

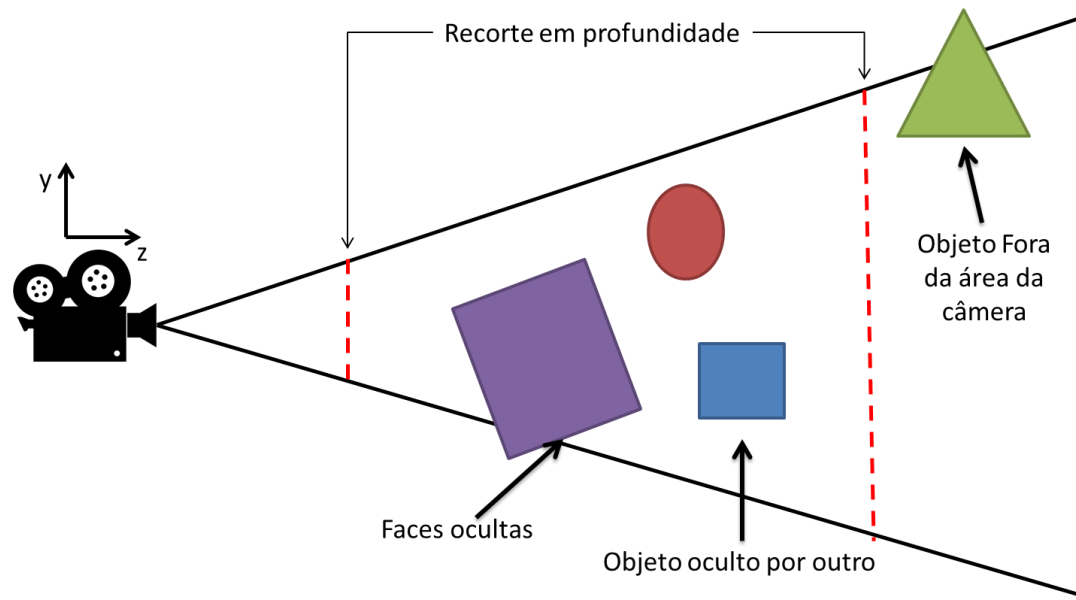
# ELIMINAÇÃO DE ELEMENTOS OCLUSOS

Prof. Dr. Bianchi Serique Meiguins

Prof. Dr. Carlos Gustavo Resque dos Santos

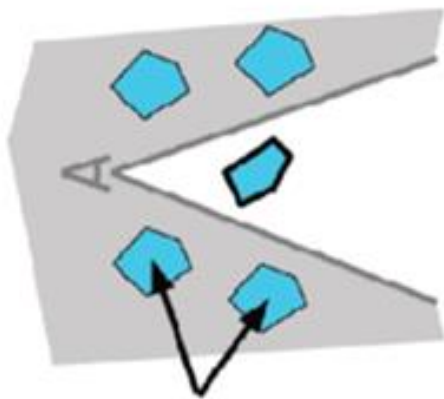
# Motivação

- Existem quatro casos base:
  - **Eliminação de faces oclusas (*backface*)**
  - **Eliminação de objetos oclusos**
  - **Eliminação por campo de visão**
  - **Eliminação de superfícies oclusas**

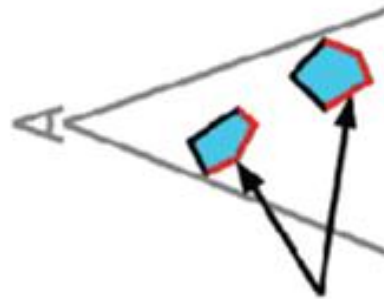


# Tipos

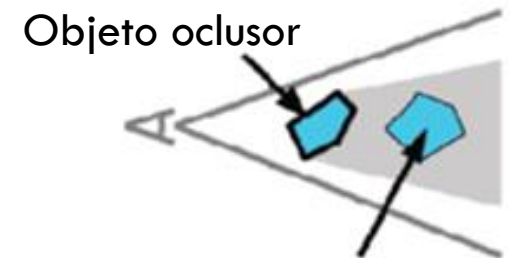
- Existem quatro casos base:
  - **Eliminação de faces oclusas (*backface*)**
  - **Eliminação de objetos oclusos**
  - **Eliminação por campo de visão**
  - **Eliminação de superfícies oclusas**



Fora do campo  
de visão



Fases de costa



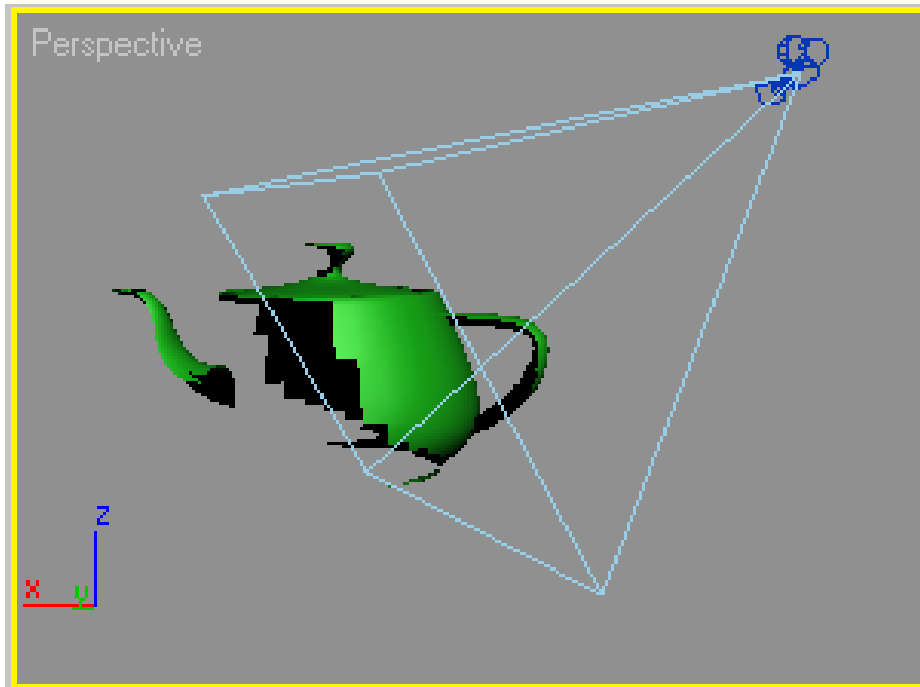
Objeto ocluido

ELIMINAÇÃO DE FACES  
OCLUSAS (BACKFACE)

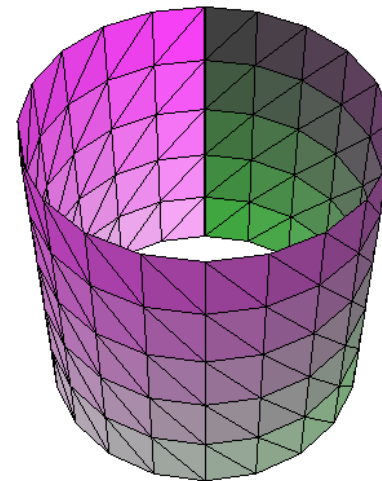
# Eliminação de Faces Oclusas

- Objetiva Eliminar faces ocultas por faces do mesmo objeto.
  - Na prática as faces que estão de costa para a câmera são eliminadas
  - Úteis para objetos fechados
  - Reduz  $\sim 50\%$  de faces de um objeto

