Ministry of Science of Ukraine

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
«IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE»
Cathedra of Design Automation of Energy Processes and Systems

Subject Spatial audio

Calculation and graphics work

Academic discipline «Methods of synthesis of virtual reality»

Student
Pereverziev Luka Ruslanovych

Kyiv – 2023

1. TASK DESCRIPTION

Requirements:

- reuse the code from practical assignment #2;
- implement sound source rotation around the geometrical center of the surface patch by means of tangible interface (the surface stays still this time and the sound source moves). Reproduce your favorite song in mp3/ogg format having the spatial position of the sound source controlled by the user;
- visualize position of the sound source with a sphere;
- add a sound filter (use BiquadFilterNode interface) per variant. Add a checkbox element that enables or disables the filter. Set filter parameters according to your taste.

2. THEORETICAL BACKGROUND

2.1. WebAudio HTML5 API.

The Web Audio API is a powerful JavaScript API that provides a way to create, manipulate, and process audio in web applications.

The API allows developers to work with audio sources, effects, and audio destinations, enabling them to build interactive and immersive audio experiences directly within web pages. There are several examples of how the Web Audio API can be used:

1. Audio Effects: The Web Audio API provides a wide range of built-in audio effects, such as reverb, delay, distortion, and equalization. These effects can be applied to audio sources or create custom audio processing pipelines.
2. Spatial Audio: The API supports spatial audio processing, allowing positioning of audio sources in a 3D virtual space. This enables creating immersive audio environments where sounds appear to come from different directions and distances. It's ideal for building virtual reality experiences.
3. Audio Recording and Manipulation: The Web Audio API can capture audio from a user's microphone or other audio sources, making it possible to build web-based audio recording applications. It's possible to manipulate the recorded audio by applying effects, trimming, merging, or exporting it to different formats.
4. Audio Streaming: The API supports streaming audio data, which means dynamically loading audio files and playing them in real-time. This is useful for building web-based music players, podcast applications, or live audio streaming platforms.

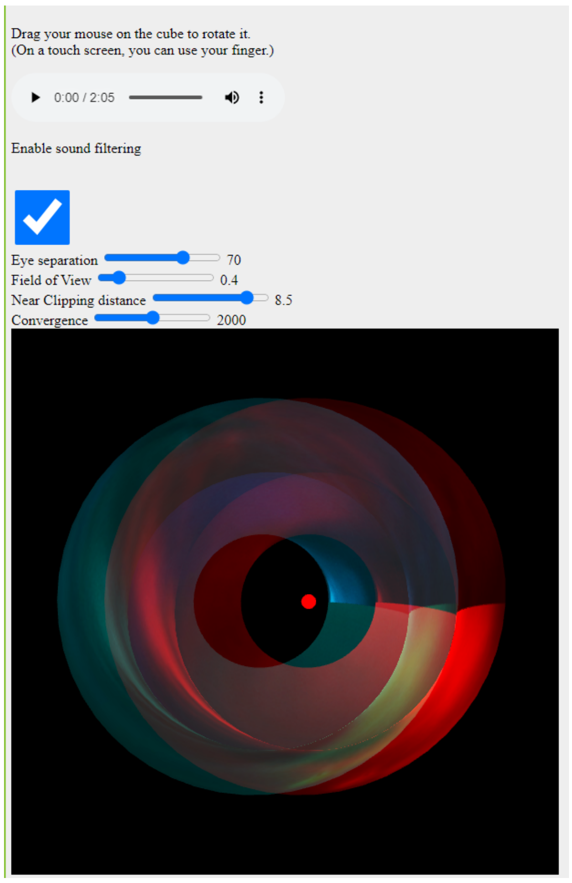
2.2. API interfaces.

The Web Audio API provides several interfaces that are essential for working with audio, of which the following were used in this CGW:

