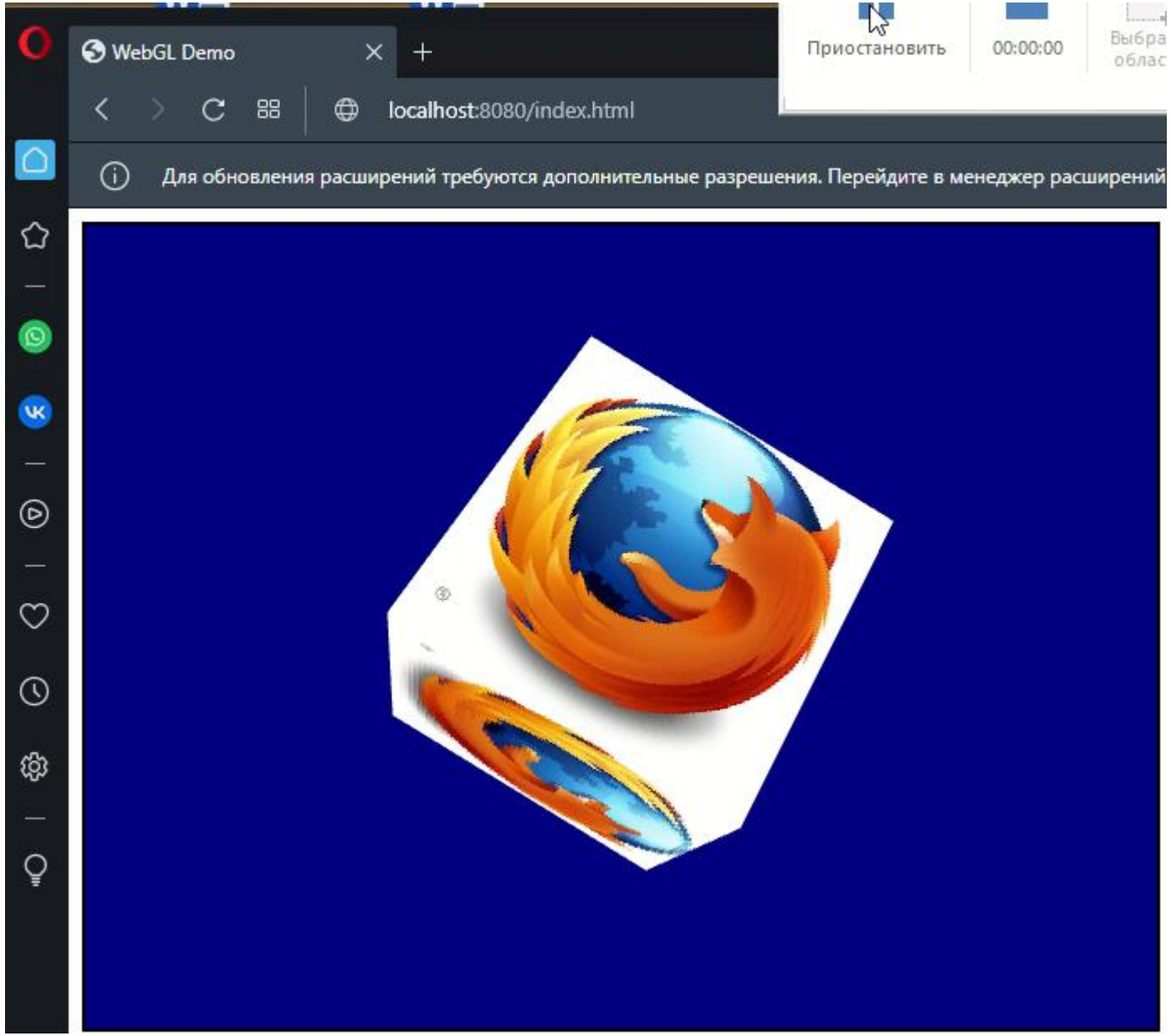
Компьютерная графика
WebGL

Лекция 4

Демяненко Я.М. ЮФУ 2024 MAGnUS


WebGL Demo
localhost:8080/index.html
Для обновления расширений требуются дополнительные разрешения. Перейдите в менеджер расширений
Приостановить
00:00:00

Что передаётся в шейдеры

```
const programInfo = {
    program: shaderProgram,
    attribLocations: {
      vertexPosition: gl.getAttribLocation(shaderProgram, 'aVertexPosition'),
      textureCoord: gl.getAttribLocation(shaderProgram, 'aTextureCoord'),
    },
    uniformLocations: {
      projectionMatrix: gl.getUniformLocation(shaderProgram, 'uProjectionMatrix'),
      modelViewMatrix: gl.getUniformLocation(shaderProgram, 'uModelViewMatrix'),
      uSampler: gl.getUniformLocation(shaderProgram, 'uSampler'),
    }
  };
```

Создание переменной

```
//глобальная переменная

var texture = gl.createTexture();
```

Загрузка текстуры

Для установки изображения в качестве текстуры нам нужен элемент
var image = new Image();

Изображение, установленное для данного элемента, и будет устанавливаться в качестве текстуры.

Два способа:

- заранее определить в структуре DOM веб-страницы элемент img и с помощью его атрибута src установить какое-либо изображение
- динамически в коде javascript создать данный элемент

Если динамически

Поскольку текстура не сразу загружается, то используем обработку события onload:

```
function setTextures(){
   texture = gl.createTexture();
   gl.bindTexture(gl.TEXTURE_2D, texture);

   var image = new Image();

   image.onload = function() {
         handleTextureLoaded(image, texture);
         setupWebGL();
         draw();
    }

    image.src = "brick_045.jpg";
    shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");
    gl.uniform1i(shaderProgram.samplerUniform, 0);
}
```

Cross-Origin Resource Sharing (CORS)

Важно помнить, что загрузка текстур следует правилам кросс-доменности, что означает, что вы можете загружать текстуры только с сайтов, для которых ваш контент является CORS доверенным.

