Программирование графических приложений

Тема 1
Введение в WebGL

Цель изучения темы. Освоение принципов программирования веб-приложений с использованием WebGL.

1.1. Общие сведения по WebGL

WebGL (Web-based Graphics Library) - программная библиотека, предназначенная для создания интерактивной трехмерной графики в веб-браузерах.

За счёт использования низкоуровневых средств поддержки OpenGL часть кода на **WebGL** может выполняться непосредственно на видеокартах, что дает выигрыш по быстродействию.

Под этим же словом «WebGL» обычно подразумевают и саму базирующуюся на OpenGL ES 2.0 технологию создания трехмерной графики с помощью одноименной библиотеки и предназначенную для рисования и отображения интерактивной 2D- и 3D-графики в веб-браузерах. Технология WebGL разрабатывается промышленным консорциумом Khronos Group, который специализируется на выработке открытых стандартов интерфейсов программирования в области создания и воспроизведения динамической графики и звука на различных платформах и устройствах. В консорциум входят более 100 компаний.

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, увеличив 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 пользовательских приложений, работающих с 2d и 3d-графикой, обычно ограничивается целевой платформой, то здесь главным ограничением является только поддержка браузером технологии WebGL. Сами же веб-приложения, построенные с использованием данной технологии, будут доступны в любой точке интернет вне зависимости от используемой платформы: будь это десктопы с ОС Windows, Linux, Mac, смартфоны, планшеты, игровые консоли.

WebGL возник из экспериментов над Canvas 3D компании Mozilla в 2006 году. Впоследствии разработчики браузеров Opera и Mozilla стали создавать свои реализации WebGL. А затем была организована рабочая группа с участием крупнейших разработчиков браузеров Apple, Google, Mozilla, Opera для работы над спецификацией технологии. И в 2011 году была представлена спецификация WebGL 1.0. В настоящее время существует спецификация WebGL 2.0.

Поддержка браузерами

В настоящий момент WebGL ***в разной степени*** поддерживается следующими 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, с сайта http://www.iewebgl.com)
- Microsoft Edge.

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;
- Google Chrome для Android.

Преимущества использования WebGL

- Кроссбраузерность и отсутствие привязки к определенной платформе. Windows, MacOS, Linux - все это не важно, главное, чтобы ваш 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.

Чтобы иметь возможность увидеть примеры на WebGL, необходимо запустить браузер. ***Рекомендуемый браузер***, поддерживающий все функции WebGL – Microsoft Edge (https://www.microsoft.com/en-us/edge). ***Тестер*** на странице отчета WebGL (https://webglreport.com/) дает подробную информацию о поддержке вашим браузером различных функций. Можно также попробовать запустить демонстрационные ***примеры*** на WebGL из репозитория Khronos (https://www.khronos.org/webgl/wiki/Demo_Repository).

Для разработки приложения на WebGL не нужно ничего, кроме браузера, однако разработчику помогут специализированные среды – для работы в интернет (JS Fiddle - https://jsfiddle.net/, JS Bin - https://jsbin.com, Code Pen - https://codepen.io) или локальные (WebStorm - https://www.jetbrains.com/webstorm, ***SubLime Text*** - http://www.sublimetext.com), а также ***локальный веб-сервер***, например, Servez - https://greggman.github.io/servez. Кроме того, практически все браузеры имеют встроенные инструменты web-разработки. Например, в браузере Mozilla Firefox инструменты разработчика вызываются по Ctrl+Shift+I.

В частности, для несложных разработок в рамках выполнения данных лабораторных работ лучше применять легкие редакторы класса SubLime Text. На сайте https://sublimetext3.ru можно посмотреть начальные инструкции по установке Sublime Text 3, познакомиться с основными свойствами интерфейса, настроек, основными полезными функциями. Там же есть краткие инструкции по использованию менеджера пакетов Sublime Package Control, предназначенного для установки плагинов на Sublime Text. Редактор минималистичен, но для него разработано большое число плагинов, выборочная установка которых позволяет настроить Sublime Text на выполнение своих задач, не перегружая среду разработки лишними функциями. Можно посмотреть, например, рекомендации: https://nicothin.pro/page/sublime-text-3-plugins.

