CCI 36 – Computação Gráfica

OpenGL – Parte 1

Instituto Tecnológico de Aeronáutica

Prof. Carlos Henrique Q. Forster – Sala 121 IEC

forster@ita.br

Luiz Felipe Simões Hoffmann

Tópicos da Aula

- OpenGL
- Histórico
- OpenGL Moderno (*Pipeline* de Renderização)
- Shaders (OpenGL *Shading Language* – GLSL)
- Exemplo
- Curva de Bézier

Referências

Dave Shreiner, Graham Sellers, John Kessenich, Bill Licea-Kane. OpenGL Programming Guide, 8th ed., Pearson Education, 2013.

Richard S. Wright Jr., Nicholas Haemel, Graham Sellers, Benjamin Lipchak. OpenGL Super Bible, 5th ed., Pearson Education, 2011.

Joey de Vries. Learn OpenGL, Joey de Vries, 2015.

OpenGL (*Open Graphics Library*)

O OpenGL é uma API (*Application Programming Interface*) para hardware gráfico. Essa API é constituída por um conjunto de funções destinadas ao desenvolvimento de gráficos e imagens de alta qualidade.

Na verdade, é uma especificação definida e mantida pelo grupo Khronos. Pode ser implementada em vários tipos diferentes de hardware ou inteiramente em software.

O OpenGL atua como um sistema cliente-servidor. Ou seja, um programa (cliente) emite comandos que são processados pelo servidor (implementação do OpenGL).

É uma grande máquina de estados finita: uma coleção de variáveis que definem como OpenGL deve operar em um determinado instante.

Histórico

1.0 – (1992) Primeira versão.

2.0 – (2004) OpenGL *Shading Language* (GLSL). *Pipeline* programável.

3.0 – (2008) *Deprecation Model*

3.1 – (2009) Remoção da maioria dos recursos obsoletos.

3.2 – (2009) *Core and compatibility profiles*.

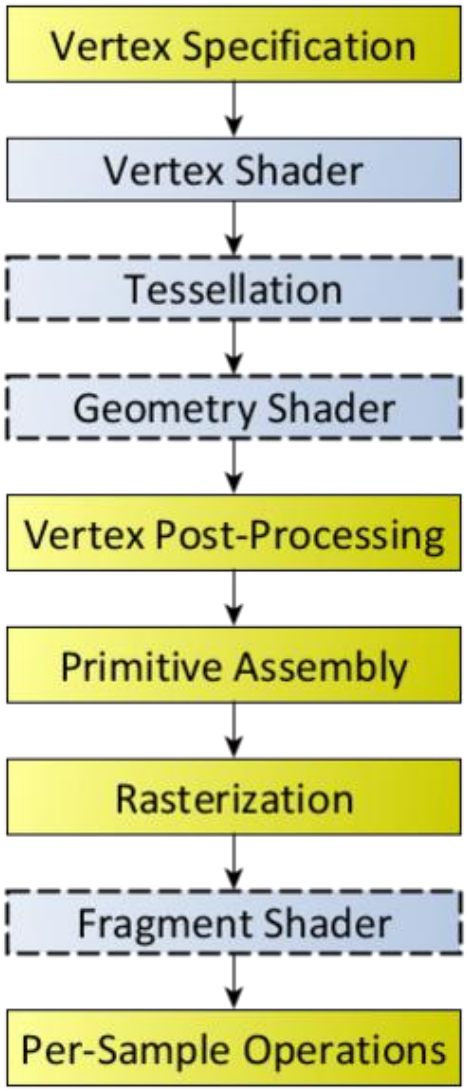
3.3 – (2010) *Backports* para funções da especificação 4.0.

4.6 – (2017) Versão atual.

