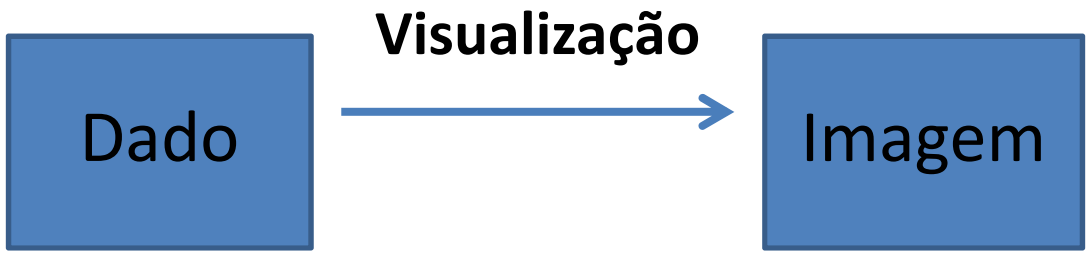
CIV 2802
PUC-Rio
Março, 2014

Computação Gráfica
Cor / OpenGL

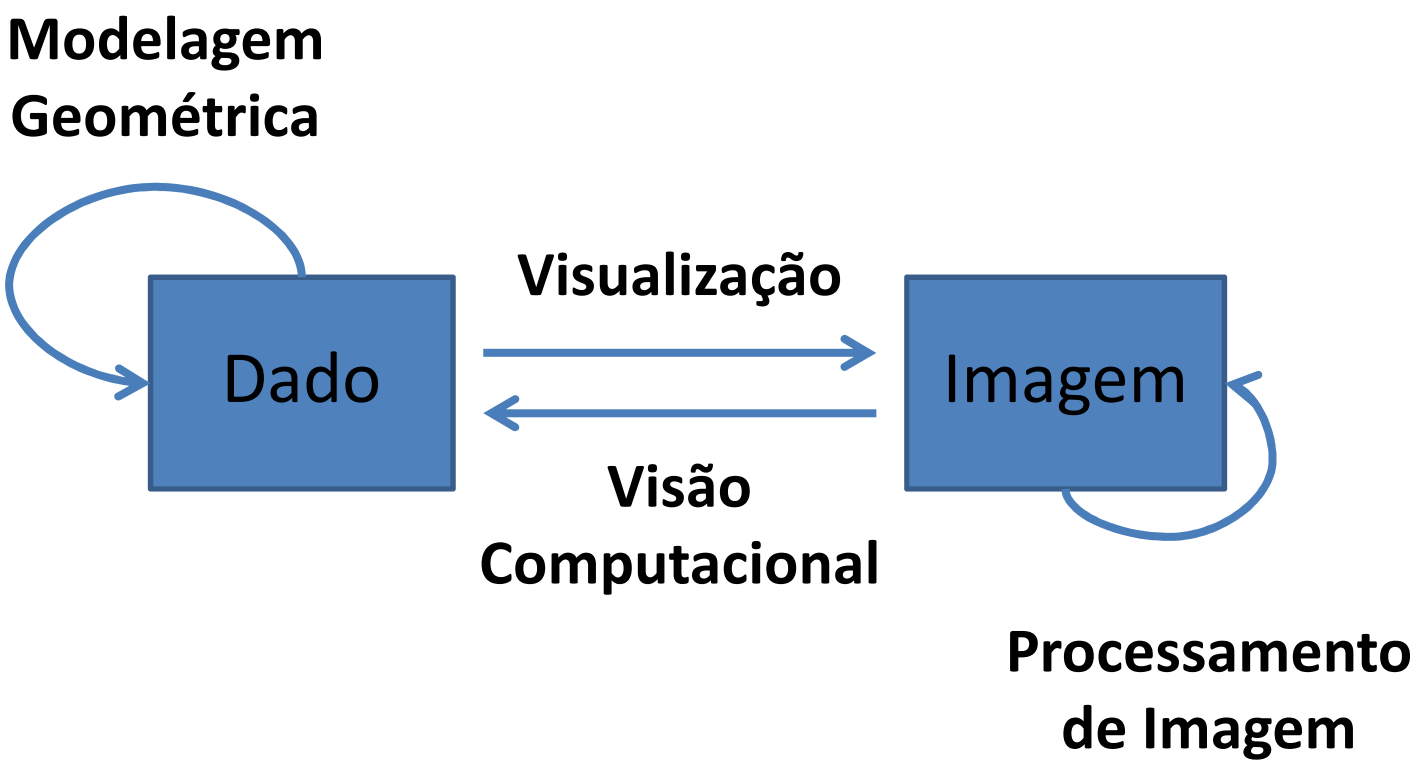
André Pereira
Luiz Fernando Martha

Computação Gráfica

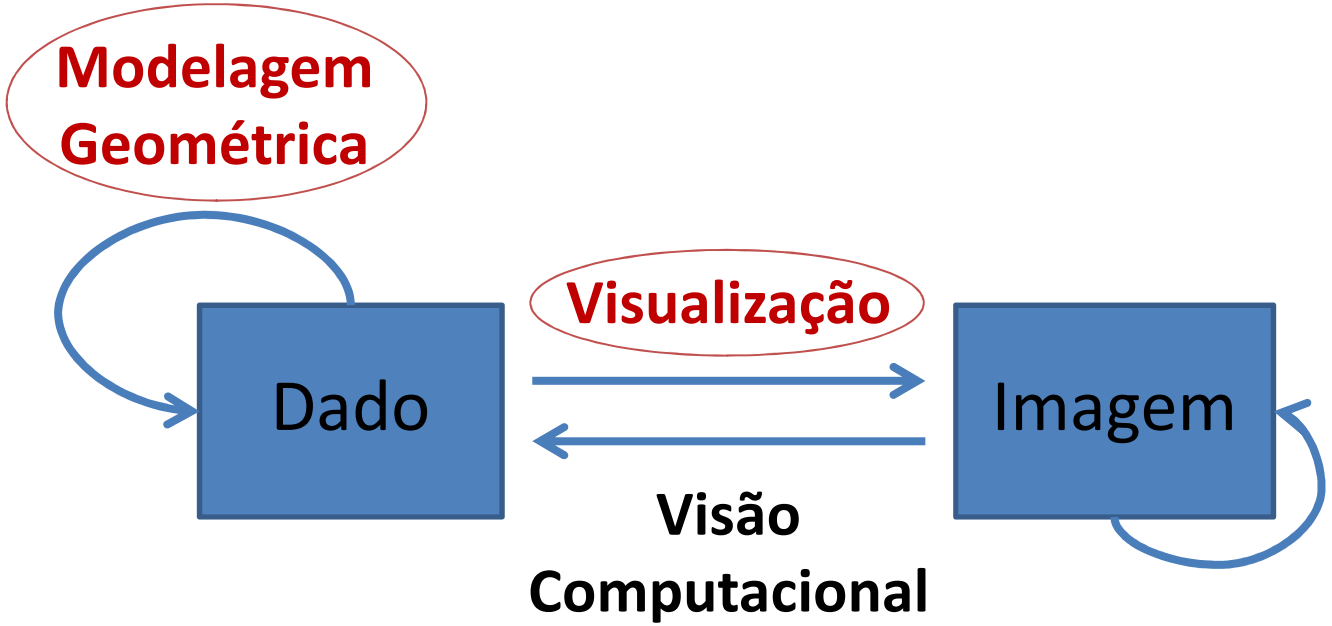
Computação Gráfica



Computação Gráfica



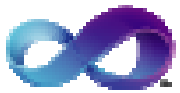
Computação Gráfica

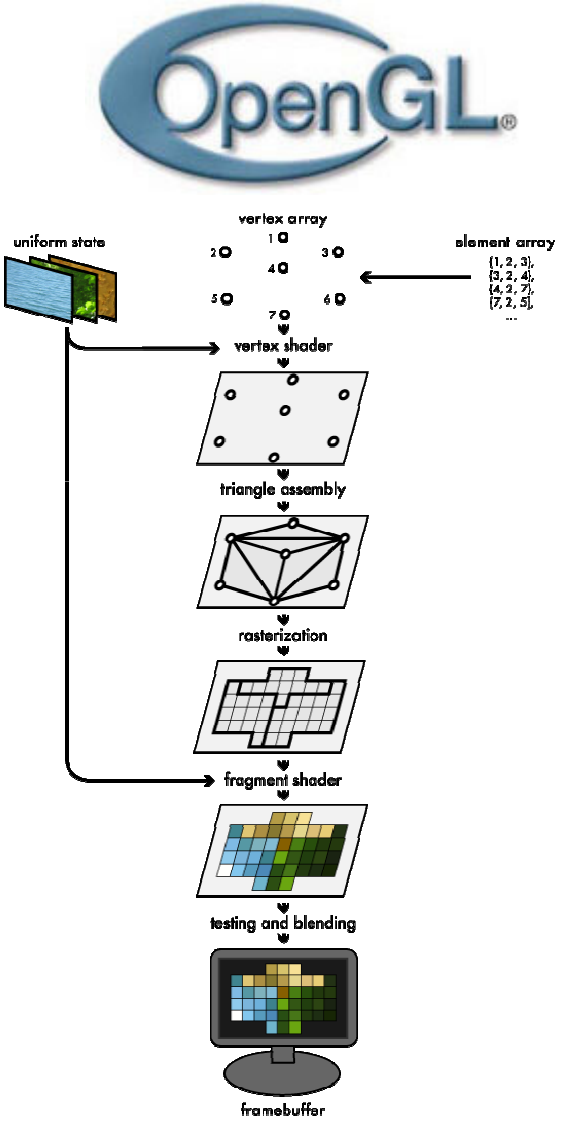


- Modelagem Geométrica
- Geração de Malha
- Geometria Computacional
- Técnicas de Visualização (Pós-processamento)

Ambiente de Desenvolvimento

C++





COR

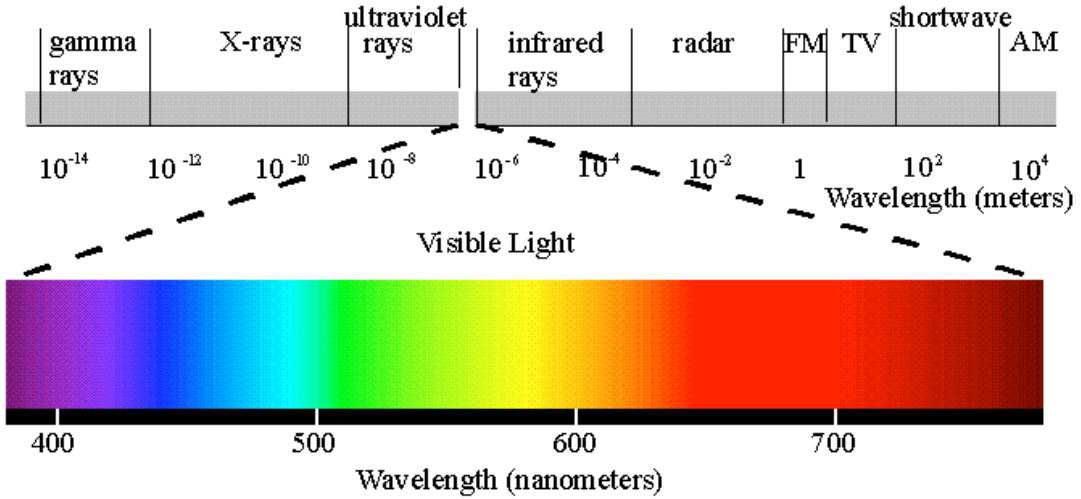
COR

Como se percebe e como se quantifica a cor?

