Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра цифрових технологій в енергетиці

[Методи синтезу віртуальної реальності](https://campus.kpi.ua/student/index.php?mode=studysheet&action=view&id=125230)

Розрахунково графічна робота

Варіант - 9

**Виконав:**

Студент 1-го курсу

групи ТР-21мп НН ІАТЕ

Кляус Кирило

**Перевірив:**

Демчишин А.А.

Київ-2023

**Завдання**

1. Імплементувати обертання джерела звуку навколо геометричного центру площини за допомогою фізичного (матеріального) інтерфейсу (на цей раз поверхня залишається нерухомою, а джерело звуку рухається). Відтворіти улюблену пісню у форматі mp3/ogg з контролем користувача над просторовим положенням джерела звуку.

2. Візуалізувати положення джерела звуку за допомогою сфери.

3. Додати звуковий фільтр (використовуючи інтерфейс BiquadFilterNode) для відповідного варіанту (№9). Додати чекбокс(checkbox), який вмикає або вимикає фільтр. Встановити параметри фільтра на свій смак.

**Теоретичні відомості**

Web Audio API - це потужний інструмент для роботи зі звуком у веб-додатках. Представлений у стандарті HTML5, цей API надає розробникам можливість створювати, маніпулювати та відтворювати аудіоефекти в реальному часі у веб-середовищі.

Web Audio API включає в себе обробку маніпуляцій зі звуком в аудіоконтексті. Основні операції зі звуком виконуються за допомогою аудіо-вузлів, які з'єднуються для формування графа маршрутизації звуку. Підтримуються декілька джерел з різними типами компонування каналів, навіть в межах одного контексту. Така модульна конструкція дозволяє гнучко створювати складні аудіо функції з динамічними ефектами.

Ключові поняття Web Audio API.

1. аудіоконтекст: аудіоконтекст - це основний об'єкт API, який ініціалізує та керує аудіосистемою. Аудіоконтекст визначає аудіосередовище і містить всі ресурси, необхідні для створення, маніпулювання і відтворення аудіо.

2. аудіо-вузол: аудіо-вузол - це процесор, який обробляє аудіосигнали. Аудіо-вузли - це процесори, які обробляють аудіо-сигнали і можуть бути з'єднані один з одним для створення аудіо-графіків. Існують різні типи аудіо-вузлів, такі як звукові генератори, ефекти, аналізатори та вузли виводу.

3. маршрутизація звуку: з'єднання аудіо-вузлів утворює аудіо-граф, який визначає потік звукових сигналів від джерела до виходу. Це дозволяє створювати складну маршрутизацію сигналів і керувати потоком сигналів.

4. відтворення аудіо Веб-аудіо API дозволяє завантажувати та відтворювати аудіофайли за допомогою аудіо-вузлів. Можна керувати такими параметрами відтворення, як гучність і панорамування.

5. звукові ефекти та обробка: API надає ряд ефектів і фільтрів, які можна застосувати до аудіосигналу. Це дозволяє створювати широкий спектр звукових ефектів та обробки, таких як реверберація, еквалайзери, затримки тощо.

Інтерфейс відкриває величезні можливості для створення інтерактивних, багатих на звук веб-додатків.

Web Audio API дозволяє застосовувати звукові фільтри до аудіосигналів. Одним з інтерфейсів, що використовується для цієї мети, є BiquadFilterNode. У цьому інтерфейсі можна встановити такі параметри фільтра, як тип фільтра, частота зрізу та коефіцієнт підсилення.

Основні кроки для роботи з фільтрами у Web Audio API такі:

1. Створіть AudioContext і завантажте аудіофайл. 2.

2. Створіть вузол BiquadFilterNode за допомогою методу createBiquadFilter() аудіоконтексту. 3.

3. 3. встановіть параметри фільтра, такі як тип фільтра (низькочастотний, високочастотний, смуговий тощо), частота зрізу, коефіцієнт підсилення тощо.

4. підключіть аудіо вузол (наприклад, AudioBufferSourceNode) до фільтра і з'єднайте його з вихідним вузлом для відтворення відфільтрованого аудіо.

5. 5. запустіть відтворення аудіо та встановіть елементи керування (наприклад, кнопки та повзунки) для зміни параметрів фільтра, щоб користувач міг налаштувати звук на свій смак.

6. додайте прапорець, щоб дозволити користувачеві вмикати та вимикати фільтр за потреби.

Інтерфейс Web Audio API також дозволяє нам контролювати, як аудіо розподіляється в просторі. Використовуючи систему, побудовану на моделі джерело-слухач, вона дозволяє керувати просторовою моделлю та регулювати звук залежно від відстані, зміна якої спричинена рухом джерела звуку.

