WebGL – это программная библиотека для JavaScript'а, которая позволяет создавать 3D графику, функционирующую в браузерах.

Three.js – это библиотека, которая опирается на возможности WebGL и делает над ним надстройку.

Скачать библиотеку и посмотреть множество интересных примеров можно на её официальном сайте (http://www.threejs.org).

Прежде чем перейти к написанию кода, нужно понять, что из себя представляет изображение в 3D; мы будем говорить об этом в терминах библиотеки three.js, чтобы в дальнейшем было проще понять и запомнить названия классов.

Основные понятия WebGL и three.js

Первое, что появляется в 3D пространстве – это рендереры (так же называется класс в three.js). **Рендерер** – это сущность, отображающая на canvas выбранную сцену с выбранной камерой. **Сцена** – это 3D пространство, в котором располагаются нужные объекты: элементы (иначе - меши) и источники света, которые освещают элементы с какой-либо стороны. Для наглядности можно представить себе куб как самую простую сцену и три оси, где располагаются элементы. От чего в первую очередь зависит внешний вид элементов? А зависит он именно от камеры – той точки, с который мы сейчас смотрим на сцену (и в ту точку, куда мы смотрим). Поэтому рендерер, чтобы сделать отображение чего-либо на canvas, всегда принимает в себя (мы говорим уже о методе объекта WebGL) сцену и камеру. Именно эти две ключевых вещи формируют отображение пространства на тег canvas.

Что ещё есть на сцене? Источник света – элемент, который создаёт освещение. Он может быть рассеянным, точечным, направленным. Самое главное и интересное понятие, которое здесь представлено, – это **меш.** Меш – это элемент сцены, который состоит из геометрии и материала. 3D объект – не простой объект, как мы хотели бы себе представить, а составная вещь. Любой объект, который мы хотим отобразить в 3D, состоит из *геометрии* и *материала*. Такой подход выбран, чтобы мы могли из простых геометрических примитивов создавать вещи, которые по-разному отображаются. Простейший пример геометрии материала: геометрия – это плоскость, материал – что-то, залитое красным цветом. В итоге получаем красную плоскость.

Геометрия – это набор вершин, которые при генерации соединяются между собой графическими примитивами. Если мы берём простейшую плоскость как геометрию, то в рамках этой плоскости при отображении точки соединяются между собой прямыми линиями, и мы видим результат. Если есть сфера, то там тоже ключевые точки по сегментам соединяются прямыми линиями. И даже если будет 3D объект машины, то и в этом случае всё соединяется примитивными элементами. Разница в том, что в плоскости - четыре ключевые точки, а в машине – четыре тысячи, но суть от этого не меняется.

Переходим непосредственно к материалу. **Материал** – это способ отображения и внешний вид элемента. Здесь кроется интересная сложность: в прошлом примере было сказано, что мы сделаем плоскость, к примеру, красной; но существует множество разных красных плоскостей, и происходит это потому, что материал – это именно способ отображения элемента. Какие-то элементы отражают свет и/или отбрасывают тени, а какие-то нет. И список можно продолжать – всё это зависит от материала. Материал – это ни в коем случае не просто цвет, это то, как он себя ведёт в рамках представления на сцене. Тени и отражение – простейшие примеры для демонстрации.

Ещё одно понятие, которое введем сразу, потому что оно есть в библиотеке, – это текстуры. **Текстура** – это изображение, которое может использоваться в рамках материала, чтобы задать внешний вид объекта. Условно говоря, плоскость можно не только покрасить в красный цвет, но и подгрузить собственную картинку и размножить её по этой плоскости.

Лабораторная работа №1

Цель: Целью данной лабораторной работы является получение базовых навыков разработки приложений трёхмерной графики с использованием библиотеки Three.js

Справка: саму библиотеку three.js можно получить по адресу: http://threejs.org/ Примеры исходного кода написаны для версии r70 и могут иметь некоторые отличия от более поздних, или ранних версий.

Запуск проектов предлагается осуществлять в браузере Google Chrome или любом другом, поддерживающем WebGL 2.0.

Примеры можно найти по адресу: http://stemkoski.github.io/Three.js/

Минимальное приложение

Минимальный проект, как правило, состоит из двух частей, html страница и связанный с ней script файл. Минимально необходимая html страница выглядит следующим образом:

```
<!DOCTYPE html>
<html>
  <head>
   <title></title>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
 <body>
  <div id="container"></div>
  <!-- Подключение библиотеки ThreeJS -->
  <script src="js/libs/three.min.js"></script>
  <!-- Подключение скрипта с графической программой -->
  <script src="js/main.js"></script>
 </body>
</html>
```

Стоит отдельно выделить три действия :

· создание div элемента с id = container. В дальнейшем, этот элемент будет использован в качестве области отображения графического

изображения.

