УДК 378:004
DOI: 10.25686/978-5-8158-2474-4-2025-23-29

Сравнительный анализ выполнения нейронных моделей для графики

Н. А. Милин[1,2], Р. О. Родионов[1,2], В. А. Галактионов[2], В. А. Фролов[1,2]
[1]Институт перспективных исследований проблем искусственного интеллекта и интеллектуальных систем МГУ имени М.В. Ломоносова, Москва, Россия
[2]Институт прикладной математики им. М.В. Келдыша РАН, Москва, Россия

Аннотация. В данной работе представлено исследование производительности различных подходов к реализации выполнения нейронных моделей в задачах компьютерной графики, связанных с представлением поверхностей и материалов трёхмерных объектов. Рассматриваются четыре различных метода, среди них как универсальные, поддерживаемые на большинстве устройств, так и использующие специальные аппаратные возможности. Экспериментальное исследование проведено на двух задачах: задание поверхности в виде нейронной функции дистанции и задание двухлучевой функции отражательной способности в виде нейронной модели. Результаты демонстрируют существенные различия в производительности между методами. Выводы, сделанные в рамках исследования, позволяют нам представить практические рекомендации для разработчиков, стремящихся интегрировать нейронные методы в приложения компьютерной графики.

***Ключевые слова*:** компьютерная графика, рендеринг реального времени, нейронные сети, функции дистанции со знаком.

A comparative study of the performance of neural models for computer graphics

N. A. Milin[1,2], R. O. Rodionov[1,2], V. A. Galaktionov[2], V. A. Frolov[1,2]
[1]IAI Moscow State University, Moscow, Russia
[2]Keldysh Institute of Applied Mathematics, Moscow, Russia

Abstract. This paper presents a performance study of different approaches to the implementation of neural models in computer graphics applications related to the representation of surfaces and materials of three-dimensional objects. Four different methods are considered, among them both universal, supported on most devices, and those that use specialized hardware capabilities. Experimental studies are performed on two tasks: representing a surface as a neural distance function and representing a bidirectional reflectance distribution function as a neural model. The results show significant differences in performance between the methods. The findings of the study allow us to provide practical recommendations for developers seeking to integrate neural methods into computer graphics applications.

***Keywords*:** computer graphics, real-time rendering, neural networks, signed distance functions.

Введение

Современная компьютерная графика переживает значительные изменения, связанные с внедрением технологий искусственного интеллекта (ИИ) в существующие алгоритмы рендеринга. Нейронные подходы к представлению геометрии и материалов открывают новые возможности для создания высококачественных визуальных эффектов, такие как моделирование многослойных материалов [9] и геометрии [11]. Кроме того, нейронные модели используются в дифференцируемом рендеринге [12, 13] и могут являться составной частью инструментов генеративного ИИ [15].

Однако интеграция нейронных методов в графические приложения реального времени сталкивается с существенными вызовами производительности [9]. Выполнение нейронных сетей на GPU требует тщательной оптимизации алгоритмов и эффективного использования архитектурных особенностей современных графических процессоров. Мотивацией данной работы является нахождение оптимальных методов выполнения нейронных моделей на графических процессорах.

Обзор существующих технологий

В современных задачах компьютерной графики применяются разнообразные архитектуры нейронных сетей, каждая из которых адаптирована под специфические требования графических вычислений. Многослойные перцептроны (MLP) широко используются в задачах трассировки лучей и расчета освещения благодаря своей способности аппроксимировать сложные функции излучения и отражения [1]. Сверточные нейронные сети (CNN) находят применение в задачах шумоподавления и повышения разрешения изображений, где пространственная корреляция данных играет ключевую

роль [2]. Рекуррентные архитектуры, включая LSTM и GRU, применяются для временной стабилизации в алгоритмах глобального освещения [3]. Архитектуры на основе трансформеров адаптируются для задач синтеза изображений, где важна способность к обработке последовательностей различной длины [4, 7].

Значительная часть исследовательских работ в области компьютерной графики опирается на библиотеки глубокого обучения общего назначения, такие как PyTorch, TensorFlow. Это обусловлено простотой их использования и обширным набором инструментов разработки. Работы [3, 6, 7, 11] используют PyTorch, однако его применение контексте графики реального времени сопряжено с существенными накладными расходами, связанными с интерпретацией Python-кода и невозможностью низкоуровневой ручной оптимизации. Решения, основанные на технологии CUDA, используемые в работах [1, 5, 8], отличаются высокой степенью оптимизации за счет адаптации к аппаратным возможностям устройств NVIDIA, однако существенным недостатком является привязка к конкретной экосистеме, что ограничивает портируемость решения на другие платформы и создает зависимость от проприетарных технологий.

