Программирование графических приложений

Тема 4
Плоские кривые линии в WebGL

4.1. Класс Geometry
4.2. Рисование осей и координатной сетки
4.3. Построение параметрических кривых
4.4. Интерполяция сплайнами
4.5. Построение кривых Безье
4.6. Построение графика функции с использованием mathbox-bundle.js

Контрольные вопросы

Цель изучения темы. Изучение способов построения плоских кривых линий при формировании моделей графических объектов с использованием WebGL.

4.1. Класс Geometry

При изучении способов построения плоских кривых линий в WebGL будем использовать библиотеку Three.js. Последовательность действий для работы с помощью Three.js приведена в теоретическом и практическом материале к теме №1. Там для построения графических объектов использовался класс Geometry.

Класс Geometry является родительским для классов, представляющих различные объекты. Основными атрибутами этого класса являются наборы вершин и граней. Набор вершин содержится в свойстве-массиве vertices, грани фигуры содержатся в массиве faces. Правила освещения фигуры реализуются с помощью ее массива вершинных нормалей normals.

Для работы с гранями фигуры в библиотеке Three.js предусмотрены специальные классы Face3 и Face4. В качестве обязательных аргументов нужно указать номера трех, или, соответственно, четырех вершин (из массива vertices).

Как и рассмотренный в примерах **ex01_02.html**, **ex01_03.html** объект BoxGeometry, другие возможные объекты (CylinderGeometry, SphereGeometry и др.) являются наследниками класса Geometry. В частности класс **TubeGeometry** представляет «трубу», которая выдавливается окружностью вдоль профиля кривой.

4.2. Рисование осей и координатной сетки

Библиотека Three.js содержит готовые методы для добавления координатных осей и координатной сетки.

Для изображения осей служит метод **AxisHelper**. Его единственный аргумент - длина осей. Этот метод рисует сразу все три оси. Если камера находится на оси z, будут видны только оси x и y.

Для изображения сетки служит метод **GridHelper**. Он имеет два параметра - **size** (длина сетки) и **step** (шаг сетки). Центр сетки приходится по умолчанию на начало координат. Метод **setColors** позволяет установить цвет линий сетки. Первый аргумент метода отвечает за цвет центральных линий, второй аргумент - за цвет остальных линий сетки. В двухмерном случае может быть изображена только сетка, находящаяся в плоскости x0y.

Тогда оси и координатную сетку двухмерной системы координат x0y можно изобразить, например, таким кодом (**ex04_01.html, ex04_01.js**):

```
var axes = new THREE.AxisHelper(150);
axes.position.set( 0,0,0 );
scene.add(axes);
var gridXY = new THREE.GridHelper(100, 20);
gridXY.position.set( 50,50,0 );
gridXY.rotation.x = Math.PI/2;
gridXY.setColors( new THREE.Color(0x0000ff), new THREE.Color(0x0000ff) );
scene.add(gridXY);
```

Результат:

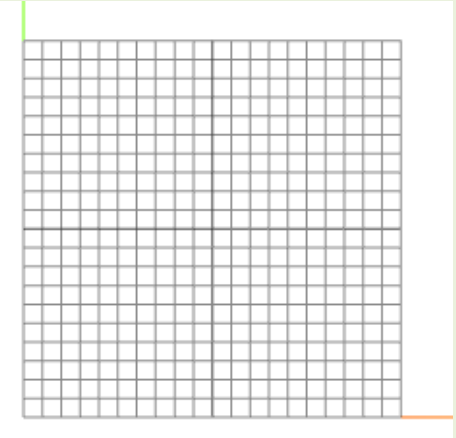

Рис. 4.1

4.3. Построение параметрических кривых

Самый простой способ построения параметрической кривой линии на плоскости - с помощью метода 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, эллипс. Параметрически эллипс можно задать так: $x=a \cdot cos(t)$, $y=b \cdot sin(t)$, $z=0$.

Объявим параметры кривой, геометрию и материал:

```
var a = 40, b = 20;
var geometry = new THREE.Geometry;
var material = new THREE.LineBasicMaterial ( { color: 0xcc0000 } );
```

Осталось внести в массив описания объекта координаты вершин:

```
for (var t = 0; t<= 6.3; t+= 0.1)
 {
  var vec =new THREE.Vector3 (a*Math.cos(t), b*Math.sin(t), 0);
  geometry.vertices.push (vec);
 }
```

Наконец, создаем линию и добавляем на сцену:

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

Линия готова (**ex04_02.html, ex04_02.js**):

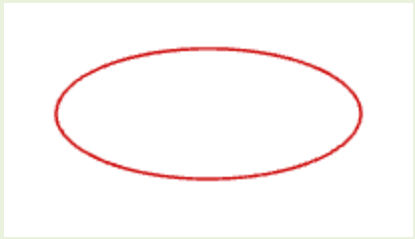

Рис. 4.2

Толщина полученной линии всегда равна единице. Для придания линии толщины можно

воспользоваться классом **TubeGeometry**. Он позволяет построить трубу с круглым сечением, которая выдавливается вдоль кривой.

