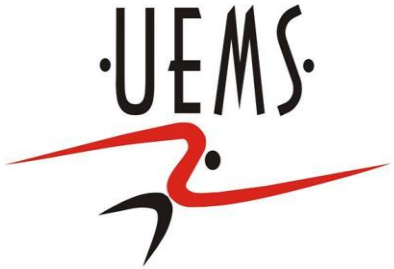


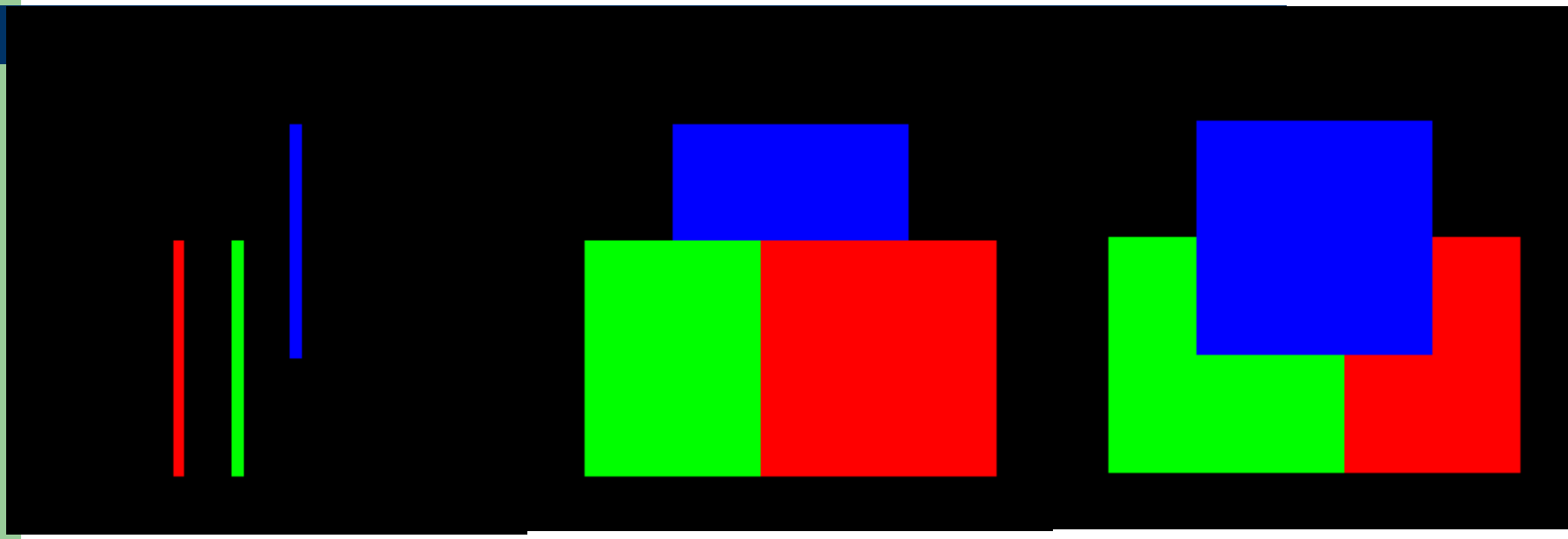
Computação Gráfica – Visibilidade

Profa. Mercedes Gonzales Márquez

Visibilidade

- O sistema de visão humana usa vários recursos para perceber a profundidade. Um deles é a oclusão de objetos, ou seja, objetos mais distantes em relação a um observador ao longo de um raio de visão são escondidos pelos objetos opacos mais próximos.

