



# Computação Gráfica I

**CRAb – Grupo de Computação  
Gráfica**

**Departamento de Computação  
UFC**

# Objetivos

- **Estudar**
  - **equipamentos, técnicas de programação e conceitos matemáticos**
- **Para**
  - **representação, manipulação e projeção de objetos bi- e tridimensionais**
  - **aplicar a problemas específicos**

# Sumário do Curso

- **Sistemas Gráficos e Modelos**
- **Programação Gráfica**
- **Input e Interação**
- **Objetos Geométricos e Transformações**
- **Visualização**
- **Pintura**
- **Técnicas Discretas**
- **Implementação de um Renderizador**

# **2. Programação Gráfica**

**2.1 O Selante (Gasket) de Sierpinski (A4)**

**2.2 Programando Aplicações 2D (A4)**

**2.2.1 Sistemas de coordenadas**

**2.3 A API OpenGL (A4)**

**2.3.1 Funções gráficas**

**2.3.2 A Pipeline gráfica e máquinas de estado**

**2.3.3 A interface OpenGL**

## 2. Programação Gráfica

### 2.4 Primitivas e Atributos (A5)

**2.4.1 Básico sobre polígonos**

**2.4.2 Tipos de polígonos no OpenGL**

**2.4.3 Desenhando uma esfera**

**2.4.4 Texto**

**2.4.5 Objetos curvos**

**2.4.6 Atributos**

# **2. Programação Gráfica**

## **2.5 Cores (A5)**

### **2.5.1 Cor RGB**

### **2.5.2 Cor indexada**

### **2.5.3 Definição de atributos de cores**

## **2.6 Visualização (A6)**

### **2.6.1 Visualização bidimensional**

### **2.6.2 A visualização ortográfica**

### **2.6.3 Modos de matrizes**

## 2. Programação Gráfica

### 2.7 Funções de Controle (A6)

2.7.1 Interação com o sistema de janelas

2.7.2 Razão de Aspecto e Viewports

2.7.3 As funções main, display e myinit

2.7.4 Estrutura de um programa

### 2.8 O Programa Gasket (A6)

### 2.9 Polígonos e recursão (A6)

## **2. Programação Gráfica**

### **2.10 O Gasket Tridimensional (A6)**

**2.10.1 Uso de pontos 3D**

**2.10.2 Uso de polígonos em 3D**

**2.10.3 Remoção de superfícies ocultas**

**(A7) Revisão para Prova (23/08/2005)**

**(A8) Prova sem consulta (25/08/2005)**

## 2.1 O Selante (Gasket) de Sierpinski

- Usado para ilustrar a estrutura de um programa em Computação Gráfica
- Processo construtivo
  - (i) Defina 3 vértices no plano xy:  
 $V_1(x_1, y_1), V_2(x_2, y_2), V_3(x_3, y_3)$
  - (ii) Selecione um ponto,  $P_i$ , dentro do triângulo
  - (iii) Selecione aleatoriamente um dos 3 vértices,  $V_j$

## 2.1 O Selante (Gasket) de Sierpinski

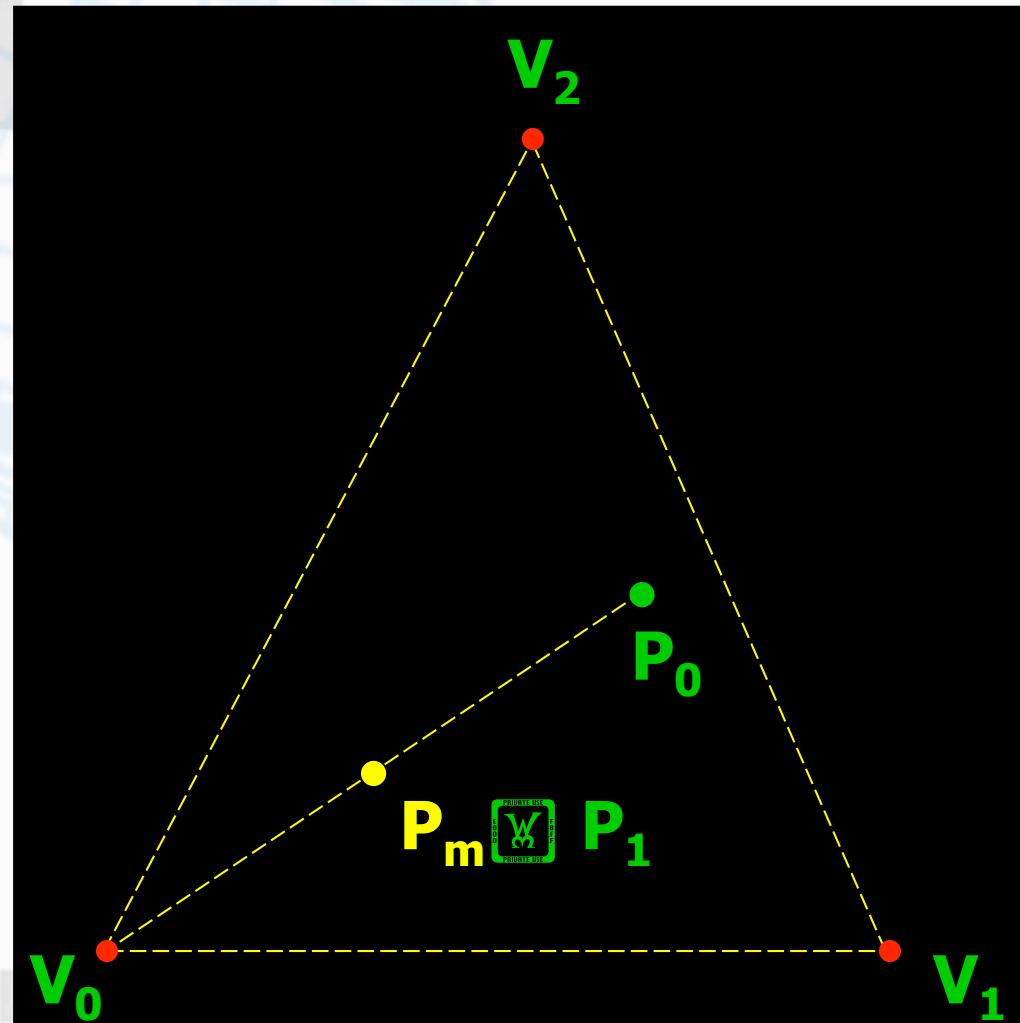
**(iv) Calcule o ponto médio do segmento  $V_jP_i$ :**

$$P_m = (V_j + P_i)/2$$

**(v) Ponha um marcador (pequeno círculo) centrado em  $P_m$**

**(vi) Considere  $P_m$  como o novo  $P_i$  e retorne ao passo (iii)**

## 2.1 O Selante (Gasket) de Sierpinski



Canvas

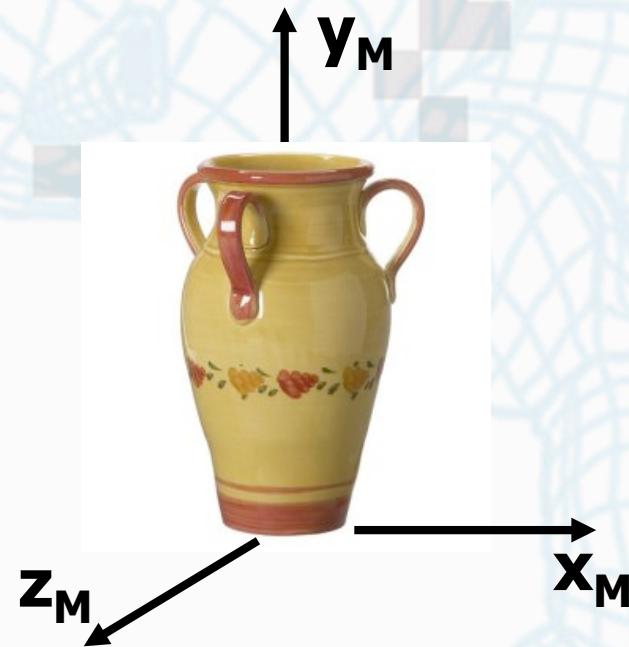
## 2.2 Programando Aplicações 2D

### 2.2.1 Sistemas de coordenadas

- Objetos e Cenários 3D sofrem transformação de mudança de sistemas de coordenadas
- Sistemas de coordenadas são referências para posicionamento e orientação de objetos no espaço

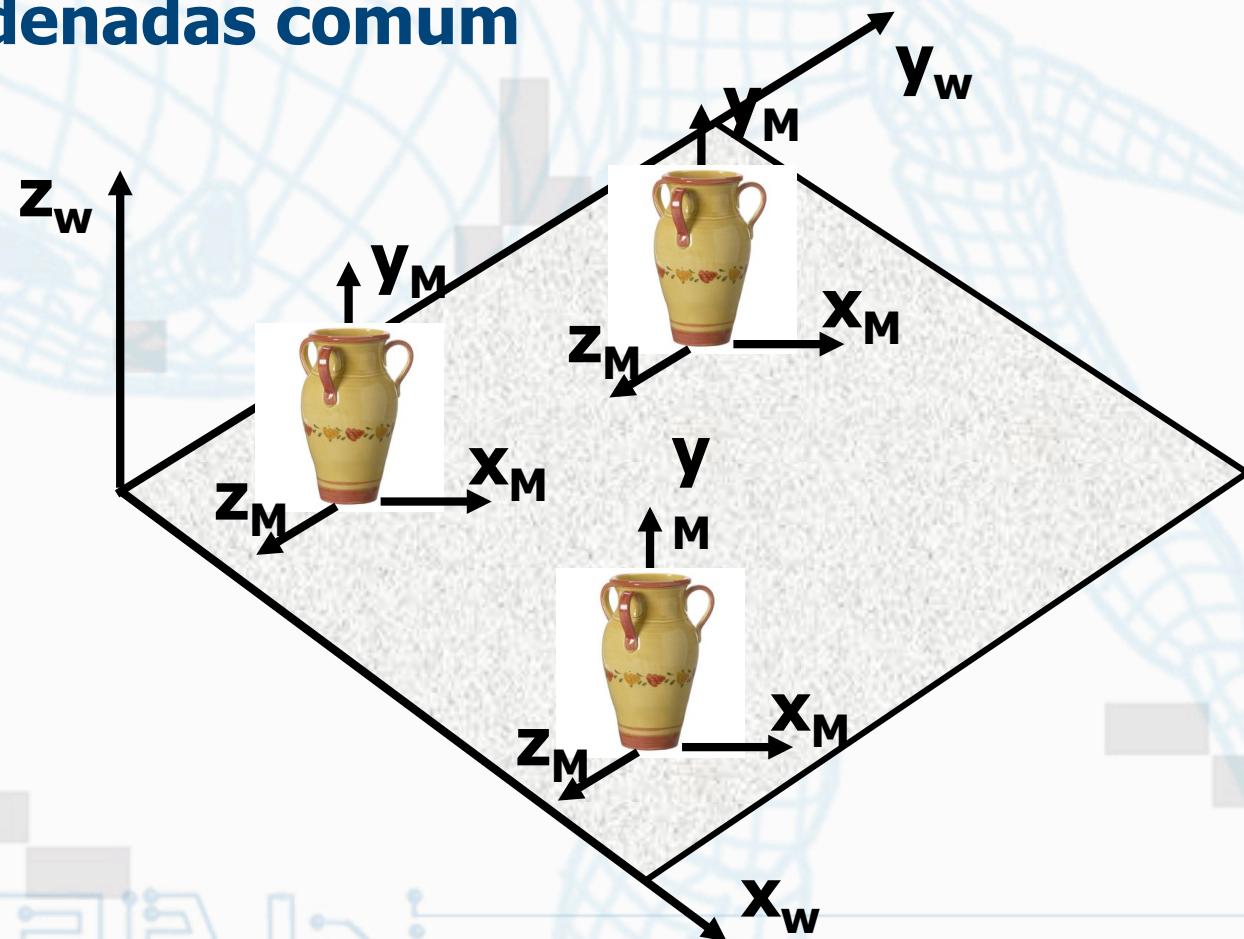
## 2.2 Programando Aplicações 2D

- Existem cinco espaços com seus sistemas de coordenadas próprios
  - Espaço do Modelo: Cada modelo tem seu próprio sistema de coordenadas



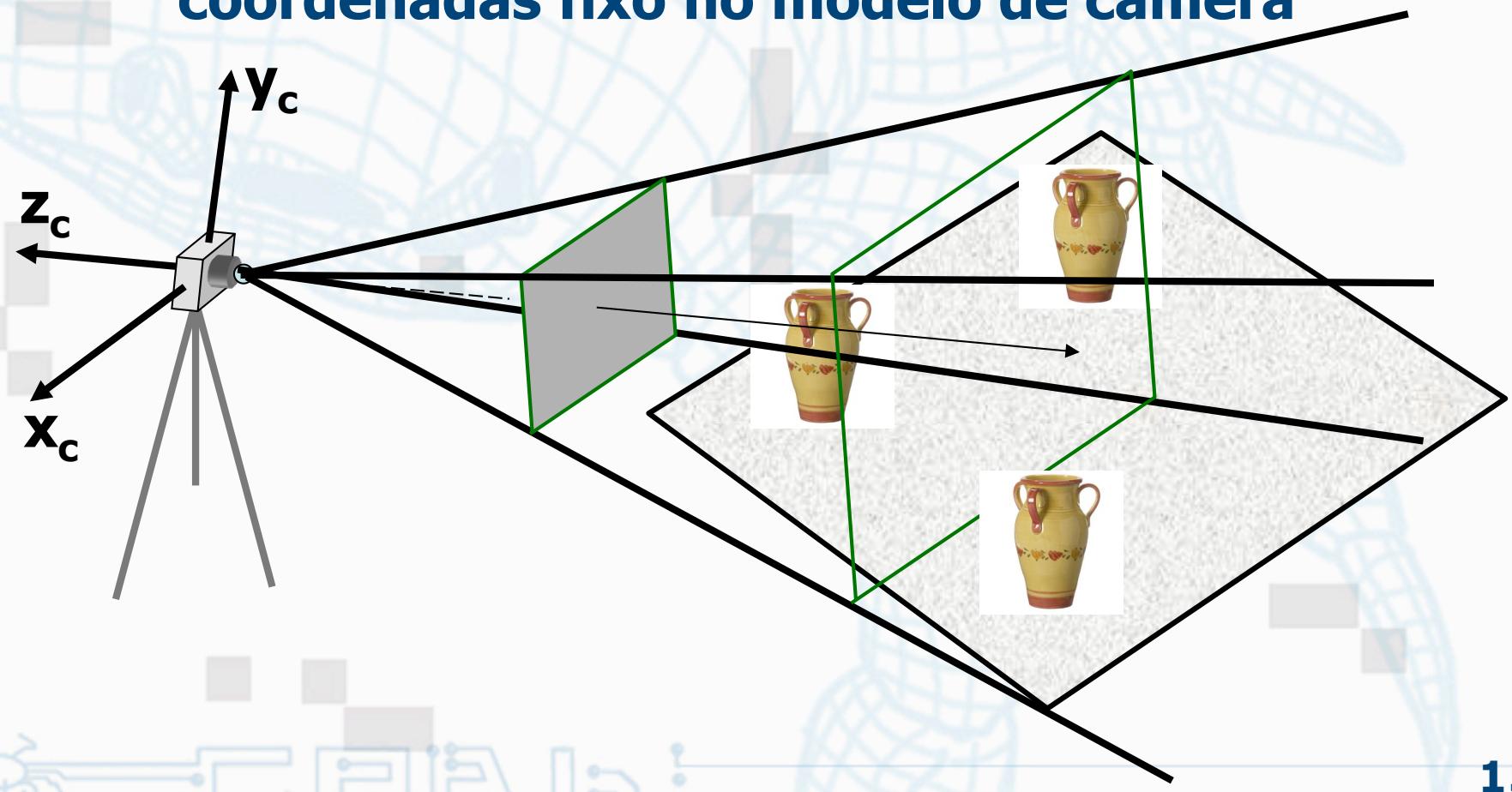
## 2.2 Programando Aplicações 2D

- **Espaço do Mundo (World): Todos os objetos do cenário posicionados relativo a um sistema de coordenadas comum**



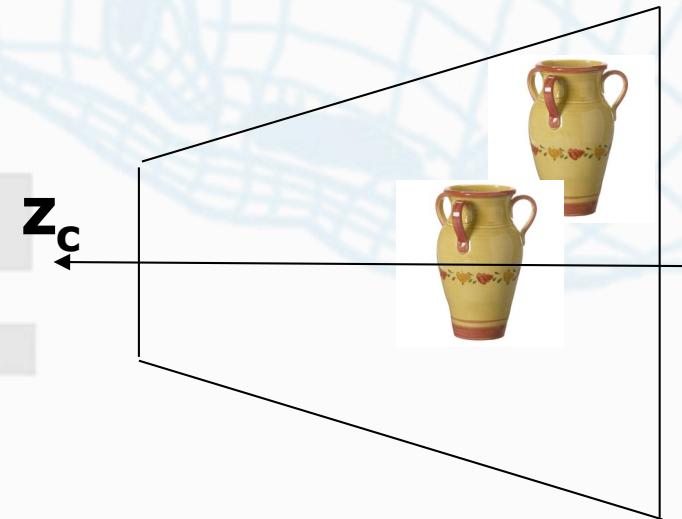
## 2.2 Programando Aplicações 2D

- **Espaço da Câmera: Todos os objetos do cenário posicionados relativo a um sistema de coordenadas fixo no modelo de câmera**

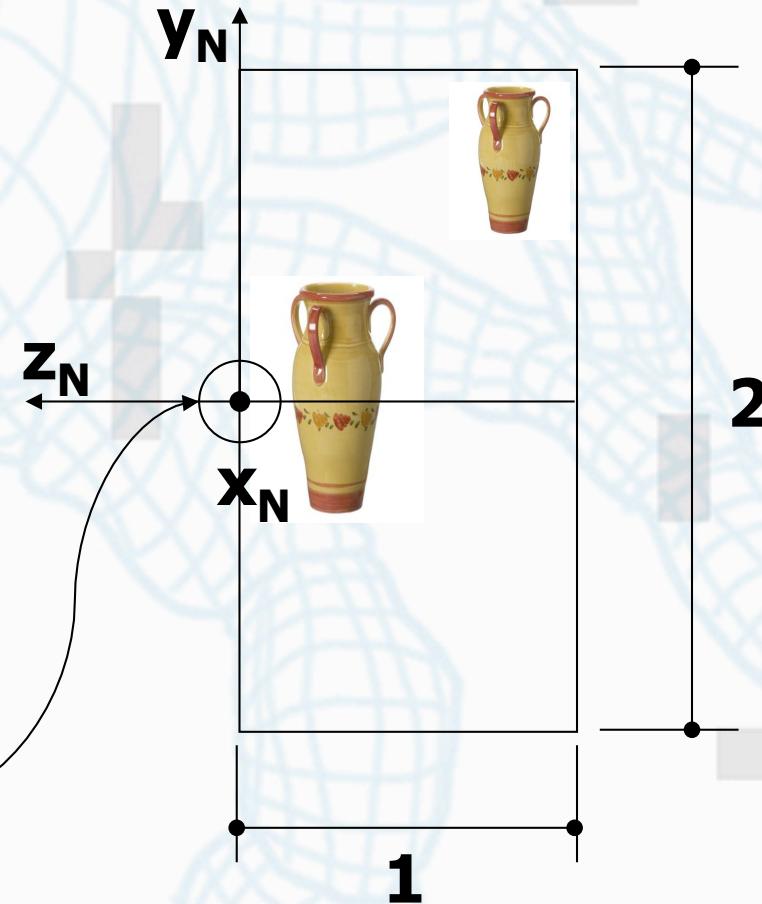


## 2.2 Programando Aplicações 2D

- Espaço de Clipping: Frustum transformado em um paralelepípedo  $2 \times 2 \times 1$

