Лабораторная работа №3:

Загрузка объектов из внешних файлов

Цель:

Целью данной лабораторной работы является получение навыков загрузки, отображения и перемещения в трёхмерном пространстве статических и анимированных трёхмерных моделей, созданных во внешнем редакторе

Справка: модели, используемые в лабораторной работе, можно скачать по адресу:
http://tf3dm.com/3d-models/architecture
и: https://github.com/mrdoob/three.js/tree/master/examples
Библиотеки для загрузки моделей из внешних файлов входят в комплект поставки `three.js`: https://github.com/mrdoob/three.js/releases

Часть 1: Загрузка объектов

Помимо прочего, библиотека three.js предоставляет набор функций для загрузки моделей из внешних файлов. Используя расширения библиотеки для загрузки различных форматов, можно добавлять в сцену как статичные, так и анимированные, объекты.

В данной работе, имеет смысл собрать подключение модулей загрузки объектов в главном js файле. Соответственно, index.html будет выглядеть следующим образом:

```
<!DOCTYPE html>
<html>
    <head>
        <!--заголовок страницы-->
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>

        <!--объект веб страницы, в котором будет отображаться графика-->
        <div id="container"></div>

        <!--подключение скрипта с графической программой-->
        <script type="module" src="js/main.js"></script>

    </body>
</html>
```

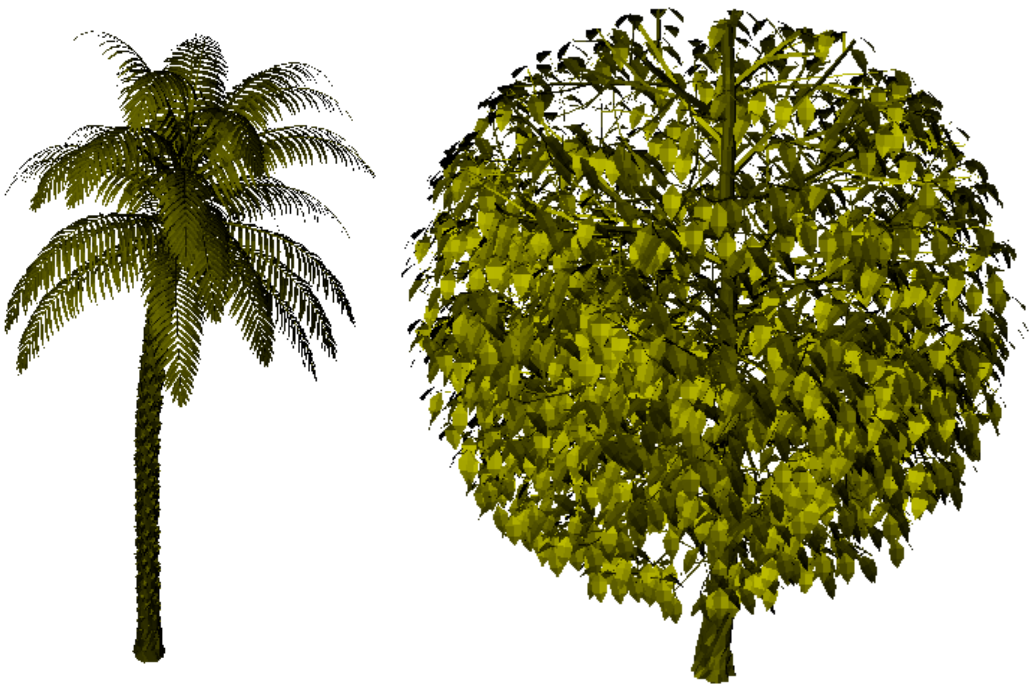
В самом начале main.js следует импортировать версию библиотеки `three.module.js`, а затем, библиотеки для работы с материалами и объектами:

```
//импорт библиотеки three.js
import * as THREE from "./lib/three.module.js";
//импорт библиотек для загрузки моделей и материалов
import { MTLLoader } from './lib/MTLLoader.js';
import { OBJLoader } from './lib/OBJLoader.js';
```