Для использования этого класса наша кривая должна быть наследником класса **Curve**:

```
chain = new THREE.Curve.create ( function(){}, function(t)
{
 t = 2 * Math.PI * t;
 var a = 40, b = 20;
 var x = a*Math.cos(t);
 var y = b*Math.sin(t);
 var z = 0;
 return new THREE.Vector3(x, y, z).multiplyScalar(2);
} );
```

Объявляем экземпляр класса и объект:

```
var mychain = new chain;
var tubegeo = new THREE.TubeGeometry (mychain, 128, 2, 12, closed = false);
```

Первый аргумент класса **TubeGeometry** ссылается на нашу кривую, второй отвечает за количество сегментов объекта. Третий аргумент - радиус трубы, четвертый - количество сегментов окружности трубы. Пятый аргумент позволяет замкнуть концы кривой (при **closed = true**). Осталось подготовить материал:

```
var material = new THREE.MeshPhongMaterial( { color: 0x9b2d30, specular: 0xd53e07,
        emissive: 0x000000, shininess: 40, shading: THREE.FlatShading, blending:
        THREE.NormalBlending, depthTest: true } );
```

Наконец, создаем сеть и добавляем её на сцену:

```
var tube = new THREE.Mesh (tubegeo, material);
scene.add (tube);
```

Результат на рисунке (**ex04_03.html, ex04_03.js**):

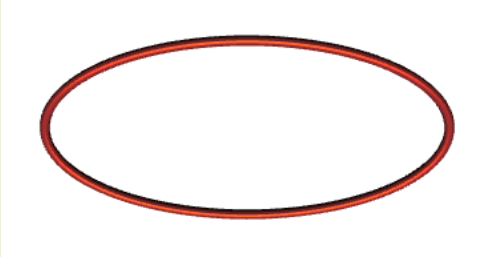

Рис. 4.3

Если применить параметр t и к третьей координате z, то получим параметрически заданную кривую в пространстве. Например, при линейной зависимости координаты z от параметра t, получим спираль с равномерным шагом: x=a·cos(t), y=b·sin(t), z= c·t. Для этого также надо увеличить диапазон изменения параметра t, чтобы у спирали было несколько витков. Например, увеличив верхнее значение параметра t до 30, получим 6 витков спирали. Константа 'c' будет определять расстояшие между витками.

4.4. Интерполяция сплайнами

Интерполирование - это способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений. С помощью интерполяции можно по заданным точкам построить непрерывную гладкую кривую, используя в качестве функции сплайн. Допустим, задано семейство точек (значения функции в узлах):

Xi	-2	-1	1	4	7	10	13	16
Yi	-1	-2	2	3	-1	2	-4	-2

Для построения по заданным точкам гладкой кривой, служит функция **SplineCurve**, аргументы которой - множество координат наших узлов:

```
spline = new THREE.SplineCurve([
new THREE.Vector2(-2, -1), new THREE.Vector2(-1, -2), new THREE.Vector2(1,2),
new THREE.Vector2(4, 3), new THREE.Vector2(7, -1), new THREE.Vector2(10, 2),
new THREE.Vector2(13, -4), newTHREE.Vector2(16, -2) ]);
```

Теперь для построения кривой можно воспользоваться ее методом **getPoint(t)**. Этот метод возвращает вектор для точки **t** кривой, где **t** находится между 0 и 1. Итак, объявим объект и материал:

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

Заносим вершины в geometry:

```
for (var i = 0; i <= 1; i+=0.01)
{
var x = spline.getPoint( i ).x;
var y = spline.getPoint( i ).y;
var vec = new THREE.Vector3( x, y, 0 );
geometry.vertices.push( vec );
}
```

Строим линию:

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

Результат (**ex04_04.html, ex04_04.js**):

