VIRTUAL STAGE: MERGING VIRTUAL REALITY TECHNOLOGIES AND INTERACTIVE AUDIO/VIDEO

Stephen Lucas, B.M., M.M.

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

May 2017

Lucas, Stephen. Virtual Stage: Merging Virtual Reality Technologies and Interactive Audio/Video. Doctor of Philosophy (Composition), May 2017, 94 pp., 25 figures, 24 code examples, bibliography, 71 titles.

Virtual Stage is a project to use virtual reality (VR) technology as an audiovisual performance interface. The depth of control, modularity of design, and user immersion aim to solve some of the representational problems in interactive audiovisual art and the control problems in digital musical instruments. Creating feedback between interaction and perception, the VR environment references the viewer's behavioral intuition developed in the real world, facilitating clarity in the understanding of artistic representation.

The critical essay discusses of interactive behavior, game mechanics, interface implementations, and technical developments to express the structures and performance possibilities. This discussion uses *Virtual Stage* as an example with specific aesthetic and technical solutions, but addresses archetypal concerns in interactive audiovisual art.

The creative documentation lists the interactive functions present in *Virtual Stage* as well as code reproductions of selected technical solutions. The included code excerpts document novel approaches to virtual reality implementation and acoustic physical modeling of musical instruments.

ACKNOWLEDGEMENTS

During the period of developing this project, I faced numerous technical, aesthetic, and logistical problems; I have only persevered with the encouragement and support of my professors, colleagues, family, and friends. I thank my advisor Jon Christopher Nelson, the other members of my committee, David Stout and Joseph Klein, for their guidance. Additionally, I would also like to thank specific professors at UNT that were prominent in my development while pursuing this degree: Paula Gaetano Adi, Nadav Assor, Alejandro Borsani, David Bard-Schwarz, Timothy Devoe, Panayiotis Kokoras, Andrew May, Elizabeth McNutt, Christopher Trebue Moore, Margaret Notley, Thomas Sovík, and Jenny Vogel. I thank several colleagues in the composition department whom I have worked with in various ways during this degree: Ermir Bejo, Jason Fick, Timothy Harenda, Joshua Harris, Jon Jackson, Joseph Lyszczarz, Mark Oliveiro, Patrick Peringer, Brad Robin, Seth Shafer, Qi Shen, Benjamin Shirey, Michael Sterling Smith, Zach Thomas, Dan Tramte, Robert Trusko, Chaz Underriner, and Jinghong Zhang. Additionally, I thank my other frequent collaborators, Rob Buttrum, Greg Dixon, Shane Hutchinson, Cory Metcalf, Chris Polcyn, and Dairy Tade, and I thank all the artists, programmers, and scholars whom I have directly and indirectly collaborated with through research. Finally, I thank my family for their love and support, and especially my wife Sarah Ruth Alexander — I couldn’t have done this without you.

TABLE OF CONTENTS

LIST OF FIGURES

Page

PART I

CRITICAL ESSAY

1. Introduction and Interactive Behavior

1.1 Virtual Environments for Performance

Interactive audiovisual art often has conflict between aesthetic and technical goals. This friction makes it difficult to maintain cohesion between how elements interact and how that interaction is perceived. By creating a feedback between interaction and perception, art references the behavioral intuition developed in the real world, facilitating clarity in aesthetic representation.

Virtual Stage is a project to use virtual reality (VR) technology as an audiovisual performance interface. The depth of control, modularity of design, and user immersion aim to solve some of the representational problems in interactive audiovisual art. Drawing influence from game mechanics, modules allow for behavioral structures that are traditionally difficult to portray, such as reflexivity and intentionality. Additionally, the interface allows for audio representation through combinations of modulated acoustic physical models that are difficult to control with orthodox computer music interfaces.

A standard model of VR places the participant in a first-person experience, isolated from the real world and any onlookers. However, in *Virtual Stage*, the VR hardware facilitates the performer to exist in the virtual world, while an additional video output displays a third-person view of the environment to the audience. The audience recognizes the performer on both the real stage and the virtual stage, allowing interactive behaviors to be directly observable (Figure 1.1)

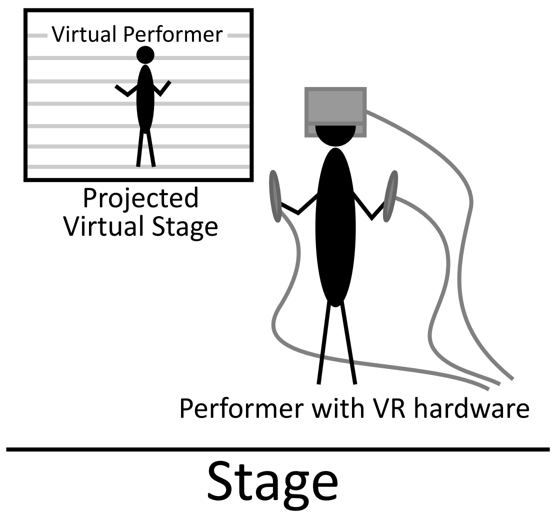


Figure 1.1 Stage orientation with controllers

However, the entire VR environment functions as an interface for the performer to control objects and functions that generate sound and visuals. The audience sees the on-stage performer's input to the VR controllers and the on-screen environment as output; when the live performer's motions directly correlate to the virtual performer, they reinforce the viewer's context for the spatiality of the virtual environment. However, the digital environment is not subject to the same physical limitations of input and output as in the real world.

The performer dictates the rate and order in which the audience is introduced to new elements in the space; this provides some malleability of composition form, but also paces a natural exposition to the audience's understanding of the environment. As new elements are introduced, there is both a spatial understanding of their virtual physics and an auditory component in the environment. This situation is similar to any interactive system, in that the audience develops a familiarity with the interactive behavior as they observe more correlation and causality. However, the development of familiarity with both the virtual instruments and environment reinforce each other, allowing for more complex and specific projections of behavior and structure.

The instruments are implemented modularly, such that developing each new instrument multiplies the possible combinations with other instruments. This widens the range of actions for the performer because new instruments evoke new ways to use existing instruments. However, this requires that the virtual interface is both intuitive and flexible for the performer to take advantage of the design modularity; if the input requires too much precision, the performer will be less likely to explore variations.

This openness of variation and modularity of function is more representative of a game than a temporally bounded piece. *Virtual Stage* consists of a structure of interactive game mechanics for creating an open-ended performance. Discussion of interactive behavior, game mechanics, interface implementations, and technical developments express the possibilities of this openness and structures to manifest them.

1.2 Framing of Interactive Systems Problems

Temporal art often functions as a transduction between different forms of energy, whether they are motion, sound, visuals, and/or data. Musical instruments convert physical motion into sound energy in their transduction of input to output. Digital systems add a layer of complexity, because energy is converted between analog and digital states. This conversion has several repercussions based in a conceptual detachment from the physical world. When energy is quantized into data, removing its bounding of spatial dimensions, it gains a relatively endless permeability through operations of logic, math, and storage. A digital system seems to behave as an instrument, in that it converts input to output, but the flexibility of permutation in digital operations makes it difficult to observe the behavior as energy transduction. A viewer understands behavior by what they observe directly, but without knowledge of the digital operations, they are limited in their understanding of the digital system's behavior.[1]

There are several prototypical paradigms of interactive digital arts systems, each with their inherent benefits and limitations to a viewer's understanding. With a performer on stage at a computer and with limited visibility of the performer's interactions, the viewer observes only the output of the system. A live performer's presence predicates some purpose but without an observable behavior, the viewer can assume only the structural implications of performance itself — the performer may be actively exerting some skill, preference, or individuality. These exertions may impact the artistic output but without direct observation, the observed result may seem distanced from any interactive behavior.

A performer with motion tracking controllers directs some correlation and/or causation between physical gestures and an output. The viewer directly observes the input, but they must extrapolate the interactive behavior from the output. If preference is given to an intended output, the interactivity is often constrained to make this result more achievable. The viewer derives understanding of the interactive behavior to the extent that these constraints maintain consistency; however, if constraints increase and the behavior seem to be scripted, the system will appear less interactive, thus limiting the representational capacity of the interaction. If the performer embellishes the input to dramatize its inherent qualities, the viewer could make erroneous assumptions about the system's behavior.

Digital artists often both design and perform an interactive system. With acoustic

[1] Within the scope of audiovisual art, the audience observes of all forms of media available to the senses. A "viewer" observes any combination of sound, image, live performance, etc. "Art" includes any combination of these observable media and their perceived structures.

instruments, it takes some amount of musicianship to build a fine instrument, but there is no prerequisite that an excellent instrument designer is also an excellent performer on that instrument. With interactive digital systems, the artist often treats the design process as a performance in itself; this can detract from their actual performance, because it distorts their ability to recognize how the viewer would understand the system. Additionally, performance practice seeks to solve inherent problems and limitations in the instrument, but the artist could always resort to design changes to affect these changes.

Although the input of a system may seem observable to the viewer, it is mitigated by how the interactive behavior is understood. A performer's input can range from "action" to "interaction" or even "reaction," depending on how much the output from the system influences their decisions. Observations and preconceptions color the viewer's understanding of the interactivity, but the performer is similarly subject to feedback from the system. When input and digital behavior don't exist in the same spatial dimensions, the correlation of input to output can be any mathematical or logical function, physically possible or not. An increase in the observability of behavior benefits both the performer's and viewer's understanding of the system.

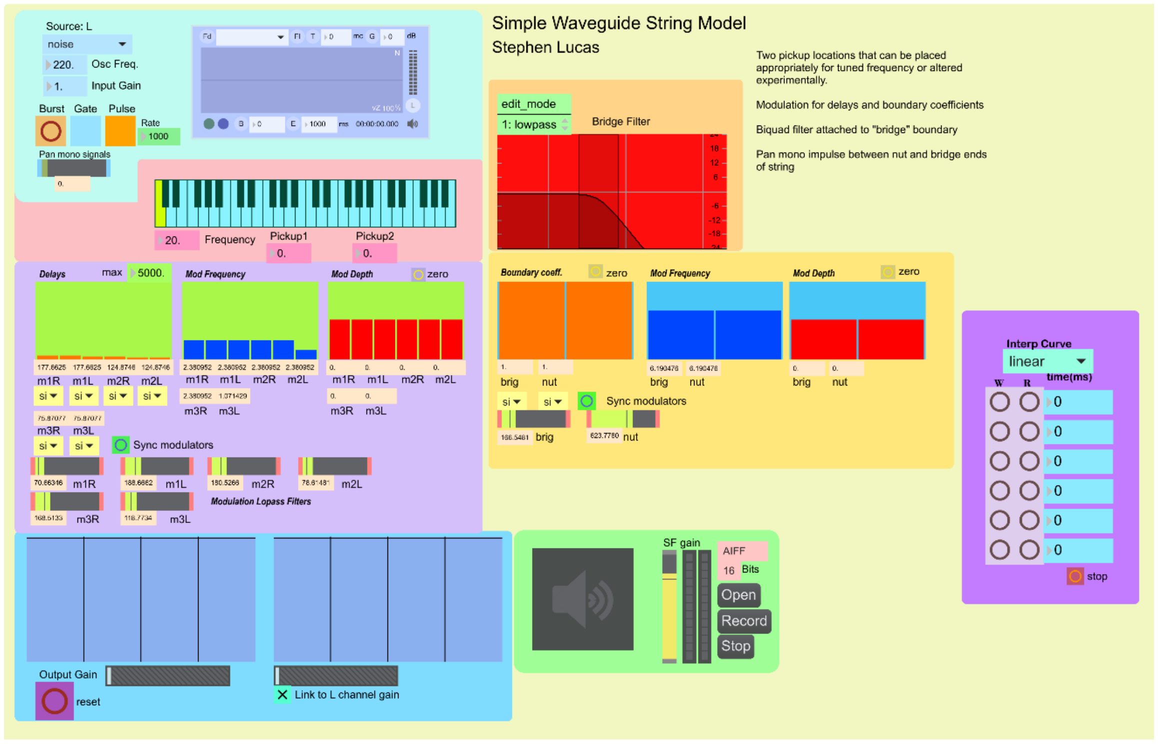
1.3 Control and Interface in Temporal Art

Observations are limited by the platonic ideal of a "piece," where art is framed within containers such as the time span of a performance or the spatial bounds of a stage; the viewer's observations are subject to speculation about how the piece and artist exist outside these bounds. Similarly, the artist exerts a different form of interaction during performance than during designing and testing. The definition of these boundaries informs the understanding of interactivity because it indicates the line between quantitative and qualitative descriptions. However, a digital system presents a level of abstraction of quantitative data that adds complexity to observing the bounds of the performance.

The technique of physical modeling allows the retention of perceptual identities and behavior by simulating the mechanics of sound production rather than the acoustic properties of hearing.[2] If a digital system structurally imitates how sound and objects behave in the real world, it grounds the modes of interaction to represent quantitative actions, demystifying the bounds of the digital system. For example, a physical model of a string instrument could have the same input and output behavior of a real, acoustic string instrument, so it would be easier to presume its interactive bounds. However, the model could be modulated in ways that are physically impossible (such as retuning the string tensions faster than mechanically possible). The digital model's presumed bounds retain influence on the understanding of its behavior, even if the final output is dissimilar to the real counterpart.

However, physical modeling presents a performance problem — when the model's control parameters extend beyond the physically possible, the physical input interface is inherently limited. For example, Figure 1.2 shows a computer interface for a physical waveguide string model, modeling how vibration travels along both directions of a string.

[2] Juraj Kojs, Stefania Serafin, and Chris Chafe, "Cyberinstruments via Physical Modeling Synthesis: Compositional Applications," *Leonardo Music Jornal* 17 (2007): 61.



Two pickup locations that can be placed appropriately for tuned frequency or altered experimentally.

Modulation for delays and boundary coefficients

Biquad filter attached to "bridge" boundary

Pan mono impulse between nut and bridge ends of string

Figure 1.2 Interface for waveguide string with modulation

Though the instrument is intended for the testing and generation of fixed media samples rather live performance in a piece, it is still subject to interface limitations. The performer manipulates numerical parameters of the model and oscillators modulate those parameters faster than a physical interface can move; some configurations are mathematically possible but do not make physical sense, such as making the string longer on the top than on the bottom.[3] While there is a grounding in a relationship to an instrument that is physically observable, there is no physical interface for simultaneously affecting all of these extended parameters. As the interface of the model becomes more abstracted from its physical counterpart, the potential of expected behavior decreases and appropriately, the output is less recognizable as a real string.

[3] This is possible because the model is two resonating delay lines that reflect and scatter audio into each other. The modeled string length is represented by of the sample durations and resonant frequencies of the delay buffers; however, there is no mathematical requirement that these two resonators are tuned to the same frequency.

While this interface is limited to the changing of one parameter at a time, preset states allow interpolation between these captured configurations at different rates. The user can assemble presets into a series of states that would more closely resemble a nuanced physical input. However, the presets are still subject to the model's abstraction from its physical counterpart, because the presets themselves are an added layer of abstraction; the interface is a model of physical interaction, but its interactivity is informed by the bounds of its control.

1.4 Modular Instrument Design

Although physical modeling simulates the mechanism of sound production, this structure does not represent the modularity of physical objects. Modular programming hierarchically abstracts functions into component parts, such that smaller, discrete modules comprise a larger application. Some benefits to this include economy of reusing work, division of labor into achievable goals, and a more clearly observable design structure.[4] Efficiency is helpful when designing complicated systems, but modularity also allows a shift in hierarchical perspective to highlight how a system operates.

Musicians are familiar with modular instrument structures because most musical instruments require coordinated interfacing of simultaneous input modes. For example, a trumpet player buzzes the mouthpiece with their lips, controls airflow with their tongue, operates the valves with one hand, and adjusts tuning slides with the other hand. Each of these inputs is modularly isolated in its function, understood conceptually, and practiced independent of the whole. Some components are hierarchically reducible or share commonalities, such as

[4] Matthias Rath, "A Strategy for Modular Implementation of Physics-Based Models," *Proceedings of the 7th International Conference on Digital Audio Effects*, October 2004, 1.

how the lips and tongue both coordinate with air speed generated by the diaphragm. Additionally, valve combinations can be practiced and reconfigured for efficiency, but are performed automatically when playing a musical passage; fingering combinations are already hierarchical aggregates of individual valves.

The parameters in Figure 1.2 are not modularly interactive because hierarchically, they are managed one at a time on the same level; there is no inherent context to organize parameters into recognizable patterns. Similarly, the preset interpolation is limited because it homogenizes all parameters and doesn't create intermediary functions in the same manner that a performer groups different input modes. Although the model represents a physical object, the interface inhibits the observability of the parameters' physical correlation.

1.5 Units and Data Mapping

Computer controllers have a physical interface, but they are modeled on basic computer science data types. For example, a button toggles or triggers a Boolean true/false value and a slider outputs a range of values. These values are readily processed with standard logical and mathematical functions because they are inherently input as numerical values. However, toggles are rarely present in the physical world and real interaction is more a function of multiple, continuously balanced parameters; when these parameters don't correlate to physical units, they lack an implication of behavior.[5] A digital piano keyboard controller affords an overlap with piano technique and implies semitone units that hold musical context through spatial relationship. Regardless if the semitone scale is remapped, a piano keyboard controller

[5] Davide Rocchesso, Federico Avanzini, Matthias Rath, Roberto Bresin, and Stefania Serafin. "Contact Sounds for Continuous Feedback," *Proceedings of the International Workshop on Interactive Sonification*, January 2004, 1–2.

implies more behavior than a slider with no output because the spatial units reference the bounds of a performance system.

Conversely, a physical instrument’s boundaries are defined by its spatial dimensions and its physical properties. The stability of physical space imparts an implication of cohesion via the persistence of spatial units. A sensor captures a performer’s input to the system and the data is bounded by both the constraints of the sensor and the system’s behavior. The data captured by the sensor loses some of its physical implication when abstracted into a scale without physical units. The observability of this data relies on the interactive behavior to indicate the spatiality in a physical representation.

Contrastingly, symbol values such as words or characters can represent units but also are meaningful in qualitative statements.[6] A symbol does not correlate to a physical dimension but requires a physical context to signify a physical relationship. Notated music parallels this limitation with a boundary of meaning between quantifiable acoustic representation and syntactical designations represented by language. Words in music notation often function as comparators (such as “faster” or “quieter”) or they consolidate an idea that is inefficient to quantify precisely.

The boundary of quantitative and qualitative designations is not necessarily absolute, but it is critical to the limitations on the observations of behavior. For example, the perspectives of a musician performing from a notated score and an audience member have different

[6] The “symbol” type selector is more commonly called a “string.” However, “string” could be confusing as it is homonymous with an acoustic “string.” “Symbol” is the type name used in, Cycling ’74’s Max, the main platform for this project.
Cycling ’74. (2017). Max (Windows version 7.3.3) [software]. Available from http://cycling74.com.

quantitative/qualitative boundaries. The performer recognizes quantifiable elements like pitch and rhythm through descriptors and instructions on the page; however, the viewer observes physical properties at the rate and manner which they are performed. The performance transduces the notational quantities to auditory quantities and the nuance of the performer's behavior effects the qualitative limits.

1.6 Virtual Reality as a Solution

Video game interfaces often use generalized peripherals with buttons and analog controllers, yet they enable players to perform complicated and nuanced combinations of inputs with contextual variation and meaning. Games directly correlate the interface with observable behavior in a virtual space and teach combinations through context and repetition.

The context of the interface and the player's in-game avatar are set against an environment or elements of the game that the player does not control. The environment is designed to behave with a scale of unpredictability, but the control interface responds predictably so that the player can maintain an understanding of the behavioral context. Variability in the environment is essential for establishing the player's context for the interface, since different environmental challenges require learning and executing variations to overcome them. Although some games introduce surprise by spontaneously changing the bounds of the interface, the player reacts to this like an environmental challenge as they attempt to reestablish the bounds of the interface. Many games are actually entirely deterministic, but the complexity of the environment's rules gives the impression of unpredictability.

While the conceptual framework of a game may differ from a performance piece, the overlaps in interface design provide methods for approaching the challenges of interactive behavior that are intrinsic to digital systems. Establishing a boundary between the interface and the environment allows for the performer and viewer to learn a context for behavioral interaction because it defines what is predictable and unpredictable. These contextual rules can correlate to observable behaviors, but allow for broader qualitative representation through the representation of nuance and combination.

Virtual reality (VR) is a mode of computer interface in which a user operates motion controllers to interact with virtual environments while using an immersive headset display. The benefit of VR is that the user's scale and perspective is synchronized to the virtual space.[7] Once the user establishes a method for movement in the virtual space, they can coordinate their own intuitive body movements with the context of the environment. Additionally, this coordination allows for a more parameters of interactive input than what is typical with a button-based hand controller.

This reference to gameplay mechanics creates other possible interactive similarities. *Virtual Stage* incorporates structures of game behavior in both design and aesthetics. This increases the combinations of interactive expression and spatial representation.

[7] Scott deLahunta, "Virtual Reality and Performance," *JAP: A Journal of Performance and Art* 24, no. 1 (2002): 106.

2. Virtual Environments

2.1 Sandbox Games

Virtual Stage is a game environment, in the sense that its boundaries are specified in the rules of how objects function, but not a dictation to the performer of how to interact with those rules. This interactive open-endedness is influenced by sandbox gaming strategies, most notably the video game Minecraft and the card game Magic: The Gathering. These games supply the player with materials and possibilities, but the narrative depends on their decisions and creativity of combining these elements.

