UFABC

Computação Gráfica

André Brandão

Aula 12

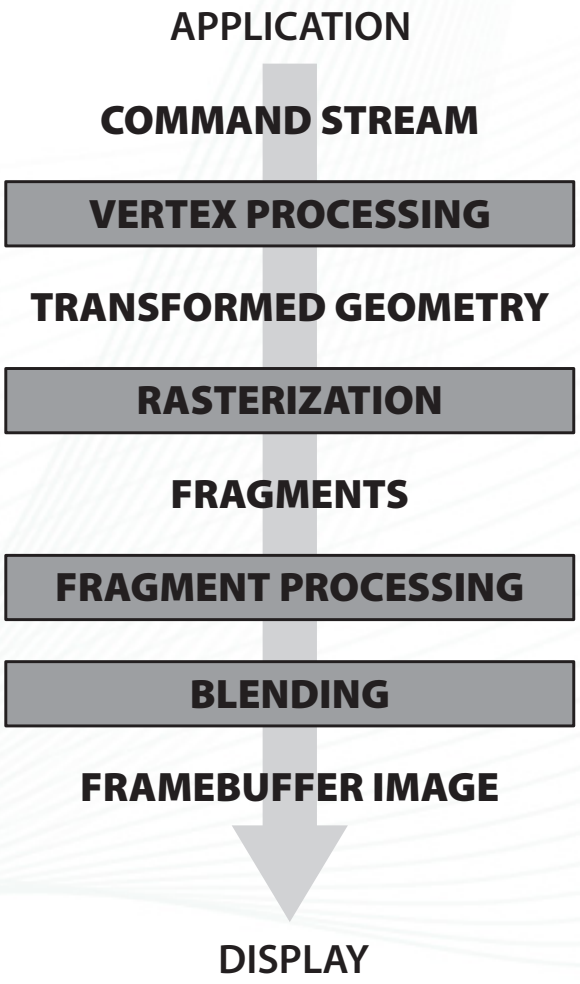
Pipeline Gráfico

(continuação)

Sumário

- Rasterização
- Operações antes e depois da rasterização
- Antialiasing simples
- Seleção de primitivas para eficiência

Etapas do Pipeline Gráfico



Sumário

- ~~Rasterização~~
- **Operações antes e depois da rasterização**
- Antialiasing simples
- Seleção de primitivas para eficiência

Operações antes e depois da rasterização

- Antes que uma primitiva possa ser rasterizada, os vértices que a definem devem estar em coordenadas de tela. Para que as cores e outros atributos que possam ser interpolados, os vértices devem ser conhecidos.
- A preparação dos dados mencionados é uma função da etapa de *Vertex processing*. Nessa etapa, os vértices são processados por meio das transformações de modelagem, visualização e de projeção. Os vértices são mapeados de suas coordenadas originais para coordenadas de tela.

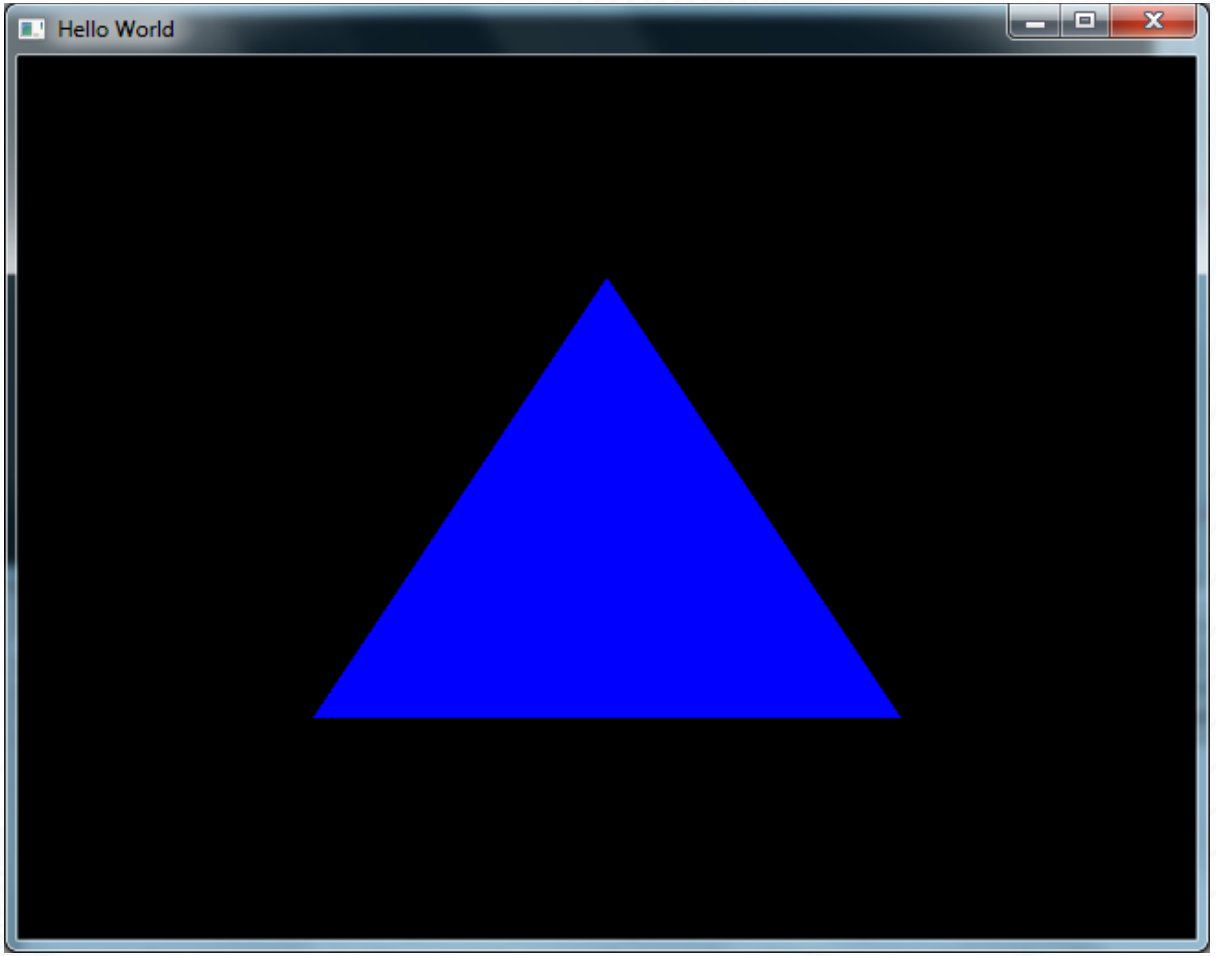
Operações antes e depois da rasterização

- Após a rasterização, mais processamentos são realizados para computar a cor e a profundidade de cada fragmento. O processamento pode ser desde interpolar cores para definir a cor atual do pixel até operações complexas de *Shading*.
- Na etapa de *blending*, os fragmentos gerados pelas primitivas, que são sobrepostos em cada pixel, são computados para a definição da cor final de um pixel. A abordagem mais comum para a definição de cor é a utilização do dado do fragmento de menor profundidade (mais perto dos olhos).

Desenho simples em 2D

- No pipeline mais simples, nada acontece nas fases de *Vertex processing* e *Fragment processing.* Na fase de *blending*, cada fragmento simplesmente define o valor do pixel atual com o mesmo valor do pixel anterior.
- A aplicação fornece as primitivas diretamente em coordenadas de pixels e a etapa de rasterização realiza todo o trabalho.
- Esse tipo de rasterização foi realizado na Aula 02.