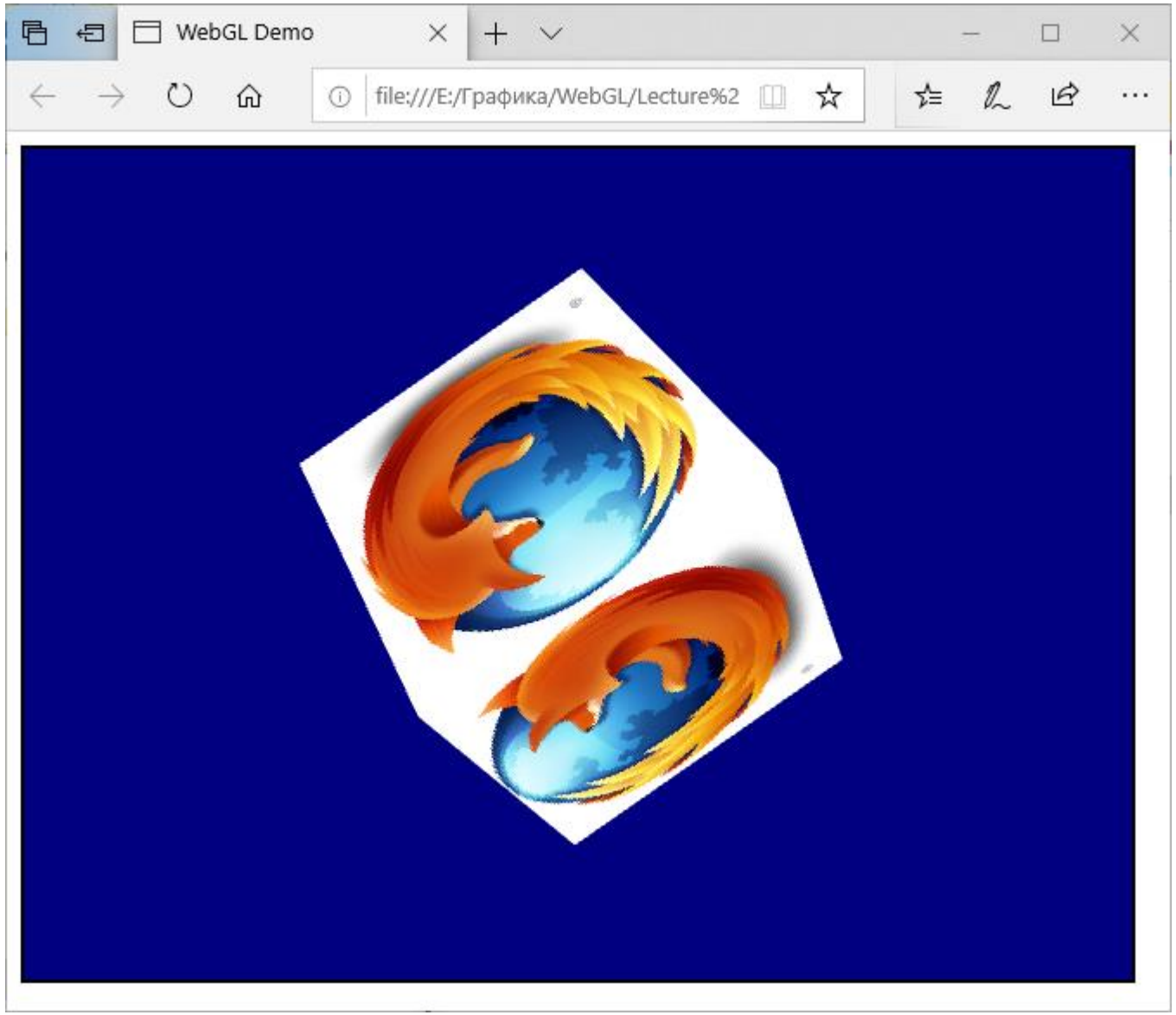
Если динамически - доработанное

Поскольку текстура не сразу загружается, то используем обработку события onload:

```
function setTextures(){
   texture = gl.createTexture();
   gl.bindTexture(gl.TEXTURE_2D, texture);

   var image = new Image();
   image.crossOrigin = "anonymous";

   image.onload = function() {
         handleTextureLoaded(image, texture);
         setupWebGL();
         draw();
    }

    image.src = "brick_045.jpg";
    shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");
    gl.uniform1i(shaderProgram.samplerUniform, 0);
}
```

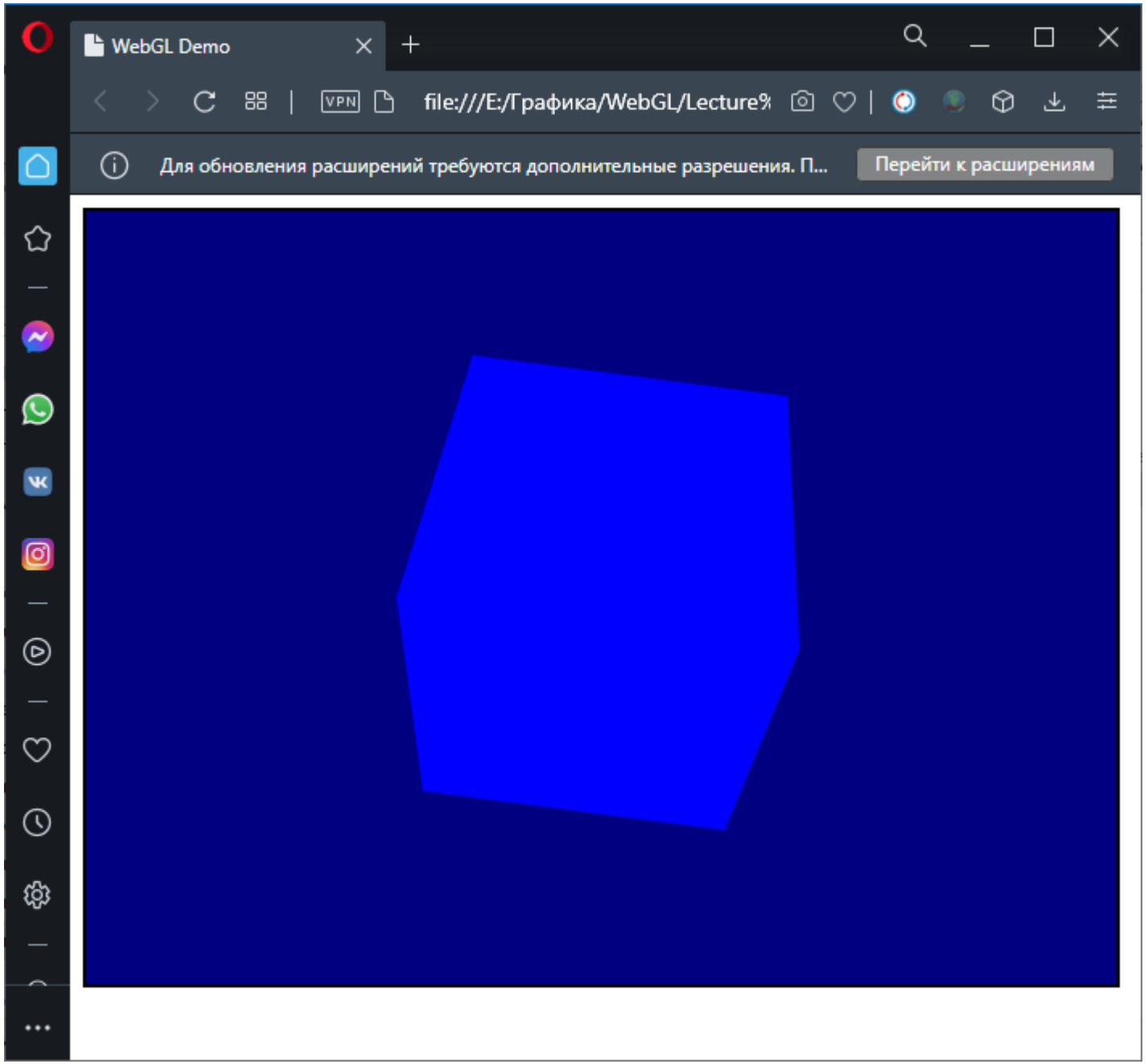
Хотя

Чтобы работало везде, видимо, действительно придется запускать локальный сервер.

