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

Звіт

з виконання розрахунково-графічної роботи на тему
«Просторове аудіо»
з дисципліни «Методи синтезу віртуальної реальності»

Виконав:
Студент 5-го курсу
ІАТЕ
групи ТР-23мп
Рудик Володимир Іванович

Перевірив:
Демчишин Анатолій Анатолійович

Київ-2023

Завдання

Варіант №16

1. Повторно використати код із практичного завдання №2.

2. Реалізувати обертання джерела звуку навколо геометричного центра поверхні за допомогою матеріального інтерфейсу (за варіантом: **software orientation sensor readings**). Цього разу поверхня залишається нерухомою, а джерело звуку переміщується. Відтворити улюблену пісню у форматі mp3/ogg з підтримкою просторового положення звуку, яким можна керувати.

3. Візуалізувати положення джерела звуку сферою.

4. Додати звуковий фільтр за допомогою інтерфейсу BiquadFilterNode (за варіантом: **фільтр високих частот**). Додати чек-бокс елемент, який вмикає та вимикає цей фільтр. Задати параметри для фільтру на свій смак.

Мета роботи: засвоїти навички роботи з просторовим аудіо та WebAudio HTML5 API.

Основне завдання: відтворити просторове аудіо за допомогою WebAudio HTML5 API.

Теоретичні відомості

Web Audio API - це високорівневе JavaScript API для обробки та синтезу звуку в веб-додатках. Метою цього API є включення можливостей, що присутні в сучасних ігрових аудіо-рушіях, а також виконання деяких завдань зі змішування, обробки та фільтрації звуку.

AudioContext призначений для керування та відтворення звуків. Щоб створити звук за допомогою Web Audio API, потрібно створити одне або кілька джерел звуку та підключити їх до призначення звуку, яке надається екземпляром AudioContext. Це підключення не обов'язково має бути прямим і може проходити через будь-яку кількість проміжних AudioNodes, які діють як модулі обробки аудіо-сигналу.

Один екземпляр AudioContext може підтримувати кілька вхідних звуків та складні аудіо-графи, тому для одного додатку потрібен лише один екземпляр.

Audio Sources у Web Audio API використовуються для представлення аудіо джерел. Вони є початковими елементами аудіо-графа і постачають аудіо-дані, які потім обробляються та відтворюються. Після того, як аудіо-сигнал пройшов крізь різні етапи обробки, він може бути підключений до аудіо-призначення (наприклад, AudioContext.destination), де відтворюється на динаміки або записується у файл.

Panner (або PannerNode) - це вузол в Web Audio API, який використовується для просторового позиціонування звукових джерел у тривимірному просторі. Він дозволяє контролювати положення звуку в просторі: панорамування (переміщення зліва направо) і нахил (переміщення вгору і вниз).

Panner приймає вхідний аудіо-сигнал і використовує різні параметри, щоб визначити положення звуку в тривимірному просторі:

- Position: визначає положення звуку в тривимірному просторі, використовуючи координати (x, y, z);

• Orientation: визначає орієнтацію звукового джерела, тобто його напрямок;

• Doppler effect: Моделює ефект Доплера, який виникає при рухомому джерелі звуку або слухачі.

BiquadFilterNode є одним з вузлів в Web Audio API і використовується для застосування фільтрації до аудіо сигналу. Він базується на математичному алгоритмі, відомому як "біквадратне рівняння" (biquadratic equation), що дозволяє виконувати різні типи фільтрації звуку.

BiquadFilterNode надає можливість створювати різноманітні фільтри, такі як нижньо- та верхньочастотні фільтри, шельфові фільтри та інші. Він також має параметри, такі як коефіцієнти фільтра, які визначають його тип і поведінку, а також рівень підсилення, частоту та Q-фактор, які визначають деталі фільтрації.

Highpass фільтр (високочастотний фільтр) є одним з типів фільтрів, який використовується для обробки звукового сигналу. Він дозволяє пропускати вищі частоти і приглушати або прибирати нижчі частоти.