COR

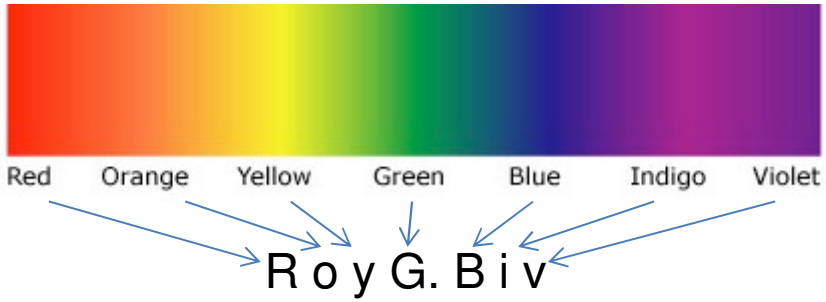
Como se percebe e como se quantifica a cor?

Luz: radiação em uma faixa particular de comprimentos de onda.

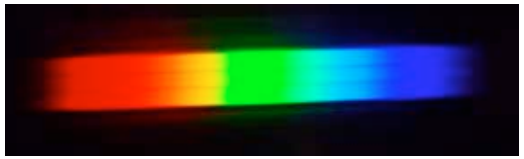


Luz de um único comprimento de onda é chamada de **monocromática**

Luz em uma única frequência

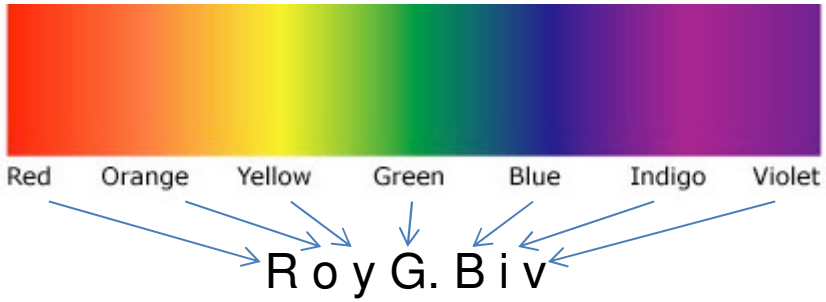


Brilhoso e distinto em aparência

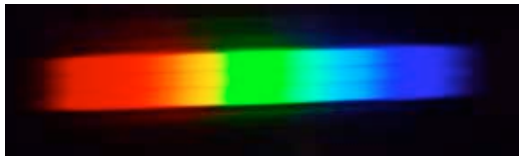


Reprodução apenas, não é um espectro de cor real.

Luz em uma única frequência



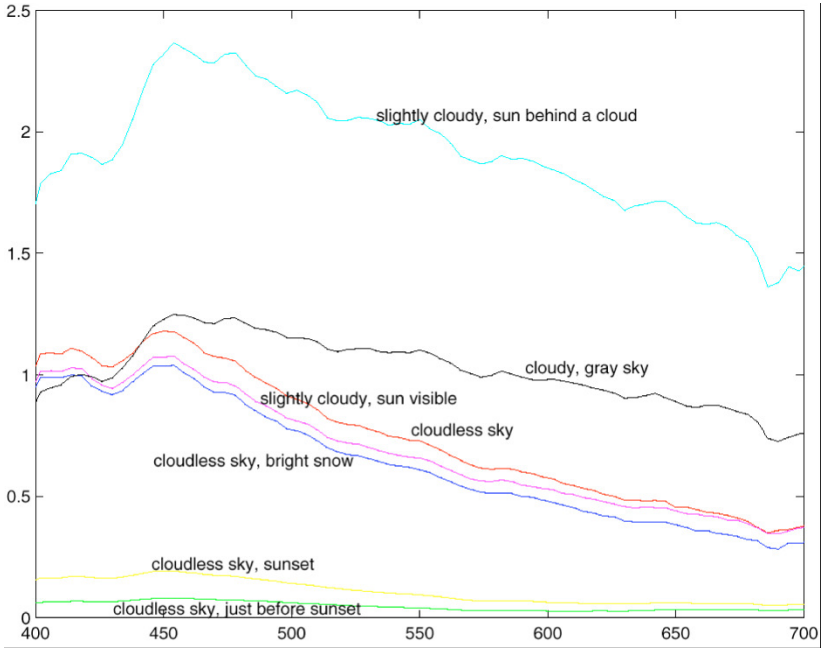
Brilhoso e distinto em aparência



Reprodução apenas, não é um espectro de cor real.

