Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

ingeniería en Ciencias de la Computación

Graficación - Primavera 2023

Evaluación 2: Sistema Solar Alternativo y Texturas

Fernando José Gil Uribe

fernando.gilu@alumno.buap.mx

10 de marzo 2023

Resumen

El resultado final de esta aplicación es un prototipo funcional que combina HTML, CSS y JavaScript para crear una experiencia interactiva y atractiva. La aplicación muestra una representación inventada de un sistema solar imaginario. Se han incluido secciones de código documentadas, lo que facilita la comprensión del desarrollo y permite la expansión de este en el futuro. Además, se describen los términos técnicos utilizados en el desarrollo para mejorar la comprensión del usuario. Las capturas de pantalla muestran el éxito de la aplicación en su objetivo principal de ofrecer una experiencia gráfica interactiva y atractiva.

Introducción

En este reporte se presenta el desarrollo de una página web que utiliza Three.js para crear una simulación interactiva de un sistema solar imaginario. La simulación está diseñada para permitir a los usuarios explorar y los planetas y el sol de este sistema solar de manera visualmente atractiva. La página web fue desarrollada utilizando Three.js para la visualización 3D y HTML, CSS y JavaScript para la interfaz de usuario y la lógica de la aplicación. A lo largo del reporte se describen los detalles técnicos de la implementación y se discuten las lecciones aprendidas durante el proceso de desarrollo. Además, se presentan algunas posibles mejoras y extensiones futuras para la página web.

Texturizado con ThreeJS

El texturizado en Three JS es relativamente fácil, primero se debe obtener la textura en archivo de imagen (en este caso se usan jpg) y guardarla en los archivos de la página, luego debemos crear un objeto TextureLoader que se encargará de cargar la textura a partir de la textura previamente guardada, posteriormente se usa la función *load()* perteneciente al objeto TextureLoader, de esta forma ya se habrá cargado la imagen al programa, luego se crea un objeto como normalmente se haría, creación de la geometría, luego declaración de un material añadiendo el mapeado del objeto (la textura) como el objeto de la textura cargada, en el caso de objetos planos se puede añadir la opción de “side : THREE.DoubleSided” para que la textura se vea por ambos lados del objeto.

El código completo de texturizado puede ser revisado tanto en este proyecto y de una forma alterna en el siguiente repositorio: https://github.com/josdirksen/learning-threejs/blob/master/chapter-10/09-canvas-texture.html

Explicación del código

```
// Variables de objetos de objetos.
let mercurio, venus, tierra, luna, marte, jupiter, saturno;

// Variables para guardar posiciones de objetos
let tierra_poss = [],
    luna_poss = [],
    mercurio_poss = [],
    venus_poss = [],
    marte_poss = [],
    jupiter_poss = [],
    saturno_poss = [];

// Variables de iteradores para animaciones
let iter_luna = 0,
    iter_tierra = 0,
    iter_mercurio = 0,
    iter_venus = 0,
    iter_marte = 0,
    iter_jupiter = 0,
    iter_saturno = 0;
```

Se crean algunas variables necesarias para el programa.

```
// Función para generar una lista de coordenadas para x,y,z de una circunferencia
function circleCoords(originX, originY, originZ, radius, stepSize = 0.1, yLowerLimit = 0, yUpperLimit = 1) {
    let positions = [];
    let t = 0;
    let y_temp = 0;
    let y_dir = 1;
    while (t < 2 * Math.PI) {

        if(y_temp > yUpperLimit || y_temp < yLowerLimit) y_dir = -y_dir;
        y_temp = ((yLowerLimit == yUpperLimit) ? 0 : y_temp + y_dir);
        positions.push([
            radius * Math.cos(t) + originX,
            y_temp,
            radius * Math.sin(t) + originZ]);
        t += stepSize;
    }
    // let a = positions.slice(0, Math.floor(positions.length / 2));
    // let b = a.slice();
    // b = b.reverse();
    // for(let i = 0; i < b.length ; i++){
    //  b[i][0] = -b[i][0];
    //  b[i][2] = -b[i][2];
    // }
    // positions = a.concat(b);
    // console.log(a);
    // console.log(b);
    // console.log(positions);
    return positions;
}
```

