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 8**

**Виконав:**

Студент 1-го курсу магістратури

ІАТЕ

групи ТР-22мп

Євтушенко Дмитро Миколайович

**Перевірив:**

Демчишин А. А.

Київ-2023

**Завдання**

Тема - Імлементування звуку у просторі за допомогою WebAudio HTML5 API

Хід роботи:

1. Повторно використати код із практичного завдання №2;
2. Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою матеріального інтерфейсу. 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.

За власним варіантом був зроблений **фільтр низьких частот.**

**Теоретичні відомості**

Web Audio API (Звукова WebAudio HTML5 API) - це JavaScript API, який дозволяє веб-розробникам маніпулювати та контролювати звук на веб-сторінці. Він надає потужні інструменти для створення, зміни та відтворення аудіо контенту у реальному часі.

Основні поняття:

* Аудіо контекст (AudioContext): Це головний об'єкт у Web Audio API, який представляє аудіо простір, в якому відбуваються всі маніпуляції зі звуком. Він керує потоками аудіо та визначає, які компоненти обробляють аудіо сигнали.
* Вузли (Nodes): Web Audio API використовує вузли для створення та маніпулювання звуком. Є кілька типів вузлів, таких як: вузли джерела (source nodes) для завантаження аудіо даних, вузли обробки (processing nodes) для зміни звукових параметрів та вузли відтворення (destination nodes) для виведення аудіо на вихідний пристрій.
* Підключення (Connections): Вузли можуть бути підключені один до одного, щоб створювати ланцюжки обробки звуку. Це дозволяє використовувати різні ефекти та обробки для створення складних звукових ефектів.
* Аналізатори (Analyzers): Web Audio API також надає можливість аналізувати аудіо сигнали, наприклад, для вимірювання гучності або виявлення спектральних характеристик.

Застосування Web Audio API:

1. Візуалізація звуку: Web Audio API може використовуватись для створення візуальних ефектів, які реагують на аудіо сигнали. Наприклад, ви можете створити анімовані спектрограми, візуалізацію звукових хвиль або реагувати на рівень гучності.
2. Музичні додатки: За допомогою Web Audio API ви можете створювати музичні інструменти, синтезатори та секвенсори, які можна відтворювати в реальному часі в браузері.
3. Аудіо ефекти: Ви можете застосовувати різні ефекти до аудіо сигналів, такі як реверберація, ехо, фільтрація, затримка (delay) та багато інших. Це дозволяє створювати цікаві аудіо ефекти та обробки прямо у браузері.
4. Звукові ігри: Web Audio API може бути використано для створення звукових ефектів та фонової музики у веб-іграх.

Web Audio API надає потужні можливості для маніпулювання та контролю звуком у веб-програмуванні. Це дозволяє створювати багатоцільові додатки, які використовують звук для створення вражаючих веб-взаємодій та досвіду користувача.

Існує кілька додаткових важливих аспектів про Web Audio API:

* Гнучкість та точність: Web Audio API дозволяє вам точно контролювати аудіо-сигнали та їхню обробку. Ви можете маніпулювати гучністю, тоном, затримкою, фазою та багатьма іншими параметрами звуку. Це дозволяє створювати складні звукові обробки та ефекти з високою точністю.
* Реалізація звукових ланцюгів: Ви можете створювати складні ланцюги обробки звуку, підключаючи різні вузли один до одного. Це дає можливість створювати послідовну або паралельну обробку звуку, де кожен вузол відповідає за певну операцію. Вузли можуть бути підключені до джерел звуку, обробляти його та виводити на вихідний пристрій.
* Завантаження та відтворення аудіо: Web Audio API надає зручні методи для завантаження та відтворення аудіо-файлів. Ви можете завантажувати аудіо з різних джерел, таких як локальний файл, мережа або потокові джерела. Після завантаження ви можете контролювати відтворення, перемотку та інші аспекти аудіо.
* Підтримка аудіоаналізу: Web Audio API надає аналізатори, які дозволяють вам отримувати інформацію про аудіо-сигнал. Ви можете отримувати дані про гучність, спектральні складові, звукові хвилі та інші параметри звуку. Це може бути корисно для візуалізації звуку або автоматичного керування звуковими параметрами.

