

Наложение текстур

Компьютерная графика

Загрузка текстур

```
GLuint texture;
```

```
glGenTextures(1, &texture);
```

```
glBindTexture(GL_TEXTURE_2D, texture);
```

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, photo_img->sizeX, photo_img->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,  
photo_img->data);
```

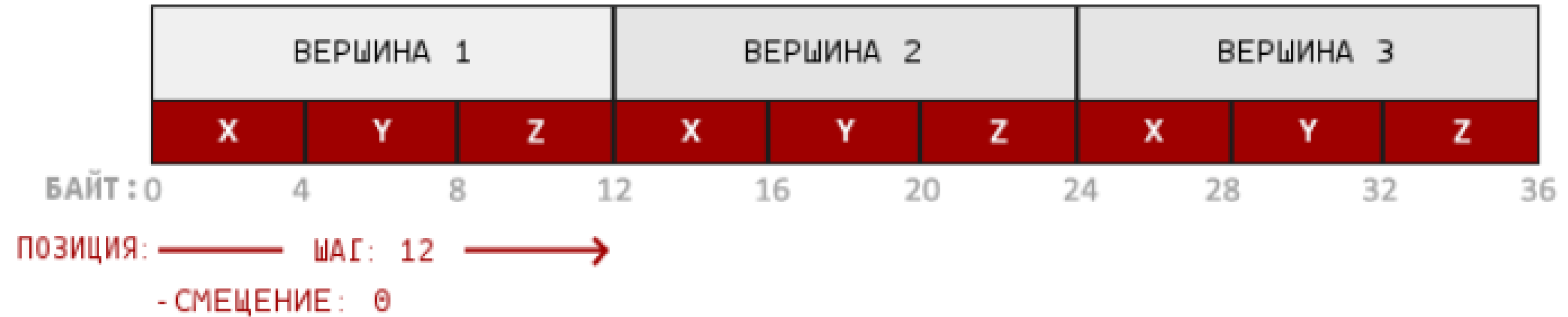
```
glTexImage2D(GL_TEXTURE_2D, 0, 3, photo_img->sizeX, photo_img->sizeY,  
0, GL_RGB, GL_UNSIGNED_BYTE, photo_img->data);
```

1. Целевой тип текстуры.
2. Мипмап-уровень, для текстуры (базовый уровень равен 0).
3. Формат хранения текстуры (RGB).
4. Ширина результирующей текстуры.
5. Высота результирующей текстуры.
6. Должен всегда быть равен 0 (наследие старого кода).
7. Формат данных исходного изображения.
8. Тип данных исходного изображения.
9. Фактические данные изображения.

Готовим данные