Función para generar una lista de coordenadas para x,y,z de una circunferencia

```
// Función para generar una lista de coordenadas en x,z para una trayectoria en espiral sobre una trayectoria elíptica
function getEllipseCoords(xRadius, zRadius, yOrigin, angleStep = 0.1) {
    let coords = [];
    let angle = 0;
    while (angle < 2 * Math.PI) {
        const x = xRadius * Math.cos(angle);
        const z = zRadius * Math.sin(angle);
        coords.push([x, yOrigin, z]);
        xRadius += -xRadius * (angleStep/(2 * Math.PI));
        zRadius += (2 * Math.PI)*angleStep;
        angle += angleStep;
    }
    let anticoords = coords.slice();
    anticoords.reverse();
    let conc_coords = coords.concat(anticoords);
    return conc_coords;
}
```

Función para generar una lista de coordenadas en x,z para una trayectoria en espiral sobre una trayectoria elíptica

```
container = document.createElement( 'div' );
document.body.appendChild( container );

// Escena y render

scene = new THREE.Scene();
renderer = new THREE.WebGLRenderer( { antialias: true } );
renderer.setPixelRatio( window.devicePixelRatio );
renderer.setSize( window.innerWidth, window.innerHeight );
container.appendChild( renderer.domElement );

// Cámara

camera = new THREE.PerspectiveCamera( 30, window.innerWidth / window.innerHeight, 1, 9000);
camera.position.set(0, 0, 8888);
cameraTarget = new THREE.Vector3( 0, 0, 0 );

// Controles de cámara
let controls = new OrbitControls( camera, renderer.domElement );
controls.screenSpacePanning = true;
controls.minDistance = camera.near;
controls.maxDistance = camera.far;
controls.target.set( 0, 0, 0 );
controls.update();

// Función para dejar la cámara en su posición inicial
function defaultCamera(){
    camera.position.set(0, 0, 8888);
    cameraTarget = new THREE.Vector3( 0, 0, 0 );
    camera.parent = null;
    controls.minDistance = camera.near;
    controls.maxDistance = camera.far;
    controls.target.set( 0, 0, 0 );
    controls.update();
}
```

Asignaciones y funciones para el manejo de la cámara, los controles de cámara y la escena.

```
// Función de utilería para actualizar la posición y objetivo de la cámara
function updateCamera(object){
    if (!object){
        defaultCamera();
        return;
    }
    cameraTarget = object.position;
    camera.parent = object;
    camera.position.set(cameraTarget.x,
                        cameraTarget.y,
                        (object.geometry.parameters.radius ?
                        object.geometry.parameters.radius*8 : object.geometry.parameters.outer_radius));
    // console.log(camera.position);
}
```

Función de utilería para actualizar la posición y objetivo de la cámara

```
// Funciones para el GUI que muestran a cada planeta.
const params = {
    Mercurio: function () {
        updateCamera(mercurio);
        console.log("Viendo a mercurio");
    },
    Venus: function () {
        updateCamera(venus);
        console.log("Viendo a Venus");
    },
    Tierra: function () {
        updateCamera(tierra);
        console.log("Viendo a Tierra");
    },
    Marte: function () {
        updateCamera(marte);
        console.log("Viendo a Marte");
    },
    Jupiter: function () {
        updateCamera(jupiter);
        console.log("Viendo a Jupiter");
    },
    Saturno: function () {
        updateCamera(saturno);
        console.log("Viendo a Saturno");
    },
    Default: function () {
        updateCamera(null);
        console.log("Perspectiva reiniciada");
    },
};
```

Funciones para el GUI que muestran a cada planeta.

```
// GUI y asignaciones de botones.
const gui = new GUI();
gui.add( params, 'Mercurio' ).name( 'Ver Mercurio' );
gui.add( params, 'Venus' ).name( 'Ver Venus' );
gui.add( params, 'Tierra' ).name( 'Ver Tierra (+ Luna)' );
gui.add( params, 'Marte' ).name( 'Ver Marte' );
gui.add( params, 'Jupiter' ).name( 'Ver Jupiter' );
gui.add( params, 'Saturno' ).name( 'Ver Saturno' );
gui.add( params, 'Default' ).name( 'Reiniciar Cámara' );
gui.open();
```

Creación del GUI y asignación de funciones para cada planeta

```
const sol_rad = 696;
const sol_texture = new THREE.TextureLoader().load(
    '../my_content/imgs/2k_sun.jpg' );
const sol_geometry = new ParametricGeometry( ParametricGeometries.mobius, 20, 20 );
// const sol_geometry = new THREE.SphereGeometry( 696, 32, 16 );
const sol_material = new THREE.MeshLambertMaterial( { color: 0xffffff, map : sol_texture, side: THREE.DoubleSide  } );
const sol = new THREE.Mesh( sol_geometry, sol_material);
sol.scale.multiplyScalar(295);
scene.add(sol);
```

Creación del sol y asignación a la escena, el sol se crea como ParametricGeometry para que se pueda crear como la tira de mobius.

```
const mercurio_texture = new THREE.TextureLoader().load(
    '../my_content/imgs/2k_mercury.jpg' );
const mercurio_geometry = new THREE.SphereGeometry( 2.44, 32, 16 );
const mercurio_material = new THREE.MeshLambertMaterial( {
    color: 0xffffff, map: mercurio_texture } );
mercurio = new THREE.Mesh( mercurio_geometry, mercurio_material );
mercurio.position.x = sol_rad + 58;
scene.add(mercurio);
```

Creación de mercurio como una esfera y con su propia textura.

Los demás planetas tienen creaciones similares a mercurio, variando únicamente la textura y la geometría.

```
// Luces
let lights = [];
const n_lights = 7;
const sphereSize = 1/n_lights;
const pl1 = new THREE.PointLight( 0xffffff, sphereSize, 10000 );
pl1.position.set( 0, 0, 0 );
lights.push(pl1);
scene.add( pl1 );
let divs = 80;
// Creación de luces en eje Z
for (let i = 0; i<n_lights ; i++){
    const ptemp = pl1.clone();
    ptemp.position.set( 0, 0, (i%2==0? divs*i : -divs*(i+1)) );
    lights.push(ptemp);
    scene.add( ptemp );
}
// Creación de luces en eje Y
for (let i = 0; i<n_lights ; i++){
    const ptemp = pl1.clone();
    ptemp.position.set( 0, (i%2==0? divs*i : -divs*(i+1)), 0);
    lights.push(ptemp);
    scene.add( ptemp );
}
```