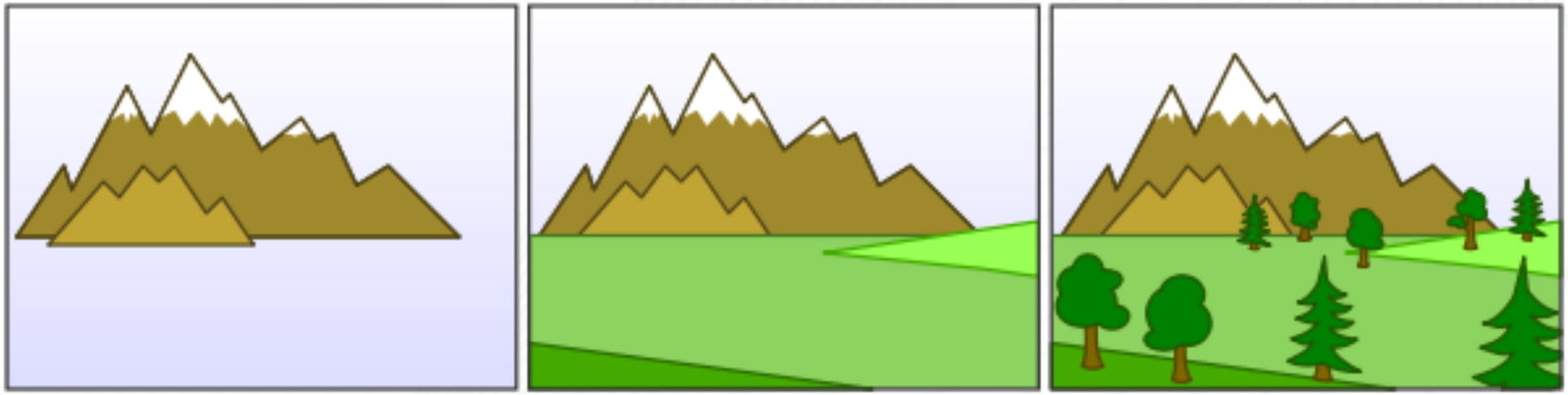
Desenho simples em 2D



Pipeline mínimo em 3D

- Para desenhar objetos em 3D, apenas uma mudança é necessária para o desenho em 2D; o *Vertex processing* multiplica as posições de vértices pelo produto das matrizes de câmera, projeção e de janela. Isso resulta no espaço de tela, que desenha os triângulos da mesma forma como se fossem especificados em 2D.
- O problema do pipeline mínimo em 3D é a relação de oclusão entre primitivas. Nesse caso, as primitivas devem ser desenhadas na ordem de trás para frente.

Algoritmo do pintor

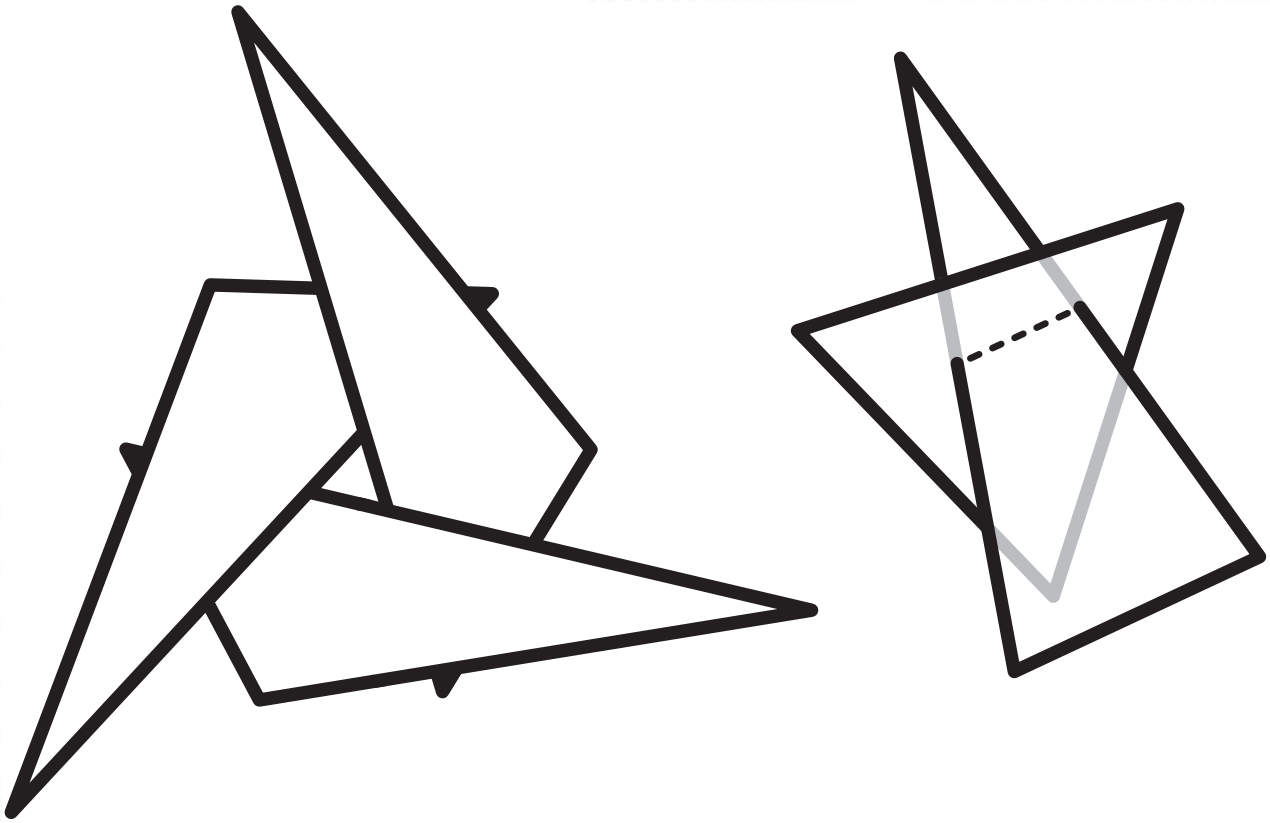


https://en.wikipedia.org/wiki/Painter%27s_algorithm

Pipeline mínimo em 3D

- Desenhar de trás para frente é conhecido como o algoritmo do pintor (*painter's algorithm*). Primeiro, pinta-se o que está no fundo, para, depois, pintar o que está na frente.
- O algoritmo do pintor tem problemas relacionados a interseções de triângulos, por conta de não existir uma ordem correta para desenhá-los. Esse é o mesmo caso de ciclos de oclusão, pois não existe uma relação de objetos que estão na frente uns dos outros.