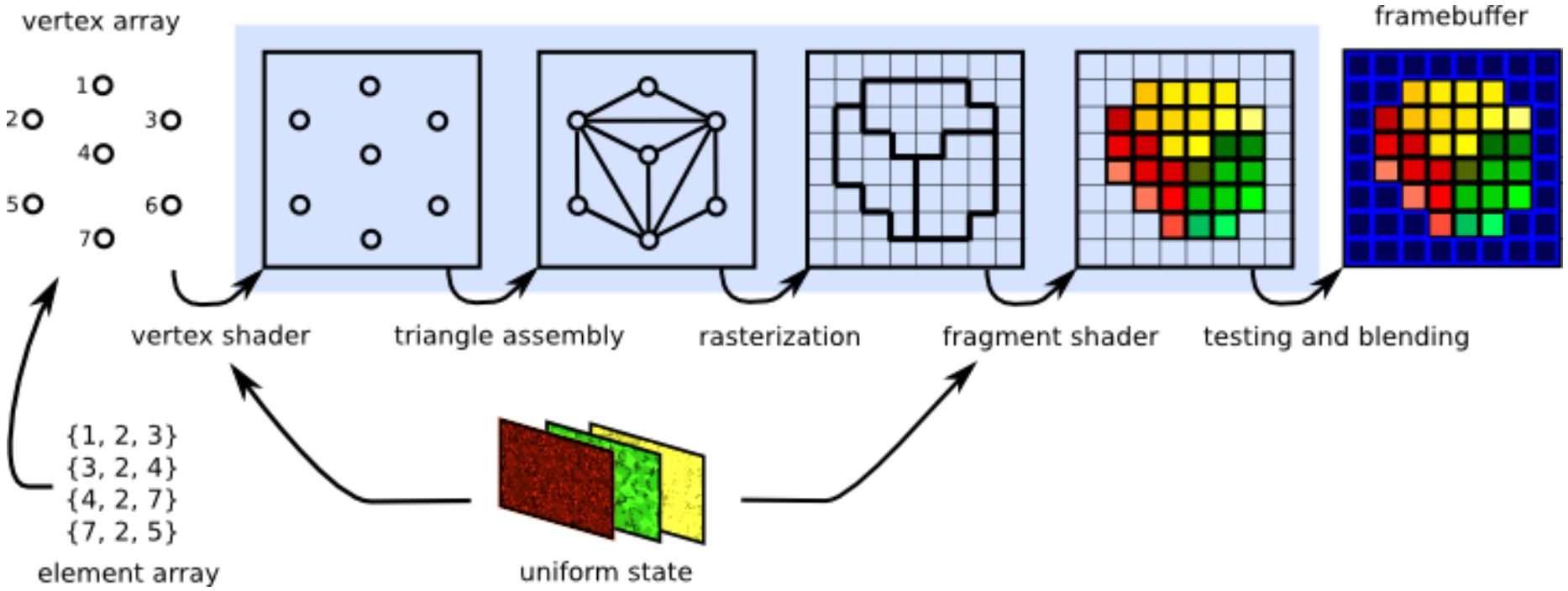
OpenGL Moderno

Pipeline de Renderização

(Fonte: *Rendering Pipeline Overview* – Khronos.org)

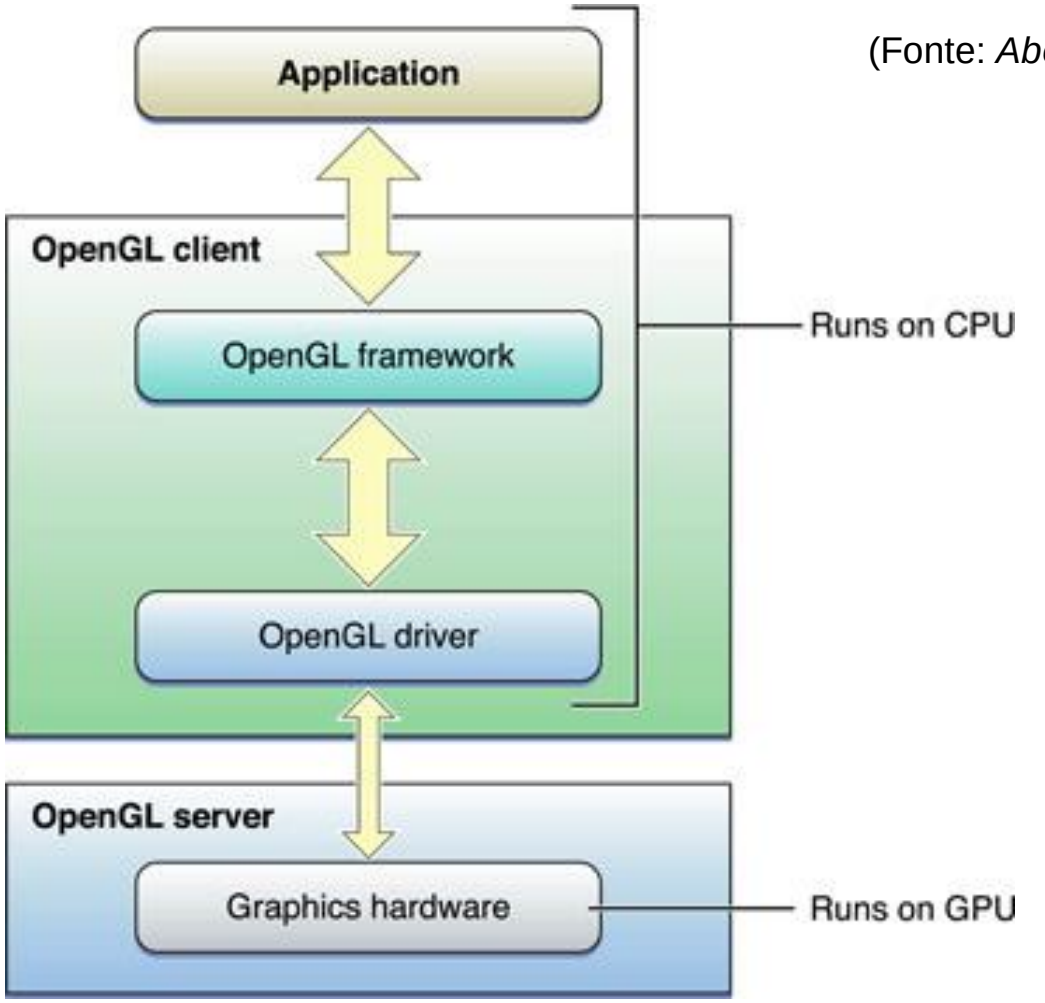


Pipeline de Renderização (Visão Geral)



