Universidade Federal do Piauí
Centro de Ciências da Natureza
Departamento de Computação

Realismo Visual e Visibilidade

Prof. Dr. Laurindo de Sousa Britto Neto

1

Realismo Visual

- São as técnicas de tratamento computacional aplicadas aos objetos sintéticos com o objetivo de lhes criar uma imagem sintética, o mais próximo da realidade que se teria se eles fossem construídos e filmados [Azevedo e Conci, 2003].
- O processo de criação sintética de imagem realista é denominado ***Rendering*** (Renderização):
  - Processo de converter dados em uma imagem realista;

2

2

Renderização

- Fotorealística
  - processo de converter dados em uma imagem sintética realística; Processo de Realismo Visual;
  - a imagem sintética chega a se confundir com uma fotografia.
- Não Fotorealística (Renderização Estilizada)
  - Processo de converter dados em uma imagem sintética que simula técnicas de expressões artísticas;
  - Processamento de objetos gráficos para que possuam o atrativo visual de obras de arte, expressando características visuais e emocionais de estilos artísticos .

3

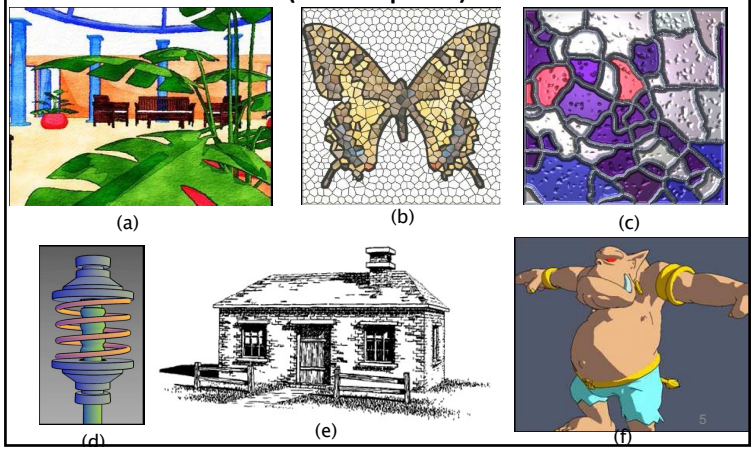
3

Renderização Fotorealística (Exemplo)

4

4

Renderização Não Fotorealística (Exemplos)



5

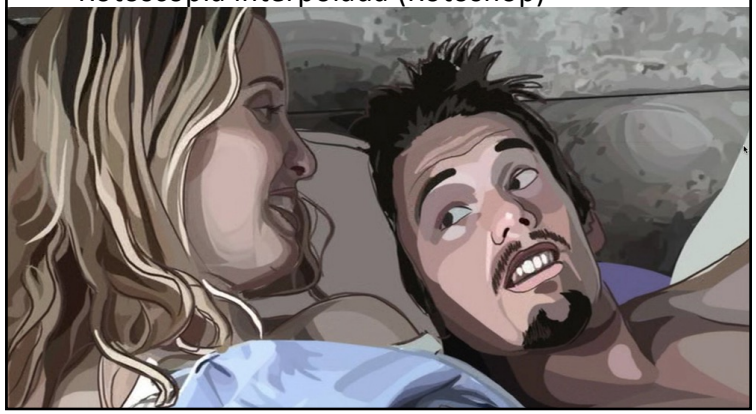
Amor Além da Vida (1998)

- Fluxo ótico para direcionar pinceladas;

6

Walking Life (2001)

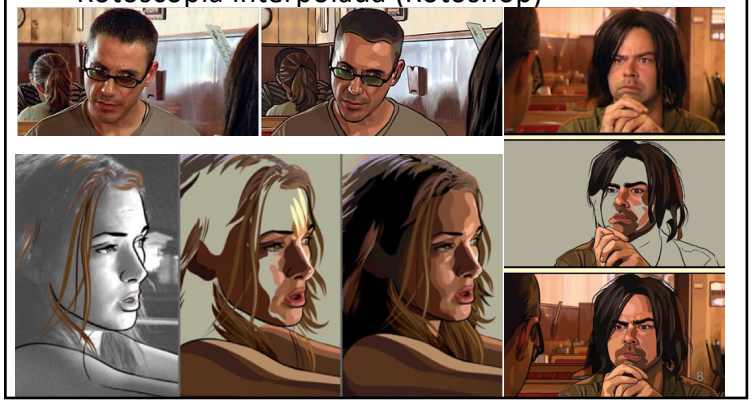
- Rotoscopia interpolada (Rotoshop)



