Implementación de grafos y shaders en ThreeJs

Gomez Hector, Pinzón Arturo, y Badran Kevin
{u6000277, u6000335, y u6000258*}@unimilitar.edu.co*
Profesor: Gabriel Eduardo Ávila Buitrago

***Resumen*— El siguiente documento es un apoyo informativo que corresponde a la realización de un proyecto realizado en Three Js, en el cual se pretendía hacer una representación de un mapa de un barrio o ciudad, el cual se implementan shaders y texturas que permiten una mejor visualización del escenario.**

I. Introducción

Para dar conocimiento a nuestro proyecto , debemos tener una perspectiva clara en cuanto a la formación y ubicación de objetos con sombra, agregado a ello un mapa donde podemos ver nuestra posición , todo esto lo hacemos realidad con Three.Js , dándole color, vida y sentido a todo lo aprendido este semestre.

A. Marco teórico

Para la realización del proyecto, es necesario definir los **Shaders** se puede especificar dos tipos diferentes de sombreadores para cada material:

- El sombreador de vértices se ejecuta primero; recibe atributos , calcula / manipula la posición de cada vértice individual y pasa datos adicionales (que varían) al sombreador de fragmentos.
- El sombreador de fragmentos (o píxeles) se ejecuta en segundo lugar; establece el color de cada "fragmento" (píxel) individual que se muestra en la pantalla.
- Hay tres tipos de variables en los sombreadores: uniformes, atributos y variaciones:
- Los uniformes son variables que tienen el mismo valor para todos los vértices; los mapas de iluminación, niebla y sombras son ejemplos de datos que se almacenarán en uniformes. Se puede acceder a los uniformes mediante el sombreador de vértices y el sombreador de fragmentos.
- Los atributos son variables asociadas con cada vértice; por ejemplo, la posición del vértice, la cara normal y el color del vértice son ejemplos de datos que se almacenarán en atributos. Solo se puede acceder a los atributos dentro del sombreador de vértices.
- Las variaciones son variables que se pasan del sombreador de vértices al sombreador de fragmentos. Para cada fragmento, el valor de cada variable se interpolara suavemente a partir de los valores de los vértices adyacentes.
- Tenga en cuenta que dentro del propio sombreador, los uniformes y los atributos actúan como constantes; solo puede modificar sus valores pasando valores diferentes a los búferes desde su código JavaScript.

Para ello procedimos a realizar un código donde se usan distintos objetos para implementar shaders.
Desde un punto de vista intuitivo un **grafo** es un conjunto de nodos unidos por un conjunto de arcos. Un ejemplo de grafo que podemos encontrar en la vida real es el de un plano de trenes. El plano de trenes está compuesto por varias estaciones (nodos) y los recorridos entre las estaciones (arcos) constituyen las líneas del trazado.

II. COMPETENCIAS A DESARROLLAR

- Diseñar y realizar una idea donde se integren las competencias propuestas (shaders y un mapa).
- Crear el código con las estructuras que representan edificios.
- Añadir una cámara de plano panorámico picado, donde se vea la ubicación del usuario a tiempo real
- Añadir árboles y señales con shaders.
- Presentar y sustentar nuestro código y resultados al docente.

III. .Cronograma

17/Sep/2020	Entrega Propuesta	
24/Sep/2020	Entrega de un Documento de investigación	Realizamos la investigación necesaria para así poder implementar shader y grafos para nuestro futuro proyecto
1/Oct	Primera entrega adelanto de proyecto	Se entregó un trabajo más desarrollado con las especificaciones que tendría nuestro proyecto
5/nov	Revisión de lo trabajado en el proyecto	En esta entrega se vio el avance que teniamos el cual fue una pequeña parte de la ciudad con la posibilidad de mover

		un una cámara simulando la primera vista de un personaje
26/Nov/2020	Entrega final	

IV. Desarrollo de la práctica

Para el desarrollo de la práctica fue necesario investigar y tener en cuenta los temas relacionados con shaders y texturas, esto antes de la realización del documento y al mismo la implementación del mismo.

Seguidamente se realizó la creación de la escena,, en este proceso se diseñó primeramente la carretera por la cual nuestro usuario iba a trasladarse, los objetos que compondrían a la misma y después la ubicación de todos y cada uno de estos.