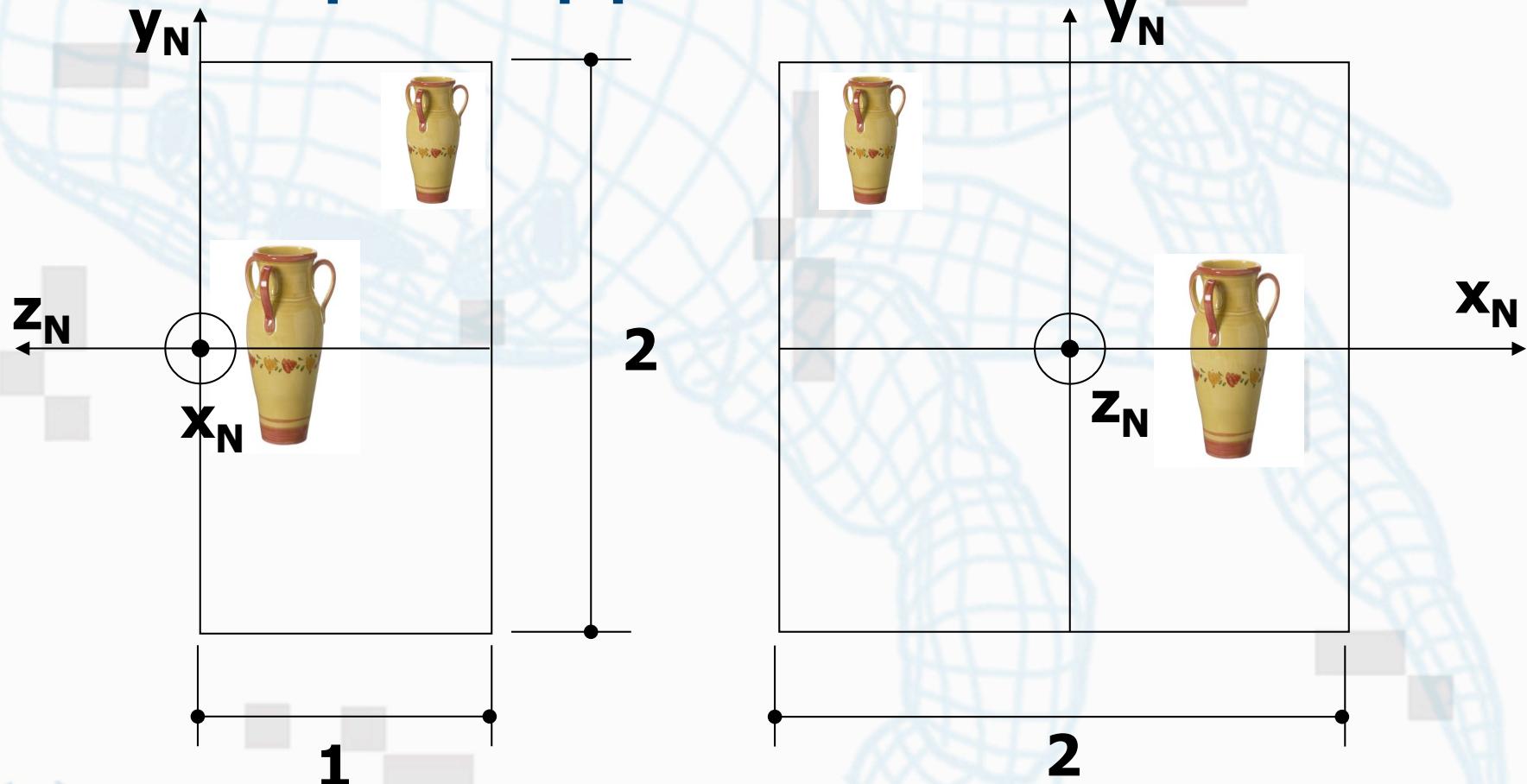


**NDC=(Normalized  
d Device  
Coordinates)**



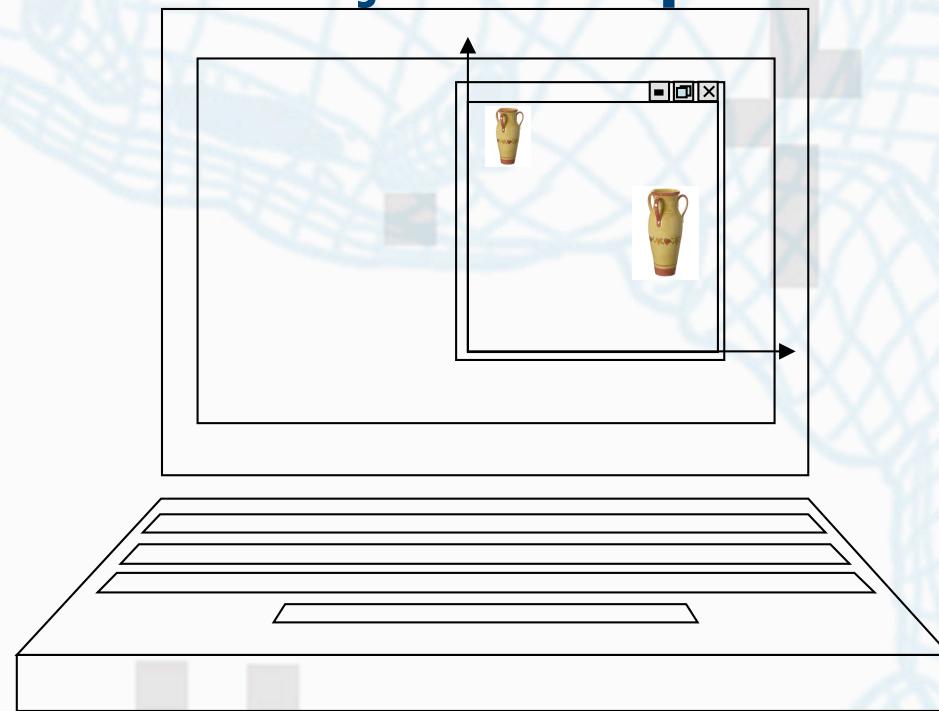
## 2.2 Programando Aplicações 2D

- Espaço de Clipping: Frustum transformado em um paralelepípedo  $2 \times 2 \times 1$



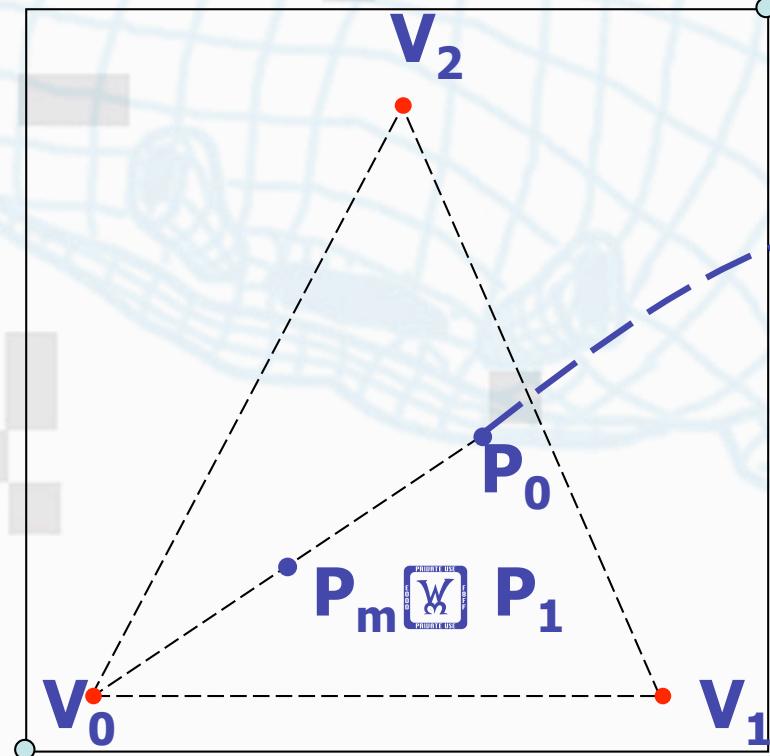
## 2.2 Programando Aplicações 2D

- **Espaço do Dispositivo:** xy em NDC é transformado para as dimensões em pixels na tela. z é preservado para algoritmo de eliminação de superfícies ocultas



## 2.1 Programando Aplicações 2D

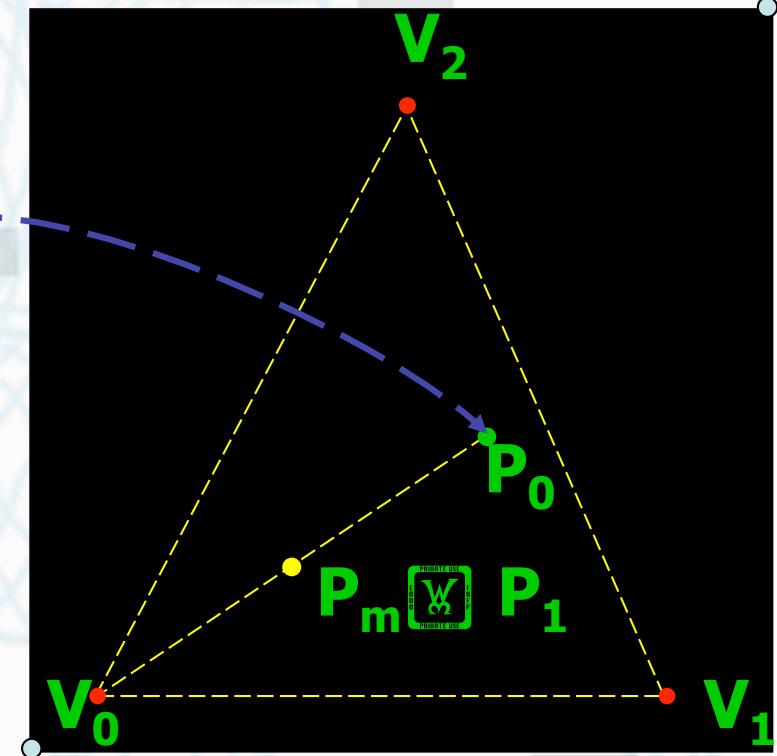
$(x_{w\_max}, y_{w\_max})$



$(x_{w\_min}, y_{w\_min})$

Janela no Mundo

$(x_{v\_max}, y_{v\_max})$

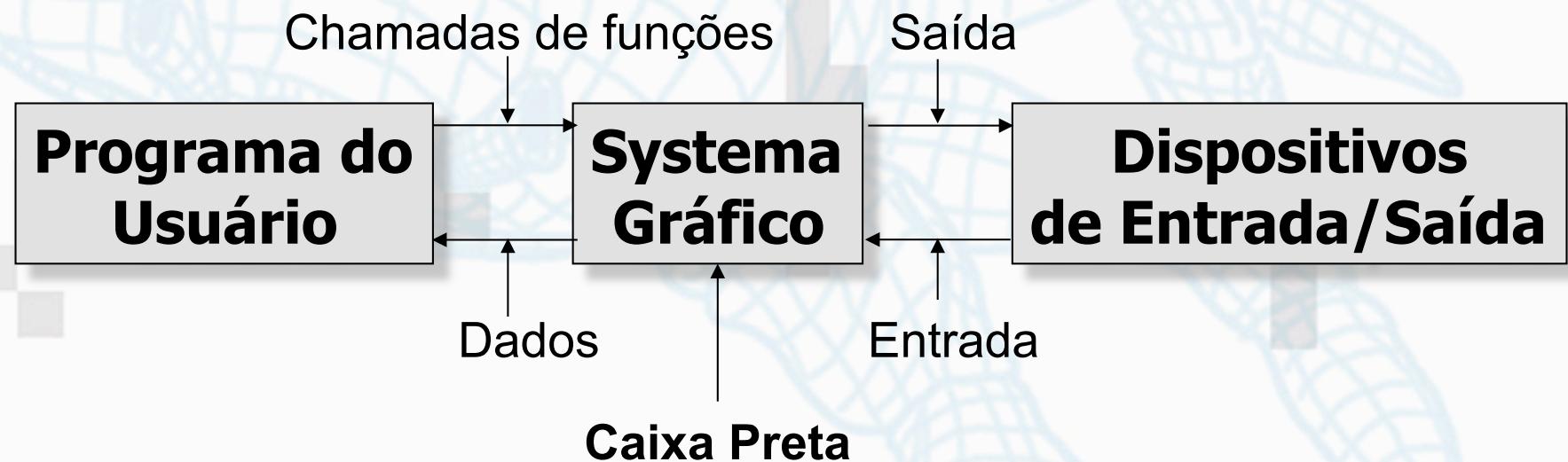


$(x_{v\_min}, y_{v\_min})$

Viewport no dispositivo

## 2.3 A API OpenGL

### 2.3.1 Funções gráficas



## 2.3 A API OpenGL

### 2.3.1 Funções gráficas

- Descrever API através das funções de sua biblioteca.
- Funções divididas em 7 grupos
  1. Primitivas
  2. Atributos
  3. Visualização
  4. Transformações
  5. Entrada
  6. Controle
  7. Inquirição

## 2.3 A API OpenGL

### 2.3.2 A Pipeline gráfica e máquinas de estado

- Sistema gráfico pode ser visto como uma Máquina de Estado (Caixa Preta que contem uma máquina de estado finito)
  - Input oriundos do programa de aplicação
    - Mudam o estado da máquina
    - Faz a máquina produzir output visível
  - Funções de dois tipos
    - Definem primitivas que fluem na pipeline (`glVertex`)
    - Mudam o estado dentro da máquina (`glEnable`)

## 2.3 A API OpenGL

### 2.3.2 A Pipeline gráfica e máquinas de estado

- No OpenGL
  - Parâmetros são persistentes
    - Valores mudam apenas através de funções que alteram o estado
    - Ex: Definindo uma cor ela fica como cor corrente até ser trocada por outra
  - Atributos não são ligados aos objetos mas são partes do estado

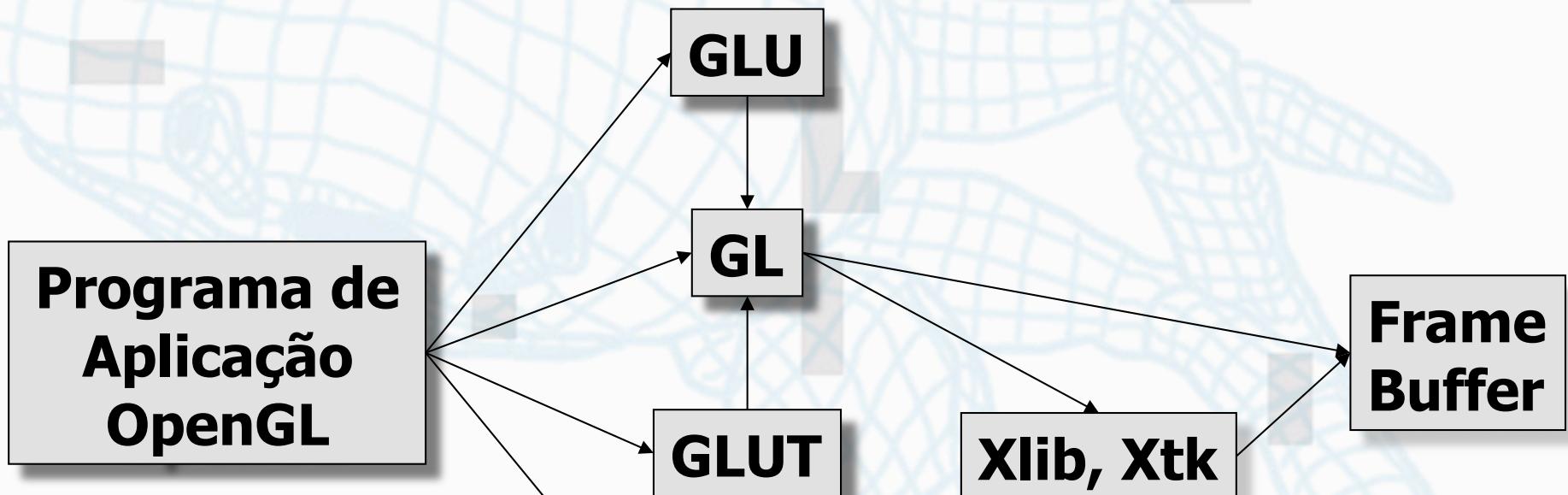
## 2.3 A API OpenGL

### 2.3.3 A interface OpenGL

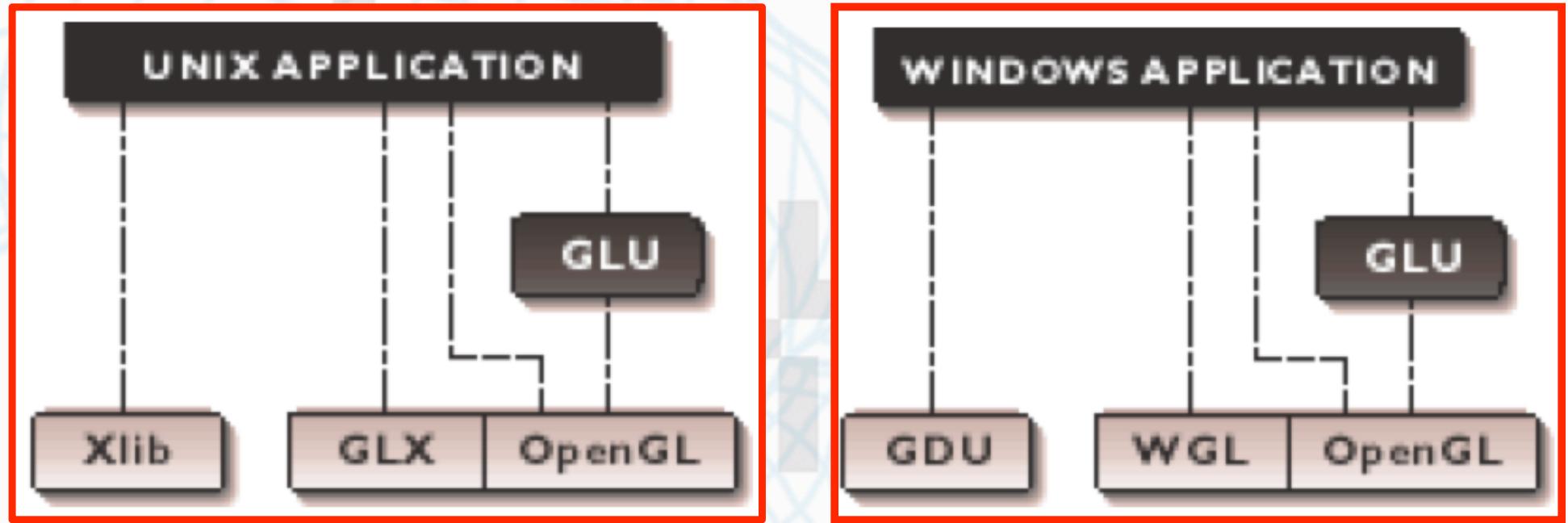
- **Biblioteca gráfica principal**
  - GL ou OpenGL no Windows
- **Bibliotecas relacionadas**
  - **GLU – Graphics Utility Library**
    - Usa GL mas contém código para criação de objetos comuns (Esferas, Cubos, ...)
  - **GLUT – Graphics Utility Toolkit**
    - Faz interface com o Sistema Windows
  - **GLX fornece a mínima ligação com X Windows**
    - Chamada pela GLUT