7

O Homem Duplo (2006)

- Rotoscopia interpolada (Rotoshop)



8

Renderização Fotorealística

- Dividido em 7 Etapas:

1. Construção do Modelo 3D (Modelagem 3D)
2. Aparência 3D (Transformações e Projeções)
3. Eliminação de Polígonos ou Faces Escondidas (Culling)
4. Recorte (Clipping)
5. Conversão da representação 3D para 2D (Rasterização)
6. Tratamento de Partes Escondidas (Visibilidade - Hidden)
7. Coloração dos Pixels (Textura, Iluminação e Sombreamento)

9

9

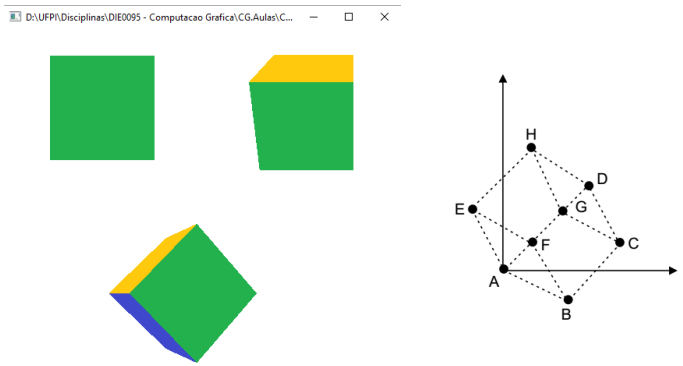
Visibilidade: Problema

- Numa cena 3D, normalmente, não é possível ver todas as superfícies de todos os objetos;
- Não queremos que objetos ou partes de objetos não visíveis apareçam na imagem;
- Problema importante que tem diversas ramificações:
  - Descartar objetos que não podem ser vistos (Culling);
  - Recortar objetos de forma a manter apenas as partes que podem ser vistas (Clipping);
  - Desenhar apenas as partes visíveis dos objetos (Hidden):
    - Em linhas/aramado/*wire-frame* (hidden line algorithms);
    - Superfícies (hidden surface algorithms);
  - Sombras (visibilidade a partir de fontes luminosas).

10

10

Exemplo: Visualizando um Cubo



11

11

Motivação

- Dispositivos matriciais sobrescrevem os objetos (aparecem os objetos desenhados por último, quando há sobreposição).
  - Em 3D, se nada for feito para corrigir a ordem de desenho, isso gera uma imagem incorreta.
- Os algoritmos de visibilidade estruturam os objetos da cena, de modo que sejam exibidos corretamente.

12

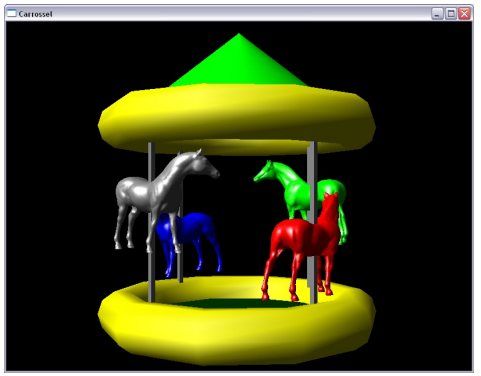
12

Exemplo: Imagem **SEM** aplicação de Algoritmos de Visibilidade

13

13

Exemplo: Imagem **COM** aplicação de Algoritmos de Visibilidade



14

14

Algoritmos de Visibilidade

- Culling
  - Algoritmo de Eliminação de Faces Ocultas pelo Cálculo da Normal (Back-face Culling)
- Hidden Surface Removal (HSR)
  - Algoritmo de Visibilidade por Prioridade (Algoritmo do Pintor)
  - Algoritmo Z-buffer

