Visualizing Aurora: Exploring the Theoretical Blueprint and Creative Potential of the Aurora Visual Synthesizer

xxx

Germany

Abstract

This paper examines the theoretical blueprint of Aurora, a software visual synthesizer built using VVVV Gamma with substantial integration of Fuse, an open-source GPU-powered library within VVVV Gamma. With a strong GPU focus, Aurora utilizes an array of GPU/GPGPU techniques like compute shaders, particle systems, and raymarching algorithms among others. Synthesizer Aurora takes as an approach transposing sound synthesis principles into the realm of visual expression, providing a platform to creatively explore the fusion of sound-related concepts into visually artistic outputs. Besides that it uses generative and procedural techniques to create interesting and original visual compositions. This study aims to shed light on Aurora's architecture, capabilities, and possible shortcommings. The overarching purpose of the paper is to dissect the prototyping process of Aurora and assess its effectiveness as a creative instrument.

1. Introduction

This paper explores the conceptual framework of Aurora, a software modular visual synthesizer built using VVVV Gamma (a visual programming environment and toolkit designed for creative coding and multimedia applications) [vvv23] in conjunction with the Fuse GPU-powered library. By utilizing the power of the GPU, Aurora employs a range of techniques, including compute shaders, particle systems, and raymarching algorithms. This exploration of Aurora aims to show its architecture, capabilities, and potential limitations.

Aurora's starting point is the idea of translating sound synthesis principles into the visual domain, facilitating the exploration of sound-related concepts in visual art. Its foundation lies in generative and procedural techniques, helping the creation of original visual compositions. The ultimate goal of this paper is to dissect Aurora's prototyping process and evaluate its effectiveness as a creative tool. Central to this exploration is an examination of Aurora's architecture, followed by an explanation of the components constituting its visual synthesis pipeline.

This paper also provides a concrete example showcasing Aurora's capabilities. I investigate the application of Fast Fourier Transform (FFT) for Frequency-Based Modulation (FBM), the modulation of colour shading based on frequency principles, and the integration of post-effects to amplify visual output as a study case.

Additionally, the paper examines Aurora's potential in the Audio/Visual Live Performance field, and it lays the groundwork for further developments, implementations, and testing of the Aurora synthesizer.

2. Background and Related Work

The historical evolution of visual synthesis in audio-visual live art traces its beginnings to the avant-garde movements of the 20th century. Artists like Thomas Wilfred and Nam June Paik started experimenting with the synchronization of visuals and music in live performances, creating immersive experiences.

As technology advanced, visual synthesis underwent a digital transformation. The 1980s marked the emergence of analog and digital video synthesizers, empowering artists to manipulate video signals in real-time shows. Steina and Woody Vasulka harnessed these tools to craft captivating audio-visual compositions that blurred the boundaries between visual art and music.

Early pioneers such as Mary Ellen Bute and Oskar Fischinger crafted abstract animations synchronized with music, laying the groundwork for audio-visual synchronization. These initial efforts involved manual techniques like frame-by-frame animation or object manipulation in front of projectors.

The mid-20th century brought analog electronic technologies, embraced by visionaries like Stan VanDerBeek and Nam June Paik, who explored analog video synthesis. This set the stage for live performances merging experimental music with real-time visuals.

The late 20th century saw the rise of digital technology, reshaping visual synthesis. The shift from analog to digital platforms provided artists with enhanced control. Software like Max/MSP and

VJ applications allowed real-time visual manipulation, synchronizing visuals with live music.

Collaborations among musicians, visual artists, and technologists surged in the late 20th and early 21st centuries. Festivals like L.E.V. and MUTEK facilitated these partnerships. Artists like Ryoji Ikeda and Carsten Nicolai ventured into the fusion of sound, math, and visual aesthetics, expanding the horizons of live performances. [S.23]

These advancements and collaborations highlight the growing need for diverse tools and visual synthesizers to continue pushing the boundaries of audio-visual live art. This is one of the reasons why I am working on creating the visual synthesizer Aurora, alongside my curiosity and personal need for such software.

3. Architecting Aurora: Components and Pipeline

Aurora's design revolves around a collection of modules, akin to building blocks. This approach provides Aurora with flexibility, adaptability, and versatility. The following sections delve into the specific components and tools that enable Aurora's artistic capabilities.

At Aurora's core lies a *Module Library*, akin to the foundational elements of sound synthesis. These modules—oscillators, envelopes, modulators, filters, and effects—lay the foundation for creating a variety of visual behaviors.

Within VVVV Gamma's expansive visual programming environment, users are presented with an expansive canvas—the *Patching Environment*. Here, modules are interwoven, parameters are mapped, and interactions are arranged. Experienced users have the opportunity to break limits by creating their own elements, models, nodes and reusable patches, elevating their work further.

An integrated sequencing mechanism *(Sequencing and Timelines)* enables users to arrange and trigger diverse visual patterns over time, using timelines to control the progression of visual events. For this purposes Keiros, a compositor/timeline tool in VVVV is used.

Fuse's shader development capabilities enable advanced visual effects through custom shaders, encompassing raymarching, complex geometry, and other GPU-driven techniques (using both node-based and traditional programming approaches). Besides that dynamic and organic visual behaviors are brought to life through Fluid Simulation and Particle Systems—modules that simulate fluid dynamics and particle interactions.

Parameter Mapping and Control Logic are established to map user inputs, like mouse movements or MIDI signals, to manipulate visual parameters, ensuring interactive and responsive visual outputs.

Optionally, modules can incorporate audio input to influence visual output by mapping audio frequencies to visual oscillations or using audio analysis to trigger specific visual events.

Aurora promotes collaboration and sharing through *Preset Management*. This system facilitates the storage and retrieval of visual presets, thereby encouraging the exchange of preferred configurations within its creative community.

To bring visuals to life, a robust *Rendering Engine* is pivotal. VVVV Gamma's Stride rendering engine transforms data from the patching environment into real-time visuals. Fuse's GPU capabilities are used for *Real-time GPU Processing*, letting Aurora to execute complex shaders and effects.

Making things run perfectly without breaks needs *Performance Optimization*. The optimization of shaders, modules, and interactions ensures seamless real-time performance, even with computationally intensive effects.

Aurora also equips users with the tools for *Debugging and Visualization*. These resources are used to visualize data flow, monitor parameter changes, and troubleshoot potential issues within the patch.

And the last but not the least, *Documentation and Learning Resources* enable users to comprehend Aurora's capabilities and use modules effectively.

Integrating these components, adhering to a modular framework, generates a potent visual synthesizer within VVVV Gamma and Fuse. This allows users to create dynamic and interactive visual compositions, supporting exploration and experimentation.

The Figure 1 is a graphical representation that captures the essence of Aurora's architecture and its foundational components, as well as schematic user interface of the synthesizer Aurora as outlined in the previous list of tools and concepts.

3.1. Constructing User Interaction: The Interface Design

The interface design for Aurora stands as a medium for user control. This interface allows users to interact with and manipulate the aspects of the visual synthesizer, thus enhancing their creative goals.

Creating a user-friendly and visually appealing UI/UX for a visual synthesizer made in VVVV Gamma requires consideration of design principles and user needs. Here's a conceptual description of how the UI/UX could be designed [wik23c]:

3.1.1. User Interface (UI) Elements:

See Figure 1.

- Workspace Canvas:
  The main workspace canvas serves as the central area for users to visually connect and manipulate nodes that represent visual synthesis components.
- Node Library:
  A side panel houses a library of pre-built visual synthesis nodes that users can drag and drop onto the workspace.
- Toolbar:
  A top toolbar provides quick access to common actions like saving, opening projects and exporting visuals.
- Inspector Panel:
  An inspector panel displays the properties of selected nodes, allowing users to customize parameters, such as colour, size, motion speed, and more.
- Timeline and Animation Controls:

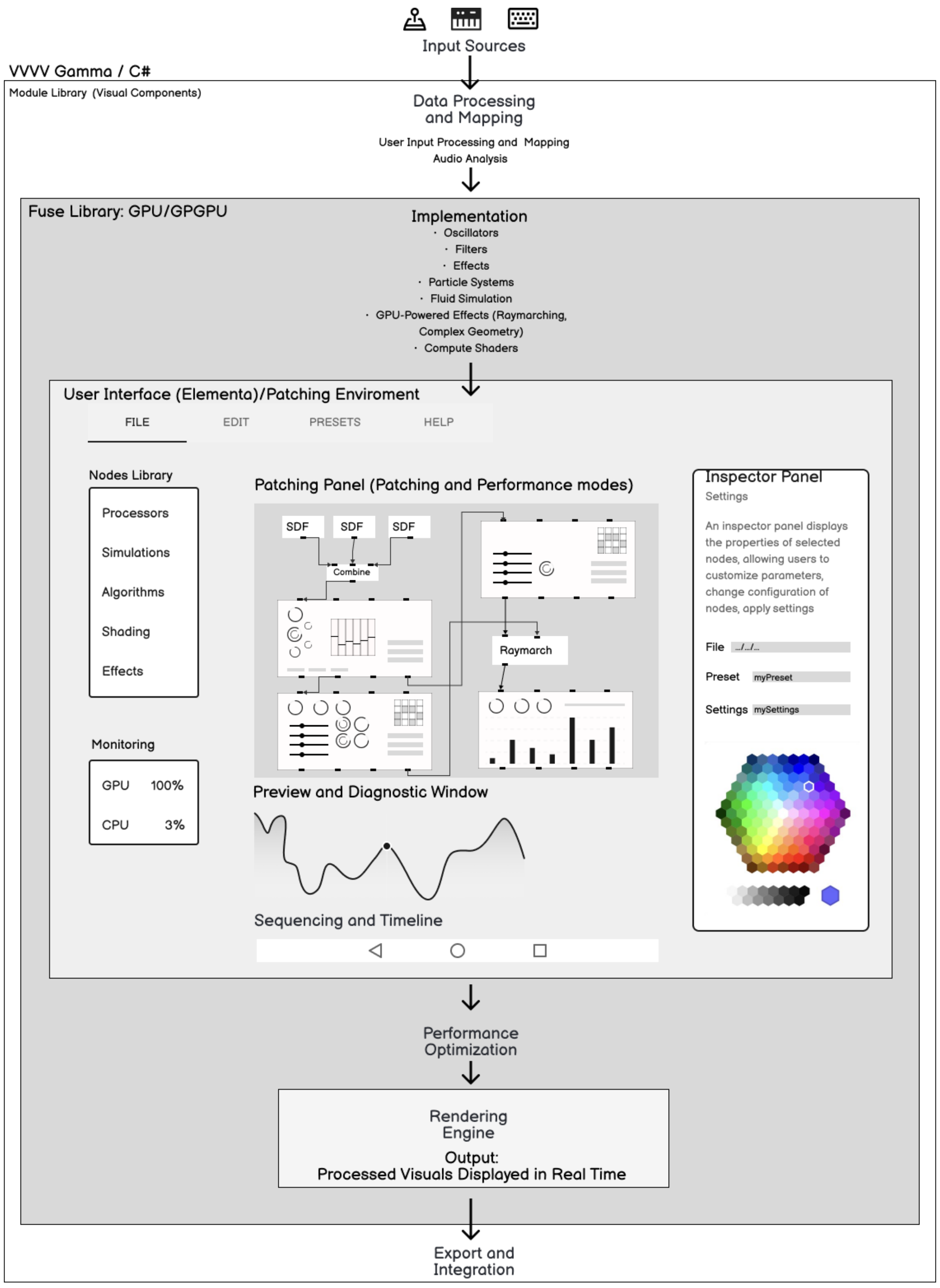

Input Sources
VVVV Gamma / C#
Module Library (Visual Components)
Data Processing and Mapping
User Input Processing and Mapping
Audio Analysis
Fuse Library: GPU/GPGPU
Implementation
· Oscillators
· Filters
· Effects
· Particle Systems
· Fluid Simulation
· GPU-Powered Effects (Raymarching, Complex Geometry)
· Compute Shaders
User Interface (Elementa)/Patching Enviroment
FILE
EDIT
PRESETS
HELP
Nodes Library
Processors
Simulations
Algorithms
Shading
Effects
Monitoring
GPU 100%
CPU 3%
Patching Panel (Patching and Performance modes)
SDF
SDF
SDF
Combine
Raymarch
Preview and Diagnostic Window
Sequencing and Timeline
Inspector Panel
Settings
An inspector panel displays the properties of selected nodes, allowing users to customize parameters, change configuration of nodes, apply settings
File ../../...
Preset myPreset
Settings mySettings
Performance Optimization
Rendering Engine
Output:
Processed Visuals Displayed in Real Time
Export and Integration

Figure 1: *Aurora Pipeline*

A timeline at the bottom enables users to create time-based visual animations. Playback controls, keyframe creation, and loop settings are accessible here.

- Modulation Panel:
  A dedicated panel allows users to apply modulation sources to visual parameters. Users can create dynamic animations by connecting modulators like LFOs to visual attributes.
- Interactive Preview Window:
  An interactive preview window displays real-time visual output, helping users visualize the effects of their settings and modifications instantly.

3.1.2. User Experience (UX) Flow:

- Workspace Interaction:
  - Users can drag and drop nodes from the library onto the canvas to build their visual synthesis setup.
  - Nodes can be connected by dragging lines between them to establish relationships.
- Node Customization:
  - Users can select nodes to access the inspector panel, where they can adjust properties and settings. - Double-clicking nodes opens a detailed node editor for advanced parameter customization.
- Modulation and Animation:
  - Users can create animation sequences by setting keyframes on the timeline. - For dynamic effects, they can use the modulation panel to link modulators to visual parameters, influencing animations over time.
- Real-Time Preview:
  - As users make changes, the interactive preview window updates in real-time, allowing them to see the immediate impact of their modifications.
- Export and Integration:
  - Users can export their visual synthesis creations in various formats, such as video files or interactive web content. - Integration with external MIDI devices or audio sources allows for interactive and synchronized performances.

3.1.3. Visual Design

- The UI should use a modern and clean design with intuitive icons and colour-coding for node categories.
- Nodes can be represented as draggable visual elements, possibly resembling the type of visual effect they generate.
- Animations and transitions within the UI should be smooth and responsive, enhancing the user experience.

3.1.4. Accessibility and User Guidance:

- Tooltips provide context-sensitive information when hovering over UI elements.
- Tutorials, tooltips, or a built-in guide help new users understand the basics of connecting nodes and applying visual effects.

3.1.5. Customizability:

- Users can rearrange UI elements and save their preferred layouts for different projects.
- Advanced users might have access to customizable shortcuts or scripting options for enhanced flexibility.

In summary, the UI/UX of a visual synthesizer made in VVVV Gamma using VVVV FUSE should prioritize ease of use, real-time visualization, and customization. It should provide an intuitive environment for artists and creators to bring their audio-visual visions to life, providing a seamless and immersive creative experience.

4. Conceptual Approach: Generating FBM Landscape with FFT and Raymarching

As a theoretical case study for the visual synthesizer Aurora, I will explore the creation of a generative landscape using the raymarch algorithm, FFT to FBM mapping, frequency modulation concepts for shading, and the Chroma Shift effect inspired by the "Chorus" effect commonly used in sound synthesis.

The schematic representation of this process is shown in Figures 2.1, 2.2, and 2.3.

4.1. FFT for Terrain Generation

[Pau98]

Figure 2.1 represents following algorithms:

1. Begin by creating a 2D noise field that will serve as the basis for the landscape. This noise field is often referred to as the height map, representing elevation data across the terrain.

2. Apply the FFT algorithm to transform this 2D noise field into the frequency domain. The result is a spectrum of frequencies that contribute to the overall terrain features.

3. Modify the amplitude of these frequencies to control the variations in the terrain. Lower frequencies will create large-scale features like mountains, while higher frequencies will introduce finer details such as valleys and ridges.