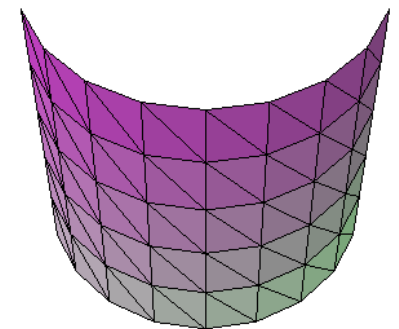
# Eliminação de Faces Oclusas



Cuidado com os objetos  
que não estão  
totalmente fechados



**off**



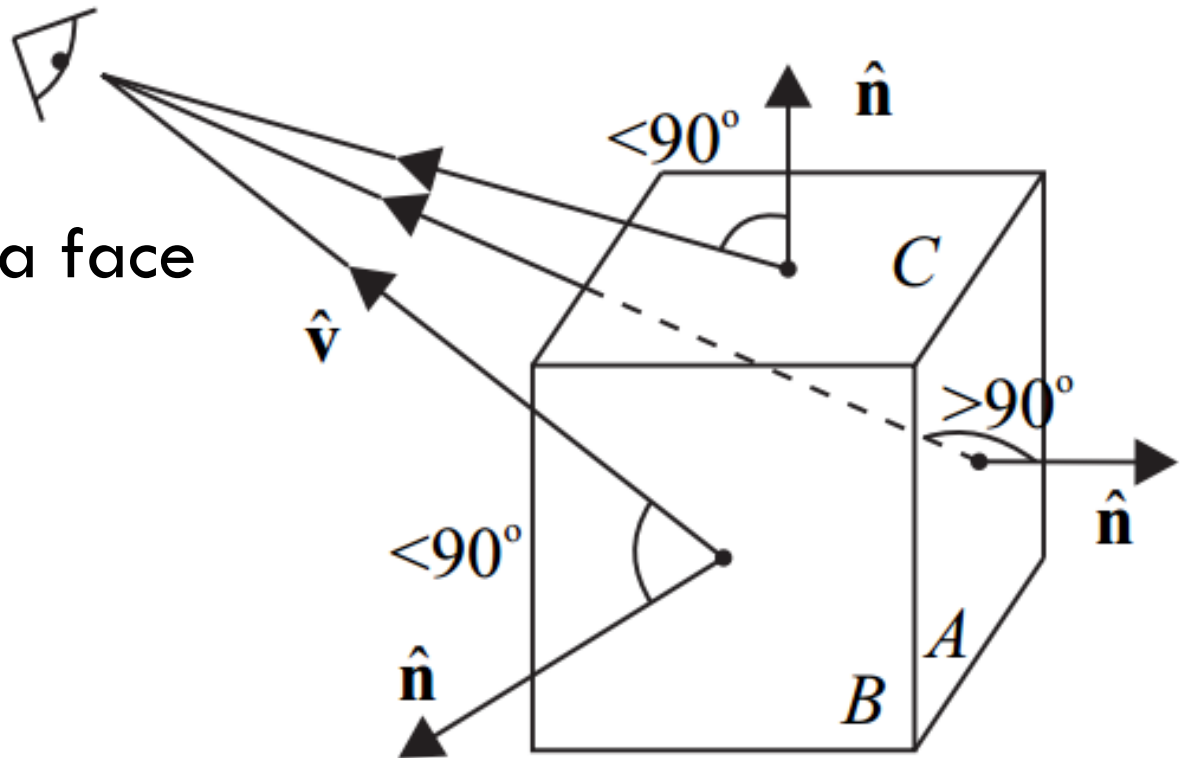
**on**

# Eliminação de Faces Oclusas

□ Se

□  $\hat{v} \cdot \hat{n} < 0$

□ Então descarte a face



Obs: “.” é o produto interno  
ou produto escalar.

\*Veja em Álgebra Linear

# Eliminação de Faces Oclusas

## □ Cálculo da Normal:

- A normal pode ser obtida pelo produto vetorial de dois vetores pertencentes ao plano.

- O produto vetorial pode ser calculado pelo determinante da matriz  $M$ :

$$\blacksquare \hat{v} \times \hat{w} = \det(M) = \begin{vmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix}$$

Sendo:

$$\hat{v} = (v_1, v_2, v_3)$$

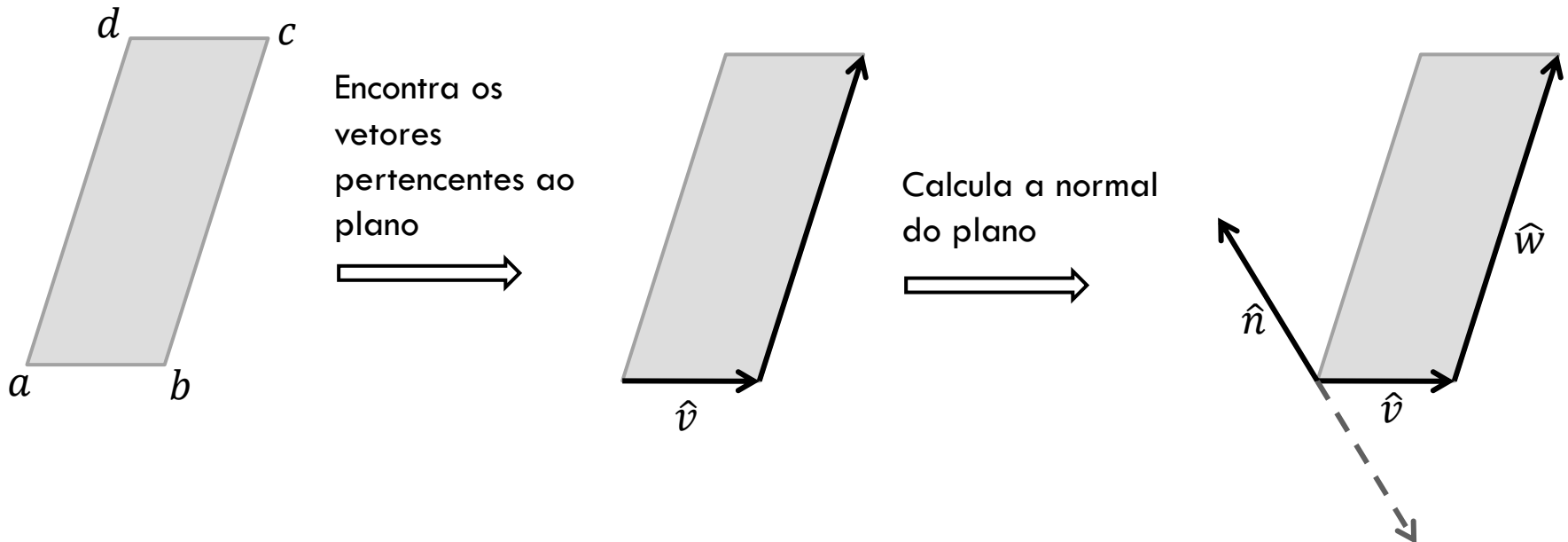
$$\hat{w} = (w_1, w_2, w_3)$$

$$\blacksquare \hat{v} \times \hat{w} = (v_2w_3 - v_3w_2, v_3w_1 - v_1w_3, v_1w_2 - v_2w_1)$$



# Eliminação de Faces Oclusas

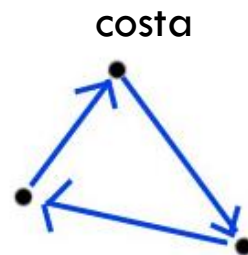
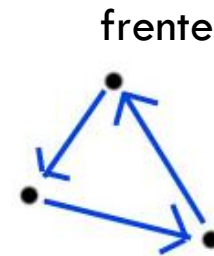
## □ Cálculo da Normal:



# Eliminação de Faces Oclusas

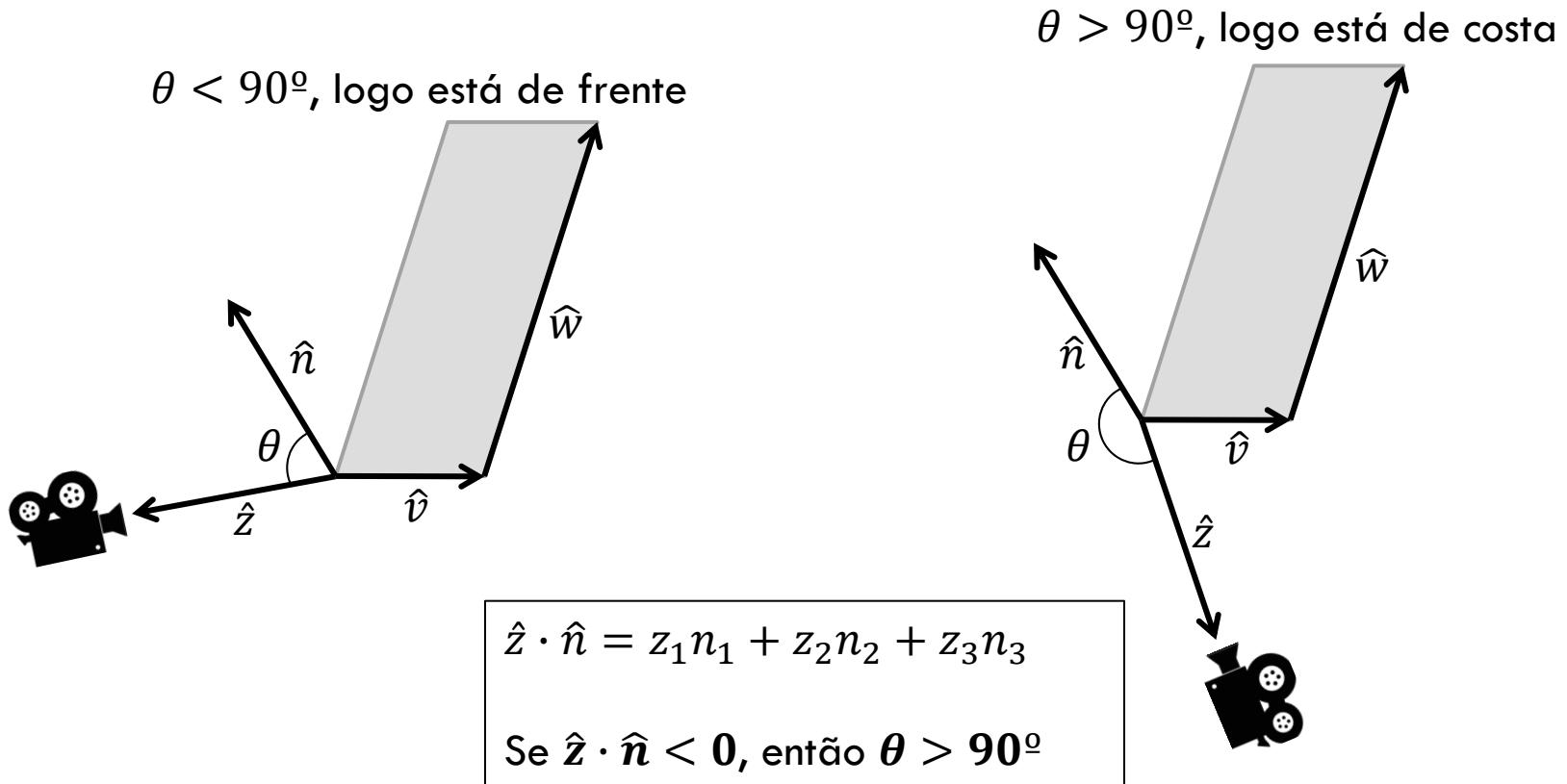
- O produto vetorial é anticomutativo, ou seja:
  - $\hat{v} \times \hat{w} = -\hat{w} \times \hat{v}$
- Esta propriedade “decide” a direção da normal
- Em termos de computação gráfica.
  - decide a frente da face (plano)

- Dica: use a regra da mão direita



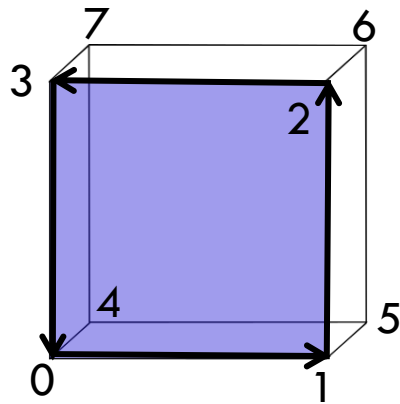
# Eliminação de Faces Oclusas

- Agora basta verificar a condição de  $90^\circ$

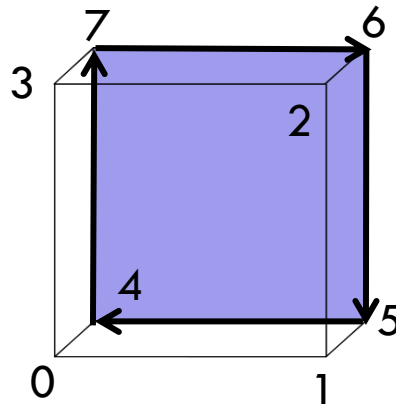


# Faces do Cubo

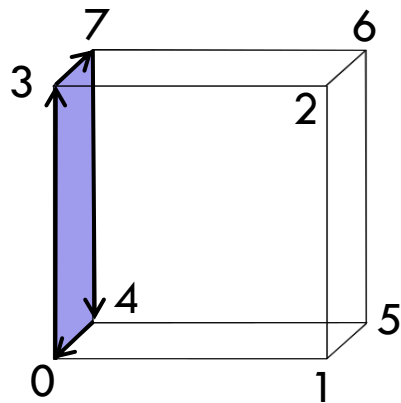
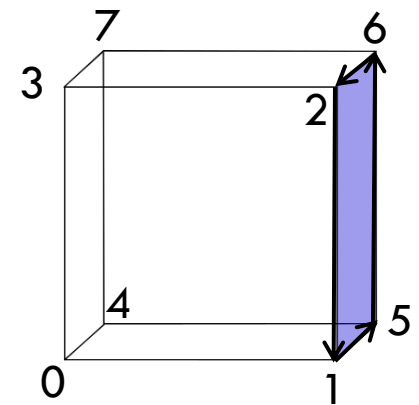
**0-1-2-3-0**



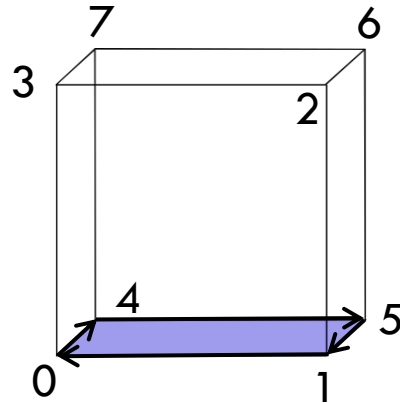
**4-7-6-5-4**



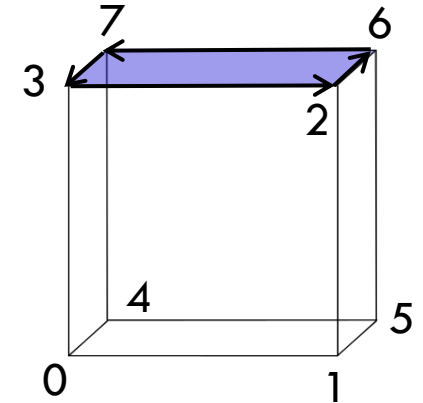
**1-5-6-2-1**



**0-3-7-4-0**



**0-4-5-1-0**

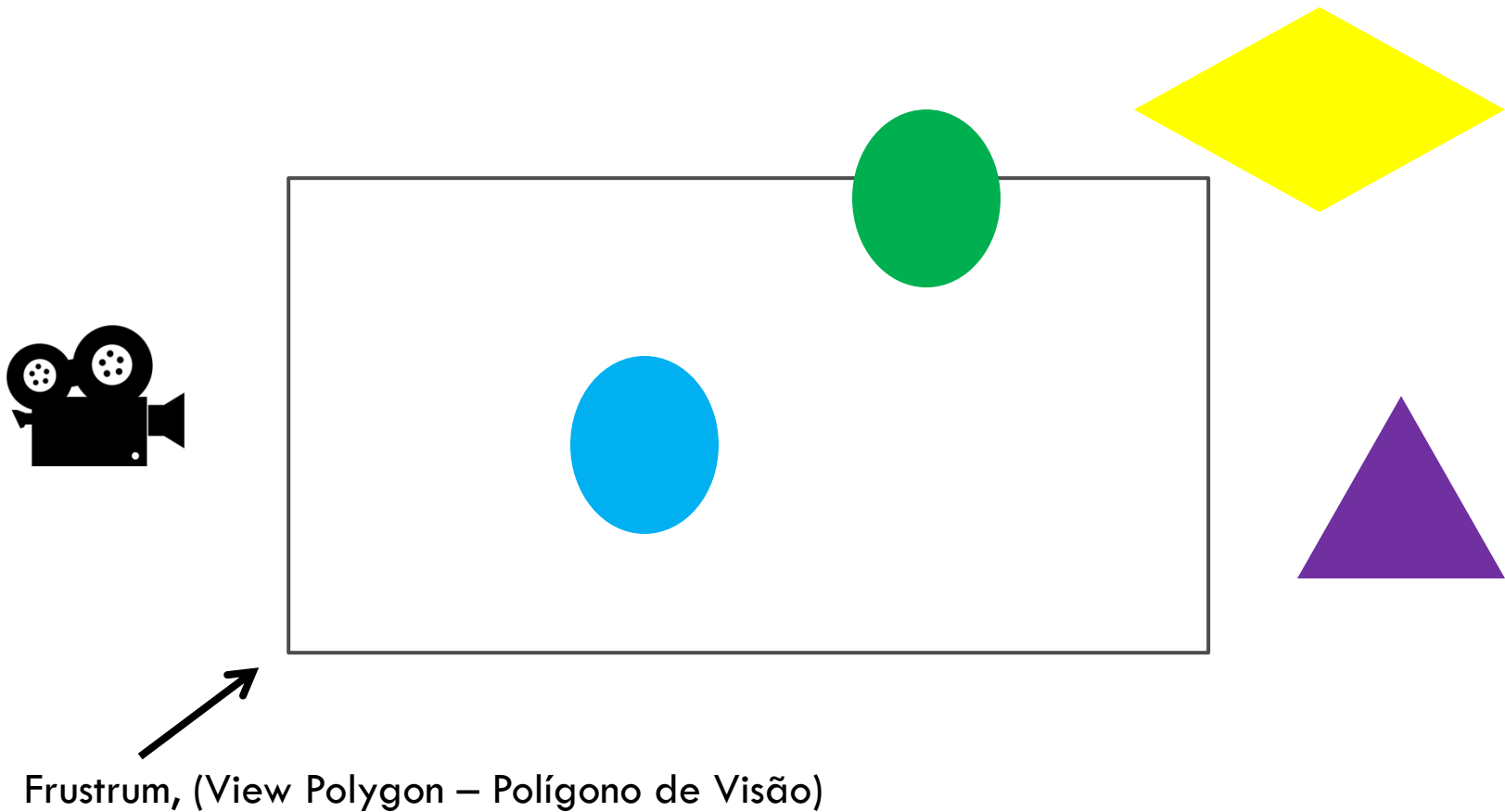


**2-6-7-3-2**

ELIMINAÇÃO POR CAMPO DE  
VISÃO (FRUSTUM CULLING)

# Recorte 3D (*Frustum Culling*)

- ❑ Elimina os objetos que não estão no alcance da câmera.



# Recorte 3D (*Frustrum Culling*)

- Os algoritmos de recorte 2D podem ser expandidos para 3D.
  - ▣ Interseção entre linha e plano
  - ▣ Teste de contém
- Pode ser realizado de forma hierárquica, evitando a comparação desnecessárias com objetos complexos.

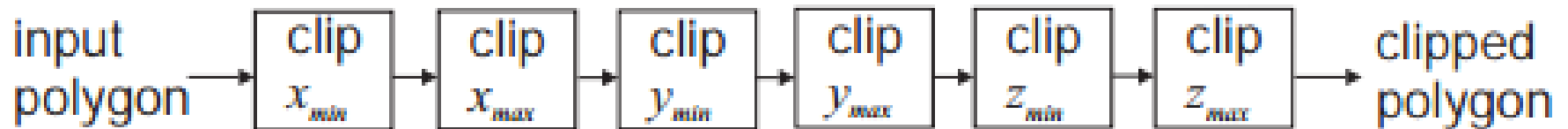
# Recorte 3D (*Frustum Culling*)

- Exemplo: Cohen-Sutherland
  - ▣ 1° bit: 1 se  $z > z_{\max}$ , 0 caso contrário
  - ▣ 2° bit: 1 se  $z < z_{\min}$ , 0 caso contrário
  - ▣ 3° bit: 1 se  $y > y_{\max}$ , 0 caso contrário
  - ▣ 4° bit: 1 se  $y < y_{\min}$ , 0 caso contrário
  - ▣ 5° bit: 1 se  $x > x_{\max}$ , 0 caso contrário
  - ▣ 6° bit: 1 se  $x < x_{\min}$ , 0 caso contrário



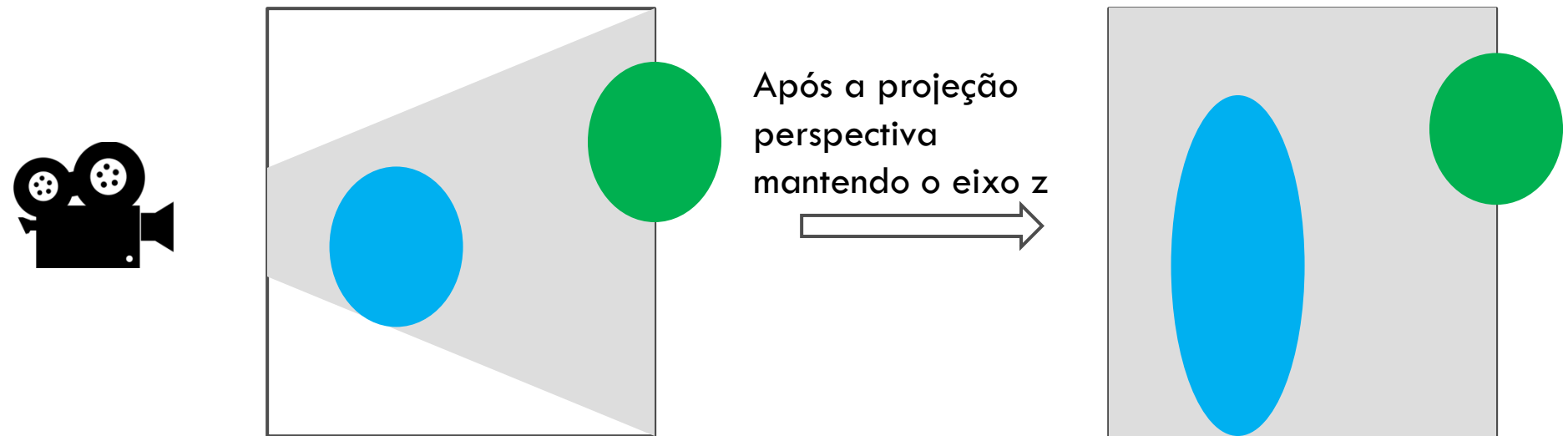
# Recorte 3D (*Frustum Culling*)

## □ Exemplo 2: Sutherland-Hodgman



# Recorte 3D (*Frustrum Culling*)

- No caso da câmera perspectiva, é possível aplicar a projeção e manter o valor original do eixo z (profundidade)
- Assim, podem ser utilizados testes simples na etapa de recorte.



# Exemplos (Vídeos)

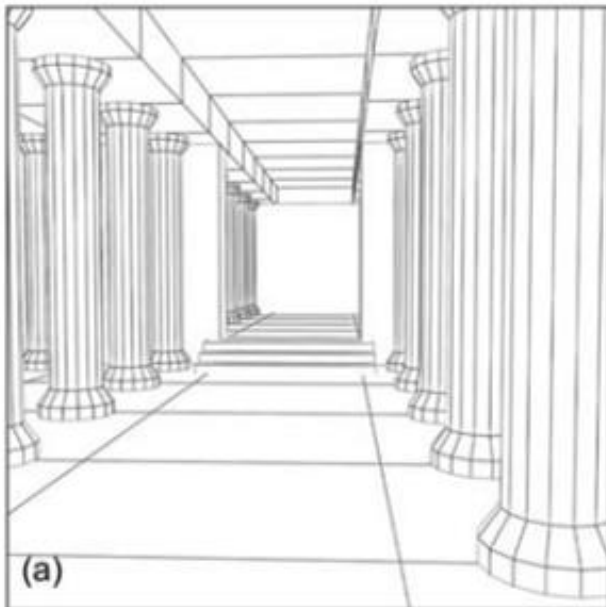
- <https://www.youtube.com/watch?v=tHrJlCq1e3Q>
- <https://www.youtube.com/watch?v=Y7qVYPBmGz4>
- <https://www.youtube.com/watch?v=XBYPWh-Ukcfk>
- <https://www.youtube.com/watch?v=XLXJY-bk82U>
- <https://www.youtube.com/watch?v=9l4lwkkh7sg>

ELIMINAÇÃO OBJETOS  
OCLUSOS

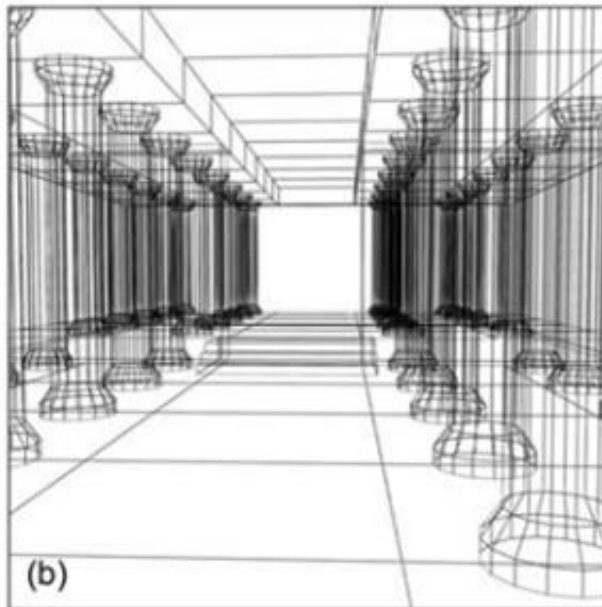
# Eliminação de Objetos Oclusos

- Elimina os objetos totalmente oclusos por outros objetos

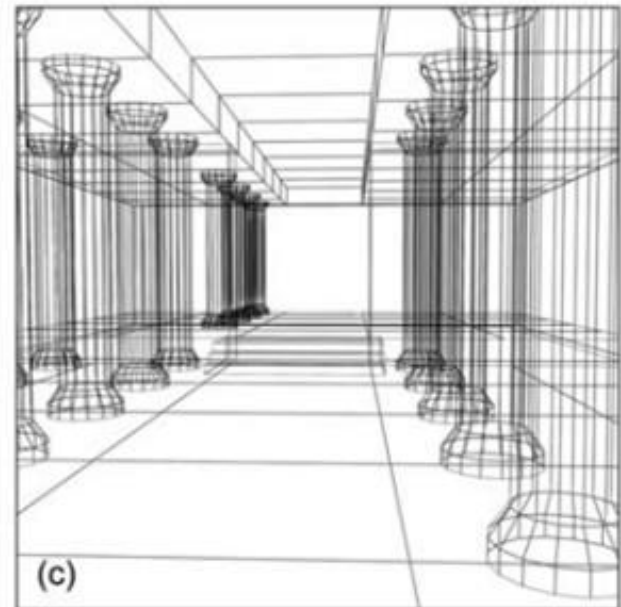
Faces



Off



On

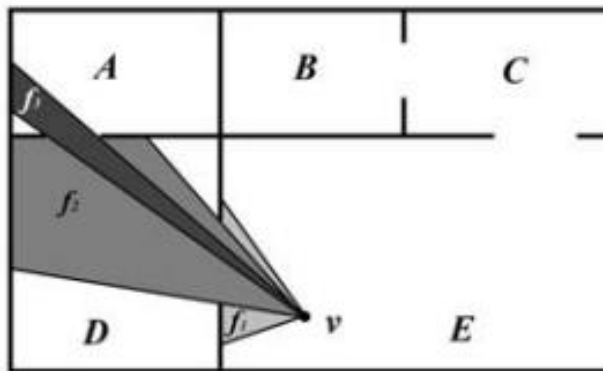


# Eliminação de Objetos Oclusos

- ❑ O objetivo é minimizar a quantidade de cálculo necessário na etapa de Eliminação de Superfície
- ❑ É eficiente em tempo de processamento
- ❑  $O(P)$ , sendo  $P$  a quantidade de polígonos

# Eliminação de Objetos Ocluídos

## □ Ambiente Indoor:



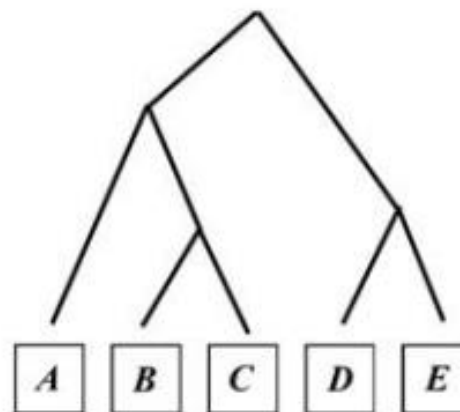
(a)



(b)

	A	B	C	D	E
A	1			1	1
B		1	1		1
C		1	1	1	1
D	1		1	1	1
E	1	1	1	1	1

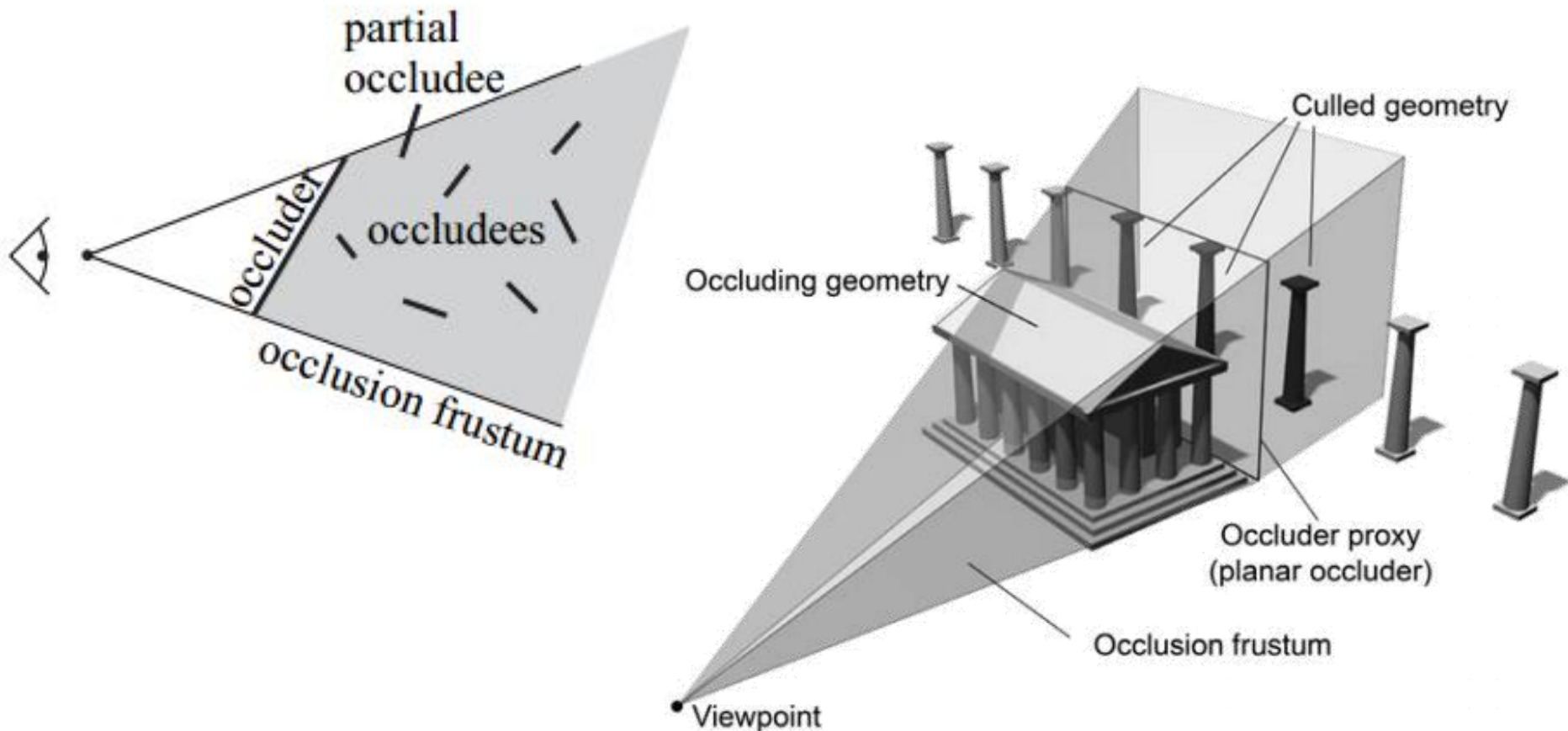
(c)



(d)