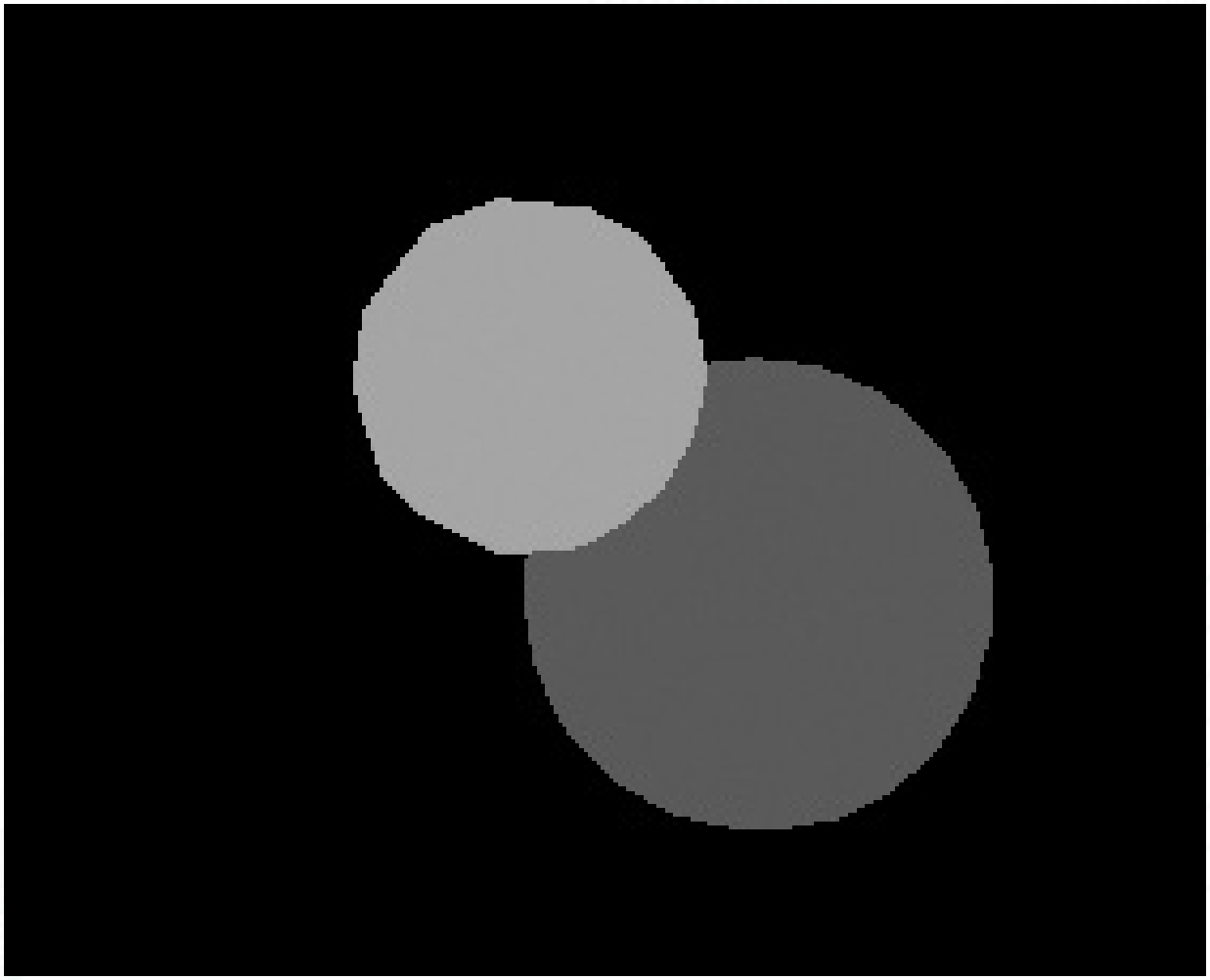
Pipeline mínimo em 3D



Pipeline mínimo em 3D

- Outro problema é que, ordenar as primitivas pela profundidade é algo, em termos computacionais, caro. Isso ocorre, especialmente, nos casos de cenas muito grandes.
- Um problema que pode ocorrer são erros de ordenação. Exemplo, desenhar uma esfera mais distante antes de desenhar a esfera mais próxima. No caso, duas esferas de mesmo tamanho.

Pipeline mínimo em 3D



Z-Buffer para superfícies oclusas

- O algoritmo do pintor dificilmente é usado.
- O algoritmo mais usado é o algoritmo do Z-Buffer.
- Para cada pixel, o algoritmo mantém o valor da distância mais próxima do plano de projeção que foi desenhado. Assim, os valores que são maiores do que a distância mais próxima guardada são descartados.

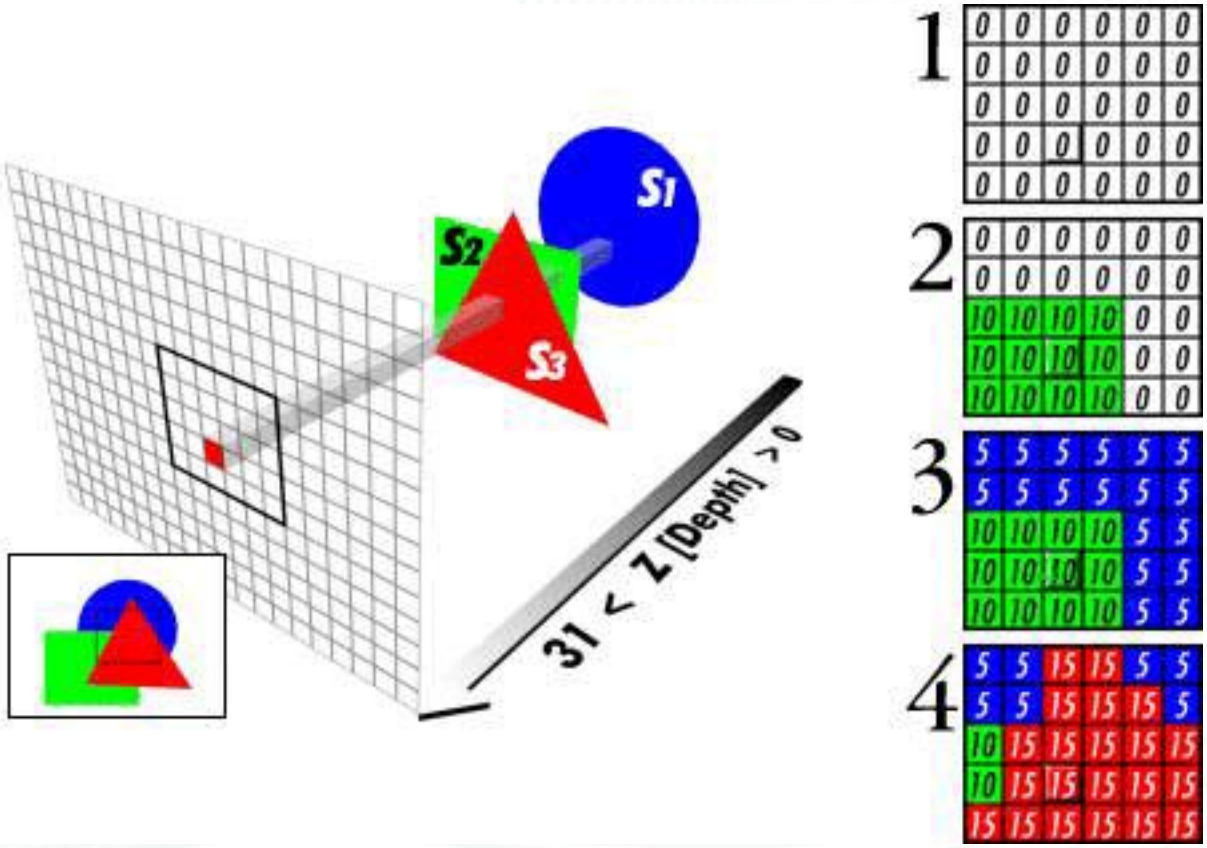
Z-Buffer para superfícies oclusas

- A distância é guardada pelos fragmentos em uma alocação extra, além dos valores RGB previamente salvos. A esse valor de profundidade, chamamos de valor Z.
- O Z-Buffer é o nome da grade de valores de Z.
- O algoritmo de Z-Buffer é implementado na etapa de *blending*.