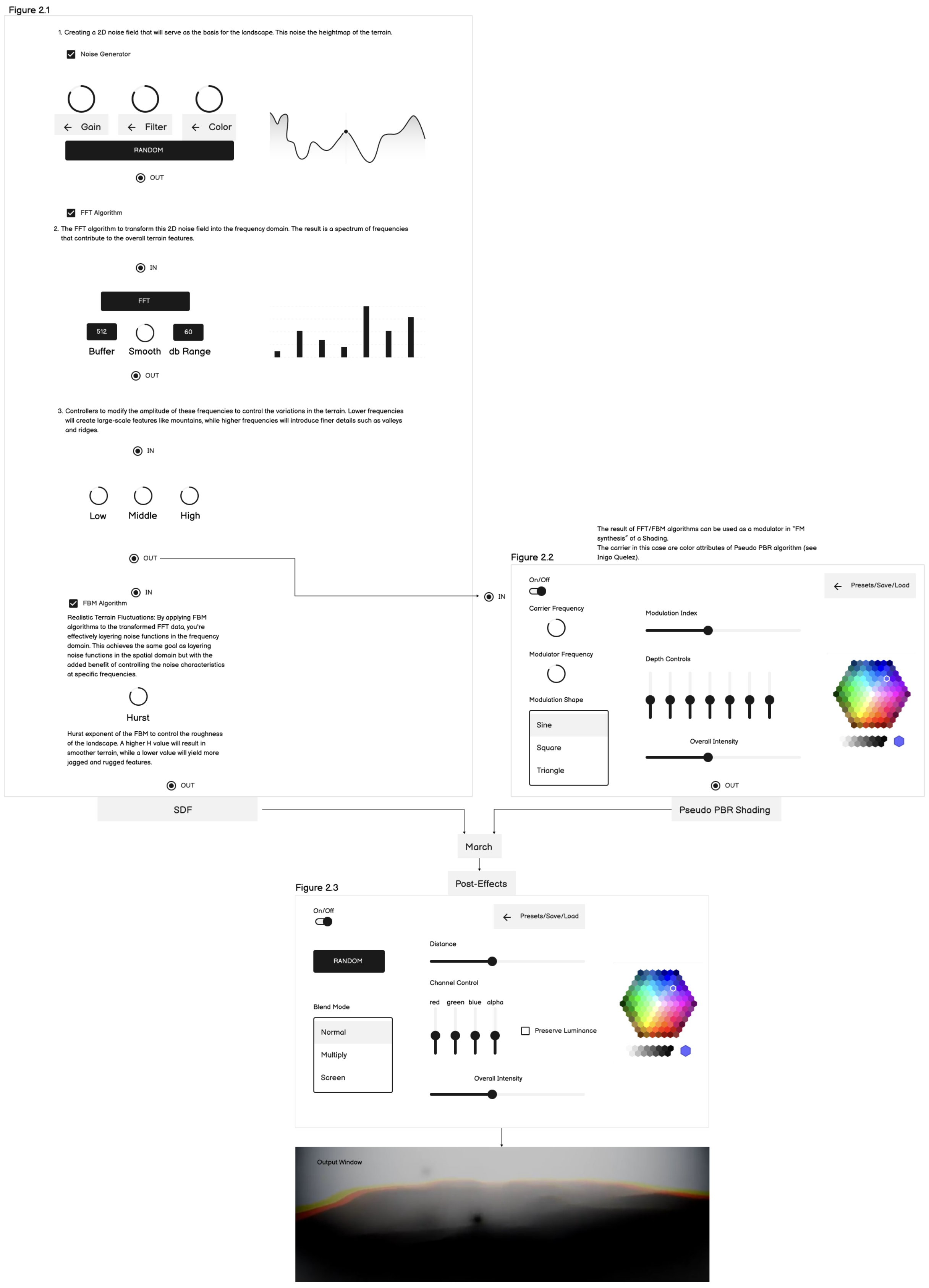
4. Fractional Brownian Motion (FBM) Integration [Ini19a]: - FBM is a method to create natural-looking randomness by layering noise functions of different scales and amplitudes. Apply FBM algorithms to the transformed FFT data to introduce realistic terrain fluctuations. - Adjust the H parameter (Hurst exponent) of the FBM to control the roughness of the landscape. A higher H value will result in smoother terrain, while a lower value will yield more jagged and rugged features.

5. Raymarching for Visualization [Ini19b]: - To visualize the generated landscape, implement raymarching techniques. Raymarching involves sending rays from a camera viewpoint and iteratively marching along the rays to determine the intersection with the terrain. - Create raymarching shaders that incorporate lighting models, shadows, and atmospheric effects to enhance the realism of the landscape. Fuse within Aurora provides the tools to develop and integrate these shaders seamlessly.