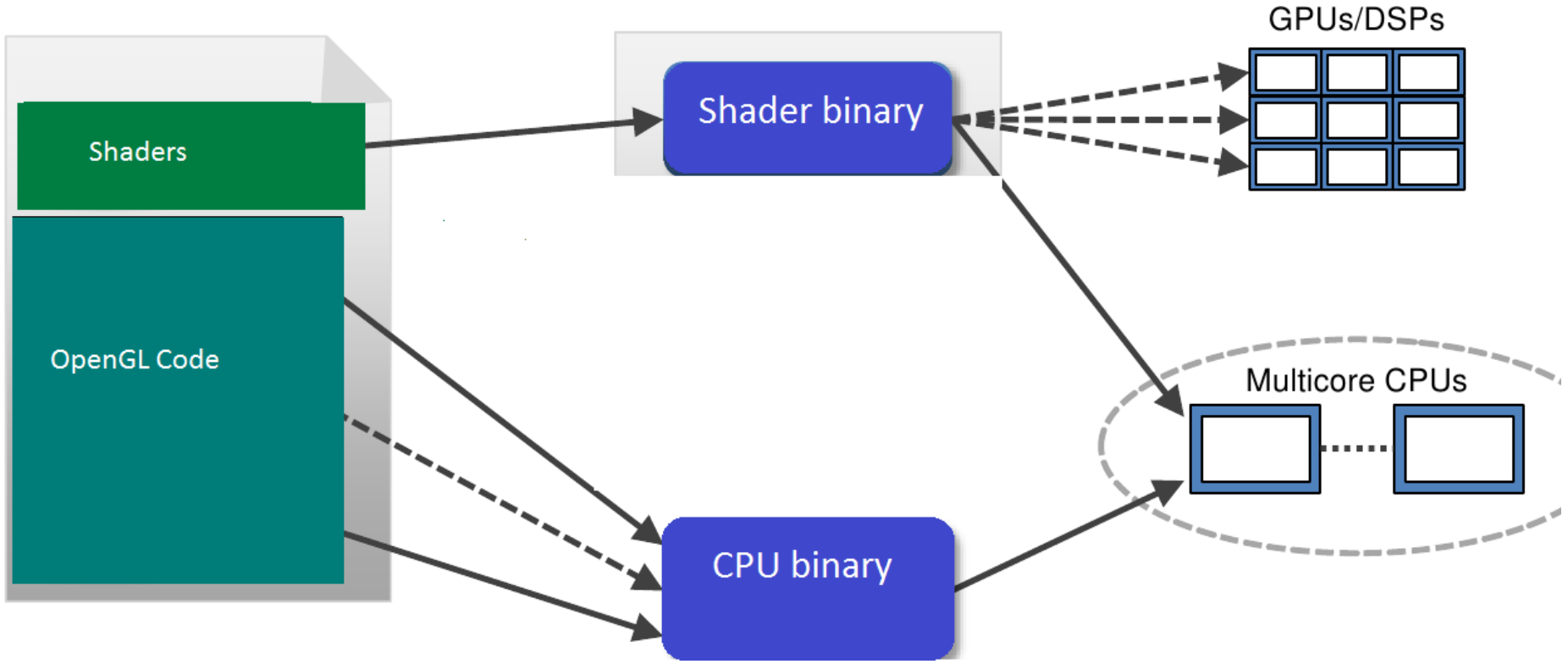
(Fonte: *Evas GL Programming Guide* – enlightenment.orghronos.org)

CPU x GPU



(Fonte: *About OpenGL for OS X* – apple.com)

CPU x GPU



(Fonte: *OpenGLES and RenderScript* – Arvind Devaraj)

Shaders

São pequenos programas escritos em *OpenGL Shading Language* (GLSL) e executados em GPUs (*Graphics Processing Units*).

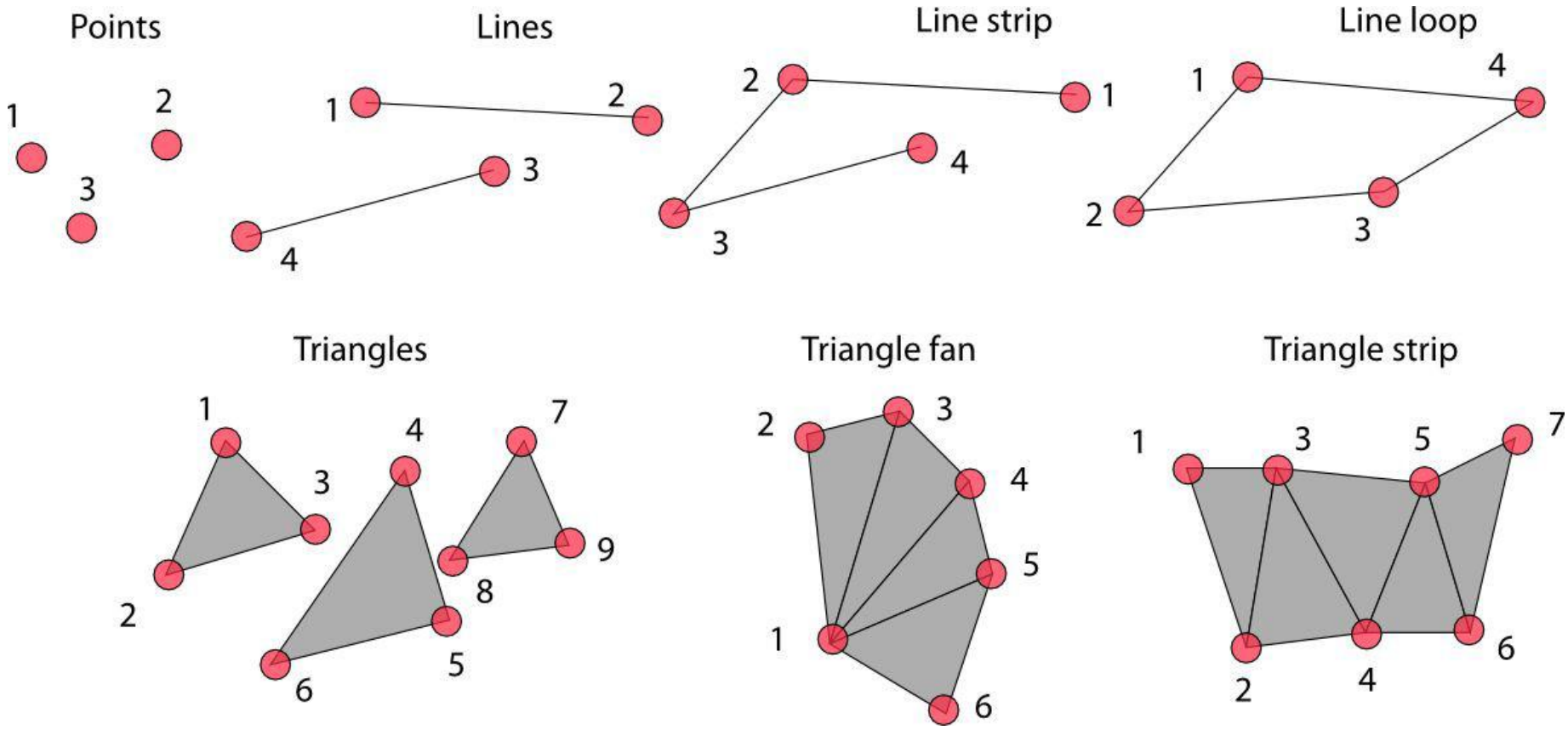
GLSL é uma linguagem parecida com “C”.

