МИНОБРНАУКИ РОССИИ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Санкт-Петербургский государственный электротехнический   
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Т. В. ГЕРАСИМОВА

Компьютерная графика:

лабораторный практикум.

# Лабораторная работа 1. Рисование геометрических объектов

Санкт-ПетербургИздательство СПбГЭТУ «ЛЭТИ»

2019

# Введение

OpenGL– это наиболее полная кроссплатформенная 3D библиотека. У нее долгая история формирования от поддержки графических ускорителей с фиксированными функциями до поддержки современного программируемого оборудования. Современный стиль OpenGL называется Core Profile. Упрощенный предшественник функций Core Profile OpenGLвдохновил OpenGL ES 2.0– безусловно, самый популярный низкоуровневый 3D API для мобильных устройств. OpenGL ES 2.0, в свою очередь, использовался в качестве основы для WebGL.

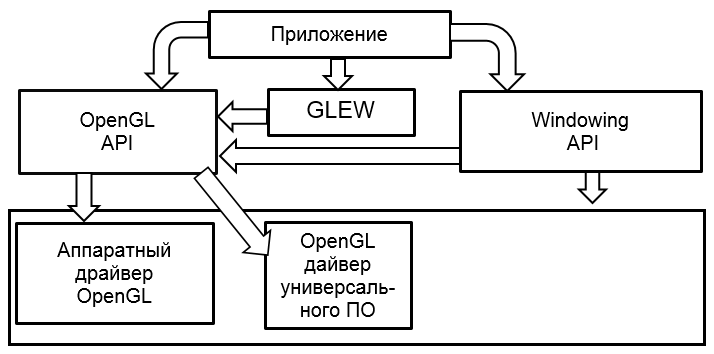
WebGL – это технология, которая позволяет рисовать, отображать и взаимодействовать со сложной интерактивной трехмерной компьютерной графикой («3D-графикой») из веб-браузеров. Традиционно 3D-графика была ограничена высококлассными компьютерами или выделенными игровыми консолями и требовала сложного программирования. Однако, поскольку персональные компьютеры и, что более важно, веб-браузеры стали более сложными, стало возможным создавать и отображать трехмерную графику, используя доступные и хорошо известные веб-технологии [1, 2].

Изучение WebGL даст возможность узнаете стиль графического программирования низкого уровня, который является общим для всех современных разновидностей OpenGL. Особенно, если вы придерживаетесь одного и того же языка программирования, переключение между версиями не является слишком сложным, и перенос кода может быть легким, если вы правильно сконфигурируете модуль

## 1.1. Общая архитектура OpenGL

Приступить написанию кода OpenGL, сложно, так как процесс отлича-ется от операционной системы к операционной системе. К счастью, сущест-вуют кроссплатформенные библиотеки, которые могут сделать ваш код чрезвычайно простым для переноса. На приведенной рис. 1.1. показан общий случай получения современной «базовой» среды OpenGL[3, 4]

* Модуль приложений разрабатывается программистом.
* Функции оконного API, специально разработаны для поддержки OpenGL, используются для настройки, выключения и обработки сигналов событий для области рисования OpenGL, называемой контекстом рендеринга (RC) в окне или компоненте. Эти функции могут быть определены операционной системой, набором сторонних виджетов или кроссплатформенной библиотекой OpenGL.