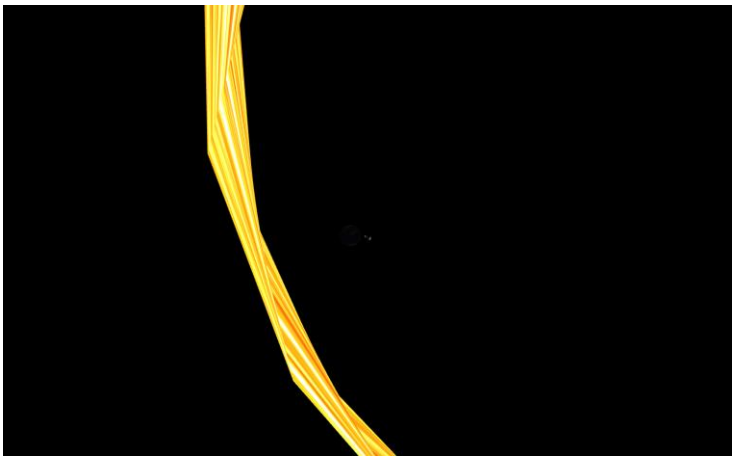
Creación de las luces, una luz central y otras *n_lights* en ejes Z y Y.

```
try {
    // Verificar que hay posiciones de animación
    if(tierra_poss){
        // Reiniciar iteración si se llega al límite
        if (iter_tierra >= tierra_poss.length) iter_tierra = 0;
        // Siguiente posición
        else iter_tierra = iter_tierra+1;
        // Mover objeto
        tierra.position.set(tierra_poss[iter_tierra][0],tierra_poss[iter_tierra][1],tierra_poss[iter_tierra][2]);
    }
    // Verificar que hay posiciones de animación
    if(luna_poss){
        // Reiniciar iteración si se llega al límite
        if (iter_luna >= luna_poss.length) iter_luna = 0;
        // Siguiente posición
        else iter_luna += 1;
        // Asignar parent de luna
        luna.parent = tierra;
        // Mover objeto
        luna.position.set(luna_poss[iter_luna][0],
        luna_poss[iter_luna][1],
        luna_poss[iter_luna][2]);
    }
    // Las siguientes operaciones son similares a la animación de la tierra
    if(mercurio_poss){
        if (iter_mercurio >= mercurio_poss.length) {
            iter_mercurio = 0;
        }
        else iter_mercurio += 1;
        mercurio.position.set(mercurio_poss[iter_mercurio][0],mercurio_poss[iter_mercurio][1],mercurio_poss[iter_mercurio][2]);
    }
    if(venus_poss){
        if (iter_venus >= venus_poss.length) iter_venus = 0;
        else iter_venus += 1;
        venus.position.set(venus_poss[iter_venus][0],venus_poss[iter_venus][1],venus_poss[iter_venus][2]);
    }
    if(marte_poss){
        if (iter_marte >= marte_poss.length) iter_marte = 0;
        else iter_marte += 1;
        marte.position.set(marte_poss[iter_marte][1],marte_poss[iter_marte][2],marte_poss[iter_marte][0]);
    }
    if(jupiter_poss){
        if (iter_jupiter >= jupiter_poss.length) iter_jupiter = 0;
        else iter_jupiter += 1;
        jupiter.position.set(jupiter_poss[iter_jupiter][0],jupiter_poss[iter_jupiter][2],jupiter_poss[iter_jupiter][1]);
    }
    if(saturno_poss){
        if (iter_saturno >= saturno_poss.length) iter_saturno = 0;
        else iter_saturno += 1;
        saturno.position.set(saturno_poss[iter_saturno][1],saturno_poss[iter_saturno][0],saturno_poss[iter_saturno][2]);
    }
}catch (error) { }
```

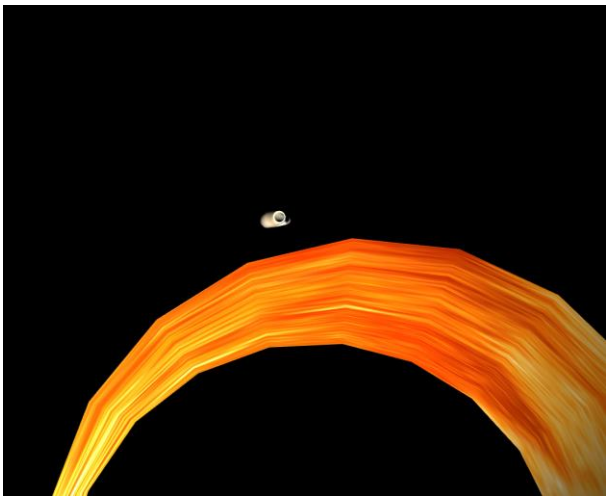
Procesamiento de las animaciones de cada objeto dentro de la función render.

Resultados

Aquí se puede ver el sol y varios planetas pequeños por la comparación de tamaño.



Aquí se ve la tierra en aumento porque se tiene la posibilidad de visualizar a un planeta en específico.



Este es saturno mostrado como una botella de Klein, igualmente está rotando alrededor del sol.

Conclusión

El trabajo que este programa implicó fue útil para entender cómo texturizar un objeto tridimensional para una página web, igualmente pude aprender cómo manejar los objetos y sus trayectorias para que varios objetos se movieran a la vez. También aprendí a utilizar las luces dentro del programa y hacer que se refleje la luz sobre los objetos de forma correcta incluso con múltiples fuentes de luz. En general el proyecto fue útil para el manejo de objetos como tal a diferencia de la tarea anterior que se centraba más al manejo de cosas no tan visuales.

Referencias

https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene

https://threejs.org/docs/index.html?q=sphe#api/en/geometries/SphereGeometry

https://threejs.org/docs/index.html?q=text#manual/en/introduction/Creating-text

https://github.com/mrdoob/three.js/