NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

“IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE”

Faculty of Heat Power Engineering

Department of Digital Technologies in Energy

Calculation and graphics work

Methods of synthesis of virtual reality

Spatial audio

Prepared by the student of TR-42mp, Harmash Diana

Київ – 2025

**CALCULATION AND GRAPHICS WORK**

**Assignment presentation:**

* Share the link to your [GIThub repo through this form](https://docs.google.com/forms/d/e/1FAIpQLScBpXP_UmSEATyyrk2qSt4Yq_xNOPYM0-_AQL253r6BNilYoA/viewform?usp=sharing" \t "_blank).
* Make sure the repo contains a paper presenting your assignment
* The practical assignment has to reside in a branch named CGW.

**Spatial audio**

Implement spatial audio through [WebAudio HTML5 API](https://www.w3.org/TR/webaudio/" \t "_blank).

**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 checkbox element that enables or disables the filter. Set filter parameters according to your taste.

**Paperwork**

Prepare a digital report that holds:

* a title page;
* a chapter describing the task (1 page);
* a chapter describing theory (2 pages);
* a chapter describing the implementation details (2 pages);
* a chapter of user's instruction with screenshots (2 pages);
* a sample of source code (2 pages).

Add report to Git branch

**PROJECT DESCRIPTION**

This project implements an interactive 3D audio-visual experience using WebGL for stereoscopic rendering and Web Audio API for spatial sound processing. The primary goals of the implementation were:

1. **Spatial Audio Implementation**: Create a sound source that rotates around a 3D surface based on smartphone sensor data. The sound source's position affects how the audio is heard, simulating a realistic 3D audio environment.
2. **Visual Representation**: Visualize the sound source as a 3D sphere that moves in sync with the audio source, allowing users to see the position of the sound in 3D space.
3. **Audio Filtering**: Implement audio processing with the Web Audio API's BiquadFilterNode, allowing users to modify the sound characteristics with adjustable parameters (frequency, Q-factor, gain).

The project builds upon anaglyphic stereo rendering (red-cyan) to create a complete multisensory experience where both visual and auditory cues are spatially coherent. This enhances immersion by providing consistent spatial information across multiple sensory channels.

The smartphone-based tangible interface enables intuitive control, transforming a common mobile device into a spatial controller that maps orientation to sound position, creating a natural interaction paradigm.

**THEORETICAL BACKGROUND**

**Spatial Audio Fundamentals**

Spatial audio reproduces sound in a way that creates the impression of a three-dimensional sound field. The human auditory system uses several cues to localize sounds:

1. **Interaural Time Difference (ITD)**: The difference in arrival time of a sound between the two ears. Sounds from the side reach the closer ear earlier.
2. **Interaural Level Difference (ILD)**: The difference in sound level between the two ears. The head creates an acoustic shadow, making sounds louder at the ear closer to the source.
3. **Head-Related Transfer Function (HRTF)**: The spectral filtering effect of the head, pinnae (outer ears), and torso, which modifies sound based on its direction.

The Web Audio API implements these principles through its PannerNode, which uses the HRTF model to create convincing 3D audio. For optimal results, this project uses the 'HRTF' panning model rather than the simpler 'equalpower' model.

**Binaural Audio Rendering**

The Web Audio API handles spatial audio through its spatializer components:

[AudioSource] → [GainNode] → [BiquadFilterNode] → [PannerNode] → [AudioDestination]

The PannerNode simulates the effect of a sound source positioned in 3D space relative to a listener. As the positions change, the audio is automatically processed to reflect the spatial relationship.

**Audio Filtering with BiquadFilterNode**

The BiquadFilterNode interface represents a simple low-order filter with various response types:

* **Peaking Filter**: Boosts or cuts a frequency band centered on a specific frequency
* **Low-pass/High-pass**: Attenuates frequencies above/below a cutoff point
* **Band-pass**: Allows frequencies within a range while attenuating others
* **Notch**: Attenuates frequencies within a specific range

Each filter is controlled by three primary parameters:

* **Frequency**: The center or cutoff frequency (in Hz)
* **Q**: The quality factor that controls the width of the frequency band
* **Gain**: The boost or cut amount (in dB) for peaking and shelving filters

For this project, a peaking filter was implemented to allow frequency-specific enhancement, creating distinctive audio effects as the sound source moves through space.