5. AudioContext: The AudioContext interface represents an audio processing graph, which is the main entry point for using the Web Audio API. It provides methods for creating audio nodes, managing audio sources and destinations, and handling audio processing. An AudioContext instance is required to interact with most other Web Audio API interfaces.
6. MediaElementAudioSourceNode: The MediaElementAudioSourceNode interface in the Web Audio API represents an audio source node that takes its input from an HTML media element, such as an <audio> or <video> element. It allows extracting the audio from the media element and using it within the Web Audio API for further processing, routing, or manipulation.
7. PannerNode: The PannerNode interface in the Web Audio API represents a node that is used for spatial audio positioning and panning. It allows positioning and control of the spatial attributes of an audio source, making it appear to come from a specific direction or move around in a virtual 3D space.
8. BiquadFilterNode: The BiquadFilterNode can apply various types of filters to audio signals, such as low-pass, high-pass, band-pass, and peaking filters. The BiquadFilterNode has properties for setting the frequency and Q factor. The frequency property determines the cutoff or center frequency depending on the filter type. The Q factor controls the width of the frequency range affected by the filter. Higher Q values create narrower filter bands, while lower Q values result in broader bands. There are also properties and methods to control the filter's gain. The gain property sets the gain (in decibels) applied by the filter.

3. IMPLEMENTATION DETAILS

In this work, the Web Audio API is used to implement spatial audio and position a sound source in a virtual 3D space using user input, namely the tilt of the user's smartphone. The secondary task consists of applying a rejection sound filter using the BiquadFilterNode interface. The basis of this calculation and graphics work is an asteroidal torus from the previous CGW. The accelerometer of the end user's device is used to track and map the tilting of the 3D surface and the position of spatial audio in virtual space. The following figure depicts the asteroidal torus from the previous CGW along with stereo effect sliders from practical assignments 1 and 2. As per the requirements, an audio player was added, along with the option to turn the rejection sound filter on and off. The red circle indicates the position of spatial audio.


Drag your mouse on the cube to rotate it.
(On a touch screen, you can use your finger.)

0:00 / 2:05

Enable sound filtering

✓

Eye separation 70
Field of View 0.4
Near Clipping distance 8.5
Convergence 2000

Fig. 3.1. CGW demo.

This was achieved by adding an mp3 audio file and initializing it using a <audio> HTML element present in index.html. To function properly, audio effects mentioned above need an AudioContext, which will contain objects related to audio manipulation. audioSource object initialized with createMediaElementSource method uses the media element as an audio source. audioPanner object initialized with createPanner will be used to position the audio in a virtual 3D space. audioFilter object initialized with createBiquadFilter is needed to apply a rejection filter to the recording. After connecting the media element source node to the other audio nodes and setting the filter properties it had to be linked to the elements in the index.html file using eventListeners.

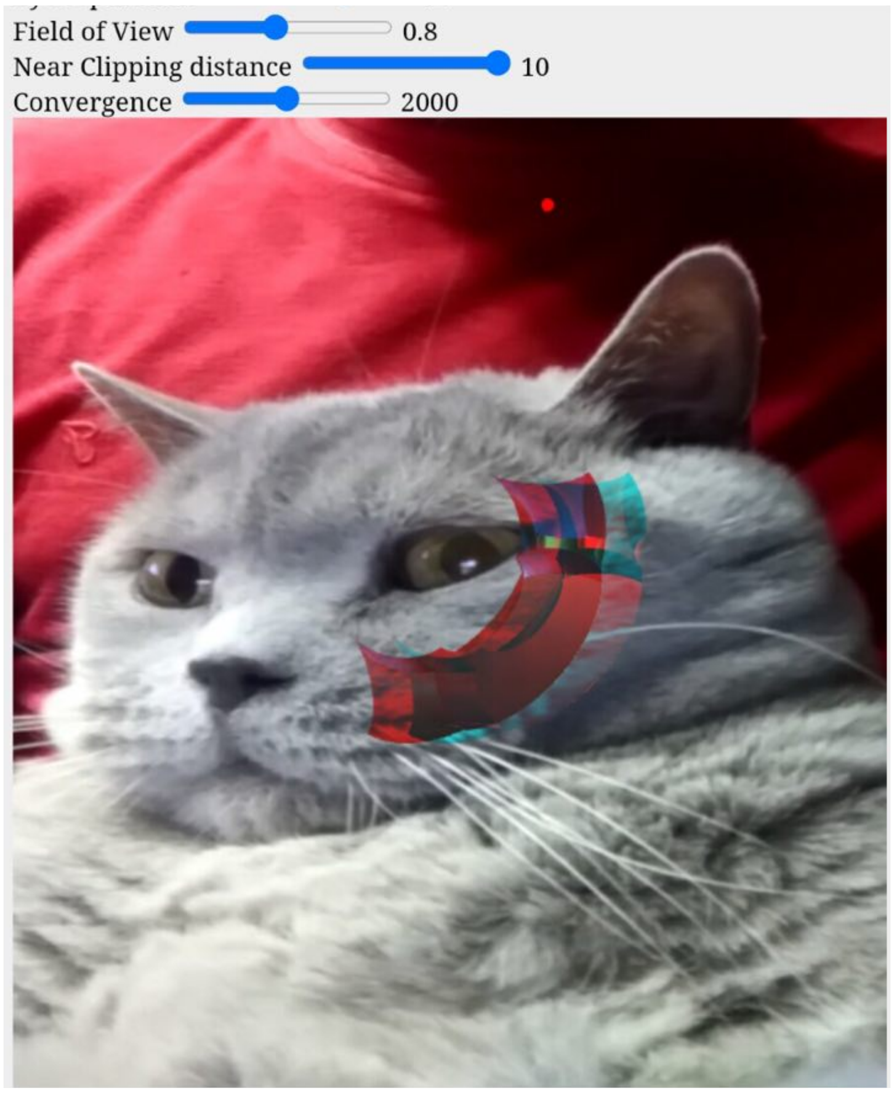
The final version of the web application allows rotating of both the asteroidal torus and the sound source based on the tilt of the user's smartphone. A anaglyphic stereo image effect is applied to the torus, with several sliders to alter the effects. The location of the sound in 3D space is represented by a red dot. The rejection filter applied to the mp3 file can be enabled and disabled at any time during the playback. The end user's webcam is used as a background.

The app features the 3D surface drawn using WebGL and texture from the previous calculation and graphics work as well as a prepared piano instrumental to demonstrate the functionality of the sound filter.

The source code of the project, all media used and photo- and video presentations of this assignment have since been uploaded to GitHub.

4. USER INSTRUCTIONS

To view the final result, the user has to clone the repository https://github.com/RewindRewind/MSVR, switch to the “CGW” branch and deploy a server to deliver the pages to the user’s smartphone. The index page of the app is depicted in Fig. 4.1.:



Field of View 0.8
Near Clipping distance 10
Convergence 2000

Fig. 4.1. index.html.

The application’s use cases include:

- rotating the astroidal torus by using touch controls or dragging the mouse on the shape;
- tilting the astroidal torus by tilting the smartphone;
- changing the anaglyphic effect applied to the image, such as eye separation, FOV, near clipping distance and convergence;

- playing back and pausing a preset audio file, as well as rotating the source of the sound by tilting the smartphone;
- enabling and disabling a rejection sound filter applied to the recording.

Each action is independent and can be performed simultaneously.

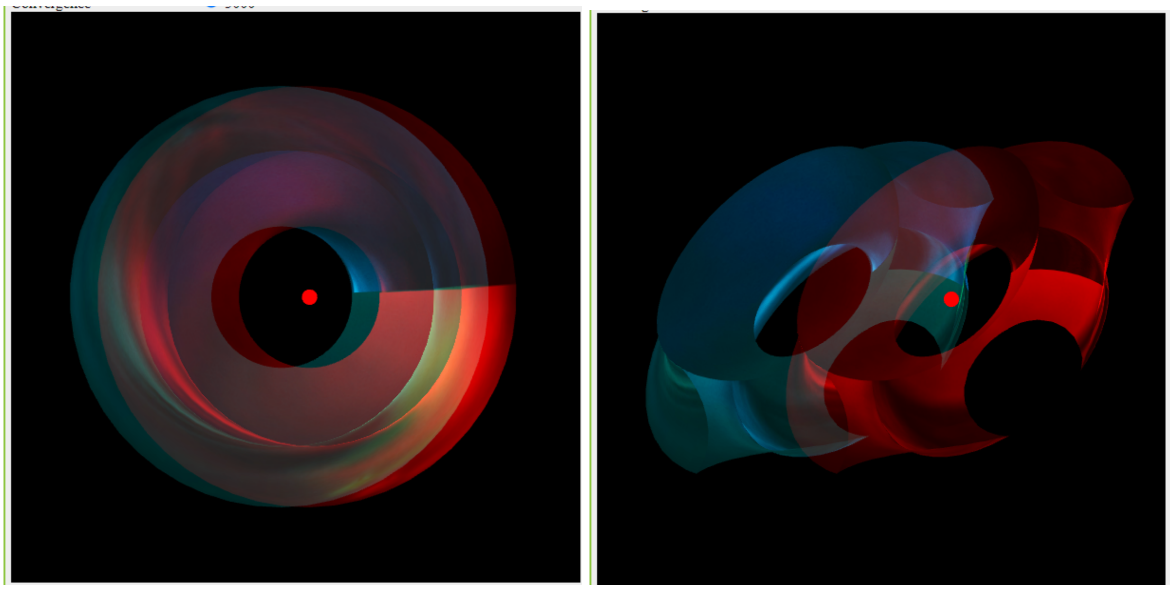
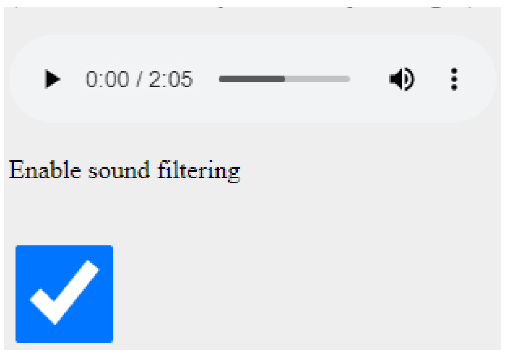


