Московский Государственный Университет им. М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Интеллектуальных Информационных Технологий

Никитин Сергей Денисович

Сравнительный анализ представлений функций дистанции со знаком

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
к.ф.-м.н.
В.А.Фролов

Москва, 2025

Аннотация

Дипломная работа посвящена сравнительному анализу функций дистанции со знаком (SDF) в задачах рендеринга 3D-моделей. Целью исследования является разработка и реализация бенчмарка для оценки различных представлений SDF по метрикам качества изображения (PSNR), времени рендеринга и размера модели. В работе рассмотрены теоретические основы SDF, включая аналитические, воксельные и нейронные представления, а также проведён обзор существующих подходов и их ограничений. Разработанный бенчмарк позволяет рендерить 3D-модели в различных представлениях и сравнивать их производительность на единой платформе. Эксперименты проведены на наборе тестовых моделей с использованием современного оборудования и программных инструментов. Работа имеет практическую значимость для компьютерной графики, игровой индустрии и 3D-моделирования.

Содержание

1 Введение

1.1 Актуальность

В современном мире комьютерная графика используется для решения широкого класса задач. Зародившись, как развлечение для небольшого числа людей, на текущий момент она породила целые индустрии, в который работают тысячи людей по всему миру. С каждым годом проработанность виртуальных миров возрастает, что в свою очередь требует увеличение вычилительных мощностей для рендера таких сцен и ресурсов для хранения полученных результатов.

В компьютерной графике существует множество представлений объектов. Однако самый популярный - полигональная сетка (меш), потому что с помощью неё можно сколь угодно точно задать поверхность. Чем сложнее поверхность, тем больше треугольников будет создано для поддержания заданной детализованности. Значит, возрастут затраты на память и рендер таких объектов.

Другим популярным представлением объектов можно назвать функцию расстояния со знаком (SDF). Функции дистанции со знаком (Signed Distance Functions, SDF) играют ключевую роль в современных задачах компьютерной графики, 3D-моделирования, рендеринга и физического моделирования. Они обеспечивают компактное и гибкое представление геометрических объектов, позволяя эффективно выполнять операции рендеринга, столкновений и оптимизации. Разнообразие подходов к реализации SDF, таких как аналитические, воксельные и нейронные представления, создаёт необходимость их сравнительного анализа для выбора оптимального решения в конкретных задачах.

SDF характеризуются рядом фундаментальных свойств, определяющих их практическую ценность в задачах компьютерной графики. Ключевым атрибутом является инвариантность относительно масштабирования и аффинных преобразований, что позволяет эффективно манипулировать геометрическими объектами без потери точности представления. Более того, SDF обладают свойством композиционности, что обеспечивает возможность построения сложных геометрических форм посредством математических операций над базовыми примитивами. Актуальность данной темы обусловлена растущим спросом на высококачественный и производительный рендеринг в игровой индустрии, виртуальной реальности и автоматизированном проектировании, где SDF находят широкое применение.

В данной работе решается актуальная задача приведения нескольких методов к формату, в котором их можно сравнить по нескольким метрикам и тем самым получить данные о плюсах и минусах рассматриваемых реализаций.

1.2 Цель работы

Цель: разработка бенчмарка для сравнительного анализа различных представлений функций дистанции со знаком по метрикам качества рендеринга (PSNR), времени рендеринга и размера модели, а также в выявлении их преимуществ и недостатков.

1. Изучить теоретические основы и существующие подходы к функциям дистанции со знаком.
2. Выбрать метрики и датасеты, которые будут максимально отражать требования к рассматриваемым методам.
3. Разработать программный бенчмарк для рендеринга 3D-моделей в различных представлениях SDF.
4. Провести экспериментальное сравнение представлений, проанализировать результаты и сформулировать рекомендации по выбору оптимальных представлений SDF.
5. Оформить полученный бенчмарк в виде докер контейнера для более простого и интуитивного запуска из "коробки".

1.3 Формальная постановка задачи

Вход:

1. Конфигурационный файл:
   (a) models $\in \mathbb{R}^{N_1 \times N_2}$ - пути до моделей, которые будут рендерится каждым выбранным методом. Всего моделей N_1 штук.
   (b) backends $\in \mathbb{R}^{N_3}$ - выбор бэкендов для рендера, это может быть CPU, GPU. N_3 - количество бэкендов.
   (c) renderers $\in \mathbb{R}^{N_4}$ - есть возможность использовать трассировку пути и обычный рендер.
   (d) render_modes $\in \mathbb{R}^{N_5}$ - массив чисел, каждое из которых представляет то, как будет рендерится каждый объект.
   (e) $w,\ h \in \mathbb{R}$ - разрешение, в котором рисуется изображение сцены.
   (f) cameras $\in \mathbb{R}$ - количество ракурсов.
   (g) spp $\in \mathbb{R}$ - количество лучей, пускаемых в каждый пиксель.
   (h) iters $\in \mathbb{R}$ - сколько раз надо отрендерить каждый ракурс для усреднения результатов.
   (i) output_folder $\in \mathbb{R}^{N_6}$ - путь до папки, куда будут сохраняться все результаты.

(j) repr_configs $\in \mathbb{R}^{\sum_{i=0}^{M} K_i \cdot B_i}$ - параметры для каждого метода, M - количество рассматриваемых методов, K_i - количество вариантов метода i, B_i - параметры для метода i.

2. slicer $\in \mathbb{R}^{N_7}$- путь до Kernel Slicer.

Выход:

1. build $\in \mathbb{R}^{M \times N_1 \times N_3 \times N_4 \times N_5 \times N_6}$ - папка, куда сохраняются все представления выбранных методов для каждой модели.

2. metrics - папка, содержащая полученные данные

   (a) build_phase $\in \mathbb{R}^{3 \times M \times N_1 \times N_3 \times N_4 \times N_5 \times N_6}$ - .csv файл, содержащий информацию о времени построения каждого метода для конкретных моделей. Например, время построения, полученный и оригинальный размер.

   (b) render_phase $\in \mathbb{R}^{8 \times M \times N_1 \times N_3 \times N_4 \times N_5 \times N_6}$ - .csv файл, куда записываются такие вещи, как время рендера, размер модели, метрики качества.

3. images $\in \mathbb{R}^{w \times h \times cameras \times M \times N_1 \times N_3 \times N_4 \times N_5 \times N_6}$ - папка с отрендеренными изображениями.

2 Обзор существующих работ

Несмотря на то, что существует множество методов сжатия 3D-геометрии, разработка метода сжатия или представления поверхности, который можно было бы использовать непосредственно во время рендеринга, остается серьезной проблемой. Это связано с фундаментальным компромиссом между достижением высокой степени сжатия, качеством и вычислительной сложностью метода декомпрессии, который может быть трудно распараллелить [Barczak et al. 2024; Kuth et al. 2024; Mlakar et al. 2024; Николаев и др. 2022]. Другой проблемой являются уровни детализации (LoD), которые обычно увеличивают потребление памяти, поскольку они хранятся в виде отдельных структур данных и также могут создавать артефакты [Haydel et al., 2023; Lloyd et al., 2020; Maggiordomo et al., 2023].

Отдельно стоит отметить выбор метода рендеринга, который так или иначе будет использоваться на протяжении всех работ. Это выбор между растеризацией и трассировкой лучей. В данном исследовании уделяется больше внимания методам, основанным на трассировке лучей, и на это есть три причины.

1. Обычно алгоритмическая сложность рендеринга с растеризацией линейно зависит от количества полигонов (либо геометрических примитивов), в то время как при трассировке лучей она линейно зависит от количества пикселей изображения и логарифмически зависит от количества полигонов. Существующие нестандартные методы рендеринга используют трассировку лучей или пирамид для определения видимости [Xue 2016].

2. Вторая причина - универсальность трассировки лучей, которая позволяет визуализировать примитивы любого вида путем алгоритмического определения пересечения луча и примитива: т.е. не требует обязательной тесселяции в треугольники.

3. Наконец, третья причина - это возможность рендеринга как на центральном, так и на графическом процессорах, даже одновременно, если это необходимо. Это важно для научной визуализации и для рендеринга CAE-данных, поскольку позволяет работать с огромными объемами данных в оперативной памяти. Например, это активно используют визуализаторы на базе OSPRay [Wald et al., 2017], а в индустрии кинопроизводства MoonRay [Lee et al., 2017]. [2017] использует этот путь из-за высокой гибкости рендерера на базе центрального процессора.

