3D Gaussian Splatting for Real Time Radiance Field Rendering

Team Members: Namratha Reddy Kareddy (nxk220058)
Paavana Sai Reddy Bandi (pxb220034)

Problem Summary: Develop a machine learning-based rendering system that uses Gaussian models to generate photorealistic images from given viewpoints in a scene. The system should be capable of refining these models through iterative training to minimize the difference between rendered images and ground-truth images, using a combination of loss functions and various rendering techniques.

Description of work: In the project, our goal was to develop a machine learning-based rendering system capable of producing photorealistic images from 3D data using Gaussian models. The work involved several phases, including initial planning, development of the rendering engine, iterative training, and evaluation of the models.

Development and Implementation:

➔ **Initial Setup and Planning:**

We began by outlining the necessary components for the project, such as the Gaussian model, the rendering pipeline, and the network interface for GUI interactions.

We identified the key performance metrics like L1 loss and SSIM, and set up a basic iterative training loop.

We have used a git repository form GraphDeco – INRIA to setup the rendering files and to setup the gaussian model and environment files. But unfortunately, we were unable to set it up on our local hardware as we have a MacBook so we use colab to run our work and we made use of an online website (which I have mentioned below) to run the .ply file.

Rendering website: https://antimatter15.com/splat/

GitHub link: https://github.com/paavansaireddy/3DGaussianSplatting

Dataset: https://huggingface.co/camenduru/gaussian-splatting/tree/main

All our work Is in the mentioned python notebook:
https://colab.research.google.com/drive/1kYJWJ9D6idA9m1bgTuYv3Tt8Izyb_mKs?usp=sharing

➔ **Training Mechanisms and Loss Functions:**
We implemented a machine learning training system focused on generating high-quality renderings from 3D models using Gaussian techniques. It is well-equipped with modern software practices such as modularity, real-time interactivity, detailed logging, and performance tracking. It follows an iterative training mechanism where the Gaussian model parameters were adjusted based on the loss computed between the rendered images and ground-truth images.

Loss functions used in our code:

➔ **L1 Loss (Absolute Difference Loss)**

$$L1(image, gt\_image) = (1/N) * \Sigma |image\_i - gt\_image\_i|$$

➔ **Structural Similarity Index (SSIM)**

$$SSIM(image, gt\_image) = [(2 * \mu\_image * \mu\_gt\_image + c1) * (2 * \sigma\_image\_gt\_image + c2)] / [(\mu\_image^2 + \mu\_gt\_image^2 + c1) * (\sigma\_image^2 + \sigma\_gt\_image^2 + c2)]$$

➔ **Combined loss function**

$$Loss = (1 - \lambda\_dssim) * L1(image, gt\_image) + \lambda\_dssim * (1 - SSIM(image, gt\_image))$$

Major Challenges Encountered:

➔ **Hardware Limitations:**
Initial Setback: Our initial attempts to implement the system using our existing computer hardware were unsuccessful. The rendering and training processes are computationally intensive, requiring significant GPU resources for efficient execution.
Specific Hardware Requirement: The project required Nvidia GPU capabilities for certain operations, particularly those involving CUDA optimizations. However, our systems were MacBooks, which are not

equipped with Nvidia GPUs. This mismatch led to significant delays and forced us to reconsider our approach.