Отдельно стоит отметить, что традиционные библиотеки глубокого обучения оптимизированы для согласованного выполнения нейронной сети, где входные данные собраны в один набор (батч) и высокая пропускная способность достигается за счет параллельной обработки большого количества образцов. Однако в графическом конвейере может потребоваться выполнить нейронную модель только для части пикселов или выполнить несколько различных моделей с разными данными на входе. Как отмечено в работе [9], инструменты для интеграции нейронных сетей с потенциально расходящимся исполнением в языки программирования шейдеров, такие как GLSL или HLSL, практически отсутствуют. Помимо этого, для повышения производительности предпочтительно использовать аппаратное ускорение матричного умножения, доступное в устройствах NVIDIA, AMD и Intel. Далее обсудим как решать эти проблемы и сравним несколько перспективных методов выполнения нейронных сетей в задачах компьютерной графики.

Описание исследуемых методов

Метод 1: Компилируемые веса

Очевидный способ – сохранить все веса нейронной сети в самом шейдере. Так они будут доступны компилятору, что позволит поместить их в кэш и провести дополнительные оптимизации, такие как разворачивание циклов и удаление избыточных инструкций. Данный метод требует создания отдельного шейдера для каждой конкретной нейронной сети, что усложняет его использование, если требуется выполнить произвольное число моделей в течение рендеринга одной сцены.

Метод 2: Веса в буфере

При данном подходе параметры нейронной сети загружаются динамически в глобальную память GPU и доступ к ним осуществляется через индексирование внутри шейдера. Это позволяет использовать код одного шейдера для моделей различных размеров, передавая данные об архитектуре сети как константы. Существенным недостатком является то, что каждый отдельный поток многократно загружает параметры модели из видеопамяти. Улучшением могло бы служить использование быстрой разделяемой памяти путем предварительной загрузки небольшого числа параметров в нее и выполнение блочного матричного умножения, но данная возможность доступна только в вычислительных (compute) шейдерах.

Метод 3: cuDNN

Чтобы ускорить стандартные для нейронных сетей операции, можно воспользоваться библиотекой cuDNN [10], которая помимо эффективного выполнения матричного умножения позволяет совмещать несколько последовательных операций в один граф и тем самым оптимизировать обращения к памяти. Для этого потребуется создать входные данные в отдельном ядре, сохранить их в глобальную память GPU, затем вызвать функцию исполнения построенного cuDNN графа операций и обработать полученный выход в следующем ядре. Таким образом можно сократить время прямого прохода сети, но потребуется использовать дополнительную память и вызывать несколько ядер, что увеличивает общие накладные расходы. Недостатком является то, что данный метод доступен только на устройствах NVIDIA и плохо интегрируется в графический конвейер.

Метод 4: Аппаратное ускорение с использованием Cooperative Vectors

С появлением этого расширения для современных графических API[1][2][3] стала возможной реализация прямого прохода нейросети, совместимая с существующей моделью программирования графических шейдеров. Его цель – предоставить высокоуровневый инструмент для описания матричного умножения в коде шейдера, в котором компилятор отвечает за объединение входных векторов с разных потоков в матрицу для эффективного перемножения с использованием тензорных ядер. Расширение не требует, чтобы данные в разных потоках были одинаковые, что позволяет обрабатывать несколько нейронных моделей в одном шейдере. Повышение производительности ожидается за счет неявной загрузки параметров модели по необходимости и выполнения умножения матриц и активаций слоев с аппаратным ускорением. Данный метод требует поддержки расширения в графическом драйвере.

Эти методы выбраны для исследования по следующим причинам. Методы 1, 2 могут быть реализованы при помощи стандартных операций графических API, поэтому применимы на большинстве платформ, включая мобильные. При этом первый теоретически способен показать лучшую производительность, если компилятору удастся расположить параметры модели в более быстрой памяти, а второй не зависит от самих параметров, предоставляет возможность их динамической загрузки. Третий использует алгоритмы умножения матриц с наилучшей оптимизацией для устройств NVIDIA, четвертый легко встраивается в существующий конвейер рендеринга простой заменой аналитических вычислений на оценку нейронной модели и предоставляет возможность выполнений нейронных сетей во всех типах шейдеров, в том числе в конвейере трассировки лучей. В методах 3, 4 также поддерживается объединение нескольких последовательных операций, таких как умножение матрицы входных данных на матрицу весов, прибавление матрицы сдвига, применение функции активации, что позволяет избежать промежуточных преобразований данных для их непрерывной обработки на тензорных ядрах.