Z-Buffer para superfícies oclusas

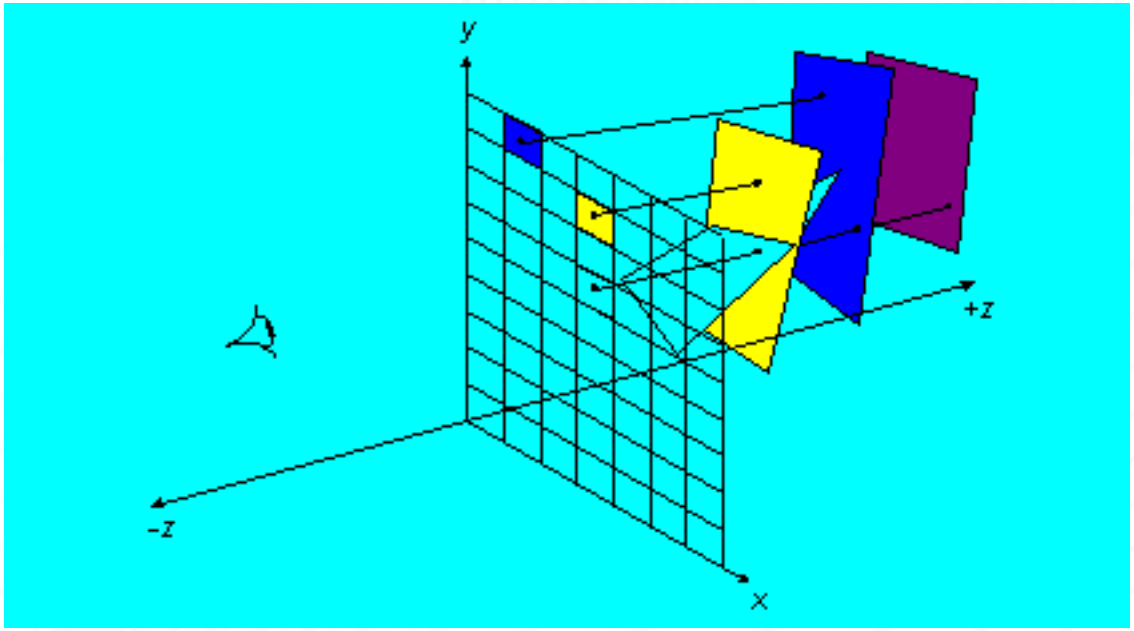
- Cada valor de Z, de cada fragmento, é comparado com o valor corrente salvo no Z-Buffer, para um dado pixel. Assim, um mesmo pixel pode ser destinado a diferentes fragmentos.
- Se o valor Z do fragmento for menor do que o valor do pixel corrente comparado no Z-Buffer, tanto a cor quanto a distância Z são sobreescritas na posição corrente da cor e da profundidade no Z-Buffer.
- Senão, os valores de cor e de Z são descartados, ou seja, o fragmento sofre oclusão por outro fragmento, mais próximo ao plano de projeção.

Z-Buffer



http://bio.gsnu.ac.kr/~youknow/graphic/3DMAX_2img/3.jpg

Z-Buffer

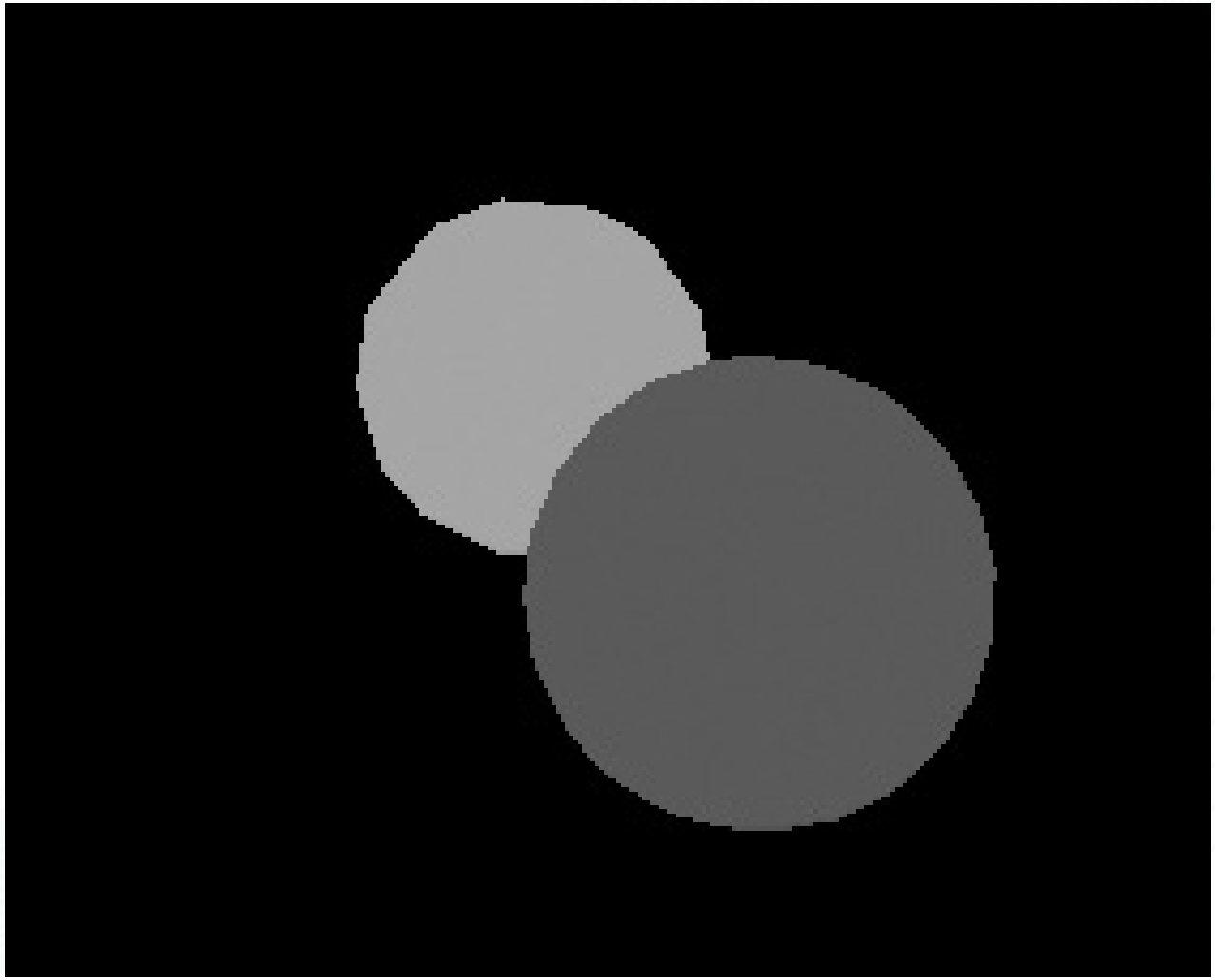


http://csis.pace.edu/~marchese/CG_Rev/Lect9New/cg_l9new_files/image027.gif

Z-Buffer para superfícies oclusas

- Afim de garantir que o primeiro pixel terá distância menor do que a distância inicial de Z-Buffer, este é inicializado com o valor máximo possível para o tipo da variável.
- Cada fragmento guarda, além da cor, a profundidade do mesmo em relação ao plano de projeção.
- O algoritmo Z-Buffer é a abordagem dominante para rasterizar objetos de uma cena.

