Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

з дисципліни “Методи синтезу віртуальної реальності”  
на тему “Spatial audio”

Варіант 24

Виконав студент групи ТР-31мп  
Руй Дмитро Ігорович

Київ 2024

1. **Завдання**

Імплементувати просторове аудіо за допомогою WebAudio HTML5 API, використовуючи код з практичного завдання 2.

Імплементувати обертання джерела звуку відносно геометричного центру поверхні з використанням дотичного інтерфейсу (смартфону, що використовує сенсори для обробки інформації, щодо положення в просторі). На відміну від попередніх робіт поверхня залишається нерухомою - рухається джерело звуку. Відтворити улюблену пісню в форматі mp3/ogg, з можливістю контролювати місцезнаходження джерела звуку в просторі користувачем.

Візуалізувати джерело звуку за допомогою сфери.

Додати звуковий фільтр за варіантом (використовуючи інтерфейс BiquadFilterNode). Додати інтерфейс користувача, який би вмикав/вимикав фільтр. Встановити параметри фільтру відповідно до вподобань.

Підготувати звіт в цифровому вигляді, який би містив необхідні частини, що сповна описують поставлені задачі та виконану роботу.

1. **Теоретичні відомості**

**WebAudio API**

Аудіо в Інтернеті до цього часу було досить примітивним і вимагало використання плагінів, таких як Flash і QuickTime, для відтворення. Введення аудіоелемента в HTML5 є значним кроком вперед, оскільки він дозволяє відтворювати базове потокове аудіо. Однак цей елемент не досить потужний для роботи зі складнішими аудіододатками. Для складних веб-ігор або інтерактивних програм необхідне інше рішення. Мета цієї специфікації полягає у впровадженні можливостей сучасних ігрових звукових движків, а також завдань мікшування, обробки та фільтрації, що використовуються у сучасних настільних програмах для створення звуку.

Інтерфейси API розроблені з урахуванням широкого спектру сценаріїв використання [webaudio-usecases]. Ідеально, API має підтримувати будь-який варіант використання, який можна реалізувати за допомогою оптимізованого C++ механізму, що керується через сценарій і виконується в браузері. Однак сучасне настільне аудіопрограмне забезпечення може мати дуже розширені можливості, деякі з яких важко або неможливо реалізувати за допомогою цієї системи. Наприклад, Apple Logic Audio підтримує зовнішні MIDI-контролери, довільні плагіни аудіоефектів і синтезаторів, оптимізоване читання/запис аудіофайлів безпосередньо на диск, тісно інтегроване розтягування часу тощо. Проте запропонована система зможе підтримувати широкий спектр досить складних ігор та інтерактивних програм, включаючи музичні, і може бути чудовим доповненням до розширених графічних функцій, що пропонує WebGL. API розроблено таким чином, щоб у майбутньому можна було додати додаткові можливості.

**BiquadFilterNode**

BiquadFilterNode — це процесор AudioNode, який реалізує дуже поширені фільтри нижчого порядку.

Фільтри нижчого порядку є будівельними блоками базових регуляторів тембру (баси, середні та високі частоти), графічних еквалайзерів і більш розширених фільтрів. Кілька фільтрів BiquadFilterNode можна комбінувати для створення більш складних фільтрів. Параметри фільтра, такі як частота, можна змінювати з часом для розгортки фільтра тощо. Кожен BiquadFilterNode можна налаштувати як один із кількох загальних типів фільтрів, як показано в IDL нижче. Тип фільтра за замовчуванням – «НЧ».

Фільтр низьких частот пропускає частоти нижче граничної частоти та послаблює частоти вище граничної. Він реалізує стандартний резонансний фільтр низьких частот другого порядку зі спадом 12 дБ/октаву.

Фільтр високих частот є протилежністю фільтру низьких частот. Частоти вище граничної частоти пропускаються, але частоти нижче граничної послаблюються. Він реалізує стандартний резонансний фільтр високих частот другого порядку зі спадом 12 дБ/октаву.

Смуговий фільтр пропускає діапазон частот і послаблює частоти нижче та вище цього діапазону частот. Він реалізує смуговий фільтр другого порядку.

