WebGL

(WEB-BASED GRAPHICS LIBRARY)

Автор: Лапцевич Д.А.

Определение

Программная библиотека для языка JavaScript предназначенная для визуализации интерактивной трехмерной графики и двухмерной графики в пределах совместимости веб-браузера без использования плагинов.

Немного истории

WebGL основан на OpenGL ES 2.0, который, в свою очередь, является специальной версией для работы на мобильных устройствах. Спецификация WebGL была выпущена в 2011 году, разрабатывается и поддерживается некоммерческой организацией Kronos Group, сайт которой частенько лежит, что ещё более усложняет изучение. Известно, что в настоящее время идёт разработка спецификации версии 2.0.

Официальный сайт: https://www.khronos.org/webgl/

Области применения

- **Браузерные игры** — игровой процесс происходит непосредственно в окне браузера с полноценным 3D движком
- **Медицина**: все виды хирургии, стоматологии, кардиологии, косметологии и др. — доктор дистанционно может ознакомиться с моделью проблемной области пациента, отслеживать динамику развития, проводить виртуальные операции и др.
- **Презентация на сайте** — позволит пользователю подробно показать план объектов строительства, с возможностью совершения виртуального тура
- **Проектирование on-line** — предоставление одновременно нескольким людям возможности проектирования с применением трёхмерной графики

Начало работы

Если вы думаете, что WebGL рисует 3D, вы ошибаетесь. WebGL ничего не знает о 3D, это скорее низкоуровневый 2D API, и всё что он умеет делать, это рисовать треугольники. Но он умеет рисовать их очень много и очень быстро.

Хотите нарисовать квадрат? Пожалуйста, соедините два треугольника. Нужна линия? Без проблем, всего лишь несколько последовательно соединенных треугольников.

В отличие от OpenGL, в WebGL для отрисовки используются только шейдеры. Шейдеры никак не связаны, как вы могли бы подумать, с тенями или затенениями. Возможно, задумывались они именно для этого, но теперь используются для рисования всего и вся повсеместно.

Наверное, имеет смысл пояснить, что вообще собой представляют шейдеры…

Шейдеры

Шейдер — это программа, выполняемая на видеокарте и использующая язык GLSL(OpenGL Shading Language — язык высокого уровня для программирования шейдеров. Синтаксис языка базируется на языке программирования ANSI C, однако, из-за его специфической направленности, из него были исключены многие возможности, для упрощения языка и повышения производительности. В язык включены дополнительные функции и типы данных, например для работы с векторами и матрицами.)

Виды шейдеров

ВЕРШИННЫЙ

Заменяет часть графического конвейера, выполняющего преобразования, связанные с данными вершин. Такие как умножение вершин и нормалей на матрицу проекции и моделирования, установка цветов вершин, установка материалов освещения. Он работает для каждой отрисованной вершины. Обязательной работой для вершинного шейдера является запись позиции вершины, в встроенную переменную **gl_Position**.

ФРАГМЕНТНЫЙ

Заменяет часть графического ковейера (ГК), обрабатывая каждый полученный на предыдущих стадиях ГК фрагмент (не пиксель). Обработка может включать такие стадии, как получение данных из текстуры, просчет освещения, просчет смешивания. Обязательной работой для фрагментного шейдера является запись цвета фрагмента во встроенную переменную **gl_FragColor** или его отбрасывания специальной командой **discard**. В случае отбрасывания фрагмента, никакие расчеты дальше с ним производиться не будут, и фрагмент уже не попадет в буфер кадра.

