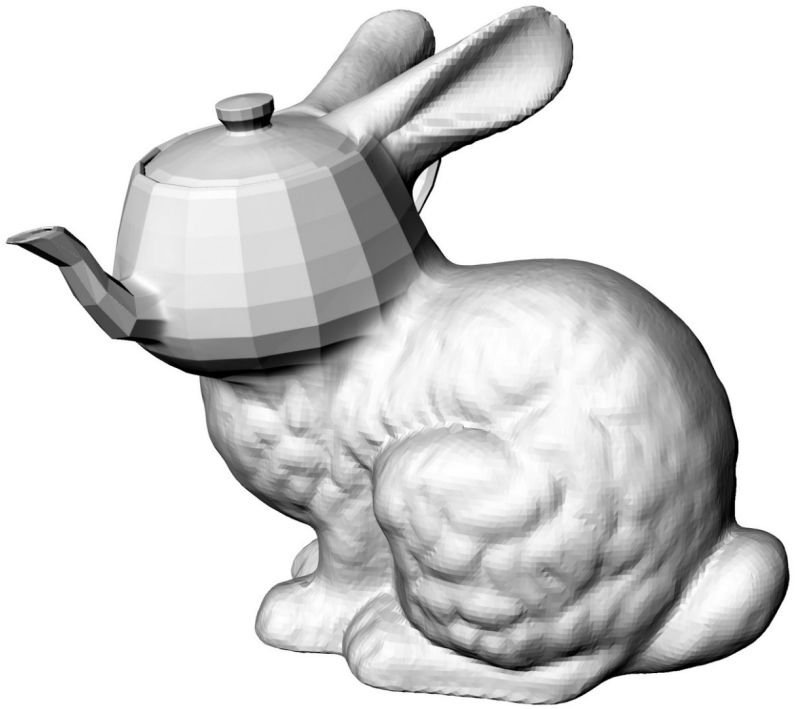


OpenGL *buffers*: VAO, VBO e EBO

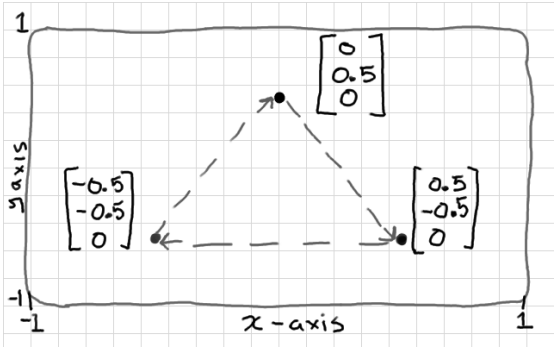
por Rossana B Queiroz

Buffers no OpenGL Moderna

- Definição de *buffer*: região de memória que serve para armazenar dados temporariamente
  - Dados provindos de algum dispositivo de entrada, antes de serem processados
  - Dados a serem enviados para algum dispositivo de saída (por exemplo, *frame buffer*) ou para serem processados pela GPU

Vertex Buffer Objects (VBOs)

- Array de dados (normalmente floats)
- Buffer para enviar dados dos vértices à GPU
  - posição, vetores normais, cores etc



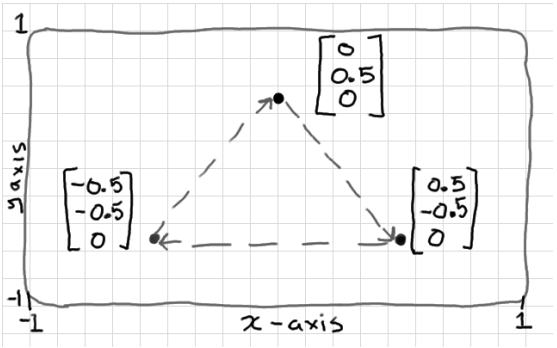
Esta forma poderia ter o vertex buffer:
{0, 0.5, 0, 0.5, -0.5, 0, -0.5, -0.5, 0}

- Os dados são alocados diretamente na memória da GPU
- Isso permite que os objetos sejam renderizados pela placa gráfica com maior velocidade

Como programar

1. Definir o array:

```
GLfloat vertices[] = { -0.5f, -0.5f, 0.0f,
                        0.5f, -0.5f, 0.0f,
                        0.0f, 0.5f, 0.0f };
```



2. Cada buffer na OpenGL precisa ter um identificador único, para isso precisamos gerar este ID usando a função glGenBuffers:

```
GLuint vbo;
glGenBuffers(1, &vbo);
```

3. OpenGL possui vários tipos de buffers. Os VBOs são do tipo *GL_ARRAY_BUFFER*. Devemos agora fazer o *bind* do novo buffer criado:

```
glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

Como programar (continuação)

4. Por ultimo, chamamos a função *glBufferData* que copia os dados do array definido para a memória do buffer da OpenGL:

```
glBufferData(GL_ARRAY_BUFFER, 9 * sizeof(GLfloat), vertices, GL_STATIC_DRAW);
```

ou `sizeof(vertices)`

O quarto parâmetro especifica como nós queremos que a placa gráfica gerencie os dados fornecidos. Há 3 formas:

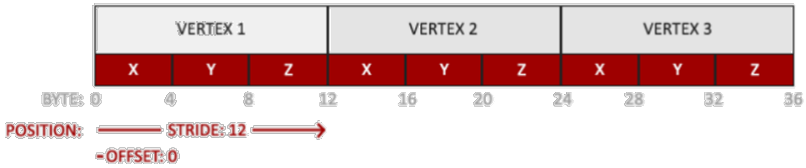
- GL_STATIC_DRAW – dados não vão mudar, ou muito raramente
- GL_DYNAMIC_DRAW – dados vão mudar com frequência
- GL_STREAM_DRAW – dados vão mudar cada vez que forem desenhados

Vertex Array Objects (VAOs)

- Fazem a ligação dos atributos de um vértice
  - Posição, cores, normais
- Define:
  - Qual o VBO que será usado
  - Localização dos dados desse VBO
  - Qual o formato desses dados

```
glVertexAttribPointer(0, 3,
GL_FLOAT, GL_FALSE, 3 *
sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
```

Em nosso exemplo anterior, os dados do buffer estão formatados da seguinte forma:



- Os valores de posição estão armazenados como floats (4 bytes)
- Cada posição é composta por 3 desses valores (x, y e z) – logo temos um total de 12 bytes por vértice
- Não possuem outros valores armazenados entre cada conjunto de 3 valores que definem o vértice – logo nosso offset é zero.
- O primeiro valor está armazenado diretamente no início do buffer (byte 0).

Vertex Array Objects (VAOs)

- Segue a explicação de cada parâmetro:

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat),
(GLvoid*)0);
```

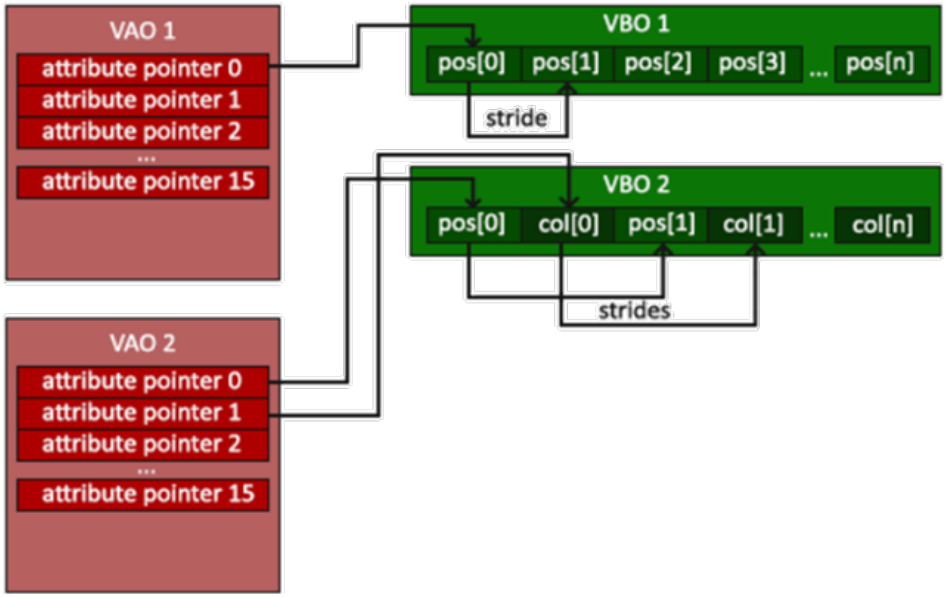
  - O primeiro parâmetro, 0, refere-se a qual o atributo estamos linkando (posição, cor, normal, coord textura...). Este número será o mesmo no vertex shader, identificado com a palavra-chave location

```
layout (location = 0) in vec3 position;
```

  - O próximo parâmetro, 3, especifica o tamanho do atributo (3 valores – xyz)
  - O próximo parâmetro, GL_FLOAT, especifica o tamanho de cada dado
  - O próximo parâmetro, GL_FALSE, especifica se os dados precisam ser normalizados (valores no interval de -1 a 1). Nossos dados já estão.
  - Por fim, passamos o deslocamento inicial no buffer, que no nosso caso é nenhum (podemos colocar 0 ou NULL se desejarmos)

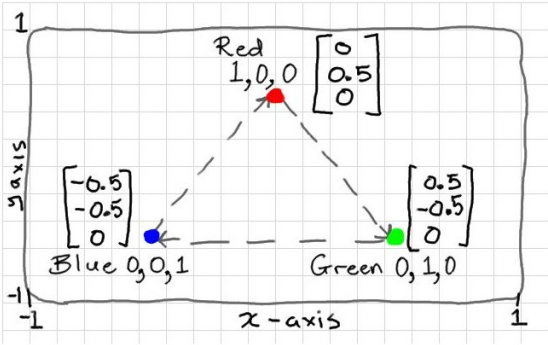
Vertex Array Objects (VAOs)

- Ideia geral:



- Podemos ter mais de um VAO, e cada VAO pode guarder mais de um atributo dos vertices (VBOs)
  - Troca de contexto entre VAOs possui um certo custo, portanto, cuidar!

Exemplo com 2 VBOs – 2 arrays e 2 atributos



1

```
GLfloat vertices[] = {
0.0f,0.5f,0.0f,
0.5f, -0.5f,0.0f,
-0.5f, -0.5f,0.0f
};

GLfloat colors[] = {
1.0f, 0.0f,  0.0f,
0.0f, 1.0f,  0.0f,
0.0f, 0.0f,  1.0f
};
```

2

```
GLuint vVBO;
glGenBuffers(1, &vVBO);
glBindBuffer(GL_ARRAY_BUFFER, vVBO);
glBufferData(GL_ARRAY_BUFFER, 9 *
sizeof(GLfloat), vertices, GL_STATIC_DRAW);
```

3

```
GLuint cVBO;
glGenBuffers(1, &cVBO);
glBindBuffer(GL_ARRAY_BUFFER, cVBO);
glBufferData(GL_ARRAY_BUFFER, 9 *
sizeof(GLfloat), colors, GL_STATIC_DRAW);
```

4

```
GLuint VAO;
glGenVertexArrays(1,
&VAO);
glBindVertexArray(VAO);
```

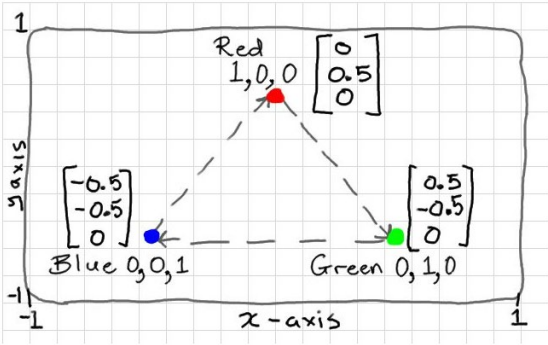
5

```
//incluindo o buffer de vértices
glBindBuffer(GL_ARRAY_BUFFER, vVBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3
* sizeof(GLfloat), NULL);
glEnableVertexAttribArray(0);
```

6

```
//incluindo o buffer de cores
glBindBuffer(GL_ARRAY_BUFFER, cVBO);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3
* sizeof(GLfloat), NULL);
glEnableVertexAttribArray(1);
```

Exemplo com 1 VBO – 1 array e 2 atributos



1

```
// Set up vertex data (and buffer(s)) and attribute
pointers
GLfloat vertices[] = {
// Positions          // Colors
0.5f, -0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   // Bottom Right
-0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   // Bottom Left
0.0f,  0.5f, 0.0f,   0.0f, 0.0f, 1.0f    // Top
};
```

2

```
GLuint VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);

glBindVertexArray(VAO);
```

3

```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
vertices, GL_STATIC_DRAW);
```

4

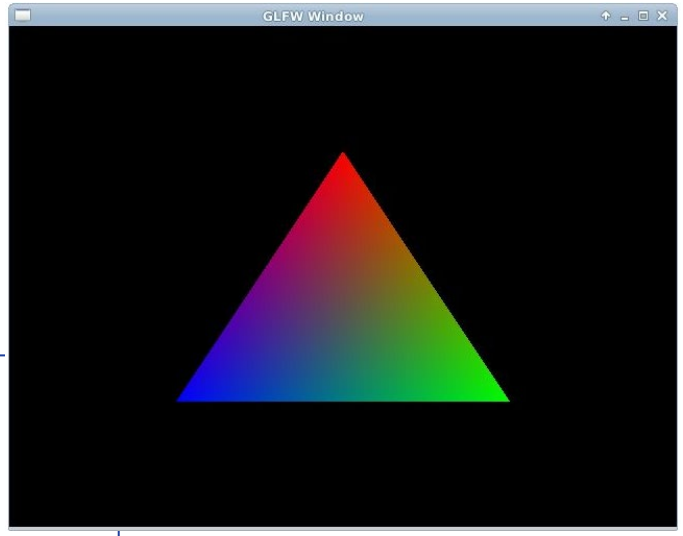
```
// Position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6
* sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
```

5

```
// Color attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 *
sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
glEnableVertexAttribArray(1);
```

Para mandar desenhar

- No loop do código:




```
glUseProgram(shaderProgram);
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0,  3);
```

Offset inicial Nro de vértices

Element Buffer Objects (EBOs)

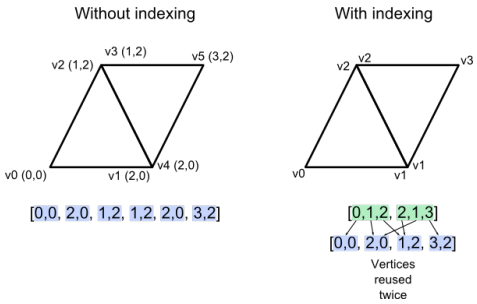
Para associar a ideia de **índices** e evitar a especificação de vértices replicados no VBO

1

```
GLfloat vertices[] = {
0.5f,  0.5f, 0.0f,  // Top Right
0.5f, -0.5f, 0.0f,  // Bottom Right
-0.5f, -0.5f, 0.0f,  // Bottom Left
-0.5f,  0.5f, 0.0f   // Top Left
};
GLuint indices[] = { // Note that we
start from 0
0, 1, 3,  // First Triangle
1, 2, 3   // Second Triangle
};
```

2

```
GLuint VBO, VAO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);
// Bind the Vertex Array
Object first, then bind and
set vertex buffer(s) and
attribute pointer(s).
glBindVertexArray(VAO);
```

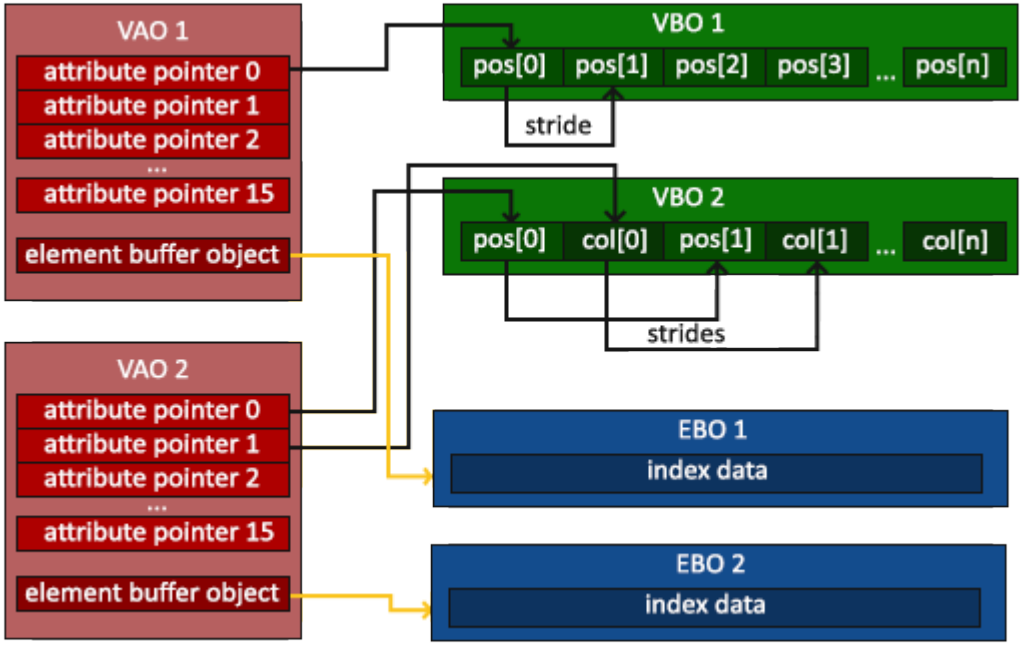


3

```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat),
(GLvoid*)0);
glEnableVertexAttribArray(0);
```

VAOs, VBOs e EBOs

- Esquema geral



Para mandar desenhar (com EBO)

- No loop do código:

```
glUseProgram(shaderProgram);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

Número total de vertices (índices)

Offset inicial

Referências

- https://learnopengl.com/
- Anton's OpenGL 4 Tutorials