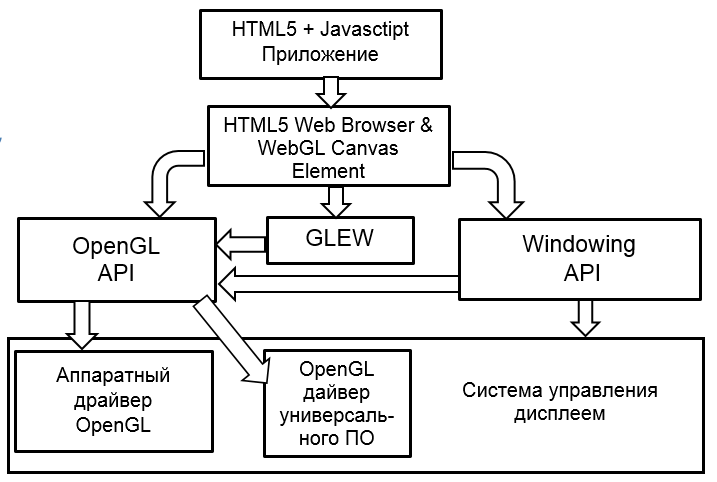
**

*Рис. 1.1 Блок-схема взаимодействия типичного приложения / OpenGL / OS.*

* Определенные ОС библиотеки: Windows: функции WGL, XWindows (Linux, большинство UNIX): функции GLX, Mac OS X: классы NSOpenGL.
* Полнофункциональные кроссплатформенные API и среды с поддержкой OpenGL: WxWidgets, FLTK, Tcl/Tk, Qt, HTML 5 (через холст WebGL) – это API для этого класса. Все, что вам нужно, встроено, включая автоматический выбор лучших драйверов для вашей платформы.
* Кроссплатформенные библиотеки OpenGL:SDL, GLFW, GLUT*.*
* ФункцииOpenGL API, определенные в gl.h и предоставляемые активным драйвером OpenGL (заданным RC), используются для рисования.
* GLEW – это популярная утилита, используемая для простого раскрытия новых возможностей OpenGL. Это очень важно для программ Windows, так как Microsoft перестала заботиться об OpenGL в версии 1.1 (текущая версия 4.5). Это также полезно для проверки возможностей и расширений OpenGL в других ОС [6, 7].
* Драйверы OpenGL реализуют детали функций OpenGL, передавая данные непосредственно на 3D-оборудование, выполняя вычисления на главном ЦП или какую-то их комбинацию.

## 1.2. архитектура приложения WEBGL

Когда вы пишете программу для WebGL, многие из деталей рис. 1.1. скрыты от вас. Фактически, вызовы WebGL могут даже не выполняться драйвером OpenGL. На Mac, где поддержка OpenGL встроена в ОС, вызовы WebGL переводятся прямо в их эквиваленты OpenGL. В Windows DirectX/D3D 9 или выше был встроен в ОС начиная с Vista, поэтому для перевода WebGL API используется специальный слой ANGLE**,** разработанный Google специально для WebGL и используемый в Chrome, Firefox, IE11 и Edge. команды в эквивалентные команды D3D9 или D3D11. В Linux поддержка драйверов чрезвычайно важна – большинство браузеров поддерживают только официальные драйверы nVidia.



*Рис.1.2 «Идеальное» взаимодействие между WebGL-приложением / OpenGL / OS.*

WebGL является 3D графической библиотекой, которая позволяет современным интернет-браузерам отрисовывать 3D-сцены стандартным и эффективным способом. WebGL имеет клиенто-ориентированный подход; элементы, которые составляют части 3D-сцены, обычно загружаются с сервера. Однако, вся дальнейшая обработка, необходимая для получения изображения выполняется локально, с помощью графического оборудования клиента [5].

По сравнению с другими технологиями (например, Java 3D, Flash и Unity Web Player Plugin) WebGL имеет ряд преимуществ:

JavaScript программирование: JavaScript это «родной» язык для веб-разработчиков и веб-браузеров. Работа с JavaScript позволяет получить доступ ко всем DOM-элементам, а также легко с ними обращаться, в отличие от общения с апплетами. Так как WebGL программируется в JavaScript, то это облегчает интеграцию WebGL-приложений с другими JavaScript – библиотеками, такими как JQuery и другими технологиями HTML5.

Автоматическое управление памятью: в отличие от OpenGL и других технологий, где есть конкретные операции выделения и освобождения памяти вручную, в WebGL нет такой необходимости. Из этого следует, что при выходе JavaScript переменной из области видимости, память, занимаемая ей, автоматически освобождается. Это чрезвычайно облегчает программирование, уменьшает объем кода, делает его более ясным и понятным.

Проницаемость: благодаря современным технологическим достижениям, веб-браузеры с поддержкой JavaScript устанавливаются на смартфоны и планшетные устройства.

Производительность: производительность приложений WebGL сопоставима с эквивалентными автономными приложениями (с некоторыми исключениями). Это происходит благодаря способности WebGL иметь доступ к локальным аппаратным ускорителям графики. До сих пор, многие веб-технологии для 3D рендеринга используют программный рендеринг.

Нулевая компиляция: учитывая, что WebGL написана на JavaScript, то нет необходимости в предварительной компиляции кода перед выполнением в веб-браузере. Это позволяет вносить изменения на лету и смотреть, как эти изменения влияют на 3D веб-приложение. Тем не менее, когда будем говорить о шейдерах, то мы поймем, что нуждаемся в некоторой компиляции. Однако, это происходит с помощью наших графических аппаратных средств, а не в нашем браузере.

Простота установки: поскольку WebGL интегрирован в HTML 5, дополнительная настройка не требуется. Чтобы написать приложение WebGL, все, что вам нужно, это текстовый редактор и веб-браузер [8].