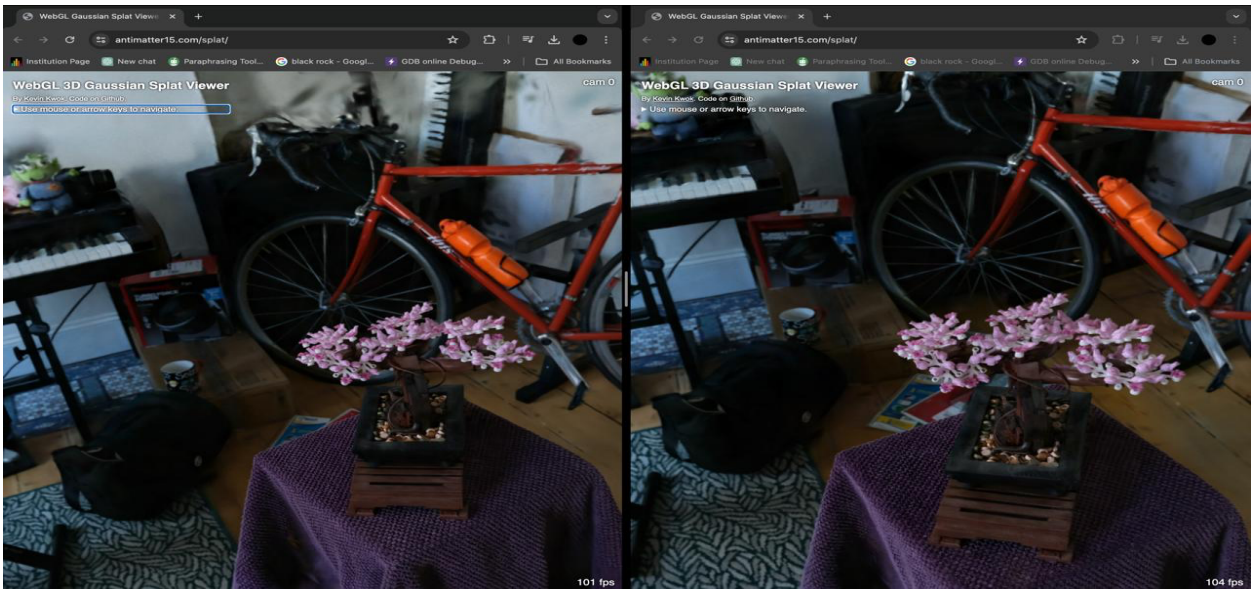
➔ **Adaptation to Google Colab:**
To circumvent the hardware limitations, we opted to use Google Colab. Google Colab provides a cloud-based environment with access to Nvidia GPUs, which allowed us to execute our training and rendering tasks without hardware constraints.
This transition involved setting up our project on Google Colab, ensuring that all dependencies and our codebase were compatible with the Colab environment, and adapting our development workflow to rely on remote execution.

➔ **File Handling and Rendering:**
.ply File Compatibility: We also encountered issues with handling and rendering .ply files, which are polygon file formats used for storing 3D data. Ensuring compatibility of these files with our rendering system required additional adjustments.
Online Rendering Tool: As an auxiliary solution, we explored the use of an online platform that could render .ply files. This allowed us to quickly visualize our 3D data without needing to fully implement rendering adjustments in our primary project setup.

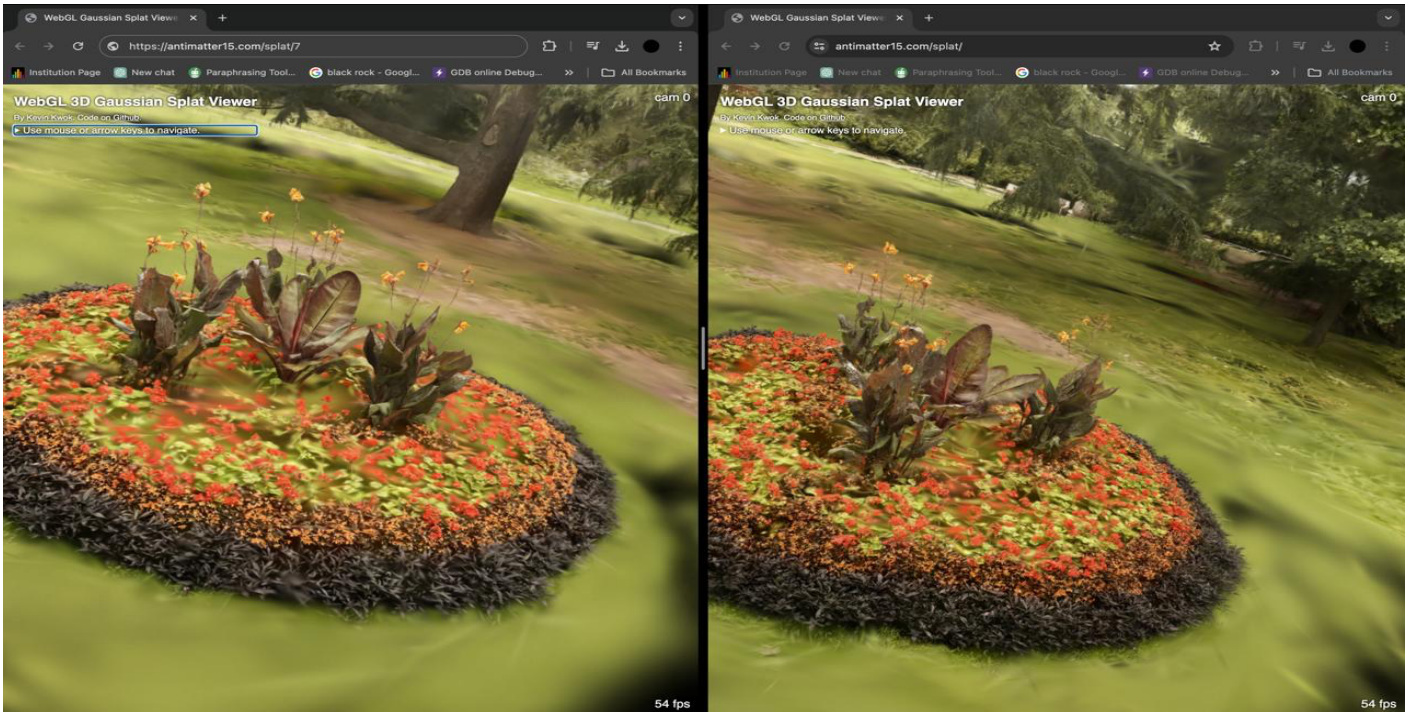
Reflections on Dead Ends:
The initial reliance on local hardware without checking compatibility with essential CUDA features was a significant oversight. It taught us the importance of thorough initial system requirement assessments.