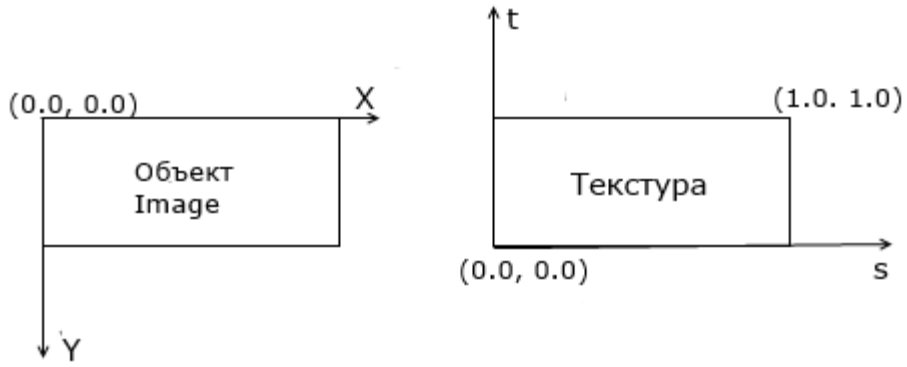

WebGL Demo
file:///E:/Графика/WebGL/Lecture%2

А пока текстура загружается…

```
const pixel = new Uint8Array([0, 0, 255, 255]);  // непрозрачный синий

gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, width, height, border, srcFormat, srcType, pixel);

const level = 0;
const internalFormat = gl.RGBA;
const width = 1;
const height = 1;
const border = 0;
const srcFormat = gl.RGBA;
const srcType = gl.UNSIGNED_BYTE;
```

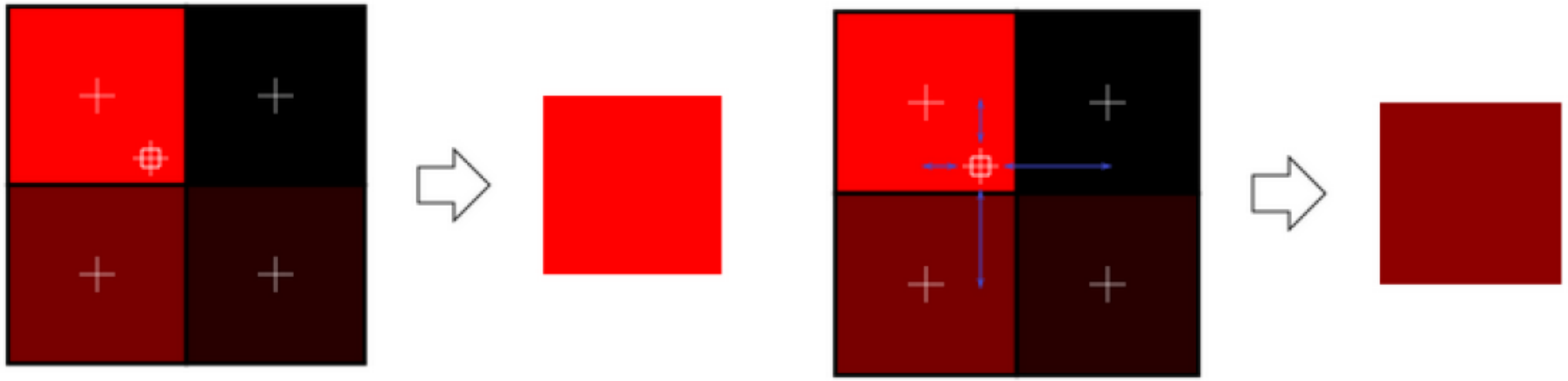

WebGL Demo
file:///E:/Графика/WebGL/Lecture%
Для обновления расширений требуются дополнительные разрешения. П...
Перейти к расширениям

Настройка всех параметров текстурирования




```
function handleTextureLoaded(image, texture) {

   gl.bindTexture(gl.TEXTURE_2D, texture);

   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
  //указывает далее идущему методу gl.texImage2D(), как текстура должна позиционироваться. Так, в данном случае мы
     передаем в качестве параметра значение  gl.UNPACK_FLIP_Y_WEBGL - этот параметр указывает методу
     gl.texImage2D(), что изображение надо перевернуть относительно горизонтальной оси.

   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
}
```