## 1.3. Графический конвейер WebGL

Рендеринг – это процесс генерации изображения из модели с использованием компьютерных программ. В графике виртуальная сцена описывается с использованием такой информации, как геометрия, точка обзора, текстура, освещение и затенение, которая передается через программу рендеринга. Результатом этой программы рендеринга будет цифровое изображение [[4, 5].

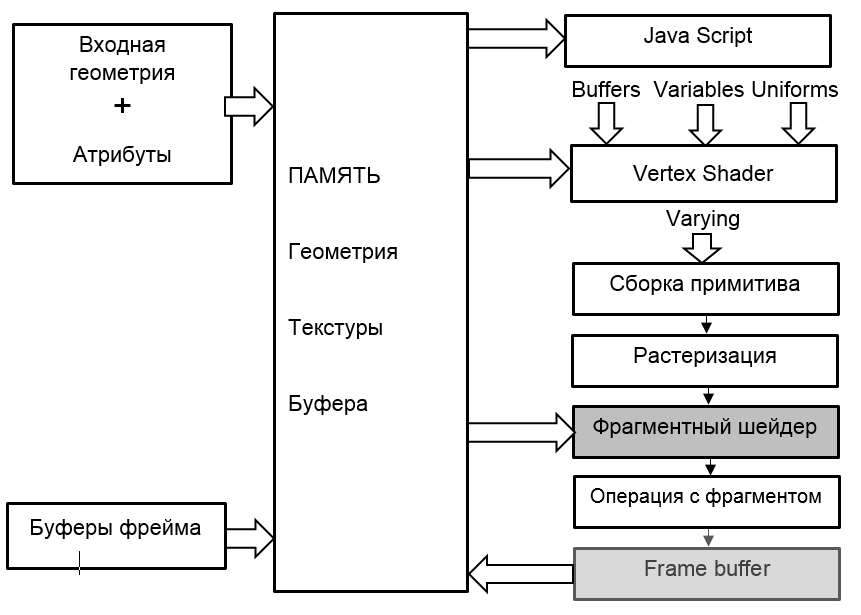
Для рендеринга 3D-графики мы должны выполнить последовательность шагов. Эти шаги известны как графический конвейер или конвейер рендеринга. На рис. 1.3. изображен графический конвейер WebGL.

### 1.3.1. JavaScript

При разработке приложений WebGL мы пишем код языка шейдеров(GLSL) для связи с графическим процессором. JavaScript используется для написания контрольного кода программы, который включает в себя следующие действия [9]:

- Инициализировать WebGL – JavaScript используется для инициализации

контекста WebGL.



*Рис. 1.3 графический конвейер WebGL*

- Создать массивы. Мы создаем массивы JavaScript для хранения данных геометрии.

- Буферные объекты – мы создаем буферные объекты (вершины и индексы), передавая массивы в качестве параметров.

- Шейдеры – мы создаем, компилируем и связываем шейдеры, используя JavaScript.

- Атрибуты – мы можем создавать атрибуты, включать их и связывать их с объектами буфера, используя JavaScript.

- Uniforms – мы также можем связать uniforms, используя JavaScript.

- Матрица преобразования – используя JavaScript, мы можем создать матрицу преобразования.

Сначала мы создаем данные для требуемой геометрии и передаем их шейдерам в виде буферов. Переменная атрибута языка шейдеров указывает на объекты буфера, которые передаются в качестве входных данных в вершинный шейдер [10, 11].

### 1.3.2. Вершинный шейдер

Когда мы запускаем процесс рендеринга, вызывая методы drawElements()

и drawArray(), вершинный шейдер выполняется для каждой вершины, предоставленной в объекте буфера вершин. Он вычисляет положение каждой вершины примитива многоугольник и сохраняет его в переменной gl\_position. Он также вычисляет другие атрибуты, такие как цвет, координаты текстуры и вершины, которые обычно связаны с вершиной.

### 1.3.3. Сборка примитивов

После вычисления положения и других деталей каждой вершины следующим этапом является этап сборки примитивов. Здесь треугольники собираются и передаются в растеризатор.

### 1.3.4. Растеризация

На этапе растеризации определяются пиксели в конечном изображении примитива. Здесь есть два шаги:

*Отбор* – вначале определяется ориентация (передняя или задняя сторона?) Многоугольника. Все те треугольники с неправильной ориентацией, которые не видны в области обзора, отбрасываются. Этот процесс называется выбраковкой.

*Отсечение* – если треугольник частично находится за пределами области просмотра, то часть вне области просмотра удаляется. Этот процесс известен как отсечение.

### 1.3.5. Фрагментный шейдер

Во фрагментный шейдер попадают следующие данные:

* данные из вершинного шейдера в varying переменных,
* примитивы из стадии растеризации, а затем
* вычисляет значения цвета для каждого пикселя между вершинами.