Описание экспериментов

В ходе экспериментов была проверена работа каждого из методов для разных размеров нейронных моделей. Методы 1, 2 и 4 были реализованы с использованием Vulkan API, в методе 4 дополнительно включено расширение VK_NV_cooperative_vector. Метод 3 был реализован на платформе CUDA по причине хорошей совместимости с cuDNN. В качестве языка для написания графических шейдеров использовался язык Slang с открытым исходным кодом [16]. Для измерения производительности в методах 1, 2 и 4 применялись временные метки Vulkan, в методе 3 – системный таймер Windows 10. Все эксперименты проводились на видеокарте NVIDIA RTX 3080 10GB с версией драйвера 572.42 в разрешении 1920x1080, параметры моделей были представлены в числах с плавающей точкой половинной точности (float16). Для оценки и сравнения методов выполнения нейронных моделей были выбраны следующие задачи:

1. Рендеринг нейронных SDF.

Функция расстояния со знаком в трёхмерном метрическом пространстве – отображение $R^3 \to R$, показывающее расстояние от данной точки до границы некоторого подмножества пространства. Знак функции определяет, находится ли точка внутри множества. Такие функции являются универсальным способом представления геометрии, способным приближать поверхности, заданные в ином виде (мэш из треугольников, воксели и другие). Традиционно SDF задаются аналитически при помощи технологии конструктивной блочной геометрии, где поверхность строится из примитивов и операций над ними. Хотя данным способом возможно задать довольно сложные поверхности, его трудно использовать для моделирования произвольных объектов. В то же время нейронные модели способны аппроксимировать сложный сигнал, такой как функция расстояния, и могут являться непрерывным и дифференцируемым представлением [11], что позволяет использовать их для задач обратного рендеринга [12, 13].

[1] https://developer.nvidia.com/blog/neural-rendering-in-nvidia-optix-using-cooperative-vectors
[2] https://registry.khronos.org/vulkan/specs/latest/man/html/VK_NV_cooperative_vector.html
[3] https://devblogs.microsoft.com/directx/cooperative-vector

В сравнении для рендеринга SDF использовался стандартный итерационный алгоритм Sphere Tracing [17] с максимальным числом шагов, равным 100. Нейронные SDF были обучены методом, предложенным в работе [11] на модели Armadillo [14].

2. Моделирование материалов с помощью нейронных BRDF.

Функции двунаправленного отражения света (Bidirectional Reflectance Distribution Function, BRDF) являются одними из главных компонентов физически корректного рендеринга. Нейронные BRDF позволяют аппроксимировать сложные и анизотропные материалы, которые трудно описать аналитическими моделями [9], и потенциально обеспечивают более точное воспроизведение реальных материалов на основе измерений [18]. В эксперименте использовались нейронные BRDF с числом параметров от 275 до 9179 и функцией активации RELU. Также проведено сравнение времени их выполнения с классической аналитической моделью Кука-Торренса.

