**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Организация графических систем  
Лабораторная работа №3

«Реализация построения объемных объектов с помощью технологии WebGL»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент | | | | |  | |  |  | | Янин С.А. | | |  |
| Группа |  | М-АС-19 | | | |  | | |  | |  |
|  | | |  |  | | | |  |  | | | |  |
| Руководитель | | |  |  | | | |  | Кургасов В.В. | | | |  |

Липецк 2020 г.

**Задание**

Реализовать программу, демонстрирующую сцену из нескольких (3 и более) объёмных фигур. Программа должна использовать вычислительные мощности GPU.

# Оглавление

Оглавление 3

Введение 4

1 Теоретические сведения 5

1.1 WebGL 5

1.2 GLSL 7

1.3 Three.js 9

1.4 Язык реализации 11

2 Исходный код 12

3 Результаты работы программы 14

Вывод 15

# Введение

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. 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 могу использовать стандартные для веб-среды технологии HTML/CSS/JavaScript и при этом также применять аппаратное ускорение графики.

Если создание настольных приложений работающих с 2d и 3d-графикой нередко ограничивается целевой платформой, то здесь главным ограничением является только поддержка браузером технологии WebGL. А сами 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.

WebGL возник из экспериментов над Canvas 3D американского разработчика сербского происхождения Владимира Вукичевича из компании Mozilla в 2006 году. Впоследствии разработчики браузеров Opera и Mozilla стали создавать свои реализации WebGL. А впоследствии была организована рабочая группа с участием крупнейших разработчиков браузеров Apple, Google, Mozilla, Opera для работы над спецификацией технологии. И в 3 марта 2011 года была представлена спецификация WebGL 1.0.

В данной лабораторной работе мы реализуем 3D объекты с помощью технологии WebGL. Языком реализации в данной работе выступает JavaScript.

# 1 Теоретические сведения

# 1.1 WebGL

WebGL (Web-based Graphics Library) — кроссплатформенный API для 3D-графики в браузере, разрабатываемый некоммерческой организацией Khronos Group. WebGL использует язык программирования шейдеров GLSL. WebGL исполняется как элемент HTML5 и поэтому является полноценной частью объектной модели документа (DOM API) браузера. Может использоваться с любыми языками программирования, которые умеют работать с DOM API, например, JavaScript, Rust, Java, Kotlin и другими. Все ведущие разработчики браузеров Google (Chrome), Opera (Opera), Mozilla (Firefox), и Apple (Safari), являются членами Khronos и реализуют WebGL в своих браузерах. За счёт использования низкоуровневых средств поддержки OpenGL часть кода на WebGL может выполняться непосредственно на видеокартах. WebGL — это контекст элемента canvas HTML, который обеспечивает API 3D графики без использования плагинов.  
WebGL выполняется на графическом процессоре компьютера. То есть необходимо написать код, который выполняется на этом процессоре. Код представлен в виде пар функций. Эти две функции - вершинный и фрагментный шейдер, и обе они написаны на очень строго типизированном языке, подобному C/C++, который называется GLSL. (GL Shader Language). Вместе эта пара функций называется программа.

Задача вершинного шейдера - вычислять положения вершин. Основываясь на положениях вершин, которые возвращает функция, WebGL затем может растеризовать различные примитивы, включая точки, линии или треугольники. В процессе растеризации этих примитивов WebGL прибегает к использованию второй функции - фрагментному шейдеру. Задача фрагментного шейдера - вычислять цвет для каждого пикселя примитива, который в данный момент отрисовывается.

Практически всё API WebGL заключается в настройке состояния для работы этих двух функций. Вы устанавливаете настройки для каждого объекта, который хотите отрисовать, а затем выполняете эти две функции через вызов gl.drawArrays или gl.drawElements, которые выполнят шейдеры на графическом процессоре.

Любые данные, которые вы хотите использовать в этих двух функциях, должны быть переданы на графический процессор. Есть 4 способа, которыми шейдер может получить данные.