· подключение библиотеки three.min.js. В данном файле, содержится описание функций для работы с графическим API WebGL.

· подключение пользовательского скрипта. Предполагается, что в нём содержится исходный код вашего проекта, отвечающий за создание, загрузку, перемещение и рендеринг графических объектов.

Минимально необходимый script файл может выглядеть следующим образом:

```
// Ссылка на эллемент веб страницы в котором будет отображаться графика
var container;
// Переменные "камера", "сцена" и "отрисовщик"
var camera, scene, renderer;
// Функция инициализации камеры, отрисовщика, объектов сцены и т.д.
init();
// Обновление данных по таймеру браузера
animate();

function init()
{
   //получение ссылки на элемент html страницы
   container = document.getElementById( 'container' );
   //создание "сцены"
   scene = new THREE.Scene();
  //Установка параметров камеры
  //45 - угол обзора
  //window.innerWidth / window.innerHeight - соотношение сторон
  //1 - 4000 - ближняя и дальняя плоскости отсечения
  camera = new THREE.PerspectiveCamera( 45, window.innerWidth /
window.innerHeight, 1, 4000 );
  //Установка позиции камеры
  camera.position.set(25, 25, 50);
  //Установка точки, на которую камера будет смотреть
  camera.lookAt(scene.position);
  // Создание отрисовщика
  renderer = new THREE.WebGLRenderer( { antialias: false } );
  renderer.setSize( window.innerWidth, window.innerHeight );
//закрашивание экрана синим цветом, заданным в 16ричной системе
  renderer.setClearColor( 0x000088, 1);
  container.appendChild( renderer.domElement );
  // Добавление функции обработки события изменения размеров окна
 window.addEventListener( 'resize', onWindowResize, false );

}
function animate()
```

```
{
requestAnimationFrame( animate );
render();
}
function render()
{
renderer.render( scene, camera );
}
```

Задание: реализовать минимальное приложение three.js При запуске, в окне браузера, должна открываться html страница, залитая красным цветом.

Лабораторная работа №2

Вершины, индексы, цвет

При использовании библиотеки three.js, работа с вершинами и индексами осуществляется через класс геометрии. Например, создание треугольника будет выглядеть следующим образом:

```
var triangleGeometry = new THREE.Geometry();

//Добавление координат вершин в массив вершин
triangleGeometry.vertices.push(new THREE.Vector3( 0.0, 5.0, 0.0));
triangleGeometry.vertices.push(new THREE.Vector3(-5.0, -5.0, 0.0));
triangleGeometry.vertices.push(new THREE.Vector3( 5.0, -5.0, 0.0));

//Добавление индексов (порядок соединения вершин) в массив индексов
triangleGeometry.faces.push(new THREE.Face3(0, 1, 2));
```

Добавление цветов осуществляется так же при помощи объекта геометрии. Для назначения цвета той или иной вершине, используется номер треугольника, которому она принадлежит:

```
//Добавление цветов для вершин
triangleGeometry.faces[0].vertexColors[0] = new THREE.Color(0xFF0000);
triangleGeometry.faces[0].vertexColors[1] = new THREE.Color(0x00FF00);
triangleGeometry.faces[0].vertexColors[2] = new THREE.Color(0x0000FF);
```

Цвета задаются в шестнадцатеричной системе, в формате RGB. В библиотеке three.js , графический объект помимо вершин, индексов и цвета имеет настройки отображения, которые хранятся в специальной структуре данных “материал”:

```
var triangleMaterial = new THREE.MeshBasicMaterial({
vertexColors:THREE.VertexColors,
side:THREE.DoubleSide
});
```

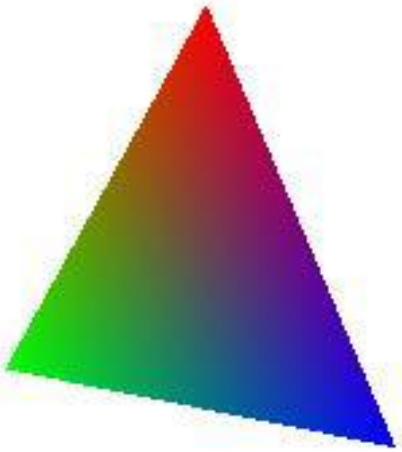
В данном примере, используется определение источника цвета объекта (указаны цвета вершин) и режим отрисовки объекта (будут прорисованы обе стороны объекта). Существует весьма полезная опция отрисовки wireframe : true, которая позволяет отображать объект в виде полигональной сетки.