Фрагментный шейдер хранит значения цвета каждого пикселя в каждом фрагменте. Эти значения цвета могут быть доступны во время операций фрагмента, которые мы собираемся обсудить далее.

### 1.3.6. Фрагментные операции

Операции над фрагментами выполняются после определения цвета каждого пикселя в примитиве. Эти операции над фрагментами могут включать в себя следующее: глубина, смешение цветовых буферов, dithering/смещение.

Как только все фрагменты обработаны, 2D изображение формируется и отображается на экране. Буфер кадра является конечным пунктом назначения конвейера рендеринга.

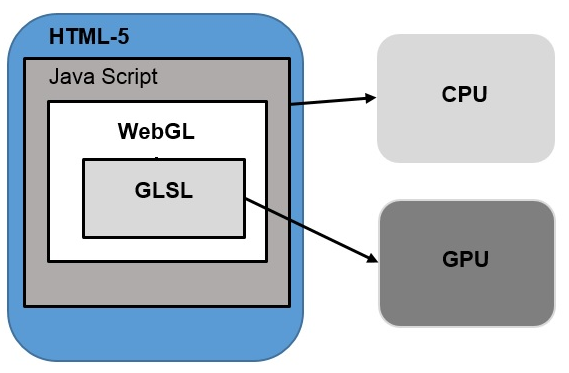
### 1.3.7. Кадровый буфер

Кадровый буфер – это часть графической памяти, в которой хранятся данные сцены. Этот буфер содержит такие детали, как ширина и высота поверхности (в пикселях), цвет каждого пикселя, а также буферы глубины и трафарета.

## 1.4. Структура приложения WebGL

Код приложения WebGL представляет собой сочетание JavaScript и языка шейдеров OpenGL. JavaScript необходим для связи с процессором. OpenGL Shader Language необходим для связи с графическим процессором.

Структура приложения показана на рис. 1.4.



*Рис.1.4 Структура WebGL приложения*

# Лабораторная работа 1. Рисование геометрических объектов

## 2.1. Процесс создания WebGL приложения

Чтобы написать приложение WebGL, первым делом нужно получить объект контекста рендеринга WebGL. Этот объект взаимодействует с буфером рисования WebGL и может вызывать все методы WebGL[8]. Следующие операции выполняются для получения контекста WebGL:

* Создать холст(canvas) HTML-5
* Получить идентификатор холста
* Получить WebGL

Шейдеры – это программы, которые работают на GPU. Шейдеры написаны на OpenGL ES Shader Language (известный как ES SL). ES SL имеет свои собственные переменные, типы данных, классификаторы, встроенные входы и выходы.

### 2.1.1. Создайте файл HTML

Файл HTML для приложения WebGL будет содержать ссылки на необ-

ходимые файлы JavaScript и ресурсы. Некоторые ресурсы, такие как шейдеры, могут быть определены в строке. Ниже приведен минимальный закомментированный HTML-файл для приложения WebGL [8].

Шаблон WebGL: HTML.

<!DOCTYPE html>

<html>

<head>

<title>WebGL Template</title>

<!-- Этот строенный скрипт является ресурсом вершинного шейдера. Шейдеры также могут быть связаны с внешним файлом. -->

<script id="vertex-shader" type="x-shader/x-vertex">

// Все вершинные шейдеры требуют ввода позиции или "атрибута".

// Имя можно изменить.

// Другие атрибуты могут быть добавлены.

attribute vec4 vPosition;

void main()

{

// gl\_Position это встроенный вывод вершинного шейдера или «varying».

// Его значение всегда должно быть установлено вершинным шейдером.

// Простейшие шейдеры просто копируют атрибут позиции прямо в

// gl\_Position

gl\_Position = vPosition;

}

</script>

<!-- Этот встроенный скрипт является ресурсом фрагментного шейдера.

Шейдеры также могут быть связаны с внешним файлом. ->

<script id="fragment-shader" type="x-shader/x-fragment">

// Устанавливает точность по умолчанию для чисел с плавающей точкой.

// Обязательно, поскольку фрагментные шейдеры не имеют точности по умолчанию.

// Варианты lowp и mediump.

precision mediump float;

void main()

{

// gl\_FragColor - это встроенный фрагментный шейдер

// В общем, это должно быть установлено, но не обязательно.

// по умолчанию gl\_FragColor не определено

gl\_FragColor = vec4(0,0,0,1);

}

</script>

<!-- это внешние файлы javascript.

Первые три библиотеки являются учебниками.

