WebGL

Дронников Игорь
СПБГЭТУ ФКТИ гр.4383

Основные положения

- **WebGL** представляет собой технологию, базирующуюся на OpenGL ES 2.0 и предназначенную для рисования и отображения интерактивной 2D- и 3D-графики в веб-браузерах. При этом для работы с данной технологией не требуются сторонние плагины или библиотеки.
- **WebGL** возник из экспериментов над Canvas 3D разработчика из компании Mozilla в 2006 году. Была организована рабочая группа с участием крупнейших разработчиков браузеров Apple, Google, Mozilla, Opera для работы над спецификацией технологии. И 3 марта 2011 года была представлена спецификация WebGL 1.0.
- Последняя версия: 1.0.2 (1 марта 2013)

Основные положения

- Вся работа веб-приложений с использованием WebGL основана на коде JavaScript, а некоторые элементы кода - шейдеры могут выполняться непосредственно на графических процессорах на видеокартах, благодаря чему разработчики могут получить доступ к дополнительным ресурсам компьютера, увеличить быстродействие.
- Таким образом, для создания приложений разработчики могут использовать стандартные для веб-среды технологии HTML/CSS/JavaScript и при этом также применять аппаратное ускорение графики.

Основные положения

- Если создание настольных приложений работающих с 2d и 3d-графикой нередко ограничивается целевой платформой, то здесь главным ограничением является только поддержка браузером технологии WebGL.
- Веб-приложения, построенные с использованием данной платформы, будут доступны в любой точке земного шара при наличии сети интернет вне зависимости от используемой платформы: то ли это десктопы с ОС Windows, Linux, Mac, то ли это смартфоны и планшеты, то ли это игровые консоли.

Поддержка браузерами

Десктопные браузеры

- Mozilla Firefox (с 4-й версии)
- Google Chrome (с 9-й версии)
- Safari (с 6-й версии, по умолчанию поддержка WebGL отключена)
- Opera (с 12-й версии, по умолчанию поддержка WebGL отключена)
- IE (с 11-й версии, для других версий можно воспользоваться сторонними плагинами, например, IEWebGL)

Мобильные браузеры и платформы

- Android-браузер (поддерживает WebGL только на некоторых устройствах, например, на смартфонах Sony Ericsson Xperia и некоторых смартфонах Samsung)
- Opera Mobile (начинаю с 12-й версии и только для ОС Android)
- IOS
- Firefox for mobile

Преимущества использования WebGL

- Кроссбраузерность и отсутствие привязки к определенной платформе. Windows, MacOS, Linux - все это не важно, главное, чтобы браузер поддерживал WebGL
- Использование языка JavaScript, который достаточно распространен
- Автоматическое управление памятью. В отличие от OpenGL в WebGL не надо выполнять специальные действия для выделения и очистки памяти
- Поскольку WebGL для рендеринга графики использует графический процессор на видеокарте (GPU), то для этой технологии характерна высокая производительность, которая сравнима с производительностью нативных приложений.

Недостатки использования WebGL

- Специалисты по вопросами безопасности британской компании Context выявили, что злоумышленник может на вредоносном сайте разместить код WebGL, который может быть напрямую передан в графический процессор, в результате чего компьютер просто выключится — тем самым можно организовать DoS атаку на компьютеры пользователей.

- Специалисты по безопасности полагают, что эту же уязвимость можно использовать для создания полноценных троянских программ. Для этого необходимо использовать технологии, позволяющие выполнять обычный код на графических ядрах — такие, как OpenCL или NVIDIA CUDA.

Недостатки использования WebGL

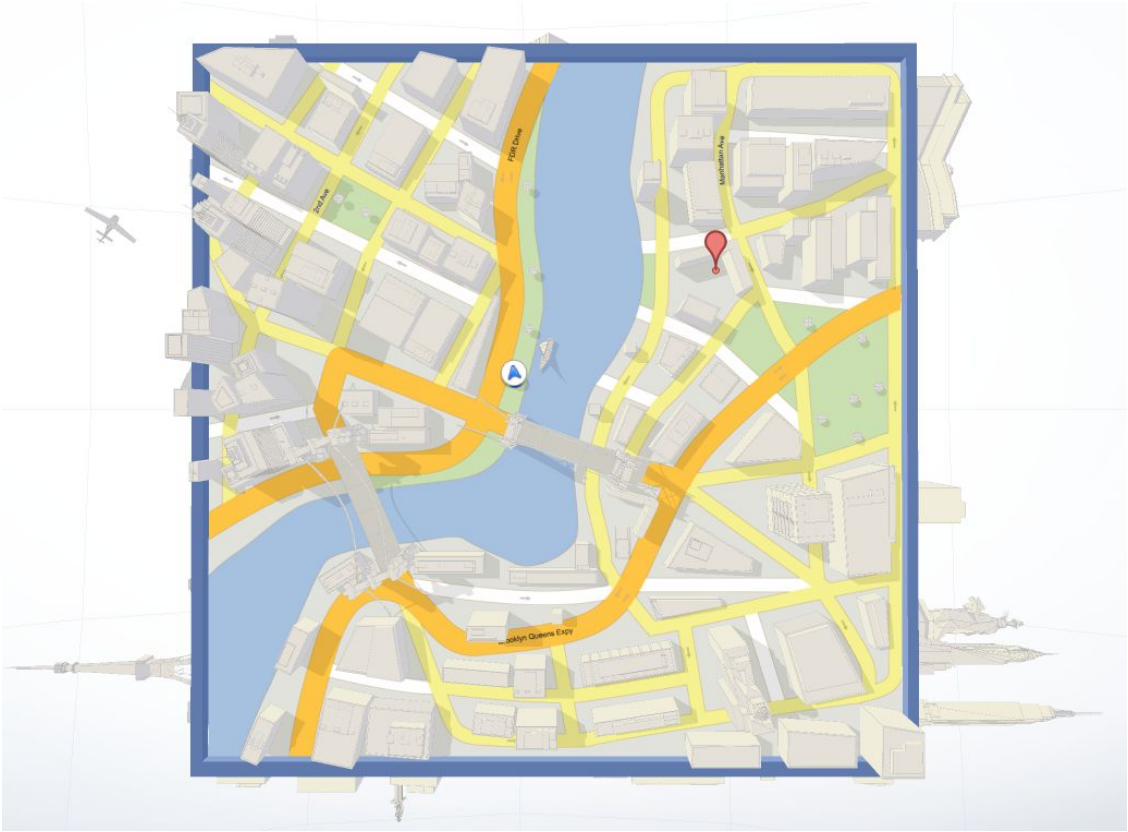
- По данным специалистов, данная уязвимость имеется как в 2D-, так и в 3D-режимах, как для чипов Nvidia, так и для AMD.

- В ответ представители Khronos Group заявили, что существует модуль для определения подобного рода атак на WebGL, который могут использовать производители графических карт. Также часть вины была переложена на производителей видеокарт, которые не выпускают обновленные драйвера с улучшенной защитой.

Примеры использования WebGL

- **Google Maps Cube**

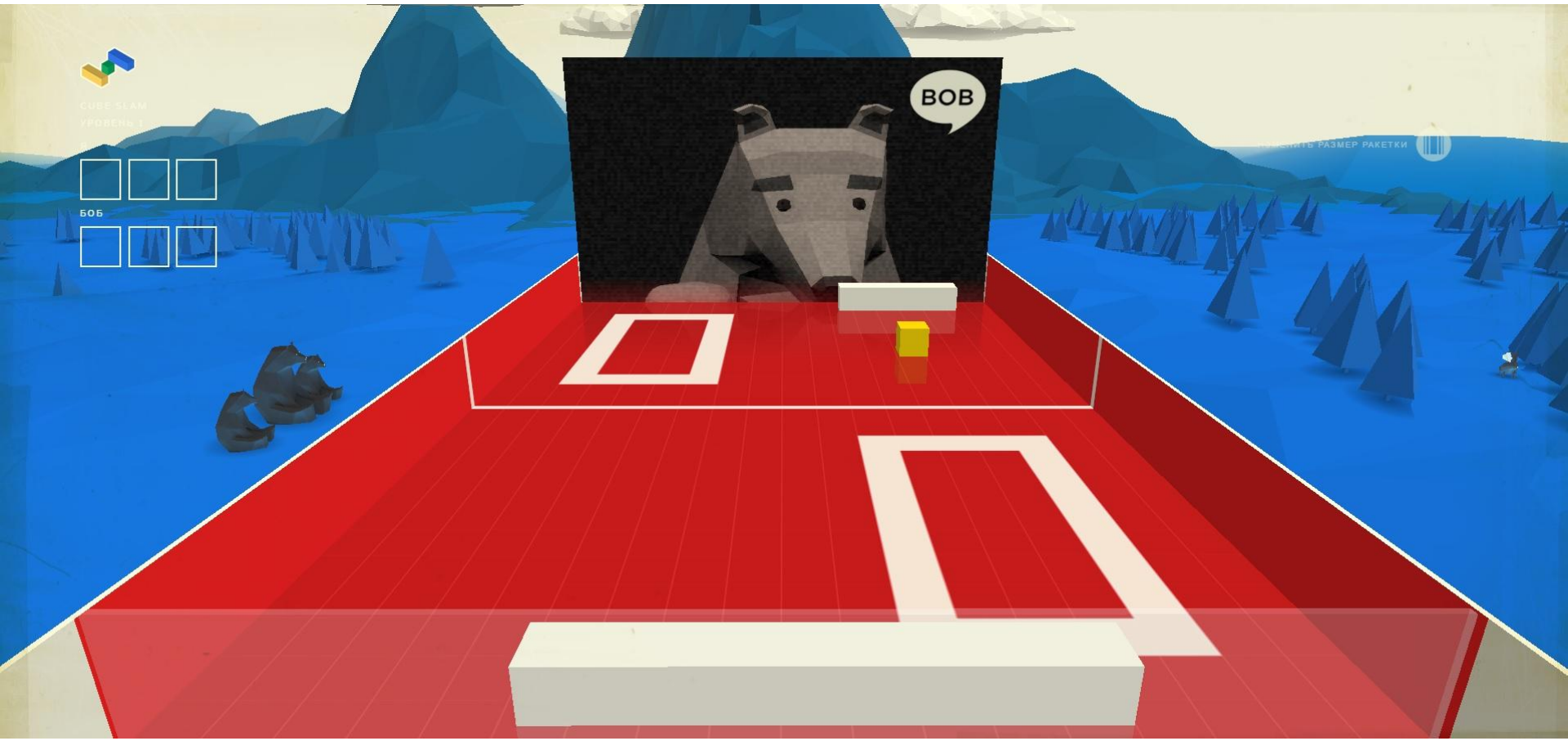
Необычная игра, где нужно гонять шарик по лабиринту из улиц Google Maps, доставляя метку в определенные места на карте. И все это расположено на кубе.



Примеры использования WebGL

- **Cube Slam**

Аэрохоккей с медведем.

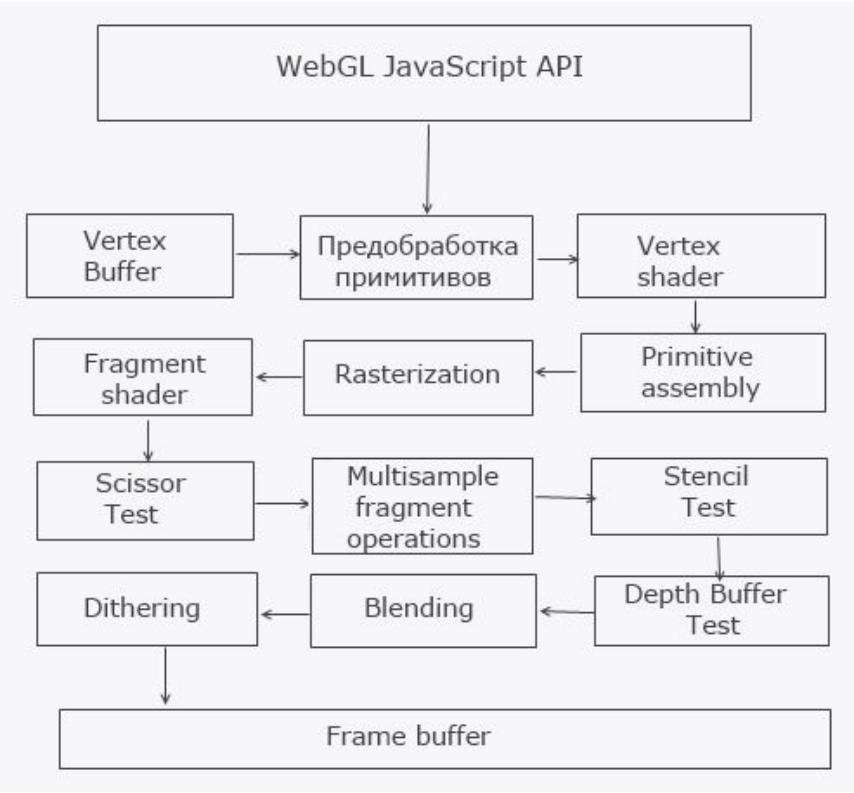


Примеры использования WebGL

- **Pearl Boy**

Реалистичная вода и закат с возможностью управления персонажем.

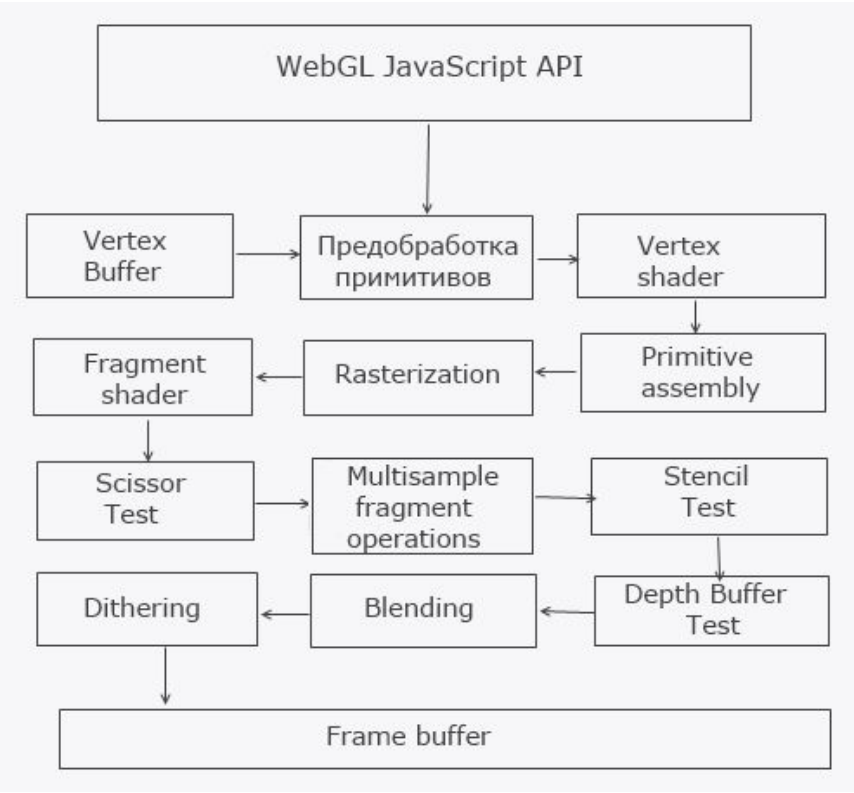
Конвейер WebGL



1. Вначале создается набор вершин в буфере вершин (Vertex Buffer). По этим вершинам впоследствии будут составлены геометрические примитивы, а из примитивов - объекты. Проводится некоторую предобработку.

2. Затем содержимое буфера вершин поступает на обработку в вершинный шейдер (Vertex Shader). Шейдер производит над вершинами некоторые трансформации, например, применяет матрицы преобразования и т.д. Шейдеры пишутся самим разработчиком, поэтому программист может применить различные преобразования по своему усмотрению.

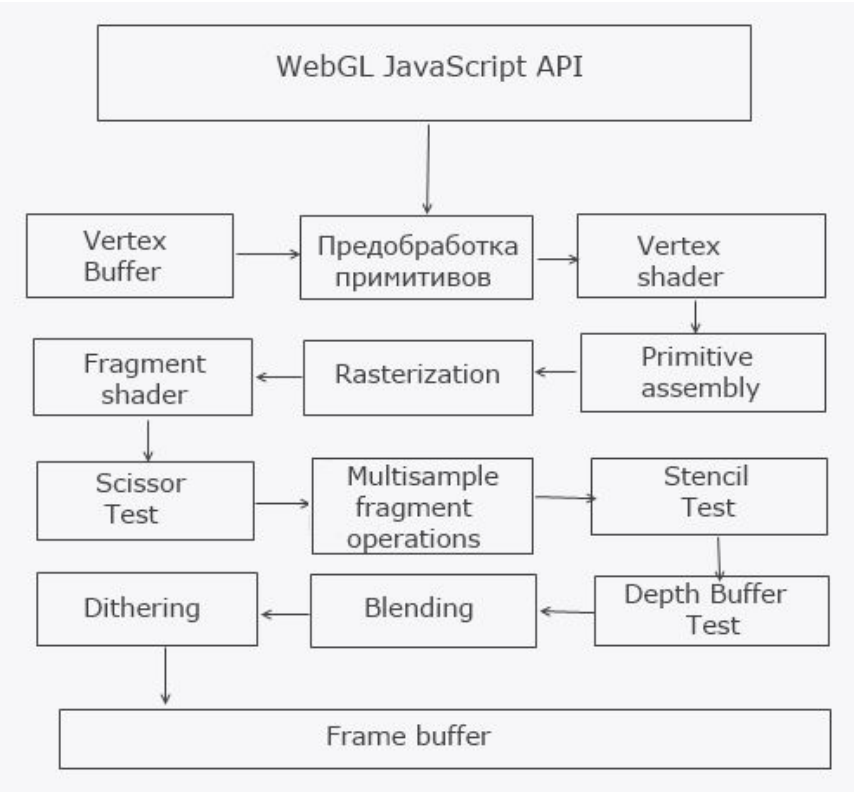
Конвейер WebGL



3. На следующем этапе (Primitive Assembly) конвейер получает результат вершинного шейдера и пытается измененные вершины сопоставить в отдельные примитивы - линии, треугольники, спрайты. Также на этом этапе определяется, входит ли примитив в видимое пространство. Если нет, то он обрезается. Оставшиеся примитивы передаются на следующий этап конвейера.

4. Далее на этапе растеризации (Rasterization) полученные примитивы преобразуются в фрагменты, которые можно представить как пиксели, которые затем будут отрисованы на экране.

Конвейер WebGL



5. И затем в дело вступает фрагментный шейдер (Fragment shader). (В технологиях Direct3D, XNA прямым аналогом является пиксельный шейдер). Фрагментный шейдер производит преобразования с цветовой составляющей примитивов, наполняет их цветом, точнее окрашивает пиксели, и в качестве вывода передает на следующий этап измененные фрагменты.

Следующий этап представляет собой ряд преобразований над полученными с фрагментного шейдера фрагментами. Собственно он состоит из нескольких подэтапов:

Конвейер WebGL

- Scissor Test: на этом этапе проверяется, находится ли фрагмент в пределах отсекающего прямоугольника. Если фрагмент находится в пределах этого прямоугольника, то он передается на следующий этап. Если же нет, то он отбрасывается и больше не принимает участия в обработке.
- Multisample Fragment Operations: на данном этапе у каждого фрагмента изменяются цветовые составляющие, производится сглаживание (anti-alising), чтобы объект выглядел более плавно на экране.
- Stencil Test: здесь фрагмент передается в буфер трафаретов (stencil buffer). Если вкратце, то в этом буфере дополнительно отбрасываются те фрагменты, которые не должны отображаться на экране. Как правило, данный буфер используется для создания различного рода эффектов, например, эффект теней.
- Depth Buffer Test - тест буфера глубины. В буфере глубины (depth buffer, а также называется, z-buffer) сравнивается z-компонента фрагмента, и если она больше значения в буфере глубины, то, следовательно, данный фрагмент расположен к смотрящему на трехмерную сцену ближе, чем предыдущий фрагмент, поэтому текущий фрагмент проходит тест. Если же z-компонента больше значения в буфере глубины, то, следовательно, данный фрагмент находится дальше, поэтому он не должен быть виден и отбрасывается.
- Blending: на данном этапе происходите небольшое смешение цветов, например, для создания прозрачных объектов.
- Dithering: здесь происходит смешение цветов, для создания тонов и полутонов.

6. Frame Buffer: и здесь наконец полученные после предобработки фрагменты превращаются в пиксели на экране.

Этапы программы

Данные этапы рисуют некоторый алгоритм действий. В реальности создание программы разбивается также на некоторые этапы:

- Создание и настройка шейдеров
- Создание и настройка буфера вершин, которые в последствии образуют геометрическую фигуру
- Отрисовка фигуры

```
<html>
<head>
<title>Triangle</title>
<meta charset="utf-8">
</head>
<body>
<canvas id="canvas3D" width="400" height="300">Ваш браузер не поддерживает элемент canvas</canvas>
<!-- фрагментный шейдер -->
<script id="shader-fs" type="x-shader/x-fragment">
  void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
  }
</script>
<!-- вершинный шейдер -->
<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;
  void main(void) {
    gl_Position = vec4(aVertexPosition, 1.0);
  }
</script>
<script type="text/javascript">
var gl;
var shaderProgram;
var vertexBuffer;
// установка шейдеров
function initShaders() {
    // получаем шейдеры
    var fragmentShader = getShader(gl.FRAGMENT_SHADER, 'shader-fs');
    var vertexShader = getShader(gl.VERTEX_SHADER, 'shader-vs');
    //создаем объект программы шейдеров
    shaderProgram = gl.createProgram();
    // прикрепляем к ней шейдеры
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    // связываем программу с контекстом webgl
    gl.linkProgram(shaderProgram);

    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert("Не удалсь установить шейдеры");
    }

    gl.useProgram(shaderProgram);
    // установка атрибута программы
    shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
    // подключаем атрибут для использования
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
}
```

```
// Функция создания шейдера по типу и id источника в структуре DOM
function getShader(type,id) {
    var source = document.getElementById(id).innerHTML;
    // создаем шейдер по типу
    var shader = gl.createShader(type);
    // установка источника шейдера
    gl.shaderSource(shader, source);
    // компилируем шейдер
    gl.compileShader(shader);

    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert("Ошибка компиляции шейдера: " + gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }
    return shader;
}
// установка буфера вершин
function initBuffers() {
 // установка буфера вершин
  vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // массив координат вершин объекта
  var triangleVertices = [
         0.0,  0.5,  0.0,
        -0.5, -0.5,  0.0,
         0.5, -0.5,  0.0
  ];
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertices), gl.STATIC_DRAW);
  // указываем кол-во точек
  vertexBuffer.itemSize = 3;
  vertexBuffer.numberOfItems = 3;
}
// отрисовка
function draw() {
    // установка области отрисовки
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);

    gl.clear(gl.COLOR_BUFFER_BIT);

    // указываем, что каждая вершина имеет по три координаты (x, y, z)
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                           vertexBuffer.itemSize, gl.FLOAT, false, 0, 0);
    // отрисовка примитивов - треугольников
    gl.drawArrays(gl.TRIANGLES, 0, vertexBuffer.numberOfItems);
}
```