Fig. 4.2, 4.3. Visual demonstration of implemented functionality.

A built-in music player can be used to play, pause, change volume, alter playback speed and download the recording. Sound filtering is controlled with a checkbox.



0:00 / 2:05

Enable sound filtering

✓

Fig. 4.4. Audio web player.

5. SOURCE CODE

main.js:

```
'use strict';

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.

let point;
let texturePoint;
let scalingKoef;



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


// Constructor
function Model(name) {
  this.name = name;
  this.iVertexBuffer = gl.createBuffer();
  this.iTextureBuffer = gl.createBuffer();
  this.count = 0;
  this.countTexture = 0;

    this.BufferData = function (vertices) {

    gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STREAM_DRAW);

    this.count = vertices.length / 3;
  }

    this.TextureBufferData = function (normals) {
```

```
    gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextureBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(normals), gl.STREAM_DRAW);

    this.countTexture = normals.length / 2;
  }

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

    gl.drawArrays(gl.TRIANGLE_STRIP, 0, this.count);
  }

    this.DisplayPoint = 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.LINE_STRIP, 0, this.count);
  }
}

function CreateSphereSurface(r = 0.05) {
  let vertexList = [];
  let lon = -Math.PI;
  let lat = -Math.PI * 0.5;
  while (lon < Math.PI) {
    while (lat < Math.PI * 0.5) {
      let v1 = sphereSurfaceData(r, lon, lat);
      vertexList.push(v1.x, v1.y, v1.z);
      lat += 0.05;
    }
    lat = -Math.PI * 0.5;
    lon += 0.05;
  }
  return vertexList;
```

```
}

function sphereSurfaceData(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 };
}


// Constructor
function ShaderProgram(name, program) {

  this.name = name;
  this.prog = program;

  // Location of the attribute variable in the shader program.
  this.iAttribVertex = -1;
  this.iAttribTexture = -1;
  // Location of the uniform matrix representing the combined transformation.
  this.iModelViewProjectionMatrix = -1;

  this.iTranslatePoint = -1;
  this.iTexturePoint = -1;
  this.iScalingKoef = -1;
  this.iTMU = -1;

  this.Use = function() {
    gl.useProgram(this.prog);
  }
}


/* 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 backPlaneView = m4.identity();

let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7);
let bPRotate = m4.axisRotation([0.707, 0.707, 0], 0.0);
let translateToPointZero = m4.translation(0, 0, -10);
let bPTranslate = m4.translation(-0.5, -0.5, -10);

let matAccum0 = m4.multiply(rotateToPointZero, modelView);
let matAccum1 = m4.multiply(translateToPointZero, matAccum0);
let bPMat0 = m4.multiply(bPRotate, backPlaneView);
let bPMat1 = m4.multiply(bPTranslate, bPMat0);
let bPMat2 = m4.multiply(m4.scaling(4, 4, 1), bPMat1);

/* 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);


gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false, projection);

gl.uniform1i(shProgram.iTMU, 0);
gl.enable(gl.TEXTURE_2D);
gl.uniform2fv(shProgram.iTexturePoint, [texturePoint.x, texturePoint.y]);
gl.uniform1f(shProgram.iScalingKoef, scalingKoef);
gl.bindTexture(gl.TEXTURE_2D, vTexture);
gl.texImage2D(
  gl.TEXTURE_2D,
  0,
  gl.RGBA,
  gl.RGBA,
  gl.UNSIGNED_BYTE,
  video
);
gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, bPMat2);
backPlane.Draw();
```

```
  // let tangibleMat = m4.multiply(m4.axisRotation([0, 1, 0], -0.5 * Math.PI *
accelerometer.x * 0.1), m4.axisRotation([1, 0, 0], 0.5 * Math.PI *
accelerometer.y * 0.1));
  // gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false,
m4.multiply(matAccum1, tangibleMat));
  gl.uniformMatrix4fv(shProgram.iModelViewMatrix, false, matAccum1);
  gl.bindTexture(gl.TEXTURE_2D, texture);
  cam.Param();
  cam.ApplyLeftFrustum();
  gl.colorMask(false, true, true, false);
  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false,
cam.mProjectionMatrix);
  surface.Draw();
  gl.clear(gl.DEPTH_BUFFER_BIT);
  cam.ApplyRightFrustum();
  gl.colorMask(true, false, false, false);
  gl.uniformMatrix4fv(shProgram.iProjectionMatrix, false,
cam.mProjectionMatrix);
  surface.Draw();
  gl.colorMask(true, true, true, true);
  // let tr = AstroidalTorus(map(texturePoint.x, 0, 1, 0, Math.PI * 2),
map(texturePoint.y, 0, 1, 0, Math.PI * 2))
  let tr = m4.translation(-0.4 * accelerometer.x + 0.8 * accelerometer.x, -0.4
* accelerometer.y + 0.8 * accelerometer.y, -0.4 * accelerometer.z + 0.8 *
accelerometer.z);
  gl.uniform3fv(shProgram.iTranslatePoint, [-0.4 * accelerometer.x + 0.8 *
accelerometer.x, -0.4 * accelerometer.y + 0.8 * accelerometer.y, -0.4 *
accelerometer.z + 0.8 * accelerometer.z])
  gl.uniform1f(shProgram.iScalingKoef, -scalingKoef);
  point.DisplayPoint();
  if (audioPanner) {
