Image View Synthesis and Differentiable Rendering in Slang

David Choi, Hayden Kwok, Rick Rodness, Taiki Yoshino

Fall 2023

1 Research Context and Problem Statement

1.1 Research Context

The application of machine learning techniques to solve complex problems within the field of computer graphics is an emerging trend. One of the most important concepts taken from machine learning has been differentiation, which is the process of taking the derivative of a function. In machine learning, differentiation is how many machine learning models learn. By calculating the derivative of a model's loss function, we can adjust the model to reduce error and improve accuracy. This idea of utilizing differentiation for computer graphics rendering tasks is called differentiable rendering.

While there are many examples demonstrating the intersection of machine learning and computer graphics via differentiable rendering, many of them are limited by ill-suited developer tools and programming languages. Developers are often forced to choose between a system designed for machine learning or a system designed for computer graphics. Machine learning languages and frameworks contain many features that make them developer-friendly. For example, many have automatic differentiation, which differentiates a model without the need for manual derivation from a developer. These conveniences liberate programmers from the intricacies of mathematics, allowing them to concentrate on model architecture. However, these systems are not optimized for rendering, leading to poor training and render performance. Programming languages used in computer graphics, on the other hand, are more performant due to their faster rendering. However, they lack machine learning and general developer conveniences, making them difficult to use, learn, and maintain. In short, while specialized systems exist designed for either machine learning development or computer graphics, it is difficult to satisfy both simultaneously.

Slang

To reconcile this disparity, Yong He et al. created a new programming language called Slang [3][1]. Slang was designed with 2 primary goals: performance and developer productivity. Table 2 shows the comparison of Slang

against other systems. In terms of performance, Slang is optimized for rendering, allowing it to match the performance of most other computer graphics systems, such as CUDA. From a developer productivity perspective, Slang has auto-differentiation features that aim to enable programmers to implement differentiable rendering algorithms without needing to hand-derive functions. Additionally, Slang implements best practices like code modularity from popular general-purpose coding languages like C++, that enhance code maintainability and extensibility. All this makes Slang extremely promising for differentiable rendering applications.

	Slang	**CUDA**	**PyTorch and TensorFlow**
Optimized Rendering	✔	✔	✖
Auto Differentiation	✔	✖	✔
Code Modularity	✔	✖	✔

Table 1: Language and Framework Features

Image View Synthesis

One such differentiable rendering application is Image View Synthesis. Image view synthesis is the process of generating new images from a scene given a sparse set of input photos. For example, as illustrated in Figure 1, when provided with a set of input images of a stuffed animal, we can generate images of the stuffed animal from every angle around a 360-degree view, including perspectives that were not originally captured. Many image view synthesis techniques have been found by adopting machine learning techniques.

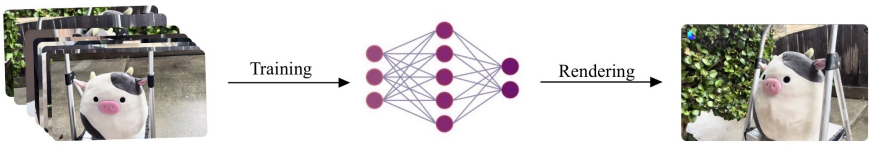

Figure 1: Image View Synthesis

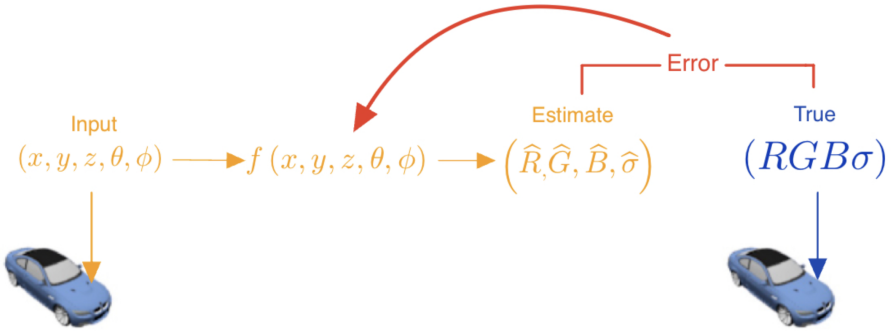

Figure 2: Overview of NeRF

One notable method is Neural Radiance Field (NeRF) [6]. A general overview of NeRF is shown in Figure 2. Given input photos of a scene the algorithm generates a neural network which can be represented as a function $f(x, y, z, \theta, \phi)$. The inputs of this function are coordinates of a point in the scene x, y, and z and viewing angles θ and ϕ. The function outputs the radiance, which can be thought of as the RGB color, and volumetric density, which can be thought of as transparency. This function is then trained on our input images, which requires us to differentiate the function $f(x, y, z, \theta, \phi)$. Once the scene is trained, we can now feed novel points and viewing angles to generate new images. Inspired by methods used in NeRF, various view synthesis algorithms have been created. The two we will examine are Plenoxels and Gaussian Splatting.

