МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

Розрахунково графічна робота

з дисципліни «Методи синтезу віртуальної реальності».

Виконав:

Студент 1-го курсу магістратури

Інституту атомної та теплової енергетики

Групи ТР-23мп  
Чорний Владислав Олександрович

Прийняв:

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:** 24 - Смуговий фільтр

**Теоретичні відомості**

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 апаратному сенсору.

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.

    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)

        }

    })

Реалізація створення візуального джерела звуку у вигляді сфери складається з:

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

function CreateSphereData() {

    let vertexList = [];

    let textCoords = [];

    let thetaStep = Math.PI / 10.0

    let phiStep = Math.PI / 10.0

    for(let theta = 0.0; theta < Math.PI; theta = theta + thetaStep)

    {

        let sinTheta = Math.sin(theta)

        let cosTheta = Math.cos(theta)

        let thetaNext = theta + thetaStep

        let sinThetaNext = Math.sin(thetaNext)

        let cosThetaNext = Math.cos(thetaNext)

        for(let phi = 0.0; phi < 2.0 \* Math.PI; phi = phi + phiStep)

        {

            let sinPhi = Math.sin(phi)

            let cosPhi = Math.cos(phi)

            let x = sphereRadius \* cosPhi \* sinTheta

            let y = sphereRadius \* sinPhi \* sinTheta

            let z = sphereRadius \* cosTheta

            vertexList.push(x, y, z)

            textCoords.push(1, 1);

            x = sphereRadius \* cosPhi \* sinThetaNext

            y = sphereRadius \* sinPhi \* sinThetaNext

            z = sphereRadius \* cosThetaNext

            vertexList.push(x, y, z)

            textCoords.push(1, 1);

        }

    }

    return [vertexList, textCoords];

}

if (sound.panner)

    {

        let Position = multiplyMatrixAndPoint(WorldViewMatrix, [1, 1, 1, 1])

        sound.panner.positionX.value = Position[0]

        sound.panner.positionY.value = Position[1]

        sound.panner.positionZ.value = Position[2]

        document.getElementById('sphereX').innerHTML = 'Sphere X: ' + Position[0]

        document.getElementById('sphereY').innerHTML = 'Sphere Y: ' + Position[1]

        document.getElementById('sphereZ').innerHTML = 'Sphere Z: ' + Position[2]

    }

**Інструкція користувача**

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



Рисунок 1 – Вигляд сторінки

Для початку програвання аудіо потрібно його увімкнути. Для включення/виключення смугового фільтру необхідно поставити/зняти галку біля “Bandpass filter”.

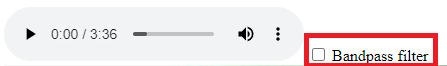


Рисунок 2 – Виключений смуговий фільтр

Використовуючи пристрій, який має сенсор можна змінювати розташування джерела звуку, при цьому створюється еффект просторового звучання. Також є інформація про поточні значення сенсора та координати розташування джерела звуку. На рисунку 3, показаний приклад зміни розташування джерела звуку.

