УДК 378:004
DOI: 10.25686/978-5-8158-2474-4-2025-94-104

# SDFMultiOctree: неявное представление поверхностей на основе функций расстояния со знаком

**Н. Д. Майоров[1], А. Р. Гарифуллин[2], А. Г. Волобой[2], В. А. Фролов[1]**
[1]Институт перспективных исследований проблем искусственного интеллекта и интеллектуальных систем МГУ им. М.В. Ломоносова, г. Москва, Россия
[2]Институт прикладной математики им. М.В. Келдыша РАН, г. Москва, Россия

***Аннотация.*** Авторами предлагается метод представления поверхностей на основе октодеревьев и функций расстояния со знаком (Signed Distance Function, SDF). В основе данного метода лежат два принципиальных нововведения. Во-первых, мы комбинируем несколько независимых SDF в одном узле октодерева, что позволяет представлять более сложные элементы модели без углубления октодерева. Во-вторых, мы помечаем отдельные SDF как тонкие, что дает возможность представлять некоторые элементы моделей более точно, чем в существующих методах на основе SDF. Предложенный метод превосходит базовый по точности, сохраняя как скорость рендеринга, так и потребляемый объём памяти.

***Ключевые слова*:** функция расстояния со знаком, трассировка лучей, неявное представление.

# SDFMultiOctree: implicit representation of surfaces based on signed distance functions

**N. D. Mayorov[1], A. R. Garifullin[2], A. G. Voloboy[2], V. A. Frolov[1]**
[1]IAI Moscow State University, Moscow, Russia
[2]Keldysh Institute of Applied Mathematics, Moscow, Russia

***Abstract.*** This paper proposes a method for surface representation based on octrees and signed distance functions (SDF). The method is founded on two key innovations. First, we combine multiple independent SDFs within a single octree node, enabling the representation of more complex model elements without deepening the octree. Second, we mark certain SDFs as thin, which allows us to represent specific model components more accurately than existing SDF-based methods. The proposed approach outperforms the baseline method in terms of accuracy while maintaining both rendering speed and memory efficiency.

***Keywords*:** signed distance function, ray tracing, implicit representation.

## Введение

В последние годы наблюдается рост популярности функций расстояния со знаком (SDF) не только благодаря их применению в генеративных моделях [1–3], задачах 3D-реконструкции [4–6], но и за счет эффективной трассировки лучей по SDF-представлениям в различных приложениях компьютерной графики [7, 8]. SDF позволяют компактно и гибко описывать сложные формы и топологии, что особенно важно для современных приложений, требующих высокой детализации и реалистичности.

В качестве базового метода хранения и рендеринга разреженных SDF-данных широко используются структуры типа Sparse Voxel Octree (SVO) [9, 10]. В данной работе предлагается модификация SVO, предназначенная для хранения вокселей с интерполированной SDF внутри, что расширяет возможности базового метода при работе с моделями, не удовлетворяющими традиционным ограничениям: открытые поверхности, неидеальная топология, тонкие детали. Наш подход позволяет корректно описывать и визуализировать такие объекты, делая трассировку лучей по SDF более универсальным инструментом для рендеринга сложной геометрии в реальном времени.

## Обзор работ

Наиболее близкими к нашему методу являются структуры, сочетающие преимущества октодерева и локальных представлений SDF, а также современные нейросетевые методы.

*1. Классические структуры*

Один из наиболее популярных подходов для хранения разреженных воксельных данных – Sparse Voxel Octree (SVO) [9, 10]. SVO хранит информацию только о занятых областях, а данные о вокселях часто агрегируются на уровне родителей для экономии памяти. Современные модификации SVO, например [11], позволяют хранить в листьях коэффициенты полиномиальной интерполяции SDF или увеличивать объём пустых областей на этапе предобработки. Такой подход дает возможность более

гибко подходить к хранению данных на границе объектов и использовать различные методы интерполяции, однако требует решения аналитических задач поиска корней, что может быть вычислительно затратно. Предложенные в данной статье модификации позволяют улучшить как раз подобные решения в ряде ситуаций, когда воксель не может содержать поверхность оригинальной модели. В ряде задач реконструкции геометрии, где модель постоянно меняется, для эффективного обновления данных вместо октодерева применяются хеш-таблицы [12, 13].

Другим заметным решением является Sparse Brick Set (SBS) [7], в котором используется комбинация BVH и регулярных воксельных решёток (bricks) внутри листьев дерева. Такой подход позволяет эффективно вычислять пересечения лучей с SDF с помощью аналитических и численных методов (например, метода Ньютона), используя значения SDF на углах вокселей, которые должны явно храниться в нужных точках.

Структура VDB [14] реализует иерархические многоуровневые решётки, которые обеспечивают балансировку дерева и ускоряют её обход. VDB хранит битовые маски активных вокселей, но не сжимает значения SDF, а сами иерархические решётки в целом характеризуются невысокой производительностью при рендеринге. Оптимизированная реализация трассировки лучей по VDB-структурам на GPU была предложена в [15], где для обхода используется 3D-дифференциальный анализатор.

*2. Нейросетевые представления*

В последние годы активно развиваются методы, использующие нейронные сети для моделирования SDF. Архитектура SIREN [16] использует периодические функции активации для улучшения гладкости и качества неявных представлений. DeepSDF [17] предлагает обучать нейронную сеть для восстановления непрерывной SDF по латентному вектору, кодирующему форму.

NGLOD [18] сочетает SVO и нейросетевой декодер: на каждом уровне октодерева хранятся векторные признаки на углах вокселей, которые вместе с координатами подаются в компактную нейросеть-декодер для оценки SDF. Это позволяет ускорять трассировку лучей без существенной потери качества и поддерживать несколько уровней детализации. Аналогичный подход реализован в CoFie [19], где форма разбивается на локальные патчи, каждый из которых моделируется отдельной нейросетью.

Интересное решение предложено в N-BVH [20], где нейросеть непосредственно предсказывает точку пересечения луча с поверхностью, используя хеш-энкодер. Такой подход обеспечивает высокую степень сжатия для сложных сцен, но уступает по качеству для простых объектов и требует гибридизации с классическими структурами для достижения баланса между качеством и эффективностью.

Также эффективным и популярным методом является Instant Neural Graphics Primitives (Instant NGP) [12], в котором используется многорезолюционный хеш-энкодер для компактного представления SDF и других неявных функций. Такой подход обеспечивает высокую скорость обучения и визуализации, а также масштабируемость для сложных сцен и различных графических задач.