Для повышения скорости разработки кроме фреймворков WebGL можно использовать и существующие библиотеки с открытым исходным кодом, содержащие набор готовых классов для создания и отображения интерактивной компьютерной 3D графики в WebGL, например, Three.js.

Но сначала рассмотрим простую программу на чистом WebGL без использования библиотек.

1.2. Простая программа на WebGL без библиотек

Построение графики в программе происходит на объекте canvas. Отрисовка в WebGL осуществляется с помощью вершинных и фрагментных (пиксельных) шейдеров.

Шейдеры - это функции, написанные на специальном языке программирования GLSL (GL Shader Language), которые выполняются графической картой и обрабатывают данные вершин и

пикселей, определяя параметры изображения объекта. Они могут включать в себя описание поглощения и рассеяния света, наложение текстуры, отражение и преломление, затенение, и т.д.

Работа с шейдерами - это достаточно трудоемкий процесс, так как нужно описать каждую вершину, каждое ребро, каждую грань, нормали к поверхности, их цвет, положение и пр.

Приведенная ниже программа показывает в действии технологии WebGL. Пока не будем детально останавливаться на этом коде - рассмотрим его основные моменты позднее. В тексте программы помогут разобраться и содержащиеся в нём подробные комментарии.

Создадим веб-страничку со следующим содержимым (**ex01_01.html**):

```
<!DOCTYPE html>
<html>
<head>
<title>Привет WebGL!</title>
</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");
```

```
    // делаем доступным атрибут для использования
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
}
// Функция создания шейдера по типу и id источника
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 x[i], y[i], z[i]
// центр системы координат совпадает с левым нижним углом треугольника
  var triangleVertices = [
         0.0,  0.0,  0.0,
         0.0,  0.5,  0.0,
         0.5,  0.0,  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 WebGL
     // 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 фон в светло-фиолетовый цвет (r=1, g=0, b=1, α=0.5)
     gl.clearColor(1.0, 0.0, 1.0, 0.5);
     // отрисовка сцены
     draw();
  }
}
</script>
</body>
</html>
```

В программе определяется элемент *canvas*, который представляет собой полотно для отрисовки сцены WebGL. Далее в виде скриптов javascript объявляются две мини-программы - фрагментный и вершинный шейдеры.

Весь код программы находится в обработчике функции *window.onload*. Вначале нам надо получить контекст (мы назвали его *gl*), через который и будут вестись все основные действия. Далее программа содержит три основные части: установка шейдеров, установка буфера вершин и сама отрисовка.

Далее идет функция, срабатывающая при загрузке страницы. Для начала устанавливаем контекст:

```
gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
```

Элемент canvas поддерживает два контекста: 2d (используется для простого рисования) и webgl, который нам и надо получить. Однако некоторые браузеры поддерживают только экспериментальный контекст - experimental-webgl, поэтому в данном случае мы также пытаемся получить и его, если контекст webgl не установлен.

Далее идет закраска. Метод gl.clearColor() принимает значения RGBA для цвета, которым будет закрашено полотно. И чтобы осуществить закраску, вызываем метод gl.clear (точнее очищаем цветовой буфер, используя константу gl.COLOR_BUFFER_BIT).

Результатом работы должен быть элемент canvas, окрашенный в заданный цвет в соответствие с цветовыми компонентами и значением A (альфа-канала). В компьютерной графике альфа-значение определяет степень прозрачности пиксела для комбинирования изображения с фоном с целью создания эффекта частичной прозрачности и применяется для сглаживания, прозрачности, создания теней, зеркал, тумана.

При отрисовке треугольника сначала срабатывает функция initShaders(), производящая инициализацию шейдеров и их настройку. Шейдеры являются обязательным звеном в конвейере WebGL. 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.

Если запустить веб-страничку в браузере, мы получим белый треугольник на светло-фиолетовом фоне:

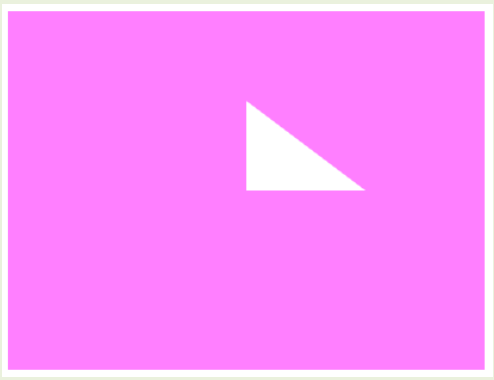

Рис. 1.1

1.3. Библиотека Three.js

Three.js –это библиотека JavaScript, содержащая набор готовых классов для создания и отображения интерактивной компьютерной 3D графики в WebGL**.**

Библиотека Three.js облегчает работу с WebGL. При использовании Three.js отпадает необходимость в написании шейдерных процедур (но эта возможность остается), и появляется возможность оперировать с более привычными и удобными понятиями сцены, света и камеры, объектами и их материалами.

Библиотека Three.js также поддерживает отображение готовых трёхмерных моделей формата Collada (который обеспечивает совместимость моделей таких программ, как Maya, 3ds Max, Blender, Unreal engine, и т.д.)

Для использования Three.js нужно скачать с сайта threejs.org из раздела build файл минимизированной версии библиотеки three.min.js (http://threejs.org/build/three.min.js), и подключить к проекту. Это библиотека, содержащая минимальный набор функций для работы с WebGL. Там же есть и полный вариант библиотеки (three.js), а также большое число других библиотек для различных целей. На сайте threejs.org доступны для скачивания примеры, исходники и документация.

Работу с 3D графикой WebGL с помощью Three.js можно условно представить в виде нескольких простых действий:

- добавление сцены;
- добавление камеры;
- добавление освещения;
- добавление графических объектов на сцену;
- создание объекта визуализации;
- рендеринг (визуализация);
- анимация (движение объектов, их взаимодействие, управление).

1.4. Простая программа на WebGL с Three.js

Программа представляет собой традиционный пример рисования вращающегося куба.

В одном каталоге с загруженной библиотекой three.min.js создадим файл .html со следующим содержанием (**ex01_02.html**):

```
<DOCTYPE html>
```

```
<html>
<head>
<title>Привет WebGL!</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() {
      // 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 объекта:  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 сетку, которая 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 структуру страницы
      document.body.appendChild(renderer.domElement);
   }
   // функция анимации
   function animate() {
       requestAnimationFrame(animate);
      // вращение сетки 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>
```

Результат:

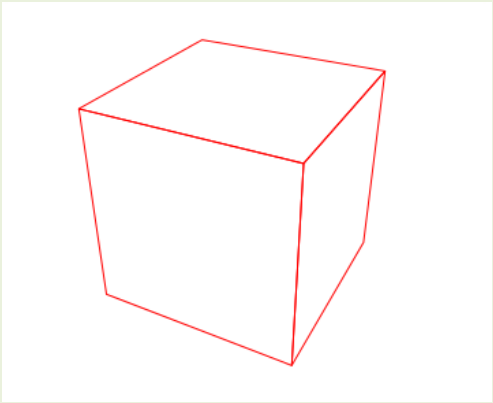

Рис. 1.2

С помощью программы **ex01_02.html** формируется более сложное изображение, чем с помощью **ex01_01.html**, но кода в программе меньше: Three.js (так же как другие библиотеки и фреймворки для работы с WebGL) позволяет значительно сократить код и упростить работу с графикой. Но в то же время даже минимизированнные версии библиотек имеют очень большой объем и в результате увеличивают проект. Какой способ следует выбрать - это дело разработчика.

1.5. Структурированная программа на WebGL с Three.js

В разделе 1.3 приведена последовательность действий для работы с 3D графикой WebGL с помощью Three.js. Создадим проект, реализующий такую структуру для формирования изображения прямоугольного параллелепипеда с реализацией всех шагов из раздела 1.3 и настройкой перечисленных там основных элементов сцены.

1.5.1. Структура проекта

Для нашего проекта будем использовать файлы:

ex01_03.html - файл главной страницы;
ex01_03.js - файл с логикой нашего приложения;
three.min.js - библиотека three;
TrackballControls.js - библиотека элементов управления.

В дальнейшем для примеров программ в основном будем использовать такую же структуру с возможным добавлением других библиотек.

На главной странице **ex01_03.html** нужно подключить наши библиотеки и создать раздел, в котором будет создан «холст» canvas для отрисовки. Для обращения к этому разделу присвоим ему, например, идентификатор «MyWebGLApp».

Тогда главная страница будет иметь вид (файл **ex01_03.html**):

```
<!DOCTYPE html>
<html>
 <head>
  <title>First WebGL</title>
  <meta charset="utf-8">
  <style>
   body { margin: 0; padding: 0; overflow: hidden; }
  </style>
  <script type="text/javascript" src="three.min_.js"></script>
  <script type="text/javascript" src="TrackballControls.js"></script>
  <script type="text/javascript" src="ex01_03.js"></script>
 </head>
```

```
<body>
  <div id="MyWebGLApp"></div>
</body>
</html>
```

Здесь мы убрали полосу прокрутки страницы при помощи стилевой команды overflow: hidden. Весь дальнейший код мы будем писать в файле **ex01_03.js**, для чего можно использовать любой подходящий (удобный) редактор.

1.5.2. Добавление сцены

Именно на сцену мы будем добавлять все созданные нами объекты. Объявим переменную:

```
var scene;
```

Переменная scene должна быть глобальной. Создается сцена просто:

```
scene = new THREE.Scene ();
```

Для добавления объектов на сцену или удаления их со сцены используются методы add и remove:

```
scene.add ( object );
scene.remove ( object );
```

1.5.3. Добавление камеры

Простейший тип камеры - это «перспективная» камера, которая воспринимает все объекты в перспективной проекции. Т.е., например, чем дальше объект находится от нас, тем он кажется нам меньше. Объявим глобальную переменную и создадим камеру:

```
var camera;
camera = new THREE.PerspectiveCamera ( 45, window.innerWidth /
          window.innerHeight, 1, 10000 );
```

Первый аргумент - это FOV (field of View - поле (угол) зрения), в примере - 45.

Второй аргумент - это пропорция, т.е. соотношение сторон (обычно ширина экрана в пикселях делится на высоту, например, 4:3=1.33).

Третий и четвертый аргументы - минимальное и максимальное расстояние от камеры, которое попадает в рендеринг. То есть очень далекие точки не будут отрисовываться вообще.

У камеры можно указать положение:

```
camera.position.set (0, -20, 100);
```

и направление обзора:

```
camera.lookAt (newTHREE.Vector3 (10, -100, 100));
```

то есть камера направлена на точку с координатами (10,-100,100). Можно следить за объектами:

```
camera.lookAt (object.position);
```

По умолчанию камера всегда смотрит в центр холста, то есть точку с координатами

(0,0,0). При изменении каких-либо параметров камеры нужно затем вызвать метод:

```
camera.updateProjectionMatrix();
```

С такой «перспективной» камерой трудно достаточно полно оценить «трехмерность» объектов на сцене. Хочется иметь возможность рассматривать объекты с разных сторон, вблизи или издалека. Для этого в Three.js имеется возможность управления обзором сцены с помощью специальных элементов управления, добавление которых позволяет менять точку обзора камеры с помощью мышки, приближаться или удаляться от сцены.

Скачайте с сайта http://threejs.org (http://threejs.org/examples/js/controls/TrackballControls.js) файл TrackballControls.js и запишите в рабочую папку проекта. Для использования этой библиотеки нужно на главной странице index.html добавить ссылку на неё:

```
<script type="text/javascript" src="TrackballControls.js"></script>
```

Теперь, после создания камеры, можно добавить наш элемент управления:

```
var controls;
controls = new THREE.TrackballControls( camera, container );
```

Первый параметр - наша камера camera. Второй параметр container указывает на раздел страницы, в котором производится отрисовка (см. ниже). После создания элемента управления можно задать, например, скорость вращения камеры при движении мыши:

```
controls.rotateSpeed = 2;
```

Можно указать будет ли изменяться положение камеры (приближаться или удаляться от сцены) при кручении колесика мыши (по умолчанию false - меняется), и скорость такого изменения:

```
controls.noZoom = false;
controls.zoomSpeed = 1.2;
```

Также можно указать будет ли камера после остановки мыши немного двигаться по инерции или же нет; по умолчанию false (движется):

```
controls.staticMoving = true;
```

Есть и другие элементы управления. Например, FirstPersonControls позволяет передвигаться по сцене с помощью стрелок (или кнопок W, A, S, D) и мышки. Пример можно посмотреть на сайте http://threejs.org (http://threejs.org/examples/webgl_geometry_minecraft.html).

Теперь можно с помощью мышки рассматривать объекты сцены со всех сторон, приближаться и удаляться от них, что вносит в приложение интерактивность.

Для удобства добавление камеры и элемента управления в файле **ex01_03.js** выделено в отдельную процедуру AddCamera().

1.5.4. Система координат в WebGL

Для расположения фигур в пространстве в WebGL используется декартова система координат:

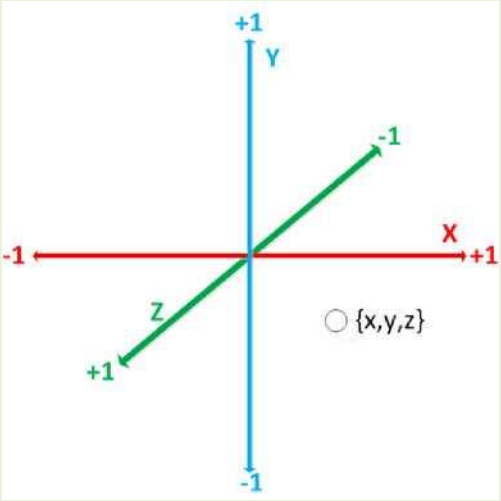

Рис. 1.3

Как обычно, для программ, работающих с 3D, наверх направлена ось Y. На рисунке также указаны правила изменения знаков. Например, при приближении точки к наблюдателю значение её координаты Z увеличивается.

При добавлении объекта на сцену в Three.js можно указать его координаты:

```
object.position.x = -50;
object.position.y = 20;
object.position.z = 60;
```

или одной строкой:

```
object.position.set ( -50, 20, 60 );
```

или с помощью класса трехмерных векторов Vector3:

```
object.position = new THREE.Vector3( -50, 20, 60 );
```

При этом в указанной точке обычно располагается геометрический центр тела. Если же координаты объекта не указаны, то они все равны нулю.

Можно задать коодинаты объекта, приравняв их к координатам другого:

```
object2.position = object1.position;
```

Для задания углов поворота тела используется свойство rotation. Углы указываются в радианах. Например, команда

```
object.rotation.y = Math.PI/2;
```

означает, что объект разворачивается на 90 градусов против часовой стрелки, если смотреть «сверху» - со стороны положительного направления оси *OY*, при этом осью вращения служит ось ординат *OY*. Синтаксис обращения к *rotation* такой же, как и у *position*.

1.5.5. Добавление света в Three.js

Освещение объектов придаст сцене большей реалистичности. Для освещения можно использовать класс *DirectionalLight*, который представляет собой источник прямого направленного освещения. Этот источник можно сравнить с солнечными лучами. С его

помощью создается поток параллельных лучей в любом направлении.

Объявим глобальную переменную для источника света:

```
var light;
```

Определим источник света:

```
light = new THREE.DirectionalLight( 0xffffff );
light.position.set ( 0, 100, 100 );
scene.add ( light );   // добавление света на сцену
```

В качестве параметра при создании света можно указать цвет освещения (обычно белый). В файле **ex01_03.js** добавление света оформлено в виде отдельной процедуры AddLight().

1.5.6. Добавление объекта визуализации

Для отображения сцены и ее объектов при помощи WebGL, в Three.js используется специальный класс WebGLRenderer. Объявим переменные для экземпляра этого класса и для элемента, с которым этот класс будет работать:

```
var container, renderer;
```

Вместе с классом создается холст canvas, который по умолчанию имеет ширину 300 пикселей и высоту 150 пикселей. Метод setSize позволяет изменить его размеры. Например, создадим объект- визуализатор, указав размеры холста на всё окно:

```
renderer = new THREE.WebGLRenderer ();
renderer.setSize (window.innerWidth, window.innerHeight );
```

Также можно указать цвет фона (в разных версиях Three.js цвет фона по умолчанию менялся). Укажем белый фон:

```
renderer.setClearColor ( 0xffffff );
```

Далее следует указать визуализатору renderer, где именно будет создаваться холст. Мы приготовили для этого раздел MyWebGLApp на главной странице:

```
container = document.getElementById ('MyWebGLApp');
container.appendChild ( renderer.domElement );
```

Теперь холст для рисования размещен в нашем разделе на главной странице.