Последний из них - ваш собственный код JavaScript. Обязательно измените имя,

чтобы оно соответствовало вашему файлу javascript. ->

<script type="text/javascript" src="../Common/webgl-utils.js"></script>

<script type="text/javascript" src="../Common/initShaders.js"></script>

<script type="text/javascript" src="../Common/MVnew.js"></script>

<script type="text/javascript" src="**yourWebGLJavascript.js**"></script>

</head>

<body>

<! - Это холст - единственный элемент HTML, который может отображать

графику WebGL. На веб-странице может быть несколько холстов WebGL, но

это Это сложно. Придерживайтесь одного на странице на данный момент. ->

<canvas id="gl-canvas" width="512" height="512">

</canvas>

</body>

</html>

### 2.1.2. Создание файла Javascript

Файл javascript вдохнет жизнь в ваше приложение WebGL. Он устанавливает контекст рендеринга WebGL, рисует и определяет ответы на различные события. Простейшая javascript-программа WebGL настроит контекст рендеринга после загрузки HTML, определив действие для события window.onload [9]. Он также может рисовать, если анимация не нужна.

Следующий javascript определяет очень минимальный шаблон программы WebGL:

var gl;

window.onload = function init() {

// Настройка контекста рендеринга WebGL в холсте HTML5

var canvas = document.getElementById("gl-canvas");

gl = WebGLUtils.setupWebGL(canvas);

if (!gl) {

alert("WebGL недоступен");

}

// Настроить WebGL

//, например,. - установить чистый цвет

// - включить проверку глубины

// Загрузить шейдеры и инициализировать буферы атрибутов

var program = initShaders(gl, "vertex-shader", "fragment-shader");

gl.useProgram(program);

// Настройка данных для рисования

// Загрузка данных в буферы данных GPU

// Связать атрибуты шейдера с соответствующими буферами данных

// Получить адреса униформы шейдера

// Либо отрисовать как часть инициализации

// render ();

// Или рисуем непосредственно перед следующим событием перерисовки

// requestAnimFrame (render ());

};

function render() {

// очищаем экран

} // рисуем

### 2.1.3. Шейдеры

Шейдер представляет собой часть шейдерной программы, которая заме-няет собой часть графического конвейера видеокарты. Тип шейдера зависит от того, какая часть конвейера будет заменена. Каждый шейдер должен вы-полнить свою обязательную работу, т. е. записать какие-то данные и передать их дальше по графическому конвейеру [11, 12].

В языке GLSL содержится обширный набор типов, включая вектор и матрицу, которые позволяют сделать код шейдерных программ для обычных 3D-графических операций более лаконичным. Существует в нем и специальный набор типов, каждый из которых определяет уникальную форму данных ввода или вывода для шейдеров. В языке есть заимствованные из C++ механизмы: перегрузка функций по типу аргумента и объявление переменных непосредственно перед использованием вместо начала блока. Язык поддерживает циклы, подпрограммы и условные выражения. Богатый набор встроенных функций обеспечивает много возможностей, необходимых для создания шейдерных алгоритмов. О языке шейдеров GLSL можно сказать следующее:

‒ высокоуровневый процедурный язык;

‒ с небольшими изменениями используется для вершинных и фрагментных шейдеров;

‒ базируется на синтаксисе и управлении C и С++;

‒ изначально в нем поддерживаются векторные и матричные операции, так как они являются неотъемлемой частью многих графических алгоритмов;

‒ более жестко проверяет типы, чем C и C++, и функции могут вызы-ваться по возвращаемому значению;

‒ использует квалификаторы типов чаще, чем управление вводом-выво-дом;

‒ у него нет ни ограничений на длину шейдера, ни необходимости ее запрашивать.

Шейдерная программа – это небольшая программа, состоящая из шейдеров (вершинного и фрагментного, возможны и др.) и выполняющаяся (Graphics Processing Unit), т. е. на графическом процессоре видео-карты [12].

Существует пять мест в графическом конвейере (см. рис.1.4), куда могут быть встроены шейдеры. Соответственно шейдеры делятся на типы:

‒ вершинный шейдер (vertex shader);

‒ геометрический шейдер (geometric shader);

‒ фрагментный шейдер (fragment shader);

Дополнительно существуют вычислительные (compute) шейдеры, кото-рые выполняются независимо от графического конвейера.

Разные шаги графического конвейера накладывают разные ограничения на работу шейдеров. Поэтому у каждого типа шейдеров есть своя специфика.

Геометрический и тессяляционные шейдеры не являются обязательны-ми. Современный OpenGL требует наличия только вершинного и фрагмент-ного шейдера. Хотя существует сценарий, при котором фрагментный шейдер может отсутствовать [13] ‒ [15].

