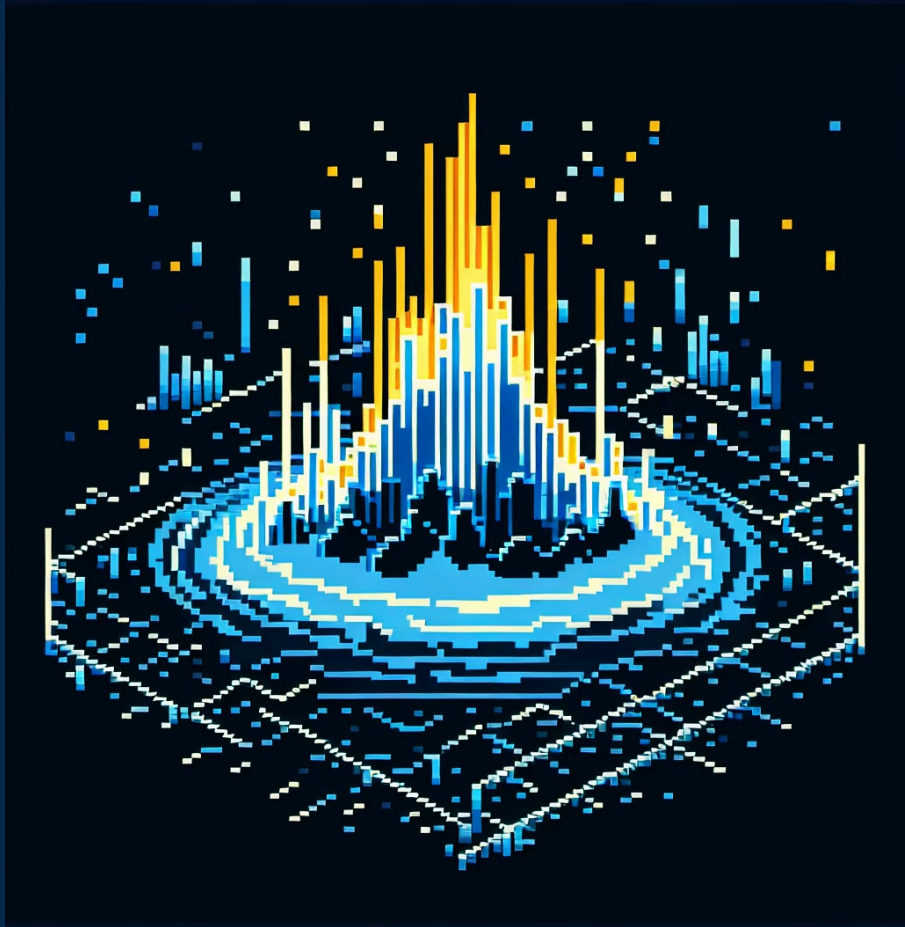
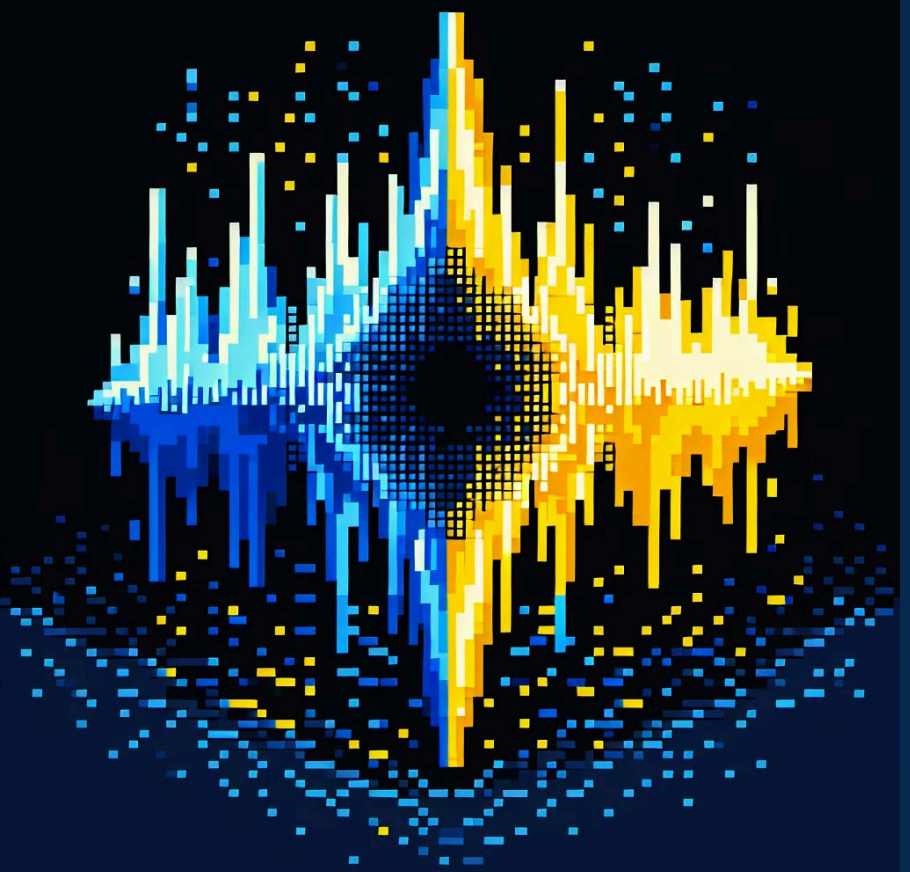
Game Audio and Implementation

9 - Game Audio Basics and Procedural Workflows





Agenda

Audio Fundamentals

- Audio Basics
- Unity Audio System

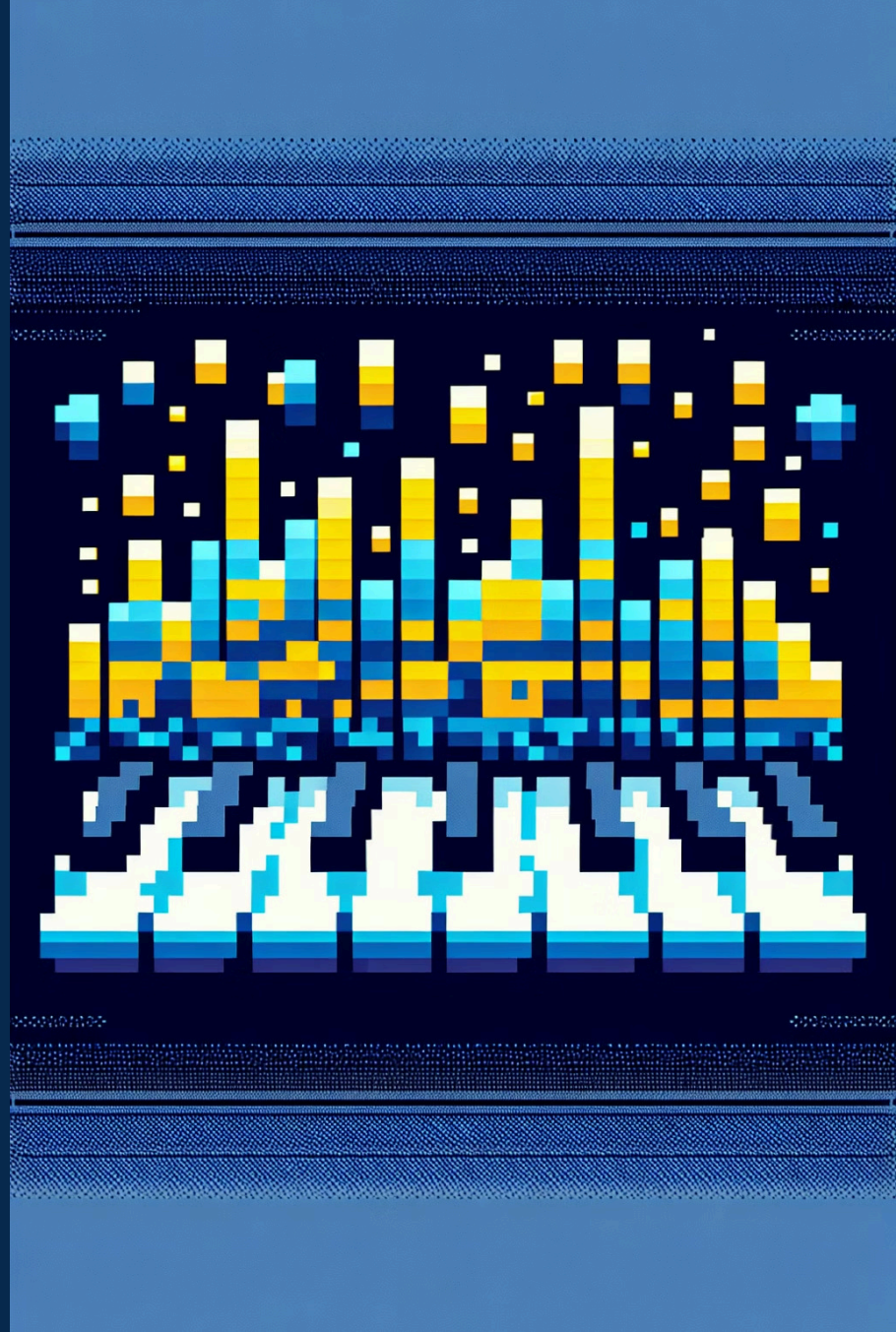
Implementation and Scripting

- Audio Implementation in Unity
- Scripting Audio in Unity

Advanced Concepts

- Advanced Audio Techniques
- Procedural Audio Introduction

Audio Basics



Importance of Audio in Games

Enhances Immersion

Audio plays a crucial role in enhancing immersion and player engagement.

Provides Feedback

Sound effects provide crucial feedback to the player.

Sets Mood

Audio is essential in setting the mood and atmosphere of a game.

Reinforces Narrative

Sound design reinforces the game's narrative and themes.

Types of Game Audio

Sound Effects (SFX)

Sound effects are audio elements in games

Music

Music sets the mood and enhances gameplay experience

Voice Acting

Voice acting brings characters to life in games

Ambient Sound

Ambient sound creates immersive game environments

Sound Effects (SFX)

Sound Effects, commonly known as SFX, are an essential component of game audio. These are short audio clips that accompany specific actions or events within the game, providing feedback.

Some common examples of sound effects in games include:

- Footsteps
- Weapon sounds
- UI feedback
- Environmental sounds (wind, rain, fire)

Music in Games

Music plays a crucial role in setting the mood for games. It can be implemented in various ways:

- **Background music:** Used to establish the atmosphere and emotional tone
- **Static music:** A single track that loops continuously
- **Dynamic music:** Changes based on the current state of the game
- **Adaptive music:** Smoothly transitions between different states

Voice Acting

Voice acting is an element in game audio that brings characters to life and provides narrative context.

Voice acting in games can take various forms, depending on the game's design and budget:

- Full-game voice acting
- Partial voice acting (key scenes only) - Cinematics
- Emotive sounds and Foley (grunts, laughs)

Ambient Sound

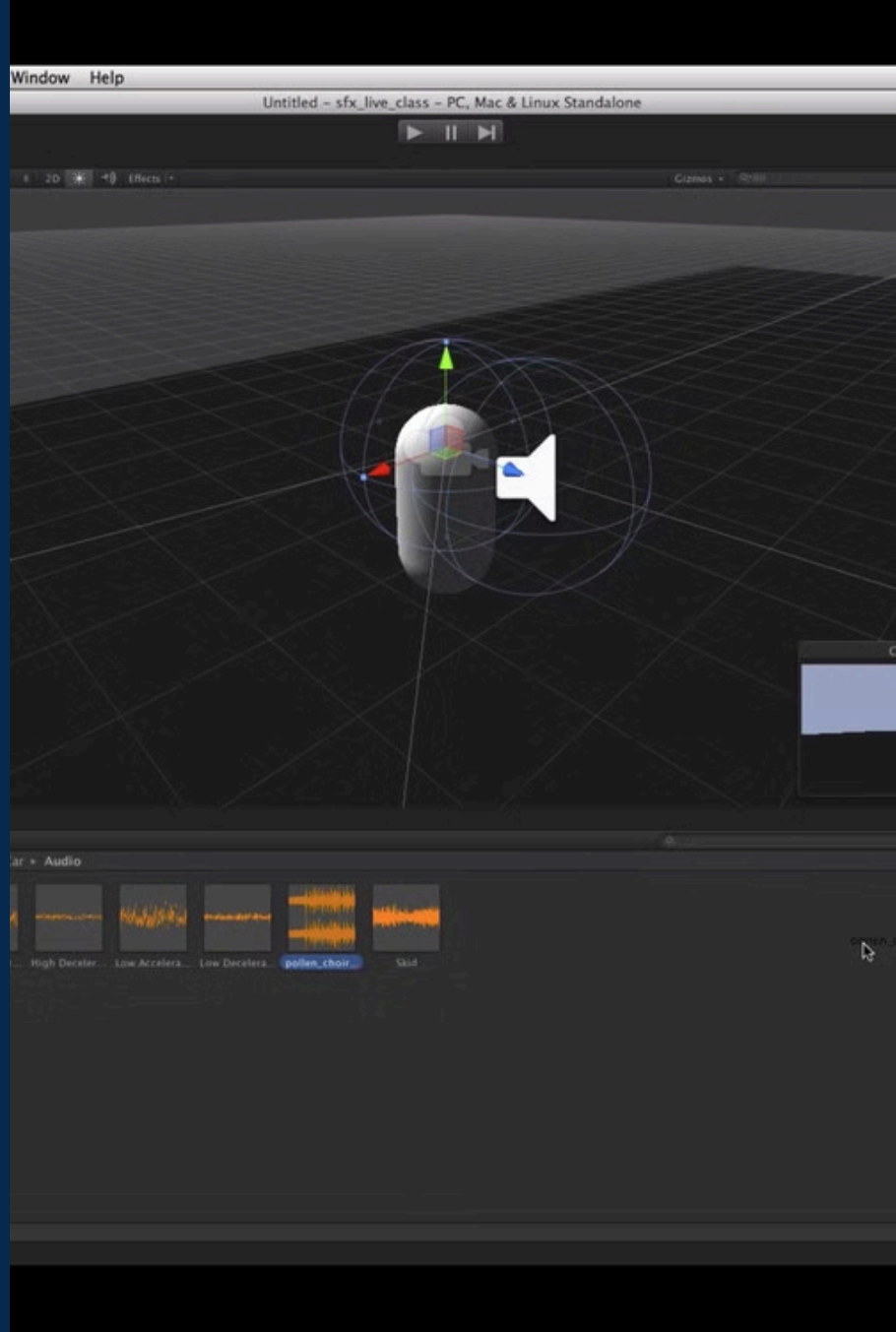
Ambient sound, often called "Background," plays an important role in game audio design:

- Creates a sense of place and atmosphere
- Often looping and subtle

Examples of ambient sound include:

- Room Tones
- Forest sounds
- City traffic
- Spaceship hum
- Etc.

