Trabajando con las caras de un objeto geométrico 3D

En algunas ocasiones necesitaremos definir un material distinto para cada una de las caras de los objetos geométricos utilizados en nuestra escena. En estos casos podemos especificar dichos materiales utilizando un mapeado UV.

Mapeado UV

El mapeado UV es un proceso en el cual puntos específicos de la imagen de la textura son asociados con vértices de la malla para dibujar la imagen en una posición precisa. El nombre se deriva de los nombres asignados a los ejes de la imagen (U corresponde a x y V corresponde a y).

Al utilizar Trhee.js el API se encarga de la mayoría de los cálculos y ofrece alterativas simples para la aplicación de texturas en las situaciones más comunes. Para figuras primitivas básicas, la librería esta incluso configurada con valores por defecto para distribuir la imagen en la forma esperada. Esta lógica pre-programada nos permite componer o mapear la imagen de acuerdo a la forma de la malla a la que queremos aplicar la textura y dejar el resto del trabajo a la librería.

Un efecto importante logrado por medio del mapeado UV es la aplicación de diferentes materiales, y por lo tanto, texturas, a la misma figura. Este proceso es generalmente reservado para mallas complejas creadas por software 3D, pero en el caso de figuras simples como cubos, Three.js ofrece una forma simple de hacerlo.

Primero debemos crear una matriz con tantos objetos como lados tenga nuestra geometría, cada uno con su textura, color o material correspondiente. Esta matriz es luego asignada como segundo parámetro del constructor Mesh() por medio del constructor MeshFaceMaterial(). Este es el constructor que toma una matriz y retorna un tipo de material compuesto que nos permite aplicar varios materiales a la misma malla.

El orden de los materiales y texturas declarados en la matriz es importante. De este modo determinamos la ubicación exacta en la que las imágenes serán dibujadas sobre la superficie de la geometría.

Ejemplo

El siguiente ejemplo muestra un cubo que tiene un color distito en cada uno de sus lados:

```
<!DOCTYPE html>
<html>
<head>
  <title>Cubo de colores</title>
  <script type="text/javascript" src="three.js"></script>
  <script type="text/javascript" src="jquery-1.9.0.js"></script>
</head>
<body>
  <div id="SalidaWebGL"></div>
  <script type="text/javascript">
  $(function () {
    var escena = new THREE.Scene();

    var camara = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight, 0.1, 1000);

    var renderizador = new THREE.WebGLRenderer();
    renderizador.setClearColorHex(0xEEEEEE, 1.0);
    renderizador.setSize(window.innerWidth, window.innerHeight);
    renderizador.shadowMapEnabled = true;

    var luz= new THREE.SpotLight( 0xffffff );
    luz.position.set( -40, 60, -10 );
    luz.castShadow = true;
    escena.add( luz );


    //Crear los materiales para cada lado del cubo (solo colores en este ejemplo)
    var materiales = [
                      new THREE.MeshLambertMaterial({color: 0x0000ff}),
                      new THREE.MeshLambertMaterial({color: 0x00ff00}),
                      new THREE.MeshLambertMaterial({color: 0x00ffff}),
                      new THREE.MeshLambertMaterial({color: 0xff0000}),
                      new THREE.MeshLambertMaterial({color: 0xff00ff}),
                      new THREE.MeshLambertMaterial({color: 0xffff00})
                     ];


    var geometriaDelCubo = new THREE.CubeGeometry(4,4,4);

    //Aquí utilizamos los materiales
    var materialDelCubo = new THREE.MeshFaceMaterial( materiales );
    var cubo = new THREE.Mesh(geometriaDelCubo, materialDelCubo);
    cubo.castShadow = true;

    //Posicionar el cubo
    cubo.position.x=-4;
    cubo.position.y=3;
    cubo.position.z=0;

    //Agregar el cubo a la escena
    escena.add(cubo);


    //Posicionamos la cámara y hacemos que apunte al centro de la escena.
    camara.position.x = -30;
    camara.position.y = 40;
    camara.position.z = 30;
    camara.lookAt(escena.position);

    //Asociamos el resultado del renderizado con el elemento html adecuado (un div o un canvas)
                       $("#SalidaWebGL").append(renderizador.domElement);
```

```
    //Renderizamos la escena
    renderizador.render(escena, camara);



    // llamada a funcion de animación
    animar();

    function animar() {

      // rotar el cubo
      cubo.rotation.x += 0.02; cubo.rotation.y += 0.02; cubo.rotation.z += 0.02;

      // renderizar con requestAnimationFrame
      requestAnimationFrame(animar);
      renderizador.render(escena, camara);
    }

  });
  </script>
</body>
</html>
```

Ejercicio.

Modifique el ejemplo anterior para mostrar un dado girando en su navegador.