# Eliminação de Objetos Ocluídos

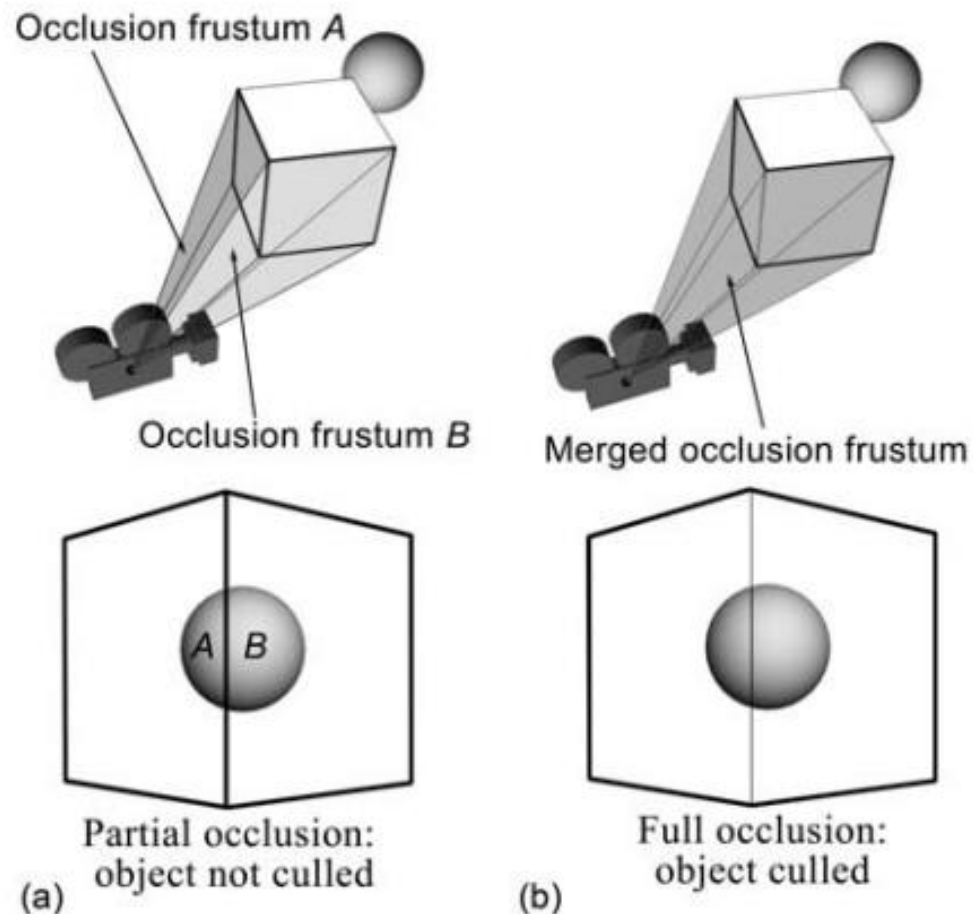
## □ Ambiente Outdoor:





# Eliminação de Objetos Ocluídos

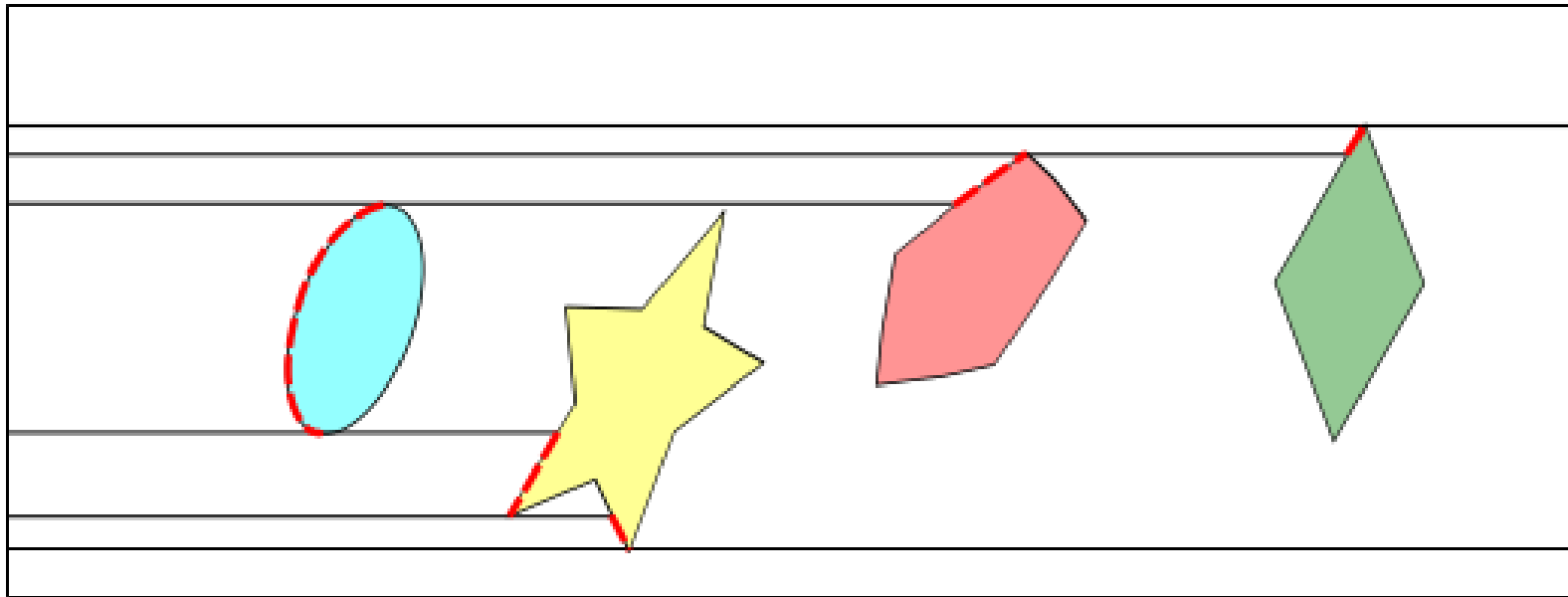
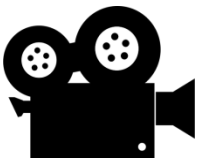
## □ Ambiente Outdoor:



# ELIMINAÇÃO SUPERFÍCIES OCLUSAS

# Eliminação de Superfícies Oclusas

- É necessário cortar superfícies oclusas para poupar processamentos de cor, iluminação, textura, etc.



# Z-Buffer

- Algoritmo clássico na área de computação gráfica.
- Trabalha no espaço da imagem digital
  - ▣ Tem a mesma ideia que o frame buffer, só que ao invés de cores são armazenados os valores de  $Z$  (profundidade).
- Este algoritmo tira proveito da coerência das primitivas (planos  $\rightarrow$  triângulos) para calcular o  $Z$  de forma incremental.

# Z-Buffer

$$F'(x + 1, y) = z = -\frac{d}{c} - \frac{a}{c}(x + 1) - \frac{b}{c}y$$

- Fórmula do plano.

- ▣  $F(x, y, z) = ax + by + cz + d = 0$

- Como estamos interessados no  $z$

- ▣  $F'(x, y) = z = -\frac{d}{c} - \frac{a}{c}x - \frac{b}{c}y$

- Aplicando a recursividade

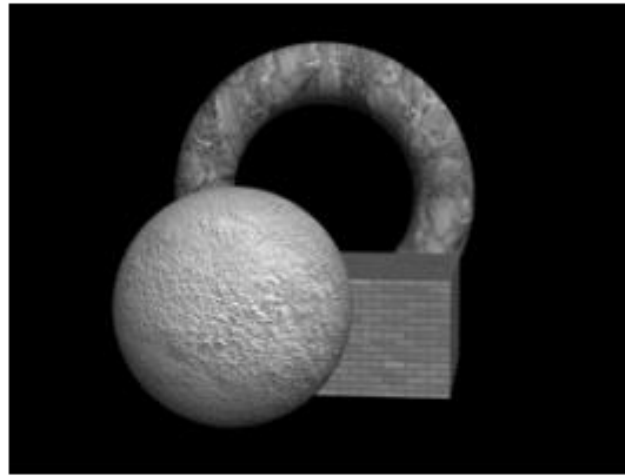
- ▣  $F'(x + 1, y) = F'(x, y) - \frac{a}{c}$

- ▣  $F'(x, y + 1) = F'(x, y) - \frac{b}{c}$

# Z-Buffer

- Inicializa  $z\_buffer$  com  $Frustrum_{z\_max}$
- Para cada pixel do Frame  $\rightarrow x, y$ 
  - Para cada polígono  $\rightarrow p$ 
    - $z_p = incrementaZ(p, x, y)$  //Igual ao bresenham (3D)
    - Se  $z_p < z\_buffer[x, y]$ 
      - $z\_buffer[x, y] = z_p$