**IMPLEMENTATION DETAILS**

**System Architecture**

The implementation follows a modular architecture with distinct components for audio processing, 3D rendering, and sensor data handling:

1. **AudioController**: Manages Web Audio API nodes, spatial audio setup, and filter controls
2. **SoundSourceModel**: Creates and renders the 3D sphere representing the sound source
3. **WebSocket Handler**: Processes sensor data from the smartphone
4. **Stereo Renderer**: Handles anaglyphic 3D rendering of the scene

The system uses event-driven communication between components, with sensor data triggering updates to both audio and visual elements.

**Spatial Audio Implementation**

The core of the spatial audio system is implemented in AudioController.js, which creates and configures the Web Audio API nodes:

// Create PannerNode for spatial audio

this.panner = this.audioContext.createPanner();

this.panner.panningModel = 'HRTF'; // Using HRTF for better 3D effect

this.panner.distanceModel = 'inverse';

this.panner.refDistance = 1;

this.panner.maxDistance = 10000;

this.panner.rolloffFactor = 1;

The audio spatialization is achieved by dynamically updating the sound source position based on smartphone orientation data:

// Update sound position in 3D space

setSoundPosition(x, y, z) {

this.soundPosition = [x, y, z];

this.panner.setPosition(x, y, z);

}

The orientation data is processed through spherical-to-cartesian coordinate transformation:

const radius = 8.0;

const x = radius \* Math.sin(alpha) \* Math.cos(beta);

const y = radius \* Math.sin(beta);

const z = radius \* Math.cos(alpha) \* Math.cos(beta);

**Sound Source Visualization**

The sound source is visualized as a sphere rendered in anaglyphic 3D. The SoundSourceModel.js module generates the sphere geometry:

this.generateSphere = function(radius, latitudeBands, longitudeBands) {

// Generate vertices and indices for sphere

// ...

return { verticesF32: this.vertices, indicesU16: this.indices };

};

Special rendering techniques ensure the sphere is visible in anaglyph 3D:

// For left eye (red channel)

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

// ...

// For right eye (cyan channel)

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

**Audio Filtering Implementation**

The audio filter is implemented using the Web Audio API's BiquadFilterNode:

// Create filter node (peak/resonant filter)

this.filterNode = this.audioContext.createBiquadFilter();

this.filterNode.type = 'peaking';

this.filterNode.frequency.value = 1000; // 1kHz

this.filterNode.Q.value = 4.0; // Narrow bandwidth

this.filterNode.gain.value = 15; // +15dB gain

The filter can be toggled on and off by reconfiguring the audio graph:

toggleFilter(enabled) {

// Disconnect current connections

this.gainNode.disconnect();

if (enabled) {

// Route through filter

this.gainNode.connect(this.filterNode);

this.filterNode.connect(this.panner);

} else {

// Bypass filter

this.gainNode.connect(this.panner);

}

}

**Integration with Smartphone Sensors**

The system uses WebSockets to connect to a smartphone and receive orientation data:

function initWebSocket() {

const phoneIP = "192.168.50.23";

const phonePort = 8080;

const wsUrl = `ws://${phoneIP}:${phonePort}/sensor/connect?type=android.sensor.orientation`;

// ...

}

This enables a tangible interface where physical phone movements directly control the audio-visual experience.

**USER INSTRUCTIONS**

**System Requirements**

* Modern web browser with WebGL and Web Audio API support (Chrome, Firefox, Edge recommended)
* Smartphone with motion sensors and WebSocket capability
* Red-cyan anaglyph 3D glasses
* Stereo headphones (recommended for optimal spatial audio experience)

**Setup Instructions**

1. **Connect Your Smartphone**:
   * Make sure your computer and smartphone are on the same network
   * Note the IP address shown in the interface (default: 192.168.50.23)
   * Open the WebSocket client app on your smartphone
   * Connect to the provided address and port
2. **Put On 3D Glasses**:

* Wear red-cyan anaglyph glasses for the stereoscopic 3D effect
* The 3D surface should appear to float in front of the screen