2.1 Представления на основе полигональной сетки

Стандартный подход с использованием уровня детализации (LoD) [Chen et al., 2023; Garland and Heckbert, 1997] не позволяет достичь двух важнейших целей.

Во-первых, он не обеспечивает компактного хранения геометрии, поскольку каждый уровень детализации постепенно увеличивает потребление памяти. Во-вторых, он по своей сути не поддерживает плавные переходы с высоким уровнем детализации и потоковую передачу. В то же время алгоритмы сжатия сетки часто создают представления, которые трудно визуализировать, особенно при трассировке лучей [Maglo et al., 2015].

Геометрические изображения [Gu et al., 2002] представляют собой метод преобразования произвольной поверхности в полностью правильную структуру. Это позволяет легко создавать новые сетки и обрабатывать геометрию как изображение, которое может быть дополнительно сжато [Edavamadathil Sivaram и др., 2024; Peyré and Mallat, 2005] и с трассировкой лучей [Carret al., 2006], обеспечивая при этом как динамическую геометрию, так и эффективный уровень детализации схемы без дополнительных затрат. Создание геометрического изображения требует хорошей параметризации 3D-поверхности, которую не всегда легко получить.

Micromesh [Maggiordomo et al., 2023] преобразует сетку с большим количеством полигонов в “базовую” сетку с низким количеством полигонов вместе со смещением. Векторы смещений сохраняются для каждой вершины в расчлененной (базовой) сетке, а карта смещений используется внутри полигона. В некотором смысле, этот подход можно рассматривать как эволюцию геометрических изображений, адаптированных к конкретным требованиям к качеству и оптимизированных для современного графического оборудования.

Концепция дерева тесселяции, представленная в [Haydel et al., 2023], направлена на сокращение передачи данных из памяти в блоки обработки при растеризации и трассировке лучей треугольных сеток. Это исследование подчеркивает важность уровней детализации для повышения производительности трассировки лучей и энергоэффективности. Однако оно не решает проблему компактного хранения геометрических данных. Вполне вероятно, что подход [Haydel et al., 2023] принят Nvidia в расширении Vulkan под названием VK_NV_cluster_acceleration_structure [VK_NV_TC, 2025].

Представления на основе сетки были и остаются современным подходом (SOTA) к рендерингу в компьютерной графике. Это подтверждается недавними демонстрациями от Nvidia: [RTXMG 2025; VK_NV_TC 2025; VK_NV_LC 2025]. Ключевым аспектом здесь является то, что как методы тесселяции [VK_NV_TC 2025], так и методы определения уровня детализации (LOD) [VK_NV_LC 2025] совместимы как с растеризацией, так и с трассировкой лучей. Учитывая, что производители графических процессоров внедряют аппаратное ускорение для этих методов, мы ожидаем, что наиболее эффективная реализация будет включать использование соответствующих функций расширения Vulkan, которые напрямую поддерживаются драйвером графического процессора.

Таким образом, программная архитектура бенчмарка сохраняет возможность обработки сеток с помощью трассировки лучей с возможностью интеграции таких подходов при необходимости.

2.2 Неявные аналитические представления

Набор разреженных блоков (SBS) [Herman et al., 2022] объединяет BVH с небольшими регулярными сетками внутри его листьев, называемыми блоками. В нем используются эффективные алгоритмы пересечения (аналитический метод и метод Ньютона) для поиска пересечений лучей и SDF. Эти алгоритмы запрашивают значения SDF в углах вокселя, которые должны явно поддерживаться структурой данных. VDB [Museth 2013] использует иерархические многоуровневые сетки (своего рода B + дерево в трехмерном пространстве), что помогает поддерживать баланс высот при построении и обходе. VDB использует битовую маску, которая хранит информацию о том, какие вокселы имеют значение; они называются “активными вокселами”. Однако VDB не поддерживает значения самой функции SDF, и в то же время иерархические сетки обычно демонстрируют низкую производительность рендеринга. Оптимизированная графическая реализация преобразования лучей с помощью структур VDB представлена в [Hoetzlein 2016], где луч пересекает объем узла с помощью дифференциального 3D-анализатора (3DDA).

Октодерево разреженных вокселей (SVO) [Laine and Karras, 2010] включает структуру, которая хранит только занятые области; большая часть воксельной информации хранится на родительском уровне, чтобы уменьшить использование памяти. Другим применением SVO является работа [Pujol and Chica 2024], в которой представлено исходное октадерево с добавлением того, что коэффициенты многочлена сохраняются в листьях, если они имеют часть поверхности; в противном случае, на этапе предварительной обработки [Pujol and Chica 2024] пытается максимизировать размер из поля, обозначающего пустую область. Это усовершенствование позволяет хранить данные, расположенные только на границе объекта. На следующем этапе [Pujol and Chica 2024] использует полиномы для интерполяции значений SDF путем нахождения аналитического пересечения. Подход Пуйоля и Чики позволяет использовать различные типы интерполяции, но он также накладывает ограничения и проблемы, связанные с аналитическим поиском корней. Алгоритм построения SDF основан на их предыдущей работе [Pujol и Chica 2023], в которой представлен подход к эффективному представлению SDF и запросам к ним путем построения адаптивной структуры данных. Он использует RMSE для принятия решения о разделении узлов. [Pujol and Chica, 2023] могут использовать несколько методов интерполяции, таких как трикубический, для решения линейной системы [Boateng and Bradach, 2023], которая требует больших вычислительных затрат.

Хэш-таблицы [Nießner et al., 2013] обычно используются вместо октодерева в приложениях реконструкции, где геометрия постоянно меняется и важно быстро обновлять

структуру данных.

Методы рендеринга на основе SDF обладают несколькими ключевыми преимуществами:

1. Унифицированные структуры данных: Одни и те же представления данных могут использоваться как для поверхностного, так и для объемного рендеринга.

2. Простота генерации уровня детализации (LOD) и переключения во время рендеринга: Лоды могут быть получены простым способом, по крайней мере, для "водонепроницаемых"объектов.

2.3 Рендеринг геометрии CSG

В моделях CAD геометрия может быть определена с помощью так называемого представления Constructive Solid Geometry (CSG). Это представление описывает геометрию как набор базовых примитивов (таких как сфера, цилиндр, конус и другие) и операции, выполняемые над этими примитивами, как над наборами. Основным преимуществом такого представления является его точность, как правило, высокая гибкость и чрезвычайная компактность (например, MPR CSG [Keeter 2020]). Основным недостатком является его высокая вычислительная сложность из-за потенциального значительного совпадения между примитивами в некоторых моделях, что вынуждает алгоритм рендеринга интенсивно перебирать поддеревья CSG. Чтобы эффективно отобразить такое представление, необходимо преобразовать древовидную иерархию CSG, минимизируя взаимные совпадения, когда это возможно [Bogolepov 2016]. Один из таких подходов называется “Повторное плетение” [Benthin et al., 2017], который уменьшает количество перекрывающихся элементов внутри объема листа. Однако недостатком ребрендинга является увеличение объема памяти и усложнение процедуры сборки.

MPR использует “ленту” для эффективного хранения данных и передачи их в графический процессор. Этот подход обеспечивает эффективную производительность и сжатие модели, состоящей из операций с множествами и самими типами множеств. Кроме того, метод предоставляет возможности для создания новых множеств с помощью пользовательских команд. С другой стороны, в отличие от формата SDF, этот формат специализируется только на обработке CSG, а не на чем-то общем. Еще одним недостатком является то, что программная реализация MPR может обрабатывать небольшое количество фигур. Поэтому сложно визуализировать что-то большое и с большим количеством деталей.

Визуализация геометрии CSG остается интересным направлением для будущих исследований и разработок. Наиболее многообещающие результаты можно увидеть в [Bogolepov 2016]. Объединив идеи [Bogolepov, 2016]; Бентин и др., 2017] и [Китер, 2020], можно было бы разработать передовую технологию.