## 2.2. Вершины и примитивы

Основные примитивы рендеринга WebGL состоят из списков вершин.

Данные вершин могут быть двух, трех или четырехмерными. Иногда требует-

ся дополнительное измерение, чтобы правильно перемещать вершины в пространстве. Данные вершин чаще всего представлены в шейдерах с типами данных vec2, vec3 и vec4. Это 2, 3 и 4-компонентные структуры с плавающей точкой. Эти данные должны быть загружены из массивов JavaScript 32-битного типа с плавающей запятой. Например, следующий двумерный массив дает двухмерные координаты для трех вершин в треугольнике:

// Треугольные позиции

var points = new Float32Array

([ 0.9, 0.9, 0.9, 0.0, 0.0, 0.9 ]);

Количество координат, предоставленных для каждой вершины, должно соответствовать типу vec, указанному на входе положения используемого вами шейдера.

### 2.2.1. Загрузка данных вершин в буферы

Если у вас есть данные вершин, вам нужно загрузить их в буферы. Каждый массив может быть загружен в отдельный буфер, или все массивы могут быть упакованы в один и тот же буфер.

### 2.2.2. Присоединение буферов к шейдерным программам

Как только буфер загружен данными, он должен быть подключен к правильному входу в вашей шейдерной программе. Для этого вы запрашиваете ввод по имени, включаете его, а затем присоединяете свои данные в текущем связанном буфере к входу с описанием того, как данные форматируются.

### 2.2.3. Точки

Можно нарисовать только один тип точки:

POINTS – рисует точку для каждой вершины.

Вы можете контролировать размер точек, задав значение встроенного вывода вершинного шейдера gl\_PointSize. Хотя для спецификации WebGL требуются только точки размера 1, почти все реализации WebGL допускают гораздо более широкий диапазон, поскольку текстурированные точки составляют основу многих интересных эффектов.

### 2.2.4. Линии

Можно создать три различных линейных примитива:

LINES – рисует отрезок линии для каждой пары вершин.

LINE\_STRIP – рисует связанную группу отрезков линии от вершины **v0** до **vn,** соединяющую линию между каждой вершиной и следующей в указанном порядке.

LINE\_LOOP – аналогично LINE\_STRIP, за исключением того, что закрывает строку от vn до v0, определяя цикл.

Некоторые реализации WebGL позволяют вам контролировать ширину линий с помощью lineWidth(). Большинство компьютеров Mac поддерживают минимальный диапазон ширины линий – от 1,0 до 1,0. Вы можете обнаружить, что ваш компьютер позволяет больше.

### 2.2.4. Треугольники

* Лицевые грани, задние грани и режимы рендеринга:
* [cullFace()](http://www.khronos.org/opengles/sdk/2.0/docs/man/xhtml/glCullFace.xml) – если CULL\_FACE включен, это указывает, какие стороны треугольника отбрасывать, FRONT, BACK или FRONT\_AND\_BACK.
* [frontFace()](http://www.khronos.org/opengles/sdk/2.0/docs/man/xhtml/glFrontFace.xml) – указать , по часовой стрелке, то ли CW или против часовой стрелки, против часовой стрелки для того точка рисунка означает треугольник перед фронтом.
* Типы треугольников
* TRIANGLE – рисует ряд отдельных трехсторонних многоугольников
* TRIANGLE\_STRIP – рисует полосу соединенных треугольников. Первые три вершины определяют полный треугольник. Каждая последующая вершина завершает треугольник с двумя предыдущими. Порядок отрисовки первых двух точек менял местами каждый треугольник, чтобы помочь поддерживать порядок отрисовки по часовой стрелке или против часовой стрелки, т.е.: вершины 0-1-2-3-4-5-6 нарисованы так: 0-1-2, 2-1-3, 2-3-4, 4-3-5, 4-5-6
* TRIANGLE\_FAN – рисует полоску треугольников, соединенных общим источником. Первые три вершины определяют полный треугольник. Каждая последующая вершина завершает треугольник с предыдущей и первой вершиной.

//

var points var =

[ vec2 (0,0, 0,0), vec2 (0,5, 0,0), vec2 (0,5, 0,5), vec2 (-0,5, 0,5), vec2 (-1,0, 0,0),

vec2 (-0,5, - 0,5]

### 2.2.5. Цвет вершины

До сих пор наш шейдер использовал жестко закодированный цвет. Вы можете изменить этот цвет в работающей программе одним из двух способов: униформой и атрибутами. Все цвета будут в формате RGBA – красный, зеленый, синий, альфа. Альфа – это дополнительный термин, используемый в операциях смешивания. Вы можете думать об этом как о «прозрачности» [16].