Plenoxles [9] differs from NeRF as it uses a special voxel grid, the 3D equivalent of a pixel, to represent a scene rather than a neural network. Each voxel contains color and volumetric density data. These properties are defined by the input images provided and refined through differentiation, similar to how the neural network in NeRF is refined. Once training is complete, we can render the scene in the same way as NeRF.

Another approach to image view synthesis is 3D Gaussian Splatting [4]. Simply put, a 3D Gaussian is a colored oval that becomes more transparent toward its edges. If you're familiar, think of a Gaussian blur. The scene can be represented by these 3D Gaussians. The shape and properties of these Gaussians are refined through differentiation based on the input images. These Gaussians can then be used to generate new images.

Each of these view synthesis algorithms has implementations in different languages and frameworks. NeRF is implemented with PyTorch and TensorFlow, Plenoxels with PyTorch, and Gaussian Splatting with CUDA. Due to the limitations of these systems, these implementations have compromises leading to stifled performance or increased code complexity.

1.2 The Problem

Slang has the potential to accelerate the research of differentiable rendering algorithms, similar to how PyTorch did for general machine learning [8], but it is a very new language. While the papers on Slang [1][3] showcase the language's potential for increasing performance and developer productivity in computer graphics applications, there is a lack of peer reviewed work to validate those results. As a result there is not much credibility, outside the original papers [1][3], that would lead to widespread adoption. The goal of our research is to see if we can verify the results of the original Slang papers, we will utilize the problem of image view synthesis to evaluate Slang's capabilities in terms of performance and developer productivity, compared with other programming languages.

2 Proposed Solution

The purpose of our research is to test how well Slang meets its goals of performance and developer productivity [3]. We are particularly interested in its utility in differentiable rendering, as Slang is one of the few languages with efficient rendering and auto differentiation. If our findings affirm the conclusions of the Slang papers [3][1], our research would give credibility to the language's advantages and adoption. If our findings contradict the conclusions of the Slang papers [3][1], we hope to provide insight into why and how the language could improve in the future. As a result, we propose three tests that we will use to evaluate Slang's training performance, render performance, and support for developer productivity.

To test the effectiveness of Slang, we plan to use the problem of image view synthesis. Image view synthesis has a few properties that make it appealing for our experiment.

- Image view synthesis has many well-established solutions that rely on differentiable rendering.
- There are image view synthesis algorithms implemented in the programming languages and frameworks we want to compare to Slang: PyTorch, TensorFlow, and CUDA.

Our plan is to implement the image view synthesis algorithms NeRF, Plenoxels, and Gaussian Splatting in Slang. We will then compare our Slang implementations with the pre-existing implementations of these algorithms, which are available publicly. NeRF was originally implemented in PyTorch [10] and TensorFlow [7], Plenoxels in PyTorch [2], and Gaussian Splatting in CUDA [5]. Through this comparison, we will discover whether Slang delivers on its performance promises.

	Slang	PyTorch	TensorFlow	CUDA
NeRF	Implement	Original	Original	N/A
Plenoxels	Implement	Original	N/A	N/A
Guassian Splatting	Implement	Original	N/A	N/A

Table 2: Image View Synthesis Algorithm Implementations

2.1 Training Performance

When measuring training performance, we want to measure how close the image appears to the ground truth and how long it takes to train a model on a scene.

We can compare the "efficiency" of each language by comparing sample images rendered by a model at fixed intervals of training time and observing the qualitative differences between them. The qualitative differences will be compared manually and through visual performance metrics PSNR, SSIM, and LPIPS used in previous research [6] [9] [4]. While the intervals of time we train for will depend on which view synthesis algorithm we are using, they will not differ between languages. That way we can compare the output images and see visually which languages lead to faster convergence. If Slang truly delivers on its performance promises, we expect the output images to outperform PyTorch and TensorFlow and match the performance of CUDA.

To measure how long the different language implementations will take to train, we will train each model for a fixed number of epochs and observe how quickly we are able to finish training on a scene. Since we are implementing the same algorithm and training on the same data, any significant discrepancy in total training time can be attributed to the programming language. We should expect that Slang enables the view synthesis models to be trained faster than PyTorch and TensorFlow and equally as fast as CUDA.