3 Исследование и построение решения задачи

3.1 Выбор метрик

В рамках исследования методов представления функций дистанции со знаком (SDF) критически важным этапом является выбор метрик, позволяющих объективно оценить компромисс между качеством реконструкции, производительностью и эффективностью сжатия. Предлагаемый набор метрик - PSNR, размер модели и время рендеринга - формирует комплексную систему оценки, учитывающую ключевые требования задач компьютерной графики: точность воспроизведения геометрии, возможность работы в реальном времени и компактность представления данных.

1. **Пиковое отношение сигнала к шуму (PSNR)**

   PSNR служит количественной мерой точности реконструкции поверхности относительно эталонного меша. Для SDF, задающих расстояние до поверхности со знаком, PSNR вычисляется через среднеквадратичную ошибку (MSE) между эталонными и восстановленными значениями расстояний:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(d_{\text{ref}}^{(i)} - d_{\text{pred}}^{(i)})^2$$

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right)$$

   где MAX - максимальное расстояние в сцене, N - количество пробных точек. Для 8-битных данных $MAX = 255$, но в 3D-контексте значение определяется масштабом сцены. Высокий PSNR (>30 дБ) указывает на близость восстановленной геометрии к эталону, что критично для приложений, требующих точного воспроизведения деталей (медицинская визуализация, инженерное моделирование).

   Ограничение PSNR - слабая корреляция с субъективным восприятием качества при наличии структурных искажений. Однако для задач, где приоритетом является метрическая точность (напр., CAD-модели), PSNR остаётся эталонным инструментом.

2. **Размер модели**

   Эффективность сжатия оценивается через объём памяти, требуемый для хранения параметров представления. Для нейросетевых методов типа DeepSDF (ссылка на статью) размер определяется числом параметров сети. Традиционные SDF в воксельном представлении требуют $O(n^3)$ памяти.

   Сравнение коэффициентов сжатия $C = \frac{V_{raw}}{V_{compressed}}$ позволяет оценить выигрыш от использования компактных представлений.

3. **Время рендеринга**

   Производительность рендеринга измеряется временем генерации кадра при использовании:

   - **Ray marching** с шагом Δt:

   $$t_{\text{total}} = N_{\text{rays}} \cdot \left(\frac{t_{\max} - t_{\min}}{\Delta t} \right) \cdot t_{\text{SDF}}$$

     где t_{SDF} - время вычисления SDF в точке.

   - **Ray tracing** (аналитический поиск персечения):

   $$t_{\text{total}} = N_{rays} \cdot (w \cdot h + \log N_{triangles})$$

     где w, h - размеры изображения.

   Нейросетевые SDF (напр., DeepSDF) увеличивают t_{SDF} из-за вычислений прямого распространения, но компенсируют это сокращением n благодаря непрерывности представления.

   Оптимизации типа иерархических структур ускорения или кэширования значений SDF критичны для достижения частоты кадров >30 FPS.

4. **Синтез требований**

   Выбранные метрики образуют треугольник компромиссов:

   - PSNR $\Leftrightarrow$ Размер модели: Повышение точности (PSNR) через увеличение разрешения или сложности модели ведёт к росту объёма данных.
   - PSNR $\Leftrightarrow$ Время рендеринга: Точные SDF с высокой детализацией требуют больше вычислений на луч.
   - Размер модели $\Leftrightarrow$ Время рендеринга: Сжатые представления (напр., нейросетевые) уменьшают объём данных, но могут замедлять рендеринг из-за вычислений параметров.

Предложенный набор метрик позволяет количественно оценить применимость методов SDF для задач, требующих баланса между точностью, производительностью и эффективностью использования ресурсов. Дальнейшие исследования могут дополнить систему оценкой структурного сходства (SSIM) и субъективным тестированием, но текущий выбор оптимален для инженерного анализа.

3.2 Выбор датасета

Для всесторонней оценки методов представления функций дистанции со знаком (SDF) разработана двухуровневая система тестирования, использующая комплементарные датасеты. Такой подход позволяет анализировать как базовую эффективность методов в штатных условиях, так и их устойчивость к экстремальным нагрузкам, что критично для валидации промышленных решений.

Эти две коллекции обеспечивают:

- Стандартизированную геометрию с размерами, нормализованными в диапазоне $[-0.5, 0.5]^3$, что устраняет артефакты масштабирования
- Водонепроницаемые меши (watertight), гарантирующие корректность расчёта SDF через алгоритм ray marching
- Низкую полигональную сложность (меньше 10^4 треугольников на модель), позволяющую тестировать базовые свойства методов без перегрузки вычислительных ресурсов

1. Базовый датасет: верификация функциональности

   В этом сравнении использовался более простой набор моделей, на которых методы, задействованные в сравнении, не нарушались. Цель состояла в том, чтобы найти перспективный существующий метод, даже если его следует доработать для устранения артефактов в целевых моделях.

   Для генерации SDF используются полигональные сетки из датасета. Это обеспечивает воспроизводимость экспериментов и прямую сопоставимость метрик PSNR между методами.

2. Стресс-тестовый датасет: анализ масштабируемости

   В качестве эталонного набора выбраны модели из ABC-Dataset, содержащий 1 млн CAD-моделей соответственно. Выбранные модели уже сложнее, поэтому для на них тестировались не все методы, а только наиболее эффективные. Также некоторые варианты, например, имеют тонкие поверхности, что является серьезным испытанием для SDF методов.

   Таким образом, собранные модели проверяют методы на способность рендерить большие модели, обладающие сложной для SDF структурой.

Использование двух комплементарных датасетов обеспечивает многоуровневую валидацию методов представления SDF, критически важную для оценки их практической применимости. Базовый датасет (напр., модели двигателей, скелетов) служит контролируемой средой для верификации корректности алгоритмов в идеализированных условиях, где метрики PSNR и время рендеринга отражают фундаментальные свойства методов без искажающего влияния шумов или артефактов. Это позволяет сравнить

методы на стандартизированной базе, обеспечивая воспроизводимость результатов и прямую сопоставимость с предыдущими исследованиями.

Одновременно стресс-тестовый датасет выявляет пределы масштабируемости, демонстрируя, как методы ведут себя при обработке неводонепроницаемых мешей, моделей с $>10^8$ треугольников. Такая комбинация позволяет оценить устойчивость алгоритмов к реальным вызовам.

Синтез результатов двух датасетов формирует треугольник компромиссов "точность-производительность-память позволяя выбрать оптимальный метод для конкретной задачи: нейронные методы для компактного представления стандартных объектов против гибридных сеток для промышленных CAD-моделей. Без многоуровневого тестирования сохраняется риск ложной оптимизации - улучшения метрик на синтетических данных при деградации на реальных сценах.

Первый датасет	
Название модели	**Размер, МБ**
Manifold	286.6
Bunny	10.37
Skeleton	196.8
Roadbike	163.3
Buddha	99.82
Crankcase	141.0
Cylinder	71.47
Stone	124.0
Buggy	634.6
Dragon	72.76
Radiator	371.4
WaterPump	72.48

Второй датасет (выборка из ABC)	
Название подмножества	**Размер, МБ**
ABC_80006	154
ABC_83870	199
ABC_88060	84
ABC_88828	20
ABC_515447	58
ABC_687231	108
ABC_701584	45
ABC_702109	122
bulldozer	1511
DEMONSTRATOR	591

Таблица 1: Сравнение размеров моделей в двух датасетах

4 Практическая реализация

4.1 Структура бенчмарка

В ходе проделанной работы был создан бенчмарк, который способен сравнивать, как методы, которые пользователь может реализовать сам, так и готовые решения, предоставляемые в виде готовых проектов. Есть возможность переносить написанные методы на графический ускоритель, используя средства Kernel Slicer [ссылка] для генерации кода для API Vulkan из C++ кода. Для такой трансформации кода написанные методы должны удовлетворять определенным тредованиям этого инструмента.

Сцена может рисоваться как в режиме Мультирендера [ссылка на LiteRT], который использует стандартную трассировку и шагающую трассировку лучей для поиска пересечения с поверхностью, так и рендер систему HydraCore3 [ссылка], разрабатываемую в лаборатории компьютерной графики и мультимедиа факультета ВМК МГУ, которая способна рендерить реалистичные сцены за счет использования трассировки пути. На рисунке 1 показана схема работы бенчмарка.