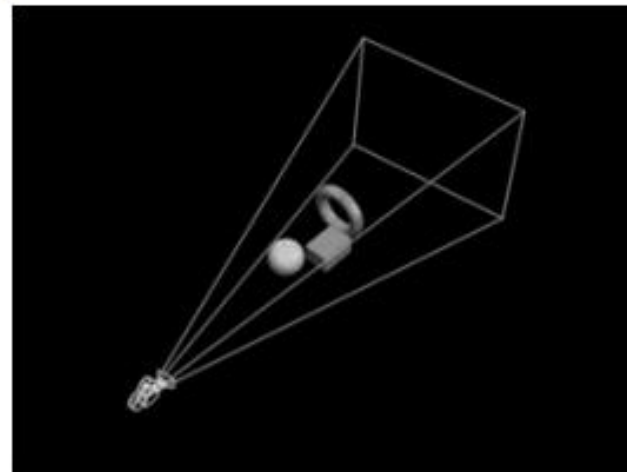
# Z-Buffer



(a)



(b)



(c)

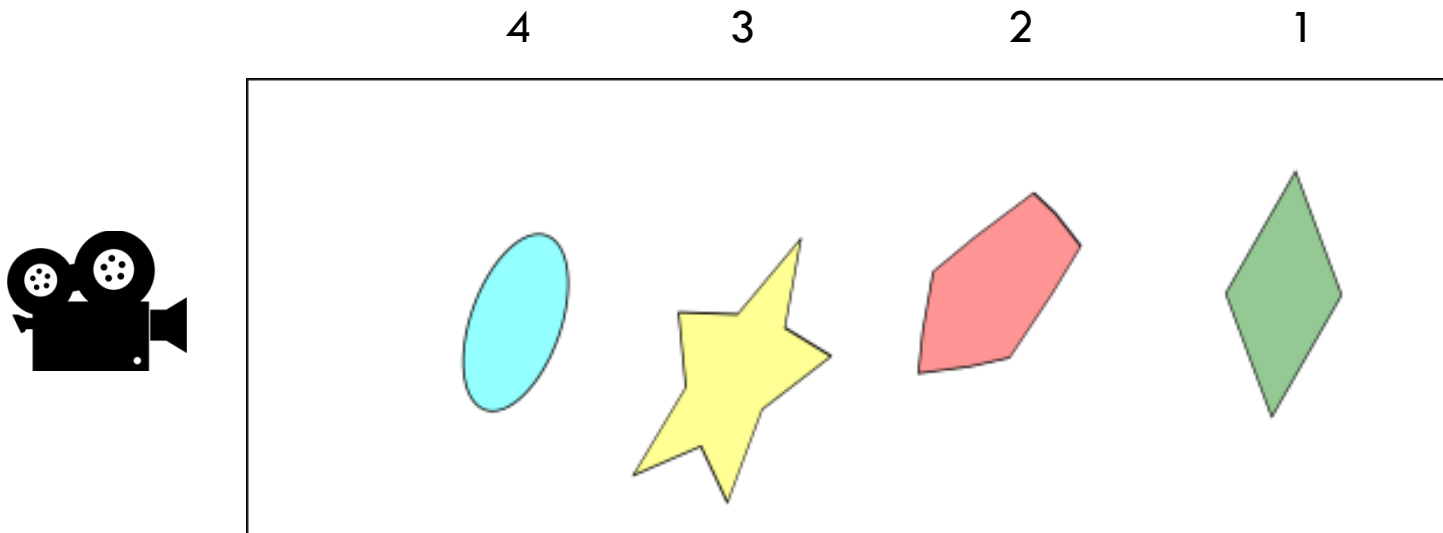
(a) Frame Buffer

(b) Z-Buffer

(c) Cena

# Algoritmo de ordenação em Profundidade

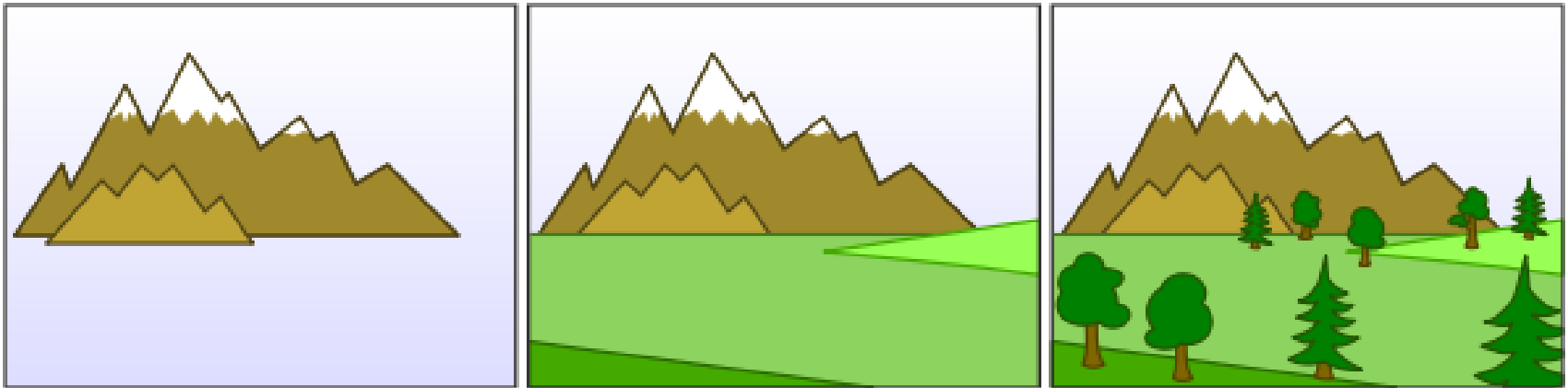
- Ordena os objetos em relação a sua distância do observador e pinta na ordem inversa.
- Também é conhecido como o algoritmo do pintor.





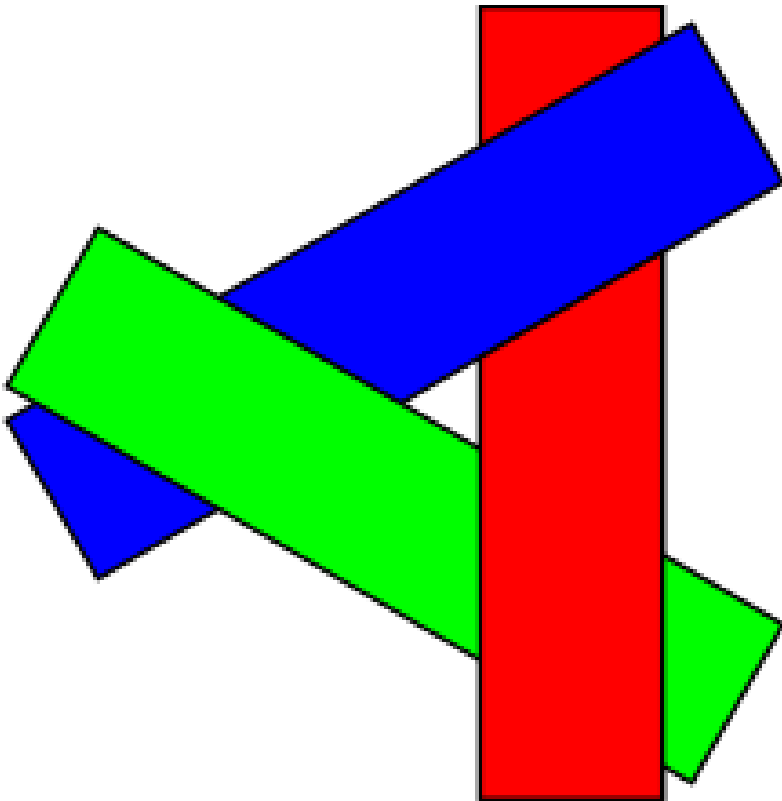
# Algoritmo de ordenação em Profundidade

