Программирование графических приложений

Тема 5
Плоские объекты в WebGL

5.1. Рисование плоских фигур
5.2. Построение круга и кольца
5.3. Спрайты

Контрольные вопросы

Цель изучения темы. Изучение способов построения плоских объектов при формировании графических изображений с использованием WebGL.

5.1. Рисование плоских фигур

Рассмотрим рисование плоских фигур в WebGL с использованием библиотеки three.js.

Для построения плоских поверхностных объектов используются классы, являющиеся наследниками класса **Geometry**:

- PlaneGeometry представляет плоский прямоугольник (кусок плоскости);
- RingGeometry используется для построения кольца;
- CircleGeometry позволяет строить плоский круг;

Для рисования полских фигур, состоящих из линий, используется метод Line(geometry, material). Первый аргумент является экземпляром класса Geometry и содержит набор вершин фигуры. Материал фигуры material выбирается либо LineBasicMaterial - для сплошных линий, либо LineDashedMaterial - для пунктирных линий.

Изобразим, например, правильный **шестиугольник** на плоскости. Каждая вершина такого шестиугольника лежит на стороне угла с величиной, кратной 60 градусам (в радианах Math.PI/3). Объявим новый объект и материал:

```
var geometry = new THREE.Geometry;
var material = new THREE.LineBasicMaterial ( { color: 0xcc0000 } );
```

и в цикле добавим все вершины:

```
for (var i=0; i<=6; i++)
{
var a = new THREE.Vector3 (50*Math.cos( Math.PI/3*i ),
50*Math.sin( Math.PI/3*i ), 0 );
geometry.vertices.push( a );
}
```

Для замыкания кривой мы добавили семь вершин, где седьмая совпадает с первой. Теперь создаем фигуру line и добавляем на сцену:

```
var line = new THREE.Line( geometry, material );
scene.add(line);
```

Результат (**ex05_01.html, ex05_01.js**):

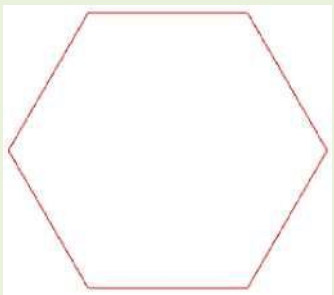

Рис. 5.1

Для изображения прямоугольной части **плоскости** используется класс **PlaneGeometry**:

PlaneGeometry(width, height, widthSegments, heightSegments);

В качестве параметров указываются ширина, высота и количество сегментов по ширине и высоте.

Изобразим плоскость(**ex05_02.html, ex05_02.js**):

```
var planeMaterial = new THREE.MeshBasicMaterial ({wireframe: true, color: 0x9999ff });
var planGeo = new THREE.PlaneGeometry( 100, 100, 1, 1);
var plane = new THREE.Mesh(planGeo, planeMaterial);
plane.position.set (50,0,50);
plane.rotation.x = Math.PI/2;
scene.add (plane);
```

Для того чтобы оставить только границы плоскости, использован параметр wireframe, который взят равным true.

Результат (справа плоскость с числом сегментов по ширине и высоте равным 5):

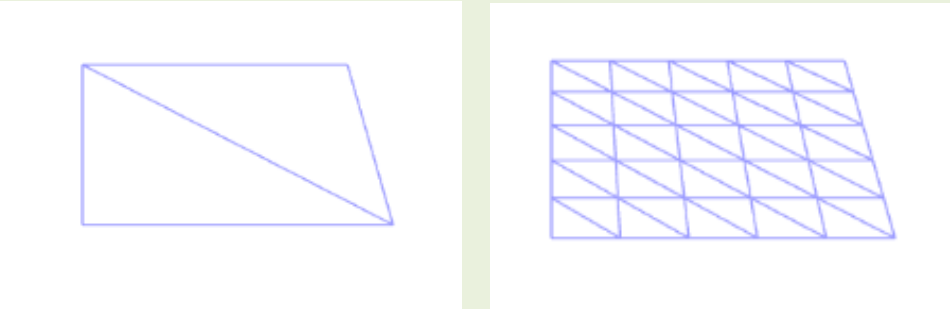

Рис. 5.2

5.2. Построение круга и кольца

Класс **CircleGeometry** позволяет строить плоский **круг** или его часть. Указываются радиус, количество сегментов, начальный угол и величина угла. Отчет идет против часовой стрелки.