**Реалізація продукту**

Для реалізації просторового аудіо в веб-додатку було використано мову програмування JavaScript та бібліотеку WebGL для візуалізації джерела звуку. І також для реалізації звуку було використано WebAudio HTML5 API.

Було реалізовано функцію яка відмальовувала сферу за координатами:

function CreateSphereSurface(r = 0.1) {

  let vertexList = [];

  let lon = -Math.PI;

  let lat = -Math.PI \* 0.5;

  while (lon < Math.PI) {

    while (lat < Math.PI \* 0.5) {

      let v1 = sphereSurfaceDate(r, lon, lat);

      vertexList.push(v1.x, v1.y, v1.z);

      lat += 0.05;

    }

    lat = -Math.PI \* 0.5

    lon += 0.05;

  }

  return vertexList;

}

function sphereSurfaceDate(r, u, v) {

  let x = r \* Math.sin(u) \* Math.cos(v);

  let y = r \* Math.sin(u) \* Math.sin(v);

  let z = r \* Math.cos(u);

  return { x: x, y: y, z: z };

}

Також було відображено зміну обертання самого девайсу:

function requestDeviceOrientation() {

  if (typeof DeviceOrientationEvent !== 'undefined' &&

    typeof DeviceOrientationEvent.requestPermission === 'function') {

    DeviceOrientationEvent.requestPermission()

      .then(response => {

        console.log(response);

        if (response === 'granted') {

          console.log('Permission granted');

          window.addEventListener('deviceorientation', e => {

            alpha = deg2rad(e.alpha)

            beta = deg2rad(e.beta)

            gamma = deg2rad(e.gamma)

          }, true);

        }

      }).catch((err => {

        console.log('Err', err);

      }));

  } else

    console.log('not iOS');

}

До того ж було встановлено роботу аудіо:

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 = 'lowpass';

      filter.Q.value = 1;

      filter.frequency.value = 1000;

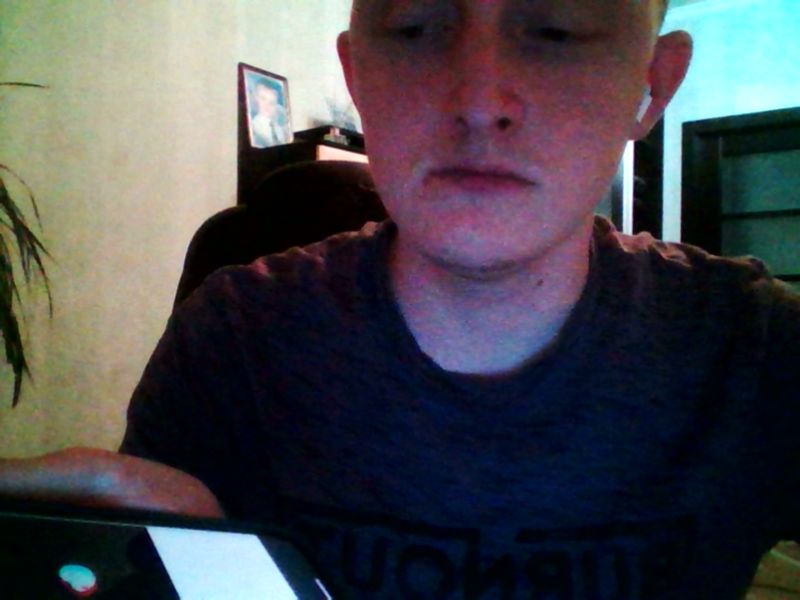
      filter.gain.value = 15;

      audioContext.resume();

