TUTORIAL 12 – AUDIO-VISUAL CONTENT SYNCHRONIZATION

Let's create a webpage that displays a 3D graphic animation where objects change their size and rotation synchronously with the audio intensity of a given audio file, and bounce back when they reach the edges of the screen. We can use **WebGL** with **Three.js** for rendering 3D graphics and **Web Audio API** for analyzing audio data.

Key Features:

- **Synchronized Motion**: The size and rotation of 3D objects (e.g., cubes) will change based on the audio intensity.
- **Size Change**: The size of the objects will change within 50% of their initial size based on the audio intensity.
- **Edge Detection and Bouncing**: Objects will bounce back when they reach the edges of the screen (camera's view area).
- **Audio Synchronization**: We'll use the **Web Audio API** to analyze the audio's frequency data and control the animation.

Full Example Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>3D Animation Synchronized with Audio Intensity</title>
 <style>
  body {
   margin: 0;
   overflow: hidden;
   background-color: #f0f0f0;
  }
  canvas {
   display: block;
  }
  #audioControls {
   position: absolute;
```

```
    top: 20px;
    left: 20px;
    z-index: 10;
   }
  </style>
</head>
<body>

  <div id="audioControls">
   <input type="file" id="audioFile" accept="audio/*">
  </div>

  <canvas id="canvas"></canvas>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>

  <script>
   let scene, camera, renderer;
   let audioContext, analyser, audioSource;
   let dataArray, bufferLength;
   let cubes = [];  // Array to hold 3D objects (cubes)
   let audio = new Audio(); // For the audio playback

   let objectSpeed = 0.1;  // Speed of the objects' movement
   let screenBounds = { xMin: -10, xMax: 10, yMin: -10, yMax: 10, zMin: -10, zMax: 10 };

   // Setup 3D scene
   function setupScene() {
    scene = new THREE.Scene();
    camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000);
    renderer = new THREE.WebGLRenderer({ canvas:
document.getElementById('canvas') });
    renderer.setSize(window.innerWidth, window.innerHeight);
    document.body.appendChild(renderer.domElement);

    // Add several cubes to the scene
    for (let i = 0; i < 5; i++) {
```

```
      const geometry = new THREE.BoxGeometry(1, 1, 1);  // Cube geometry
      const material = new THREE.MeshBasicMaterial({ color: Math.random() * 0xffffff });
      const cube = new THREE.Mesh(geometry, material);
      cube.position.set(Math.random() * 10 - 5, Math.random() * 10 - 5, Math.random() * 10 - 5);
      scene.add(cube);
      cubes.push(cube);
    }

    // Set camera position
    camera.position.z = 20;
  }

  // Setup Web Audio API for analyzing audio data
  function setupAudio() {
    audioContext = new (window.AudioContext || window.webkitAudioContext)();
    analyser = audioContext.createAnalyser();
    analyser.fftSize = 256;  // Size of the FFT (frequency bin)
    bufferLength = analyser.frequencyBinCount;
    dataArray = new Uint8Array(bufferLength);

    // Connect the audio source to the analyser and audio context
    audioSource = audioContext.createMediaElementSource(audio);
    audioSource.connect(analyser);
    analyser.connect(audioContext.destination);
  }

  // Function to start the audio playback
  function startAudio(url) {
    audio.src = url;
    audio.play();
    audioContext.resume();  // Resume the audio context after a user interaction (required by browsers)
  }

  // Update the size, rotation, and motion of cubes based on audio intensity
  function animate() {
    requestAnimationFrame(animate);
```

```
// Get the frequency data from the analyser
analyser.getByteFrequencyData(dataArray);

// Calculate the average intensity of the audio
let sum = 0;
for (let i = 0; i < bufferLength; i++) {
 sum += dataArray[i];
}
const average = sum / bufferLength;

// Adjust the size, velocity, and motion of the cubes based on the audio intensity
cubes.forEach((cube, index) => {
 // Size scaling factor based on intensity, within 50% of the initial size
 const scale = 1 + (average / 100) * 0.5;  // 50% scaling range
 cube.scale.set(scale, scale, scale);

 // Move cubes based on audio intensity
 cube.position.x += Math.sin(index + average * 0.01) * objectSpeed;
 cube.position.y += Math.cos(index + average * 0.02) * objectSpeed;
 cube.position.z += Math.sin(index + average * 0.03) * objectSpeed;

 // Rotation effect: Rotate cubes based on audio intensity
 cube.rotation.x += 0.01 + (average / 1000);
 cube.rotation.y += 0.01 + (average / 1000);

 // Bounce effect: Detect edge collision and reverse direction
 if (cube.position.x > screenBounds.xMax || cube.position.x < screenBounds.xMin) {
  objectSpeed = -objectSpeed;
 }
 if (cube.position.y > screenBounds.yMax || cube.position.y < screenBounds.yMin) {
  objectSpeed = -objectSpeed;
 }
 if (cube.position.z > screenBounds.zMax || cube.position.z < screenBounds.zMin) {
  objectSpeed = -objectSpeed;
 }
});
```

```
    // Render the 3D scene
    renderer.render(scene, camera);
  }

  // Handle audio file input
  document.getElementById('audioFile').addEventListener('change', function(event) {
    const file = event.target.files[0];
    if (file) {
      const url = URL.createObjectURL(file);
      setupAudio();  // Set up Web Audio API with new audio
      startAudio(url);  // Start the audio playback
      animate();  // Start the animation loop
    }
  });

  // Initialize the scene
  setupScene();
 </script>
</body>
</html>
```

Key Features:

1. 3D Animation (Cubes as Placeholders):

- We're using **cubes** to represent the objects that change in size and motion based on the audio. You can replace these with more complex models like **GLTF** models if needed.
- The cubes are randomly placed within the scene with random colors.

2. Audio Synchronization:

- We use the **Web Audio API** to process the audio and retrieve the frequency data.
- The AnalyserNode provides the frequency data, which we use to calculate the audio's intensity (the average of the frequency data).
- The average intensity is used to modify the size and motion of the cubes.

3. Size and Motion:

- **Size Scaling**: The cubes' size changes in a range from their original size to 50% larger based on the audio's intensity.
- **Motion**: The cubes move in 3D space, with their position changing in a sinusoidal pattern. The direction and speed are affected by the intensity of the audio.

4. Bouncing Objects:

- Objects will bounce when they reach the boundaries of the scene, defined by the **screenBounds** object. When an object reaches the edge, its motion reverses, creating the "bounce" effect.

5. Web Audio API:

- **AnalyserNode** is used to gather frequency data from the audio file.
- The frequency data is analyzed, and the average intensity is used to control the animation.

How It Works:

1. **Audio Playback**: The user uploads an audio file using the file input. The audio is then played using the HTML audio element, and the Web Audio API is used to analyze the audio in real-time.
2. **Real-Time Animation**: The animate() function updates the size, rotation, and position of the cubes based on the average audio intensity.
3. **Edge Detection & Bouncing**: When the cubes reach the edge of the defined screen bounds, their direction reverses, making them "bounce."

Customization:

- **3D Models**: Replace the cubes with more complex models by using Three.js loaders (like GLTFLoader) to load 3D models in **GLTF** or **OBJ** format.
- **Motion and Scaling**: Adjust the scaling factor and motion dynamics to create different visual effects based on the audio's frequency data.
- **Edge Detection**: You can tweak the screenBounds values or implement more complex edge collision detection if needed.