

VAO, VBO, EBO

Компьютерная графика

Vertex buffer objects (VBO) — объекты вершинного буфера

Объявление идентификатора

Инициализация буфера

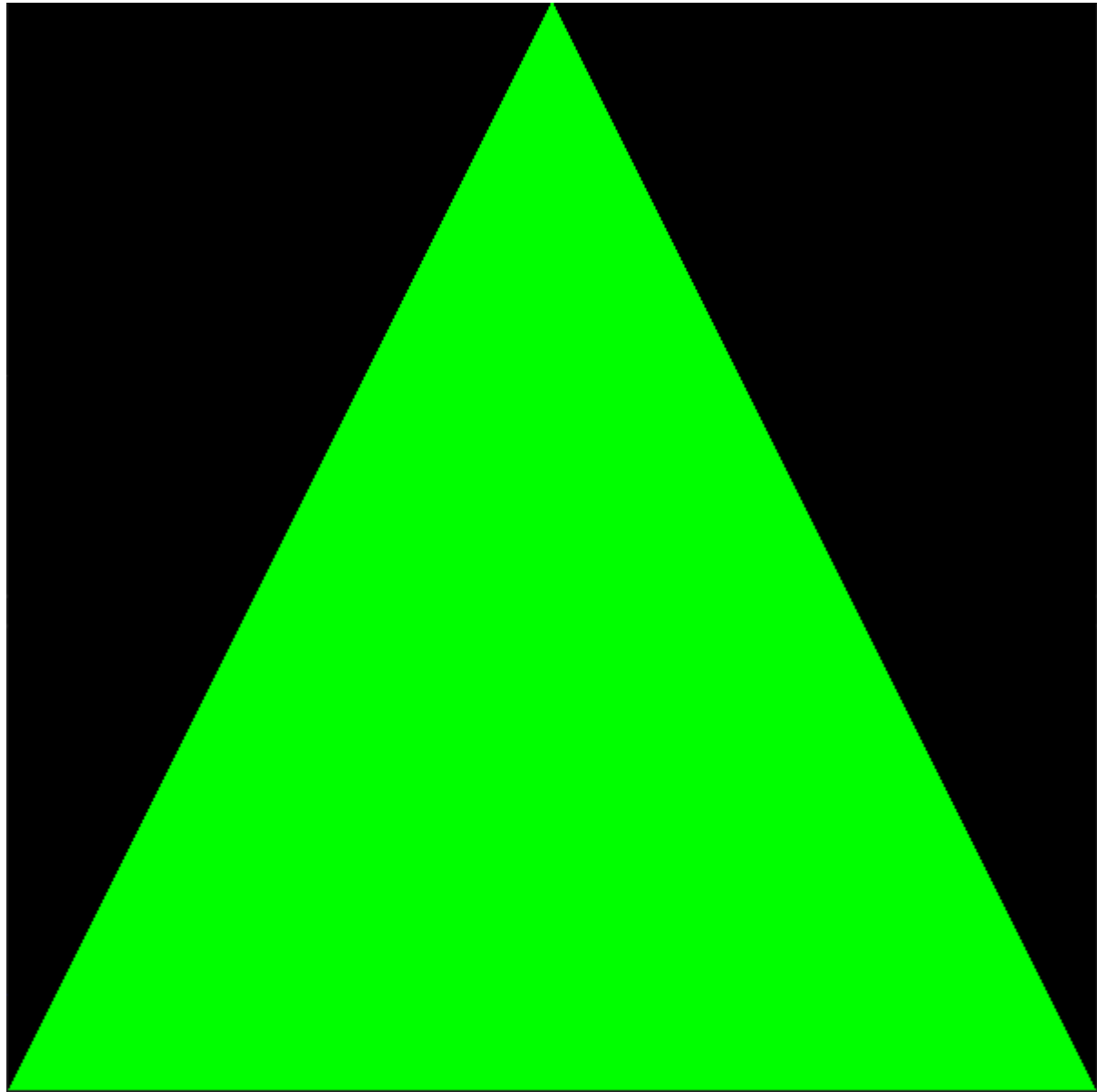
Основная часть (управление выводом)

Освобождение буфера

Объявление идентификатора

```
// ID Vertex Buffer Object
```

```
GLuint VBO;
```

