Лабораторная работа №2: Перемещение объектов в трёхмерном пространстве

Цель:

Целью данной лабораторной работы является получение базовых навыков расчёта позиции объектов в трёхмерном пространстве.

Справка: текстуры, используемые в лабораторной работе, можно скачать по адресу: http://planetpixelemporium.com/earth.html
Документация по классам и функциям three.js доступна по адресу: https://threejs.org/docs/index.html#api/en/core/Object3D

Часть 1: Позиционирование объектов

Помимо прочего, библиотека three.js предоставляет набор функций для создания простых моделей. Подобные модели удобно использовать для создания простых объектов и отработки приёмов программирования не связанных с графикой непосредственно.

Список стандартных объектов можно найти в документации к библиотеке three.js, в том числе по адресу: http://threejs.org/docs/ (запрос: geometry)

В данной лабораторной работе, нам понадобится только **сфера**:

```
//создание геометрии сферы
var geometry = new THREE.SphereGeometry( 10, 32, 32 );

//загрузка текстуры
var tex = loader.load( "imgs/sunmap.jpg" );
tex.minFilter = THREE.NearestFilter;

//создание материала
var material = new THREE.MeshBasicMaterial({
    map: tex,
    side: THREE.DoubleSide
});

//создание объекта
var sphere = new THREE.Mesh( geometry, material );

//размещение объекта в сцене
scene.add( sphere );
```

Параметры метода `THREE.SphereGeometry`:

- радиус сферы
- число сегментов по горизонтали
- число сегментов по вертикали

Строчка `tex.minFilter = THREE.NearestFilter,` означает, что используемая текстура будет автоматически масштабирована до ближайшей степени двух. В остальном, работа со сферой, ни чем не отличается от работы с пользовательским объектом из лабораторной работы номер один.

Задать позицию для объекта можно при помощи свойства `position`, одним из следующих способов:

```
sphere.position.x = x;
sphere.position.y = y;
sphere.position.z = z;

sphere.position.set(x, y, z);

var pos = new THREE.Vector3(x, y, z);
sphere.position.copy(pos);
```

Помимо этого, для объекта может быть указан его **поворот**, при помощи свойства: rotation. Поворот указывается в виде углов Эйлера, заданных в радианах.

Задание:

реализовать набор функций, позволяющий создавать объекты типа: “Солнце”, ”Планета”, ”Звёздное небо”. Все объекты должны содержать в себе информацию о текстуре, размере объекта и его начальной позиции. Помимо этого, объекты типа “Планета”, должны содержать информацию о собственном вращении и перемещении.

При помощи разработанных функций, отобразить на экран монитора неподвижную модель солнечной системы. Желательно соблюсти относительный масштаб планет и их удаление от солнца.