Esferas do mesmo tamanho e com distâncias diferentes



Z-Buffer para superfícies oclusas

- Os valores Z são inteiros, pois a comparação entre inteiros é menos custosa do que a comparação de números do tipo *float*.

Shading por vértice

- Em alguns casos, a definição das cores dos vértices podem ser atribuídas diretamente ou atribuídas por interpolação.
- Em outros casos, pode ser desejável realizar uma tonalização (*Shading*), que depende: da cor do fragmento, da normal, do ângulo do raio de luz e do ângulo do observador.
  - *Lambert Shading*
  - *Blinn-Phong Shading*

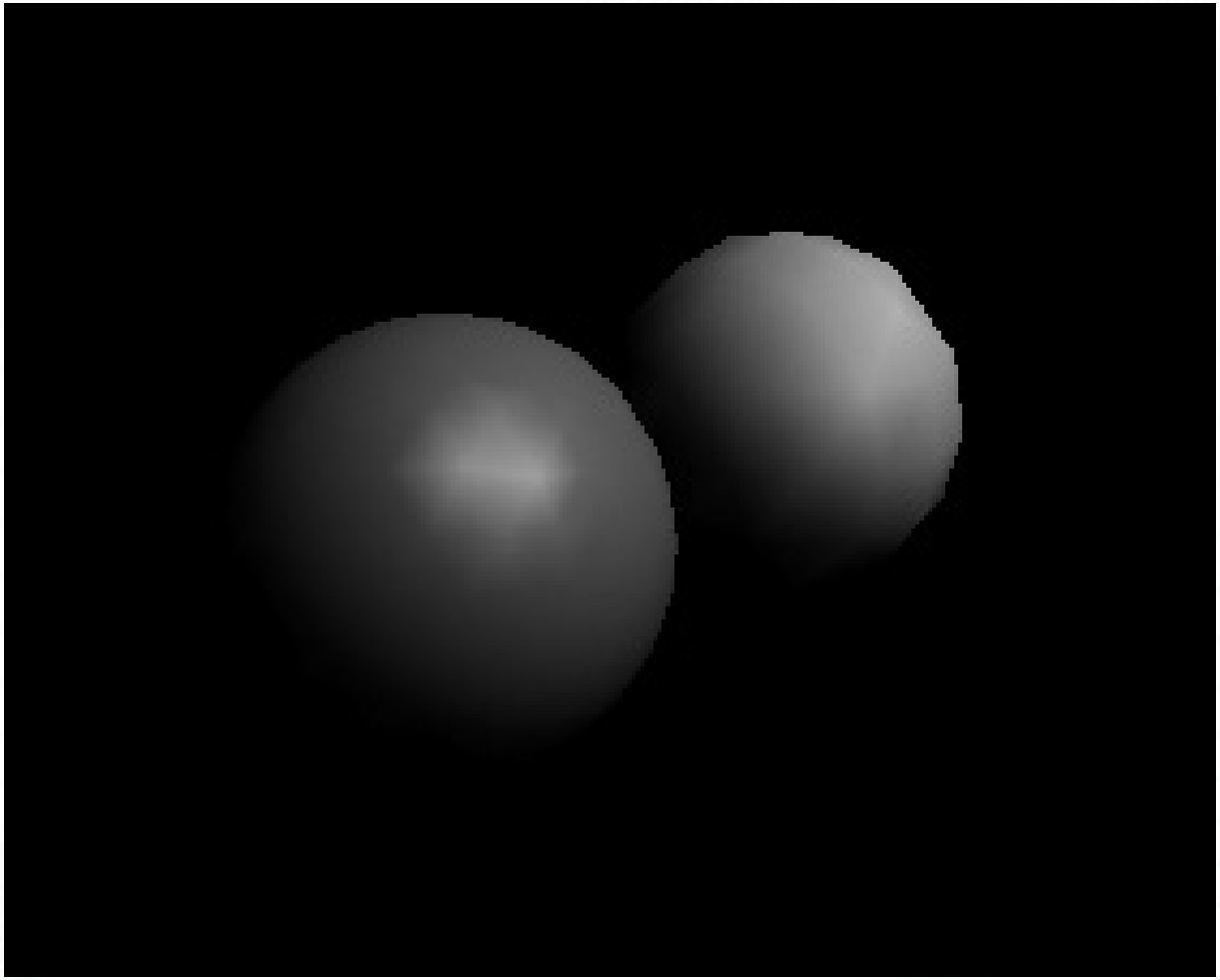
Shading por vértice

- Uma maneira de se aplicar Shading é na etapa de *Vertex processing*. Assim, a aplicação envia as normais de cada vértice, juntamente com os mesmos.
- Para cada vértice, a direção do observador e direção de cada fonte de luz são computadas baseadas nas posições da câmera, das luzes e do vértice.

Shading por vértice

- O modelo de Shading computa a cor do pixel e a envia para a etapa de rasterização como a cor do vértice.
- É comum de se chamar o Shading por vértice como ***Gouraud shading***.