# 2.3 A API OpenGL

## 2.3.3 A interface OpenGL



## 2.3 A API OpenGL



## 2.3 A API OpenGL

### 2.3.3 A interface OpenGL

- Macros são usados para facilitar a leitura do código
  - **GL\_FILL**
  - **GL\_POINTS**

# 2.4 Primitivas e Atributos

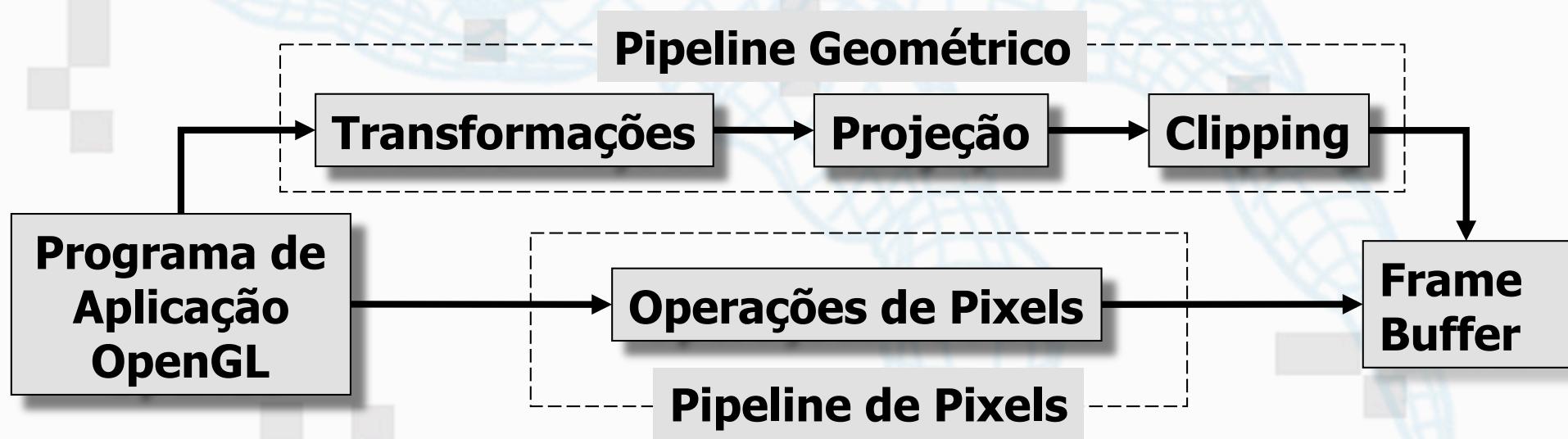
## 2.4.0 Introdução

- O que deve ser considerado primitiva em uma API?
  - Conjunto mínimo suportado por todos os tipos de hardware
    - Linhas, polígonos, alguma forma de texto
  - Otogonalidade: funcionalidade que não pode ser obtida pelas outras primitivas
  - OpenGL: número rasoável de primitivas

## 2.4 Primitivas e Atributos

### – Primitivas do OpenGL

- Primitivas Geométricas
  - Sofrem transformações espaciais
- Primitivas de Raster
  - Arrays de pixels



## 2.4 Primitivas e Atributos

### – Primitivas do OpenGL

- Pontos: **GL\_POINTS**
  - Cada vértice tem o tamanho de no mínimo 1 pixel
- Segmentos de reta: **GL\_LINES**
  - Pares de vértices
  - Geralmente segmentos são desconexos
- Poligonais: **GL\_LINE\_STRIP**, **GL\_LINE\_LOOP**
  - Abertas e fechadas

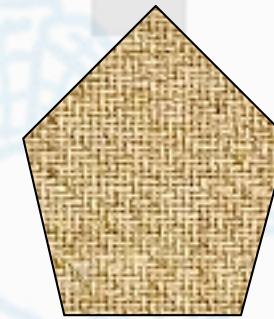
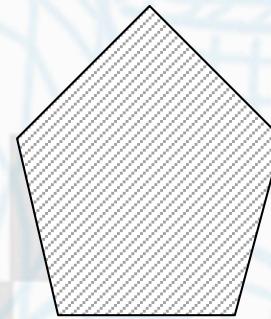
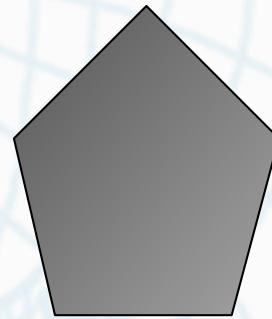
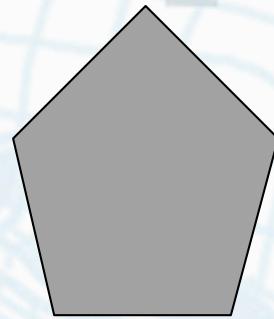
# 2.4 Primitivas e Atributos

## 2.4.1 Básico sobre polígonos

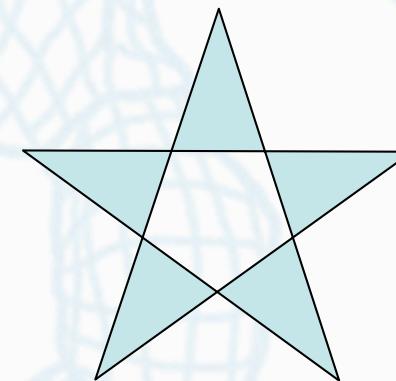
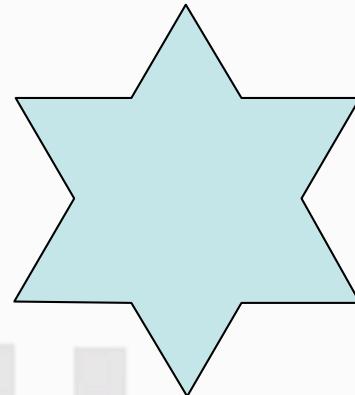
- **Polígono: Objeto com**
  - **Borda descrita por um “line loop”**
  - **Região Interior**
- **Usados para representar superfícies curvas (aproximação poliedral)**
- **Propriedades obrigatórias**
  - **Simples**
  - **Convexo**
  - **Plano**

## 2.4 Primitivas e Atributos

- **Exibição de polígonos**

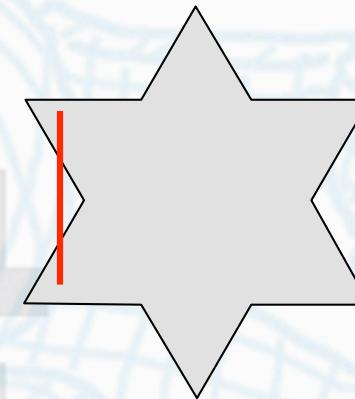
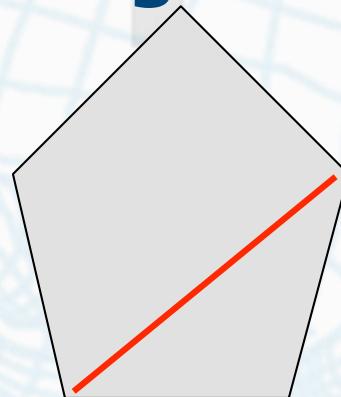


- **Polígonos Simples e Não-simples**

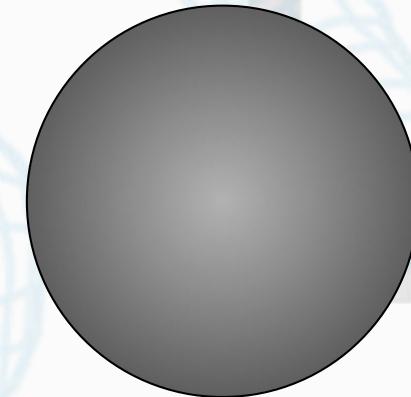
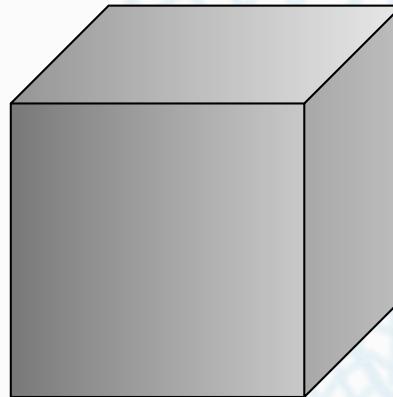
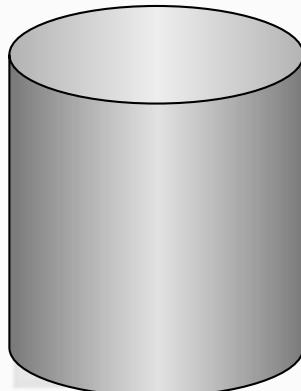


## 2.4 Primitivas e Atributos

- **Polígonos Convexos e Côncavos**



- **Objetos convexos**



## 2.4 Primitivas e Atributos

### 2.4.2 Tipos de polígonos no OpenGL

- Polígonos

- **GL\_POLYGON**

- Triângulos e quadriláteros

- **GL\_TRIANGLES**
  - **GL\_QUADS**

- Faixas e leques

- **GL\_TRIANGLE\_STRIP, GL\_QUAD\_STRIP**
  - **GL\_TRIANGLE\_FAN**

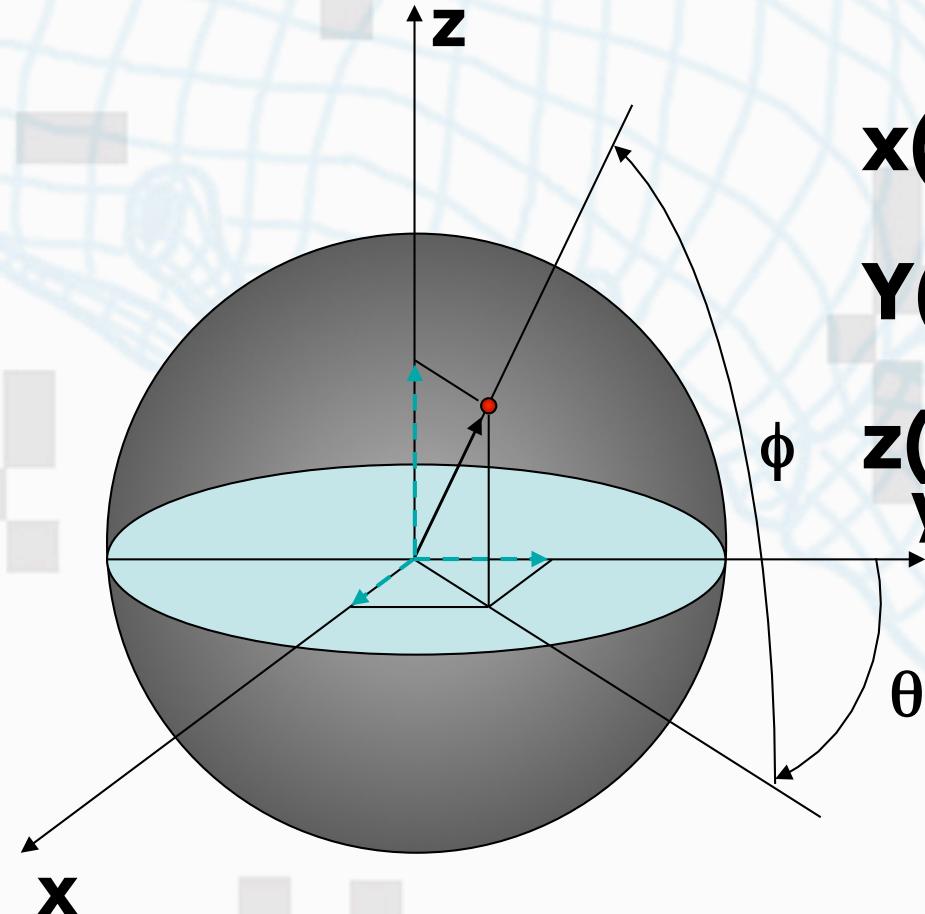
## 2.4 Primitivas e Atributos

### 2.4.3 Desenhando uma esfera

- Aproximação por poliedro
  - Definir vértices no cruzamento de meridianos e paralelos
    - Definir em coordenadas esféricas
    - Transformar para coordenadas cartesianas
  - Entre dois paralelos, utilizar
    - **GL\_QUAD\_STRIP ou GL\_TRIANGLE\_STRIP**
  - Entre um polo e o paralelo vizinho, utilizar
    - **GL\_TRIANGLE\_FAN**

## 2.4 Primitivas e Atributos

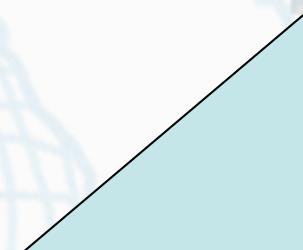
### – Coordenadas dos nós (raio = 1)



$$x(\theta, \phi) = \sin\theta (1 \cdot \cos\phi)$$

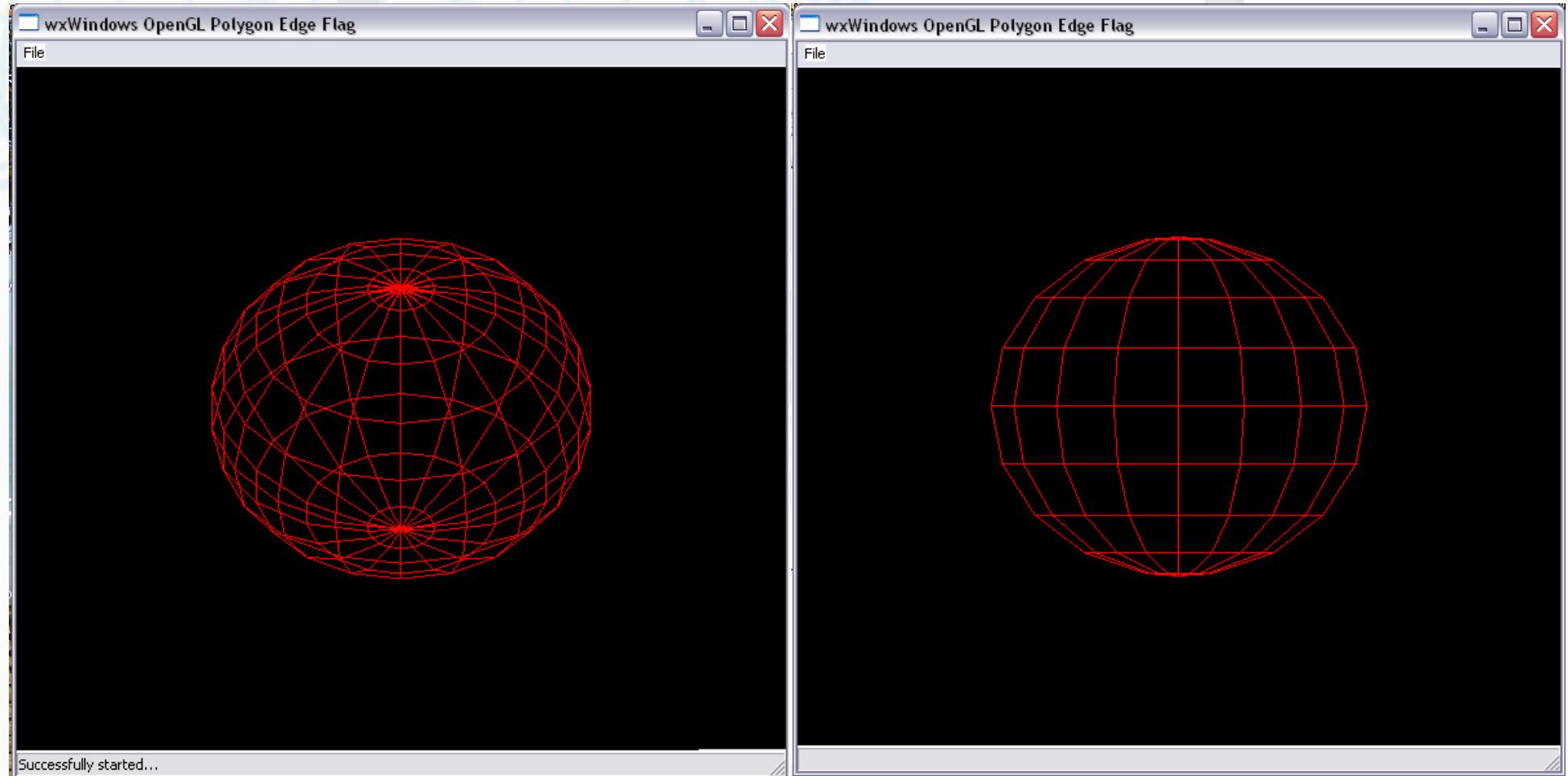
$$y(\theta, \phi) = \cos\theta (1 \cdot \cos\phi)$$

$$\phi \quad z(\theta, \phi) = \sin\phi$$



## 2.4 Primitivas e Atributos

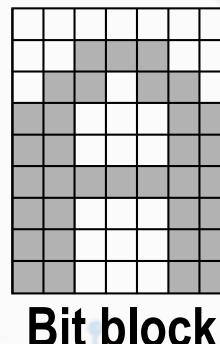
### – Wireframe da esfera



# 2.4 Primitivas e Atributos

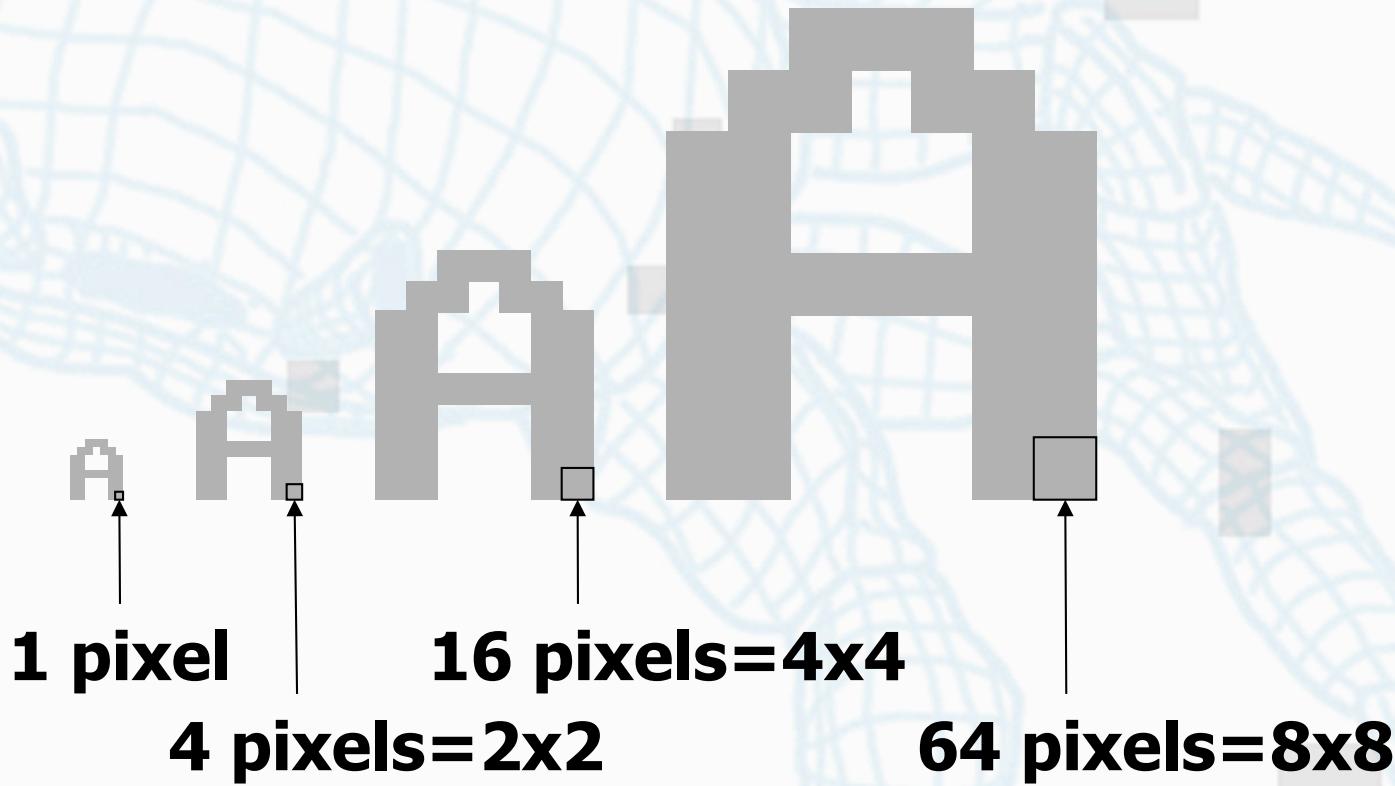
## 2.4.4 Texto

- Problemático em aplicações gráficas
- Duas formas
  - **Stroke text (vetorial): definido como qualquer outro objeto em CG**
  - **Raster(bitmaped): Um bloco de bits pode ser transferido com uma única instrução para o frame buffer (BITBLT: Bit-Block-Transfer)**