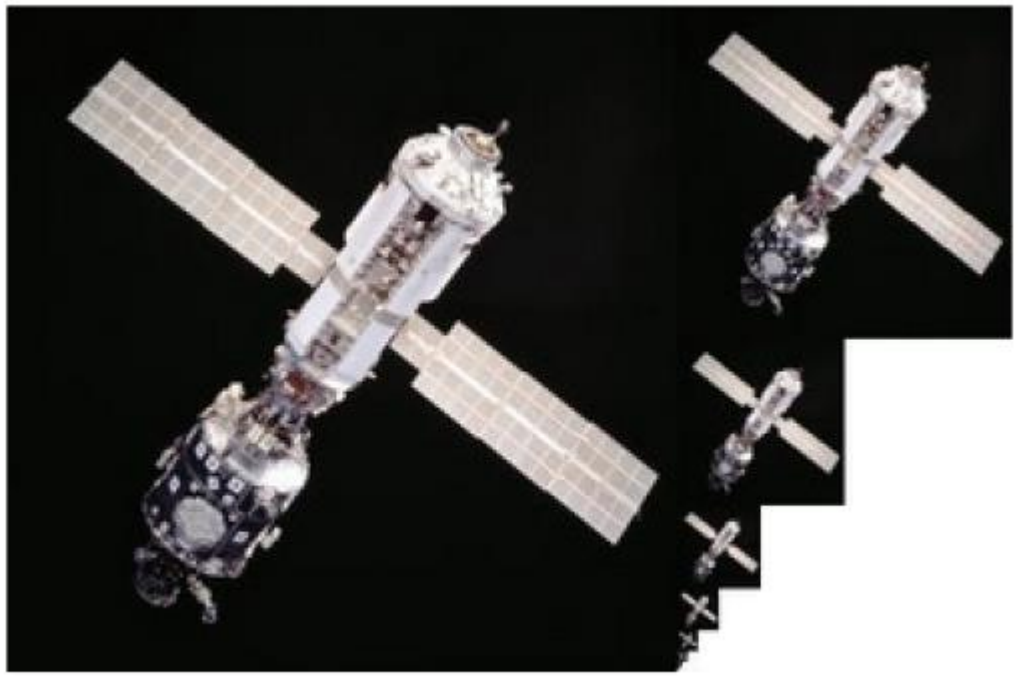
NEAREST и LINEAR



Варианты

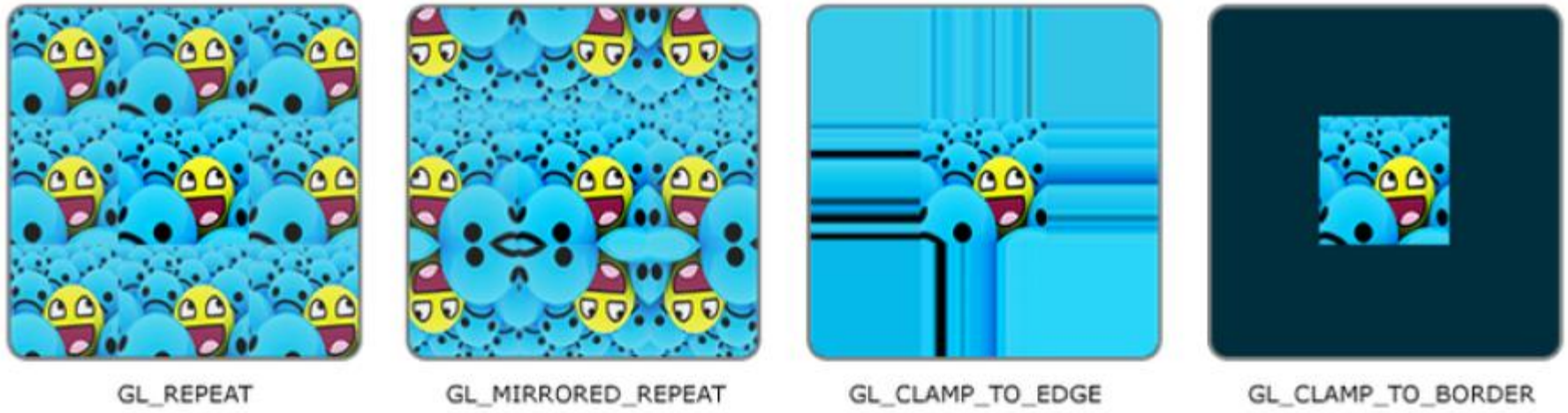
```
 // У WebGL1 разные требования к изображениям, имеющим размер степени 2,
// и к не имеющим размер степени 2

if (isPowerOf2(image.width) && isPowerOf2(image.height)) {
       // Размер соответствует степени 2. Создаем MIP'ы.
       gl.generateMipmap(gl.TEXTURE_2D);
    } else {
       // Размер не соответствует степени 2.
       // Отключаем MIP'ы и устанавливаем натяжение по краям

       // также разрешено gl.NEAREST вместо gl.LINEAR, но не mipmap.
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);

      // Не допускаем повторения по s-координате.
     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);

     // Не допускаем повторения по t-координате.
     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
}
```

MIPMAP



Если текстурные координаты выйдут за промежуток 0-1

GL_REPEAT: по умолчанию. Повторяет текстуру
GL_MIRRORED_REPEAT: Похож на GL_REPEAT, но отражается
GL_CLAMP_TP_EDGE: привязывает координаты между 0 и 1. Выход за пределы координат будут привязаны к границам
GL_CLAMP_TO_BORDER: Координаты, выходящие за пределы, будут цвета границы



Координаты текстуры

```
const textureCoordinates = [
   // Front
   0.0,  0.0,          1.0,  0.0,          1.0,  1.0,          0.0,  1.0,
   // Back
   0.0,  0.0,          1.0,  0.0,          1.0,  1.0,          0.0,  1.0,
   // Top
   0.0,  0.0,          1.0,  0.0,          1.0,  1.0,          0.0,  1.0,
   // Bottom
   0.0,  0.0,          1.0,  0.0,          1.0,  1.0,          0.0,  1.0,
   // Right
   0.0,  0.0,          1.0,  0.0,          1.0,  1.0,          0.0,  1.0,
   // Left
   0.0,  0.0,          1.0,  0.0,          1.0,  1.0,          0.0,  1.0,
 ];
```

Координаты текстуры лежат в промежутке между 0.0 и 1.0. Размерность текстуры нормализуется в пределах между 0.0 и 1.0, независимо от реального размера изображения

Вершинный шейдер

```
attribute vec4 aVertexPosition;
attribute vec2 aTextureCoord;

uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;

out highp vec2 vTextureCoord;

void main(void) {
  gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
  vTextureCoord = aTextureCoord;
}
```

Фрагментный шейдер

```
precision highp float;
uniform sampler2D uSampler;
in vec2 vTextureCoords;

 void main(void) {
   gl_FragColor = texture2D(uSampler, vTextureCoords);
 }
```

Указываем WebGL, как извлечь текстурные координаты из буфера

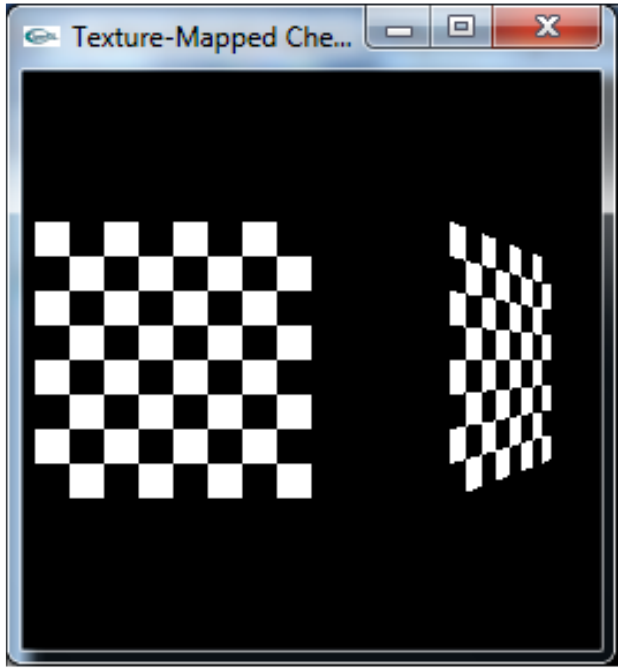
```
const num = 2; // каждая координата состоит из 2 значений
const type = gl.FLOAT; // данные в буфере имеют тип 32-bit float
const normalize = false; // не нормализуем
const stride = 0; // сколько байт между одним набором данных и следующим
const offset = 0; // стартовая позиция в байтах внутри набора данных

gl.bindBuffer(gl.ARRAY_BUFFER, buffers.textureCoord);
gl.vertexAttribPointer(programInfo.attribLocations.textureCoord, num, type, normalize, stride, offset);
gl.enableVertexAttribArray(programInfo.attribLocations.textureCoord);
```

Далее

```
// Указываем WebGL, что мы используем текстурный регистр 0
  gl.activeTexture(gl.TEXTURE0);

  // Связываем текстуру с регистром 0
  gl.bindTexture(gl.TEXTURE_2D, texture);

  // Указываем шейдеру, что мы связали текстуру с текстурным регистром 0
  gl.uniform1i(programInfo.uniformLocations.uSampler, 0);
```

WebGL имеет минимум 8 текстурных регистров; первый из них gl.TEXTURE0

Генерация текстур




```
checkImage [ checkImageHeight ] [ checkImageWidth ] [ 4 ] ;
```

```
for ( i =0; i<checkImageHeight ; i++) {
    for ( j =0; j<checkImageWidth ; j++) {
            c=((( i&0x8 )==0)^(( j&0x8 )==0))255;
            checkImage [ i ] [ j ] [ 0 ]=( GLubyte ) c ;
            checkImage [ i ] [ j ] [ 1 ]=( GLubyte ) c ;
            checkImage [ i ] [ j ] [ 2 ]=( GLubyte ) c ;
            checkImage [ i ] [ j ] [ 3 ]=( GLubyte ) 255 ;
            }
     }
```