15

15

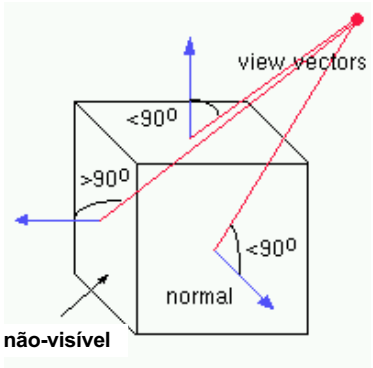
Back-face Culling

- Utiliza o teste visibilidade da normal (Robert's visibility test)
  - Verifica a magnitude do ângulo (**β**) formado pela normal da face em consideração à linha de visão;
  - Se o valor absoluto do ângulo estiver no intervalo de 0^o a 90^o, então a superfície é visível;
  - Caso contrário, a superfície é não-visível;

16

16

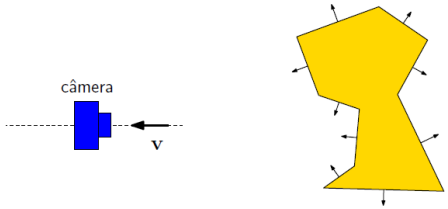
Back-face Culling: Exemplo 1



17

17

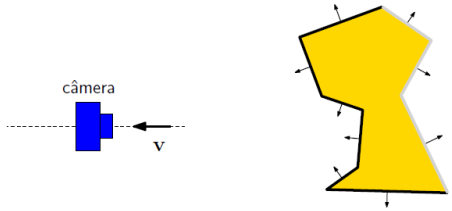
Back-face Culling: Exemplo 2 (1/2)



18

18

Back-face Culling: Exemplo 2 (2/2)



19

19

Algoritmo Back-face Culling

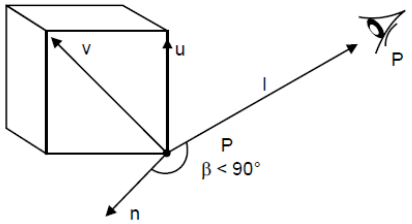
- Ler as coordenadas do objeto no espaço 3D, considerando um ponto de referência e armazená-las em forma de matriz;
- Localizar no espaço a posição do observador;
- Calcular o vetor normal (**n**) de cada face do objeto;
- Calcular o vetor linha (**l**) de visibilidades para cada face do objeto;
- Realizar o teste de visibilidade para os dois vetores;
- Definir os vértices das faces do objeto e armazená-los de forma raster;
- Verificar os vértices visíveis, com seus respectivos posicionamentos;
- Rasterizar as faces visíveis;

20

20

Algoritmo Back-face Culling

- Teste de Visibilidade: Calcular ângulo β;



- Vetores de orientação (u e v), vetor normal (n) e vetor linha de visibilidade (l);

21

21

Algoritmo Back-face Culling

- Teste de visibilidade
  - Dois vetores de orientação (u e v) associados a face ou superfície;
  - Vetor normal (n) de cada uma dessas faces ou superfícies;

$$n = u \times v = (y_u z_v - y_v z_u,\ x_v z_u - x_u z_v,\ x_u y_v - x_v y_u)$$

  - Vetor linha de visibilidade (l): diferença entre as coordenadas do observador e de um dos vértice da face do objeto
  - Cálculo do ângulo (β) entre (n e l):
    Se (**|β| < 90º**) então **visível = true**; senão **visível = false**;

$$n \cdot l = \|n\| \, \|l\| \cos\beta \Rightarrow \beta = \arccos\left(\frac{n \cdot l}{\|n\| \, \|l\|}\right)$$