Unity Audio System



Unity Audio System Overview

Emitter-Listener Model

The Unity Audio System is based on emitters and listeners

Key Components

The system consists of four essential components:

1 Audio Source

Emits sound in the game environment

2 Audio Listener

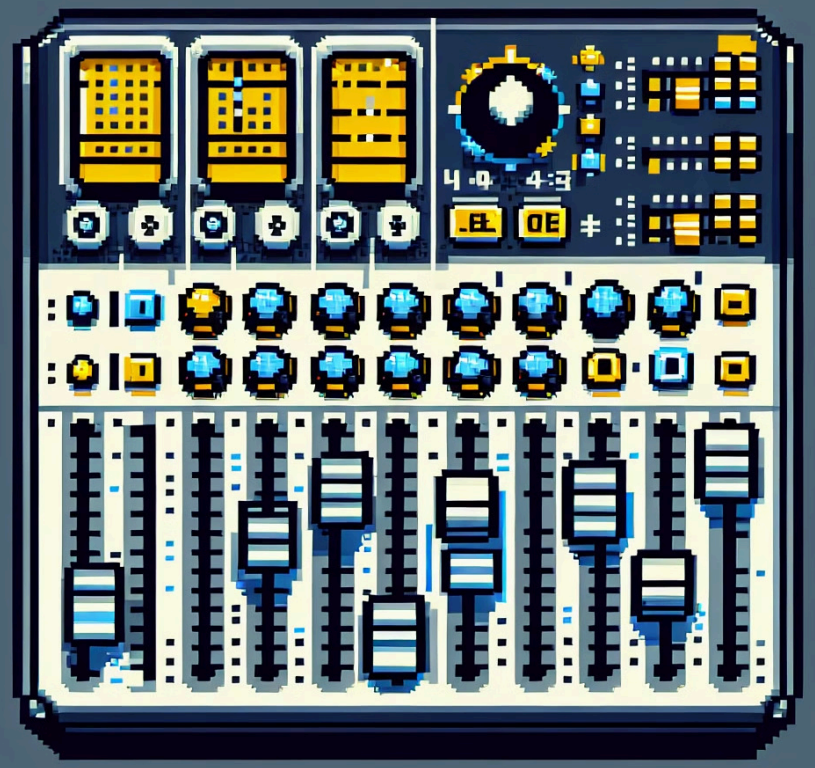
Receives sound in the game environment

3 Audio Clip

Contains the actual sound data

4 Audio Mixer

Manages and mixes multiple audio sources



Audio Source Component

The Audio Source Component is the element in Unity's audio system responsible for playing back Audio Clips within a game environment.

Key properties of the Audio Source Component include:

- Clip: Determines which audio file will be played
- Volume: Controls the loudness of the audio
- Pitch: Adjusts the frequency of the sound
- Spatial Blend: Allows for seamless transition between 2D and 3D audio
- Loop: Enables continuous playback of the audio clip

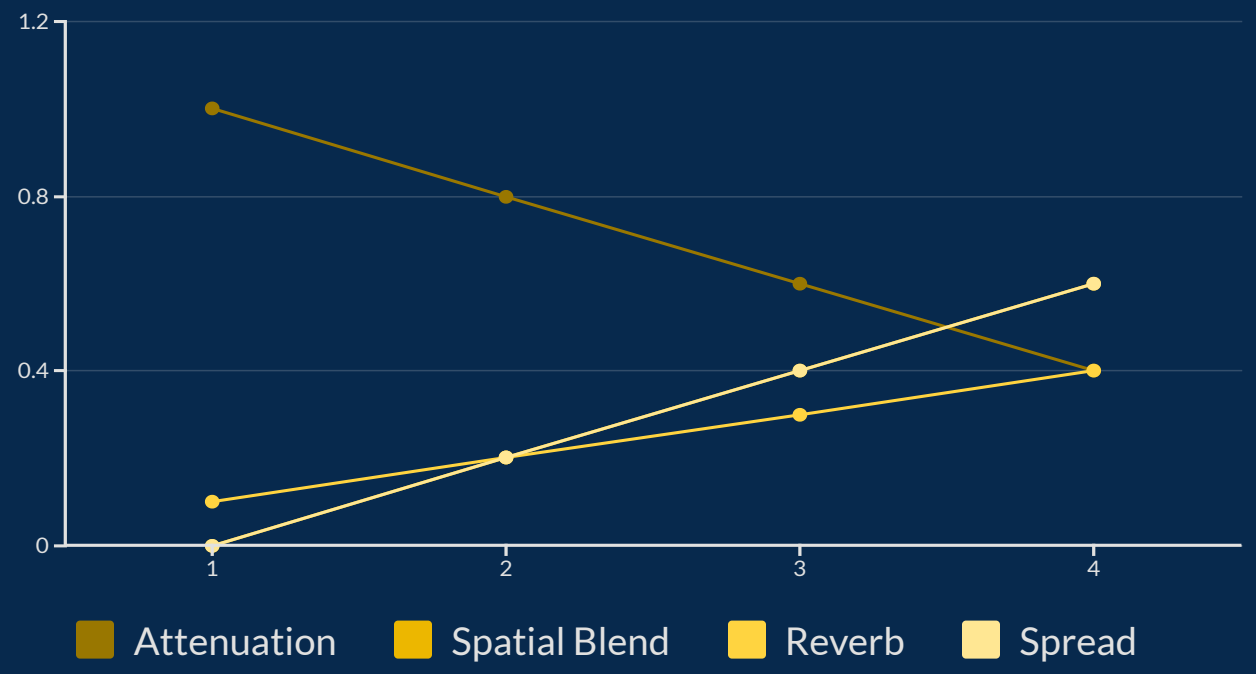
docs.unity3d.com

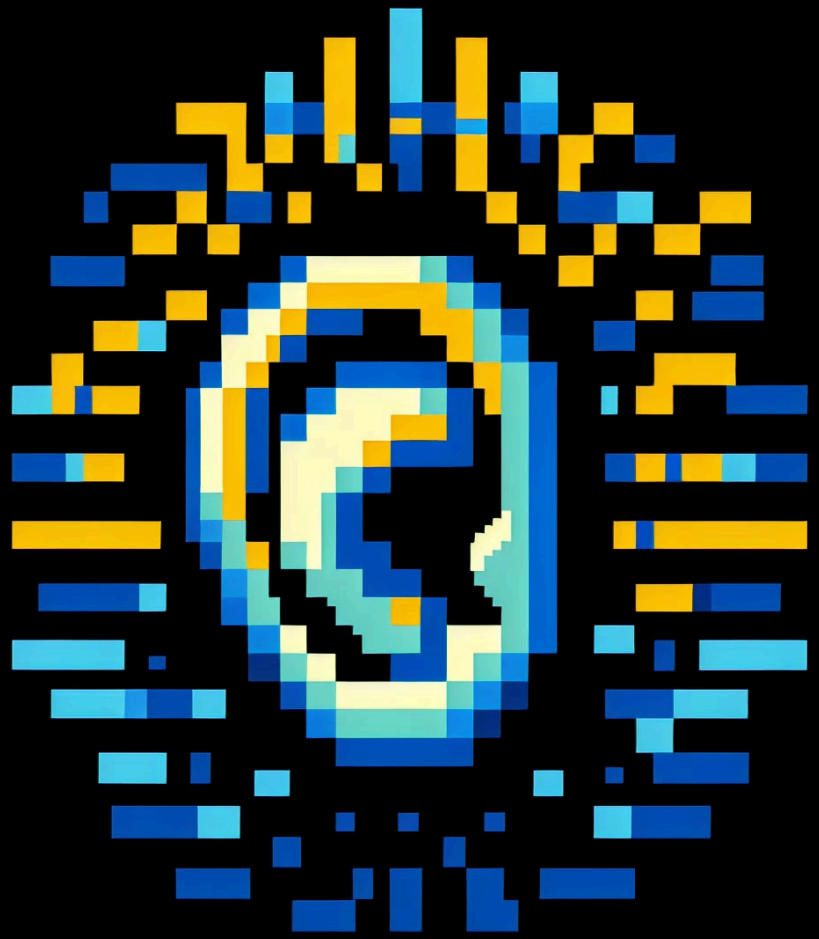
Unity - Manual: Audio Source

The Audio Source plays back an Audio Clip in the sceneA Scene contains the environments and menus of your game. Think of each unique Scene file as ...

