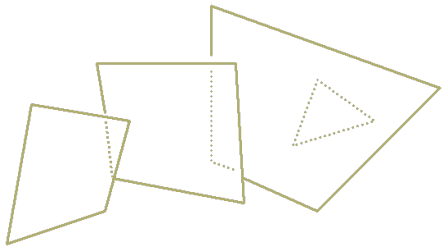
Exercícios

Exercícios I

Para os exercícios a seguir, use as coordenadas da pirâmide obtidas na aula anterior.

1. Encontre os vetores normais (que podem ser obtidos através do produto vetorial entre dois lados não paralelos) às superfícies da pirâmide.
2. Utilize o *Back-Face Culling* para determinar quais faces/vértices são visíveis e quais estão ocultos.
3. Considere a figura a seguir:

Exercícios II



Atribua uma cor para cada polígono e represente aproximadamente a imagem gerada usando o método *z-Buffer*, em um plano de visão projetado posteriormente ao apresentado (isto é, com o observador visualizando a cena por trás, com as profundidades dos objetos opostas às da figura).