A maioria das cores vistas são mistura de luz de vários comprimentos de onda.

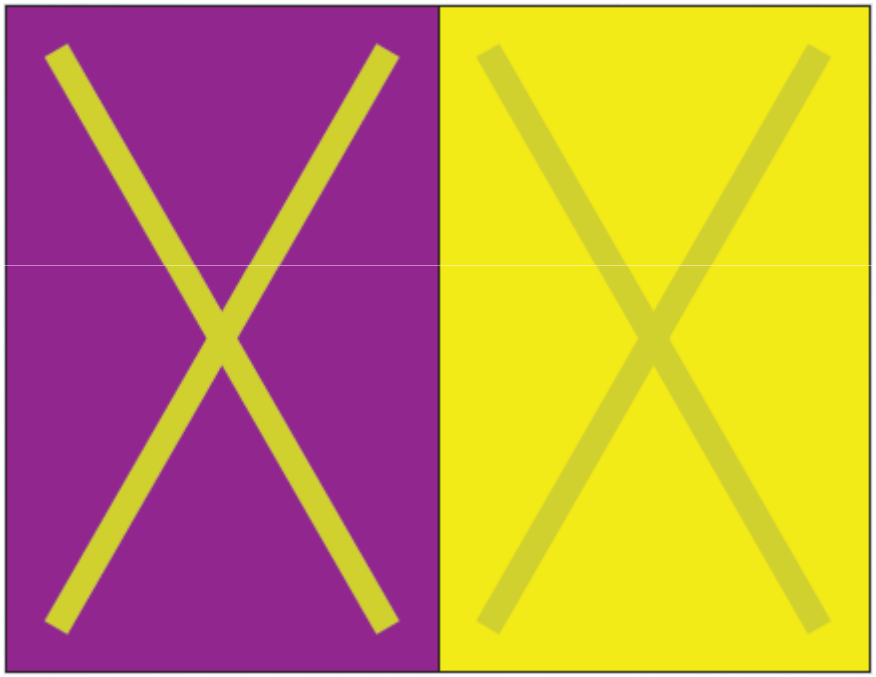
Curvas descrevem a composição espectral $\Phi(\lambda)$ de estímulos.



Percepção x Medição

Não “vemos” o espectro de luz.

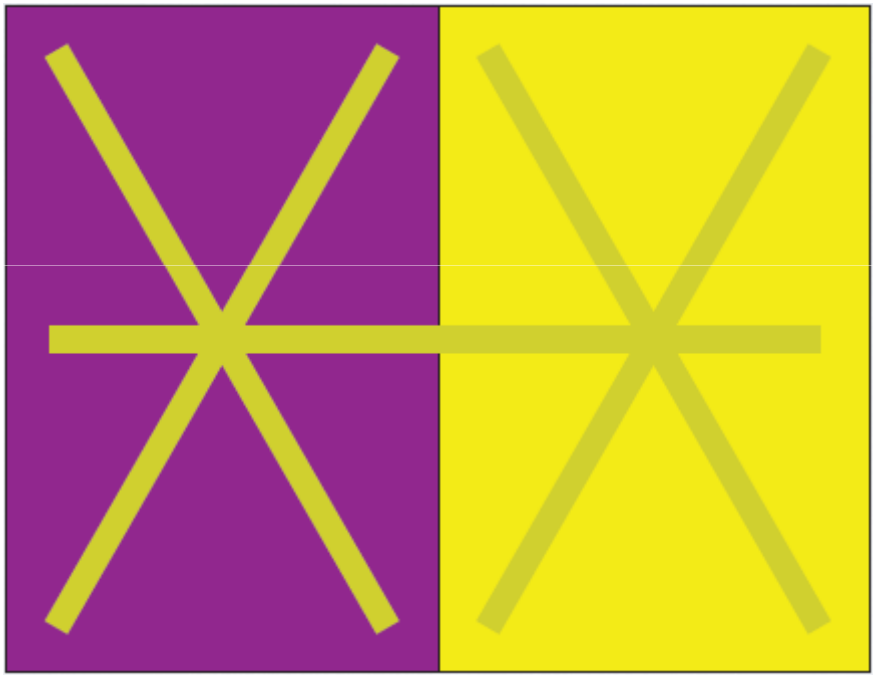
Tudo é relativo!



Percepção x Medição

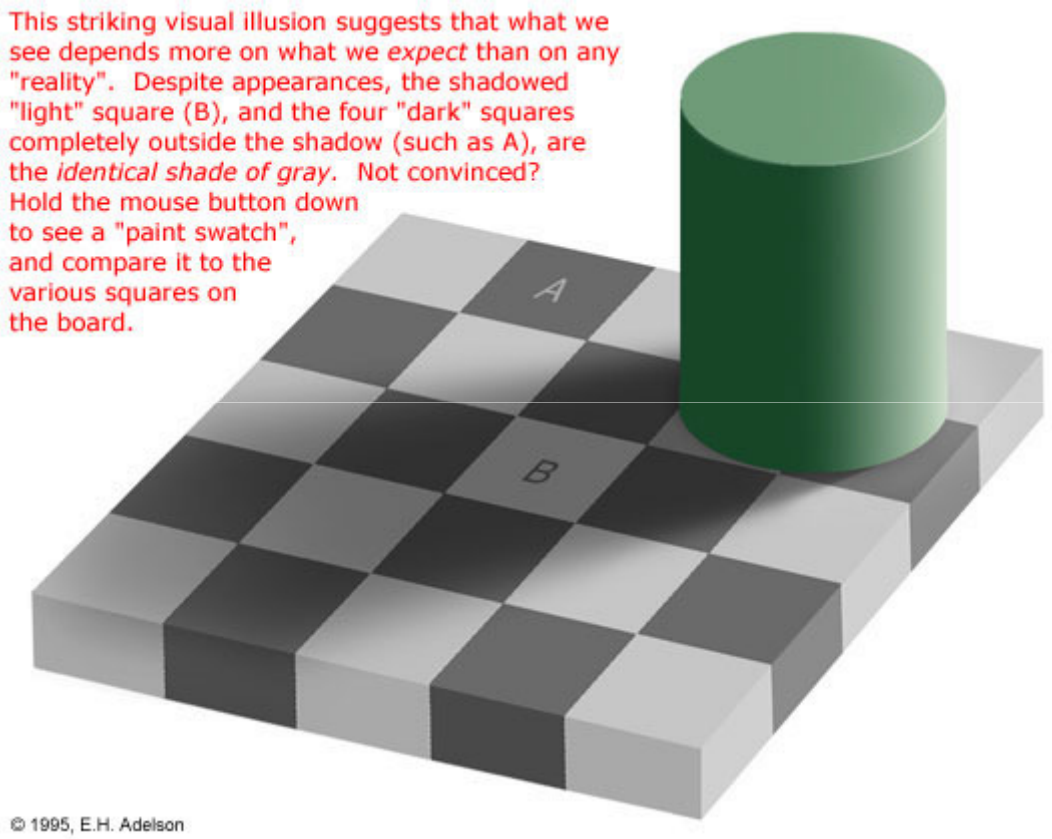
Não “vemos” o espectro de luz.

Tudo é relativo!