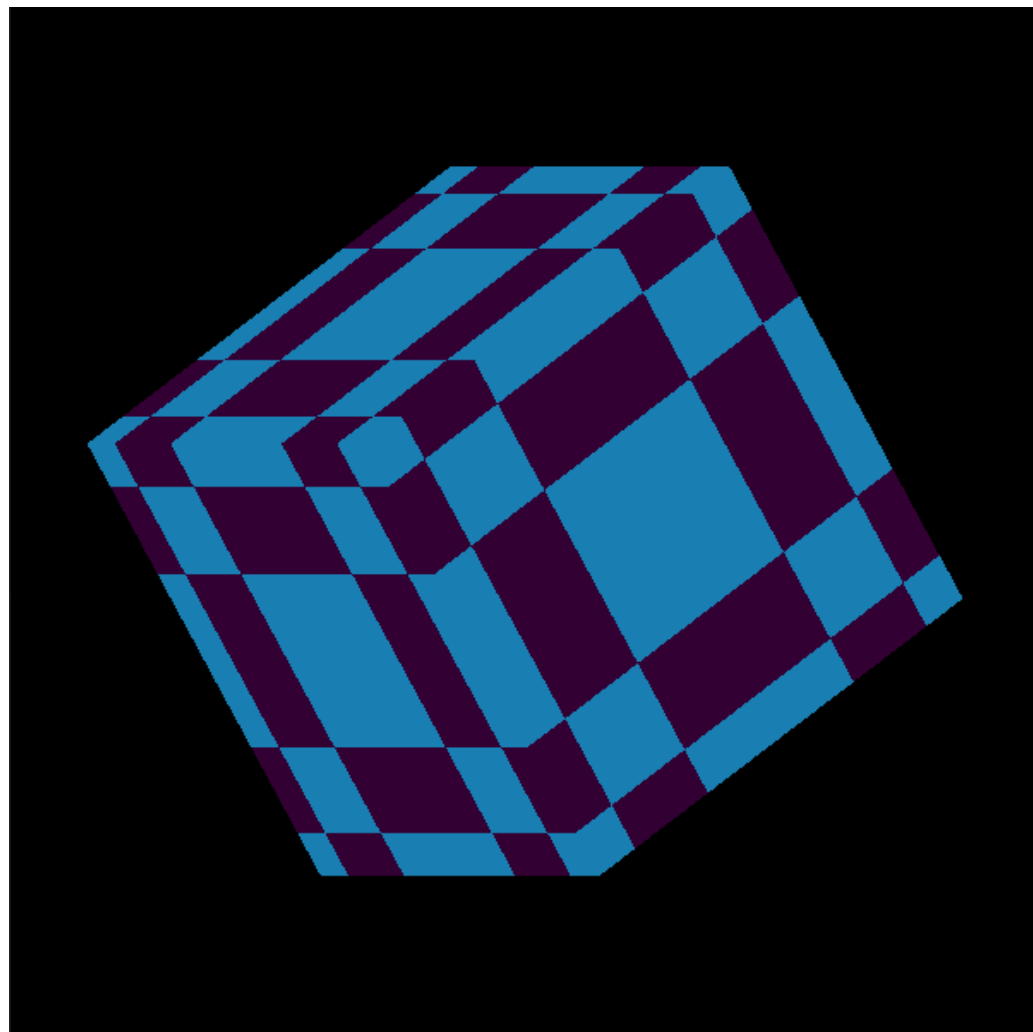


Инициализация буфера

```
void InitVBO() {  
  
    glGenBuffers(1, &VBO); //создание буфера  
  
    Vertex triangle[3] = { { -1.0f, -1.0f }, { 0.0f, 1.0f }, { 1.0f, -1.0f } }; // Вершины треугольника  
  
    glBindBuffer(GL_ARRAY_BUFFER, VBO); // Привязка или активация буфера  
  
    glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW); // Передаём вершины в буфер  
  
    checkOpenGLerror();  
}
```

3 режима

- `GL_STATIC_DRAW`: данные либо никогда не будут изменяться, либо будут изменяться очень редко
- `GL_DYNAMIC_DRAW`: данные будут меняться довольно часто
- `GL_STREAM_DRAW`: данные будут меняться при каждой отрисовке



```

void InitVBO() {
    glGenBuffers(1, &VBO);
    Vertex triangle[] = { // Вершины кубика
        { -0.5, -0.5, +0.5 }, { -0.5, +0.5, +0.5 }, { +0.5, +0.5, +0.5 },
        { +0.5, +0.5, +0.5 }, { +0.5, -0.5, +0.5 }, { -0.5, -0.5, +0.5 },
        { -0.5, -0.5, -0.5 }, { +0.5, +0.5, -0.5 }, { -0.5, +0.5, -0.5 },
        { +0.5, +0.5, -0.5 }, { -0.5, -0.5, -0.5 }, { +0.5, -0.5, -0.5 },

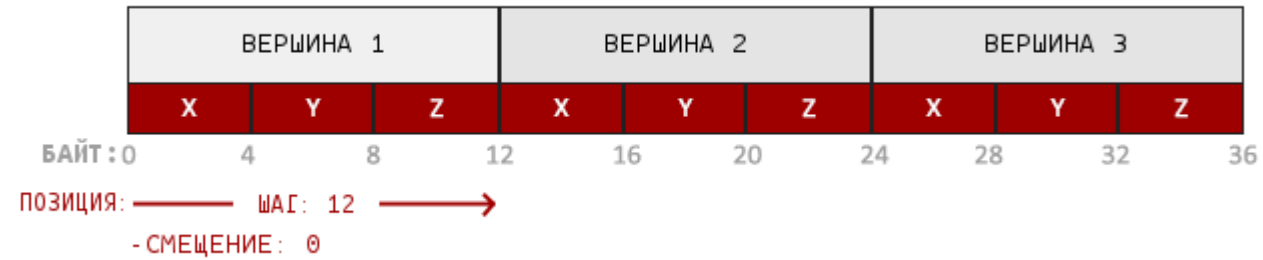
        { -0.5, +0.5, -0.5 }, { -0.5, +0.5, +0.5 }, { +0.5, +0.5, +0.5 },
        { +0.5, +0.5, +0.5 }, { +0.5, +0.5, -0.5 }, { -0.5, +0.5, -0.5 },
        { -0.5, -0.5, -0.5 }, { +0.5, -0.5, +0.5 }, { -0.5, -0.5, +0.5 },
        { +0.5, -0.5, +0.5 }, { -0.5, -0.5, -0.5 }, { +0.5, -0.5, -0.5 },

        { +0.5, -0.5, -0.5 }, { +0.5, -0.5, +0.5 }, { +0.5, +0.5, +0.5 },
        { +0.5, +0.5, +0.5 }, { +0.5, +0.5, -0.5 }, { +0.5, -0.5, -0.5 },
        { -0.5, -0.5, -0.5 }, { -0.5, +0.5, +0.5 }, { -0.5, -0.5, +0.5 },
        { -0.5, +0.5, +0.5 }, { -0.5, -0.5, -0.5 }, { -0.5, +0.5, -0.5 },
    };
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // Передаем вершины в буфер
    glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);
    checkOpenGLError();
}

```


Формат вершинного буфера

Плотно упакованный буфер



```
void Init() {  
    InitShader();  
    InitVBO();  
    // Включаем проверку глубины  
    glEnable(GL_DEPTH_TEST);  
}  
  
//А при отрисовке не забываем  
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Соотносим данные в программе и в шейдере

```
// Вытягиваем ID атрибута вершин из собранной программы  
Attrib_vertex = glGetAttribLocation(Program, "coord");
```

Найди отличия в следующих 2-х слайдах