Виды шейдеров (для чайнико

ВЕРШИННЫЙ

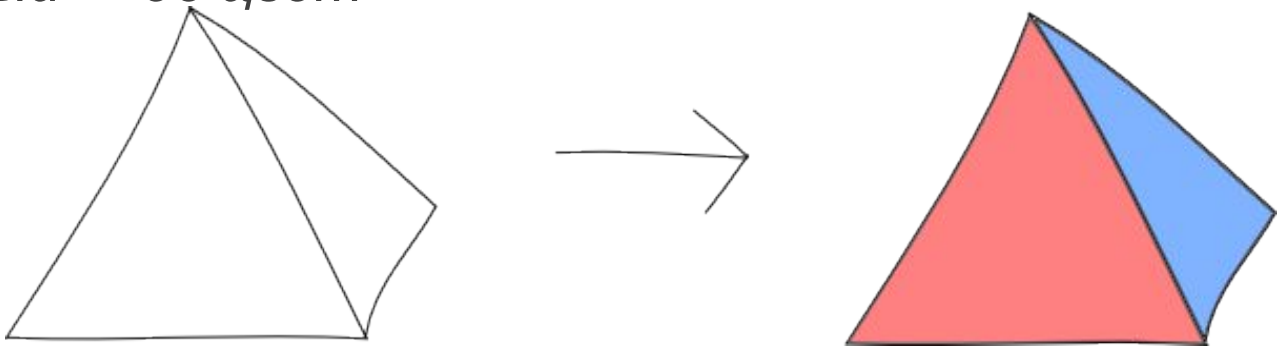
Предположим, что вы хотите нарисовать куб или любую другую фигуру со множеством вершин. Для этого вам нужно задать её геометрию, а геометрия в свою очередь задаётся с помощью указания координат вершин. Было бы накладно самим каждый раз вычислять новые координаты всех вершин при изменении положения куба в пространстве. Такую работу лучше переложить с процессора на видеокарту, для этого и существует вершинный шейдер.
В него передаются координаты вершин фигуры и положение локальной системы координат, в которой эти вершины заданы. Вершинный шейдер вызывается для каждой из вершин, он вычисляет их положение в глобальной системе координат и передаёт дальше для работы фрагментного шейдера.
Вершинный шейдер всегда вычисляет положение вершин, но попутно он может выполнять и другую работу, например, подсчёт угла падения света.

Виды шейдеров (для чайников)

ФРАГМЕНТНЫЙ

Знания положения фигуры недостаточно, чтобы её нарисовать. Необходима также информация о том, как должна быть раскрашена фигура, для этого служит фрагментный шейдер. Он вызывается для каждой точки поверхности фигуры и на основе переданной информации вычисляет цвет пикселя на экране.

Если вершинный шейдер определяет геометрию фигуры, то фрагментный — её цвет



Примеры шейдеров для треугольника

ВЕРШИННЫЙ

```
attribute vec3 a_position;

attribute vec3 a_color;

uniform vec3 u_position;

varying vec3 v_color;



void main(void) {

    v_color = a_color;

    gl_Position = vec4(u_position +
a_position, 1.0);}
```

ФРАГМЕНТНЫЙ

```
precision mediump float;

varying vec3 v_color;



void main(void) {

    gl_FragColor = vec4(v_color.rgb,
1.0);

}
```

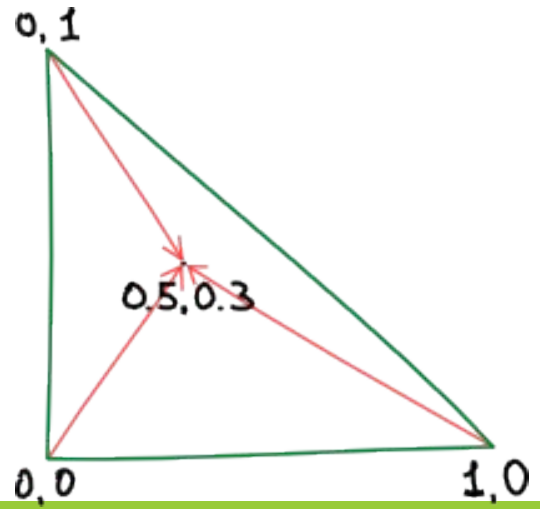
Код состоит из переменных и главной функции, возвращающей основной результат работы шейдера: gl_Position передаёт координаты, а gl_FragColor устанавливает цвет.

Шейдеры

Шейдеры имеют три типа переменных, которые передаются из основной программы:

1. ***attributes*** — доступны только в вершинном шейдере, разные для каждой из вершин;
2. ***uniforms*** — доступны в обоих шейдерах и одинаковы для всех вызовов шейдера;
3. ***varying*** — служат для передачи информации от вершинного шейдера к фрагментному.

При вызове фрагментного шейдера для конкретной точки, значения ***varying*** переменных линейно интерполируются между вершинами треугольника, которому принадлежит данная точка.



Начало написания приложений WebGL

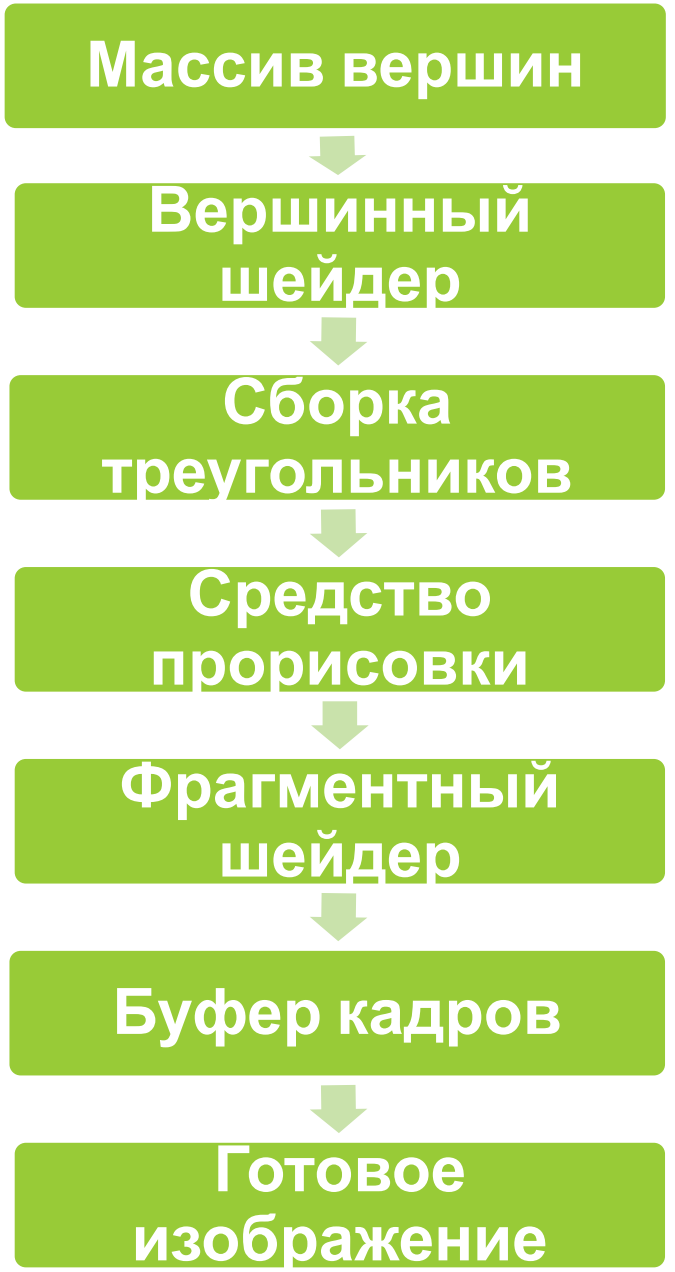
Приложение передает фигуру в GPU в виде массива векторов, который обычно представляет набор треугольников. Треугольники для GPU могут быть описаны:

- тремя вершинами (для отдельных треугольников);
- полосами треугольников: после первого треугольника для каждого последующего треугольника добавляется только одна дополнительная вершина.

Кроме того, вы можете описать для GPU линии, ломаные или точки. При передаче массива векторов в GPU следует указать, как его считывать: как отдельные треугольники, линии или полосы.

Конвейер отрисовки

- Приложение передает координаты в массиве векторов, который указывает на буфер векторов. Координаты векторов поочередно передаются в вершинный шейдер.
- Вершинный шейдер обрабатывает вершину относительно других вершин, перемещая координаты, добавляя ссылки на цвета и выполняя другие действия.
- Треугольники собираются и передаются в средство прорисовки, которое вычисляет пиксели, лежащие между вершинами треугольников, с помощью интерполяции.
- На этапе подробной проверки определяется, виден ли пиксель. Пиксели (и объекты) могут отсутствовать в области просмотра, находиться слишком далеко впереди или сзади (в зависимости от координат по оси Z) или закрываться другим объектом. Если они не видны, о них забудут.
- Шейдер фрагментов закрашивает пиксели. Цвета или ссылки на изображения могут быть переданы в шейдер фрагментов из вершинного шейдера. Задать цвета можно и в шейдере фрагментов.
- В завершение пиксель отправляется в буфер кадров, который отображает его на экране.