Visibilidade



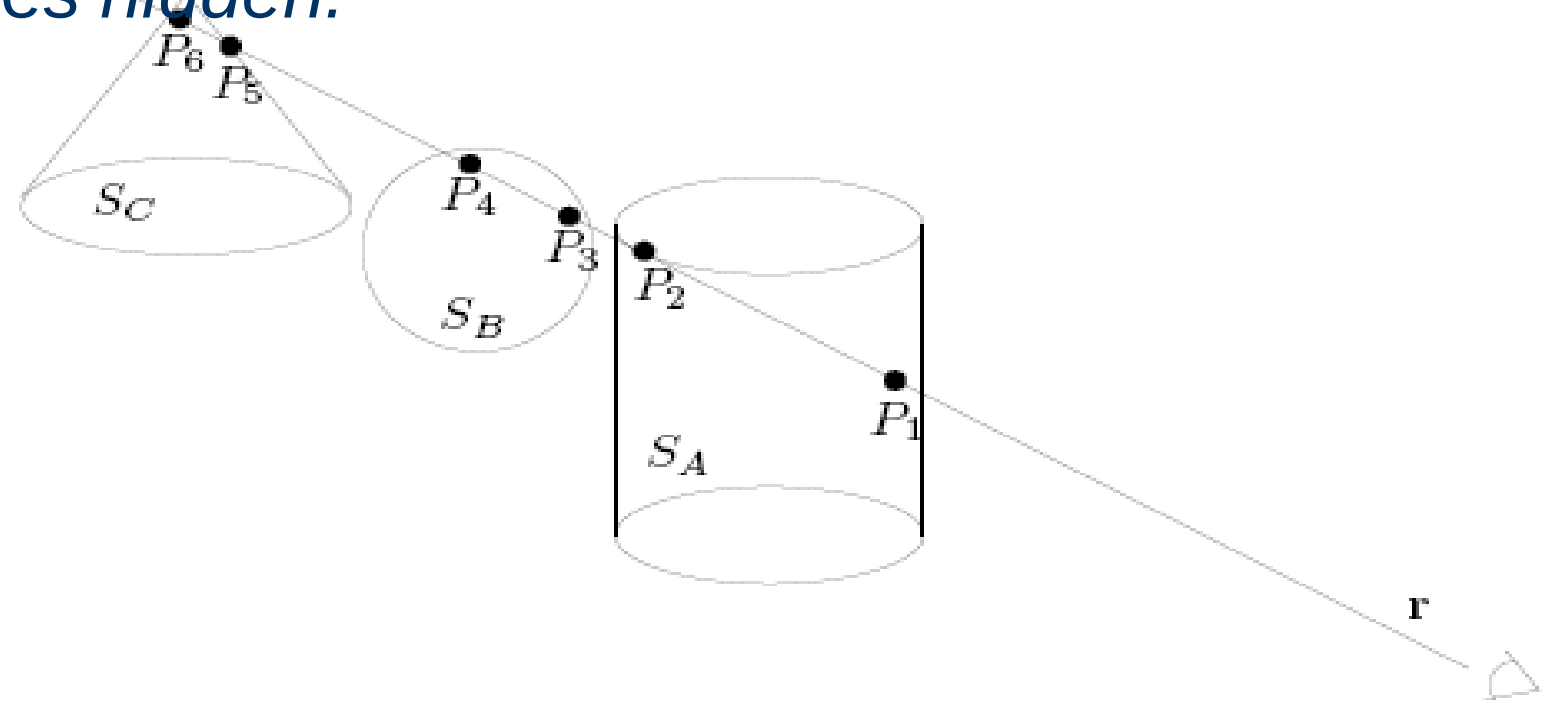
(a) Objetos rotacionados em um ângulo de 90 graus no eixo y (azul tem menor profundidade).

(b) Visualização incorreta

(c) Visualização correta

Visibilidade

- Na Figura, a superfície SA é visível em relação ao observador e as outras ficam escondidas, em inglês *hidden.*



Problema de Visibilidade

A essência dos algoritmos de visibilidade é muito simples:
Remover as partes que “não devem ser vistas” pelo computador.

Algoritmos de Visibilidade

- São classificados em três grupos:

  (a) técnicas baseadas em espaço de imagem,

  (b) técnicas baseadas em espaço do objeto ou da cena e

  (c) técnicas mistas.

Algoritmos de Visibilidade

- As técnicas baseadas no espaço de imagem tem resolução a nível de pixels.

- As baseadas em espaço da cena tem resolução a nível do espaço de representação das figuras geométricas.

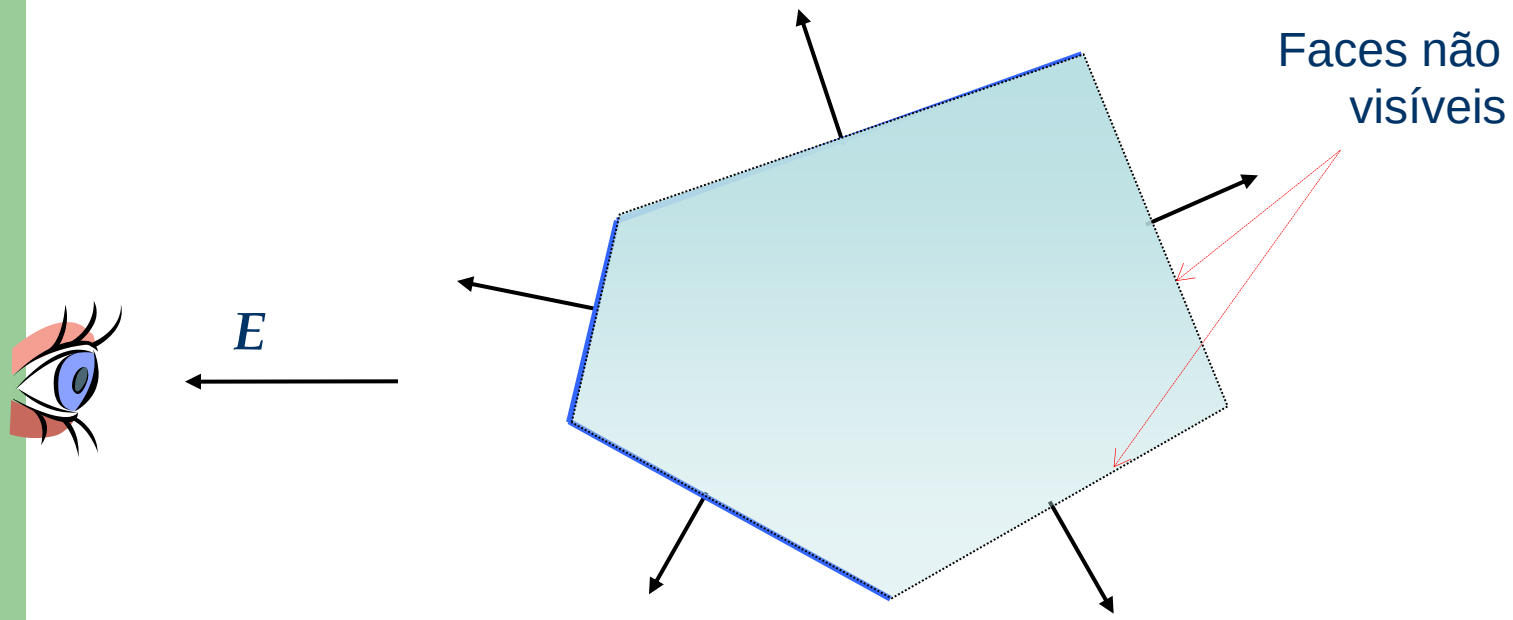
Algoritmos de Visibilidade

- Alguns métodos:
  - Remoção das faces posteriores (*backface culling*) - Técnica baseada no espaço da cena.
  - Depth-buffer (*z-buffer*) – Técnica baseada no espaço da imagem.

Backface Culling (Remoção das faces posteriores)

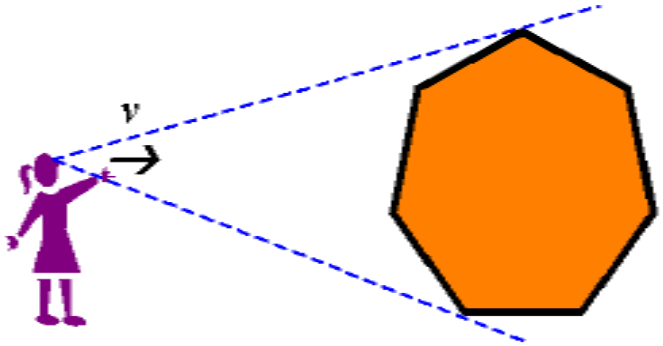
- Considere-se o caso de um objeto o qual pode ser aproximado por um poliedro sólido: então as suas faces poligonais envolvem-no completamente, ou seja, possui um volume interior fechado
- Assumindo que cada polígono constituinte é definido de forma a que a sua normal aponte para o exterior do objeto, então os polígonos cuja normal aponte na direção oposta à do observador encontram-se completamente tapados pelos restantes polígonos. Estas faces, denominada de posteriores (*backfaces*), podem ser removidas.

