Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра цифрових технологій в енергетиці

# «Методи синтезу віртуальної реальності»

Розрахунково-графічна робота

Виконав:

студент групи ТР-21мп

Узун А.С.

Варіант 25

Перевірив:

Демчишин А.А.

Київ – 2023

# Постановка задачі

Імплементувати шелфовий фільтр низьких частот.

- Завдання повинно бути завантажено в репозиторій на GitHub
- Завдання повинно міститися в гілці, що має назву CGW
- В репозиторії повинен міститися звіт до розрахунково-графічної роботи
- реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою інтерфейсу сенсора (цього разу поверхня залишається нерухомою, а джерело звуку рухається).
- Відтворити улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем;
- візуалізувати положення джерела звуку за допомогою сфери; додайте звуковий фільтр (використовуйте інтерфейс BiquadFilterNode) для кожного варіанту.
- додати шельфовий фільтр низьких частот.
- додати перемикач, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак.

# Теорія

Шельфовий фільтр низьких частот, також відомий як lowshelf фільтр, є одним з основних типів фільтрів, які використовуються в аудіо обробці. Він впливає на спектральний зміст звуку, знижуючи амплітуду вищих частот і залишаючи низькі частоти без змін.

У web audio API, яка є стандартною API для обробки аудіо веб-додатків, також підтримується шельфовий фільтр низьких частот. Це дозволяє розробникам створювати веб-додатки, які можуть змінювати спектральний зміст аудіо сигналу в режимі реального часу.

Шельфовий фільтр низьких частот в web audio API може бути використаний для обрізання амплітуди вищих частот або для створення ефекту "теплого" звуку, додавання басів чи приглушення яскравості звучання. Цей фільтр має параметри, такі як "підсилення" (gain), яке визначає магнітуду зміни амплітуди, та "частота зрізу" (cutoff frequency), яка визначає точку, де починається зниження амплітуди.

Web audio API також надає інші типи фільтрів, такі як highshelf фільтр (шельфовий фільтр високих частот) і peaking фільтр, які дозволяють розробникам змінювати спектральний зміст звуку в більш гнучкий спосіб.

Загалом, шельфовий фільтр низьких частот в web audio API є потужним інструментом для обробки аудіо сигналів у веб-додатках, дозволяючи змінювати спектральні характеристики звуку для досягнення бажаного звукового ефекту. Використання цього фільтру дозволяє додати гнучкості і творчого потенціалу до веб-аудіо додатків.

WebGL (Web Graphics Library) - це JavaScript API, який дозволяє рендерити 2D та 3D графіку у веб-браузері без необхідності використання плагінів. Він базується на мові програмування OpenGL ES, яка використовується для розробки графічних додатків для мобільних пристроїв.

WebGL надає потужні можливості для візуалізації графіки, створення 3D сцен та взаємодії з ними. За допомогою WebGL можна відображати 3D моделі, створювати анімації, застосовувати текстури, освітлення та ефекти. Веб-розробники можуть використовувати WebGL для створення ігор, віртуальної реальності, додатків для обробки об'ємних даних та багато іншого.

Для роботи з WebGL використовуються шейдери - мали програми, які працюють на графічних процесорах і відповідають за обробку графічних об'єктів. Шейдери дозволяють контролювати процеси рендерингу, включаючи освітлення, тіні, текстури, колір та матеріали.

WebGL є потужним інструментом для створення веб-графіки високої якості. Він надає можливість створювати інтерактивні та захоплюючі веб-додатки, які працюють безпосередньо у браузері користувача, без необхідності встановлення додаткових програм або плагінів. WebGL використовує апаратну акселерацію графіки, що дозволяє досягати високої продуктивності та швидкодії рендерингу.

Загалом, WebGL відкриває широкі можливості для створення вражаючих графічних веб-додатків, розширюючи межі можливостей браузера і дозволяючи розробникам втілити свої творчі ідеї у сучасних веб-проектах.

## Хід роботи

У ході виконання роботи було розроблено програмний продукт, який реалізовує просторове аудіо за допомогою веб-технологій Web Audio API та WebGL. Основною метою проекту було створення візуального та звукового ефекту, де звукові об'єкти уявно розміщені у просторі та звучать залежно від їх позиції.

Починаючи з реалізації веб-додатку, було створено HTML-сторінку, яка містить необхідні елементи для відображення візуального та аудіо вмісту. За допомогою WebGL було відрендерено просту фігуру, наприклад, куб або сферу, за допомогою функції gl.drawArrays().

Навколо цієї фігури було створено сферу, яка відтворює роль об'єкту у просторі, і можна керувати її позицією за допомогою акселерометра.

Ось приклад JavaScript-коду для отримання даних з акселерометра:

```
// Отримання доступу до акселерометра

if (window.DeviceMotionEvent) {

  window.addEventListener('devicemotion', handleMotion);

} else {

  console.log('Akselerometera nav!');

}


// Обробник події руху пристрою

function handleMotion(event) {

  const acceleration = event.accelerationIncludingGravity;

  const x = acceleration.x;

  const y = acceleration.y;

  const z = acceleration.z;
```

```
  // Виконати необхідні дії зі значеннями акселерометра
  // Наприклад, оновлення позиції сфери у просторі
}
```

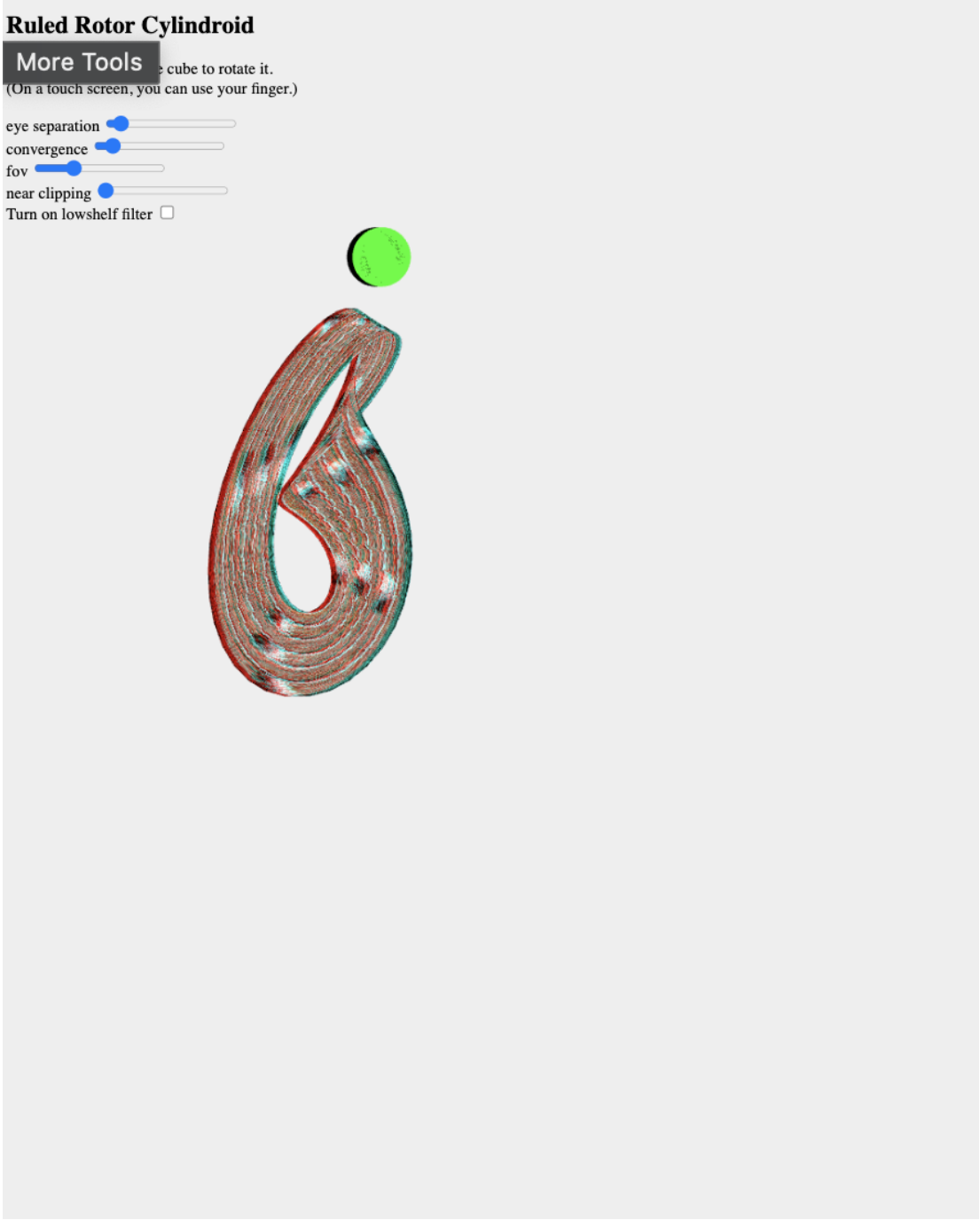
Для досягнення просторового звукового ефекту, використовувався Web Audio API. Було створено аудіо контекст, в якому були налаштовані різні параметри звуку, наприклад, гучність та частоти. Координати сфери, яка відображала позицію звуку, були передані до об'єкта PannerNode, який відповідав за розташування звукового джерела у просторі.

Також, було реалізовано функцію увімкнення та вимкнення шелфового фільтру низьких частот (lowshelf filter).

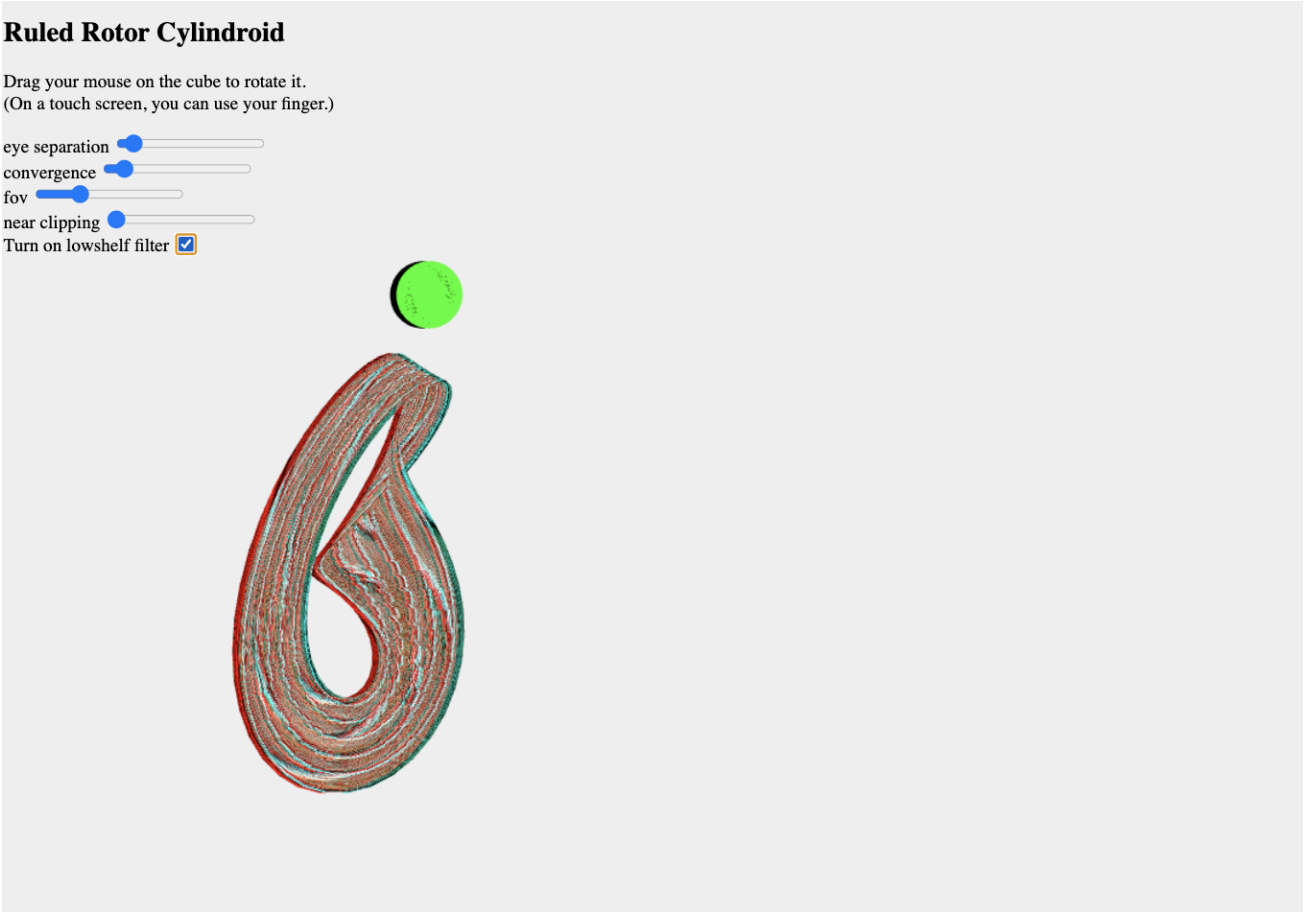
У результаті виконання лабораторної роботи було досягнуто мети створення програмного продукту, який реалізовує просторове аудіо за допомогою веб-технологій Web Audio API та WebGL. Цей продукт дозволяє створювати звукові ефекти, які змінюються залежно від позиції звукового джерела у просторі та можуть бути керовані за допомогою акселерометра. Також він надає можливість увімкнення та вимкнення шелфового фільтру низьких частот, що дозволяє змінювати еквалайзер звуку та створювати різні звукові ефекти для користувача.

```
// Створення lowshelf фільтру
const filter = audioContext.createBiquadFilter();
filter.type = 'lowshelf';
// Увімкнення шелфового фільтру
function enableLowshelfFilter() {
  filter.frequency.value = 100; // Налаштування частоти
  filter.gain.value = 10; // Налаштування підсилення
  source.connect(filter);
```

# Скриноти виконання




На рисунку зображено фігуру зі сферою, що обертається навколо неї.




На рисунку зображено увімкнений шельфовий фільтр низьких частот.

# Програмний код

```
function CreateSphereSurface(r = 0.2) {
	let vertexList = [];
	let lon = -Math.PI;
	let lat = -Math.PI * 0.5;
	while (lon < Math.PI) {
	while (lat < Math.PI * 0.5) {
		let ty = sphereSurfaceDate(r, lon, lat);
		vertexList.push(ty.x, ty.y, ty.z);
		lat += 0.05;
	}
	lat = -Math.PI * 0.5
	lon += 0.05;
	}
	return vertexList;
}

function sphereSurfaceDate(r, u, v) {
  const offset = 1.8;
	let x = r * Math.sin(u) * Math.cos(v) + offset;
	let y = r * Math.sin(u) * Math.sin(v) + offset;
	let z = r * Math.cos(u) + offset;
	return { x: x, y: y, z: z };
}

// Constructor
function ShaderProgram(name, program) {

	this.name = name;
	this.prog = program;

	// Location of the attribute variable in the shader program.
	this.iAttribVertex = -1;
	this.iAttribTexture = -1;
	// Location of the uniform specifying a color for the primitive.
	this.iColor = -1;
	// Location of the uniform matrix representing the combined
transformation.
	this.iModelViewProjectionMatrix = -1;
	this.iTranslatePoint = -1;
	this.iTexturePoint = -1;
	this.iRotateValue = -1;
	this.iTMU = -1;

	this.Use = function () {
```

```
	gl.useProgram(this.prog);
	}
}

function leftFrustum(stereoCamera) {
	const { eyeSeparation, convergence, aspectRatio, fov, near, far } =
stereoCamera;
	const top = near * Math.tan(fov / 2);
	const bottom = -top;

	const a = aspectRatio * Math.tan(fov / 2) * convergence;
	const b = a - eyeSeparation / 2;
	const c = a + eyeSeparation / 2;

	const left = -b * near / convergence;
	const right = c * near / convergence;

	return m4.frustum(left, right, bottom, top, near, far);
}

function rightFrustum(stereoCamera) {
	const { eyeSeparation, convergence, aspectRatio, fov, near, far } =
stereoCamera;
	const top = near * Math.tan(fov / 2);
	const bottom = -top;

	const a = aspectRatio * Math.tan(fov / 2) * convergence;
	const b = a - eyeSeparation / 2;
	const c = a + eyeSeparation / 2;

	const left = -c * near / convergence;
	const right = b * near / convergence;
	return m4.frustum(left, right, bottom, top, near, far);
}
function Model(name) {
	this.name = name;
	this.iVertexBuffer = gl.createBuffer();
	this.count = 0;
	this.iTextureBuffer = gl.createBuffer();

	this.BufferData = function (vertices) {

	gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
	gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STREAM_DRAW);
```