Results: We ran the code for 4 different models and below are the results at 2 different checkpoint.

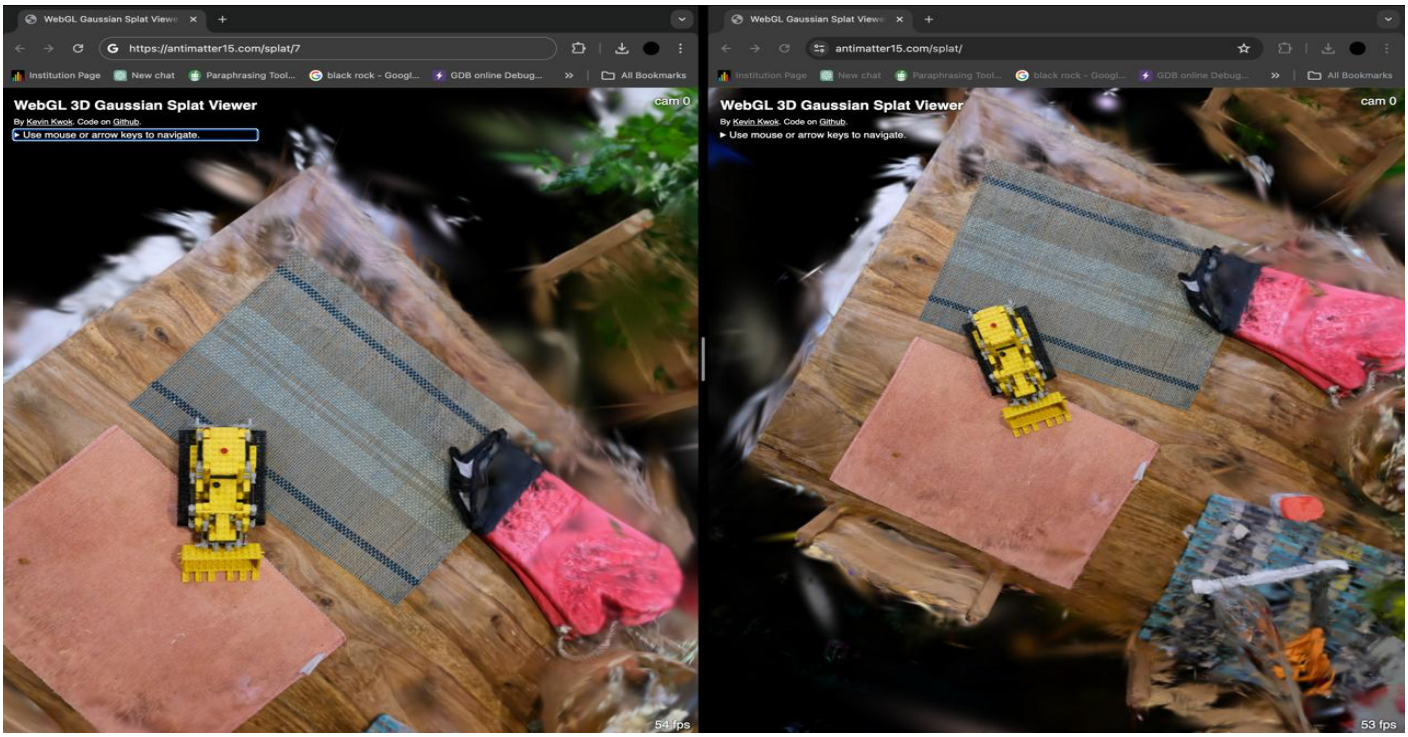
Bonsai with 7k iterations and 30k iterations:



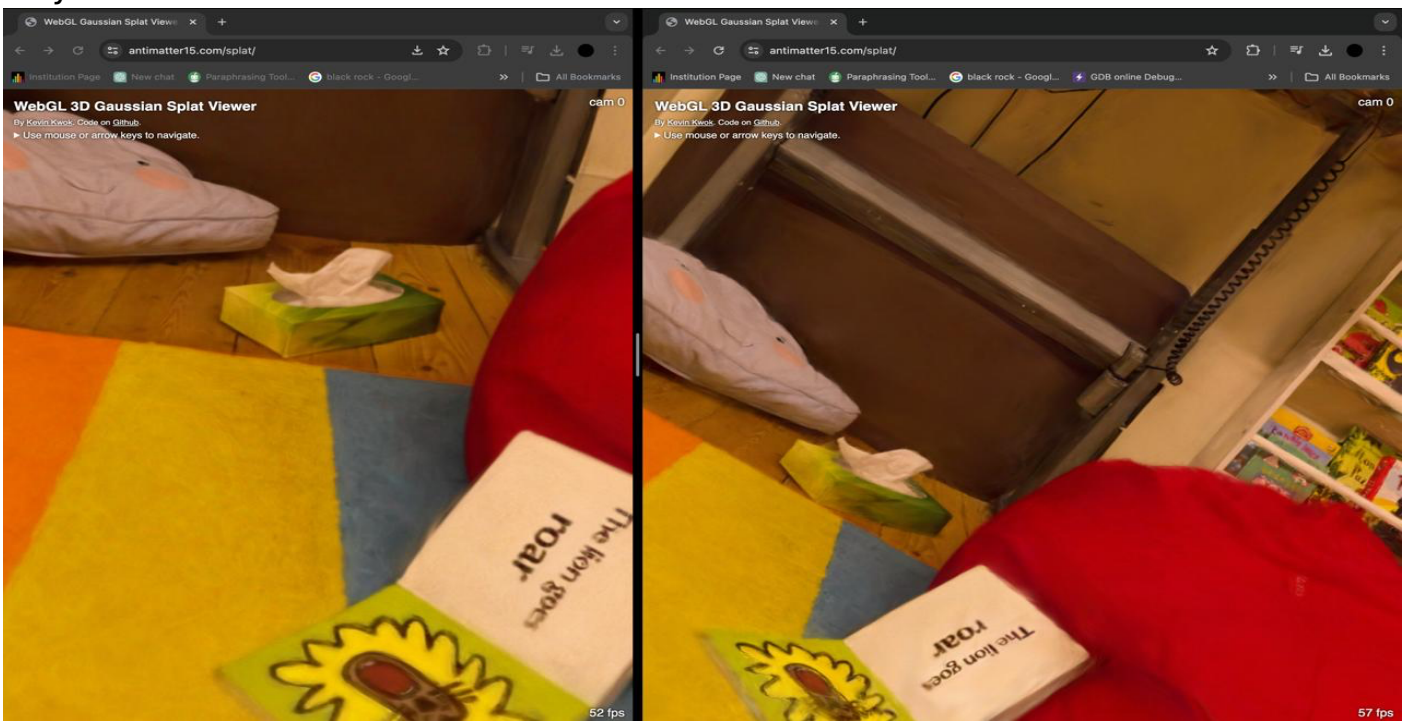
Bonsai with 7k iterations and 30k iterations:



Kitchen with 7k and 30k iterations:



Playroom with 30k and 80k iterations



Analysis of Work

In evaluating the success of our project to develop a machine learning-based rendering system using Gaussian models, we must consider the extent to which we met our original goals, the challenges we faced, and the solutions we implemented to address these challenges. Here's a detailed analysis based on these criteria:

Meeting Original Goals:

Original Goals:

- Develop a rendering system that can produce photorealistic images from 3D data.
- Utilize Gaussian models to capture complex scene details and lighting.
- Implement an iterative training loop that adjusts model parameters to minimize the discrepancy between rendered images and ground-truth images.
- Create a responsive system capable of incorporating real-time adjustments through a network GUI.