4.2. FM Synthesis for Shading

Figure 2.2 represents following algorithms:

Frequency Modulation (FM) synthesis involves modulating the frequency of one waveform (the carrier) using another waveform (the modulator). In the context of visual synthesis, we can interpret



1. Creating a 2D noise field that will serve as the basis for the landscape. This noise the heightmap of the terrain.

2. The FFT algorithm to transform this 2D noise field into the frequency domain. The result is a spectrum of frequencies that contribute to the overall terrain features.

3. Controllers to modify the amplitude of these frequencies to control the variations in the terrain. Lower frequencies will create large-scale features like mountains, while higher frequencies will introduce finer details such as valleys and ridges.

Realistic Terrain Fluctuations: By applying FBM algorithms to the transformed FFT data, you're effectively layering noise functions in the frequency domain. This achieves the same goal as layering noise functions in the spatial domain but with the added benefit of controlling the noise characteristics at specific frequencies.

Hurst exponent of the FBM to control the roughness of the landscape. A higher H value will result in smoother terrain, while a lower value will yield more jagged and rugged features.

The result of FFT/FBM algorithms can be used as a modulator in "FM synthesis" of a Shading.
The carrier in this case are color attributes of Pseudo PBR algorithm (see Inigo Quelez).

Figure 2: *Generating FBM Landscape with FFT and Raymarching*

these waveforms as visual attributes (such as colour, brightness, or texture), and the modulation process is dynamically altering these attributes based on the shapes' positions [wik23b].

When applying FM synthesis to shading, we can consider the following components:

1. Modulator and Carrier Attributes: Choose two sets of visual attributes (e.g., colour, texture) to represent the modulator and carrier components. The modulator will influence the carrier's attributes over space or time.

2. Frequency Modulation: The frequency of the modulator will determine the rate at which the carrier attributes change. A higher modulator frequency will lead to more rapid changes.

3. Modulator Frequency Knob: A knob that adjusts the frequency of the modulating waveform. Users can control how fast the modulation occurs, affecting the rate of change in the visual attributes.

4. Carrier Frequency Knob: A knob to adjust the frequency of the carrier waveform. This determines the main frequency of the visual output.

5. Modulation Index Slider: A slider to control the modulation index, which influences the depth and intensity of the FM modulation. This can create different levels of visual variation.

6. Depth Controls: Sliders or knobs that determine the depth of FM modulation for different visual components. Users can control how much each component is modulated.

7. Modulation Shape Selector: A dropdown or buttons to select different modulation waveforms (sine, square, triangle, etc.). Users can experiment with different modulation shapes and their impact on the shading effect.

8. Overall Intensity Slider: A master slider to control the overall intensity or strength of the FM modulation effect.

9. Color Picker: As before, an interactive colour picker to select the colours of different visual components influenced by FM synthesis.

10. Presets/Save/Load: Buttons to save and load presets for different FM synthesis configurations, along with options to share or export these presets.

Overall, FM synthesis in visual shading introduces dynamic variations in visual attributes based on the principles of frequency modulation, resulting in responsive shading effects as the shapes move through the scene.

4.3. Chorus" for post-effects

Figure 2.3

"Chroma Shift" effect for Aurora's visual synthesizer. This effect is inspired by the "Chorus" effect in sound synthesis, which creates a richer, thicker sound by introducing slight variations in pitch and timing. In the visual context, Chroma Shift effect introduces subtle color variations and shifts to enhance the visual experience [wik23a].

The Chroma Shift effect aims to simulate the visual equivalent of a chorus effect in sound synthesis by introducing slight color variations and shifts to create a more dynamic and vibrant visual composition.

To achieve this effect, we'll use a compute shader to process the rendered image and apply controlled color variations to nearby pixels, simulating the visual equivalent of pitch and timing variations in a chorus effect.

To control and interact with the Chroma Shift effect in Aurora's user interface (UI), various knobs, sliders, and controllers can be incorporated, that allow users to adjust the parameters of the effect in real-time. Here are some UI elements that could be used:

1. Intensity Slider: Use a slider control to adjust the intensity of the chroma shift effect. This slider would control the amount of color variation introduced by the effect. Users can slide the control to the left for subtle shifts and to the right for more pronounced variations.

