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 10, 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 Recent Advances in Software Development

1.2 Recent User Experience Trends

1.3 Games and Accessibility

Chapter 2

Literature Review

2.1 Computer Vision

2.2 AI & Machine Learning

2.3 Audification

2.3.1 Spatial Audio

2.4 Games

2.4.1 Games and Accessibility

2.4.2 Games and Audification

2.4.3 Games and Computer Vision

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 (**Re**-usable **A**udification **L**ibrary) is a software framework 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. I will demonstrate this in a later chapter, creating one implementation of ReAL Sound using the original corner detection algorithm—the *Harris corner detector*[12], originally developed in 1986.

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 display monitors from major manufacturers like Dell[14]. Future patents promise even greater advancements through advanced techniques like automatic HRTF adjustments specific to the user's unique 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] have seen popular adoption. Legacy libraries with widespread adoption, such as the quarter-century old *Qt* application development framework[18], have also introduced support for modern spatial audio technologies in recent versions[19]. The implementations of ReAL Sound presented later in later sections utilize Qt as a backend framework—although numerous other libraries feature equivalent functionality.

Conclusion In this section, I proposed ReAL Sound and provided justifications for its utility and novelty by considering contemporary technology innovations and emergent user trends.

3.2 Concepts

Introduction In this section, I explain in detail the theoretical concepts which underpin ReAL Sound. I later use these concepts to rigidly define the system's

core structure. To begin, I use formal math notation to construct a semi-formal definition of a 'video game.' I then explore this definition through the lens of automata theory. Following this, I use the construct to abstract methods of interpreting a game via visual analysis.

3.2.1 Anatomy of a Game

Introduction

Here, I construct a semi-rigid abstraction of video games using formal notation. Later, I will use these constructs to more clearly explore related concepts.

Game

A Game, termed M (for *meta*), can be conceived 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!'* onscreen.

In abstract, L can be imagined as a series of conditional requirements ('rules') L_C that yield specific game actions ('responses') L_A:

$$L : L_C \longrightarrow L_A$$

Conditions A state also contains conditional rules defining when to exit the state and transition to different state. 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" to or "how" to transition to a different state after a condition in C has evaluated to `true`. In essence, T maps a conditional statement C_S to 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:

$$T(C) \longrightarrow E_N$$

Summary

To summarize, a game M can be conceptualized as:

1. A collection of attributions (A), which describe aspects of the game's overarching meta-state.

2. A set of game states (G), which each contain internal game-logic (L), state-specific attributes (A), as well as rules for when to transition to another state (C) and exactly which state to transition to (T).

3. A set of entities (I) which serve as the game's passive and active subjects, each having their own set of attributes (A) and states (E). Each state compasses the same qualities of a state enumerated above (A, L, C, and T).

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

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

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

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

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[20]—but these structures will serve as a useful framework for our following definitions and concepts.

3.2.2 State Models

Introduction

In this section, I briefly review automata theory theory. Through concepts like finite state automata, I produce the Game and Entity (GSM and ESM) state machine constructs, which I will continue to utilize in following sections.

Automata Theory

The previous section's underlying theory will likely seem familiar for those with a background in fields such at theory of computation or advanced linguistics. I am, of course, referring to the theory of *automata* a core concept which underpins numerous fields–including computing and, by natural extension, interactive software such as video games.

Automata are generally categorized into four major types as defined by the Chomsky Hierarchy—ranging from Type-0, describing the classical Turning machine, to Type-3, which describes finite state automata [21]. Games, as I have described them above, can (in general) be described as a Type-3 automata—an automation of finite state and quality.

Finite State Automata

Introduction and Concepts Carroll defines (deterministic) finite automata as the "mathematical model of a machine that accepts a particular set of words over some alphabet Σ.[22]" He conceptualizes FSMs as "black boxes" that combine an input tape (consisting of symbols in the alpha Σ), a *'read head'* which processes this input tape of symbols, and an 'acceptance light' which indicates the acceptance/rejection of an input symbol, and whose activity is governed by

the read head's reaction to the given input. In essence, the machine accepts input, makes decisions for the acceptance light based upon the input, and then moves to a new position to receive another input. The machine's acceptance/rejection of a symbol, as well as the position it moves to next, is contingent on its own internal logic—some rule that dictates *"If symbol X is read, turn on (if accepted) or off (if rejected) the acceptance light, and move read head to position Y."* The machine may also read a symbol which finally halts its operation.

Comparison with Turing Machines This conceptual structure bares some resemblance to a Turning machine[23], although there are some distinct differences. A Turning machine has an infinite amount of memory and is capable of accepting languages with recursive qualities—as the turing machine is able to 'write' to tape–modifying a symbol after reading it. A Turing machine is well known to be capable of implementing and computing any possible computing algorithm[24]. It is consequently classified as a Type-0 (also known as a "recursively enumerable" or an "unrestricted") grammar.

An FSM, by contrast, is limited in several ways. As the name implies, FSM have a *finite* set of states. More importantly, the system is incapable of storing memory—lacking the 'writing' mechanism of a Turning machine[22].