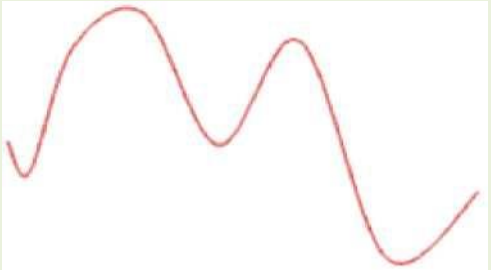

Рис. 4.4

Для придания линии толщины вновь можно воспользоваться классом **TubeGeometry**. Для этого надо использовать трехмерную интерполяцию с помощью **SplineCurve3** (кривая строится на трехмерных векторах):

```
spline = new THREE.SplineCurve3([
  new THREE.Vector3(-2, -1, 0), new THREE.Vector3(-1, -2, 0),
  new THREE.Vector3(1, 2, 0), new THREE.Vector3(4, 3, 0),
  new THREE.Vector3(7, -1, 0), new THREE.Vector3(10, 2, 0),
  new THREE.Vector3(13, -4, 0), new THREE.Vector3(16, -2, 0),
]);
```

Объявляем экземпляр класса и объект:

```
var tubegeo = new THREE.TubeGeometry ( spline, 128, 1, 12 );
```

Готовим материал:

```
var material = new THREE.MeshPhongMaterial ( { color: 0x9b2d30, specular: 0xd53e07,
        emissive: 0x000000, shininess: 40, shading: THREE.FlatShading, blending:
        THREE.NormalBlending, depthTest: true } );
```

Создаем сеть и добавляем на сцену:

```
var tube = new THREE.Mesh ( tubegeo, material );
scene.add ( tube );
```

Результат (**ex04_05.html, ex04_05.js**):

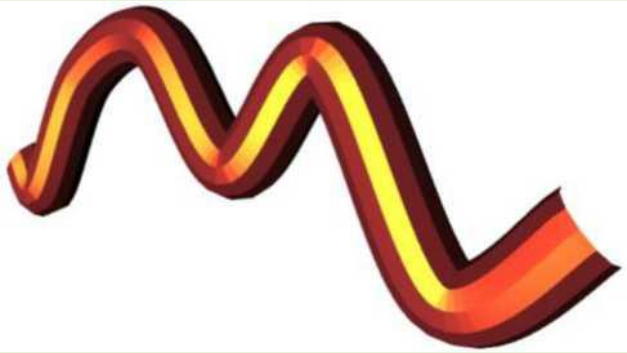

Рис. 4.5

Линия приобрела толщину, но по сути осталась плоской (координаты z всех её точек равны нулю).

Вместо метода **getPoint(t)** можно было воспользоваться методом **getPoints(N)**, который возвращает готовый массив из **N** точек кривой. Тогда следует написать:

```
var points = spline.getPoints( 100 );
for (var i = 0; i < points.length; i++)
{
var x = points[i].x;
var y = points[i].y;
var vec = new THREE.Vector3( x, y, 0 );
```

```
geometry.vertices.push( vec );
}
```

Остальной код программы тот же самый.

4.5 Построение кривых Безье

Для построения квадратичной кривой Безье используется метод **QuadraticBezierCurve**, параметрами которого являются три двумерных вектора. Изобразим, например, кривую Безье по трем опорным точкам (0,44), (84,80) и (120,0).

Опишем три трехмерных вектора:

```
v0 = new THREE.Vector2 ( 0, 44 );
v1 = new THREE.Vector2 ( 84, 80 );
v2 = new THREE.Vector2 ( 120, 0 );
```

Вычисляем кривую Безье:

```
var curve = new THREE.QuadraticBezierCurve ( v0, v1, v2 );
```

Осталось описать геометрию и материал и заполнить геометрию точками кривой Безье:

```
var geometry = new THREE.Geometry;
var material = new THREE.LineBasicMaterial ( { color: 0x00cc00 } );
for (var i = 0; i <= 1; i+=0.01)
 {
  var x = curve.getPoint ( i ).x;
  var y = curve.getPoint ( i ).y;
  var vec = new THREE.Vector3 ( x, y, 0 );
  geometry.vertices.push ( vec );
 }
var line = new THREE.Line ( geometry, material );
scene.add ( line );
```

Результат (**ex04_06.html, ex04_06.js**):

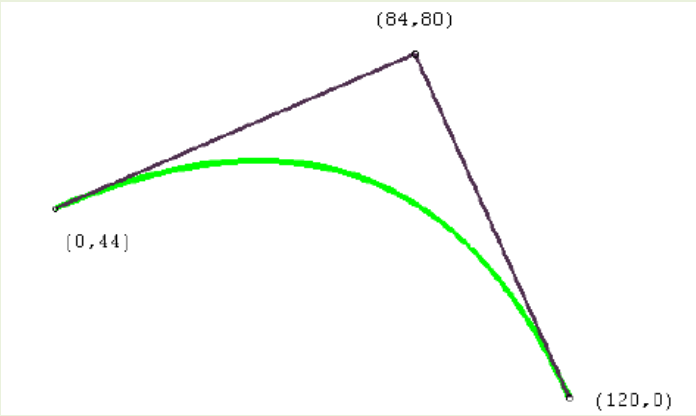

Рис. 4.6

Для построения кубической кривой Безье по четырем заданным двумерным векторам используется функция **CubicBezierCurve (v0, v1, v2, v3)**. Кроме того, существуют трехмерные аналоги этих кривых **QuadraticBezierCurve3 (v0, v1, v2)** и **CubicBezierCurve3 (v0, v1, v2, v3)**.