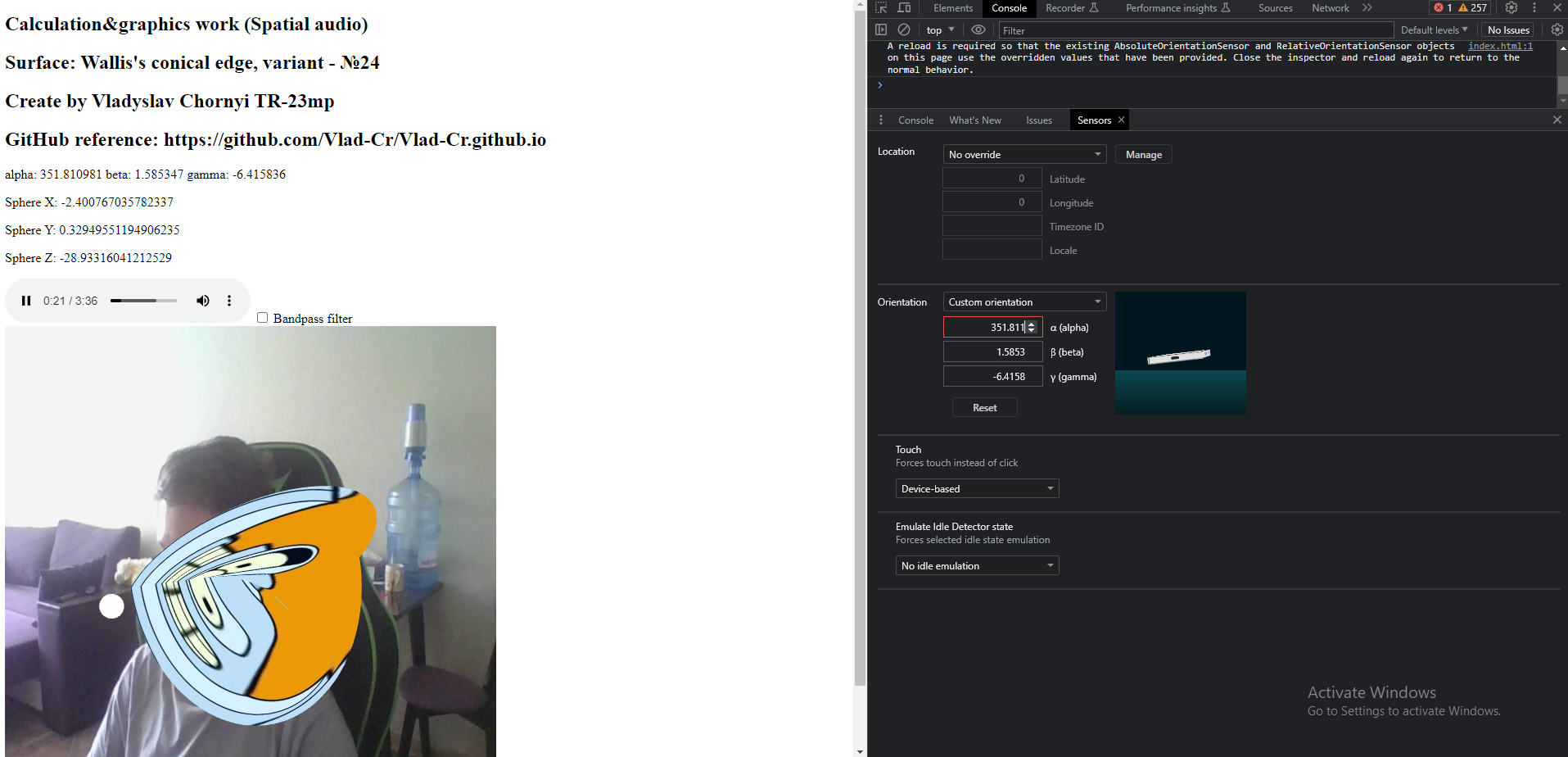


Рисунок 3 – Зміна розташування джерела звуку

**Лістинг програми**

...

function draw() {

    gl.clearColor(0,0,0,1);

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

    gl.enable(gl.CULL\_FACE);

    gl.enable(gl.DEPTH\_TEST);

    DrawWebCamVideo();

    gl.clear(gl.DEPTH\_BUFFER\_BIT);

    DrawSurface();

    DrawSphere();

}

function DrawSphere()

{

    let ModelView = getRotationMatrix(SensorAlpha, SensorBeta, SensorGamma)

    let WorldMatrix = m4.translation(0, 0, -30);

    let ProjectionMatrix = m4.perspective(Math.PI / 8, 1, 1, 100)

    let WorldViewMatrix = m4.multiply(ModelView, WorldMatrix );

    let ModelViewProjection = m4.multiply(ProjectionMatrix, WorldViewMatrix);

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, ModelViewProjection );

    gl.uniform4fv(shProgram.iColor, [1.0,1.0,1.0,1] )

    if (sound.panner)

    {

        let Position = multiplyMatrixAndPoint(WorldViewMatrix, [1, 1, 1, 1])

        sound.panner.positionX.value = Position[0]

        sound.panner.positionY.value = Position[1]

        sound.panner.positionZ.value = Position[2]

        document.getElementById('sphereX').innerHTML = 'Sphere X: ' + Position[0]

        document.getElementById('sphereY').innerHTML = 'Sphere Y: ' + Position[1]

        document.getElementById('sphereZ').innerHTML = 'Sphere Z: ' + Position[2]

    }

    AudioSphere.Draw()

}

function DrawSurface()

{

    let ModelView = m4.translation(0, 0, 0);

    let WorldMatrix = m4.translation(0, 0, -15);

    let ProjectionMatrix = m4.perspective(Math.PI / 8, 1, 1, 100)