- Analogia do pintor



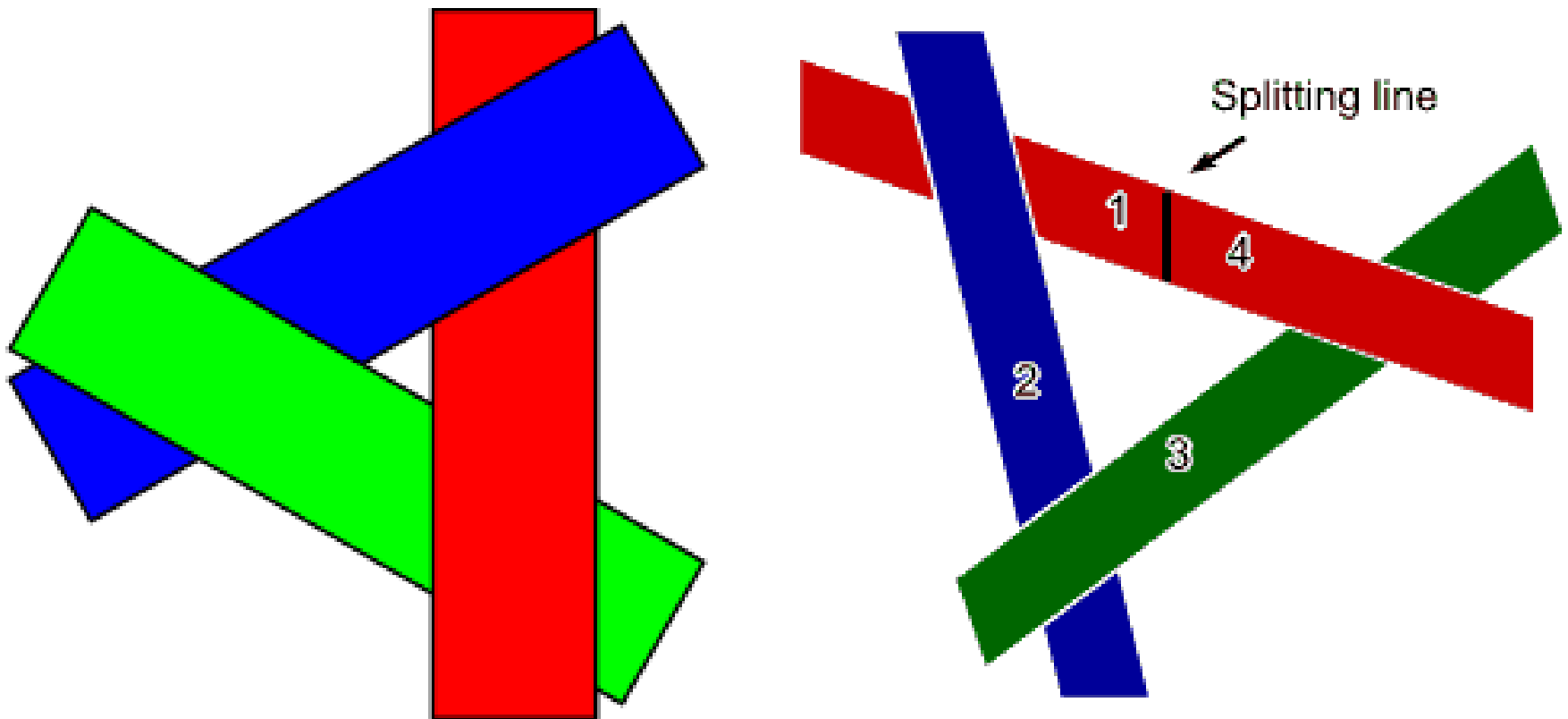
# Algoritmo de ordenação em Profundidade

□ Problema....



# Algoritmo de ordenação em Profundidade

- Problema.... Uma solução.



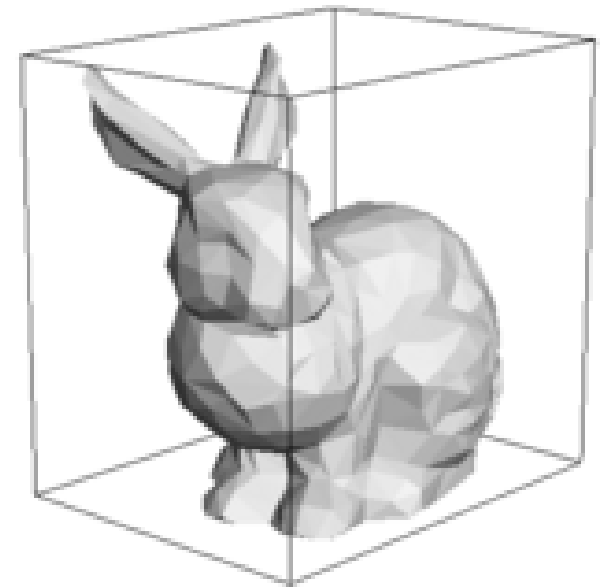
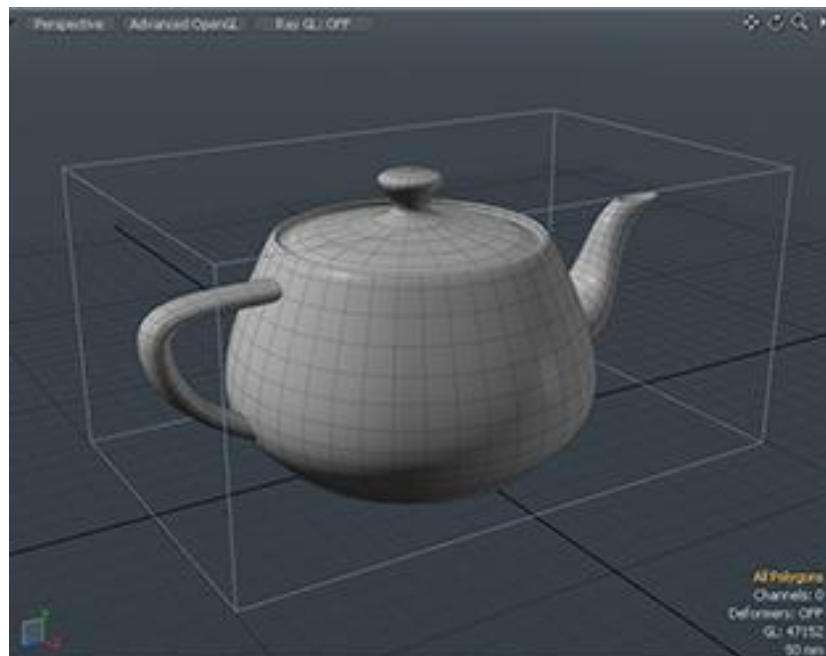
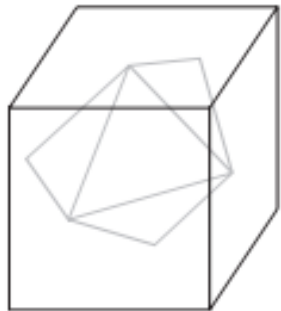
# Algoritmo de ordenação em Profundidade

- Para cada polígono
  - ▣ Encontrar o  $Z_{\min}$  e o  $Z_{\max}$  de cada  $p$
- Ordenar de acordo com o  $Z_{\min}$
- Resolver sobreposições entre  $Z_{\min}$  e  $Z_{\max}$  para cada par de polígonos
- Mostrar polígonos na ordem inversa

# Extras

- Melhorias no desempenho:

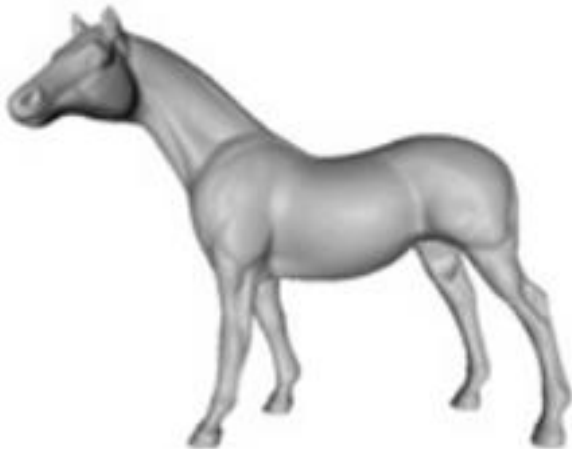
- ▣ Bounding Boxes:



# Extras

- Melhorias no desempenho:
  - ▣ Progressive Hull:

96.699 polígonos



2.000 polígonos



200 polígonos