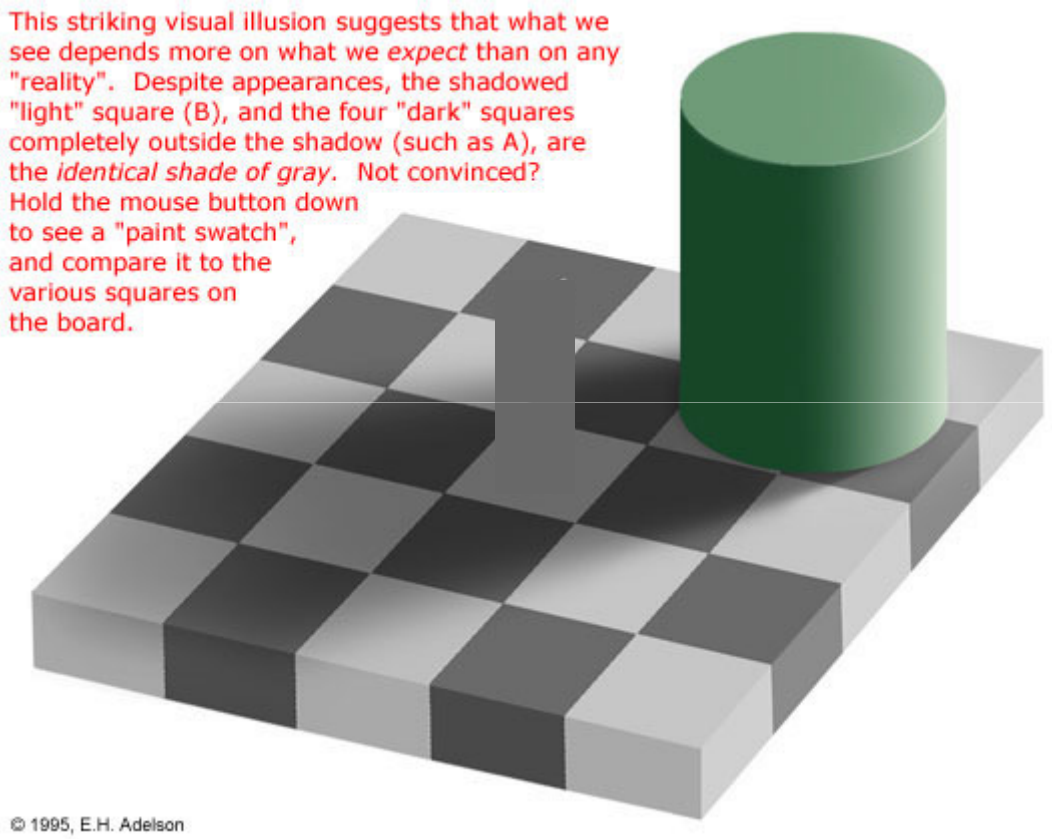
Percepção

O olho não vê valores de intensidade



Percepção

O olho não vê valores de intensidade



Olho como um Sensor

O olho grava cor por 3 medições
Nós podemos “enganar” com combinação de 3 sinais.

Assim, os dispositivos de vídeo (monitores, impressoras, etc) podem gerar cores perceptíveis com a mistura de 3 primárias.

A resposta ao estímulo Φ1 é (L1, M1, S1)
A resposta ao estímulo Φ2 é (L2, M2, S2)

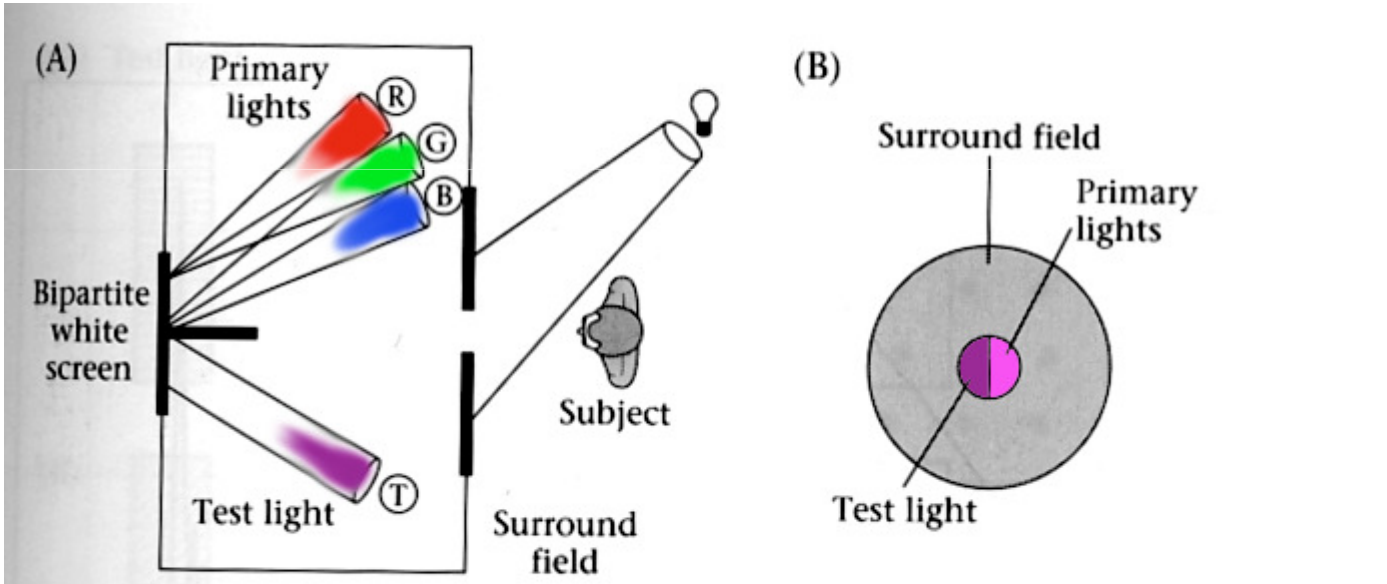
Então a resposta a Φ1+ Φ2 é (L1+L2, M1+M2, S1+S2)

A resposta para n Φ1 é (n L1, n M1, n S1)

Sistema que obedece **superposição** e **escala** é chamado de **Sistema Linear**.

Mistura Aditiva

Dadas três primárias estamos de acordo sobre p1, p2, p3
Combine luz de entrada genérica com $\Phi = \alpha\, p1 + \beta\, p2 + \gamma\, p3$
Negativa não percebida, mas pode adicionar primária na luz de teste.
A cor é agora descrita por α, β, γ
Ex: monitor de computador [R,G,B]

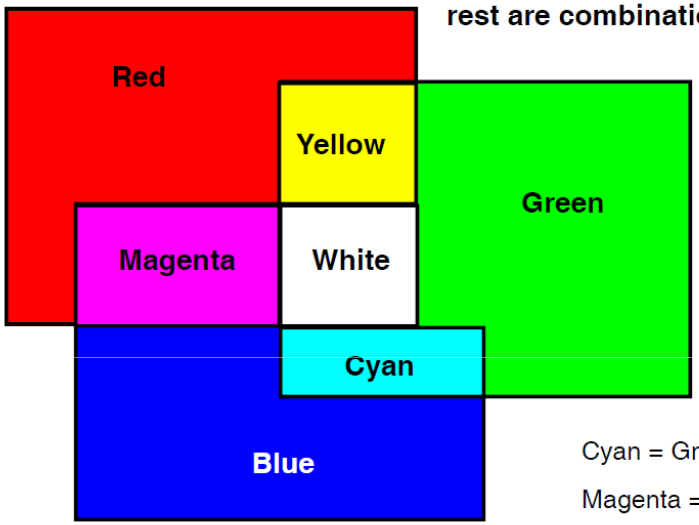


4.10 THE COLOR-MATCHING EXPERIMENT. The observer views a bipartite field and adjusts the intensities of the three primary lights to match the appearance of the test light. (A) A top view of the experimental apparatus. (B) The appearance of the stimuli to the observer. After Judd and Wyszecki, 1975.

Representação de COR

The Framebuffer Uses Additive Colors (RGB)

Red, Green, and Blue are provided. The rest are combinations of those three.

