ReAL Sound: Outline of a Reusable Audification Library to Improve Game Accessibility for the Visually Impaired

Meir Arani
Kyushu University
Graduate School of Design

January 9, 2025

Abstract

In a world of ever-increasing software complexity, there has been a growing demand for interoperable, reusable technologies that function in many problem domains. This is especially true in the world of game development, where tools, structures, and architectures often change from title to title. At the same time, the specificity of software user needs has also grown immensely, bringing an increased demand for advanced accessibility tools with it. To address these trends, we propose ReAL Sound: the ReUsable Audification Library, which abstracts the creation of visual accessibility technology for impaired persons in the realm of game design using computer vision and machine learning techniques.

Contents

Chapter 1

Introduction

1.1 Game Development

1.2 Accessibility

1.3 Computer Vision

1.4 Machine Learning

Chapter 2

Literature Review

Chapter 3

Real Sound

Introduction In this chapter, I propose and outline the ***ReAL Sound*** system: a **Re**-usable **A**udification **L**ibrary that abstracts the design of accessibility features for the visually impaired via the use of computer vision, spatial audio, and audification techniques.

3.1 Proposal

Introduction In this section, I propose ReAL Sound and justify its novelty and utility based on the literature review performed in the previous chapter.

What is ReAL Sound?

ReAL Sound is a software concept that generalizes the creation of visual accessibility features for games. Using computer vision techniques and modern spatial audio technology, ReAL Sound aims to abstract the process of moment-to-moment analysis of a game's state as well as the generation of 3D spatial audio objects. Through this abstraction, implementors of /rs (as distinguished from *users* of ReAL Sound—the visually impaired persons benefitting from the features) can sidestep many of the language, platform, and architecture specific headaches involved in the creation of game accessibility features, especially in the case of after-market games.

ReAL Sound achieves this generalization by abstracting the process into a few core steps for any given target game:

Planning The creation of simple 'rules' and 'definitions' which accurately describe the game.

Training The act of preparing computer vision techniques based on the **Planning** stage to analyze the game's current state data.

Design The creation of simple 'rules' and 'definitions' for translating the data analyzed in the **Training** phase into spatial audio objects.

Execution The real-time marriage of the previous stages. Uses a computer vision model **trained** on the **planned** rules which translates the game's real-time visual data into spatial audio objects as **designed** by the implementor.

With ReAL Sound, a implementor with only a moderate amount of technical knowledge could feasibly add accessibility features to any target game—all with zero knowledge of the game's underlying source code or architecture.

Why is ReAL Sound?

As discussed in the previous chapter, there is a well-established and ever-growing need for software interoperability and platform agnostic support, as well as a growing demand for improved software accessibility. Implementors of game accessibility features already face numerous game-specific challenges in the design process—as each game's bespoke nature demands equally unique accessibility design. The added barriers of specific architecture, engine, and language implementation have especially stymied the development of accessibility features in the gaming industry—where many development tools target a single architecture or operating system, are company specific, and are often used on a per-game basis.

ReAL Sound significantly streamlines these problems through abstraction—significantly simplifying this troubled implementation process. As a consequence, greater flexibility and more time are handed to feature designers—who are now better quipped to craft higher quality features at a faster clip.

Moreover, the generalization process requires zero knowledge of the game's actual codebase, meaning games can be modified in the after-market by dedicated enthusiasts. As a consequence, the fan-driven movement of retrofitting of older titles with modern accessibility features becomes easier—allowing visually impaired persons access to a wealth of classic titles while costing modern developers zero resources. This is becoming especially crucial in modern times, as some estimates show that over 85% of games are functionally abandoned—with no publisher or developer entity maintaining ownership [1]. This fact effectively renders the vast majority of published games, *'abandonware'* with little hope of official improvements by original developers [2].

How is ReAL Sound?

ReAL Sound is made possible through modern machine learning, computer vision, and spatial audio technologies. Using the latest computer vision techniques through libraries like OpenCV[3], even novice developers can implement object detection algorithms—which analyze visual input and return semantic information about the its contents—with ease.

The Modern AI Boom On top of this, the recent 'boom' in AI technologies[4][5] has brought an equally intense focus to the development, improvement, and democratization of AI tools and systems[6][7]. Ecosystems like *HuggingFace*[8] and

projects like Google's *TensorFlow*[9] have dramatically changed the landscape of AI technology development—trivializing many tasks considered inaccessible to the common developer just a few years ago.

Agnostic Vision Requirements Considering the ever-changing state of modern AI and computer vision technology, ReAL Sound does not specifically call for any one particular solution for its **Training** or **Execution** stages. Instead, ReAL Sound only requires the *successful* real-time identification of in-game objects *as defined by the implementor*. The means by which this goal is accomplished is left up to the implementor, and may be achieved by any available means.