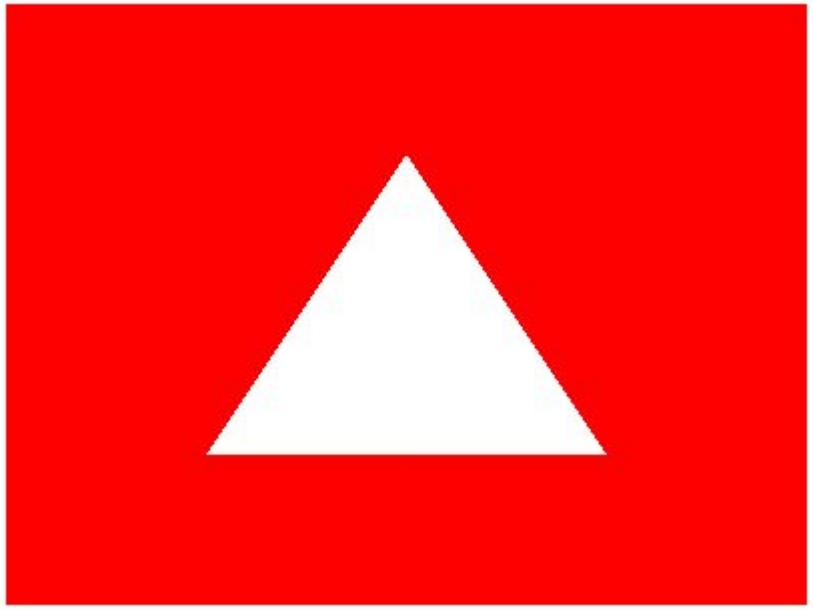
```
window.onload=function(){
    // получаем элемент canvas
    var canvas = document.getElementById("canvas3D");
    try {
        // Сначала пытаемся получить стандартный контекст WegGK.
        // Если не получится, обращаемся к экспериментальному контексту
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }
    catch(e) {}

    // Если контекст не удалось получить, выводим сообщение
      if (!gl) {
        alert("Ваш браузер не поддерживает WebGL");
      }
    if(gl){
        // установка размеров области рисования
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
        // установка шейдеров
        initShaders();
        // установка буфера вершин
        initBuffers();
        // покрасим в красный цвет фон
        gl.clearColor(1.0, 0.0, 0.0, 1.0);
        // отрисовка сцены
        draw();
    }
}
</script>
</body>
</html>
```

В данном примере, сначала срабатывают функция initShaders(), производящая инициализацию шейдеров и их настройку. Шейдеры являются обязательным звеном в конвейере WebGL, поэтому без них сложно будет построить программу.

Затем в дело вступает функция initBuffers(), устанавливающая буфер точек, по которым идет отрисовка.

И на финальном этапе происходит отрисовка в функции draw() - при помощи шейдеров буфер вершин превращается в геометрическую фигуру.

Результат



Реализация с помощью библиотек

- Пример: вращающийся куб.

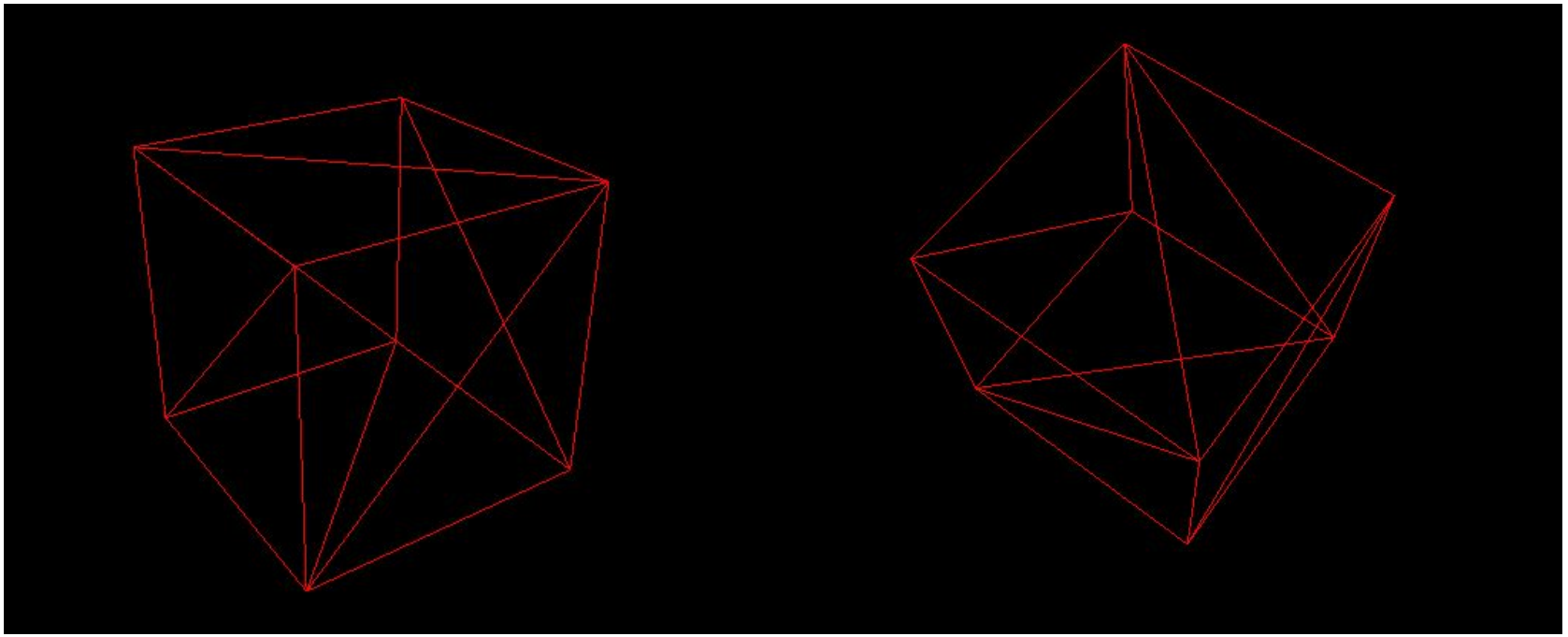
Потребуется специальная библиотека *Three.js (*легковесная кроссбраузерная библиотека JavaScript, используемая для создания и отображения анимированной компьютерной 3D графики при разработке веб-приложений). Эта библиотека не является необходимой для работы с WebGL, однако она упрощает работу.

- В одном каталоге с загруженной библиотекой создадим файл *index.html* со следующим содержанием:

```
<html>
<head>
<title>Coube</title>
<script src="three.min.js"></script>
<script type='text/javascript'>
window.onload=function(){
    var camera, scene, renderer;
    var geometry, material, mesh;

    init();
    animate();
    // инициализация начальных значений
    function init() {
```

```
    function init() {
        // создаем камеру - перспективная проекция
        camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 1, 1000);
        // установка z-координаты камеры
        camera.position.z = 600;
        // настройка сцены
        scene = new THREE.Scene();
        // настройка геометрии - в качестве геометрии будет куб
        // настроим его ширину, высоту и длину по оси z
        geometry = new THREE.CubeGeometry(200, 200, 200);
        // настройка материала - установка цвета
        material = new THREE.MeshBasicMaterial({ color: 0xff0000, wireframe: true});
        // настраиваем меш, который будет отображать куб
        mesh = new THREE.Mesh(geometry, material);
        scene.add(mesh);
        // создаем объект для рендеринга сцены
        renderer = new THREE.WebGLRenderer();
        // установка размеров
        renderer.setSize(window.innerWidth, window.innerHeight);
        // встраиваем в DOM-структуру страницы
        document.body.appendChild(renderer.domElement);
    }
    // функция анимации
    function animate() {

        requestAnimationFrame(animate);
        // вращение меша вокруг осей
        mesh.rotation.x += 0.01;
        mesh.rotation.y += 0.02;
        // рендеринг сцены - метод, производящий по сути отрисовку
        renderer.render(scene, camera);
    }
}
</script>
</head>
<body>
</body>
</html>
```

Что получилось?



Источники

- **Введение в WebGL** - http://metanit.com/web/webgl/1.1.php
- **Слабое место убийцы Flash** -http://internetno.net/category/analitika/slaboe_mesto_ubijcy_flash/
- **Впечатляющие примеры WebGL** - https://habrahabr.ru/post/190388/
- **WebGL** - https://developer.mozilla.org/ru/docs/Web/API/WebGL_API
- **WebGL** - https://ru.wikipedia.org/wiki/WebGL