Backface Culling
(Remoção das faces posteriores)



Remoção das faces posteriores

- Este método pode ser aplicado em um único poliedro convexo fechado na cena.



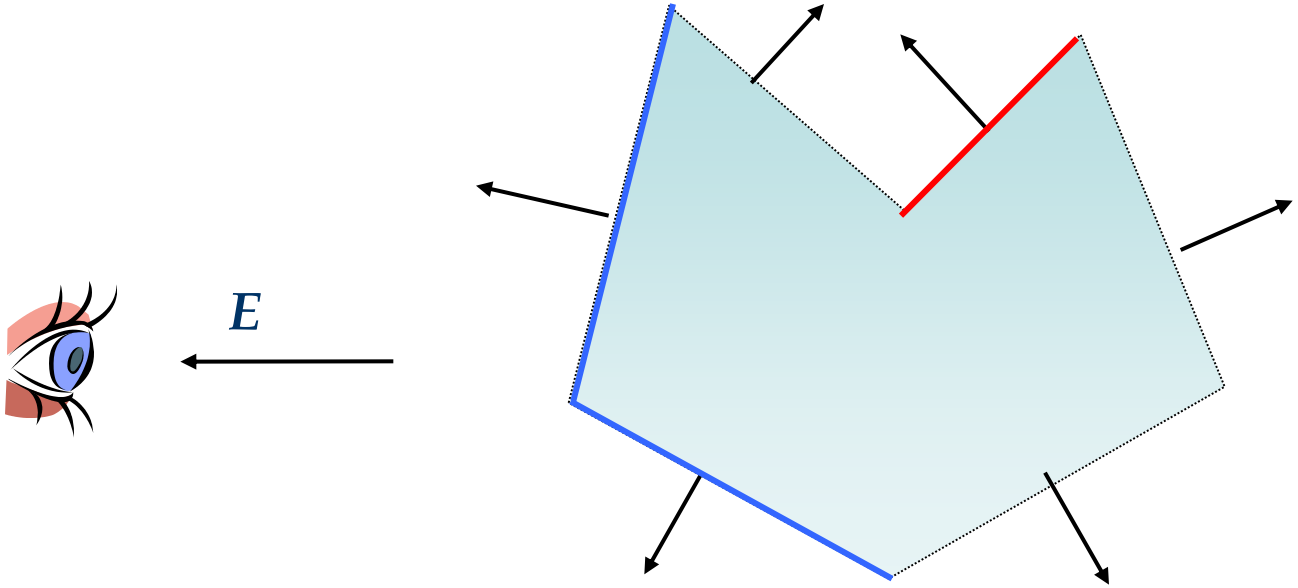
Não pode ser aplicado :

- - Em poliedros côncavos
- - ou quando existem mais objetos

(Por simplicidade estamos mostrando nas ilustrações polígonos no lugar de poliedros).

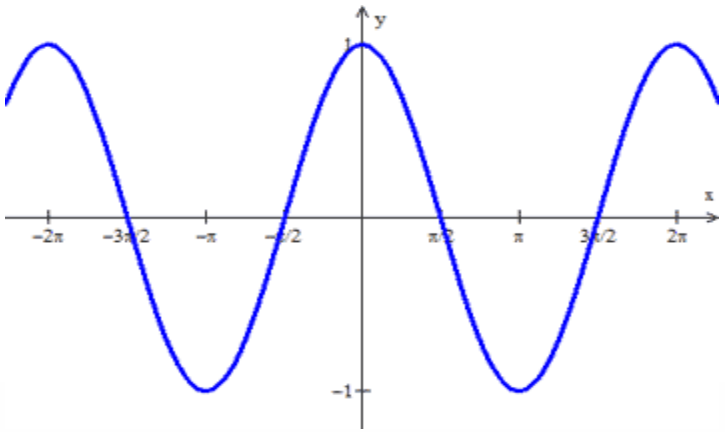
Remoção das faces posteriores

- Observe o poliedro côncavo da figura. Veja que a face pintada de vermelho tem vetor normal na mesma direção do observador e portanto o algoritmo considera erradamente como face visível ou face da frente.