Для придания моделям более реалистичного вида в 3D графике применяют сглаживание. В Three.js допускается использовать сглаживание при помощи параметра antialias. Тогда объявление можно изменить следующим образом:

```
renderer = new THREE.WebGLRenderer ( { antialias: true } );
```

Далее во всех примерах этот параметр будет установлен.

Также в Three.js имеется класс CanvasRenderer, который обеспечивает визуализацию простой графики в браузерах без поддержки WebGL. Тогда можно составить более сложную конструкцию:

```
try { renderer= new THREE.WebGLRenderer ( { antialias: true } ); }
```

```
catch (err)
{
alert ('Ваш браузер не поддерживает WebGL!');
try { renderer = new THREE.CanvasRenderer; }
}
```

При отображении в браузерах без поддержки WebGL пострадают качество и скорость отображения, а также изображаться корректно будут лишь объекты с простыми материалами.

1.5.7. Рендеринг и анимация

Рендеринг (отрисовка) производится с помощью созданного в предыдущем пункте объекта renderer. Для этого у него есть метод render, в параметрах которого указываются сцена и камера:

```
renderer.render (scene, camera);
```

Анимация есть, по сути, последовательный неоднократный рендеринг сцены (для отображения динамики). Такое последовательное отображение осуществляется специально для этого предназначенной функцией requestAnimationFrame. Ее можно назвать аналогом другой Javascript-функции setInterval, но она оптимизирована для отображения неподвижных объектов.

В демонстрационных примерах Three.js для инициализации, анимации и рендеринга используются функции init(), animate() и render(); мы будем использовать те же назваания.

В функции init() происходит инициализация сцены, камеры, света, добавление всех объектов, которые мы хотим увидеть на сцене. Функция animate() будет использовать requestAnimationFrame и постоянно вызывать функцию render(), которая уже и будет отрисовывать все изменения:

```
function animate()
 {
  requestAnimationFrame(animate);
  render();
 }
```

Функция render() будет содержать все изменения на сцене. Например, задать движение объекта Cube можно такой функцей:

```
var Cube;   // переменная Cube должна быть глобальной
function render()
 {
  Cube.position.x = Cube.position.x + 1;
  Cube.rotation.y = Cube.rotation.y + 0.01;
  controls.update ();
  renderer.render (scene, camera);
 }
```

Теперь объект Cube движется вправо и вращается вокруг своей оси, параллельной оси ординат *OY*.

Здесь также добавлена возможность управления обзором мышкой при помощи controls. При движении мышки положение камеры меняется, и нужно каждый раз рисовать новую картину сцены. Метод update как раз служит для обновления картины при манипуляциях с мышкой.

1.5.8. Добавление графических объектов

После вызова отдельной процедуры init() для создания сцены, добавления света, рендерера, камеры и т.д., можно добавлять объекты.

Добавление объектов производится следующим образом. Сначала объявляется вид объекта. В простейших случаях это может быть параллелепипед, сфера, цилиндр и т.д. Параметры объекта включают обычно линейные размеры и количество сегментов, отвечающее за точность изображения.

Например, для прямоугольного параллелепипеда предусмотрен специальный класс BoxGeometry (в старых версиях Three.js класс назывался CubeGeometry):

```
var geometry = new THREE.BoxGeometry( 200, 100, 150);
```

Указываются ширина параллелепипеда (вдоль оси X), высота (вдоль оси Y) и длина (вдоль оси Z).