## 2.4 Primitivas e Atributos

– Escala de caractere por replicação de pixels



## 2.4 Primitivas e Atributos

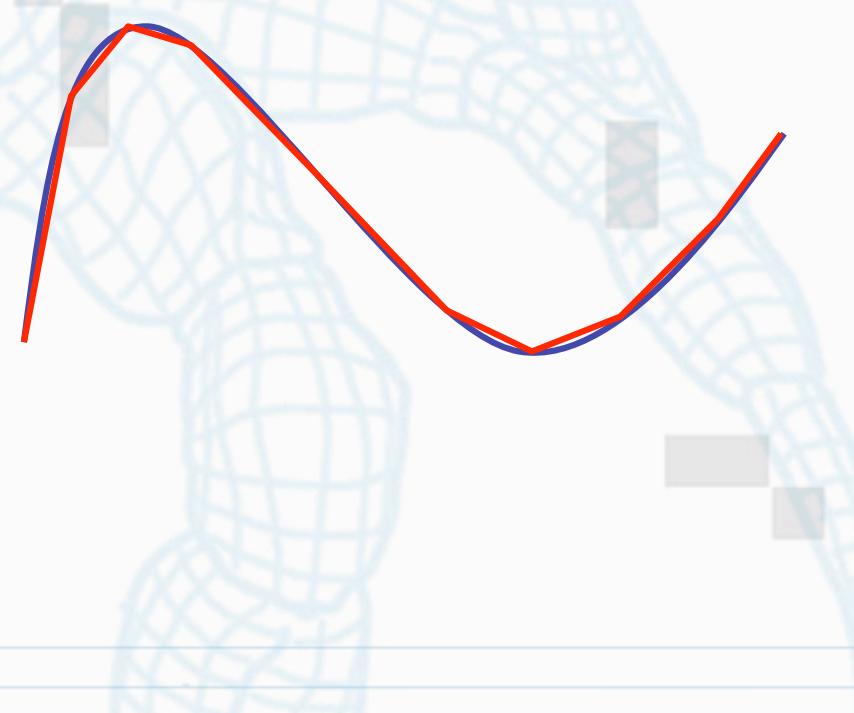
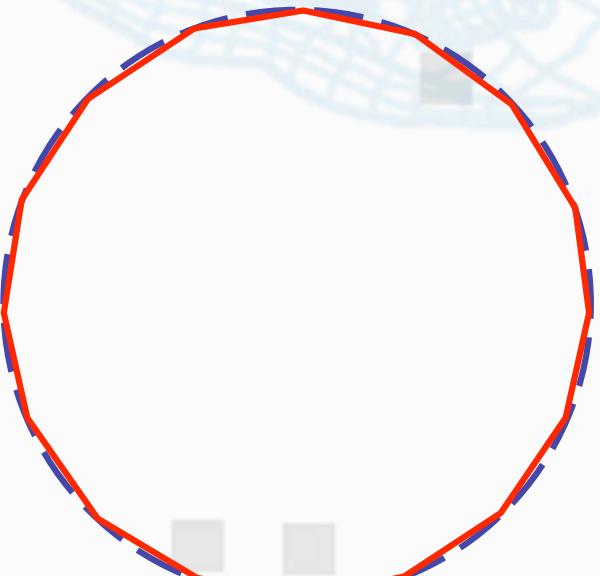
- OpenGL não tem primitiva de texto
    - Textos podem ser criados por meio de outras primitivas
  - GLUT provê caracteres stroke e bitmap
    - Definidos em software
    - Portáveis
- `glRasterPos2i(x, y);`
- `glutBitmapCharacter(GLUT_BITMAP_8_BY_13,C);`

## 2.4 Primitivas e Atributos

### 2.4.5 Objetos curvos

– Dois tipos de representação

- Poliédrica (Tessellation) ——————
- Matemática ——————



## 2.4 Primitivas e Atributos

### 2.4.6 Atributos

- Propriedade que determina como uma primitiva geométrica será desenhada
- Cor
- Espessura
- Tipo de linha
- Padrão de preenchimento
- ...

## 2.4 Primitivas e Atributos

### 2.4.6 Atributos

— Linha cheia preta fina

— Linha cheia preta espessa

— Linha cheia vermelha fina

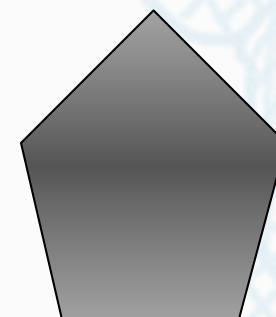
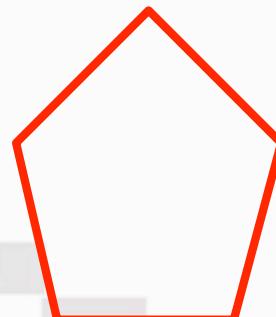
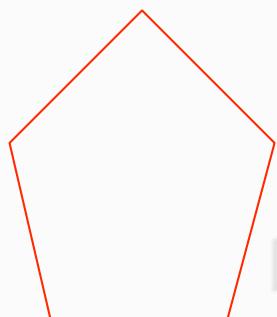
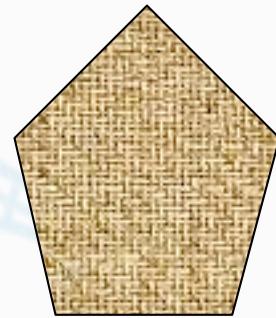
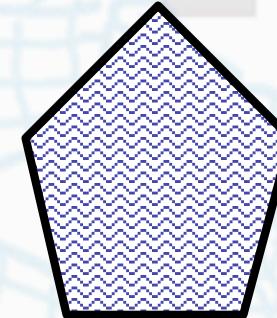
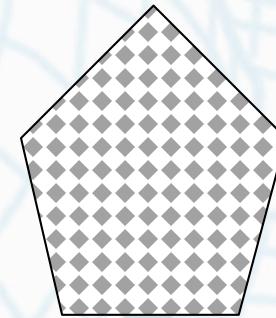
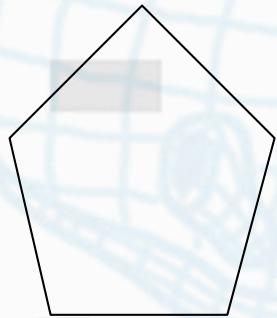
— · — · — · — · Linha traço ponto preta espessa

— — — — — - Linha tracejada vermelha espessa



## 2.4 Primitivas e Atributos

### 2.4.6 Atributos





# Fim da Aula 5

# 2.5 Cores

## 2.5.0 Introdução

- Luz: Onda eletromagnética entre (350 e 780 nm: 1mm/1 000 000)
- Cor é um fenômeno psicofísico
  - Percepção de ondas
    - ~350 → azul
    - ~780 → vermelho
    - ~560 → verde



350 nm

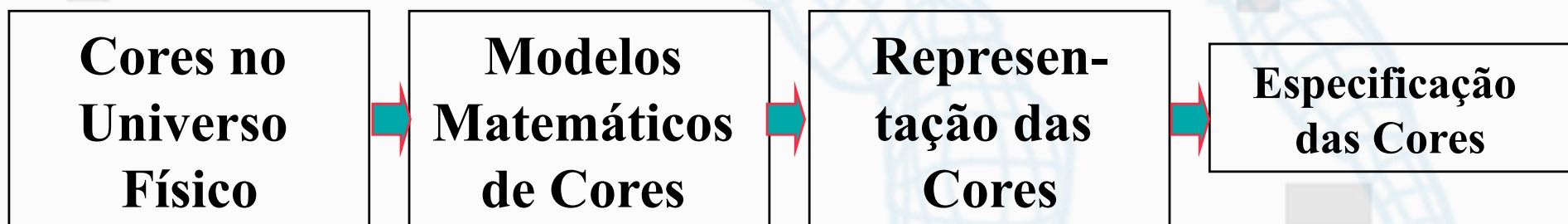
780 nm

## 2.5 Cores

- **Fonte luminosa**
  - A maioria das fontes distribui energia irregularmente entre as ondas componentes
- **Distribuição espectral (SPD)**
  - gráfico de Potência (Energia/s) x comprimento-de-onda
  - **Espectro-radiômetro**
    - Instrumento para medir a energia de irradiação em todo o espectro

## 2.5 Cores

- Cor é um fenômeno psicofísico
  - depende da interação entre luz e o sistema visual humano
  - manifestação perceptiva da luz
- Níveis de abstração no estudo das cores

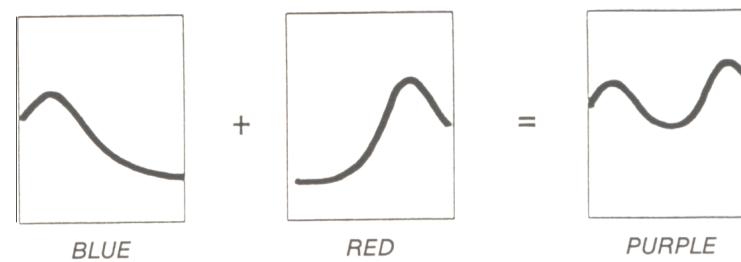


## 2.5 Cores

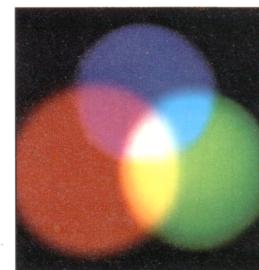
- **Processos de formação de cores**
  - Processo aditivo
  - Processo subtrativo

# 2.5 Cores

- **Processo aditivo**
  - resultado da mistura de luzes
    - combinação das SPD's de dada fonte

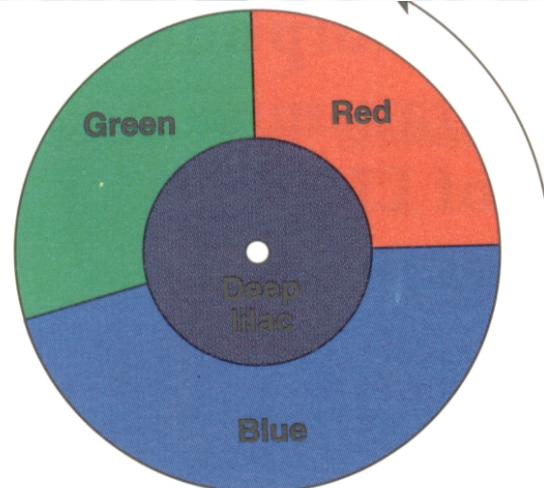


- **Experimento de Thomas Young em 1801**



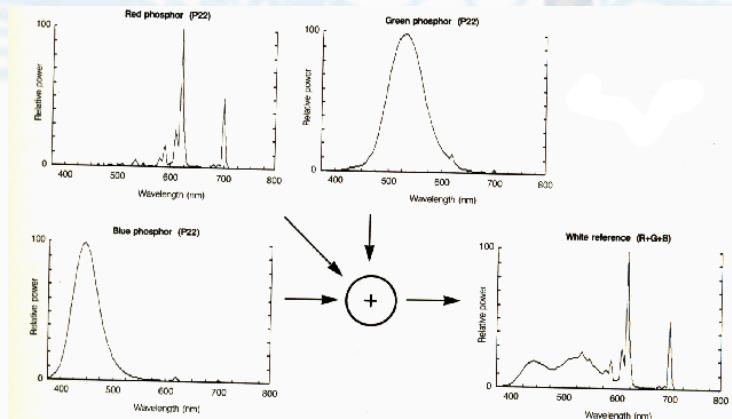
# 2.5 Cores

- Processo aditivo
  - Discos de cores de Maxwell
    - Setores ajustáveis
    - Coincidir com cor central



# 2.5 Cores

- Processo aditivo
  - Os monitores CRT Coloridos
    - dispositivos aditivos
    - mistura da luz emitida pelos três fósforos



# 2.5 Cores

- **Processo substrativo**
  - Filtros
  - Pigmentação
- **Filtros**
  - Luz passa através de um material contendo colorante
    - Lei de Lambert
    - Lei de Beer

## 2.5 Cores

- **Filtros**
  - **Lei de Lambert**
    - **espessuras iguais de material → absorção igual independente da intensidade da fonte**
  - **Lei de Beer**
    - **quantidades iguais de material absorvente → quantidades iguais de absorção**

# 2.5 Cores

- **Filtros**
  - **Forma combinada da Lei Beer-Lambert**

$$A(\lambda) = a(\lambda) \cdot b \cdot c = \ln\left(\frac{1}{T(\lambda)}\right)$$

T( $\lambda$ ) = transmitância

A( $\lambda$ ) = absorvência  
(é aditiva)

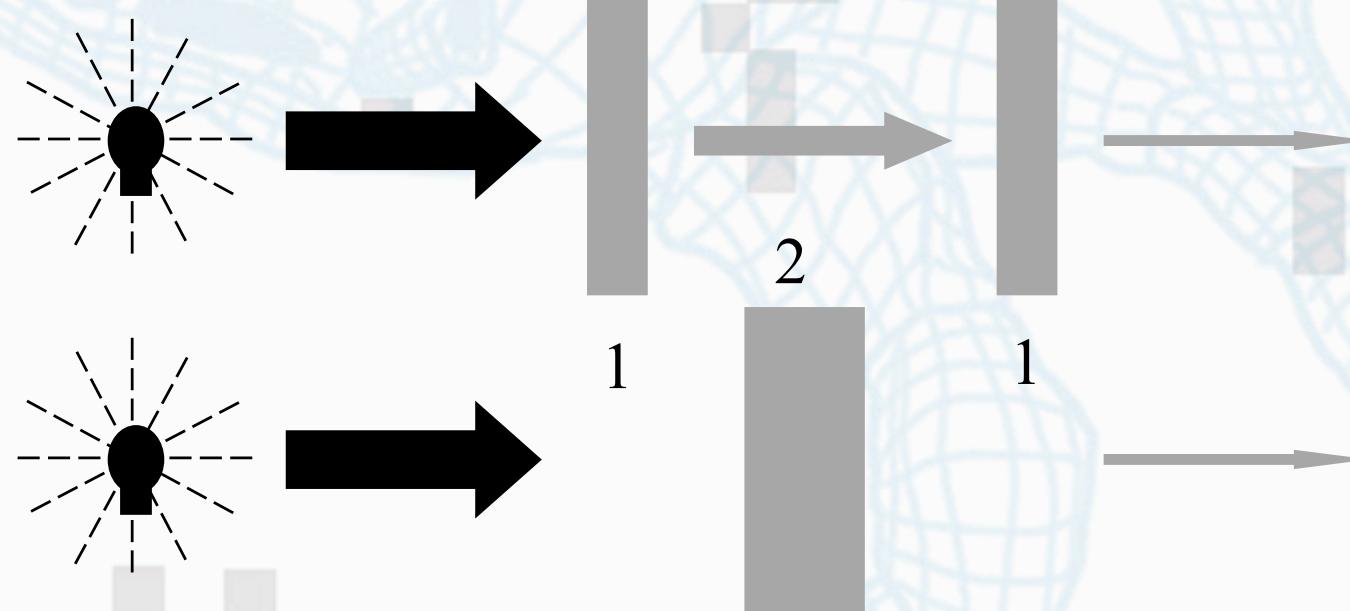
a( $\lambda$ ) = absorbilidade do material

