RayTone:
A Node-based Audiovisual Sequencing Environment

Eito Murakami
CCRMA, Stanford University
Stanford, CA, United States
eitom@ccrma.stanford.edu

John Burnett
UC San Diego
La Jolla, CA, United States
john.burnett.c@gmail.com

Ge Wang
CCRMA, Stanford University
Stanford, CA, United States
ge@ccrma.stanford.edu

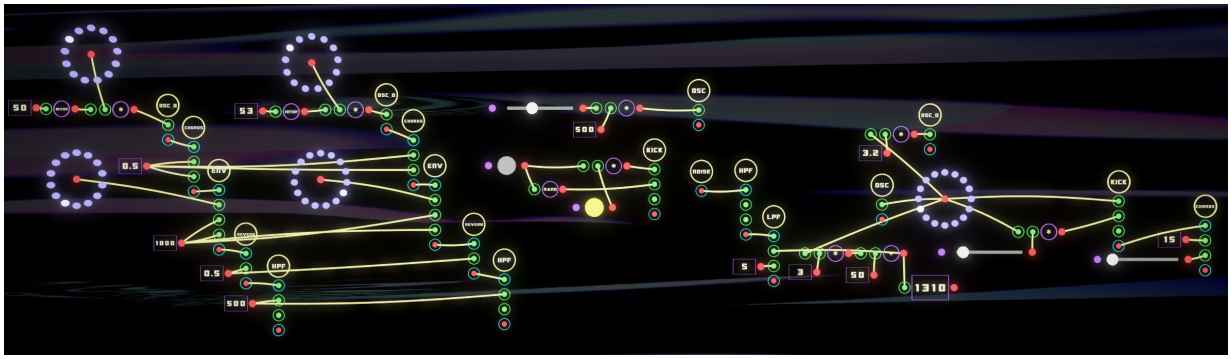

Figure 1: An example RayTone patch.

ABSTRACT

RayTone is a node-based software environment for creating audiovisual compositions. The software emphasizes the aesthetics and joy of patching procedures, aiming to promote a playful workflow for transforming creative ideas into artistic content. RayTone allows for native access to ChucK music programming language and OpenGL Shading Language (GLSL), affording programmability of arbitrary complexity inside each node on canvas. The ability to sequentially connect nodes as well as to live script functionalities therein makes RayTone suitable for users with widely varying skill levels and as an entry point to digital signal processing and shader programming. In this paper, we discuss RayTone's patching and scripting workflow and how it is designed to facilitate the integration of audio and graphics. RayTone was used as a primary educational tool for a five-day summer workshop, introducing participants to building original live multimedia performances. We present selected student compositions from the workshop as example projects, and evaluate the expressiveness of RayTone as a digital art-making platform.

Author Keywords

RayTone, multimedia, DSP, ChucK, OpenGL

CCS Concepts

•Human-centered computing → Systems and tools for interaction design; •Applied computing → Sound and music computing; Media arts;

NIME'24, 4–6 September, Utrecht, The Netherlands.

1. INTRODUCTION

In this paper, we present RayTone, a node-based software environment for creating audiovisual compositions. The software is freely available, open-source, and currently supported on Windows and macOS. RayTone is designed to be a digital canvas for sequencing audio and graphics materials simultaneously for live performances, installations, and fixed media. Users can patch nodes (called Units in RayTone) in the style of modular synthesis to manipulate control-rate or audio-rate signals and construct chains of synthesizers and audio effects. RayTone values the aesthetics and joy of patching procedures alongside technical functionalities, and seeks to reinforce users' creativity through experimentation on the canvas. The sound engine of RayTone is provided by ChucK—a strongly-timed music programming language [16]. With ChucK embedded in RayTone, users can script and modify behaviors of audio nodes that perform DSP at run time. Additionally, RayTone allows users to load OpenGL Shading Language (GLSL) [4] scripts for programming audio-reactive graphics on canvas.

