МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

ЦИФРОВІ ТЕХНОЛОГІЇ В ЕНЕРГЕТИЦІ

Графічно-розрахункова робота з дисципліни

«Методи синтезу віртуальної реальності»

Варіант-3

**Виконала:**

студентка гр. ТР-21мн

Федорова Ю. Є.

**Перевірив:**

Демчишин А. А.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2023

**Завдання для графічно-розрахункової роботи**

1. Повторно використовувати код із практичного завдання №2;
2. Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою матеріального інтерфейсу (цього разу поверхня залишається нерухомою, а джерело звуку рухається). Відтворити свою улюблену пісню у форматі mp3/ogg із просторовим розташуванням джерела звуку, керованим користувачем;
3. Візуалізувати положення джерела звуку за допомогою сфери;
4. Додати звуковий фільтр (використовуйте інтерфейс BiquadFilterNode) для кожного варіанта. Додайте елемент прапорця, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак.
5. Підготувати цифровий звіт.
6. Додати звіт до гілки Git

Варіант - Шелфовий фільтр високих частот

**Теорія**

1. Концепція Просторового Звуку:

Просторовий звук - це важливий аспект аудіо-візуальних додатків, що створює враження присутності та напрямку звуку в тривимірному просторі. У проекті просторовий звук використовується для імітації руху звукового джерела, що дозволяє відчути його присутність у віртуальному просторі. Ця техніка досягається шляхом динамічної зміни гучності та ефектів в залежності від положення джерела відносно слухача.

2. Використання BiquadFilterNode для звукового Фільтру:

BiquadFilterNode - це потужний інструмент веб-аудіо, що надає можливість фільтрації звуку. У нашому випадку, ми використовуємо високочастотний фільтр типу "highshelf"(Шелфовий фільтр високих частот). Це дозволяє модифікувати частотні характеристики звуку в залежності від висоти джерела. Такий фільтр підсилює ефект зміни частоти та додає просторовий аспект до аудіо-візуального досвіду.

3. Обертання Звукового Джерела:

Обертання звукового джерела навколо геометричного центру використовується для створення враження руху та просторової динаміки. Математичні обчислення визначають положення джерела на кожній ітерації анімації, створюючи враження обертання звуку. Це взаємодіє з положенням слухача, дозволяючи відчути напрямок та рух джерела в просторі.

4. Візуалізація Сфери:

Сфера візуалізує місцезнаходження джерела звуку у тривимірному просторі. Кожне положення сфери відображає поточне місце розташування звукового джерела, створюючи інтерактивний елемент для користувача. Це допомагає візуально сприймати рух звуку та віддавати перевагу іммерсивному аудіо-візуальному досвіду.

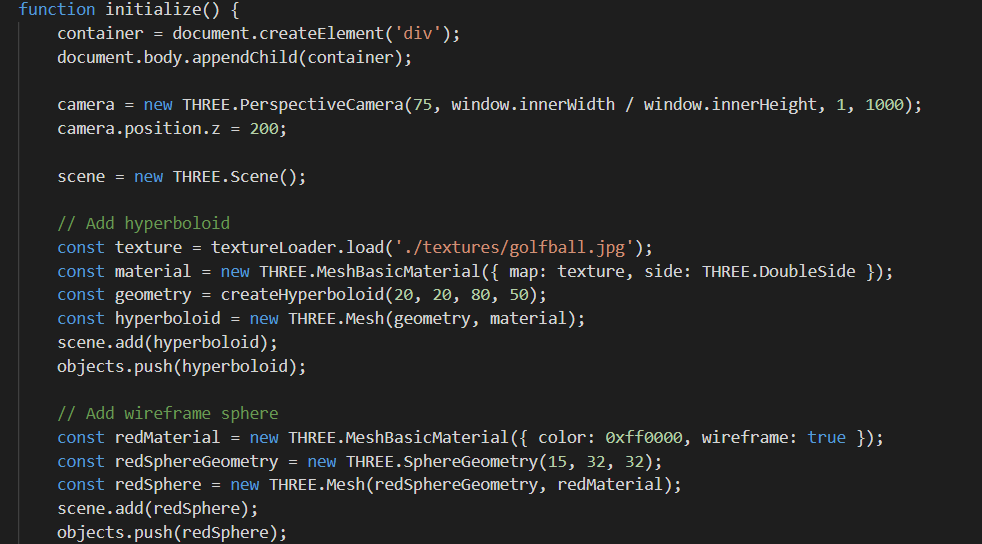
**Деталі впровадження**

Код програми представляє собою HTML-сторінку з вбудованим JavaScript, яка використовує бібліотеку Three.js для візуалізації тривимірної сцени. Основні елементи коду:

* HTML-розмітка: У блоку <body> створюється контейнер для Three.js і відображається кнопка "Play" та чекбокс "highshelf filter".
* CSS-стилі: Задаються стилі для об'єктів на сторінці, таких як положення та вигляд блоків, кнопок, чекбоксу.
* JavaScript-код:

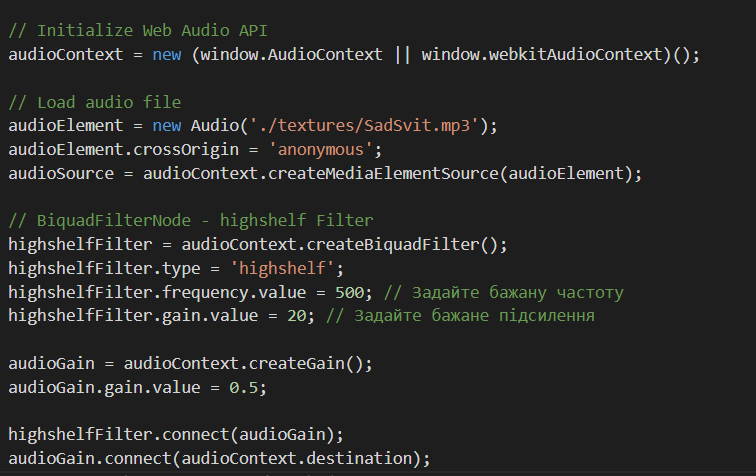
Використовується імпорт Three.js та деяких його додаткових модулів.

В цьому фрагменті коду відбувається ініціалізація сцени та об'єктів. Створюється контейнер для Three.js, камера, сцена, гіперболоїд і сфера. Гіперболоїд створюється за допомогою функції createHyperboloid, яка використовує ParametricGeometry для генерації геометрії об'єкта.

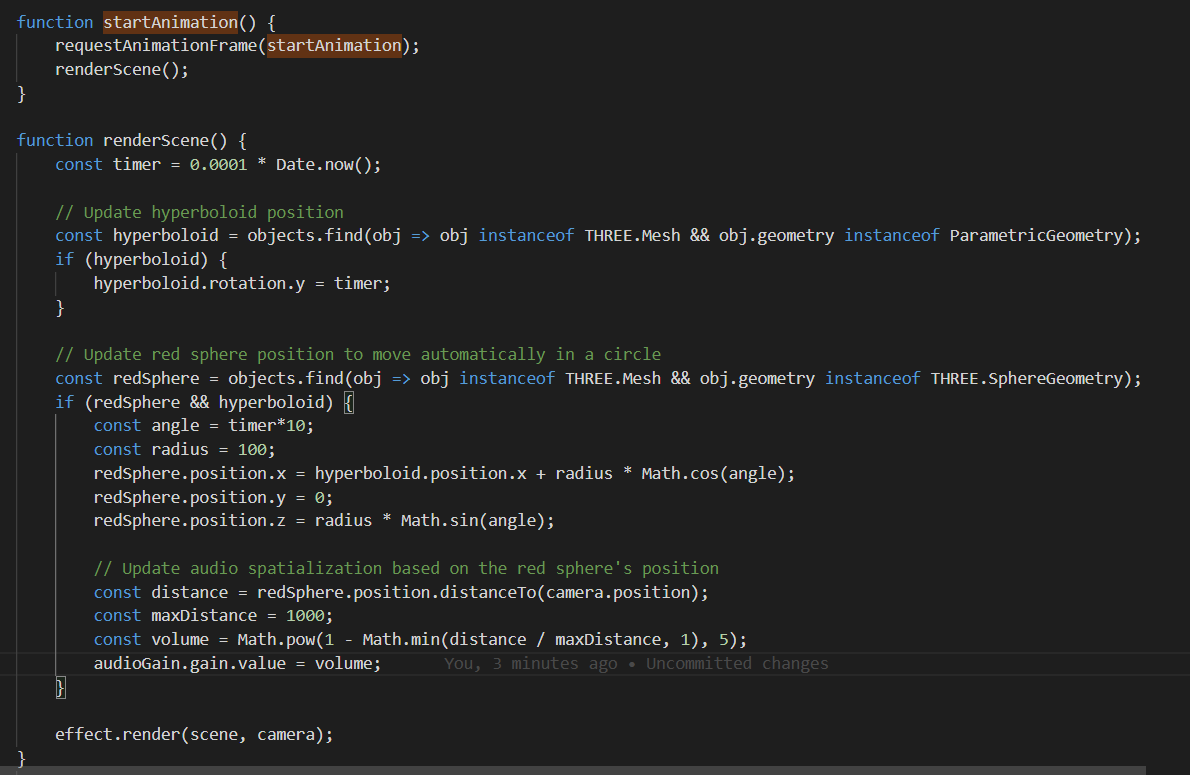


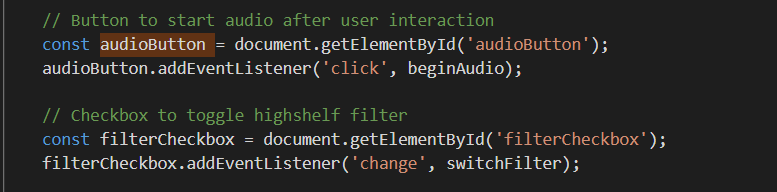
Використання Web Audio API для обробки аудіофайлу та просторової аудіо-спрямованості відносно позиції сфери:

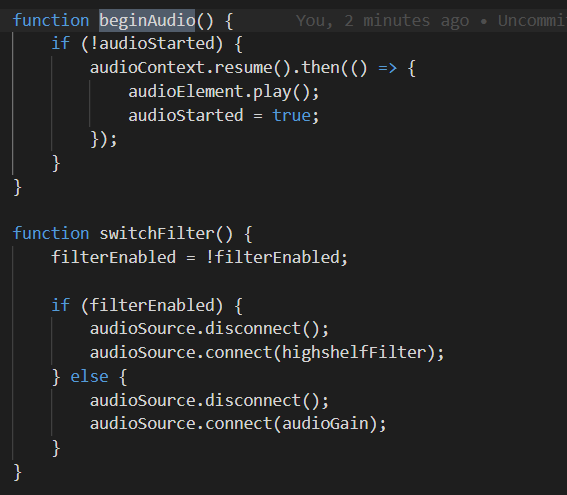
У цьому фрагменті коду створюється об'єкт AudioContext, завантажується аудіофайл, створюються об'єкти BiquadFilterNode для обробки звуку, і встановлюються звукові шляхи.



Анімація та рендеринг: Функція startAnimation використовує requestAnimationFrame для початку анімації, а функція renderScene відпововідає за оновлення позицій об'єктів та рендерінг сцени. Гіперболоїд обертається, а сфера рухається в колі. Крім того, просторова аудіо-спрямованість змінюється в залежності від позиції сфери відносно камери.

* 
* Обробка подій: Тут визначаються обробники подій для кнопки "Play" та чекбоксу "highshelf filter". Кнопка "Play" запускає аудіо, а чекбокс включає або виключає ефект "highshelf filter" для обробки звуку.

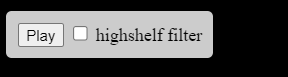


Функції для роботи з аудіо: Функція beginAudio використовується для початку відтворення аудіо після взаємодії користувача, а функція switchFilter дозволяє перемикати ефект "highshelf filter" для обробки звуку, в залежності від стану чекбоксу.

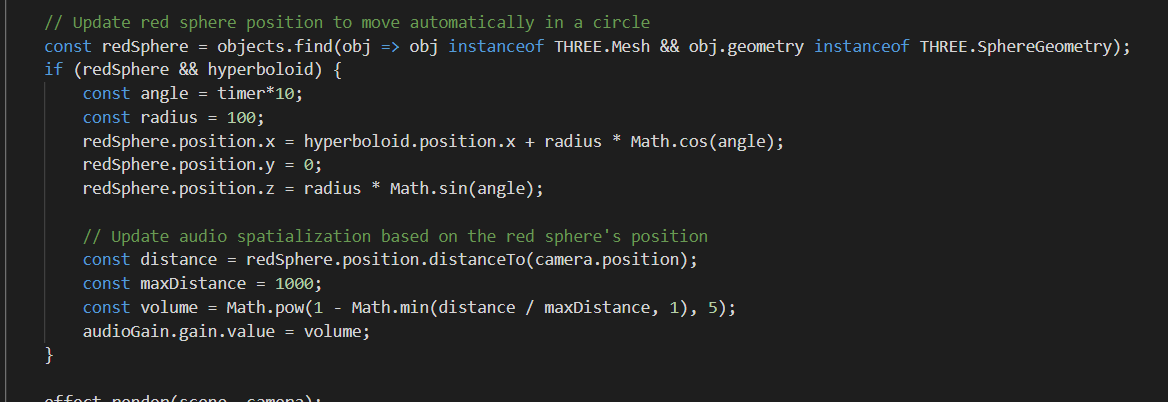
**Інструкції для користувача**

Налаштування аудіо та фільтра

* Початок відтворення аудіо:
  + Щоб розпочати відтворення аудіо потрібно натиснути кнопку "Play".
  + Чекбокс "highshelf filter" включає або виключає Шелфовий фільтр високих частотдля обробки звуку.

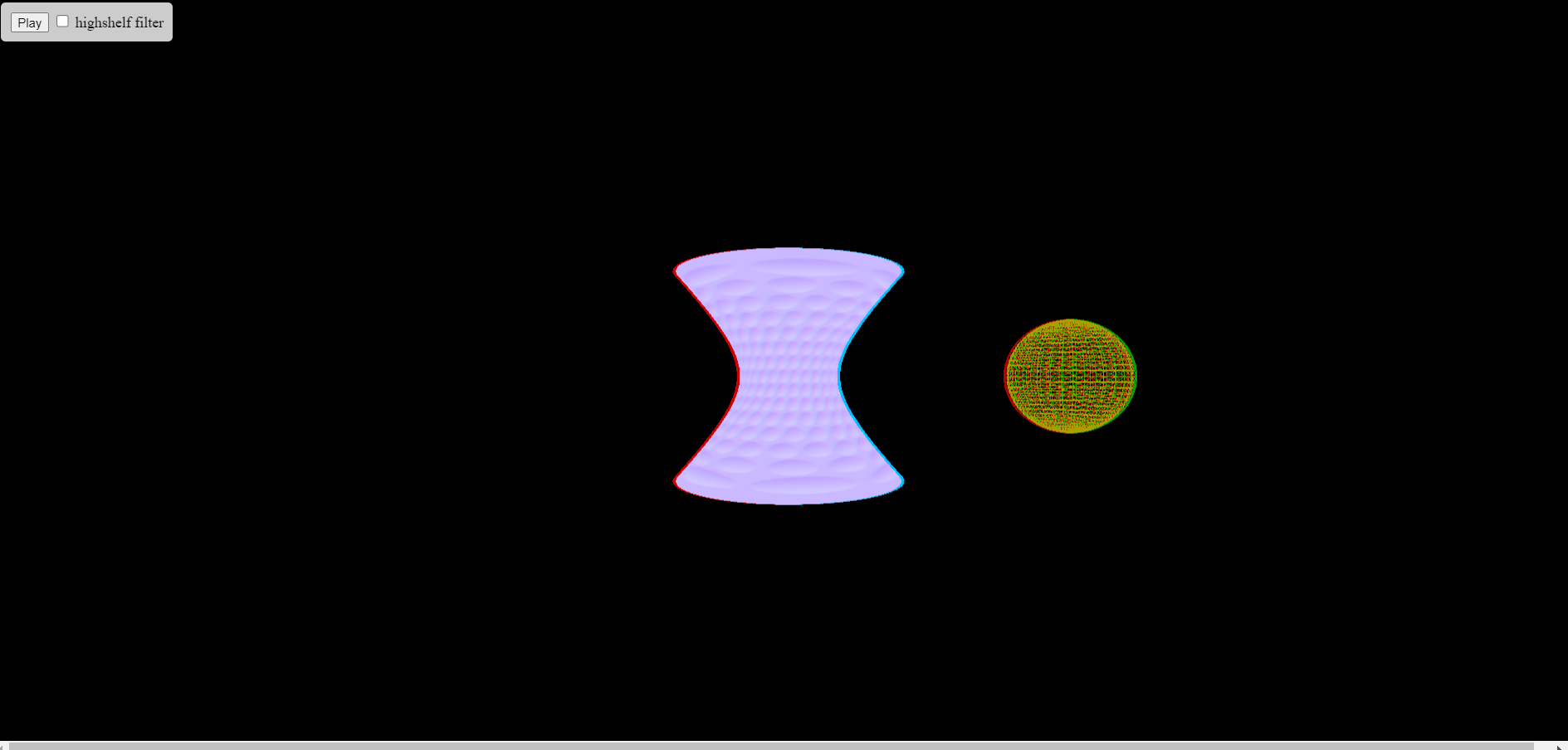


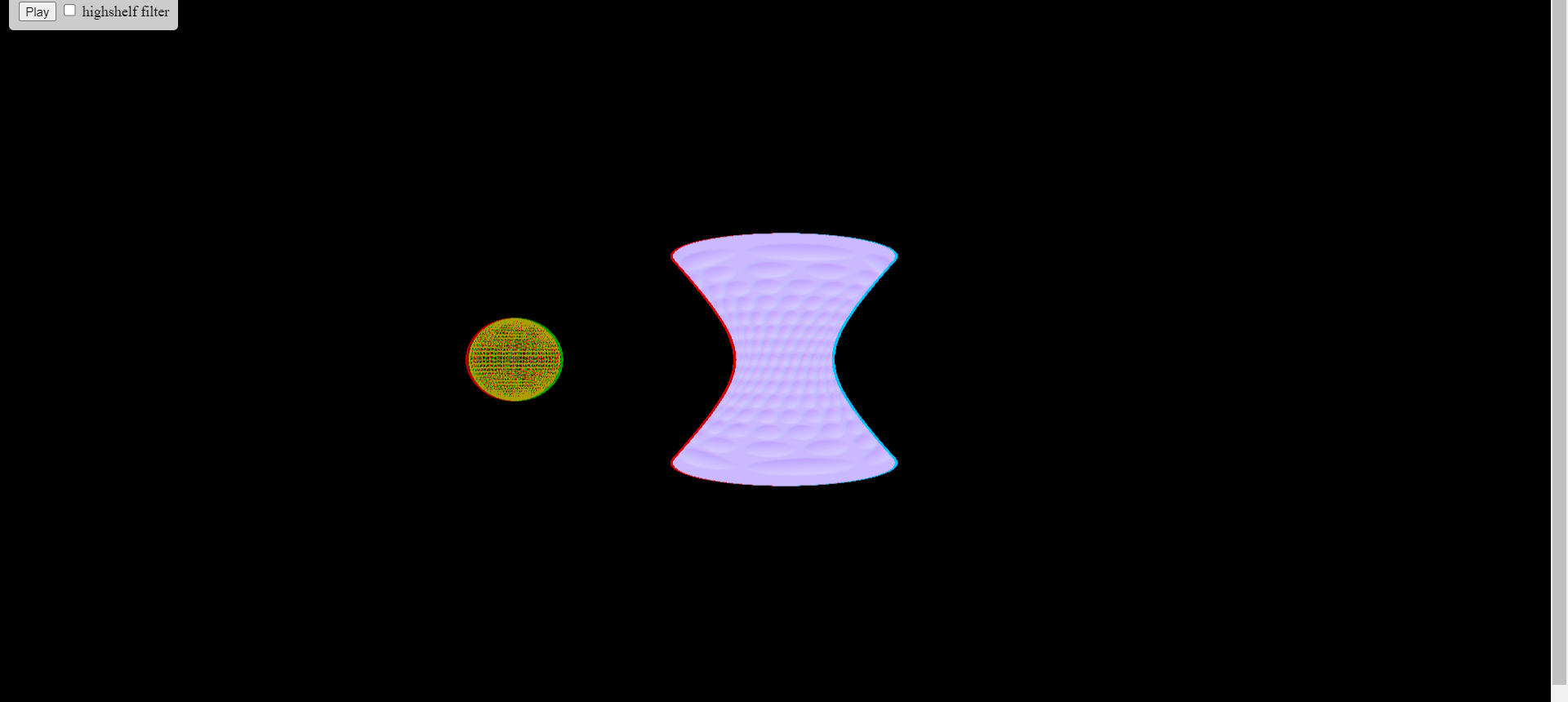
Автоматичний рух сфери можна пояснити наступним фрагментом коду:



* + Змінна angle визначає кут, який змінюється в залежності від часу.
  + Змінна radius визначає радіус кола, по якому рухається сфера.
  + Координати сфери оновлюються з використанням тригонометричних функцій (Math.cos та Math.sin), щоб визначити нове положення сфери в залежності від кута та радіусу.
* Оновлення звукової просторовості:
  + Сфера використовується для визначення звукової просторовості. Відстань від сфери до камери обчислюється, і чим ближче сфера до камери, тим більше гучності звуку встановлюється (audioGain.gain.value).
  + Гучність обчислюється так, щоб було виражене відносне приглушення гучності на основі відстані.

Ця система створює ефект руху сфери в колі і одночасно визначає звукову просторовість, залежно від положення цієї сфери відносно камери.





Вихідний код:

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<title>Textured Hyperboloid of Revolution</title>*

*<meta charset="utf-8">*

*<meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">*

*<style>*

*body { margin: 0; }*

*canvas { display: block; }*

*#fovBlock, #clippingBlock {*

*position: absolute;*

*padding: 10px;*

*background-color: rgba(255, 255, 255, 0.8);*

*border-radius: 5px;*

*}*

*#fovBlock { top: 10px; left: 10px; }*

*</style>*

*</head>*

*<body>*

*<div id="fovBlock">*

*<button id="audioButton">Play</button>*

*<!-- Checkbox to toggle bandpass filter -->*

*<input type="checkbox" id="filterCheckbox">*

*<label for="filterCheckbox">highshelf filter</label>*

*</div>*

*<script type="importmap">*

*{*

*"imports": {*

*"three": "./build/three.module.js",*

*"three/addons/": "./"*

*}*

*}*

*</script>*

*<script type="module">*

*import \* as THREE from 'three';*

*import { Anaglyph } from 'three/addons/Anaglyph.js';*

*import { ParametricGeometry } from 'three/addons/ParametricGeometry.js';*

*let container, camera, scene, renderer, effect;*

*const objects = [];*

*const textureLoader = new THREE.TextureLoader();*

*let sliderValue = 0;*

*let audioContext, audioSource, audioGain, audioElement, highshelfFilter;*

*let audioStarted = false;*

*let filterEnabled = false;*

*initialize();*

*startAnimation();*

*function initialize() {*

*container = document.createElement('div');*

*document.body.appendChild(container);*

*camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 1, 1000);*

*camera.position.z = 200;*

*scene = new THREE.Scene();*

*// Add hyperboloid*

*const texture = textureLoader.load('./textures/golfball.jpg');*

*const material = new THREE.MeshBasicMaterial({ map: texture, side: THREE.DoubleSide });*

*const geometry = createHyperboloid(20, 20, 80, 50);*

*const hyperboloid = new THREE.Mesh(geometry, material);*

*scene.add(hyperboloid);*

*objects.push(hyperboloid);*

*// Add wireframe sphere*

*const redMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000, wireframe: true });*

*const redSphereGeometry = new THREE.SphereGeometry(15, 32, 32);*

*const redSphere = new THREE.Mesh(redSphereGeometry, redMaterial);*

*scene.add(redSphere);*

*objects.push(redSphere);*

*// Initialize Web Audio API*

*audioContext = new (window.AudioContext || window.webkitAudioContext)();*

*// Load audio file*

*audioElement = new Audio('./textures/SadSvit.mp3');*

*audioElement.crossOrigin = 'anonymous';*

*audioSource = audioContext.createMediaElementSource(audioElement);*

*// BiquadFilterNode - highshelf Filter*

*highshelfFilter = audioContext.createBiquadFilter();*

*highshelfFilter.type = 'highshelf';*

*highshelfFilter.frequency.value = 500; // Задайте бажану частоту*

*highshelfFilter.gain.value = 20; // Задайте бажане підсилення*

*audioGain = audioContext.createGain();*

*audioGain.gain.value = 0.5;*

*highshelfFilter.connect(audioGain);*

*audioGain.connect(audioContext.destination);*

*// Button to start audio after user interaction*

*const audioButton = document.getElementById('audioButton');*

*audioButton.addEventListener('click', beginAudio);*

*// Checkbox to toggle highshelf filter*

*const filterCheckbox = document.getElementById('filterCheckbox');*

*filterCheckbox.addEventListener('change', switchFilter);*

*// Check filter state and set filter by default*

*if (filterEnabled) {*

*audioSource.connect(highshelfFilter);*

*filterCheckbox.checked = true;*

*} else {*

*audioSource.connect(audioGain);*

*}*

*renderer = new THREE.WebGLRenderer();*

*renderer.setPixelRatio(window.devicePixelRatio);*

*container.appendChild(renderer.domElement);*

*const width = window.innerWidth || 2;*

*const height = window.innerHeight || 2;*

*effect = new Anaglyph(renderer, 0.064);*

*effect.setSize(width, height);*

*}*

*function onSliderChange(event) {*

*sliderValue = parseFloat(event.target.value);*

*}*

*function beginAudio() {*

*if (!audioStarted) {*

*audioContext.resume().then(() => {*

*audioElement.play();*

*audioStarted = true;*

*});*

*}*

*}*

*function switchFilter() {*

*filterEnabled = !filterEnabled;*

*if (filterEnabled) {*

*audioSource.disconnect();*

*audioSource.connect(highshelfFilter);*

*} else {*

*audioSource.disconnect();*

*audioSource.connect(audioGain);*

*}*

*}*

*function resizeWindow() {*

*camera.aspect = window.innerWidth / window.innerHeight;*

*camera.updateProjectionMatrix();*

*effect.setSize(window.innerWidth, window.innerHeight);*

*}*

*function startAnimation() {*

*requestAnimationFrame(startAnimation);*

*renderScene();*

*}*

*function renderScene() {*

*const timer = 0.0001 \* Date.now();*

*// Update hyperboloid position*

*const hyperboloid = objects.find(obj => obj instanceof THREE.Mesh && obj.geometry instanceof ParametricGeometry);*

*if (hyperboloid) {*

*hyperboloid.rotation.y = timer;*

*}*

*// Update red sphere position to move automatically in a circle*

*const redSphere = objects.find(obj => obj instanceof THREE.Mesh && obj.geometry instanceof THREE.SphereGeometry);*

*if (redSphere && hyperboloid) {*

*const angle = timer\*10;*

*const radius = 100;*

*redSphere.position.x = hyperboloid.position.x + radius \* Math.cos(angle);*

*redSphere.position.y = 0;*

*redSphere.position.z = radius \* Math.sin(angle);*

*// Update audio spatialization based on the red sphere's position*

*const distance = redSphere.position.distanceTo(camera.position);*

*const maxDistance = 1000;*

*const volume = Math.pow(1 - Math.min(distance / maxDistance, 1), 5);*

*audioGain.gain.value = volume;*

*}*

*effect.render(scene, camera);*

*}*

*function createHyperboloid(a, b, height, segments) {*

*const geometry = new ParametricGeometry((u, v, target) => {*

*const theta = u \* 2 \* Math.PI;*

*const y = v \* height - height / 2;*

*const x = a \* Math.sqrt(1 + (y \* y) / (b \* b)) \* Math.cos(theta);*

*const z = a \* Math.sqrt(1 + (y \* y) / (b \* b)) \* Math.sin(theta);*

*target.set(x, y, z);*

*}, segments, segments);*

*return geometry;*

*}*

*</script>*

*</body>*

*</html>*