Era necesario implementar las texturas a los objetos creados, para lo cual se descargaron diferentes tipos de materiales en un formato de imágen, el cual después fue importado a la escena (objeto por objeto) para que los elementos de la misma se adaptaran bien, como se puede ver en la imagen:

```
var texture = new THREE.TextureLoader().load("textures/3.png");
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.RepeatWrapping;
texture.repeat.set(8, 8);
var materialCallesita = new THREE.MeshLambertMaterial({ map: texture });
var cll1 = new THREE.Mesh(new THREE.BoxGeometry(500, 0.2, 500, 150), materialCallesita);
cll1.receiveShadow = true;
```

visualizandose de la siguiente manera:

En la imagen se puede ver como la textura es implementada en la escena. Teniendo en cuenta dicha imágen, observamos que dentro del proyecto encontramos árboles, los cuales fueron creados a partir de curvas, como se puede ver a continuación:

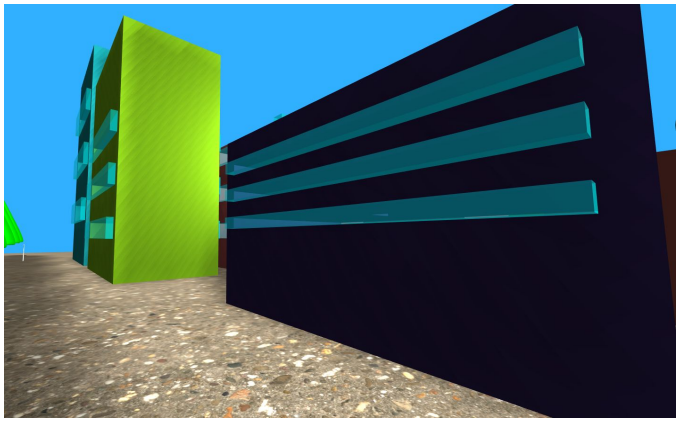
```
static arbolitosOme(x, z){
    var curve2D = [];
    curve2D[0] = new THREE.Vector2(0.7, 0);
    curve2D[1] = new THREE.Vector2(0.8, 7);
    curve2D[2] = new THREE.Vector2(7, 8);
    curve2D[3] = new THREE.Vector2(9, 10);
    curve2D[4] = new THREE.Vector2(0, 30);


    var shape = new THREE.Shape();

    shape.moveTo(0, 0);
    shape.splineThru(curve2D);

    var resolution = 50;
    var points = shape.getPoints(resolution);

    var geometry = new THREE.LatheGeometry(points, 10);
    var material = new THREE.MeshLambertMaterial({color:'green'});

    const geometryCil = new THREE.CylinderGeometry(0.8, 0.8, 15, 32);
    const materialCil = new THREE.MeshLambertMaterial({ color: 0x725f0a });
```

Los edificios creados se crearon con base en operaciones entre primitivas creadas con CSG, los cuales toman colores aleatorios cada vez que se recarga la página y el programa vuelve a correr.

En la siguiente imagen se ve la manera en la que se le asigna el material a los edificios.

```
function materialEdifisito() {

    var colorRandom;


    var n = Math.floor(Math.random() * 6);
    switch (n) {
        case 0: colorRandom = new THREE.Color(0x1C1234);
            break;
        case 1: colorRandom = new THREE.Color(0x514E57);
            break;
        case 2: colorRandom = new THREE.Color(0x776B73);
            break;
        case 3: colorRandom = new THREE.Color(0x803B38);
            break;
        case 4: colorRandom = new THREE.Color(0x537B0E);
            break;
        case 5: colorRandom = new THREE.Color(0x0C6164);
            break;
    }

    var materialEdifisito = new THREE.MeshStandardMaterial({
        color: colorRandom,
        metalness: 0.5,
        roughness: 0.5,
        opacity: 0.75,
        side: THREE.DoubleSide,
        transparent: false

    });

    return materialEdifisito;
}
```

En la siguiente imagen se puede mirar cómo quedaron los edificios a partir de la resta de las primitivas de CSG