```
void Draw() { //треугольник

    glUseProgram(Program); // Устанавливаем шейдерную программу текущей

    glEnableVertexAttribArray(Attrib_vertex); // Подключаем массив атрибутов

    glBindBuffer(GL_ARRAY_BUFFER, VBO); // Подключаем VBO

    // Указываем, как и куда читаем данные из VBO

    glVertexAttribPointer(Attrib_vertex, 2, GL_FLOAT, GL_FALSE, 0, 0);

    glBindBuffer(GL_ARRAY_BUFFER, 0); // Отключаем VBO

    glDrawArrays(GL_TRIANGLES, 0, 3); // Рисуем

    glDisableVertexAttribArray(Attrib_vertex); // Отключаем массив атрибутов

    glUseProgram(0); // Отключаем шейдерную программу

    checkOpenGLError();

}
```

```
void Draw() { //куб

    glUseProgram(Program); // Устанавливаем шейдерную программу текущей

    glEnableVertexAttribArray(Attrib_vertex); // Включаем массив атрибутов

    glBindBuffer(GL_ARRAY_BUFFER, VBO); // Подключаем VBO

    // Указываем, как и куда читаем данные из VBO

    glVertexAttribPointer(Attrib_vertex, 3, GL_FLOAT, GL_FALSE, 0, 0);

    glBindBuffer(GL_ARRAY_BUFFER, 0); // Отключаем VBO

    glDrawArrays(GL_TRIANGLES, 0, 36); // Рисуем

    glDisableVertexAttribArray(Attrib_vertex); // Отключаем массив атрибутов

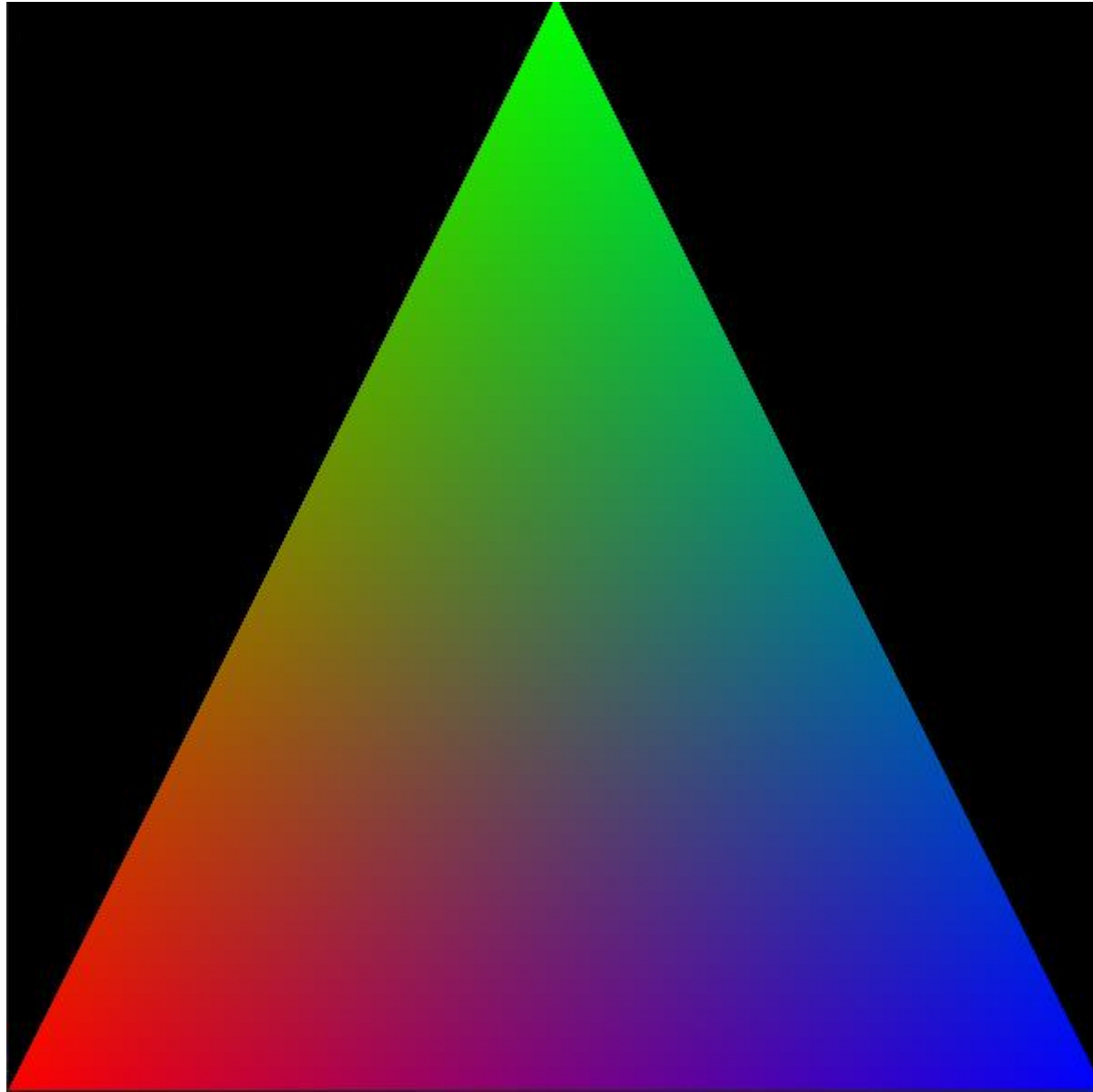
    glUseProgram(0); // Отключаем шейдерную программу

    checkOpenGLError();

}
```

Освобождение буфера

```
void ReleaseVBO() {  
    glBindBuffer(GL_ARRAY_BUFFER, 0);  
    glDeleteBuffers(1, &VBO);  
}
```



Переменные с идентификаторами ID

```
// ID атрибута вершин
```

```
GLuint Attrib_vertex;
```

```
// ID атрибута цвета
```

```
GLuint Attrib_color;
```

```
// ID VBO вершин
```

```
GLuint VBO_position;
```

```
// ID VBO цвета
```

```
GLuint VBO_color;
```

Два значения для каждой вершины

```
void InitVBO() {
    glGenBuffers(1, &VBO_position);
    glGenBuffers(1, &VBO_color);
    // Вершины треугольника
    Vertex triangle[3] = { { -1.0f, -1.0f , 0.0f }, { 0.0f, 1.0f , 0.0f }, { 1.0f, -1.0f , 0.0f } };
    // Цвет треугольника
    float colors[3][4] = { { 1.0, 0.0, 0.0}, { 0.0, 1.0, 0.0}, { 0.0, 0.0, 1.0}, };