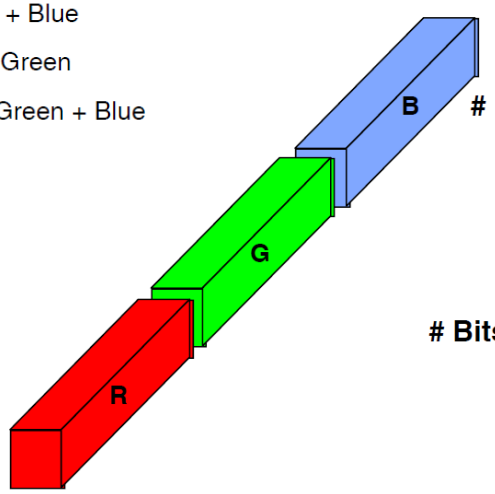


Cyan = Green + Blue

Magenta = Red + Blue

Yellow = Red + Green

White = Red + Green + Blue

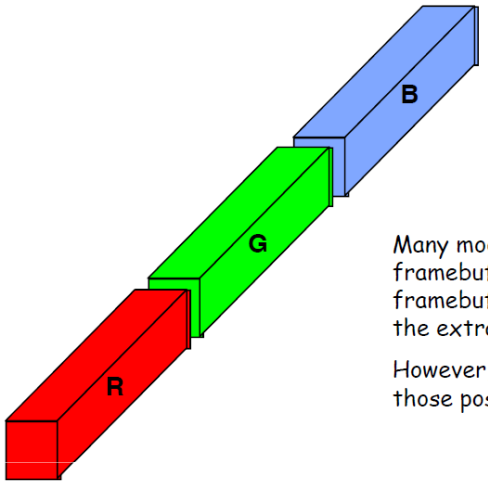


# Bits/color	# Intensities per color
8	$2^8 = 256$
10	$2^{10} = 1024$
12	$2^{12} = 4096$

# Bits/pixel	Total colors:
24	$2^{24} = 16.7$ M
30	$2^{30} = 1$ B
36	$2^{36} = 69$ B

The Framebuffer:
Floating Point Color Storage

- ***16- or 32-bit floating point for each color component***



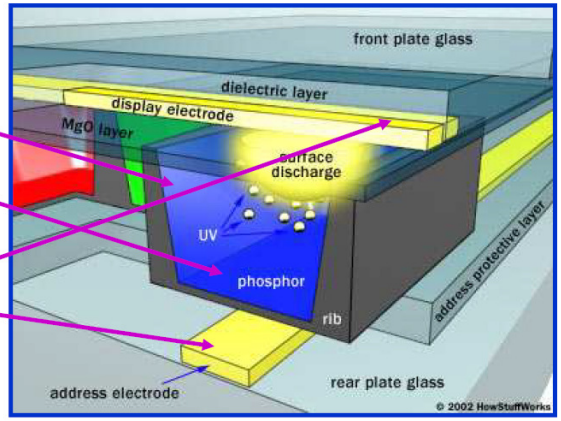
Why so much?

Many modern algorithms do arithmetic on the framebuffer color components, or treat the framebuffer color components as data. They need the extra precision during the arithmetic.

However, the display system cannot display all of those possible colors.

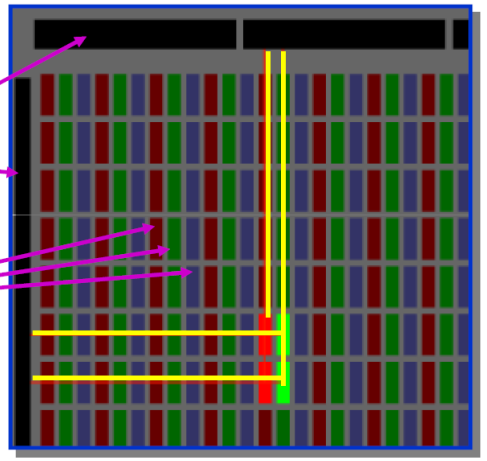
Displaying Color on a Plasma Monitor

- **Gas cell**
- **Phosphor**
- **Grid of electrodes**

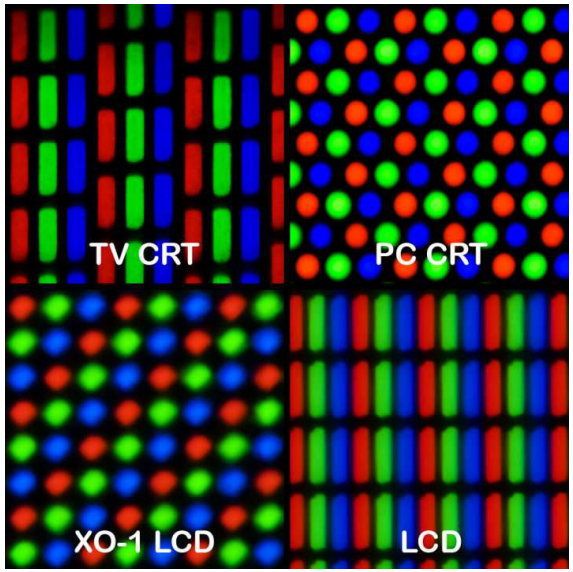


Displaying Color on a Computer Graphics LCD Monitor

- Grid of electrodes
- Color filters



Source: http://electronics.howstuffworks.com

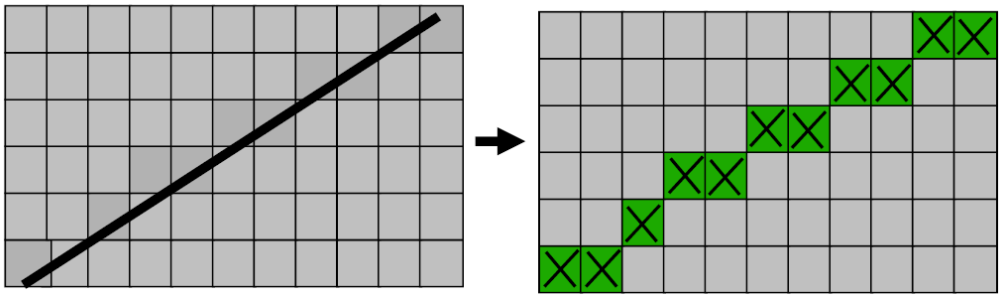


Display Resolution

- ***Pixel*** resolutions (1280x1024, 1600x1200, 1920x1152 are common on the desktop)
- Screen size (13", 16", 19", 21" are common)
- Human acuity: 1 arc-minute is achieved by viewing a 19" monitor with 1280x1024 resolution from a distance of ~40 inches

Rasterization

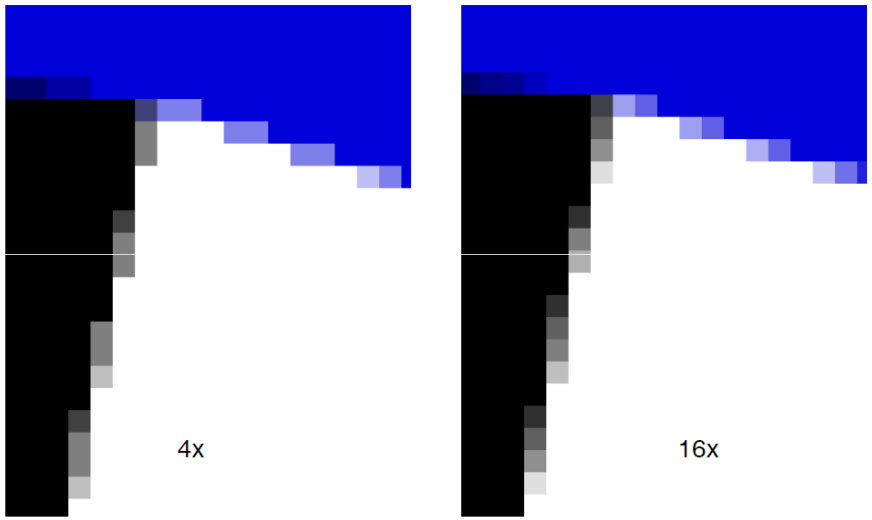
- Turn screen space vertex coordinates into pixels that make up lines and polygons
- A great place for custom electronics
- Anti-aliasing is often built-in