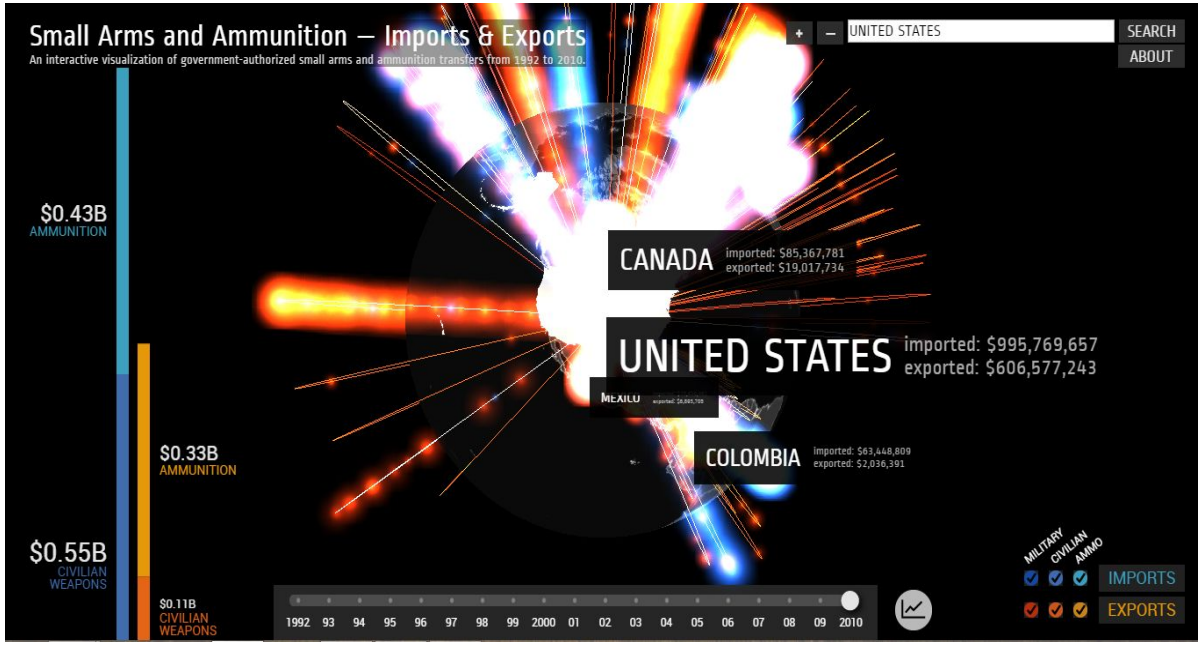


Примеры WebGL

Interactive Globe

Инфографика, показывающая динамику импорта и экспорта вооружения разных стран с 1982 по 2010 годы.