Audio Source: Spatial Blend

- Controls how the audio is perceived in 3D space
- 0 = 2D (no spatial effect)
- 1 = 3D (full spatial effect)
  - Can be set anywhere in between for partial effect





Audio Listener Component

- Acts as the "ears" of the scene
- Usually attached to the main camera
- Only one active Audio Listener per scene

docs.unity3d.com

Unity - Manual: Audio Listener

The Audio Listener acts as a microphone-like device. It receives input from any give...

Audio Clip

An Audio Clip is a fundamental component in Unity's audio system that contains the actual audio data. It serves as the source of sound that can be played through Audio Sources in your game.

The supported formats for Audio Clips include:

- WAV
- MP3
- OGG
- AIFF

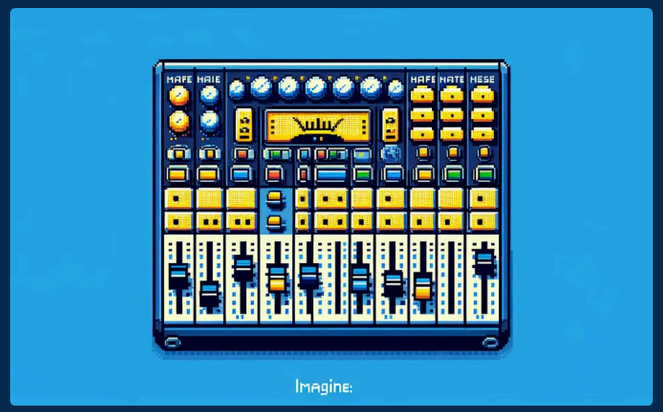
When choosing a format for your Audio Clips, it's crucial to consider both file size and quality.

docs.unity3d.com

Unity - Manual: Audio Clip

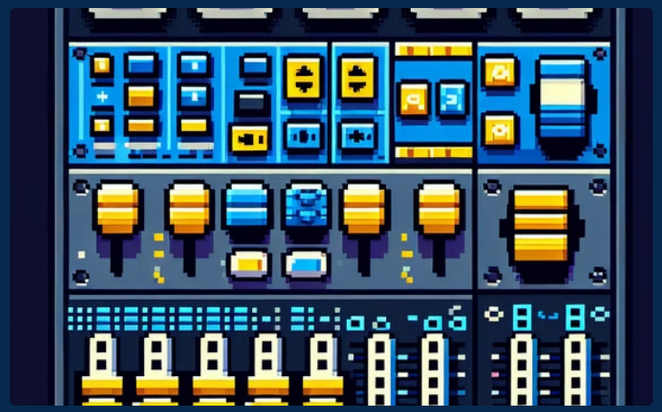
Audio clips contain the audio data used by Audio SourcesA component which plays back an Audio Clip in the scene to an audio listener or through an audi...

Audio Mixer



Structured Audio Management

The Audio Mixer allows developers to create audio groups for things like SFX, Music, and Dialogue, providing a structured approach to organizing different types of sounds.



Applying Audio Effects

The Audio Mixer enables developers to apply effects to audio groups, enhancing the overall audio experience.

Dynamic Volume Control

The Audio Mixer allows for real-time adjustments to audio levels based on gameplay events or player preferences, ensuring each component is heard clearly.

docs.unity3d.com

Unity - Manual: Audio Mixer

When you select Add you can select from a list of effects that can be applied to the group.

Game Audio Implementation - Unity

Basic Audio Implementation Steps

1

1. Import Audio Files

Import audio files into Unity

2

2. Add Audio Source

Add Audio Source component to game object(s)

3

3. Assign Audio Clip

Assign Audio Clip to Audio Source in Editor or via Script

4

4. Adjust Properties

Adjust Audio Source properties

5

5. Play Audio

Play audio through script or component settings

3D Sound in Unity

- Use **Spatial Blend** setting in Audio Source to create 3D sound effects
- Adjust 3D Sound Settings for realistic audio:
  - **Doppler Level:** Controls pitch change based on motion
  - **Volume Rolloff:** Determines how volume decreases with distance
  - **Min/Max Distance:** Sets the audible range of the sound
- Many sound settings can be represented by an Animation Curve

docs.unity3d.com

Unity - Manual: Audio Spatializer SDK

The audio spatializer SDK provides controls to change the way your application transmits audio from an audio sourceA component which plays...

Audio Mixer: Practical Use

1

Create and Organize

1. Create Audio Mixer asset
2. Add groups (e.g., SFX, Music, Dialogue)

2

Assign and Enhance

3. Assign Audio Sources to groups
4. Add effects to groups (e.g., reverb, EQ)

3

Control and Adjust

5. Control group volumes via script or UI

Scripting Audio in Unity

Explore the power of programming to control and manipulate audio elements in your game.

Playing Audio with C#: Concept

- Access the AudioSource component on game objects
- Utilize methods such as Play(), Stop(), and Pause() to control audio playback
- Capability to play one-shot sounds or looping audio as needed
- Dynamically control various audio properties including volume and pitch

Playing Audio with C#: Code Example

Here's a practical example of how to play audio in Unity using C#:

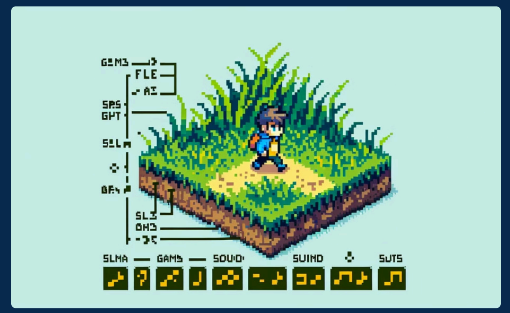
```
using UnityEngine;

public class AudioPlayer : MonoBehaviour
{
    public AudioSource audioSource;
    public AudioClip soundEffect;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    void PlaySound()
    {
        audioSource.PlayOneShot(soundEffect);
    }
}
```

This code demonstrates a simple AudioPlayer class that can be attached to a GameObject in Unity. It uses an AudioSource component to play a sound effect. The Start() method gets the AudioSource component, and the PlaySound() method plays the specified sound effect using the PlayOneShot() function.

Terrain-based Footstep Sounds: Concept

Detect Terrain Texture

The game engine continuously monitors the surface type the player is walking on.

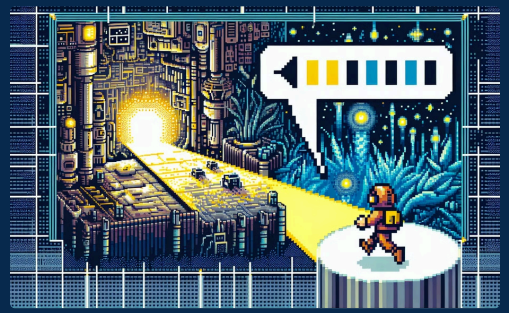


Select Appropriate Sounds

Once the terrain is identified, the system chooses a corresponding sound effect that matches the surface material.

Play Varied Sounds

To avoid repetition and increase realism, the system plays a randomly selected variation of the appropriate footstep sound.