    // Передаем вершины в буфер
    glBindBuffer(GL_ARRAY_BUFFER, VBO_position);
    glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, VBO_color);
    glBufferData(GL_ARRAY_BUFFER, sizeof(colors), colors, GL_STATIC_DRAW);
    checkOpenGLError();
}
```

```
void Draw() {
    glUseProgram(Program); // Устанавливаем шейдерную программу текущей
// Включаем массивы атрибутов
glEnableVertexAttribArray(Attrib_vertex);
glEnableVertexAttribArray(Attrib_color);

// Подключаем VBO_position
glBindBuffer(GL_ARRAY_BUFFER, VBO_position);
glVertexAttribPointer(Attrib_vertex, 3, GL_FLOAT, GL_FALSE, 0, 0);

// Подключаем VBO_color
glBindBuffer(GL_ARRAY_BUFFER, VBO_color);
glVertexAttribPointer(Attrib_color, 3, GL_FLOAT, GL_FALSE, 0, 0);

glBindBuffer(GL_ARRAY_BUFFER, 0); // Отключаем VBO

glDrawArrays(GL_TRIANGLES, 0, 3); // Передаем данные на видеокарту(рисуем)

// Отключаем массивы атрибутов
glDisableVertexAttribArray(Attrib_vertex);
glDisableVertexAttribArray(Attrib_color);

glUseProgram(0); checkOpenGLError();
}
```

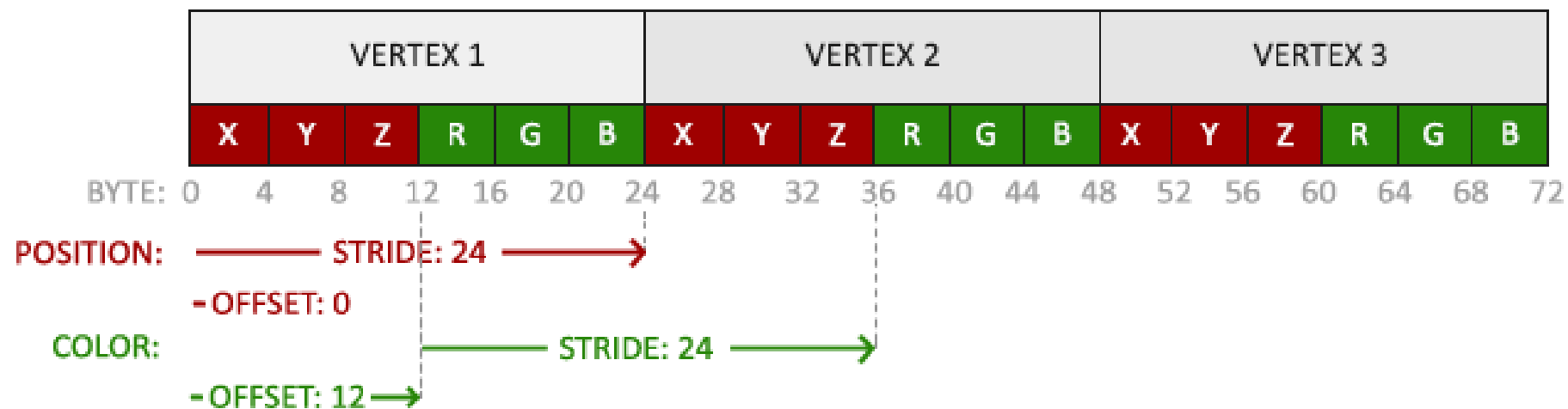
Два значения для каждой вершины в одном буфере

```
void InitVBO() {
    glGenBuffers(1, &VBO);
    // Вершины и цвет треугольника
    float attrib[3][6] = { {-1.0f, -1.0f, 0.0, 1.0, 0.0, 0.0}, {0.0f, 1.0f, 0.0, 0.0, 1.0, 0.0}, {1.0f, -1.0f, 0.0, 0.0, 0.0, 1.0}, };

    // Передаем вершины в буфер
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(attrib), attrib, GL_STATIC_DRAW);

    checkOpenGLError();
}
```

VBO в памяти: координаты и цвет



VBO: координаты и цвет