Remoção das faces posteriores

- **Como determinar se a face é da frente ou visível?**
- **Face da frente –** o ângulo θ formado pelo vetor da linha de vista E e o vetor normal N é menor ou igual a 90º



Se $0^o \leq \theta \leq 90^o$, então $0 \leq \cos\theta \leq 1$

Se $90^o \leq \theta \leq 180^o$, então $-1 \leq \cos\theta \leq 0$

Dado que $\cos\theta = \dfrac{E.N}{|E||N|}$ então a fase é visível se $0 \leq \dfrac{E.N}{|E||N|} \leq 1$

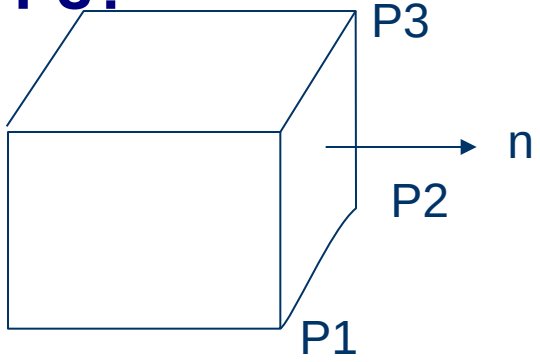
Remoção das faces posteriores

Dado que $\cos\theta = \frac{E.N}{|E||N|}$ então a fase é visível se $0 \leq \frac{E.N}{|E||N|} \leq 1$ ou $E.N \geq O$

- **Resumindo**
  - $E.N \geq O \rightarrow$ Face da frente
  - $E.N < O \rightarrow$ Face de trás

Remoção das faces posteriores

- **Como determinar o vetor normal n da face com pontos P1, P2 e P3?**



$$n = \vec{u} \times \vec{v}, onde\ u = \overline{P2P3}\ e\ v = \overline{P2P1}$$

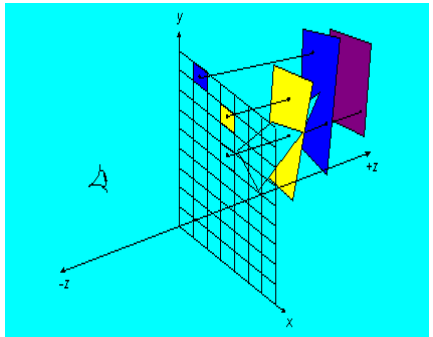
$$n_x = (P3_y - P2_y)(P1_z - P2_z) - (P1_y - P2_y)(P3_z - P2_z)$$
$$n_y = (P3_z - P2_z)(P1_x - P2_x) - (P1_z - P2_z)(P3_x - P2_x)$$
$$n_z = (P3_x - P2_x)(P1_y - P2_y) - (P1_x - P2_x)(P3_y - P2_y)$$

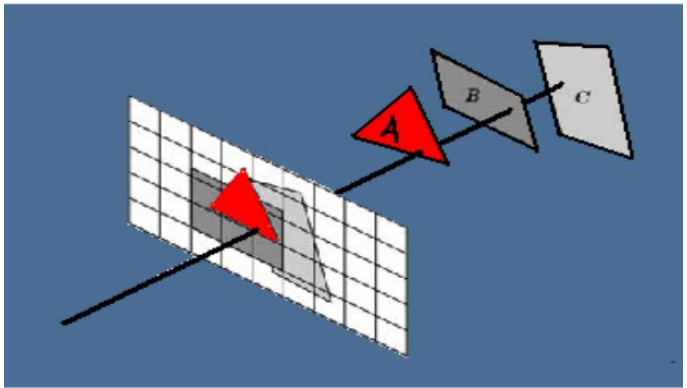
Z-Buffer

- Método que opera no espaço da imagem. Precisa de duas áreas de memória, uma para a construção da imagem (designada por *frame-buffer*) e outra para armazenamento de profundidades Z(designada por *z-buffer*).
- Ele consiste em colorir cada pixel com a cor da face que estiver mais próxima do observador (menor profundidade) em dois passos:

  1. Cada face do cenário é rasterizada e

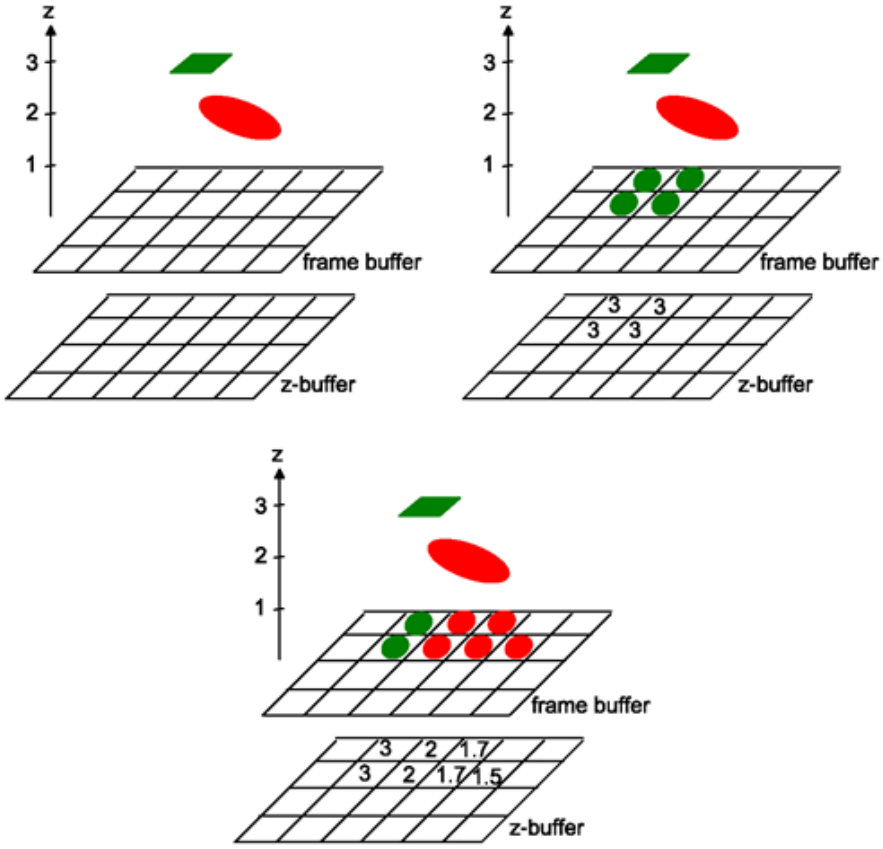
  2. cada pixel (produto da rasterização) recebe a cor do objeto mais próximo ao observador.





Z-Buffer - Algoritmo

- Ao início fazemos:
  - *z-buffer* = profundidade máxima
  - *Buffer* de cor = cor de fundo
- Durante a rasterização de cada polígono, cada pixel passa por um *teste de profundidade*
  - Se a profundidade do pixel for menor que a registrada no *z-buffer*
    - Pintar o pixel (atualizar o buffer de cor)
    - Atualizar o buffer de profundidade
  - Caso contrário, ignorar



Z-Buffer

- OpenGL:
  - Habilitar o z-buffer:

    glEnable (GL_DEPTH_TEST);
  - Não esquecer de alocar o z-buffer → GLUT_DEPTH
    - Número de bits por pixel depende de implementação / disponibilidade de memória
  - Ao gerar um novo quadro, limpar também o z-buffer:

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
  - Ordem imposta pelo teste de profundidade pode ser alterada

    glDepthFunc(…)
- Veja os exemplos em OpenGL: hidplanes.c, planes.c e triangles.c