*3. Особенности и ограничения базового метода*

В качестве базового метода для представления разреженных SDF-данных в настоящей работе выбран Sparse Voxel Octree (SVO) [9, 10]. SVO широко применяется благодаря своей компактности, способности адаптировать плотность разбиения к локальной сложности геометрии и эффективной поддержке операций поиска и трассировки лучей. Иерархическая структура октодерева позволяет автоматически повышать детализацию в областях с мелкими деталями, при этом экономя память за счет пропуска пустых регионов. Благодаря этим преимуществам, SVO стал де-факто стандартом для хранения и рендеринга сложных трёхмерных сцен с использованием SDF.

Однако классический подход к построению октодерева обладает рядом ограничений, которые становятся критичными в современных приложениях. Во-первых, для корректной работы требуется строго замкнутая поверхность модели. В противном случае в местах разрывов могут появляться заметные артефакты при визуализации. Во-вторых, для корректного отображения тонких элементов, ширина которых меньше размера вокселя, требуется локальное увеличение глубины дерева, что приводит к росту объёма данных и снижению производительности. Эта проблема также иллюстрируется на рисунке 1, а возникающие на практике артефакты заметны на рисунке 2. Кроме того, если модель

состоит из нескольких независимых частей (рис. 3), стандартный подход может некорректно заполнять пространство между ними, что ухудшает точность представления сложных объектов.

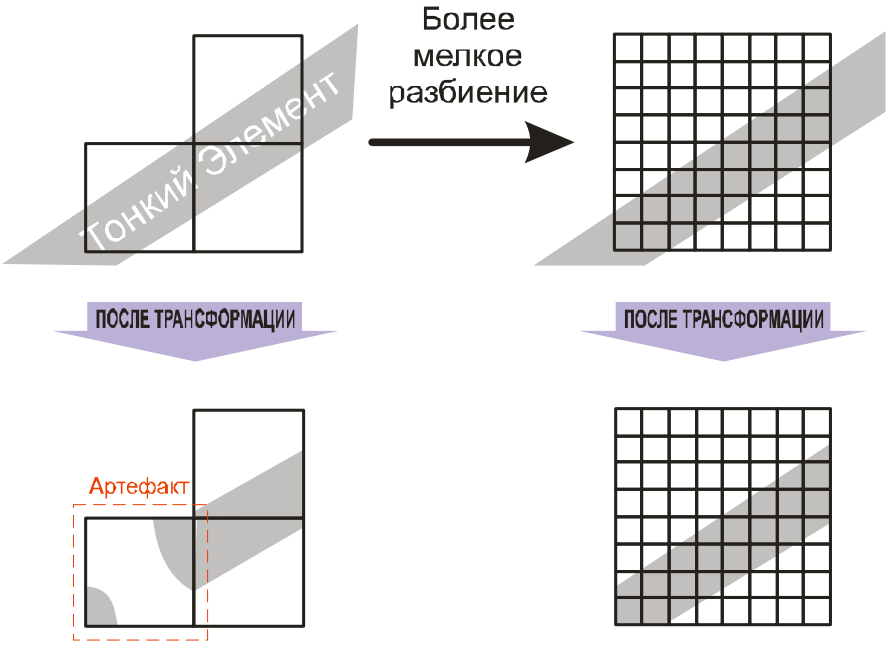


Рисунок 1. *Слева*: иллюстрация артефактов, возникающих при наличии тонких элементов у модели (из-за линейной интерполяции элемент модели может стать «дырявым» или вовсе пропасть). *Справа*: наивный способ решения этой проблемы – локальное увеличение глубины октодерева

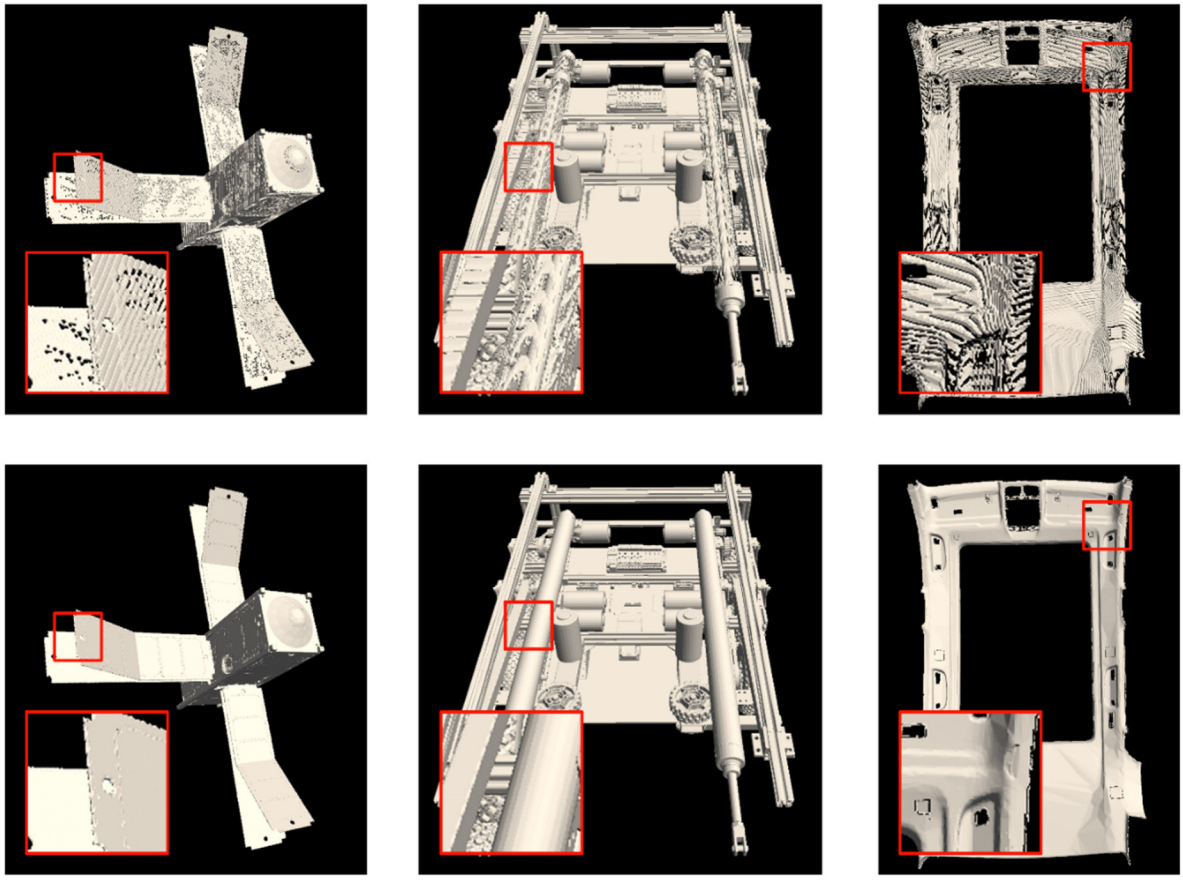

Рисунок 2. *Сверху*: октодеревья глубины 9, построенные базовым методом по моделям (слева направо) ABC 80006, ABC 83870 и ABC 515447. *Снизу*: октодеревья глубины 9, построенные нашим методом по тем же моделям. На этих моделях отчётливо видно артефакты, которые образуются при попытке приблизить тонкие элементы геометрии

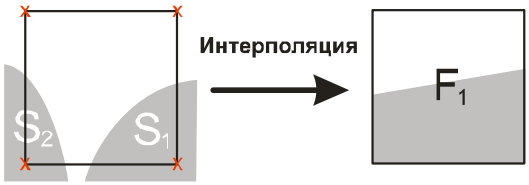


Рисунок 3. Иллюстрация артефактов, возникающих из-за близкого расположения разных частей модели (на расстоянии меньшем, чем размер вокселя)

**Предложенные модификации**

Для устранения выявленных недостатков и повышения гибкости структуры были разработаны несколько модификаций базового алгоритма. Эти изменения направлены на улучшение качества аппроксимации поверхности, особенно в случаях наличия тонких элементов и незамкнутых поверхностей, однако предложенные изменения требуют добавления небольшого количества дополнительной информации в узлы октодерева.

*1. Множественные узлы*

Основная идея первой модификации заключается в том, что каждый узел октодерева может содержать не один, а несколько вокселей данных, каждый из которых соответствует отдельной части модели, находящейся в пределах данного региона пространства, как на рисунке 4. Такой подход позволяет в рамках одного узла описывать сложные случаи, например, резкие стыки между поверхностями или несколько не связанных между собой фрагментов объекта, которые не могут быть корректно аппроксимированы одним вокселем. Для построения такой структуры необходимо определить, какие полигоны исходной модели относятся к какой компоненте внутри узла. В случае мешей это реализуется разбиением пересекающихся с узлом полигонов на связные компоненты, для каждой из которых строится отдельный воксель. Если в исходной модели присутствует информация о принадлежности полигонов к частям объекта, её также можно использовать для инициализации множественных вокселей, хотя на практике эта информация не всегда бывает полной или корректной.

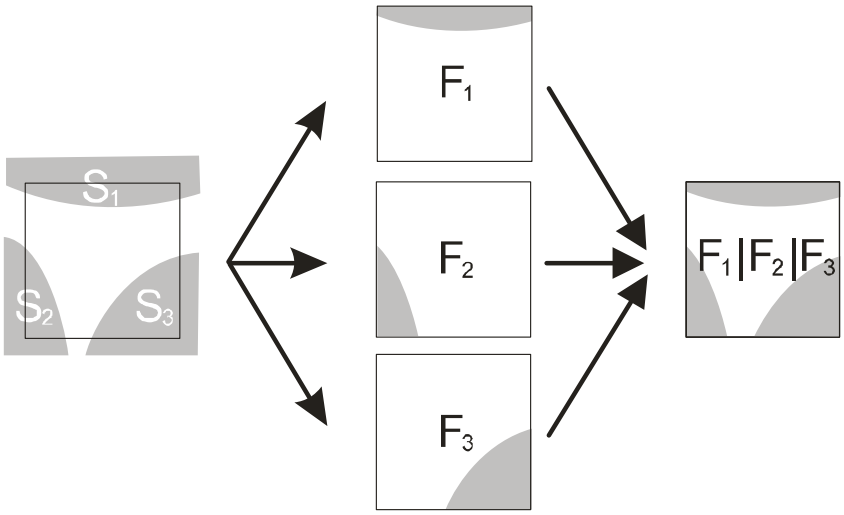


Рисунок 4. Идея «множественного узла»: *слева*: регион пространства, в котором находятся несколько независимых поверхностей; *посередине*: воксели, построенные по поверхностям, составляющие множественный узел; *справа*: множественный узел

Для эффективного хранения данных в каждом узле дополнительно сохраняется число, определяющее количество вокселей в данном регионе. На рисунке 5 показано соответствующее поле в структуре узла октодерева. Для ограничения объёма памяти и предотвращения избыточного разбиения на практике мы ввели верхний предел на число вокселей в одном узле (например, 8–10). Превышение этого значения обычно связано с недостатками алгоритма выделения компонент, поэтому такие случаи желательно отслеживать и корректировать на этапе построения дерева. Для сокращения количества вокселей в узле нужно уметь объединять несколько вокселей в один и убирать воксели, которые не влияют на внешний вид модели. На рисунке 6 схематично изображено, как это происходит.

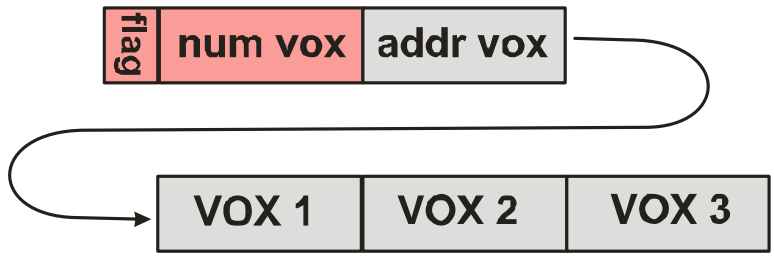


Рисунок 5. Модифицированная структура листа в октодереве. Красным выделены добавленные поля, необходимые для описания поверхностных представлений и множественных узлов