Outcome and Analysis:

➔ **Photorealistic Rendering:**

Success Level: Partially met.

We succeeded in developing a basic rendering system that used Gaussian models. However, the quality of photorealism was below our initial expectations due to the limitations in hardware and subsequent adjustments in computational resources (using Google Colab).

The use of cloud resources introduced variability in performance due to dependency on internet connectivity and available cloud GPU resources at different times.

➔ **Iterative Training and Model Adjustment:**

Success Level: Mostly met.

The iterative training mechanism was effectively implemented, allowing us to adjust Gaussian model parameters based on computed losses. This core functionality worked as expected and demonstrated potential for further refinement. Real-time feedback and adjustments were limited by our transition to Google Colab, as the initial plan to integrate with a local

network GUI had to be abandoned. This was a compromise to ensure that we had the necessary computational resources.

➔ **Integration with Network GUI:**
Success Level: Not met.
Our goal to integrate real-time GUI interactions was not achieved. The shift from local hardware to Google Colab meant that direct integration with a network GUI for real-time adjustments was not feasible within the scope of this project phase.

Reasons for Not Fully Meeting Goals:

➔ **Hardware Limitations:**
The lack of Nvidia GPUs in our original MacBook setups significantly hindered our ability to perform necessary computations locally. This was a major oversight in the planning phase, affecting our ability to meet all project goals, especially those involving high computational loads and real-time interactions.

➔ **Adaptation to New Tools:**
The need to adapt to Google Colab meant learning and configuring new workflows, which took time away from other development tasks. While it solved the computational issue, it introduced constraints on how interactively we could develop and test the system.

➔ **Resource Allocation:**
Dependency on external resources (cloud GPUs) introduced unpredictability in performance and availability, impacting our ability to consistently train and refine the model as planned.

Reflections on Project Execution:
The project was successful in demonstrating the feasibility of using Gaussian models for rendering 3D scenes. It also highlighted critical aspects of dependency on specific hardware for machine learning tasks, especially in graphics-intensive applications.
The adjustments made to overcome hardware limitations were necessary and beneficial for continuing the project, but they also highlighted areas for

improvement in planning and execution, particularly in hardware-resource assessment and contingency planning.

Conclusion: The project aimed to develop a machine learning-based rendering system using Gaussian models to produce photorealistic images from 3D data. The initiative showed significant promise in integrating advanced rendering techniques with Gaussian modeling to handle complex scene details and lighting effectively. While the project partially met its goals, several insights and lessons were gleaned from the challenges encountered during implementation.

Future work:

Enhanced Hardware Setup: Invest in a robust local hardware setup equipped with the necessary GPUs or establish a dedicated cloud environment tailored specifically for high-performance computing tasks related to rendering.

Advanced Model Development: Focus on advancing the capabilities of the Gaussian models to enhance the quality of photorealism. This can involve integrating more sophisticated machine learning techniques and exploring deeper neural network architectures.

Real-time Interaction Enhancement: Develop alternative solutions to facilitate real-time interaction and feedback even when operating in cloud-based environments. This might include developing more responsive cloud-based GUIs or utilizing APIs that allow better synchronization between cloud resources and local interfaces.

References:

https://github.com/graphdeco-inria/gaussian-splatting
https://medium.com/@AriaLeeNotAriel/numbynum-3d-gaussian-splatting-for-real-time-radiance-field-rendering-kerbl-et-al-60c0b25e5544
https://huggingface.co/camenduru/gaussian-splatting/tree/main
https://so.csdn.net/so/search?spm=1001.2101.3001.4498&q=3D%20Gaussian%20Splatting%20for%20Real-time%20Radiance%20Field%20Rendering&t=&u=