    audioPanner.setPosition(-0.4 * accelerometer.x + 0.8 * accelerometer.x,
-0.4 * accelerometer.y + 0.8 * accelerometer.y, -0.4 * accelerometer.z + 0.8 *
accelerometer.z)
  }
}

function reDraw() {
  draw()
  window.requestAnimationFrame(reDraw);
}
```

```
function CreateSurfaceData() {
  let vertexList = [];
  let uMax = Math.PI * 2;
  let vMax = Math.PI * 2;
  let uStep = uMax / 50;
  let vStep = vMax / 50;

  for (let u = 0; u <= uMax; u += uStep) {
    for (let v = 0; v <= vMax; v += vStep) {
      let vert = AstroidalTorus(u, v);
      let avert = AstroidalTorus(u + uStep, v);
      let bvert = AstroidalTorus(u, v + vStep);
      let cvert = AstroidalTorus(u + uStep, v + vStep);

      vertexList.push(vert.x, vert.y, vert.z);
      vertexList.push(avert.x, avert.y, avert.z);
      vertexList.push(bvert.x, bvert.y, bvert.z);

      vertexList.push(avert.x, avert.y, avert.z);
      vertexList.push(cvert.x, cvert.y, cvert.z);
      vertexList.push(bvert.x, bvert.y, bvert.z);
    }
  }

  return vertexList;
}

function CreateTexture() {
  let texture = [];
  let uMax = Math.PI * 2;
  let vMax = Math.PI * 2;
  let uStep = uMax / 50;
  let vStep = vMax / 50;

  for (let u = 0; u <= uMax; u += uStep) {
    for (let v = 0; v <= vMax; v += vStep) {
      let u1 = map(u, 0, uMax, 0, 1);
      let v1 = map(v, 0, vMax, 0, 1);
      texture.push(u1, v1);
      u1 = map(u + uStep, 0, uMax, 0, 1);
      texture.push(u1, v1);
      u1 = map(u, 0, uMax, 0, 1);
      v1 = map(v + vStep, 0, vMax, 0, 1);
      texture.push(u1, v1);
      u1 = map(u + uStep, 0, uMax, 0, 1);
      v1 = map(v, 0, vMax, 0, 1);
      texture.push(u1, v1);
      v1 = map(v + vStep, 0, vMax, 0, 1);
      texture.push(u1, v1);
      u1 = map(u, 0, uMax, 0, 1);
      v1 = map(v + vStep, 0, vMax, 0, 1);
      texture.push(u1, v1);
    }
  }

  return texture;
}

function map(val, f1, t1, f2, t2) {
  let m;
  m = (val - f1) * (t2 - f2) / (t1 - f1) + f2;
  return Math.min(Math.max(m, f2), t2);
}

function AstroidalTorus(u, v) {
  let r = 1;
  let a = 0.5;
  let t = 0.5 * Math.PI;
  let x = (r + a * Math.cos(u) ** 3 * Math.cos(t) - a * Math.sin(u) ** 3 *
Math.sin(t)) * Math.cos(v);
  let y = (r + a * Math.cos(u) ** 3 * Math.cos(t) - a * Math.sin(u) ** 3 *
Math.sin(t)) * Math.sin(v);
  let z = a * Math.cos(u) ** 3 * Math.sin(t) + Math.sin(u) ** 3 * Math.cos(t);
  return { x: x, y: y, z: z }
}


/* 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.iAttribTexture = gl.getAttribLocation(prog, "texture");
  shProgram.iModelViewMatrix = gl.getUniformLocation(prog, "ModelViewMatrix");
  shProgram.iProjectionMatrix = gl.getUniformLocation(prog,
"ProjectionMatrix");
  shProgram.iTranslatePoint = gl.getUniformLocation(prog, 'translatePoint');
  shProgram.iTexturePoint = gl.getUniformLocation(prog, 'texturePoint');
  shProgram.iScalingKoef = gl.getUniformLocation(prog, 'scalingKoef');
  shProgram.iTMU = gl.getUniformLocation(prog, 'tmu');

  surface = new Model('AstroidalTorus');
  surface.BufferData(CreateSurfaceData());
  LoadTexture();
  surface.TextureBufferData(CreateTexture());
  point = new Model('Point');
  point.BufferData(CreateSphereSurface())
  backPlane = new Model('Plane')
  backPlane.BufferData([0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
0]);
  backPlane.TextureBufferData([1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1]);
  gl.enable(gl.DEPTH_TEST);
}


/* 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 StereoCamera(
  Convergence,
  EyeSeparation,
  AspectRatio,
  FOV,
  NearClippingDistance,
  FarClippingDistance
) {
  this.mConvergence = Convergence;
  this.mEyeSeparation = EyeSeparation;
  this.mAspectRatio = AspectRatio;
  this.mFOV = FOV;
  this.mNearClippingDistance = NearClippingDistance;
  this.mFarClippingDistance = FarClippingDistance;

  this.mProjectionMatrix = null;
  this.mModelViewMatrix = null;

    this.ApplyLeftFrustum = function () {
    let top, bottom, left, right;
    top = this.mNearClippingDistance * Math.tan(this.mFOV / 2);
    bottom = -top;

    let a = this.mAspectRatio * Math.tan(this.mFOV / 2) * this.mConvergence;
    let b = a - this.mEyeSeparation / 2;
    let c = a + this.mEyeSeparation / 2;

    left = (-b * this.mNearClippingDistance) / this.mConvergence;
    right = (c * this.mNearClippingDistance) / this.mConvergence;

    // Set the Projection Matrix
    this.mProjectionMatrix = m4.frustum(
      left,
      right,
      bottom,
      top,
      this.mNearClippingDistance,
      this.mFarClippingDistance
    );

    // Displace the world to right
    this.mModelViewMatrix = m4.translation(
      this.mEyeSeparation / 2,
      0.0,
      0.0
    );
  };

    this.ApplyRightFrustum = function () {
    let top, bottom, left, right;
    top = this.mNearClippingDistance * Math.tan(this.mFOV / 2);
    bottom = -top;

    let a = this.mAspectRatio * Math.tan(this.mFOV / 2) * this.mConvergence;