2. Distance Slider: Implement a slider to control the maximum distance over which the chroma shift effect is applied. This parameter determines the range within which nearby pixels are affected by the color shifts.

3. Randomization Button: a button that triggers a randomization of the chroma shift parameters. This can be a fun feature that encourages experimentation and can lead to unexpected visual results.

4. Color Pickers: allows to specify the color of the shift introduced by the effect. This can add an artistic touch to the visuals.

5. Channel Control Sliders: for adjusting the shift intensity for red, green, blue and alpha channels individually.

7. Preserve Luminance Checkbox: a checkbox that enables users to preserve the luminance (brightness) of the original colors while applying the chroma shifts.

5. Performance and optimization

Performance and optimization are crucial factors when designing a visual synthesizer to ensure smooth real-time operation and efficient use of system resources. Here are some considerations to keep in mind:

1. Node Efficiency:

- Optimization of the code and logic within each node to reduce computational overhead.

- Minimization of unnecessary calculations and operations to improve overall performance.

2. Real-Time Rendering:

- Prioritization of techniques that allow for efficient real-time rendering of visuals.

- GPU acceleration to offload rendering tasks and enhance performance.

3. GPU Resources:

- Monitoring of the GPU's memory and processing capabilities.

- Avoiding too many complex visual elements simultaneously, as it can lead to slowdowns.

4. Caching and Precomputation:

- Caching mechanisms for frequently used visuals to avoid redundant calculations.

- Precomputation/storage of intermediate results for certain effects to reduce real-time computational demands.

5. Level of Detail (LOD):

- Level of detail techniques to adjust the complexity of visuals based on their distance from the viewer. This can reduce the GPU load and improve performance for distant or less visible elements.

6. Dynamic Quality Settings:

- Adjustable quality settings to allow users to balance visuals and performance according to their system capabilities.

7. Asynchronous Processing:

- Asynchronous processing where possible to offload non-real-time tasks from the main rendering thread. This can help maintain responsiveness even during computationally intensive operations.

8. Frame Rate Management:

- Frame rate management to maintain a stable performance without straining the system.

9. Resource Management:

- to avoid memory leaks and ensure smooth operation.

10. Parallelization:

- to leverage multi-core processors for better performance. Distributing workload across multiple cores can help distribute computational demands.

Balancing visual complexity and real-time performance is a constant challenge in visual synthesis. By implementing optimization strategies and considering the system's capabilities, a visual synthesizer can deliver smooth user experience while making efficient use of resources.

6. Creative Applications and Artistic Potential of Aurora

- Audio-Visual Performances - Aurora empowers live performers to create audio-visual experiences, synchronizing visuals with music or soundscapes in real-time.
- Interactive Installations - artists can design interactive art installations that respond to touch, motion, or sound, allowing viewers to influence and shape the visuals they experience.
- Digital Art Exhibitions - visual artists can use Aurora to generate digital artworks that evolve over time, creating ever-changing visual narratives in gallery setting.
- Music Videos and Visual Storytelling - Aurora's real-time animation capabilities enable the creation of unique music videos that synchronize with the rhythms and emotions of the music.

7. Discussion: Difficulties and Limitations in Implementing Aurora

The conceptual framework of Aurora might hold the potential for the new way of visual synthesis, however, like any ambitious project, there are certain challenges and limitations that must be acknowledged and addressed. In this section, those difficulties and limitations that might arise during the implementation of Aurora are discussed.

1. *Complexity of GPU Programming*: While the utilization of GPU acceleration is a cornerstone of Aurora's design, programming for the GPU can be intricate and demanding. Developing efficient shaders and compute kernels requires a deep understanding of graphics programming, and optimizing them for real-time performance can be a complex task. A lot of time needed to be invested in learning and mastering GPU programming concepts to employ Aurora's potential.

2. *Performance vs. Complexity Trade-off*: The pursuit of creating visually interesting effects can sometimes come at the cost of performance. Complex shaders and effects might strain the GPU and impact the real-time rendering capabilities of Aurora. Keeping the right balance between visual complexity and real-time performance becomes a crucial consideration.

