Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

ingeniería en Ciencias de la Computación

Graficación - Primavera 2023

Evaluación 3: Modelos 3D y Audio

Fernando José Gil Uribe

fernando.gilu@alumno.buap.mx

28 de abril 2023

Resumen

El proyecto se enfocó en crear una experiencia interactiva y visualmente atractiva utilizando tecnología web. Se utilizó Threejs, una biblioteca de JavaScript para crear la escena tridimensional. El proyecto incluye una variedad de modelos 3D importados desde un repositorio de modelos tipo OBJ y utilizados para decorar la escena o para animaciones complejas. Además, se ha compartido el código utilizado en la creación del proyecto para futuras referencias y desarrollo. En el informe se presentan capturas de pantalla que muestran el desempeño de la aplicación, dentro de los límites de lo que una imagen puede transmitir. Este proyecto demuestra el potencial de la tecnología web para crear experiencias 3D interactivas y atractivas, con posibilidades de desarrollo en el futuro.

Introducción

Este informe presenta un proyecto desarrollado en Three.js, una biblioteca de JavaScript para crear y ver gráficos 3D en un navegador web. En esta aplicación, Three.js se usa para crear una escena 3D interactiva con objetos seleccionados de un repositorio de modelos 3D de tipo OBJ. El objetivo de este proyecto es mostrar la capacidad de Three.js para crear experiencias visuales y de audio. Además de la descripción general de la aplicación, este informe incluye detalles sobre el proceso de desarrollo y el código utilizado para crear la aplicación. También se proporcionan capturas de pantalla para mostrar el resultado y la calidad de la experiencia que se puede crear con esta tecnología.

Uso de modelos tridimensionales

El uso y carga de modelos tridimensionales en Threejs se vuelve un poco simple cuando se lee la documentación y gracias a experiencia previa con la biblioteca, se facilita desarrollar este proyecto. Para empezar, se debe conseguir un modelo 3D en un modelo que Threejs tenga la posibilidad de cargar, hay diferentes tipos de archivo que se pueden cargar como MTL, DRC o 3DM, pero en este caso todos los archivos utilizados fueron de tipo OBJ, se cargan usando el OBJLoader importado de THREE.js, el cual recibe una función de carga la cual puede modificar el archivo al gusto del programador, e igualmente de ser necesario se puede trasladar la variable del objeto creado a una variable global del programa para poder manipularla posteriormente para las animaciones.

Explicación del código

Creación de la curva paramétrica para animar uno de los objetos de la escena:

```
curve = new THREE.CatmullRomCurve3( [
    new THREE.Vector3( 200, 0, 200 ),
    new THREE.Vector3( -200, 50, 200 ),
    new THREE.Vector3( -200, 0, -200 ),
    new THREE.Vector3( 200, 50, -200 ),
],true);
curvePoints = curve.getPoints( 50 );
const geometry = new THREE.BufferGeometry().setFromPoints( curvePoints );
const material = new THREE.LineBasicMaterial( { color: 0xff0000 } );
const curveObject = new THREE.Line( geometry, material );
scene.add(curveObject)
```

Asignación de las variables necesarias para el procesamiento, reproducción y análisis del audio:

```
audio_listener = new THREE.AudioListener();
camera.add( audio_listener );
sound = new THREE.Audio( audio_listener );
audio_analyser = new THREE.AudioAnalyser( sound, 32 );

// load a sound and set it as the Audio object's buffer
audio_loader = new THREE.AudioLoader();
audio_loader.load( 'audio/'+songs[0]+'.mp3', function( buffer ) {
    sound.setBuffer( buffer );
    sound.setLoop(true);
    sound.setVolume(0.5);
});
```

Ejemplo de carga de un modelo, la función de carga toma 4 diferentes parámetros, la dirección relativa del modelo a cargar, una función de carga, una función de notificación de éxito y una de

notificación de errores:

```
// instantiate a loader
const model_loader = new OBJLoader();
// load a resource
model_loader.load(
    'models/objs/capybara.obj',
    function ( obj ) {
        obj.position.set(0, 0, 0)
        obj.rotation.x = Math.PI * 0.5;
        obj.rotation.y = Math.PI * 1;
        obj.rotation.z = Math.PI * 1;
        obj.scale.multiplyScalar(4);
        scene.add(obj)
        capybara = obj;
    },
    function ( xhr ) {console.log( ( xhr.loaded / xhr.total * 100 ) + '% loaded' );},
    function ( error ) {console.log("Model error:" + error);}
);
```