    let b = a - this.mEyeSeparation / 2;
    let c = a + this.mEyeSeparation / 2;

    left = (-c * this.mNearClippingDistance) / this.mConvergence;
    right = (b * this.mNearClippingDistance) / this.mConvergence;

    // Set the Projection Matrix
    this.mProjectionMatrix = m4.frustum(
      left,
      right,
      bottom,
      top,
      this.mNearClippingDistance,
      this.mFarClippingDistance
    );

    // Displace the world to left
    this.mModelViewMatrix = m4.translation(
      -this.mEyeSeparation / 2,
      0.0,
      0.0
    );
  };

    this.Param = function () {
    let ins = document.getElementsByClassName("in");
    let outs = document.getElementsByClassName("out");
    let eyeSeparation = 70.0;
    eyeSeparation = ins[0].value;
    outs[0].innerHTML = eyeSeparation;
    this.mEyeSeparation = eyeSeparation;
    let FOV = 0.8;
    FOV = ins[1].value;
    outs[1].innerHTML = FOV;
    this.mFOV = FOV;
    let nearClippingDistance = 5.0;
    nearClippingDistance = ins[2].value - 0.0;
    outs[2].innerHTML = nearClippingDistance;
    this.mNearClippingDistance = nearClippingDistance
    let convergence = 2000.0;
    convergence = ins[3].value;
    outs[3].innerHTML = convergence;
    this.mConvergence = convergence;
  }
}

function init() {
    cam = new StereoCamera(
        2000,         // Convergence
        70.0,         // Eye Separation
        1,            // Aspect Ratio
        0.8,          // FOV in degrees * PI / 180 degrees
        10,           // Near Clipping Distance
        20000         // Far Clipping Distance
  );
  texturePoint = { x: 0.1, y: 0.1 }
  scalingKoef = 1.0;
  let canvas;
  startAudio();
  try {
    canvas = document.getElementById("webglcanvas");
    gl = canvas.getContext("webgl");
    video = document.createElement('video');
    video.setAttribute('autoplay', true);
    window.vid = video;
    getWebcam();
    vTexture = CreateWebCamTexture();
    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;
  }

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

let texture, video, track;
let vTexture, backPlane, cam;

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/RewindRewind/WebGL/CGW/texture.jpg";
  image.onload = () => {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(
      gl.TEXTURE_2D,
      0,
      gl.RGBA,
      gl.RGBA,
      gl.UNSIGNED_BYTE,
      image
    );
    draw()
  }
}

function getWebcam() {
    navigator.getUserMedia({ video: true, audio: false }, function (stream) {
    video.srcObject = stream;
    track = stream.getTracks()[0];
    }, function (e) {
    console.error('Rejected!', e);
  });
}

function CreateWebCamTexture() {
  let textureID = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, textureID);
  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);
  return textureID;
}

let audioElement = null;
let audioContext;
```

```javascript
let audioSource;
let audioPanner;
let audioFilter;

function initializeAudio() {
  audioElement = document.getElementById('audio');

  audioElement.addEventListener('play', handlePlay);

  audioElement.addEventListener('pause', handlePause);
}

function handlePlay() {
  console.log('play');
  if (!audioContext) {
    audioContext = new AudioContext();
    audioSource = audioContext.createMediaElementSource(audioElement);
    audioPanner = audioContext.createPanner();
    audioFilter = audioContext.createBiquadFilter();

    // Connect audio nodes
    audioSource.connect(audioPanner);
    audioPanner.connect(audioFilter);
    audioFilter.connect(audioContext.destination);

    // Set filter parameters
    audioFilter.type = 'notch';
    audioFilter.Q.value = 0.1;
    audioFilter.frequency.value = 900;
    audioFilter.gain.value = 19;

    audioContext.resume();
  }
}

function handlePause() {
  console.log('pause');
  audioContext.resume();
}

function toggleFilter() {
  let filterCheckbox = document.getElementById('filterCheckbox');
  if (filterCheckbox.checked) {
    // Connect filter when checkbox is checked
    audioPanner.disconnect();
    audioPanner.connect(audioFilter);
    audioFilter.connect(audioContext.destination);
  } else {
    // Disconnect filter when checkbox is unchecked
    audioPanner.disconnect();
    audioPanner.connect(audioContext.destination);
  }
}

function startAudio() {
  initializeAudio();

  let filterCheckbox = document.getElementById('filterCheckbox');
  filterCheckbox.addEventListener('change', toggleFilter);

  audioElement.play();
}

const accelerometer = new Accelerometer({ frequency: 60 });
accelerometer.addEventListener("reading", () => {
  console.log(`Acceleration along the X-axis ${accelerometer.x}`);
  console.log(`Acceleration along the Y-axis ${accelerometer.y}`);
  console.log(`Acceleration along the Z-axis ${accelerometer.z}`);
});
accelerometer.start();
```