b = espessura da amostra

c = concentração da tintura

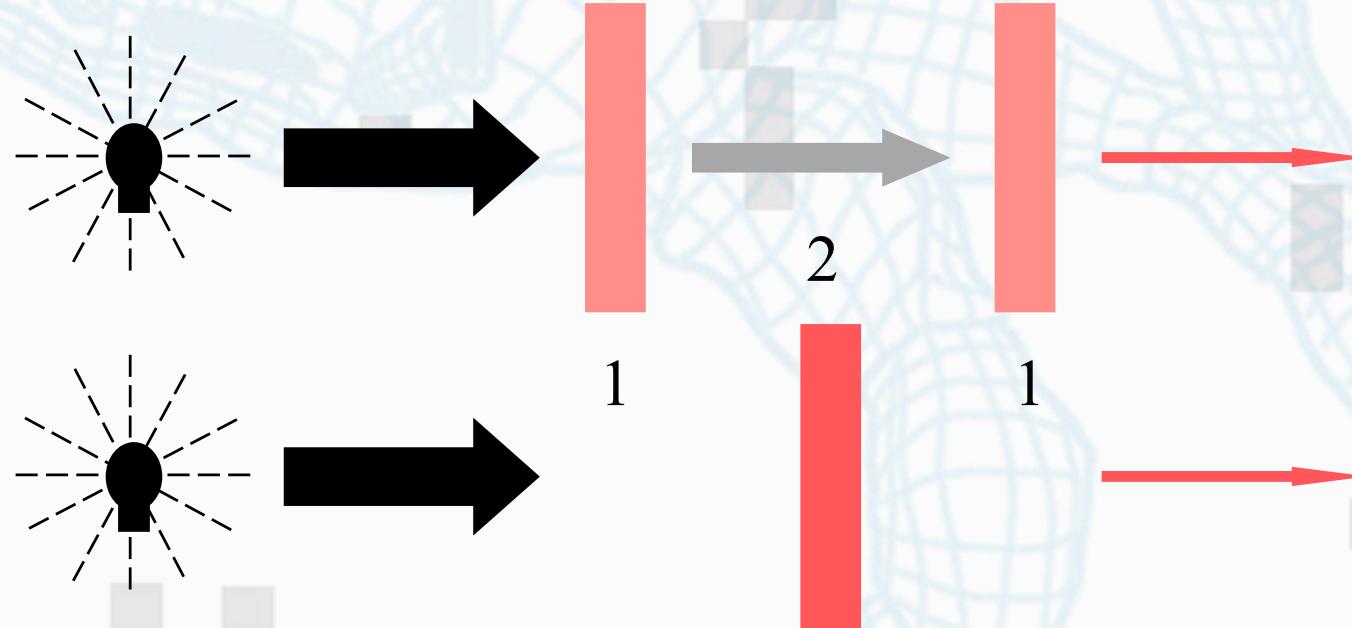
# 2.5 Cores

- Filtros
  - Lei de Lambert



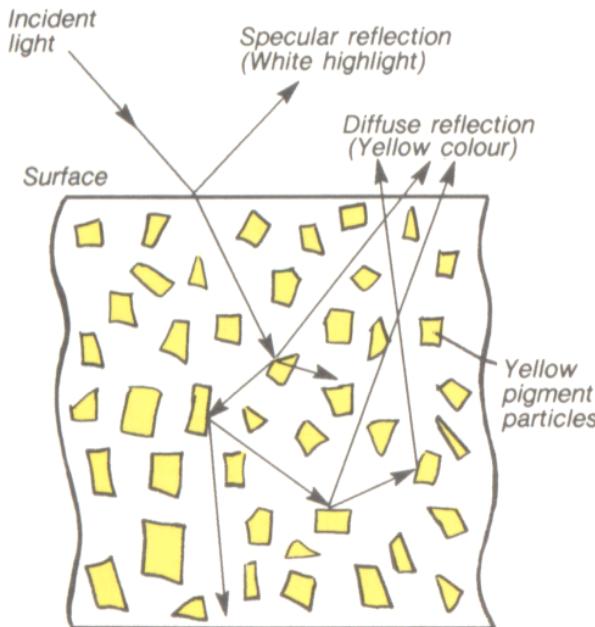
# 2.5 Cores

- **Filtros**
  - **Lei de Beer (baixas a médias concentrações)**



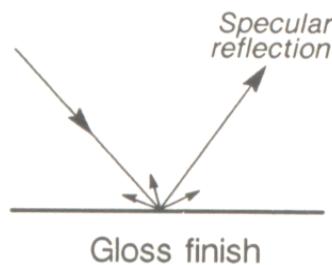
# 2.5 Cores

- **Pigmentação**

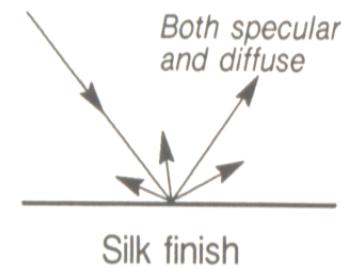
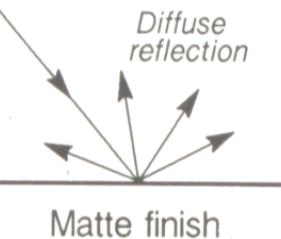


# 2.5 Cores

- Pigmentação



**10a** Polar distribution of reflected light from three different types of surface.



## 2.5 Cores

- **Fonte luminosa irradia  $n(\lambda)$  photons para cada  $\lambda$  (distribuição espectral de photons)**
- **Se  $n(\lambda)d\lambda$  photons são emitidos por segundo na faixa  $(\lambda, \lambda + d\lambda)$** 
  - Energia por segundo (Potência):
    - $P = h.c.\lambda^{-1}.n(\lambda)d\lambda$   $\langle h = 6.626\ 068\ 96\ e^{-34}\ J.s \rangle$
- **Fonte monocromática**
  - emite photons do mesmo  $\lambda$
  - cor espectral ou cor pura

## 2.5 Cores

- Trocar modelo contínuo por discreto
- Trocar o espaço espectral (dimensão infinita) por espaço de dimensão finita
- Amostragem de cores (receptores)
  - aproxima o espaço de dimensão infinita por um espaço de dimensão finita
- Reconstrução de cores (emissores)
  - obtém a cor espectral a partir de amostras

## 2.5 Cores

- Amostragem de cores
  - receptores de cores
    - número finito de sensores,  $s_i$
    - cada qual com uma função de resposta espectral,  $s_i(\lambda)$  → indica o peso com que uma onda luminosa de comprimento  $\lambda$  contribui para a saída do sensor
  - $C(\lambda) \rightarrow$  SPD da luz
  - Sinal resultante do sensor  $s_i$

$$C_i = \int C(\lambda) s_i(\lambda) d\lambda$$

# 2.5 Cores

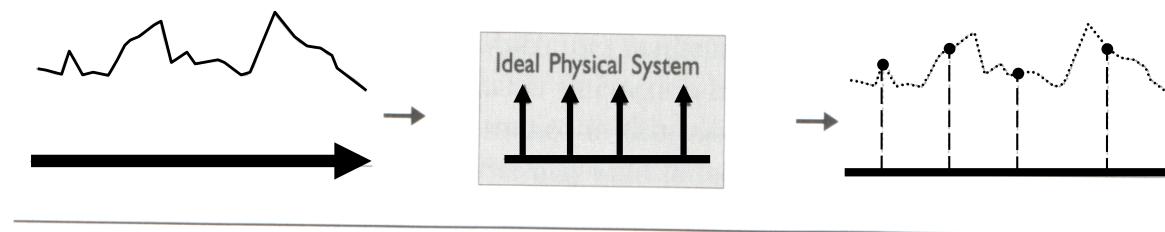
## – Receptor ideal

- função de resposta espectral é a função de Dirac

$$C_i = \int_R C(\lambda) \delta(\lambda - \lambda_i) d\lambda = C(\lambda_i)$$

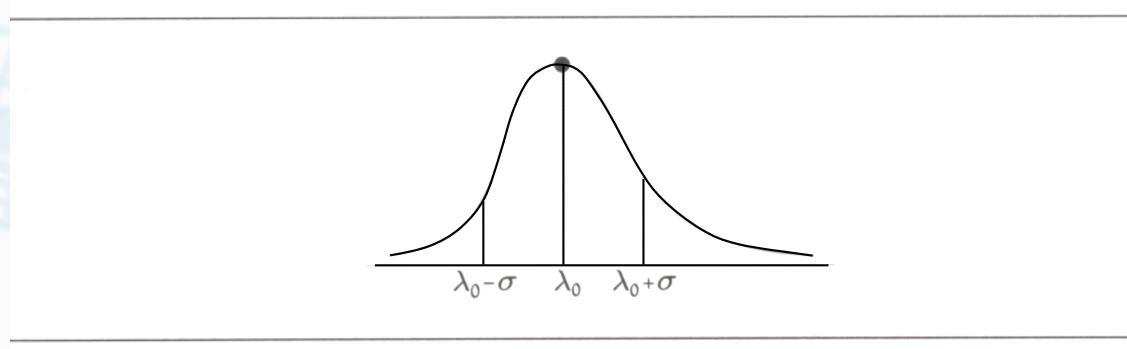
- Executa amostragem pontual

Figure 3.3. Point sampling of color in an ideal receptor.



# 2.5 Cores

- Receptor típico
  - função de resposta espectral



- Quanto menor o desvio padrão
  - mais localizada a resposta do receptor
  - maior a sensibilidade naquela faixa do espectro

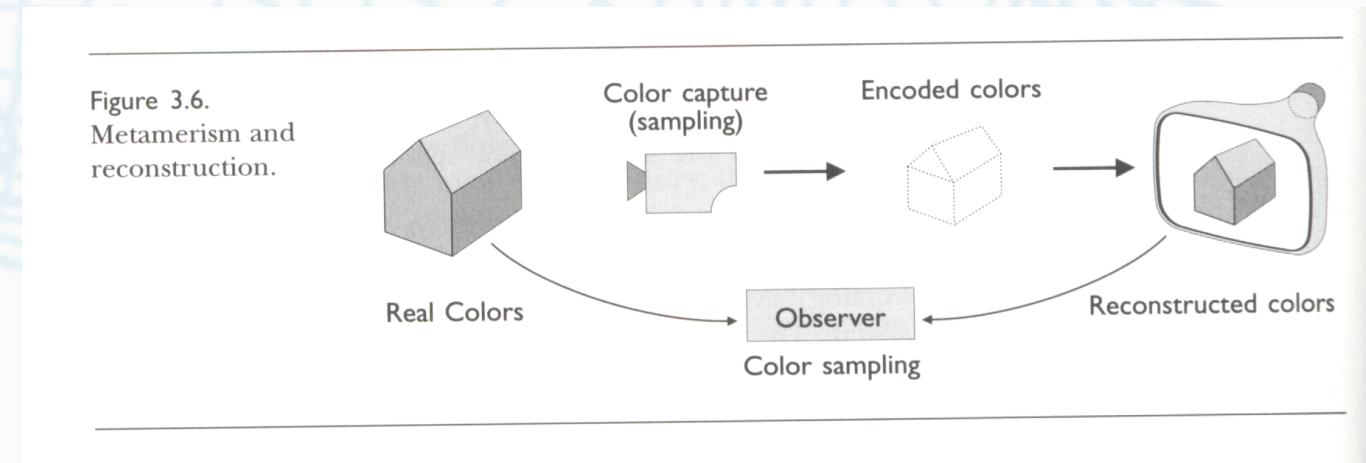
## 2.5 Cores

- Transformação de representação,  $R:E \rightarrow R^n$
- $R(C) = (C_1, \dots, C_n)$
- relação entre o espaço espectral de cores e sua representação finita  $R^n$
- Relação de **metamerismo**  $\approx$  em  $E$

$$C(\lambda) \approx C'(\lambda) \Leftrightarrow R(C) = R(C')$$

## 2.5 Cores

- **Reconstrução de cores (emissores)**
  - obtém a cor espectral a partir de amostras



- O receptor (olho humano) deve gerar a mesma amostragem para as cores reais e para as cores reconstruídas

## 2.5 Cores

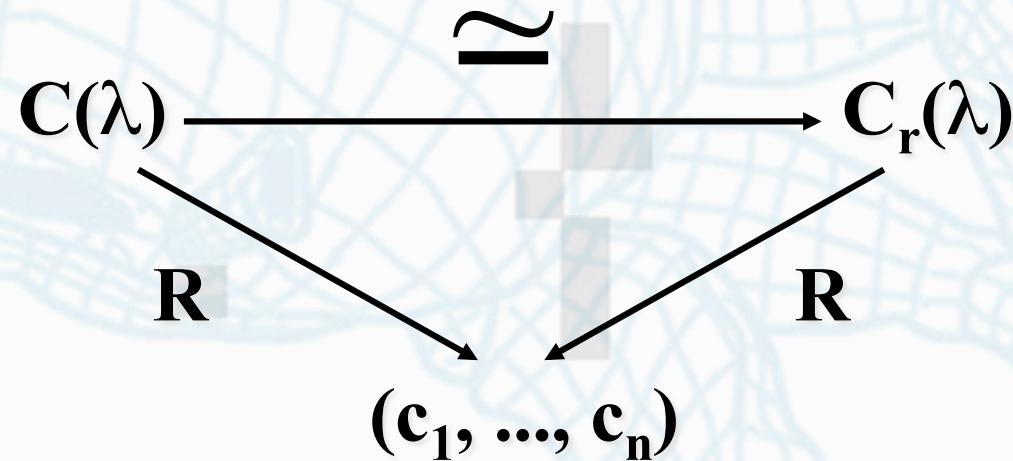
- Reconstrução: Obtém Cor  $C_r(\lambda)$  no espaço de cores do emissor  $S_e$

$$C_r(\lambda) = \sum_{k=1}^n \beta_k P_k(\lambda)$$

- $P_k$  são as cores primárias
- $\beta_k$  são coeficientes de combinação linear (componentes primárias)

## 2.5 Cores

- Cor reconstruída deve ser metamérica da cor real com relação ao receptor



- Se  $s_i(\lambda)$  são as curvas de resposta espectral de um receptor, a representação da cor  $C(\lambda)$  é dada pelo vetor  $(\alpha_1(C), \dots, \alpha_n(C))$

## 2.5 Cores

- **Funções de reconstrução de cores,  $C_k(\lambda)$**

- Utilizadas devido à dificuldade de determinar  $s_i(\lambda)$  experimentalmente
  - Definidas de forma que

$$\delta(\lambda - \lambda_o) = \sum_{k=1}^n C_k(\lambda_o) P_k(\lambda)$$

- $C_k(\lambda_o)$  é fator de combinação linear da cor primária  $P_k(\lambda)$  para obtenção da cor pura com comprimento de onda  $\lambda_o$  e distribuição  $\delta(\lambda - \lambda_o)$

## 2.5 Cores

- Representação CIE-RGB
  - Cores Primárias

$$P_1(\lambda) = \delta(\lambda - \lambda_1) \quad \text{para} \quad \lambda_1 = 700\text{nm(red)}$$

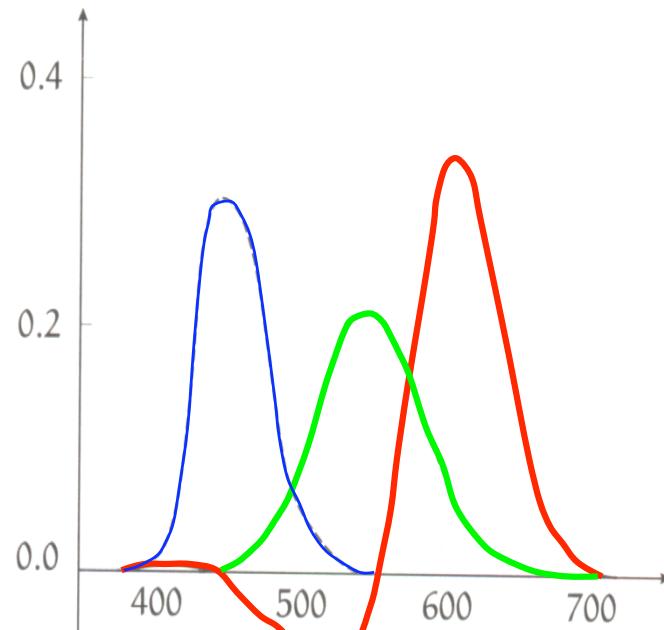
$$P_2(\lambda) = \delta(\lambda - \lambda_2) \quad \text{para} \quad \lambda_2 = 546\text{nm(green)}$$

$$P_3(\lambda) = \delta(\lambda - \lambda_3) \quad \text{para} \quad \lambda_3 = 435.8\text{nm(blue)}$$

## 2.5 Cores

- **Fuções de reconstrução de cores**

Figure 3.7. Color reconstruction functions in the CIE-RGB system.



## 2.5 Cores

- Determinação experimental das funções de reconstrução de cores

## 2.5 Cores

- Luminância e Cromaticidade
  - Percepção de cor em dois passos
    - Sinais R, G e B gerados pelas células fotosensitivas
    - Combinações de sinais enviados para o cérebro ( $R+G$ ), ( $R-G$ ),  $B - (R+G)$ .
  - B tem pouca influência na percepção de cor escura ou clara
    - ( $R+G$ ) corresponde à **Luminância**
    - ( $R-G$ ) e  $B - (R+G)$  codificam o resto da informação de cor, a **Cromaticidade**

## 2.5 Cores

- **Luminância e Cromaticidade**
  - Olho envia ao cérebro
    - Componentes Bidimensionais de cromaticidade
    - Componente unidimensional de luminância

## 2.5 Cores

- **Luminância de  $C(\lambda)$**

$$L(C(\lambda)) = K \int_{\mathcal{R}} C(\lambda) V(\lambda) d\lambda$$

- **K≈680lumens/Watt**
- **V( $\lambda$ ) é a função eficiência luminosa relativa**

$$V(\lambda) = \sum_{i=1}^n a_i s_i(\lambda)$$

- **Luminâncias de duas cores metaméricas são idênticas**
  - $L(C)=L(C')$

## 2.5 Cores

- Se  $C(\lambda)$  é uma cor visível e  $t > 0$  é Real,  
 $t \cdot C(\lambda)$  também é visível
  - O conjunto de cores visíveis é um cone no espaço espectral de cores
- Se  $C_1$  e  $C_2$  são visíveis e  $t \in [0, 1]$  é Real,  
 $C = (1 - t) C_1 + t C_2$  também é visível
  - O conjunto de cores visíveis é convexo