**Виконання роботи**

1. **Система контролю версій**Перед початком роботи було створено гілку в GIT з назвою CGWsecond, для зручного зберігання та контролю версій програми.
2. **Audio Context**Спочатку створюється аудіоконтекст. Через обмеження автовідтворення аудіо в сучасних браузерах, контекст не може бути створений без взаємодії користувача з додатком. Тому було створено eventListener, який ініціалізує контекст тільки після натискання кнопки відтворення, який є частиною елемента<audio>і активується, коли елементу присвоюється параметр controls.
3. const audioElement = document.getElementById("audioElement");
4. audioElement.addEventListener("play", () => {
5. if (!audioContext) {
6. let AudioContext = window.AudioContext || window.webkitAudioContext;
7. audioContext = new AudioContext();
8. listener = audioContext.listener;
9. listener.positionX.value = 0;
10. listener.positionY.value = 0;
11. listener.positionZ.value = -5;
12. listener.forwardX.value = 0;
13. listener.forwardY.value = 0;
14. listener.forwardZ.value = -1;
15. sourceNode = audioContext.createMediaElementSource(audioElement);
16. pannerNode = audioContext.createPanner();
17. // Connect audio nodes and set up spatial audio properties
18. // Create a BiquadFilterNode
19. filterNode = audioContext.createBiquadFilter();
20. filterNode.type = "highpass"; // Set filter type to highpass
21. filterNode.frequency.value = 7777; // Set cutoff frequency
22. filterNode.Q.value = 1; // Set resonance/Q value
23. sourceNode.connect(pannerNode);
24. pannerNode.connect(audioContext.destination);
25. }
26. audioElement.play();
27. });
28. }

Приклад створення аудіоконтерксу з evenrListener

<audio id="audioElement" src="cherry.mp3" controls></audio><br>

Приклад елементу аудіо

Вихідні параметри призначаються джерелам звуку, наприклад позиція. Створені вузли є вихідними вузлами та просторовими вузлами. Після цього з’єднав вузли та створив свого роду ланцюжок викликів джерело-простір вузол-адресат (звуковий пристрій для фізичного інтерфейсу). Після цього починаємо відтворення звуку. Даний приклад під’єднання вузлів зображено на прикладі вище.

1. **Джерело звуку**Сфера - джерело звуку, була створена за допомогою функції createSurface з попередньої роботи. Вершини, нормалі та текстури з функції createSphere переносяться в буфери. Сфера рухається по колу, дотримуючись певної відстані від фігури. Контролюється магнітометром. Коли межа розділу матеріалу нахилена, розраховується нова матриця обертання, яка негайно множиться на матрицю переміщення. Сфера рухається, а у функції moveModelCGWRotationMatrix джерелу звуку призначаються нові координати шляхом. Вони обчислюються в цій функції та оновлюються у вузлі за допомогою функції setPosition.
2. **Фільтр**Щоб створити високочастотний фільтр, вузол фільтра ініціалізується типом високої частоти, частотою зрізу та резонансною частотою.

filterNode = audioContext.createBiquadFilter();

      filterNode.type = "highpass";

      filterNode.frequency.value = 7777;

      filterNode.Q.value = 1;

Приклад налаштування фільтру

Для роботи фільтра створено відповідний чекбокс для події. Коли прапорець увімкнуто, вузол простору вимкнено, підключений до вузла фільтра, який, у свою чергу, підключений до пункту призначення (звукового пристрою, призначення). При переході в стан «ВИМК.» відбувається зворотний процес. Приклад імплементації події зміни стану чекбоксу зображено нижче.

function toggleFilter() {

  const checkbox = document.getElementById("filterCheckbox");

  const filterEnabled = checkbox.checked;

  if (filterEnabled) {

    pannerNode.disconnect();

    filterNode.connect(audioContext.destination);

    pannerNode.connect(filterNode);

  } else {

    filterNode.disconnect();

    pannerNode.connect(audioContext.destination);

  }

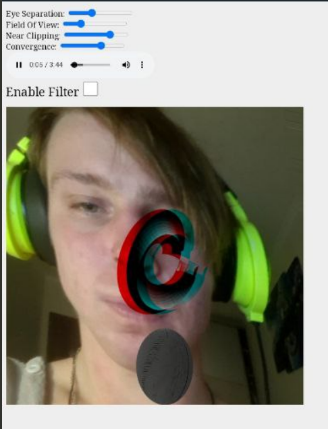
}

**Результат роботи**

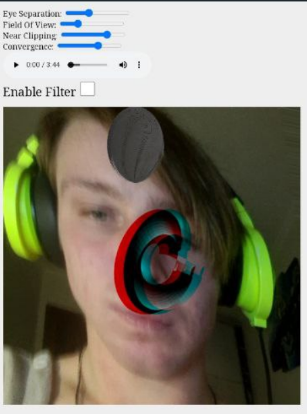
У прикладах нижче зображено фігуру та сферу, елементи інтерфейсу - результат виконання розрахунково графічної. Сфера є джерелом звуку та контролюється нахилом матеріального інтерфейсу вперед та назад. Сфера ближче до користувача (тобто знизу) – звук лунає гучніше з основного динаміку. В інтерфейсі додався флаг для фільтру та аудіодоріжка.