Смешивание текстур

Имитация смешивания — Отбрасывание фрагментов

Реальное смешивание

Рендер полупрозрачных текстур

Имитация смешивания — Отбрасывание фрагментов

Отбрасываем фрагменты, содержащие прозрачные части текстуры, не сохраняя их в буфере цвета

Что делать?)

gl.TexImage2D(GL_TEXTURE_2D, 0, **GL_RGBA**, width, height, 0, **GL_RGBA**, GL_UNSIGNED_BYTE, data);

Во фрагментом шейдере выборка в вектор с 4мя компонентами, чтобы не остаться только с RGB значениями:

```
void main() {
   // FragColor = vec4(vec3(texture(texture1, TexCoords)), 1.0);
   FragColor = texture(texture1, TexCoords);
}
```

Как убрать?)

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoords;

uniform sampler2D texture1;

void main() {
    vec4 texColor = texture(texture1, TexCoords);
    if(texColor.a < 0.1)
        discard;
    FragColor = texColor;
}
```

Как избежать артефакта рамки?

Для избежания артефакта появления полупрозрачной цветной рамки вокруг выведенной текстуры при использовании текстур с прозрачностью нужно параметр повтора установить в GL_CLAMP_TO_EDGE.

```
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

Смешивание — это не то же самое, что прозрачность,
это просто одна из техник, которая может быть использована для получения эффекта прозрачности.

Реальное смешивание

Для рендера изображений с объектами, имеющими разную степень непрозрачности мы должны включить режим смешивания.

gl.Enable(GL_BLEND);

Как это работает?

Смешивание OpenGL выполняется по следующей формуле

$$\bar{C}_{result} = \bar{C}_{source} * F_{source} + \bar{C}_{destination} * F_{destination}$$

где C_source – вектор цвета источника. Это значение цвета, полученное из текстуры.
C_destination – вектор цвета приемника. Это значение цвета, хранимое на данный момент в буфере цвета.
F_source – множитель источника. Задает степень влияния альфа-компоненты на цвет источника.
F_destination – множитель приемника. Задает степень влияния альфа-компоненты на цвет приемника.

Как настроить коэффициенты смешивания?

gl.BlendFunc(GLenum sfactor, GLenum dfactor)

Первый параметр функции **gl.blendFunc** определяет исходный множитель, а второй — целевой множитель.

Исходный фрагмент — отрисовываем сейчас, а целевой фрагмент — уже находится во фреймбуфере

В нашем первом примере
значение прозрачности исходного фрагмента постоянное значение — единица (непрозрачный)

gl.blendFunc(gl.SRC_ALPHA, gl.ONE);

Параметр	Значение множителя
GL_ZERO	0
GL_ONE	1
GL_SRC_COLOR	$\overline{C}_{source}$
GL_ONE_MINUS_SRC_COLOR	$1 - \overline{C}_{source}$
GL_DST_COLOR	$\overline{C}_{destination}$
GL_ONE_MINUS_DST_COLOR	$1 - \overline{C}_{destination}$
GL_SRC_ALPHA	Равен альфа-компоненте *alpha* вектора $\overline{C}_{source}$
GL_ONE_MINUS_SRC_ALPHA	1 — *alpha* вектора $\overline{C}_{source}$
GL_DST_ALPHA	Равен альфа-компоненте *alpha* вектора $\overline{C}_{destination}$
GL_ONE_MINUS_DST_ALPHA.	1 — *alpha* вектора $\overline{C}_{destination}$
GL_CONSTANT_COLOR	Равен вектору цвета $\overline{C}_{constant}$
GL_ONE_MINUS_CONSTANT_COLOR	$1 - \overline{C}_{constant}$
GL_CONSTANT_ALPHA	Равен альфа-компоненте *alpha* вектора $\overline{C}_{constant}$
GL_ONE_MINUS_CONSTANT_ALPHA	1 — *alpha* вектора $\overline{C}_{constant}$

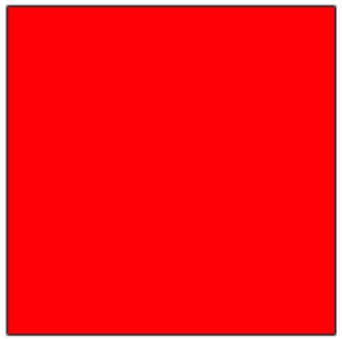
Если смешиваются цвета

Если **gl.blendFunc**(gl.SRC_ALPHA, gl.ONE);

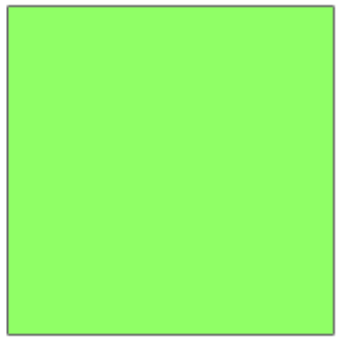
То будет так

$R_{result} = R_s * A_s + R_d$
$G_{result} = G_s * A_s + G_d$
$B_{result} = B_s * A_s + B_d$
$A_{result} = A_s * A_s + A_d$

Как получить?

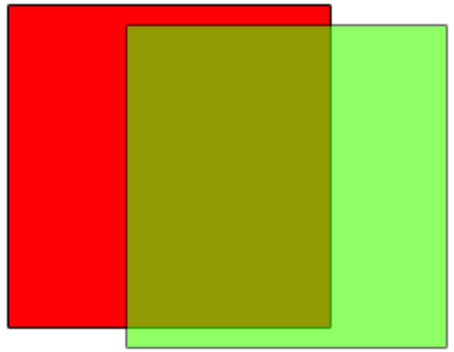


(1.0, 0.0, 0.0, 1.0)



(0.0, 1.0, 0.0, 0.6)

$$\bar{C}_{result} = \begin{pmatrix} 0.0 \\ 1.0 \\ 0.0 \\ 0.6 \end{pmatrix} * 0.6 + \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 1.0 \end{pmatrix} * (1 - 0.6)$$



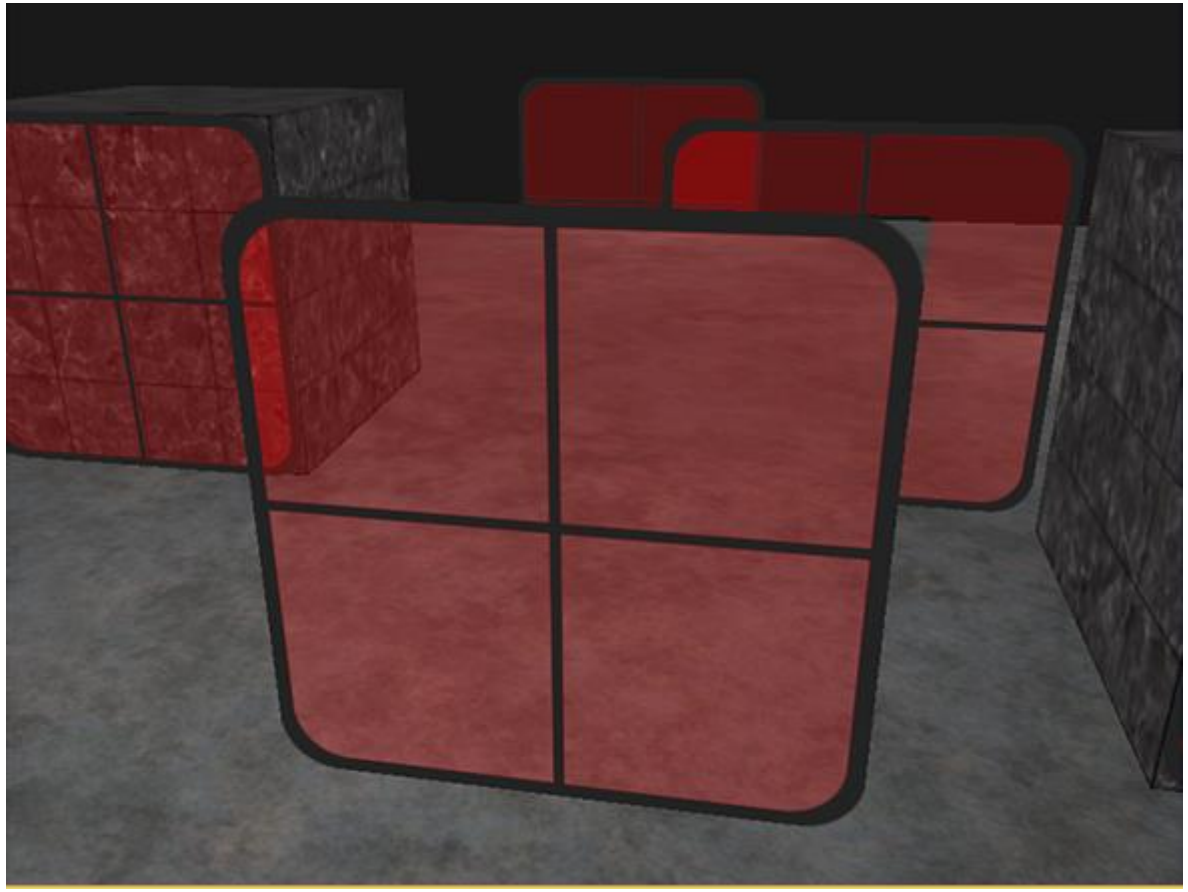
(0.4, 0.6, 0.0, 0.76)

Следует выбрать такие параметры, чтобы коэффициент источника равнялся alpha (значение альфа-компоненты) цвета источника, а коэффициент приемника равнялся 1 – alpha.
Что равнозначно вызову:
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

А можно и так

```
gl.enable(gl.BLEND); //смешивание выключено по умолчанию

gl.disable(gl.DEPTH_TEST); // отключить проверку глубины


in vec2 vTextureCoord;
in vec3 vLightWeighting;

uniform float uAlpha;

uniform sampler2D uSampler;

void main(void) {
   vec4 textureColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
   gl_FragColor = vec4(textureColor.rgb * vLightWeighting, textureColor.a * uAlpha);
}
```

Артефакты. Почему так?



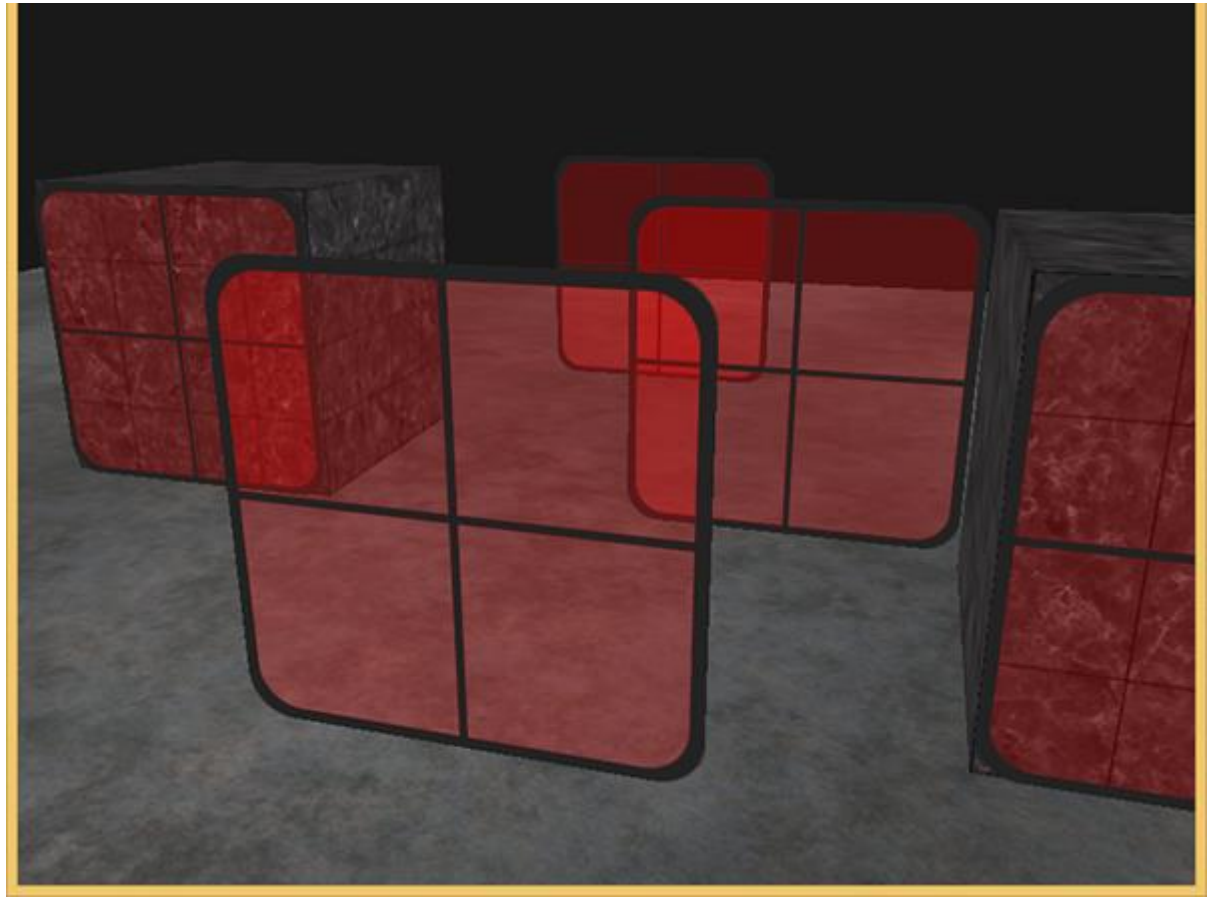
Так как же получить «настоящую» прозрачность?

Устанавливаем исходный множитель в SRC_ALPHA, а целевой множитель в ONE_MINUS_SRC_ALPHA.