Например, нарисуем два сектора разных цветов (**ex05_03.html, ex05_03.js**):

```
//фиолетовый сектор
var material = new THREE.MeshBasicMaterial({color: 0xe100ff});
var geometry =new THREE.CircleGeometry( 100, 16, 0, Math.PI );
var figure = new THREE.Mesh(geometry, material);
scene.add( figure );
//зеленый сектор
var material = new THREE.MeshBasicMaterial({ color: 0x177245 });
var geometry =new THREE.CircleGeometry( 100, 16, Math.PI, 2*Math.PI/3);
var figure = new THREE.Mesh(geometry, material);
scene.add( figure );
```

Результат:

Рис. 5.3

При этом фигура будет видна только с одной стороны. Изменить это можно с помощью параметра материала **side**:

```
var material = new THREE.MeshBasicMaterial({color: 0xe100ff,
                        side:THREE.DoubleSide});
```

Если включить этот параметр в объявление материала для каждого сектора, наша фигура станет видна с обеих сторон.

Для построения **кольца** используется класс **RingGeometry**. Синтаксис вызова следующий:

```
RingGeometry ( innerRadius, outerRadius, thetaSegments, phiSegments, thetaStart,
                thetaLength );
```

Указываются размеры внутреннего и внешнего радиусов, количество угловых и радиальных сегментов, начальный угол и величина угла.

Создадим кольцо, видимое с обеих сторон (**ex05_04.html, ex05_04.js**):

```
var material = new THREE.MeshLambertMaterial ({color: 0x082567, side:
                        THREE.DoubleSide});
var geometry = new THREE.RingGeometry (20, 100, 20, 10, 0, 2*Math.PI);
ring = new THREE.Mesh ( geometry, material );
scene.add ( ring );
```

Результат:

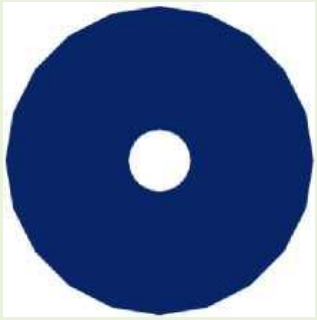

Рис. 5.4

5.3. Спрайты

Спрайт – двухмерный графический объект в компьютерной графике. Чаще всего это растровый объект, при отображении которого на экране создаётся эффект меняющегося изображения: меняться могут его геометрические и визуальные параметры. Например, объект движется, меняет свою форму или цвет. При этом графика анимируется с помощью перебора изображений из атласа спрайтов. Атлас спрайтов является растровым изображением, объединяющим спрайты в набор раскадровок анимации, или комплектом таких изображений.

Ниже прритведен код программы, использующей спрайты для анимации объекта (**ex05_05.html**).

Для изображения прямоугольной части плоскости, на которой будет двигаться объект, используется класс PlaneGeometry. Сам спрайт анимируется с помощью класса TextureAnimator:

```
TextureAnimator (texture, horiz, vert, total, duration);
```