Приклад інтерфейсу

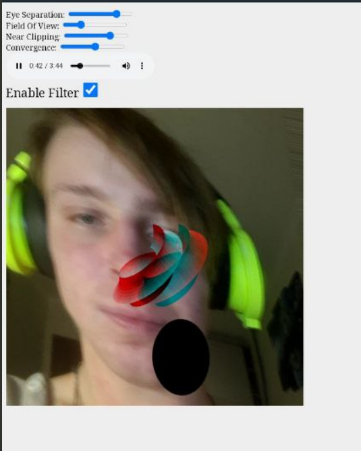


Сфера знизу



Сфера зверху

Під час роботи програми все ще можна змінювати параметри основної фігури. Натискання на чекбокс вмикає й вимикає фільтр високих частот.



Поверхня зі зміненими параметрами, чекбокс активований

**Лістинг**

**index.html**

<!DOCTYPE html>

<html>

<head>

  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

  <title>CGW</title>

  <style>

    body {

      background-color: #EEEEEE;

    }

    input.larger {

        width: 30px;

        height: 30px;

    }

    .bigger {

      font-size: 24px;

    }

  </style>

  <script src="./Utils/trackball-rotator.js"></script>

  <script src="./Utils/m4.js"></script>

  <script src="./shader.gpu"></script>

  <script src="./main.js"></script>

  <script src="./constants.js"></script>

</head>

<body onload="init()">

  <p>

    <label for="eyeSeparation">Eye Separation:</label>

    <input id="eyeSeparation" value="70" type="range" min="0" max="200" step="2" onchange="reDraw()" /><br>

    <label for="fieldOfView">Field Of View:</label>

    <input id="fieldOfView" value="0.8" type="range" min="0.01" max="3.14" step="0.01" onchange="reDraw()" /><br>

    <label for="nearClipping">Near Clipping:</label>

    <input id="nearClipping" value="5" type="range" min="-10" max="10" step="0.5" onchange="reDraw()" /><br>

    <label for="Convergence">Convergence:</label>

    <input id="Convergence" value="2000" type="range" min="100" max="3000" step="100" onchange="reDraw()" /><br>

    <audio id="audioElement" src="cherry.mp3" controls></audio><br>

    <label for="filterCheckbox" class="bigger">Enable Filter</label>

    <input type="checkbox" id="filterCheckbox" class="larger" onchange="toggleFilter()">

  </p>

  <div id="canvas-holder">

    <canvas width="600" height="600" id="webglcanvas" style="background-color:red"></canvas>

  </div>

</body>

</html>

**main.js**

"use strict";

let gl; // The webgl context.

let surface; // A surface model

let shProgram; // A shader program

let spaceball; // A SimpleRotator object that lets the user rotate the view by mouse.

let background; // A background model

let video; // Object holding video stream from camera

let stereoCam; // Object holding stereo camera calculation parameters

let textureWebCam; // Texture to optimize convertation from png to bmp and so on

let texture\_object; // Texture object holds a texture of type gl.TEXTURE\_2D for surface

let magnetometerData; // Object that holds data from magnetometer

let audioContext;

let sourceNode;

let pannerNode;

let filterNode;

let audioElement;

let playButton;

let listener;

let sphere;

let sphereTexture;

let point = { u: 200, v: 200 };

let b = 3;

let c = 2;

let d = 4;

let X = (u, v) =>

  0.05 \*

  (f(a, b, v) \*

    (1 + Math.cos(u) + (d \*\* 2 - c \*\* 2) \* ((1 - Math.cos(u)) / f(a, b, v)))) \*

  Math.cos(v);

let Y = (u, v) =>

  0.05 \*

  (f(a, b, v) \*

    (1 + Math.cos(u) + (d \*\* 2 - c \*\* 2) \* ((1 - Math.cos(u)) / f(a, b, v)))) \*

  Math.sin(v);

let Z = (u, v) =>

  0.05 \* (f(a, b, v) - (d \*\* 2 - c \*\* 2) / f(a, b, v)) \* Math.sin(u);

function f(a, b, j) {

  return (

    (a \* b) / Math.sqrt(a \*\* 2 \* Math.sin(j) \*\* 2 + b \*\* 2 \* Math.cos(j) \*\* 2)

  );

}

function deg2rad(angle) {

  return (angle \* Math.PI) / 180;

}

let pValue = 0;

const getCircleCords = () => {

  const p = Math.sin(pValue) \* 2.5;

  return [p, 10, -10 + p \* p];

