Программирование графических приложений

Тема 12
Материалы и карты текстур

12.1. Материалы в Three.js
12.2. Параметры материалов в Three.js
12.3. Параметры карт текстур в Three.js
12.4. Применение разных материалов к одному объекту
12.5. Применение нескольких материалов к одному объекту
12.6. Добавление теней

Контрольные вопросы

Цель изучения темы. Изучение методов работы с материалами и картами текстур в WebGL с использованием библиотеки Three.js.

12.1. Материалы в Three.js

Материалы - это объекты Three.js, задающие правила и настройки, по которым будут закрашиваться грани объектов сцены. В WebGL для определения цвета каждого пикселя грани используются специальные шейдер-программы, написанные на языке GLSL, которые выполняются на стороне видеокарты.

Библиотека Three.js содержит несколько заготовленных шейдеров, которые компонуются в зависимости от типа и настройки материала, примененного к объекту.

В предыдущих темах уже использовались различные типы материалов для закрашивания полигональных моделей, в частности:

- MeshBasicMaterial - простой материал, не учитывающий освещение;
- MeshPhongMaterial - выполняет закрашивание поверхности методом Фонга;
- MeshLambertMaterial - выполняет закрашивание поверхности методом Ламберта.

Создаются они соответственно:

```
var basic_material = new THREE.MeshBasicMaterial();
var phong_material = new THREE.MeshPhongMaterial();
var lambert_material = new THREE.MeshLambertMaterial();
```

В скобках можно указать параметры материала в виде {ключ:значение}, например:

```
var basic_material = new THREE.MeshBasicMaterial ({color:0xFF0000, shininess:10});
```

12.2. Параметры материалов в Three.js

При задании материалов в Three.js можно передать в конструктор следующие параметры материала в виде {ключ:значение}:

color (цвет)

Задает базовый цвет материала в формате 0xrrggbb. По умолчанию принимает значение 0xFFFFFF (белый цвет).

Пример:

```
{ color:0xFF0000 }
```

shading (закрашивание)

Определяет способ закрашивания граней. Устанавливается значением констант THREE.SmoothShading (плавное закрашивание) и THREE.FlatShading (плоское закрашивание).

Константа THREE.SmoothShading (плавное закрашивание) устанавливается по умолчанию. При отрисовке сглаживает переходы между гранями.

Пример (полный код в файлах **ex12_001.html**, **ex12_001.js**):

```
{ shading: THREE.SmoothShading }
```

Результат:

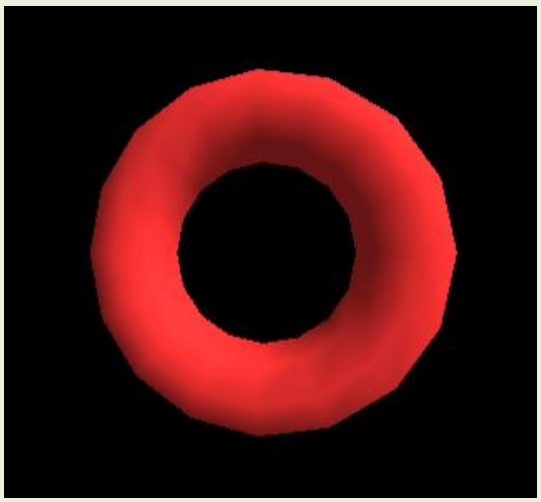

Рис. 12.1

Константа THREE.FlatShading (плоское закрашивание) закрашивает, расчитывая освещение поверхности по нормали к граням. В результате видны резкие переходы между ними.

Пример (полный код в файлах **ex12_002.html**, **ex12_002.js**):

```
{ shading: THREE.FlatShading }
```

Результат:

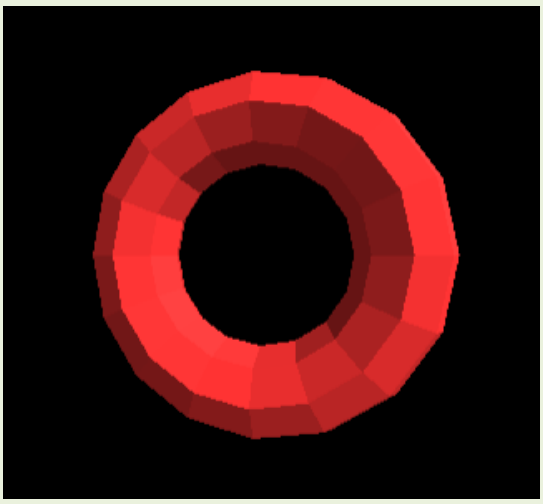

Рис. 12.2

specular (отражение)

Используется в MeshPhongMaterial**;** задается в формате 0xrrggbb (0x111111 по умолчанию). Определяет оттенок бликов.

Пример:

```
{specular:0x171717}
```

shininess (блик)

Используется в MeshPhongMaterial, определяет четкость блика. Чем выше значение, тем поверхность более глянцевая. Значение по умолчанию – 10.

Пример:

```
{ shininess:10}
```

Пример (полный код в файлах **ex12_003.html**, **ex12_003.js**) включает установку отражения и блика. Программа выводит в canvas движки, позволяющие пользователю установить значение коэффициентов цветовых компонент отражения и коэффициент блика.

Результат:

Рис. 12.3

metal (металл)

Принимает значения **true** или **false** (по умолчанию). Придает материалу больше схожести с металлом, за счет увеличения контраста.

Пример:

```
{ metal: true }
```

emissive (излучение)

Задает цвет, который излучает объект; по умолчанию - 0x000000. Используется в MeshPhongMaterial и MeshLambertMaterial, фактически прибавляет этот цвет к расчетному.

Пример:

```
{ emissive: 0x111111}
```

wireframe (каркас)

Задается булевым значение true или false (по умолчанию).

При включенном состоянии отображает объект в виде каркасной сетки. При этом отображаются только ребра, а грани остаются пустыми.

Пример:

```
{ wireframe: true }
```

fog (туман)

Определяет влияние тумана при отрисовке, по умолчанию true. Если установить false - влияние тумана не будет распространяться на объекты с этим материалом.

Пример:

```
{ fog: false }
```

12.3. Параметры карт текстур в Three.js

Некоторые параметры материалов в Three.js требуют для передачи в качестве значения заранее подготовленный объект - текстуру.

Текстура - это растровое изображение, которое проецируется на грани трехмерного объекта по определенным правилам. Можно представить каждый пиксель изображения в виде

вектора, содержащего значения RGB (аналогично тому, как вектор координат содержит значения XYZ), а само изображение как векторную карту.

Следующие параметры материала передаются в виде карт текстур {ключ:значение}:

map

Самый простой способ проецирования, где цвет пикселя текстуры становится цветом фрагмента грани.

Для примера возьмем изображение 128x128px – файл **texture.jpg**:

Рис. 12.4

Создадим из него текстуру:

```
var texture = new THREE.ImageUtils.loadTexture('texture.jpg');
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.RepeatWrapping;
texture.repeat.set( 2, 2 );
sphere_material = new THREE.MeshPhongMaterial ({map:texture});
```

Пример (полный код в файлах **ex12_004.html**, **ex12_004.js**) демонстрирует применение карты текстур map – строит текстурированный вращающийся шар.

Результат:

Рис. 12.5

normalMap (карта нормалей)

В материалах, у которых цвет фрагментов граней зависит от нормали в этой точке, мы можем задать значение нормали в ней с помощью normalMap

Для оптимизации сцены обычно делают 2 модели: высокополигональную (hi-poly) с высокой детализацией и большим количеством полигонов и низкополигональную (lo-poly), которая сможет быстро обрабатываться в режиме реального времени.

Затем с высокополигональной модели снимается карта нормалей и накладывается на низкополигональную. Таким образом мы получаем отражение света от lo-poly модели как от hi-poly. Возьмём изображение **normal.jpg**:

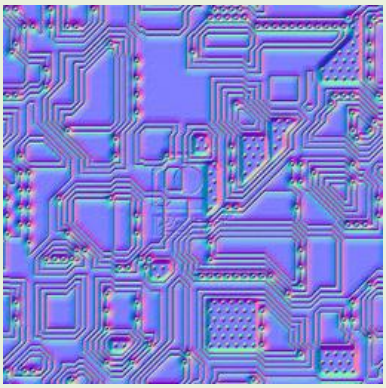

Рис. 12.6

Создадим текстуру:

```
var normalmap = new THREE.ImageUtils.loadTexture('normal.jpg');
normalmap.wrapS = THREE.RepeatWrapping;
normalmap.wrapT = THREE.RepeatWrapping;
normalmap.repeat.set( 4, 4 );
sphere_material = new THREE.MeshPhongMaterial({  map:normalmap });
```

Пример (полный код в файлах **ex12_005.html**, **ex12_005.js**) демонстрирует применение карты нормалей к шару.

Результат:

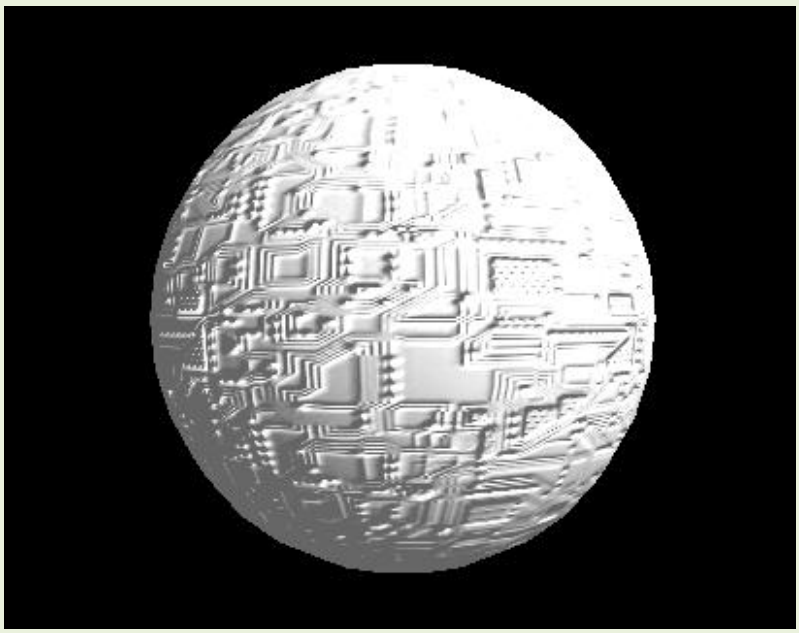

Рис. 12.7

lightMap (карта освещения)

Накладывает карту освещения на объект, при этом белый цвет карты (0xFFFFFF) дает текущее освещение, а черный (0x000000) - полное отсутствие освещения на фрагменте полигона. Использование lightMap имеет смысл при наложении теней на статические объекты взамен динамического просчета.

Карта освещения требует собственные текстурные координаты. Поэтому в примере ниже мы укажем дополнительные текстурные координаты для карты освещения, эквивалентные координатам обычной текстуры.

Изображение обычной текстуры **texture.jpg**:

Рис. 12.8

Изображение карты освещения **lightmap.jpg**:

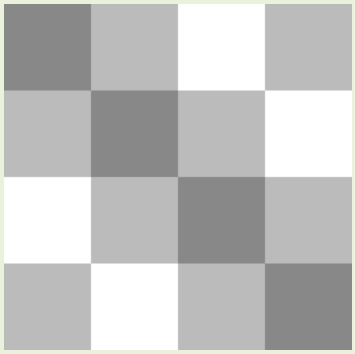

Рис. 12.9

Используем эти текстуры:

```
  var texture = new THREE.ImageUtils.loadTexture('texture.jpg');
  texture.wrapS = THREE.RepeatWrapping;
  texture.wrapT = THREE.RepeatWrapping;
  texture.repeat.set( 4, 4 );
  var lightmap = new THREE.ImageUtils.loadTexture('lightmap.jpg');
  lightmap.wrapS = THREE.RepeatWrapping;
  lightmap.wrapT = THREE.RepeatWrapping;
  lightmap.repeat.set( 4, 4 );
  var sphere_material = new THREE.MeshPhongMaterial({
    map:texture,
    lightMap:lightmap
  });
var sphere_geometry = new THREE.SphereGeometry(1, 16, 16);
//указываем текстурные координаты для lightmap
  sphere_geometry.faceVertexUvs[1] = sphere_geometry.faceVertexUvs[0];
```

```
sphere = new THREE.Mesh( sphere_geometry, sphere_material );
scene.add( sphere );
```

Пример (полный код в файлах **ex12_006.html**, **ex12_006.js**) демонстрирует применение карты освещения к текстурированному объекту.

Результат:

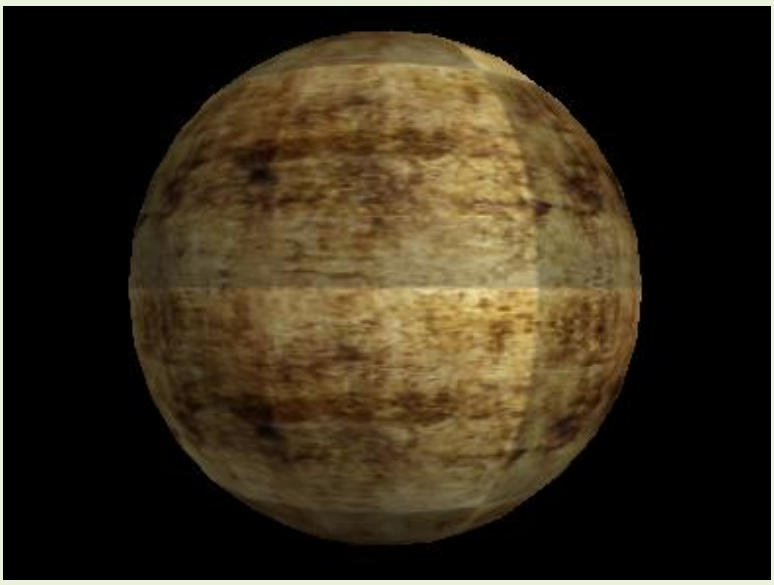

Рис. 12.10

bumpMap (карта высот)

Это карта высот, она очень похожа по действию на normalMap, но задает не значение нормалей, а перепады высот на поверхности (от которых берутся нормали). Обработка bumpMap более ресурсоемка, поэтому предпочтительнее использовать normalMap.

Работает в паре с коэффициентом **bumpScale** (по умолчанию равно 1).

Возьмем карту **bumpmap.jpg**:

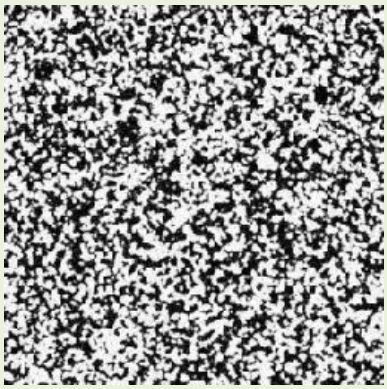

Рис. 12.11

```
var bumpmap = new THREE.ImageUtils.loadTexture('bumpmap.jpg');
bumpmap.wrapS = THREE.RepeatWrapping;
bumpmap.wrapT = THREE.RepeatWrapping;
bumpmap.repeat.set( 4, 4 );
var sphere_material = new THREE.MeshPhongMaterial({
  bumpMap:bumpmap,
  bumpScale:0.01
});
```

Пример (полный код в файлах **ex12_007.html**, **ex12_007.js**) демонстрирует применение карты высот к объекту.

Результат:

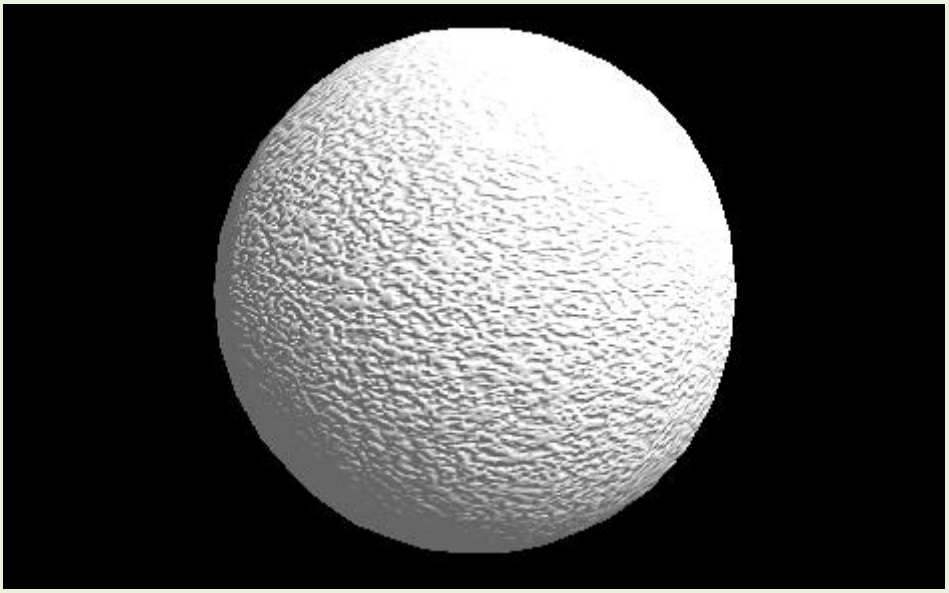

Рис. 12.12

alphaMap (карта прозрачности)

Определяет прозрачность фрагментов полигонов.

Для работы карты прозрачности необходимо включить свойство материала **transparent** (по умолчанию **false**).

Карта прозрачности работает по зеленому каналу текстуры. FF - полностью непрозрачный, 00 - полностью прозрачный.

Можно использовать параметр **alphaTest**. Он принимает значение от 0 (по умолчанию) до 1. При этом не будут отрисовываться фрагменты с прозрачностью меньше величины **alphaTest**. Возьмем карту **alphamap.jpg**:

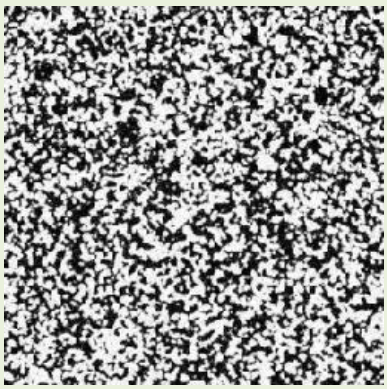

Рис. 12.13

Для лучшего эфекта включим отрисовку граней с двух сторон:

```
var alphamap = new THREE.ImageUtils.loadTexture('alphamap.jpg');
alphamap.wrapS = THREE.RepeatWrapping;
alphamap.wrapT = THREE.RepeatWrapping;
alphamap.repeat.set( 1, 1 );
var sphere_material = new THREE.MeshPhongMaterial({
 transparent:true,
```

```
 color:0xFF0000,
 alphaMap:alphamap,
 alphaTest:0.5,
 side: THREE.DoubleSide
});
var sphere_geometry = new THREE.SphereGeometry(1, 16, 16);
sphere = new THREE.Mesh( sphere_geometry, sphere_material );
```

Пример (полный код в файлах **ex12_008.html**, **ex12_008.js**) демонстрирует применение карты прозрачности к объекту.

Результат:

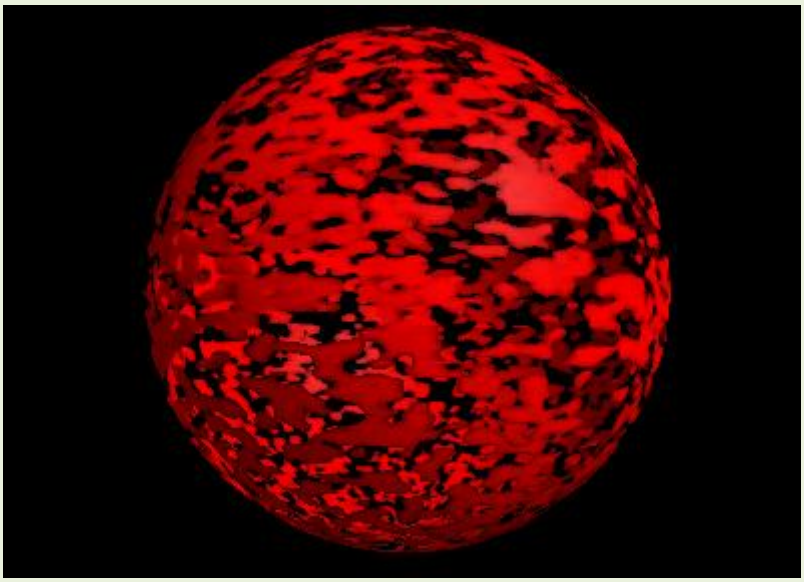

Рис. 12.14

envMap (карта окружения)

Это карта Environment map - применяется для эмуляции отражения на поверхности объекта. Просчет отражения - математически очень дорогая операция, поэтому в некоторых случаях имеет смысл наложить готовую картинку. Карта **envMap** принимает на вход кубическую текстуру, содержущую изображение окружающего пространства.

Возьмем 6 изображений, образующих безшовный куб пространства (**px.png, nx.png, py.png, ny.png, pz.png, nz.png**):

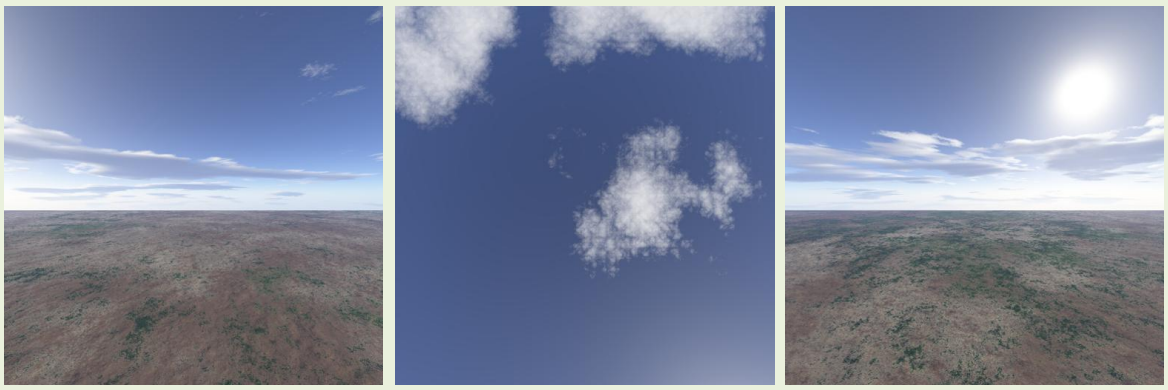

Рис. 12.15

Расположение изображений на резвертке куба:

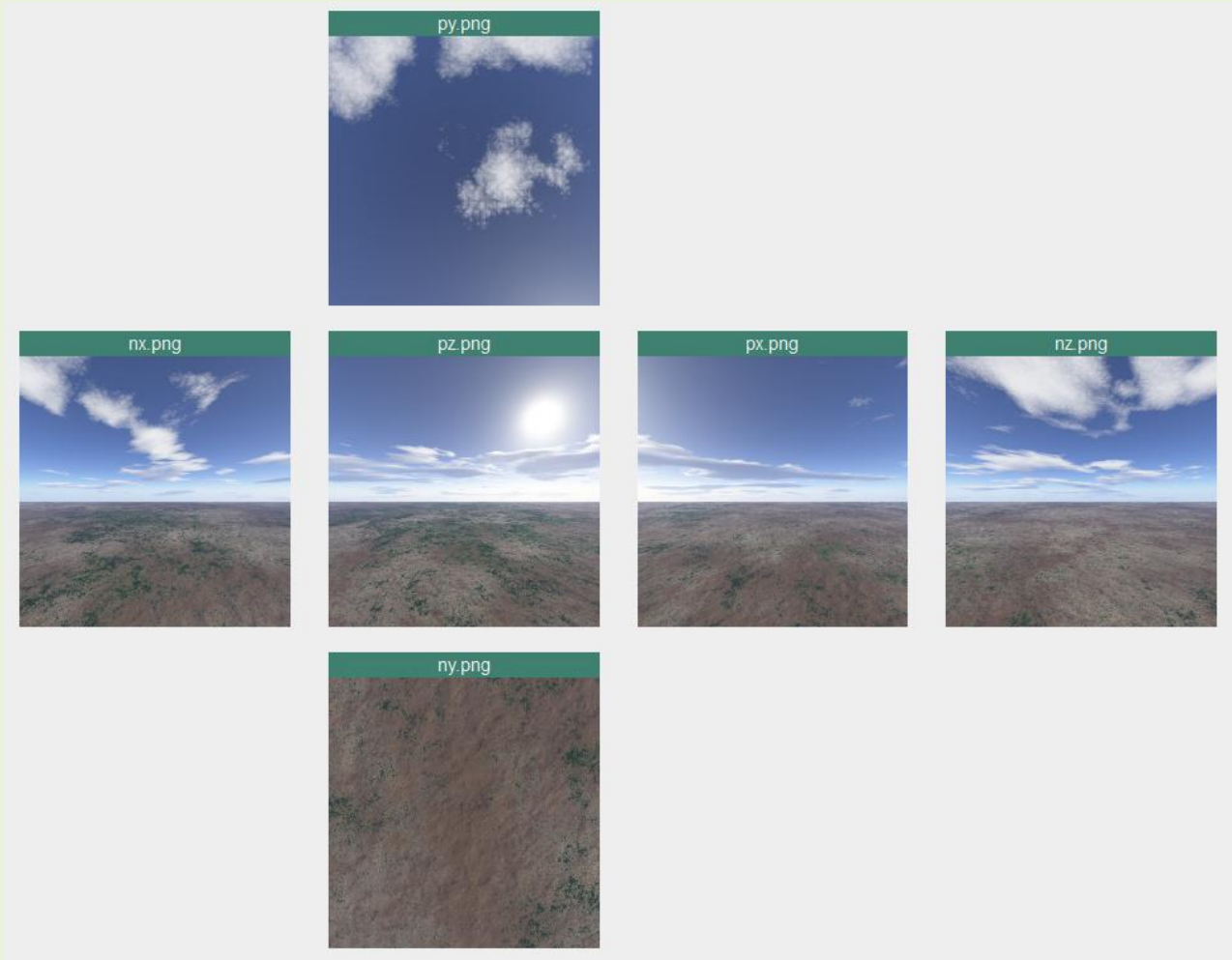

Рис. 12.16

```
var cubemap = new THREE.ImageUtils.loadTextureCube([
 'px.png', 'nx.png', 'py.png', 'ny.png', 'pz.png', 'nz.png'
]);
cubemap.mapping = THREE.CubeReflectionMapping;
var sphere_material = new THREE.MeshPhongMaterial({ envMap: cubemap });
var sphere_geometry = new THREE.SphereGeometry(1, 16, 16);
sphere = new THREE.Mesh( sphere_geometry, sphere_material );
```

Пример (полный код в файлах **ex12_009.html**, **ex12_009.js**) демонстрирует применение карты окружения к объекту.

Результат:

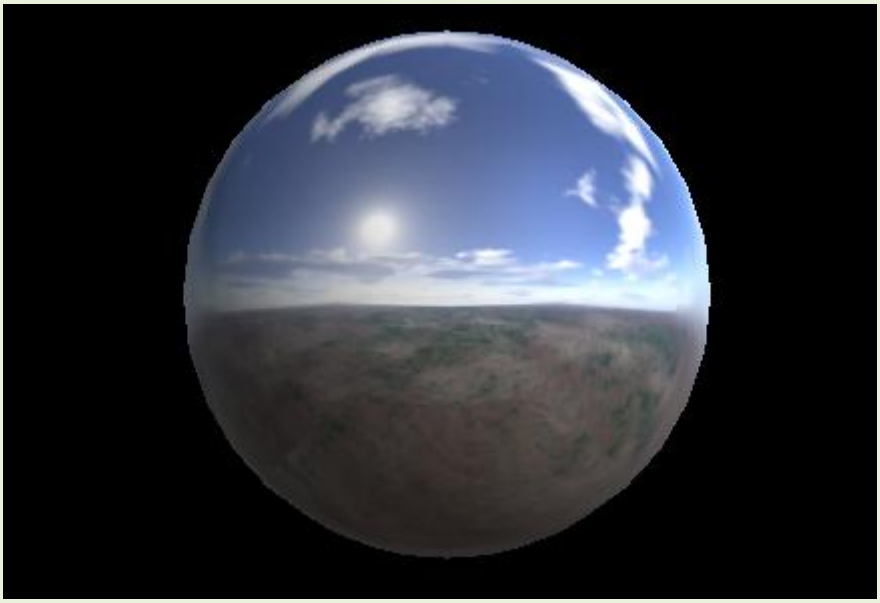

Рис. 12.17

12.4. Применение разных материалов к одному объекту

Если необходимо назначить граням одного объекта разные материалы, следует создать коллекцию THREE.MeshFaceMaterial([material1,material2,...]).

В качестве аргумента в конструктор передается массив материалов. Затем для каждой грани объекта необходимо выставить индекс материала в поле geometry.faces[].materialIndex (по умолчанию он равен 0 и если не проставить индексы, ко всем граням применится первый материал коллекции).

Файл **ex12_10.html:**

```
<!DOCTYPE HTML>
<html>
 <head>
  <script src="three.min.js"></script>
 </head>
 <body>
 </body>
  <script src=" ex12_10.js "></script>
</html>
```

Файл **ex12_10.js:**

```
var scene,camera,renderer,cylinder;
init();
animate();
function init(){
    renderer = new THREE.WebGLRenderer();
    renderer.setSize(800,600);
    document.body.appendChild( renderer.domElement );
 camera = new THREE.PerspectiveCamera( 70, 1.5, 1, 1000 );
 camera.translateZ(5);
```

```
camera.lookAt(new THREE.Vector3(  0, 0, 0 ));
scene = new THREE.Scene();
var cylinder_materials = new THREE.MeshFaceMaterial([
        new THREE.MeshPhongMaterial({color:0xff0000, metal:true,
        speculat:0x999999, emissive:0x700000, shines:90}),
        new THREE.MeshPhongMaterial({color:0x00ff00, metal:true,
        speculat:0x999999, emissive:0x007000, shines:90}),
        new THREE.MeshPhongMaterial({color:0x0000ff, metal:true,
        speculat:0x999999, emissive:0x000070, shines:90})
]);
var cylinder_geometry = new THREE.CylinderGeometry( 1, 1, 5, 15, 5 );
for (var i=0;i<(cylinder_geometry.faces.length-30)/2;i++){
        cylinder_geometry.faces[2*i].materialIndex=i%3;
        cylinder_geometry.faces[2*i+1].materialIndex=i%3;
}
cylinder = new THREE.Mesh( cylinder_geometry, cylinder_materials );
var light1 = new THREE.AmbientLight( 0xffffff );
var light2 = new THREE.DirectionalLight( 0xffffff, 1 );
light2.position.set(3, 3, 3);
light2.lookAt(new THREE.Vector3(  0, 0, 0 ));
scene.add( light1, light2, cylinder );
animate();
}
function animate(){
 cylinder.rotateX( Math.PI/360 );
   renderer.render( scene, camera );
   requestAnimationFrame( animate );
}
```

Результат:

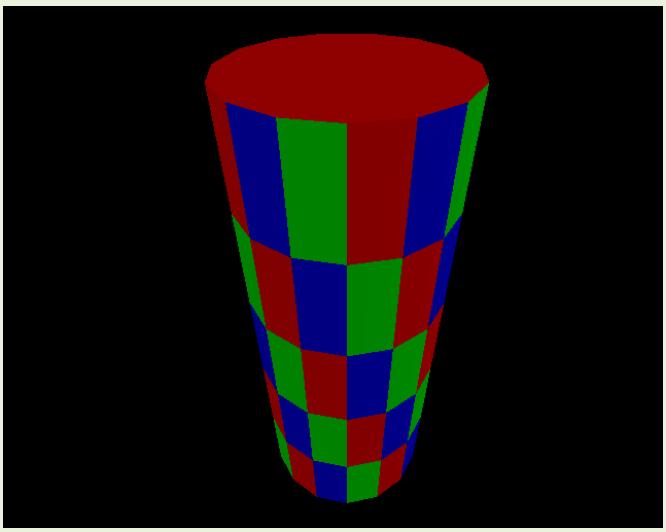

Рис. 12.18

12.5. Применение нескольких материалов к одному объекту

Бывают ситуации, когда необходимо назначить несколько материалов для каждой грани объекта. Например, мы хотим поверх обычного цвета видеть каркасную сетку. Если включить параметр {wireframe:true}, то грани перестанут отрисовываться, а начнут отрисовываться ребра. Можно воспользоваться методом

THREE.SceneUtils.createMultiMaterialObject(геометрия, [материал1, материал2,....])

Этот метод создаст экземпляр Object3D и назначит ему потомками несколько meshs (по колличеству материалов), имеющих общую геометрию, но различные материалы.

Файл **ex12_11.html:**

```
<!DOCTYPE HTML>
<html>
 <head><script src="three.min.js"></script></head>
 <body></body>
 <script src=" ex12_11.js "></script>
</html>
```

Файл **ex12_11.js:**

```
var scene,camera,renderer,cylinder;
init();
animate();
function init(){
    renderer = new THREE.WebGLRenderer();
    renderer.setSize(800,600);
    document.body.appendChild( renderer.domElement );

 camera = new THREE.PerspectiveCamera( 70, 1.5, 1, 1000 );
 camera.translateZ(3);
 camera.lookAt(new THREE.Vector3(  0, 0, 0 ));
 scene = new THREE.Scene();
 var obj_material = [
            new THREE.MeshPhongMaterial( { wireframe: false, color: 0xff0000 } ),
            new THREE.MeshBasicMaterial( { wireframe: true, color: 0xffffff } )
 ];
 var box_geometry = new THREE.BoxGeometry( 1, 1, 1);
 box = THREE.SceneUtils.createMultiMaterialObject( box_geometry, obj_material );
 var light1 = new THREE.AmbientLight( 0x666666 );
 var light2 = new THREE.DirectionalLight( 0xffffff, 0.9 );
 light2.position.set(5, 5, 5);
 light2.lookAt(new THREE.Vector3(  0, 0, 0 ));
 scene.add( light1, light2, box );
 animate();
}
function animate(){
 cylinder.rotateX( Math.PI/360 );
    renderer.render( scene, camera );
    requestAnimationFrame( animate );
}
```

Результат:

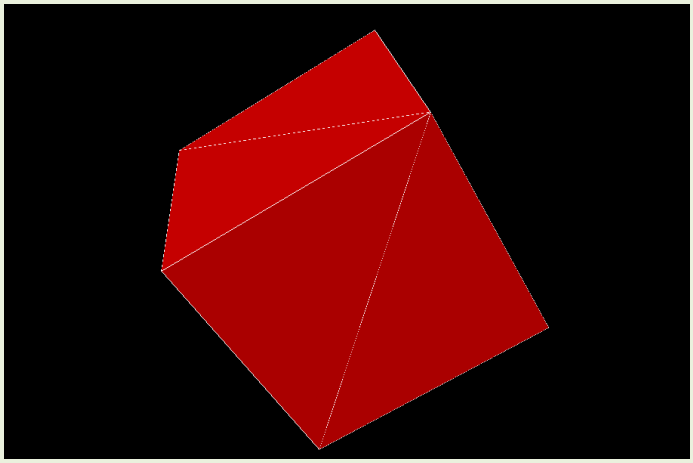

Рис. 12.19

12.6. Добавление теней

Для наложения теней на сцене нужно предварительно включить их в рендере:

```
renderer.shadowMapEnabled = true;
```

Далее нужно включить образование теней у источника света:

```
light.castShadow = true;
```

Это же свойство castShadow нужно включить у тех тел, от которых требуется отбрасывание теней.

Рассмотрим пример. Пусть дана сфера:

```
var material = newTHREE.MeshLambertMaterial( { color: 0x33CCFF } );
var geometry = new THREE.SphereGeometry(20, 50, 50);
var sphere1 = new THREE.Mesh( geometry, material );
sphere1.position.set( 0, 60, 0 );
sphere1.castShadow = true;
scene.add ( sphere1 );
```

Расположим под ней две плоскости:

```
var planeMaterial1 = new THREE.MeshLambertMaterial(
    { color: 0x9999ff, side: THREE.DoubleSide });
var planGeo1 = new THREE.PlaneGeometry( 200, 200, 1, 1);
var plane1 = new THREE.Mesh(planGeo1, planeMaterial1);
plane1.rotation.x = Math.PI/2;
scene.add( plane1 );
var planeMaterial2 = new THREE.MeshLambertMaterial(
    { color: 0xff00ff, side: THREE.DoubleSide });
var planGeo2 = new THREE.PlaneGeometry( 300, 300, 1, 1);
var plane2 = new THREE.Mesh(planGeo2, planeMaterial2);
```

Но как видим на рисунке, теней пока нет:

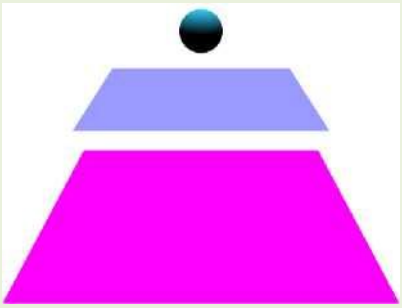

Рис. 12.20

Чтобы плоскости воспринимали тени, нужно включить у них свойство receiveShadow:

```
plane2.position.set (0,-120,0);
plane2.rotation.x = Math.PI/2;
scene.add ( plane2 );
plane1.receiveShadow = true;
plane2.receiveShadow = true;
```

Результат:

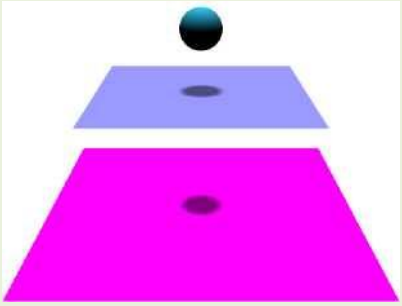

Рис. 12.21

Чтобы тень правильно отбрасывалась не только на верхнюю плоскость, но и на нижнюю, необходимо включить свойство castShadow у первой плоскости:

```
plane1.castShadow = true;
```

Результат (коды – в файлах **ex12_12.html**, **ex12_12.js)**:

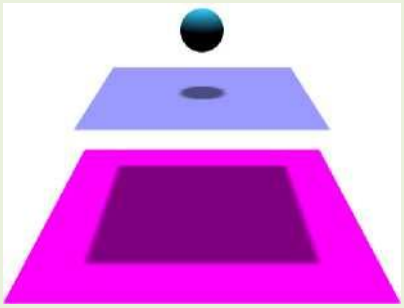

Рис. 12.22

Контрольные вопросы

1. Параметры материалов.
2. Параметры карт текстур.
3. Применение нескольких материалов к объектам.