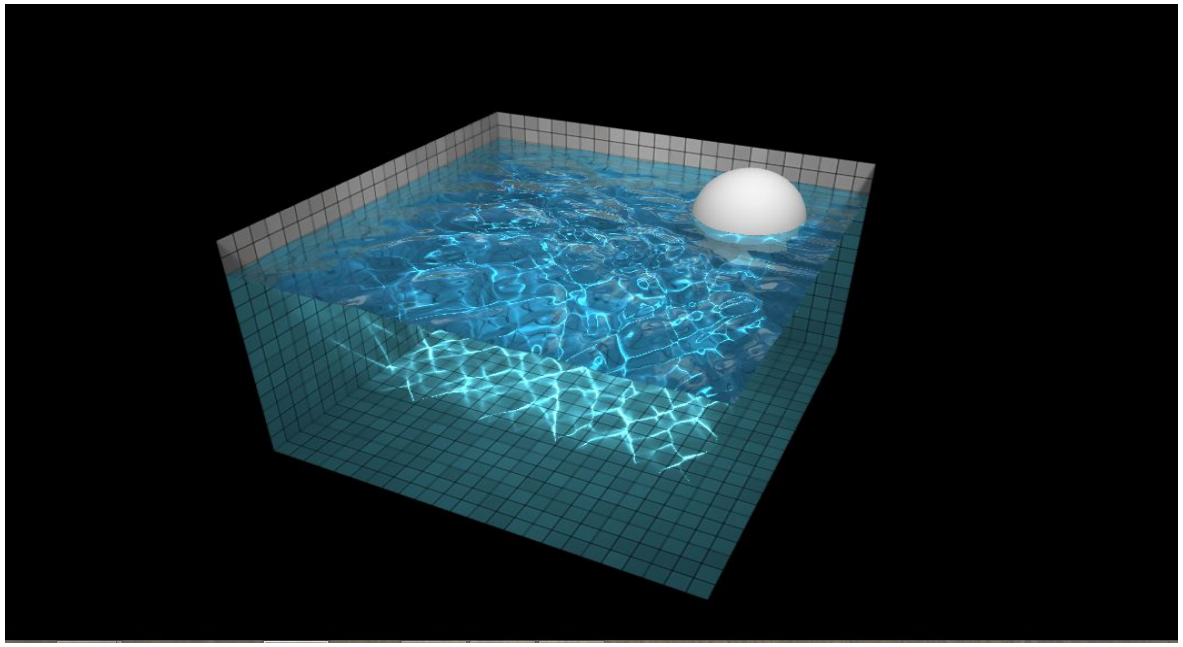
http://armsglobe.chromeexperiments.com/



WebGL Water

Удивительно реалистичный эффект воды, можно управлять шариком внутри бассейна, а также вращать всю сцену. Автор Evan Wallace.

http://madebyevan.com/webgl-water/



Преимущества и недостатки использования WebGL

Преимущества использования

- 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.
- Для создания приложений разработчики могут использовать стандартные для веб-среды технологии HTML/CSS/JavaScript и при этом также применять аппаратное ускорение графики.

Преимущества использования

- 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.

Преимущества использования

- 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

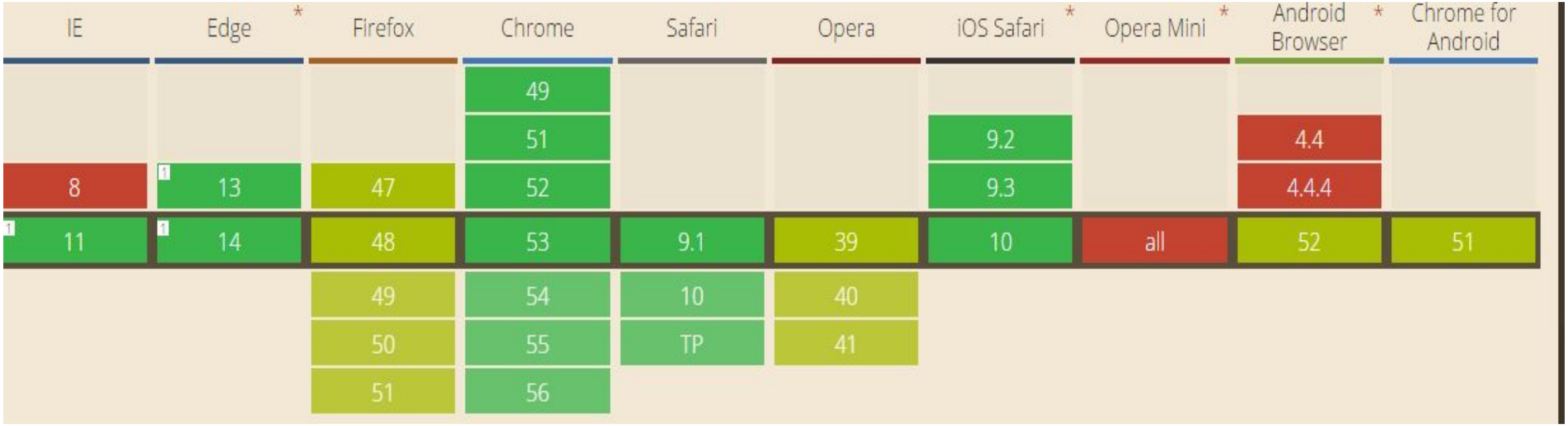
- Специалисты по вопросами безопасности британской компании Context выявили, что злоумышленник может на вредоносном сайте разместить код WebGL, который может быть напрямую передан в графический процессор, в результате чего компьютер просто выключится — тем самым можно организовать DoS атаку на компьютеры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