Фільтр Lowshelf пропускає всі частоти, але додає підсилення (або ослаблення) нижніх частот. Він реалізує фільтр низького рівня другого порядку.

Фільтр Highshelf є протилежністю фільтру Lowshelf і пропускає всі частоти, але додає посилення до вищих частот. Він реалізує фільтр високої полиці другого порядку

Піковий фільтр пропускає всі частоти, але додає підсилення (або ослаблення) до діапазону частот.

Режекторний фільтр (також відомий як смуговий або смуговий фільтр) є протилежністю смуговому фільтру. Він дозволяє пропускати всі частоти, крім набору частот.

1. **Аспекти імплементації**

Вдалося створити обертання джерела звуку навколо геометричного центру ділянки поверхні протягом певного часу (у даному випадку джерело звуку рухається, а поверхня - нерухома). Джерело звуку створено у вигляді сферичної геометрії.

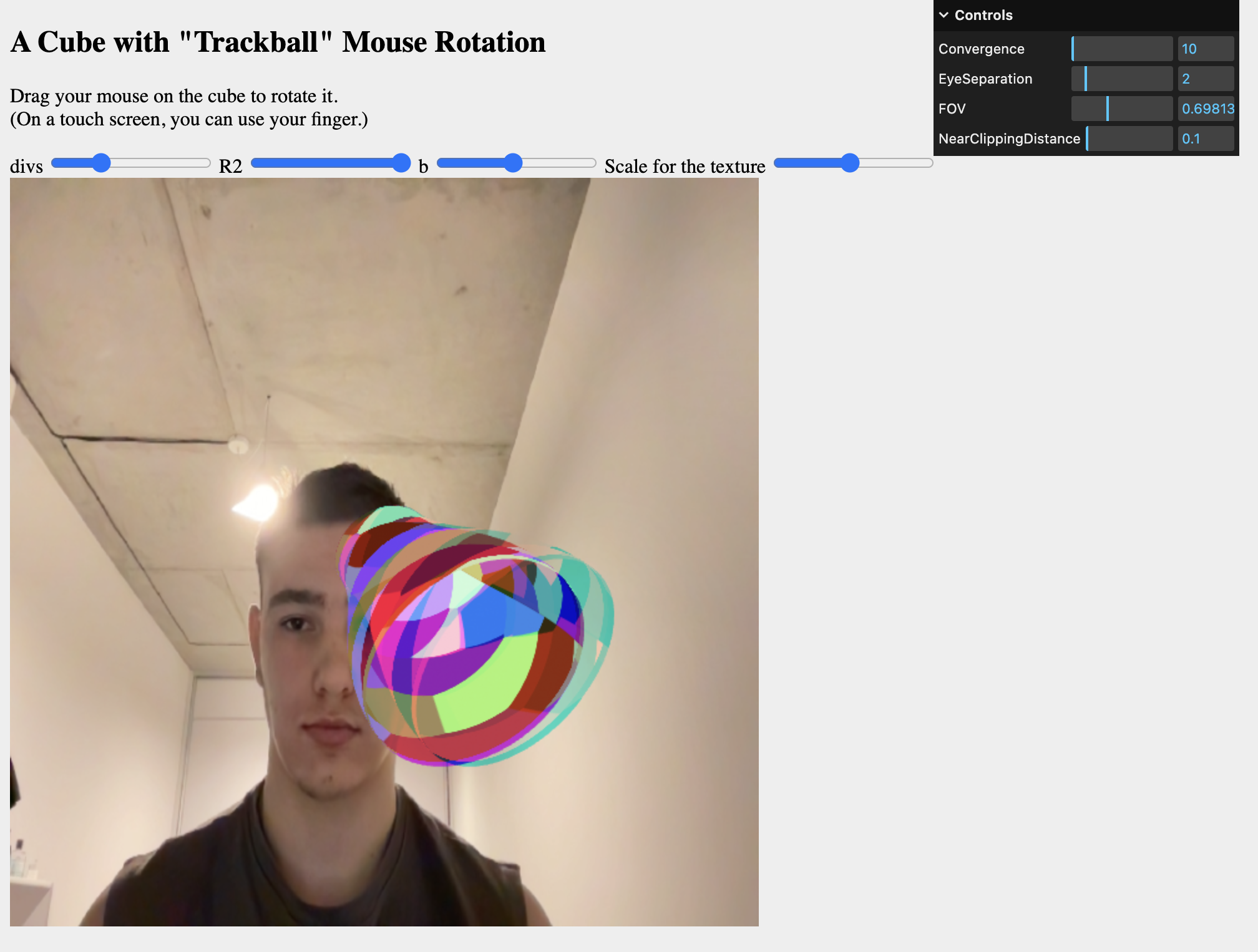


Рисунок 3.1 - 2 ЛР

В подальшому цей функціонал дозволить керувати положенням джерела звуку просторового аудіо всередині програми.

Першим кроком для імплементації просторового аудіо є створення HTML-елементу <audio>, який містить інформацію про джерело аудіо-доріжки в моєму випадку це [heavy-rain-nature-sounds-8186](https://github.com/dimasambo/computer-graphics-2/blob/CGW/heavy-rain-nature-sounds-8186.mp3) в форматі mp3. Цей елемент керування дозволить зупиняти та продовжувати відтворення аудіо-доріжки. Після цього за допомогою JavaScript, а саме WebAudio API було створено обʼєкт аудіоконтексту (AudioContext), для якого було створено та підʼєднано 3 основних обʼєкти:

* джерело звуку (MediaElementSource)
* обʼєкт обробки просторового аудіо (Panner)
* звуковий фільтр (BiquadFilter)

Згідно з варіантом було обрано смуговий фільтр.

Наступним кроком було встановлено параметри обраного фільтра. Взагалі, BiquadFilter має 3 параметри для налаштування:

* frequency (частота)
* Q (ширина смуги)
* gain (підсилення)

Особливість смугового фільтру в тому, що:

* frequency - центр смуги частот
* Q - контролює ширину смуги. Ширина стає вужчою зі збільшенням значення Q.
* gain - не використовується в цьому типі фільтра

Джерело звуку, обробка якого в просторі здійснюється обʼєктом класу Panner зображено у WebGL контексті у вигляді сфери, щоб може переміщуватись - відповідно змінюючи параметри обʼєкту Panner, щоб при прослуховуванні аудіо зі стерео звуком був ефект переміщення джерела звуку відповідно до місцезнаходження сфери в системі координат.