No OpenGL moderno, é necessário definir pelo menos o *vertex shader* e o *fragment shader*.

Os *shaders* definidos pelo usuário são compilados em tempo de execução.

A comunicação entre *shaders* ocorre por meio de variáveis globais especiais.

Primitivas do OpenGL



(Fonte: *Curve and Surfaces Modeling* – Christian Jacquemin)

Estrutura de um *Shader*

```
#version version_number

in type in_variable_name;

out type out_variable_name;

uniform type uniform_name;

int main()
{
    // Código
}
```

Tipos de Dados

Tipo	Descrição
bool	Booleano
int	Inteiro de 32 bits
uint	Inteiro de 32 bits, sem sinal
float	Ponto flutuante de precisão simples (IEEE-754)
double	Ponto flutuante de precisão dupla (IEEE-754)

GLSL também suporta vetores e matrizes.

Vetores

Vetores de 2, 3 ou 4 componentes (*n* = 2, 3, 4).

Tipo	**Descrição**
bvec*n*	Vetor de booleanos
ivec*n*	Vetor de inteiros
uvec*n*	Vetor de inteiros, sem sinais
vec*n*	Vetor de pontos flutuantes de precisão simples
dvec*n*	Vetor de pontos flutuantes de precisão dupla

Vetores (Utilização)

```
vec4 someVec;

vec4 someVec = vec4(8.0, 1.0, 0.0, 1.0);

someVec.xy = vec2(3.0, 2.0);

vec3 otherVec = someVec.zyw;

vec4 anotherVec = vec4(otherVec, 1.0);

// Utilizando cores
vecColor.rgba = vec4(0.5, 0.2, 1.0, 1.0);
```

Matrizes

Matrizes são do tipo ponto flutuante (precisão simples ou dupla) e ordenadas por coluna (*column-major order*).

Tipo	**Descrição**
mat*n*x*m*	Matriz com *n* colunas e *m* linhas.
mat*n*	Matriz com *n* colunas e *n* linhas.

n, *m* = 2, 3, 4.

Utilizar *dmat* (OpenGL 4.0 ou superior), no lugar de *mat*, para definição de matriz com dupla precisão.

Matrizes (Utilização)

```
mat3 someMat = mat3(1.0, 2.0, 3.0,
                    4.0, 5.0, 6.0,
                    7.0, 8.0, 9.0);

vec3 col0 = vec3(1.0, 2.0, 3.0);
vec3 col1 = vec3(4.0, 5.0, 6.0);
vec3 col2 = vec3(7.0, 8.0, 9.0);

mat3 someMat = mat3(col0, col1, col2);
```

Resulta em:

$$\begin{pmatrix} 1.0 & 4.0 & 7.0 \\ 2.0 & 5.0 & 8.0 \\ 3.0 & 6.0 & 9.0 \end{pmatrix}$$

Qualificadores de Armazenamento (*Storage Qualifiers*)

Definem o tipo de armazenamento de uma variável.

Tipo	**Descrição**
<none>	Variável local, sem visibilidade externa.
const	Constante (tempo de compilação).
in	Variável passada por uma etapa anterior.
out	Variável passada à próximo etapa.
uniform	Variável passado pelo código do cliente e não muda em diferentes múltiplas execuções de um shader.

Ferramentas

NumPy (Funções matemáticas)

PyOpenGL (Implementação OpenGL)

PyQt5 (Interface gráfica)

Pyrr (Funções matemáticas com suporte para OpenGL)

Exemplo: Triângulo

Passo 1: definir os vértices.

```
#                       Posições    Cores
vertices = np.array([-0.5, -0.5, 1.0, 0.0, 0.0, # Esquerdo
                      0.5, -0.5, 0.0, 1.0, 0.0, # Direito
                      0.0,  0.5, 0.0, 0.0, 1.0  # Superior
                    ], dtype=np.float32)
```

Exemplo: Triângulo

Passo 2: alocar memória na GPU.

```
# Cria um vertex buffer object
vbo = glGenBuffers(1)

# Conecta objeto vbo
glBindBuffer(GL_ARRAY_BUFFER, vbo)

# Copia dados na memória
glBufferData(GL_ARRAY_BUFFER,
             vertices.size * vertices.itemsize,
             vertices, GL_STATIC_DRAW)
```