En las siguientes imágenes se podrá observar la manera en la que se crean los edificios, en el siguiente orden: creación de edificio, creación de las ventanas (armadas como una estructura similar a un árbol, denominadas en el código como "ARBOLITOME"), implementación de las operaciones de CSG.

```
var E2CSG = THREE.CSG.fromMesh(E2);
var E3CSG = THREE.CSG.fromMesh(E3);
var E4CSG = THREE.CSG.fromMesh(E4);
var E5CSG = THREE.CSG.fromMesh(E5);
var E6CSG = THREE.CSG.fromMesh(E6);
var E7CSG = THREE.CSG.fromMesh(E7);
var E8CSG = THREE.CSG.fromMesh(E8);
var E9CSG = THREE.CSG.fromMesh(E9);
```

```
var ARBOLITOME = vCSG.union(v1CSG).union(v2CSG).union(v3CSG);

var result3 = ARBOLITOME.subtract(E9CSG);
var result4 = E9CSG.subtract(ARBOLITOME);


var arbustofeoOME = THREE.CSG.toMesh(ARBOLITOME);

var CasitasOME = THREE.CSG.toMesh(result3);
CasitasOME.material = materialVentanita;

var Casitas2OME = THREE.CSG.toMesh(result4);
Casitas2OME.material = materialEdifisito();

//CLONAR
ARBOLITOME2 = arbustofeoOME.clone();
ARBOLITOME2.applyMatrix4(new THREE.Matrix4().makeScale(1, .6, 1));
ARBOLITOME2.position.x = 50;
ARBOLITOME2.position.y = 0;
ARBOLITOME2.position.z = 0;
```

```
var result11 = ARBOLITOME5CSG.subtract(E5CSG);
var result12 = E5CSG.subtract(ARBOLITOME5CSG);
```

```
var Casitas3OME = THREE.CSG.toMesh(result5);
Casitas3OME.material = materialVentanita;
var Casitas4OME = THREE.CSG.toMesh(result6);
Casitas4OME.material = materialEdifisito();
```

```
Casitas2OME.castShadow = true;
Casitas2OME.receiveShadow= true;
Casitas4OME.castShadow = true;
Casitas4OME.receiveShadow= true;
Casitas6OME.castShadow = true;
Casitas6OME.receiveShadow= true;
Casitas8OME.castShadow = true;
Casitas8OME.receiveShadow= true;
Casitas10OME.castShadow = true;
Casitas10OME.receiveShadow= true;
Casitas11OME.castShadow = true;
Casitas11OME.receiveShadow= true;
Casitas12OME.castShadow = true;
Casitas12OME.receiveShadow= true;
Casitas14OME.castShadow = true;
Casitas14OME.receiveShadow= true;
Casitas16OME.castShadow = true;
Casitas16OME.receiveShadow= true;
```

La cámara utilizada es una cámara en primera persona con la que el usuario puede recorrer todo el barrio, y la cual puede observar su entorno moviendo el mouse y las teclas "w,a,s,d", adicional a ello, al personaje se le implementan velocidades y un máximo del ángulo que este puede girar con respecto a la rotación en el eje y.

```
var raycaster;
var blocker = document.getElementById('blocker');
var instructions = document.getElementById('instructions');
// https://www.html5rocks.com/en/tutorials/pointerlock/intro/
var havePointerLock = 'pointerLockElement' in document || 'mozPointerLockElement' in document || 'webkitPointerLockElement' in document;
if (havePointerLock) {
    var element = document.body;
    var pointerlockchange = function (event) {
        if (document.pointerLockElement === element || document.mozPointerLockElement === element || document.webkitPointerLockElement === element) {
            controlsEnabled = true;
            controls.enabled = true;
            blocker.style.display = 'none';
        } else {
            controls.enabled = false;
            blocker.style.display = '-webkit-box';
            blocker.style.display = '-moz-box';
            blocker.style.display = 'box';
            instructions.style.display = '';
        }
    };
    var pointerlockerror = function (event) {
        instructions.style.display = '';
    };
```

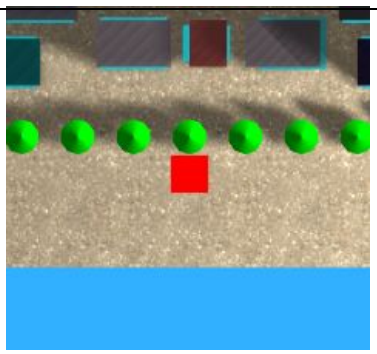
```
// Hook pointer lock state change events
document.addEventListener('pointerlockchange', pointerlockchange, false);
document.addEventListener('mozpointerlockchange', pointerlockchange, false);
document.addEventListener('webkitpointerlockchange', pointerlockchange, false);
document.addEventListener('pointerlockerror', pointerlockerror, false);
document.addEventListener('mozpointerlockerror', pointerlockerror, false);
document.addEventListener('webkitpointerlockerror', pointerlockerror, false);
instructions.addEventListener('click', function (event) {
    instructions.style.display = 'none';
    // Ask the browser to lock the pointer
    element.requestPointerLock = element.requestPointerLock || element.mozRequestPointerLock || element.webkitRequestPointerLock;
    if (/Firefox/i.test(navigator.userAgent)) {
        var fullscreenchange = function (event) {
            if (document.fullscreenElement === element || document.mozFullscreenElement === element || document.mozFullScreenElement === element) {
                document.removeEventListener('fullscreenchange', fullscreenchange);
                document.removeEventListener('mozfullscreenchange', fullscreenchange);
                element.requestPointerLock();
            }
        };
        document.addEventListener('fullscreenchange', fullscreenchange, false);
        document.addEventListener('mozfullscreenchange', fullscreenchange, false);
        element.requestFullscreen = element.requestFullscreen || element.mozRequestFullscreen || element.mozRequestFullScreen || element.webkitRequest
        element.requestFullscreen();
    } else {
        element.requestPointerLock();
    }
}, false);
} else {
    instructions.innerHTML = 'Your browser doesn\'t seem to support Pointer Lock API';
```