Enhance Immersion

This technique significantly improves the player's sense of presence in the game world by offering auditory cues that match their visual surroundings.

Terrain-based Footstep Sounds: Code Example (Part 1)

1. Code structure: **using UnityEngine;** statement and **public class FootstepAudio : MonoBehaviour** declaration
2. Audio clip arrays: **public AudioClip[] grassSteps, woodSteps, stoneSteps;** for different terrain types
3. Private variables: **private AudioSource audioSource;** and **private Terrain terrain;** for audio playback and terrain detection
4. **Start()** method: Initializes **audioSource** and **terrain** variables
5. **Update()** method: Checks for spacebar input to simulate a step and calls **PlayFootstep()** method

Terrain-based Footstep Sounds: Code Example (Part 2)

```
void PlayFootstep() {
    int terrainTextureIndex = GetMainTexture(transform.position);
    AudioClip[] currentSteps;
    switch (terrainTextureIndex) {
        case 0:
            currentSteps = grassSteps;
            break;
        case 1:
            currentSteps = woodSteps;
            break;
        case 2:
            currentSteps = stoneSteps;
            break;
        default:
            currentSteps = grassSteps;
            break;
    }
    AudioClip stepToPlay = currentSteps[Random.Range(0, currentSteps.Length)];
    audioSource.PlayOneShot(stepToPlay);
}
```

- **Switch statement:** Determines the appropriate footstep sound based on terrain texture index
- **Random selection:** Chooses a random audio clip from the selected terrain's footstep array
- **PlayOneShot:** Plays the selected footstep sound without interrupting other audio

docs.unity3d.com

Unity - Scripting API: AudioSource.PlayOneShot

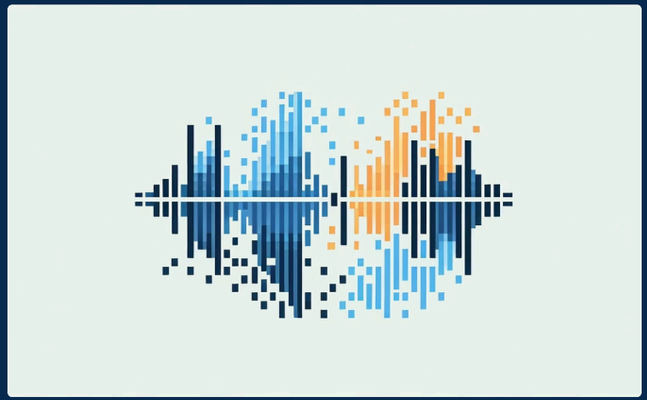
Thank you for helping us improve the quality of Unity Documentation. Although we cannot accept all submissions, we do read each suggested...

Unity Learn

Switch Statements - Unity Learn

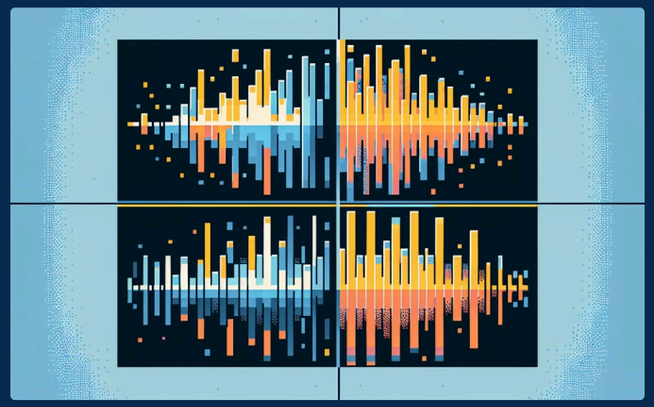
Switch statements act like streamline conditionals. They are useful for when you want to compare a single variable against a series of constants. In this...

Smooth Audio Transitions: A Seamless Experience



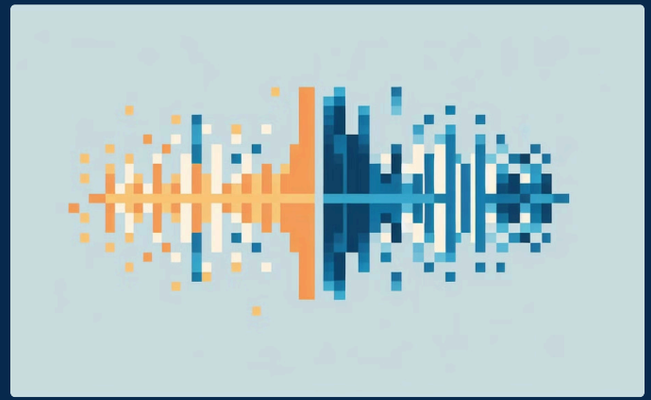
Gradual Volume Changes

Smooth audio transitions involve gradually fading out one audio track while simultaneously fading in another, creating a seamless transition between different audio elements.



Precise Timing with Coroutines

Developers often utilize coroutines for time-based operations, allowing for precise control over the timing of volume changes and ensuring a fluid transition between audio tracks.



Responsive Soundscapes

By implementing smooth audio transitions, game developers can create more dynamic and responsive soundscapes that adapt to changes in gameplay, environment, or narrative without jarring interruptions.

Unity Learn

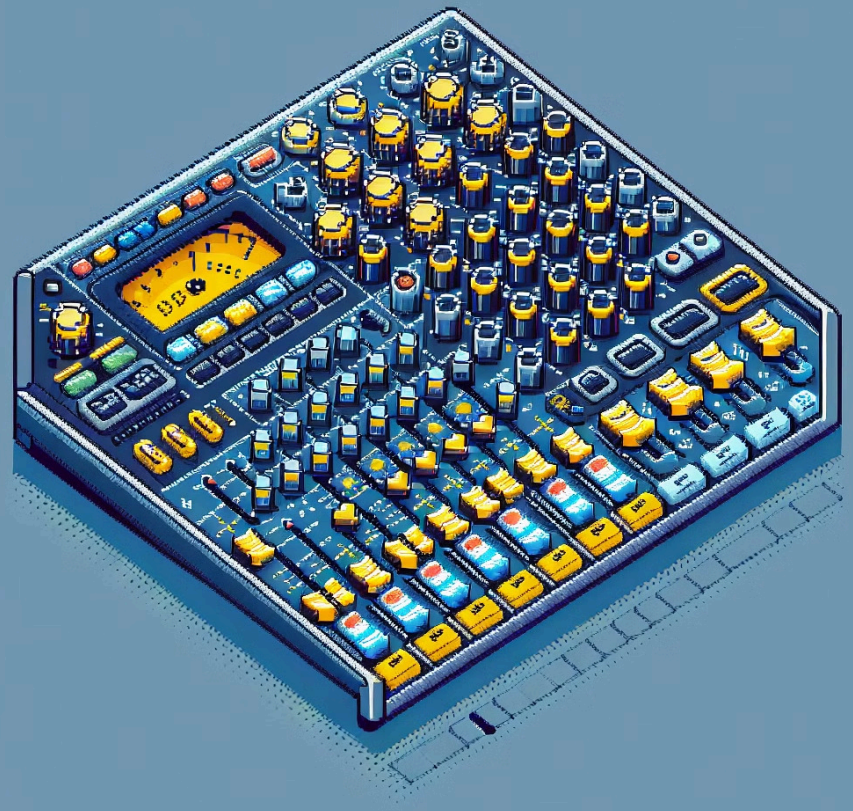
Coroutines - Unity Learn

Coroutines are a way of performing an operation over time instead of instantly. They can be useful, but there are some important things you need ...