Deterministic and Non-deterministic FSAs The FSMs we detail here should also be distinguished as *deterministic*, as opposed to *non-deterministic* FSMs. Some FSMs are considered non-deterministic when their states are capable of transitioning to several possible outputs given the *same input*—making their reaction to an input non-deterministic in nature[25]. For our purposes, we define games as a type of deterministic FSM–where the same inputs always yield the same results.

Obstacles and Limitations You may be wondering how the provided definition of games fit the category of FSM considering the inclusion of attributes A—which may persist as a game or entity transitions from state-to-state. This may appear like a kind of 'memory' on first glance, but is actually just a shorthand notation that simplifies our FSM conceptualization. In reality the proper FSM of a game has *many* more states than we describe here.

To provide some insight into this concept, imagine that for each state G_S, the conditions of C and the transitional rules of T also encode the attributes of A. This means that each state G_S actually has numerous variations—each relating to the specific attributes of A and the conditions of C. For example, the aforementioned sport game's **match** state may really have 10 different **match** states, one describing each possible value of the scoring attribute S. If the game has just begun, the first scoring condition C_0 will transition the machine to $match_1$, and so on. Eventually, the final **match** state's C_9 condition will transition the FSM to the **finish** state.

Useful Adaptions As you can see, this more-accurate representation of FSMs is considered unproductive in many domains where FSMs remain useful concepts[26]. Thankfully, numerous alternative notations have arisen to address these shortcomings.

For example, FSMs which produce output contingent on a given input—often termed *transducers*[22]—are frequently conceptualized as *Mealy machines*. The output of a Mealy machine state is also contingent on its input (`IF input(X) AND IN state(Y) THEN transition(Z)`). Later designs—such as the *unified modeling language*—have extended this notation further to leverage the theoretical benefits of FSAs while avoiding their limiting notation schemes[27]. In a similar fashion, I make use of FSAs on a conceptual basis while abstracting away these notational hindrances through the usage of the attributes A of each object.

Automata and Games

It is possible to define games more clearly through their relation to FSAs using our definitions. For example we can consider a game to be a collection of two principal state machines:

Game State Machine (GSM) The game state machine comprises the general 'flow' of a game—its levels, combat encounters, win conditions, story sequences, game-over menus, boss fights, item inventories, and any other 'macro' loop that is reasonably distinguishable within the gaming experience.

Entity State Machine (ESM) An entity state machine comprises the general 'life' and 'activities' of a given entity—every action it can take, as well as its response to any given input. A player may run, jump, or die, while a stage platform may move or vanish.

These two concepts—the GSM and ESM—prove to be crucial in the conceptualization of ReAL Sound. We will continue to analyze them in later sections using other frames of analysis.

Conclusion

In this section I reviewed numerous aspects of automata theory—namely finite state automata—in order to present the concept of the Game and Entity state machines (GSM and ESM), which will prove to be useful constructs in the total construction of ReAL Sound.

Visual Analysis of a Game

Introduction In this section, I provide a framework for analyzing a game's state information based on the game's visual output by using the constructs detailed in the previous section. With this framework, I argue in the next section that a user who is aware of the game's external logic (the 'rules' a player

intuitively learns via playing the game) can wield this visual analysis in building their own state model of the game.

3.3 Structural Outline

3.3.1 Vision Layer

3.3.2 Decision Layer

3.3.3 Audification Layer

3.4 Process Outline

In this section, I outline ReAL Sound's principal operational process—the **Planning**, **Training**, **Design**, **Execution** loop. To conclude, I consider the structure's strengths and weaknesses.

3.4.1 Design

3.4.2 Training

3.4.3 Implementation

3.4.4 Play

3.5 Demonstration: Pong

3.6 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% missing: The disappearance of classic video games," Nov 2023. [Online]. Available: https://gamehistory.org/87percent/

[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. Blanchette and M. Summerfield, *C++ GUI Programming with Qt 4*, 2nd ed. USA: Prentice Hall PTR, 2008.

[19] T. Q. Company, "Qt spatial audio 6.8.1." [Online]. Available: https://doc.qt.io/qt-6/qtspatialaudio-index.html

[20] 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/

[21] N. Chomsky, "Three models for the description of language," *IRE Trans. Inf. Theory*, vol. 2, pp. 113–124, 1956. [Online]. Available: https://api.semanticscholar.org/CorpusID:17432009

[22] J. Carroll and D. Long, *Theory of Finite Automata with an Introduction to Formal Languages*, 01 1989.

[23] A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1937. [Online]. Available: https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s2-42.1.230

[24] A. Turing, *The Essential Turing*, B. J. Copeland, Ed. Oxford, England: Clarendon Press, 2004.

[25] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM Journal of Research and Development*, vol. 3, no. 2, pp. 114–125, 1959.

[26] M. Samek, "Who moved my state?" *The C/C++ Users Journal*, vol. 21, pp. 28+30–34, 04 2003.

[27] OMG, *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1*, Object Management Group Std., Rev. 2.4.1, August 2011. [Online]. Available: http://www.omg.org/spec/UML/2.4.1