1. **Load Audio File**:

* Click the "Select Audio File" button
* Choose an MP3 or OGG file from your computer
* Click "Play" to start audio playback

**Using the Interface**

1. **Control Sound Position**:
   * Toggle "Enable Sensor Control" to activate smartphone orientation tracking
   * Move your smartphone to change the sound source position
   * The yellow sphere represents the sound source location
   * You should hear the audio change as the source moves around you
2. **Adjust Audio Filter**:

* Toggle "Enable Peak Filter" to activate audio filtering
* Adjust the frequency slider to target different frequency ranges
* Modify the Q value to change the width of the frequency band
* Adjust the gain to increase or decrease the effect intensity

1. **Fine-tune 3D Settings**:

* Adjust "Eye Separation" to change the strength of the 3D effect
* Modify "Field of View" to change the perspective
* Adjust "Convergence" to set where objects appear in space

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**SOURCE CODE SAMPLES**

Sound Source Position Update (main.js)

function updateSoundPosition(alpha, beta, gamma) {

const radius = 8.0;

// Normalize angles in radians for calculation

alpha = alpha % (2 \* Math.PI);

beta = Math.max(-Math.PI/2, Math.min(Math.PI/2, beta));

gamma = Math.max(-Math.PI/2, Math.min(Math.PI/2, gamma));

// Convert spherical coordinates to Cartesian

const x = radius \* Math.sin(alpha) \* Math.cos(beta);

const y = radius \* Math.sin(beta);

const z = radius \* Math.cos(alpha) \* Math.cos(beta);

console.log("Sound position updated:", x, y, z);

soundSourcePosition = [x, y, z];

if (audioController && audioController.initialized) {

audioController.setSoundPosition(x, y, z);

}

if (soundSourceModel) {

soundSourceModel.setPosition(x, y, z);

}

requestAnimationFrame(drawScene);

}

Audio Filter Implementation (AudioController.js)

export default class AudioController {

constructor() {

// Initialize AudioContext and processing nodes

this.audioContext = null;

this.audioElement = null;

this.audioSource = null;

this.panner = null;

this.gainNode = null;

this.filterNode = null;

this.initialized = false;

this.isPlaying = false;

this.filterEnabled = false;

this.listenerPosition = [0, 0, 0];

this.soundPosition = [0, 0, 0];

}

async init() {

try {

this.audioContext = new (window.AudioContext || window.webkitAudioContext)();

this.audioElement = document.createElement('audio');

this.audioElement.loop = true;

this.audioSource = this.audioContext.createMediaElementSource(this.audioElement);

this.panner = this.audioContext.createPanner();

this.panner.panningModel = 'HRTF';

this.panner.distanceModel = 'inverse';

this.panner.refDistance = 1;

this.panner.maxDistance = 10000;

this.panner.rolloffFactor = 1;

this.gainNode = this.audioContext.createGain();

this.gainNode.gain.value = 0.8;

this.filterNode = this.audioContext.createBiquadFilter();

this.filterNode.type = 'peaking';

this.filterNode.frequency.value = 1000; // 1kHz

this.filterNode.Q.value = 4.0; // Narrow bandwidth

this.filterNode.gain.value = 15; // +15dB gain

this.audioSource.connect(this.gainNode);

this.gainNode.connect(this.panner);

this.panner.connect(this.audioContext.destination);

// Set up listener at origin

this.setupListener();

this.panner.setPosition(...this.soundPosition);

this.initialized = true;

console.log("Audio system initialized successfully");

return true;

} catch (error) {

console.error("Error initializing audio system:", error);

return false;

}

}

**CONCLUSION**

This implementation successfully combines WebGL 3D rendering with Web Audio API spatial processing to create an immersive audio-visual experience. The smartphone-based tangible interface provides an intuitive control method, while the audio filtering capabilities add another dimension of customization.

Future enhancements could include:

* Multiple sound sources with different characteristics
* More advanced filter configurations and presets
* Room acoustic simulation with reverb and reflections
* Integration with VR headsets for fully immersive experience

The project demonstrates the powerful capabilities of modern web technologies for creating interactive 3D audio-visual applications without requiring specialized hardware or software installations.