Функция для загрузка статичной модели формата obj + mtl:

```
function loadModel(path, oname, mname) //где path – путь к папке с моделями
{
    const onProgress = function ( xhr ) { //выполняющаяся в процессе загрузки
        if ( xhr.lengthComputable ) {
            const percentComplete = xhr.loaded / xhr.total * 100;
            console.log( Math.round( percentComplete, 2 ) + '% downloaded' );
        }
    };
    const onError = function () { }; //выполняется в случае возникновения ошибки

    const manager = new THREE.LoadingManager();
    new MTLLoader( manager )
        .setPath( path )                        //путь до модели
        .load( mname, function ( materials ) {  //название материала
            materials.preload();
            new OBJLoader( manager )
                .setMaterials( materials )      //установка материала
                .setPath( path )                //путь до модели
                .load( oname, function ( object ) { //название модели
                    //позиция модели по координате X
                    object.position.x = 200;
                    //масштаб модели
                    object.scale.set(0.2, 0.2, 0.2);
                    //добавление модели в сцену
                    scene.add( object );
                }, onProgress, onError );
        } );
}
// вызов функции загрузки модели (в функции Init)
loadModel('models/', "Tree.obj", "Tree.mtl");
```

Пример статичных моделей, используемых в лабораторной работе:



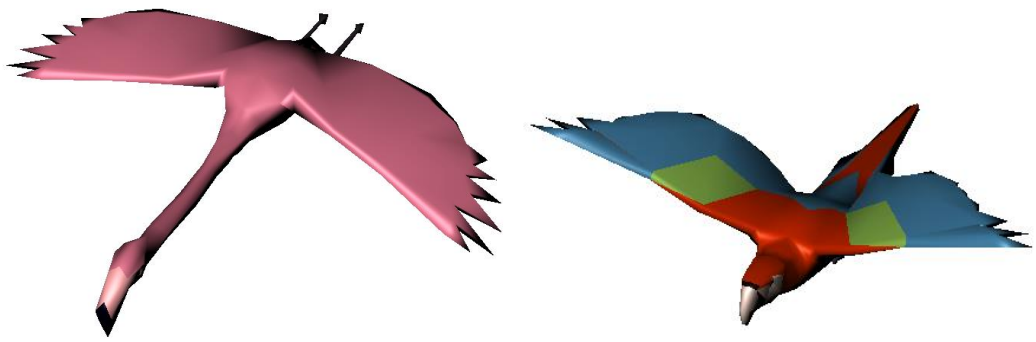
Загрузка анимированной модели формата glb:

```
//импорт библиотеки для загрузки моделей в формате glb
import { GLTFLoader } from './lib/GLTFLoader.js';
//глобальные переменные для хранения списка анимаций
var mixer, morphs = [];
//создание списка анимаций в функции Init
mixer = new THREE.AnimationMixer( scene );
…
//функция загрузки анимированной модели
function loadAnimatedModel(path) //где path – путь и название модели
{
    var loader = new GLTFLoader();

    loader.load( path, function ( gltf ) {
        var mesh = gltf.scene.children[ 0 ];
        var clip = gltf.animations[ 0 ];
        //установка параметров анимации (скорость воспроизведения и стартовый фрейм)
        mixer.clipAction( clip, mesh ).setDuration( 1 ).startAt( 0 ).play();

        mesh.position.set( 20, 20, -5 );   //установка позиции объекта
        mesh.rotation.y = Math.PI / 8;     //поворот модели вокруг оси Y
        mesh.scale.set( 0.2, 0.2, 0.2 );   //масштаб модели

        scene.add( mesh );                 //добавление модели в сцену
        morphs.push( mesh );
    } );
}
// воспроизведение анимаций (в функции animate)
var delta = clock.getDelta();
mixer.update( delta );
for ( var i = 0; i < morphs.length; i ++ )
{
    var morph = morphs[ i ];
}
```

Загружать модель имеет смысл только один раз. В случае если вам требуется добавить в сцену несколько объектов имеющих одинаковую модель, её можно скопировать при помощи метода:

```
object.clone();
```

Пример анимированных моделей, используемых в лабораторной работе:



Задание:

- разработать универсальную функцию загрузки объектов из файлов типа obj + mtl
- разработать универсальную функцию загрузки анимированных объектов типа json
- добавить в программу, разработанную в рамках выполнения лабораторной работы №1 10 статических моделей (деревьев, расположенных случайным образом) и 2 анимированные

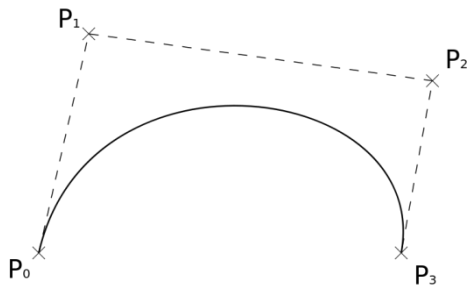
Часть 2: Перемещение объектов по траектории

Для того, чтобы организовать перемещение анимированных моделей по сложным траекториям, можно воспользоваться стандартным классом, позволяющим создавать кубические кривые Безье и получать произвольное число точек на них.

Создание кубической кривой Безье, и получение точек на ней в виде массива векторов, выглядит следующим образом:

```
var curve = new THREE.CubicBezierCurve3(
    new THREE.Vector3( -10, 0, 0 ),  //P0
    new THREE.Vector3( -5, 15, 0 ),  //P1
    new THREE.Vector3( 20, 15, 0 ),  //P2
    new THREE.Vector3( 10, 0, 0 )    //P3
);

var vertices = [];
// получение 20-ти точек на заданной кривой
vertices = curve.getPoints( 20 );
```

полученная кривая будет иметь следующий вид:



Для того, чтобы объединить несколько кривых, можно использовать метод `concat`:

```
vertices = curve1.getPoints( 20 );
vertices = vertices.concat(curve2.getPoints( 20 ));
```

После чего, можно сгенерировать кривую, проходящую через все точки траектории:

```
// создание кривой по списку точек
var path = new THREE.CatmullRomCurve3(vertices);
// является ли кривая замкнутой (зацикленной)
path.closed = true;
```

Для отображения рассчитанной кривой, можно использовать следующую конструкцию:

```
//создание геометрии из точек кривой
var geometry = new THREE.BufferGeometry().setFromPoints( vertices );
var material = new THREE.LineBasicMaterial( { color : 0xffff00 } );
```

```
//создание объекта
var curveObject = new THREE.Line( geometry, material );
scene.add(curveObject); //добавление объекта в сцену
```

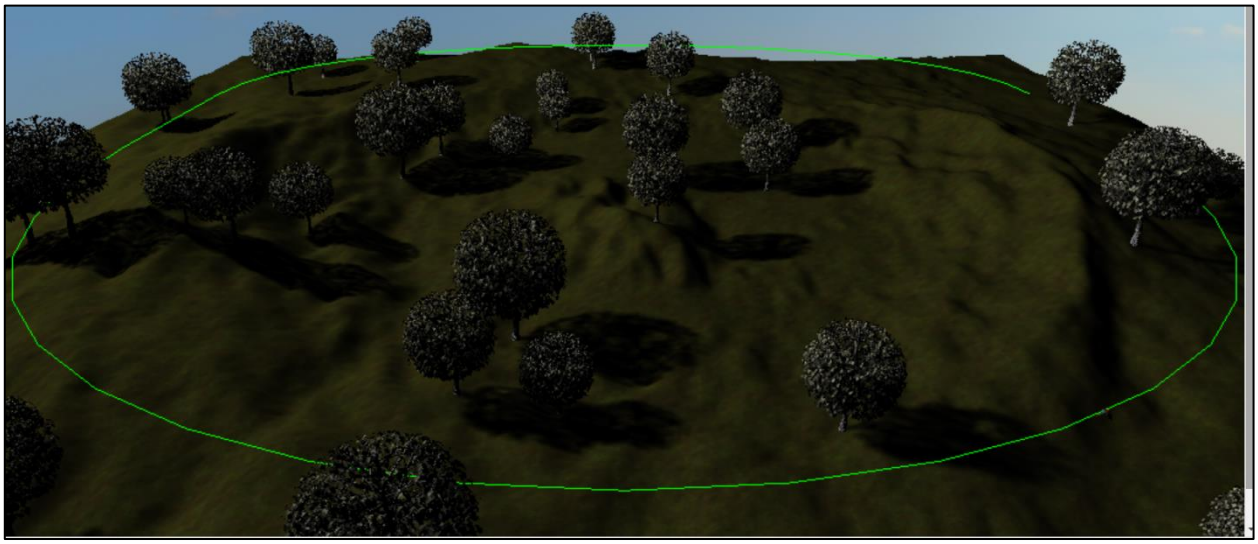
Переместить объект можно найдя точку на траектории, соответствующую прошедшему с начала движения времени, и использование этой точки в качестве позиции объекта.