shader.gpu:

```
// Vertex shader
const vertexShaderSource = `
attribute vec3 vertex;
attribute vec2 texture;
uniform mat4 ModelViewMatrix;
uniform mat4 ProjectionMatrix;

varying vec2 v_texcoord;
```

```
uniform vec3 translatePoint;
uniform vec2 texturePoint;
uniform float scalingKoef;

mat4 translation(float tx, float ty, float tz) {
        mat4 dst;

        dst[0][0] = 1.0;
        dst[0][1] = 0.0;
        dst[0][2] = 0.0;
        dst[0][3] = 0.0;
        dst[1][0] = 0.0;
        dst[1][1] = 1.0;
        dst[1][2] = 0.0;
        dst[1][3] = 0.0;
        dst[2][0] = 0.0;
        dst[2][1] = 0.0;
        dst[2][2] = 1.0;
        dst[2][3] = 0.0;
        dst[3][0] = tx;
        dst[3][1] = ty;
        dst[3][2] = tz;
        dst[3][3] = 1.0;
        return dst;
}
mat4 scaling(float s){
        mat4 dst;

        dst[0][0] = s;
        dst[0][1] = 0.0;
        dst[0][2] = 0.0;
        dst[0][3] = 0.0;
        dst[1][0] = 0.0;
        dst[1][1] = s;
        dst[1][2] = 0.0;
        dst[1][3] = 0.0;
        dst[2][0] = 0.0;
        dst[2][1] = 0.0;
        dst[2][2] = s;
        dst[2][3] = 0.0;
        dst[3][0] = 0.0;
        dst[3][1] = 0.0;
```

```
	dst[3][2] = 0.0;
	dst[3][3] = 1.0;

	return dst;
}

void main() {
	mat4 sMatrix = scaling(scalingKoef);
	mat4 t1Matrix = translation(texturePoint.x,texturePoint.y,0.0);
	mat4 t2Matrix = translation(-texturePoint.x,-texturePoint.y,0.0);
	vec4 texture1 = vec4(texture,0.0,0.0)*t1Matrix;
	vec4 textureRotated = texture1*sMatrix;
	vec4 texture2 = textureRotated*t2Matrix;
	v_texcoord = vec2(texture2.x,texture2.y);
	vec4 vertPos4 = ModelViewMatrix * vec4(vertex, 1.0);
	gl_Position =ProjectionMatrix* vertPos4;
	if(scalingKoef<0.0){
	vec4 tPoint =
translation(translatePoint.x,translatePoint.y,translatePoint.z)*vertPos4;
	gl_Position = ProjectionMatrix*tPoint;
	}
}`;


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

varying vec2 v_texcoord;
uniform sampler2D tmu;
uniform float scalingKoef;

void main() {
	//Applying texture
	vec4 texColor = texture2D(tmu, v_texcoord);
	gl_FragColor = texColor;
	if(scalingKoef<0.0){
	gl_FragColor = vec4(0.,0.,0.,0.);
	}
```

