МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

# Розрахунково-графічна робота
з дисципліни «Методи синтезу віртуальної реальності»

Варіант №2

**Виконав:**
Студент 2-го курсу магістратури
ІАТЕ
групи ТР-22мп
Бачинський В. І.

**Перевірив:**
Демчишин А.А.

Київ-2023

# Завдання

Використавши код із практичного завдання №2:

- 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 перемикач, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак.
- Згідно до варіанту додати фільтр високих частот.

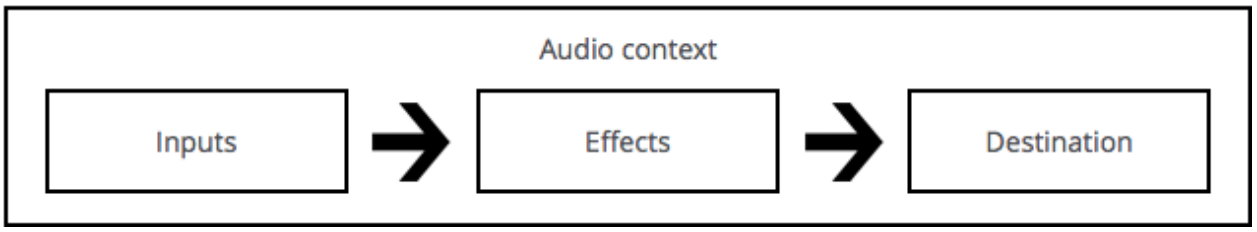
# Опис Теорії

Web audio API - потужний і багатогранний інструмент для маніпуляції звуковими компонентами на веб-сторінці, який дає можливість розробникам вибирати джерела, додавати до них спеціальні звукові ефекти (такі як панорамування), візуалізувати їх і багато іншого.

Загальні концепції та використання веб-аудіо API веб-аудіо дозволяє обробляти операції над аудіо за допомогою спеціального аудіоконтексту (аудіоконтексту) і спроектовано з використанням модульної маршрутизації (модульної маршрутизації). Базові операції виконуються за допомогою аудіо вузлів (аудіовузлів), які об'єднуються разом, формуючи аудіо-маршрутизатор *таблицю (граф маршрутизації аудіо). Нескілько джерел - з різними видами поточних схем - підтримуються навіть усередині простого контексту. Ця модульна концепція забезпечує гнучкість у створенні складних функцій для динамічних ефектів. Аудіо узли об'єднуються в ціпи і прості мережі їх вводами і виводами. Вони, як правило, запускаються з одним або більш джерелами. Источники являють собою масиви семплів на одиницю часу. Наприклад, при частоті дискретизації 44100 Гц, у кожній секунді кожного каналу розміщено 22050 семплів. Вони можуть бути або оброблені математики (див.: OscillatorNode), або вважатися звуковими/відеофайлами (див.: AudioBufferSourceNode і MediaElementAudioSourceNode) або з аудіопотоками (див.: MediaStreamAudioSourceNode). По суті, звукові файли - просто запис звукових коливань, які виходять від мікрофона та музичних інструментів, змішуючи в одну складну хвилю. Виведені дані цих узлів можуть бути пов'язані з вхідними іншими, які змішують або модифікують потоки звукових зразків в інших потоках. Популярна модифікація - збільшення зразка на значення, щоб зробити вихідний звук менш або більш громким (дивіться : GainMode). Коли звук був успішно оброблений призначеним йому ефектом, він може бути прив’язаний до вихідного потоку (дивіться : AudioContext.destination), який направляє звук в динаміку або

мікрофон. Цей крок потрібен, лише якщо ви краще дати можливість користувачеві услишати ваші шедеври. Простой, типовий порядок виконання маніпуляцій над звуком виглядає так:

1. Створюємо звуковий контекст
2. Всередині нашого контексту визначені джерела - такі як <аудіо>, генератор (осцилятор), потік
3. Определим узлы эффектов, таких как реверберация (reverb), біквадратний фільтр (biquad filter), панорамирование (panner), стиснення (компресор)
4. Вибираємо конечну точку аудіосигналу, наприклад ваші системні звукові пристрої
5. Прив'язуємо наші джерела до ефектів, і ефекти до кінцевого сигналу.




Розподіл часу контролюється з високою точністю і низькими затримками, що дозволяє розробникам писати код, який точно реагує на подію і в стані опрацювати образець навіть на високій кількості зразків (частота дискретизації). Так що такі додатки як ритм-комп'ютер і програмний автомат завжди під рукою. API веб-аудіо також дає нам можливість контролювати те, яке аудіо є в просторі. Використовуючи особливу систему, яка базується на моделі джерело-слухач, вона дозволяє контролювати модель панорамування та обходитися без дистанційно-визваного ослаблення (загасання, викликаного відстанню) або дупплерівського зсуву, викликаного здвигом джерела (або здвигом слухача).

## Деталі Реалізації

Для виконання цієї роботи було ознайомлено та використано технологію Web Audio API. У найпростішій формі, його можна використати наступним чином:

```
const audioCtx = new AudioContext();
const analyser = audioCtx.createAnalyser();
```

Однак для того щоб запустити аудіо доріжку - її потрібно спочатку завантажити. Для цього необхідно використати статичний веб-сервер, до якого буде звертатися клієнт. Для цього можна використати будь який сервер, у даному випадку був використаний sirv-cli, перевагою якого є простота адже його можна запустити з командного рядку. Для завантаження використовувався сучасний fetchAPI, який підтримують всі сучасні веб-браузери. Головною характеристикою цього АПІ є те, що його він має first-class підтримку асинхронних викликів та промісів, тому можнна комбінувати ці виклики наступним чином:

```
const audioContext = new AudioContext();
const decodedAudioData = await fetch("/WebGL_labs2/music.mp3")
    .then(response => response.arrayBuffer())
    .then(audioData => audioContext.decodeAudioData(audioData));
const source = audioContext.createBufferSource();
source.buffer = decodedAudioData;
source.connect(audioContext.destination);
source.start();
```

Також важливим факктором є те, що аудіо контектст може бути ініціалізований тільки після взаємодії користувача з сайтом.

Застосунок використовує Magnetometer API для отримання даних про магнітне поле та service worker для їх обробки. Ці дані використовуються для оновлення позиції рухомої сфери на канвасі.

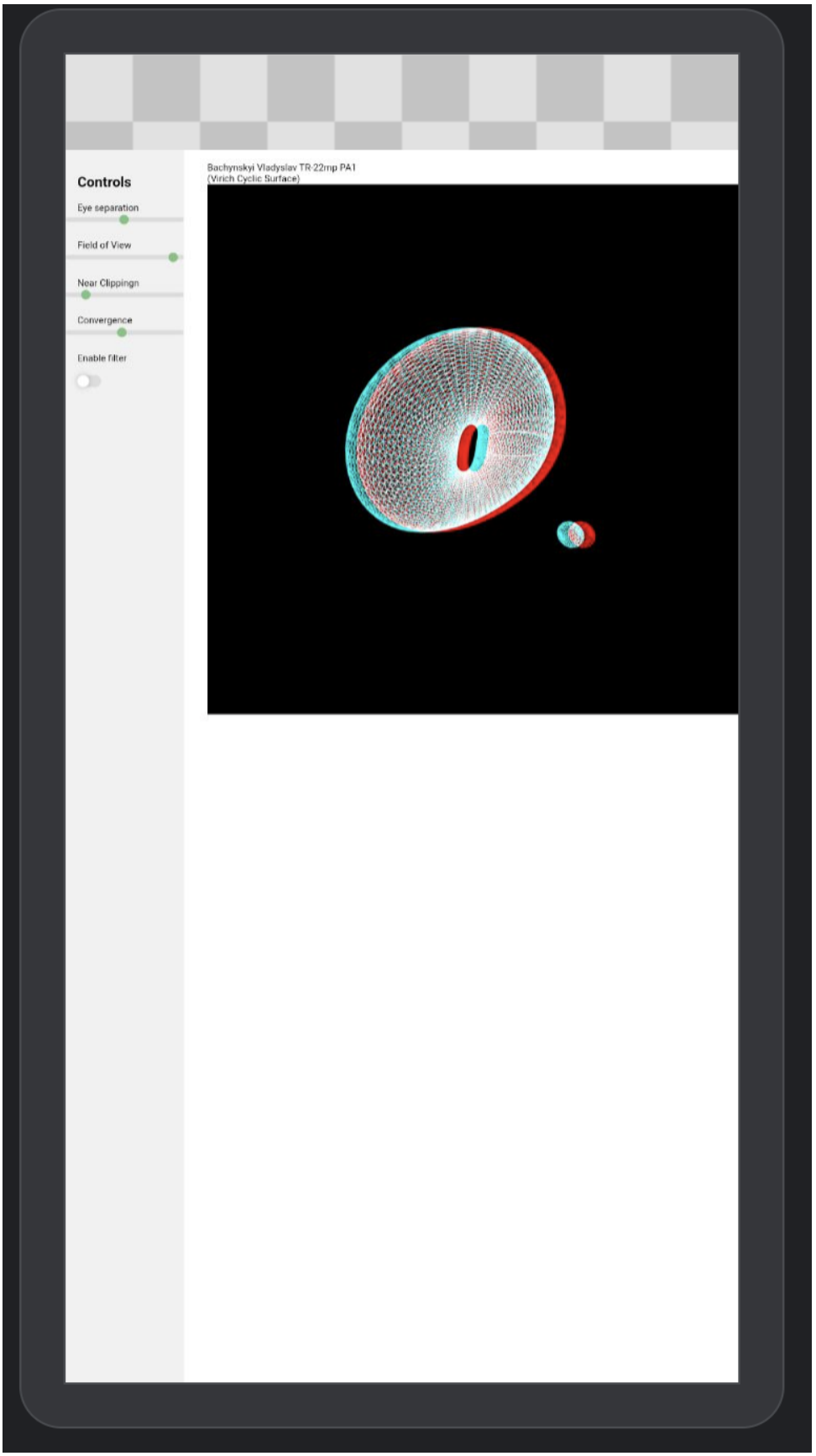
За допомогою WebGL було досягнуто відрендерення двох фігур на канвасі. Шейдери використовуються для розрахунку положення вершин та кольорів фігур, що дозволяє створити ефект руху.

Web Audio API використовується для завантаження аудіо доріжки та контролю звуку. Аудіо доріжка відтворюється з використанням AudioBufferSourceNode, а її позиція змінюється залежно від координат рухомої сфери. Це додає враження обертання звуку у просторі, що збільшує іммерсивність візуально-аудіального досвіду.

Додатково, застосунок надає можливість включення highpass фільтра для звуку. Цей фільтр обмежує частотний діапазон звуку, приглушуючи нижчі частоти.

У результаті поєднання цих технологій та API, розроблений веб-застосунок створює вражаючий візуально-аудіальний досвід, дозволяючи користувачам насолоджуватися рухом фігур на канвасі, обертанням звуку у просторі та динамічним звуковим ефектом за допомогою highpass фільтра.

# Скриншоти


Controls
Eye separation
Field of View
Near Clippingn
Convergence
Enable filter
Bachynskyi Vladyslav TR-22mp PA1
(Virich Cyclic Surface)

## Controls

Eye separation

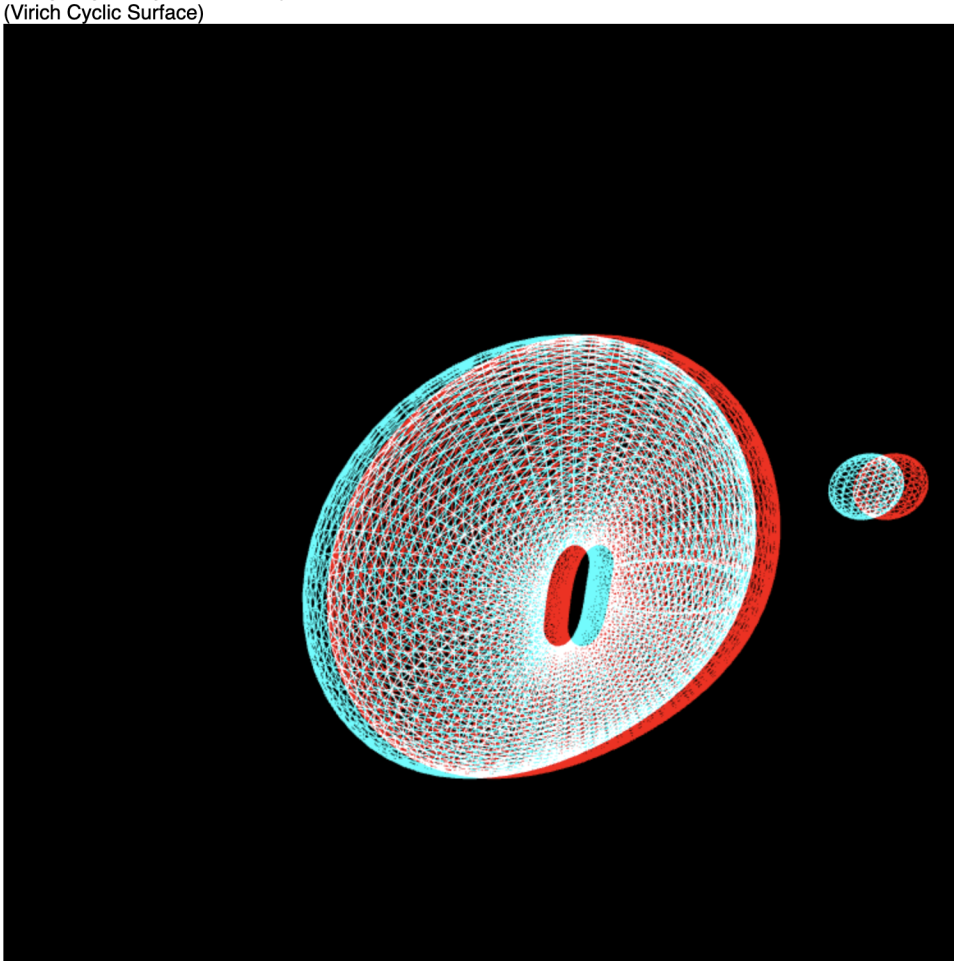
Field of View

Near Clippingn

Convergence

Enable filter

Bachynskyi Vladyslav TR-22mp PA1
(Virich Cyclic Surface)



## Вихідний код

```
function createSphereData() {
  const offset = 3.2;
  const radius = 0.5;
  const slices = 16;
  const stacks = 16;
  const vertices = [];

  for(let stackNumber = 0; stackNumber <= stacks; stackNumber++) {
        const theta = stackNumber * Math.PI / stacks;
        const nextTheta = (stackNumber + 1) * Math.PI / stacks;

        for(let sliceNumber = 0; sliceNumber <= slices; sliceNumber++) {
        const phi = sliceNumber * 2 * Math.PI / slices;
        const nextPhi = (sliceNumber + 1) * 2 * Math.PI / slices;
        const x1 = radius * Math.sin(theta) * Math.cos(phi);
        const y1 = radius * Math.cos(theta);