```
float vertices[] = {  
    // координаты    // цвета    // текстурные координаты  
    0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, // верхняя правая  
    0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, // нижняя правая  
    -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, // нижняя левая  
    -0.5f, 0.5f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f // верхняя левая  
};
```

Плотно упакованный буфер

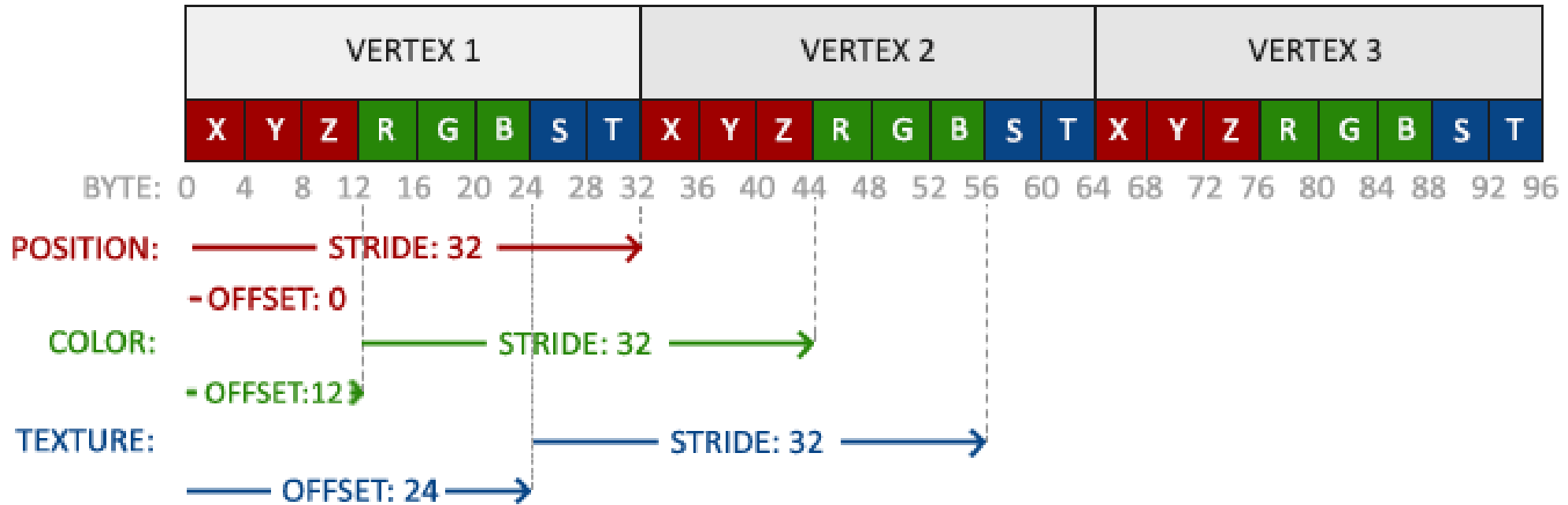


Информация о каждой координате позиции хранится в 32 битном (4 байта) значении с плавающей точкой. Каждая позиция формируется из 3-х значений (координат).

Не существует никакого разделителя между наборами из 3 значений. Такой буфер называется плотно упакованным.

Первое значение в переданных данных — это начало буфера.

Один буфер — три набора данных



glVertexAttribPointer

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

1-й аргумент описывает какой аргумент шейдера мы хотим настроить. Мы хотим специфицировать значение аргумента `position`, позиция которого может быть указана следующим образом: `layout (location = 0)`.

2-й аргумент описывает размер аргумента в шейдере. Поскольку мы использовали `vec3` то мы указываем 3.

3-й аргумент описывает используемый тип данных. Мы указываем `GL_FLOAT`, поскольку `vec` в шейдере использует числа с плавающей точкой.

4-й аргумент указывает необходимость нормализовать входные данные. Если `GL_TRUE`, то все данные будут расположены между 0 (-1 для знаковых значений) и 1. Нам нормализация не требуется, поэтому `GL_FALSE`.

5-й аргумент — шаг — расстояние между наборами данных. Если шаг = 0, тогда OpenGL высчитает шаг (работает только с плотно упакованными наборами данных).

6-й аргумент — смещение начала данных в буфере.

Атрибуты вершин

// 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);
```


Вершинный шейдер

```
layout (location = 0) in vec3 position;  
layout (location = 1) in vec3 color;  
layout (location = 2) in vec2 texCoord;  
  
out vec3 ourColor;  
out vec2 TexCoord;  
  
void main() {  
    gl_Position = vec4(position, 1.0f);  
    ourColor = color;  
    TexCoord = texCoord;  
}
```

Фрагментный шейдер

```
in vec3 ourColor;
```

```
in vec2 TexCoord;
```

```
out vec4 color;
```

```
uniform sampler2D ourTexture;
```

```
void main() {
```

```
    color = texture(ourTexture, TexCoord);
```

```
}
```

Результат наложения текстуры



Место в общей структуре

```
void Draw() {  
    // Устанавливаем шейдерную программу текущей  
    // Включаем массив атрибутов  
    // Подключаем 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);  
    ...  
    // Отключаем VBO  
    // Передаем данные на видеокарту(рисуем)  
    // Отключаем массив атрибутов  
    // Отключаем шейдерную программу  
}
```