Exemplo: Triângulo

Passo 3: criar o *vertex shader*.

```
VERTEX_SHADER = '''#version 330 core
layout (location = 0) in vec2 position;
layout (location = 1) in vec3 color;
out vec3 vertex_color;
void main()
{
    gl_Position = vec4(position, 0.0, 1.0);
    vertex_color = color;
}'''
```

Exemplo: Triângulo

Passo 4: criar o *fragment shader*.

```
FRAGMENT_SHADER = '''#version 330 core
in vec3 vertex_color;
out vec4 color;
void main()
{
    color = vec4(vertex_color, 1.0f);
}'''
```

Exemplo: Triângulo

Passo 5: compilar e ativar os *shaders*.

```
vertex_shader = shaders.compileShader(VERTEX_SHADER,
                                      GL_VERTEX_SHADER)
fragment_shader = shaders.compileShader(FRAGMENT_SHADER_SRC,
                                        GL_FRAGMENT_SHADER)
program = shaders.compileProgram(vertex_shader, fragment_shader)

glDeleteShader(vertex_shader)
glDeleteShader(fragment_shader)
glUseProgram(program)
```

Exemplo: Triângulo

Passo 6: vincular os vértices.

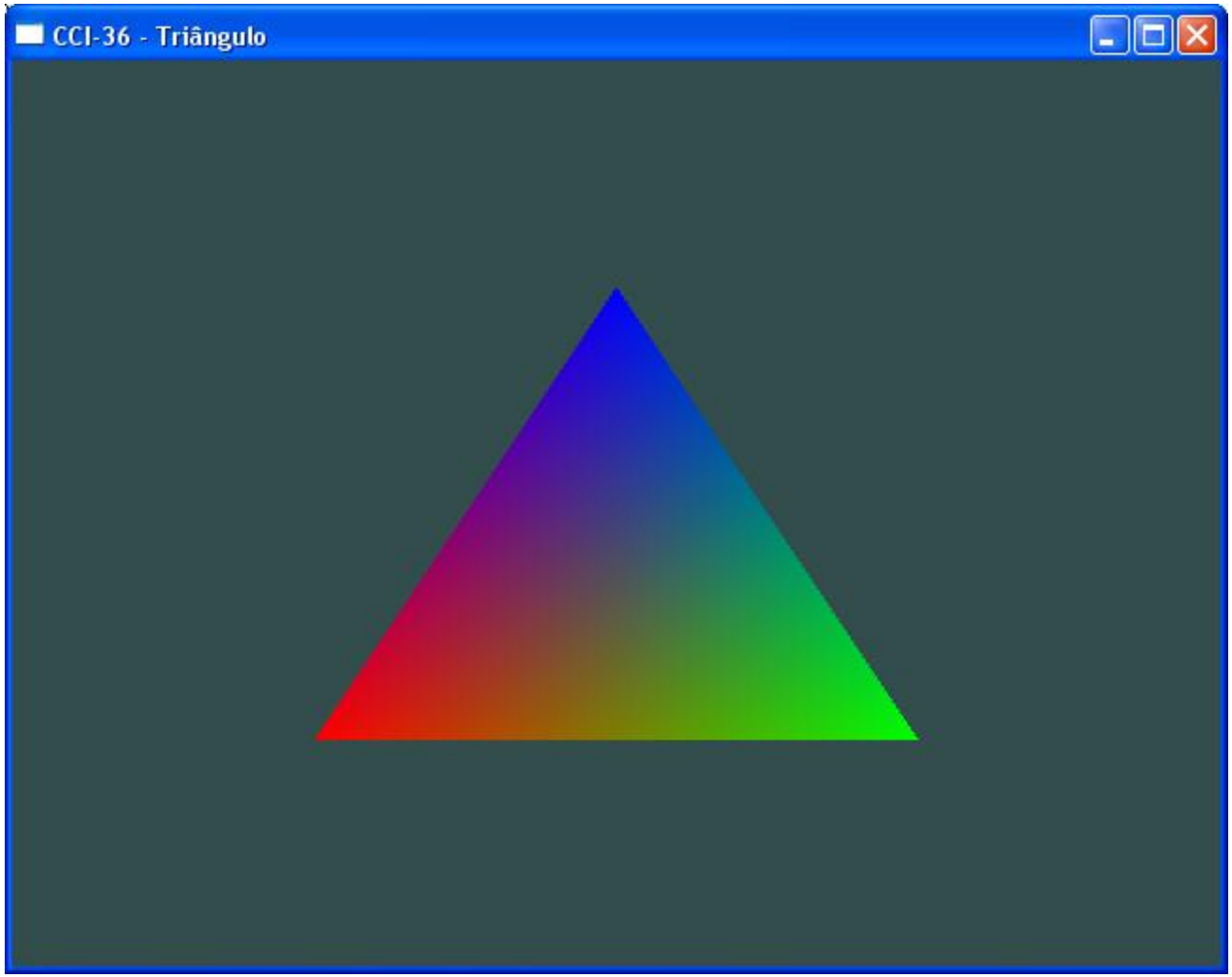
```
# Posição
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE,
                      5 * vertices.itemsize, None)
glEnableVertexAttribArray(0)

# Cor
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
                      5 * vertices.itemsize,
                      ctypes.c_void_p(2 * vertices.itemsize))
glEnableVertexAttribArray(1)
```

Exemplo: Triângulo

Passo 7: desenhar o triângulo.

```
glDrawArrays(GL_TRIANGLES,                    # Tipo de primitiva
             0,                               # Posição inicial
             3                                # Número de vértices
            )
```

Exemplo: Triângulo



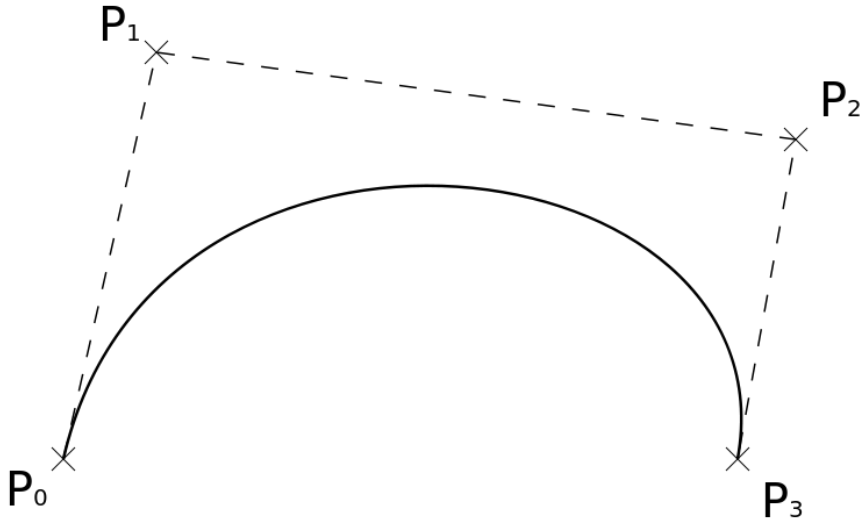
Curva de Bézier

A curva de Bézier é uma curva polinomial desenvolvida em 1962 pelo francês Pierre Bézier. É uma interpolação linear, por meio de alguns pontos representativos (pontos de controle), utilizada para modelar curvas suaves.

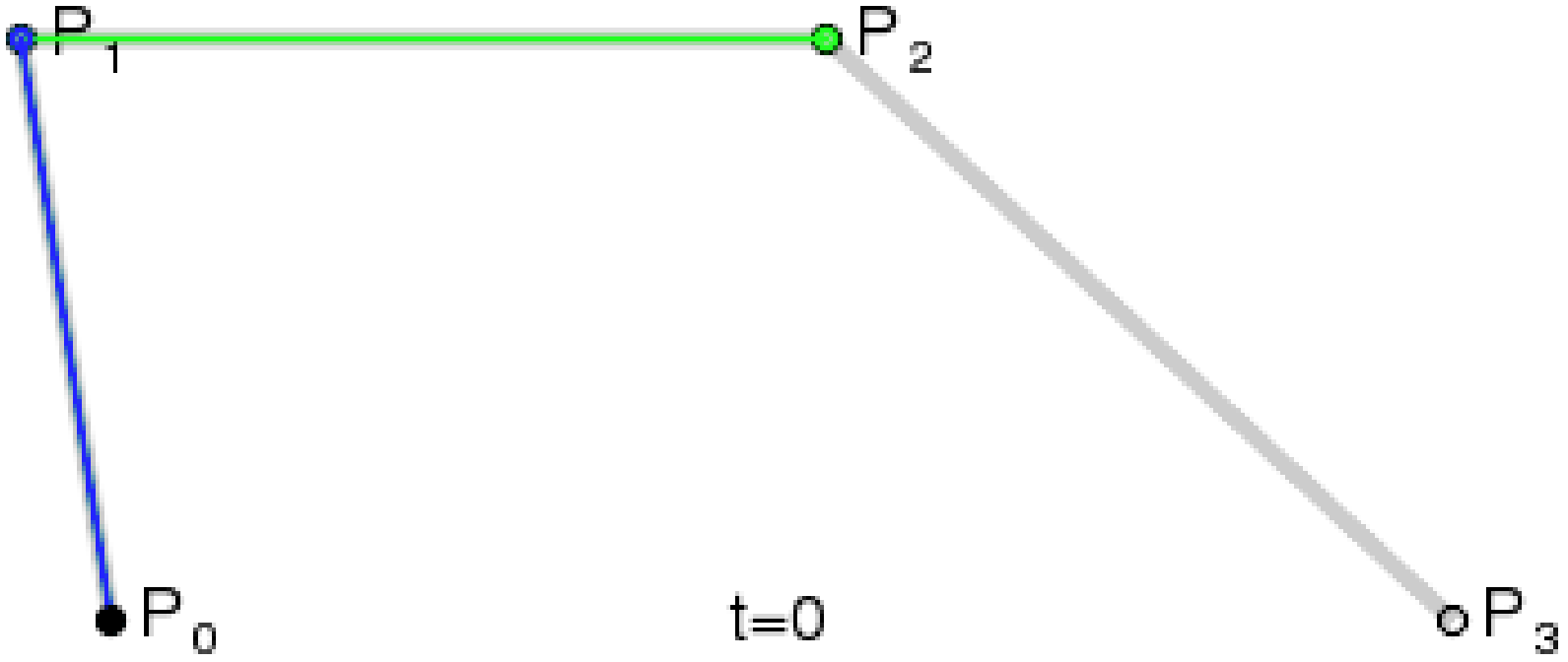
Utilizada em diversas aplicações gráficas como: GIMP, Photoshop, e CorelDRAW.

Exemplo de uma curva cúbica.

