Міністерство освіти й науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра цифрових технологій в енергетиці

Звіт
«Візуалізація графічної та геометричної інформації»

Розрахунково графічна робота
Варіант №18

Виконав:
студент 5-го курсу
групи ТР-23мп ІАТЕ
Сірик О.О.
Перевірив: Демчишин А.А.

Київ-2023

Завдання

Використавши код із практичного завдання №2:

- реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою інтерфейсу сенсора (цього разу поверхня залишається нерухомою, а джерело звуку рухається). Відтворити улюблену пісню у форматі mp3/ogg, маючи просторове розташування джерела звуку, кероване користувачем;
- візуалізувати положення джерела звуку за допомогою сфери; додайте звуковий фільтр (використовуйте інтерфейс BiquadFilterNode) для кожного варіанту.
- додайте перемикач, який вмикає або вимикає фільтр. Встановіть параметри фільтра на свій смак.

Теоритичні відомості

Web Audio API - це високорівневий інтерфейс JavaScript для обробки і синтезу аудіоданих в веб-додатках. Він надає об'єктно-орієнтований фреймворк для маніпулювання аудіо на веб-сторінці, дозволяючи розробникам генерувати звук, декодувати аудіофайли, завантажувати їх віддалено, контролювати гучність, створювати візуалізації і багато іншого.

Для початку роботи з Web Audio API потрібно створити аудіоконтекст. Він представляє аудіо-оточення і дозволяє створювати об'єкти, які виробляють або обробляють звук.

```
var audioCtx = new (window.AudioContext ||
window.webkitAudioContext)();
```

Web Audio API включає ряд вбудованих вузлів для генерування і обробки звуку. Наприклад, для створення синусоїди можна використовувати OscillatorNode.

```
var oscillator = audioCtx.createOscillator();
oscillator.type = 'sine'; // тип волни
oscillator.frequency.value = 440; // частота в Гц
oscillator.connect(audioCtx.destination); //
підключення до вихідного вузла
oscillator.start(); // запуск генерації звуку
```

Web Audio API також дозволяє розробникам працювати з аудіо буферами для більш складних задач, наприклад, розробки інструментів для змішування звуку або синтезу музики.

Аудіо вузли об'єднуються в ланцюги та прості мережі їх введеннями та висновками. Вони зазвичай запускаються з одним або більше джерелами. Джерела є масиви семплів на одиницю часу. Наприклад, при частоті дискретизації 44100 Гц, у кожній секунді кожного каналу розташовано 22050 семплів. Вони можуть бути або оброблені математично (дивіться: OscillatorNode), або зчитані з звуко/відео файлів (дивіться: AudioBufferSourceNode і MediaElementAudioSourceNode) або з аудіо потоків (дивіться: MediaStreamAudioSourceNode). По факту, звукові файли – просто запис звукових коливань, які надходять від мікрофона та музичних інструментів, змішуючись в одну складну хвилю. Вивідні дані цих вузлів можуть бути пов'язані з вступними інших, що змішують або модифікують потоки звукових зразків в інші потоки. Популярна модифікація - множення зразка на значення, щоб зробити вихідний звук меншим або більш гучним (дивіться GainMode). Коли звук був успішно оброблений призначеним йому ефектом, він може бути прив'язаний до вихідного потоку (дивіться AudioContext.destination), що направляє звук в динаміки або мікрофон. Даний крок потрібен лише якщо ви надасте можливість користувачу почути ваші шедеври.

Хід роботи

У ході виконання роботи було розроблено веб-застосунок, який використовує браузерні API, такі як Magnetometer, WebAudio та WebGL. Застосунок має на меті відрендерити фігуру на канвасі та створити рухому сферу, яка рухається з використанням магнетометра. Застосунок також надає можливість змінювати позицію аудіо доріжки, яка грає на фоні, залежно від координат сфери. Додатково, реалізовано можливість включення lowpass фільтра для звуку.

Опишемо детально технічні аспекти розробки цього застосунку:

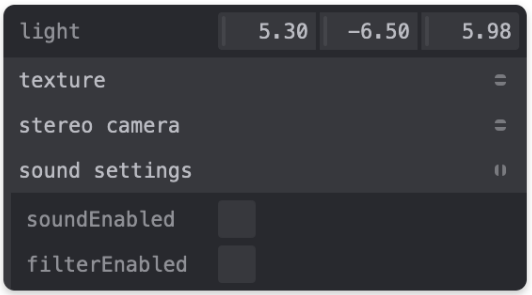
1. Підключення API Magnetometer за допомогою service worker:
    - Створюємо файл service worker з підтримкою Magnetometer API.
    - Реєструємо service worker у головному скрипті застосунку.
    - Встановлюємо прослуховувач подій Magnetometer API для отримання даних про магнітне поле. При отриманні даних з магнетометра, оновлюємо позицію сфери.
2. Використання WebGL для відрендерення двох фігур в одному буфері:
    - Ініціалізуємо WebGL контекст на канвасі.
    - Створюємо та компілюємо шейдери (vertex shader і fragment shader) для рендерингу фігур.
    - Створюємо та налаштовуємо буфер вершин для фігур, включаючи координати вершин, колір і текстурні координати.
    - Завантажуємо фігури до буферу вершин.

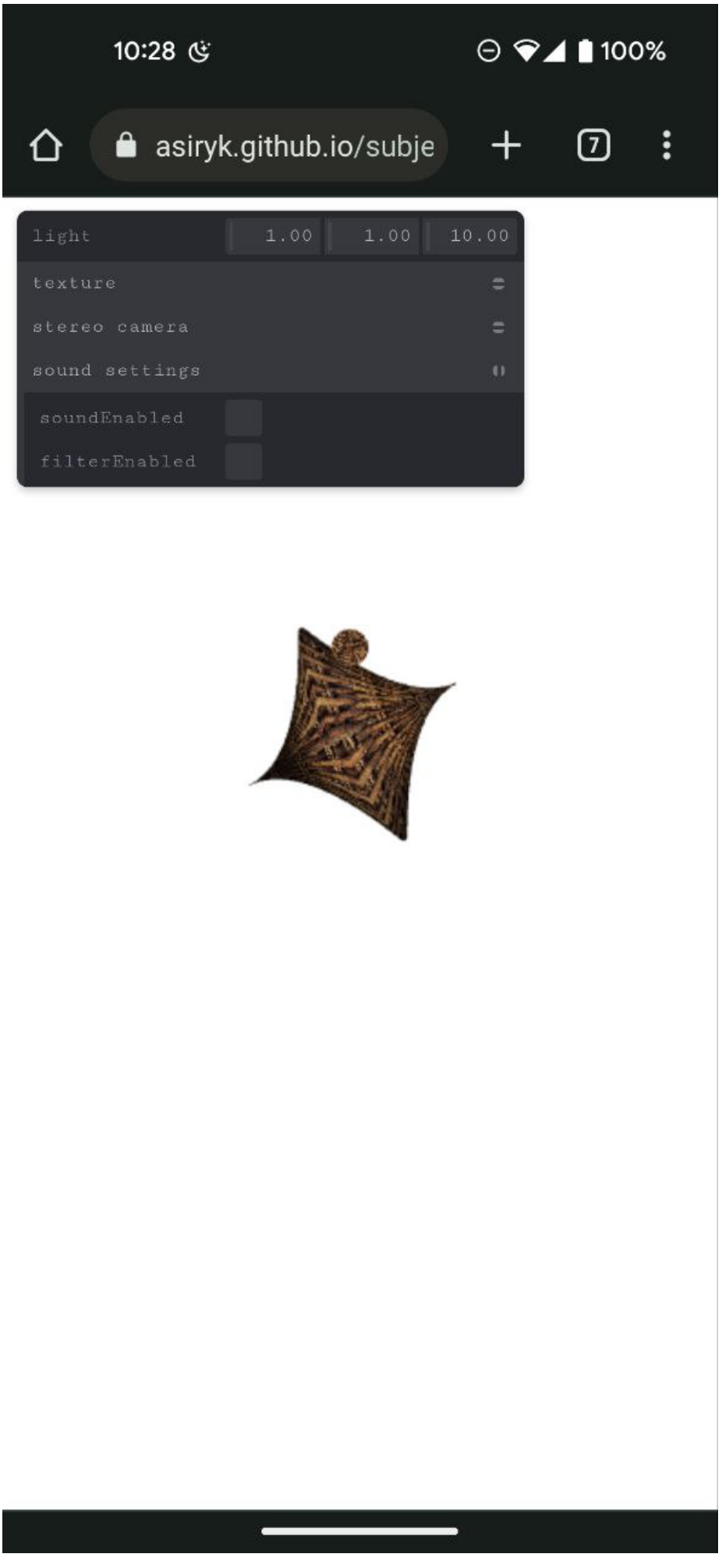
    - Виконуємо рендеринг фігур з використанням шейдерів та буфера вершин.
3. Завантаження аудіо доріжки до Web Audio Context та зміна позиції звуку:
    - Створюємо Web Audio Context.
    - Завантажуємо аудіо доріжку у веб-застосунок.
    - Створюємо AudioBufferSourceNode для відтворення аудіо доріжки.
    - Налаштовуємо параметри звуку, такі як гучність, панорамування та швидкість відтворення.
    - Збираємо дані з магнетометра та використовуємо їх для зміни позиції звуку у просторі. Наприклад, можна використовувати координати сфери для визначення панорамування звуку.
4. Створення lowpass фільтра:
    - Створюємо AudioContext.destination для виведення звуку.
    - Створюємо BiquadFilterNode з типом "lowpass".
    - Встановлюємо значення параметрів фільтра, таких як частота зрізу і підсилення.
    - Підключаємо AudioBufferSourceNode до BiquadFilterNode і BiquadFilterNode до AudioContext.destination.

Виконавши ці кроки, отримаємо програму що відповідає всім вимогам. Переконатися в цьому можна поглянувши на результати виконання у наступному розділі:

Результати виконання

На рисунках зображено фігуру зі сферою що обертається за допомогою сенсора магнетометра і відображає поточне положення звуку відносно фігури.




10:28
100%
asiryk.github.io/subje
7
light
1.00
1.00
10.00
texture
stereo camera
sound settings
soundEnabled
filterEnabled

Вихідний код

```
async function initAudio() {
  const audioContext = new AudioContext();
  const decodedAudioData = await fetch("/subject_ms-VR/sound.mp3")
      .then(response => response.arrayBuffer())
      .then(audioData => audioContext.decodeAudioData(audioData));
  const source = audioContext.createBufferSource();
  source.buffer = decodedAudioData;
  source.connect(audioContext.destination);
  source.start();

  while(audioContext.state === "suspended") {
      await new Promise(resolve => window.setTimeout(resolve, 2000));
      audioContext.resume();
  }
  audioContext.suspend(); // suspend to enable with tweakpane

  const panner = audioContext.createPanner();
  const volume = audioContext.createGain();
  volume.connect(panner);

  // create low-pass filter, but don't enable it
  const lowpass = audioContext.createBiquadFilter();
  lowpass.type = "lowpass";
  lowpass.frequency.value = 15000; // Hz filter

  audio.panner = panner;
  audio.context = audioContext;
  audio.filter = lowpass;
  audio.source = source;
  source.connect(lowpass);

  window.setAudioPosition = (x: number, y: number, z: number) => {
      panner.positionX.value = x;
      panner.positionY.value = y;
      panner.positionZ.value = z;
```

```
  }
}

export function init(attachRoot: HTMLElement) {
  try {
      const size = Math.min(600, window.innerWidth - 50);
      const { gl, canvas } = initCanvas(size, size);

      const program = new Program(gl, vertex, fragment);
      const { vertices: surface, uvs } = createVertices();
      const {vertices: sphere, uvs: uvsSphere } = createSphere();
      program.initBuffer(Attributes.Vertices, [surface, sphere],
Attributes.Uvs, uvs.concat(uvsSphere));

      const eyeSeparation = 0.004;
      const convergence = 1;
      const fov = radians(15);
      const near = 0.001;
      const far = 30;
      const pane = initTweakpane();
      const camera = new Camera(eyeSeparation, convergence, fov, near, far);

      const rotator = new TrackballRotator(canvas, null, 0);
      const dummyRotation = new Matrix4().identity();

      rotator.setCallback(() => draw(gl, program, rotator, camera,
dummyRotation));
      draw(gl, program, rotator, camera, dummyRotation);

      program.setUniform(Uniforms.TextureScale, new Vector2(1, 1));
      program.setUniform(Uniforms.LightPosition, new Vector3(1, 1, 10));
      program.setUniform(Uniforms.TextureRotAxis, new Vector2(0, 0));
      program.setUniform(Uniforms.TextureRotAngleDeg, 0);
      draw(gl, program, rotator, camera, dummyRotation);
```