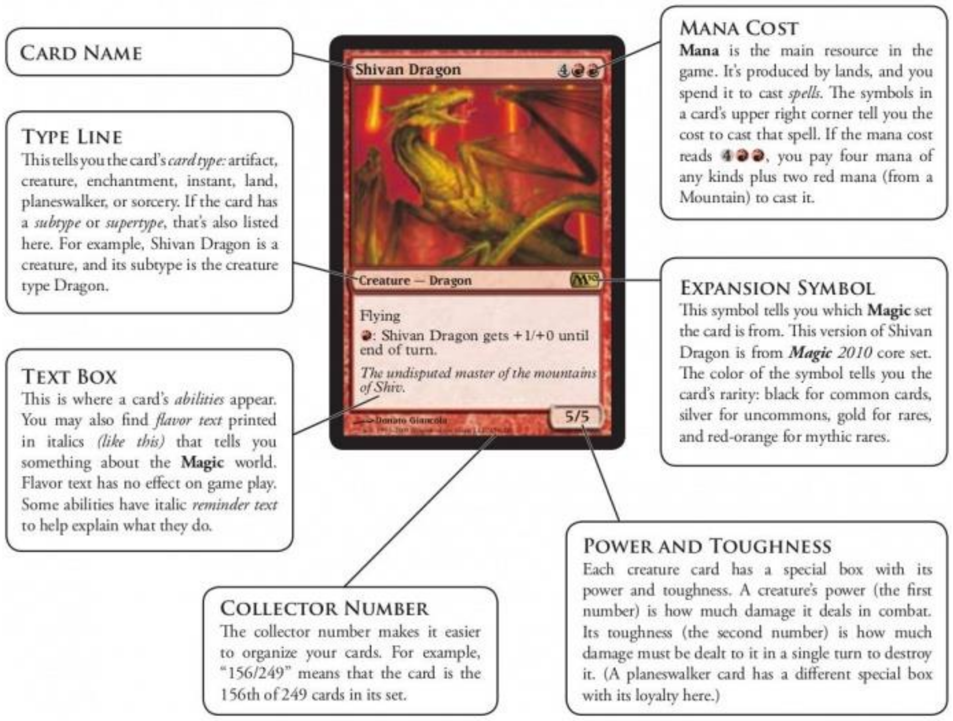
Minecraft is an interactive system for building structures out of virtual cubes with no inherent game narrative or goal.[8] Minecraft's player constructs their own narrative through how they decide to combine materials together. There are two modes to this constructing — a scripted progression through collecting and combining materials to build increasingly advanced tools, and free-form combinations of the player's devising like houses or sculptures. These modes reinforce each other because the creative construction is imparted a narrative through the conflict of acquiring the creative materials. Making a wooden house requires crafting an axe and chopping trees; building underwater requires establishing a complicated tunneling system.[9] *Virtual Stage* is an open system during the development of new instruments, but when the performer is in the VR space, the narrative of performance is linked to the process of combining instruments.

[8] "About Minecraft," *Minecraft Wiki, Gamepedia*, accessed November 10, 2016, http://minecraft.gamepedia.com/Minecraft_Wiki.
[9] Matthew Belinkie, "What Makes Minecraft So Addictive," *Overthinking It*, November 11, 2010, http://overthinkingit.com/2010/11/11/minecraft-vs-second-life.

In early tests for *Virtual Stage*, controller buttons were each mapped to single functions, while the motion controllers were mapped differently depending on which function was active. This limits the number of possible functions, especially when including utility functions like calibration or camera control. One possible solution is for a button function to toggle a mode that changes the function of other buttons; however, the complexity of this type of structure increases the difficulty of navigating between functions and requires memorization of the structure.

Virtual Stage's organizational structure for functions is inspired by the competitive card game Magic: The Gathering.[10] In Magic, each card represents an individual spell (function) with details about the resources it requires and its possible effects on other cards. (Figure 2.1)

[10] "Magic: The Gathering," *Wikipedia*, accessed November 10, 2016, http://en.wikipedia.org/wiki/Magic:_The_Gathering.


Shivan Dragon
Creature — Dragon
Flying
: Shivan Dragon gets +1/+0 until end of turn.
The undisputed master of the mountains of Shiv.
Donato Giancola
5/5

CARD NAME

TYPE LINE
This tells you the card's *card type:* artifact, creature, enchantment, instant, land, planeswalker, or sorcery. If the card has a *subtype* or *supertype*, that's also listed here. For example, Shivan Dragon is a creature, and its subtype is the creature type Dragon.

TEXT BOX
This is where a card's *abilities* appear. You may also find *flavor text* printed in italics *(like this)* that tells you something about the **Magic** world. Flavor text has no effect on game play. Some abilities have italic *reminder text* to help explain what they do.

MANA COST
Mana is the main resource in the game. It's produced by lands, and you spend it to cast *spells.* The symbols in a card's upper right corner tell you the cost to cast that spell. If the mana cost reads 4, you pay four mana of any kinds plus two red mana (from a Mountain) to cast it.

EXPANSION SYMBOL
This symbol tells you which **Magic** set the card is from. This version of Shivan Dragon is from ***Magic** 2010* core set. The color of the symbol tells you the card's rarity: black for common cards, silver for uncommons, gold for rares, and red-orange for mythic rares.

POWER AND TOUGHNESS
Each creature card has a special box with its power and toughness. A creature's power (the first number) is how much damage it deals in combat. Its toughness (the second number) is how much damage must be dealt to it in a single turn to destroy it. (A planeswalker card has a different special box with its loyalty here.)

COLLECTOR NUMBER
The collector number makes it easier to organize your cards. For example, "156/249" means that the card is the 156th of 249 cards in its set.

Figure 2.1 Parts of a Magic Card[11]

Cards are categorized into types and subtypes that indicate their rules some of their possible interactions; the primary categorization consists of spells that summon an instance of a permanent object to the game space and spells that activate a momentary effect and then resolve.[12] Permanent objects may inflict a continuous state on the game and/or have momentary abilities that can be activated. Spells and abilities are restricted by limits like the player's turn, the availability of the card in the player's hand, and a mana cost — a color-coded representation of the card's resource requirements. The mana colors imply thematic categories and potential synergies with other cards.[13] Players must combine effects and permanents that

[11] "Parts of a Card," *MTG Salvation Wiki, Gamepedia*, accessed November 11, 2016, http://mtgsalvation.gamepedia.com/Parts_of_a_card.
[12] "Spell," *MTG Salvation Wiki,* accessed December 7, 2016, http://mtgsalvation.gamepedia.com/Spell.
[13] "Mana cost," *MTG Salvation Wiki*, accessed December 7, 2016, http://mtgsalvation.gamepedia.com/Mana_cost.

function well together while accumulating resources to activate them; strategies are based on efficiency of resources, countering the opponent's spells, and building strong spell combinations.

Virtual Stage references this structure by encapsulating the idea of a spell as a digital function. A function can affect the environment, target an object, or instantiate objects with their own effects; this normalized approach to functions modularizes the workflow of designing and implementing them. However, in Magic, the costs and conditions on effects are a mitigation of how quickly players can deploy their strategies; the different types of limitations create variations in deck design. In *Virtual Stage*, there is no competitive structure so it is only limited by the performer's ability to create. As in Minecraft, the performer must manually prepare and lay out the component parts to a combination, reinforcing the viewer's awareness of the spatiality of the virtual environment.

A Magic card's mana cost describes both a philosophical category and consequently an implication of how effect combinations interact. For example green and black are opposites — green can accelerate its resources to play tough creatures but black counters it by being able to easily destroy creatures regardless of toughness (Figure 2.2).[14]

[14] Notice that the relatively strong creature Alpha Tyrranax has a significantly higher converted mana cost (6) than Go for the Throat (2), but Go for the Throat will destroy it. Converted mana cost is total count of how many individual mana are required for the spell/effect. Colorless mana can be paid by any color (though not vice-versa).

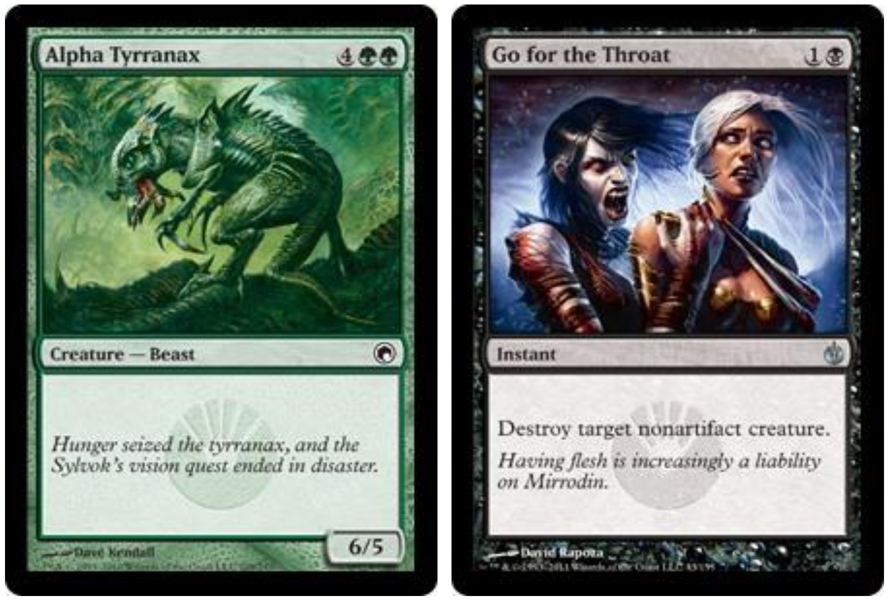

Alpha Tyrranax
4
Creature — Beast
Hunger seized the tyrranax, and the Sylvok's vision quest ended in disaster.
Dave Kendall
6/5
Go for the Throat
1
Instant
Destroy target nonartifact creature.
Having flesh is increasingly a liability on Mirrodin.
David Rapoza

Figure 2.2 Opposing card colors counter each other efficiently[15]

Multicolored cards contain aspects from the included colors while uncolored cards are neutral artifact types. Many cards share the same mana cost, which ostensibly connotes the card's function and complexity (Figure 2.3).

[15] "Gatherer," *Magic: The Gathering,* accessed December 1, 2016. http://gatherer.wizards.com/Pages/Default.aspx.

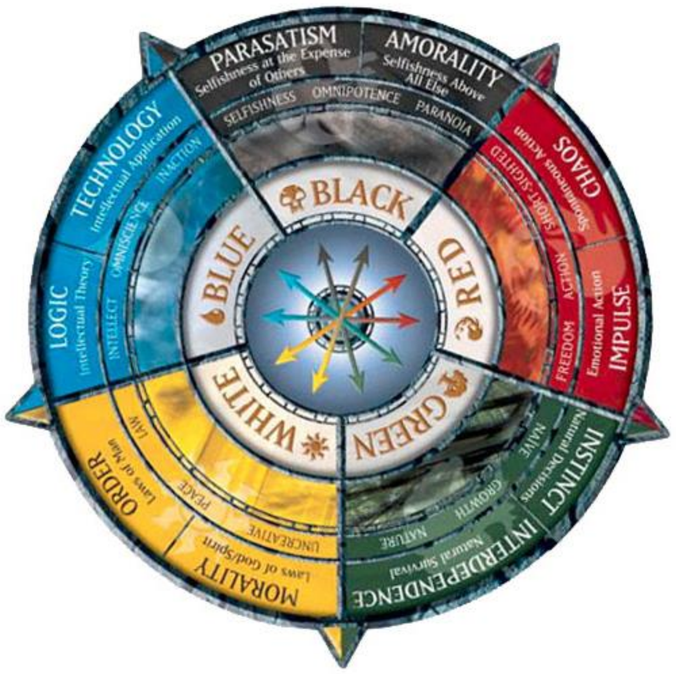


Figure 2.3 Magic: the Gathering color themes[16]

Virtual Stage borrows this format by referencing functions as sequences of colors, which solves multiple problems. Instead of occupying controller buttons with single functions, each button represents a color, and much like an alphabetical character, strings of colors spell out a function — each function is literally a spelling or "spell" of a string of color characters. *Virtual Stage* uses five colors because this fits the input controller.[17] Each color represents a functional category such that the color spelling acts as both the function's button sequence and a memory aid for the performer (Figure 2.4).

[16] "Color Pie," *MTG Salvation Wiki, Gamepedia*, accessed November 11, 2016, http://mtgsalvation.gamepedia.com/Color_Pie.

[17] The Razer Hydra controller is the primary development controller. See section I.2.3 for more info.

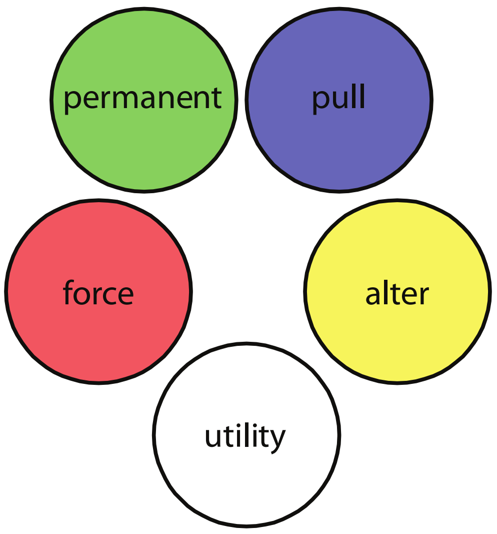


Figure 2.4 *Virtual Stage* color symbols[18]

While Magic's mana cost is a resource restriction, button spellings in *Virtual Stage* don't require any external resource, though it is slightly more time consuming to enter more complicated sequences. There is some balance between the most intuitive color spellings and minimizing the number of button presses.[19]

This paradigm is powerful in that each hand can independently and simultaneously execute a separate spell. Instead of limiting each hand to a set of parameters, each hand contextually changes its function, allowing two active, concurrent functions to interact. Additionally, this requires all functions to be modularly available, which is a prerequisite to allowing non-player objects to activate functions.

2.2 Goal Orientation

The game environment of *Virtual Stage* challenges traditional structural boundaries of interactive art since it is both a digital media composition and an instrument. It functions as a

[18] See section I.3.5 for details on color themes.
[19] See section II.1.1 for specific examples of spells and their color combinations.

composition because it establishes an on-stage boundary and setting for an audience but it functions as an instrument because there is not necessarily a mandated method for operating within this setting. Additionally, the system is designed to be expandable and reconfigurable, increasing flexibility and potential with multiple performance iterations. Therefore, the initial set of functions are designed to both illustrate the boundaries of the system and stimulate new design directions.

The primary goal of the system is to combine functions such that their aggregate effect becomes an interface that exceeds the limits of the interface. Because the environment integrates with the interactive behavior, the viewer should evaluate the behavior of objects with spatial assumptions of how their combinations could interact. However, this requires a guided progression through functions to optimize the rate of behavioral learning. Performance instructions would notate a sequence of function combinations, though the performer would be free to decide what interaction is best suited for the combination.

A secondary goal for the system is that the function combinations may symbolically illustrate a conceptual structure that is removed from the environment itself. In this way, the interactive behaviors would be dependent on the same subjective understanding inherent in any art. This type of structure first requires the establishment of the viewer's behavioral understanding and a syntactical combination of functions in reference to that understanding.

Additionally, the open and expandable nature of the system makes it effective as a presentation tool for a meta-discussion of its own properties. Functions could be useful for demonstration purposes in a way that may not be interesting during an actual performance. Despite having limited performance value, this direction facilitates an additional method to enhance the viewer's spatial and behavioral understanding of the virtual environment.

These goals serve as a directive for the initial and future development of the system, with an understanding that the design phase is a facet of the interactive process. Spell functions themselves are constructed of common sub-components, allowing for improvisation in design; this is mirrored by the improvisational performance in the VR environment. Ideally, this reinforces the grounding of structure in observable interactivity, rather than creating discontinuity between design and performance practice.

2.3 Hardware Technical Considerations

In recent years, consumer-priced VR technology has become much more available. Using consumer-level hardware provides ease of replacement, significantly reduced cost, and more ample software community support. Ideally, the equipment and computing restrictions should not influence the overall design and the hardware setup could be upgraded or substituted without significant reprogramming.

A VR headset contains a video display and convex lenses and the main determination of quality is the format of the computing base station.[20] The HTC Vive and Oculus Rift headsets use a PC base station so they have the most potential computing and graphics power; other platforms like Gear VR are peripherals to consumer-level smartphones, encapsulating the video, head-tracking, and computing into a wireless, battery-powered device. Given that interactive audio-visual environments already have heavy computing requirements, *Virtual Stage* is not intended to run on a fully mobile platform and is developed for PC with a wired VR headset.[21]

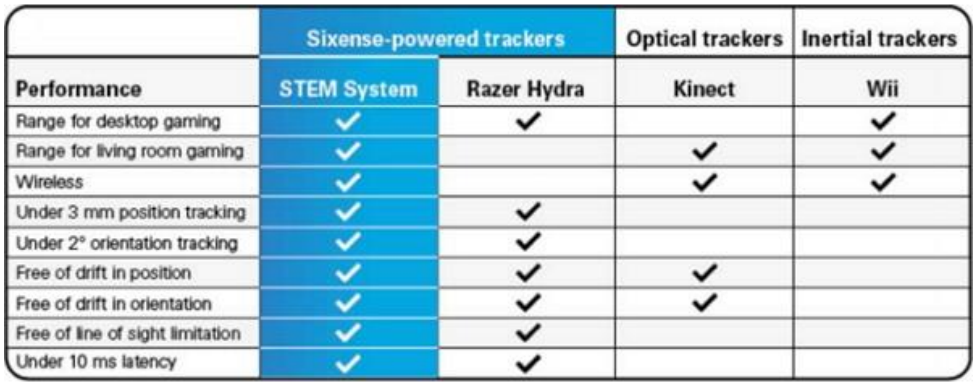
[20] Will Shanklin, "2016 VR Comparison Guide," *New Atlas*, October 18, 2016, http://newatlas.com/best-vr-headsets-comparison-2016/45984.
[21] See section I.4.3 for details on headset integration.

The Vive and Rift differ in their manner of head-tracking and controller integration. Both headsets have their own controllers with different button formats; however, both rely on camera-sensor-based methods for positional tracking.[22] The Vive employs mounted boxes that emit invisible lasers while the headset and controllers have sensors to orient in reference to the boxes. Conversely, the Rift requires a mounted camera to track positions of infrared emitters on the controllers, with additional cameras to increase a wider degree of body motion. The limitation of these tracking methods is additional setup and testing time to ensure that the performance space is optimized for tracking. In an interactive installation piece, long setup to optimize space considerations is expected, but in performance pieces, it is ideal to minimize setup time by keeping the system self-contained and not reliant on constraints of the performance space.

Sixense developed a third tracking method for consumer-level controllers utilizing magnetic fields in the Razer Hydra and the STEM System. These controllers use a base station that emits as a magnetic reference field, while the controllers sense their relative position and rotation to the base station.[23] This method has a more limited range than camera-sensors, but they are less restricted by the performance space or ambient light conditions. The STEM System is wired but the Hydra controllers are wired to the base station, preventing 360° body rotation (Figure 2.5).

[22] Shanklin.

[23] The Hydra has been available since 2011 but presently, the STEM System has not met projected release deadlines and is still in pre-order. The software API should be similar enough that development for the Hydra should be transferrable once the STEM System is available.



	Sixense-powered trackers		Optical trackers	Inertial trackers
Performance	STEM System	Razer Hydra	Kinect	Wii
Range for desktop gaming	✓	✓		✓
Range for living room gaming	✓		✓	✓
Wireless	✓		✓	✓
Under 3 mm position tracking	✓	✓		
Under 2° orientation tracking	✓	✓		
Free of drift in position	✓	✓	✓	
Free of drift in orientation	✓	✓	✓	
Free of line of sight limitation	✓	✓		
Under 10 ms latency	✓	✓		

Figure 2.5 VR controller comparison[24]

In an ideal performance tech rehearsal, the interface would require minimal testing and allows focus on the space's sound and video projection equipment.

Two additional controller methods that have benefits but were not pursued include IR body-tracking cameras like the Microsoft Kinect and the LEAP Motion and inertial trackers like the Nintendo Wii Remote (Wiimote) and the Nod Backspin. Body-tracking cameras are especially susceptible to line-of-sight tracking of body parts, which limits the reliability of data and/or the performer's available motions. Additionally, body-tracking cameras are good at tracking gross body movements but easily lose detail in smaller parts like fingers; precise finger motions are more easily tracked by physical controllers, which also add a level of haptic feedback.

Inertial hand tracking controllers sense acceleration rather than absolute spatial position; if a system references this sensor data directly, there are some non-intuitive mapping problems. A spike in controller acceleration is inevitably countered by a negative spike representing deceleration; an interface must compensate for this or intentionally contextualize

[24] Ben Lang, "Sixense STEM VR Motion Controller Kickstarter is Live – Prices Start at $149," *Road to VR*, September 12, 2013, http://www.roadtovr.com/sixense-stem-vr-motion-controller-kickstarter-prices-149-tiers.

the oscillating acceleration data. As a VR controller, inertial tracker data must be translated into spatial position by calibrating them to a reference point; however, the lack of a reference sensor makes them prone to drifting out of calibration. Hand calibration would cancel out any offset from body movement so an additional positional sensor is necessary for tracking the central body position.

Virtual Stage requires a minimum of two hand tracking controllers and the rotational tracking of the VR headset. Presently, it operates with the Hydra controllers and the Oculus Rift Developer Kit 2 (DK2). This combination tracks absolute hand position relative to the base sensor and head rotation. An additional sensor tracks absolute head positions, but due to the Hydra's limitation on body movement and rotation, head position is relatively static. Using these devices provides an upgrade path to the STEM System and a higher video resolution headset as they become available.[25]

2.4 Software Technical Considerations

These hardware considerations influence the possible solutions for building the system on the computer. The assumption of maximizing computing and graphics power as much as logistically possible limits the project to a Windows PC, rather than a Macintosh or a smartphone. Following this same assumption, the hardware developers optimize their drivers for PC, making other platforms even more difficult to implement. Since designing an entire software platform from scratch is not feasible for this scale of project, the two environment/engines considered with available VR resources include the Unity game engine

[25] The Rift DK2 maximum video resolution is 1920x1080 while the Rift CV1 (Consumer version 1) is 2160x1200. While this would increase the visual experience for the performer, it also increases the processing overhead for the PC. The performer's quality of immersion is important but not a primary concern.

and Cycling ’74’s Max audio-visual programming environment.[26]

Unity is one of the most popular development platforms for VR because it has built-in support for most VR device drivers and an extensive community for sharing and marketing software assets between developers.[27] During the initial development period for *Virtual Stage*, Unity released Unity 5 and changed their subscription model to allow free access to advanced features that were previously quite expensive for individual developers.[28]

Unity is relatively limited in its digital audio signal processing capabilities, compared to popular computer music platforms.[29] There are some additional software assets to overcome these limitations but it appeared more efficient to develop a method for passing data between Unity, which would handle interface and graphics, and Max, which would handle audio. Open Sound Control is a protocol that facilitates data transfer between platforms or computers with minimal latency, which is necessary for real-time audio response.[30] I developed and tested an interface for the Open Sound Control format that would facilitate this data transfer, which was primarily based on Jorge Garcia Martin’s UnityOSC.[31] Additionally, I was successful in integrating this with InstantVR Advanced, a software asset that facilitates controller drivers with Unity and

[26] Unity Technologies. (2015). Unity (Windows version 5.1) [software]. Available from http://unity3d.com.
[27] “VR Overview,” *Unity*, accessed November 15, 2016, http://unity3d.com/learn/tutorials/topics/virtual-reality/vr-overview.
[28] John Riccitiello, “Unity 5 Launch,” *Unity Blog*, March 3, 2015, http://blogs.unity3d.com/2015/03/03/unity-5-launch.
[29] “Audio in Unity 5.0,” *Unity Documentation*, accessed November 10, 2016, http://docs.unity3d.com/Manual/UpgradeGuide5-Audio.html.
[30] “Introduction to OSC,” *opensoundcontrol.org an Enabling Encoding for Media Applications*, accessed November 15, 2016, http://opensoundcontrol.org/introduction-osc.
[31] See section II.2.24, based on Jorge Garcia Martin’s UnityOSC code.
Jorge Garcia Martin. (2015). UnityOSC (Version 1.2) [C# code]. Available from http://github.com/jorgegarcia/UnityOSC.

establishes the framework of an on-screen performer needed in *Virtual Stage*.[32]

Although these developments were successful, Unity's rendering output conditions were limited at that time. Rendering a first person view to the VR headset and a separate third-person display for the audience required either two, networked instances of Unity, or scripts that interrupt the rendering pipeline to create an additional output window; both of these situations would have added an unforeseeable amount of development complication and processing overhead. Though this limitation was resolved in subsequent release versions of Unity, it highlighted the limitations in Unity's features and documentation that would impair development.[33]

Around the time of these developments, users on the Cycling '74 forum began to solve integration problems for VR headsets directly in Max.[34] Although Unity has a more advanced rendering engine, I was still developing in Max for other projects so the benefit of keeping the entire project on one platform was even more appealing. The switch to Max also opened more possibility of providing cross-platform support on Macintosh, which would generally help with development, even though the main goal was to utilize a PC to run the performance software. However, later integration development of the Rift in Max created a tradeoff between allowing current Rift drivers and restricting the VR interface to Windows.[35]

The decisions on software platforms were a combination of the technical needs of the

[32] Passer VR. (2015). InstantVR Advanced (Windows version 3.1.0) [Unity package]. Available from http://passervr.com/products/instantvr-advanced.
[33] "Multi-display," *Unity Documentation*, accessed November 16, 2016, http://docs.unity3d.com/Manual/MultiDisplay.html.
[34] "Oculus Rift," *Cycling '74 Forums*, accessed September 5, 2015, http://cycling74.com/forums/topic/oculus-rift/#.WEEiveYrKUk.
[35] Graham Wakefield. (2017). Max_Worldmaking_Package (Windows commit 93e573a) [Max external]. Available from http://github.com/worldmaking/Max_Worldmaking_Package.

piece, the logistical needs of development, and projected hardware and software availability. The chosen hardware is possible on either Unity or Max, but there must be a judgment between the benefits of a platform and how heavily the development difficulties would mitigate those benefits.

2.5 Modulation of Virtual Instruments

The virtual instruments in *Virtual Stage* are designed modularly to maximize the possible combinations of instruments and increase the observability of their behaviors. Functions are represented visually and spatially so the event that represents a point of influence between functions is a spatial collision. The performer combines dynamic objects (influenced by physics forces) and kinematic objects (manually or automatically positioned) to sequence the collisions intentionally, incidentally, or accidentally. Collisions in Max are output as a dictionary object that indicates which objects are colliding, the velocity of impact, and the spatial properties of the collision point.[36] However, if the objects remain in contact on successive simulation frames, the simulation reports a new dictionary each frame.

This demonstrates a frequent issue in *Virtual Stage* and other systems — the difference between an event onset and a state toggling on. Some functions, like the instantiation of new permanent objects, exist only as an event, whereas other functions influence objects only when they are toggled on, like a field activating a force. If physics collisions can activate either type of function, the question is whether the state is retriggered by successive collisions (Figure 2.6).

[36] Max dictionaries are multi-dimensional data structures that hold symbol and number data. "Dictionaries – Structured Data," *Max 7 Documentation*, accessed December 7, 2016, http://docs.cycling74.com/max7/vignettes/dictionaries.

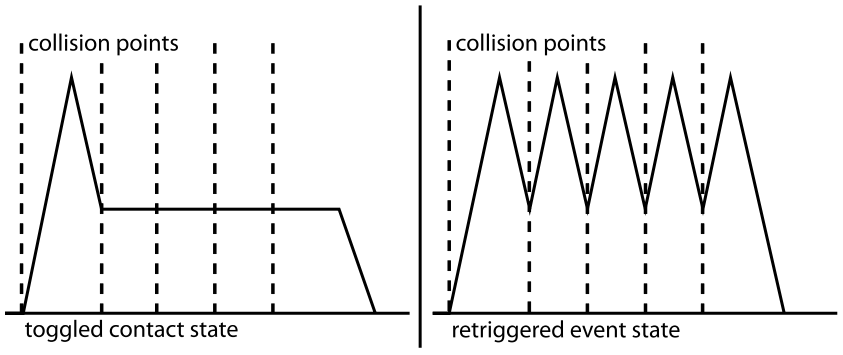


Figure 2.6 Toggled vs retriggered states over time

A toggled contact state is similar to an audio ADSR envelope in that there is a threshold of time before collisions would be allowed to retrigger the state, rather than sustaining the toggle.[37] The choice between these two models is useful because it allows for either type of contact to be represented by an audio signal; the contact between the two is mapped as an index of modulation (Figure 2.7).

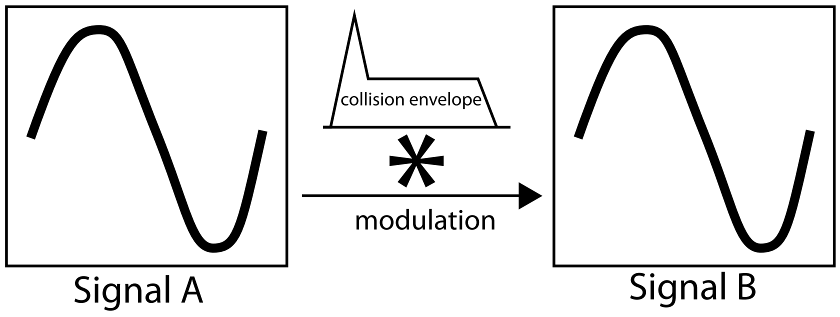


Figure 2.7 Collision envelope as a modulation index

This method works on audio output signals but also applies to physical models. For example, both models could have a continuous audio output, but the collision state modulates the routing level from model A to model B.

[37] ADSR – attack, decay, sustain, release envelope. This is typically used to modulate the amplitude of another audio signal, segmenting it into discreet events.

If every possible collision between two objects is a potential routing connection, it would be impractical to have a massive routing matrix because the majority of the routing connections would be inactive most of the time. A better method is for each collision to activate a polyphonic voice that busses the signal from input to output module for the duration of the collision envelope. This allows for either the retriggered or toggled type of collision, because it only requires that a message is sent to the envelope voice instructing it how to react to retriggers.

Although Max physics simulations mostly consist of rigid body dynamics, they can also track contact between overlapping kinematic objects. If the objects are not repelled by collision dynamics, it is interesting to track their difference in spatial position and rotation as a continuous modulation signal (Figure 2.8).

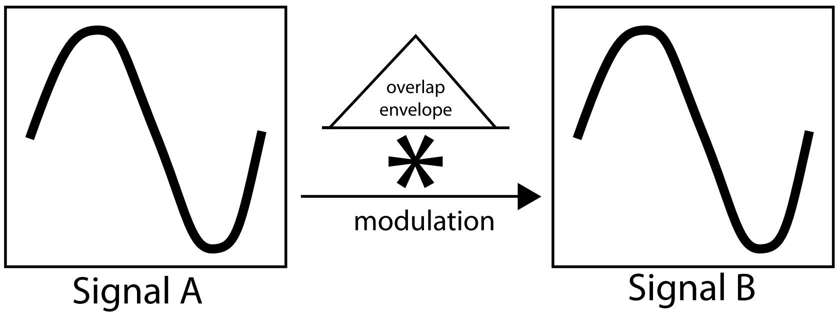


Figure 2.8 Overlap modulation as continuous control

Treating each model as a continuous audio signal at least allows for amplitude modulation, but collisions or overlaps between different types of objects can modulate different types of parameters, creating a variety of sonic effects with a modular structure.

3. Virtual Stage

3.1 Environment Description

Virtual Stage appears as a wide, empty landscape in which the performer summons instances of interactive objects. The virtual performer flies around the space, sets up combinations of functions and intentionally (or accidentally) acts upon them to create electroacoustic sounds (Figure 3.1).

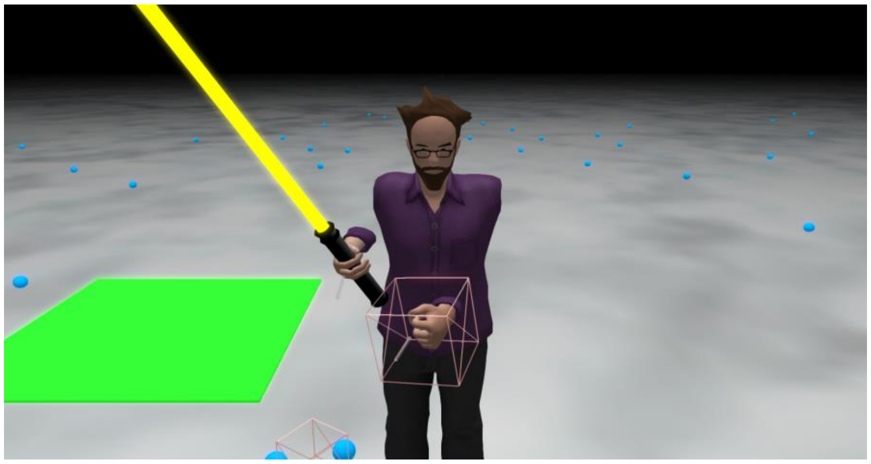


Figure 3.1 *Virtual Stage* third-person view

The performer's body is correlated to the input from the VR controllers, such that the viewer should recognize the virtual performer as the performance agent of the piece, rather than watching the live performer directly. Because I am the intended performer of the piece, the virtual performer is specifically designed to resemble me, in features and dress; this increases the viewer's immediate ability to recognize the correlation between on-stage and virtual performer.[38]

The perspective of the audience is modulated in that they are subject to the orientation

[38] See section I.4.1 for details on body scanning and mapping.

of the camera in the virtual space. The performer can manipulate the camera or there are presets for tracking the performer with camera distance and rotation parameters (Figure 3.2).

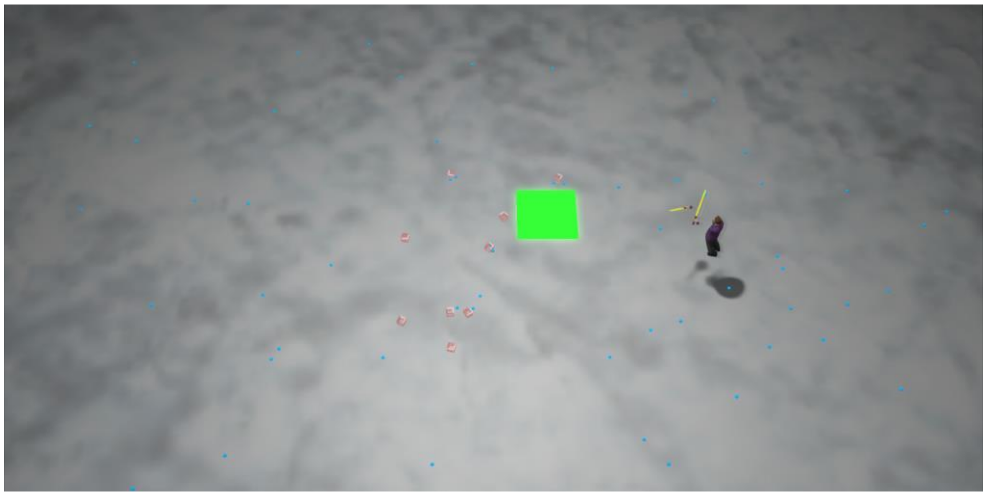


Figure 3.2 Third-person camera zoomed out

Additionally, the performer controls the viewer's video fade, similar to an on-stage lighting cue. This allows for clearer segmentations in the performance and virtual setup time between events.

Graphical manipulations to the viewer's video can be activated by performance functions. For example, motion blur and lighting adjustments are included as part of the general visual aesthetic so they can be modulated for dramatic effect (Figure 3.3).

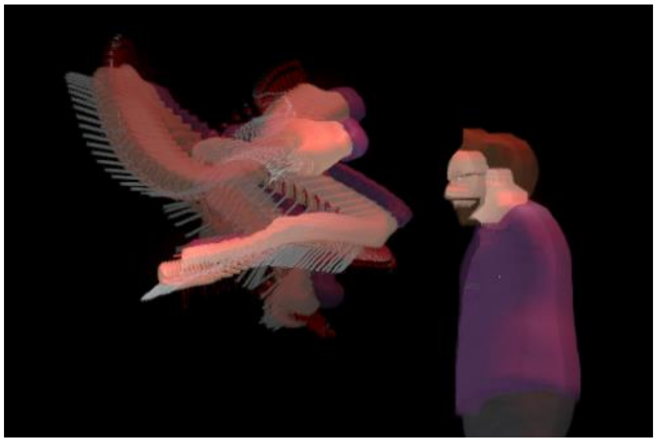


Figure 3.3 Motion blur effects on third-person view

The audio is directly correlated to the activation of physically modeled virtual instruments. The main audio instrument is a 3D Slab that is struck and spatially manipulated. Object collisions to this Slab are sonified with spatial position relative to the third-person camera (Figure 3.4).[39]

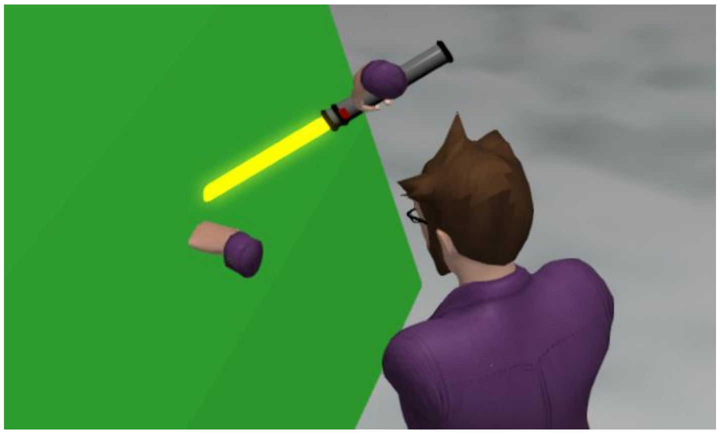


Figure 3.4 Sword collisions to sonify physical model via Slab

[39] It would be possible to provide the performer with a first-person spatialization of the audio, but unlike the video perspectives, it is difficult to simultaneously monitor both audio perspectives. Calculating audio spatialization twice also considerably increases audio processing costs.

3.2 Graphical User Interface

The performer’s graphical interface is a first-person view of the virtual environment, which allows the performer to feel closely connected with the performance materials. There is also a functional benefit to more naturally looking around the environment, in that the hand controllers are not constrained to the line of sight; this allows the hands to function independent from vision and for the performer to multi-task more efficiently.

Although the VR headset has a limited viewing angle and screen space must be treated economically, some heads-up display (HUD) features are used to extend parts of the interface that are not represented spatially (Figure 3.5). The performer sees a miniature version of the third-person camera in the HUD so they can optimize camera angles; without being able to easily look at the screen the viewer sees, this keeps the performer aware of the audience’s experience of the piece.

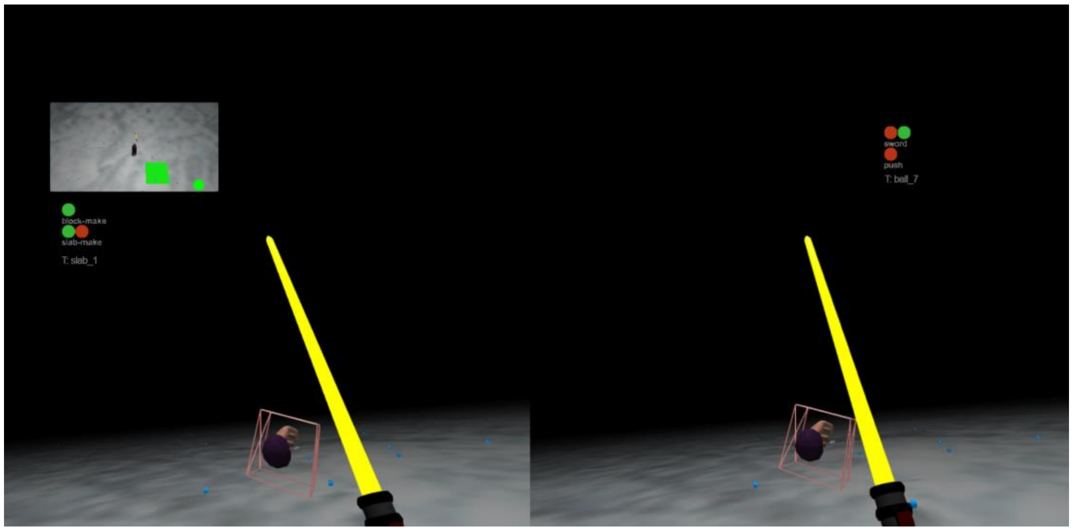


Figure 3.5 User interface on headset[40]

The HUD third-person monitor also displays a green or red dot in the corner, indicated whether the viewer's display is faded up or down. This allows the performer to position the camera and any visible objects, while still being able to see what will be in frame when the view is faded up (Figure 3.6).

[40] The performer's display appears as a split-screen because this is the signal sent to the VR headset. Because the HUD is not mapped to both sides of the split-screen, they appear in only one eye and don't completely block that part of the visual field.

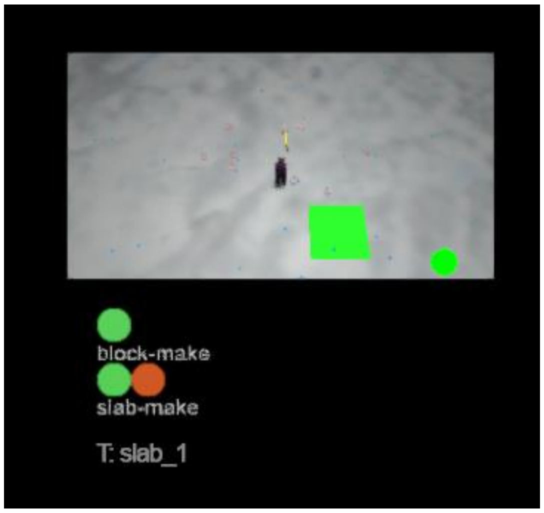


Figure 3.6 Performer’s heads-up display — left eye

The HUD displays the current spell for the hand on each side, as well as the spell that is currently being queued up (Figure 3.6). This allows the performer to verify the intended function before activating it; the HUD also verifies that the queued spelling is a valid function, and displays “XXX” if the entered spell is not in the library.

The bottom of the HUD displays the hand’s active target, which changes the behavior of certain spells. If the target is manually cleared or the target object is no longer active, the target indicator clears and any targeting spells do not activate or default to the position of the triggering hand.

3.3 Controller Interface

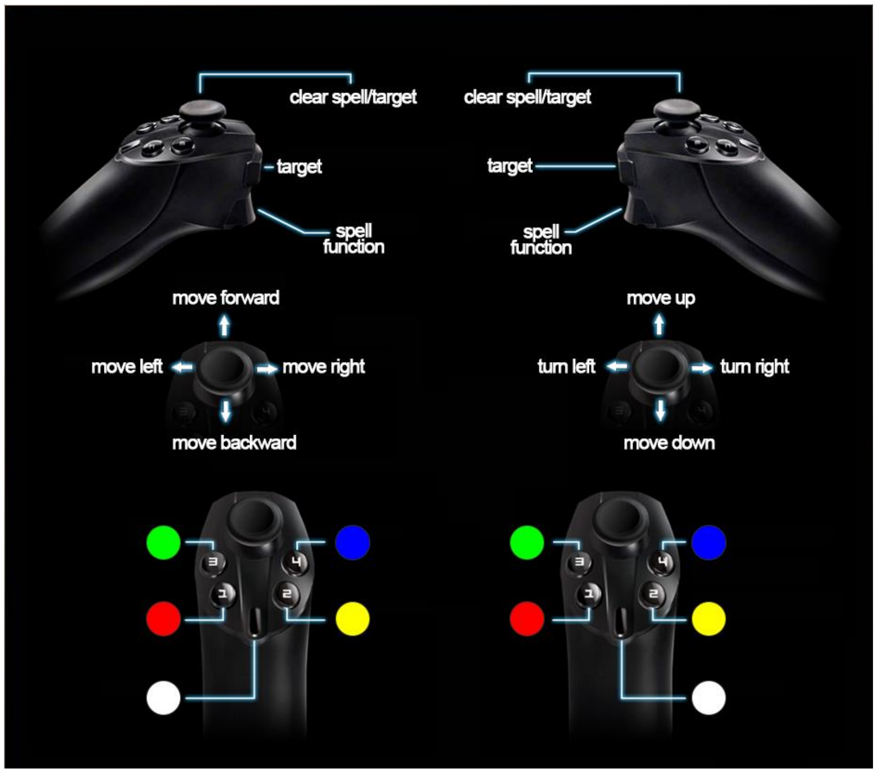


Figure 3.7 Hydra controller interface[41]

The controller interface is designed to maximize the efficiency of spell queueing and enable each hand to independently queue and activate spells (Figure 3.7). When entering a color sequence to queue a spell, the performer's HUD shows the spell that is ready or indicates if the spelling is not valid. Pressing the joystick in clears the selection, allowing the spelling to be restarted; pressing the joystick a second time will also clear the current spell. When a spell is queued, pressing the trigger activates the spell. Some spells use the trigger to toggle or have an instant effect when the trigger is depressed, while other spells can be partially activated by depressing the trigger as a continuous controller. The bumper button chooses a target object,

[41] Adapted from Sixense MotionCreator graphic for Skyrim control scheme.
"Razer Hydra Control Profile – Skyrim," *Sixense*, accessed December 5, 2016, http://sixense.com/skyrim.

which is selected by collision with the target sensor projecting from each hand; holding the bumper and pressing the joystick clears the current target. If multiple targetable objects are in contact with the target sensor, the bumper will cycle through the possible targets.

The joysticks are used to navigate the virtual performer around the space. The left joystick changes the body position and the speed can be regulated by pushing the joystick partially. The right joystick moves the body up and down and turns the body left and right. Since the Hydra controller restricts the live performer to the same rotation orientation, the joystick is necessary to be able to rotate the virtual performer. If the control interface is upgraded to a wireless solution, the performer would have freedom to rotate with their body and the right joystick X-axis could be used to rotate on a different axis, allowing the performer to spin upside-down.

Because the joystick and color buttons are depressed by the thumb on each hand, there are some situations where it is difficult to navigate while queueing combinations of spells. One of the benefits of being able to use either hand for any spell function is that the performer can optimize their hand selection to account for necessary joystick manipulation.