2. RELATED WORK

RayTone draws from a lineage of node-based programming environments such as Pure Data [13], Max/MSP [12], and TouchDesigner [6]. RayTone is also influenced by recently introduced audiovisual software environments, such as Extempore [14], Fragment [1], Hydra [2], and ChuGL [9], which give equal precedence to audio and visual material. Each of these systems offers its unique own way of thinking and doing (and therein lies their respective value propositions). Furthermore, each environment strikes its own balance between programming paradigms (e.g., graphical-patching and text-based coding), and between support for audio and visual programming. For example, while Max/MSP/Jitter and TouchDesigner are both node-based programming environments that support audio and visuals, the nuanced differences in ethos, tool sets, and user communities result in their respective workflows.

Similarly, while RayTone is influenced by all the above systems, it offers its unique workflow and a playful approach to working with audiovisual programming. RayTone embodies an ethos of giving equal emphasis to audio and visuals; the graphics elements co-exist with audio units on the same canvas rather than a separate window. The immediacy of seeing the visual output as one edits audio parameters invites them to regard the two domains as interdependent media. Node-based patching in RayTone is designed to be quick and fun, while the integrated ChucK and shader code affords programmability of arbitrary complexity. This approach is similar to programs such as OpenMusic [11], where each node is simply an abstraction of underlying LISP code. This design makes RayTone suitable for users with widely varying skill levels and can serve as an entry point to raw DSP and shader programming.

3. DESIGN

In this section, we introduce the main components and features of RayTone as well as a typical workflow for creating a patch with different unit types.

3.1 Control + Voice + Graphics Units

There are three categories of units that comprise a RayTone patch: Voice, Graphics, and Control. Voice units process audio signals and can be scripted using ChucK code. RayTone natively includes more than 30 ready-to-use Voice units, so that there is no need for users to be familiar with the ChucK language or DSP programming to start making music. Graphics units process existing media (images, videos, camera input, etc.) as textures, which users can load in the fragment shader to be displayed on canvas. The Control units manipulate parameters made available by Voice and Graphics units. An example of a Control unit is a circular sequencer, which is quantized to the global clock and outputs a value at the corresponding step. Figure 2 illustrates the interface for editing sequencer values and properties. Control units are also used for math operations and communication with external systems through protocols such as MIDI and Open Sound Control (OSC) [17].

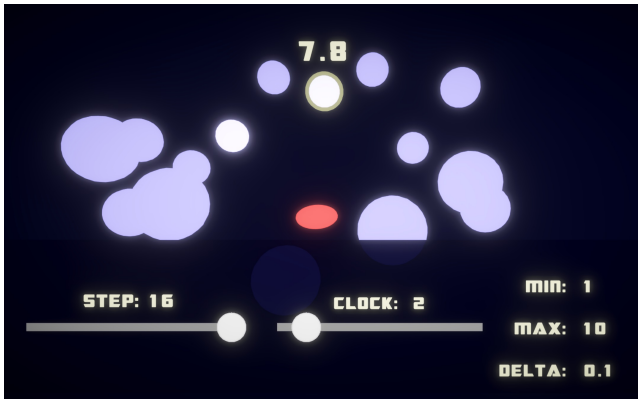

Figure 2: A sequencer unit in edit mode.
A sequence of values is represented by their height in a circular buffer. Users can edit individual step values in the specified range and increment.

3.2 Canvas Workflow

The RayTone canvas is intended to be a playful environment that promotes a non-deterministic approach to composition through experimentation. Art-making in the digital domain is influenced by tools that one chooses to work with, and the interface design should be more than simply functional to support creative exploration. [15] RayTone does not enforce traditional musical scales or dynamics, and does not feature a piano roll as seen in many digital audio workstations. One of the unique features of RayTone is the use of canvas to pan Voice units relative to the camera position. Voice units can start making sound as soon as they are placed on canvas, and invite users to experiment with placement of units and create associations between the visual patch and spatial musical output. This canvas workflow encourages users to treat musical phrases as dynamically transforming agents; a smooth transition between sections in a composition is possible by shifting the camera position to center different Voice units rather than clicking on a button or moving a slider. Figure 3 shows a patch that demonstrates the use of Voice units and Control units with graphical user interface (GUI) features, such as slider, toggle button, and value monitor. A green dot with a ring represents an inlet to the unit, and the red dot represents an outlet.[1] Voice unit sockets that carry audio-rate signals are marked by a blue ring. Two units are connected by dragging a cable from an outlet to an inlet either using a computer mouse or a finger when using a touch-screen display. (The patch cable gently swings, as if inviting the user to playfully experiment.)

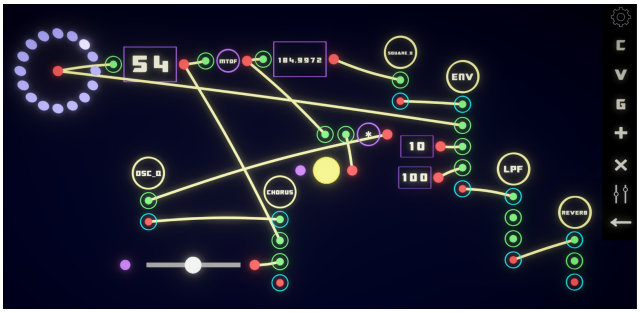

Figure 3: A patch with GUI units and chained Voice units.

4. AUDIOVISUAL INTEGRATIONS

One of the powerful features of RayTone is the ability to customize Voice and Graphics units using ChucK and GLSL. In order to facilitate communication between these scripting languages and a RayTone patch, users have access to preprocessor macros to read input values and define unit properties from code. As Figure 4 shows, syntax errors made by users in either language are displayed on the console integrated as part of RayTone canvas. This is useful for debugging code while live-scripting, and also as an educational tool for learning the two programming languages. The following sections introduce an example of writing an original Voice unit and a VFX (visual effects) unit.

[1]The sockets are designed with hover text to make the interface inclusive for color blindness.

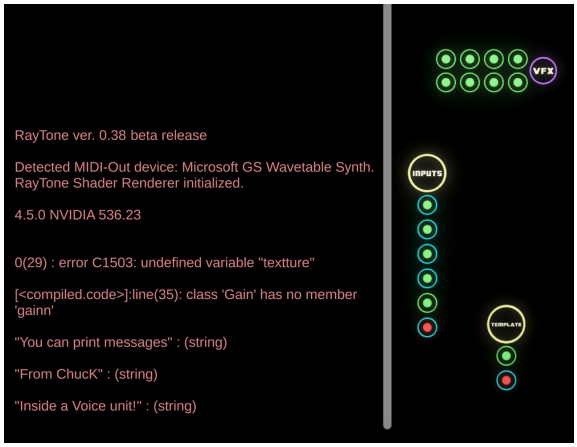

Figure 4: The console that displays syntax errors in ChucK and GLSL. It can also be used to print log messages from ChucK scripts.

4.1 Scripting DSP with a Voice Unit

A ChucK script inside a Voice unit allows users to read and process each audio sample, which opens up possibilities to create various types of signal processors, including multi-pole filters, granular synthesizers, physical models, and Fourier analyzers. A Voice unit can receive an arbitrary number of audio-rate inputs and up to 10 control-rate values. The following ChucK script is an example of using RayTone-specific preprocessor macros to define inlets and implement a playback sampler. This Voice unit loads a WAV file from a file browser, triggers the clip with `inlet 1`, and controls the playback rate with `inlet 2`.

```
//-----------------------------------------------
//Define RayTone inputs & inlets & file requirement
RAYTONE_DEFINE_INPUTS();
RAYTONE_DEFINE_INLETS("trigger", "rate");
RAYTONE_DEFINE_OUTLET(false);
RAYTONE_LOADFILE(true);
//-----------------------------------------------

// Create a signal chain using the SndBuf class
// to play an audio clip
SndBuf buf => RAYTONE_OUTPUT => dac;

// Load a .wav file to SndBuf
RAYTONE_FILEPATH => buf.read;
0 => buf.gain;

// infinite loop
while(true)
{
    // Advance time until the global clock
    RAYTONE_TICK => now;

    // Check if inlet1 received a trigger
    if(RAYTONE_TRIG(0) == 1)
    {
        // Reset SndBuf gain and playback position
        RAYTONE_LOCAL_GAIN => buf.gain;
        0 => buf.pos;

        // Read playback rate from inlet2.
        // If inlet2 is 0, set playback rate to 1.
        if(RAYTONE_INLET(1) == 0)
        {
            1 => buf.rate;
        }
        else
        {
            RAYTONE_INLET(1) => buf.rate;
        }
    }
}
```