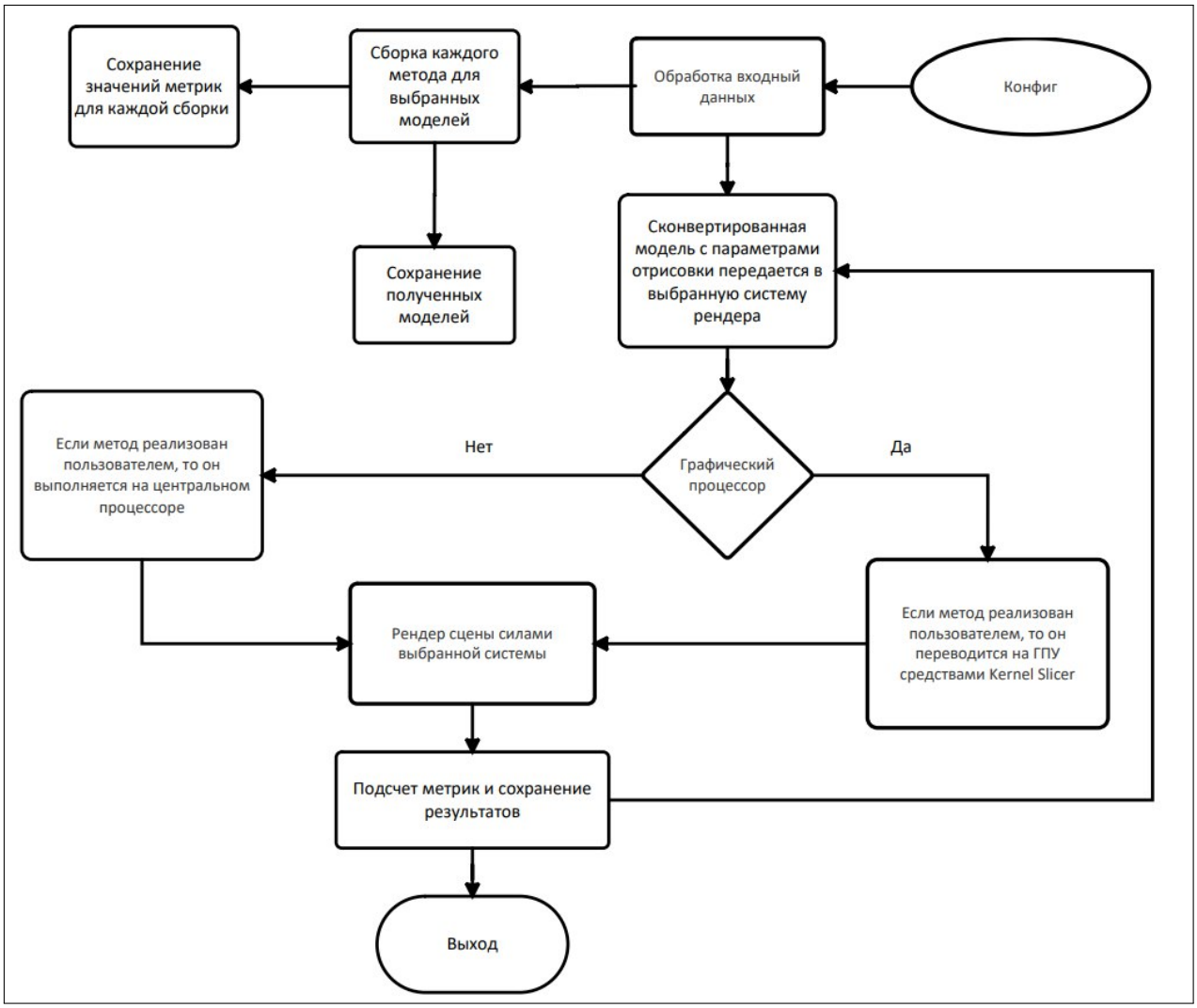

Рис. 1: Схема реализация сравнений методов между собой

```
<geometry_lib total_chunks="4">
  <mesh id="0" name="my_box" type="vsgf" bytesize="1304" loc="data/cornell_open.vsgf" offset="0" vertNum="20" triNum="10" dl="0" path='
    <positions type="array4f" bytesize="320" offset="24" apply="vertex" />
    <normals type="array4f" bytesize="320" offset="344" apply="vertex" />
    <tangents type="array4f" bytesize="320" offset="664" apply="vertex" />
    <texcoords type="array2f" bytesize="160" offset="984" apply="vertex" />
    <indices type="array1i" bytesize="120" offset="1144" apply="tlist" />
    <matindices type="array1i" bytesize="40" offset="1264" apply="primitive" />
  </mesh>
  <normalized_mesh id="1" name="dragon" type="vsgf" type="normalized_mesh" loc="../../saves/dragon/mesh.vsgf" mat_id="4">
  </normalized_mesh>
  <mesh id="2" name="my_area_light_lightmesh" type="vsgf" bytesize="280" loc="data/chunk_00003.vsgf" offset="0" vertNum="4" triNum="2"
    <positions type="array4f" bytesize="64" offset="24" apply="vertex" />
    <normals type="array4f" bytesize="64" offset="88" apply="vertex" />
    <tangents type="array4f" bytesize="64" offset="152" apply="vertex" />
    <texcoords type="array2f" bytesize="32" offset="216" apply="vertex" />
    <indices type="array1i" bytesize="24" offset="248" apply="tlist" />
    <matindices type="array1i" bytesize="8" offset="272" apply="primitive" />
  </mesh>
</geometry_lib>
<lights_lib>
  <light id="0" name="my_area_light" type="area" shape="rect" distribution="diffuse" visible="1" mat_id="5" mesh_id="2">
    <size half_length="1" half_width="1" />
    <intensity>
      <color val="1 1 1" />
      <multiplier val="25" />
    </intensity>
  </light>
</lights_lib>
<cam_lib>
  <camera id="0" name="my camera" type="uvn">
    <fov>45</fov>
    <nearClipPlane>0.01</nearClipPlane>
    <farClipPlane>100.0</farClipPlane>
    <up>0 1 0</up>
    <position>0 -1.5 8</position>
    <look_at>0 -1.5 0</look_at>
  </camera>
</cam_lib>
```

Рис. 2: Пример хранения информации о модели, её расположении и тд.

Работа бенчмарка идет в два этапа:

1. Сборка каждым методом всех моделей с конкретными параметрами рендера. Полученные данные сохраняются в отдельные папки, откуда будут считываться во время рендера. Также записывается время конвертации, размер и другие данные в специальный .csv файл.

2. Во время рендера из каждой папки берутся данные и рендерятся. Результат сравнивается с референсным методом (по умолчанию это полигональная сетка) и сохраняется в .csv файл.

4.2 Контейнеризация

Про докер

4.3 Возникшие проблемы

Во время написания программы возниклом множество проблем. Их можно разделить на 4 группы:

1. Методы, имеющие открытую программную реализацию, часто не имели нужных входных параметров. Например, позиция камеры, позиция источника света. В

некоторых методах модель нормализуется в куб $[-1, 1]^3$, а не $[-0.5, 0.5]^3$, что давало неверную картинку. Аналогичную проблему вызывали разные настройки камеры, потому что данные о ней могли по-разному храниться и использоваться. Учитывая, что изначально метод мог вообще не принимать на вход информацию о ней, то приходилось это исправлять. Другой проблемой было освещение, потому что, например, в нескольких методах используется модель физического освещения, а в других обходятся обычным рассчетом мест затенения, поэтому приходилось убирать такую сложную модель освещения и заменять на более простую, чтобы полученные картинки совпадали по теням и цвету.

2. Возникла проблема организации хранения данных, потому что очень много параметров, которые меняют выходной результат. Таким образом, на первом этапе бенчмарк сохраняет в отдельную папку бинарный файл с моделью и .xml файл, который используется для её дальнейшей загрузки в выбранную систему рендера. На втором этапе создается другая папка, которая будет содержать отрендеренные изображения. На этом шаге сначала рисуется эталонный метод, чтобы другие реализации могли сразу с ним сравниваться. Все полученные результаты метрик записываются в третью папку. Выбранный подход позволяет получить на каждом этапе простую иерархию данных.