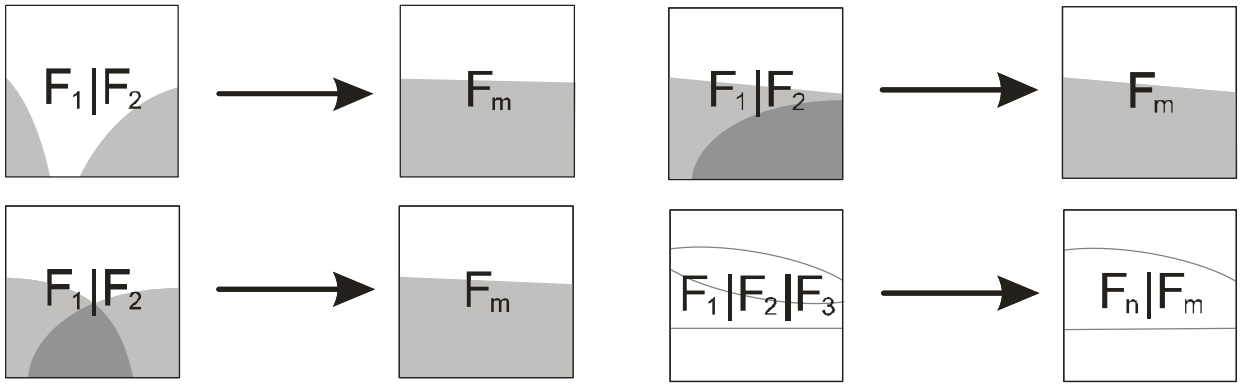


Рисунок 6. Алгоритм уменьшения количества вокселей, при необходимости уменьшения их количества:
*слева:* для объёмных вокселей с потерей качества;
*справа наверху*: для объёмных вокселей без потери качества;
*справа внизу*: для поверхностных вокселей

*2. Поверхностные узлы*

Вторая модификация направлена на расширение возможностей представления незамкнутых поверхностей и тонких элементов модели. В рамках этого подхода допускается хранение вокселей, описывающих не объём, а только поверхность: значения функции расстояния в таком вокселе определяют лишь разделение пространства на два полупространства относительно поверхности, без явного выделения внутренней и внешней областей. На рисунке 7 продемонстрированы главные отличия таких узлов от обычных. Это особенно важно для моделей, не являющихся строго замкнутыми. А при комбинированной работе двух предложенных модификаций это может быть использовано для корректного отображения тонких элементов, ширина которых меньше размера вокселя. В базовой реализации такие элементы часто теряются, что приводит к появлению дыр в модели. Это можно понять по схеме, представленой на рисунке 7. В свою очередь, на рисунке 2 можно увидеть, как наш метод на практике справляется с такими артефактами.

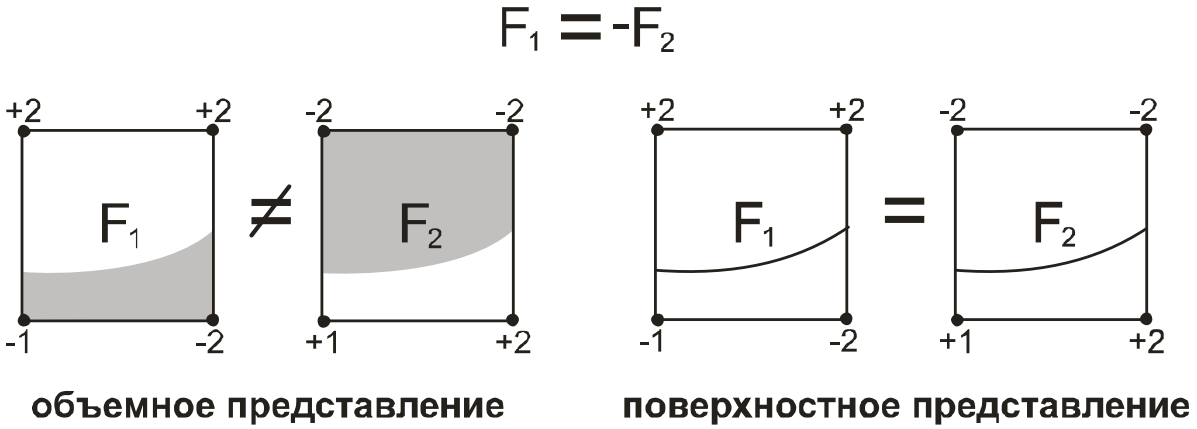


Рисунок 7. Демонстрация различий между стандартным объёмным представлением вокселя и поверхностным представлением

Однако полностью переходить к поверхностным узлам нецелесообразно: при недостаточной точности аппроксимации могут возникать щели между вокселями, как на рисунке 8. Поэтому в предложенной структуре поддерживается одновременное хранение как объёмных, так и поверхностных вокселей. Для этого в каждом узле дополнительно сохраняется флаг, определяющий тип вокселей, что позволяет гибко комбинировать оба подхода в рамках одной структуры.

На рисунке 5 можно увидеть соответствующую модификацию в структуре октодерева. В результате достигается более корректное представление сложных и разнородных объектов, а также существенно расширяется область применимости метода для моделей с дефектами топологии и открытыми поверхностями.