(Fonte: Wikipedia)

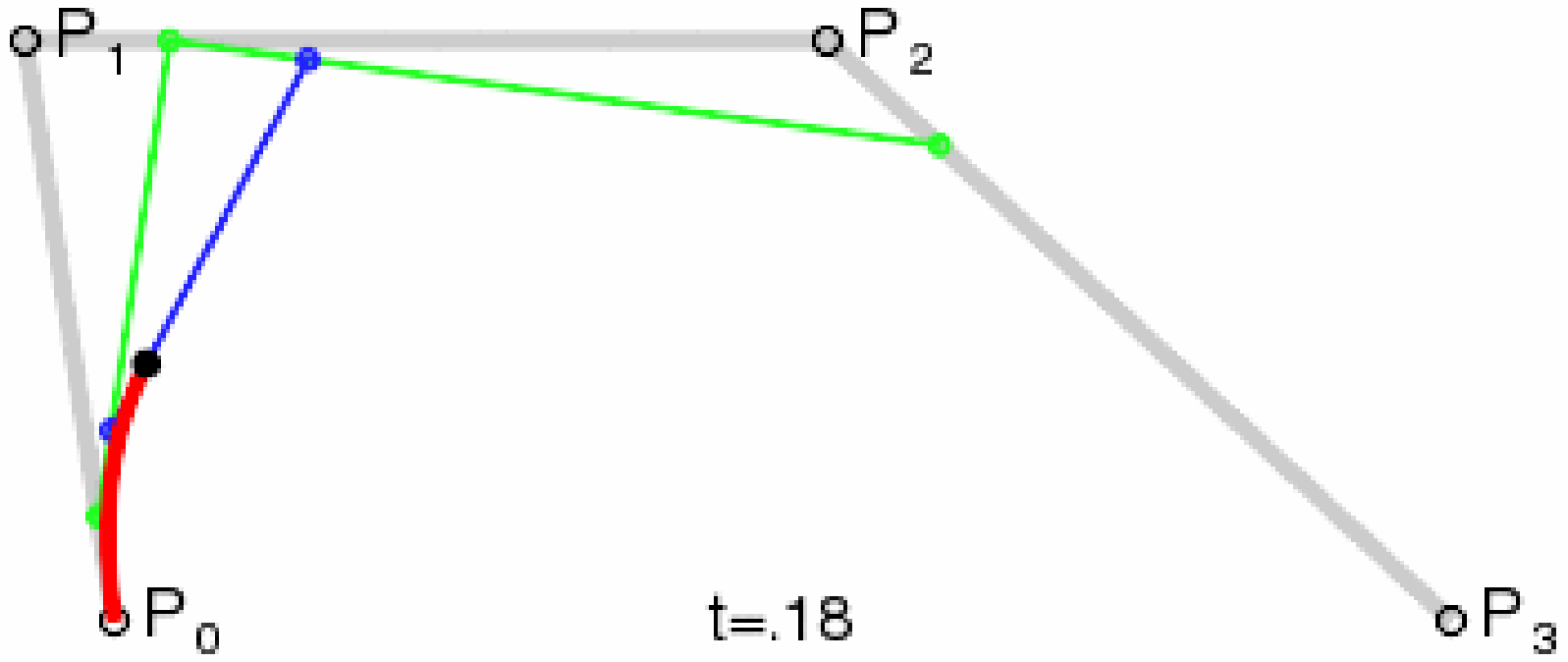


Construção da Curva de Bézier Cúbica (T = 0,00)



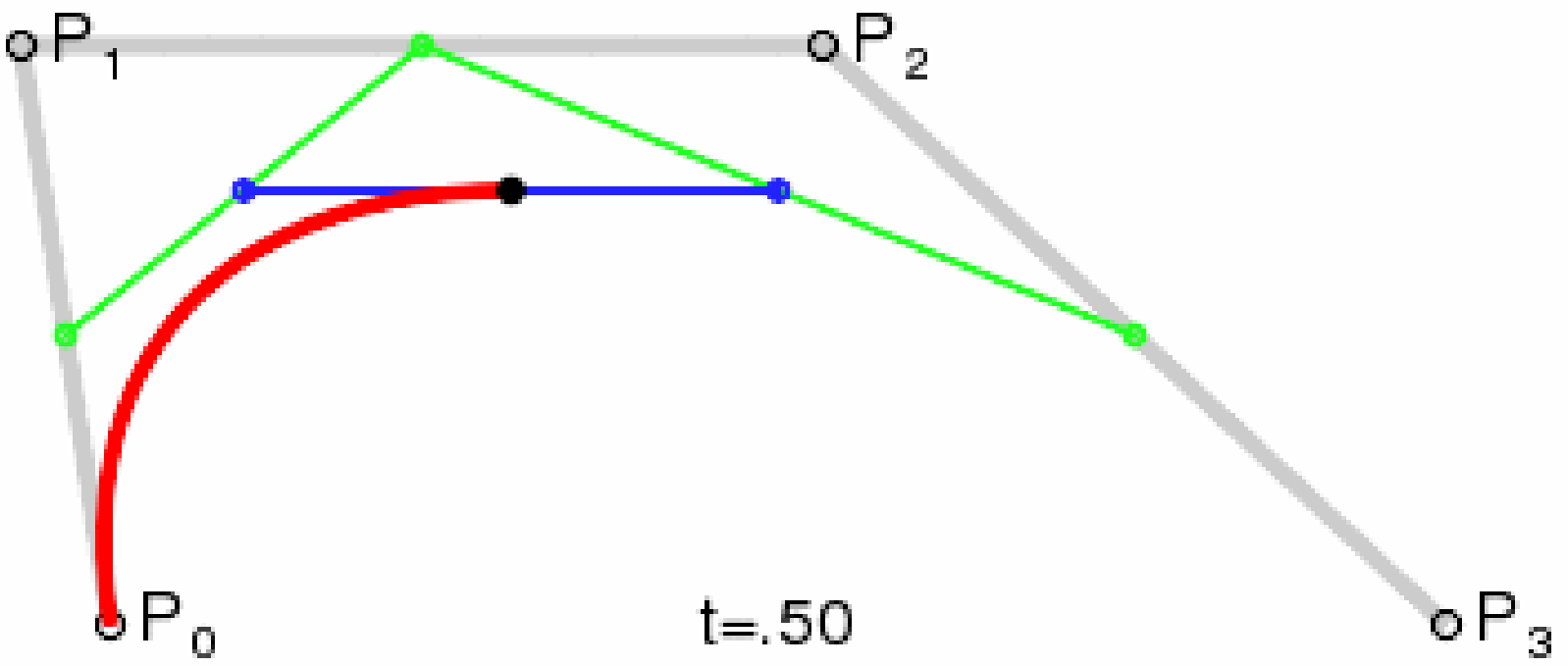
(Fonte: Wikipedia)

