НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

Розрахунково графічна робота

З дисципліни: «Методи синтезу віртуальної реальності»

Варіант 12

**Виконав:**

студент 5 курсу

групи ТР-23мп, ТЕФ

Кондратюк Ігор Віталійович

**Перевірив:**

Демчишин А. А.

Київ – 2023

**Завдання**

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). Відтворюйте улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем;
3. Візуалізувати положення джерела звуку за допомогою сфери;
4. Додайте звуковий фільтр (використовуйте інтерфейс BiquadFilterNode) для кожного варіанту . Додайте елемент прапорця, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак. **Варіант 12 «Шельфовий фільтр високих частот»**

**Теоретична інформація**

Просторове аудіо в WebAudio HTML5 API використовується для створення іммерсивного звукового середовища у веб-додатках. Ця технологія дозволяє розміщувати звукові джерела у тривимірному просторі, що дозволяє точно визначити їх положення та рух.

Основні поняття, пов'язані з просторовим аудіо, включають:

1. Аудіо-контекст (AudioContext): Це головний об'єкт, який ініціалізує та керує аудіо-процесором. Він представляє собою центральну точку для створення та маніпулювання аудіо-об'єктами.
2. Звукові джерела (Audio Sources): Це аудіо-джерела, які можуть бути розміщені в тривимірному просторі. Вони можуть бути представлені як різні типи об'єктів, такі як аудіо-елементи, буфери аудіо та синтезовані звуки.
3. Просторові властивості: Кожне звукове джерело має просторові властивості, такі як позиція (x, y, z), швидкість руху, орієнтація та розмір зони поширення звуку. Ці властивості використовуються для створення ефектів просторового звуку, таких як звучання з різних напрямків та відстані.
4. Аудіо-простір (Audio Space): Весь простір, в якому розміщуються звукові джерела, представляє аудіо-простір. Він може бути тривимірним, де координати визначають положення звукових джерел, а також служити для визначення властивостей пов'язаних зі звуком об'єктів, таких як слухач (listener).
5. Слухач (Listener): Це об'єкт, який представляє точку прослуховування в аудіо-просторі. Він може бути рухливим і мати свою власну позицію та орієнтацію. Звукові джерела відтворюються відносно цієї точки, що створює враження звуку, що наближається або віддаляється від слухача.
6. Просторові аудіо-ефекти: WebAudio HTML5 API надає набір ефектів, які можуть бути застосовані до звукових джерел. Це включає панорамування (panning), просторову еквалізацію, розміщення віртуальних мікрофонів та багато іншого.

За допомогою просторового аудіо в WebAudio HTML5 API можна створювати реалістичні звукові ефекти, імітуючи звучання об'єктів у реальному світі. Він дозволяє розміщувати звукові джерела у тривимірному просторі та контролювати їх положення, рух та взаємодію зі слухачем, створюючи захоплюючий аудіальний досвід для користувачів веб-додатків.

**Деталі реалізації**

Для реалізації просторового аудіо в HTML5 API я використав мову програмування JavaScript та бібліотеку WebGL для візуалізації джерела звуку. Процес складається з таких етапів:

1. Перше, що потрібно зробити, це додати елемент «audio» та mp3 файл до нього на html-сторінку. А також допоміжні inputs для управління джерелом звуку.
2. Створюємо сферу для візуалізації джерела звуку та додаємо автоматичне і ручне її положення. Розіб’ємо даний процес на підпункти:

* Додаємо функцію для побудови сфери:

function createSphere(radius, latitudeBands, longitudeBands) {  
 const positions = [];  
  
 for (let lat = 0; lat <= latitudeBands; lat++) {  
 const theta = (lat \* ***Math***.PI) / latitudeBands;  
 const sinTheta = ***Math***.sin(theta);  
 const cosTheta = ***Math***.cos(theta);  
  
 for (let lon = 0; lon <= longitudeBands; lon++) {  
 const phi = (lon \* 2 \* ***Math***.PI) / longitudeBands;  
 const sinPhi = ***Math***.sin(phi);  
 const cosPhi = ***Math***.cos(phi);  
  
 const x = radius \* cosPhi \* sinTheta;  
 const y = radius \* cosTheta;  
 const z = radius \* sinPhi \* sinTheta;  
  
 positions.push(x, y, z);  
 }  
 }  
  
 return positions;  
}

* Додаємо траекторію руху сфери (коло):

function drawCircle(currentSide, numSides, centerX, centerZ, radius) {  
 const angle = (currentSide / numSides) \* ***Math***.PI \* 2;  
 const x = centerX + ***Math***.cos(angle) \* radius;  
 const z = centerZ + ***Math***.sin(angle) \* radius;  
 ***xPosition*** = x;  
 ***zPosition*** = z;  
}

* Налаштовуємо автоматичну зміну положення сфери (рух по колу), якщо активний відповідний чекбокс:

function autoPositionChange() {  
 const centerX = 0;  
 const centerZ = 0;  
 const radius = 1;  
 const numSides = 20;  
 let currentSide = 0;  
  
 ***changePositionInterval*** = autoInterval(currentSide, numSides, centerX, centerZ, radius);  
   
 let autoChangePosition = ***document***.getElementById('autoChangePosition');  
 autoChangePosition.addEventListener('change', function() {  
 if (autoChangePosition.checked) {  
 ***changePositionInterval*** = autoInterval(currentSide, numSides, centerX, centerZ, radius);  
 } else {  
 clearInterval(***changePositionInterval***);  
 }  
 });  
}

1. Виконую налаштування аудіо через WebAudio HTML5 API, за наступними кроками:

* Створюємо аудіо-контекст за допомогою new (window.AudioContext || window.webkitAudioContext)(), який забезпечує інтерфейс для роботи з аудіо в браузері. Потім створюються та налаштовуються аудіо-вузли, такі як джерело звуку source, панорама panner і фільтр filter.
* Підключаємо звукове джерело до панорами, панорама підключається до фільтра, а фільтр підключається до аудіо-контексту, який потім направляє звук на вихідний пристрій (audioContext.destination). За варіантом налаштовую параметри фільтра (Шельфовий фільтр високих частот) filter.type = 'highshelf'. Наступні параметри налаштовую за вподобанням: частоту filter.frequency.value = 1000 і підсилення filter.gain.value = 15. Після цього продовжую роботу аудіо-контексту (audioContext.resume()).

***filter***.type = 'highshelf';  
***filter***.frequency.value = 1000;  
***filter***.gain.value = 15;

* Далі налаштовуємо фінальну функцію для аудіо startAudio(), яка викликає функцію AudioSetup() та регулює додання фільтру в залежності від значення чекбоксу в input.

1. Додаємо зміну самого положення звуку, в залежності від заданих координат:

if (***panner***) {  
 let x = parseFloat(***xPosition***);  
 let y = parseFloat(***yPosition***);  
 let z = parseFloat(***zPosition***);  
 ***panner***.setPosition(x, y, z);  
}

**Інструкція користувача**

Нижче наведені зображення для користування та демонстрації функціоналу програми.



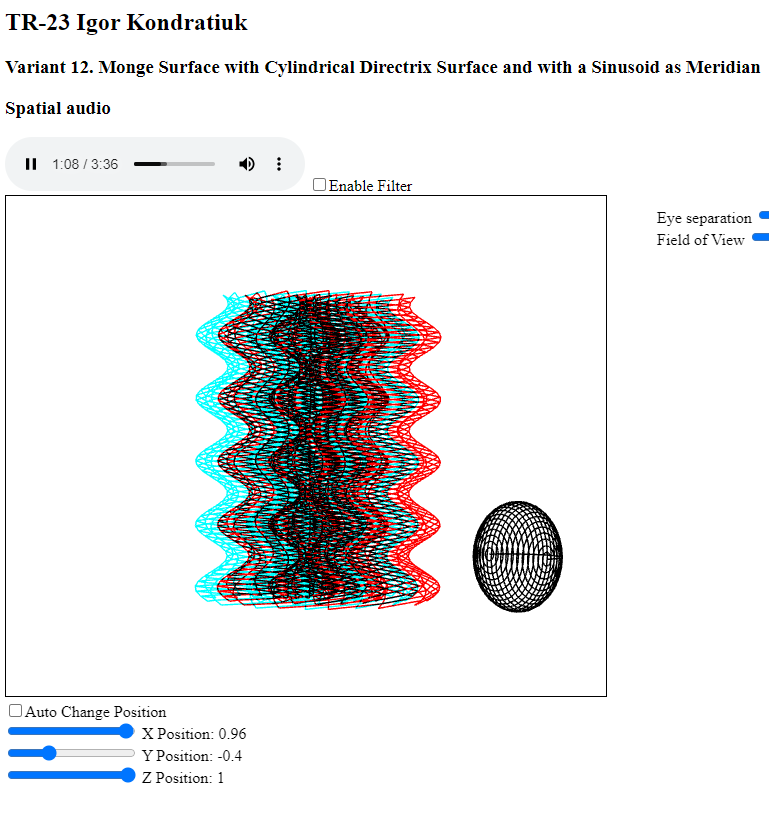
*Рис.1 Приклад початкового вигляду*

При вимкненні чекбоксу «Auto Change Position», джерело звуку та сфера перестають змінювати своє положення. І далі за допомогою зміни параметрів inputs (X Position, Y Position, Z Position) користувач може змінювати позицію джерела звуку. Відповідно і сфера, яка відображає джерело звуку, також змінить своє положення.



*Рис.2 Приклад ручної зміни джерела звуку*

Фільтр можна вимикати і вмикати під час відтворення аудіо за допомогою чекбоксу «Enable Filter».



*Рис.3 Приклад відключення фільтра (Шельфовий фільтр високих частот)*

**Вихідний код**

// Sphere Setup  
function updatePosition() {  
 const xPositionInput = ***document***.getElementById('xPosition');  
 xPositionInput.addEventListener('input', handleInputChange);  
  
 const yPositionInput = ***document***.getElementById('yPosition');  
 yPositionInput.addEventListener('input', handleInputChange);  
  
 const zPositionInput = ***document***.getElementById('zPosition');  
 zPositionInput.addEventListener('input', handleInputChange);  
}  
  
// Handle user input changes  
function handleInputChange(event) {  
 switch (event.target.id) {  
 case 'xPosition':  
 ***xPosition*** = parseFloat(event.target.value);  
 break;  
 case 'yPosition':  
 ***yPosition*** = parseFloat(event.target.value);  
 break;  
 case 'zPosition':  
 ***zPosition*** = parseFloat(event.target.value);  
 break;  
 default:  
 alert('Error')  
 break;  
 }  
  
 updatePosition();  
}  
  
function AudioSetup() {  
 ***audio*** = ***document***.getElementById('audio');  
  
 ***audio***.addEventListener('play', () => {  
 ***console***.log('play');  
 if (!***audioContext***) {  
 ***audioContext*** = new (***window***.***AudioContext*** || ***window***.webkitAudioContext)();  
 ***source*** = ***audioContext***.createMediaElementSource(***audio***);  
 ***panner*** = ***audioContext***.createPanner();  
 ***filter*** = ***audioContext***.createBiquadFilter();  
  
 // Connect audio nodes  
 ***source***.connect(***panner***);  
 ***panner***.connect(***filter***);  
 ***filter***.connect(***audioContext***.destination);  
  
 // highshelf filter parameters  
 ***filter***.type = 'highshelf';  
 ***filter***.frequency.value = 1000;  
 ***filter***.gain.value = 15;  
 ***audioContext***.resume();  
 }  
 })  
  
  
 ***audio***.addEventListener('pause', () => {  
 ***console***.log('pause');  
 ***audioContext***.resume();  
 })  
}  
  
function startAudio() {  
 AudioSetup();  
  
 let filterCheckbox = ***document***.getElementById('filterCheckbox');  
 filterCheckbox.addEventListener('change', function() {  
 if (filterCheckbox.checked) {  
 // Connect filter when checkbox is checked  
 ***panner***.disconnect();  
 ***panner***.connect(***filter***);  
 ***filter***.connect(***audioContext***.destination);  
 } else {  
 // Disconnect filter when checkbox is unchecked  
 ***panner***.disconnect();  
 ***panner***.connect(***audioContext***.destination);  
 }  
 });  
  
 ***audio***.play();  
}  
  
function createSphere(radius, latitudeBands, longitudeBands) {  
 const positions = [];  
  
 for (let lat = 0; lat <= latitudeBands; lat++) {  
 const theta = (lat \* ***Math***.PI) / latitudeBands;  
 const sinTheta = ***Math***.sin(theta);  
 const cosTheta = ***Math***.cos(theta);  
  
 for (let lon = 0; lon <= longitudeBands; lon++) {  
 const phi = (lon \* 2 \* ***Math***.PI) / longitudeBands;  
 const sinPhi = ***Math***.sin(phi);  
 const cosPhi = ***Math***.cos(phi);  
  
 const x = radius \* cosPhi \* sinTheta;  
 const y = radius \* cosTheta;  
 const z = radius \* sinPhi \* sinTheta;  
  
 positions.push(x, y, z);  
 }  
 }  
  
 return positions;  
}  
  
function autoPositionChange() {  
 const centerX = 0;  
 const centerZ = 0;  
 const radius = 1;  
 const numSides = 20;  
 let currentSide = 0;  
  
 ***changePositionInterval*** = autoInterval(currentSide, numSides, centerX, centerZ, radius);  
  
 let autoChangePosition = ***document***.getElementById('autoChangePosition');  
 autoChangePosition.addEventListener('change', function() {  
 if (autoChangePosition.checked) {  
 ***changePositionInterval*** = autoInterval(currentSide, numSides, centerX, centerZ, radius);  
 } else {  
 clearInterval(***changePositionInterval***);  
 }  
 });  
}  
  
function drawCircle(currentSide, numSides, centerX, centerZ, radius) {  
 const angle = (currentSide / numSides) \* ***Math***.PI \* 2;  
 const x = centerX + ***Math***.cos(angle) \* radius;  
 const z = centerZ + ***Math***.sin(angle) \* radius;  
 ***xPosition*** = x;  
 ***zPosition*** = z;  
}  
  
function autoInterval(currentSide, numSides, centerX, centerZ, radius) {  
 return setInterval(() => {  
 if (currentSide < numSides) {  
 drawCircle(currentSide, numSides, centerX, centerZ, radius);  
 currentSide++;  
 } else {  
 currentSide = 0;  
 drawCircle(currentSide, numSides, centerX, centerZ, radius)  
 }  
 }, 200);  
}