Недостатки использования

- 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.

- поддерживают не все браузеры (около 17% пользователей не имеют возможности использовать WebGL в своих браузерах)

Поддержка браузерами



IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49						
			51			9.2		4.4	
8	13 [1]	47	52			9.3		4.4.4	
11 [1]	14 [1]	48	53	9.1	39	10	all	52	51
		49	54	10	40				
		50	55	TP	41				
		51	56						

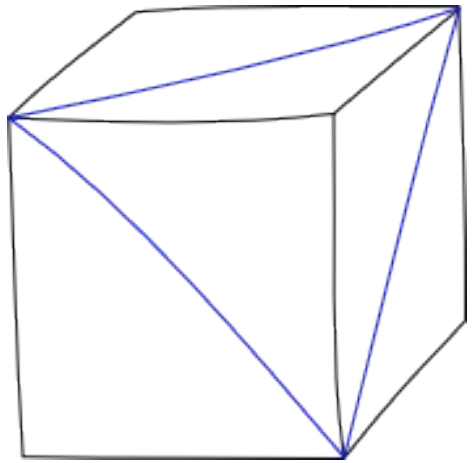
- не поддерживается
- поддерживается частично
- поддерживается полностью

Статистика поддержки WebGL разными браузерами взята с сайта caniuse.com

Как нарисовать объемную фигуру и заставить ее вращаться

Как нарисовать объемную фигуру и заставить ее вращаться

Положение плоской фигуры в пространстве задавалось с помощью вектора размерности три. Но фигура может не только менять положение, она может ещё вращаться и масштабироваться. Поэтому в трёхмерной графике используются не вектор положения, а матрица.



Известно, что матрица поворота в трёхмерном пространстве задаётся с помощью матрицы размером 3×3. К этой матрице добавляется вектор положения, таким образом, в итоге используется матрица 4×4.

WebGL никак не помогает нам работать с матрицами, поэтому, для удобства работы используется довольно известная библиотека glMatrix.

Как нарисовать объемную фигуру и заставить ее вращаться

За это преобразование отвечает матрица перспективы

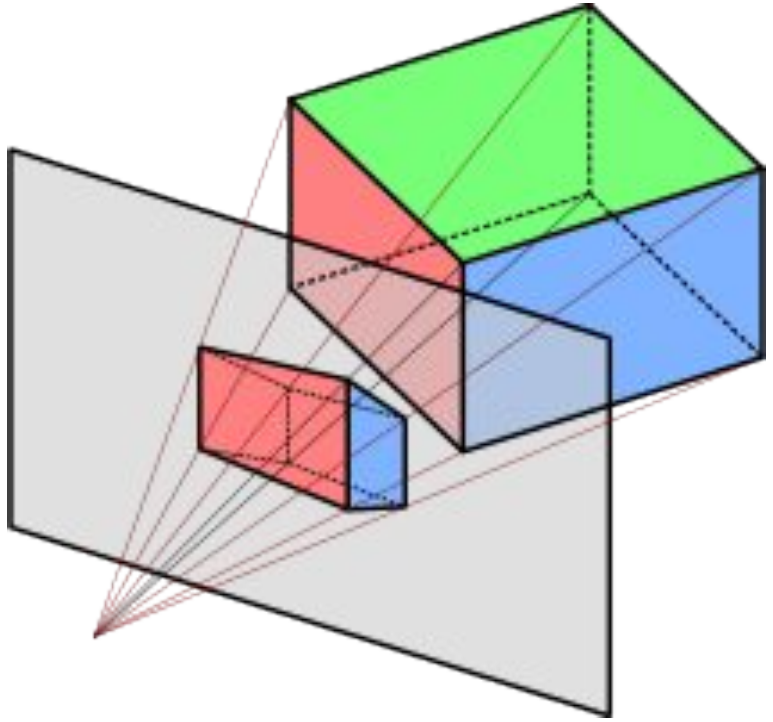
Перспективы

Чтобы спроецировать трёхмерный объект, нам нужно ввести понятие камеры.