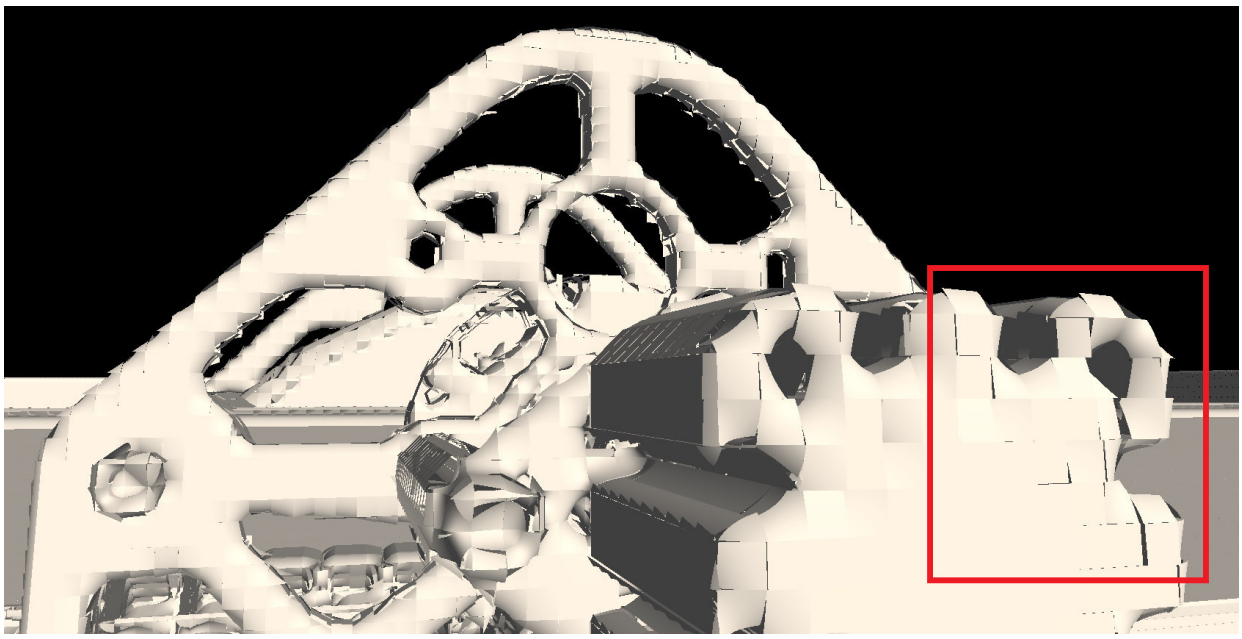

Рисунок 8. Пример модели, где все воксели считаются поверхностными, но была немного уменьшена точность вычисления значений в них. За счёт этого возникают артефакты в виде «щелей» между вокселями

**Сравнения**

*1. Сравнение с октодеревьями*

В этом разделе будет произведено сравнение с методом, который строит обычные разреженные октодеревья [9, 10]. Мы сравнили базовый метод с предложенным на нескольких моделях из ABC датасета [21, 22], используя разные параметры глубины. На рисунках 9 и 10 продемонстрированы графики, иллюстрирующие зависимость среднего FLIP и PSNR для нескольких ракурсов от размера полученного октодерева для конкретных моделей из датасета. Чем больше PSNR и чем меньше размер октодерева, тем, соответственно, лучше результат. Как видно из графиков, предложенный метод даёт лучший результат и позволяет получить больший PSNR, используя меньшую глубину октодерева и получая при этом меньшую модель по размеру. В таблице 1 тоже представлены данные для конкретной модели с октодеревьями разной глубины.

В таблице 2 приведены данные для нескольких моделей с зафиксированной глубиной, равной 9. Как видим, предложенные модификации повышают качество модели, но при этом и увеличивают её размер. Однако, как показывает анализ представленных выше графиков, увеличение глубины октодерева для базового метода даёт меньший прирост в качестве и больший прирост в размере, что доказывает преимущество нашего метода.

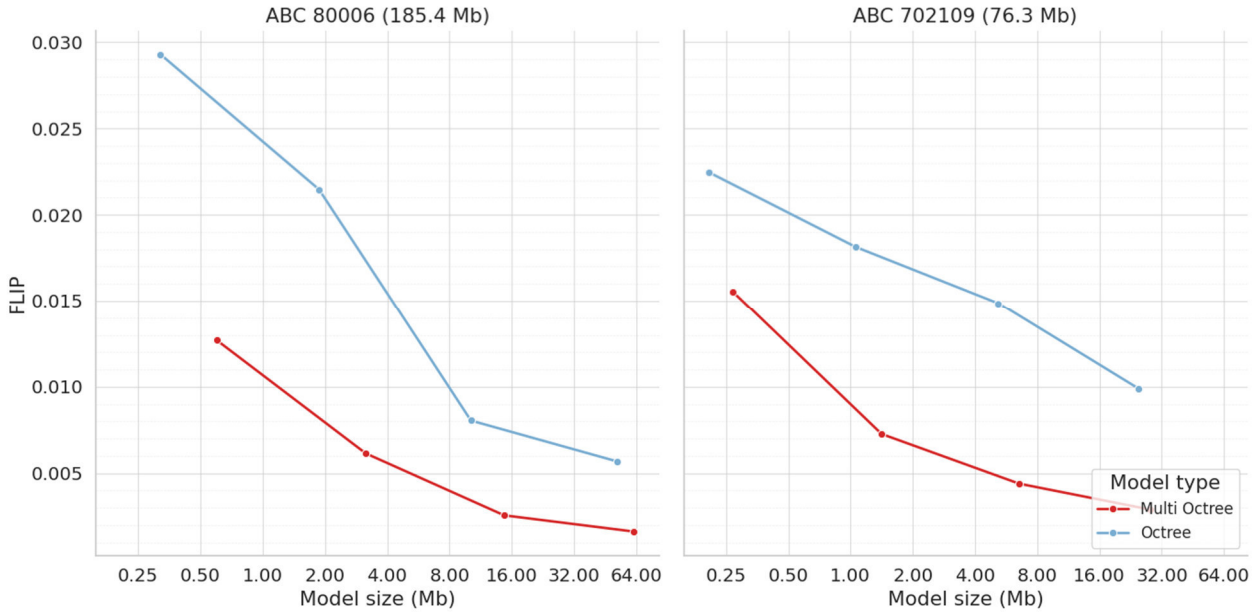


Рис. 9. График сравнения базового октодерева с нашим методом по FLIP и размеру октодеревьев разной глубины. Чем график ближе к левому нижнему углу, тем метод более качественный