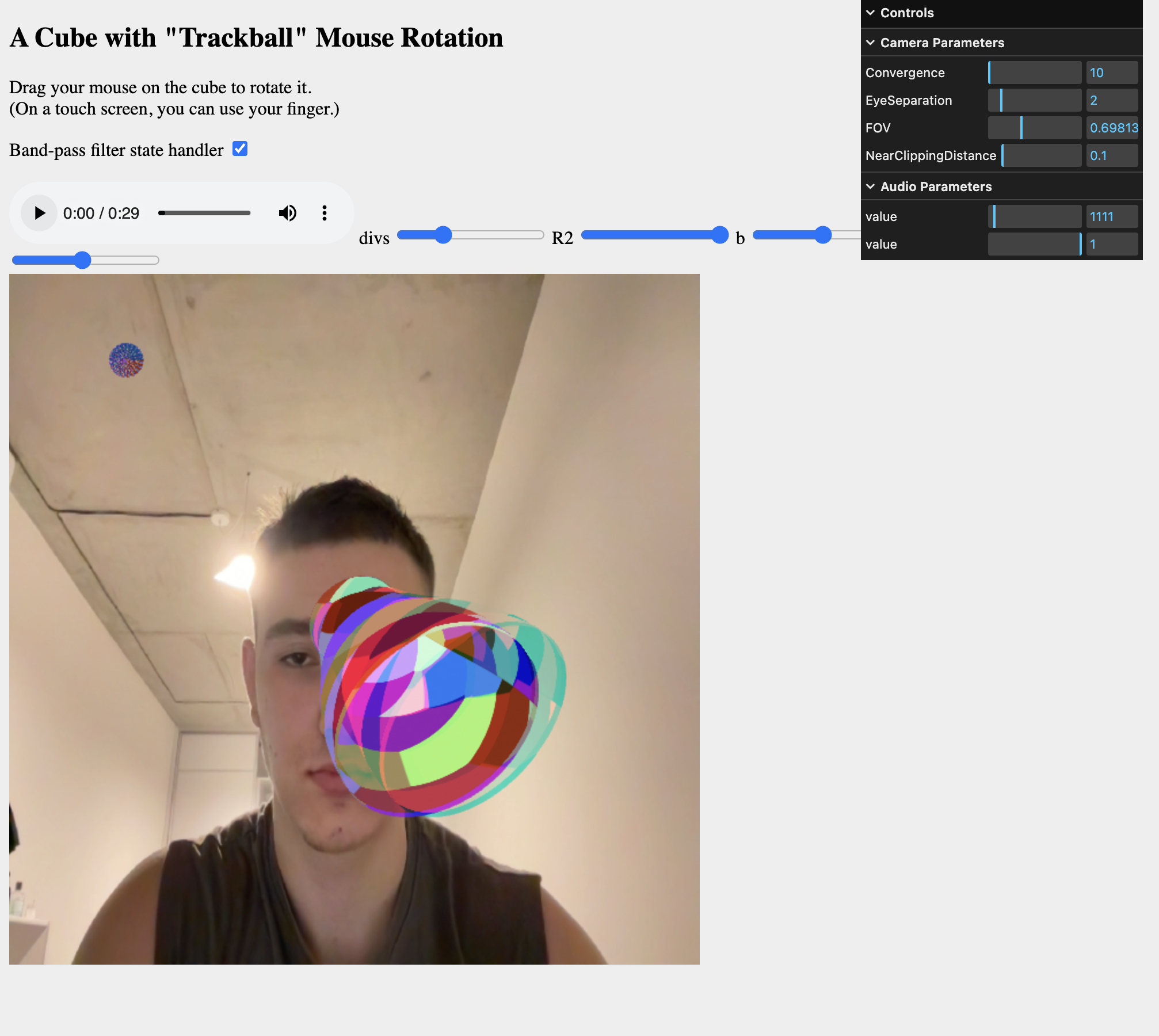


Рисунок 3.2 - РГР

1. **Інструкція користувача**

Для налаштування стерео зображення використовуються 4 слайдери, в який можна налаштувати наступні параметри:

* Convergence (збіжність)
* Eye separation (відстань між очима)
* Field of view (поле зору)
* Near clipping distance (відстань ближньої площини, що відсікає зображену на екрані геометрію)