Камера, как и любой объект, имеет своё положение в пространстве.

Она также определяет, какие объекты будут видны на экране, и отвечает за преобразование фигур так, чтобы на экране у нас создалась иллюзия 3D.

За это преобразование отвечает матрица перспективы



Перспектива куба на экране

Матрица перспективы

Метод `mat4.perspective(matrix, fov, aspect, near, far)` принимает пять параметров:

1. `matrix` — матрица, которую необходимо изменить;
2. `fov` — угол обзора в радианах;
3. `aspect` — соотношение сторон экрана;
4. `near` — минимальное расстояние до объектов, которые будут видны;
5. `far` — максимальное расстояние до объектов, которые будут видны.

Шейдеры для куба

ВЕРШИННЫЙ

```
attribute vec3 a_position;

attribute vec3 a_color;

uniform mat4 u_cube;//матрица положения

uniform mat4 u_camera;//матрица камеры

varying vec3 v_color;

void main(void) {

    v_color = a_color;

    gl_Position = vec4(u_position +
a_position, 1.0);}
```

В отличие от треугольника, в шейдерах для куба дополнительно используется матрица положения и матрица камеры

ФРАГМЕНТНЫЙ

```
precision mediump float;

varying vec3 v_color;



void main(void) {

    gl_FragColor = vec4(v_color.rgb,
1.0);

}
```

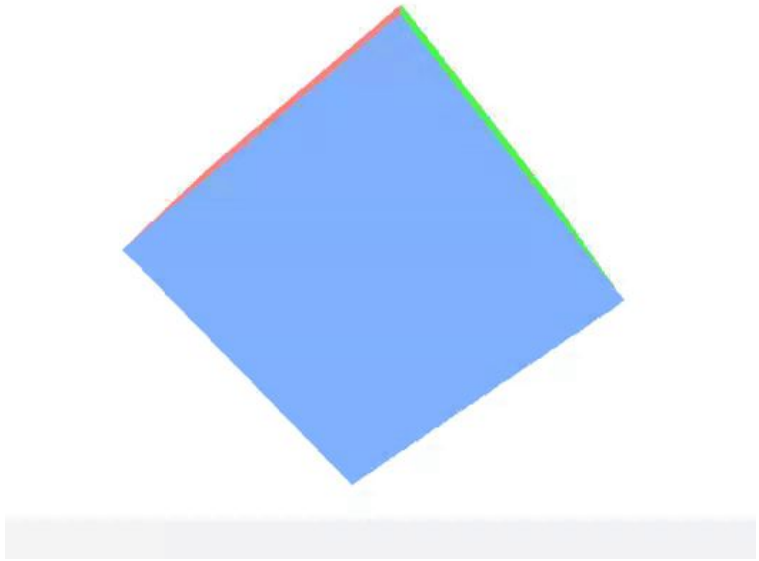
Фрагментный шейдер остается без изменений

Заставляем фигуру вращаться

Чтобы куб не стоял на месте, а вращался, необходимо постоянно менять его положение и обновлять кадр. Обновление происходит по средствам вызова встроенной функции `requestAnimationFrame`.

В отличие от других подобных методов, `requestAnimationFrame` вызывает переданную функцию только когда видеокарта свободна и готова к отрисовке следующего кадра.