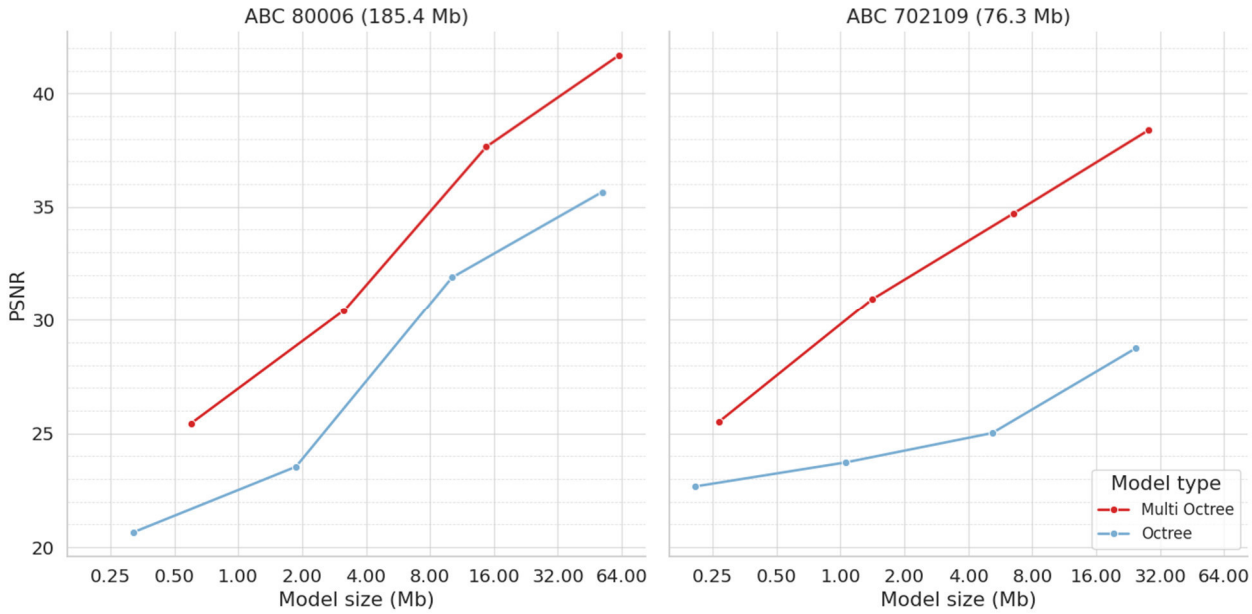


Рис. 10. График сравнения базового октодерева с нашим методом по PSNR и размеру октодеревьев разной глубины. Чем график ближе к левому верхнему углу, тем метод более качественный

Таблица 1. **Сравнение PSNR, FLIP, размеров октодеревьев и их времени рендеринга на модели ABC 88060 (71.06 MB) для разных настроек глубины**

CPU: Intel Core i5-8600 @ 3.10 GHz, 16 GB RAM. GPU не использовался.

| Depth | Sparse Voxel Octree | | | | Ours | | | |
|---|---|---|---|---|---|---|---|---|
| | PSNR | FLIP | Size *MB* | Time *ms* | PSNR | FLIP | Size *MB* | Time *ms* |
| 7 | 29.19 | 0.0087 | **0.36** | **96.3** | **31.51** | **0.0072** | 0.42 | 97.2 |
| 8 | 32.47 | 0.0066 | **1.69** | 104.2 | **35.36** | **0.0051** | 1.96 | **103.2** |
| 9 | 38.38 | 0.0040 | **7.83** | **108.8** | **39.59** | **0.0036** | 8.10 | 109.2 |
| 10 | 41.94 | 0.0030 | **33.46** | **114.7** | **42.82** | **0.0028** | 34.10 | 115.5 |

Таблица 2. **Сравнение PSNR, FLIP, размеров октодеревьев глубины 9 и их времени рендеринга для разных моделей из датасета, указанных на рисунке 11**

CPU: Intel Core i5-8600 @ 3.10 GHz, 16 GB RAM. GPU не использовался.

| Model | Sparse Voxel Octree | | | | Ours | | | |
|---|---|---|---|---|---|---|---|---|
| | PSNR | FLIP | Size *MB* | Time *ms* | PSNR | FLIP | Size *MB* | Time *ms* |
| ABC 80006 (185.36) | 31.89 | 0.0080 | **10.16** | **61.2** | **37.65** | **0.0026** | 14.71 | 64.1 |
| ABC 88060 (71.06) | 38.38 | 0.0040 | **7.83** | **108.8** | **39.59** | **0.0036** | 8.10 | 109.2 |
| ABC 88828 (14.43) | 22.73 | 0.0262 | **3.10** | **76.2** | **29.47** | **0.0093** | 6.92 | 85.2 |
| ABC 83870 (149.35) | 28.14 | 0.0150 | **22.93** | 84.9 | **33.04** | **0.0086** | 25.02 | **80.6** |
| ABC 515447 (57.43) | 28.52 | 0.0120 | **4.38** | 96.4 | **33.79** | **0.0055** | 5.47 | **90.0** |
| ABC 687231 (76.30) | 30.31 | 0.0101 | **17.19** | 101.6 | **30.78** | **0.0097** | 22.58 | **97.9** |
| ABC 701584 (55.15) | 34.74 | 0.0050 | **7.47** | 70.9 | **35.23** | **0.0041** | 8.94 | **70.2** |
| ABC 702109 (76.27) | 25.01 | 0.0149 | **5.18** | 93.1 | **34.71** | **0.0044** | 6.53 | **83.6** |

*2. Сравнение с нейросетевым методом*

Проведем сравнение с методом NGLOD [18], так как это интересный современный метод, сочетающий в себе идею октодеревьев и нейросетевой декодер. Произведём сравнение на тех же моделях из того же датасета [21, 22], используя разные параметры глубины для нашего метода и разные параметры детализации для NGLOD. В таблице 3 представлены данные для конкретной модели с разными настройками. Очевидно, что при увеличении размера нейросетевой модели, качество увеличивается недостаточно. Модель меньшего размера, построенная нашим методом, показывает более качественный результат.

Таблица 3. **Сравнение PSNR, FLIP, размеров и времени рендеринга моделей нашего метода с моделями нейросетевого метода NGLOD [18] на модели ABC 88060 (71.06 *MB*) для разных настроек методов**

CPU: Intel Core i5-8600 @ 3.10 GHz, 16 GB RAM. GPU не использовался.

| Depth / Detail | NGLOD | | | | Ours | | | |
|---|---|---|---|---|---|---|---|---|
| | PSNR | FLIP | Size *MB* | Time *ms* | PSNR | FLIP | Size *MB* | Time *ms* |
| 7 / 2 | 27.28 | 0.0152 | **0.14** | 719.2 | **31.51** | **0.0072** | 0.42 | **97.2** |
| 8 / 3 | 30.06 | 0.0104 | **0.76** | 1014.1 | **35.36** | **0.0051** | 1.96 | **103.2** |
| 9 / 4 | 29.52 | 0.0105 | **5.16** | 1242.4 | **39.59** | **0.0036** | 8.10 | **109.2** |
| 10 / 5 | 33.21 | 0.0069 | 38.71 | 1623.3 | **42.82** | **0.0028** | **34.10** | **115.5** |

В таблицах 4 и 5 представлены данные для нескольких моделей с зафиксированной глубиной, равной 9, и зафиксированной детализацией, так чтобы размер нейросетевой модели не был значительно больше наших моделей. Как можно наблюдать из этой таблицы, предложенный метод даёт лучшее качество в сравнении с NGLOD, при том имея меньший размер. Скорость рендеринга также значительно выше у нашего метода. Из анализа данных таблицы 3 можно сделать вывод, что нейросетевой метод даёт довольно хорошие результаты при необходимости экстремального сжатия.

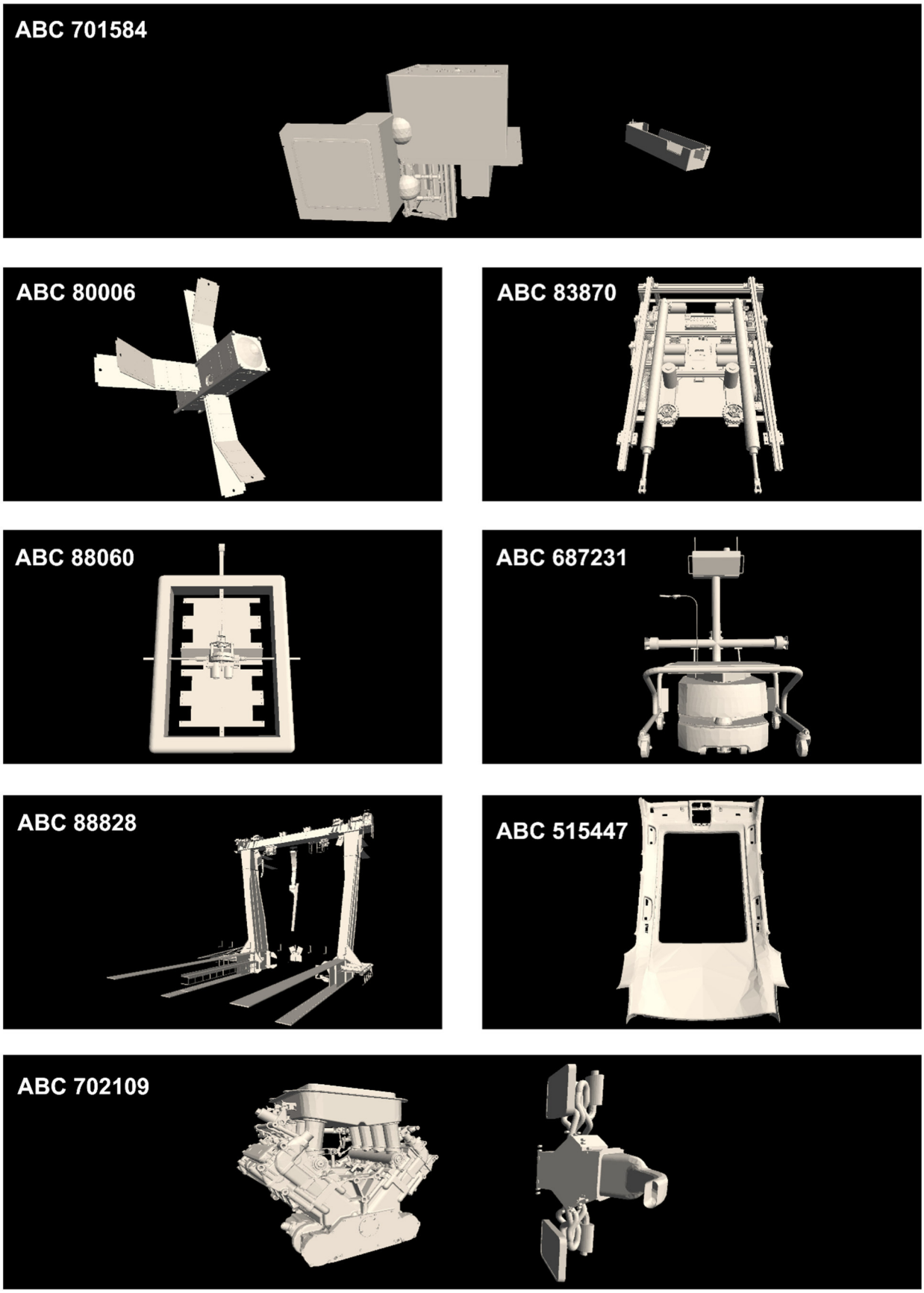


Рис. 11. Модели из датасета [21, 22], на которых производились сравнения

Таблица 4. **Сравнение PSNR, FLIP, размеров и времени рендеринга октодеревьев глубины 9 и моделей, полученных с помощью метода NGLOD [18] со сравнимым размером, для разных моделей из датасета [21, 22], указанных на рисунке 11**

CPU: Intel Core i5-8600 @ 3.10 GHz, 16 GB RAM. GPU не использовался.

| Model | NGLOD | | | | Ours | | | |
|---|---|---|---|---|---|---|---|---|
| | PSNR | FLIP | Size *MB* | Time *ms* | PSNR | FLIP | Size *MB* | Time *ms* |
| ABC 80006 (185.36) | 25.47 | 0.0128 | **5.163** | 942.3 | **37.65** | **0.0026** | 14.71 | **64.1** |
| ABC 88060 (71.06) | 29.52 | 0.0105 | **5.163** | 1242.4 | **39.59** | **0.0036** | 8.10 | **109.2** |
| ABC 88828 (14.43) | 23.46 | 0.0209 | **5.163** | 1561.1 | **29.47** | **0.0093** | 6.92 | **85.2** |
| ABC 83870 (149.35) | 22.41 | 0.0322 | **5.163** | 1623.1 | **33.04** | **0.0086** | 25.02 | **80.6** |
| ABC 515447 (57.43) | 29.91 | 0.0090 | **5.163** | 1528.1 | **33.79** | **0.0055** | 5.47 | **90.0** |
| ABC 687231 (76.30) | 27.42 | 0.0141 | **5.163** | 1461.1 | **30.78** | **0.0097** | 22.58 | **97.9** |
| ABC 701584 (55.15) | 27.25 | 0.0135 | **5.163** | 1311.2 | **35.23** | **0.0041** | 8.94 | **70.2** |
| ABC 702109 (76.27) | 28.69 | 0.0097 | **5.163** | 1817.7 | **34.71** | **0.0044** | 6.53 | **83.6** |

Таблица 5. **Сравнение PSNR, FLIP, размеров и времени рендеринга октодеревьев глубины 9 и моделей, полученных с помощью метода NGLOD [18] с не меньшим размером, для разных моделей из датасета [21, 22], указанных на рисунке 11**