```
glEnableVertexAttribArray(0);
```

```
glEnableVertexAttribArray(1);
```

```
// Подключаем VBO
```

```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

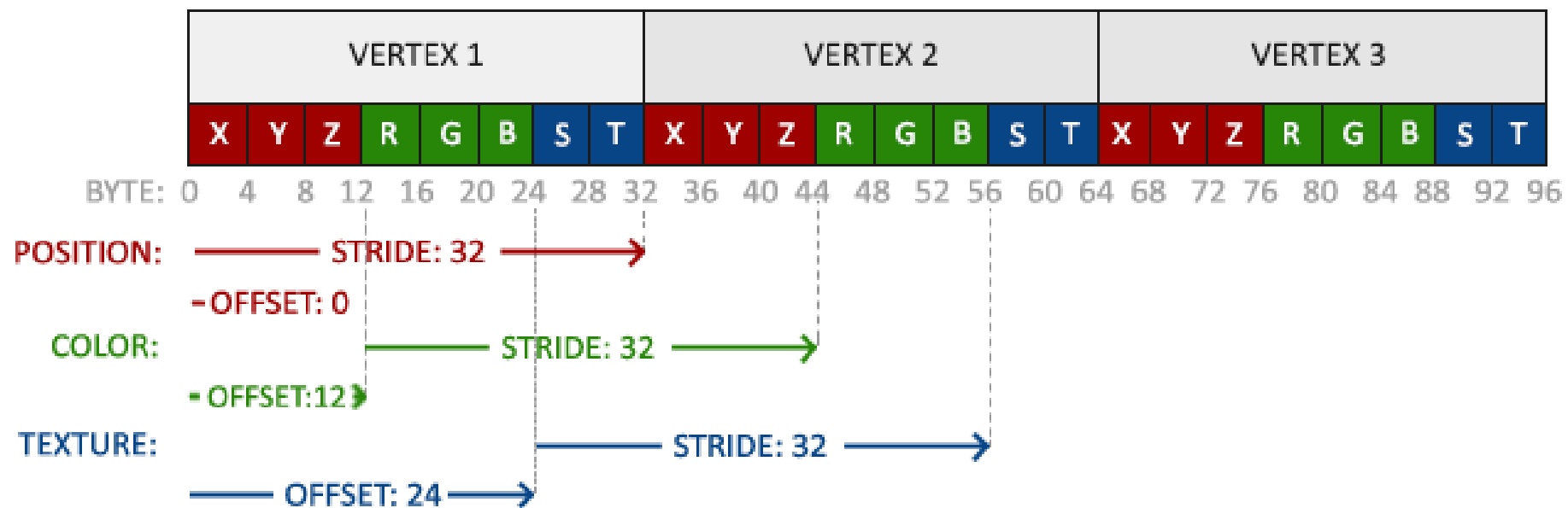
```
// Атрибут с координатами
```

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
```

```
// Атрибут с цветом
```

```
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
```

VBO в памяти: координаты, цвет и текстура



VBO: координаты, цвет и текстура

```
glEnableVertexAttribArray(0);  
glEnableVertexAttribArray(1);  
glEnableVertexAttribArray(2);
```

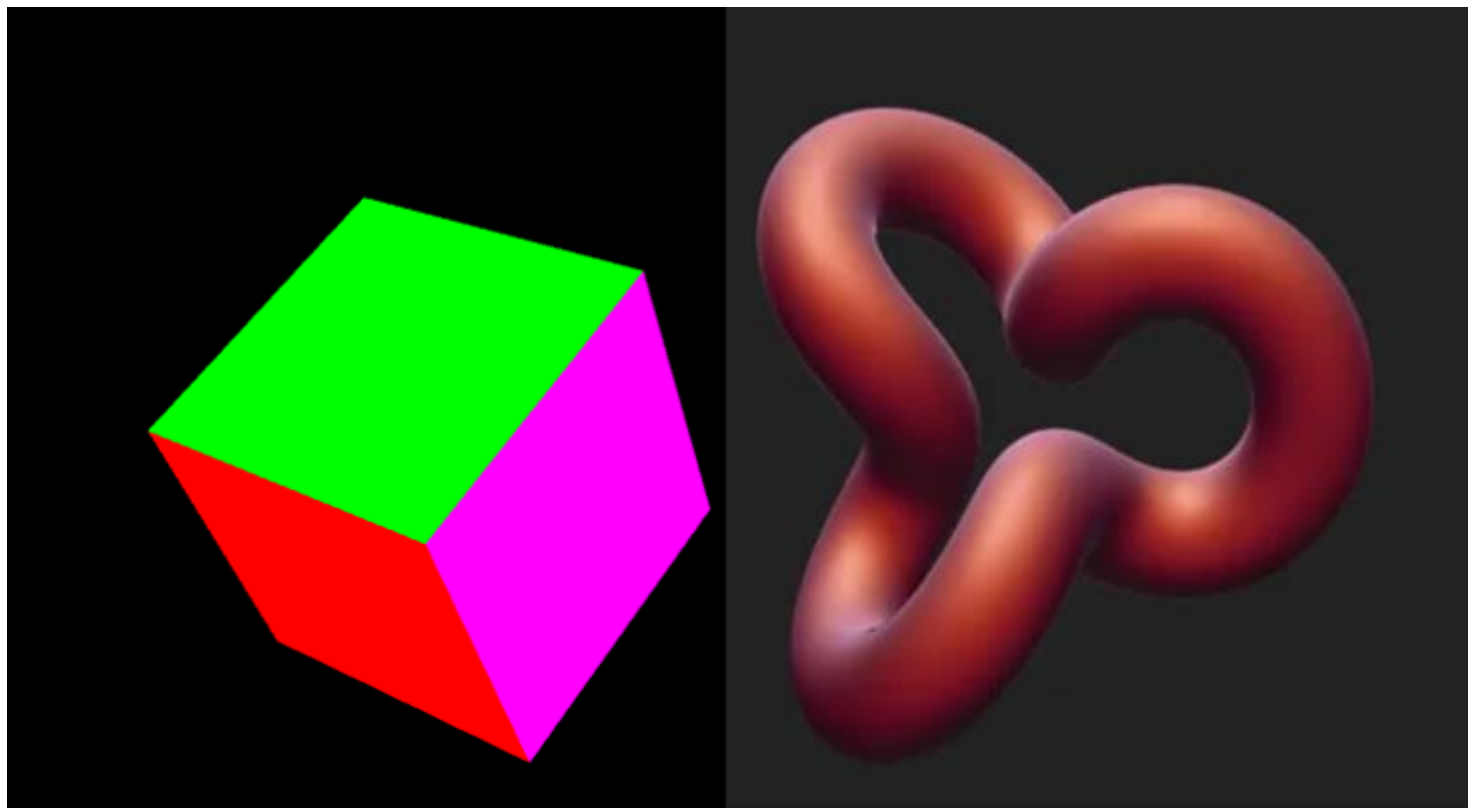
```
// Подключаем VBO  
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

```
// Атрибут с координатами  
glVertexAttribPointer(0,3,GL_FLOAT,GL_FALSE, 8 * sizeof(GLfloat),(GLvoid*)0);  
glEnableVertexAttribArray(0);
```

```
// Атрибут с цветом  
glVertexAttribPointer(1,3,GL_FLOAT,GL_FALSE, 8 * sizeof(GLfloat),(GLvoid*)(3 * sizeof(GLfloat)));  
glEnableVertexAttribArray(1);
```

```
// Атрибут с текстурными координатами  
glVertexAttribPointer(2, 2,GL_FLOAT,GL_FALSE, 8 * sizeof(GLfloat),(GLvoid*)(6 * sizeof(GLfloat)));  
glEnableVertexAttribArray(2);
```


Как вывести два разных объекта?