## 2.5 Cores

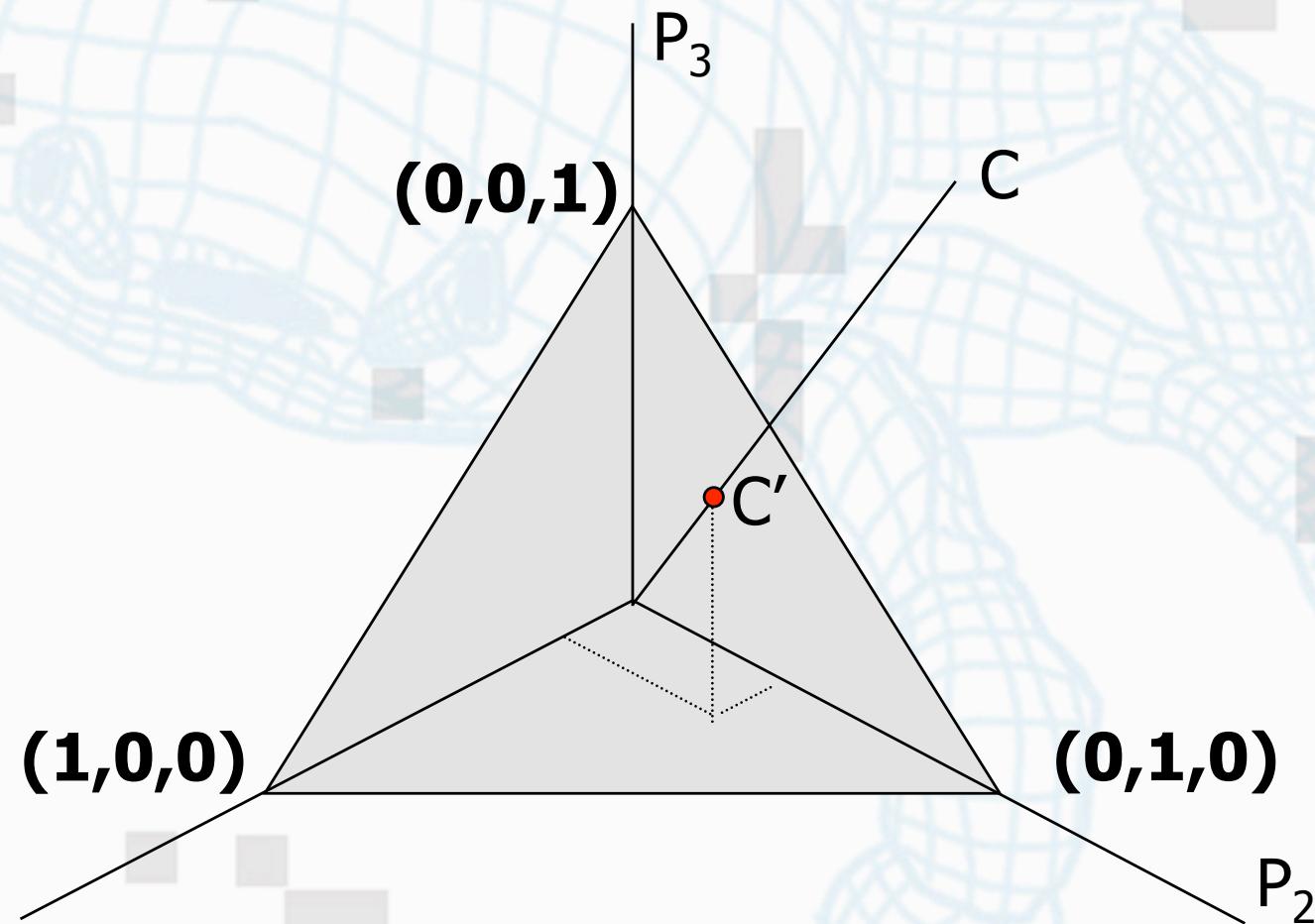
- No espaço discreto de representação de cores (subespaço do  $R^n$ ) o cone convexo é chamado de Sólido de Cores
- A projeção radial,  $C'$ , das cores do sólido de cores no plano de Maxwell define o **diagrama de cromaticidade**
  - $C' = t_o C$
  - $C' \in$  Plano de Maxwell  $\Rightarrow C'_1 + C'_2 + C'_3 = 1$
  - Assim,  $t_o = 1 / (C_1 + C_2 + C_3)$

## 2.5 Cores

- No espaço discreto de representação de cores (subespaço do  $R^n$ ) o cone convexo é chamado de Sólido de Cores
- A projeção radial,  $C'$ , das cores do sólido de cores no plano de Maxwell define o **diagrama de cromaticidade**
  - $C' = t_o C$
  - $C' \in$  Plano de Maxwell  $\Rightarrow C'_1 + C'_2 + C'_3 = 1$
  - Assim,  $t_o = 1 / (C_1 + C_2 + C_3)$

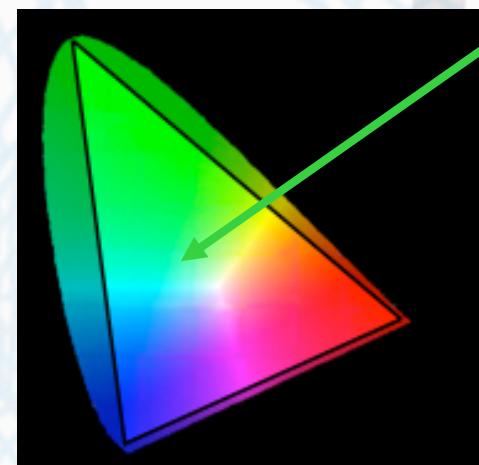
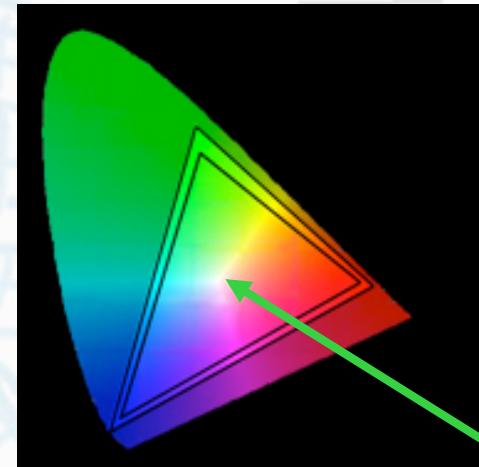
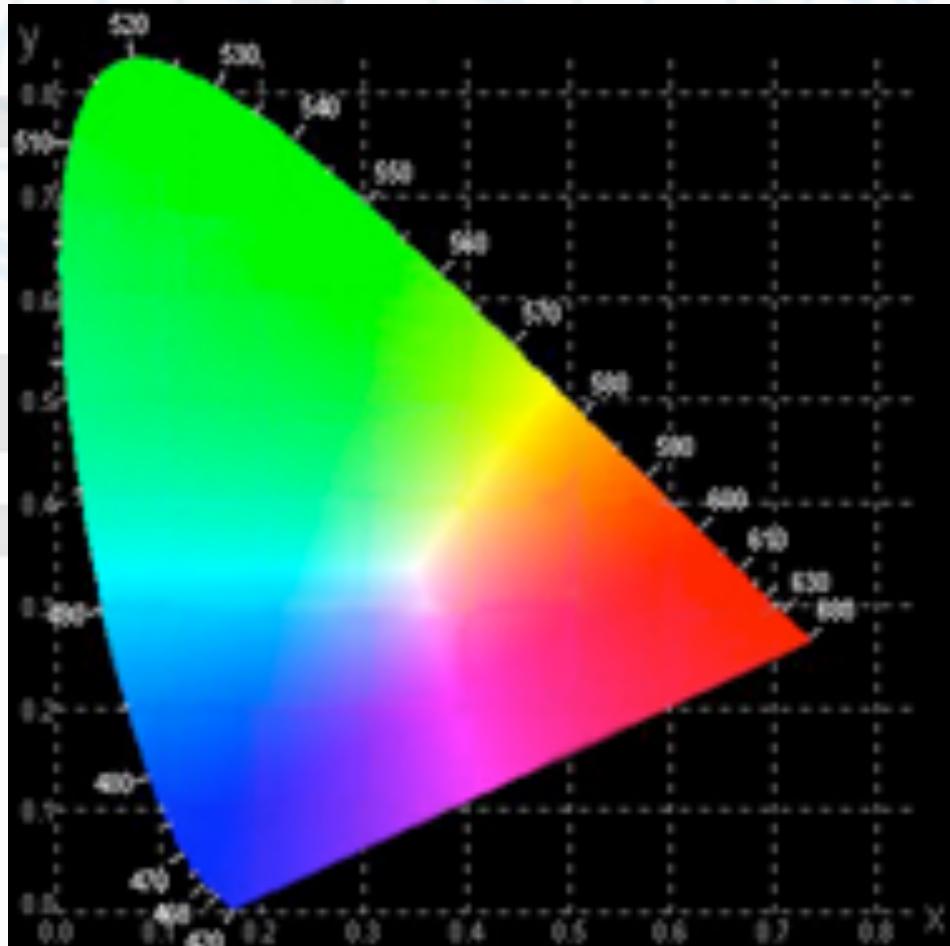
# 2.5 Cores

- Triângulo de Maxwell



# 2.5 Cores

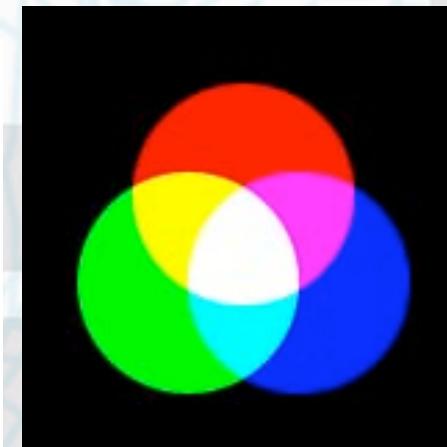
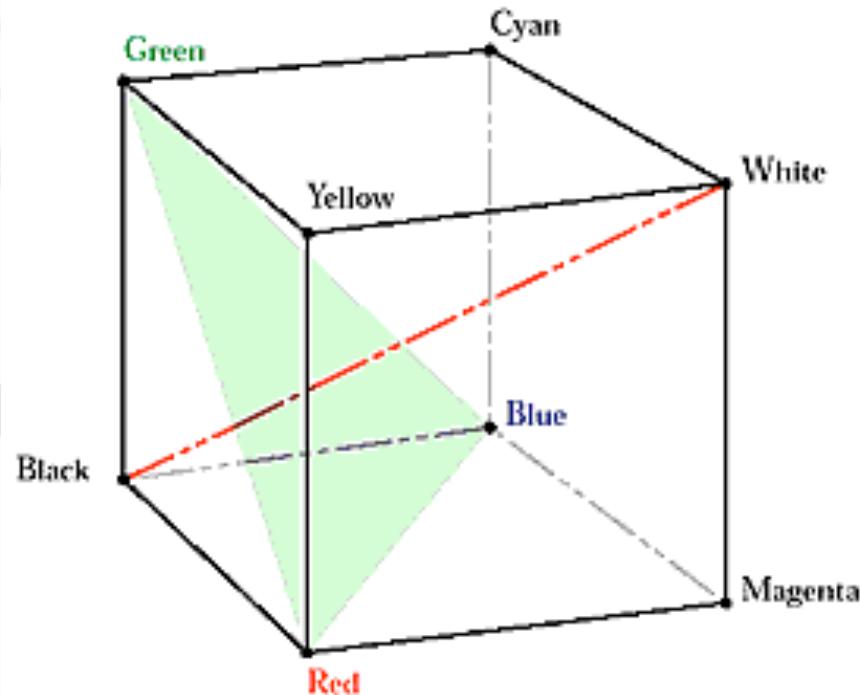
- CIE – Diagrama de cromaticidade



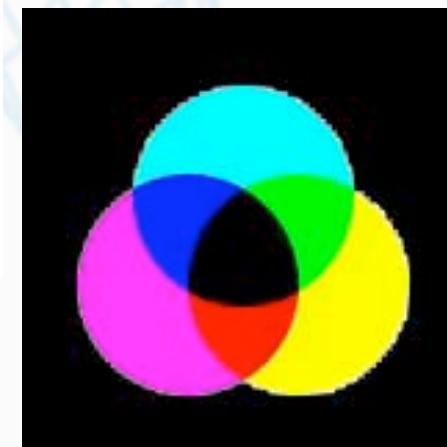
Gamut  
De  
cores

# 2.5 Cores

- Cores complementares



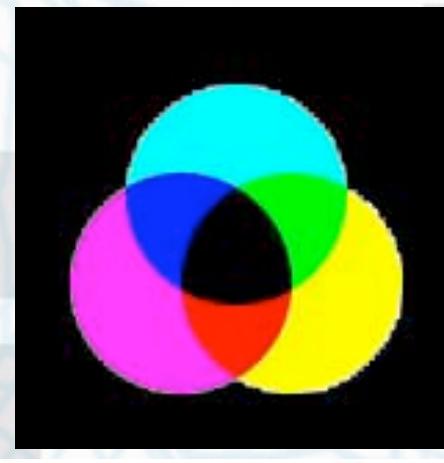
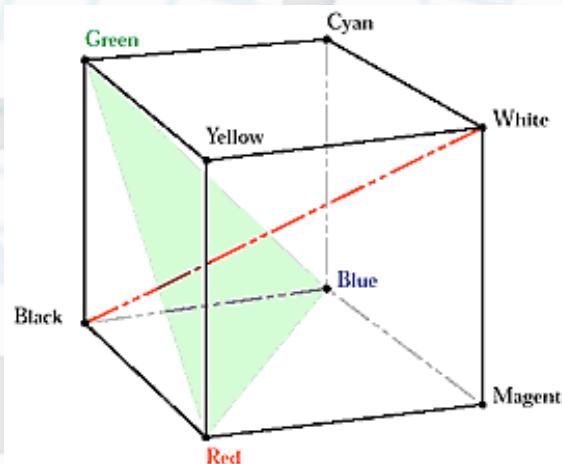
**Aditivas**



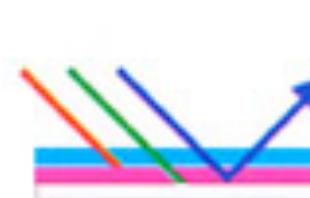
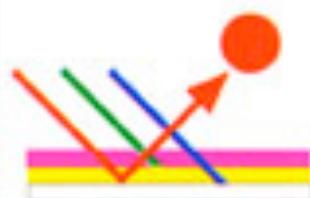
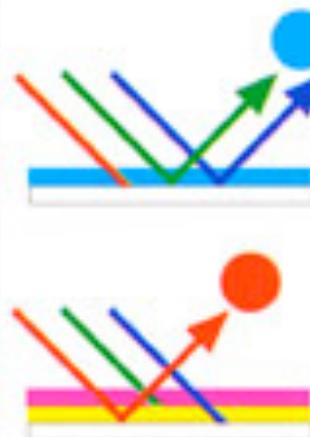
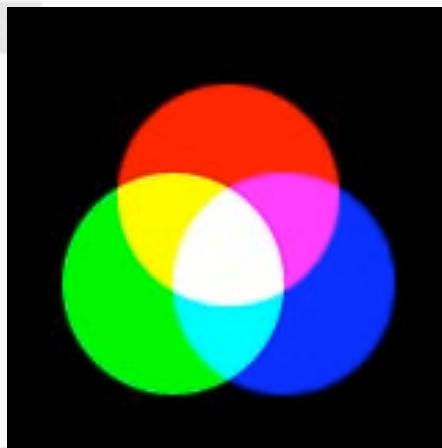
**Subtrativas**

# 2.5 Cores

- **Cores complementares**



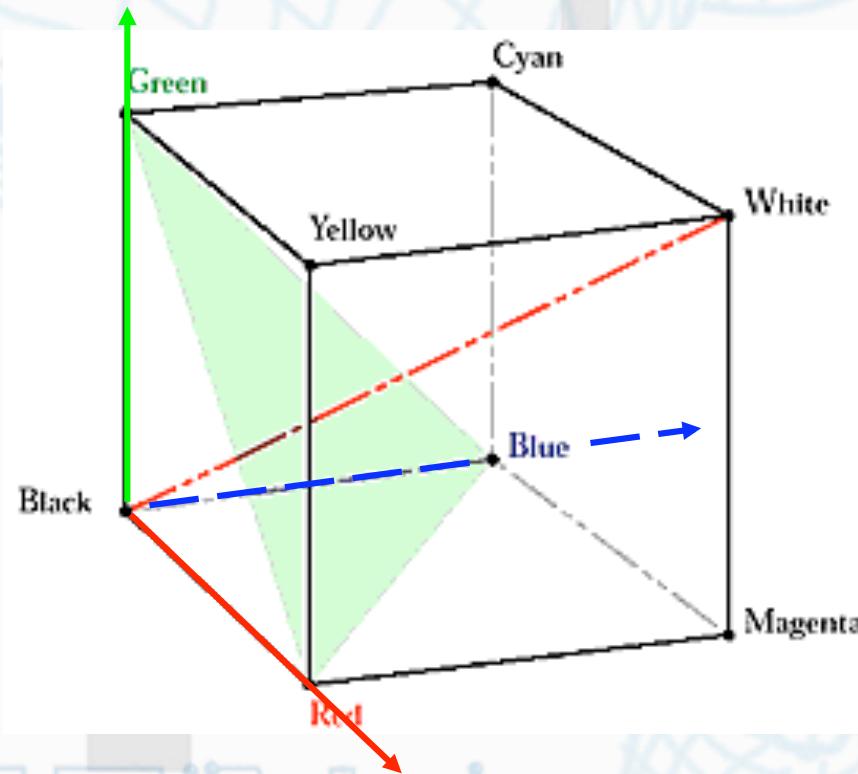
**Subtrativas**



# 2.5 Cores

## 2.5.1 Cor RGB

- Usa o cubo de cores com uma primária em cada eixo de coordenadas



# 2.5 Cores

## 2.5.1 Cor RGB

- Há sistemas com
  - 4 bits por cor
  - 8 bits por cor
  - 12 bits por cor (ou mais)
- API usa números no intervalo [0, 1]
  - Independência do hardware
- OpenGL exemplo: `glColor3f(1.0, 0.0, 0.0);`
- Em sistemas com 32 bits por pixel
  - 24 bits para cores e 8 bits para canal alpha

## 2.5 Cores

- Em sistemas raster RGB

- número de bits/pixel define faixa de cores disponíveis
- conjunto mínimo de cores: 3 bits/pixel
  - um bit para cada canhão (R, G e B)
  - cada canhão está ligado ou desligado

## 2.5 Cores

- 3 bits/pixel: 1 bit para cada canhão
- cada canhão está ligado ou desligado

Código	Red	Green	Blue	Cor
0	0	0	0	Preto
1	0	0	1	Azul
2	0	1	0	Verde
3	0	1	1	Ciano
4	1	0	0	Vermelho
5	1	0	1	Magenta
6	1	1	0	Amarelo
7	1	1	1	Branco

# 2.5 Cores

- Em sistemas de 24 bits/pixel
  - 8 bits por canhão (256 possibilidades)Totalizando **16.777.216 cores**
- Para sistema de **1.024 x 1.024 pixels**
  - tamanho do frame buffer = **3.145.728 bytes**

# 2.5 Cores

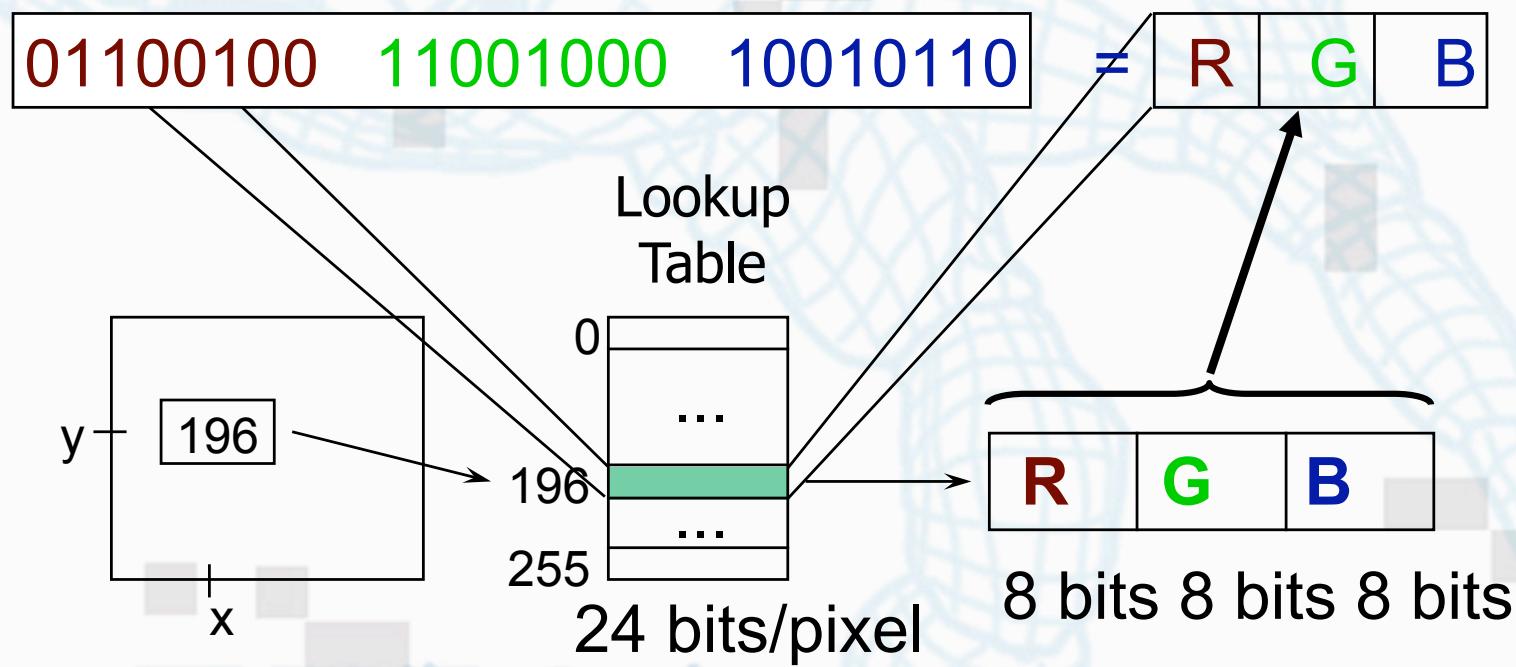
## 2.5.2 Cor indexada

- Reduzem tamanho do raster
- 256 cores carregadas em uma tabela
- A paleta total continua a mesma (16.777.216 cores)
- Armazenamento total requerido:
  - Raster =  $(1.024 \times 1.024 \times 8)/8 = 1.048.576$  bytes
  - Tabela =  $(256 \times 24)/8 = 768$  bytes
  - Total = 1.049.344 bytes  $\approx$  1 megabyte

# 2.5 Cores

## – Esquema típico

- **196 = Conteúdo armazenado no endereço de memória no frame buffer correspondente ao pixel (x, y)**



## 2.5 Cores

- Cores são modificadas recriando entradas na tabela de cores
  - **glIndexi(int address);**
  - **glutSetColor(int color, GLfloat red, GLfloat green, GLfloat blue);**

## 2.5 Cores

### 2.5.3 Definição de atributos de cores

- `glClearColor(1.0, 1.0, 1.0, 1.0);`
- `glClearIndex(0.0);`
- `glColor3f(1.0, 0.0, 0.0);`



# Fim da Aula 6

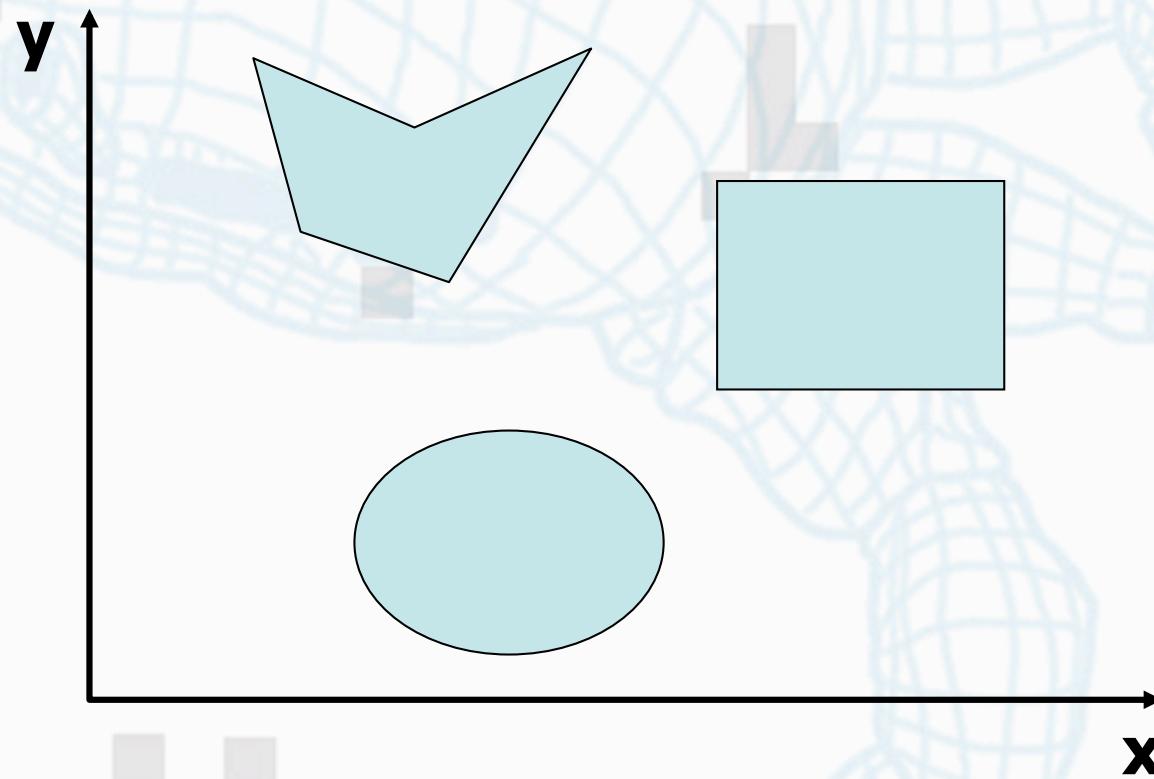
## 2.6 Visualização

### 2.6.0 Introdução

- O cenário está montado
  - Todos os modelos estão posicionados em coordenadas do mundo
- Como posicionar a câmera para ver todo o cenário ou parte do cenário?

# 2.6 Visualização

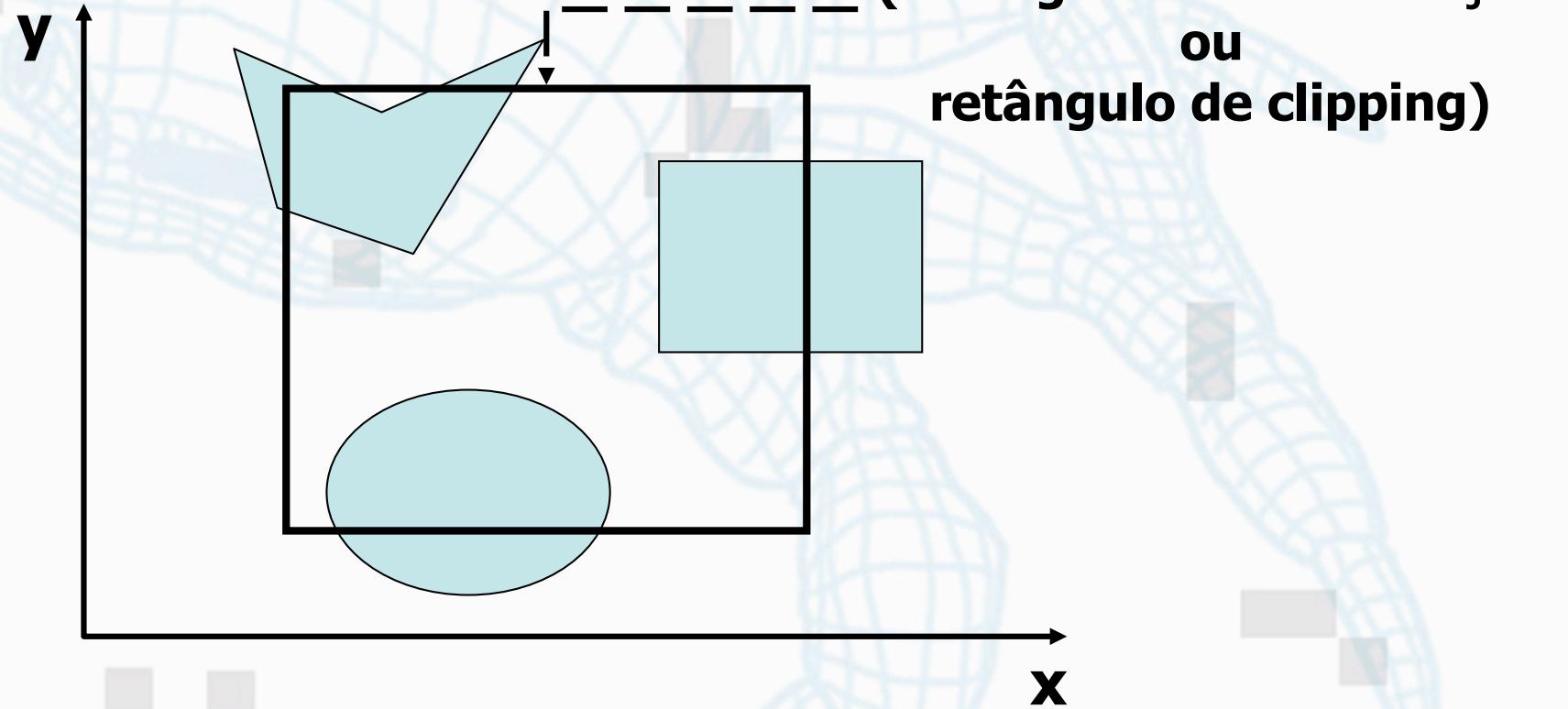
## 2.6.1 Visualização bidimensional – Cenário 2D



# 2.6 Visualização

## 2.6.1 Visualização bidimensional

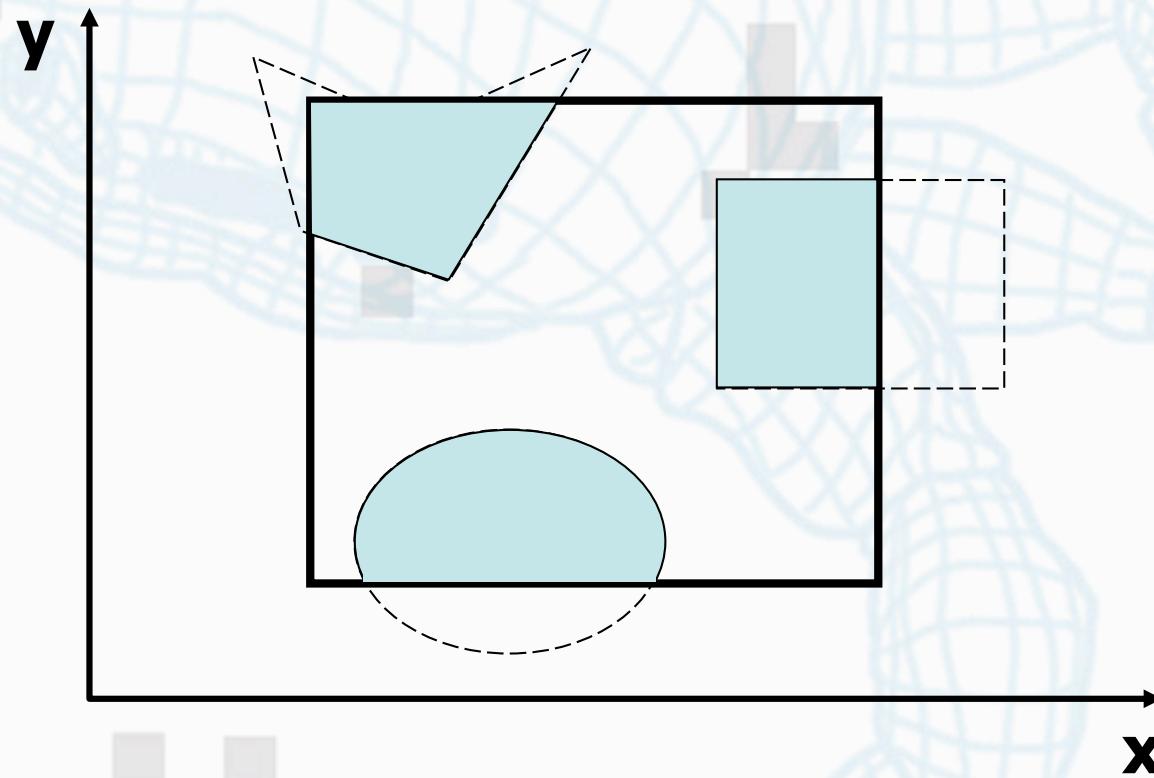
### – Cenário 2D + Janela



# 2.6 Visualização

## 2.6.1 Visualização bidimensional

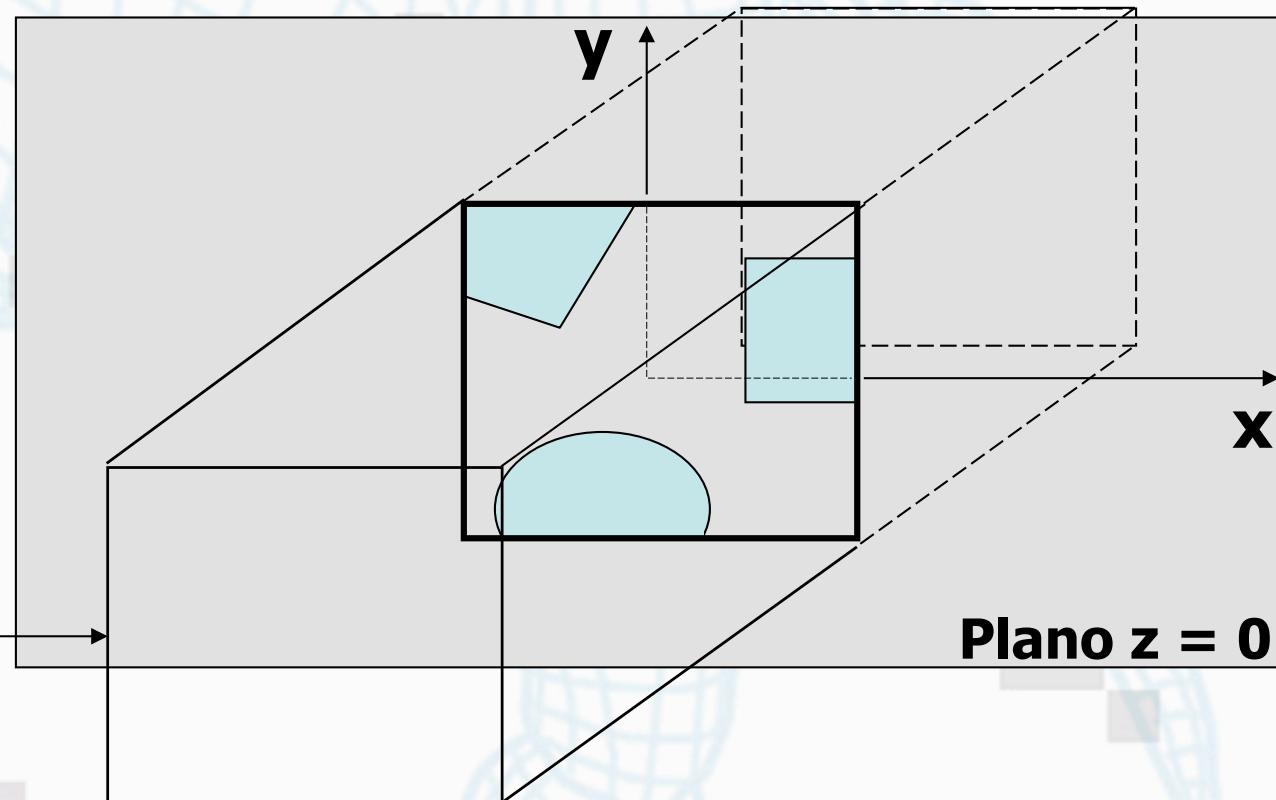
– Cenário 2D dentro da janela após recorte



# 2.6 Visualização

## 2.6.1 Visualização bidimensional – Cenário 2D é caso particular de 3D

Volume de visualização default do OpenGL  
 $2 \times 2 \times 2$



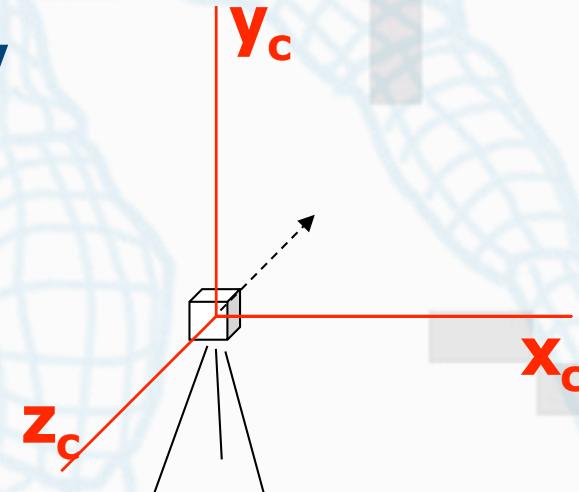
## 2.6 Visualização

### 2.6.2 A visualização ortográfica

- A visualização que aparece na janela 2D da transparência anterior é um caso particular de uma **projeção ortográfica**
  - Ponto  $(x, y, z)$  é projetado no plano  $z = 0$  como  $(x, y, 0)$
  - No Cenário 2D, todos os pontos estão em  $z=0$ 
    - Projeção não causa nenhuma alteração

## 2.6 Visualização

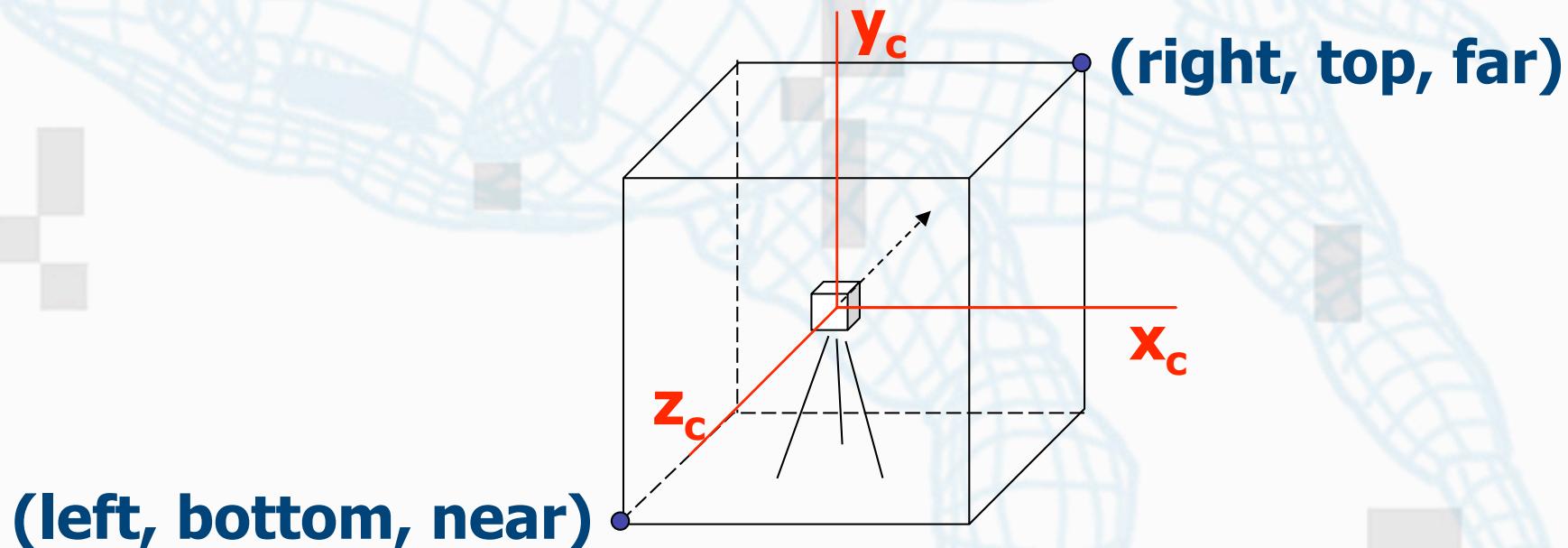
- **Comando do OpenGL**  
`glOrtho( GLdouble left, GLdouble right,  
GLdouble bottom, GLdouble top,  
GLdouble near, GLdouble far );`
- **Câmera do OpenGL por default**
  - Posicionada no plano xy
  - Apontando para z-