Их аргументы являются трехмерными векторами.

4.6. Построение графика функции с использованием mathbox-bundle.js

Приведём текст постой программы построения графика функции y=f(x) на интервале -2π<x<2π, использующей библиотеку **mathbox-bundle.js**. Код не требует особых комменттариев (**ex04_07.html**):

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>Graph</title>
 <script src="mathbox-bundle.js"></script>
 <link rel="stylesheet" href="mathbox.css">
 <meta name="viewport" content="initial-scale=1, maximum-scale=1">
</head>
<body>
 <script>
  var mathbox = mathBox({
   plugins: ['core', 'controls', 'cursor', 'mathbox'],
   controls: {klass: THREE.OrbitControls}
  });
  if (mathbox.fallback) throw "WebGL не поддерживается"
  var three = mathbox.three;
  three.renderer.setClearColor (new THREE.Color(0xFFFFFF), 1.0);
   var camera = mathbox.camera ( { proxy: true, position: [0, 0, 3] } );
   var view = mathbox.cartesian ( {range: [[-6.28,6.28], [-2,2]], scale: [2,1]} );
   var xAxis = view.axis ({axis:1, width:8, detail:40, color:"blue"});
   var yAxis = view.axis ({axis:2, width:8, detail:40, color:"green"});
   var grid = view.grid ({width:2, divideX:20, divideY:10, opacity:0.25});
   var graphData = view.interval ({
    expr: function (emit,x,i,t) {emit(x,Math.cos(x)*Math.sin(10*x)+Math.atan(20*x)); },
    width: 256,   channels: 2,
    });
  var graphView = view.line ( {width: 4, color: "red"} );
 </script>
</body>
</html>
```

Результат:

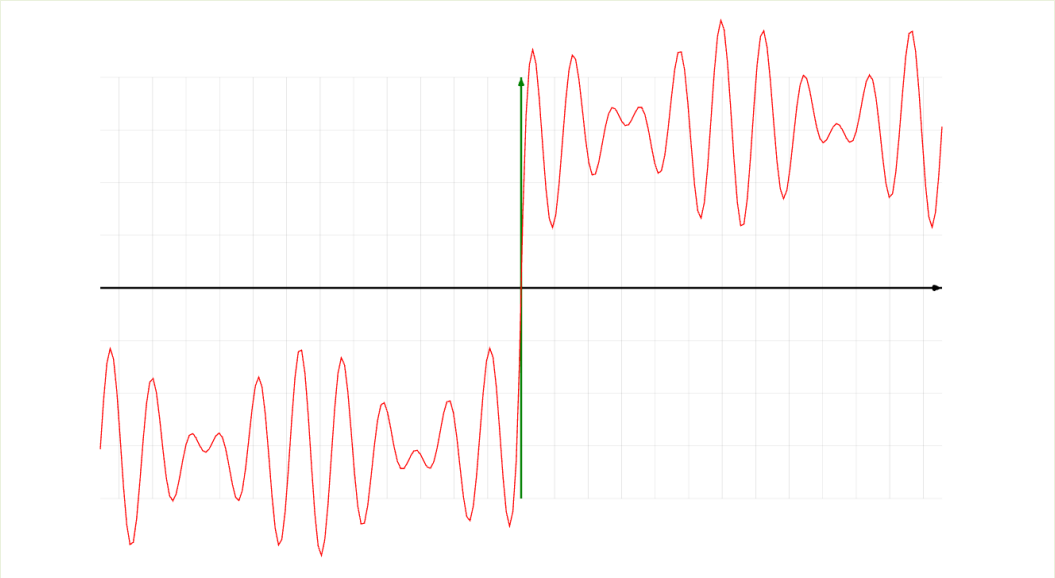

Рис. 4.7

Контрольные вопросы

1. Применение класса Geometry из библиотеки Three.js для построения плоских кривых линий.
2. Построение плоских параметрических кривых линий в WebGL.
3. Построение сплайнов и кривых Безье с использованием библиотеки Three.js в WebGL.