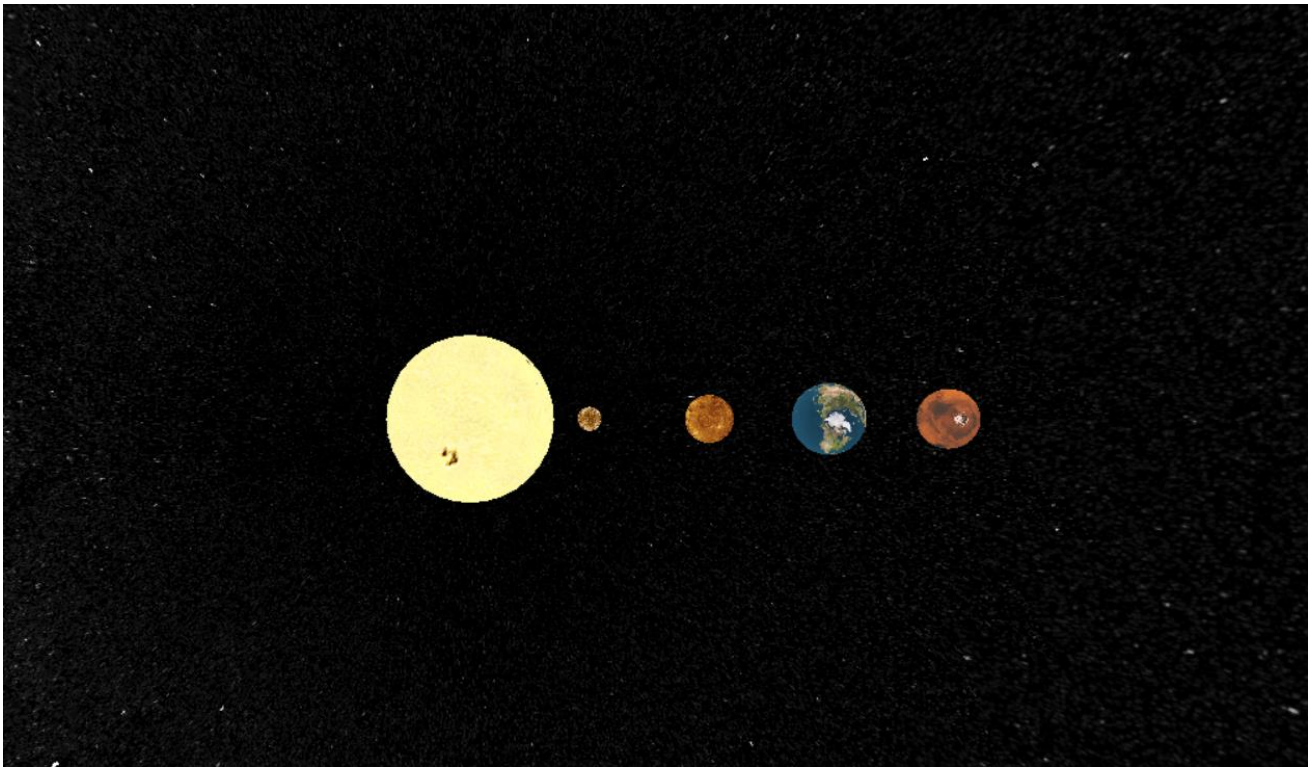
Справка: работа с массивами в JavaScript:

```
var planets = [];                          //создание
planets.push(planet);                      //добавление элемента
for (var i = 0; i < planets.length; i++)   //перебор
```

Создание класса в JavaScript:

```
var planet = {};              //создание
planet.planet = sphere;       //добавление поля planet
planet.pos = pos;             //добавление поля pos
```

Пример (текстура звёздного неба наложена на большую сферу, внутри которой находится сцена):



Часть 2: Перемещение объектов

Очевидным способом перемещения объектов в трёхмерном пространстве, является аналитический расчёт текущей позиции объекта и размещении его при помощи свойства, рассмотренного в предыдущей части.

Например, движение по окружности с центром в произвольной точке можно записать через параметрическое уравнение окружности:

```
pos.x = center.x + r * Math.cos(a);
pos.z = center.z + r * Math.sin(a);
```

где `pos` – точка на окружности радиуса r с центром в точке `center`.

Вращение объекта можно осуществить, задав поворот вокруг осей XYZ при помощи углов Эйлера:

```
object.rotation.x = angleX;
object.rotation.y = angleY;
object.rotation.z = angleZ;
```

Другой вариант, использование преобразований поворота и переноса:

```
//перенос вдоль одной из локальных координатных осей
object.translateX(distance);
object.translateY(distance);
object.translateZ(distance);

//вектор axis задаёт ось Y
var axis = new THREE.Vector3(0, 1, 0);

//перенос вдоль оси, заданной вектором в локальных координатах
object.translateOnAxis(axis, distance);

//поворот вокруг оси, заданной в локальных координатах
object.rotateOnAxis(axis, angle);
```

При этом, получить текущую позицию объекта можно при помощи следующих методов:

```
//получение матрицы позиции из матрицы объекта
var m = new THREE.Matrix4();
m.copyPosition(object.matrix);

//получение позиции из матрицы позиции
var pos = new THREE.Vector3(0, 0, 0);
pos.setFromMatrixPosition(m);
```

Примечание: получить позицию в вектор можно напрямую из матрицы объекта.

Третий вариант, расчёт “матрицы мира” объекта при помощи комбинирования преобразований:

```
//создание набора матриц
var m = new THREE.Matrix4();
var m1 = new THREE.Matrix4();
var m2 = new THREE.Matrix4();

//создание матрицы поворота (вокруг оси Y) в m1 и матрицы перемещения в m2
m1.makeRotationY( angle );
m2.setPosition(new THREE.Vector3(x, y, z));
```

```
//запись результата перемножения m1 и m2 в m
m.multiplyMatrices( m1, m2 );

//установка m в качестве матрицы преобразований объекта object
object.matrix = m;
object.matrixAutoUpdate = false;
```

В рамках выполнения данной лабораторной работы, вам могут понадобиться следующие методы класса THREE.Matrix4:

```
Matrix.identity();          //очистка данных
Matrix.multiply( mat );     //умножение матрицы Matrix на матрицу mat
//создание матрицы переноса в точку xyz
Matrix.makeTranslation( x, y, z );
//создание матрицы поворота вокруг оси axis на угол angle (в радианах)
Matrix.makeRotationAxis( axis, angle );
```

Задание: реализовать функцию, осуществляющую перемещение планет по круговым траекториям одним из выше указанных методов. Реализовать функцию отрисовки орбит планет пунктирной линией.

Справка: для того что бы скорость перемещения объектов не зависела от производительности компьютера, рекомендуется использовать функцию получения времени, прошедшего с предыдущего вызова этой функции:

```
var clock = new THREE.Clock();
var delta = clock.getDelta();
//возвращает время, прошедшее с момента предыдущего вызова этой функции
```

Для отрисовки пунктирной линии, рекомендуется использовать стандартный примитив:

```
//создание материала для пунктирной линии
var dashed_material = new THREE.LineDashedMaterial( {
        color: 0xffff00,  //цвет линии
        dashSize: 2,      //размер сегмента
        gapSize: 2,       //величина отступа между сегментами
    } );

var points = [];    //массив для хранения координат сегментов

points.push( new THREE.Vector3( 0, 0, 0 ) );   //начало линии
points.push( new THREE.Vector3( 10, 0, 10 ) ); //завершение линии

var geometry = new THREE.BufferGeometry().setFromPoints( points ); //создание геометрии

var line = new THREE.Line( geometry, dashed_material ); //создание модели
line.computeLineDistances(); //вычисление дистанции между сегментами

scene.add(line); //добавление модели в сцену
```

Итоговое задание:

Разработать приложение, моделирующее и визуализирующее упрощённую модель солнечной системы. Модель должна включать следующие объекты:

- солнце
- карта звёздного неба
- меркурий
- венера
- земля и луна

В качестве траекторий движения планет и луны, можно использовать окружность.

Расстояния между планетами, их скорость вращения вокруг собственной оси и солнца должны отражать реальные отношения размеров, расстояний и скоростей. (Марс меньше Земли, Меркурий меньше Марса и т.д.)

Реализовать режим слежения за планетой для камеры. По нажатию на клавиши 1 – 4, фокус камеры должен смещаться вслед за позицией соответствующей номеру планеты. По нажатию кнопки 0, должен включаться “общий” вид на модель. По нажатию “стрелок”, камера должна поворачиваться вокруг планеты.

Справка: для обработки нажатия клавиш, вам понадобится библиотека THREEx.KeyboardState.js, доступная по адресу: https://github.com/jeromeetienne/threex.keyboardstate

```
var keyboard = new THREEx.KeyboardState();
if (keyboard.pressed("0")) {}
```

Перемещение камеры осуществляется, так же, как и перемещение трёхмерных объектов.
Для использования карты высот, необходимо задать соответствующие параметры материала:

```
//загрузка карты высот
var bump = loader.load( bump_texture_name );

//назначение карты и масштабирования высот
var material = new THREE.MeshPhongMaterial({
    map: tex,
    bumpMap: bump,
    bumpScale: 0.05,
    side: THREE.DoubleSide
});
```

Добавление карты бликов происходит аналогичным образом:

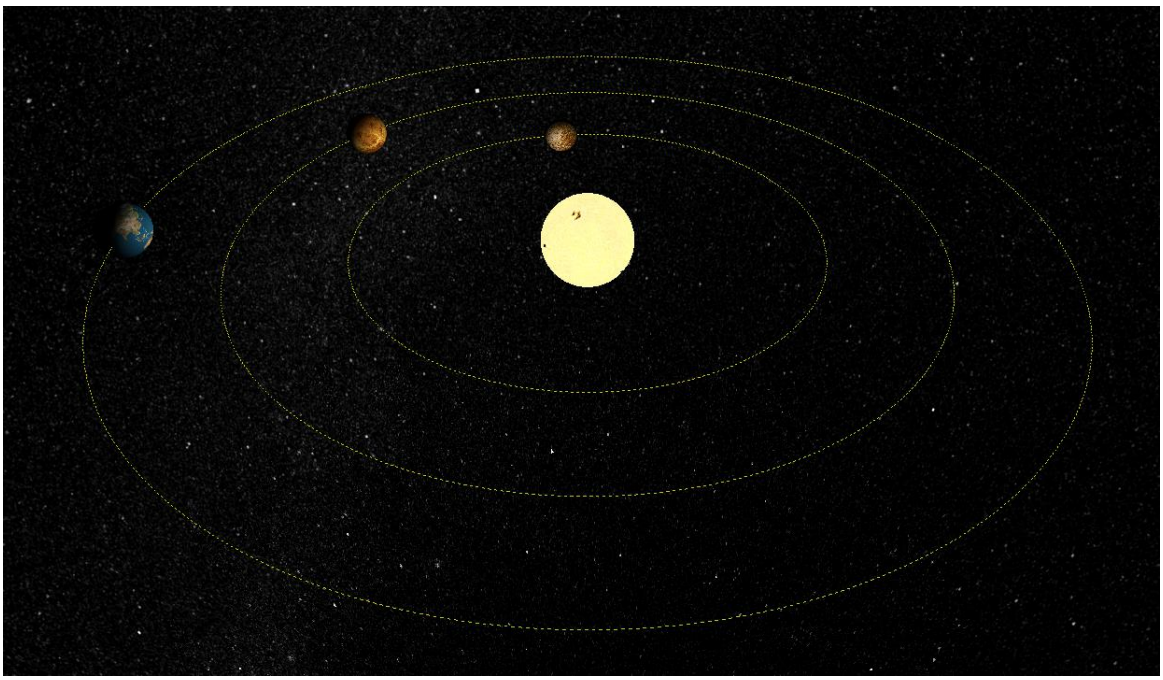
```
//загрузка карты бликов
var specMap = loader.load( spec_texture_name );

//назначение карты и цвета бликов
var material = new THREE.MeshPhongMaterial({
    map: tex,
    specularMap: specMap,
    specular: new THREE.Color('grey'),
    side: THREE.DoubleSide
});
```

Функция создания облачной сферы:

```
function createEarthCloud()
{
    // create destination canvas
    var canvasResult    = document.createElement('canvas');
    canvasResult.width  = 1024;
    canvasResult.height = 512;
    var contextResult   = canvasResult.getContext('2d');

    // load earthcloudmap
    var imageMap    = new Image();
    imageMap.addEventListener("load", function()
    {

      // create dataMap ImageData for earthcloudmap
      var canvasMap = document.createElement('canvas');
      canvasMap.width = imageMap.width;
      canvasMap.height = imageMap.height;
      var contextMap = canvasMap.getContext('2d');
      contextMap.drawImage(imageMap, 0, 0);
      var dataMap = contextMap.getImageData(0, 0, canvasMap.width, canvasMap.height);

      // load earthcloudmaptrans
      var imageTrans    = new Image();
      imageTrans.addEventListener("load", function()
      {
          // create dataTrans ImageData for earthcloudmaptrans
          var canvasTrans       = document.createElement('canvas');
          canvasTrans.width = imageTrans.width;
          canvasTrans.height    = imageTrans.height;
          var contextTrans  = canvasTrans.getContext('2d');
          contextTrans.drawImage(imageTrans, 0, 0);
          var dataTrans     = contextTrans.getImageData(0, 0, canvasTrans.width,
canvasTrans.height);
          // merge dataMap + dataTrans into dataResult
          var dataResult = contextMap.createImageData(canvasMap.width, canvasMap.height);
          for(var y = 0, offset = 0; y < imageMap.height; y++)
          for(var x = 0; x < imageMap.width; x++, offset += 4)
          {
              dataResult.data[offset+0] = dataMap.data[offset+0];
              dataResult.data[offset+1] = dataMap.data[offset+1];
              dataResult.data[offset+2] = dataMap.data[offset+2];
              dataResult.data[offset+3] = 255-dataTrans.data[offset+0];
          }
        // update texture with result
        contextResult.putImageData(dataResult,0,0)
        material.map.needsUpdate = true;
      });

      imageTrans.src = 'images/earthcloudmaptrans.jpg';
    }, false);

    imageMap.src = 'images/earthcloudmap.jpg';
```

```
    var geometry = new THREE.SphereGeometry(0.51, 32, 32);
    var material = new THREE.MeshPhongMaterial({
        map: new THREE.Texture(canvasResult),
        side: THREE.DoubleSide,
        transparent: true,
        opacity: 0.8,
    });

    var mesh = new THREE.Mesh(geometry, material);
    return mesh;
}
```

Общий вид:



Режим слежения: