НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

Звіт до розрахунково-графічної роботи

З дисципліни: «Методи синтезу віртуальної реальності»

Варіант 3

**Виконав:**

студент 5 курсу

групи ТР-21мп, ІАТЕ

Бойко Єгор Вікторович

**Перевірив:**

Демчишин А. А.

Київ – 2023

**Завдання:**

1. Повторно використати 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. Додати звуковий фільтр (використовуючи інтерфейс BiquadFilterNode) для кожного варіанту. Додайте елемент прапорця, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак.

**Варіант:** 3 - Смуговий фільтр

**Теорія**

Аудіо в Інтернеті до цього моменту було досить примітивним і донедавна доводилося доставляти через плагіни, такі як Flash і QuickTime. Введення елемента аудіо в HTML5 є дуже важливим, оскільки дозволяє відтворювати базове потокове аудіо. Але він недостатньо потужний, щоб працювати зі складнішими аудіододатками. Для складних веб-ігор або інтерактивних програм потрібне інше рішення. Ціль Web Audio API полягає в тому, щоб включити можливості сучасних ігрових звукових движків, а також деякі завдання мікшування, обробки та фільтрації, які є в сучасних настільних програмах створення звуку.

Використовуючи PannerNode, аудіопотік можна розподілити або розташувати в просторі відносно AudioListener. BaseAudioContext міститиме один AudioListener. І панорами, і слухачі мають позицію в 3D-просторі за допомогою правої декартової системи координат. Одиниці, які використовуються в системі координат, не визначені, і це не обов’язково, оскільки результати, обчислені за допомогою цих координат, є незалежними/інваріантними щодо будь-яких конкретних одиниць, таких як метри або фути. Об’єкти PannerNode (що представляють вихідний потік) мають вектор орієнтації, що вказує, у якому напрямку проектується звук. Крім того, вони мають звуковий конус, що вказує на те, наскільки спрямований звук. Наприклад, звук може бути всенаправленим, і в цьому випадку його буде чути будь-де, незалежно від його орієнтації, або він може бути більш спрямованим і чутним, лише якщо він звернений до слухача. Об’єкти AudioListener (які представляють вуха людини) мають вектори вперед і вгору, що представляють напрямок, у якому людина дивиться. Під час рендерингу PannerNode обчислює азимут і висоту. Ці значення використовуються внутрішньо реалізацією, щоб відтворити ефект просторування.

BiquadFilterNode — це процесор AudioNode, який реалізує дуже поширені фільтри нижчого порядку.

Фільтри нижчого порядку є будівельними блоками базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів і більш розширених фільтрів. Кілька фільтрів BiquadFilterNode можна комбінувати для створення більш складних фільтрів. Параметри фільтра, такі як частота, можна змінювати з часом для розгортки фільтра тощо. Кожен BiquadFilterNode можна налаштувати як один із кількох загальних типів фільтрів

**Деталі реалізації**

Для реалізації просторового аудіо та накладення фільтру, використовуючи Web Audio API, була використана мова Javascript та бібліотека WebGl, для зображення джерела звуку. Крім того, керування позиціонування джерела звуку здійснюється завдяки сенсору гіроскопа.

Реалізація налаштування аудіо складається з:

* Налаштування аудіо контексту, який надає функціонал для роботи з аудіо для подальшого використання;
* Налаштування інтерфейсу MediaElementAudioSourceNode, який представляє собою джерело аудіо з аудіо або відео елементу;
* Налаштування інтерфейсу PannerNode, який є вузлом обробки та позиціонує/розподіляє вхідний аудіопотік у тривимірному просторі;
* Налаштування інтерфейсу BiquadFilterNode , який дозволяє використовувати поширені фільтри низького порядку;
* З’єднання всіх AudioNode для отримання вихідного аудіо.
* Продовження чи призупинення аудіоконтексту, в залежності чи була натиснута пауза чи програвання.

function setupAudio() {

  audioSource = document.getElementById('audio')

  audioSource.addEventListener('play', () => {

    if (!sound.audioCtx) {

      sound.audioCtx = new window.AudioContext()

      sound.source = sound.audioCtx.createMediaElementSource(audioSource)

      sound.panner = sound.audioCtx.createPanner()

      sound.filter = sound.audioCtx.createBiquadFilter()

      // Filter settings

      sound.filter.type = 'bandpass'

      sound.filter.detune.value = 10

      sound.filter.frequency.value = 700

      // Connecting nodes

      sound.source.connect(sound.panner)

      sound.panner.connect(sound.audioCtx.destination)

    }

    sound.audioCtx.resume()

  })

  audioSource.addEventListener('pause', () => {

    if (sound.audioCtx) {

      sound.audioCtx.suspend()

    }

  })

}

Реалізація початку використання аудіо складається з:

* Налаштування аудіо;
* Отримання вибору використання фільтру;
* В залежності від вибору, з’єднання або роз’єднання AudioNode фільтру до послідовності.

function beginAudio() {

  setupAudio()

  let filterCheck = document.getElementById('filterCheck')

  filterCheck.addEventListener('change', () => {

    if (filterCheck.checked) {

      sound.panner.disconnect()

      sound.panner.connect(sound.filter)

      sound.filter.connect(sound.audioCtx.destination)

    } else {

      sound.panner.disconnect()

      sound.panner.connect(sound.audioCtx.destination)

    }

  })

  audioSource.play()

}

Реалізація створення візуального джерела звуку у вигляді сфери складається з:

* Створення масиву координат сфери;
* Додавання зміщення центру сфери, використовуючи гіроскоп;
* Використання координат центру сфери для зміни значень розташування PannerNode;
* Створення матриці зміщення для відображення зміщення сфери.

function createSphereCoordinates(radius, widthSegments, heightSegments) {

  let coordinates = []

  for (var i = 0; i <= heightSegments; i++) {

    let theta = (i \* Math.PI) / heightSegments

    let sinTheta = Math.sin(theta)

    let cosTheta = Math.cos(theta)

    for (var j = 0; j <= widthSegments; j++) {

      let phi = (j \* 2 \* Math.PI) / widthSegments

      let sinPhi = Math.sin(phi)

      let cosPhi = Math.cos(phi)

      let x = cosPhi \* sinTheta

      let y = cosTheta

      let z = sinPhi \* sinTheta

      coordinates.push(x \* radius, y \* radius, z \* radius)

    }

  }

  return coordinates

}

sphereX += sensor.x

    sphereY += sensor.y

    sphereZ += sensor.z

if (sound.panner) {

    sound.panner.positionX.value = sphereX

    sound.panner.positionY.value = sphereY

    sound.panner.positionZ.value = sphereZ

    document.getElementById('sphereX').innerHTML = 'Sphere X: ' + sphereX

    document.getElementById('sphereY').innerHTML = 'Sphere Y: ' + sphereY

    document.getElementById('sphereZ').innerHTML = 'Sphere Z: ' + sphereZ

  }

let translateSphere = m4.translation(sphereX, sphereY, sphereZ)

  let matAccum2 = m4.multiply(translateToPointZero, matAccum0)

  let matAccum3 = m4.multiply(translateSphere, matAccum2)

**Інструкція користувача**

Для використання зміни положення джерела звуку необхідно, щоб пристрій мав підтримку гіроскопу. Для прослуховування та накладання фільтру, сенсори не потрібно.

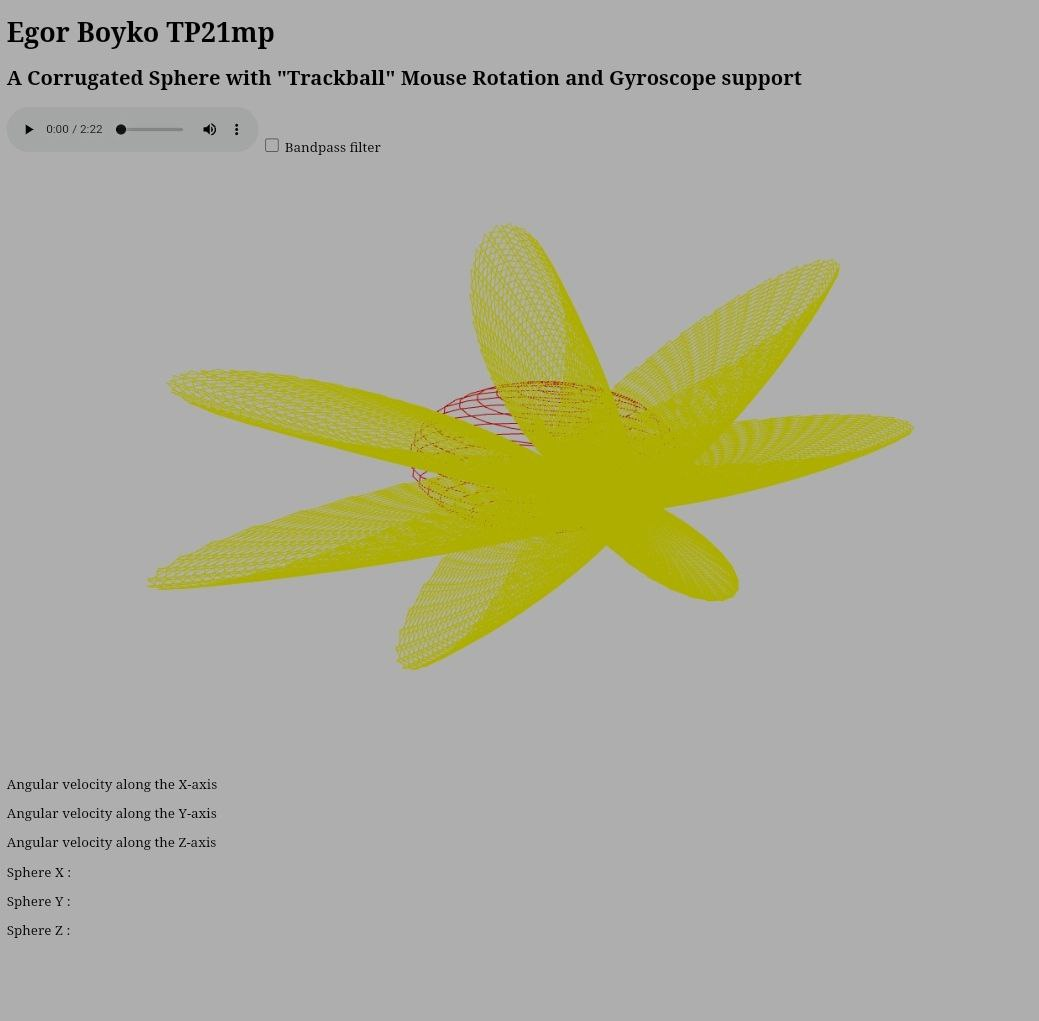


Рисунок 1 – Початковий вигляд сторінки

Прослуховування аудіо має початися автоматично, при завантаження сторінки, але якщо це не відбулось, через особливості браузера, необхідно натиснути кнопку програвання у управлінні аудіофайлу. Для включення/виключення смугового фільтру необхідно поставити (рисунок 2)/зняти (рисунок 3) галку біля “Bandpass filter”.



Рисунок 2 – Виключений смуговий фільтр



Рисунок 3 – Включений смуговий фільтр

Використовуючи пристрій, який має сенсор гіроскопу можна змінювати розташування джерела звуку, при цьому створюється еффект просторового звучання. Також, під областю відмалювання сфери та поверхні, є інформація про поточні значення сенсора гіроскопу та координати розташування джерела звуку. На рисунках 4 та 5, показані приклади зміни розташування джерела звуку.



Рисунок 4 – Зміна розташування джерела звуку

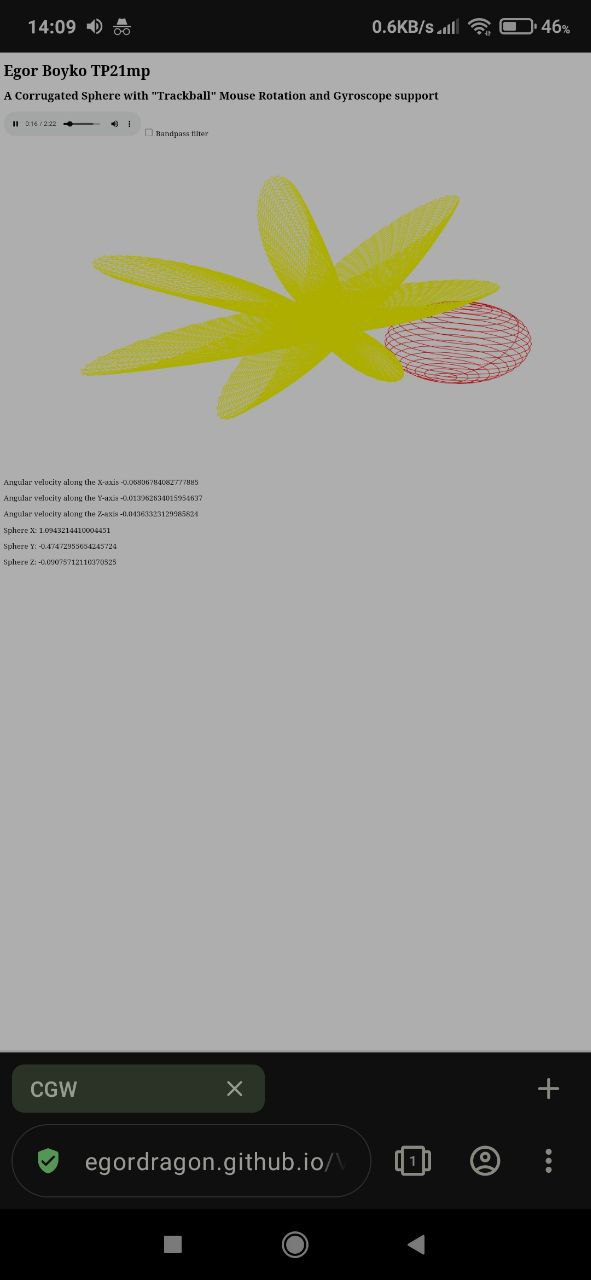


Рисунок 5 – Зміна розташування джерела звуку

**Приклад вихідного коду**

function setupAudio() {

  audioSource = document.getElementById('audio')

  audioSource.addEventListener('play', () => {

    if (!sound.audioCtx) {

      sound.audioCtx = new window.AudioContext()

      sound.source = sound.audioCtx.createMediaElementSource(audioSource)

      sound.panner = sound.audioCtx.createPanner()

      sound.filter = sound.audioCtx.createBiquadFilter()

      // Filter settings

      sound.filter.type = 'bandpass'

      sound.filter.detune.value = 10

      sound.filter.frequency.value = 700

      // Connecting nodes

      sound.source.connect(sound.panner)

      sound.panner.connect(sound.audioCtx.destination)

    }

    sound.audioCtx.resume()

  })

  audioSource.addEventListener('pause', () => {

    if (sound.audioCtx) {

      sound.audioCtx.suspend()

    }

  })

}

function beginAudio() {

  setupAudio()

  let filterCheck = document.getElementById('filterCheck')

  filterCheck.addEventListener('change', () => {

    if (filterCheck.checked) {

      sound.panner.disconnect()

      sound.panner.connect(sound.filter)

      sound.filter.connect(sound.audioCtx.destination)

    } else {

      sound.panner.disconnect()

      sound.panner.connect(sound.audioCtx.destination)

    }

  })

  audioSource.play()

}

function createSphereCoordinates(radius, widthSegments, heightSegments) {

  let coordinates = []

  for (var i = 0; i <= heightSegments; i++) {

    let theta = (i \* Math.PI) / heightSegments

    let sinTheta = Math.sin(theta)

    let cosTheta = Math.cos(theta)

    for (var j = 0; j <= widthSegments; j++) {

      let phi = (j \* 2 \* Math.PI) / widthSegments

      let sinPhi = Math.sin(phi)

      let cosPhi = Math.cos(phi)

      let x = cosPhi \* sinTheta

      let y = cosTheta

      let z = sinPhi \* sinTheta

      coordinates.push(x \* radius, y \* radius, z \* radius)

    }

  }

  return coordinates

}

function draw() {

  gl.clearColor(1, 1, 1, 1)

  gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT)

  if (sound.panner) {

    sound.panner.positionX.value = sphereX

    sound.panner.positionY.value = sphereY

    sound.panner.positionZ.value = sphereZ

    document.getElementById('sphereX').innerHTML = 'Sphere X: ' + sphereX

    document.getElementById('sphereY').innerHTML = 'Sphere Y: ' + sphereY

    document.getElementById('sphereZ').innerHTML = 'Sphere Z: ' + sphereZ

  }

  /\* Set the values of the projection transformation \*/

  let projection = m4.perspective(Math.PI / 8, 1, 8, 12)

  /\* Get the view matrix from the SimpleRotator object.\*/

  let modelView = spaceball.getViewMatrix()

  let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7)

  let translateToPointZero = m4.translation(0, 0, -10)

  let matAccum0 = m4.multiply(rotateToPointZero, modelView)

  let translateSphere = m4.translation(sphereX, sphereY, sphereZ)

  //let matAccum3 = m4.multiply(orientationRotateMatrix, matAccum0)

  let matAccum2 = m4.multiply(translateToPointZero, matAccum0)

  let matAccum3 = m4.multiply(translateSphere, matAccum2)

  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccum3)

  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projection)

  gl.uniform4fv(shProgram.iColor, [1, 0, 0, 1])

  sphere.Draw()

gl.clear(gl.DEPTH\_BUFFER\_BIT)

  matAccum0 = m4.multiply(rotateToPointZero, modelView)

  matAccum3 = m4.multiply(translateToPointZero, matAccum0)

  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccum3)

  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projection)

  gl.uniform4fv(shProgram.iColor, [1, 1, 0, 1])

  surface.Draw()

}