Anti-aliasing is Implemented by Oversampling within Each Pixel

No AA 4x 16x

Anti-aliasing is Implemented by Oversampling within Each Pixel



4x 16x

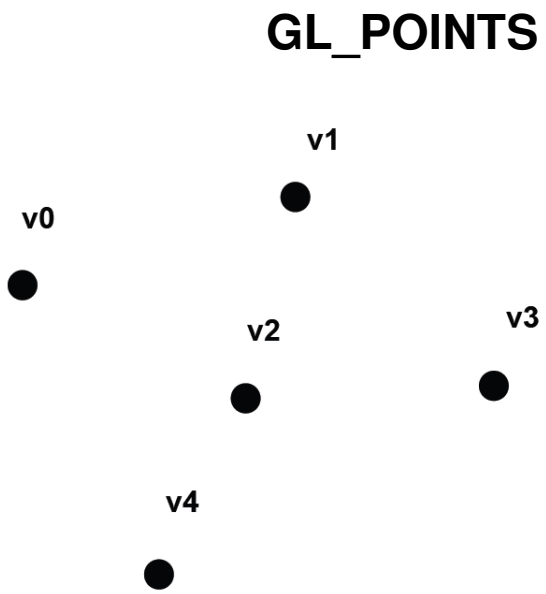
OpenGL

OpenGL

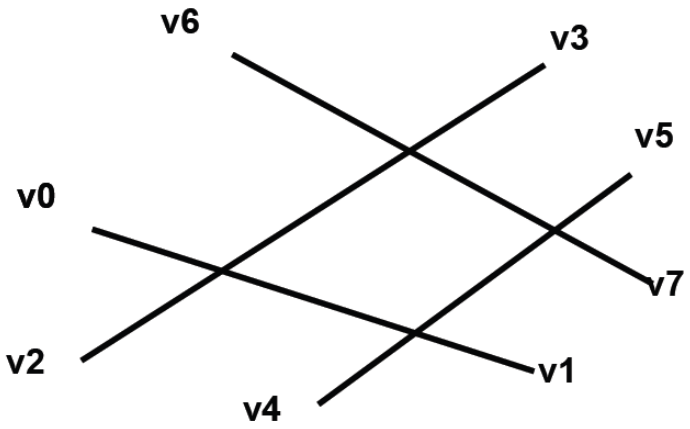
OpenGL
- Programming API (API) for hardware accelerated 2D/3D graphics
- Platform independent
- Generic
- Flexible
- Low level...

Drawing
- All drawing accomplished using 10 primitives
- Same basic principle

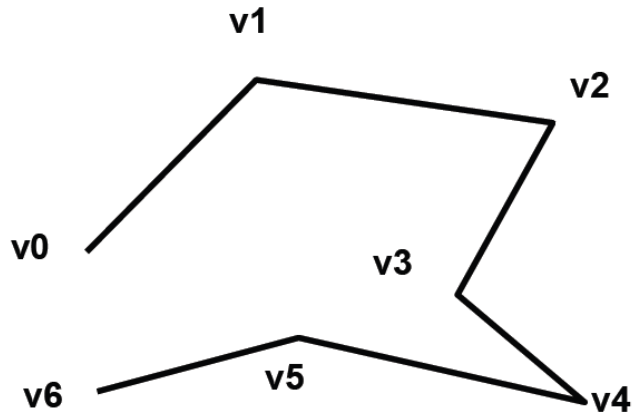
```
// Draw 4 points
glBegin(GL_POINTS)
glVertex2i(-50,-50)
glVertex2i(50,-50)
glVertex2i(50,50)
glVertex2i(-50,50)
glEnd()
```