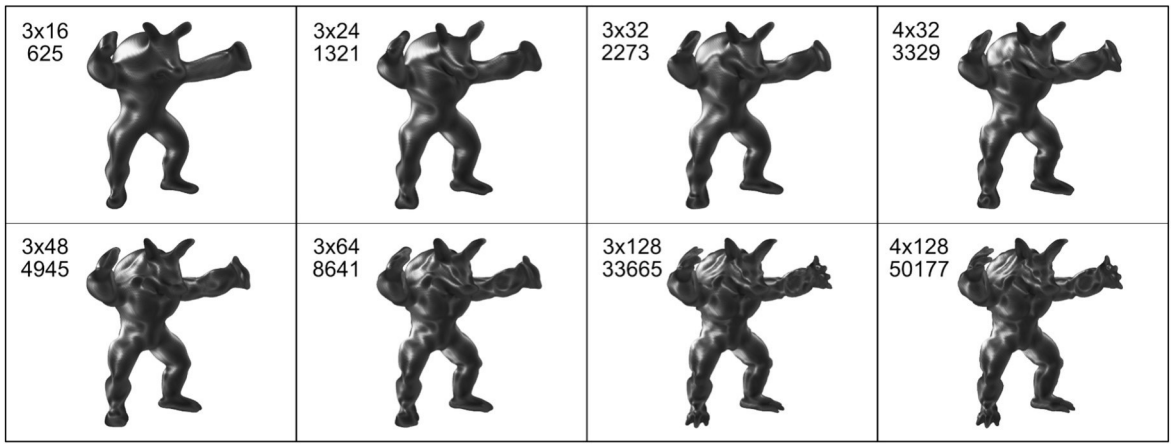

Рисунок 1. Нейронная SDF, построенная по модели Armadillo при разных размерах сети (в обозначении: количество скрытых слоев, размер скрытого слоя, на следующей строке – общее число параметров)

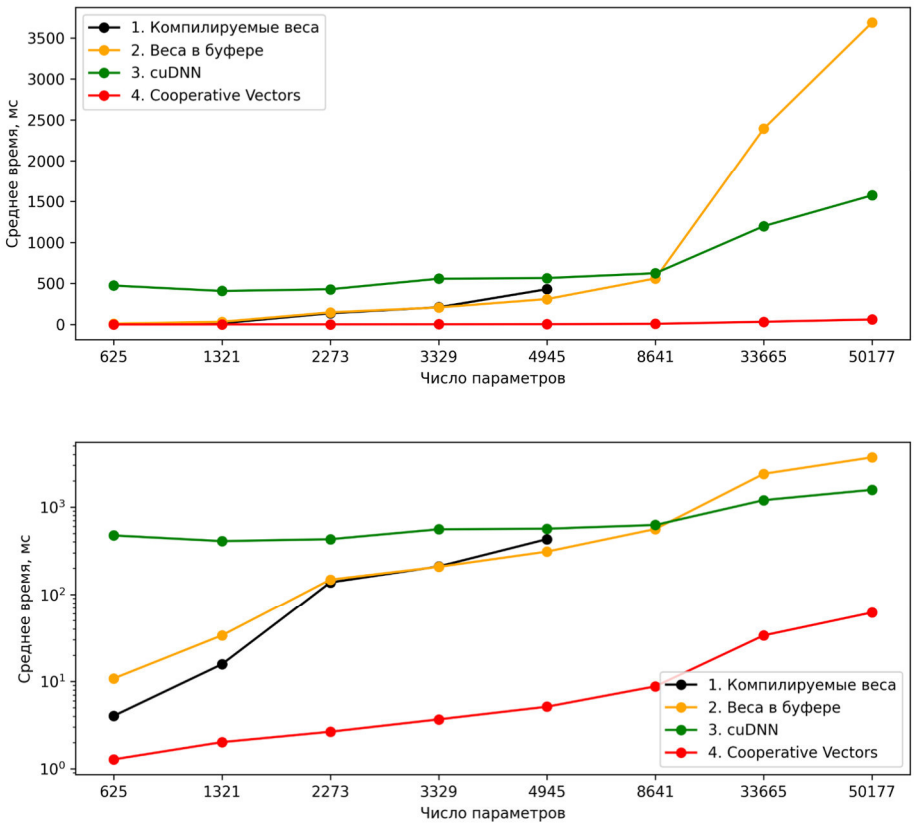

Рисунок 2. Время выполнения нейронной SDF при разном числе параметров модели (в линейной и логарифмической шкалах)

