**Uv坐标系**

纹理的坐标是uv坐标系，坐标的宽度合高度都是1



**几何体的uv坐标**

给几何体贴纹理我们需要设置几何体每个顶点对应的uv坐标，如下

const geometry = new THREE.BufferGeometry();

// 几何体顶点

let points = [

    -50, -50, 0,

    -50, 50, 0,

    50, 50, 0,

    -50, -50, 0,

    50, 50, 0,

    50, -50, 0

];

geometry.attributes.position = new THREE.Float32BufferAttribute(points, 3)

// 几何体顶点对应的uv坐标

let uvs = [

    0, 0,

    0, 1,

1, 1,

    0, 0,

    1, 1,

    1, 0,

];

geometry.attributes.uv = new THREE.Float32BufferAttribute(uvs, 2);

// 材质，这里不填材质颜色

var material = new THREE.MeshBasicMaterial({

    side: THREE.DoubleSide

});

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

scene.add(mesh);

设置好uv坐标后，我们就可以加载纹理了

// TextureLoader创建一个纹理加载器对象，可以加载图片作为几何体纹理

var textureLoader = new THREE.TextureLoader();

// 执行load方法，加载纹理贴图成功后，返回一个纹理对象Texture

textureLoader.load(myimgPath, function (texture) {

    // 设置模型的材质为texture

    mesh.material.map = texture;

    // needsUpdate为true会更新材质

    mesh.material.needsUpdate = true;

});

**纹理的其他设置**

1. 重复

如下方法设置纹理再uv坐标系上的重复数量

        // uv两个方向纹理重复数量

        texture.repeat.set(3, 2);



1. 偏移

偏移后，空白的uv部分会默认重复使用贴图

        // 偏移效果

        texture.offset = new THREE.Vector2(0.2, 0)



1. 重复贴图

默认的纹理设置为如下，贴图会在uv坐标上重复

        // 设置阵列模式

        texture.wrapS = THREE.RepeatWrapping;

        texture.wrapT = THREE.RepeatWrapping;

如下，我们设置为不重复

        // 设置阵列模式

        texture.wrapS = THREE.RepeatWrapping;

        texture.wrapT = THREE.RepeatWrapping;



1. 旋转贴图

        // 旋转弧度

        texture.rotation = Math.PI / 4;

        // 设置纹理的旋转中心，默认(0,0)

        texture.center.set(0.5, 0.5);

**视频纹理**

视频可以作为纹理贴图，如游戏里的主角在看电视就是视频纹理贴图

let video = document.createElement('video');

video.src = "1086x716.mp4"; // 设置视频地址

video.autoplay = "autoplay"; //要设置播放

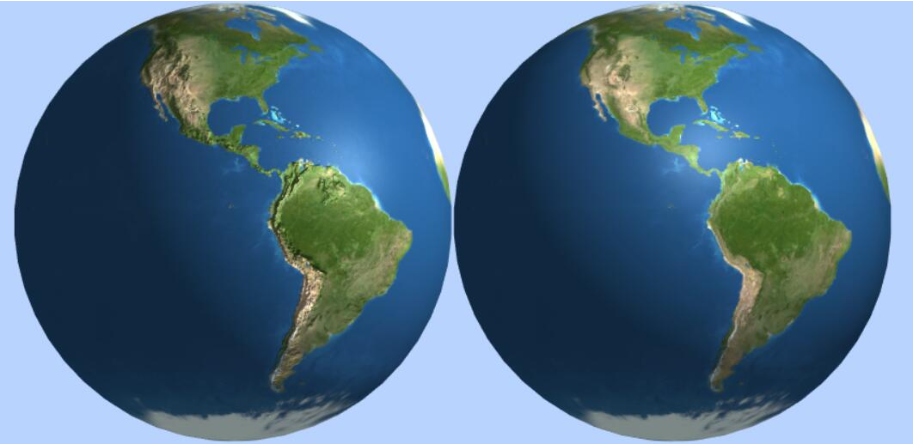
// video对象作为VideoTexture参数创建纹理对象

var texture = new THREE.VideoTexture(video);

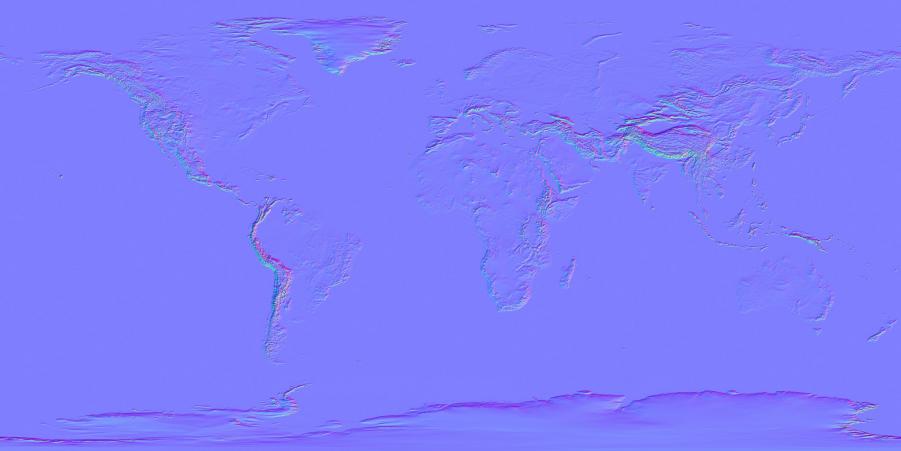
**法线贴图**

如下，对比2个地球，左边的更有立体感，其实这2个地球都是一个球形几何体+地球贴图，

但左边的多加了一个法线贴图



如下是使用的法线贴图，法线贴图也是一张图片，但这张图片的rgb保存的是法线xyz



利用ps可以制作图片的法线贴图

下面我们应用法线贴图，首先生成模型

    const geometry = new THREE.SphereGeometry(100, 50, 50);

    // 材质，这里不填材质颜色

    var material = new THREE.MeshPhongMaterial({

        color: 0xffffff

    });

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

    scene.add(mesh);

然后加载法线贴图

    // TextureLoader创建一个纹理加载器对象，可以加载图片作为几何体纹理

    var textureLoader = new THREE.TextureLoader();

    // 执行load方法，加载纹理贴图成功后，返回一个纹理对象Texture

    textureLoader.load(myimgPath, function (texture) {

        // 设置模型的材质为texture

        mesh.material.normalMap = texture;

        // needsUpdate为true会更新材质

        mesh.material.needsUpdate = true;

    });

}

**高光贴图**

默认整个贴图的反光强度是一直的，如果想要贴图的某个部分反光强，某个部分弱（如地球的陆地部分反光弱，海洋部分反光强），则需要光照贴图

使用方法与法线贴图类似

mesh.material.specularMap = texture;

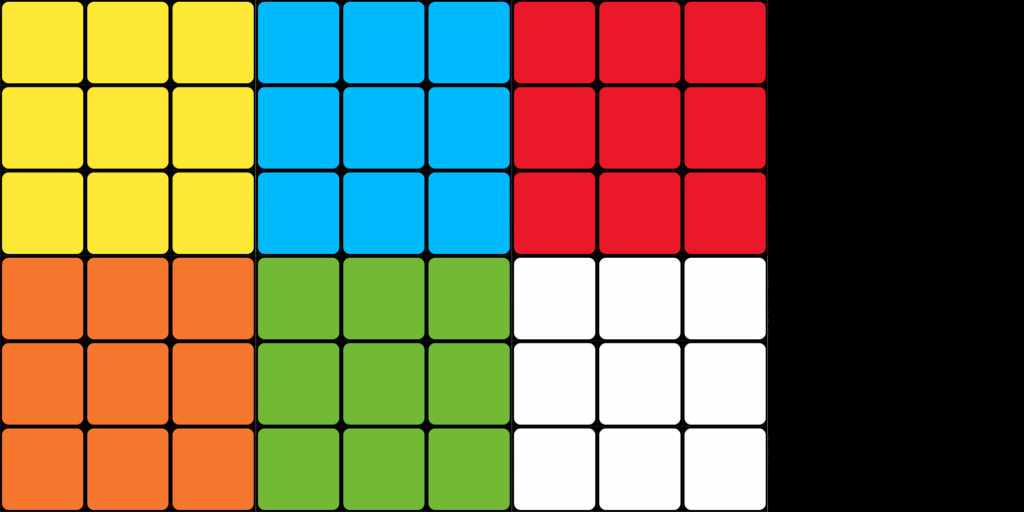
**环境贴图**

环境贴图可以让物体在没有周边环境的情况下看到周边环境的反射

mesh.material.envMap = texture;

**示例：魔方**

如下，我们给正方体贴上这样的纹理



给几何体的uv坐标如下

const geometry = new THREE.BufferGeometry();

// 几何体顶点

let points = [

    -50, -50, -50,

    -50, 50, -50,

    50, -50, -50,

    -50, 50, -50,

    50, 50, -50,

    50, -50, -50,

    -50, -50, 50,

    50, -50, 50,

    -50, 50, 50,

    -50, 50, 50,

    50, -50, 50,

50, 50, 50,

    -50, 50, -50,

    -50, 50, 50,

    50, 50, -50,

    -50, 50, 50,

    50, 50, 50,

50, 50, -50,

    -50, -50, -50,

    50, -50, -50,

    -50, -50, 50,

    -50, -50, 50,

    50, -50, -50,

50, -50, 50,

    -50, -50, -50,

    -50, -50, 50,

    -50, 50, -50,

    -50, -50, 50,

    -50, 50, 50,

    -50, 50, -50,

    50, -50, -50,

    50, 50, -50,

    50, -50, 50,

    50, -50, 50,

    50, 50, -50,

    50, 50, 50,

];

geometry.attributes.position = new THREE.Float32BufferAttribute(points, 3)

// 几何体顶点对应的uv坐标

let uvs = [

    0, 0,

    0, 0.5,

    0.25, 0,

    0, 0.5,

    0.25, 0.5,

0.25, 0,

    0.25, 0,

    0.5, 0,

    0.25, 0.5,

    0.25, 0.5,

    0.5, 0,

0.5, 0.5,

    0.5, 0,

    0.5, 0.5,

    0.75, 0,

    0.5, 0.5,

    0.75, 0.5,

0.75, 0,

    0, 0.5,

    0.25, 0.5,

    0, 1,

    0, 1,

    0.25, 0.5,

0.25, 1,

    0.25, 0.5,

    0.25, 1,

    0.5, 0.5,

    0.25, 1,

    0.5, 1,

    0.5, 0.5,

    0.5, 0.5,

    0.75, 0.5,

    0.5, 1,

    0.5, 1,

    0.75, 0.5,

    0.75, 1,

];

geometry.attributes.uv = new THREE.Float32BufferAttribute(uvs, 2);

// 材质，这里不填材质颜色

var material = new THREE.MeshBasicMaterial({

});

var mesh = new THREE.Mesh(geometry, material); //网格模型对象Mesh

scene.add(mesh);

设置好后我们就能看到一个魔方了