Вариант: два буфера, две шейдерные программы

```
void Draw() { //Тетраэдр
    glUseProgram(ProgramT);
    glEnableVertexAttribArray(Attrib_vertexT);
    glBindBuffer(GL_ARRAY_BUFFER, VBOT);
    glVertexAttribPointer(Attrib_vertexT, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDrawArrays(GL_TRIANGLES, 0, 12);
    glDisableVertexAttribArray(Attrib_vertexT);
    glUseProgram(0);
    checkOpenGLError();
}
```

```
void Draw() { //куб
    glUseProgram(ProgramC);
    glEnableVertexAttribArray(Attrib_vertexC);
    glBindBuffer(GL_ARRAY_BUFFER, VBOC);
    glVertexAttribPointer(Attrib_vertexC, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDrawArrays(GL_TRIANGLES, 0, 36);
    glDisableVertexAttribArray(Attrib_vertexC);
    glUseProgram(0);
    checkOpenGLError();
}
```

Вариант: два буфера, одна шейдерная программа

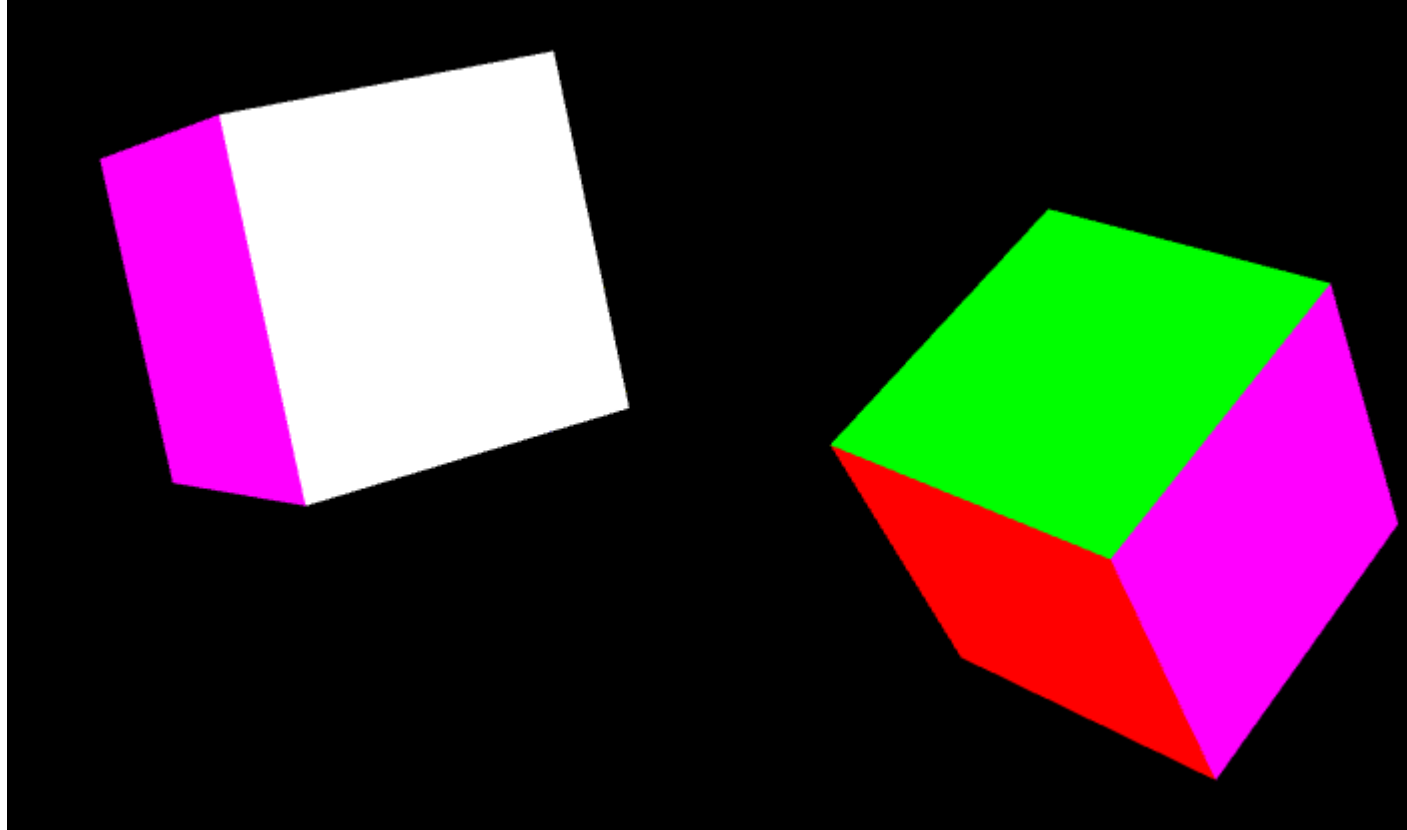
```
glUseProgram(ProgramT);
glEnableVertexAttribArray(Attrib_vertexT);
glEnableVertexAttribArray(Attrib_vertexC);

glBindBuffer(GL_ARRAY_BUFFER, VBOT);
glVertexAttribPointer(Attrib_vertexT, 3, GL_FLOAT, GL_FALSE, 0, 0);
glDrawArrays(GL_TRIANGLES, 0, 12);

glBindBuffer(GL_ARRAY_BUFFER, VBOC);
glVertexAttribPointer(Attrib_vertexC, 3, GL_FLOAT, GL_FALSE, 0, 0);
glDrawArrays(GL_TRIANGLES, 0, 36);

glBindBuffer(GL_ARRAY_BUFFER, 0);
glDisableVertexAttribArray(Attrib_vertexT);
glDisableVertexAttribArray(Attrib_vertexC);
glUseProgram(0);
```

Как вывести два одинаковых объекта?





Различные виды буферов

VBO (объекты буфера вершин)

Объект буфера вершин — это область буфера памяти, созданная в области памяти графической карты, которая используется для хранения различных типов информации атрибутов вершин, таких как координаты вершин, векторы вершин и данные цвета вершин.

Во время рендеринга различные **атрибутные данные** вершин могут быть **взяты непосредственно из VBO**. Поскольку VBO находится в видеопамяти, ему не нужно передавать данные из CPU, и **эффективность** обработки выше.

Можно создать много VBO, и каждый VBO имеет свой уникальный идентификационный идентификатор в OpenGL. Этот идентификатор соответствует конкретному адресу видеопамяти VBO. Через этот идентификатор можно получить доступ к данным в конкретном VBO.

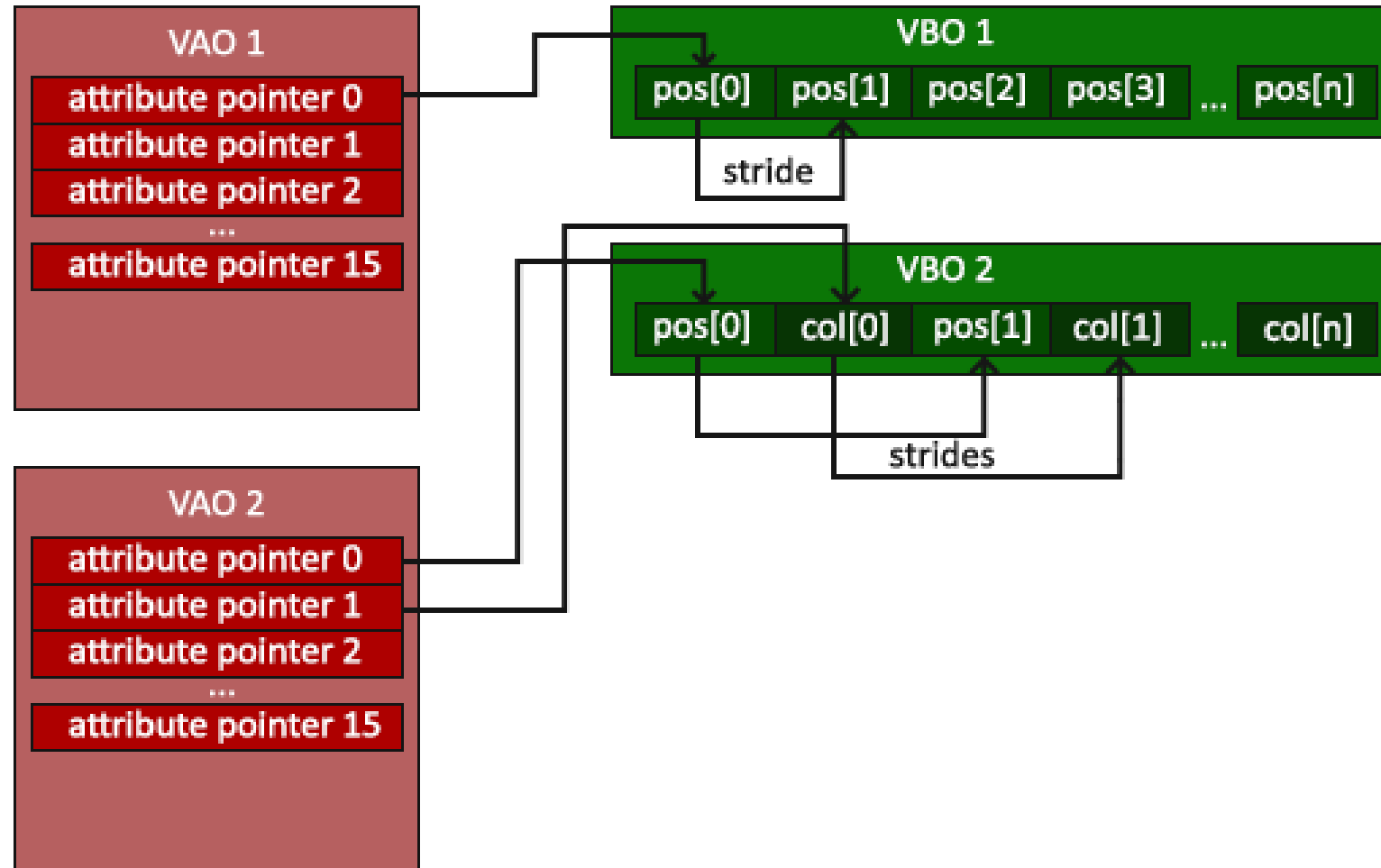
VAO (объект Vertex Array)

- Сохраняет **комбинацию состояний** всех атрибутов данных вершины, сохраняет **формат данных** вершины и **ссылку на VBO**, требуемую данными вершины.

Vertex Array Object (VAO) — объект вершинного массива

- Объект вершинного массива (VAO) может быть также привязан как и VBO, и после этого все последующие вызовы вершинных атрибутов будут храниться в VAO.
- Преимущество этого метода в том, что нам требуется **настроить атрибуты** лишь **единожды**, а все последующие разы будет использована **конфигурация VAO**.
- Также такой метод упрощает смену вершинных данных и конфигураций атрибутов простым привязыванием различных VAO.

Взаимосвязи объектов



VAO хранит

- Вызовы **glEnableVertexAttribArray** или **glDisableVertexAttribArray**
- Конфигурации атрибутов, выполненная через **glVertexAttribPointer**
- VBO, ассоциированные с вершинными атрибутами с помощью **glVertexAttribPointer**

Процесс генерации VAO

```
GLuint VAO;
```

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

Что нужно, чтобы использовать VAO?

```
//Код инициализации выполняется единожды, если, конечно, объект не будет часто изменяться  
glBindVertexArray(VAO); //Привязываем VAO
```

```
// Копируем массив вершин в буфер для OpenGL  
glEnableVertexAttribArray(Attrib_vertex);  
glBindBuffer(GL_ARRAY_BUFFER, VBO);  
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);  
//Устанавливаем указатели на вершинные атрибуты  
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);  
glEnableVertexAttribArray(0);
```

```
//Отвязываем VAO  
glBindVertexArray(0);
```

И тогда ...