Consequently, ReAL Sound does not technically demand the usage of AI *at all.* Many well-proven pre-AI techniques—such as *template matching*[10] and *corner detection*[11]—have proven to be successful in many simple use-cases. To demonstrate this, I will present an implementation of ReAL Sound using the one of the original corner detection algorithms—the *Harris corner detector*[12], originally developed in 1986—in a later chapter.

Spatial Audio Running alongside the AI boom has been a comparably smaller (yet not insignificant) boom of 3D and spatial audio technologies. As discussed in the previous chapter, many advances have been made in sonification, audification, and spatial audio techniques in recent years. This trend has given end-users an abundance of choice for low-cost, high quality audio playback devices with native spatial audio support–from in-ear monitors like the the Apple *AirPods*[13] to even computer monitors from major manufacturers like Dell[14]. Future patents promise even greater advancements through techniques like automatic HRTF adjustments specific to the user's physiology[15].

Many new software development libraries been developed to address this boom in consumer demand. Fan driven efforts like the *Spatial_Audio_Framework*[16] as well as company-produced projects such as Valve Software's Steam Audio[17].

3.2 Structural Outline

3.2.1 Overview

Anatomy of a Game

Introduction Here, I construct a semi-formal abstraction of video games using mathematics notation. Later, I will use these constructs to more rigidly define concepts.

Game A Game, lets call it M (for *meta*) can be modeled as the combination of three sets: a collection of meta attributes A, a series of in-game states, G, and a group of entities, I:

$$M = \{A, G, I\}$$

Meta Attributes The game's meta-attributes M_A can be imagined as data that is preserved over an entire game session. This data can persist between state transitions and usually describes overarching information such as current playtime or the currently active game level.

Game State Each state, G_S (referred to interchangeably as *game state* hereafter) is comprised of four components—internal state-attributes A, internal state-logic L, as well as conditions C for inter-state transitions T.

$$G_S = \{A, L, C, T\}$$

State-Attributes A state's attributes A act similarly to the game's overarching meta attributes M_A, but on a per-state context. These attributes may store information such as the current time spent within the state or other data describing the state in specific. State data is lost upon transitioning to a different state.

Internal Logic The state's internal logic L defines all the behaviors and activities that are carried out *within* the state. For example, a racing game may contain a **start** state—which contains logic for playing a special sound effect when the race is started, or routines to visually display the text *'START!'* on-screen.

In abstract, L can be imagined as a series of conditional requirements L_C that yield specific actions L_A:

$$L : L_C \longrightarrow L_A$$

Conditions A state also contains conditional rules defining when to exit this state and transition to another. C defines these rules as a set of boolean statements—which evaluate to either `true` or `false`. Take, for example, a sports game which transitions from a **match** state to a **finish** state after ten points are scored in the match. Imagine that the current match score is stored as an integer value through the state attribute S:

$$S : [0, 10], S \in G_A$$

Then there is a condition within C, lets call it C_S, which might look like this:

$$C_S = S \geq 10$$

When C_S evaluates to true (i.e., when the match score has reached ten points), the state will transition transition to the **finish** state.

Transitions The transition function T defines the mapping of "where" or "how" to transition after a condition in C has evaluated to `true`. It maps a conditional statement C_S into the next game state G_N to transition to:

$$T : C_S \longrightarrow G_N$$

This function can be generalized by instead taking a specific state G_S and a generic condition C as input:

$$T : G_S \times C \longrightarrow G_N$$

Entities Entities I are the objects which constitute a game's internal structure—the 'actors' of the game. An entity can be anything contained within a game, such as the player, enemies, items, level construction, etc. Usually, an entity's behavior can be described using verbs—in either an active (*Mario jumps on the platform*) or passive (*The platform was jumped on by Mario*) sense.

Each entity has attributes A as well a collection of states E:

$$I = \{A, E\}$$

Entity Attributes Similar to the previous definitions, entity attributes describe entity-specific attributes. For example, an entity's location point P in a 2D game's world space might be modeled as:

$$P = \{(x, y) : x, y \in \mathbf{R}^2\}$$

Similar information, like a enemy's current health, an item's purchase price, or a bomb's damage radius are also considered Entity attributes.

Entity States An entity state, then, describes an entity's different states of being. A player entity might be able to **jump**, while a spell entity might be **cast** or an enemy entity might be **killed**, to name a few examples.