  - Se **n·l > 0**, face visível /* |β| < 90º */
  - Se **n·l = 0**, (l) é paralelo ao plano /* |β| = 90º */
  - Se **n·l < 0**, face não-visível /* |β| > 90º */

$$n \cdot l = n_x l_x + n_y l_y + n_z l_z$$

22

22

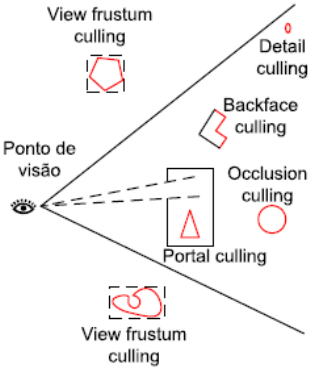
Observações: Back-face Culling

- O algoritmo sozinho não constitui uma técnica completa para determinação de faces ocultas;
- Utilizada como um pré-filtro de faces para redução no tempo de processamento de outras fases da renderização, como sombreamento e definição da cor das superfícies;
- OpenGL: glEnable(GL_CULL_FACE);

23

23

Outros Algoritmos de Culling



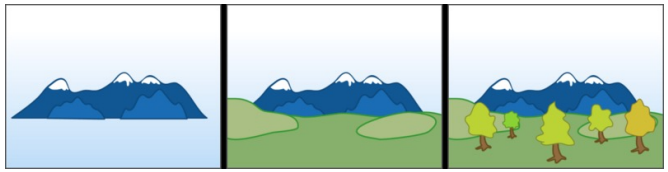
24

24

Algoritmo de Visibilidade por Prioridade (Algoritmo do Pintor)

– Solução mais simples;

– Simula a forma como um pintor faria;



– Se A bloqueia a visão de B então B está mais distante do que A

25

25

Algoritmo do Pintor

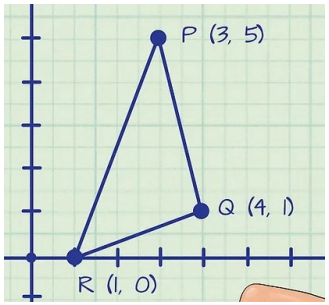
- Calcular a distância ao observador de todas as faces poligonais da cena; (centróide, baricentro, centro de gravidade)
  - Observador na origem: $D = \sqrt{x^2 + y^2 + z^2} \Rightarrow D = |x| + |y| + |z|$
- Ordenar todos os polígonos pelo valor da distância ao observador;
- Resolver as ambiguidades;
- Desenhar primeiro os polígonos que estiverem mais distantes do observador;

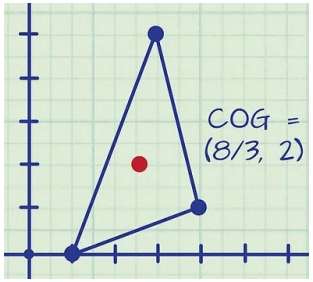
26

26

Calculando Centróide do Triângulo

- Calcular média aritmética das coordenadas



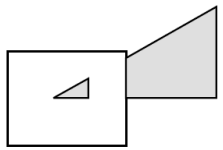


27

27

Problema do Algoritmo do Pintor

- Falha quando ocorre sobreposição de polígonos:
  - Domínio: se A bloqueia B, B não pode bloquear A;
    - Funciona: polígonos convexos sem sobreposição;
  - Falha: polígonos não-convexos e disposições "exóticas";
    - Podem bloquear uns aos outros;
  - Solução: usar vários pontos do polígono;



28

28

Z-Buffer

- Catmull, 1974 (https://pt.wikipedia.org/wiki/Edwin_Catmull);
- Simples de implementar tanto em software como em hardware;
- Alto custo de memória e processamento;
- Atua no Espaço da Imagem;
  - Entrada vetorial e saída matricial;
- Buffer de profundidade ou z-buffer;
- Buffer de imagem ou "rascunho".

29

29

Principais Vantagens e Desvantagens

- Principais vantagens:
  - Sempre funciona;
  - fácil implementação;
- Principais desvantagens:
  - Alteração frequente no valor do pixel;
  - Dificulta o uso de transparência ou técnicas de anti-aliasing;

30

30

Z-Buffer no OpenGL

- Habilitar o z-buffer:
  - glEnable (GL_DEPTH_TEST);
- Não esquecer de alocar o z-buffer
  - Ex.: glutInitDisplayMode(GLUT_RGB|GLUT_DEPTH);
- Ao gerar um novo quadro, limpar também o z-buffer:
  - glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
- A ordem imposta pelo teste de profundidade pode ser alterada:
  - Ex.: glDepthFunc (GL_GREATER);