Получаем вращающийся куб:



Отрисовка фигур в WebGL

gl.drawArrays()

Метод отрисовывает объекты последовательно по вершинам в буфере вершин. Данный метод принимает три параметра: `gl.drawArrays(mode, index, count)`

- **mode:** данный параметр указывает на отрисуемый примитив и может принимать следующие значения: gl.POINTS, gl.LINES, gl.LINE_LOOP, gl.LINE_STRIP, gl.TRIANGLES, gl.TRIANGLE_STRIP, gl.TRIANGLE_FAN.
- **index:** второй параметр указывает, какой номер вершины в буфере вершин будет первой для примитива.
- **count:** третий параметр указывает, сколько вершин будет использоваться для отрисовки.

gl.drawElements()

Если выше рассмотренный метод имеет дело с буфером вершин, то `gl.drawElements()` работает с буфером индексов. Он имеет следующую сигнатуру: `gl.drawElements(mode, count, type, offset)`:

- **mode**: режим, указывающий на тип примитива. В качестве примитивов используются те же, что и для метода `gl.drawArrays()`
- **count**: число элементов для отрисовки
- **type**: тип значений в буфере индексов. Может иметь значение `UNSIGNED_BYTE` или `UNSIGNED_SHORT`
- **offset**: смещение - с какого индекса будет проводиться отрисовка

Примитивы WebGL

В WebGL определены следующие виды примитивов:

- **gl.LINES** : набор линий, при этом все линии рисуются отдельно, если у двух линий определена общая точка, тогда визуально они соединяются, но фактически это две разные линии
- **gl.LINE_STRIP** : набор точек последовательно соединяются линиями, а незамкнутый контур, образуемый линиями, представляет единое целое
- **gl.LINE_LOOP** : то же самое, что и `gl.LINE_STRIP`, только последняя точка в наборе дополнительно еще соединяется с первой. Таким образом получается замкнутый контур
- **gl.TRIANGLES** : набор треугольников
- **gl.TRIANGLE_STRIP** : набор треугольников, при этом вершины последовательно соединяются в треугольники
- **gl.TRIANGLE_FAN** : набор треугольников, при этом для всех треугольников есть одна общая вершина в центре
- **gl.POINTS** : набор обычных точек, не соединенных между собой

Методы

- **`createProgram()`** – создание объекта программы шейдеров
- **`createShader(type)`** – создание шейдера по заданному типу type
- **`shaderSource(shader, source)`** – установка источника кода source для шейдера shader
- **`compileShader(shader)`** – компиляция шейдера shader в бинарный код для дальшейшего использования программой

Методы

- **`attachShader(program, shader)`** – добавление к объекту программы шейдера
- **`linkProgram(program)`** – связывание программы с контекстом webgl
- **`useProgram(program)`** - запуск программы для рендеринга

```
var program = gl.createProgram();

// Attach pre-existing shaders
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);

gl.linkProgram(program);
gl.useProgram(program);
```

Методы

- **`createBuffer()`** – создает буфер (хранилище) для данных, таких как вершины или цвета
- **`bindBuffer(target, buffer)`** – задаёт буферу «цель»: координаты вершин, координаты текстур, цвета, индексы элементов
- **`bufferData(data, size, usage)`**- добавление данных в созданный ранее буфер

```
vertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER,
vertexBuffer); 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);
```

Методы

- **`viewport(x, y, width, height)`** – установка области отрисовки
- **`clear(mask)`** – очистка буфферов заданной маской
- **`vertexAttribPointer(index, size, type, normalized, stride, offset)`** - устанавливаем атрибуты для каждой вершины: индекс атрибутов для использования, количество компонентов (1-4), тип компонентов, нормализация числе с плавающей запятой, шаг смещения, смещение

```
gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
gl.clear(gl.COLOR_BUFFER_BIT);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
vertexBuffer.itemSize, gl.FLOAT, false, 0, 0);
gl.drawArrays(gl.TRIANGLES, 0, vertexBuffer.numberOfItems);
```

Источники материалов

Статья «Начало работы с WebGL»:
https://msdn.microsoft.com/ru-ru/library/dn385807(v=vs.85).aspx
Официальный сайт WebGL:
https://www.khronos.org/webgl/
Статья на habrahabr «WebGL для всех» :
https://habrahabr.ru/company/2gis/blog/273735/
Примеры использования WebGL взяты из статей Давида Уолша(David Walsh):
https://davidwalsh.name/webgl-demos
https://davidwalsh.name/webgl-demo
А также с сайта WebGL Experiments:
https://www.chromeexperiments.com/webgl