Gouraud shading



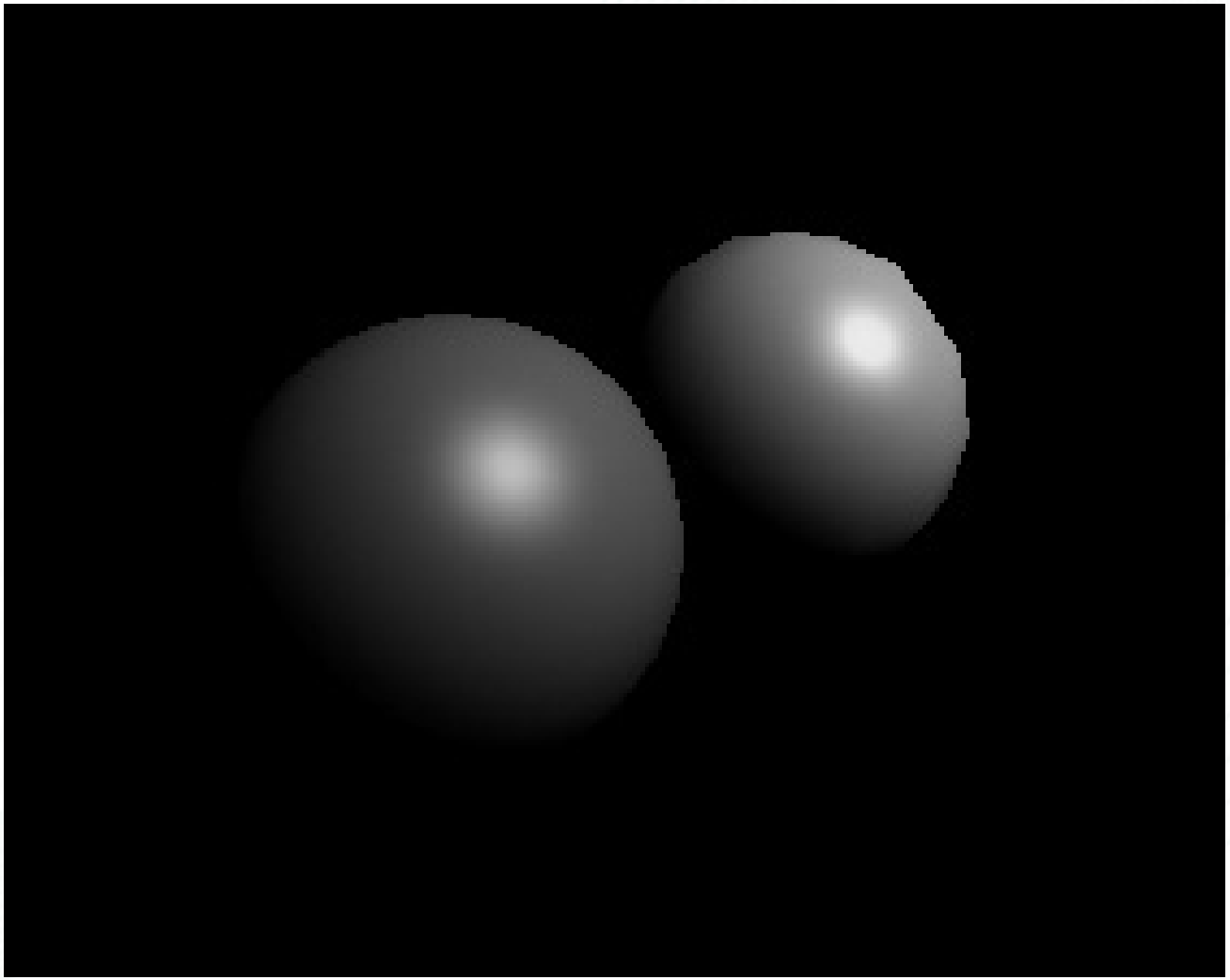
Shading por fragmento

- A tonalização, ou Shading, também pode ser executada na etapa de *Fragment processing*.
- No Shading por fragmento, os modelos de Shading são avaliados, mas são avaliados para cada fragmento, usando vetores interpolados, ao invés de usar vetores para cada vértice, que viriam da aplicação.

Shading por fragmento

- No Shading por fragmento, a informação geométrica necessária para a tonalização é passada à etapa de Rasterização como atributos dos vértices.
- Assim, a etapa de *Vertex processing* deve ser sincronizada com a etapa de *Fragment processing*, afim de preparar os dados apropriadamente.

Shading por fragmento



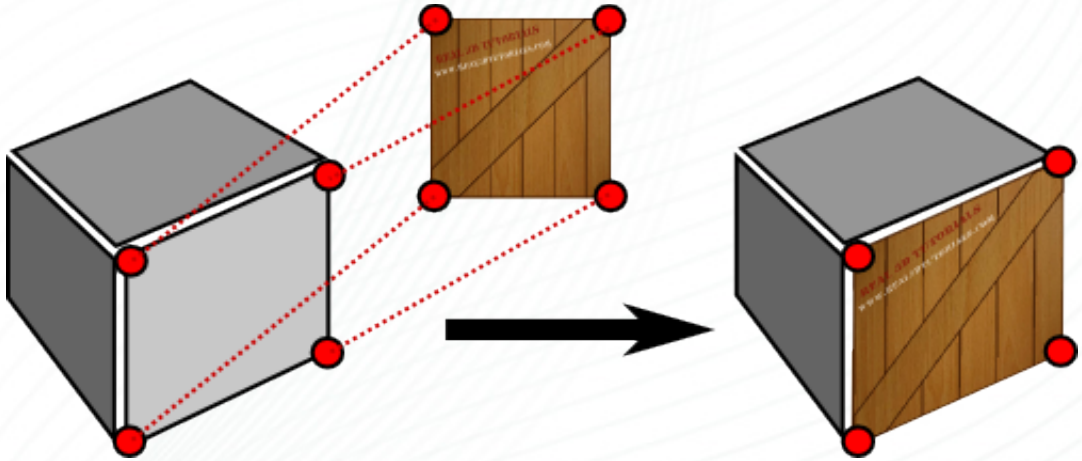
Mapeamento de textura

- Texturas são imagens que são usadas para adicionar detalhes extras na tonalização de superfícies que, de outra forma, dariam uma aparência muito homogênea e artificial.
- Cada Shading é computada, nós lemos um dos valores usados na computação de tonalização de uma textura, ao invés de usar os valores de atributos que estão anexados na geometria renderizada.

Mapeamento de textura

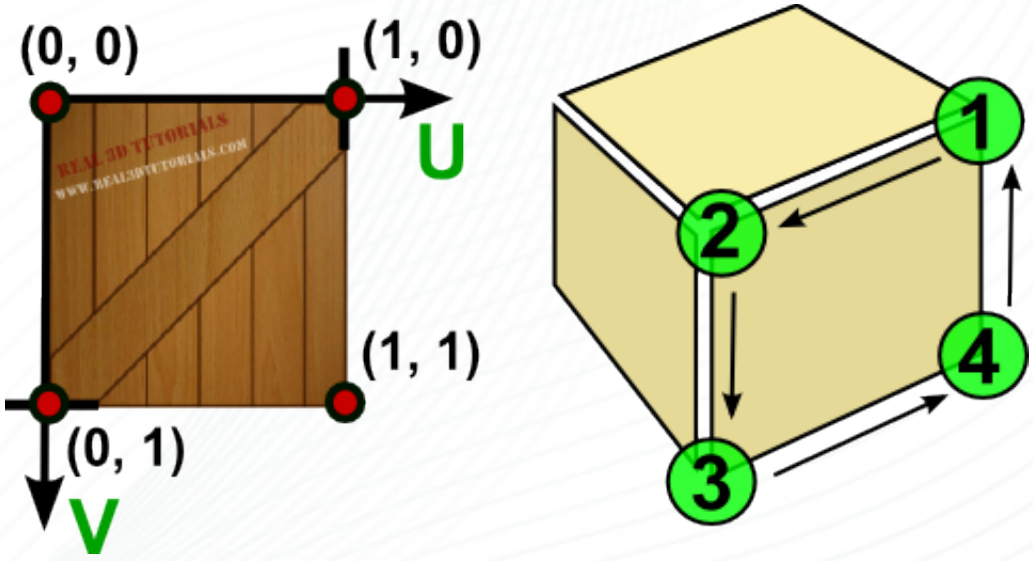
- O código de shading especifica a coordenada de textura, um ponto no domínio de textura e o sistema de mapeamento de textura encontra o valor no ponto da imagem de textura.
- Então, o valor da textura é usado na operação de Shading.