};

function StereoCamera(

  Convergence,

  EyeSeparation,

  AspectRatio,

  FOV,

  NearClippingDistance,

  FarClippingDistance

) {

  this.mConvergence = Convergence;

  this.mEyeSeparation = EyeSeparation;

  this.mAspectRatio = AspectRatio;

  this.mFOV = FOV;

  this.mNearClippingDistance = NearClippingDistance;

  this.mFarClippingDistance = FarClippingDistance;

  this.mProjectionMatrix;

  this.mModelViewMatrix;

  this.ApplyLeftFrustum = function () {

    let top, bottom, left, right;

    top = this.mNearClippingDistance \* Math.tan(this.mFOV / 2);

    bottom = -top;

    let a = this.mAspectRatio \* Math.tan(this.mFOV / 2) \* this.mConvergence;

    let b = a - this.mEyeSeparation / 2;

    let c = a + this.mEyeSeparation / 2;

    left = (-b \* this.mNearClippingDistance) / this.mConvergence;

    right = (c \* this.mNearClippingDistance) / this.mConvergence;

    // Set the Projection Matrix

    this.mProjectionMatrix = m4.frustum(

      left,

      right,

      bottom,

      top,

      this.mNearClippingDistance,

      this.mFarClippingDistance

    );

    // Displace the world to right

    this.mModelViewMatrix = m4.translation(this.mEyeSeparation / 2, 0.0, 0.0);

  };

  this.ApplyRightFrustum = function () {

    let top, bottom, left, right;

    top = this.mNearClippingDistance \* Math.tan(this.mFOV / 2);

    bottom = -top;

    let a = this.mAspectRatio \* Math.tan(this.mFOV / 2) \* this.mConvergence;

    let b = a - this.mEyeSeparation / 2;

    let c = a + this.mEyeSeparation / 2;

    left = (-c \* this.mNearClippingDistance) / this.mConvergence;

    right = (b \* this.mNearClippingDistance) / this.mConvergence;

    // Set the Projection Matrix

    this.mProjectionMatrix = m4.frustum(

      left,

      right,

      bottom,

      top,

      this.mNearClippingDistance,

      this.mFarClippingDistance

    );

    // Displace the world to left

    this.mModelViewMatrix = m4.translation(this.mEyeSeparation / 2, 0.0, 0.0);

  };

}

// Constructor

function Model(name) {

  this.name = name;

  this.iVertexBuffer = gl.createBuffer();

  this.iNormalsBuffer = gl.createBuffer();

  this.iTextureBuffer = gl.createBuffer();

  this.count = 0;

  this.BufferData = function ({ vertexList, normalsList, textureList }) {

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

    gl.bufferData(

      gl.ARRAY\_BUFFER,

      new Float32Array(vertexList),

      gl.STATIC\_DRAW

    );

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

    gl.bufferData(

      gl.ARRAY\_BUFFER,

      new Float32Array(normalsList),

      gl.STATIC\_DRAW

    );

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

    gl.bufferData(

      gl.ARRAY\_BUFFER,

      new Float32Array(textureList),

      gl.STATIC\_DRAW

    );

    this.count = vertexList.length / 3;

  };

  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.iNormalsBuffer);

    gl.vertexAttribPointer(shProgram.iNormalsVertex, 3, gl.FLOAT, true, 0, 0);

    gl.enableVertexAttribArray(shProgram.iNormalsVertex);

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

    gl.enableVertexAttribArray(shProgram.iTextureCoords);

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

    if (this.name == "Sphere") {

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

    } else {

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

    }

  };

}

// Constructor

function ShaderProgram(name, program) {

  this.name = name;

  this.prog = program;

  // Location of the attribute variable in the shader program.

  this.iAttribVertex = -1;

  this.iNormalsVertex = -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.iViewWorldPosition = -1;

  this.iWMatrix = -1;

  this.iWInverseTranspose = -1;

  this.iLightWorldPosition = -1;

  this.iLightDir = -1;

  this.iTextureCoords = -1;

  this.iTMU = -1;

  this.iFScale = -1;

  this.iFPoint = -1;

  this.Use = function () {

    gl.useProgram(this.prog);

  };

}

/\* Draws a colored cube, along with a set of coordinate axes.

 \* (Note that the use of the above drawPrimitive function is not an efficient

 \* way to draw with WebGL.  Here, the geometry is so simple that it doesn't matter.)

 \*/

function draw() {

  gl.clearColor(0, 0, 0, 1);

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

  /\* 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 matAccumRotate0 = m4.multiply(rotateToPointZero, modelView);

  let matAccumTrans0 = m4.multiply(translateToPointZero, matAccumRotate0);

  // correct positioning of webcam texture

  let rotateToCenter = m4.axisRotation([0.0, 0.0, 1.0], Math.PI);

  let translateToCenter = m4.translation(2, 2, -10);

  let enlargedBackground = m4.scaling(4, 4.5, 1);

  // An identity matrix is a matrix that effectively represents 1.0

  // so that if you multiply by the identity nothing happens

  let matAccumIdent = m4.multiply(rotateToCenter, m4.identity());

  let matAccumEnlarge = m4.multiply(enlargedBackground, matAccumIdent);

  let matAccumView = m4.multiply(translateToCenter, matAccumEnlarge);

  /\* Multiply the projection matrix times the modelview matrix to give the

     combined transformation matrix, and send that to the shader program. \*/

  let modelViewProjection = m4.multiply(projection, matAccumTrans0);

  let worldInverseMatrix = m4.inverse(matAccumTrans0); // possible //no effect

  let worldInverseTransposeMatrix = m4.transpose(worldInverseMatrix);

  gl.uniformMatrix4fv(

    shProgram.iModelViewProjectionMatrix,

    false,

    modelViewProjection

  );

  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumView); //possible // very likely ?

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

  gl.uniform3fv(shProgram.iViewWorldPosition, [0, 0, 0]);

  gl.uniform3fv(shProgram.iLightWorldPosition, getCircleCords());

  gl.uniform3fv(shProgram.iLightDir, [0, -1, 0]);

  gl.uniformMatrix4fv(

    shProgram.iWInverseTranspose,

    false,

    worldInverseTransposeMatrix

  ); // possible

  gl.uniformMatrix4fv(shProgram.iWMatrix, false, matAccumTrans0); // possible

  /\* Draw the six faces of a cube, with different colors. \*/

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

  const scaleU = 1; //document.getElementById('textureScaleU').value;  // To disfunction sliders

  const scaleV = 1; //document.getElementById('textureScaleV').value;

  // console.log(scaleU, scaleV, point)

  gl.uniform2fv(shProgram.iFScale, [scaleU, scaleV]);

  gl.uniform2fv(shProgram.iFPoint, [X(point.u, point.v), Y(point.u, point.v)]);

  gl.uniform1i(shProgram.iTMU, 0);

  // Set values from sliders to stereo camera

  stereoCam.mEyeSeparation = document.getElementById("eyeSeparation").value;

  stereoCam.mFOV = document.getElementById("fieldOfView").value;

  stereoCam.mNearClippingDistance =

    document.getElementById("nearClipping").value - 0.001; // Not sure why, but need to substract some value

  stereoCam.mConvergence = document.getElementById("Convergence").value;

  // Get projection matrixes for both eyes

  stereoCam.ApplyLeftFrustum();

  let projectionR = stereoCam.mProjectionMatrix;

  stereoCam.ApplyRightFrustum();

  let projectionL = stereoCam.mProjectionMatrix;

  // First bind and draw is for video from webcam

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

  gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, video);

  gl.uniform1f(shProgram.iL, 0);

  background.Draw();

  gl.uniform1f(shProgram.iL, 10);

  // Need to call uniformMatrix4fv to correctly position and fix webcam texture.

  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumTrans0); //definetly not the best place for this line, but demonstration and convinient code reading is priority for now

  // Second bind and 2 draws is for surface texture for each eye (must be a better way)

  gl.bindTexture(gl.TEXTURE\_2D, texture\_object);

  // First pass for left eye, drawing blue+green (mask red)

  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionL);

  gl.colorMask(true, false, false, false); //reference for colours - https://registry.khronos.org/OpenGL-Refpages/gl4/html/glColorMask.xhtml

  surface.Draw();

  // Second pass for the right eye, drawing red (mask blue+green)

  gl.clear(gl.DEPTH\_BUFFER\_BIT);

  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projectionR);

  gl.colorMask(false, true, true, false);

  surface.Draw();

  gl.colorMask(true, true, true, true);

  //CGW

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

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

  gl.uniform1f(shProgram.iL, 10);

  gl.uniform1i(shProgram.iTexture, 0);

  gl.uniformMatrix4fv(

    shProgram.iModelViewMatrix,

    false,

    m4.multiply(

      projection,

      moveModelCGWRotationMatrix(calculateSurfaceRotation())

    )

  );

  sphere.Draw();

  gl.clear(gl.DEPTH\_BUFFER\_BIT);

  //CGW

}

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 = HelicoidSurface(u, v);

      let avert = HelicoidSurface(u + uStep, v);

      let bvert = HelicoidSurface(u, v + vStep);

      let cvert = HelicoidSurface(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);

    }

  }

  let normals = [];

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

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

      let vert = HelicoidSurface(u, v);

      let avert = HelicoidSurface(u + uStep, v);

      let bvert = HelicoidSurface(u, v + vStep);

      let cvert = HelicoidSurface(u + uStep, v + vStep);

      let verta0 = {

        x: avert.x - vert.x,

        y: avert.y - vert.y,

        z: avert.z - vert.z,

      };

      let vertb0 = {

        x: bvert.x - vert.x,

        y: bvert.y - vert.y,

        z: bvert.z - vert.z,

      };

      let vertca = {

        x: cvert.x - avert.x,

        y: cvert.y - avert.y,

        z: cvert.z - avert.z,

      };

      let vertba = {

        x: bvert.x - avert.x,

        y: bvert.y - avert.y,

        z: bvert.z - avert.z,

      };

      let norm = vec3Cross(verta0, vertb0);

      vec3Normalize(norm);

      let norma = vec3Cross(vertca, vertba);

      vec3Normalize(norma);

      normals.push(norm.x, norm.y, norm.z);

      normals.push(norm.x, norm.y, norm.z);

      normals.push(norm.x, norm.y, norm.z);

      normals.push(norma.x, norma.y, norma.z);

      normals.push(norma.x, norma.y, norma.z);

      normals.push(norma.x, norma.y, norma.z);

    }

  }

  return { vertexList, normalsList: normals, textureList: normals };

}

function vec3Cross(a, b) {

  let x = a.y \* b.z - b.y \* a.z;

  let y = a.z \* b.x - b.z \* a.x;

  let z = a.x \* b.y - b.x \* a.y;

  return { x: x, y: y, z: z };

}

function vec3Normalize(a) {

  var mag = Math.sqrt(a.x \* a.x + a.y \* a.y + a.z \* a.z);

  a.x /= mag;

  a.y /= mag;

  a.z /= mag;

}

c = 0.5;

function HelicoidSurface(u, v) {

  const a = 1;

  const b = c / Math.PI;

  const t = Math.PI / 2;

  let x = (a + x0(v) \* Math.cos(t) + y0(v) \* Math.sin(t)) \* Math.cos(u);

  let y = (a + x0(v) \* Math.cos(t) + y0(v) \* Math.sin(t)) \* Math.sin(u);

  let z = b \* u - x0(v) \* Math.sin(t) + y0(v) \* Math.cos(t);

  return { x: x, y: y, z: z };

}

function x0(v) {

  return c \* Math.pow(Math.cos(v), 3);

}

function y0(v) {

  return c \* Math.pow(Math.sin(v), 3);

}

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

let prog;

function initGL() {

  prog = createProgram(gl, vertexShaderSource, fragmentShaderSource);

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

  shProgram.Use();

  shProgram.iModelViewProjectionMatrix = gl.getUniformLocation(

    prog,

    "ModelViewProjectionMatrix"

  );

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

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

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

  shProgram.iNormalsVertex = gl.getAttribLocation(prog, "normal");

  shProgram.iColor = gl.getUniformLocation(prog, "color");

  shProgram.iWInverseTranspose = gl.getUniformLocation(

    prog,

    "wInverseTranspose"

  );

  shProgram.iWMatrix = gl.getUniformLocation(prog, "wMatrix");

  shProgram.iViewWorldPosition = gl.getUniformLocation(

    prog,

    "ViewWorldPosition"

  );

  shProgram.iLightWorldPosition = gl.getUniformLocation(

    prog,

    "LightWorldPosition"

  );

  shProgram.iLightDir = gl.getUniformLocation(prog, "lightDir");

  shProgram.iL = gl.getUniformLocation(prog, "l");

  shProgram.iTextureCoords = gl.getAttribLocation(prog, "textureCoords");

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

  shProgram.iFScale = gl.getUniformLocation(prog, "fScale");

  shProgram.iFPoint = gl.getUniformLocation(prog, "fPoint");

  surface = new Model("Surface");

  surface.BufferData(CreateSurfaceData());

  sphere = new Model("Sphere");

  sphere.BufferData(createSphere(1, 70, 70));

  LoadSphereTexture();

  stereoCam = new StereoCamera(2000, 35.0, 1.3, 45.0, 10.0, 20000); // "If something doesn't work - try to change numbers a bit"

  // Set to global video variable (or define and initialize variable globally(not good for consistency)).

  video = document.createElement("video");

  var constraints = { video: true };

  navigator.mediaDevices

    .getUserMedia(constraints)

    .then((stream) => {

      video.srcObject = stream;

      video.onloadedmetadata = function (e) {

        video.play();

      };

    })

    .catch(function (err) {

      console.log(err.name + ": " + err.message);

    }); // always check for errors at the end.

  gl.enable(gl.DEPTH\_TEST);

  // There is another way of doing this, but I failed when tried to implement.

  // reference - https://webglfundamentals.org/webgl/lessons/webgl-2d-drawimage.html

  background = new Model();

  background.BufferData({

    vertexList: [0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0],

    normalsList: [0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1],

    textureList: [1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1],

  });

  CreateWebCamTexture();

  LoadTexture();

  gl.enable(gl.DEPTH\_TEST);

  const audioElement = document.getElementById("audioElement");

  audioElement.addEventListener("play", () => {

    if (!audioContext) {

      let AudioContext = window.AudioContext || window.webkitAudioContext;

      audioContext = new AudioContext();

      listener = audioContext.listener;

      listener.positionX.value = 0;

      listener.positionY.value = 0;

      listener.positionZ.value = -5;

      listener.forwardX.value = 0;

      listener.forwardY.value = 0;

      listener.forwardZ.value = -1;

      sourceNode = audioContext.createMediaElementSource(audioElement);

      pannerNode = audioContext.createPanner();

      // Connect audio nodes and set up spatial audio properties

      // Create a BiquadFilterNode

      filterNode = audioContext.createBiquadFilter();

      filterNode.type = "highpass";

      filterNode.frequency.value = 7777;

      filterNode.Q.value = 1;

      sourceNode.connect(pannerNode);

      pannerNode.connect(audioContext.destination);

    }

    audioElement.play();

  });

}

function toggleFilter() {

  const checkbox = document.getElementById("filterCheckbox");

  const filterEnabled = checkbox.checked;

  if (filterEnabled) {

    pannerNode.disconnect();

    filterNode.connect(audioContext.destination);

    pannerNode.connect(filterNode);

  } else {

    filterNode.disconnect();

    pannerNode.connect(audioContext.destination);

  }

}

/\* Creates a program for use in the WebGL context gl, and returns the

 \* identifier for that program.  If an error occurs while compiling or

 \* linking the program, an exception of type Error is thrown.  The error

 \* string contains the compilation or linking error.  If no error occurs,

 \* the program identifier is the return value of the function.

 \* The second and third parameters are strings that contain the

 \* source code for the vertex shader and for the fragment shader.

 \*/

function createProgram(gl, vShader, fShader) {

  let vsh = gl.createShader(gl.VERTEX\_SHADER);

  gl.shaderSource(vsh, vShader);

  gl.compileShader(vsh);

  if (!gl.getShaderParameter(vsh, gl.COMPILE\_STATUS)) {

    throw new Error("Error in vertex shader:  " + gl.getShaderInfoLog(vsh));

  }

  let fsh = gl.createShader(gl.FRAGMENT\_SHADER);

  gl.shaderSource(fsh, fShader);

  gl.compileShader(fsh);

  if (!gl.getShaderParameter(fsh, gl.COMPILE\_STATUS)) {

    throw new Error("Error in fragment shader:  " + gl.getShaderInfoLog(fsh));

  }

  let prog = gl.createProgram();

  gl.attachShader(prog, vsh);

  gl.attachShader(prog, fsh);

  gl.linkProgram(prog);

  if (!gl.getProgramParameter(prog, gl.LINK\_STATUS)) {

    throw new Error("Link error in program:  " + gl.getProgramInfoLog(prog));

  }

  return prog;

}

/\*\*

 \* initialization function that will be called when the page has loaded

 \*/

function init() {

  let canvas;

  try {

    canvas = document.getElementById("webglcanvas");

    gl = canvas.getContext("webgl");

    if (!gl) {

      throw "Browser does not support WebGL";

    }

  } catch (e) {

    document.getElementById("canvas-holder").innerHTML =

      "<p>Sorry, could not get a WebGL graphics context.</p>";

    return;

  }

  try {

    initGL(); // initialize the WebGL graphics context

    //    initGLSphere();

  } catch (e) {

    document.getElementById("canvas-holder").innerHTML =

      "<p>Sorry, could not initialize the WebGL graphics context: " +

      e +

      "</p>";

    return;

  }

  spaceball = new TrackballRotator(canvas, draw, 0);

  setInterval(draw, 1 / 20);

}

window.addEventListener("keydown", (event) => {

  switch (event.key) {

    case "ArrowLeft":

      pValue -= 0.1;

      draw();

      break;

    case "ArrowRight":

      pValue += 0.1;

      draw();

      break;

    case "w":

      point.v = point.v + step;

      draw();

      break;

    case "s":

      point.v = point.v - step;

      draw();

      break;

    case "d":

      point.u = point.u + step;

      draw();

      break;

    case "a":

      point.u = point.u - step;

      draw();

      break;

    default:

      break;

  }

});

function LoadTexture() {

  // Use global object "texture object"

  texture\_object = gl.createTexture();

  let image = new Image();

  image.src =

    "https://www.the3rdsequence.com/texturedb/download/116/texture/jpg/1024/irregular+wood+planks-1024x1024.jpg";

  image.crossOrigin = "anonymous";

  image.onload = () => {

    // Make the "texture object" be the active texture object. Only the

    // active object can be modified or used. This also declares that the

    // texture object will hold a texture of type gl.TEXTURE\_2D. The type

    // of the texture, gl.TEXTURE\_2D, can't be changed after this initialization.

    gl.bindTexture(gl.TEXTURE\_2D, texture\_object);

    // Set parameters of the texture object.

    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\_T, gl.CLAMP\_TO\_EDGE);

    //www.the3rdsequence.com/texturedb/download/116/texture/jpg/1024/irregular+wood+planks-1024x1024.jpg

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

    // Tell gl to flip the orientation of the image on the Y axis. Most

    // images have their origin in the upper-left corner. WebGL expects

    // the origin of an image to be in the lower-left corner.

    gl.pixelStorei(gl.UNPACK\_FLIP\_Y\_WEBGL, 1);

    // Store in the image in the GPU's texture object

    gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

    draw();

  };

}

// Create texture and assign to a global texture variable

function CreateWebCamTexture() {

  textureWebCam = gl.createTexture();

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

  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);

  gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.FLOAT, video);

}

// CGW

function LoadSphereTexture() {

  sphereTexture = gl.createTexture();

  let image = new Image();

  image.src =

    "https://images.pexels.com/photos/1545743/pexels-photo-1545743.jpeg?cs=srgb&dl=pexels-yurii-hlei-1545743.jpg&fm=jpg";

  image.crossOrigin = "anonymous";

  image.onload = () => {

    // Make the "texture object" be the active texture object. Only the

    // active object can be modified or used. This also declares that the

    // texture object will hold a texture of type gl.TEXTURE\_2D. The type

    // of the texture, gl.TEXTURE\_2D, can't be changed after this initialization.

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

    // Set parameters of the texture object.

    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\_T, gl.CLAMP\_TO\_EDGE);

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

    // Tell gl to flip the orientation of the image on the Y axis. Most

    // images have their origin in the upper-left corner. WebGL expects

    // the origin of an image to be in the lower-left corner.

    gl.pixelStorei(gl.UNPACK\_FLIP\_Y\_WEBGL, 1);

    // Store in the image in the GPU's texture object

    gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

    draw();

  };

}

function createSphere(radius, latitudeBands, longitudeBands) {

  const positions = [];

  const indices = [];

  const textureList = [];

  for (let lat = 0; lat <= latitudeBands; lat++) {

    const theta = (lat \* Math.PI) / latitudeBands;

    const sinTheta = Math.sin(theta);

    const cosTheta = Math.cos(theta);

    for (let long = 0; long <= longitudeBands; long++) {

      const phi = (long \* 2 \* Math.PI) / longitudeBands;

      const sinPhi = Math.sin(phi);

      const cosPhi = Math.cos(phi);

      const x = cosPhi \* sinTheta;

      const y = cosTheta;

      const z = sinPhi \* sinTheta;

      positions.push(radius \* x, radius \* y, radius \* z);

      textureList.push(long / longitudeBands, lat / latitudeBands);

    }

  }

  for (let lat = 0; lat < latitudeBands; lat++) {

    for (let long = 0; long < longitudeBands; long++) {

      const first = lat \* (longitudeBands + 1) + long;

      const second = first + longitudeBands + 1;

      indices.push(first, second, first + 1);

      indices.push(second, second + 1, first + 1);

    }

  }

  // console.log("VertexList is: " + positions);

  return {

    vertexList: positions,

    normalsList: indices,

    textureList: textureList,

  };

}

function moveModelCGWRotationMatrix(compassHeadingM) {

  const centerX = 0;

  const centerY = 0;

  const centerZ = 0;

  // Define the radius of circular motion

  const radius = 3; // Adjust the radius as needed

  // Initialize the angle

  let angle = compassHeadingM / 4;

  const objectX = centerX + Math.cos(angle) \* radius;

  const objectY = centerY;

  const objectZ = centerZ + Math.sin(angle) \* radius;

  if (pannerNode != undefined) {

    pannerNode.setPosition(objectX, objectY, objectZ); // Update the position of the audio source

    pannerNode.setOrientation(0, 0, -1); // Set the orientation of the audio source

    // pannerNode.distanceModel = 'linear'; // Change the distance model if needed

  }

  let rotationMatrix = new Float32Array([

    Math.cos(angle),

    0,

    Math.sin(angle),

    0,

    0,

    1,

    0,

    0,

    -Math.sin(angle),

    0,

    Math.cos(angle),

    0,

    0,

    0,

    0,

    1,

  ]);

  let translateToCenter2 = m4.translation(objectX, objectZ, 0); //-10

  rotationMatrix = m4.multiply(translateToCenter2, rotationMatrix);

  return rotationMatrix;

}

function compassHeading(alpha, beta, gamma) {

  var degtorad = Math.PI / 180;

  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);

  // Calculate Vx and Vy components

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

  var Vy = -sZ \* sY + cZ \* sX \* cY;

  // Calculate compass heading

  var compassHeading = Math.atan(Vx / Vy);

  // Convert compass heading to use whole unit circle

  if (Vy < 0) {

    compassHeading += Math.PI;

  } else if (Vx < 0) {

    compassHeading += 2 \* Math.PI;

  }

  return compassHeading \* (180 / Math.PI); // Compass Heading (in degrees)

}

//60 times per second seems reasonable // turned out that NO - Maximum allowed frequency value for this sensor type is 10 Hz.

// reference: https://developer.mozilla.org/en-US/docs/Web/API/Magnetometer

let magSensor = new Magnetometer({ frequency: 10 });

magSensor.addEventListener("reading", (e) => {

  const alpha = magSensor.x;

  const beta = magSensor.y;

  const gamma = magSensor.z;

  magnetometerData = [alpha, beta, gamma];

});

magSensor.start();

function calculateSurfaceRotation() {

  if (magnetometerData != null) {

    // Calculate rotation

    return compassHeading(

      magnetometerData[0],

      magnetometerData[1],

      magnetometerData[2]

    );

  }

}