3. *Learning Curve and Accessibility*: Aurora's modular architecture can offer a wide array of creative possibilities. However, this modularity can also lead to a steep learning curve, especially for newcomers to visual synthesis and programming. Providing accessible documentation, tutorials, and resources becomes essential to help users unlock the software's potential.

4. *Hardware Limitations*: Aurora's dynamic and real-time nature heavily relies on the capabilities of the hardware it runs on. Users with older or less powerful hardware may experience limitations in terms of the complexity of effects they can create and the real-time performance they can achieve. As a result, users may need to adjust their expectations based on their system specifications.

5. *Artistic Expression vs. Technical Constraints*: The balance between artistic expression and technical constraints can sometimes be challenging to deal with. In such cases, compromises might be necessary to maintain performance and usability.

6. *Integration Challenges*: Aurora's architecture relies on a seamless integration of various components and libraries, including VVVV Gamma and the Fuse GPU-powered library. Ensuring these components work harmoniously together and receive timely updates and support can pose integration challenges. Compatibility issues and updates to individual components might require continuous monitoring and adaptation.

7. *Real-Time Debugging and Performance Optimization*: Real-time environments demand efficient debugging and optimization techniques. Debugging complex shader code and real-time interactions can be challenging due to the dynamic nature of visual synthesis. Implementing tools and workflows that facilitate debugging and performance optimization becomes crucial to ensure a smooth experience.

8. *Resource Management and Stability*: Real-time rendering places considerable strain on system resources, especially GPU

memory and processing power. Proper resource management and stability become critical to prevent crashes, freezes, and other undesirable behaviors that could hinder the user experience.

9. *Scaling and Flexibility*: Maintaining scalability and flexibility becomes important. The software should be able to accommodate projects of varying sizes and complexities while ensuring consistent performance and stability.

10. *Iterative Development and User Feedback*: Given the complexity of Aurora's architecture, it might require iterative development cycles and ongoing user feedback to refine its features, performance, and usability.

In conclusion, realization of the Aurora's conceptual blueprint involves tackling challenges related to GPU programming, performance optimization, user accessibility, and balancing artistic intent with technical constraints.

8. Conclusion

This paper makes an attempt to exemine a space where sound principles intersect with visual art. Through utilizing the GPU's power and adopting a modular approach, Aurora offers a platform for creative potential.

Starting from its foundation in translating sound into visuals, Aurora's generative and procedural techniques can provide artists with a versatile toolbox. The user-friendly interface, sequencing tools, and shaders combine to enable real-time experimentation. Whether it's working on live performances, interactive installations, or music videos, Aurora may serve as a tool for artistic expression.

However, the process has its own challenges. Handling the difficulties of GPU programming, balancing performance with complexity, limitations imposed by hardware, the need for artistic compromise and addressing the learning curve have been discussed. Additionally, an optimizing resource usage, ensuring system stability are all important parts of creating such a software have to be taken into consideration.

In conclusion, this paper serves as a foundation for future work on the Aurora visual synthesizer, which, hopefully, will be valuable for artists, musicians, and technicians, providing a platform for exploration and experimentation in the realm of audio-visual art.

References

[Ini19a] Inigo Q.: fbm - 2019. *https://iquilezles.org/articles/fbm/* (2019). 4

[Ini19b] Inigo Q.: Raymarching sdf. *https://iquilezles.org/articles/raymarchingdf/* (2019). 4

[Pau98] Paul B.: Generating noise with different power spectra laws. *https://paulbourke.net/* (1998). 4

[S.23] S. G.: *Live visuals history, theory, practice*. Routledge, 2023. 2

[vvv23] Chorus (audio effect). https://visualprogramming.net/, 2023. Viewed August 28, 2023. 1

[wik23a] Chorus (audio effect). https://en.wikipedia.org/wiki/Chorus_(audio_effect), 2023. Viewed August 28, 2023. 6

[wik23b] Frequency modulation synthesis. https://en.wikipedia.org/wiki/Frequency_modulation_synthesis, 2023. Viewed August 28, 2023. 6

[wik23c] Principles of user interface design. https://en.wikipedia.org/wiki/Principles_of_user_interface_design, 2023. Viewed August 28, 2023. 2