Финальным этапом является создание графического объекта на основе геометрии и материала, и добавление его в сцену:

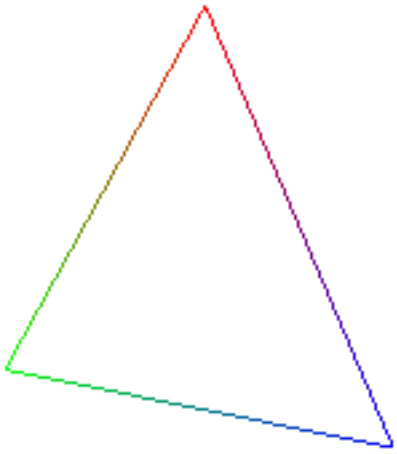
```
// Создание объекта и установка его в определённую позицию
var triangleMesh = new THREE.Mesh(triangleGeometry, triangleMaterial);
```

```
triangleMesh.position.set(-15.0, 0.0, -10.0);
scene.add(triangleMesh);
```

После выполнения этих действий, при запуске проекта, в окне браузера должно получиться следующее изображение:



А в случае использования wireframe : true:



Задание: разработать программу, отображающую на экран монитора регулярную полигональную сетку заданного размера. Часть

Лабораторная работа №3
Текстуры

Задание текстурных координат осуществляется так же при помощи объекта геометрия и выглядит следующим образом:

```
squareGeometry.faceVertexUvs[0].push([new THREE.Vector2(0, 0),
new THREE.Vector2(1, 0),
new THREE.Vector2(1, 1)]);
squareGeometry.faceVertexUvs[0].push([new THREE.Vector2(0, 0),
new THREE.Vector2(1, 1),
new THREE.Vector2(0, 1)]);
```

Данный пример исходного кода показывает, каким образом можно задать прямое, полное отображение текстуры на четырёхугольник. Непосредственно загрузка текстуры осуществляется следующим образом :

```
var tex = new THREE.ImageUtils.loadTexture( 'tex.jpg');
```

а установка текстуры в материал так:

```
var mat = new THREE.MeshBasicMaterial({
map:tex,
side:THREE.DoubleSide
});
```

Текстурированный подобным образом четырёхугольник может выглядеть так **Задание**: модифицировать программу из прошлого раздела таким образом, что бы для регулярной сетки рассчитывались текстурные координаты.

Итоговое задание: Разработать программу осуществляющую построение и визуализацию трёхмерного ландшафта по карте высот. Под картой высот, подразумевается grayscale растровое изображение, тёмные участки которого обозначают низины ландшафта, а светлые – возвышенности. Визуализируемая модель должна быть текстурирована.
Пример кода функций, для чтения пикселей изображения:

```
var canvas = document.createElement('canvas');
var context = canvas.getContext('2d');
var img = new Image();
img.onload = function()
{
canvas.width = img.width;
canvas.height = img.height;
context.drawImage(img, 0, 0 );
imagedata = context.getImageData(0, 0, img.width, img.height);
CreateTerrain();
}
img.src = 'cDsYZ.jpg';
function getPixel( imagedata, x, y )
{

var position = ( x + imagedata.width * y ) * 4, data = imagedata.data;
return data[ position ];;
}
```

Пример использования функции: //получение цвета пикселя в десятом столбце десятой строки изображения

```
var heigh = getPixel( imagedata, 10, 10 );
```

Так же, при выполнении задания, может быть использован встроенный механизм расчёта освещения, включающий в себя три этапа.

Добавление источника освещения:

```
//создание точечного источника освещения заданного цвета
var spotlight = new THREE.PointLight(0xffff00);
//установка позиции источника освещения
spotlight.position.set(0,20, 0);
//добавление источника в сцену
scene.add(spotlight);
```

Расчёт нормалей для геометрии:
geometryObj.computeFaceNormals();
geometryObj.computeVertexNormals();

Использование материала, поддерживающего расчёт освещения :
var planeMaterial = new THREE.MeshLambertMaterial({ map: tex, side: THREE.DoubleSide });

Предполагаемый результат до наложения текстур :