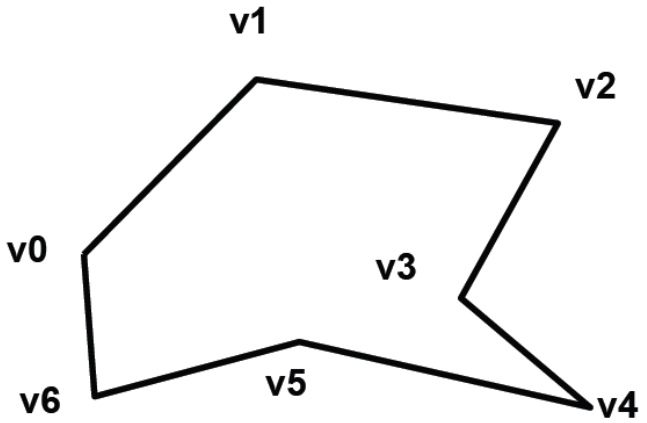
GL_LINES



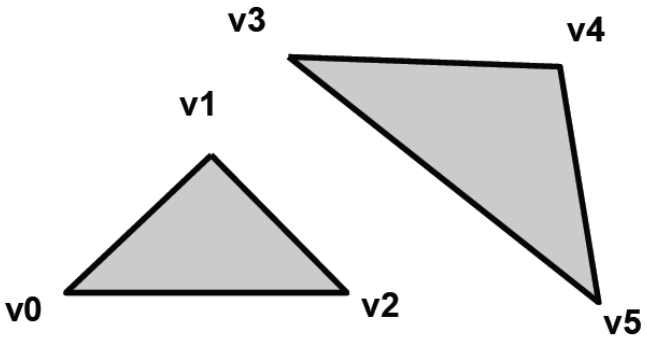
GL_LINE_STRIP



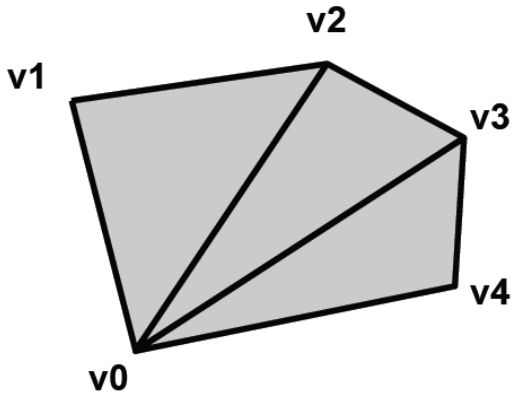
GL_LINE_LOOP



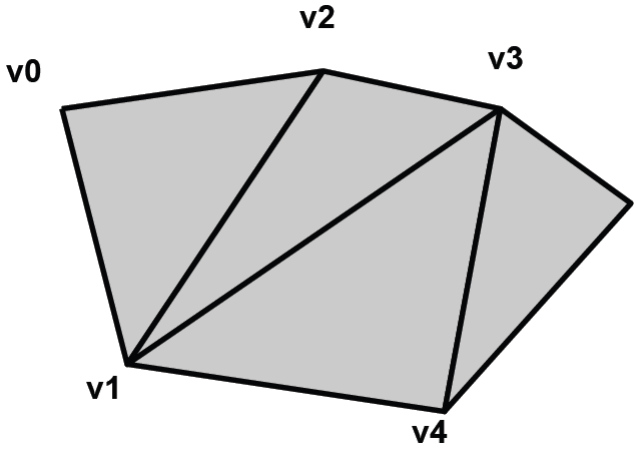
GL_TRIANGLES



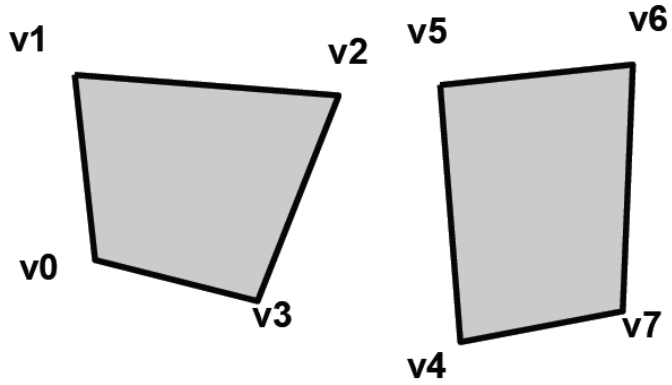
GL_TRIANGLE_FAN



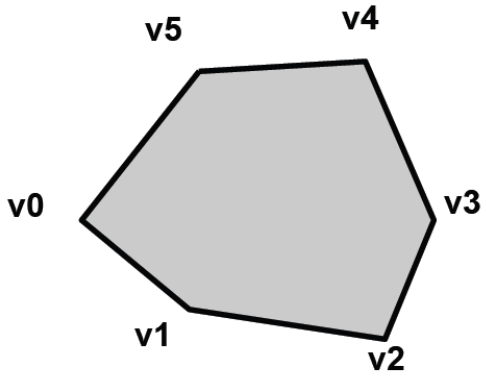
GL_TRIANGLE_STRIP



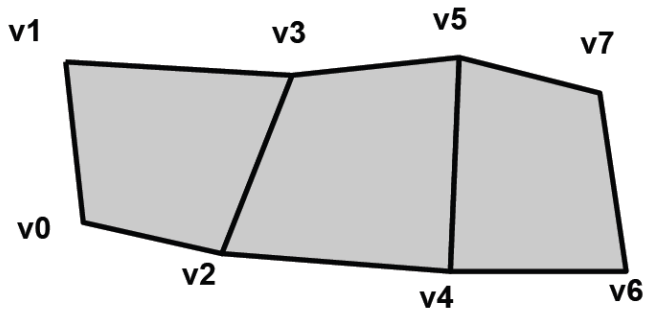
GL_QUADS



GL_POLYGON

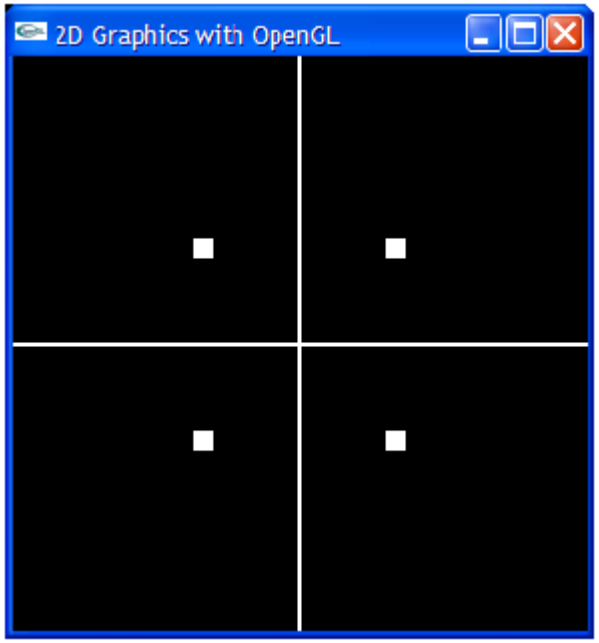


GL_QUAD_STRIP



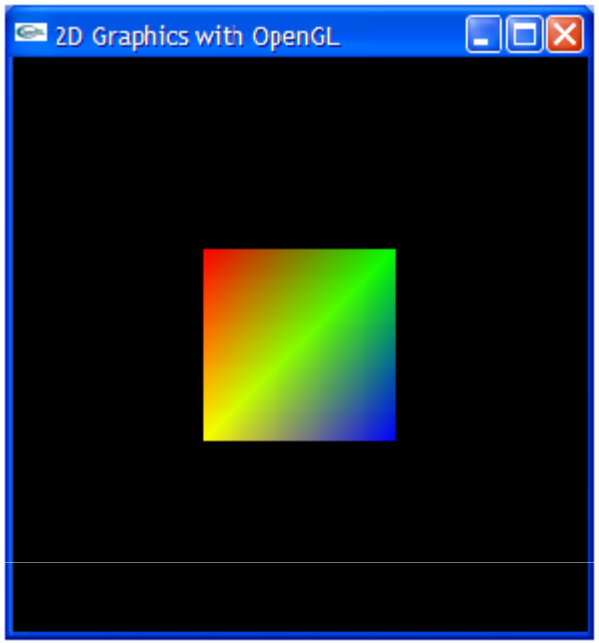
Primitive properties

- Points and lines
  - Outside glBegin()/glEnd()
  - line width, glLineWidth(2.0)
  - Point size
- Color
  - Given on a vertex level
  - Inside glBegin()/glEnd()
  - Given in RGB, where 1.0 max intensity and 0.0 is minimum intensity
  - Color is interpolated between vertices



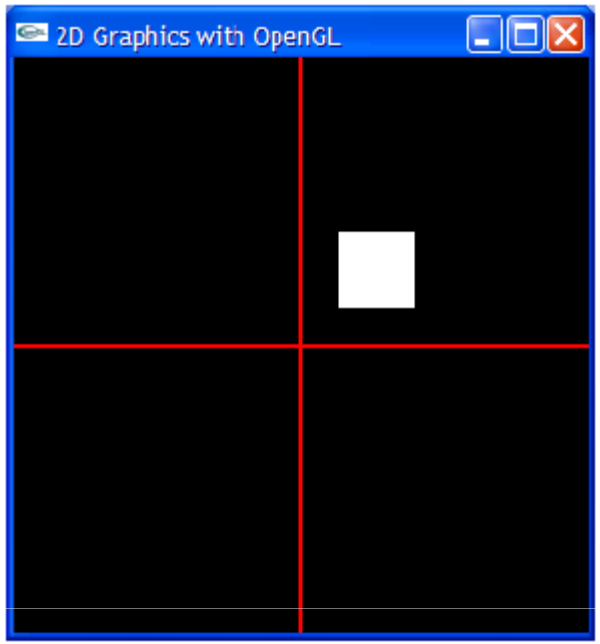

```
// Set the line width
glLineWidth(2.0)
glBegin(GL_LINES)
glVertex2i(-1000,0);
glVertex2i(1000,0);
glVertex2i(0,-1000);
glVertex2i(0, 1000);
glEnd()
// Set point size
glPointSize(5)
glBegin(GL_POINTS)
glVertex2i(-50, -50)
glVertex2i( 50, -50)
glVertex2i( 50, 50)
glVertex2i(-50, 50)
glEnd()
```

```
glBegin(GL_QUADS)
glColor3f(1.0, 0.0, 0.0) // Red color
glVertex2i(-50, -50)
glColor3f(0.0, 1.0, 0.0) // Green color
glVertex2i( 50, -50)
glColor3f(0.0, 0.0, 1.0) // Blue color
glVertex2i( 50, 50)
glColor3f(1.0, 1.0, 0.0) // Yellow color
glVertex2i(-50, 50)
glEnd()
```



Geometric transformations

- Transformations are important in computer graphics
  - Translation
  - Rotation
  - Scaling
- OpenGL
  - Transformation matrices implemented in hardware
  - Model matrix - glMatrixMode(GL_MODELVIEW)
  - Project matrix - glMatrixMode(GL_PROJECTION)



Initialising matrices

```
// Initialise model view matrix to identity
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
```

Translation

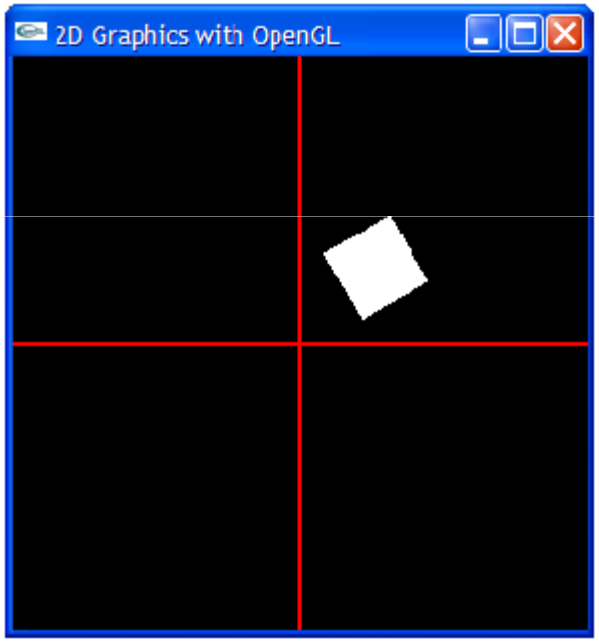
- Translating coordinate systems
- glTranslatef(x, y, z)
- Current matrix in multiplied by a translation matrix

```
glTranslatef(40.0,40.0,0.0)
glBegin(GL_QUADS)
glColor3f(1.0,1.0,1.0)
glVertex2i(-20,-20)
glVertex2i(20,-20)
glVertex2i(20,20)
glVertex2i(-20,20)
glEnd()
```