Highpass фільтр працює за допомогою алгоритму, який використовує змінні параметри, такі як частота розсічення (cutoff frequency) і нахил (slope). Частота розсічення визначає точку, де фільтр починає приглушувати нижчі частоти і пропускати вищі. Нахил визначає, наскільки різко зменшується амплітуда нижчих частот.

При проходженні звукового сигналу через highpass фільтр, усі частоти нижче встановленої частоти розсічення приглушуються або відкидаються, тоді як вищі частоти проходять без змін. Це дозволяє виділити або підсилити високочастотні компоненти звуку, що призводить до більш прозорого або свіжого звучання.

Деталі реалізації

Перш за все було створено нову гілку з назвою CGW у репозиторії GitHub. Після виконання завдання програмний код, відео-демонстрацію та звіт було завантажено туди.

Потім було створено сферу для візуалізації джерела аудіо, весь процес якої аналогічний створенню фігури з попереднього кредитного модулю, тож далі будуть наведені лише основні моменти:

1) За допомогою параметричного рівняння сфери будуються її точки (Рисунок 1), з яких потім народжується сама фігура методом TRIANGLE_STRIP (цей процес аналогічний створенню основної фігури).

```
const CreateSphereData = (radius) => {
  const vertexList = [];
  const textureList = [];
  const splines = 20;

  const maxU = Math.PI;
  const maxV = 2 * Math.PI;
  const stepU = maxU / splines;
  const stepV = maxV / splines;

  const getU = (u) => {
    return u / maxU;
  };
  const getV = (v) => {
    return v / maxV;
  };

  for (let u = 0; u <= maxU; u += stepU) {
    for (let v = 0; v <= maxV; v += stepV) {
      const x = radius * Math.sin(u) * Math.cos(v);
      const y = radius * Math.sin(u) * Math.sin(v);
      const z = radius * Math.cos(u);
      vertexList.push(x, y, z);
      textureList.push(getU(u), getV(v));

      const xNext = radius * Math.sin(u + stepU) * Math.cos(v + stepV);
      const yNext = radius * Math.sin(u + stepU) * Math.sin(v + stepV);
      const zNext = radius * Math.cos(u + stepU);
      vertexList.push(xNext, yNext, zNext);
      textureList.push(getU(u + stepU), getV(v + stepV));
    }
  }

  return {
    verticesSphere: vertexList, texturesSphere: textureList,
  };
};
```

Рисунок 1 – Функція для побудови сфери

2) Перед відмалюванням сфери на неї накладається текстура (Рисунок 2), яка попередньо завантажується у функції loadSphereTexture (Рисунок 3).

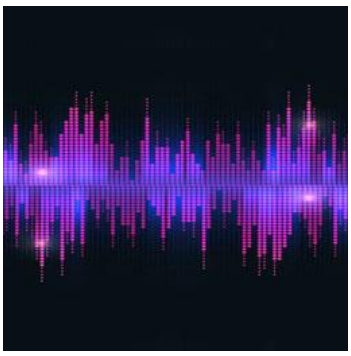

Рисунок 2 – Текстура для сфери

```
576  const loadSphereTexture = () => {
577    const image = new Image();
578    image.crossOrigin = "anonymous";
579    image.src = "./sphere-texture.jpg";
580
581    image.addEventListener("load", () => {
582      textureSphere = gl.createTexture();
583      gl.bindTexture(gl.TEXTURE_2D, textureSphere);
584      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
585      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
586      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
587    });
588  };
```

Рисунок 3 – Функція створення текстури для сфери

Після цього сфера рендериться у функції draw. На цьому етапі створення фігури завершено, далі потрібно додати звук і прив’язати його до її координат.