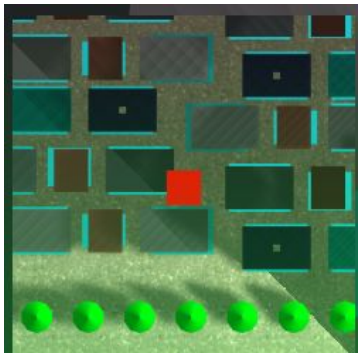
```
    var controlsEnabled = false;
    var moveForward = false;
    var moveBackward = false;
    var moveLeft = false;
    var moveRight = false;
    var canJump = false;
    var prevTime = performance.now();
    var velocity = new THREE.Vector3();
    var direction = new THREE.Vector3();
    var plano, raycast, mousePosX, mousePosY, mousePos;
    var interseccion, objetos = [], objetoInterseccion;
    // ----------------------------------------------------------------

var scene, aspect, camera, renderer, controls;
var ARBOLITOME2,ARBOLITOME3,ARBOLITOME4,ARBOLITOME5,ARBOLITOME6,ARBOLITOME7,ARBOLITOME8;
var keyboard = {};
var player = { height:3, speed:0.1, turnSpeed:Math.PI*0.02 };
var USE_WIREFRAME = false;
var upArrow = false;
var downArrow = false;
var leftArrow = false;
var rightArrow = false;
var objects = [];

var insetHeight, insetWidth;
var origin = new THREE.Vector3(0, 0, 0);
var sphere, playerX, playerZ;
init();
animate();
```

```
if (controlsEnabled) {
    raycaster.ray.origin.copy(controls.getObject().position);
    raycaster.ray.origin.y -= 10;

    var intersections = raycaster.intersectObjects(objects);
    var onObject = intersections.length > 0;
    var isOnObject = intersections.length > 0;
    var time = performance.now();
    var delta = (time - prevTime) / 1000;

    velocity.x -= velocity.x * 10.0 * delta;
    velocity.z -= velocity.z * 10.0 * delta;
    velocity.y -= 9.8 * 100.0 * delta; // 100.0 = mass

    direction.z = Number(moveForward) - Number(moveBackward);
    direction.x = Number(moveRight) - Number(moveLeft);
    direction.normalize(); // this ensures consistent movements in all directions

    if (moveForward || moveBackward) velocity.z -= direction.z * 400.0 * delta;
    if (moveLeft || moveRight) velocity.x -= direction.x * 400.0 * delta;

    if (onObject === true) {
        velocity.y = Math.max(0, velocity.y);
        canJump = true;
    }

    controls.moveRight(- velocity.x * delta);
    controls.moveForward(- velocity.z * delta);

    controls.getObject().position.y += (velocity.y * delta); // new behavior

    if (controls.getObject().position.y < 10) {
        velocity.y = 0;
        controls.getObject().position.y = 10;
        canJump = true;
    }
    prevTime = time;
}
```

Por último se realizó la implementación de la cámara que siempre está mirando al usuario, es decir, la sigue cada vez que el usuario se mueve, tal como se ve a continuación:

```
requestAnimationFrame(animate);
playerX=camera.position.x;
playerZ=camera.position.z;
sphere.position.set(playerX, 90, playerZ);


cameraOrtho.position.set(playerX, 100, playerZ);
cameraOrtho.lookAt(playerX, 0, playerZ);
```





En el siguiente enlace se encuentra un video en el cual se puede observar el proyecto finalizado:

https://www.youtube.com/watch?v=IulSRNtSZLg

V. Conclusiones

- Se lograron implementar adecuadamente todas las características que deseábamos tener en nuestro código, de igual manera añadimos objetos y líneas que nos ayudaron a expandir el mundo que no se tenían en la planificación del proyecto, al mismo tiempo es necesario resaltar que se entendió el propósito del trabajo que se realizó y se vinculó, según el grupo, a la perfección, con respecto a lo visto a lo largo del curso en el semestre.

Referencias

Nuestro código fue basado en:

[1] https://jsfiddle.net/f2Lommf5/11653/
[2] https://threejs.org/docs/#examples/en/controls/PointerLockControls
[3] https://github.com/mrdoob/three.js/blob/master/examples/jsm/controls/PointerLockControls.js
[4] https://threejs.org/docs/#api/en/materials/ShaderMaterial
Donde los tomamos como referencia respetando el trabajo de los autores.