Следующим шагом указывается материал будущего объекта (о материалах подробнее чуть позже):

```
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
```

Здесь мы указали зеленый цвет материала будущего объекта. И, наконец, создается сам объект с выбранными параметрами и материалом. Делается это через класс Mesh - сетку будущего объекта:

```
Cube = new THREE.Mesh( geometry, material );
```

Осталось добавить объект на сцену. Можно указать позицию (свойство position) и поворот относительно осей координат - rotation:

```
Cube.position.z = -100;
Cube.rotation.z = Math.PI / 6;
scene.add( Cube );
```

Полный код файла **ex01_03.js**:

```
// глобальные переменные
var container, camera, controls, scene, renderer, light;
var Cube;

// начинаем рисовать после полной загрузки страницы
window.onload = function()
 {
  init();
  animate();
 }

function init()
 {
  scene = new THREE.Scene(); //создаем сцену
  AddCamera( 0, 300, 500); //добавляем камеру
  AddLight( 0, 0, 500 ); //устанавливаем белый свет
```

```
 //создаем рендерер
 renderer = new THREE.WebGLRenderer( { antialias: true } );
 renderer.setClearColor( 0xffffff );
 renderer.setSize( window.innerWidth, window.innerHeight );
 container = document.getElementById('MyWebGLApp');
 container.appendChild( renderer.domElement );

 //добавляем куб
 var geometry = new THREE.BoxGeometry( 200, 100, 150);
 var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
 Cube = new THREE.Mesh( geometry, material );
 Cube.position.z = -100;
 Cube.rotation.z = Math.PI / 6;
 scene.add( Cube );
}

function animate()
 {
  requestAnimationFrame(animate);
  render();
 }

function render()
 {
 Cube.position.x = Cube.position.x + 0; // +1 - куб движется
 Cube.rotation.y = Cube.rotation.y + 0.01; //и вращается вокруг оси
 controls.update();
 renderer.render(scene, camera);
}

function AddCamera(X,Y,Z)
 {
 camera = new THREE.PerspectiveCamera( 45, window.innerWidth /
          window.innerHeight, 1, 10000 );
 camera.position.set(X,Y,Z);
 controls = new THREE.TrackballControls( camera, container );
 controls.rotateSpeed = 2;
 controls.noZoom = false;
 controls.zoomSpeed = 1.2;
 controls.staticMoving = true;
 }

function AddLight(X,Y,Z)
 {
 light = new THREE.DirectionalLight( 0xffffff );
 light.position.set(X,Y,Z);
 scene.add( light );
 }
```

Результат:

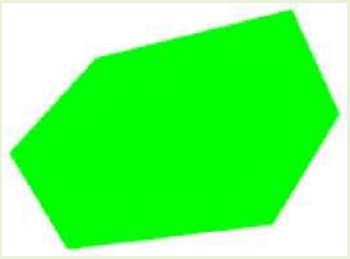

Рис. 1.4

Если теперь добавить код из пункта 1.5.4, то наш параллелепипед будет двигаться вправо и вращаться вокруг оси. При этом переменная Cube должна быть глобальной.

При выбранном материале не так заметна трехмерность объекта. Применим другой вид материала:

```
var material = new THREE.MeshNormalMaterial();
```

Тогда за счет разноцветной окраски объекта лучше видна его трехмерность:

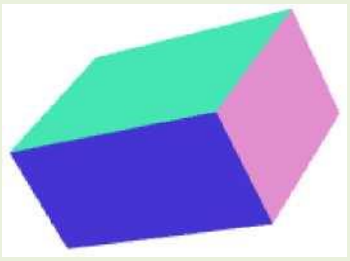

Рис. 1.5

Благодаря controls наш параллелепипед можно рассматривать в браузере со всех сторон, приближаться к нему и удаляться от него. Достаточно лишь движения мышки и ее колеса.

Для геометрии поверхности и материала необязательно объявлять отдельные переменные. Первые три строки создания куба эквивалентны, например, паре таких команд:

```
var material = new THREE.MeshBasicMaterial ( { color: 0x00ff00 });
var Cube = new THREE.Mesh (new THREE.BoxGeometry( 200, 300, 50), material );
```

или даже одной:

```
var Cube = new THREE.Mesh (new THREE.BoxGeometry( 200, 300, 50),
                           new THREE.MeshBasicMaterial( { color: 0x00ff00 } ) );
```

Контрольные вопросы

1. Отличительные особенности WebGL.
2. Построение простых изображений в WebGL.
3. Библиотека Three.js.