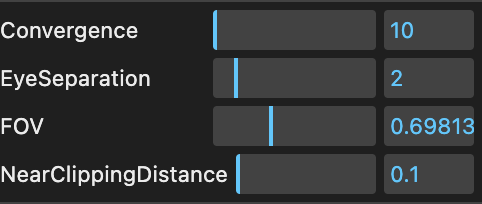


Рисунок 4.1 - слайдери

Згідно до встановлених параметрів стерео зображення фігури виглядає дещо інакше

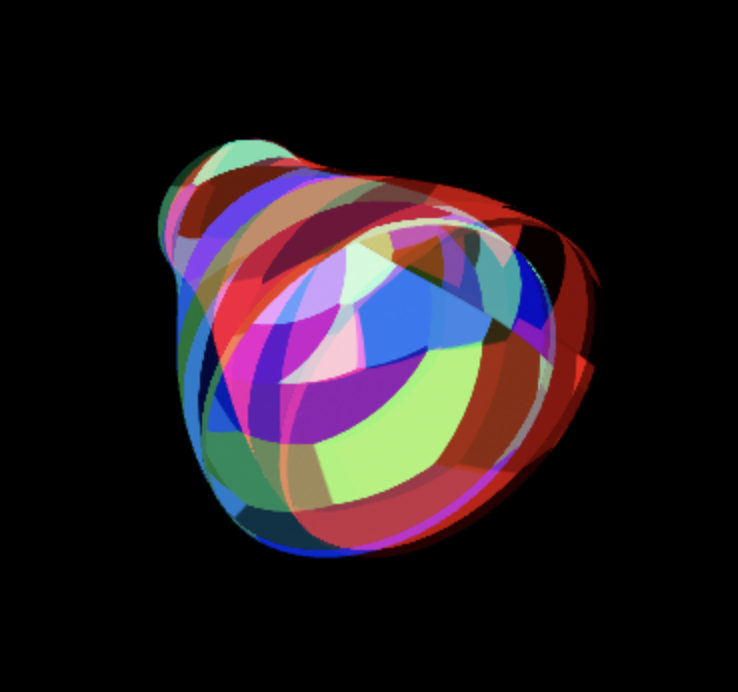


Рисунок 4.2 - фігура з початковими параметрами

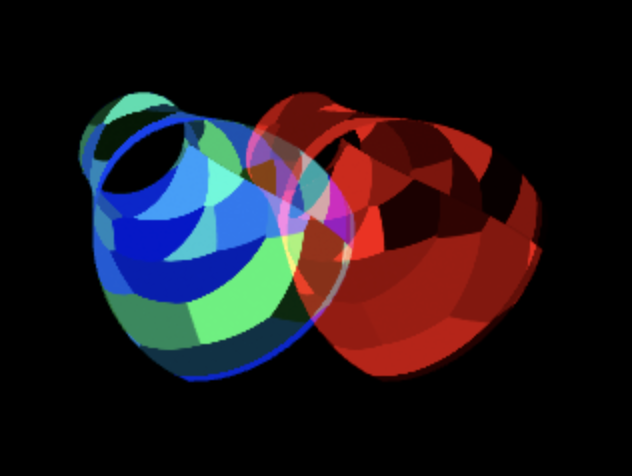


Рисунок 4.3 - фігура зі зміненими параметрами

Вище згаданий HTML-елемент audio дозволяє керувати аудіо-доріжкою, а саме зупиняти та продовжувати відтворення, перемотувати на потрібний час, керувати гучністю.

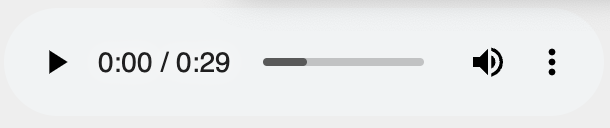


Рисунок 4.4 - UI-елемент плеєра

“Чекбокс” дозволяє керувати станом звукового фільтру: вмикати та вимикати його. В залежності від стану фільтру можна помітити зміну звучання.

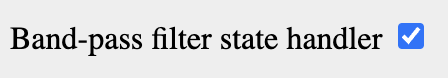


Рисунок 4.5 - чекбокс

1. **Код програми**

main.js:

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.

const { sin, cos, pow, *PI* } = *Math*

let *sphere*;

let *texCoord* = [0, 0]

let *cam*;

let *gui*;

let *bkg*;

let *wbc*;

let *track*;

let *texture*;

let *txr*;

function deg2rad(angle) {

return angle \* *Math*.PI / 180;

}

// Constructor

function Model(name) {

this.name = name;

this.iVertexBuffer = *gl*.createBuffer();

this.iNormalBuffer = *gl*.createBuffer();

this.iTextureBuffer = *gl*.createBuffer();

this.count = 0;

this.BufferData = function (vertices, normals, textures) {

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

*gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(vertices), *gl*.*STREAM\_DRAW*);

*gl*.bindBuffer(*gl*.*ARRAY\_BUFFER*, this.iNormalBuffer);

*gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(normals), *gl*.*STREAM\_DRAW*);

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

*gl*.bufferData(*gl*.*ARRAY\_BUFFER*, new *Float32Array*(textures), *gl*.*STREAM\_DRAW*);

this.count = vertices.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.iNormalBuffer);

*gl*.vertexAttribPointer(*shProgram*.iAttribNormal, 3, *gl*.*FLOAT*, false, 0, 0);

*gl*.enableVertexAttribArray(*shProgram*.iAttribNormal);

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

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

*gl*.enableVertexAttribArray(*shProgram*.iAttribTexture);

*gl*.drawArrays(*gl*.*TRIANGLES*, 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;

// 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.Use = function () {

*gl*.useProgram(this.prog);

}

}

function changed() {

*surface*.BufferData(...CreateSurfaceData());

draw()

}

/\* 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 matAccum0 = m4.multiply(rotateToPointZero, modelView);

let matAccum1 = m4.multiply(translateToPointZero, matAccum0);

/\* 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, matAccum1);

const normalMatrix = m4.identity();

m4.inverse(modelView, normalMatrix);

m4.transpose(normalMatrix, normalMatrix);

*gl*.uniformMatrix4fv(*shProgram*.iModelViewProjectionMatrix, false, modelViewProjection);

*gl*.uniformMatrix4fv(*shProgram*.iNormalMatrix, false, normalMatrix);

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

*gl*.uniform4fv(*shProgram*.iColor, [...hexToRgb(*document*.getElementById('c').value), 1]);

*gl*.uniform3fv(*shProgram*.iLightPos, [1 \* cos(Date.now() \* 0.001), 2 \* sin(Date.now() \* 0.001), 0]);

*gl*.uniform2fv(*shProgram*.iTT, *texCoord*);

*gl*.uniform1f(*shProgram*.iScale, *document*.getElementById('s').value);

*gl*.uniformMatrix4fv(*shProgram*.iModelViewProjectionMatrix, false, m4.identity());

*gl*.bindTexture(*gl*.*TEXTURE\_2D*, *txr*);

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

*bkg*.Draw();

*gl*.clear(*gl*.*DEPTH\_BUFFER\_BIT*);

let pnrX = *Math*.sin(Date.now() \* 0.0005)

let pnrY = *Math*.cos(Date.now() \* 0.0005)

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

if (*pnr*) {

*pnr*.setPosition(pnrX, pnrY, 0);

}

*gl*.uniformMatrix4fv(*shProgram*.iModelViewProjectionMatrix, false, m4.translation(pnrX, pnrY, 0));

*sphere*.Draw();

*cam*.ApplyLeftFrustum();

modelViewProjection = m4.multiply(*cam*.projection, m4.multiply(*cam*.modelview, matAccum1));

*gl*.uniformMatrix4fv(*shProgram*.iModelViewProjectionMatrix, false, modelViewProjection);

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

*surface*.Draw();

*gl*.clear(*gl*.*DEPTH\_BUFFER\_BIT*);

*cam*.ApplyRightFrustum();

modelViewProjection = m4.multiply(*cam*.projection, m4.multiply(*cam*.modelview, matAccum1));

*gl*.uniformMatrix4fv(*shProgram*.iModelViewProjectionMatrix, false, modelViewProjection);

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

*surface*.Draw();

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

// gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.multiply(

// modelViewProjection,

// m4.translation(...cyl2(texCoord[0] \* 2 \* b, texCoord[1] \* 2 \* PI))

// ));

// gl.uniform4fv(shProgram.iColor, [1, 1, 1, 100]);

// sphere.Draw();

}

function animation() {

draw()

*window*.requestAnimationFrame(animation)

}

function hexToRgb(hex) {

var result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);

return [

parseInt(result[1], 16) / 255,

parseInt(result[2], 16) / 255,

parseInt(result[3], 16) / 255]

}

let *r2* = 1,

*r1* = *r2* \* 0.5,

*b* = 1;

function CreateSurfaceData() {

let vertexList = [];

let normalList = [];

let textureList = [];

const NUM\_STEPS\_I = parseInt(*document*.getElementById('numStepsI').value);

const NUM\_STEPS\_J = 50;

*r2* = *document*.getElementById('r2').value

// r1 = document.getElementById('r1').value

*b* = *document*.getElementById('b').value

const ist = 2 \* *b* / NUM\_STEPS\_I;

const jst = 2 \* *PI* / NUM\_STEPS\_J;

for (let i = 0; i < 2 \* *b*; i += ist) {

for (let j = 0; j < 2 \* *PI*; j += jst) {

vertexList.push(

...cyl2(i, j),

...cyl2(i + ist, j),

...cyl2(i, j + jst),

...cyl2(i, j + jst),

...cyl2(i + ist, j),

...cyl2(i + ist, j + jst),

);

normalList.push(

...normalAnalytical(i, j),

...normalAnalytical(i + ist, j),

...normalAnalytical(i, j + jst),

...normalAnalytical(i, j + jst),

...normalAnalytical(i + ist, j),

...normalAnalytical(i + ist, j + jst),

)

textureList.push(

i / (2 \* *b*), j / (2 \* *PI*),

(i + ist) / (2 \* *b*), j / (2 \* *PI*),

i / (2 \* *b*), (j + jst) / (2 \* *PI*),

i / (2 \* *b*), (j + jst) / (2 \* *PI*),

(i + ist) / (2 \* *b*), j / (2 \* *PI*),

(i + ist) / (2 \* *b*), (j + jst) / (2 \* *PI*),

)

}

}

return [vertexList, normalList, textureList];

}

function CreateSphereData() {

let vertexList = [];

let u = 0,

t = 0;

while (u < *Math*.PI \* 2) {

while (t < *Math*.PI) {

let v = getSphereVertex(u, t);

let w = getSphereVertex(u + 0.1, t);

let wv = getSphereVertex(u, t + 0.1);

let ww = getSphereVertex(u + 0.1, t + 0.1);

vertexList.push(v.x, v.y, v.z);

vertexList.push(w.x, w.y, w.z);

vertexList.push(wv.x, wv.y, wv.z);

vertexList.push(wv.x, wv.y, wv.z);

vertexList.push(w.x, w.y, w.z);

vertexList.push(ww.x, ww.y, ww.z);

t += 0.1;

}

t = 0;

u += 0.1;

}

return [vertexList, vertexList, vertexList];

}

const *radius* = 0.05;

function getSphereVertex(long, lat) {

return {

x: *radius* \* *Math*.cos(long) \* *Math*.sin(lat),

y: *radius* \* *Math*.sin(long) \* *Math*.sin(lat),

z: *radius* \* *Math*.cos(lat)

}

}

const *eps* = 0.0001

function normalAnalytical(ii, jj) {

let u1 = cyl2(ii, jj),

u2 = cyl2(ii + *eps*, jj),

v1 = cyl2(ii, jj),

v2 = cyl2(ii, jj + *eps*);

const dU = [], dV = []

for (let i = 0; i < 3; i++) {

dU.push((u1[i] - u2[i]) / *eps*)

dV.push((v1[i] - v2[i]) / *eps*)

}

const n = m4.normalize(m4.cross(dU, dV))

return n

}

function cyl2(a, b) {

let x = r(a) \* cos(b);

let y = r(a) \* sin(b);

let z = a;

return [x, y, z];

}

function r(a) {

let rr = (*r2* - *r1*) \* pow(sin(*PI* \* a / (4 \* *b*)), 2) + *r1*

return rr;

}

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

function initGL() {

let prog = createProgram(*gl*, vertexShaderSource, fragmentShaderSource);

*shProgram* = new ShaderProgram('Basic', prog);

*shProgram*.Use();

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

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

*shProgram*.iAttribTexture = *gl*.getAttribLocation(prog, "texture");

*shProgram*.iModelViewProjectionMatrix = *gl*.getUniformLocation(prog, "ModelViewProjectionMatrix");

*shProgram*.iNormalMatrix = *gl*.getUniformLocation(prog, "NormalMatrix");

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

*shProgram*.iLightPos = *gl*.getUniformLocation(prog, "lightPos");

*shProgram*.iTT = *gl*.getUniformLocation(prog, "tt");

*shProgram*.iScale = *gl*.getUniformLocation(prog, "s");

*surface* = new Model('Surface');

*surface*.BufferData(...CreateSurfaceData());

*sphere* = new Model()

*sphere*.BufferData(...CreateSphereData())

*bkg* = new Model('Background');

let bkgVerts = getBkgVerts();

*bkg*.BufferData(bkgVerts, bkgVerts, bkgTxrs());

*gl*.enable(*gl*.*DEPTH\_TEST*);

}

function getBkgVerts() {

const verts = [

[-1, -1, 0],

[1, 1, 0],

[1, -1, 0],

[-1, 1, 0]

]

const inds = [1, 0, 3, 0, 1, 2]

let vertexList = []

inds.forEach(i => {

vertexList.push(...verts[i])

})

return vertexList;

}

function bkgTxrs() {

const txrs = [

[1, 1],

[0, 0],

[0, 1],

[1, 0]]

const inds = [1, 0, 3, 0, 1, 2]

let textureList = []

inds.forEach(i => {

textureList.push(...txrs[i])

})

return textureList;

}

/\* 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() {

AccessWbc()

*cam* = new StereoCamera(10, 2, 1, 40, 0.1, 40);

*gui* = new GUI();

const camParam = *gui*.addFolder('Camera Parameters');

camParam.add(*cam*, 'Convergence', 10, 500, 10)

camParam.add(*cam*, 'EyeSeparation', 0, 15, 0.1)

camParam.add(*cam*, 'FOV', 0, 2, 0.01)

camParam.add(*cam*, 'NearClippingDistance', 0.1, 19, 0.1)

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

}

catch (e) {

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

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

return;

}

initCtx()

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

CreateWbcTxr()

LoadTexture()

animation();

}

function CreateWbcTxr() {

*txr* = *gl*.createTexture();

*gl*.bindTexture(*gl*.*TEXTURE\_2D*, *txr*);

*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*);

}

function AccessWbc() {

*wbc* = *document*.createElement('video');

*wbc*.setAttribute('autoplay', true);

*navigator*.getUserMedia({ video: true, audio: false }, function (stream) {

*wbc*.srcObject = stream;

*track* = stream.getTracks()[0];

}, function (e) {

*console*.error('Rejected!', e);

});

}

function LoadTexture() {

*texture* = *gl*.createTexture();

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

*gl*.texParameteri(*gl*.*TEXTURE\_2D*, *gl*.*TEXTURE\_MIN\_FILTER*, *gl*.*LINEAR*);

*gl*.texParameteri(*gl*.*TEXTURE\_2D*, *gl*.*TEXTURE\_MAG\_FILTER*, *gl*.*LINEAR*);

const image = new Image();

image.crossOrigin = 'anonymus';

image.src = "https://raw.githubusercontent.com/dimasambo/computer-graphics/CGW/pic-colors.png";

image.onload = () => {

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

*gl*.texImage2D(

*gl*.*TEXTURE\_2D*,

0,

*gl*.*RGBA*,

*gl*.*RGBA*,

*gl*.*UNSIGNED\_BYTE*,

image

);

*console*.log("imageLoaded")

draw()

}

}

*window*.onkeydown = (e) => {

if (e.keyCode == 87) {

*texCoord*[0] = *Math*.min(*texCoord*[0] + 0.01, 1);

}

else if (e.keyCode == 83) {

*texCoord*[0] = *Math*.max(*texCoord*[0] - 0.01, 0);

}

else if (e.keyCode == 68) {

*texCoord*[1] = *Math*.min(*texCoord*[1] + 0.01, 1);

}

else if (e.keyCode == 65) {

*texCoord*[1] = *Math*.max(*texCoord*[1] - 0.01, 0);

}

}

let *ctx*, *audio*, *src*, *bps*, *pnr*;

function initCtx() {

*audio* = *document*.getElementById('audioid');

*audio*.addEventListener('play', () => {

if (!*ctx*) {

*ctx* = new AudioContext();

*src* = *ctx*.createMediaElementSource(*audio*);

*pnr* = *ctx*.createPanner();

*bps* = *ctx*.createBiquadFilter();

*src*.connect(*pnr*);

*pnr*.connect(*bps*);

*bps*.connect(*ctx*.destination);

*bps*.type = 'bandpass';

*bps*.frequency.value = 1111;

*bps*.Q.value = 1;

const audioParam = *gui*.addFolder('Audio Parameters');

audioParam.add(*bps*.frequency, 'value', 0, 20000, 1)

audioParam.add(*bps*.Q, 'value', 0, 1, 0.01)

*ctx*.resume();

}

})

*audio*.addEventListener('pause', () => {

*console*.log('pause');

*ctx*.resume();

})

const bpsEnabled = *document*.getElementById('bps');

bpsEnabled.addEventListener('change', function () {

if (bpsEnabled.checked) {

*pnr*.disconnect();

*pnr*.connect(*bps*);

*bps*.connect(*ctx*.destination);

} else {

*pnr*.disconnect();

*pnr*.connect(*ctx*.destination);

}

});

*audio*.play();

}