3.4 Software Interface

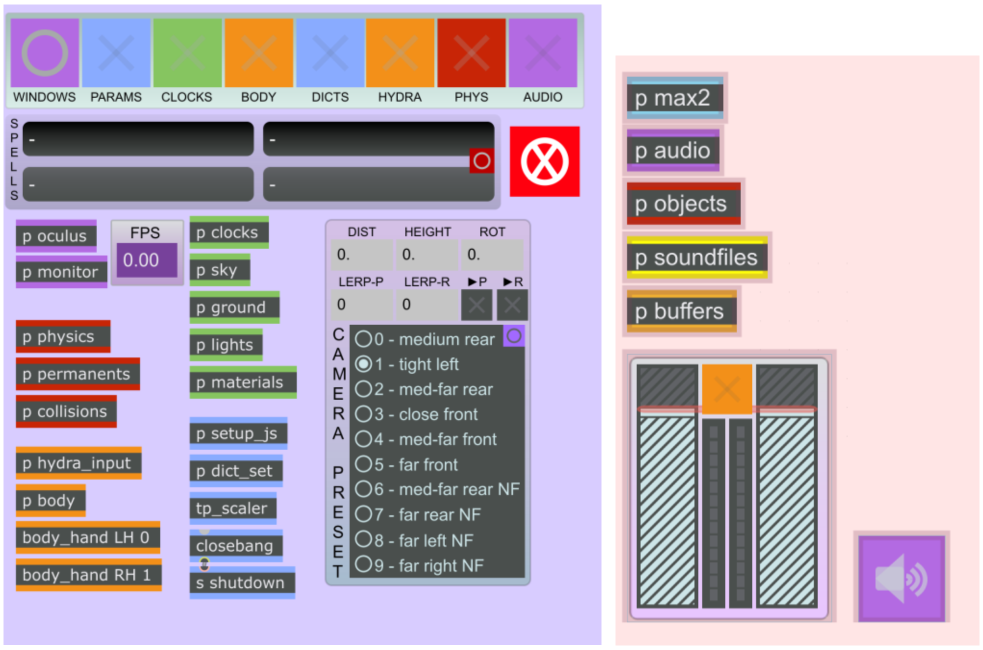


Figure 3.8 *Virtual Stage* Max GUIs

The software interface is designed to facilitate development and economize screen space while providing access to sub-sections as needed. The top row of toggles are executed in order on startup to activate sections of the software. This is useful in debugging because it segments parts of the Max patch and isolates problems occurring on startup. It can also be useful during testing to shut sections off, like the controller input, so that that test messages can be inserted.

The “SPELLS” display duplicates the function of the HUD spell display, because it is impractical to wear the headset all the time during software development. This GUI displays each hand’s current and queued spell, along with a button to re-scan the spell library.

The audio section of the software runs in a separate instance of Max, which helps optimize multithreading and protect latency in video and physics processing from interfering with audio triggering. Similarly, sections are opened separately to conserve screen space. A master audio slider and meter could be monitored by a technician during performance, but is primarily available for testing.[42]

3.5 Spell Functions

The five buttons to queue spells correspond to both the color in the designated spell sequence and a categorical representation of the spell's theme. These categories are loose and not all-encompassing, but help the performer to encapsulate spells with visual cues. For example, green-red summons a Slab, which is a permanent object that the performer can inflict force on, while red-green equips a Sword, which is a held instrument for inflicting force on permanents. There are other force instruments, but the Sword is has the simplest purpose (Figure 3.9).[43]

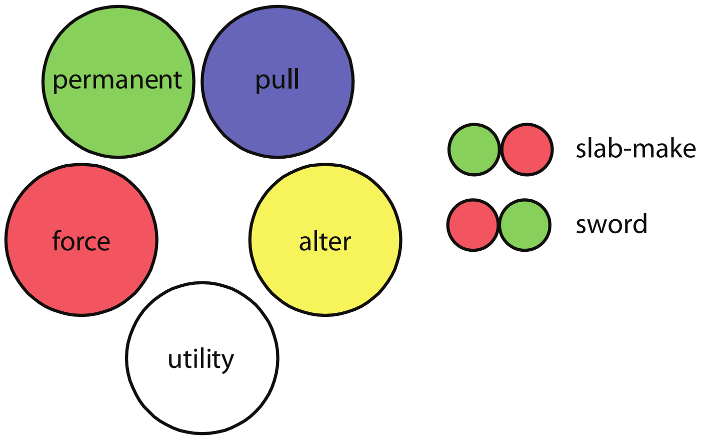


Figure 3.9 *Virtual Stage* color symbols with examples

The permanent (green) category instantiates objects in the world that use the physics

[42] See section I.4.5 for technical details on function spelling.
[43] See section II.1.1 for details on all spell functions.

engine. The electroacoustic sounds are created by physically modeled interactions between these physics objects so this category also directly relates to audio generation.

The force (red) category includes spells that create repulsion force on physics objects, as well as equipped instruments that strike or influence permanents.

The alter (yellow) category has spells that change the properties of permanent objects, which is reliant on the hand targeting. Alter also includes fields that will influence object properties while they are in contact. Fields can be attached in space or can be attached to permanent objects, tracking to their position.

The pull (blue) category creates attraction force fields for physics objects. This can be used to hold groups of objects together and then throw them. This category also includes some hand held instruments similar to the force category.

The utility (white) category contains any extra functions that don’t specifically affect the environment. For example, the Recenter function can be used to calibrate and zero out the controllers and headset position.

4. Technical Context

4.1 Body Scanning

With the goal of an immersive experience for the performer and a visual representation displayed to the audience, body scanning and 3d rigging techniques were employed to create a virtual avatar that visually represents the performer's appearance.[44] The process uses a Microsoft Kinect v2 camera to do a depth scan of my physical body — via Body Labs' BodySnap software (Figure 4.1).[45]

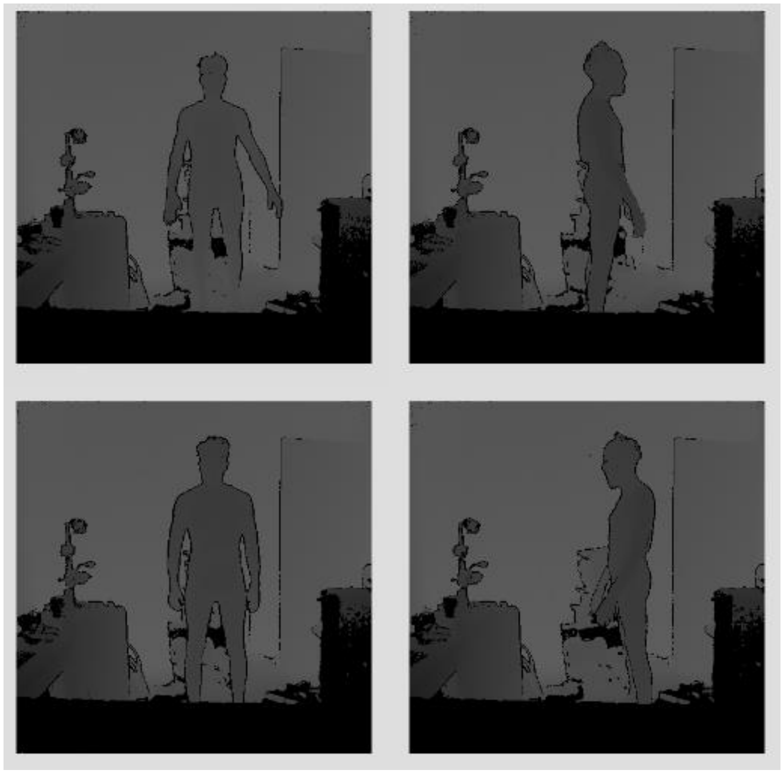


Figure 4.1 BodySnap body depth scanning for virtual avatar model

[44] See section I.3.1 for why my personal proportions and appearance are used for the virtual avatar.
Chantel Benson, "Create a 3D Avatar of Yourself with Kinect for Windows v2 and Fuse," *Mixamo Blog*, August 7, 2014, http://blog.mixamo.com/how-to-scan-yourself-into-a-game-ready-3d-model-using-kinect-and-fuse.
[45] Julia Gilroy, "Kinect Your Body," *Bodylabs*, December 11, 2014, http://www.bodylabs.com/blog/2014/12/11/kinect-your-body.
Body Labs. (2015). BodySnap. (Windows version 16) [software]. Available from http://bodylabs.com.

This software exports an OBJ model of the body shape with proportions relative to the performer's body. Mixamo/Adobe's Fuse imports the OBJ file, scans it, and fits clothing models to the body shape and textures 3d materials on components of the model (Figure 4.2).[46]

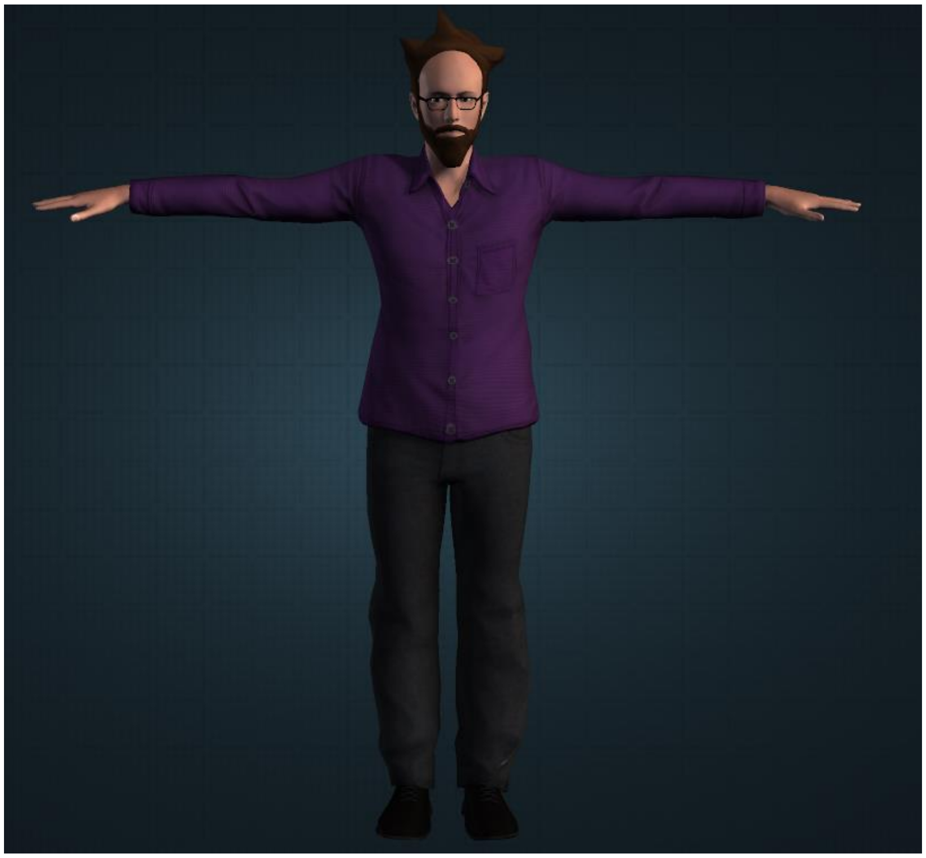


Figure 4.2 Mixamo Fuse – adding clothing to OBJ model

However, Fuse's apparel library is limited so the OBJ is exported to Blender to edit the default hair and eyewear components, increasing the resemblance to the physical attributes (Figure 4.3).[47]

[46] Mixamo, Inc. (2015). Fuse (Windows version 1.3) [software]. Available from http://mixamo.com/fuse.
[47] Blender.org. (2016). Blender (Windows version 2.78) [software]. Available from http://blender.org.

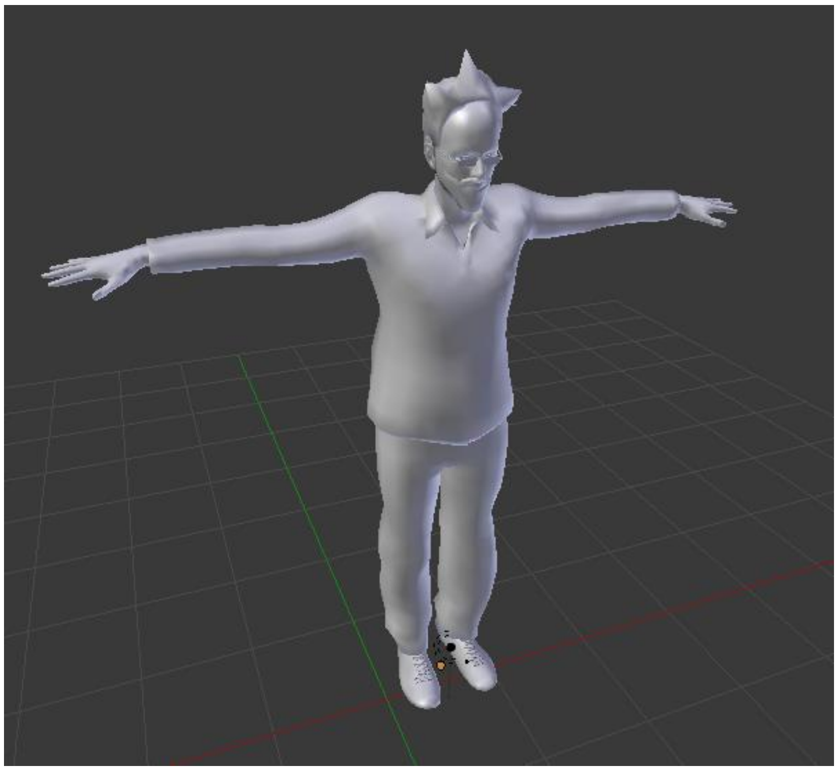


Figure 4.3 Edited hair and eyewear models in Blender

The adjusted OBJ is uploaded to the Fuse online platform for automatic animation rigging. This maps a weighted armature to the vertices of the model, effectively creating a skeleton structure that deforms the model for animation (Figure 4.4).

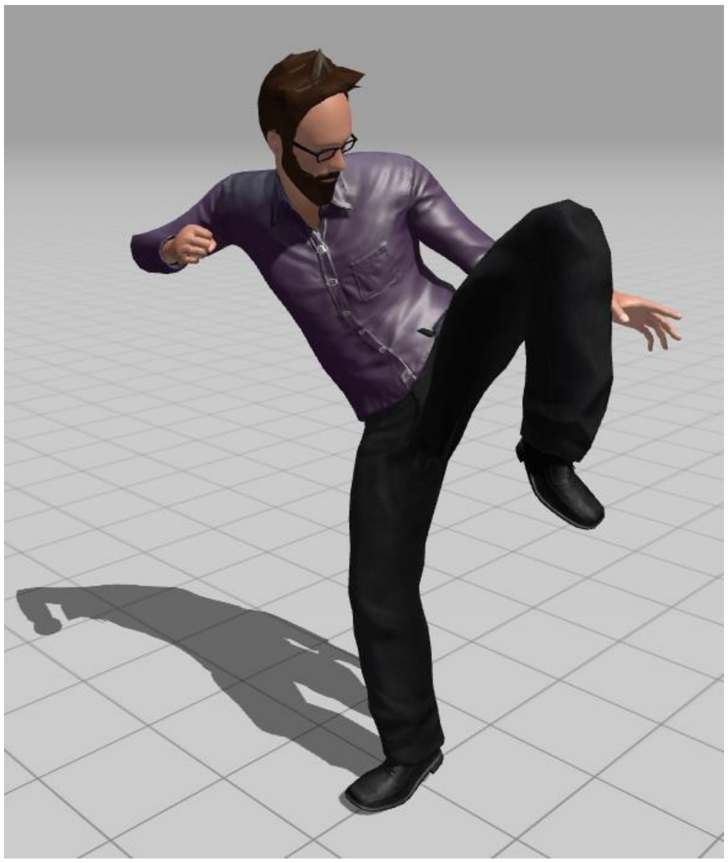


Figure 4.4 Rigged model in Fuse online interface[48]

Fuse can export multiple 3D formats, but Collada DAE is the best file format to include animation rigging data to import into Max. This also allows the packaging of material and texture data into the file, simplifying the Max material configuration. However, Max does not use the same axis orientation as Blender so the armature and model must be rotated in Blender and exported to swap the Y-axis and Z-axis (Figure 4.5).

[48] Note that the online platform does not import customizations in 3d material shaders so this screenshot appears more glossy than in the editing software and other platforms.

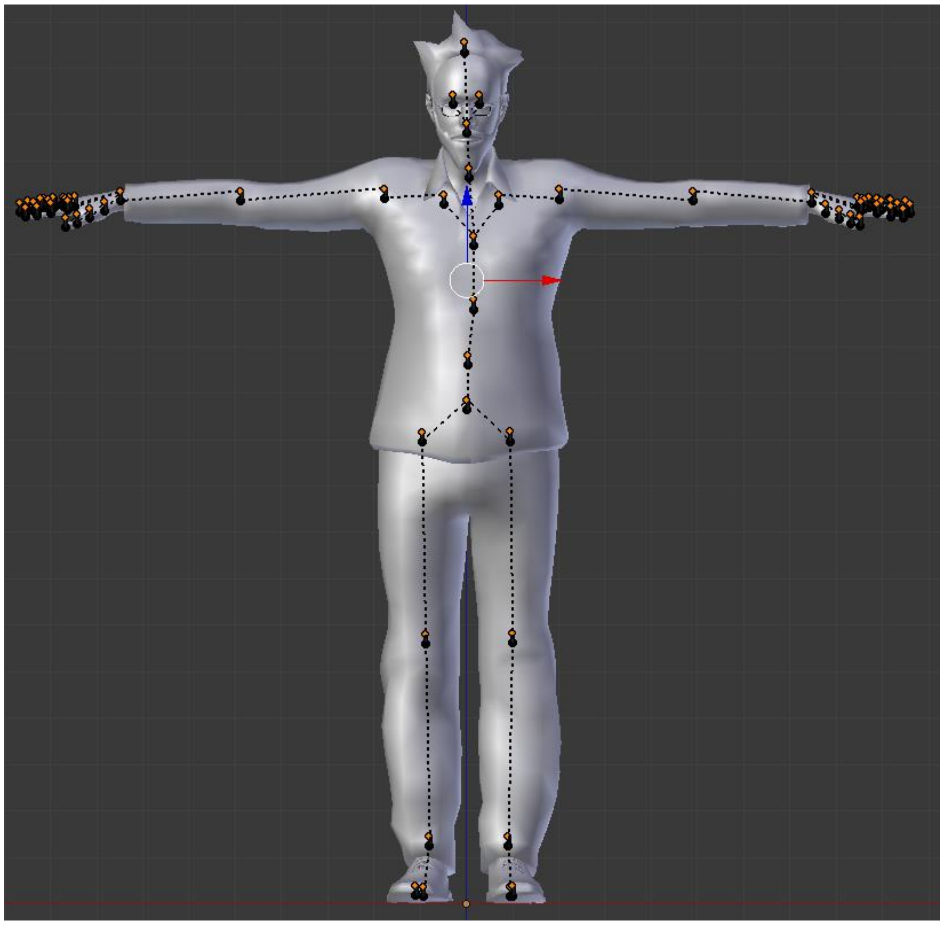


Figure 4.5 Rigged model in Blender

This axis swapping is problematic because the controller data is represented by a 4-dimensional quaternion rotation that needed axis adjustment, which cannot be directly rotated or swapped as a single axis in a 3-dimensional Euler rotation. Applying rotations to the armature and model rotations before importing into Max maintains the appropriate visual axis orientation (so that the models head points up). Before the controller data maps onto the animation points, they are converted to Euler rotation values and rotated to align properly to the model.

With the use of hand controllers, only two animation nodes are tracked but this requires animation of the arm nodes. This requires an Inverse Kinematics (IK) calculation, which retroactively calculates armature positions based on the positions of nodes hierarchically down the skeleton. Max does not have a built-in IK solver so the suggested IK solution is to accomplish this is to create physics constraints that link the nodes together and solve using rigid body physics.[49] Testing of this method encountered problems: calculating the constraint positions and forces becomes increasingly complicated when also scaling the body model; there is a tradeoff between convincing physical motion and a stable physics simulation; and this physics solver does not appear as realistic as most IK solvers. Because of these, the arms were removed from the model and the tracked hand positions float, detached from the body; this eliminates inconsistent animation of the arms, with an assumption that it is less distracting and less computationally expensive.[50]

4.2 Controller Software

The Razer Hydra controllers track motion in 6-degrees-of-freedom (6dof) and have multiple buttons and analog inputs on each controller.[51] There is no Windows Max external to track these parameters so the use of the controllers requires a Windows software application to pull data from the controller drivers and stream it to Max.

The Sixense Software Development Kit (SDK) includes the driver files and an Application Program Interface (API) for developing an application using the Hydra controller data.[52]

[49] "Physics Patch-a-day," *Cycling '74*, accessed November 15, 2015, http://cycling74.com/2012/09/19/00-physics-patch-a-day.

[50] After these development decisions, Caliko, an Inverse Kinetmatics Java library was released. This has been tested using Max's [mxj] Java object and is not computationally prohibitive and can interface properly with a rigged Collada model. However, coordinating the geometry axes of the Max/Jitter space, the model joint rotations, and the Caliko structure space is not trivial. This requires further development but should allow for a reintegration of the 3d body's arms.
Alastair Lansley, et al., "Caliko: An Inverse Kinematics Software Library Implementation of the FABRIK Algorithm," *Journal of Open Research Software* 4(1), e36, accessed December 25, 2017, DOI: http://doi.org/10.5334/jors.116.

[51] 6-degrees-of-freedom (6dof) refers to the axes of position and rotation, without being inhibited by 3d calculation problems like gimbal lock.

[52] Sixense Entertainment, Inc. (2012). Sixense SDK (Windows version 102215) [software library]. Available from http://sixense.com/windowssdkdownload.

MrMormon on the Sixense Forum developed a Win32 application that utilizes these drivers this purpose.[53] This application also makes use of the OscPkt library, which is an efficient solution for streaming Open Sound Control (OSC) parameter data to a network port.[54]

When retrieving data from a network port, Max requires a polling interval to schedule the port query.[55] The Hydra drivers output 4-dimensional quaternion rotation values (to allow for 6-degrees-of-freedom) and streams each of these values to a separate OSC register. Because Max's polling interval is not synchronized to the rate of OSC streaming, there are intermittent discontinuities in the quaternion values.[56] This required recompiling MrMormon's Win32 application to pack the quaternion values into a single message list, enforcing quaternion axis synchronization in Max.[57] This also allows preemptive optimization of the register symbols to efficiently route values in Max.

Max splits the individual controller parameters in a routing patch and assigns to [value] registers, which can be referenced by any other module in Max.[58] This patch also parses positional and rotational velocity and acceleration of the 3-dimensional Euler values, which is also assigned to [value] registers.

4.3 Headset Software

Although there were early tests for using the Oculus Rift in Unity, the implementation in

[53] "The Most Versatile Musical Instrument, *Sixense Forum*, accessed February 10, 2016, http://sixense.com/forum/vbulletin/archive/index.php/t-2870.html.
[54] Julien Pommiers. (2015). OscPkt (Windows version 1.2) [C++ OSC library]. Available from http://gruntthepeon.free.fr/oscpkt.
[55] See section II.2.13 for Hydra OSC parsing in Max.
[56] This cannot be solved by directly interpolating the dimension values because quaternion rotation requires a smooth rotational transition between 0 and 360 degrees.
[57] See section II.2.1 for Hydra Win32 application code.
[58] See section II.2.14 for Hydra data routing in Max.

Max is facilitated by the [oculusrift] external by Graham Wakefield and discussed on the Cycling '74 forum.[59] The multi-window setup required in *Virtual Stage* uses the [jit.gl.node] Max object to hierarchically structure video render nodes.[60]

This requires that [jit.gl.render o] is the parent render context and [jit.gl.node o @name world] renders inside of it. All of the environment's 3d objects render inside the "world" node. Separately, [jit.gl.node o @name o_monitor] renders all of the 2d HUD context like text and spell colors, as well as a [jit.gl.videoplane] with the third-person HUD camera. These are alpha-blended with transparency to overlay on the first-person cameras textures and the HUD together onto the headset display.

4.4 Render Contexts

The additional render context for the third-person camera also implements post-processing effects to make it more visually appealing.[61] These aren't as important for the performer's first-person view, so they were removed for graphics efficiency. The post-processing effects used are: Fast Approximate Anti-Aliasing to smooth out jagged edges; Bloom-Tonemap to rescale color values and add blur to highlights; and Slide Motion-Blur, which interpolates/motion-blurs the image and can be turned up for dramatic, visual effect.

The third-person camera's [jit.gl.camera world] must be set to "@capture 1," enabling output to a GL video texture, rather than rendering directly onto the render context. This routes it to two separate [jit.gl.videoplane]s — one for the audience's video window, and one for the [jit.gl.node o @name o_monitor] node that renders on the performer's HUD.

[59] See section I.2.4 for details on *Unity* tests.
[60] See section II.2.16 for Oculus Rift render context in Max.
[61] See section II.2.17 for third-person camera and post-processing Max code.

Depth-of-field filters were tested, which would add photorealism to the third-person camera display.[62] Unfortunately, Max's GL render engine is limited in output of multiple render context textures, so it can either output depth information or alpha-blended transparency.[63] Alpha-blending is more useful in *Virtual Stage* and is required for semi-transparent objects, so this was selected over of the depth-of-field filters.

4.5 Function Spelling

The performer's input of spell functions requires a reliable system for checking button sequences against a library of possible spell combinations. This type of process is inefficient in Max because it requires a multi-dimensional data structure and functions for checking if symbol values or keys are inside the data structure. Implementing this in Max requires conditional iteration through the entire list, which is inefficient in Max.

A prototype Javscript solution stores button/color sequences and check them against the spells.json library file. Unfortunately, the Max Javascript [js] object operates in the low-priority thread so it is prone to timing error; the script was ported to a Java class for running with Max's [mxj] Java wrapper.[64] A separate Java class parses the output and draws the color symbols and spell names onto the HUD using [jit.gl.sketch] instructions.[65]

A Max patch tracks the controller input data, implements these [mxj] classes, and sends messages to the spell functions on the relevant hand.[66]

[62] Depth of field is an effect where the distance between near and far focus creates a blur effect.
[63] "Re: Re: RE: Depth of field style blur shader + global illum/soft shadow shader?," *Cycling '74 Forum*, accessed July 2016, http://cycling74.com/forums/topic/re-re-re-depth-of-field-style-blur-shader-global-illumsoft-shadow-shader/#.WEgCXOYrKUk.
[64] See section II.2.3 for the Java spell.json parser.
[65] See section II.2.2 for Java [jit.gl.sketch] drawing instructions.
[66] See section II.2.22 for Max function spell checker.

4.6 Physical Modeling

The primary audio physical model in *Virtual Stage* for electroacoustic sound is based on a waveguide string model.[67] Implementing in Max is facilitated by [gen~], which allows Max-style programming of signal processing with single-sample delays.[68] This can be implemented by creating a series of [delay] objects in [gen~], with boundary filters/coefficients, and impulse signal inputs (Figure 4.6).

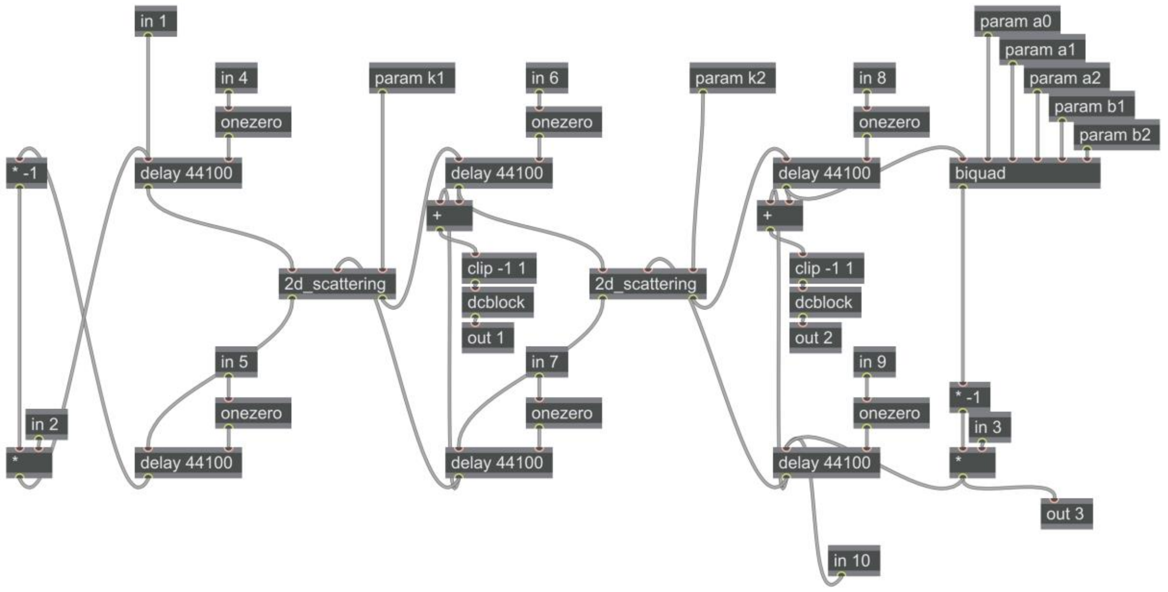


Figure 4.6 Waveguide string with two output pickups in [gen~]

However, this model is limited in that the input signal must be directly routed into the model and all input signals are synchronized to the same insertion point. Sonifying multiple impulse points on the model at simultaneously requires multiple insertion points. A solution implements the standard waveguide delay lines as feedback read/write points in a multi-

[67] Perry R. Cook, *Real Sound Synthesis for Interactive Applications* (Natick, Massachusetts: A K Peters, 2002), 97–106.
[68] "Gen~ for Beginniners," *Cycling '74 Wiki*, accessed December 7, 2016, http://cycling74.com/wiki/index.php?title=gen~_For_Beginners.

channel circular buffer. A [phasor] cycles through the buffer and the sample distance between the read and write points determines the resonant frequency of the feedback.[69]

This model assumes the phasor steps through the buffer sequentially; however, if the write-point is modulated at the audio rate to create modulation sidebands, it skips over buffer indices and causes audible signal discontinuities. Detection of discontinuity in buffer writing and conditional interpolation fills in skipped indices faster than the audio rate.[70] High-frequency modulation correction is computationally expensive and causes audio glitches that are worse than the writing discontinuities. The solution limits the maximum interpolation steps and relies on constrained modulation frequencies to prevent distortion from discontinuities.

With this physical model, any external audio module may write into the buffer channels.[71] A collision detector polyphonically assign voices to write into the buffer at different insertion points, simulating impulses at different points of the string. A global phasor synchronizes the model's buffer read/write points the delay size is recorded to a global buffer; this is referenced to synchronize the impulse's write point against the delay line.

This configuration allows for a more realistic representation of sounding object (string) and impulse actuator (plectrum/mallet). However, a waveguide string does not sound especially interesting when modulating the insertion point unless something else is manipulating the delay line. A separate module scatters signal across two string models with separate tuning, resulting in more effect from multiple impulse positions.[72]

[69] See section II.2.5 for [gen~] patch with buffer writing waveguide string.
[70] See section II.2.6 for interpolating buffer write [gen~] patch, based significantly off Graham Wakefield's work.
[71] See section II.2.7 for physical model buffer poker.
[72] See section II.2.8 for cross-physical model scattering.

This method for a physical model is modular because additional impulse sources can be developed to activate the model without creating new versions of the waveguide. It is robust at creating complex timbres when modulating parameters with audio signals, and allows signal-rate modulation of multiple parameters. This is the basis for electroacoustic sound in *Virtual Stage* because each audio module is represented by a discreet object in 3d space. Mapping spatial parameters of the objects onto the physical parameters of the models allows multiple methods to drive the same type of model and a visible correlation to the sonification.

5. Conclusions

Modular design facilitates the creation of a virtual environment for digital interaction by integrating the audio and video structures. Physically modeled digital instruments have a capacity for nuanced control and visualizing them spatially creates a representational framework for meaningful interactive gestures. Additionally, a visualized interface of sound increases the viewer's potential for understanding the interactive vocabulary. This approach combines and blurs the relationship between performance space and instrumental interface. *Virtual Stage* enhances the possibilities of perceived interaction in digital systems; the viewer's understanding of the environment relies on their connection to the performer's interactions and is heightened by the behavioral process.

The future development of *Virtual Stage* will expand the number of interactive functions to expand the interactive possibilities. Advancements in computational power and graphics processing will allow for a denser population of virtual objects and a wider array of simultaneous interactive possibilities. Available consumer-grade virtual reality control and display hardware continues to increase in accuracy and reliability; this will increases the immersive potential for the performer and make it more viable to extend the system to other computer platforms. These advancements could also lead to potential for multi-user performances.

The additional possibilities of interactive performance understanding should challenge both the performer and the viewer to evaluate their conceptions of physicality and observation. It is difficult to maintain a coherent idea of behavioral structure when data is abstracted and manipulated by digital processes. Ultimately, this challenges us to develop our understanding of space and time, leading to new ideas and problem solving methods. Immersive interfaces and open structures are tools that reinforce these developments. However, they rely on our own thoughtfulness and attention to expand our awareness in the world.

6. Bibliography

Abbas, Niran B. “Narrative as Virtual Reality: Immersion and Interactivity in Literature and Electronic Media by Marie-Laure Ryan,” *The Yearbook of English Studies* 34 (2004): 277–278.

“About Minecraft,” *Official Minecraft Wiki, Gamepedia*, accessed November 10, 2016, http://minecraft.gamepedia.com/Minecraft_Wiki.

“Audio in Unity 5.0,” *Unity Documentation*, accessed November 10, 2016, http://docs.unity3d.com/Manual/UpgradeGuide5-Audio.html.

Belinkie, Mathhew. “What Makes Minecraft So Addictive,” *Overthinking It*, November 11, 2010, http://www.overthinkingit.com/2010/11/11/minecraft-vs-second-life.

Benson, Chantel. “Create a 3D Avatar of Yourself with Kinect for Windows v2 and Fuse,” *Mixamo Blog*, August 7, 2014, http://blog.mixamo.com/how-to-scan-yourself-into-a-game-ready-3d-model-using-kinect-and-fuse.

Bernardini, Nicola. “Should Musical Instruments Be Dreams?,” *Computer Music Journal* 15, no. 4 (1991): 78–81.

Bizri, Hisham. “Story Telling in Virtual Reality,” *Leonardo* 33, no. 1 (2000): 17–19.

Blender.org. (2016). Blender (Windows version 2.78) [software]. Available at http://blender.org.

Body Labs. (2015). BodySnap. (Windows version 16) [software]. Available at http://bodylabs.com.

Brown, Richard D. “Virtual Unreality and Dynamic Form: An Exploration of Space, Time and Energy,” *Leonardo* 33, no. 1 (2000): 21–25.

Cadoz, Claude. “The Physical Model as Metaphor for Musical Creation: “pico..TERA”, a Piece Entirely Generated by Physical Model,” *Proceedings of International Computer Music Conference* (2002): 305–312.

Cai, Y. Y., B. F. Lu, Z. W. Fan, C. W. Chan, K. T. Lim, L. Qi, and L. Li. “Proteins, Immersive Games and Music,” *Leonardo* 39, no. 2 (2006): 134–137.

Christensen, Damaris. “Sculpting Virtual Reality” *Science News*, 156, no. 12 (1999): 184–186.

"Color Pie," *MTG Salvation Wiki, Gamepedia*, accessed November 11, 2016, http://mtgsalvation.gamepedia.com/Color_Pie.

Cook, Perry, "Physically Inspired Sonic Modeling (PhISM): Synthesis of Percussive Sounds," *Computer Music Journal* 21, no. 3 (1997): 38–49.

Cook, Perry R. *Real Sound Synthesis for Interactive Applications.* Natick, Massachusetts: A K Peters, 2002.

Coyne, Richard. "Heidegger and Virtual Reality: The Implications of Heidegger's Thinking for Computer Representations," *Leonardo* 27, no. 1 (1994): 65–73.

Cycling '74. (2017). Max (Windows version 7.3.3) [software]. Available from http://cycling74.com.

deLahunta, Scott. "Virtual Reality and Performance," *JAP: A Journal of Performance and Art* 24, no. 1 (2002): 150–114.

"Dictionaries – Structured Data," *Max 7 Documentation*, accessed December 7, 2016, http://docs.cycling74.com/max7/vignettes/dictionaries.

Dove, Toni. "Theater without Actors: Immersion and Response in Installation," *Leonardo* 27, no. 4 (1994): 281–287.

Dryer, Ivan. "Virtual Realities and the Future of the Arts," *Leonardo* 28, no. 4 (1995): 346–347.

Dyson, Freeman. "Is Life Analog or Digital?," *Edge*, March 13, 2001, http://www.edge.org/conversation/freeman_dyson-is-life-analog-or-digital.

Fischlin, Daniel, Andrew Taylor, Toni Dove, and Michael Mackenzie. "Cybertheater, Postmodernism, and Virtual Reality: An Interview with Toni Dove and Michael Mackenzie," *Science Fiction Studies* 21, no. 1 (1994): 1–23.

"Gatherer," *Magic: The Gathering*, accessed December 1, 2016. http://gatherer.wizards.com/Pages/Default.aspx.

"Gen~ for Begininers," *Cycling '74 Wiki*, accessed December 7, 2016, http://cycling74.com/wiki/index.php?title=gen~_For_Beginners.

Gigliotti, Carol. "Aesthetics of a Virtual World," *Leonardo* 28, no. 4 (1995): 289–295.

Gilroy, Julia. “Kinect Your Body,” *Bodylabs*, December 11, 2014, http://bodylabs.com/blog/2014/12/11/kinect-your-body.

Goslin, Mike and Jacquelyn Ford Morie. “'Virtopia': Emotional Experiences in Virtual Environments,” *Leonardo* 29, no. 2 (1996): 95–100.

Goto, Suguru. “The Aesthetics and Technological Aspects of Virtual Musical Instruments: The Case of the SuperPolm MIDI Violin,” *Leonardo Music Journal* 9 (1999): 115–120.

“Introduction to OSC,” *opensoundcontrol.org an Enabling Encoding for Media Applications*, accessed November 15, 2016, http://opensoundcontrol.org/introduction-osc.

Jones, Stephen. “Towards a Philosophy of Virtual Reality: Issues Implicit in 'Consciousness Reframed,'” *Leonardo* 33, no. 2 (2000): 125–132.

Kaye, Nick and Gabriella Giannachi. “Acts of Presence: Performance, Mediation, Virtual Reality,” *TDR* 55, no. 4 (2011): 88–95.

Kendall, Gary S. “Visualization by Ear: Auditory Imagery for Scientific Visualization and Virtual Reality,” *Computer Music Journal* 15, no. 4 (1991): 70–73.

Kojs, Juraj, Stefania Serafin, and Chris Chafe. “Cyberinstruments via Physical Modeling Synthesis: Compositional Applications,” *Leonardo Music Journal* 17 (2007): 61–66.

Lang Ben. “Sixense STEM VR Motion Controller Kickstarter is Live – Prices Start at $149,” *Road to VR*, September 12, 2013, http://www.roadtovr.com/sixense-stem-vr-motion-controller-kickstarter-prices-149-tiers.

Lansley, Alastair,et al., “Caliko: An Inverse Kinematics Software Library Implementation of the FABRIK Algorithm,” *Journal of Open Research Software* 4(1), e36, accessed December 25, 2017, DOI: http://doi.org/10.5334/jors.116.

Latowska, F. Gregory and Dan Hunter. “The Laws of the Virtual Worlds,” *California Law Review* 92, no. 1 (2004): 1–73.

“Magic: The Gathering,” *Wikipedia*, accessed November 10, 2016, http://en.wikipedia.org/wiki/Magic:_The_Gathering.

Mixamo, Inc. (2015). Fuse. (Windows version 1.3) [software]. Available from http://mixamo.com/fuse.

"Mana cost," *MTG Salvation Wiki*, accessed December 7, 2016, http://mtgsalvation.gamepedia.com/Mana_cost.

Martin, Jorge Garcia. (2015). UnityOSC (Version 1.2) [C# code]. Available from http://github.com/jorgegarcia/UnityOSC.

Mateas, Michael. "Expressive AI: A Hybrid Art and Science Practice," *Leonardo* 34, no. 2 (2002): 149–153.

McKenzie, Jon. "Virtual Reality: Performance, Immersion, and the Thaw," *TDR* (1994): 83–106.

"Multi-display," *Unity Documentation*, accessed November 16, 2016, http://docs.unity3d.com/Manual/MultiDisplay.html.

Murray, Craig D. and Judith Sixsmith. "The Corporeal Body in Virtual Reality," *Ethos* 27, no. 3 (1999): 315–343.

Nechvatal, Joseph. "Towards an Immersive Intelligence," *Leonardo* 34, no. 5 (2001): 417–422.

"Oculus Rift," *Cycling '74 Forums*, accessed September 5, 2015, http://cycling74.com/forums/topic/oculus-rift/#.WEEiveYrKUk.

Orr, David W. "Virtual Nature," *Conservation Biology* 10, no. 1 (1996): 8–9.

O'Sullivan, Dan. "Choosing Tools for Virtual Environments," *Leonardo* 27, no. 4 (1994): 297–302.

"Parts of a Card," *MTG Salvation Wiki, Gamepedia*, accessed November 11, 2016, http://mtgsalvation.gamepedia.com/Parts_of_a_card.

Passer VR. (2015). InstantVR Advanced (Windows version 3.1.0) [Unity package] Available from http://passervr.com/products/instantvr-advanced.

Perelman, Bob. *Virtual Reality.* New York: Roof Books, 1993.

Pommiers, Julien. (2015). OscPkt (Windows version 1.2) [C++ OSC library]. Available from http://gruntthepeon.free.fr/oscpkt.

Rath, Matthias. "A Strategy for Modular Implementation of Physics-Based Models," *Proceedings of the 7th International Conference on Digital Audio Effects*, October 2004, 1–4.

"Razer Hydra Control Profile – Skyrim," *Sixense*, accessed December 5, 2016, http://sixense.com/skyrim.

"Re: Re: RE: Depth of field style blur shader + global illum/soft shadow shader?," *Cycling '74 Forum*, accessed July 2016, http://cycling74.com/forums/topic/re-re-re-depth-of-field-style-blur-shader-global-illumsoft-shadow-shader/#.WEgCXOYrKUk.

Riccitiello, John. "Unity 5 Launch," *Unity Blog*, March 3, 2015, http://blogs.unity3d.com/2015/03/03/unity-5-launch.

Rizzo, Albert A. and Maria T. Schultheis. "Expanding the Boundaries of Psychology: The Application of Virtual Reality," *Psychological Inquiry* 13, no. 2 (2002): 134–140.

Rocchesso, Davide, Federico Avanzini, Matthias Rath, Roberto Bresin, and Stefania Serafin. "Contact Sounds for Continuous Feedback," *Proceedings of the International Workshop on Interactive Sonification*, January 2004.

Shanklin, Will. "2016 VR Comparison Guide," *New Atlas*, October 18, 2016, http://newatlas.com/best-vr-headsets-comparison-2016/45984.

Siegel, Wayne and Jens Jacobsen. "The Challenges of Interactive Dance: An Overview and Case Study," *Computer Music Journal* 22, no. 4 (1996): 29–43.

Sixense Entertainment, Inc. (2012). Sixense SDK (Windows version 102215) [software library]. Available from http://sixense.com/windowssdkdownload.

Spalter, Anne Mortan, Phillip Andrew Stone, Barbara J. Meier, Timothy S. Miller, and Rosemary Michelle Simpson. "Interaction in an IVR Museum of Color: Constructivism Meets Virtual Reality," *Leonardo* 35, no. 1 (2002): 87–90.

"Spell," *MTG Salvation Wiki,* accessed December 7, 2016, http://mtgsalvation.gamepedia.com/Spell.

Stevens, Jane Ellen. "The Growing Reality of Virtual Reality," *BioScience* 45, no. 7 (1995): 435–439.

"The Most Versatile Musical Instrument, *Sixense Forum*, accessed February 10, 2016, http://sixense.com/forum/vbulletin/archive/index.php/t-2870.html.

Toenjes, John. "Composing for Interactive Dance: Paradigms and Perception," *Perspectives of New Music* 45, no. 2 (2007): 28–50.

Unity Technologies. (2015). Unity (Windows version 5.1) [software]. Available from http://unity3d.com.

"VR Overview," *Unity*, accessed November 15, 2016, http://unity3d.com/learn/tutorials/topics/virtual-reality/vr-overview.

Wakefield, Graham. (2017). Max_Worldmaking_Package (Windows commit 93e573a) [Max External]. Available from http://github.com/worldmaking/Max_Worldmaking_Package.

PART II

CREATIVE DOCUMENTATION

1. Spell Notation

1.1 Spell Library

The Spell Library (spells.json) is a reconfigurable mapping of the color spelling of each function. This is where functions are defined and their color combinations are set. Each spell corresponds to a function abstraction that requires a trigger sent to [function]-[activator].

spells.json requires a syntax of "[comma-separated number spelling]":"[function]",

```
{       "0":"recenter",      //0=white
        "0,1":"speed-mode",
        "0,2":"light-hand",
        "0,3":"target-mode",
        "0,0":"camera-mode",

        "1":"push",          //1=red
        "1,1":"gun",
        "1,2":"fan",
        "1,3":"sword",
        "1,4":"whip",

        "2":"grab",          //2=yellow
        "2,1":"force-field",
        "2,2":"resize",
        "2,3":"delete",
        "2,4":"pull-field",
        "2,0":"clone",

        "3":"block-make",  //3=green
        "3,1":"slab-make",
        "3,3":"ball-make",
        "3,0":"dog",

        "4":"hold",          //4=blue
        "4,1":"glove",
        "4,4":"pull",
        "4,4,4":"mass-pull" }
```

Function descriptions in the remainder of this section are formatted:
[function name]
[color spelling] – [number spelling] – [description]

1.2 Recenter

○ – 0 – Zeroes the position and rotation of the hand controllers and the headset. Look straight ahead and hold the controllers to your shoulders– left to left and right to right.

1.3 Speed-Mode

○● – 01 – Toggles a mode that quadruples performer position and doubles rotation speed. Also lightens skin color as a visual cue that the mode is active.

1.4 Light-Hand

○● – 02 – Toggles the viewer's display to fade up or down. A green or red dot circle on the performer's HUD displays the current state of the fade.

1.5 Target-Mode

○● – 03 – Toggles the hand's targeting state between short range and long range.

1.6 Camera-Mode

○○ – 00 – Cycles between camera presets after a one second delay. Uses a separate, smaller set of presets than the software interface GUI.

1.7 Push

● – 1 – Equips a small force-field around the hand that repels physics objects.

1.8 Gun

●● – 11 – Equips an instrument that shoots fast projectiles.

1.9 Fan

●● – 12 – Equips an instrument that emits a force-field proportionate to the hand's velocity. Activates Slabs with noise impulses.

1.10 Sword

●● – 13 – Equips a long, rigid instrument used for striking slabs and other physics objects. Activates Slabs with s triangle wave signal.

1.11 Whip

●● – 14 – Equips a long, constraint chain instrument, used for striking slabs and other physics objects. Activates Slabs with sawtooth wave signal.

1.12 Grab

🟡 – 2 – Begins to offset the current target’s position and rotation. If no object is targeted, effects the first available target.

1.13 Force-Field

🟡🔴 – 21 – Creates an instance of a floating force-field that repels physics objects.

1.14 Resize

🟡🟡 – 22 – Begins to offset the current target’s scale. If no object is targeted, effects the first available target.

1.15 Delete

🟡🟢 – 23 – Removes the instance of the current target. If no object is targeted, effects the first available target.

1.16 Pull-Field

🟡🔵 – 24 – Creates an instance of a floating force-field that attracts physics objects.

1.17 Clone

🟡○ – 20 – Duplicates the instance of the current target. If no object is targeted, effects the first available target.

1.18 Block-Make

🟢 – 3 – Creates an instance of a small physics cube. Activates Slabs with a DC impulse.

1.19 Slab-Make

🟢🔴 – 31 – Creates an instance of a physical model audio slab. Receives impulses from certain physics objects and equipped instruments. Overlapping Slabs feed into each other.

1.20 Ball-Make

🟢🟢 – 33 – Creates an instance of a small bouncy sphere. Activates Slabs with a DC impulse.

1.21 Dog

●○ – 30 – Creates an instance of an autonomous robot that randomly roams around the environment and repels physics objects on contact. If Dog is already active, the spell summons it to the current position.

1.22 Hold

● – 4 – Equips a small force-field around the hand that attracts physics objects.

1.23 Glove

●● – 41 – Equips a small field that modulates Slab audio.

1.24 Pull

●● – 44 – Equips a long, thin force-field around the hand that attracts physics objects.

1.25 Mass-Pull

●●● – 444 – Equips a large force-field around the hand that attracts physics objects.

2. Code Documentation

2.1 Hydra Controller Win32 Command Line Solution

Hydra.cpp

```
#include "../glfw.h"
#include "../sixense.h"
#define _WINSOCK_DEPRECATED_NO_WARNINGS 1;
#define _CRT_SECURE_NO_WARNINGS 1;
#define OSCPKT_OSTREAM_OUTPUT
#pragma comment(lib, "../sixense.lib")
#pragma comment(lib, "../sixense_utils.lib")
#pragma comment(lib, "../libglfw.a")
#pragma comment(lib, "../libopengl32.a")
#include "../oscpkt.hh"
#include "../udp.hh"

using namespace oscpkt;
#include <iostream>
#include "stdafx.h"
using namespace std;

int main(){
        sixenseInit();
        sixenseSetActiveBase(0);    //operates on one base only
        sixenseSetFilterEnabled(1);
        sixenseControllerData dat;
        UdpSocket sok;
        sok.connectTo("localhost", 7022);    //port is hard coded
        std::cout << "GLFW time set to .033" << std::endl;
        for (glfwInit();; glfwSetTime(glfwGetTime() - .033)){      //set interval time
                for (; glfwGetTime()<.033;);
                PacketWriter pkt;
                pkt.startBundle(); //create OSC packet
                Message lpa("l p"),          //left – position
                        ljx("l jx"),         //joystick X
                        ljy("l jy"),         //joystick Y
                        ltt("l t"),          //trigger
                        lb0("l b 0"),        //buttons
                        lb1("l b 1"),
                        lb2("l b 2"),
                        lb3("l b 3"),
                        lb4("l b 4"),
                        lb5("l b 5"),
                        lb6("l b 6"),
                        lra("l r"),          //rotation
                        rpa("r p"),          //right
                        rjx("r jx"),
                        rjy("r jy"),
                        rtt("r t"),
                        rb0("r b 0"),
                        rb1("r b 1"),
				rb2("r b 2"),
				rb3("r b 3"),
				rb4("r b 4"),
				rb5("r b 5"),
				rb6("r b 6"),
				rra("r r");
			sixenseGetNewestData(0, &dat);	//append data to OSC packet
			pkt.addMessage(lpa.pushFloat(dat.pos[2]))
				.addMessage(lpa.pushFloat(dat.pos[1]))
				.addMessage(lpa.pushFloat(dat.pos[0]))
				.addMessage(ljx.pushFloat(dat.joystick_x))
				.addMessage(ljy.pushFloat(dat.joystick_y))
				.addMessage(ltt.pushFloat(dat.trigger))
				.addMessage(lb0.pushInt32(dat.buttons&SIXENSE_BUTTON_START ? 1 : 0))
				.addMessage(lb1.pushInt32(dat.buttons&SIXENSE_BUTTON_1 ? 1 : 0))
				.addMessage(lb2.pushInt32(dat.buttons&SIXENSE_BUTTON_2 ? 1 : 0))
				.addMessage(lb3.pushInt32(dat.buttons&SIXENSE_BUTTON_3 ? 1 : 0))
				.addMessage(lb4.pushInt32(dat.buttons&SIXENSE_BUTTON_4 ? 1 : 0))
				.addMessage(lb5.pushInt32(dat.buttons&SIXENSE_BUTTON_BUMPER ? 1 : 0))
				.addMessage(lb6.pushInt32(dat.buttons&SIXENSE_BUTTON_JOYSTICK ? 1 : 0))
				.addMessage(lra.pushFloat(dat.rot_quat[3]))
				.addMessage(lra.pushFloat(dat.rot_quat[2] * -1))
				.addMessage(lra.pushFloat(dat.rot_quat[1] * -1))
				.addMessage(lra.pushFloat(dat.rot_quat[0]));
			sixenseGetNewestData(1, &dat);
			pkt.addMessage(rpa.pushFloat(dat.pos[2]))
				.addMessage(rpa.pushFloat(dat.pos[1]))
				.addMessage(rpa.pushFloat(dat.pos[0]))
				.addMessage(rjx.pushFloat(dat.joystick_x))
				.addMessage(rjy.pushFloat(dat.joystick_y))
				.addMessage(rtt.pushFloat(dat.trigger))
				.addMessage(rb0.pushInt32(dat.buttons&SIXENSE_BUTTON_START ? 1 : 0))
				.addMessage(rb1.pushInt32(dat.buttons&SIXENSE_BUTTON_1 ? 1 : 0))
				.addMessage(rb2.pushInt32(dat.buttons&SIXENSE_BUTTON_2 ? 1 : 0))
				.addMessage(rb3.pushInt32(dat.buttons&SIXENSE_BUTTON_3 ? 1 : 0))
				.addMessage(rb4.pushInt32(dat.buttons&SIXENSE_BUTTON_4 ? 1 : 0))
				.addMessage(rb5.pushInt32(dat.buttons&SIXENSE_BUTTON_BUMPER ? 1 : 0))
				.addMessage(rb6.pushInt32(dat.buttons&SIXENSE_BUTTON_JOYSTICK ? 1 : 0))
				.addMessage(rra.pushFloat(dat.rot_quat[3]))
				.addMessage(rra.pushFloat(dat.rot_quat[2] * -1))
				.addMessage(rra.pushFloat(dat.rot_quat[1] * -1))
				.addMessage(rra.pushFloat(dat.rot_quat[0]));
			sok.sendPacket(pkt.packetData(), pkt.packetSize());
		}
		while (true)
		{
			Sleep(10000);
		}
	}
```

2.2 [mxj] Spell Displaying Overlay Drawer

SpellDisplayer.java

```
import com.cycling74.max.Atom;
import com.cycling74.max.MaxObject;

public class SpellDisplayer extends MaxObject {

  int x = 1;
        int maxspell = 6; //maximum number of characters
        String[] colors =    {"1 1 1 1", //build color array
                              "1 .25 .1 1",
                              "1 1 .25 1",
                              ".25 1 .25 1",
                              ".25 .25 1 1",
                              "0 0 0 0"};

  int[][] n = {{0,3,6,9,12,15},  //circle row positions
            {18,21,24,27,30,33}
  };

    SpellDisplayer() { //Max object init
    declareIO(1,1);
  }

  public void loadbang() {
    bang();
  }

  public void disp(Atom[] a){           //set row colors
    reset(a[0].getInt());
    for (int i = 1; i < a.length; i++) {
        replace(a[0].getInt(),i,a[i].getInt());
    }
}

  public void reset(int x){    //resets the given row
    for (int i = 0; i < maxspell; i++){
        replace(x,i+1,5);
    }
}

public void replace(int r, int p, int c){//change colors
        outlet(0,"cmd_replace " + n[r][p-1] + " glcolor " + colors[c]);
}

public void circ(double r, double g, double b, double a, double x, double y){ //draw circles
        outlet(0,"glcolor " + r + " " + g + " " + b + " " + a);
        outlet(0,"moveto " + x + " " + y + " 0");
        outlet(0,"circle 0.25");
}

public void bang() {  //set up drawing
```

```
    outlet(0, "reset");
    circ(0,0,0,0,-1.3,.535);
    circ(0,0,0,0,-.8,.535);
    circ(0,0,0,0,-.3,.535);
    circ(0,0,0,0,.2,.535);
    circ(0,0,0,0,.7,.535);
    circ(0,0,0,0,1.2,.535);
    circ(0,0,0,0,-1.3,-.268);
    circ(0,0,0,0,-.8,-.268);
    circ(0,0,0,0,-.3,-.268);
    circ(0,0,0,0,.2,-.268);
    circ(0,0,0,0,.7,-.268);
    circ(0,0,0,0,1.2,-.268);
    }
}
```

2.3 [mxj] JSON Parser and Spell Reader with Controller Spelling

ButtonSpell.java

```
import com.cycling74.max.MaxObject;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;

import org.json.JSONException;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

public class ButtonSpell extends MaxObject {
  String spells = "M:/virtual_stage/spells/spells.json";
    JSONParser parser = new JSONParser();
    JSONObject jsonObject = null;                 //loaded json from file
    Object xObject = null;                        //object holder
    String spelst = " ";                          //string holder
    String spelc = " ";                           //string with commas removed
    int[] c = new int[]{0,0,0,0,0,0,0,0,0};       //detect change
    int maxspell = 6;                             //maximum characters in spelling
    List<Integer> speller = new ArrayList<Integer>(1);  //appends buttons for spelling
    List<Integer> spelt = new ArrayList<Integer>(1);    //list holder
    boolean res = true;                 //remember reset state
    boolean check = false;              //stores if spell is valid

    ButtonSpell() {             //Max object init
      declareIO(1,4);
      //outlets:
      //1-symbols to display parser
      //2-text for top row
      //3-text for bottom row
      //4-clear messages
    }

        public void read() throws JSONException {      //JSON file reader
          System.out.println(spells);
          try{
            Object obj = parser.parse(new FileReader(spells));
            jsonObject = (JSONObject) obj;
          } catch (Exception e) {
            e.printStackTrace();
          }
          reset();              //re-init after loading
        }

        public void reset() { //re-initialize
          outlet(0,1);         //bottom row
          outlet(0,0);         //top row
          outlet(4,0);         //clear hand state
          ResetSpeller();    //reset speller to bottom row
          outlet(2, "-");      //bottom row
          outlet(1, "-");      //top row
        }

        public void list(int x, int y) { //buttons index, state
                if (y == 1 && c[x] == 0) {              //check for state change
                   if (x < 5) {                         //spell buttons
                      res = true;                       //clear reset state
                      if (speller.size() < maxspell+1) { //check for max spell length
                         speller.add(x);                //append input to spell
                         SpellConvert();                //set bottom row
                         Spelled(2);                    //check/out for spell name bottom row
                      }
                   } else if (x == 6) {         //joystick button(clear)
                      if (res == false) {       //check for reset state 0
                         outlet(0,0);           //clear top row
                         outlet(3,0);           //clear top text
                      } else {                  //reset state 1
                         res = false;           //ready to clear top row
                         outlet(0,1);           //clear bottom row
                         outlet(3,1);           //clear bottom text
                         ResetSpeller();        //reset speller to bottom row
                      }
                   }
                }
    c[x] = y;  //update state change
}

public void bang(){                     //trigger on(>0) priority bang
    if (check == true) {                //is spell valid?
       if (speller.size() > 1) {        //is anything spelled?
          speller.set(0, 0);            //set top row output
          SpellConvert();               //set spell colors
          Spelled(1);                   //check/out for spell name on top row
          outlet(0,1);                  //clear bottom row
          outlet(2,"-");                //clear bottom text
          ResetSpeller();               //reset speller to bottom row
       }
    }
}

public void SpellConvert() {
    spelst = speller.toString();
    spelst = spelst.replace("[", "");  //remove the right bracket
    spelst = spelst.replace("]", "");  //remove the left bracket
    spelst = spelst.replace(",", "");  //remove commas
    outlet(0,spelst);
}

public void Spelled(int x) {                    //x = row number
    spelt = new ArrayList<Integer>(speller);    //copy list for processing
    spelt.remove(0);                            //remove first entry
    spelc = spelt.toString();                   //convert to string
    spelc = spelc.replace(" ", "");             //remove spaces
    spelc = spelc.replace("[", "");             //remove the right bracket
    spelc = spelc.replace("]", "");             //remove the left bracket
    if (jsonObject.get(spelc) != null) {        //check isset
```

```
            xObject = jsonObject.get(spelc);            //store as Object
            outlet(3,x-1);
            outlet(x,xObject.toString());               //output spell text to x row
            check = true;                               //spell is valid
        } else {
            outlet(x,"XXX");                            //unset text
            check = false;                              //spell is invalid
        }
    }

    public void ResetSpeller() {            //reset speller list to [1]
        speller = new ArrayList<Integer>(0);
        speller.add(1);
    }
}
```

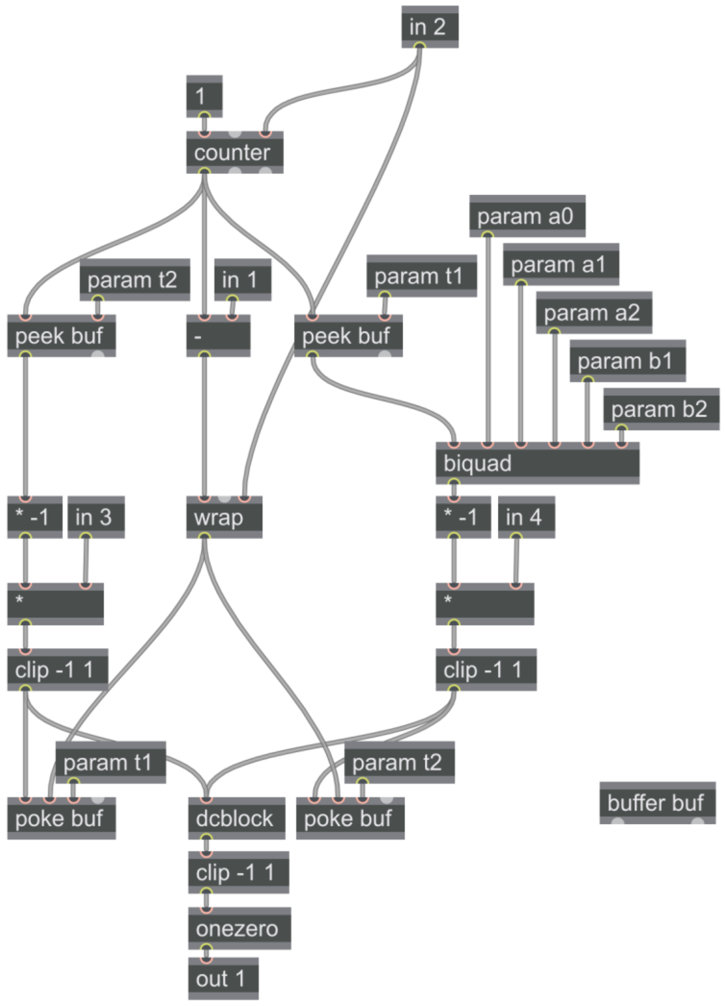
2.4 [gen~] 3-Axis Waveguide String with Aligned Scattering Junction

xd_string.gendsp

```
//samp,left-buffer,right-buffer,left-delay,right-delay,left-reflect,right-reflect,scatter,buffer-sym,buffer-max,input,pan
scat(t, lb, rb, ld, rd, lr, rr, k, std, bm, i, p)
{
   ls = -1 * peek(std, wrap(t - bm * ld, 0, bm), lb) * lr; //left summing
   rs = -1 * peek(std, wrap(t - bm * rd, 0, bm), rb) * rr; //right summing
   li = (ls * k) + ((1 - k) * rs) + i * p; //left input
   ri = (ls * (1 + k)) + (-1 * k * rs) + i * (1 - p); //right input
   poke(std, li, t, lb);  //write left
   poke(std, ri, t, rb); //write right
   return 0;
}

Buffer std("#0-string_delays"); //delay buffer
History t(0); //index
Param bm(4410); //buffer max
Param xrl(0); //X Reflect coeff
Param xrr(0);
Param yrl(0); //Y "
Param yrr(0);
Param zrl(0); //Z "
Param zrr(0);
Param xk(0); //X Scatter coeff
Param yk(0); //Y
Param zk(0); //Z
Param xp(0); //X Pan
Param yp(0); //Y
Param zp(0); //Z

t = wrap(t + 1, 0, bm); //accum sample
scat(t, 0, 1, in1, in2, xrl, xrr, xk, std, bm, in7, xp);  //X-axis string
scat(t, 2, 3, in3, in4, yrl, yrr, yk, std, bm, in7, yp);  //Y-axis string
scat(t, 4, 5, in5, in6, zrl, zrr, zk, std, bm, in7, zp);  //Z-axis string
out1 = peek(std, t, 0);
out2 = peek(std, t, 1);
out3 = peek(std, t, 2);
out4 = peek(std, t, 3);
out5 = peek(std, t, 4);
out6 = peek(std, t, 5);
```

2.5 [gen~] Buffer-Based Waveguide String Model

z_string.gendsp



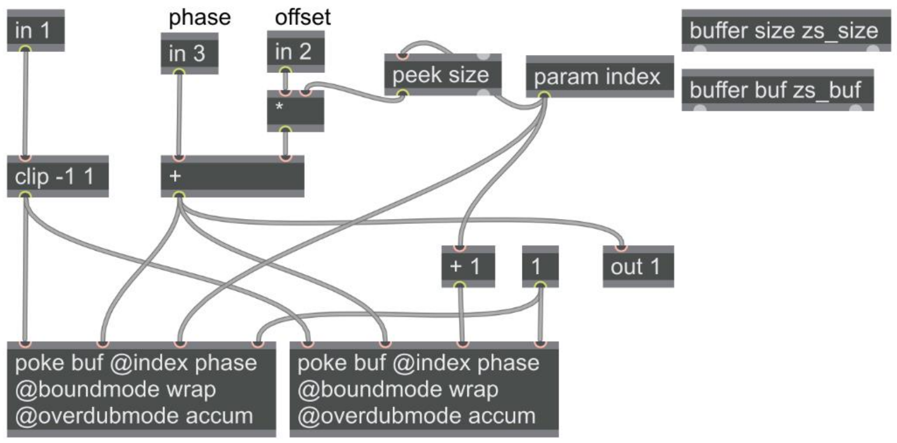
2.6 [gen~] Interpolating Buffer Write

ipoke.gendsp

```
// adapted from Graham Wakefield’s ipoke code
// main difference is the limiting of interpolation steps to prevent runaway loops
History idx0;
Delay vd(4);  // previous input values
val1 = in1;
idx1 = in2*dim(buf);
chan = in3;
overdub = in4;
maxstp = in5; //maximum step
// cache index for next time:
idx0_next = idx1;
// store input value into the delay buffer:
vd.write(val1);
// handle wrap around:
if (idx1 < idx0) { idx1 += dim(buf); }
// get interpolation scalar:
iscale = fixnan(1/(idx1 - idx0));
// write samples:
idxi0, idxi1 = floor(idx0), floor(idx1);
dif = abs(idxi1-idxi0); //determine step amount
stp = round(dif/clip(dif,1,maxstp)); //enforce maximum step amount
for (i=idxi0; i<idxi1; i+=stp) {
   // interpolated read from delay:
   a = (i - idx0)*iscale;
   v = vd.read(1-a);
   // write into buffer (with overdub)
   poke(buf, v, i, chan, overdub, boundmode="wrap");
}
// output recorded samples per input sample factor:
// update old index for next time:
idx0 = idx0_next;
out1 = stp;
```

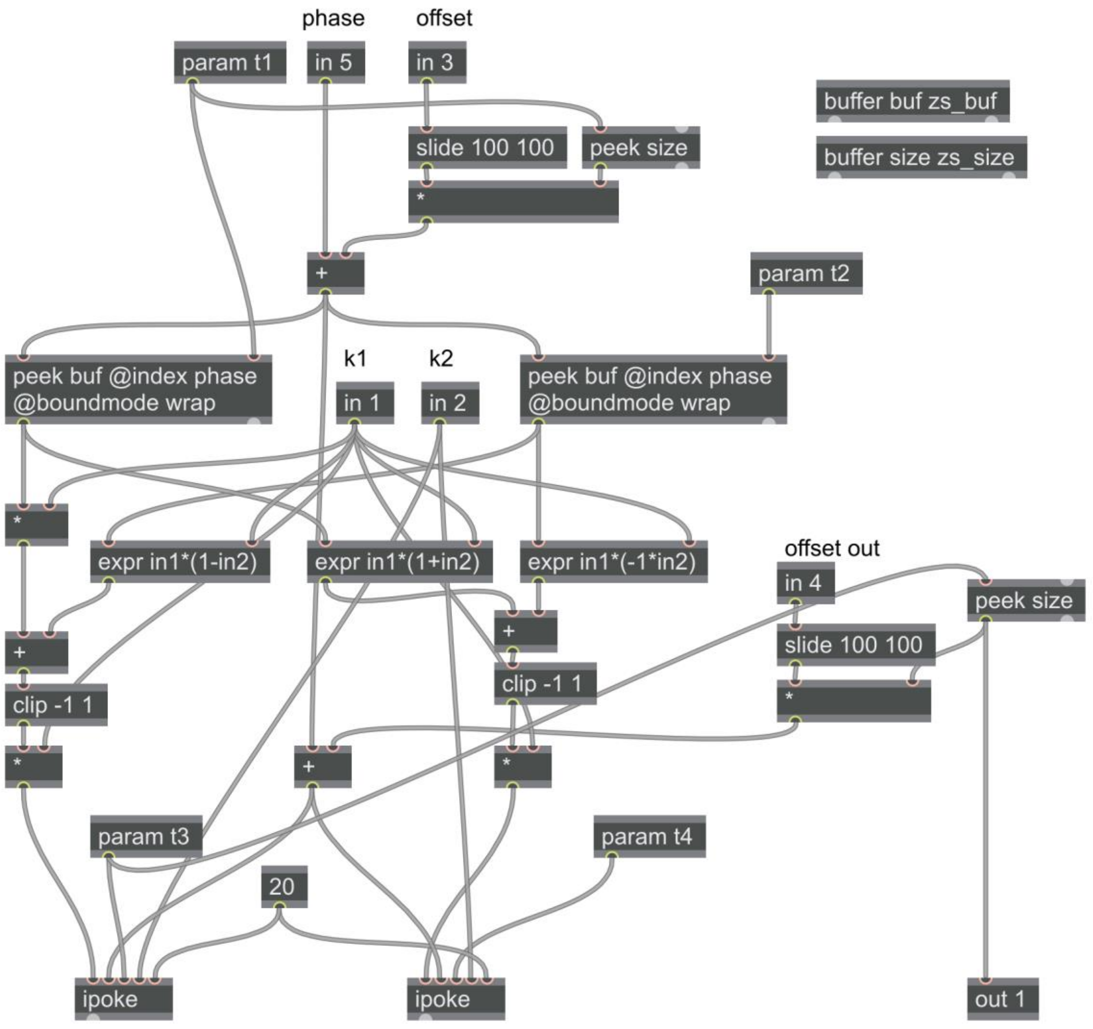
2.7 [gen~] Physical Model Buffer Poker

poker2.gendsp


in 1
phase
in 3
offset
in 2
peek size
param index
buffer size zs_size
buffer buf zs_buf
*
clip -1 1
+
+ 1
1
out 1
poke buf @index phase @boundmode wrap @overdubmode accum
poke buf @index phase @boundmode wrap @overdubmode accum

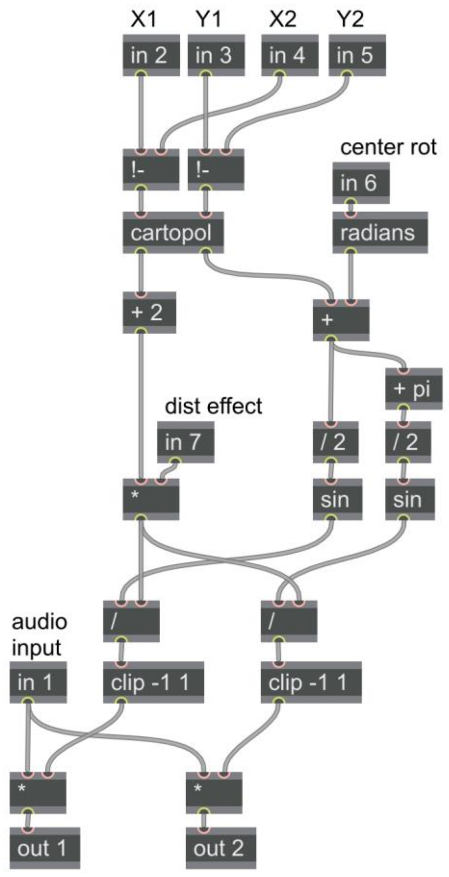
2.8 [gen~] Physical Model Cross-Buffer Scattering

buf_scat.gendsp



2.9 [gen~] Cartesian to Polar Audio Panning

cartopol_pan.gendsp



2.10 [js] Iterated Physics Instance Setup Example

ball.pe.ac_defaults.js

```
var d = [ //c-count, s-scale, m-mass, p-phys, x-multiple, g-gridshape
    "c 100",
    "s 0.05 0.05 0.05",
    "m mass 55",
    "p rolling_friction 0.5",
    "p restitution 1.5",
    "p shape sphere",
    "g shape sphere",
    "g dim 20 20",
    "g poly_mode 0 0",
    "g lighting_enable 1",
    "g color 0 .6 1 1",
    ];

function bang()
{
    for (var i = 0; i < d.length; i++)
    {
        outlet(0, d[i]);
    }
}
```

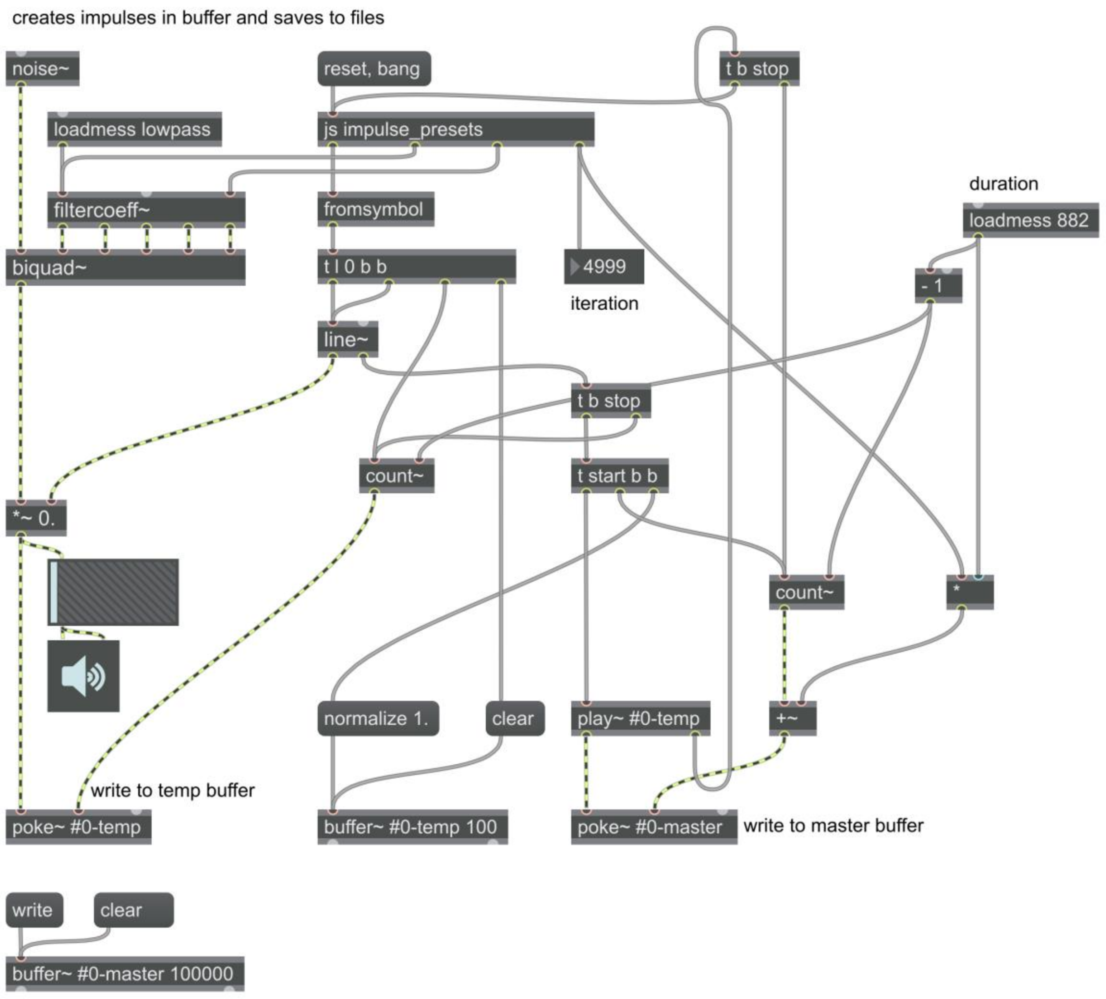
2.11 [js] Iterative Impulse Generator Setup

impulse_presets.js

```
inlets = 1;
outlets = 4;

var i = 0; //counter
var c = 1000;  //max count

var l = [   //amplitude functions
            "1. 0. 0. 20.",
            "1. 5. 0. 15.",
            "1. 10. 0. 10.",
            "1. 15. 0. 15.",
            "1. 20. 0. 0."
];

var r = [.5, 1, 2, 3, 4, 5, 7.5, 10, 15, 20]; //resonance values


function reset(b) { //reset
  i = 0;    //reset counter
  c = b;   //set max count
}

function bang() {   //iterate toward max count on bang
  if (i < c) {
    start(i);
  };
  i = i + 1;
}

function start(v) {
  z = trunc(v / 1000);                        //thousands place
  y = (trunc(v % 1000 / 100));           //hundreds place
  x = trunc(v % 1000 % 100);            //tens and ones place
  outlet(3, (z * 1000 + y * 100 + x));  //index
  outlet(2, r[y]);                              //resonance
  outlet(1, Math.pow((x / 2), 2) * 5 + 40); //frequency
  outlet(0, l[z]);                              //line function
}

function trunc(number) { //truncate right of decimal
  return number > 0
      ? Math.floor(number)
      : Math.ceil(number);
}
```

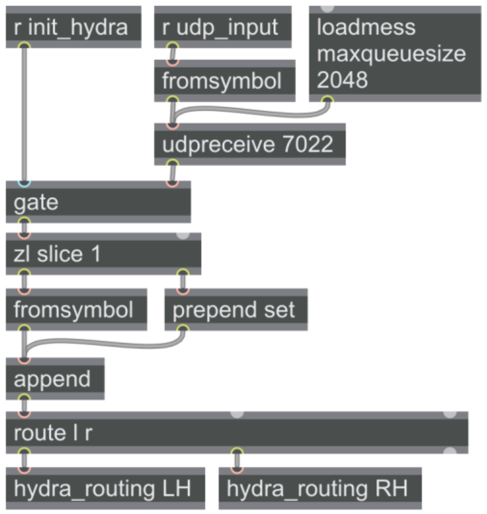
2.12 [maxpat] Iterative Impulse Generator Patch

impulse_generator.maxpat



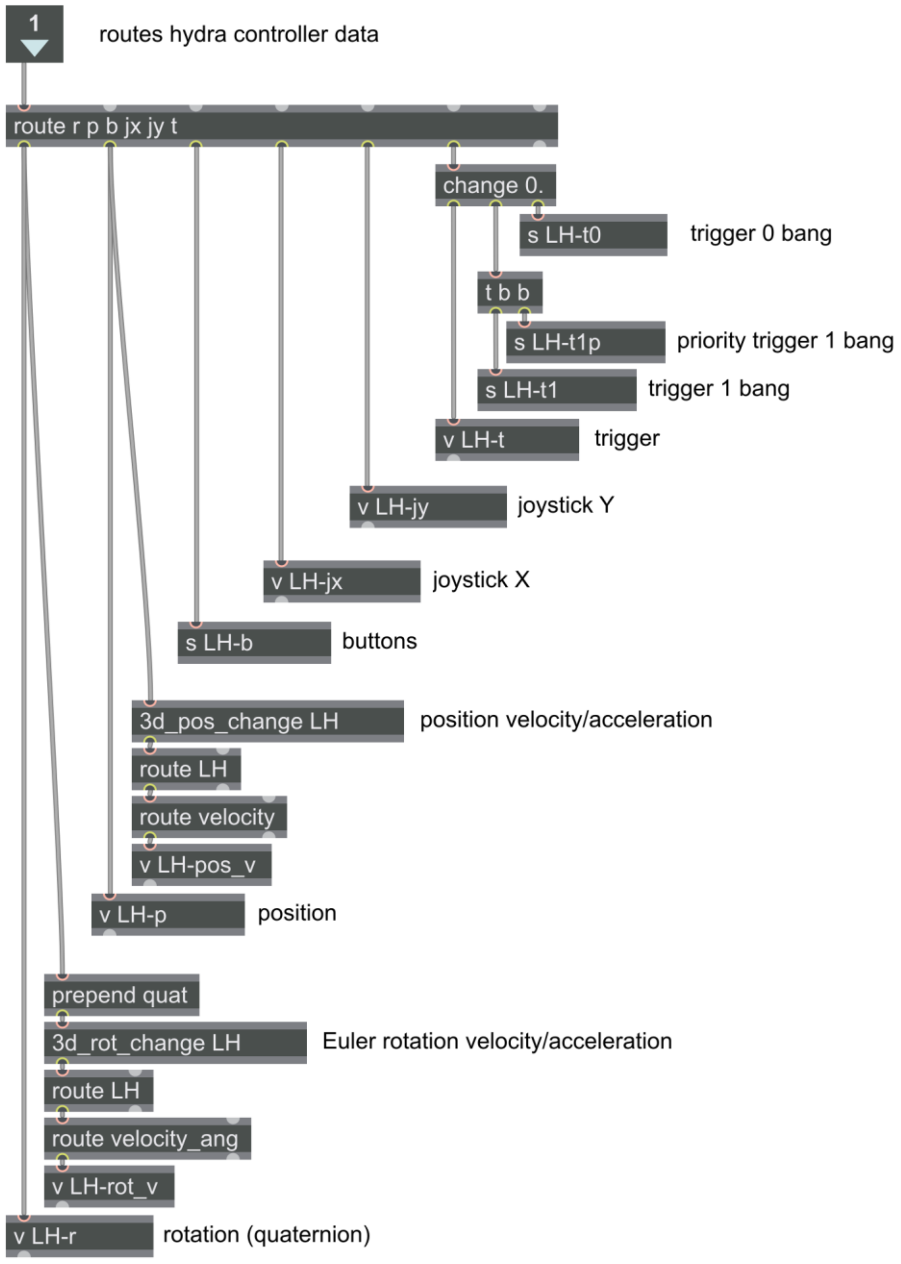
2.13 [maxpat] Hydra Input Parsing

hydra_input.maxpat



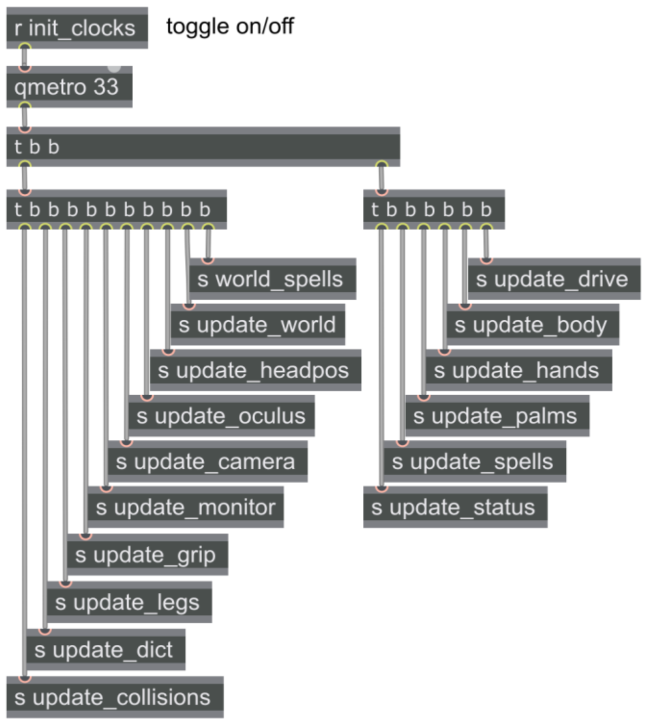
2.14 [maxpat] Hydra Parameter Routing

hydra_routing.maxpat



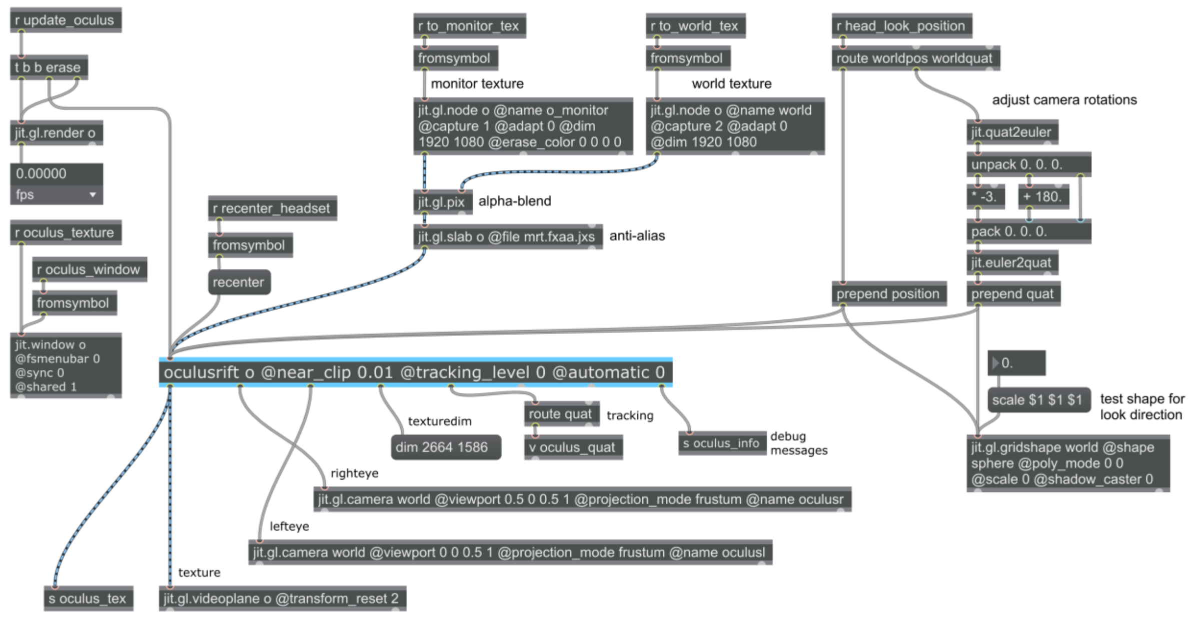
2.15 [maxpat] Update Clocks

clocks.maxpat



2.16 [maxpat] Oculus Headset Rendering

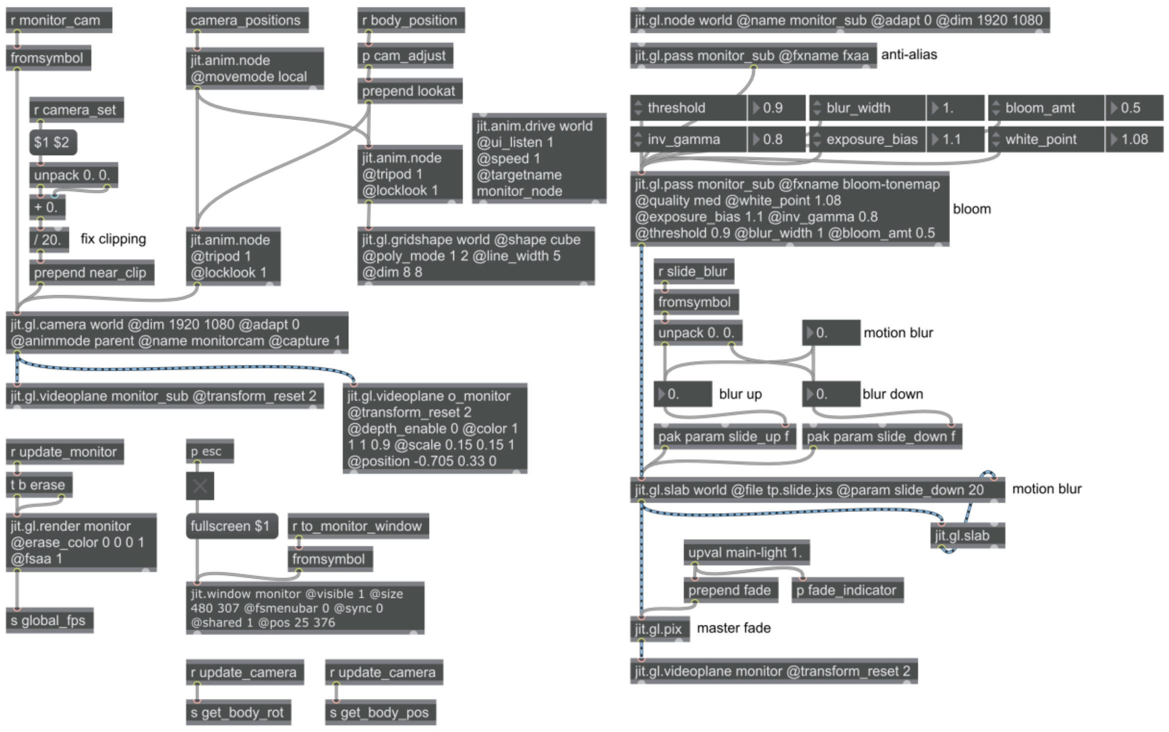
oculus.maxpat



```
r update_oculus
t b b erase
jit.gl.render o
0.00000
fps
r oculus_texture
r oculus_window
fromsymbol
jit.window o @fsmenubar 0 @sync 0 @shared 1
r recenter_headset
fromsymbol
recenter
r to_monitor_tex
fromsymbol
jit.gl.node o @name o_monitor @capture 1 @adapt 0 @dim 1920 1080 @erase_color 0 0 0 0
r to_world_tex
fromsymbol
jit.gl.node o @name world @capture 2 @adapt 0 @dim 1920 1080
jit.gl.pix
jit.gl.slab o @file mrt.fxaa.jxs
r head_look_position
route worldpos worldquat
jit.quat2euler
unpack 0. 0. 0.
* -3.
+ 180.
pack 0. 0. 0.
jit.euler2quat
prepend position
prepend quat
oculusrift o @near_clip 0.01 @tracking_level 0 @automatic 0
dim 2664 1586
route quat
v oculus_quat
s oculus_info
0.
scale $1 $1 $1
jit.gl.gridshape world @shape sphere @poly_mode 0 0 @scale 0 @shadow_caster 0
jit.gl.camera world @viewport 0.5 0 0.5 1 @projection_mode frustum @name oculusr
jit.gl.camera world @viewport 0 0 0.5 1 @projection_mode frustum @name oculusl
s oculus_tex
jit.gl.videoplane o @transform_reset 2
```

2.17 [maxpat] Third-Person Camera Rendering and Post-Processing

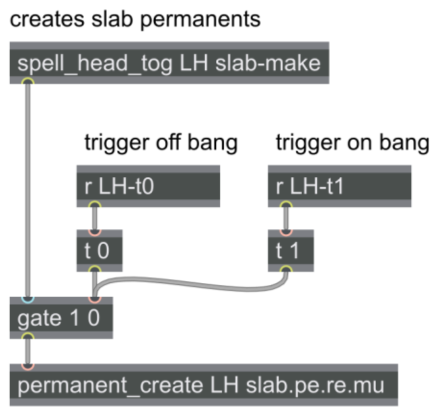
monitor.maxpat



```
r monitor_cam
fromsymbol
r camera_set
$1 $2
unpack 0. 0.
+ 0.
/ 20.    fix clipping
prepend near_clip

camera_positions
jit.anim.node @movemode local
jit.anim.node @tripod 1 @locklook 1

r body_position
p cam_adjust
prepend lookat
jit.anim.node @tripod 1 @locklook 1
jit.anim.drive world @ui_listen 1 @speed 1 @targetname monitor_node
jit.gl.gridshape world @shape cube @poly_mode 1 2 @line_width 5 @dim 8 8

jit.gl.camera world @dim 1920 1080 @adapt 0 @animmode parent @name monitorcam @capture 1
jit.gl.videoplane monitor_sub @transform_reset 2
jit.gl.videoplane o_monitor @transform_reset 2 @depth_enable 0 @color 1 1 0.9 @scale 0.15 0.15 1 @position -0.705 0.33 0

r update_monitor
t b erase
jit.gl.render monitor @erase_color 0 0 0 1 @fsaa 1
s global_fps

p esc
fullscreen $1
r to_monitor_window
fromsymbol
jit.window monitor @visible 1 @size 480 307 @fsmenubar 0 @sync 0 @shared 1 @pos 25 376

r update_camera
s get_body_rot
r update_camera
s get_body_pos

jit.gl.node world @name monitor_sub @adapt 0 @dim 1920 1080
jit.gl.pass monitor_sub @fxname fxaa    anti-alias
threshold 0.9    blur_width 1.    bloom_amt 0.5
inv_gamma 0.8    exposure_bias 1.1    white_point 1.08
jit.gl.pass monitor_sub @fxname bloom-tonemap @quality med @white_point 1.08 @exposure_bias 1.1 @inv_gamma 0.8 @threshold 0.9 @blur_width 1 @bloom_amt 0.5    bloom
r slide_blur
fromsymbol
unpack 0. 0.    0.    motion blur
0.    blur up    0.    blur down
pak param slide_up f    pak param slide_down f
jit.gl.slab world @file tp.slide.jxs @param slide_down 20    motion blur
jit.gl.slab
upval main-light 1.
prepend fade    p fade_indicator
jit.gl.pix    master fade
jit.gl.videoplane monitor @transform_reset 2
```

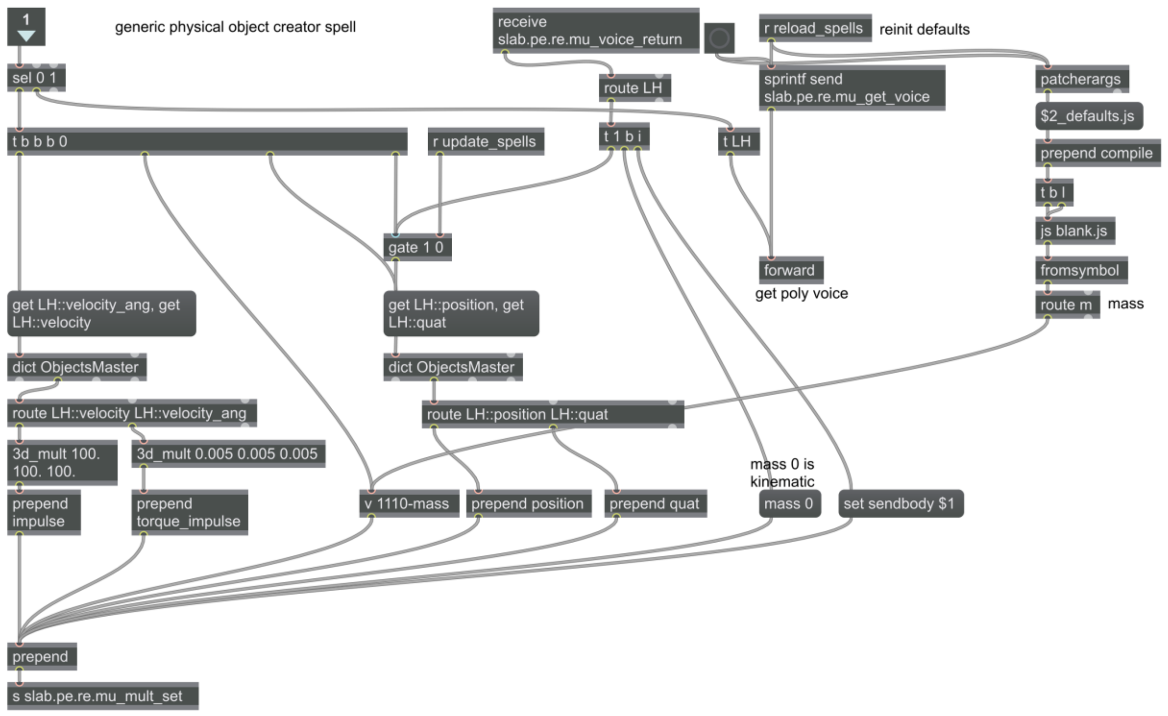
2.18 [maxpat] Slab Spell

slab-make.maxpat



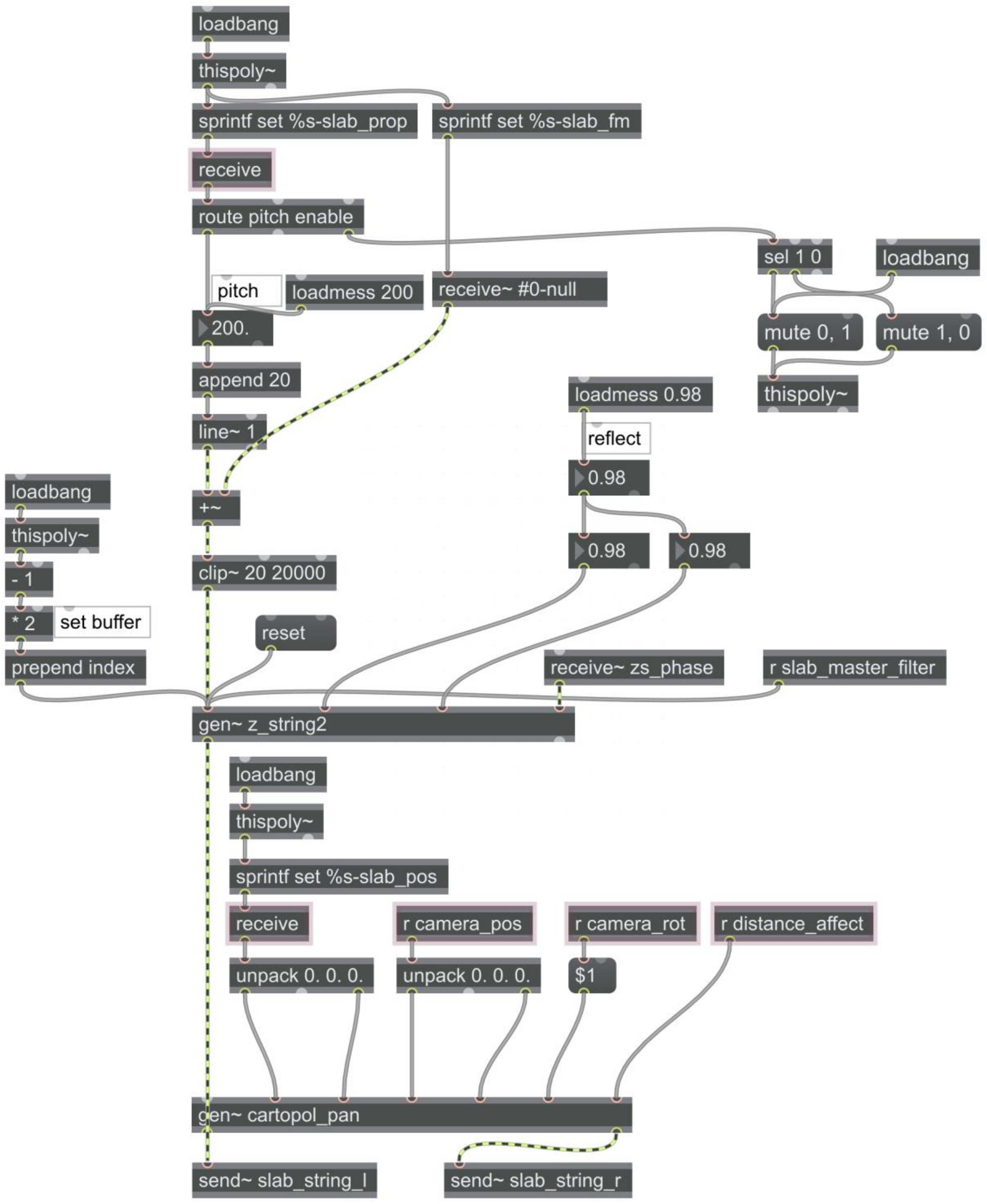
2.19 [maxpat] Permanent Instance Creator

permanent_create.maxpat



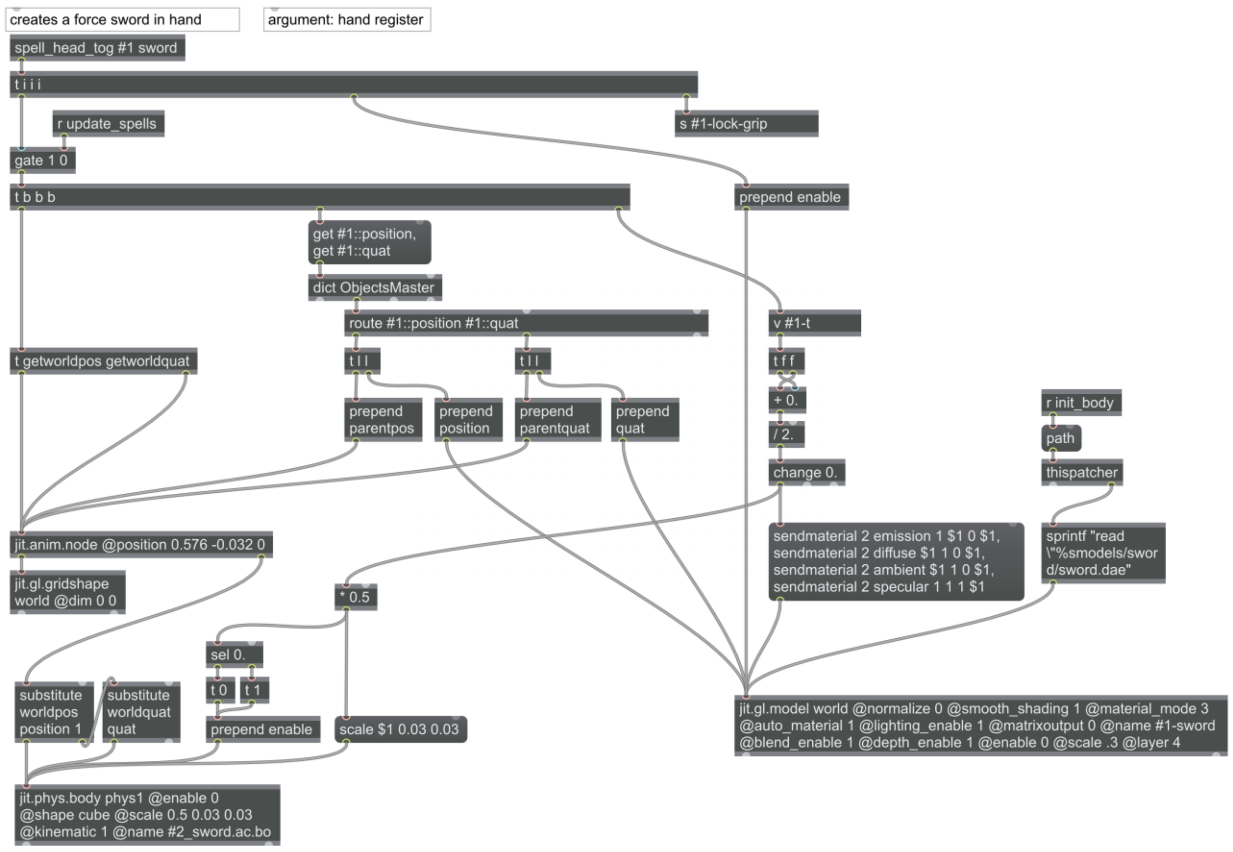
2.20 [maxpat] Slab Audio Generator

slab_audio.maxpat



2.21 [maxpat] Sword Spell

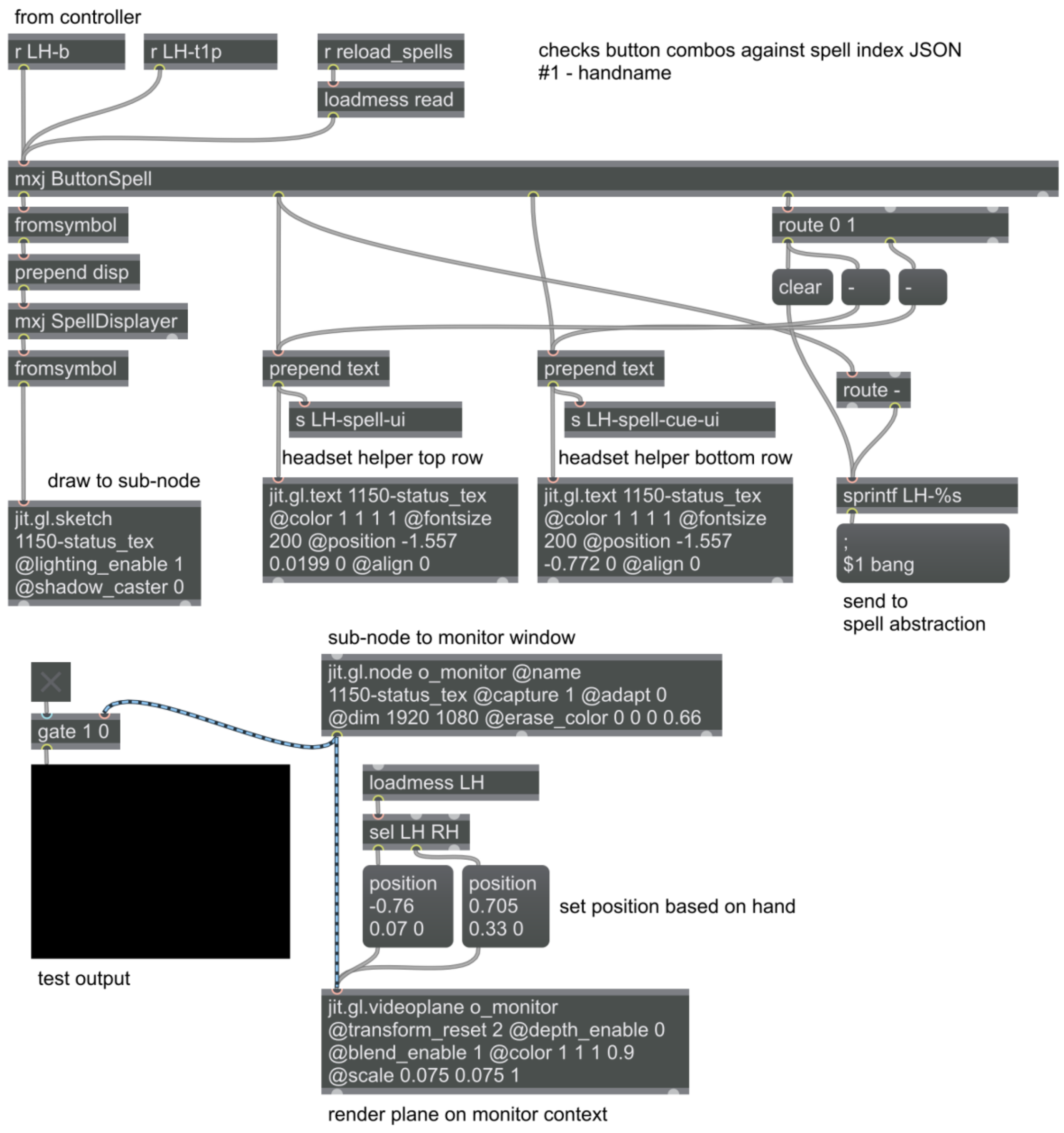
sword.maxpat



```
creates a force sword in hand
argument: hand register
spell_head_tog #1 sword
t i i i
r update_spells
s #1-lock-grip
gate 1 0
t b b b
prepend enable
get #1::position,
get #1::quat
dict ObjectsMaster
route #1::position #1::quat
v #1-t
t getworldpos getworldquat
t l l
t l l
t f f
+ 0.
/ 2.
change 0.
prepend
parentpos
prepend
position
prepend
parentquat
prepend
quat
r init_body
path
thispatcher
jit.anim.node @position 0.576 -0.032 0
sendmaterial 2 emission 1 $1 0 $1,
sendmaterial 2 diffuse $1 1 0 $1,
sendmaterial 2 ambient $1 1 0 $1,
sendmaterial 2 specular 1 1 1 $1
sprintf "read
\"%smodels/swor
d/sword.dae"
jit.gl.gridshape
world @dim 0 0
* 0.5
sel 0.
t 0
t 1
substitute
worldpos
position 1
substitute
worldquat
quat
prepend enable
scale $1 0.03 0.03
jit.gl.model world @normalize 0 @smooth_shading 1 @material_mode 3
@auto_material 1 @lighting_enable 1 @matrixoutput 0 @name #1-sword
@blend_enable 1 @depth_enable 1 @enable 0 @scale .3 @layer 4
jit.phys.body phys1 @enable 0
@shape cube @scale 0.5 0.03 0.03
@kinematic 1 @name #2_sword.ac.bo
```

2.22 [maxpat] Spell checker and router

spell_maker.maxpat



from controller
r LH-b
r LH-t1p
r reload_spells
loadmess read
checks button combos against spell index JSON
#1 - handname
mxj ButtonSpell
fromsymbol
prepend disp
mxj SpellDisplayer
fromsymbol
route 0 1
clear
-
-
prepend text
prepend text
route -
s LH-spell-ui
s LH-spell-cue-ui
headset helper top row
headset helper bottom row
draw to sub-node
jit.gl.sketch 1150-status_tex @lighting_enable 1 @shadow_caster 0
jit.gl.text 1150-status_tex @color 1 1 1 1 @fontsize 200 @position -1.557 0.0199 0 @align 0
jit.gl.text 1150-status_tex @color 1 1 1 1 @fontsize 200 @position -1.557 -0.772 0 @align 0
sprintf LH-%s
; $1 bang
send to spell abstraction
sub-node to monitor window
jit.gl.node o_monitor @name 1150-status_tex @capture 1 @adapt 0 @dim 1920 1080 @erase_color 0 0 0 0.66
gate 1 0
loadmess LH
sel LH RH
position -0.76 0.07 0
position 0.705 0.33 0
set position based on hand
test output
jit.gl.videoplane o_monitor @transform_reset 2 @depth_enable 0 @blend_enable 1 @color 1 1 1 0.9 @scale 0.075 0.075 1
render plane on monitor context

2.23 [js] Global Parameter Initialization

initParams.js

```
function bang() {
   //Spells
   messnamed("reload_spells", "bang");
   //Controllers
   messnamed("udp_input", "port 7022");
   messnamed("udp_input", "maxqueuesize 1024");
   //Body
   messnamed("hand_speedlim", 5);
   messnamed("foot_adjust", "bang");
   messnamed("leg_wiggle_toggle", 1);
   messnamed("breathing_toggle", 1);
   messnamed("recenter_headset", "bang");
   messnamed("RH-target_shape_up", .05);
   messnamed("RH-target_shape_up", "bang");
   messnamed("LH-target_shape_up", .05);
   messnamed("LH-target_shape_up", "bang");
   messnamed("RH-force_shape_up", 0);
   messnamed("RH-force_shape_up", "bang");
   messnamed("LH-force_shape_up", 0);
   messnamed("LH-force_shape_up", "bang");
   //Shader processing
   messnamed("slide_blur", "2 2");
   //World
   messnamed("worldbox", "enable 1");
   messnamed("worldbox", "gl_color 0 0 0 1");
   messnamed("camera_preset", 0);
   messnamed("ground_enable", 1);
   messnamed("init_permanents", "bang");
   messnamed("main-light", 1.);
   messnamed("main-light", "bang");
}
```

2.24 Unity OSC Interface, Based on Jorge Garcia Martin's UnityOSC

```
//		UnityOSC - Example of usage for OSC receiver
//		Copyright (c) 2012 Jorge Garcia Martin
//		Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated
//		documentation files (the "Software"), to deal in the Software without restriction, including without limitation
//		the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
//		and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
//		The above copyright notice and this permission notice shall be included in all copies or substantial portions
//		of the Software.
//
//		THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING
BUT NOT LIMITED
//		TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL
//		THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF
//		CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE
USE OR OTHER DEALINGS IN THE SOFTWARE.

using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using UnityOSC;

public class oscControl : MonoBehaviour {

	private Dictionary<string, ServerLog> servers;

	void Start() {		// Script initialization
		OSCHandler.Instance.Init(); //init OSC
		servers = new Dictionary<string, ServerLog>();
	}

	void Update() {
		OSCHandler.Instance.UpdateLogs();
		servers = OSCHandler.Instance.Servers;
		foreach( KeyValuePair<string, ServerLog> item in servers ){
			// If we have received at least one packet,
			// show the last received from the log in the Debug console
			if(item.Value.log.Count > 0) {
				int lastPacketIndex = item.Value.packets.Count - 1;
				UnityEngine.Debug.Log(String.Format("SERVER: {0} ADDRESS: {1} VALUE 0: {2}",
					    item.Key, // Server name
					    item.Value.packets[lastPacketIndex].Address, // OSC address
					    item.Value.packets[lastPacketIndex].Data[0].ToString()));
							//First data value
			}
		  }
	 }
```