        const z1 = radius * Math.sin(theta) * Math.sin(phi);
        const x2 = radius * Math.sin(nextTheta) * Math.cos(nextPhi);
        const y2 = radius * Math.cos(nextTheta);
        const z2 = radius * Math.sin(nextTheta) * Math.sin(nextPhi);

        vertices.push(x1 + offset, y1 + offset, z1 + offset);
        vertices.push(x2 + offset, y2 + offset, z2 + offset);
        }
  }

  return vertices;
}

// Constructor
function Model(name) {
  this.name = name;
  this.iVertexBuffer = gl.createBuffer();

  this.BufferData = function () {
        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(vertices.concat(sphereVertices)), gl.STREAM_DRAW);
        gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(shProgram.iAttribVertex);
  };
}
```

```
// Constructor
function ShaderProgram(name, program) {
  this.name = name;
  this.prog = program;

  // Location of the attribute variable in the shader program.
  this.iAttribVertex = -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.Use = function () {
        gl.useProgram(this.prog);
  };
}

function LeftPOV(stereoCamera) {
  const { eyeSeparation, convergence, aspectRatio, fov, nearClipping, far } =
stereoCamera;
  const top = nearClipping * 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 * nearClipping) / convergence;
  const right = (c * nearClipping) / convergence;

  return m4.frustum(left, right, bottom, top, nearClipping, far);
}
function RightPOV(stereoCamera) {
  const { eyeSeparation, convergence, aspectRatio, fov, nearClipping, far } =
stereoCamera;
  const top = nearClipping * 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 * nearClipping) / convergence;
  const right = (b * nearClipping) / convergence;
  return m4.frustum(left, right, bottom, top, nearClipping, far);
}
```