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

    let matAccum0 = m4.multiply(rotateToPointZero, ModelView)

    let WorldViewMatrix = m4.multiply(WorldMatrix, matAccum0 );

    let ModelViewProjection = m4.multiply(ProjectionMatrix, WorldViewMatrix);

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, ModelViewProjection );

    gl.uniform4fv(shProgram.iColor, [0.0,0.0,0.0,1] );

    gl.bindTexture(gl.TEXTURE\_2D, SurfaceTexture);

    gl.uniform1i(shProgram.iTexture, 0);

    surface.Draw();

}

function CreateSphereData() {

    let vertexList = [];

    let textCoords = [];

    let thetaStep = Math.PI / 10.0

    let phiStep = Math.PI / 10.0

    for(let theta = 0.0; theta < Math.PI; theta = theta + thetaStep)

    {

        let sinTheta = Math.sin(theta)

        let cosTheta = Math.cos(theta)

        let thetaNext = theta + thetaStep

        let sinThetaNext = Math.sin(thetaNext)

        let cosThetaNext = Math.cos(thetaNext)

        for(let phi = 0.0; phi < 2.0 \* Math.PI; phi = phi + phiStep)

        {

            let sinPhi = Math.sin(phi)

            let cosPhi = Math.cos(phi)

            let x = sphereRadius \* cosPhi \* sinTheta

            let y = sphereRadius \* sinPhi \* sinTheta

            let z = sphereRadius \* cosTheta

            vertexList.push(x, y, z)

            textCoords.push(1, 1);

            x = sphereRadius \* cosPhi \* sinThetaNext

            y = sphereRadius \* sinPhi \* sinThetaNext

            z = sphereRadius \* cosThetaNext

            vertexList.push(x, y, z)

            textCoords.push(1, 1);

        }

    }

    return [vertexList, textCoords];

}

function init() {

...

    SetUpWebCamTexture()

    spaceball = new TrackballRotator(canvas, draw, 0)

    LoadTexture()

    playVideo()

    setupAudio()

}

function playVideo(){

    draw();

    setInterval(playVideo, 1/24);

}

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()

      }

    })

    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)

        }

    })

  }

const dataContainerOrientation = document.getElementById('dataContainerOrientation');

if(window.DeviceOrientationEvent)

{

    dataContainerOrientation.innerHTML = 'alpha: ' + SensorAlpha + '  beta: ' + SensorBeta + '  gamma: ' + SensorGamma;

    window.addEventListener('deviceorientation', function (event) {

        SensorAlpha = event.alpha;

        SensorBeta = event.beta;

        SensorGamma = event.gamma;

        if(SensorAlpha!=null || SensorBeta!=null || SensorGamma!=null)

        {

            dataContainerOrientation.innerHTML = 'alpha: ' + SensorAlpha + '  beta: ' + SensorBeta + '  gamma: ' + SensorGamma;

        }

        draw();

    });

}

function getRotationMatrix( alpha, beta, gamma ) {

    var degtorad = Math.PI / 180; // Degree-to-Radian conversion

    var \_x = beta  ? beta  \* degtorad : 0; // beta value

    var \_y = gamma ? gamma \* degtorad : 0; // gamma value

    var \_z = alpha ? alpha \* degtorad : 0; // alpha value

    var cX = Math.cos( \_x );

    var cY = Math.cos( \_y );

    var cZ = Math.cos( \_z );

    var sX = Math.sin( \_x );

    var sY = Math.sin( \_y );

    var sZ = Math.sin( \_z );

    //

    // ZXY rotation matrix construction.

    //

    var m11 = cZ \* cY - sZ \* sX \* sY;

    var m12 = - cX \* sZ;

    var m13 = cY \* sZ \* sX + cZ \* sY;

    var m21 = cY \* sZ + cZ \* sX \* sY;

    var m22 = cZ \* cX;

    var m23 = sZ \* sY - cZ \* cY \* sX;

    var m31 = - cX \* sY;

    var m32 = sX;

    var m33 = cX \* cY;

    let dst = new Float32Array(16);

    dst[0] = m11;

    dst[1] = m12;

    dst[2] = m13;

    dst[3] = 0.0;

    dst[4] = m21;

    dst[5] = m22;

    dst[6] = m23;

    dst[7] = 0.0;

    dst[8] = m31;

    dst[9] = m32;

    dst[10] = m33;

    dst[11] = 0.0;

    dst[12] = 0.0;

    dst[13] = 0.0;

    dst[14] = 0.0;

    dst[15] = 1.0;

    return dst;

}