Перед налаштуванням та роботою зі звуком потрібно додати його у html файл (Рисунок 4) за допомогою тегу audio та вказати посилання на файл або його розташування у тезі source. Також було вказано параметри для відображення кнопок управління звуком та для його зациклення. У якості аудіо в цій роботі використовується головна тема з серіалу Гра Престолів.

```
151      <audio
152        id="audio"
153        style="margin-top: 15px; margin-bottom: 15px"
154        controls
155        loop
156      >
157        <source src="audio.mp3" type="audio/mpeg" />
158        Your browser does not support the audio element.
159      </audio>
```

Рисунок 4 – Аудіо елемент в html файлі

Тепер можна виконувати налаштування аудіо:

1) Спочатку потрібно створити аудіо-контекст за допомогою window.AudioContext або window.webkitAudioContext для старіших версій

браузерів, початковий елемент AudioSource та вузли для обробки, такі як PannerNode та BiquadFilterNode (Рисунок 5).

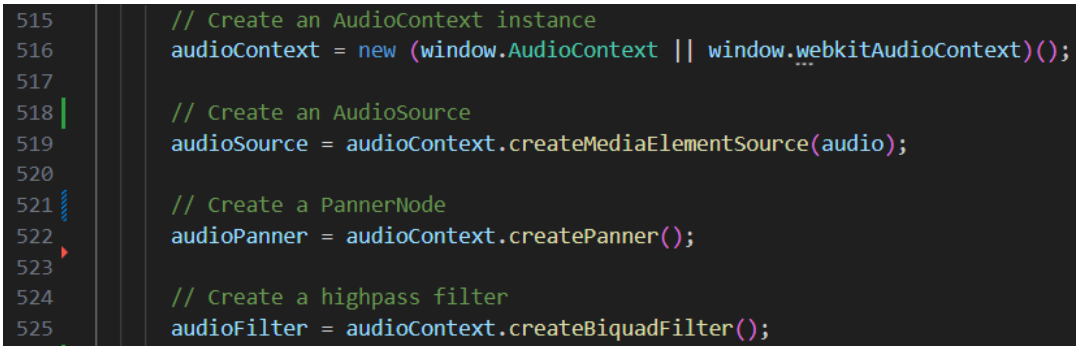

Рисунок 5 – Створення основних об’єктів для роботи з аудіо

2) Після створення об’єктів потрібно виконати їх налаштування (Рисунок 6). Для об’єкту audioPanner було вказано модель панування (panningModel) зі значенням **HRTF** (Head-Related Transfer Function), та модель відстані (distanceModel) зі значенням **linear** для лінійного зменшення звуку пропорційно до відстані. Для об’єкту audioFilter вказано тип фільтру згідно з варіантом **highpass** (високочастотний фільтр) та частоту зрізу (початкове значення **500**), яку можна змінювати у режимі реального часу.

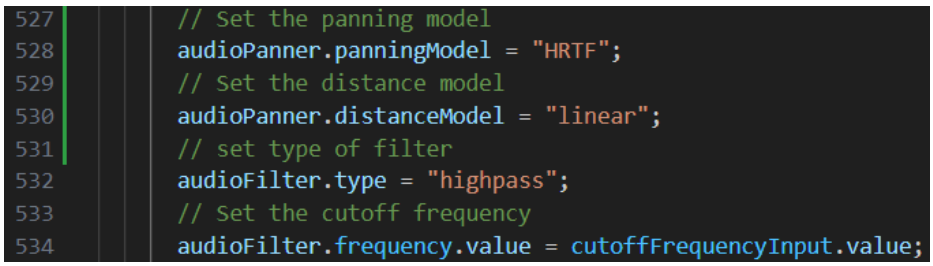

Рисунок 6 – Налаштування PannerNode та BiquadFilterNode

Для завершення налаштувань потрібно зв’язати отриманий ланцюжок за допомогою функцій connect (Рисунок 7).

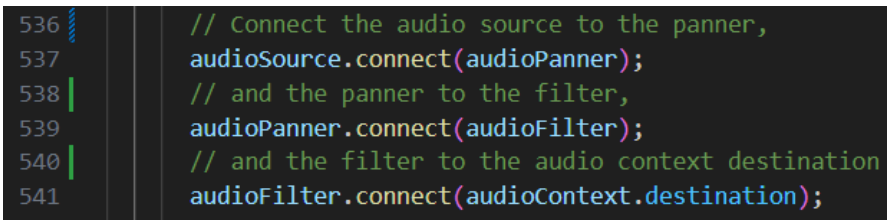

Рисунок 7 – Зв’язка об’єктів у систему

Також було реалізовано зміну координат сфери та частоти зрізу, ввімкнення та вимкнення фільтру через інтерфейс додатку, однак їх реалізація є очевидною і не потребує пояснень.

Інструкція користувача

Розроблений додаток має наступний вигляд (Рисунок 8). Зверху знаходиться опис, потім набір елементів для зміни різних параметрів фігури, звуку, сфери.

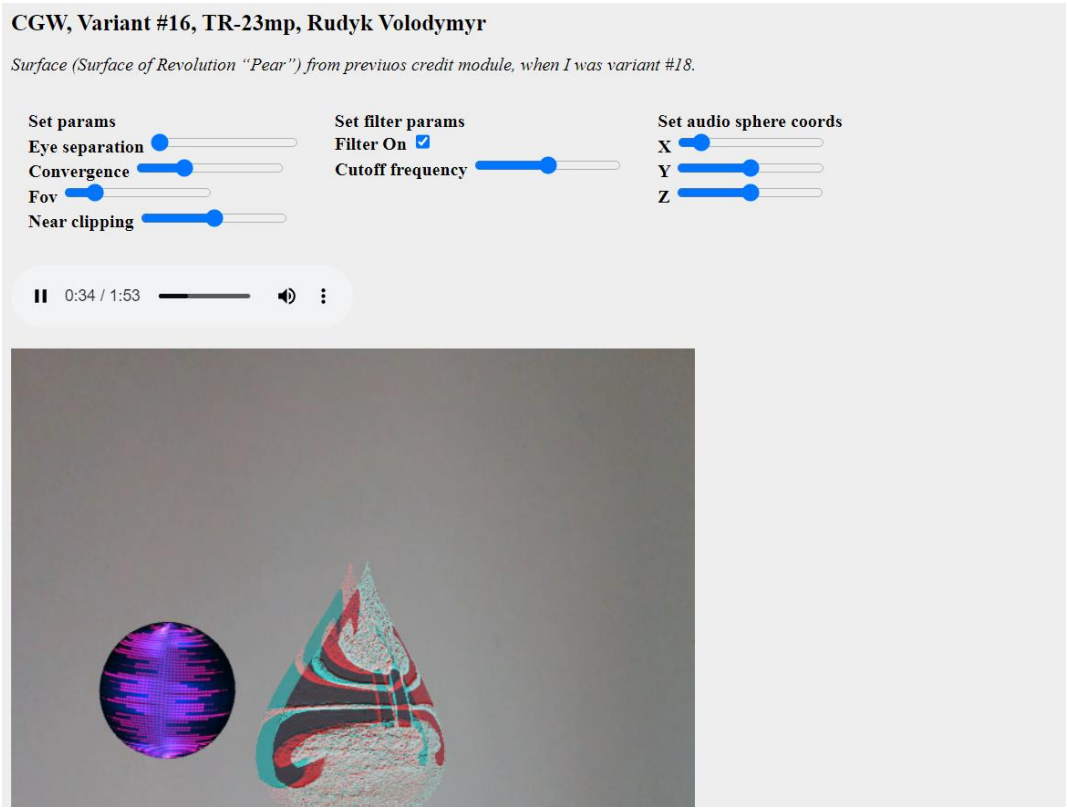

Рисунок 7 – Інтерфейс додатку

Перша колонка елементів відповідає за зміну параметрів 3D основної фігури, таких як eye separation, convergence distance, field of view та near clipping distance.

Друга колонка відповідає за зміну параметрів фільтру. Тут можна ввімкнути або вимкнути фільтр (за замовчуванням він увімкнений) та змінити значення частоти зрізу. Усі зміни моментально відображаються на екрані та впливають на звук.

Третя колонка відповідає за зміну координат сфери (за замовчуванням ця колонка не потрібна, оскільки сфера обертається навколо фігури на основі даних від датчика орієнтації).

Нижче знаходиться елемент аудіо, який складається з кнопки паузи\старту відтворення, відображення часу відтворення, кнопки для регулювання звуку та додаткових опцій (завантажити аудіо файл або змінити швидкість програвання).

І в кінці сторінки знаходиться елемент canvas, у якості фону на якому зображене відео з веб-камери пристрою, поверх якого знаходяться дві фігури: основна фігура та сфера-джерело-аудіо.

На рисунках 8-9 можна побачити зміну положення сфери в залежності від положення девайсу.

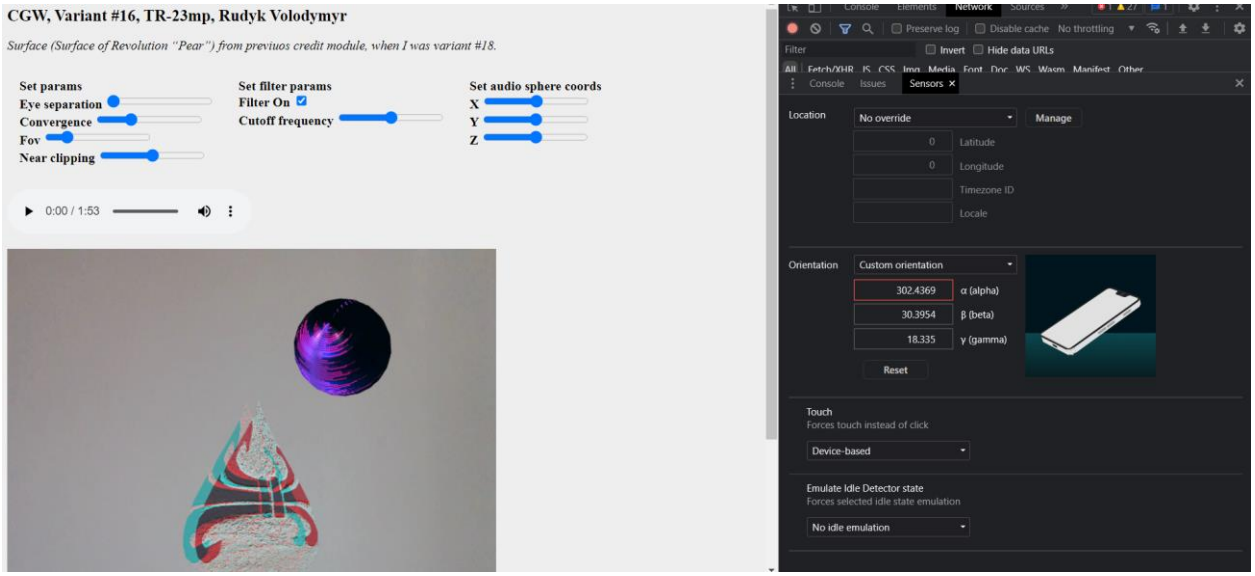

Рисунок 8 – Стан додатку після зміни положення девайсу

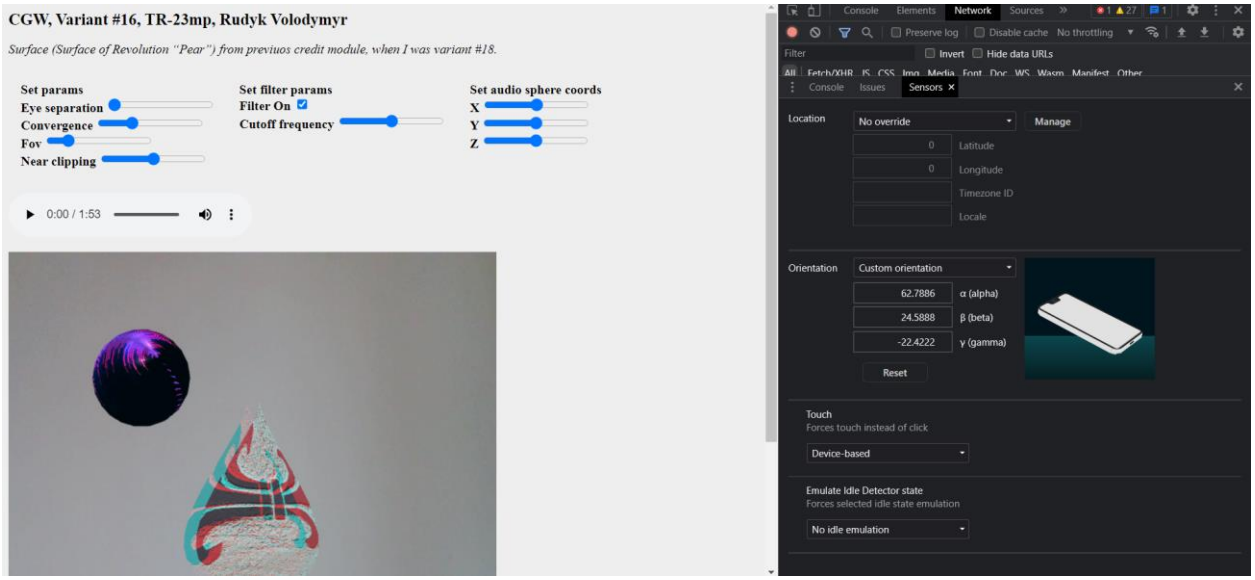

Рисунок 9 – Стан додатку після ще однієї зміни положення девайсу

Лістинг коду

main.js

```
let sphere, textureSphere;
let audio = null;
let audioContext, audioSource, audioPanner, audioFilter, cutoffFrequency;
let xPosition = 0;
let yPosition = 0;
let zPosition = 0;

function draw() {
  gl.clearColor(0, 0, 0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  let projection = m4.orthographic(0, 1, 0, 1, -1, 1);
  let modelView = spaceball.getViewMatrix();
  let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0);

...

  let modelView1 = null;
  gl.bindTexture(gl.TEXTURE_2D, textureSphere);
  xPosition = parseFloat(document.getElementById("xPosition").value);
  yPosition = parseFloat(document.getElementById("yPosition").value);
  zPosition = parseFloat(document.getElementById("zPosition").value);

  if (
    orientationEvent.alpha &&
    orientationEvent.beta &&
    orientationEvent.gamma
  ) {
    let rotationMatrix = getRotationMatrix(
      orientationEvent.alpha,
      orientationEvent.beta,
      orientationEvent.gamma
    );
    let translationMatrix = m4.translation(0, 0, -1);

    xPosition = orientationEvent.gamma;
    yPosition = orientationEvent.beta;
    zPosition = orientationEvent.alpha;
    modelView1 = m4.multiply(rotationMatrix, translationMatrix);
  }

  if (audioPanner) {
    cutoffFrequency = document.getElementById("cutoffFrequency");
    audioFilter.frequency.value = parseFloat(cutoffFrequency.value);
    let pannerCoef = 0.7;
```

```
    audioPanner.setPosition(
      xPosition * pannerCoef,
      yPosition * pannerCoef,
      zPosition
    );
  }

  let translateToPointZero = m4.translation(50, 50, 0);
  const translationMatrix = m4.translation(xPosition, yPosition, zPosition);
  let matAccumSphere = m4.multiply(
    rotateToPointZero,
    modelView1 ? modelView1 : modelView
  );
  let translationMatrixSphere = m4.multiply(translationMatrix, matAccumSphere);
  const scaleMatrix = m4.scaling(0.01, 0.01, 0.01);
  const matAccumSphere0 = m4.multiply(
    scaleMatrix,
    m4.multiply(translateToPointZero, translationMatrixSphere)
  );
  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projection);
  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccumSphere0);
  sphere.Draw();
  gl.clear(gl.DEPTH_BUFFER_BIT);

...
}

const CreateSphereData = (radius) => {
  const vertexList = [];
  const textureList = [];
  const splines = 20;
  const maxU = Math.PI;
  const maxV = 2 * Math.PI;
  const stepU = maxU / splines;
  const stepV = maxV / splines;
  const getU = (u) => {
    return u / maxU;
  };
  const getV = (v) => {
    return v / maxV;
  };
  for (let u = 0; u <= maxU; u += stepU) {
    for (let v = 0; v <= maxV; v += stepV) {
      const x = radius * Math.sin(u) * Math.cos(v);
      const y = radius * Math.sin(u) * Math.sin(v);
      const z = radius * Math.cos(u);
      vertexList.push(x, y, z);
      textureList.push(getU(u), getV(v));
      const xNext = radius * Math.sin(u + stepU) * Math.cos(v + stepV);
      const yNext = radius * Math.sin(u + stepU) * Math.sin(v + stepV);
      const zNext = radius * Math.cos(u + stepU);
```

```
      vertexList.push(xNext, yNext, zNext);
      textureList.push(getU(u + stepU), getV(v + stepV));
    }
  }

  return {
    verticesSphere: vertexList,
    texturesSphere: textureList,
  };
};

function initGL() {
  ...

  sphere = new Model("Sphere");
  const { verticesSphere, texturesSphere } = CreateSphereData(10);
  sphere.BufferData(verticesSphere, texturesSphere);

...

  loadSphereTexture();

  gl.enable(gl.DEPTH_TEST);
}

function init() {

...

  // get custom audio sphere coords
  const xPositionInput = document.getElementById("xPosition");
  const yPositionInput = document.getElementById("yPosition");
  const zPositionInput = document.getElementById("zPosition");
  const cutoffFrequencyInput = document.getElementById("cutoffFrequency");
  // add eventListeners for inputs custom audio sphere coords
  xPositionInput.addEventListener("input", draw);
  yPositionInput.addEventListener("input", draw);
  zPositionInput.addEventListener("input", draw);
  cutoffFrequencyInput.addEventListener("input", draw);

  ...

  // get audio element
  audio = document.getElementById("audio");

  // add eventListener for audio pause
  audio.addEventListener("pause", () => {
    audioContext.resume();
  });

  // add eventListener for audio play
```

```
audio.addEventListener("play", () => {
  if (!audioContext) {
    // Create an AudioContext instance
    audioContext = new (window.AudioContext || window.webkitAudioContext)();

    // Create an AudioSource
    audioSource = audioContext.createMediaElementSource(audio);

    // Create a PannerNode
    audioPanner = audioContext.createPanner();

    // Create a highpass filter
    audioFilter = audioContext.createBiquadFilter();

    // Set the panning model
    audioPanner.panningModel = "HRTF";
    // Set the distance model
    audioPanner.distanceModel = "linear";
    // set type of filter
    audioFilter.type = "highpass";
    // Set the cutoff frequency
    audioFilter.frequency.value = cutoffFrequencyInput.value;

    // Connect the audio source to the panner,
    audioSource.connect(audioPanner);
    // and the panner to the filter,
    audioPanner.connect(audioFilter);
    // and the filter to the audio context destination
    audioFilter.connect(audioContext.destination);

    audioContext.resume();
  }
});

// get element of filter
let filterOn = document.getElementById("filterCheckbox");

// add eventListener for change filter checkbox
filterOn.addEventListener("change", function () {
  if (filterOn.checked) {
    audioPanner.disconnect();
    audioPanner.connect(audioFilter);
    audioFilter.connect(audioContext.destination);
  } else {
    audioPanner.disconnect();
    audioPanner.connect(audioContext.destination);
  }
});

audio.play();
```

```
  reDraw();
}

const loadSphereTexture = () => {
  const image = new Image();
  image.crossOrigin = "anonymous";
  image.src = "./sphere-texture.jpg";

  image.addEventListener("load", () => {
    textureSphere = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, textureSphere);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
  });
};
```