Smooth Audio Transition: Code Example

1. Code structure: **MusicTransition** class inheriting from MonoBehaviour
2. Public variables: **source1**, **source2** (AudioSource), **transitionTime** (float)
3. Public method: **TransitionMusic()** to initiate transition
4. Private coroutine: **Transition()** for smooth volume adjustment
5. Transition logic:
   - Start second source
   - Gradually decrease volume of first source
   - Simultaneously increase volume of second source
   - Stop first source after transition completes
6. Uses **Mathf.Lerp()** for smooth volume interpolation
7. Transition duration controlled by **transitionTime** variable

```
using UnityEngine;
using System.Collections;

public class MusicTransition : MonoBehaviour
{
    public AudioSource source1;
    public AudioSource source2;
    public float transitionTime = 2.0f;

    public void TransitionMusic()
    {
        StartCoroutine(Transition());
    }

    private IEnumerator Transition()
    {
        float t = 0;
        source2.Play();
        while (t < transitionTime)
        {
            t += Time.deltaTime;
            source1.volume = Mathf.Lerp(1, 0, t / transitionTime);
            source2.volume = Mathf.Lerp(0, 1, t / transitionTime);
            yield return null;
        }
        source1.Stop();
    }
}
```



Scripting with Audio Mixer: Concept

- Access the Audio Mixer through scripts for dynamic control
- Modify group volumes dynamically during gameplay
- Utilize logarithmic scale for natural-sounding volume changes
- Enable runtime audio adjustments (e.g., volume sliders in options menu)

Scripting with Audio Mixer: Code Example

Here's a practical code example demonstrating how to script with the Audio Mixer in Unity:

```
using UnityEngine;
using UnityEngine.Audio;

public class VolumeControl : MonoBehaviour
{
    public AudioMixer audioMixer;

    public void SetMasterVolume(float volume)
    {
        audioMixer.SetFloat("MasterVolume", Mathf.Log10(volume) * 20);
    }

    public void SetSFXVolume(float volume)
    {
        audioMixer.SetFloat("SFXVolume", Mathf.Log10(volume) * 20);
    }
}
```

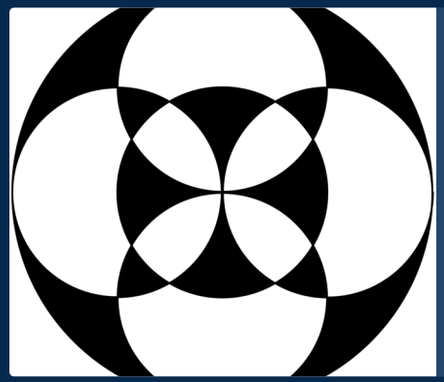
This code snippet shows a **VolumeControl** class that interacts with an AudioMixer to adjust volume levels. It includes methods for setting both the master volume and SFX volume using a logarithmic scale for more natural-sounding volume changes.

Advanced Audio Techniques

Exploring cutting-edge methods in game audio design and implementation

Ambisonics

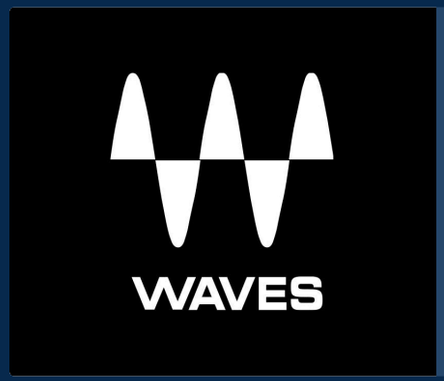
- **Full-sphere surround sound technique** for immersive audio experiences
- Captures and reproduces a complete **3D sound field**
- **Ideal for VR and 360° video applications**, enhancing realism
- **Supported in Unity** for certain audio formats, enabling game developers to implement advanced audio



Wikipedia

Ambisonics

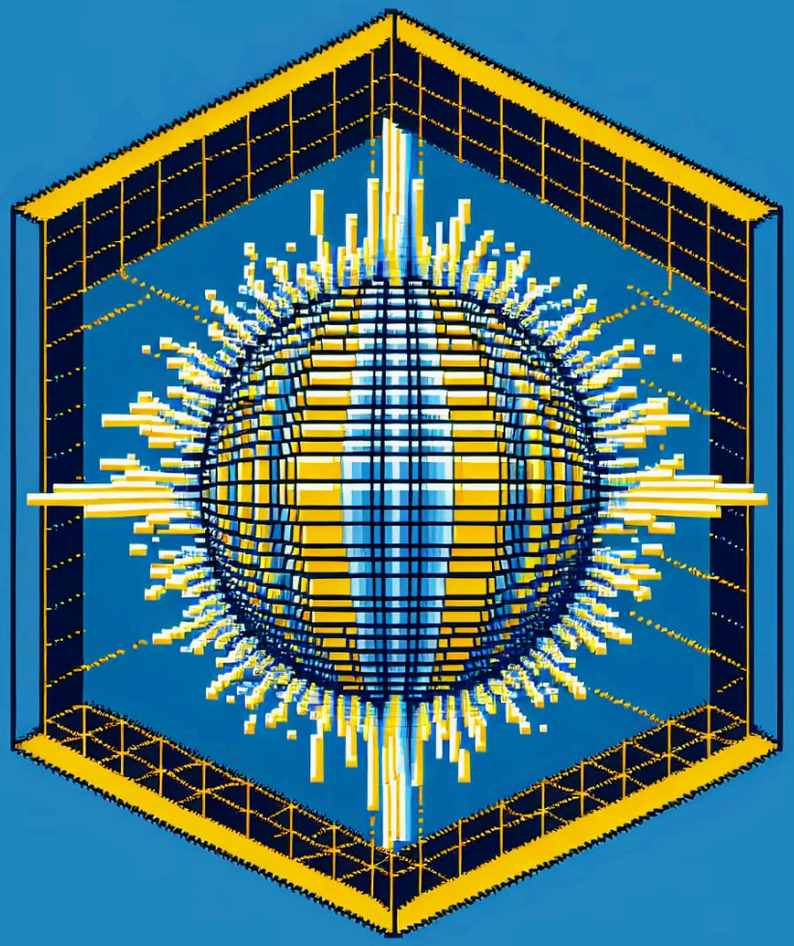
Ambisonics is a full-sphere surround sound format: in addition to the horizontal plane,...



waves.com

Ambisonics Explained: A Guide for ...

Need to mix audio for 360 videos or VR projects and not sure where to start? Get ...



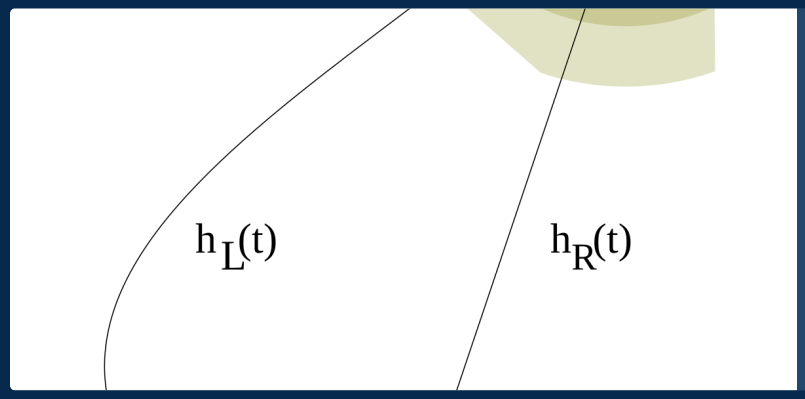
Binaural Audio

HRTF (Head-Related Transfer Function)

Binaural audio utilizes HRTF to create a realistic soundscape that closely mimics natural hearing processes.