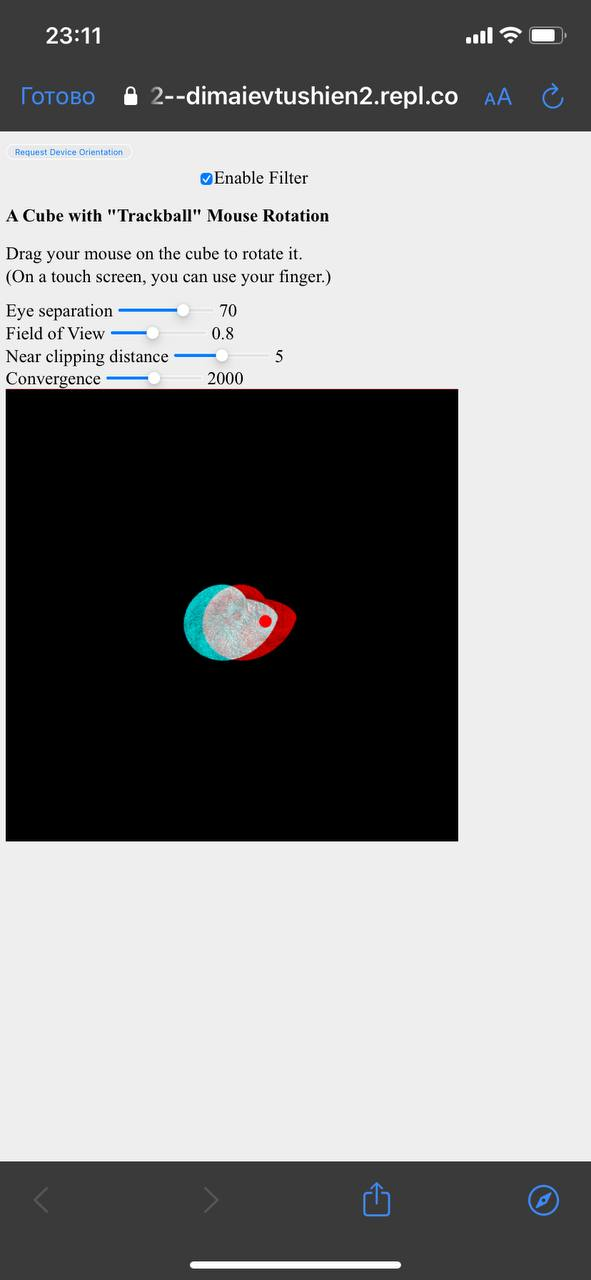
    }

  })

**Результати програми**

****

На цьому рисунку представлена робота, де сфера рухається і за цим рухом змінюється звук тону навушників



На цьому рисунку представлена сфера, яка працює та рухається за нахилом девайсу.

**Лістинг програми**

this.updateValues = function() {

    let inputs = document.getElementsByClassName("setValue");

    let spans = document.getElementsByClassName("currentValue");

    let eyeSep = 70.0;

    eyeSep = inputs[0].value;

    spans[0].innerHTML = eyeSep;

    this.mEyeSeparation = eyeSep;

    let ratio = 1.0;

    let fov = 0.8;

    fov = inputs[1].value;

    spans[1].innerHTML = fov;

    this.mFOV = fov;

    let nearClip = 5.0;

    nearClip = inputs[2].value - 0.0;

    spans[2].innerHTML = nearClip;

    this.mNearClippingDistance = nearClip

    let convergence = 2000.0;

    convergence = inputs[3].value;

    spans[3].innerHTML = convergence;

    this.mConvergence = convergence

  }

}

function getWebcam() {

  navigator.getUserMedia({ video: true, audio: false }, function(stream) {

    video.srcObject = stream;

    track = stream.getTracks()[0];

  }, function(e) {

    console.error('Rejected!', e);

  });

}

function CreateWebCamTexture() {

  videoTexture = gl.createTexture();

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

  gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

  gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.LINEAR);

  gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

  gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.CLAMP\_TO\_EDGE);

}

function deg2rad(angle) {

  return angle \* Math.PI / 180;

}

// Constructor

function Model(name) {

  this.name = name;

  this.iVertexBuffer = gl.createBuffer();

  this.iTextureBuffer = gl.createBuffer();

  this.count = 0;

  this.countText = 0;

  this.BufferData = function(vertices) {

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

    gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices), gl.STREAM\_DRAW);

    this.count = vertices.length / 3;

  }

  this.TextureBufferData = function(normals) {

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer);

    gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(normals), gl.STREAM\_DRAW);

    this.countText = normals.length / 2;

  }

  this.Draw = function() {

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

    gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);

    gl.enableVertexAttribArray(shProgram.iAttribVertex);

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer);

    gl.vertexAttribPointer(shProgram.iAttribTexture, 2, gl.FLOAT, false, 0, 0);

    gl.enableVertexAttribArray(shProgram.iAttribTexture);

    gl.drawArrays(gl.TRIANGLE\_STRIP, 0, this.count);

  }

  this.DisplayPoint = function() {

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer);

    gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);

    gl.enableVertexAttribArray(shProgram.iAttribVertex);

    gl.drawArrays(gl.LINE\_STRIP, 0, this.count);

  }

}

function CreateSphereSurface(r = 0.1) {

  let vertexList = [];

  let lon = -Math.PI;

  let lat = -Math.PI \* 0.5;

  while (lon < Math.PI) {

    while (lat < Math.PI \* 0.5) {

      let v1 = sphereSurfaceDate(r, lon, lat);

      vertexList.push(v1.x, v1.y, v1.z);

      lat += 0.05;

    }

    lat = -Math.PI \* 0.5

    lon += 0.05;

  }

  return vertexList;

}

function sphereSurfaceDate(r, u, v) {

  let x = r \* Math.sin(u) \* Math.cos(v);

  let y = r \* Math.sin(u) \* Math.sin(v);

  let z = r \* Math.cos(u);

  return { x: x, y: y, z: z };

}

function drawDraw() {

  draw()

  window.requestAnimationFrame(drawDraw)

}

function CreateSurfaceData() {

  let vertexList = [];

  let u = 0;

  let v = 0;

  let uMax = Math.PI \* 2

  let vMax = Math.PI \* 2

  let uStep = uMax / 50;

  let vStep = vMax / 50;

  for (let u = 0; u <= uMax; u += uStep) {

    for (let v = 0; v <= vMax; v += vStep) {

      let vert = Cornucopia(u, v)

      let avert = Cornucopia(u + uStep, v)

      let bvert = Cornucopia(u, v + vStep)

      let cvert = Cornucopia(u + uStep, v + vStep)

      vertexList.push(vert.x, vert.y, vert.z)

      vertexList.push(avert.x, avert.y, avert.z)

      vertexList.push(bvert.x, bvert.y, bvert.z)

      vertexList.push(avert.x, avert.y, avert.z)

      vertexList.push(cvert.x, cvert.y, cvert.z)

      vertexList.push(bvert.x, bvert.y, bvert.z)

    }

  }

  return vertexList;

}

function CreateTexture() {

  let texture = [];

  let u = 0;

  let v = 0;

  let uMax = Math.PI \* 2

  let vMax = Math.PI \* 2

  let uStep = uMax / 50;

  let vStep = vMax / 50;

  for (let u = 0; u <= uMax; u += uStep) {

    for (let v = 0; v <= vMax; v += vStep) {

      let u1 = map(u, 0, uMax, 0, 1)

      let v1 = map(v, 0, vMax, 0, 1)

      texture.push(u1, v1)

      u1 = map(u + uStep, 0, uMax, 0, 1)

      texture.push(u1, v1)

      u1 = map(u, 0, uMax, 0, 1)

      v1 = map(v + vStep, 0, vMax, 0, 1)

      texture.push(u1, v1)

      u1 = map(u + uStep, 0, uMax, 0, 1)

      v1 = map(v, 0, vMax, 0, 1)

      texture.push(u1, v1)

      v1 = map(v + vStep, 0, vMax, 0, 1)

      texture.push(u1, v1)

      u1 = map(u, 0, uMax, 0, 1)

      v1 = map(v + vStep, 0, vMax, 0, 1)

      texture.push(u1, v1)

    }

  }

  return texture;

}

function map(val, f1, t1, f2, t2) {

  let m;

  m = (val - f1) \* (t2 - f2) / (t1 - f1) + f2

  return Math.min(Math.max(m, f2), t2);

}

function Cornucopia(u, v) {

  const k = 0.25

  const p = 0.15

  const m = 0.1

  let x = (Math.E \*\* (m \* u) + Math.E \*\* (p \* u) \* Math.cos(v)) \* Math.cos(u)

  let y = (Math.E \*\* (m \* u) + Math.E \*\* (p \* u) \* Math.cos(v)) \* Math.sin(u)

  let z = Math.E \*\* (p \* u) \* Math.sin(v)

  return { x: x \* k, y: y \* k, z: z \* k }

}

/\* Initialize the WebGL context. Called from init() \*/

function initGL() {

  let prog = createProgram(gl, vertexShaderSource, fragmentShaderSource);

  shProgram = new ShaderProgram('Basic', prog);

  shProgram.Use();

  shProgram.iAttribVertex = gl.getAttribLocation(prog, "vertex");

  shProgram.iAttribTexture = gl.getAttribLocation(prog, "texture");

  shProgram.iModelViewMatrix = gl.getUniformLocation(prog, "ModelViewMatrix");

  shProgram.iProjectionMatrix = gl.getUniformLocation(prog, "ProjectionMatrix");

  shProgram.iTranslatePoint = gl.getUniformLocation(prog, 'translatePoint');

  shProgram.iTexturePoint = gl.getUniformLocation(prog, 'texturePoint');

  shProgram.iRotateValue = gl.getUniformLocation(prog, 'rotateValue');

  shProgram.iTMU = gl.getUniformLocation(prog, 'tmu');

  virtualCamera = new StereoCamera(

    2000,

    70.0,

    1,

    0.8,

    5,

    100

  )

  surface = new Model('Surface');

  surface.BufferData(CreateSurfaceData());

  LoadTexture()

  surface.TextureBufferData(CreateTexture());

  point = new Model('Point');

  point.BufferData(CreateSphereSurface())

  gl.enable(gl.DEPTH\_TEST);

}

function requestDeviceOrientation() {

  if (typeof DeviceOrientationEvent !== 'undefined' &&

    typeof DeviceOrientationEvent.requestPermission === 'function') {

    DeviceOrientationEvent.requestPermission()

      .then(response => {

        console.log(response);

        if (response === 'granted') {

          console.log('Permission granted');

          window.addEventListener('deviceorientation', e => {

            alpha = deg2rad(e.alpha)

            beta = deg2rad(e.beta)

            gamma = deg2rad(e.gamma)

          }, true);

        }

      }).catch((err => {

        console.log('Err', err);

      }));

  } else

    console.log('not iOS');

}

function getRotationMatrix() {

  var \_x = beta; // beta value

  var \_y = gamma; // gamma value

  var \_z = alpha; // 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;

  return [

    m11, m12, m13, 0,

    m21, m22, m23, 0,

    m31, m32, m33, 0,

    0, 0, 0, 1

  ];

};

var audio = null;

var audioContext;

var source;

var panner;

var filter;

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 = 'lowpass';

      filter.Q.value = 1;

      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();

}