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

Розрахункова графічна робота з дисципліни
“Методи синтезу віртуальної реальності”
Варіант №17

Виконав:
студент 5-го курсу, ІАТЕ
групи ТР-31мп
Міщенко А. А.
Перевірив:
Демчишин А. А.

Київ – 2024

Завдання

1. Використовуючи програмний код з другої лабораторної роботи, додати аудіо.
2. Реалізувати обертання джерела звуку навколо геометричного центру ділянки поверхні за допомогою дотичного інтерфейсу (цього разу поверхня залишається нерухомою, а джерело звуку рухається). Відтворити улюблену пісню у форматі mp3/ogg, при цьому положення джерела звуку керується користувачем.
3. Візуалізувати положення джерела звуку за допомогою сфери.
4. Додати звуковий фільтр (використовуйте інтерфейс BiquadFilterNode) для кожного варіанту. Додати елемент checkbox, який вмикає або вимикає фільтр. Налаштуйте параметри фільтра на свій смак.
5. Розроблений програмний код завантажити на віддалений GitHub репозиторій у гілку CGW.

Теорія

WebGL (Web Graphics Library) - це JavaScript API для рендерингу високопродуктивної інтерактивної 3D і 2D графіки в будь-якому сумісному веб-браузері без використання плагінів. WebGL робить це, представляючи API, який тісно пов'язаний з OpenGL ES 2.0, що може бути використаний в елементах HTML <canvas>. Ця відповідність дозволяє API використовувати переваги апаратного прискорення графіки, що надається пристроєм користувача.

Web Audio API - це високорівневий JavaScript API для обробки та синтезу аудіо у веб-застосунках. Він розроблений для використання разом з іншими API на веб-платформі. Деякі з його функцій включають модульну маршрутизацію, високий динамічний діапазон і точне за часом відтворення запланованого звуку.

Web Audio API складається з кількох основних компонентів:

- AudioContext: Це об'єкт, який представляє аудіосистему браузера. Він використовується для створення інших компонентів Web Audio API.
- AudioNode: Це базовий клас для всіх компонентів обробки аудіо. До нього належать такі об'єкти, як AudioBufferSourceNode, AudioGainNode та AudioFilterNode.
- AudioBuffer: Це об'єкт, який представляє собою дані аудіо. Він використовується для створення об'єктів AudioBufferSourceNode.
- AudioWorklet: Це тип JavaScript-коду, який може використовуватися для створення власних компонентів обробки аудіо.

BiquadFilterNode - це тип AudioNode, який використовується для загальних фільтрів низького порядку, таких як:

- Низькочастотний (low pass)
- Високочастотний (high pass)
- Смуговий (band pass)
- Низькочастотна полиця (low shelf)

- Високочастотна полиця (high shelf)
- Піковий (peaking)
- Зарубковий (notch)
- Всепрохідний (allpass)

BiquadFilterNode має ряд параметрів, які можна використовувати для керування його поведінкою, включаючи:

- frequency: Частота, на якій фільтр має максимальний ефект.
- Q: Добротність фільтра. Це визначає ширину смуги пропускання фільтра.
- gain: Посилення фільтра. Це визначає, скільки посилення застосовується до частот, які проходять через фільтр.
- type: Тип фільтра. Це визначає форму частотної характеристики фільтра.

BiquadFilterNode можна використовувати для створення широкого кола ефектів обробки аудіо. Наприклад, його можна використовувати для створення еквалайзера, компресора або реверберatора.

BiquadFilterNode можна використовувати як смуговий фільтр (band pass). Смуговий фільтр пропускає діапазон частот, послаблюючи частоти нижче та вище зазначеного діапазону.

Наприклад, смуговий фільтр з центральною частотою 2000 Гц і шириною смуги пропускання 1000 Гц пропускатиме частоти від 1500 Гц до 2500 Гц, послаблюючи частоти нижче 1500 Гц і вище 2500 Гц.

Смугові фільтри часто використовуються для виділення певного діапазону частот з аудіосигналу. Наприклад, смуговий фільтр можна використовувати для виділення вокалу з музичного запису або для виділення певного інструменту з оркестрового запису.