Mapeamento de textura



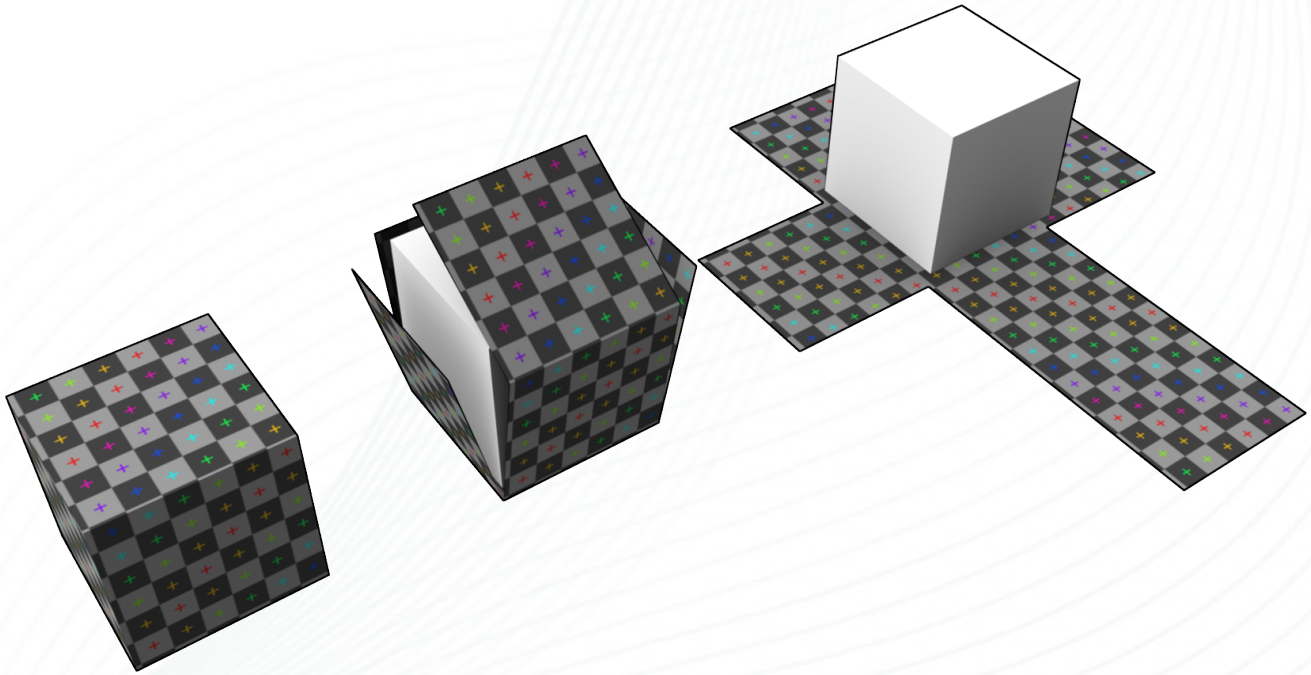
http://www.real3dtutorials.com/images/img00017.png

Mapeamento de textura



http://www.real3dtutorials.com/images/img00018.png

Mapeamento de textura



https://upload.wikimedia.org/wikipedia/commons/f/fe/Cube_Representative_UV_Unwrapping.png

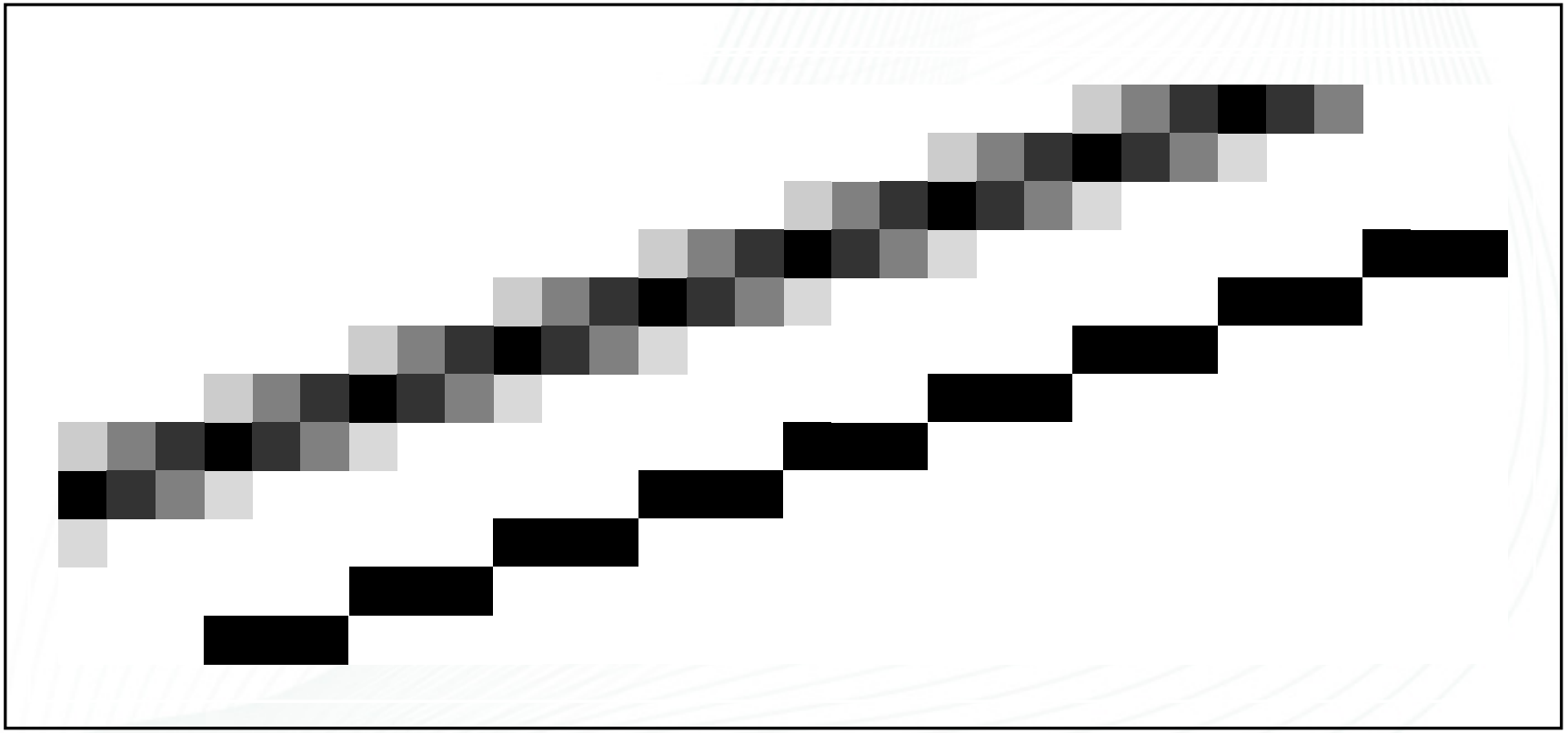
Sumário

- ~~Rasterização~~
- ~~Operações antes e depois da rasterização~~
- **Antialiasing simples**
- Seleção de primitivas para eficiência

Antialiasing simples

- Assim como no Ray Tracing, na rasterização, ao adotar abordagens “tudo-ou-nada” para definirmos se um pixel pertence ou não a uma primitiva, linhas irregulares podem ocorrer na imagem renderizada.
- Uma possível solução é permitir que pixels estejam parcialmente presentes na primitiva. Isso causa um “borramento” no conjunto de pixels que estão próximos dos limites dessas primitivas.