Для нахождения такой точки может быть использован метод кривой `getPointAt`:

```
var pos = new THREE.Vector3();
pos.copy(path.getPointAt(t/T));
```

где t – время, прошедшее с момента начала движения по траектории, а T – время прохождения всей траектории.

Пример траектории состоящей из двух кривых:



Для примера были использованы следующие опорные точки:

P0 - (300, 20, 120), P1 – (300, 20, 25), P2 - (50, 20, 25), P3 - (50, 20, 120)

P0 - (50, 20, 170), P1 - (50, 20, 325), P2 - (300, 20, 325), P3 - (300, 20, 170)

Размер ландшафта: 350x350

Для поворота ориентированных объектов в направлении движения по траектории может быть использован метод:

```
object.lookAt(nextPoint);
```

где `nextPoint` – является следующей точкой на траектории, которую можно найти следующим образом:

```
var nextPoint = new THREE.Vector3();
nextPoint.copy(path.getPointAt((t+0.1)/T));
```

Задание:

- разработать функцию расчёта траекторий
- организовать движение анимированных моделей по независимым, цикличным траекториям
- реализовать режим слежения за моделью

Справка: слежение за объектом может быть реализовано следующим образом:

```
// установка смещения камеры относительно объекта
var relativeCameraOffset = new THREE.Vector3(0,3,-15);

var m1 = new THREE.Matrix4();
var m2 = new THREE.Matrix4();
// получение поворота объекта
m1.extractRotation(object.matrixWorld);
// получение позиции объекта
m2.extractPosition(object.matrixWorld);
m1.multiplyMatrices(m2, m1);
// получение смещения позиции камеры относительно объекта
var cameraOffset = relativeCameraOffset.applyMatrix4(m1);
// установка позиции и направления взгляда камеры
camera.position.copy(cameraOffset);
camera.lookAt(object.position );
```

Итоговое задание:

Добавить в полученное в процессе разработки программное приложение:

- сферу с текстурой неба
- расчёт теней
- модель птицы, управляемую пользователем
- переключение режима слежения между моделями, перемещающимися по траекториям и моделью, управляемой пользователем

Настройки рендера для включения режима расчёта теней:

```
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFShadowMap;
```

Настройки источника освещения:

```
//создание точечного источника освещения, параметры: цвет, интенсивность, дальность
const light = new THREE.PointLight( 0xffffff, 1, 1000 );
light.position.set( 300, 200, 128 );    //позиция источника освещения
light.castShadow = true;                //включение расчёта теней от источника освещения
scene.add( light );                     //добавление источника освещения в сцену

//настройка расчёта теней от источника освещения
light.shadow.mapSize.width = 512;   //ширина карты теней в пикселях
light.shadow.mapSize.height = 512;  //высота карты теней в пикселях
light.shadow.camera.near = 0.5;     //расстояние, ближе которого не будет теней
light.shadow.camera.far = 1500;     //расстояние, дальше которого не будет теней
```

Настройка параметров модели на приём и отбрасывание тени:

```
mesh.receiveShadow = true;
mesh.castShadow = true;
```

В случае если модель имеет иерархическую структуру, настроить её подмножества можно при помощи метода traverse:

```
object.traverse( function ( child )
{
    if ( child instanceof THREE.Mesh )
    {
        child.castShadow = true;
    }
} );
```

Настройки фильтрации текстуры, для минимизации эффекта пикселизации (может понадобиться для текстуры неба):

```
var maxAnisotropy = renderer.getMaxAnisotropy();
texture.anisotropy = maxAnisotropy;
```

Пример работы: