A REPORT

ON

Project Boxxy: Open Source GPU Remote Access & Development Suite

BY

JOEL TONY **2021A7PS2077G**

AT

Coditation Systems

A Practice School – I Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI, GOA CAMPUS

July 2023

A REPORT

ON

Project Boxxy: Open Source GPU Remote Access & Development Suite

BY

JOEL TONY **2021A7PS2077G**

Prepared in partial fulfilment of the Practice School-I Course BITS F221

AT

Coditation Systems

A Practice School – I Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI, GOA CAMPUS

July 2023

Report Details
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI - (RAJASTHAN)

Practice School Division

Station: Coditation Systems	**Centre:** Pune, India
Duration: 30th May - 22nd July 2023	**Date of Submission:** 24th June 2023

Title: Project Boxxy: Open Source GPU Remote Access & Development Suite

Name of student: Joel Tony
ID Number: 2021A7PS2077G

Name of Experts	**Designation**
Mr. Shubham Kamthania	Technical Lead
Ms. Madhuri Pulipaka	Project Manager

Name of PS Faculty	**Designation**
Dr. Ravi Kadlimatti	Asst. Professor, Dept. of EEE

Keywords: Open Source, GPU, Remote, Graphics, Graphics Programming, WebGPU, WebRTC

Project Areas: Graphics Programming

Signature of Student **Signature of PS Faculty**

Acknowledgements

I would like to extend my special thanks to our mentor Dr. Ravi Kadlimatti, for being a constant support throughout the duration of the project. Also, I would like to thank Coditation Systems for providing me with the opportunity of working on an interesting project. Lastly, I would like to acknowledge the role of BITS Pilani in giving me this exposure through the Practice School-I course.

Contents

Chapter 1

Project Abstract

This project report presents the progress and outcomes of Project Boxxy, which I was a part of during my Practice School-I (BITS F221) program. Project Boxxy aims to develop an open-source sandbox environment tailored for graphics payloads. The project includes:

- A WebRTC component for delivering graphics streams on the web.
- A native WebGPU component for real-time graphics rendering on the server.
- DevSync for real-time code editing.
- A remote caching system for compiled resources.

The primary objective is to provide affordable and accessible GPU resources to developers and researchers working on resource-intensive tasks. By offering a sandbox environment specifically designed for graphics development and deployment, Project Boxxy addresses the challenge of expensive and inaccessible high-end GPUs. Through an open-source approach, it fosters collaboration and innovation while democratizing GPU resources. The project seeks to empower developers and researchers with an affordable and efficient platform for developing and deploying graphics-intensive applications.

Chapter 2

Project Overview

2.1 Project Description

Project Boxxy focuses on developing an open-source sandbox environment specifically designed to handle graphics payloads. The project encompasses various components, including a WebRTC module for streaming graphics on the web, a native WebGPU module for real-time graphics rendering on the server, DevSync for seamless real-time code editing, and a remote caching system for compiled resources.

The main objective of Project Boxxy is to create an accessible and cost-effective solution for developers and researchers working on graphics-intensive tasks. The project addresses the challenge of expensive and limited access to high-end GPUs by offering a specialized sandbox environment. By leveraging the open-source model, Project Boxxy promotes collaboration, innovation, and inclusivity within the graphics development community.

The sandbox environment provided by Project Boxxy serves as a platform for developers and researchers to experiment, create, and deploy graphics-intensive applications without costly hardware investments. Through the integration of the WebRTC component, graphics streams can be efficiently delivered on the web, enabling remote access and interaction with graphical applications. The native WebGPU component ensures real-time graphics rendering capabilities on the server, facilitating high-performance graphics processing within the sandbox environment. Additionally, DevSync allows for real-time code editing, enhancing productivity and teamwork. The remote caching system optimizes performance by reducing the need for repetitive compilation, resulting in faster execution times for resource-intensive tasks.

Overall, Project Boxxy aims to democratize GPU resources and empower developers and researchers by providing an affordable, accessible, and feature-rich sandbox environment for developing and deploying graphics payloads.

2.2 Motivation Behind the Project

The motivation behind Project Boxxy stems from the high cost and limited accessibility of high-end GPUs, which pose significant challenges for developers and researchers working on resource-intensive tasks. The expense of acquiring high-end GPUs makes them out of reach for many individuals and organizations, hindering their ability to delve into graphics development and innovation.

To address this issue, Project Boxxy aims to provide an alternative solution by offering an affordable and accessible sandbox environment designed for developing and deploying graphics payloads. By creating this platform, the project seeks to democratize GPU resources and allow individuals and organizations with limited resources to participate in graphics-driven development actively.