Headphone Audio Immersion

Binaural audio is particularly effective for headphone users, providing an unparalleled level of audio immersion.



Wikipedia

Head-related transfer function

A head-related transfer function (HRTF) is a response that characterizes how an ear receives a sound from a point in space. As sound strikes the listener,...

YouTube

What Is HRTF? (Head Related Transfer Functions)

Kaushik Sunder, Embody's Principal Acoustic Researcher, explains a breakdown of what an HRTF (Head Related Transfer Function) means and...

YouTube

Hellblade Development Diary 15: Binaural Audio Tests

Hellblade Development Diary 15: Binaural Audio Tests. Wear headphones to hear binaural sound. Hellblade Senua's Sacrifice is releasing August 8t...

Facebook

Experience voices & auditory hallucinations in #Hellblade

See how the sound of psychosis is being crafted in Hellblade: Senua's Sacrifice #PS4 #PCGaming

Dynamic Music Systems

Adaptive Music

Dynamic music systems allow game music to adapt to player actions and gameplay events in real-time, creating a more immersive experience.

Location-based Music

The music can change based on factors like the player's location, game state, or even their health and stress levels.

Implementation Approaches

Dynamic music is typically achieved through techniques like multi-track audio or generative music, allowing for a more responsive and interactive soundscape.

YouTube

Introduction to Adaptive Music in Games (Listening to Videogame...

A primer on adaptive music (a.k.a. dynamic music) in video games, talking about it's purpose and a few techniques of implementation. Adaptive music ...

Audio Occlusion and Obstruction

- Simulates how sound is affected by objects in the environment
- Occlusion: Sound passing through objects
- Obstruction: Sound going around objects
- Can be implemented using raycasts and audio filters

audiokinetic

ak audiokinetic

Documentation | Audiokinetic

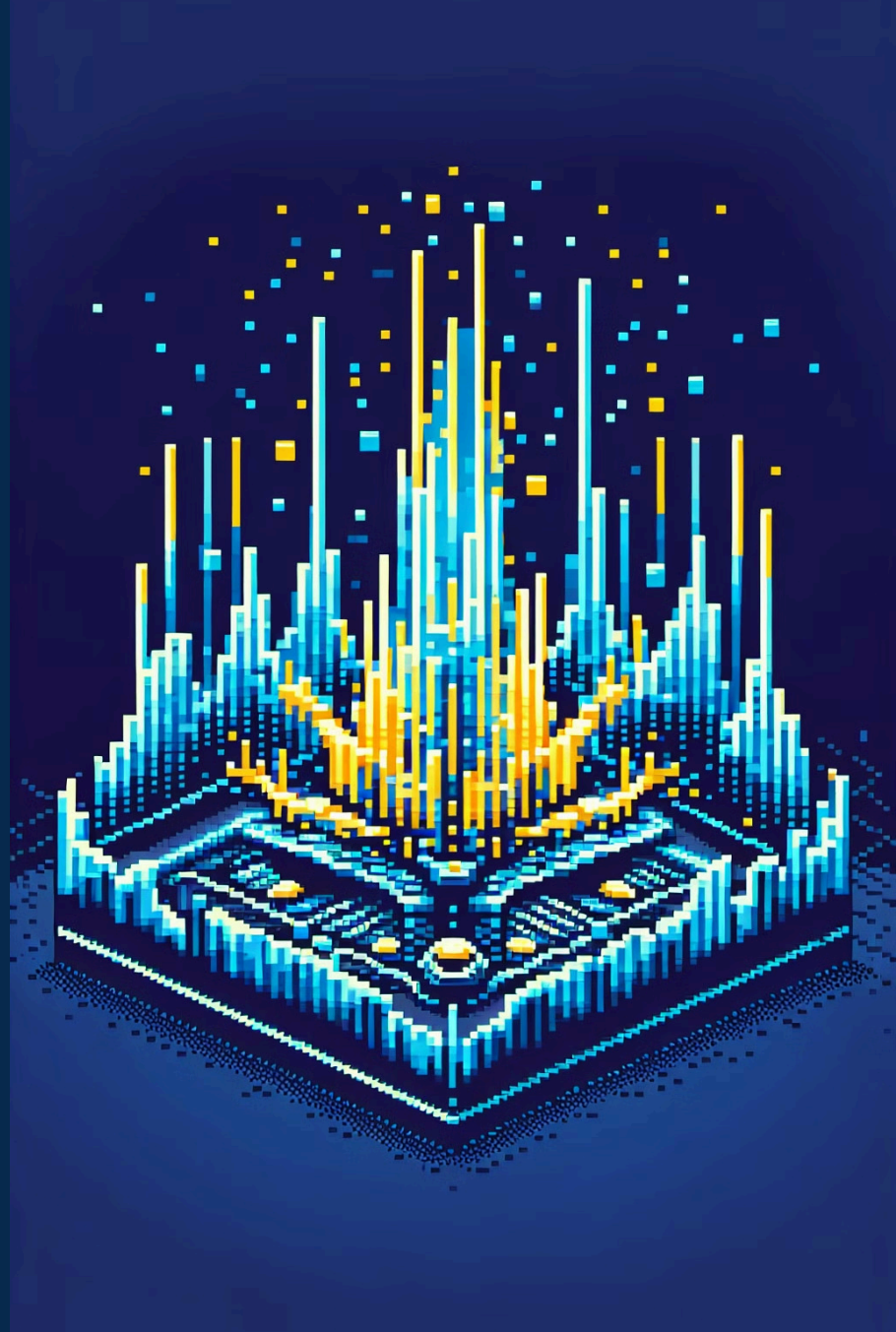
Audiokinetic - The global leader in interactive audio technology.

Procedural Audio: Introduction

An introduction to the concept of procedural audio in game development, exploring its definition, applications, and potential impact on game sound design.

What is Procedural Audio?

- Audio generated or manipulated in real-time
- Not pre-recorded, but created algorithmically[1]
- Can be responsive to game state and player actions
- Reduces memory usage for audio assets