CPU: Intel Core i5-8600 @ 3.10 GHz, 16 GB RAM. GPU не использовался.

| Model | NGLOD | | | | Ours | | | |
|---|---|---|---|---|---|---|---|---|
| | PSNR | FLIP | Size *MB* | Time *ms* | PSNR | FLIP | Size *MB* | Time *ms* |
| ABC 80006 (185.36) | 26.69 | 0.0110 | 38.71 | 1104.7 | **37.65** | **0.0026** | **14.71** | **64.1** |
| ABC 88060 (71.06) | 33.21 | 0.0069 | 38.71 | 1623.3 | **39.59** | **0.0036** | **8.10** | **109.2** |
| ABC 88828 (14.43) | 25.65 | 0.0163 | 38.71 | 1758.7 | **29.47** | **0.0093** | **6.92** | **85.2** |
| ABC 83870 (149.35) | 24.70 | 0.0243 | 38.71 | 2083.5 | **33.04** | **0.0086** | **25.02** | **80.6** |
| ABC 515447 (57.43) | 30.12 | 0.0087 | 38.71 | 1723.8 | **33.79** | **0.0055** | **5.47** | **90.0** |
| ABC 687231 (76.30) | 28.35 | 0.0132 | 38.71 | 1735.6 | **30.78** | **0.0097** | **22.58** | **97.9** |
| ABC 701584 (55.15) | 28.53 | 0.0121 | 38.71 | 1412.4 | **35.23** | **0.0041** | **8.94** | **70.2** |
| ABC 702109 (76.27) | 29.41 | 0.0090 | 38.71 | 1897.1 | **34.71** | **0.0044** | **6.53** | **83.6** |

**Выводы**

Авторы данной статьи предлагают модификации базового метода разреженных октодеревьев. Эти изменения позволяют хранить в каждом узле дерева несколько поверхностей вместо одной в базовом алгоритме. Также появляется возможность представлять элементы модели не только как объёмные узлы, но и как поверхности. Эксперименты показывают, что такие модификации немного повышают размер итоговой модели, однако при этом значительно повышают их качество, а время рендеринга изменённых моделей при этом сравнимо с рендерингом базовых моделей.

**Список литературы**

1. G. Chou, Y. Bahat and F. Heide, "Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)", Diffusion-SDF: Conditional Generative Modeling of Signed Distance Functions, Paris, France, 2023.

2. L. Yariv, O. Puny, N. Neverova, O. Gafni and Y. Lipman, "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)", Mosaic-SDF for 3D Generative Models, Seattle, WA, USA, 2024.

3. J. Shim, C. Kang and K. Joo, "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)", Diffusion-Based Signed Distance Fields for 3D Shape Generation, Vancouver, BC, Canada, 2023.

4. C.-H. Lin, C. Wang and S. Lucey, "SDF-SRN: Learning Signed Distance 3D Object Reconstruction from Single-View Images", arXiv preprint arXiv:2010.10505, 2020.

5. C. Sommer, L. Sang, D. Schubert and D. Cremers, "Gradient-SDF: A Semi-Implicit Surface Representation for 3D Reconstruction", arXiv preprint arXiv:2111.13652, 2022.

6. M. Slavcheva, W. Kehl, N. Navab and S. Ilic, "Proceedings of the European Conference on Computer Vision (ECCV)", SDF-2-SDF: Highly Accurate 3D Object Reconstruction, Amsterdam, The Netherlands, 2016.

7. H. Hansson-Söderlund, A. Evans and T. Akenine-Möller, "Ray Tracing of Signed Distance Function Grids", Journal of Computer Graphics Techniques, vol. 11, no. 3, p. 94–113, 2022.

8. A. R. Garifullin, V. A. Frolov, A. S. Budak and V. A. Galaktionov, "Study of Surface Representation Methods Based on Signed Distance Functions", Programming and Computer Software, vol. 51, no. 1, pp. 131-139, 2025.

9. C. Crassin, F. Neyret, S. Lefebvre and E. Eisemann, "Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)", GigaVoxels: ray-guided streaming for efficient and detailed voxel rendering, Boston, Massachusetts, 2009.

10. S. Laine and T. Karras, "Proceedings of the 2010 Symposium on Interactive 3D Graphics and Games (I3D '10)", в Efficient sparse voxel octrees, New York, NY, USA, 2010.

11. E. Pujol and A. Chica, "Rendering piecewise approximations of SDFs through analytic intersections", Computers & Graphics, vol. 122, no. C, p. 9, November 2024.

12. T. Müller, A. Evans, C. Schied and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding", ACM Transactions on Graphics, vol. 41, no. 4, pp. Article 102, 1–15, July 2022.

13. M. Nießner, M. Zollhöfer, S. Izadi and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing", ACM Transactions on Graphics, vol. 32, no. 6, pp. Article 169, 1–11, November 2013.

14. K. Museth, "VDB: High-resolution sparse volumes with dynamic topology", ACM Transactions on Graphics, vol. 32, no. 3, pp. Article 27, 1–22, July 2013.

15. R. K. Hoetzlein, "Proceedings of High Performance Graphics (HPG '16)", GVDB: raytracing sparse voxel database structures on the GPU, Dublin, Ireland, 2016.

16. V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell and G. Wetzstein, "Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)", Implicit neural representations with periodic activation functions, Red Hook, NY, USA, 2020.

17. J. J. Park, P. R. Florence, J. Straub, R. A. Newcombe and S. Lovegrove, "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, 2019.

18. T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire and S. Fidler, "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)", Neural Geometric Level of Detail: Real-Time Rendering with Implicit 3D Shapes, 2021.

19. H. Jiang, H. Yang, G. Pavlakos and Q. Huang, "CoFie: Learning Compact Neural Surface Representations with Coordinate Fields", arXiv preprint arXiv:2406.03417, 2024.

20. P. Weier, A. Rath, É. Michel, I. Georgiev, P. Slusallek and T. Boubekeur, "ACM SIGGRAPH 2024 Conference Papers (SIGGRAPH '24)", N-BVH: Neural ray queries with bounding volume hierarchies, Denver, CO, USA, 2024.

21. S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin and D. Panozzo, "ABC Dataset", Доступно под MIT лицензией, 2019.

22. S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin and D. Panozzo, "ABC: A Big CAD Model Dataset For Geometric Deep Learning", arXiv preprint arXiv:1812.06216, 2019.