4.2 Shader Programming with Graphics Units

RayTone's GLSL implementation allows for shader interaction via a single fragment shader and supports 2D approaches to shader programming. This paradigm is comparable to the style of shader programming seen on the website Shadertoy [5] or in programs such as KodeLife [3]. This approach allows for techniques such as procedural 2D graphics and for processing of input textures supplied by the node graph. RayTone includes a simple raymarching and signed-distance field (SDF) library written in GLSL. Users have access to a number of SDF functions representing primitive shapes in addition to functions for basic linear transforms, constructive geometry operations, and SDF manipulations. The preprocessor macro used to retrieve inlet values is identical to that of a Voice unit, facilitating the transition between audio and graphics domains. The visual output on canvas is updated in real-time as users edit the code. The following demonstrates a basic GLSL script that sets RGB values from three inlets, and samples a texture passed to the first inlet of the Textures unit:

```
#version 410

//-----------------------------------------------------------
//RAYTONE HEADER
uniform vec3 iResolution;
uniform float iTime;
uniform float inlets[8];
uniform sampler2D textures[8];
uniform vec3 textureResolutions[8];

#define RAYTONE_RESOLUTION iResolution
#define RAYTONE_TIME iTime
#define RAYTONE_INLET(i) inlets[i]
#define RAYTONE_TEXTURE(i) textures[i]
#define RAYTONE_TEXTURE_RESOLUTION(i) textureResolutions[i]
//-----------------------------------------------------------

out vec4 fragColor;
void main()
{
    // Define uv coordinates based on canvas resolution
    vec2 uv = gl_FragCoord.xy / RAYTONE_RESOLUTION.xy;

    // Read and interpret inlet values as RGB
    float r = RAYTONE_INLET(0);
    float g = RAYTONE_INLET(1);
    float b = RAYTONE_INLET(2);

    // Define texture coordinates based on its resolution
    vec2 tex1_uv =
        gl_FragCoord.xy / RAYTONE_TEXTURE_RESOLUTION(0).xy;

    // Sample texture
    vec4 tex1 = texture(RAYTONE_TEXTURE(0), tex1_uv);

    // Output is RGB + texture
    fragColor = vec4(r, g, b, 1.0) + tex1;
}
```

5. IMPLEMENTATION

RayTone for Windows and macOS is developed with Unity [8] mainly due to its cross platform compatibility, including the possibility to support iOS devices in the future. ChucK is embedded in RayTone through Chunity [10], which internally maintains a precise audio clock independent of the graphics frame rate. The shader rendering is handled by a custom plugin written in C++. The plugin takes advantage of Unity's multithreaded rendering scheme, allowing the output to be rendered on the main RayTone canvas where a patch exists, rather than in a separate window.

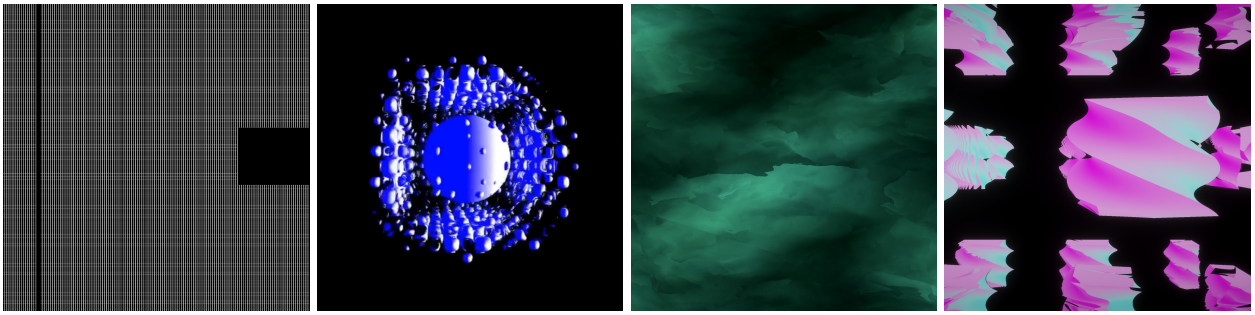

Figure 5: Screenshots of student compositions.