}`;

index.html:

```html
<!DOCTYPE html>
<html><head><meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">


    <title>CGW</title>
    <style>
        body {
            background-color: #EEEEEE;
        }
    </style>

    <script src="./Utils/trackball-rotator.js"></script>
    <script src="./Utils/m4.js"></script>
    <script src="./shader.gpu"></script>
    <script src="./main.js"></script>

</head>
<body onload="init()">

    <h2>Astroidal Torus</h2>

    <p id="message">Drag your mouse on the cube to rotate it.<br>
        (On a touch screen, you can use your finger.)</p>

    <audio id="audio" src="./109 - Aphex Twin - Avril 14th.mp3" controls
loop></audio><br>
    <p>Enable sound filtering</p><br>
    <input style="width: 60px; height: 60px;" type="checkbox"
id="filterCheckbox" checked>

    <div>
        <span>Eye separation</span>
        <input type="range" class="in" value="70" min="0" max="100" step="">
        <span class="out"></span>
    </div>
    <div>
        <span>Field of View</span>
        <input type="range" class="in" value="0.8" min="0.2" max="1.6"
step="0.2">
        <span class="out"></span>
    </div>
    <div>
    <span>Near Clipping distance</span>
    <input type="range" class="in" value="10" min="0" max="10" step="0.5">
        <span class="out"></span>
    </div>
    <div>
        <span>Convergence</span>
        <input type="range" class="in" value="2000" min="1000" max="3000"
step="100">
        <span class="out"></span>
    </div>

    <div id="canvas-holder">
        <canvas width="600" height="600" id="webglcanvas"
style="background-color:red"></canvas>
    </div>



</body></html>
```