## 2.6 Visualização

### – Comando glOrtho

- Todo objeto dentro do volume de visualização é projetado no plano  $x_cy_c$



## 2.6 Visualização

### – Comando **gluOrtho2D**

```
gluOrtho2D(GLdouble left, GLdouble right,  
GLdouble bottom, GLdouble top );
```

## 2.6 Visualização

### 2.6.3 Modos de matrizes

- **Matrizes são parte do estado do sistema**
  - Permanecem ativas até serem alteradas
  - **Matrizes de Transformação são concatenadas**
- **Matrizes Model-View**
  - **Afetam a construção do cenário**
- **Matrizes de Projeção**
  - **Afetam a forma de visualização do cenário**

## 2.6 Visualização

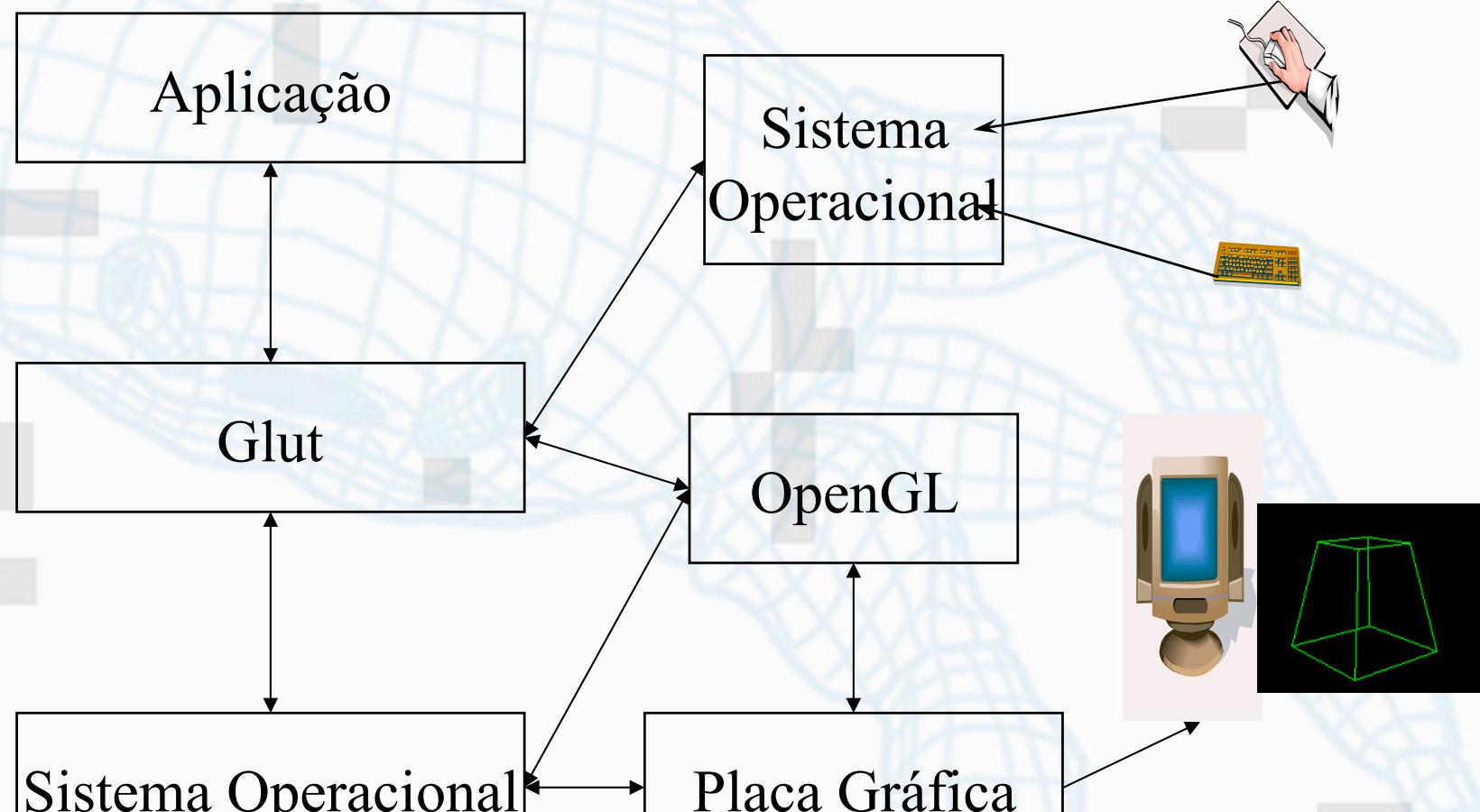
- Seqüência para visualização 2D  
**glMatrixMode(GL\_PROJECTION);**  
**glLoadIdentity();**  
**gluOrtho2D(0.0, 500.0, 0.0, 500.0)**  
**glMatrixMode(GL\_MODELVIEW)**

## 2.7 Funções de Controle

### 2.7.0 Problema de interface com sistema Operacional e de janelas

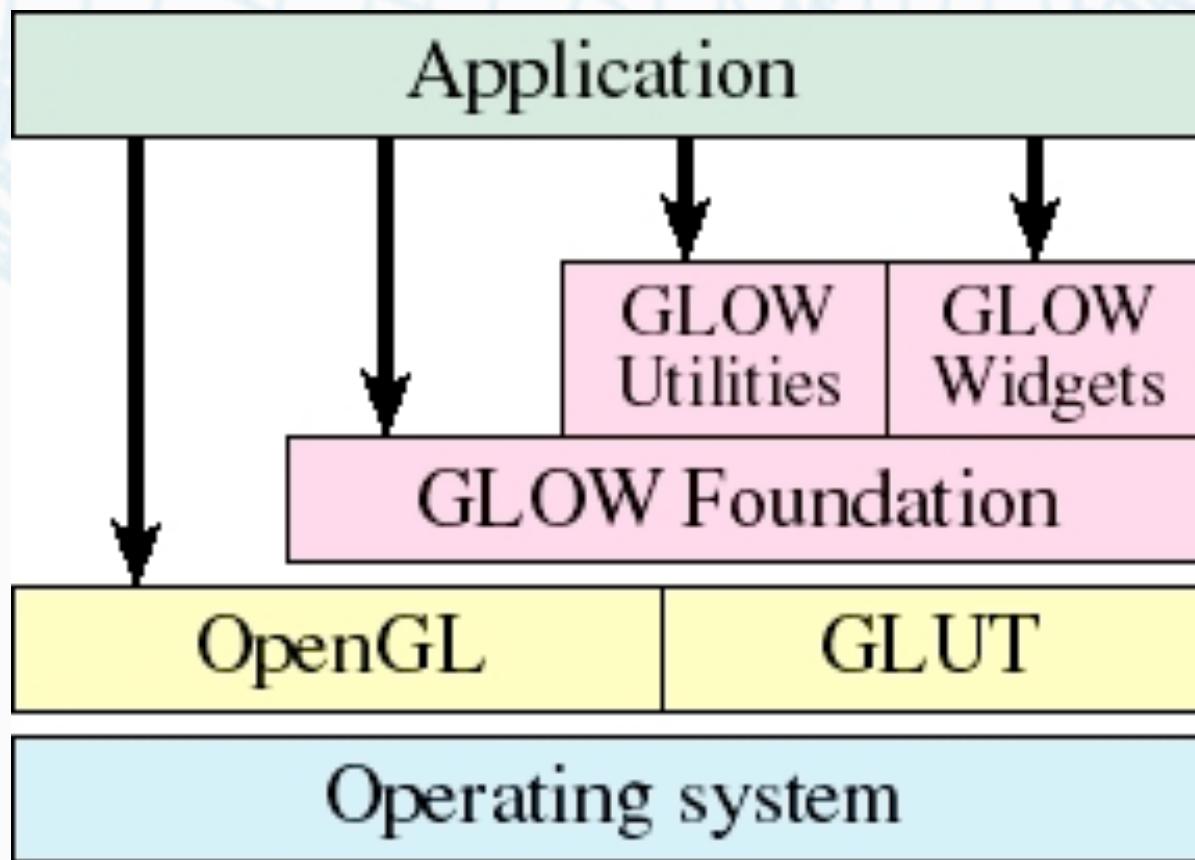
- GLUT
  - Interface comum com múltiplas plataformas
  - Gerencia
    - Windows
    - double buffering,
    - text rendering,
    - keyboard input
    - menus

## 2.7 Funções de Controle



## 2.7 Funções de Controle

- **GLOW ( OpenGL Object-oriented Windowing toolkit )**



## 2.7 Funções de Controle

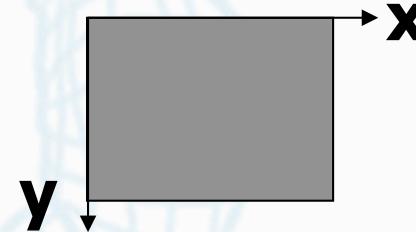
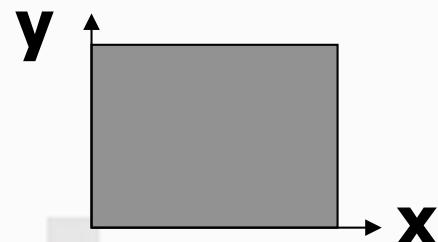
### 2.7.1 Interação com o sistema de janelas

- **Window ou screen window**
  - Área retangular no monitor
  - Mostra o conteúdo do frame buffer
  - Posições de pontos em coordenadas de window (unidades de pixels)
- **Sistema de janelas (window system)**
  - Ambiente multijanelas disponibilizado por
    - Sistema X Window
    - Microsoft Window

## 2.7 Funções de Controle

### – Janela Gráfica

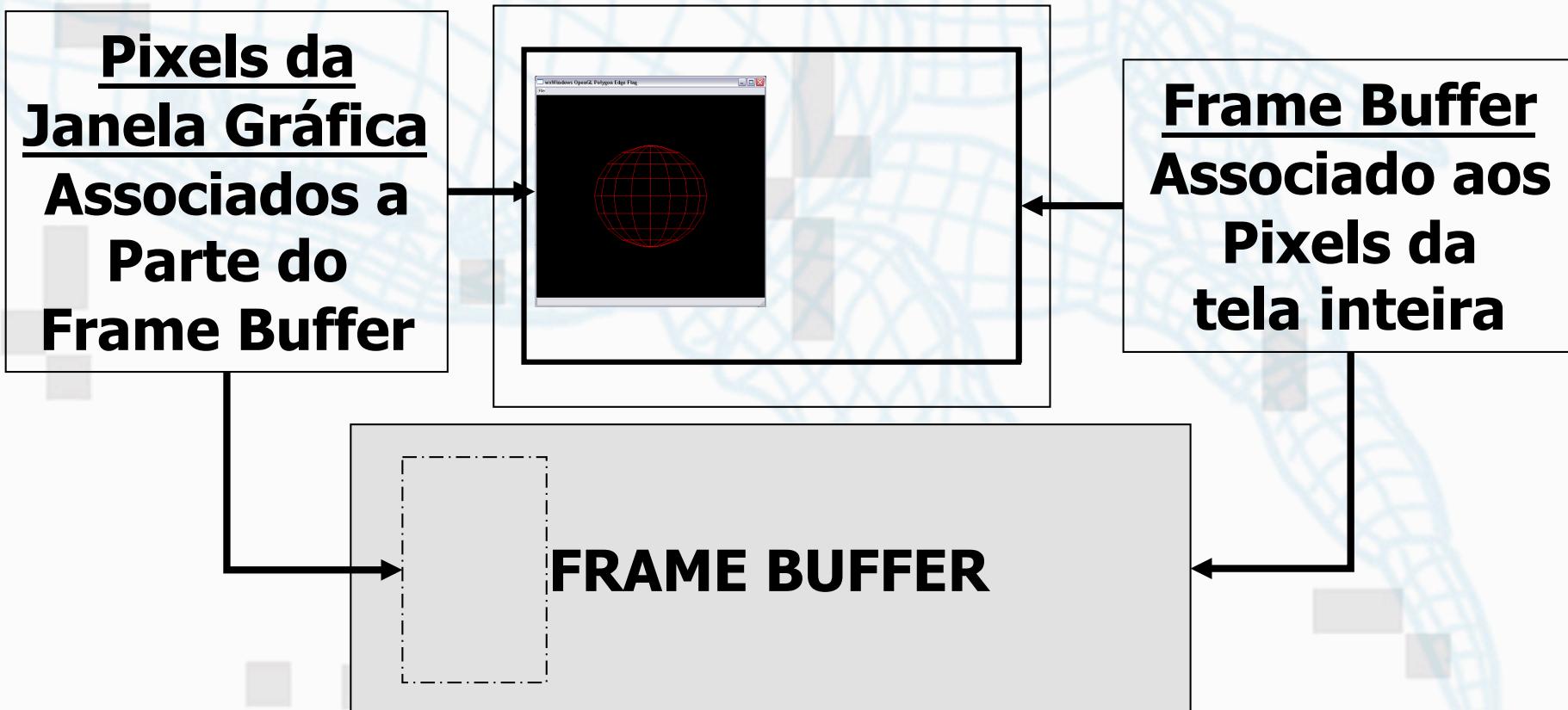
- Um tipo de janela gerenciado pelo sistema de janelas
- Gráficos podem ser exibidos
- Objetos gráficos podem ser renderizados
- Posições são relativas a um canto da janela
  - Geralmente o canto inferior esquerdo: (0,0)
  - Sistema de janelas usam canto superior esquerdo



## 2.7 Funções de Controle

### – Janela Gráfica

- Pode ser menor ou igual ao screen



## 2.7 Funções de Controle

- **Antes de abrir janela**
  - Deve haver interação entre o sistema de janelas e o OpenGL
  - GLUT inicia essa interação através de **glutInit(int \*argcp, char \*\*argv);**
    - Inicializa a biblioteca GLUT
    - Negocia uma sessão com o window system
  - Definir características da janela
    - glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH | GLUT\_DOUBLE);**
    - glutInitWindowSize(480, 640);**
    - glutInitWindowPosition(0, 0);**

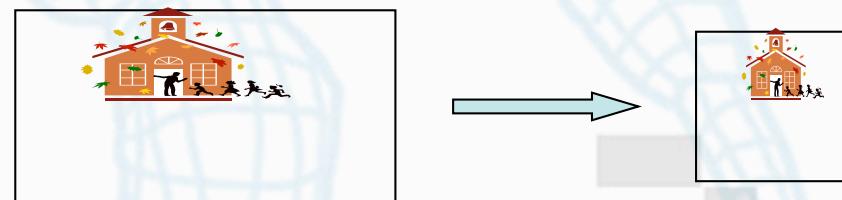
## 2.7 Funções de Controle

- GLUT abre uma OpenGL window  
`glutCreateWindow(char *title);`
  - Tamanho 480 x 640
  - Posicionada no canto superior esquerdo
  - Usa RGB diretamente no Framebuffer ao invés de cor indexada (GLUT\_INDEX)
  - Usa um buffer de profundidade para remoção de superfícies ocultas
  - Usa double buffering ao invés de single buffering (GLUT\_SINGLE)
  - Default seria (GLUT\_RGB, nenhum buffer de profundidade, GLUT\_SINGLE)

## 2.7 Funções de Controle

### 2.7.2 Razão de Aspecto e Viewports

- **Aspect ratio de um retângulo**
  - W/H (largura sobre altura)
- **Por default o conteúdo da janela de visualização no cenário é mapeada na OpenGL window aberta pelo sistema de janelas**
- **Se os aspect ratios desses retângulos forem diferentes**
  - Imagem distorcida



## 2.7 Funções de Controle

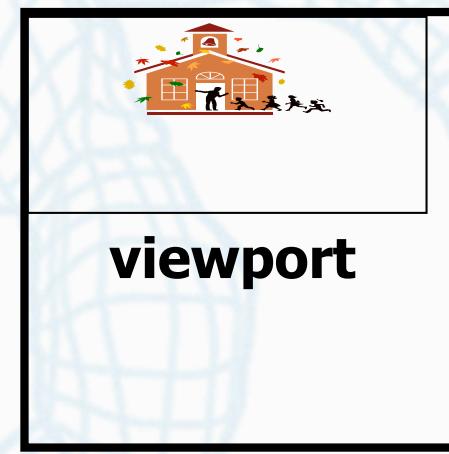
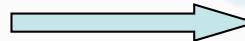
### – Definir Viewport

`glViewport()`

- Define subregião da OpenGL window para fazer o mapeamento
- Pode ser usado para dividir a OpenGL window par múltiplas visões do cenário



Janela de visualização



OpenGL window

## 2.7 Funções de Controle

```
void glViewport(GLint x, GLint y,  
               GLsizei width, GLsizei height);
```

- (*x*, *y*): canto inferior esquerdo da viewport
- *width* and *height* : largura e altura da viewport
- Default: (0, 0, *winWidth*, *winHeight*)

## 2.7 Funções de Controle

### 2.7.3 As funções main, display e myinit

- Código exemplo

```
#include <GL/glut.h>
void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Sierpinski Gasket");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

## 2.7 Funções de Controle

### – **glutMainLoop()**

- Entra no laço de processamento de eventos do GLUT
- Deve ser chamada no máximo uma vez no programa
- Ela nunca retorna depois de chamada
- Responsável pela chamada de funções registradas como callbacks

## 2.7 Funções de Controle

– **glutDisplayFunc(void (\*func)(void))**

- **func: nome da função que será chamada sempre que o sistema de janelas determinar que a OpenGL window precisa ser redesenhada**
- **Exemplo: ponha o código de desenho do Sierpinski Gasket na função display**

## 2.7 Funções de Controle

### – **myinit()**

- **Usada para colocar comandos de inicialização de variáveis de estado do OpenGL de visualização e atributos que preferirmos separar do processo de display**

## 2.7 Funções de Controle

- **#include <GL/glut.h>**
  - Adiciona os cabeçalhos para
    - Biblioteca GLUT
    - Biblioteca OpenGL (gl.h)
    - Biblioteca OpenGL Utility (glu.h)

## 2.7 Funções de Controle

### 2.7.4 Estrutura de um programa

- Ver programa do Selante de Sierpinski

## 2.7 Funções de Controle

## 2.8 O Programa Gasket

**Código Fonte**

**Executável**

## 2.7 Funções de Controle

### 2.9 Polígonos e recursão

**Código Fonte**

**Executável**

# Sumário do Curso

## 2. Programação Gráfica (cont.)

### 2.10 O Gasket Tridimensional (A6)

**2.10.1 Uso de pontos 3D**

**2.10.2 Uso de polígonos em 3D**

**2.10.3 Remoção de superfícies ocultas**



# Fim da Aula 7