Some examples may seem unintuitive, but still work within this framework. A game's menu system, for instance can also be considered an entity. In this case, the act of clicking on the menu—triggering some sort of effect, can be modeled as a **clicked** state with its own internal logic and goals.

As you might guess, a entity state looks similar to a game state, with its own internal logic L, a series of transition conditionals C, and transition mappings T:

$$E = \{L, C, T\}$$

One example of an entity state is **jumping**, which may transition from **idle** or **walking** or **running**. When transitioned to, the **jumping** state might, according to L, play a jumping sound effect. When a condition in C is satisfied (the jumping entity finally touches the ground again, or perhaps falls into a pit and dies), then the transition function T moves to the next state E_N:

$$T(C) \longrightarrow E_N$$

Conclusion Here, I have provided a general outline of games in formal notation. I do not guarantee nor contend that these structures form a complete closure over the entire concept of video games—whose definition is still fiercely debated to this day[18]—but these structures will serve as a useful framework for our following definitions and concepts.

3.2.2 Design

3.2.3 Training

3.2.4 Implementation

3.2.5 Play

3.3 Considerations

Chapter 4

Sample Implementation

4.1 OpenCV

4.2 Qt

Chapter 5

Experiments

5.1 Pong Demonstration

5.1.1 Results

Chapter 6

Conclusions

6.1 Limitations

6.2 Future Work

6.3 Thanks

Chapter 7

Bibliography

[1] K. Lewin, "87

[2] G. Costikyan, "New front in the copyright wars: Out-of-print computer games," May 2000. [Online]. Available: https://archive.nytimes.com/www.nytimes.com/library/tech/00/05/circuits/articles/18aban.html

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] W. Knight, "Google's gemini is the real start of the generative ai boom," *Wired*, Dec. 2023. [Online]. Available: https://www.wired.com/story/google-gemini-generative-ai-boom/

[5] S. Meredith, "A 'thirsty' generative ai boom poses a growing problem for big tech," Dec. 2023. [Online]. Available: https://www.cnbc.com/2023/12/06/water-why-a-thirsty-generative-ai-boom-poses-a-problem-for-big-tech.html

[6] E. Brynjolfsson and A. McAfee, "The business of artificial intelligence," *Harvard Business Review*, Jul. 2017. [Online]. Available: https://hbr.org/2017/07/the-business-of-artificial-intelligence

[7] M. Heikkilä, "Inside a radical new project to democratize ai," Jul. 2022. [Online]. Available: https://www.technologyreview.com/2022/07/12/1055817/inside-a-radical-new-project-to-democratize-ai/

[8] K. Wiggers, "Inside bigscience, the quest to build a powerful open language model," Jan. 2022. [Online]. Available: https://venturebeat.com/ai/inside-bigscience-the-quest-to-build-a-powerful-open-language-model/

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray,

C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[10] R. Brunelli, *Template matching techniques in computer vision.* Hoboken, NJ: Wiley-Blackwell, Mar. 2009.

[11] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 207, no. 1167, p. 187–217, Feb. 1980. [Online]. Available: http://dx.doi.org/10.1098/rspb.1980.0020

[12] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. AVC*, 1988, pp. 23.1–23.6, doi:10.5244/C.2.23.

[13] I. Apple, "Airpods: Control spatial audio and head tracking." [Online]. Available: https://support.apple.com/guide/airpods/control-spatial-audio-and-head-tracking-dev00eb7e0a3/web

[14] I. Campbell, "Dell's new 4k qd-oled monitor comes with spatial audio," Jan. 2025. [Online]. Available: https://www.engadget.com/computing/accessories/dells-new-4k-qd-oled-monitor-comes-with-spatial-audio-194551957.html

[15] A. J. Vanne, D. A. Satongar, J. W. Vandyke, J. O. Merimaa, M. E. Johnson, and T. A. Huttunen, "Spatial audio reproduction based on head-to-torso orientation," U.S. Patent 20 240 357 308, Oct., 2024.

[16] L. McCormack, C. Hold, M. McCrea, juhanipaasonen, S. Saue, jananifernandez, MarcoBin, nmkahlen, M. Lavallée, R. Daugintis, magnus nomono, L. Strand, L. Hannink, exargon, N. Wakefield, O. Larkin, bijulette, and M. Kjeldgaard, "leomccormack/spatial_audio_framework," 6 2024. [Online]. Available: https://github.com/leomccormack/Spatial_Audio_Framework

[17] V. Software, "Steam audio." [Online]. Available: https://valvesoftware.github.io/steam-audio/

[18] J. Knoop, "Epic v apple judge grapples with the big question: What is a videogame?" Sep 2021. [Online]. Available: https://www.pcgamer.com/videogame-definition-legal/