Construção da Curva de Bézier Cúbica (T = 0,18)



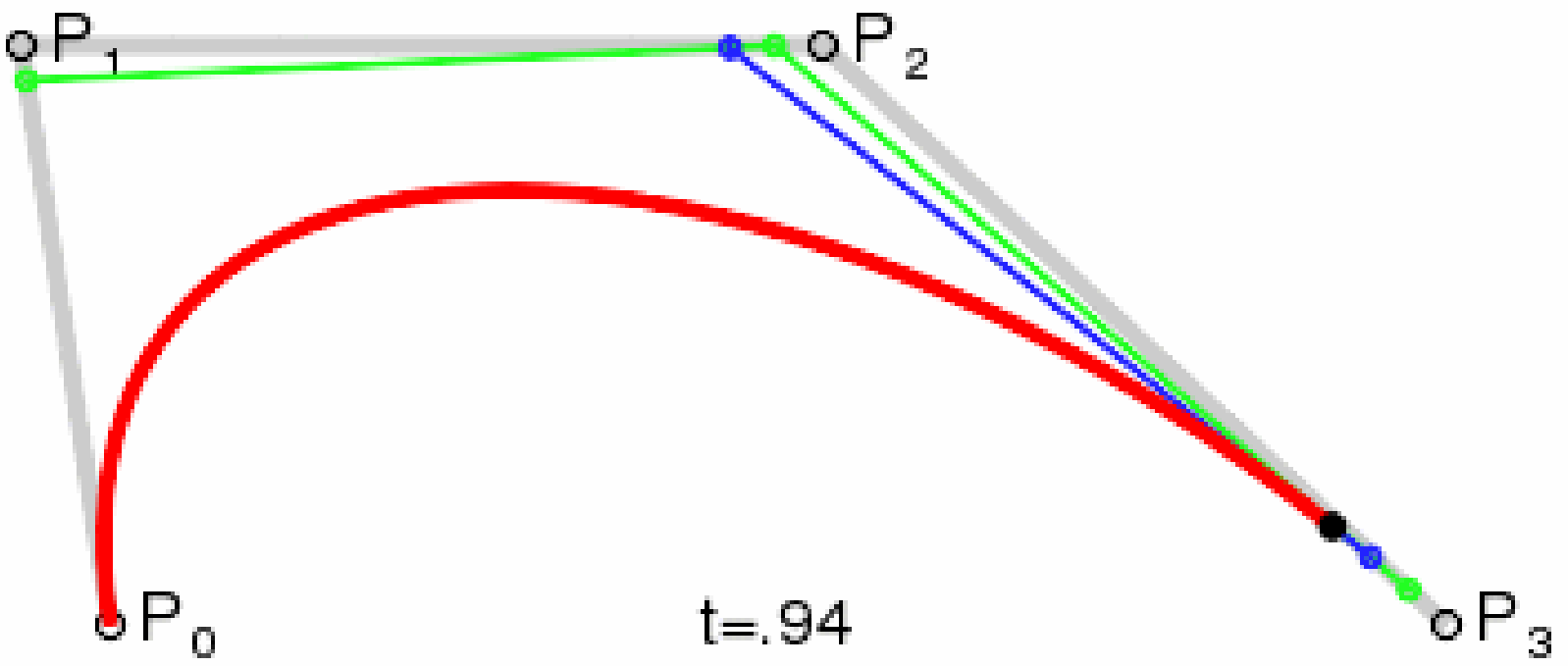
(Fonte: Wikipedia)

Construção da Curva de Bézier Cúbica (T = 0,50)



(Fonte: Wikipedia)

Construção da Curva de Bézier Cúbica (T = 0,94)



(Fonte: Wikipedia)

Equações

Equação geral:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i}\, t^i P_i \quad 0 \le t \le 1$$

Curva de Bézier Quadrática:

$$B(t) = (1-t)^2 P_0 + 2t\,(1-t)\,P_1 + t^2 P_2 \quad 0 \le t \le 1$$

Curva de Bézier Cúbica:

$$B(t) = (1-t)^3 P_0 + 3t^2\,(1-t) P_1 + 3t(1-t)^2 P_2 + t^3 P_3 \quad 0 \le t \le 1$$

Exemplo OpenGL da Curva de Bézier Cúbica