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


Element Buffer Object (EBO) или Index Buffer Object (IBO)

- Предположим, что нам надо отрисовать не треугольник, а четырехугольник.
- Мы можем отрисовать четырехугольник с помощью 2 треугольников.

Четырёхугольник

```
GLfloat vertices[] = {  
    // Первый треугольник  
    0.5f, 0.5f, 0.0f, // Верхний правый угол  
    0.5f, -0.5f, 0.0f, // Нижний правый угол  
    -0.5f, 0.5f, 0.0f, // Верхний левый угол  
    // Второй треугольник  
    0.5f, -0.5f, 0.0f, // Нижний правый угол  
    -0.5f, -0.5f, 0.0f, // Нижний левый угол  
    -0.5f, 0.5f, 0.0f // Верхний левый угол  
};
```

Лучше так

```
GLfloat vertices[] = {  
    0.5f, 0.5f, 0.0f, // Верхний правый угол  
    0.5f, -0.5f, 0.0f, // Нижний правый угол  
    -0.5f, -0.5f, 0.0f, // Нижний левый угол  
    -0.5f, 0.5f, 0.0f // Верхний левый угол  
};  
GLuint indices[] = {  
    0, 1, 3, // Первый треугольник  
    1, 2, 3 // Второй треугольник  
};
```

Индексный буферный объект EVO

- Аналог индексного массива для массива вершин
- Содержимое, хранящееся в EVO, является индексом местоположения.
- Также является частью буфера памяти в видеопамяти.

Создание EBO (IBO)

```
GLuint IBO;
```

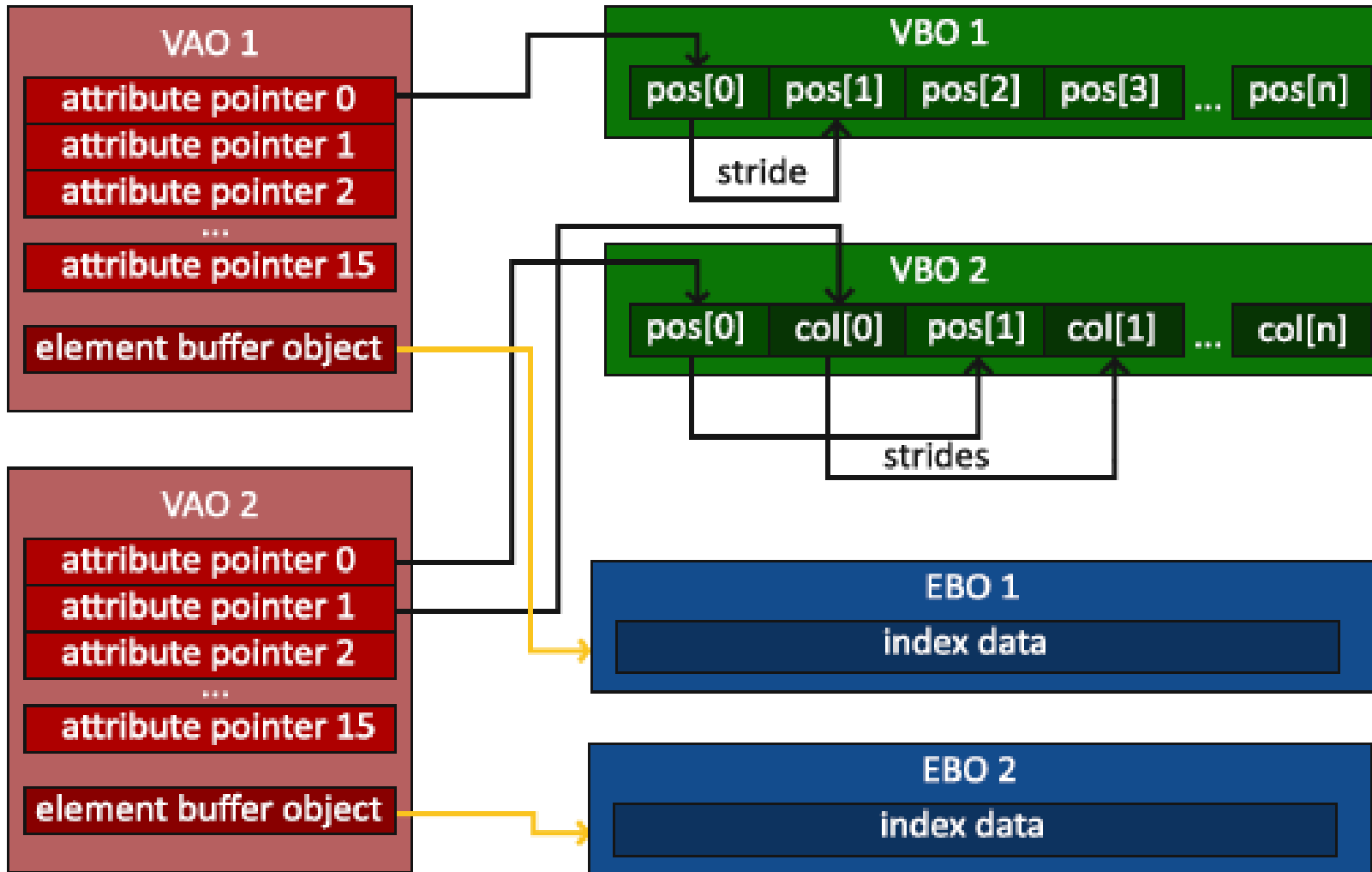
```
glGenBuffers(1, &IBO);
```

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
```

...

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);  
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

VAO с VBO и IBO (EBO)



```
// ...: Код инициализации :: ..
// 1. Привязываем VAO
glBindVertexArray(VAO);
// 2. Копируем наши вершины в буфер для OpenGL
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// 3. Копируем наши индексы в в буфер для OpenGL
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
// 3. Устанавливаем указатели на вершинные атрибуты
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
// 4. Отвязываем VAO (НЕ IBO)
glBindVertexArray(0);
```

[...]

```
// ...: Код отрисовки (в игровом цикле) :: ..
glUseProgram(shaderProgram);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0)
glBindVertexArray(0);
```