### 2.2.6. Vertex Color Arrays

Цветовые массивы работают так же, как массивы позиций вершин. Вам нужно будет настроить второй вход массива для вашего вершинного шейдера, создать массив цветов, загрузить его в буфер и прикрепить к шейдеру.

Следующий код определяет входной атрибут с именем vColor. Это похоже на код, используемый для vPosition. Вы должны назначить значение в vColor для вывода цвета:

attribute vec4 vColor; // Ввод цвета вершины

## 2.3. Цель, требования и рекомендации к выполнению задания

Цель выполнения задания: ознакомление с основными примитивами

WebGL. Требования и рекомендации к выполнению задания:

– проанализировать полученное задание, выделить информационные объекты и действия;

– разработать программу с использованием требуемых примитивов и атрибутов.

## 2.4. Задания

Получите удобное рисование с буферами вершин, униформой и шейдерами:

* рисование нескольких вещей с отдельными командами рисования
* используя разные примитивы
* изменение размеров линий и точек по умолчанию
* изменение цвета на лету

## 2.5. Пример выполнения задания

Ранее мы обсуждали, как следовать пошаговому процессу, чтобы нарисовать примитив. Мы объяснили процесс в пять шагов. Вам нужно повторять эти шаги каждый раз, когда вы рисуете новую форму. Объясним, как рисовать точки с трехмерными координатами в WebGL. Прежде чем двигаться дальше, давайте взглянем на пять шагов.

### 2.5.1. Требуемые шаги

Для создания приложения WebGL для рисования точек необходимы следующие шаги.

Шаг 1. Подготовьте холст и получите контекст рендеринга WebGL

На этом этапе мы получаем объект контекста рендеринга WebGL, используя метод getContext ().

Шаг 2. Определите геометрию и сохраните ее в буфере объектов

Поскольку мы рисуем три точки, мы определяем три вершины с трехмерными координатами и сохраняем их в буферах.

var vertices = [ -0.5,0.5,0.0, 0.0,0.5,0.0, -0.25,0.25,0.0, ];

Шаг 3. Создание и компиляция шейдерных программ

На этом этапе вам нужно написать программы вершинного шейдера и фрагментного шейдера, скомпилировать их и создать объединенную программу, связав эти две программы.

Вершинный шейдер – В вершинном шейдере данного примера мы определяем векторный атрибут для хранения трехмерных координат и присваиваем его переменной gl\_position.

gl\_pointsize – это переменная, используемая для назначения размера точке. Назначаем размер точки как 10.

var vertCode = 'attribute vec3 coordinates;

void main(void) { gl\_Position = vec4(coordinates, 1.0); 'gl\_PointSize = 10.0; '}';

Шаг 4 – Связать шейдерные программы для буферизации объектов

На этом этапе мы связываем объекты буфера с программой шейдера.

Шаг 5 – Рисование необходимого объекта

Мы используем метод drawArray() для рисования точек. Поскольку количество точек, которые мы хотим нарисовать, равно трем, значение счетчика равно 3.

gl.drawArrays(gl.POINTS, 0, 3)

Пример – нарисуйте три точки, используя WebGL

Вот полная программа WebGL, чтобы нарисовать три точки –

<!doctype html>

<html>

<body>

<canvas width = "570" height = "570" id = "my\_Canvas"></canvas> <script>

/\*================ Создание холста =================\*/

var canvas = document.getElementById('my\_Canvas');

gl = canvas.getContext('experimental-webgl');

/\*========== Определение и хранение геометрии =======\*/

var vertices = [

-0.5,0.5,0.0, 0.0,0.5,0.0, -0.25,0.25,0.0, ];

// Создать пустой буферный объект для хранения буфера вершин

var vertex\_buffer = gl.createBuffer();

// Привязать к нему соответствующий буфер массива

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

// Привязать к нему соответствующий массива буфер

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices),

gl.STATIC\_ DRAW);

gl.bindBuffer(gl.ARRAY\_BUFFER, null); // Убрать буфер

/\*=====================Shaders========================\*/

// исходный код вершинного шейдера

var vertCode ='attribute vec3 coordinates;' +'void main(void)

{' +' gl\_Position = vec4(coordinates, 1.0);' + 'gl\_PointSize = 10.0;'+ '}'

// Создать объект вершинного шейдера

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

// Прикрепить исходный код вершинного шейдера

gl.shaderSource(vertShader, vertCode);

gl.compileShader(vertShader); // Скомпилируйте вершинный шейдер

// исходный код фрагментного шейдера

var fragCode = 'void main(void) {' +' gl\_FragColor = vec4(0.0, 0.0, 0.0, 0.1);' +'}';

