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 5 курсу

групи ТР-21мп, ІАТЕ

Островой Денис Олегович

Київ – 2023

Пояснення завдання

Суттю завдання є ознайомитись з роботою зі звуком в WebGL, а саме створення джерела звуку додавання фільтру частот в залежності від варіанту, в моєму випадку це фільтр високих частот (варіант 16), додавання можливості рухати джерело звуку і в залежності від цього змінювати уявну відстань, яку проходить звук від чого зменшується або збільшується гучність, чим ближче тим гучніше і навпаки, іншими словами, створити об’ємний звук. Для виконання цього завдання треба користуватись напрацюваннями з 2 лабораторної роботи і зверху імплементувати функціонал Web Audio API. Також потрібно візуалізувати джерело звуку потрібно для позначення використати сферу. За бажанням можна використати улюблену пісню для представлення звуку.

Теоретичні відомості

Для ознайомлення нам були представленні лекції викладача, а також посилання на інтернет ресурс де чітко показано як працювати з технологією, представлено список функцій та параметрів цих функцій.

Основна ідея за Web Audio API полягає у створенні аудіо-графа, який складається з різних аудіо-вузлів. Ці вузли представляють оброблювальні блоки, які здійснюють різні операції зі звуком, такі як генерація звуку, зміна гучності, фільтрація, ефекти та інше. Вони можуть бути з'єднані між собою для створення складних звукових ланцюжків.

Для HTML5 введення елемента аудіо було важливим кроком у поліпшенні можливостей веб-аудіо. Однак для складніших аудіододатків, таких як веб або флеш-ігри або інтерактивні сайти, потрібно більше. Ця технологія була розроблена з метою включення сучасних звукових можливостей, обробки та фільтрації аудіо, які використовуються в професійних програмах для створення звуку. Web Audio API розроблено таким чином, щоб задовольняти різноманітні потреби користувачів веб-аудіо, включаючи складні ігри та музичні додатки. Хоча API може не мати всіх можливостей програмного забезпечення для настільних комп'ютерів, воно є потужним інструментом для реалізації різноманітних аудіо функцій у браузері, особливо в поєднанні з графічними можливостями WebGL.

Web Audio API включає в себе базовий набір функціоналу, що дозволяють працювати зі звуков в браузері, наприклад:

1. Аудіо-контекст (AudioContext) - це основний об'єкт API, який ініціалізує та керує аудіо-системою.
2. Аудіо-вузли (Audio Nodes) - це блоки, які обробляють аудіо-сигнали і можуть бути з'єднані між собою для створення аудіо-графів.
3. Роутинг аудіо (Audio Routing) - це процес з'єднання аудіо-вузлів для створення маршрутів аудіо-сигналу від джерела до виходу.
4. Відтворення звуку - можливість завантажувати аудіо-файли та відтворювати їх з використанням аудіо-вузлів.
5. Ефекти та обробка звуку - наявність ефектів та фільтрів, які можна застосовувати до аудіо-сигналу для створення різноманітних звукових ефектів. Існують такі фільтри:
    a. Фільтр низьких частот (Low Pass)
    b. Фільтр високих частот (High Pass)

c. Полосовий фільтр (Band Pass)
d. Фільтр низького рівня (Low Shelf)
e. Фільтр верхнього рівня(High Shelf)
f. Піковий фільтр (Peaking)
g. Фільтр пропускання (Notch)
h. Фільтр всіх частот (Allpass)

6. Потік аудіо (Audio stream) - створення та маніпуляція потоками звуку завдяки скриптам.

Web Audio API відкриває широкі можливості для створення інтерактивних та звуко насичених веб-додатків. Одним з інтерфейсів, який дозволяє застосовувати звукові фільтри, є BiquadFilterNode. Цей інтерфейс дозволяє налаштовувати параметри фільтра, такі як тип фільтра, частоту зрізу та підсилення, для обробки аудіо-сигналу.

Також є підтримка модульної маршрутизації, яка дозволяє довільні з'єднання між різними об'єктами аудіо вузлів. Кожен вузол може мати входи та/або виходи. Вихідний вузол не має входів і єдиний вихід. Вузол призначення має один вхід і не має виходів. Інші вузли, такі як фільтри, можна розмістити між вузлами джерела та призначення. Розробнику не потрібно турбуватися про деталі низькорівневого формату потоку, коли два об'єкти з'єднані разом. Наприклад, якщо монофонічний аудіопотік під'єднано до стерео входу, він має просто належним чином мікшувати лівий і правий канали.

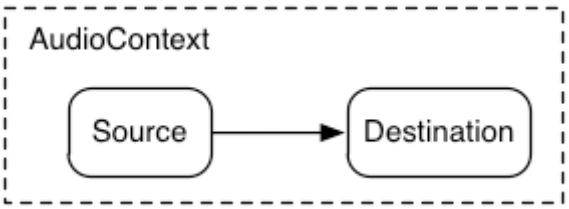

Рис. 2.1 Приклад модульної маршрутизації.

В висновку хочу сказати, що цей інструмент є незамінним в сьогоденні. Web Audio API є важливим компонентом для створення багатофункціональних та інтерактивних веб-додатків, які працюють зі звуком. Він надає веб-розробникам широкі можливості для контролю та маніпуляції звуковими елементами, що додає потрібну всім функціональність та взаємодію з користувачем.

Деталі виконання роботи

Так, як для створення розрахункової роботи потрібно було використовувати код з 2 лабораторної роботи, то про розробку усього крім завдання РГР сенсу говорити немає.

Спочатку було створено нові зміні для роботи зі звуком.

```
let audio;
let filter;
let soundtrack;
let source;
let checkBox;
let listener;
let sphereSurface;
```

Рис. 3.1 Список змінних для розробки.

Потім було створено чекбокс для ввімкнення та вимкнення фільтру, та ініціалізовано та прописано шля для звукового файла в HTML документі.

Далі, було створено аудіо контекст для подальшої маніпуляції за звуком.

```
if (!audio) {
  audio = new (window.AudioContext || window.webkitAudioContext)();
```

Рис. 3.2 Створення контексту.

Далі, було створено джерело звуку, панораматора для передачі інформації про рух звуку та фільтр звуку і підв’язані до контексту.

```
source = audio.createMediaElementSource(track);
panner = audio.createPanner();
filter = audio.createBiquadFilter();

source.connect(panner);
panner.connect(filter);
filter.connect(audio.destination);
```

Рис. 3.3 Створення Джерела, панораматора та фільтру.

Далі відбувається налаштування фільтра, вибір його типу, межі частот, такаож за бажанням можна виставити інші властивості

```
filter.type = 'highpass';
filter.frequency.value = 4000;
```

Рис. 3.4 Налаштування фільтру.

Далі було створено функцію enableAudio(), яка містить в собі виклик попередньої функції, ініціалізація чекбоксу для роботи з фільтром, а також робота зі звуком.

```
function enableAudio() {
  audioPlay();
  const checkBox = document.getElementById('filter');
  const track = document.getElementById('audioElement');
  track.addEventListener('pause', () => {
    audio.resume();
  })

  checkBox.addEventListener('change', function() {
    if (checkBox.checked) {
      panner.disconnect();
      panner.connect(filter);
      filter.connect(audio.destination);
    } else {
      panner.disconnect();
      panner.connect(audio.destination);
    }
  });
  track.play();
}
```

Рис. 3.5 Функція enableAudio().

Далі було додано сферу на екран, і пов’язано її рух джерелом звуку.

```
gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, m4.multiply(m4.translation(t[0]*1.5,t[1]*1.5,t[2]*1.5),matAccumRight));
sphereSurface.Draw();
if(panner){
  panner.setPosition(t[0]*1.5,t[1]*1.5,t[2]*1.5)
```

Рис. 3.6 Створення сфери.

Взаємодія користувача з додатком

На сторінці користувач буде бачити назву, головне вікно з фігурою, джерелом звуку, повзунками, які допомагають змінювати розмір фігури, полодення частин фігури, чекбокс з можливістю управління фільтром, і музичний програвач.

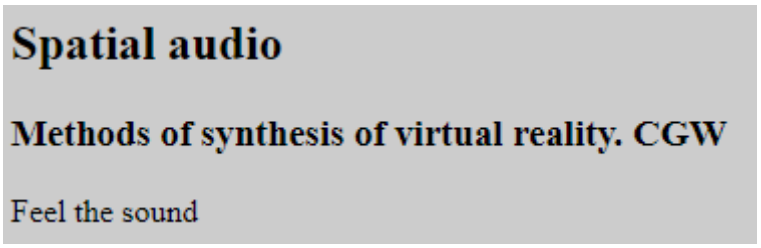

Рис. 4.1 Назва.

Рси. 4.2 Музикальний програвач, фільтр та повзунки управління.

Рис 4.3 Площина з джерелом звуку зверху.

На рисунку (4.3) видно, що джерело звуку знаходиться зверху, тобто звук буде йти зверху і майже не буде помітно різницю між каналами.

При повороті телефону джерело звуку змінить свою позицію.

Рис. 4.4 Джерело зліва.

Внаслідок знаходження джерела зліва звук гарно чути тільки в лівому каналі(навушнику). При повороті в іншу сторону звук починає краще звучати в правому каналі.

Рис 4.5 Джерело справа.

Також працює фільтр високих частот, який ріже звук і робить його більш плоским.

Приклад коду

```
let audio;
let filter;
let soundtrack;
let source;
let checkBox;
let listener;
let sphereSurface;
```

...

```
function sphere(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 enableSphere(){
  sphereSurface = new Model('sphereSurface');
  sphereSurface.BufferData(createSphereSurface(0.2))
  sphereSurface.TextureBufferData(createSphereSurface(0.2))
}
```

...

```
let track = null;
let panner;
function audioPlay() {
  const track = document.getElementById('audioElement');
  track.addEventListener('play', initializeAudio);
}

function initializeAudio() {
  const track = document.getElementById('audioElement');
  if (!audio) {
    audio = new (window.AudioContext || window.webkitAudioContext)();
    source = audio.createMediaElementSource(track);
    panner = audio.createPanner();
    filter = audio.createBiquadFilter();

    connectAudioNodes();

    configureFilter();
```

```
    audio.resume();
  }
}

function connectAudioNodes() {
  source.connect(panner);
  panner.connect(filter);
  filter.connect(audio.destination);
}

function configureFilter() {
  filter.type = 'highpass';
  filter.frequency.value = 4000;
}
```

...

```
function enableAudio() {
  audioPlay();
  const checkBox = document.getElementById('filter');
  const track = document.getElementById('audioElement');
  track.addEventListener('pause', () => {
    audio.resume();
  })

  checkBox.addEventListener('change', function() {
    if (checkBox.checked) {
      panner.disconnect();
      panner.connect(filter);
      filter.connect(audio.destination);
    } else {
      panner.disconnect();
      panner.connect(audio.destination);
    }
  });
  track.play();
}
```

...

```
function rotateAroundZ(angle, vector) {
  const rotZ = [
      [Math.cos(angle), -Math.sin(angle), 0],
      [Math.sin(angle), Math.cos(angle), 0],
      [0, 0, 1]
  ];
```

```
  return multiplyMatrixVector(rotZ, vector);
}

function rotateAroundY(angle, vector) {
  const rotY = [
      [Math.cos(angle), 0, Math.sin(angle)],
      [0, 1, 0],
      [-Math.sin(angle), 0, Math.cos(angle)]
  ];
  return multiplyMatrixVector(rotY, vector);
}

function rotateAroundX(angle, vector) {
  const rotX = [
      [1, 0, 0],
      [0, Math.cos(angle), -Math.sin(angle)],
      [0, Math.sin(angle), Math.cos(angle)]
  ];
  return multiplyMatrixVector(rotX, vector);
}
```