Атрибуты и буферы

Буферы - это массивы бинарных данных, загруженных в графический процессор. Обычно буферы содержат вещи вроде положений вершин, нормалей, координат текстур, цветов вершин и т.д., хотя вы вольны положить в них что угодно.

Атрибуты определяют, каким образом данные из ваших буферов передаются в вершинный шейдер. Например, вы можете поместить положения вершин в буфер как три 32-битных числа с плавающей точкой на одно положение. Вы указываете конкретному атрибуту, откуда брать положения вершин, какой тип данных используется (три 32-битных числа с плавающей точкой), начиная с какого индекса в буфере начинаются положения вершин и какое количество байтов нужно получить от одного положения до следующего.

Доступ к буферам не произвольный. Вместо этого вершинный шейдер выполняется заданное количество раз и каждый раз, когда он выполняется, выбирается следующее значение каждого из указанных буферов и назначается атрибуту.

Uniform-переменные

Uniform-переменные - это глобальные переменные, которые устанавливаются перед выполнением программы шейдера.

Текстуры

Текстуры - это массивы данных, к которым есть произвольный доступ в программе шейдера. Чаще всего в текстуру помещается картинка, но текстура - это просто набор данных и вы можете запросто поместить в неё что-то отличное от набора цветов.

Varying-переменные

Varying-переменные позволяют передавать данные из вершинного шейдера фрагментному шейдеру. Во фрагментном шейдере мы получим интерполированные значения вершинного шейдера - зависит от того, отображаем ли мы точки, линии или треугольники.

# 1.2 GLSL

GLSL (OpenGL Shading Language, Graphics Library Shader Language) — язык высокого уровня для программирования шейдеров. Разработан для выполнения математики, которая обычно требуется для выполнения растеризации графики. 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.

Основное преимущество GLSL перед другими шейдерными языками — переносимость кода между платформами и ОС.

Шейдеры используют GLSL (OpenGL Shading Language), специальный язык затенения OpenGL с синтаксисом, аналогичным C. GLSL выполняется непосредственно графическим конвейером. Существует два типа шейдеров: вершинные шейдеры и фрагментные (пиксельные) шейдеры. Шейдеры вершин преобразуют положения фигуры в координаты 3D-чертежа. Шейдеры фрагментов вычисляют отображение цветов фигуры и других атрибутов.

GLSL не так интуитивно понятен, как JavaScript. GLSL строго типизирован, и есть много математики, включающей векторы и матрицы. Это может быть очень сложно — очень быстро.

Вершина - это точка в трехмерной системе координат. Вершины могут иметь и обычно имеют дополнительные свойства. Трехмерная система координат определяет пространство, а вершины помогают определять формы в этом пространстве.

Типы шейдеров

Шейдер-это по существу функция, необходимая для рисования чего-либо на экране. Шейдеры выполняются на GPU (графическом процессоре), который оптимизирован для таких операций. Использование графического процессора для работы с шейдерами разгружает часть числа crunching от центрального процессора. Это позволяет процессору сосредоточить свою вычислительную мощность на других задачах, таких как выполнение кода.

Шейдеры вершин

Шейдеры вершин управляют координатами в трехмерном пространстве и вызываются один раз для каждой вершины. Назначение вершинного шейдера заключается в настройке gl\_Positionпеременной — это специальная, глобальная и встроенная переменная GLSL. gl\_Positionиспользуется для сохранения положения текущей вершины.

void main() – Функция является стандартным способом определения gl\_Positionпеременной. Все, что находится внутриvoid main(), будет выполнено вершинным шейдером. Шейдер вершин дает переменную, содержащую способ проецирования положения вершины в трехмерном пространстве на 2D-экран.

Шейдеры фрагментов

Шейдеры фрагментов (или текстур) определяют цвета RGBA (красный, синий, зеленый, Альфа) для каждого обрабатываемого пикселя — один шейдер фрагментов вызывается один раз на пиксель. Целью шейдера фрагментов является настройка gl\_FragColorпеременной. gl\_FragColorявляется встроенной переменной GLSL, как gl\_Position.

В результате вычислений получается переменная, содержащая информацию о цвете RGBA.

# 1.3 Three.js

Three.js — легковесная кроссбраузерная библиотека JavaScript, используемая для создания и отображения анимированной компьютерной 3D графики при разработке веб-приложений. Three.js скрипты могут использоваться совместно с элементом HTML5 CANVAS, SVG или WebGL.

Three.js часто путают с WebGL, поскольку чаще всего, но не всегда, three.js использует WebGL для рисования 3D. WebGL - это система очень низкого уровня, которая рисует только точки, линии и треугольники. Чтобы сделать что-нибудь полезное с WebGL, как правило, требуется немало кода, и именно здесь приходит Three.js. Он обрабатывает такие вещи, как сцены, источники света, тени, материалы, текстуры, 3D-математику, все, что вам нужно было бы написать самостоятельно. если бы вы использовали WebGL напрямую.

Чтобы работать с three.js, необходимо иметь представление о структуре приложения three.js. Приложение three.js требует, чтобы вы создали группу объектов и соединили их вместе.

Элементы Three.js:

* Renderer. Это, возможно, основной объект three.js. Вы передаете сцену и камеру в модуль рендеринга, и он рендерит (рисует) часть трехмерной сцены, которая находится внутри области камеры, как двухмерное изображение на холсте;
* Scene. Существует сценарий, который представляет собой древовидную структуру, состоящую из различных объектов, таких как объект Scene, нескольких объектов Mesh, объектов Light, GroupObject3D и объектов Camera. Объект Scene определяет корень графа сцены и содержит такие свойства, как цвет фона и туман. Эти объекты определяют иерархическую структуру типа родительского / дочернего дерева и представляют, где объекты появляются и как они ориентированы. Дети расположены и ориентированы относительно своего родителя. Например, колеса на автомобиле могут быть дочерними элементами автомобиля, так что перемещение и ориентация объекта автомобиля автоматически перемещают колеса. Вы можете прочитать больше об этом в статье о сценограммах.
* В Three.js, в отличие от других объектов, камера не обязательно должна быть в графе сцены, чтобы функционировать. Как и другие объекты, камера, как дочерний элемент какого-либо другого объекта, будет перемещаться и ориентироваться относительно своего родительского объекта;
* Mesh. Объекты сетки представляют собой чертеж определенной геометрии с определенным материалом. Объекты Material и Geometry могут использоваться несколькими объектами Mesh. Например, чтобы нарисовать 2 синих куба в разных местах, нам может понадобиться 2 объекта Mesh для представления положения и ориентации каждого куба. Нам понадобится только 1 геометрия для хранения данных вершин куба, и нам понадобится только 1 материал для определения синего цвета. Оба объекта Mesh могут ссылаться на один и тот же объект Geometry и один и тот же объект Material;
* Geometry. Объекты геометрии представляют данные вершин некоторого фрагмента геометрии, такие как сфера, куб, плоскость, собака, кошка, человек, дерево, здание и т.д.. Three.js предоставляет множество видов встроенных геометрических примитивов. Также имеется возможность создавать собственные геометрии и загружать геометрию из файлов;
* Material. Объекты материала представляют свойства поверхности, используемые для рисования геометрии, включая такие вещи, как используемый цвет и насколько он блестящий. Материал также может ссылаться на один или несколько объектов текстуры, которые можно использовать, например, для наложения изображения на поверхность геометрии;
* Texture. Объекты текстуры обычно представляют изображения, загруженные из файлов изображений, сгенерированные с холста или сгенерированные из другой сцены;
* Light. Cветовые объекты, представляющие разные виды света.

# 1.4 Язык реализации

JavaScript — динамический, интерпретируемый язык со слабой типизацией, обычно используемый для написания скриптов на стороне клиента. JavaScript поддерживается всеми современными браузерами и позволяет выполнять на стороне клиента достаточно сложные вычисления. Для JS существует большое количество фреймворков для простого рендеринга 3D-сцен, отрисовки пользовательских интерфейсов.

Кроме того, современные браузеры также поддерживают и использование GPU API WebGL. WebGL предполагает использование языка шейдеров GLSL, который имеет много общего с С. GLSL (OpenGL Shading Language, Graphics Library Shader Language) — язык высокого уровня для программирования шейдеров. Разработан для выполнения математики, которая обычно требуется для выполнения растеризации графики. 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. Основное преимущество GLSL перед другими шейдерными языками — переносимость кода между платформами и ОС. Язык GLSL используется в OpenGL, в OpenGL ES и WebGL используется язык GLSL ES (OpenGL ES Shading Language).

# 2 Исходный код

Файл «index.html»

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset**="utf-8"**>  
 <title>**Лабораторная №3**</title>  
 <style>  
 body { *margin*: 0; *padding*: 0; *font-size*: 0; }  
 canvas { *width*: 100%; *height*: 100%; }  
 </style>  
</head>  
<body>  
 <script src**="js/three.min.js"**></script>  
 <script src**="js/app.js"**></script>  
</body>  
</html>

Файл «app.js»

//Получаем высоту и ширину области, в которой отображается HTML документ в пикселях  
var *WIDTH* = *window*.innerWidth;  
var *HEIGHT* = *window*.innerHeight;  
  
//Создаем экземпляр средства визуализации, параметр передаваемый в конструктов (antialias) со значением true указывает, что необходимо выполнить сглаживание  
var *renderer* = new THREE.WebGLRenderer({antialias:true});  
//Указываем отображаемую область, в данном случае - это весь экран  
*renderer*.setSize(*WIDTH*, *HEIGHT*);  
//Устанавливаем на фон белый цвет  
*renderer*.setClearColor(0xFFFFFF, 1);  
//Установка визуализации в теге body  
*document*.body.appendChild(*renderer*.domElement);  
  
//Создаем экземпляр сцены. Сцена позволяет настроить, что и где должно отображаться, например объекты, источники света и камеры  
var *scene* = new THREE.Scene();  
  
/\*\*  
 \* Экземпляр класса "камера с перспективной проекцией", по сути имитирует то, что видит человеческий глаз  
 \* Первый параметр определяет дальность от зрителя, чем больше параметр, тем "дальше" находятся объекты  
 \* Второй параметр отвечает за поворот по оси x. Т.е. при увеличении или уменьшении данного параметр, мы будем как будто объодить отрисованные объекты слева или справа  
 \* Третий параметр как бы отсекает изображение на указанном расстоянии от нас. т.е. если там будет указано 30, то это значит, что на расстоянии на 30 единиц от нас не будут отражаться объекты  
 \* Четвертый параметр работает по принципу третьего, только ограничивает всё после указанного расстояния, а не до  
 \*/  
  
var *camera* = new THREE.PerspectiveCamera(70, *WIDTH*/*HEIGHT*, 0.1, 1000000);  
//Позиция z - это наша позиция как зрителя, т.е. расстояние до объектов составляет 50 единиц  
*camera*.position.z = 50;  
//В сцену добавляется камера  
*scene*.add(*camera*);  
  
/\*\*  
 \* Библиотека предоставляет методы для создания объектов, например при помощи BoxGeometry создается кубоид  
 \* Первый параметр - длинна по оси x, второй - длинна по оси y, третий - длинна по оси z  
 \* Т.е. при указании например: 10, 20, 30 - мы получим прямоугольный параллепипед  
 \*/  
var *boxGeometry* = new THREE.BoxGeometry(10, 10, 10);  
//Создаем экземпляр материала, в данном случае простоое закрашивание (данный способ не подвержен влиянию света)  
var *basicMaterial* = new THREE.MeshBasicMaterial({color: 0xff6a4f});  
//Класс Mesh предоставляет объекты на основе треугольной сетки  
var *cube* = new THREE.Mesh(*boxGeometry*, *basicMaterial*);  
/\*\*  
 \* Далее двигаем куб по оси x, т.к. по умолчанию оно рендерится на середине экрана  
 \* задаем чере метод rotation вращение, где первый параметр вращение по x, второй по y, третий по z  
 \* добавляем куб в сцену  
 \*/  
*cube*.position.x = -25;  
*cube*.rotation.set(0.3, 0.2, 0);  
*scene*.add(*cube*);  
  
/\*\*  
 \* TorusGeometry создает объект типа тороид  
 \* Первый параметр - радиус  
 \* второй - радиус внутреннего пустого пространства  
 \* третий - количество радиальных сегментов, чем больше значении, тем более круглым будет торус в одной из проекций  
 \* четвертый - другие сегменты в другой проекции, действует по принципу третьего параметра  
 \*/  
var *torusGeometry* = new THREE.TorusGeometry(5, 1, 16, 100);  
/\*\*  
 \* Данный класс позволяет имитировать отражательную способность с зеркальными бликами, или блестящие поверхности  
 \* Например может использоваться для лакированного дерева  
 \*/  
var *phongMaterial* = new THREE.MeshPhongMaterial({color: 0xb52ff});  
var *torus* = new THREE.Mesh(*torusGeometry*, *phongMaterial*);  
*scene*.add(*torus*);  
  
/\*\*  
 \* DodecahedronGeometry создает правильный додекаэдр, который состоит из 12 правильных пятиугольников  
 \* Первый параметр указывает радиус додекаэдра  
 \*/  
var *strangeGeometry* = new THREE.DodecahedronGeometry(5);  
// Данный класс позволяет имитировать отражательную способность, но без имитации блестящих поверхностей, или зеркальных бликов  
var *lambertMaterial* = new THREE.MeshLambertMaterial({color: 0xff00});  
var *dodecahedron* = new THREE.Mesh(*strangeGeometry*, *lambertMaterial*);  
// Двигаем объект по оси x вправо, т.к. в центре уже имеется торус  
*dodecahedron*.position.x = 25;  
*scene*.add(*dodecahedron*);  
  
//Создаем экземпляр класса света, в конструктоор передаем белый цвет  
var *light* = new THREE.PointLight(0xFFFFFF);  
//Устанавливаем где будет находиться источник света. В конструктор передаем x, y, z соответственно  
*light*.position.set(-10, 15, 50);  
*scene*.add(*light*);  
  
//Переменная времени  
var *t* = 0;  
//Функция отрисовки сцены со всеми объектами и камеры, рендера  
function render() {  
 *t* += 0.01;  
 //Функция самого JS, которая указывает браузеру на то, что необходимо произвести анимацию, параметром передаем функцию, которая вызывается, когда придет время обновлять анимацию  
 requestAnimationFrame(render);  
 //Вращаем куб по осям x и y, а торус по оси x  
 *cube*.rotation.y += 0.01;  
 *cube*.rotation.x += 0.01;  
 *torus*.rotation.x += 0.01;  
 //рассчитываем синусоиду по y, что позволит додекаэдру двигаться по оси y  
 *dodecahedron*.position.y = 7\**Math*.sin(*t*\*2);  
 //Осуществляем само отображение объектов  
 *renderer*.render(*scene*, *camera*);  
}  
//При запуске скрипта и инициализации всех объектов вызывается функция описанная выше  
render();

# 3 Результаты работы программы

Для того чтобы проверить работу программы, необходимо открыть в любом поддерживающим WebGL браузере файл index.html. При запуске на странице браузера сразу появляются 3D объекты.

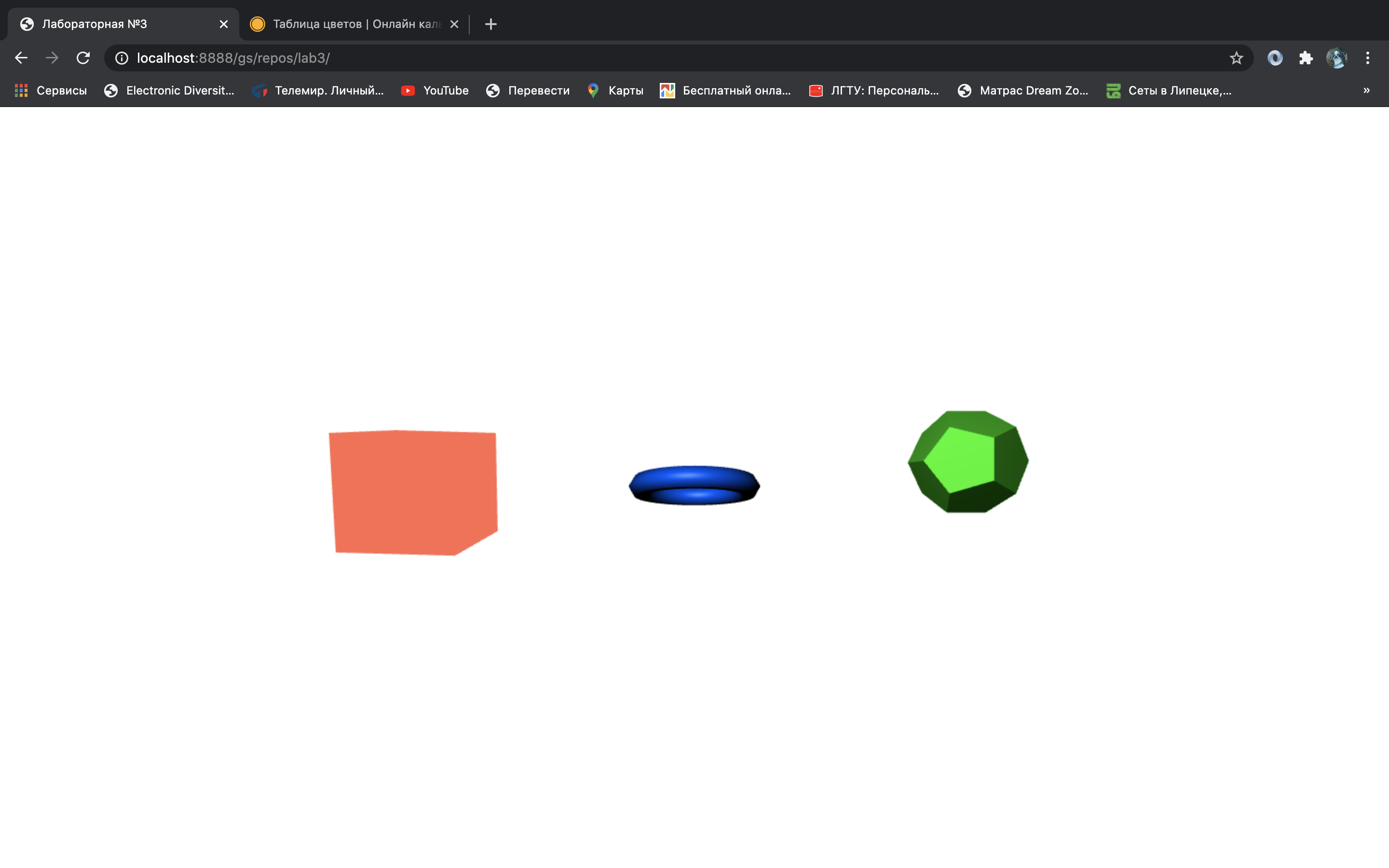


Рисунок 1 – Кадр 1

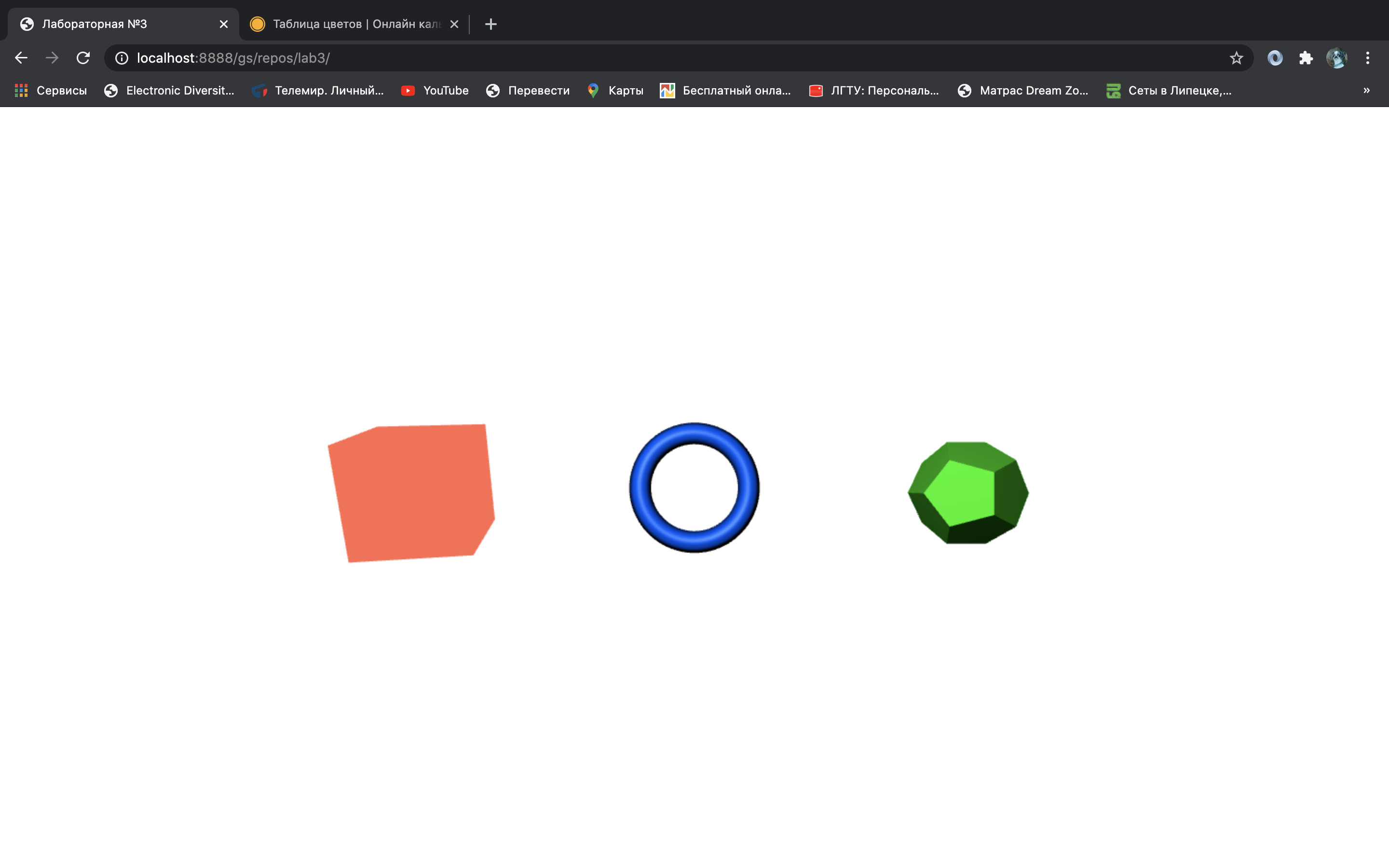


Рисунок 2 – Кадр 2

# Вывод

В ходе выполнения данной лабораторной работы были получены практические навыки обработки графики, а именно работа с движущимися 3D объектами на сцене. Это было достигнуто с помощью библиотеки WebGL, THREE.JS и языка JavaScript.