Імплементація програмного коду

Для початку завантажимо аудіо з яким будемо працювати в розрахунковій роботі за допомогою вище описаного Web Audio API. Аудіо, яке буде програватися, буде завантажено за допомогою XMLHttpRequest класу з віддаленого GitHub репозиторію. А використовуючи Web Audio API буде створено аудіо контекст, фільтр та панер, які будуть підключені одні до одного, щоб відтворювати звук згідно заданих налаштувань.

```
   function createAudio() {
        audioContext = new window.AudioContext();
        audioSource = audioContext.createBufferSource();
        createBandpassFilter();
        createAudioPanner();
        const request = new XMLHttpRequest();
        request.open("GET",
"https://raw.githubusercontent.com/twistedmisted/surf-rev-pear-vr/CGW/audio.mp3", true);
        request.responseType = "arraybuffer";
        request.onload = () => {
        const audioData = request.response;
        audioContext.decodeAudioData(audioData, (buffer) => {
                audioSource.buffer = buffer;
                if (useFilter) {
                        audioSource.connect(audioFilter);
                        audioFilter.connect(audioPanner);
                } else {
                        audioSource.connect(audioPanner);
                }
                audioPanner.connect(audioContext.destination);
                audioSource.loop = true;
                }, (err) => {alert(err)}
        );
        };
        request.send();
   }
```

Згідно варіанту необхідно було реалізувати смуговий фільтр (band pass). Для цього необхідно створити BiquadFilter використовуючи AudioContext та встановити цьому фільтру бажаний тип.

```
   function createBandpassFilter() {
        audioFilter = audioContext.createBiquadFilter();
        audioFilter.type = "bandpass";
        audioFilter.frequency.value = 1000;
        audioFilter.Q.value = 1;
   }
```

Також необхідно створити Panner, його буде використано для можливості керування аудіо та розміщення його в просторі за допомогою зміни положення телефону в просторі.

```
   function createAudioPanner() {
        audioPanner = audioContext.createPanner();
        audioPanner.panningModel = "HRTF";
        audioPanner.distanceModel = "inverse";
        audioPanner.refDistance = 1;
        audioPanner.maxDistance = 1000;
```

```
    audioPanner.rolloffFactor = 1;
    audioPanner.coneInnerAngle = 360;
    audioPanner.coneOuterAngle = 0;
    audioPanner.coneOuterGain = 0;
}
```

Для можливості вимикання та вмикання фільтра під час програвання звуку необхідно додати checkbox та приєднати/від'єднати фільтр відповідно до встановленого значення в цьому полі.

```
function setupUseFilterEvent() {
    const checkbox = document.getElementById('useFilter');
    checkbox.addEventListener('change', (event) => {
    if (event.target.checked) {
            useFilter = true;
            if (audioContext) {
            audioSource.disconnect();
            audioPanner.disconnect();
            audioSource.connect(audioFilter);
            audioFilter.connect(audioPanner);
            audioFilter.connect(audioContext.destination);
            }
    } else {
            useFilter = false;
            if (audioContext) {
            audioSource.disconnect();
            audioPanner.disconnect();
            audioSource.connect(audioPanner);
            audioPanner.connect(audioContext.destination);
            }
    }
    });
}
```

Для можливості змінювати звук в просторі було використано EventListener на веб-сторінці, який буде обробляти прискорення телефону і відповідно до цього змінювати положення аудіо в просторі.

```
window.addEventListener('devicemotion', (event) => {
    if(audioPanner) {
            audioPosition.x += deg2rad(event.acceleration.x);
            audioPosition.y += deg2rad(event.acceleration.y);
            audioPosition.z += deg2rad(event.acceleration.z);

            sphereRotation.x = 2 * Math.cos(audioPosition.y) * Math.cos(audioPosition.x);
            sphereRotation.y = 2 * Math.sin(audioPosition.y);
            sphereRotation.z = 2 * Math.cos(audioPosition.y) * Math.sin(audioPosition.z);

            audioPanner.setPosition(sphereRotation.x, sphereRotation.y, sphereRotation.z);
            audioPanner.setOrientation(0,0,0);

            redraw();
    }
});
```