2.2 Render Performance

Measuring render performance is much simpler than training performance. After fully training each model, we will measure how long each implementation of the algorithm takes to fully render. Then, we can compare the outputs visually, both manually and through PSNR, SSIM, and LPIPS. We will see which languages perform better by seeing which language leads to an accurate output in the least time.

Our expectations are that Slang should outperform PyTorch and TensorFlow, and match CUDA. We expect that all languages will perform the same visually. Since the algorithm does not change between the different language implementations, neither should the output. Any visual discrepancy between languages would signal a significant issue in render or training accuracy, either due to implementation or the language. Where we expect the languages to differ

is in the time taken to render. Slang should take the same amount of time to render as CUDA, as both platforms are optimized for rendering. Slang should also outperform PyTorch and TensorFlow as those languages are not optimized for rendering, bottlenecked by their RAM usage.

If our Slang implementations meet the performance expectations when compared to Pytorch, Tensorflow, and CUDA we can confidently support the original Slang papers' claims in performance [3][1]. If Slang fails to meet these expectations, we can point to specific areas of improvement for the developers to improve training and render performance in the future.

2.3 Developer Productivity

Evaluating how effectively a programming language can improve developer productivity is a challenging task, as it involves the subjective experiences and perceptions of the users. We can get an idea through lines of code, but that's a very limited perspective. Traditional methods (citation of PyTorch?) often measure productivity based on the language's popularity and usability experiments. However, due to the limited resources and niche nature of Slang's audience, it is hard to get a large enough population to conduct these measurements. So we propose a survey of 10 - 15 Computer Science Graduate students with Computer Graphics experience at the University of California, San Diego. Our survey will consist of the following questions and more:

- What is your experience with (PyTorch/TensorFlow/CUDA/Slang): None, Beginner Intermediate or Advanced?
- Consider the following situation. You are working on a personal differentiable rendering project. You will be the only person to edit this code in the future. Which of the following languages would you use?
- Consider the following situation. You are working on a differentiable rendering project for a large company. You know many people will have to read, maintain, and extend your code in the future. Which of the following languages would you use?
- On a scale of 1 (very difficult) to 5(very easy) how learnable are the following languages (PyTorch/TensorFlow/CUDA/Slang)
- On a scale of 1 (very difficult) to 5(very easy) what is your experience extending pre-existing code in the following languages: PyTorch or TensorFlow, CUDA, Slang
- On a scale of 1 (very unreadable) to 5(very readable) how readable is this snippet of code

These questions will give insight into how Slang compares to other languages used in differentiable rendering applications. The expected outcome is that Slang will be preferred to CUDA but fall behind PyTorch and TensorFlow.

In summary, we propose three methods to test the claims of the original Slang papers [3][1]. Ultimately by either affirming or rejecting these claims, we aim to provide insight into the language's viability in differentiable rendering applications.

3 Evaluation Plan

Our evaluation plan is designed to confirm the validity of our experimental results. By showing that our tests are reliable we can affirm our conclusions about the effectiveness of Slang in differentiable rendering applications. We will be conducting our evaluations based on the two aspects of our proposed solution: performance tests and developer productivity.

3.1 Performance

We will implement a series of rigorous statistical tests to validate our findings for render performance and training performance. Our goal is to ensure that the differences in performance metrics observed within our implementations are statistically significant and not a byproduct of random variation.

The first is a T-Test for independent samples, which would compare the means of the training times and rendering speed between Slang and each competing framework. We hope to find whether or not there is statistically significant evidence for Slang's performance being better across the majority of categories regarding differentiable rendering. In the event the conditions for normality are not met, where our collected data is not retrieved from a random sample that demonstrates independence or includes outliers, we will employ a Non-Parametric U Test. We will use this Non-Parametric U Test, also known as the Wilcoxon Rank-Sum Test, to compare the datasets as an alternative to our previously disclosed method.

A potential shortcoming to relying solely on the T-Test for independent samples is that it is limited to comparing the metrics of only two samples. As a result, we will also employ ANOVA (analysis of variance) amongst all the group means that implement the same rendering algorithm to highlight the statistically significant differences between all of the programming systems. In addition, we will calculate Cohen's d to visualize the size of the differences to provide a more comprehensive understanding of their significance.

Lastly, we will employ a 95% confidence interval to our calculated sample means to provide a range within which we can be confident the true difference of means amongst all datasets lies. This will allow us to quantify the uncertainty around our sample means, allowing us to make more compelling inferences and conclusions.

After the completion of these tests, we will graph and log our findings with data visualization tools in Python or R-studio for a better visual understanding of our findings. Some examples of which would include error mapping, comparison graphs, as well as a breakdown of their implications. Overall, we

employ these methodologies across multiple dimensions in order to bolster the credibility of our findings.