31

31

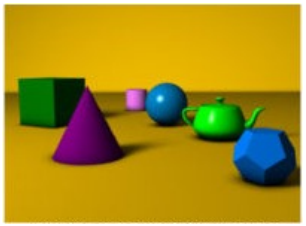
Algoritmo Z-Buffer

- Algoritmo:
  - Inicializar o z-buffer com o valor da profundidade máxima;
  - Inicializar rascunho com cor de fundo;
  - Para cada pixel projetado no rascunho:
    - Computar a coordenada **Z** para cada ponto do polígono;
    - Testar a profundidade **Z** do ponto de cada superfície;
    - Atualizar o valor no z-buffer e a cor no rascunho se **Z** estiver mais próximo do observador.

32

32

Algoritmo Z-Buffer



Cena 3D

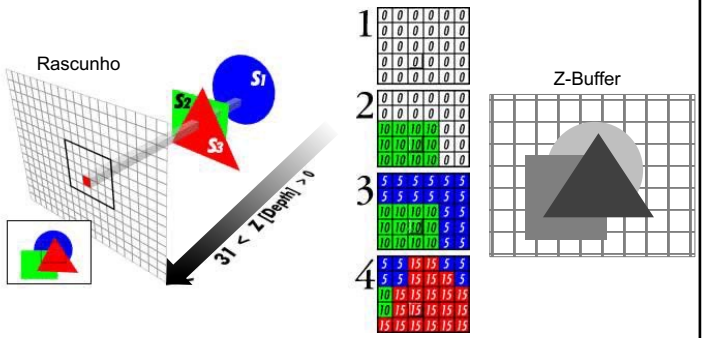
Z-buffer

- Para cada poligono P da cena
  - Para cada pixel (x,y) de um polígono P
    - Computar z_depth na posição x, y
    - Se z_depth < z_buffer (x,y) então
      - Defina_pixel(x, y, color)
      - Troque o valor : z_buffer (x,y) = z_depth

33

33

Exemplo: Entendendo o Z-Buffer

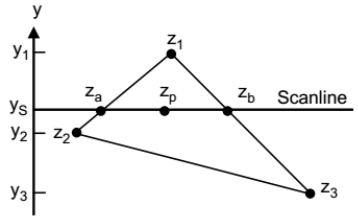


34

34

Computando a coordenada Z

- Determinar $\mathbf{z_p}$ pela interpolação de $\mathbf{z_a}$ e $\mathbf{z_b}$;
  - Equação paramétrica da reta: $P(t) = P_0 + t(P_1 - P_0)$ $0 \le t \le 1$



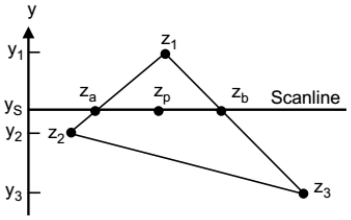
- $z_a = z_1 + t_a(z_2 - z_1)$, com $t_a = (y_s - y_1)/(y_2 - y_1)$
- $z_b = z_1 + t_b(z_3 - z_1)$, com $t_b = (y_s - y_1)/(y_3 - y_1)$
- $z_p = z_a + t_p(z_b - z_a)$, com $t_p = (x_p - x_a)/(x_b - x_a)$

35

35

Tarefa

1. Determine $\mathbf{z_p}$ sabendo que $P_1(3,7,1)$, $P_2(1,1,8)$, $P_3(5,2,5)$, $P_p(3,4,z_p)$, e o que os pixels de 2 a 4 foram selecionados para rasterização na ordenada do *scanline* atual;
   - Equação paramétrica da reta: $P(t) = P_0 + t(P_1 - P_0)$ $0 \le t \le 1$



- $z_a = z_1 + t_a(z_2 - z_1)$, com $t_a = (y_s - y_1)/(y_2 - y_1)$
- $z_b = z_1 + t_b(z_3 - z_1)$, com $t_b = (y_s - y_1)/(y_3 - y_1)$
- $z_p = z_a + t_p(z_b - z_a)$, com $t_p = (x_p - x_a)/(x_b - x_a)$

36

36