Но исходный и целевой фрагмент обрабатываются по-разному, и поэтому остается зависимость от порядка отрисовки.

Полностью прозрачные примитивы следует отрисовывать первыми,
затем частично прозрачные в порядке от отдаленных к ближним.

Корректная прозрачность



Опять про буферы

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.

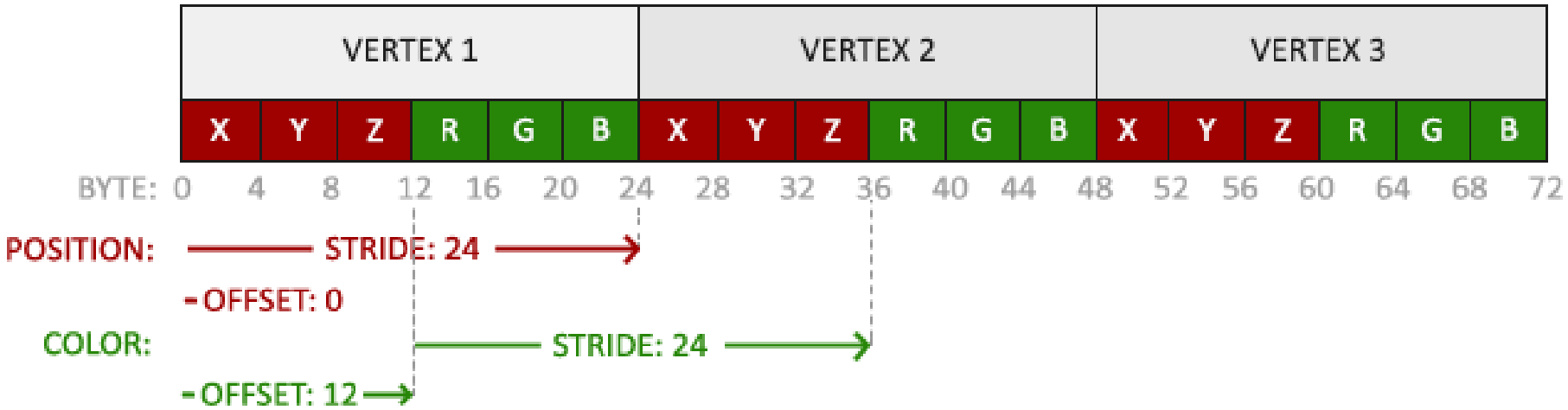
Во **время рендеринга** различные **атрибутные данные** вершин могут быть **взяты непосредственно из VBO**. Поскольку VBO находится в видеопамяти, ему не нужно передавать данные из CPU, и **эффективность** обработки выше.

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.

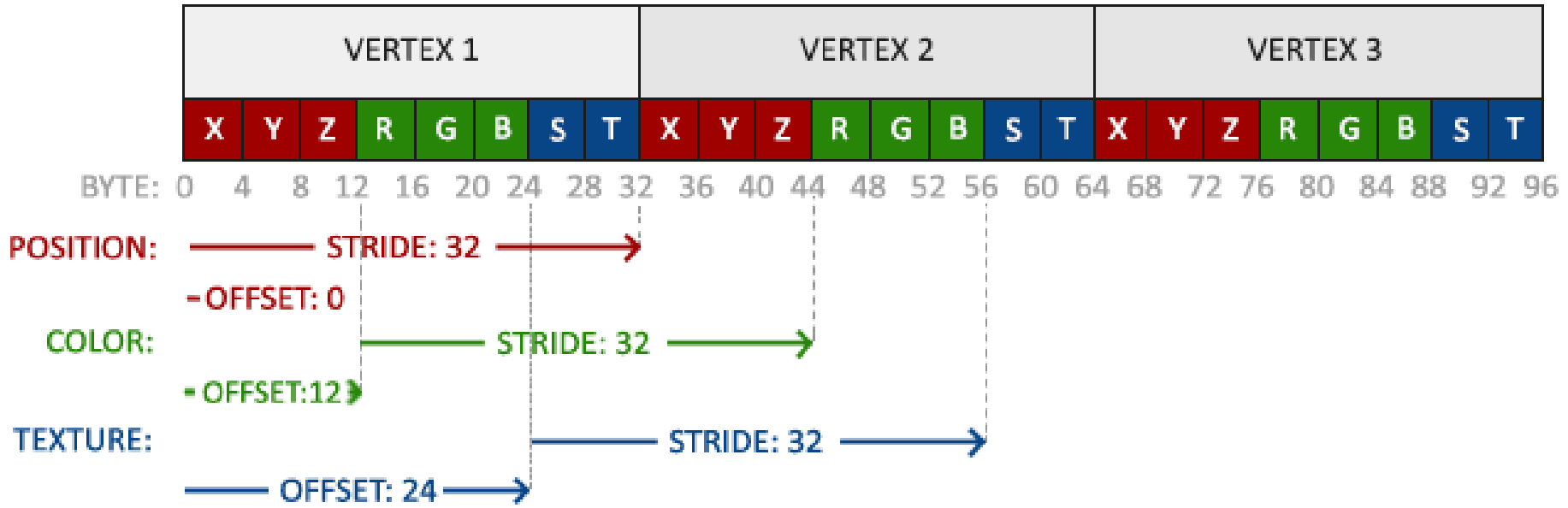
Формат вершинного буфера

Плотно упакованный буфер

VBO в памяти: координаты и цвет



VBO в памяти: координаты, цвет и текстура



VBO: координаты, цвет и текстура

```
gl.EnableVertexAttribArray(0);
gl.EnableVertexAttribArray(1);
gl.EnableVertexAttribArray(2);

// Подключаем VBO
gl.BindBuffer(GL_ARRAY_BUFFER, VBO);

// Атрибут с координатами
gl.VertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
// Атрибут с цветом
gl.VertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(3* sizeof(GLfloat)));
// Атрибут с текстурой
gl.VertexAttribPointer(2, 2, GL_FLOAT,GL_FALSE,   8 * sizeof(GLfloat), (GLvoid*)(6 * sizeof(GLfloat)));
```

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.