3.2 Developer Productivity

In order to better understand how well Slang enables developer productivity, our proposed survey, alongside a simple comparison of lines of code, aims to capture the sentiment regarding Slang's ease of use. More specifically we want to look at developers' attitudes towards Slang in terms of its utility in differentiable rendering, learnability, and ease of code maintenance. We acknowledge our sample size of 10 to 15 Computer Science Graduates is much too small for any conclusive evidence. However, we still aim to note any interesting patterns observed and provide a starting point for discussion. Furthermore, we will provide an outline of tests which may be used in the future on a larger sample to draw more conclusive results.

To analyze our results from the proposed survey, we will be conducting hypothesis tests in order to infer our alternate hypothesis: the majority of UCSD Graduate Students studying Computer Graphics have a positive view of Slang in each question category. Otherwise, we will fail to reject our null hypothesis, which would suggest that there is no significant preference for Slang among this population. However, there is a possibility that normality will not be met and thus verify the limitations of this survey.

Through our evaluation plan we hope to affirm the results of our performance tests and survey. Ultimately, our goal is to test Slang's potential and practicality in the realm of differentiable rendering. The desired outcomes of our project are to provide feedback on Slang and provide additional context to the computer graphics community about its viability as a tool for both academic research and industry applications.

3.3 Timeline

Winter Quarter 24:

Week 1-2

- Reread and analyze NeRF paper and algorithm.
- Begin the implementation of NeRF in Slang.
- Install and run pre-exisiting implementations of NeRF in PyTorch and TensorFlow.

Week 3-4

- Collect relevant performance metrics from pre-exisiting implementations of NeRF.
- Refine NeRF in Slang, troubleshoot issues, and start preliminary testing.

Week 5-8

- Finalize our implementation of NeRF in Slang.
- Collect relevant performance metrics from our implementation of NeRF in Slang.
- Perform statistical analysis on all performance metrics.

Week 9-10

- Draft and distribute the survey to collect data regarding participants' firsthand experiences and their opinions on the overall user-friendliness of PyTorch, TensorFlow, CUDA, and Slang.
- Additionally, gather insights on their learning experiences and the perceived difficulty levels associated with using these technologies.

Spring Quarter 24:
Week 1-2

- Reread and analyze Plenoxels and Gaussian Splatting papers and algorithms.
- Begin the implementation of Plenoxels and Gaussian Splatting in Slang.
- Install and run pre-exisiting implementations of Plenoxels and Gaussian Splatting in Pytorch and CUDA.
- Analyze survey results and perform hypothesis tests if possible.

Week 3-4 - Data Analysis and Report Drafting

- Collect relevant performance metrics from pre-exisiting implementations of Plenoxels and Gaussian Splatting.
- Refine Plenoxels and Gaussian Splatting in Slang, troubleshoot issues, and start preliminary testing.

Week 5-8

- Finalize our implementation of Plenoxels and Guassian Splatting in Slang.
- Collect relevant performance metrics from our implementations of Plenoxels and Gaussian Splatting in Slang.
- Perform statistical analysis on all performance metrics.

Finalization and Poster Presentation Prep
Week 9

- Finalize results and draw conclusions into a crafted summary.
- Utilize presentation skills and structure demonstrated during research lab group meetings to outline poster presentations.

- Prepare for poster presentation, rehearse, refine data collection, visuals, etc.

Week 10

- Present research poster and submit accompanying work for potential future use cases.

References

[1] Sai Bangaru, Lifan Wu, Tzu-Mao Li, Jacob Munkberg, Gilbert Bernstein, Jonathan Ragan-Kelley, Fredo Durand, Aaron Lefohn, and Yong He. Slang.d: Fast, modular and differentiable shader programming. In *ACM Transactions on Graphics (SIGGRAPH Asia)*, volume 42, pages 1–28, 2023.

[2] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Github repository: svox2. https://github.com/sxyu/svox2, 2022. Accessed: December 8, 2023.

[3] Kayvon Fatahalian He, Yong and Tim Foley. Slang: language mechanisms for extensible real-time shading systems. In *ACM Transactions on Graphics (TOG) 37.4 (2018): 1-13*, 2018.

[4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.

[5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. Github repository: gaussian-splatting. https://github.com/graphdeco-inria/gaussian-splatting, 2023. Accessed: December 8, 2023.

[6] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[7] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Github repository: nerf. https://github.com/bmild/nerf, 2021. Accessed: December 8, 2023.

[8] Adam et al. Paszke. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32*, page in NeurIPS, 2019.

[9] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022.

[10] Lin Yen-Chen. Github repository: Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020. Accessed: December 8, 2023.