Параметры: texture - текстура, horiz - число спрайтов в атласе по горизонтали, vert - число спрайтов по вертикали, total – общее число спрайтов, duration - интервал времени для сканирования всего атласа.

```
<!doctype html>
<html lang="en">
<head>
<title>Спрайт</title>
<meta content="charset=utf-8">
<meta name="viewport" content="width=device-width, user-scalable=no, minimum-
        scale=1.0, maximum-scale=1.0">
</head>
<body>

<script src="Three.js"></script>
<script src="Detector.js"></script>
<script src="OrbitControls.js"></script>
<script src="THREEx.KeyboardState.js"></script>
<script src="THREEx.FullScreen.js"></script>
<script src="THREEx.WindowResize.js"></script>

<div id="infoButton"></div>
<div id="ThreeJS" style="position: absolute; left:0px; top:0px"></div>
<script>
var container, scene, camera, renderer, controls;
var keyboard = new THREEx.KeyboardState();
var clock = new THREE.Clock();
var annie;
init();
animate();
function init()
 {
  scene = new THREE.Scene();
  var SCREEN_WIDTH = window.innerWidth, SCREEN_HEIGHT =
        window.innerHeight;
  var VIEW_ANGLE = 45, ASPECT = SCREEN_WIDTH / SCREEN_HEIGHT, NEAR =
        0.1, FAR = 20000;
  camera = new THREE.PerspectiveCamera( VIEW_ANGLE, ASPECT, NEAR, FAR);
  scene.add(camera);
  camera.position.set(0,150,400);
  camera.lookAt(scene.position);
  if ( Detector.webgl )       renderer = new THREE.WebGLRenderer( {antialias:true} );
  else      renderer = new THREE.CanvasRenderer();
  renderer.setSize(SCREEN_WIDTH, SCREEN_HEIGHT);
  container = document.getElementById( 'ThreeJS' );
  container.appendChild( renderer.domElement );
  THREEx.WindowResize(renderer, camera);
  THREEx.FullScreen.bindKey({ charCode : 'm'.charCodeAt(0) });
  controls = new THREE.OrbitControls( camera, renderer.domElement );
  var light = new THREE.PointLight(0xffffff);
  light.position.set(0,250,0);
```

```
 scene.add(light);
 var runnerTexture = new THREE.ImageUtils.loadTexture( 'marsch.png' );
 annie = new TextureAnimator( runnerTexture, 6, 1, 6, 100 );
 var runnerMaterial = new THREE.MeshBasicMaterial( { map: runnerTexture,
          side:THREE.DoubleSide } );
 var runnerGeometry = new THREE.PlaneGeometry(50, 50, 1, 1);
 var runner = new THREE.Mesh(runnerGeometry, runnerMaterial);
 runner.position.set(-100,25,0);
 scene.add(runner);
}
function animate()
{
 requestAnimationFrame( animate );
 render();
 update();
}
function update()
{
 var delta = clock.getDelta();
 annie.update(1000 * delta);
 controls.update();
 stats.update();
}
function render() { renderer.render( scene, camera ); }
function TextureAnimator(texture, tilesHoriz, tilesVert, numTiles, tileDispDuration)
{
 this.tilesHorizontal = tilesHoriz;
 this.tilesVertical = tilesVert;
 this.numberOfTiles = numTiles;
 texture.wrapS = texture.wrapT = THREE.RepeatWrapping;
 texture.repeat.set( 1 / this.tilesHorizontal, 1 / this.tilesVertical );
 this.tileDisplayDuration = tileDispDuration;
 this.currentDisplayTime = 0;
 this.currentTile = 0;
 this.update = function( milliSec )
 {
  this.currentDisplayTime += milliSec;
  while (this.currentDisplayTime > this.tileDisplayDuration)
  {
   this.currentDisplayTime -= this.tileDisplayDuration;
   this.currentTile++;
   if (this.currentTile == this.numberOfTiles)
 this.currentTile = 0;
   var currentColumn = this.currentTile % this.tilesHorizontal;
   texture.offset.x = currentColumn / this.tilesHorizontal;
   var currentRow = Math.floor( this.currentTile / this.tilesHorizontal );
   texture.offset.y = currentRow / this.tilesVertical;
  }
 };
}
</script>
</body>
```

```
</html>
```

Атлас спрайтов – файл **marsch.png**:

Рис. 5.5

Прямоугольную область со спрайтом, полученную в браузере в результате выполнения программы, можно перемещать по кнопке мыши как обычный плоский объект.

Контрольные вопросы

1. Построение двухмерных объектов в WebGL.
2. Использование спрайтов в WebGL.