3. При добавлении метода "Массово-параллельный рендеринг сложных неявных поверхностей замкнутой формы"(MPR) пришлось решать проблему, связанную с тем, что его было не с чем сравнивать, потому что он рендерит конструктивную сплошную геометрию, представленную в виде дерева, а не полигональной сеткой. Для этого метода пришлось реализовывать свое синтаксическое дерево, которое уже могло использоваться, как функция дистанции. Следовательно, появился способ сравнивать SDF методы с методами, реализующими конструктивную сплошную геометрию. Такой подход позволяет использовать бенчмарк для более широкого класса методов, которые изначально могут не иметь возможность сравнения с уже добавленными вариантами.

4. Ещё не маловажная задача - подбор параметров для каждого метода. Так как хотелось бы все модели получить примерно одинакового веса, чтобы методы могли честно соревноваться в скорости рендера и качестве полученных картинок. Для выполнения этой задачи было решено первый датасет сжать до 2 Мб. Второй датасет сжимался в 10 раз.

5 Эксперименты и результаты

Поскольку различные методы имеют разные ограничения, было некорректно сравнивать их на моделях, на которых некоторые из методов просто не работали бы. Поэтому сначала выбирается упрощенный набор 3D-моделей, которые удовлетворяют ограничениям для всех существующих методов, это сравнение по длинному списку. Затем выбирается подмножество наиболее эффективных методов и проводится сравнение 3D-моделей из набора данных ABC. Это сравнение по короткому списку: Instant NGP, NG LOG, N-BVH, LoD97.

5.1 Пример конфигурационного файла

```
{
backends:arr = { "CPU" }
renderers:arr = { "MR"}
render_modes:arr = { "LAMBERT_NO_TEX" }
models:arr =
{
  "obj/ABC_80006.obj"
}
width:i = 1024
height:i = 1024
cameras:i = 4
iters:i = 1
spp:i = 4
output_folder:s = "results"
render_results_filename:s = "render"
build_results_filename:s = "build"
ref_type:s = "MESH"

repr_configs
{
 MESH     { default {                                              } }
 INGP     { ingp    { iters:i = 8000 path:s = "../instant-ngp"     } }
 SBS      { sbs     {depth:i = 8 brick_size:i = 3 brick_pad:i  = 0 } }
 OPENVDB  { openvdb { w:i = 5 size:r = 0.01                        } }
}
}
```

5.2 Длинный список

Сравнение по длинному списку было выполнено во время работы над статьей по SCom дереву. В этом сравнении мы использовали более простой набор моделей, на которых методы, задействованные в сравнении, не нарушались (рис. 3). Цель состояла в том, чтобы найти перспективный существующий метод, даже если его следует доработать для устранения артефактов в целевых моделях.

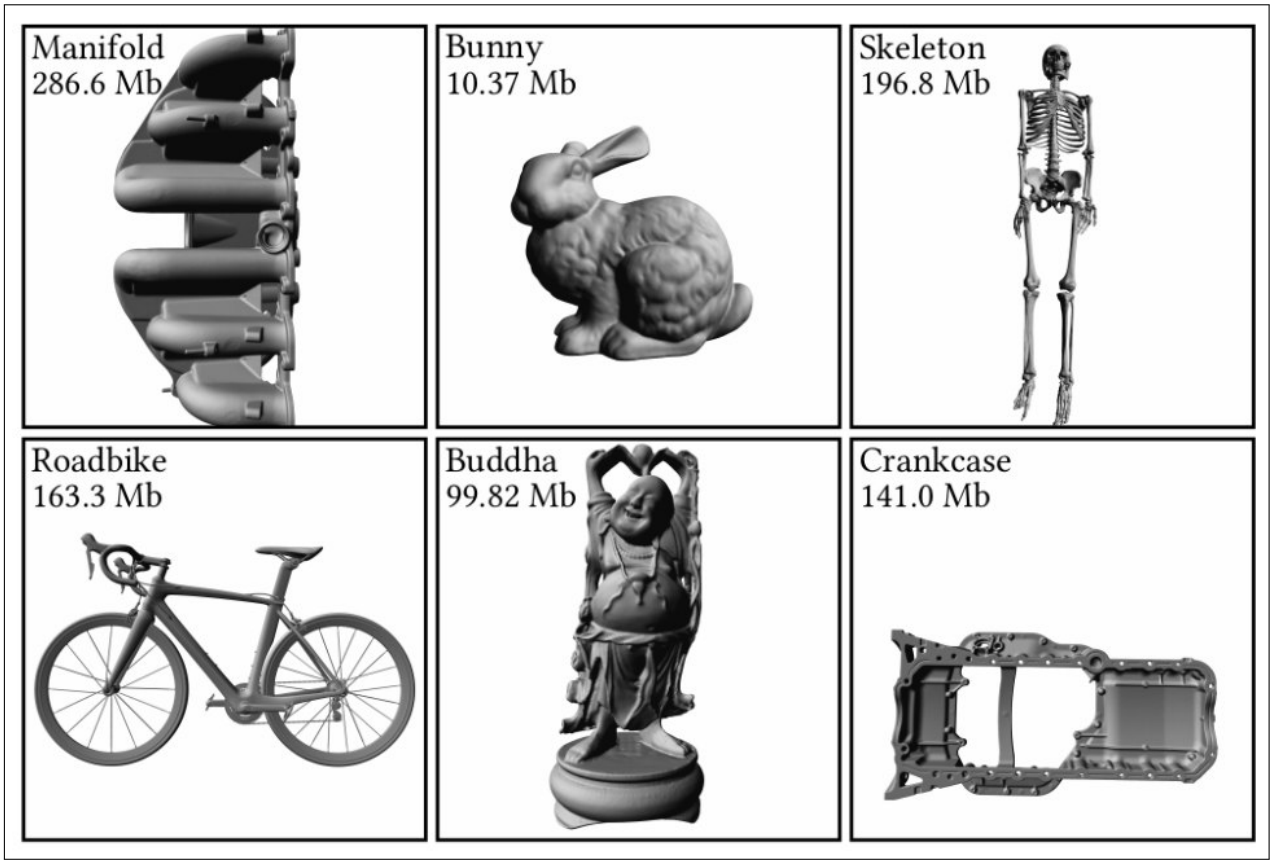

Рис. 3

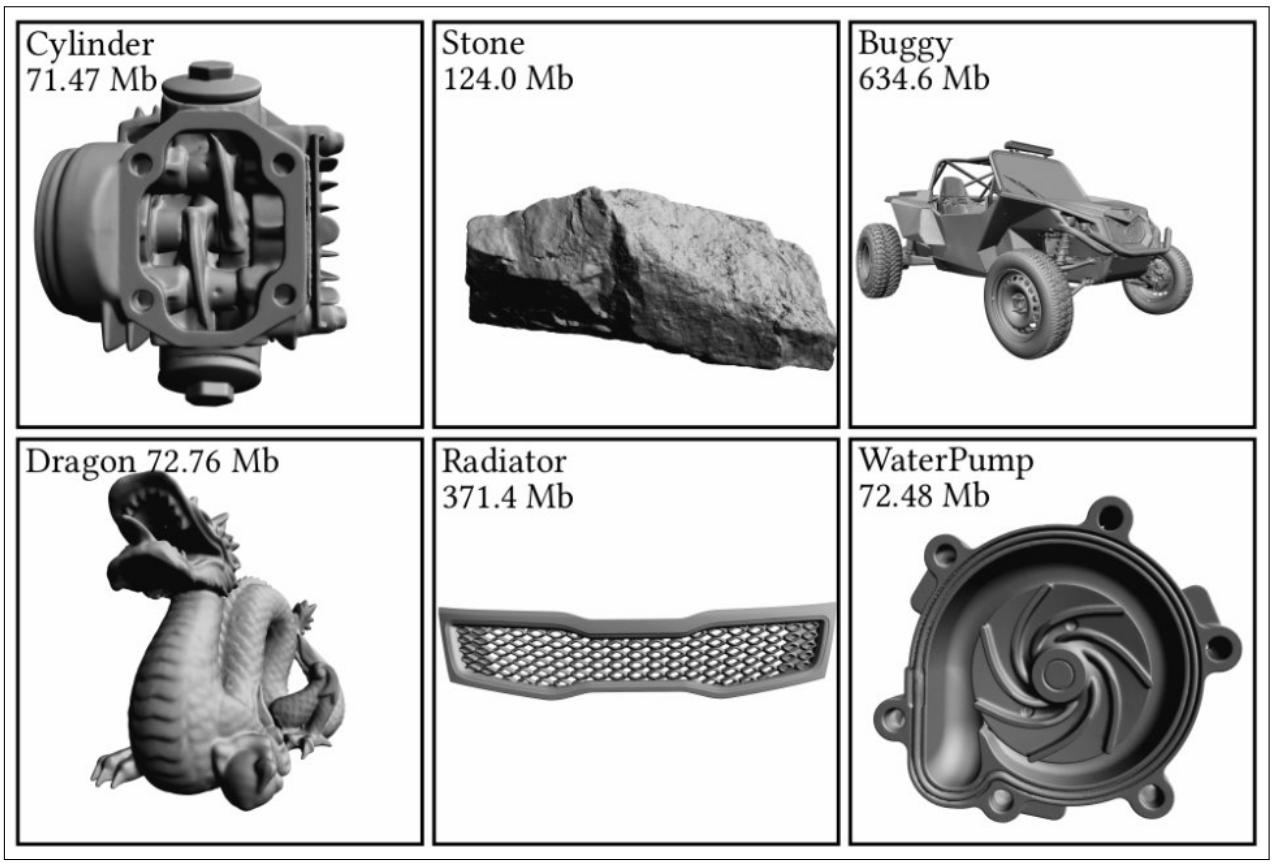

Рис. 4: Тестовые сцены для методов из длинного списка.

Выбранные методы:

- Decimation [Garland and Heckbert 1997]
- RoLoPM [Chen et al. 2023]
- Micromesh [Maggiordomo et al. 2023]
- N-BVH [Weier et al. 2024]
- NGLOD [Takikawa et al. 2021]
- GVDB [Hoetzlein 2016]
- SBS [Herman et al. 2022]
- SDF Octree [Pujol and Chica 2023; Pujol and Chica 2024]

Scene	Decimation			RoLoPM			Micromesh		N-BVH		NGLOD		GVDB		SBS			SDF Octree			Ours		
	PSNR	t_{sw}	t_{hw}	*PSNR*	t_{sw}	t_{hw}	*PSNR*	*t*	*PSNR*	*t*	*PSNR*	*t*	*PSNR*	*t*	*PSNR*	t_{sw}	t_{hw}	*PSNR*	t_{sw}	t_{hw}	*PSNR*	t_{sw}	t_{hw}
Buddha	34.71	0.99	**0.48**	29.41	1.10	0.49	36.41	**0.71**	29.34	3.00	32.16	1813.51	32.10	40.69	35.25	3.85	3.38	27.36	1.12	1.07	**41.29**	5.97	6.14
Buggy	25.82	**1.25**	**0.52**	25.59	1.33	0.53	-	-	**29.00**	3.88	19.77	1641.11	23.50	40.58	24.30	5.37	4.67	20.61	1.50	1.35	28.60	7.02	7.23
Bunny	42.41	1.40	0.57	34.62	1.19	**0.56**	40.30	**0.71**	37.79	7.17	35.59	1622.76	40.41	37.82	42.98	5.66	4.97	34.03	1.60	1.44	**44.11**	9.39	9.60
Crankcase	34.47	0.76	**0.46**	33.09	0.84	0.47	36.69	**0.68**	37.12	2.71	21.90	1139.05	37.65	57.71	34.51	2.68	2.32	27.55	1.00	0.93	**48.11**	4.64	4.84
Cylinder	31.25	0.72	**0.28**	25.76	**0.70**	0.31	32.67	0.71	33.37	6.60	30.67	1665.35	30.66	36.00	29.63	2.10	1.82	21.23	0.74	0.69	**40.95**	3.30	3.36
Dragon	33.91	1.36	**0.53**	27.08	1.41	0.56	33.99	**0.71**	35.25	6.46	21.78	1280.27	34.25	39.37	37.54	5.27	4.59	27.95	1.50	1.37	**41.89**	7.29	7.45
Manifold	30.32	1.15	0.51	29.24	1.27	**0.51**	31.80	**0.71**	34.31	5.02	21.35	1421.02	31.04	40.11	36.29	4.99	4.40	23.59	1.53	1.37	**42.10**	6.73	6.93
Pump	33.13	1.36	**0.54**	32.04	1.37	0.55	34.94	**0.71**	34.87	4.77	31.08	1479.67	33.35	37.79	31.64	4.16	3.76	27.83	1.52	1.36	**44.62**	7.21	7.34
Radiator	36.71	0.45	0.24	33.23	**0.43**	**0.23**	34.28	0.70	36.96	2.80	31.56	1637.45	33.65	58.77	36.82	1.23	0.98	27.49	0.44	0.44	**45.65**	1.58	1.80
Roadbike	20.93	1.40	**0.47**	25.42	1.14	0.49	19.67	**0.71**	**32.81**	3.22	19.36	1455.56	28.89	57.90	29.33	3.43	2.76	25.56	0.96	0.91	32.42	5.08	5.30
Skeleton	35.09	0.69	**0.23**	30.92	0.78	0.24	30.66	0.78	36.71	2.50	30.25	1576.27	32.32	58.42	36.81	1.20	1.02	27.33	**0.41**	0.34	**44.86**	1.55	1.75
Stone	31.49	0.92	**0.48**	33.13	1.00	0.49	32.68	**0.70**	35.89	3.34	33.34	1727.28	33.60	40.41	36.87	3.33	2.85	30.41	0.99	0.94	**41.37**	4.83	5.05

Рис. 5: Сравнение с другими методами на сценах из длинного списка. Параметры всех методов были скорректированы для получения модели размером 2 Мб. Для моделей Buggy и Roadbike этого размера недостаточно из-за большого количества деталей на них, что приводит к снижению PSNR. Рендеринг в разрешении 1400 × 1400 на графическом процессоре Nvidia RTX 3070. **Средний PSNR = 40,8**.

5.3 Короткий список

Список методов:

- Decimation [Garland and Heckbert 1997]
- Intant NGP (IMGP) [Müller et. al 2022]
- N-BVH [Weier et al. 2024]
- NGLOD [Takikawa et al. 2021]
- SBS [Herman et al. 2022]

Scene compressed in Mb	INGP		MESH LOD		NBVH		NGLOD		SDF SBS		Ours	
	PSNR	t	PSNR	t	PSNR	t	PSNR	t	PSNR	t	PSNR	t
ABC_80006 (15.4)	24.4373	2.0208	20.8237	1.25595	30.0934	1.38029	32.9684	729.374	29.6357	1.39803	**33.2679**	1.39631
ABC_83870 (19.9)	21.9288	1.5898	**33.4161**	0.7658	27.4981	1.10991	27.1096	1217.91	27.6569	1.92584	30.7536	2.04484
ABC_88060 (8.4)	25.4127	1.6002	31.6253	0.94485	32.1536	3.56534	31.427	1187.38	33.7094	0.810781	**36.724**	1.08488
ABC_88828 (2.0)	25.3924	1.5706	21.2852	0.2608	**30.956**	3.84097	26.6941	953.696	22.4191	0.43615	22.7819	0.6985
ABC_515447 (5.8)	25.0376	1.54	**37.2475**	0.3861	29.8905	1.40661	29.2517	1093.44	26.0431	0.671531	32.6907	0.832969
ABC_687231 (10.8)	24.386	1.6312	26.6885	0.38195	29.5108	4.58807	28.7878	927.968	31.1062	1.63666	**32.2423**	1.60022
ABC_701584 (4.5)	23.4694	1.615	28.3015	0.3198	22.3917	0.903047	30.4694	995.088	29.347	0.428219	**34.3476**	1.28916
ABC_702109 (12.2)	24.2761	1.5688	22.9275	0.43865	31.0398	4.03193	**31.4955**	829.598	24.6758	0.956313	26.5279	1.26706
bulldozer (151.1)	-	-	-	-	-	-	-	-	24.1367	1.39813	**26.843**	3.43475
DEMONSTRATOR (59.1)	26.4516	1.6082	-	-	**31.4977**	0.810695	29.8707	828.228	24.5772	0.645969	28.1639	0.955281

Таблица 2: Сравнение с другими методами на наборе данных ABC из короткого списка, сжатом примерно в 10 раз. Рендеринг в разрешении 1024 × 1024 на графическом процессоре Nvidia RTX 4080. **Среднее PSNR = 29,9**.

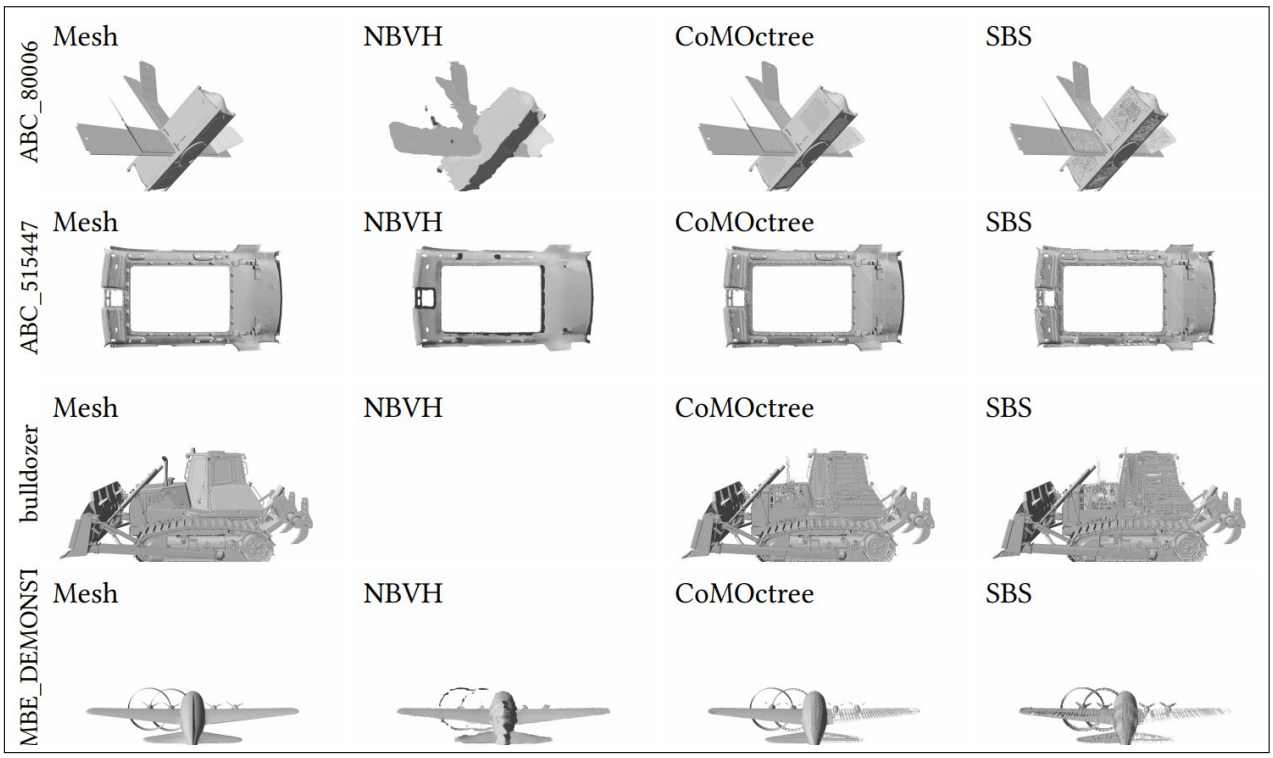

Рис. 6: Визуальное сравнение методов из короткого списка.

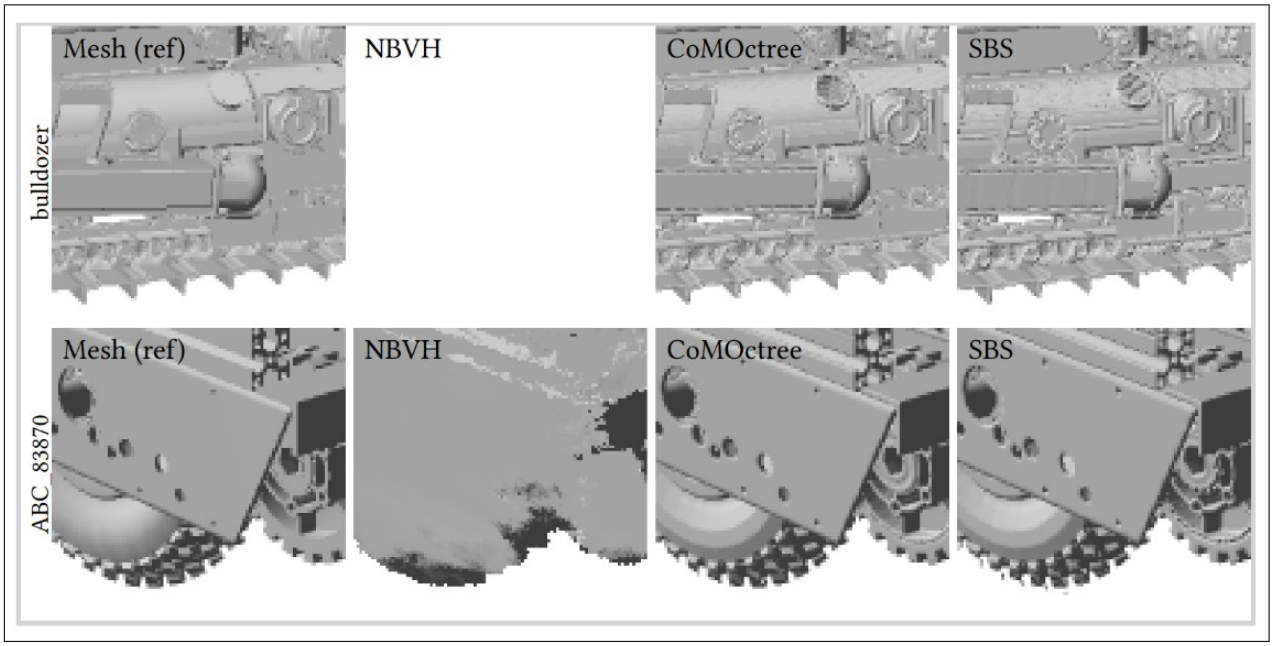

Рис. 7: Визуальное сравнение методов из короткого списка в приближении.

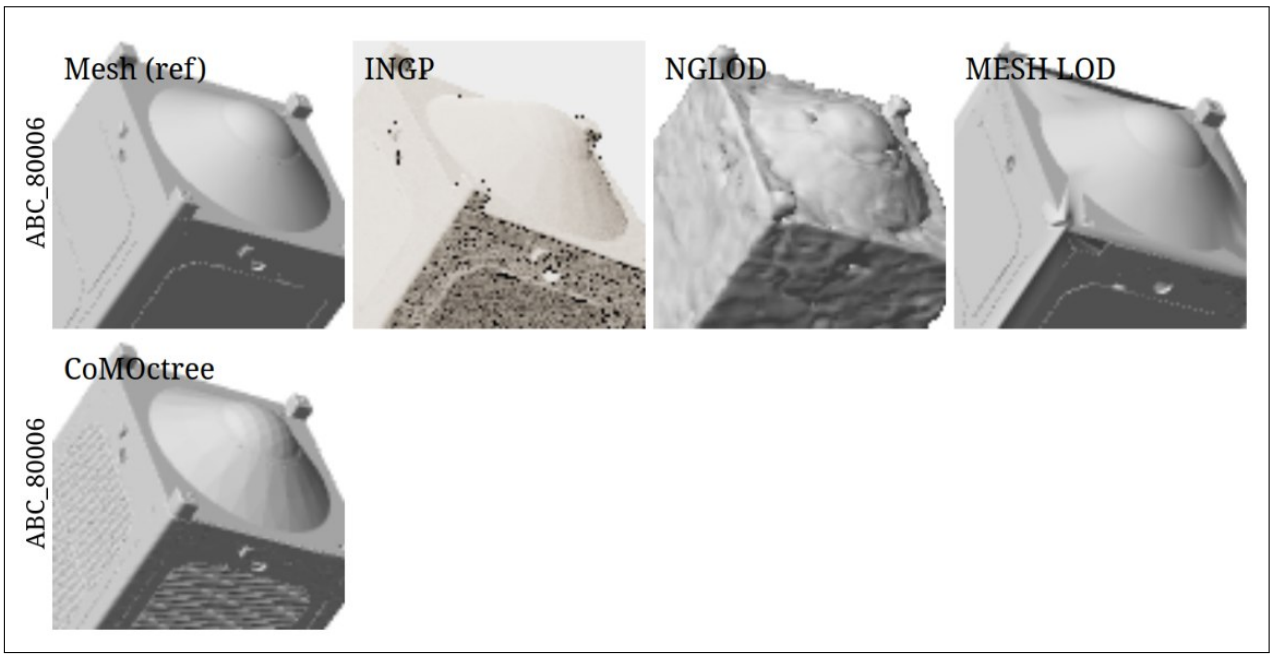

Рис. 8: Визуальное сравнение методов из короткого списка в приближении.

5.4 Вывод

На основании результатов [Таблица 2, Рис 5, 6, 7, 8], полученных на наборе данных ABC, можно заметить, что CoMOctree демонстрирует высокое качество. Хотя он не достигает самых высоких показателей сжатия по сравнению со всеми другими подходами, он все же обеспечивает конкурентоспособную производительность, сохраняя мелкие детали, которые часто теряются при использовании альтернативных методов. Кроме того, наш метод способен обрабатывать самые большие и сложные модели, такие как бульдозер в наборе данных, которые большинство других методов не могут обработать.

Основываясь на визуальном сравнении можно сказать, что нейросетевые методы [ссылки] достаточно хорошо сжимают информацию о модели, однако не способны в должной мере отрисовать мелкие детали и, в целом, выглядят мыльно, шумно или имеют волнистую форму, что может свидетельствовать о том, что они хорошо запоминают границы объекта, но со временем перестают должным образом его улучшать, возможно, из-за низкой ошибки.

Алгоритмические методы [ссылки] хоть и не могут сравниться с нейросетями по сжатию, но способны дать более качественный результат, и они не падают на больших моделях. Также время рендера у таких подходов меньше, так как не надо перемножать кучу весов, чтобы найти значение растояния до поверхности в каждом пикселе. Существующая проблема тонких поверхностей, как видно на примере модели DEMONSTRATOR в полной мере не решается, тут нейросети в какой-то мере лучше, однако новый подход предложенный в CoMOctree позволяет эту проблему победить, в отличие от SBS[ссылка], Octree[ссылка] и тд.

5.5 Сравнение с MPR

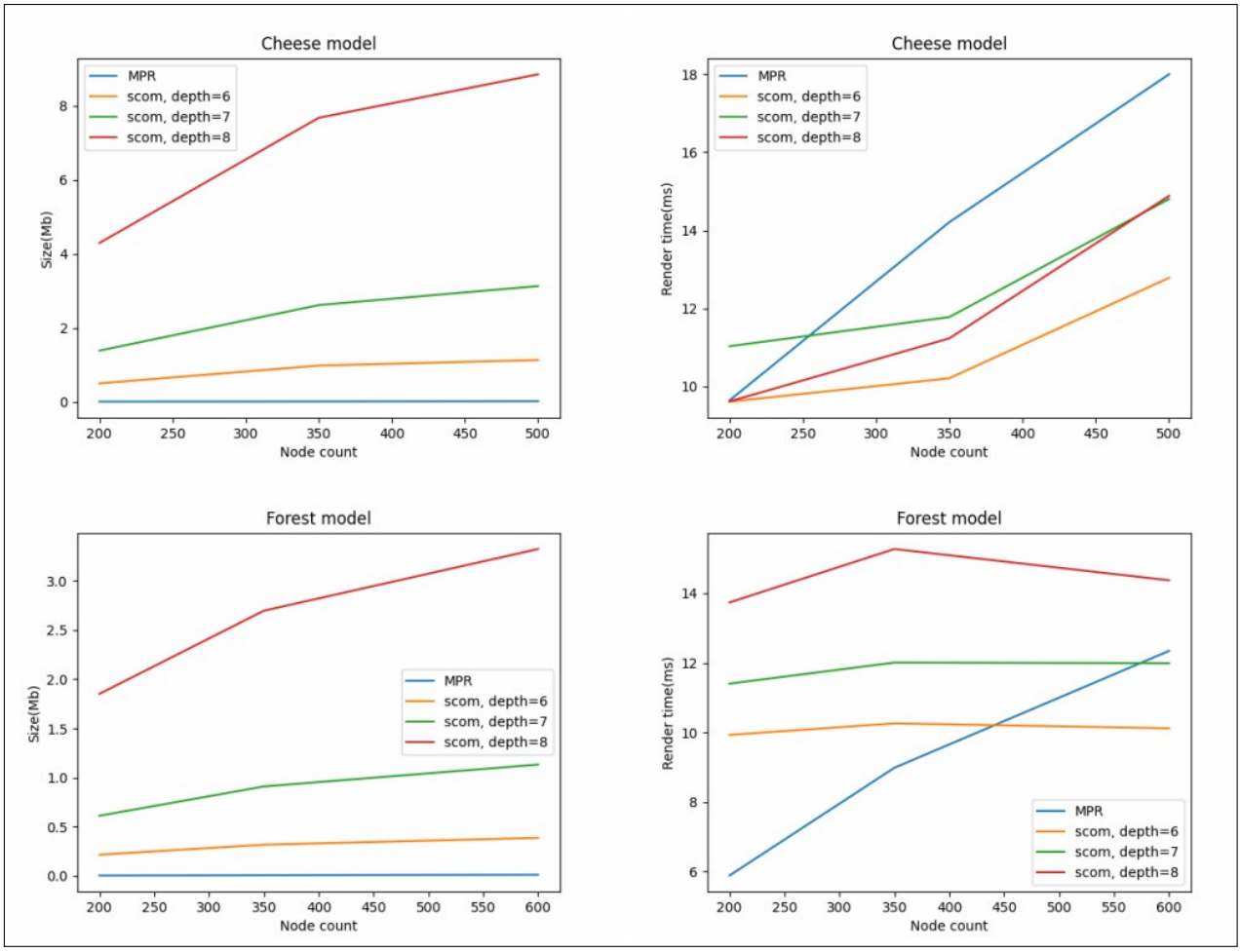

Рис. 9: Сравнение нашего метода с MPR. Сравнения слева основаны на памяти, а справа на времени рендеринга.

Scene	# Prims	Tree Depth	Intel 4000		Radeon 7870		GTX 680			
			–	+	–	+	–	+	OpenSG	IceSL
City	3385	14	7	7.5	50	60	51	57		
	343K	22	1.8	4.5	6.5	17	8	22		
	987K	24	2.3	7	6.7	18	8.3	21		
Cheese	502	10	1.2	31	7.5	193	11.2	211	21	1.1
	1002	11	0.4	17	4.6	110	5.8	128	6.5	0.3
	8002	14	N/A	6.5	0.5	28	0.5	32	N/A	N/A
	32002	17	N/A	0.5	N/A	3.7	N/A	4	N/A	N/A
Satellites	87.5K	7	5	9	26	67	29	65		
	1120K	7	2.8	4.5	8	18	7	15		

Рис. 10: Таблица со сравнениями из статьи [Bogolepov 2016] показывает, что предложенный может обрабатывать огромное количество примитивов, в то время как MPR не может [Рис. 9].

6 Заключение

В ходе работы были получены следующие результаты:

- Проведен обзор существующих работ по представлениям функций дистанции со знаком.
- На основе выбранных методов был реализован бенчмарк, позволяющий сравнивать разные методы по нескольким метрикам, при желании можно без проблем добавить новые варианты сравнения.
- Полученная программа при необходимости способна переводить код с C++ на графический процессор с целью ускорения выполнения вычислений.
- Подготовлен ряд сцен для демонстрации достоинств и недостатков каждого из методов.
- Проведено сравнение выбранных методов, замерены такие показатели, как время рендера, размер модели, качество полученных изображений.

7 Список литературы