Для відображення позиції звука в просторі було додано сферу, яка змінює своє положення разом із звуком.

```
function CreateSphereData()
{
    let radius = 1.2;
    let vertexList = [];
    const stepU = 10;
    const maxDegree = 360;
```

```
        for (let u = 0; u <= maxDegree; u += stepU) {
        for(let v = 0; v <= maxDegree; v += stepU) {
                let tempA = deg2rad(u);
                let tempB = deg2rad(v);
                let tempA2 = deg2rad(u + stepU);
                let tempB2 = deg2rad(v + stepU);
                vertexList.push(sphereRotation.x + (radius *   Math.cos(tempA) * Math.sin(tempB)),
sphereRotation.y + (radius *    Math.sin(tempA) * Math.sin(tempB)), sphereRotation.z + (radius *
Math.cos(tempB)));
                vertexList.push(sphereRotation.x + (radius *   Math.cos(tempA2) * Math.sin(tempB2)),
sphereRotation.y + (radius *   Math.sin(tempA2) * Math.sin(tempB2)), sphereRotation.z + (radius *
Math.cos(tempB2)));
        }
        }
        return vertexList;
   }

   this.DrawSphere = 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.TRIANGLE_STRIP, 0, this.verticesLength);
   };
```

Приклади роботи з програмою

Приклад фігури з негативним паралаксом, вебкамери з нульовим паралаксом, сфери, яка позначає позицію звука в просторі та інтерфейсу користувача (рис. 1).

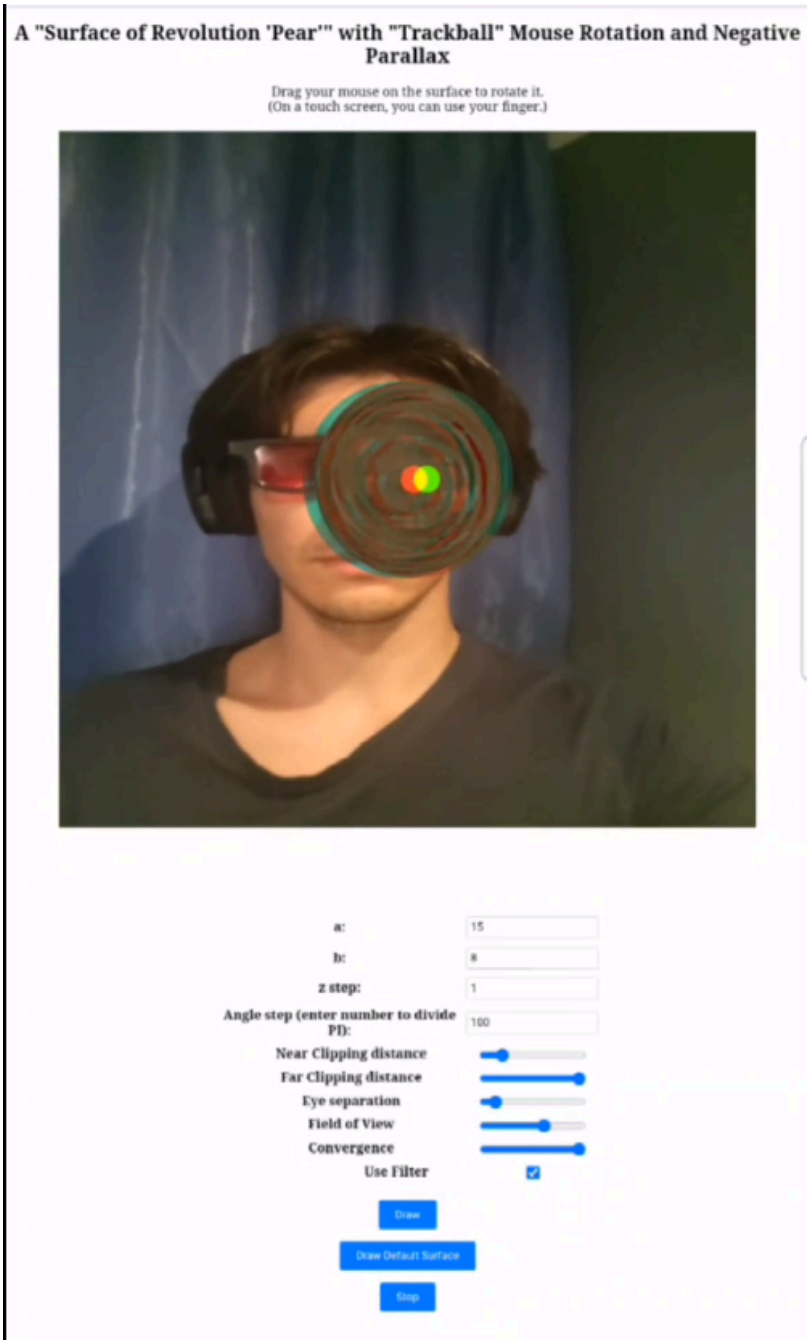

Рис. 1. – Фігура з початковими параметрами

На рисунку 2 зображено зміну позиції телефона в просторі, відповідно і звуку та сфери.

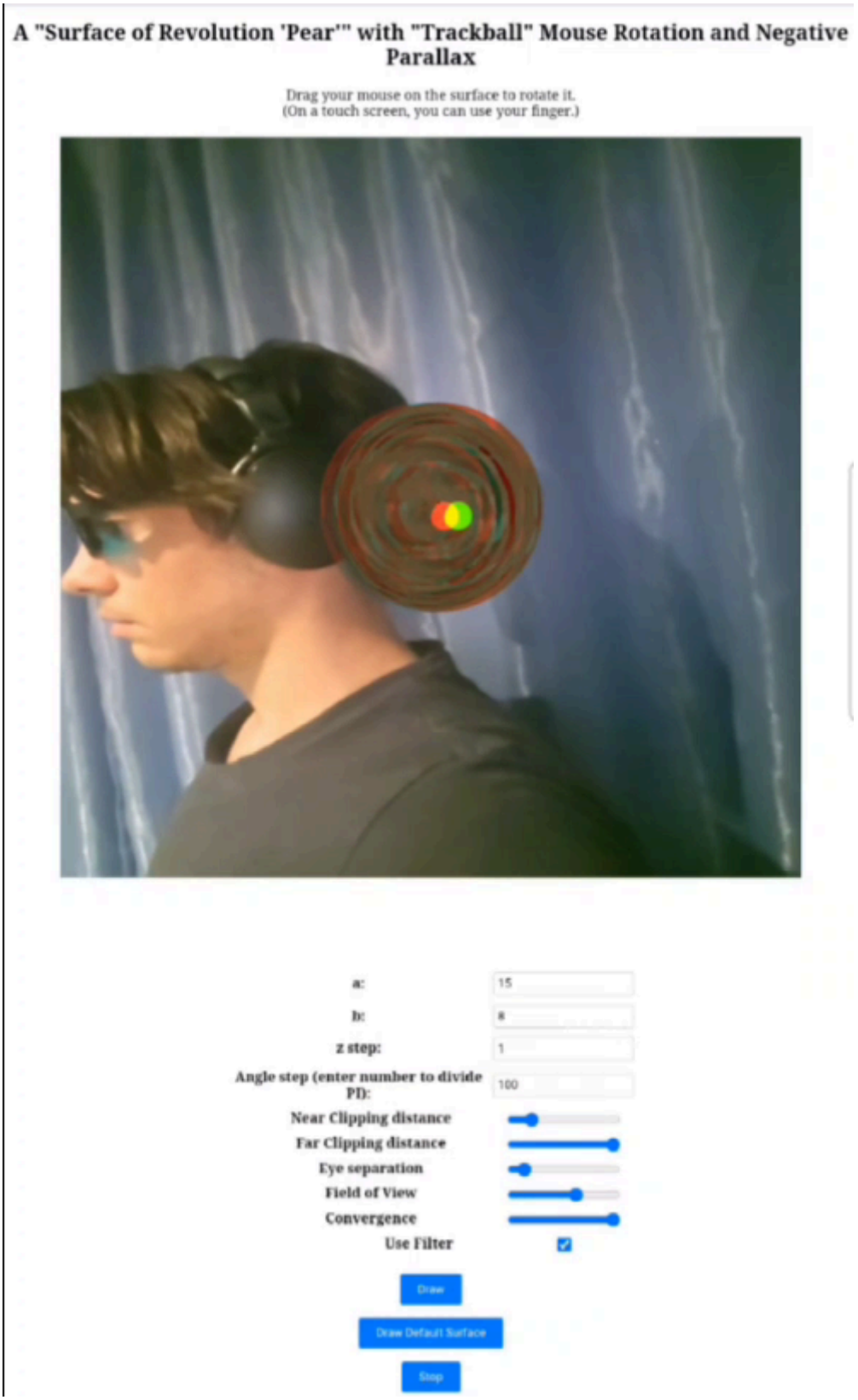

Рис. 2. – Зміна положення звуку в просторі.

На рисунку 3 зображено вимкнутий фільтр для аудіо.

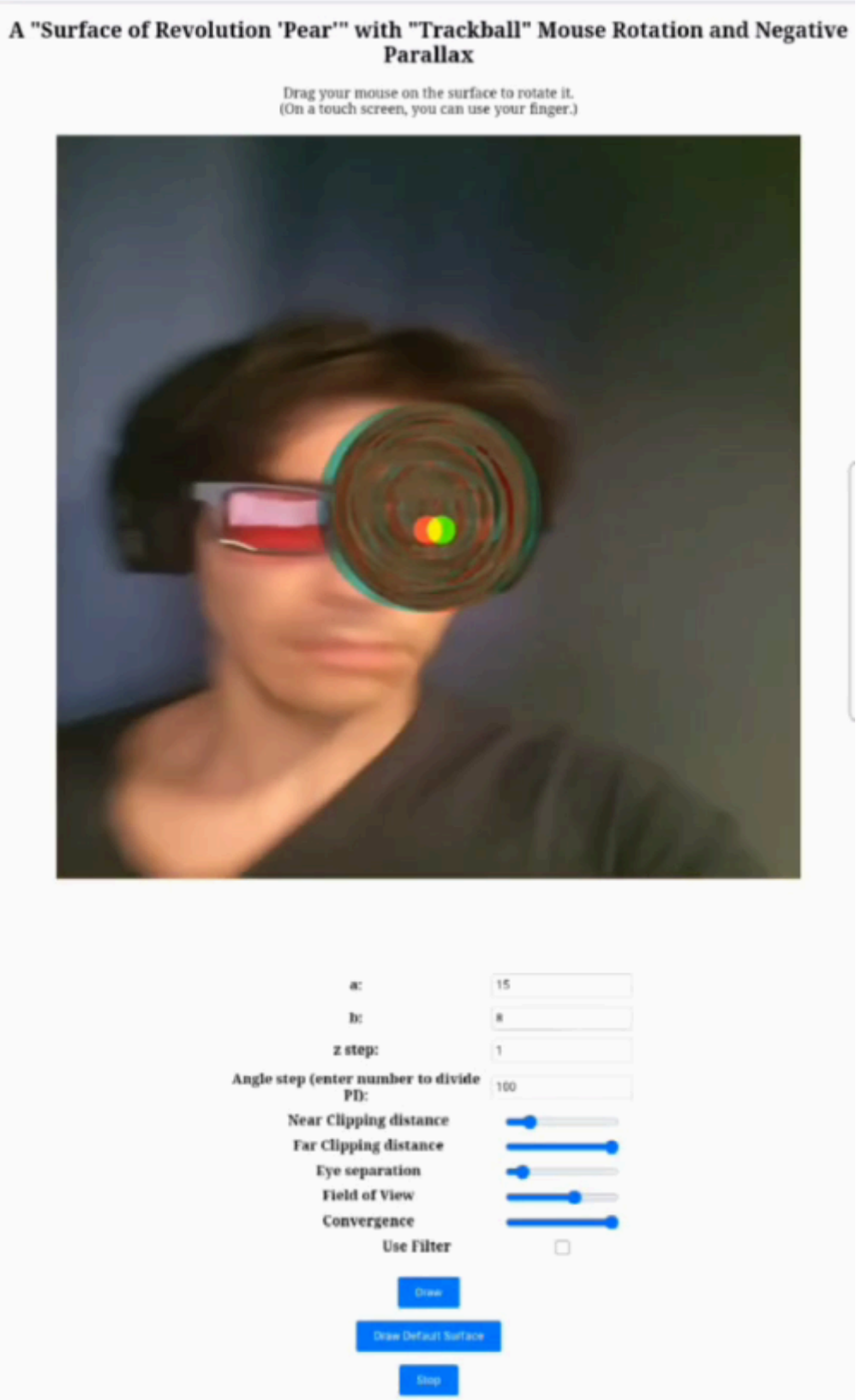

Рис. 3. – Вимкнутий аудіо фільтр

Приклад вихідного коду

```
    let audioContext;               // An audio context
    let audioSource;                // Holds audio settings
    let audioPanner;                // An audio panner
    let audioFilter;                // An audio bandpass filter
    let defaultFrequency;           // A default frequency of sound filter
    let audioPosition;              // Audio position in space
    let useFilter = true;          // Indicates if use filter for audio

    let sphere;                        // A sphere model to vizualize position of the sound in the
space
    let sphereRotation;             // Sphere rotation point
    function initParameters() {
        sphereRotation = new Point(0, 0, 0);
        audioPosition = new Point(1, 0, 0);

        parameters = {
            a: 15,
            b: 8,
            zStep: 1,
            angleStep: 100,
            nearClippingDistance: 8,
            farClippingDistance: 20000,
            eyeSeparation: 0.45,
            FOV: Math.PI * 3,
            convergence: 350,
            audioPlay: false
        };

        for (let key in parameters) {
            let element = document.getElementById(key);
            if (element) {
                element.value = parameters[key];
            }
        }
    }
    /**
     * Creates data for audio sphere
     */
    function CreateSphereData()
    {
        let radius = 1.2;
        let vertexList = [];
        const stepU = 10;
        const maxDegree = 360;
        for (let u = 0; u <= maxDegree; u += stepU) {
            for(let v = 0; v <= maxDegree; v += stepU) {
                let tempA = deg2rad(u);
                let tempB = deg2rad(v);
                let tempA2 = deg2rad(u + stepU);
```

```
            let tempB2 = deg2rad(v + stepU);
             vertexList.push(sphereRotation.x + (radius *  Math.cos(tempA) * Math.sin(tempB)), sphereRotation.y + (radius *  Math.sin(tempA) * Math.sin(tempB)), sphereRotation.z + (radius * Math.cos(tempB)));
                        vertexList.push(sphereRotation.x + (radius *  Math.cos(tempA2) * Math.sin(tempB2)), sphereRotation.y + (radius *  Math.sin(tempA2) * Math.sin(tempB2)), sphereRotation.z + (radius *  Math.cos(tempB2)));
          }
      }
      return vertexList;
  }

  function deg2rad(angle) {
      return angle * Math.PI / 180;
  }


  /**
   * initialization function that will be called when the page has loaded
   */
  function init() {
      //...
      window.addEventListener('devicemotion', (event) => {
          if(audioPanner) {
              audioPosition.x += deg2rad(event.acceleration.x);
              audioPosition.y += deg2rad(event.acceleration.y);
              audioPosition.z += deg2rad(event.acceleration.z);

              sphereRotation.x = 2 * Math.cos(audioPosition.y) * Math.cos(audioPosition.x);
              sphereRotation.y = 2 * Math.sin(audioPosition.y);
              sphereRotation.z = 2 * Math.cos(audioPosition.y) * Math.sin(audioPosition.z);

              audioPanner.setPosition(sphereRotation.x, sphereRotation.y, sphereRotation.z);
              audioPanner.setOrientation(0,0,0);

              redraw();
          }
      });

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

      setupUseFilterEvent();
      updateSurfaces();
  }
  function setupUseFilterEvent() {
      const checkbox = document.getElementById('useFilter');
      checkbox.addEventListener('change', (event) => {
          if (event.target.checked) {
              useFilter = true;
              if (audioContext) {
                  audioSource.disconnect();
```

```
                audioPanner.disconnect();
                audioSource.connect(audioFilter);
                audioFilter.connect(audioPanner);
                audioFilter.connect(audioContext.destination);
            }
        } else {
            useFilter = false;
            if (audioContext) {
                audioSource.disconnect();
                audioPanner.disconnect();
                audioSource.connect(audioPanner);
                audioPanner.connect(audioContext.destination);
            }
        }
    });
}
/**
 * Plays audio from .mp3 file on HTML page
 */
function playMusic() {
    if (parameters.audioPlay) {
        audioContext.suspend();
        document.getElementById('play-audio-btn').textContent = 'Resume';
    } else {
        if (audioContext) {
            audioContext.resume();
        } else {
            createAudio();
            audioSource.start(0);
        }
        document.getElementById('play-audio-btn').textContent = 'Stop';
    }
    parameters.audioPlay = !parameters.audioPlay;
}

/**
 * Creates audio from .mp3 file to play on HTML page
 */
function createAudio() {
    audioContext = new window.AudioContext();
    audioSource = audioContext.createBufferSource();
    createBandpassFilter();
    createAudioPanner();
    const request = new XMLHttpRequest();
                                                                request.open("GET",
"https://raw.githubusercontent.com/twistedmisted/surf-rev-pear-vr/CGW/audio.mp3", true);
    request.responseType = "arraybuffer";
    request.onload = () => {
        const audioData = request.response;
        audioContext.decodeAudioData(audioData, (buffer) => {
                audioSource.buffer = buffer;
```

```
                if (useFilter) {
                    audioSource.connect(audioFilter);
                    audioFilter.connect(audioPanner);
                } else {
                    audioSource.connect(audioPanner);
                }
                audioPanner.connect(audioContext.destination);
                audioSource.loop = true;
            }, (err) => {alert(err)}
        );
    };
    request.send();
}

/**
 * Sets up bandpass filter for audio
 */
function createBandpassFilter() {
    audioFilter = audioContext.createBiquadFilter();
    audioFilter.type = "bandpass";
    audioFilter.frequency.value = 1000;
    audioFilter.Q.value = 1;
}

/**
 * Creates audio panner
 */
function createAudioPanner() {
    audioPanner = audioContext.createPanner();
    audioPanner.panningModel = "HRTF";
    audioPanner.distanceModel = "inverse";
    audioPanner.refDistance = 1;
    audioPanner.maxDistance = 1000;
    audioPanner.rolloffFactor = 1;
    audioPanner.coneInnerAngle = 360;
    audioPanner.coneOuterAngle = 0;
    audioPanner.coneOuterGain = 0;
}
```

Фрагментний шейдер:

```
// Vertex shader
const vertexShaderSource = `
attribute vec3 vVertex;
attribute vec2 texCoord;
uniform mat4 ModelViewMatrix, ProjectionMatrix;

uniform bool isCamera;
uniform bool isSphere;
```

```
varying vec2 texInterp;

void main() {
  if (isCamera) {
     vec4 vertPos4 = ProjectionMatrix * vec4(vVertex, 1.0);
     gl_Position = vertPos4;
  } else {
     vec4 vertPos4 = ModelViewMatrix * vec4(vVertex, 1.0);
     gl_Position = ProjectionMatrix*vertPos4;
  }

  texInterp = texCoord;
}`;


// Fragment shader
const fragmentShaderSource = `
#ifdef GL_FRAGMENT_PRECISION_HIGH
   precision highp float;
#else
   precision mediump float;
#endif

varying vec2 texInterp;
uniform sampler2D tmu;
uniform float angleRad;

uniform bool isSphere;
uniform vec4 color;

void main() {
  if (isSphere) {
     gl_FragColor = color;
  } else {
     gl_FragColor = texture2D(tmu, texInterp);
  }
}`;
```