Rotation

- Rotates coordinate system
- glRotatef(angle, axis_x, axis_y, axis_z)
- Right-hand rule
- Positive Z-axis out of the screen

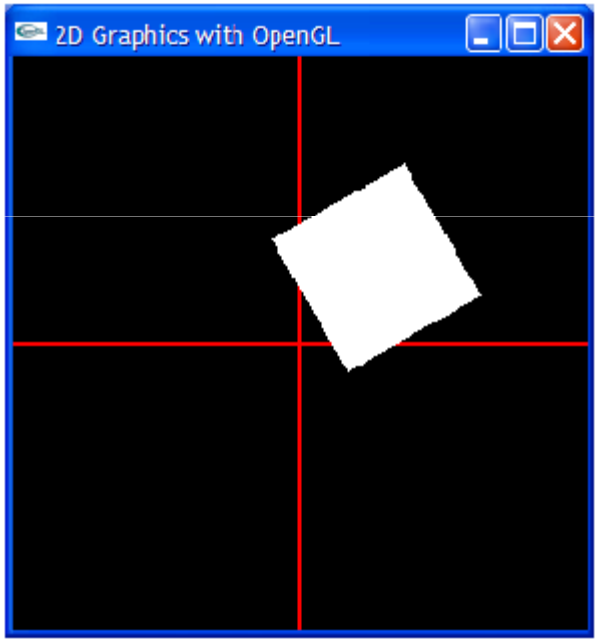
glTranslatef(40.0,40.0,0.0)
glRotatef(30.0,0.0,0.0,1.0)
glBegin(GL_QUADS)
glColor3f(1.0,1.0,1.0)
glVertex2i(-20,-20)
glVertex2i(20,-20)
glVertex2i(20,20)
glVertex2i(-20,20)
glEnd()



Scaling

- Scales current coordinate system
- glScalef(scale_x, scale_y, scale_z)

```
glTranslatef(40.0,40.0,0.0)
glRotatef(30.0,0.0,0.0,1.0)
glScalef(2.0,2.0,0.0)
glBegin(GL_QUADS)
glColor3f(1.0,1.0,1.0)
glVertex2i(-20,-20)
glVertex2i(20,-20)
glVertex2i(20,20)
glVertex2i(-20,20)
glEnd()
```



Problem with current method

- Matrices constantly needs initialising
- Difficult implement hierarchical tranformations
- Many matrix multiplications

OpenGL Matrix stack

- Stack of matrices
- Top is the current matrix
- If a matrix is added it is assigned the values of the top level matrix.
  - glPushMatrix()
- Matrices can be discarded using glPopMatrix()
- Reduce the matrix multiplications
- Speeds up the code
- Implemented in hardware

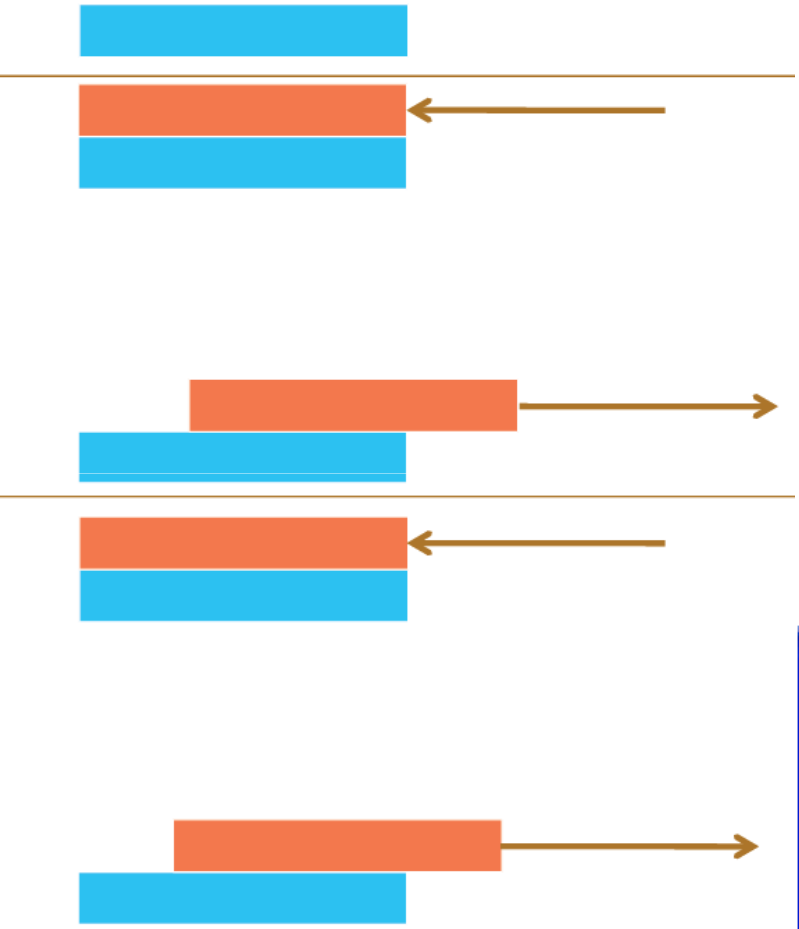
glPushMatrix()/glPopMatrix()

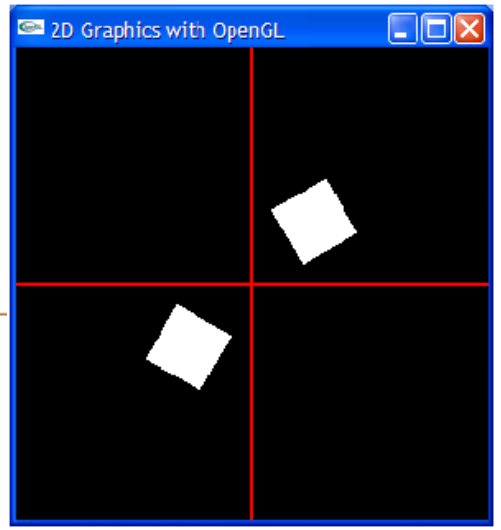
```
glPushMatrix()
glTranslatef(40.0, 40.0, 0.0)
glRotatef(30.0, 0.0, 0.0, 1.0)

glBegin(GL_QUADS)
glColor3f(1.0, 1.0, 1.0)
glVertex2i(-20, -20)
glVertex2i( 20, -20)
glVertex2i( 20,  20)
glVertex2i(-20,  20)
glEnd()
glPopMatrix()
```

```
glPushMatrix()
glTranslatef(-40.0, -40.0, 0.0)
glRotatef(-30.0, 0.0, 0.0, 1.0)

glBegin(GL_QUADS)
glColor3f(1.0, 1.0, 1.0)
glVertex2i(-20, -20)
glVertex2i( 20, -20)
glVertex2i( 20,  20)
glVertex2i(-20,  20)
glEnd()
glPopMatrix()
```





Drawing in the screen buffer

- Must be cleared for every frame
- glClear(GL_COLOR_BUFFER_BIT)
- Background color
  - glClearColor(red, green, blue)
- Double buffering
  - Reduces flickering
  - All drawing in back buffer
  - Switch between front and back buffer after drawing

Projection and screen view

- The projection matrix maps model coordinates to screen coordinates
- glMatrixMode(GL_PROJECTION)
- 2D = Orthographic projection
  - gluOrtho2D(left, right, top, bottom)

Initialising project matrix

```
// Initiate project matrix
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
// Create a 2D projection matrix
gluOrtho2D(0, width, 0, height)
// Initialise the modelview matrix to identity
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
```

Viewport

- Defines where in a window the drawing is to be done
  - glViewport(x,y,width,height)
- Enables multiple views in a single window
- Must be updated when window is resized