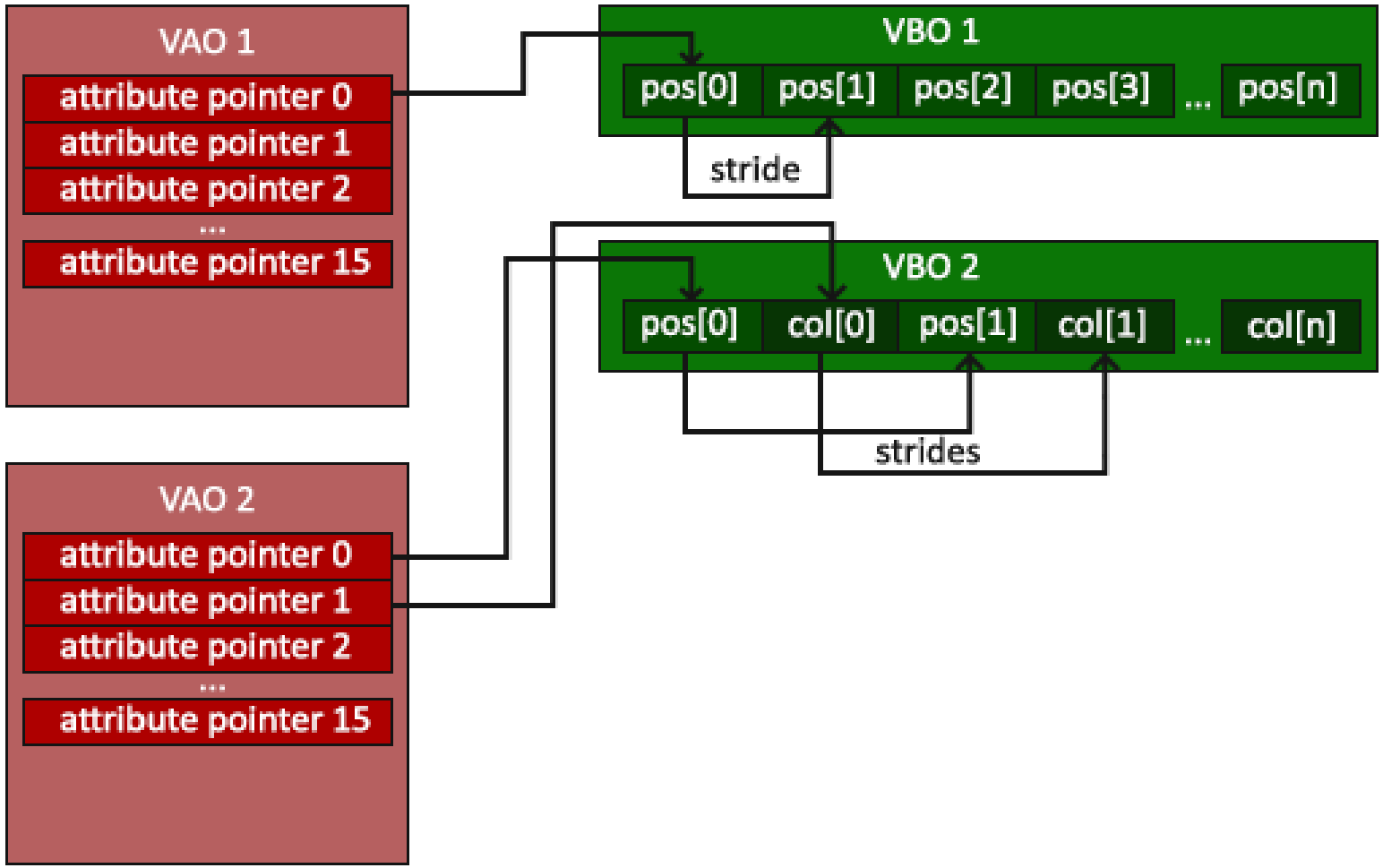
- Преимущество этого метода в том, что нам требуется **настроить атрибуты** лишь **единожды**, а все последующие разы будет использована **конфигурация VAO**.

- Также такой метод упрощает смену вершинных данных и конфигураций атрибутов простым привязыванием различных VAO.

VAO хранит

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

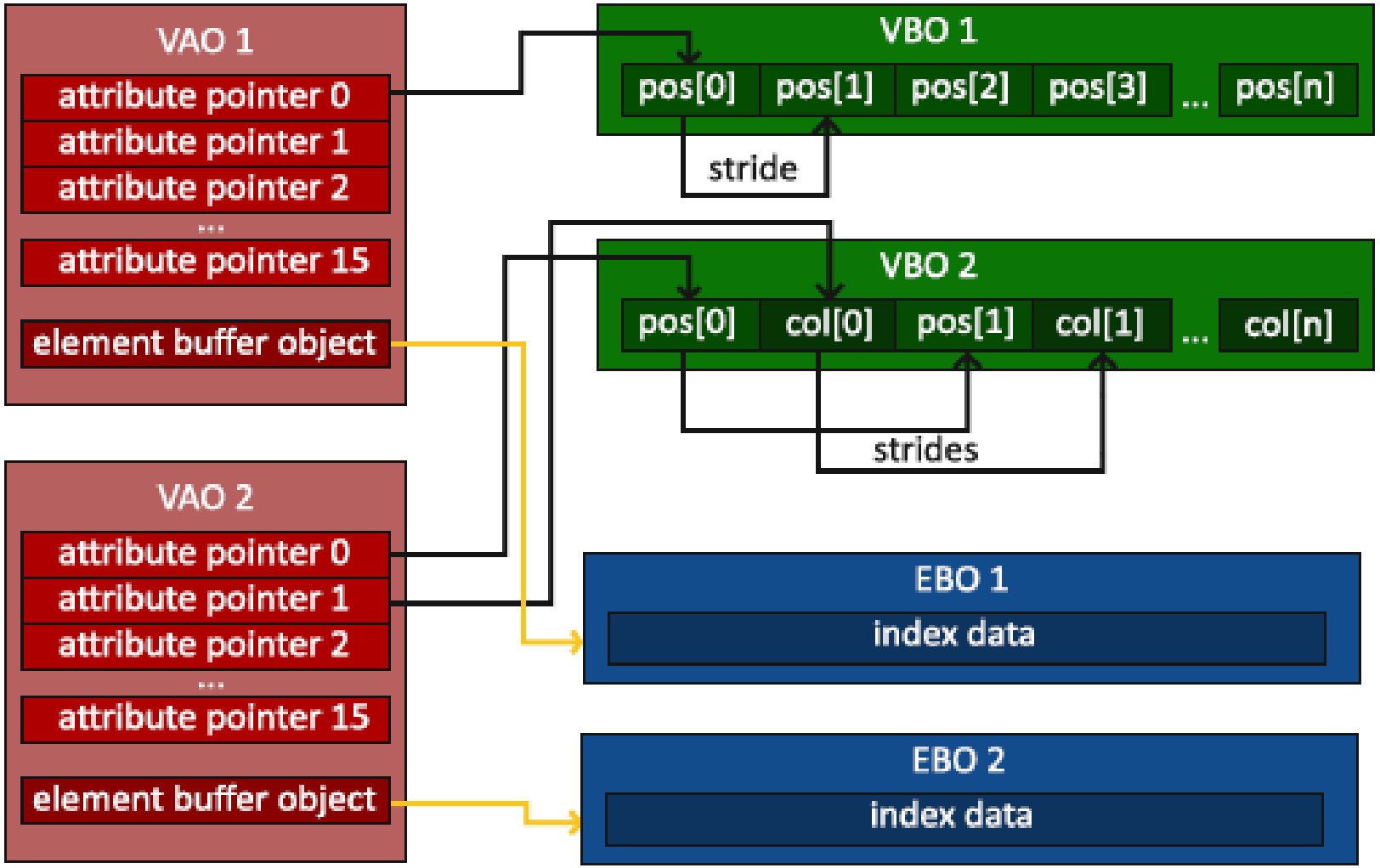
Взаимосвязи объектов



Индексный буферный объект EBO

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

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);
```