// Создать фрагмент шейдерного объекта

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

// Прикрепить фрагмент исходного кода шейдера

gl.shaderSource(fragShader, fragCode);

gl.compileShader(fragShader); // Скомпилируйте фрагментный шейдер

// Создать объект программы шейдера для

// хранения комбинированной программы шейдера

var shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertShader); // Присоедините вершинный шейдер

gl.attachShader(shaderProgram, fragShader); // Прикрепить фрагмент шейдера

gl.linkProgram(shaderProgram); // связать обе программы

// Используйте комбинированный шейдерный программный объект

gl.useProgram(shaderProgram);

/\*======== Связывание шейдеров для буферизации объектов ========\*/

// Привязать объект буфера вершин

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

// Получить атрибут местоположения

var coord = gl.getAttribLocation(shaderProgram, "coordinates");

// Укажите атрибут для текущей привязки VBO

gl.vertexAttribPointer(coord, 3, gl.FLOAT, false, 0, 0);

gl.enableVertexAttribArray(coord); // Включить атрибут

/\*============= Рисование примитива ===============\*/

gl.clearColor(0.5, 0.5, 0.5, 0.9); // Очистить холст

gl.enable(gl.DEPTH\_TEST); // Включить тест глубины

// Очистить бит буфера цвета

gl.clear(gl.COLOR\_BUFFER\_BIT);

// Установите порт просмотра

gl.viewport(0,0,canvas.width,canvas.height);

// Нарисуй треугольник

gl.drawArrays(gl.POINTS, 0, 3);

</script> </body> </html>

# Литература

1. Diego Cantor, Brandon Jones. WebGL Beginner's Guide. Paperback – June 15, 2012
2. Мацуда Коичи, Ли Роджер. WebGL. Программирование трехмерной графики. ДМК-Пресс, 2015
3. Anyuru Andreas. Professional WebGL Programming: Developing 3D Graphics for the Web.
4. https://webgl2fundamentals.org. Дата обращения 02.01.2019 г.
5. Трёхмерная инфографика в браузере — Chrome Experiments WebGL http://infographer.ru/3d-infographics-in-browser-chrome. Дата обращения 12.02.2019 г.
6. Эйнджел Э. Интерактивная компьютерная графика. Вводный курс на базе OpenGL. 2-е изд. пер. с англ. ‒ M.: Вильямc, 2001.
7. Дэн Гинсбург,Будирижанто Пурномо. OpenGL ES 3.0. Programming Guide Руководство разработчика, Издательство «ДМК Пресс» . 2015
8. Parisi Tony. Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages.1st Edition. O’Reilly, 2011.
9. Браун Этан. Изучаем JavaScript: руководство по созданию современных веб-сайтов, 3-е издание
10. http://www.gamedev.ru/code/tip/catmull\_rom. Дата обращения 22.01.2019 г.
11. John Kessenich. The OpenGL Shading Language. 2006 https://www.khronos.org/registry/ OpenGL/specs/gl/GLSLangSpec.1.20.pdf  
    Дата обращения 19.02.2019 г.
12. Randi J. Rost. OpenGL Shading Language. 2nd Edition. Published by Addison-Wesley, 2006.
13. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов. пер. с англ. ‒ M.: ДМК Пресс, 2016.
14. Рост Р. Дж. OpenGL. Трёхмерная графика и язык программирования шейдеров.- СПб.: БХВ – Петербург, 2006.
15. Боресков А.В. Разработка и отладка шейдеров. СПб.: БХВ – Петербург, 2006.
16. Методы повышения реалистичности изображений: метод. указания к лаб. работам/ Санкт-Петербургский государственный электротехнический университет им. В.И. Ульянова (Ленина) "ЛЭТИ" ; [сост. Т.В. Герасимова]. - СПб. : Изд-во СПбГЭТУ "ЛЭТИ", 2008. 32 с.
17. ПоревВ. Н*.* Компьютерная графика*.* ‒ СПб*.*: БХВ-Петербург, 2002*.’*
18. Джеф Проузис*.* Как работает компьютерная графика*. ‒* СПб.: Питер, 2008. - 654 с
19. Герасимова Т.В. Использование открытой библиотеки Open GL 4.0 для разработки графических приложений.: учебное пособие. СПб.; Изд-во СПбГЭТУ «ЛЭТИ», 2011. 63 с.
20. Т.В. Герасимова. Повышение реалистичности изображений в системах 3-D графики: учеб. пособие, СПб.; Изд-во СПбГЭТУ «ЛЭТИ», 2010, 64 с.