The motivation to offer an open-source sandbox environment for graphics payloads is driven by the belief in collaboration and knowledge sharing within the graphics development community. By providing a cost-effective solution, Project Boxxy empowers developers and researchers to explore the full potential of graphics-intensive tasks, encouraging innovation and fostering a more inclusive environment.

Ultimately, the motivation behind Project Boxxy is to bridge the gap between the prohibitive costs of high-end GPUs and the need for affordable and accessible GPU resources. By providing a dedicated sandbox environment, the project enables individuals and organizations to overcome financial barriers and engage in efficient and cost-effective graphics development.

Chapter 3

Project Requirements

3.1 Functional Requirements

3.1.1 WebRTC

- Enable real-time delivery of graphics streams on the web.
- Support secure and efficient streaming of graphics data.
- Ensure compatibility with modern web browsers and platforms.

3.1.2 WebGPU

- Develop a native component for real-time graphics rendering on the server using WebGPU.
- Support efficient and optimized graphics rendering techniques.
- Handle shader languages such as GLSL or SPIR-V.

3.1.3 DevSync

- Enable real-time code editing, compilation, and hot reloading of the graphics viewport.
- Support a seamless development experience for developers working on graphics payloads.
- Integrate development tools and frameworks that facilitate live code updates and hot module replacement.

3.1.4 Compiler Infrastructure

- Implement remote caching of compiled resources to reduce the turnaround time for graphics compilation.
- Utilize distributed caching systems or cloud-based storage solutions for storing compiled resources.
- Optimize resource management and ensure efficient compilation processes.

3.1.5 Collaboration and Coherence

- Enable collaboration among developers and researchers working on the project.
- Utilize version control systems (e.g., Git) and project management tools (e.g., GitHub, Jira) for effective collaboration.
- Foster a coherent and collaborative end-to-end developer experience.

3.1.6 Open Source

- Follow open-source development practices and guidelines.
- Enable easy contribution from the community.
- Comply with open-source licenses and requirements.

3.1.7 Cloud Infrastructure

- Enable deployment of the remote access system using cloud-based infrastructure and services.
- Manage GPU resources efficiently within the cloud environment.
- Ensure scalability, reliability, and security of the cloud infrastructure.

3.1.8 Security

- Employ authentication and authorization mechanisms to control user access.
- Ensure secure remote access to the GPU-based applications.
- Protect sensitive data and prevent unauthorized access or data breaches.

3.1.9 Documentation and Testing

- Provide clear and comprehensive documentation for developers and users.
- Utilize testing frameworks and methodologies to ensure the reliability and quality of the applications.

3.2 Non-Functional Requirements

3.2.1 Performance

- Provide real-time graphics rendering and streaming with minimal latency.
- Optimize resource utilization for efficient compilation and caching processes.
- Support a scalable infrastructure to handle multiple concurrent users.

3.2.2 Usability

- Offer an intuitive and user-friendly interface for developers and researchers.
- Provide clear instructions and documentation for setup, configuration, and usage.
- Support seamless integration with popular development environments and tools.

3.2.3 Reliability

- Ensure high availability and uptime of the remote access system.
- Implement error handling and recovery mechanisms to handle exceptions and failures.
- Regularly monitor and maintain system stability and performance.

3.2.4 Security

- Implement robust security measures to protect user data and prevent unauthorized access.
- Follow security best practices and industry standards.
- Regularly update and patch the system to address security vulnerabilities.

3.2.5 Scalability

- Support a growing user base and increasing demands for GPU resources.
- Scale the infrastructure seamlessly to handle additional users and workloads.
- Optimize resource allocation and management for scalability.

3.2.6 Compatibility

- Support major web browsers and platforms for web-based delivery of graphics streams.
- Integrate with popular development tools and frameworks for code editing and compilation.
- Ensure compatibility with various operating systems and hardware configurations.

Chapter 4

Risks and Mitigation Strategies

Risk	Mitigation Strategy
Insufficient community involvement and lack of contributions	• Implement effective community engagement strategies • Provide clear guidelines for contributions • Actively seek feedback from the community
Performance and scalability issues arise due to increased usage	• Regularly assess and optimize system resources • Allocate resources according to usage patterns • Introduce load balancing techniques to handle additional demands

Chapter 5

My Contributions on Project Boxxy

5.1 Build Systems

I spend the initial week exploring building systems. A build system plays a crucial role in the development workflow of software projects. It automates compiling source code, linking dependencies, and generating executable or deployable artefacts. They also help in managing the project dependencies and their versions.

There are two significant kinds of build systems, artefact-based and task-based. Most early build systems like `make` and `ant` are task-based. They execute a series of tasks step-by-step to generate the final artefact. This means that for every small change, all the steps are performed again. The other type of build system is artifact-based. They execute only the tasks that are required to generate the final artefact. This is done by maintaining a dependency graph of the tasks and their inputs and outputs. This means that for every small change, only the tasks affected are executed, saving time.

I explored Bazel, a popular artefact-based build system developed by Google. I also studied some optimisations it has, like remote caching and remote execution. Remote caching allows users to utilise a shared cache of build artefacts, which can be used to speed up builds. Remote execution will enable users to execute build tasks on remote machines, which can be used to speed up builds and build for different architectures.

5.2 Dependency Management for C++ Projects

The following week, I spent time understanding how C++ projects manage their dependencies, and integrate them with a build system for OS-agnostic builds. I looked into `vcpkg` which is a package manager used to manage C/C++ libraries. It is often used in tandem with CMAKE, which is a popular build system for C++ projects.

5.3 WebGPU

We then pivoted roles, and I started working on the WebGPU implementation. I began by reading the WebGPU specification and understanding the API and its features. Since I did not have prior experience with graphics programming, I spent some time learning the basics of graphics programming and the graphics pipeline. This was an exciting experience, and I learned much about graphics programming. I learnt about vertices, rasterisation, ray tracing and shaders. I also learnt about the various file formats used in graphics programming, like GLSL, SPIR-V, and HLSL.

WebGPU currently has two significant distributions: Dawn from Google, written in C++ and GPU-native from Mozilla, written in Rust. Dawn has better quality-of-life features, like better error messages and better documentation. However, it required to be built from the source, which was a challenging process. Conversely, GPU-native is available as a pre-built binary and is easier to set up. Wgpu-native would be preferred for minor projects due to its simple setup process.

5.4 Point Clouds

The last stage of the project deals with rendering point clouds. We were provided with raw pint-cloud data from a 3D scanner in a proprietary format. This was then to be investigated and rendered. Following this, we could convert the data to a standard format of coordinate colours and render it using `open3d` in Python. We then exported the point cloud as a PCD file, an industry-standard format for point-cloud data. This was a complex feat, as the data from the 3D scanner was not in a standard format and required some processing before it could be rendered. I then understood the PCD format specification and how its header related to the succeeding data.

We are now exploring Three.js for loading and rendering the PCD file in the browser. We are also exploring the possibility of rendering the point cloud in the browser using WebGPU.

Chapter 6

Conclusion

Project Boxxy is an open-source solution that develops a sandbox environment for GPU-based development and deployment. Its goal is to provide accessible and affordable resources that address the challenges related to high-end GPU costs and limited collaborative development experiences. We could illustrate the project's functional, non-functional, and general requirements through diverse layouts and imagery.

I learned a lot about graphics programming and how systems are used in the industry. My PS1 at Coditation Systems proved to be a very enriching experience, with many learnings for me. I am glad that I could work on a project I was passionate about.

Chapter 7

References

- “Apache Ant - Welcome.” Ant.apache.org, ant.apache.org/.
- Atlassian. “Jira Cloud.” Atlassian, 2019, www.atlassian.com/software/jira.
- “Bazel.” Bazel, bazel.build/.
- “CMake.” Cmake.org, 2018, cmake.org/.
- “Dawn - Git at Google.” Dawn.googlesource.com, dawn.googlesource.com/dawn.
- Git. “Git.” Git-Scm.com, 2019, git-scm.com/.
- GitHub. “GitHub.” GitHub, 2023, github.com/.
- Open3D – a Modern Library for 3D Data Processing. www.open3d.org/.
- “SPIR - the Industry Open Standard Intermediate Language for Parallel Compute and Graphics.” The Khronos Group, 20 Jan. 2014, www.khronos.org/spir/. Accessed 17 July 2023.
- stevewhims. “High-Level Shader Language (HLSL) - Win32 Apps.” Learn.microsoft.com, learn.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl.
- “WebGPU.” Www.w3.org, www.w3.org/TR/webgpu/.
- “WebRTC Home — WebRTC.” Webrtc.org, 2017, webrtc.org/.
- “Wgpu-Native.” GitHub, 14 July 2023, github.com/gfx-rs/wgpu-native. Accessed 17 July 2023.
- Vcpkg.io, 2023, vcpkg.io/. Accessed 17 July 2023.