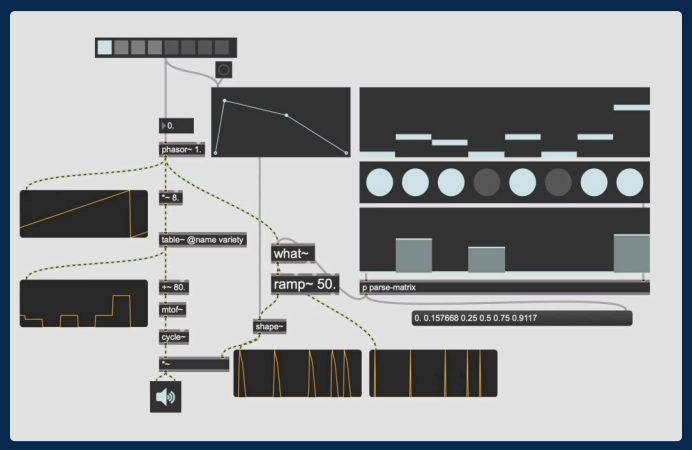
Tools for Procedural Audio

Pure Data

Open-source visual programming language for audio synthesis

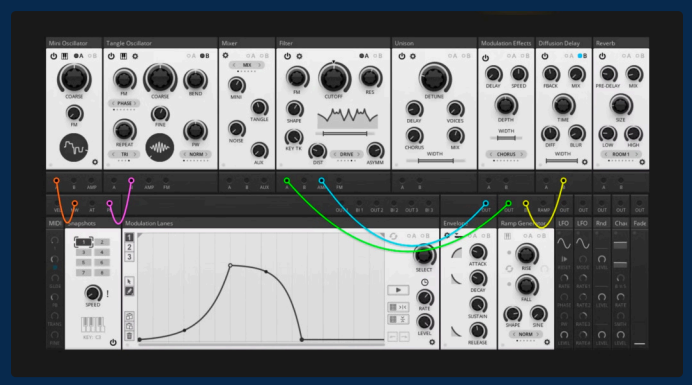
Max/MSP

Visual programming language for music and multimedia



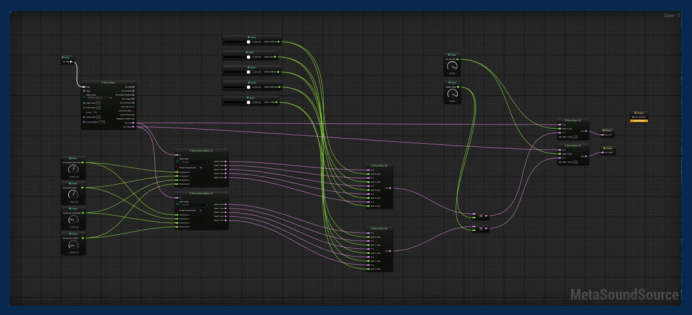
Reaktor

Modular sound design and synthesis environment



Unreal MetaSounds

Integrated audio tools within the Unreal game engine

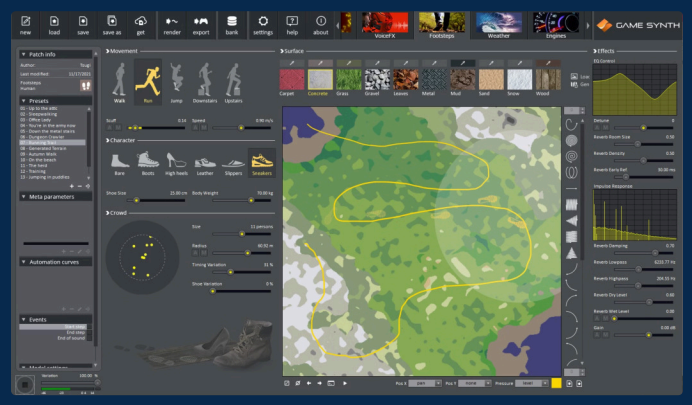


Custom Scripts (C#/C++/Etc.)

Tailored audio synthesis through programming

Tsugi GameSynth

A software sweet for procedural and algorithmic sound effects



There are MANY more tools out there…

Benefits of Procedural Audio

Infinite Variations

Infinite variations of sounds

Responsive to Gameplay

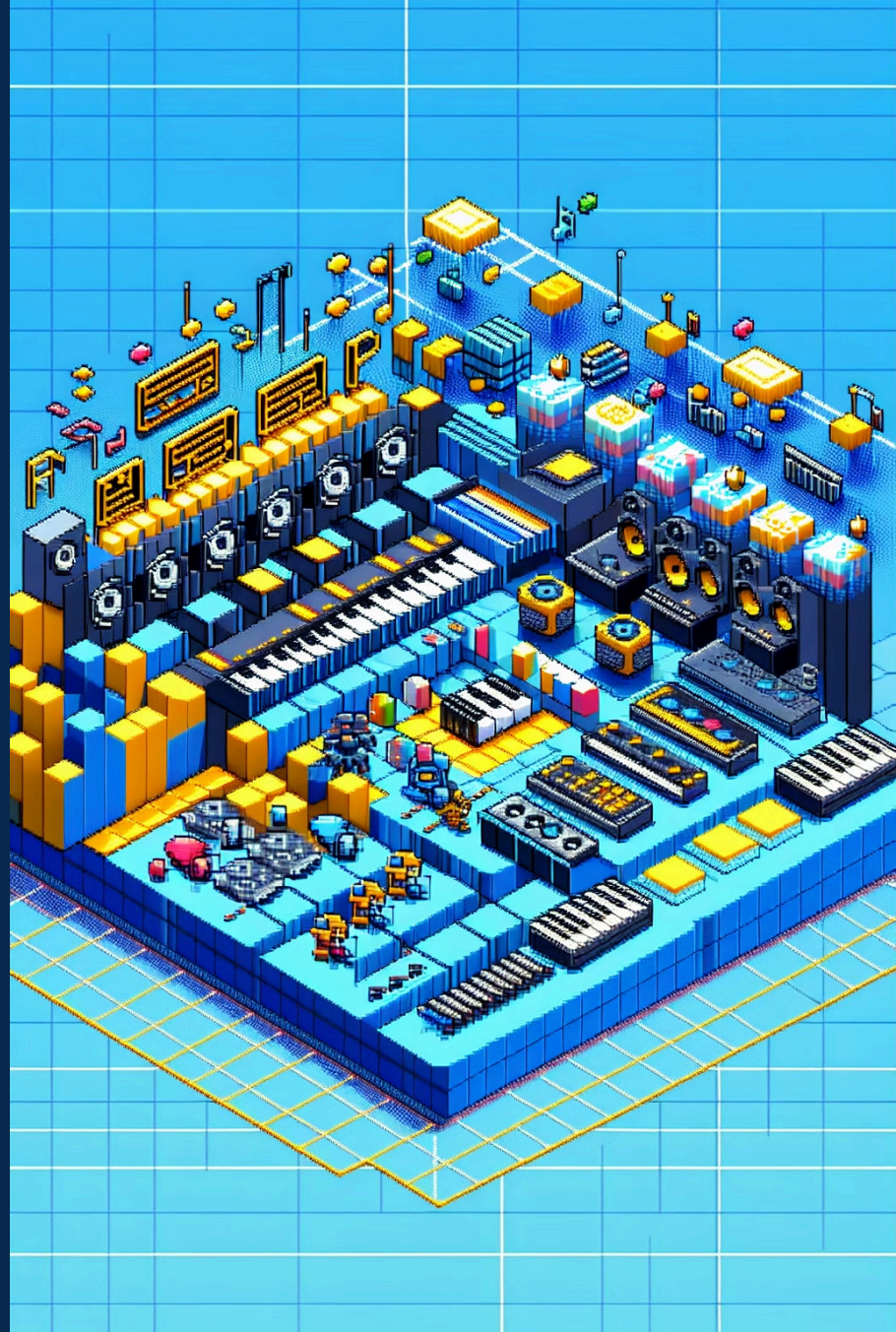
Responsive to gameplay parameters

Reduced Memory

Reduced memory footprint

Smaller File Sizes

Potentially smaller file sizes for games



Challenges of Procedural Audio

1. **Resource Intensive**

   Can be CPU-intensive, potentially impacting game performance

2. **Skill Set Requirements**

   Requires different skill set (programming + sound design), increasing the complexity of audio development

3. **Quality Concerns**

   May lack the nuance of carefully crafted recorded audio, potentially affecting the overall audio experience

4. **Evolving Landscape**

   Tools and workflows still evolving, making it challenging to establish standardized practices

YouTube

So You Wanna Make Games?? | Episode 8: Sound Design

What makes a good sound? In games, strong sound design effectively communicates what's happening to and around you, provides feedback o...

Final Project - Game Audio + Implementation

Integrating immersive spatial audio and responsive sound effects are crucial elements of a captivating game experience. By strategically positioning audio sources and designing audio systems that react to player actions, developers can enhance the sense of realism and player immersion.

- Implement **at least 2 distinct audio/music effects** that contribute to the game's atmosphere and feedback
- Ensure **proper 3D audio spatialization** for sound sources, creating a convincing sense of depth and directionality, where appropriate
- (Optional) Develop **reactive audio systems** that dynamically adjust volume, pitch, or other properties based on player movements or interactions

Lab Time

Now, practice implementing some of these audio techniques for yourself