А можно было без layout (location = 0)

```
// Устанавливаем связь между параметрами в программе и шейдере
```

```
...
```

```
const char* attr_name_p = "position"; //имя в шейдере  
const char* attr_name_c = "color"; //имя в шейдере  
const char* attr_name_t = "texCoord"; //имя в шейдере
```

```
// Вытягиваем ID атрибута из собранной программы
```

```
Attrib_vertex_p = glGetAttribLocation(Program, attr_name_p);  
Attrib_vertex_c = glGetAttribLocation(Program, attr_name_c);  
Attrib_vertex_t = glGetAttribLocation(Program, attr_name_t);
```

```
...
```

Тогда

```
// Атрибут с координатами
```

```
glVertexAttribPointer(Attrib_vertex_p,3,GL_FLOAT,GL_FALSE, 8 * sizeof(GLfloat),(GLvoid*)0);  
glEnableVertexAttribArray(0);
```

```
// Атрибут с цветом
```

```
glVertexAttribPointer(Attrib_vertex_c,3,GL_FLOAT,GL_FALSE, 8 * sizeof(GLfloat),(GLvoid*)(3* sizeof(GLfloat)));  
glEnableVertexAttribArray(1);
```

```
// Атрибут с текстурными координатами
```

```
glVertexAttribPointer(Attrib_vertex_t, 2,GL_FLOAT,GL_FALSE, 8 * sizeof(GLfloat),(GLvoid*)(6 *  
sizeof(GLfloat)));  
glEnableVertexAttribArray(2);
```

Хотим смешать текстуру и цвет

Фрагментный шейдер: текстура + цвет

```
in vec3 ourColor;  
in vec2 TexCoord;  
out vec4 color;  
uniform sampler2D ourTexture;  
  
void main() {  
    Color=texture(ourTexture,TexCoord)* vec4(ourColor,1.0f);  
}
```


Результат: текстура + цвет



Мультитекстурирование: фрагментный шейдер

```
uniform sampler2D ourTexture1;  
uniform sampler2D ourTexture2;  
void main(){  
    color = mix(texture(ourTexture1, TexCoord), texture(ourTexture2, TexCoord), 0.2);  
}
```

Мультитекстурирование: загрузка текстур

```
GLuint texture1;  
glGenTextures(1, &texture1);
```

```
GLuint texture2;  
glGenTextures(1, &texture2);
```

```
glBindTexture(GL_TEXTURE_2D, texture1);  
glTexImage2D(GL_TEXTURE_2D, 0, 3, photo1_img->sizeX, photo1_img->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,  
photo1_img->data);
```

```
glBindTexture(GL_TEXTURE_2D, texture2);  
glTexImage2D(GL_TEXTURE_2D, 0, 3, photo2_img->sizeX, photo2_img->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,  
photo2_img->data);
```

Мультитекстурирование: привязка текстуры

```
glActiveTexture(GL_TEXTURE0);  
glBindTexture(GL_TEXTURE_2D, texture1);  
glUniform1i(glGetUniformLocation(ourShader.Program, "ourTexture1"), 0);
```

```
glActiveTexture(GL_TEXTURE1);  
glBindTexture(GL_TEXTURE_2D, texture2);  
glUniform1i(glGetUniformLocation(ourShader.Program, "ourTexture2"), 1);
```

С помощью glUniform1i мы можем присвоить значение местоположения текстурному сэмплеру для возможности использования нескольких текстур в одном фрагментном шейдере.

Местоположение текстуры чаще называется **текстурным блоком**.

Текстурный блок по умолчанию — 0, который означает текущий активный текстурный блок.

Результат мультитекстурирования



Иногда такой результат текстурирования



Пути решения проблемы

1. Изменить текстурные координаты в вершинных данных и перевернуть ось Y: вычесть Y координату из 1
2. Изменить вершинный шейдер для переворачивания Y координаты:
`TexCoord=vec2(texCoord.x,1.0f — texCoord.y)`
3. Лучше решить на этапе загрузки изображения