Creación del GUI para controlar la aplicación:

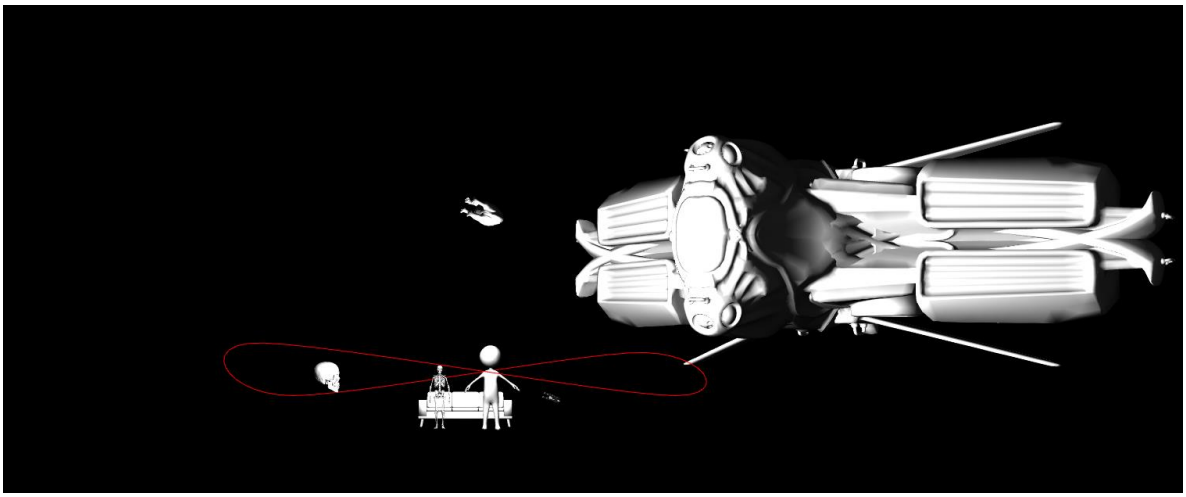
```
// Funciones para el GUI que muestran a cada planeta.
const params = {
    Default: function () {
        updateCamera(null);
        console.log("Perspectiva reiniciada");
    },
    PlayPause: function () {
        if(sound.isPlaying) sound.pause();
        else sound.play();
    },
    ChangeSong: function () {
        songIndex = songIndex + 1 > songs.length-1 ? 0 : songIndex+1;
        audio_loader.load('audio/'+songs[songIndex]+'.mp3', function( buffer ) {
            sound.stop();
            sound.setBuffer( buffer );
            sound.setLoop(true);
            sound.setVolume(0.5);
            sound.play();
        });
    },
};
const gui = new GUI();
gui.add( params, 'Default' ).name( 'Reiniciar Cámara' );
gui.add( params, 'PlayPause' ).name( 'Iniciar/Detener música' );
gui.add( params, 'ChangeSong' ).name( 'Cambiar canción' );
gui.open();
// Fin de GUI
```

Animación de la aplicación utilizando los frames de la aplicación como medida de cambio para las diferentes animaciones:

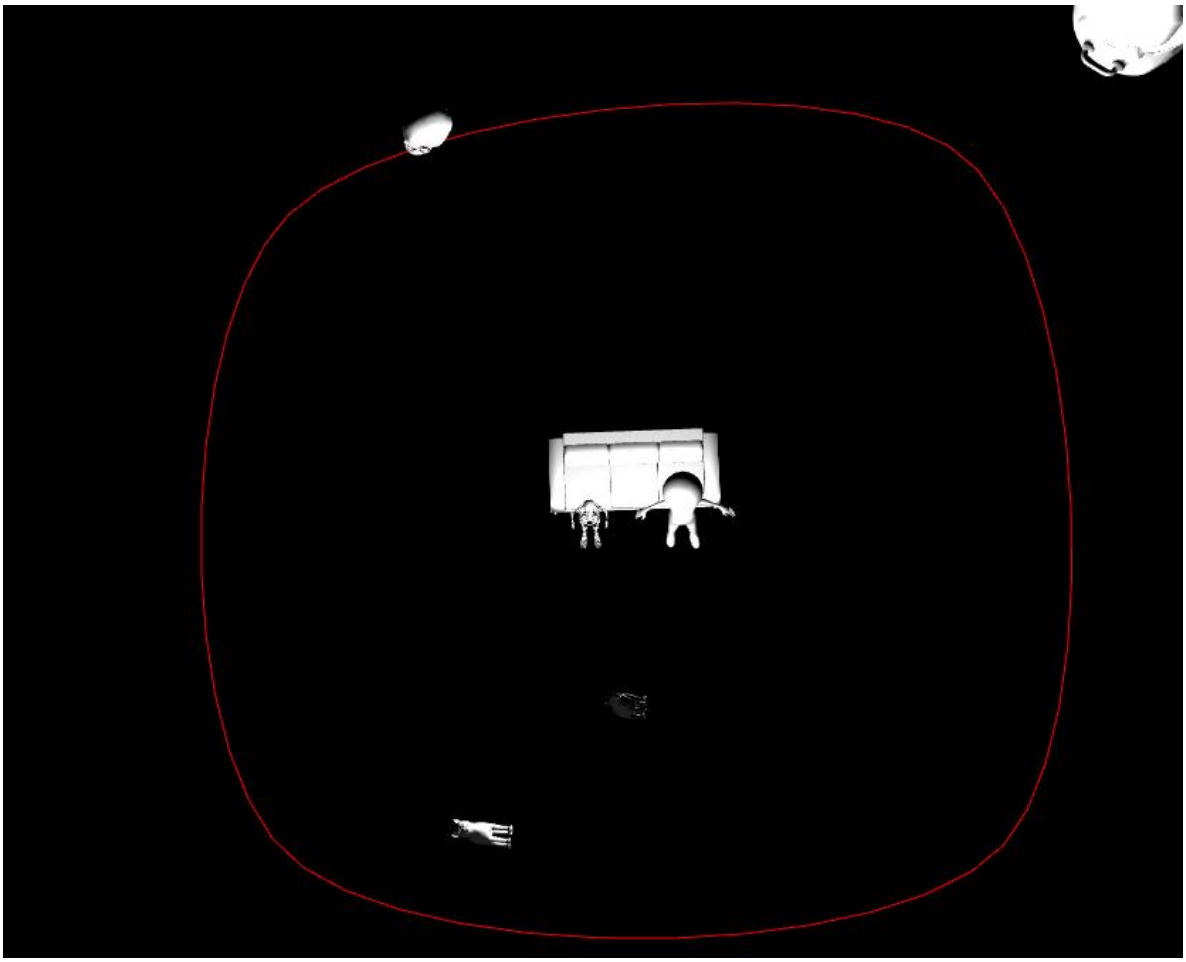
```
function animate() {
    time = .00025* performance.now();
    t = time % 1;
    varsize = Math.abs(Math.cos(time*10/2) * 150)
    if(cat){
        cat.position.x = -Math.cos(time*10/2) * animationRadius;
        cat.position.z = Math.sin(time*10/2) * animationRadius;
        cat.rotation.y = Math.PI * (time*1.5);
        cat.rotation.z = Math.PI * (time*2);
        cat.rotation.x = Math.PI * (time*3);
        cat.scale.multiplyScalar((Math.cos(time*10)>0 ? 1.01 : 0.99))
    }
    if(capybara){
        capybara.position.y = -Math.cos(time*10/2+20) * animationRadius;
        capybara.position.z = Math.sin(time*10/2+20) * animationRadius;
        capybara.rotation.y = Math.PI * (time*1.5);
        capybara.rotation.z = Math.PI * (time*2);
        capybara.rotation.x = Math.PI * (time*3);
        capybara.scale.multiplyScalar((-Math.cos(time*10)>0 ? 1.01 : 0.99))
    }
    if(skull1 && audio_analyser){
        if(curveInd > curvePoints.length-1)  curveInd = 0
        const pointOnCurve = curvePoints[curveInd];
        skull1.position.x = pointOnCurve.x
        skull1.position.z = pointOnCurve.z
        const scale = audio_analyser.getAverageFrequency()/8
        const setscale = audio_analyser.getAverageFrequency() ?
            scale : 5;
        skull1.scale.set(setscale,setscale,setscale)
        skull1.position.y = pointOnCurve.y + audio_analyser.getAverageFrequency();
        currentRot +=1
        if(currentRot == rotationDelay) {
            currentRot = 0
            curveInd += 1
        }
        skull1.rotation.y = Math.PI * (time*1.5);

    }
    if(spaceship){
        spaceship.position.y = (Math.cos(time*10)>0 ?
        spaceship.position.y + 0.1 : spaceship.position.y-0.1)
        spaceship.rotation.x = -Math.PI * (Math.cos(time*10)*.01);
    }
    iter++
    requestAnimationFrame( animate );
    render();
}
```

Resultados

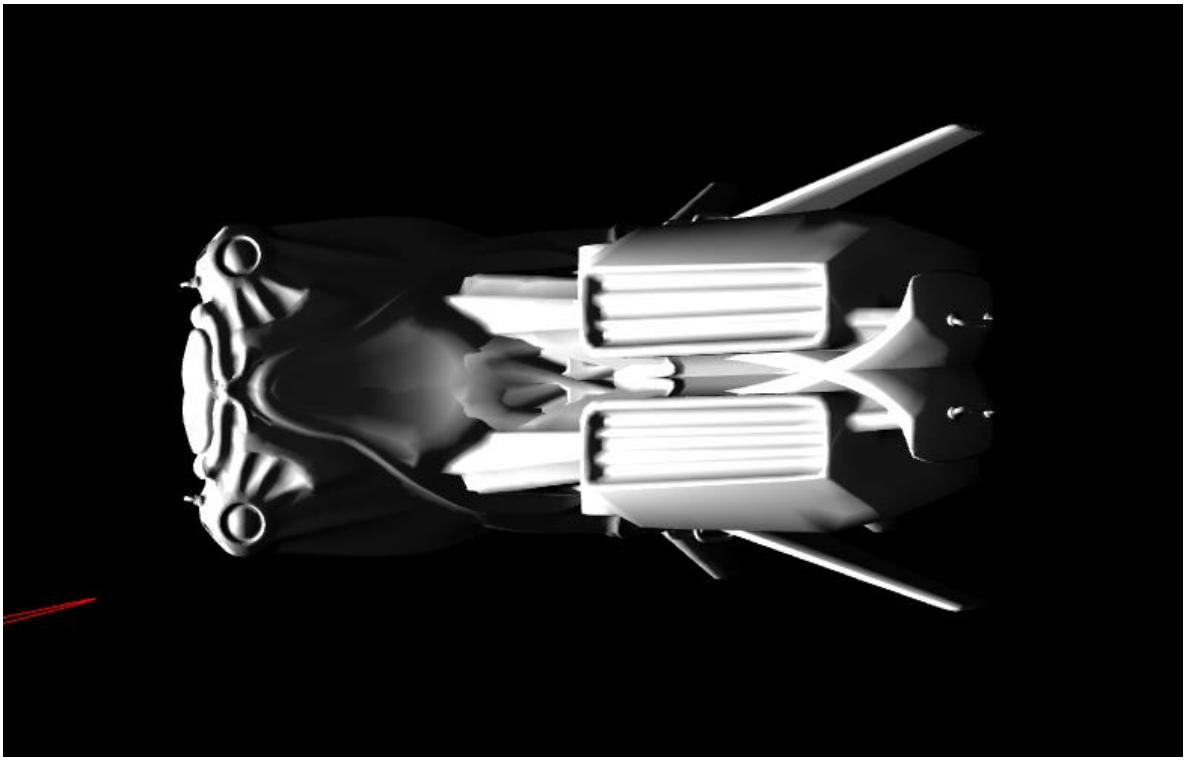
Escena general:



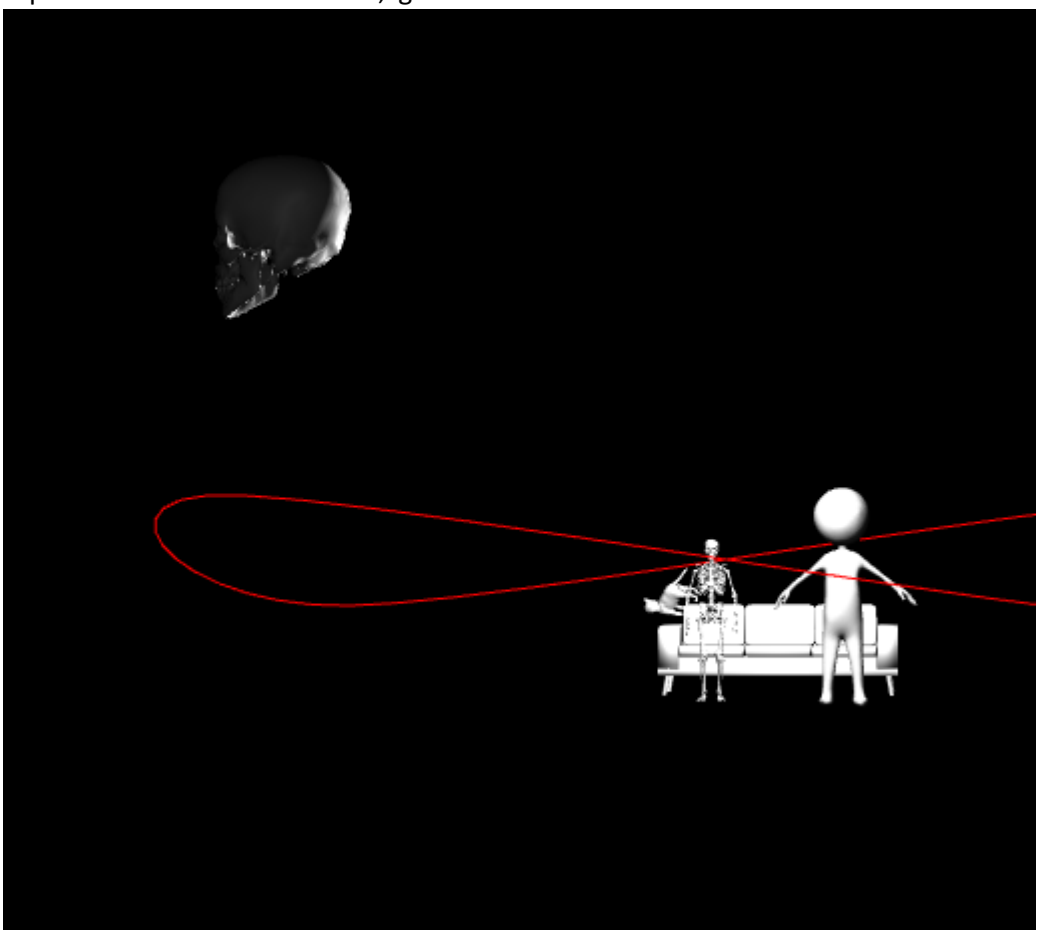
El modelo sobre la línea roja es el que sigue una curva paramétrica, mientras que los otros dos modelos chicos no centrados siguen trayectorias designadas por funciones trigonométricas:



Una nave espacial que se mueve de acuerdo con funciones trigonométricas:



El modelo de cráneo modifica su posición en Y de acuerdo con la frecuencia del audio que se esté reproduciendo en el momento, igualmente se modifica su escala en función de lo mismo:



Conclusión

El trabajo en este programa fue interesante para entender cómo cargar modelos externos a la librería base de ThreeJS, lamentablemente ThreeJS pareciera tener algunas limitaciones en cuanto a sus funciones como poder dar un material a partes específicas de un modelo de forma sencilla, pero igualmente provee de herramientas útiles para poder modificar y administrar las utilizades de los modelos cargados. En cuanto a la función de análisis de audio, no le veo demasiada utilidad o usos prácticos en un proyecto formal, pero posiblemente tenga un potencial para animaciones como esta misma, en las que se tiene que hacer algo dependiendo del audio que se esté reproduciendo.

Referencias

https://threejs.org/docs/index.html?q=loader#examples/en/loaders/OBJLoader

https://threejs.org/docs/index.html?q=audio#api/en/audio/Audio

https://threejs.org/docs/index.html?q=audio#api/en/loaders/AudioLoader

https://threejs.org/docs/index.html?q=audio#api/en/audio/AudioAnalyser

https://threejs.org/docs/index.html?q=audio#api/en/audio/AudioListener

https://threejs.org/docs/index.html?q=curve#api/en/extras/curves/CatmullRomCurve3

https://threejs.org/docs/index.html?q=curve#api/en/extras/core/Curve