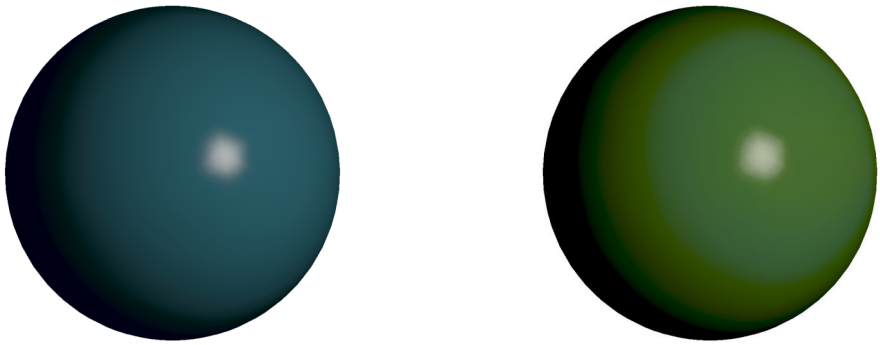

Рисунок 3. Изображения сферы, полученные с использованием нейронных BRDF

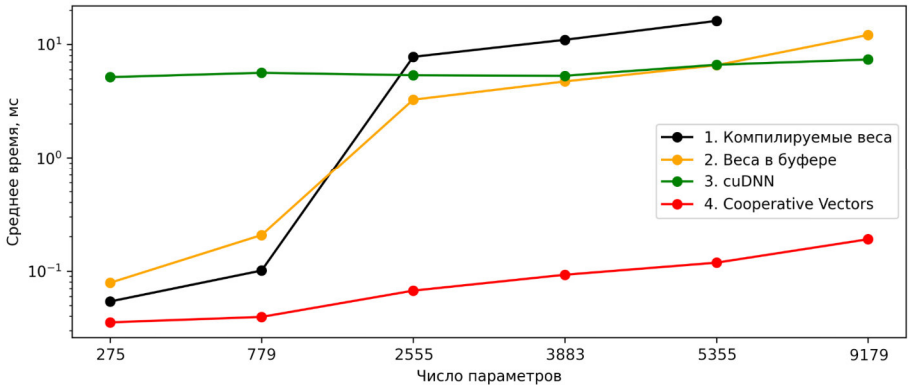

Рисунок 4. Время выполнения нейронной BRDF при разном числе параметров модели (в логарифмической шкале)

Таблица 1. **Время выполнения нейронной SDF**

Метод реализации	Количество параметров модели							
	3x16 625	3x24 1321	3x32 2273	4x32 3329	3x48 4945	3x64 8641	3x128 33665	4x128 50177
	Среднее время, мс							
1. Компилируемые веса	4.018	15.748	137.843	209.946	430.141	E	E	E
2. Веса в буфере	10.797	33.752	148.149	207.818	309.995	561.047	2393.544	3690.585
3. cuDNN	474.025	407.932	429.275	557.170	565.323	623.787	1197.341	1572.553
4. Cooperative Vectors	1d.277	2.014	2.650	3.673	5.131	8.787	33.703	61.728

Таблица 2. **Время выполнения нейронной BRDF**

Метод реализации	Количество параметров модели					
	275	779	2555	3883	5355	9179
	Среднее время, мс					
1. Компилируемые веса	0.054	0.101	7.780	10.984	16.111	E
2. Веса в буфере	0.079	0.207	3.263	4.727	6.560	12.119
3. cuDNN	5.157	5.629	5.357	5.296	6.630	7.371
4. Cooperative Vectors	0.035	0.040	0.067	0.093	0.118	0.190
Классическая Cook-Torrance	0.027					

Анализ результатов

В таблицах 1-2 приведены результаты замеров времени выполнения нейронных SDF и BRDF методами 1-4 при разном числе параметров модели. Эксперименты показали, что различия во времени выполнения для одной и той же модели могут достигать трех порядков. Так, метод с компилируемыми весами демонстрирует относительно хорошую производительность при малом числе параметров (примерно до 2000), однако затем время выполнения резко возрастает, после чего возникают ошибки создания конвейера в драйвере (обозначены как E в таблицах) и его применение становится невозможным при дальнейшем увеличении размера модели. Метод 2 показывает линейный рост времени выполнения при увеличении числа параметров. Для малых моделей он уступает в производительности, но сохраняет применимость при любом размере модели, хотя и с существенными расходами на доступ к памяти. Метод 3, основанный на библиотеке cuDNN, несмотря на ожидаемую оптимизацию, демонстрирует высокие и практически постоянные накладные расходы, связанные с переключением между ядрами и синхронизацией, что делает его малоэффективным для моделей с небольшим числом параметров, особенно в задаче рендеринга SDF, где приходится многократно запускать разные ядра. При моделировании BRDF, однако, он обходит методы 1-2 уже при меньших размерах нейронной сети. Метод 4 (Cooperative Vectors) показывает лучшую производительность и стабильно работает при всех рассмотренных размерах моделей.

В целом для задач рендеринга SDF в реальном времени подходят только модели небольшого размера, так как в традиционно используемом алгоритме Sphere Tracing требуется многократная оценка функции дистанции. Из рассмотренных методов для этой задачи наиболее применимы метод 4 и методы 1, 2 но только для наиболее малых моделей. Будущие исследования могут включать разработку специальной архитектуры нейронных SDF, для которых нахождение хотя бы приближенного решения $Sdf(x) = 0$ будет возможно без выполнения модели десятки раз.

Задачи моделирования BRDF менее требовательны к вычислениям, и все методы показывают приемлемое время выполнения, а для небольших моделей метод 4 достигает производительности, сопоставимой с классической реализацией аналитической модели Кука-Торренса, что делает возможным прямую замену аналитических моделей нейронными.

Таким образом, можно сделать следующие обобщения. Методом 1 можно выполнять только небольшие модели (до примерно 2000 параметров на видеокарте NVIDIA RTX 3080), но он позволяет получить высокую производительность и не ограничен конкретной платформой. Метод 2 подходит, если нужно переносимое решение, работающее с любым числом параметров, и задача не требует большого числа оценок модели. Метод 3 не рекомендуется для графических приложений реального времени из-за высоких накладных расходов и сложной интеграции с графическими API, несмотря на его эффективность в традиционных задачах глубокого обучения. Метод 4 предпочтительнее, если целевым оборудованием являются современные графические ускорители, имеющие поддержку Cooperative Vectors.

Заключение

В рамках исследования был проведен сравнительный анализ различных методов выполнения нейронных моделей в задачах рендеринга геометрии и материалов в контексте графики реального времени. Эксперименты включали реализацию и измерение производительности различных подходов к выполнению нейронных SDF и BRDF. Полученные данные позволяют сделать ряд обоснованных выводов о применимости, эффективности и ограничениях каждого метода. Результаты подтверждают, что эффективное выполнение нейронных моделей в задачах графики реального времени возможно, однако требует тщательного выбора стратегии реализации с учетом специфики задачи и целевой платформы. Предметом будущих исследований может служить разработка новых методов, а также развитие предложенных в данной работе.

Источник финансирования

Исследование выполнено за счет гранта Российского научного фонда № 25-11-00054, https://rscf.ru/project/25-11-00054/

Список литературы

1. Müller, T., et al. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding.” ACM Transactions on Graphics, 2022.

2. Chaitanya, C. R. A., et al. “Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder.” ACM Transactions on Graphics, 2017.

3. Munkberg, J., et al. “Neural Denoising with Layer Embeddings.” Computer Graphics Forum, 2020.

4. Kulhánek, J., et al. “ViewFormer: NeRF-free Neural Rendering from Few Images Using Transformers” European Conference on Computer Vision, 2022.

5. Meister, D., et al. “N-BVH: Neural Bounding Volume Hierarchies for Real-Time Ray Tracing.” SIGGRAPH Conference Papers, 2024

6. Mildenhall, B., et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” European Conference on Computer Vision, 2020.

7. “RenderFormer: Transformer-based Neural Rendering of Triangle Meshes with Global Illumination.” ACM SIGGRAPH 2025 Conference Papers, 2025

8. Müller, T., et al. “Instant neural graphics primitives with a multiresolution hash encoding.” ACM Transactions on Graphics, 2022.

9. Zeltner, T., et al. “Real-Time Neural Appearance Models.” ACM Transactions on Graphics, 2024

10. Chetlur, S., et al. “cuDNN: Efficient Primitives for Deep Learning.” ArXiv abs/1410.0759, 2014

11. Sitzmann, V., et al. “Implicit Neural Representations with Periodic Activation Functions.” Proceedings of the 34th International Conference on Neural Information Processing Systems, 2020

12. Zhang, D., et al. “RISE-SDF: A Relightable Information-Shared Signed Distance Field for Glossy Object Inverse Rendering.” International Conference on 3D Vision, 2025

13. Zhang, K., et al. “IRON: Inverse Rendering by Optimizing Neural SDFs and Materials from Photometric Images.” IEEE Conference on Computer Vision and Pattern Recognition, 2022

14. The Stanford 3D Scanning Repository. URL: http://graphics.stanford.edu/data/3Dscanrep

15. Li, Z., et al. “Sparc3D: Sparse Representation and Construction for High-Resolution 3D Shapes Modeling.” arXiv preprint arXiv:2505.14521, 2025

16. Bangaru, S. P., et al. “SLANG.D: Fast, Modular and Differentiable Shader Programming.” ACM Transactions on Graphics, 2023

17. Hart, J. “Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces.” The Visual Computer, 1995

18. Sztrajman, A., et al. “Neural BRDF Representation and Importance Sampling.” Computer Graphics Forum, 2021