6. EXAMPLES AND FEEDBACK

One of the intended uses of RayTone is rapid prototyping and performing multimedia content in real-time. In order to support live patching and scripting as a form of performance, RayTone has a dual-display feature that allows users to interact with the patch on their computer screen and display the raw visual output (without units) on a separate screen for the audience. RayTone's functionalities can be further extended with the use of tactile interfaces, such as MIDI controllers and portable tablet devices. Users can take advantage of third-party OSC-compatible software, such as TouchOSC [7], to wirelessly interface with a RayTone patch on iOS and Android devices. These forms of interaction introduce tactile feedback and allow RayTone to act as a hub of communication for existing hardware and software.

6.1 Workshop Student Examples

In the summer of 2023, the authors hosted a five-day workshop at CCRMA, Stanford University titled "Real-Time AudioVisual Composition With RayTone". The background of the participants ranged from little to no experience with either computer music or computer graphics to a high degree of familiarity with these subjects. Additionally, the prior experience with text-based programming was generally low. The participants were introduced to digital signal processing and shader programming, and crafted live multimedia performances of their design. The materials taught in the workshop were split between node-based patching and text-based programming such that every participant learned to write DSP code with ChucK. The majority of participants chose to solely use the RayTone node-based interface for their final presentations, and one participant chose to customize a Voice unit with ChucK for advanced audio analysis. The following is a list of compositions by students who agreed to share their works publicly, accompanied by a shortened form of descriptions that they wrote without further alteration. Some of these compositions were created with an earlier version of RayTone that utilized High-Level Shader Language (HLSL) instead of GLSL.

- **beep-v3** (Figure 5A)
  [...] Randomized values used to control the sound are also used as inputs for the shader, alongside some extra sequencers that serve as constant beats, in order to draw lines and rectangles paired to the music.

- **Flipbook** (Figure 5B)
  In this project, [...] I had 3 sections: A, B, and C. A was more focused on tonal harmonies and chord changes through the use of ratio based modulation while B and C implemented 1 voice each, serving as contrast for when the master sequencer moved to the next section. [...]

- **Swara's Shenanigans** (Figure 5C)
  This patch incorporates several sitar note passages that can be presented in an order or at random. [...] Each of the note passages are associated with a color scheme and when triggered, results in the respective color scheme appearing in the background.

- **Wakey-wakey Dreamy-dreamy** (Figure 5D)
  Using the mic as input, participant blows across it to move the clouds to the left. Percussive sound causes the clouds to reveal their shape as twisted cubes. The RMS of the mic signal is accumulated to a value that translates the replicated cubes in the negative x direction. [...]

The showcase video that includes all of the compositions above is on the project website.

6.2 Workshop Feedback

The feature in RayTone that was most valued by the workshop participants was the ability to customize DSP and graphics code. A student stated that they found the boilerplate code provided with RayTone to be helpful. Another student commented that they enjoyed the process of sharing RayTone projects with other students. Once saved as a project file with an assets folder, a RayTone project can be opened on another computer to further modify or study the patch. Students have also indicated existing limitations of the software as a tool for live performance. First, RayTone currently uses the default audio device on the operating system, and lacks control over input and output latency (this is due to limitations in the Unity audio system). Second, a student remarked that RayTone was too computationally expensive for their performance practice. Third, native support for the Linux operating system was strongly requested. These constraints affect the usability and accessibility of the software, and will be addressed in future versions of RayTone. Overall, the current strengths of RayTone can be summarized as a wide range of customization options through the two scripting languages to create original audiovisual content. Nevertheless, it is evident from the feedback that RayTone requires further development for enhanced stability, efficiency, and inclusion of additional operating systems.

7. CONCLUSION AND FUTURE WORK

RayTone offers a unique digital environment for learning, prototyping, and performing audiovisual content. The software is guided by the principle that creativity can be reinforced through a playful interaction with the interface,

and aims to promote experimentation as an approach to composing. Live scripting DSP and graphics is an integral component of RayTone, which offers customization options and educational value. The student compositions from the workshop exemplify that RayTone can help artists create a diversity of sound-driven multimedia compositions. RayTone is designed to complement rather than replace existing audiovisual frameworks, and can act as a hub of communication through its sequencing and interconnectivity capabilities. In future versions of RayTone, we intend to address limitations indicated in the previous section, and expand a library of tutorial and example patches. We will also work towards supporting multi-channel audio output for building installations with spatial audio, as well as Linux compatibility. Finally, we foresee deploying RayTone to iOS devices as a long-term project goal.

8. PROJECT WEBSITE

A documentation of all the features as well as a showcase video from the workshop can be found on the following project website: `https://www.raytone.app`.

9. ACKNOWLEDGMENTS

The authors would like to thank CCRMA for providing equipment and resources towards the development of RayTone over the past two years. We also thank participants of the summer workshop and those who voluntarily betatested RayTone for their valuable feedback.

10. ETHICAL STANDARDS

RayTone is distributed for free as an educational tool and the authors seek no interest in commercializing the software. The summer workshop 2023 was offered solely for the purpose of inspiring audiovisual programming/compositions, and the feedback provided by the participants was optional. The composers of example projects listed on this paper or on the project website have given prior consent to share their creations.

11. REFERENCES

[1] Fragment - Web-based audio-visual live coding environment. `https://www.fsynth.com/`. Accessed: May 7, 2024.

[2] Hydra - Live Coding Video Synth. `https://hydra.ojack.xyz/`. Accessed: May 7, 2024.

[3] KodeLife. `https://hexler.net/kodelife`. Accessed: May 7, 2024.

[4] OpenGL - The Industry's Foundation for High Performance Graphics. `https://www.opengl.org/`. Accessed: May 7, 2024.

[5] Shadertoy. `https://www.shadertoy.com/`. Accessed: May 7, 2024.

[6] Touchdesigner. `https://derivative.ca/`. Accessed: May 7, 2024.

[7] TouchOSC. `https://hexler.net/touchosc`. Accessed: May 7, 2024.

[8] Unity. `https://unity.com/`. Accessed: May 7, 2024.

[9] A. Z. Aday and G. Wang. ChuGL: Unified Audiovisual Programming in ChucK. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, September 2024.

[10] J. Atherton and G. Wang. Chunity: Integrated Audiovisual Programming in Unity. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 102–107. Virginia Tech, June 2018.

[11] J. Bresson, C. Agon, and G. Assayag. OpenMusic: visual programming environment for music composition, analysis and research. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, page 743–746, New York, NY, USA, 2011. Association for Computing Machinery.

[12] M. S. Puckette. Combining Event and Signal Processing in the MAX Graphical Programming Environment. *Computer Music Journal*, 15(3):68–77, 1991.

[13] M. S. Puckette. Pure Data. In *Proceedings: International Computer Music Conference 1997*, pages 224–227. The International Computer Music Association, 1997.

[14] A. C. Sorensen. *Extempore: The design, implementation and application of a cyber-physical programming language*. PhD thesis, College of Engineering and Computer Science, The Australian National University, 2018.

[15] G. Wang. *Artful Design: Technology in Search of the Sublime, a Musicomic Manifesto*. Stanford University Press, 2018.

[16] G. Wang, P. R. Cook, and S. Salazar. ChucK: A Strongly-timed Computer Music Language. *Computer Music Journal*, 39(4):10–29, December 2015.

[17] M. Wright. Open Sound Control: an enabling technology for musical networking. *Organised Sound*, 10(3):193–200, December 2005.