Antialiasing simples



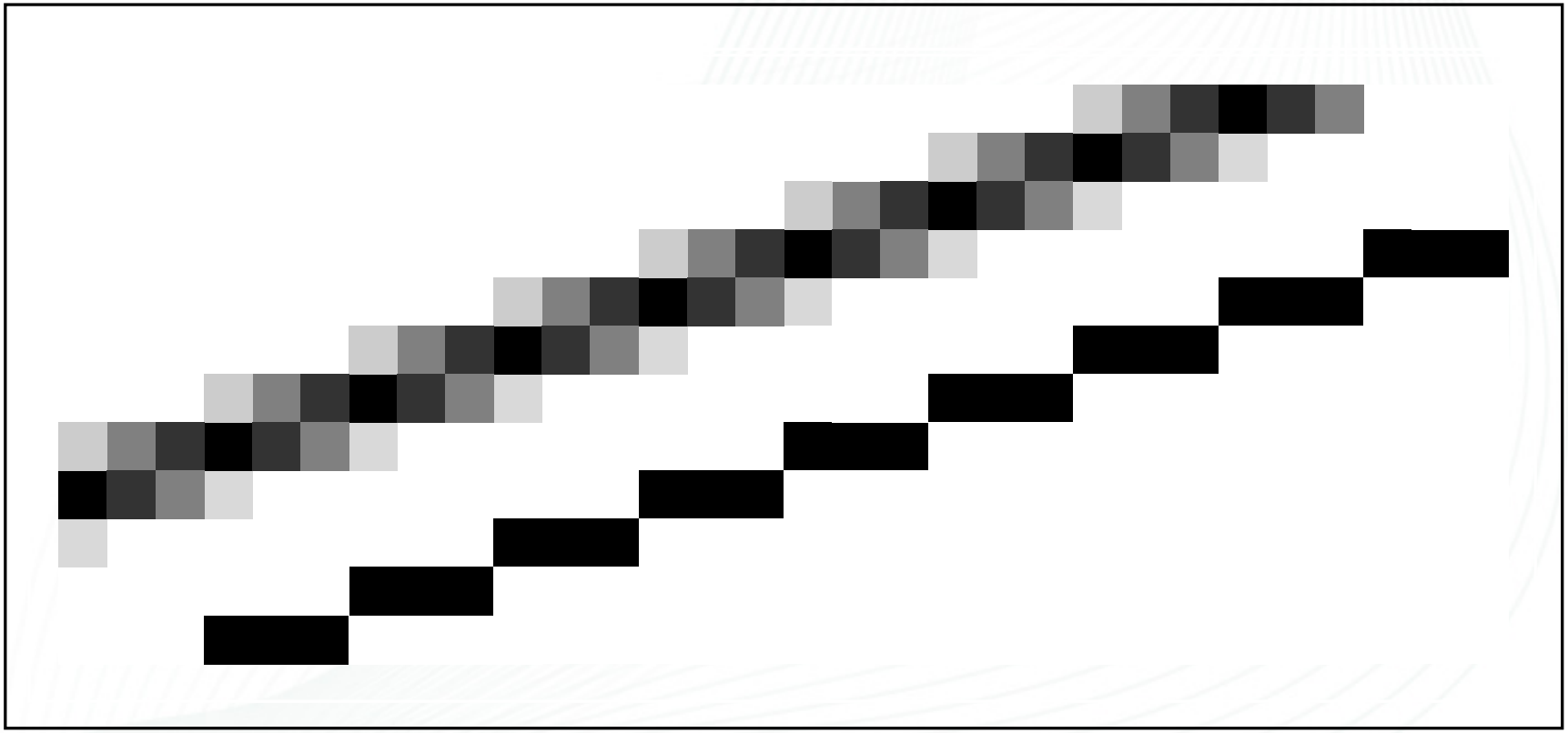
Antialiasing simples

- Para realizar a operação de antialiasing, existem diversas abordagens. Podemos produzir uma imagem sem aliasing ao definir cada valor de pixel como a média das cores dos pixels vizinhos em uma área quadrada.
- Essa abordagem é conhecida como *box filtering.*

Antialiasing simples

- Assim, temos de pensar em todas as entidades “desenháveis” como áreas bem definidas.
- Por exemplo, a linha na figura pode ser pensada como uma aproximação de uma linha de comprimento de um pixel.

Antialiasing simples



Antialiasing simples

- Uma outra maneira de se tratar o aliasing é, rasterizar uma versão superestimada da imagem. Após essa rasterização superestimada, diminui-se a escala do objeto a ser rasterizado, com informações suficientes que evitam o aliasing.
- Por exemplo, ao rasterizar um objeto afim de termos uma imagem de 256 x 256 pixels, podemos rasterizar uma versão deste objeto em uma imagem de 1024 x 1024 e, depois, diminuir a escala deste objeto.

Sumário

- ~~Rasterização~~
- ~~Operações antes e depois da rasterização~~
- ~~Antialiasing simples~~
- **Seleção de primitivas para eficiência**

Seleção de primitivas para eficiência

- O ponto forte da renderização por ordem de objeto (*object-order rendering*) é que ele requere apenas uma passagem sobre toda a geometria da cena. Porém, esse é, também, um ponto fraco, se considerarmos cenas complexas.
- Por exemplo, em um modelo que contém uma cidade inteira, apenas alguns prédios serão desenhados na imagem final da tela.
- Então um grande esforço computacional para calcular as distâncias dos objetos até o plano de projeção é realizado para apenas alguns objetos aparecerem.

Seleção de primitivas para eficiência

http://www.hpp-international.com/abbildungen/presentation/001334.jpg

Seleção de primitivas para eficiência

- A identificação e descarte de geometria invisível para poupar tempo que seria dedicado para processá-la é conhecida como ***culling***.
- Entre as abordagens de ***culling***, podemos destacar três delas.

Culling

- ***View volume culling**:* remove geometria que está fora do volume de visão.
- ***Occlusion culling**:* remove a geometria que pode estar no volume de visão mas está obscura ou oclusa por outra geometria, mais próxima à camera
- ***Backface culling**:* remove as primitivas que estão fora do campo de visão da câmera.

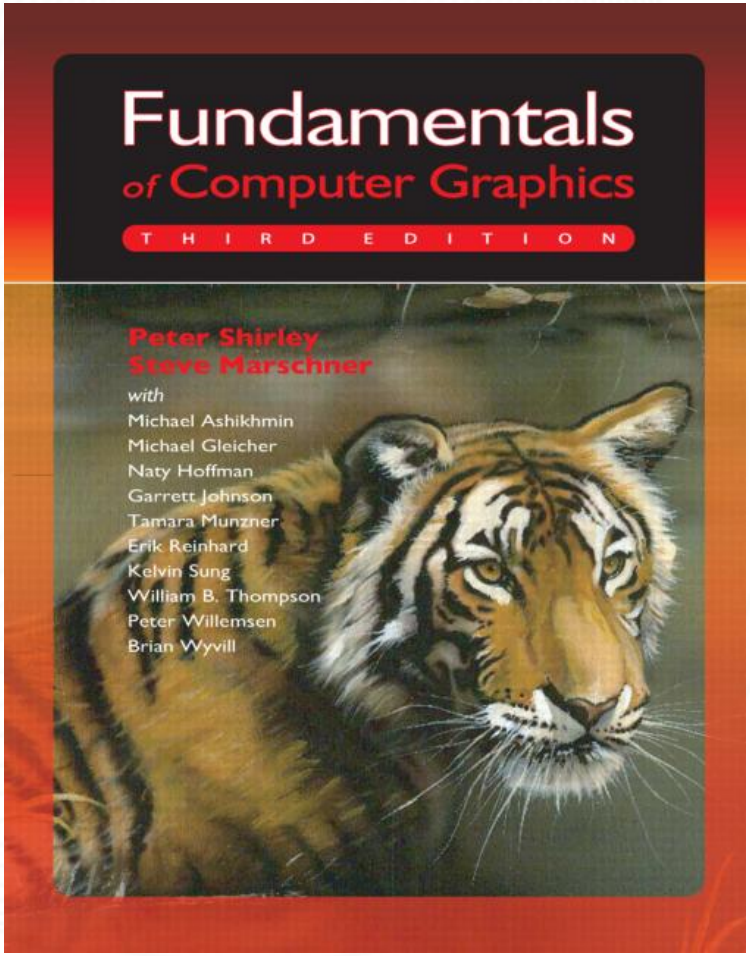
Sumário

- ~~Rasterização~~
- ~~Operações antes e depois da rasterização~~
- ~~Antialiasing simples~~
- ~~Seleção de primitivas para eficiência~~

Aula de hoje

Shirley, Peter, Michael Ashikhmin, and Steve Marschner. Fundamentals of computer graphics. CRC Press, 3rd Edition, 2009.

- **Capítulo 8**



Fim da Aula 12

André Luiz Brandão