Оптимизация многоуровневой ускоряющей структуры пространственного разбиения хеш-таблиц и kd-дерева

А.И. Лысых [1], И.Е. Кинев [1], Д.Д. Жданов [1]

[1] *ИТМО, Кронверкский пр-кт, д. 49, лит. А, Санкт-Петербург, 197101, Российская Федерация*

Аннотация

Производится анализ существующих типов ускоряющих структур пространственного разбиения. Рассматриваются свойства и особенности исходной комбинированной структуры пространственного разбиения, построенной на основе хеш-таблиц и kd-дерева. Выделяются три типа случаев применения структуры: динамический, сбалансированный и статический. Предлагаются методы оптимизации алгоритмов построения структур и поиска ячеек. Для трёх типов применения структуры предлагаются алгоритмы определения размера и уровней расположения хеш-таблиц. Производится анализ оптимальных значений параметров этих алгоритмов для каждого из случаев применения. Выполняется сравнение и анализ эффективности модифицированной структуры с исходной комбинированной структурой и kd-деревом. Сравнение показало, что полученная структура может значительно повысить эффективность поиска ячеек для рассмотренных типов применения структуры.

Ключевые слова

Структуры пространственного разбиения, оптимизация, рендеринг, фотонные карты, трассировка лучей.

Optimization of Multilevel Spatial Partitioning Accelerating Structure of Hash Tables and Kd-tree

A.I. Lysykh [1], I.E. Kinev [1], D.D. Zhdanov [1]

[1] *ITMO, Kronverksky Pr. 49, bldg. A, St. Petersburg, 197101, Russia*

Abstract

This research conducts an in-depth analysis of existing types of spatial partitioning acceleration structures. It investigates the properties and characteristics of the original combined spatial partitioning structure, based on hash tables and kd-trees. Three types of use cases for this structure are identified: dynamic, balanced, and static. Methods for optimizing the structure construction and cell search algorithms are proposed. For each of the three usage scenarios, algorithms for determining the size and placement levels of hash tables are introduced. An analysis of the optimal parameter values for these algorithms in each usage case is performed. Furthermore, a comparative efficiency analysis of the modified structure against the original combined structure and kd-tree is carried out. The comparison reveals that the resulting structure significantly enhances cell search efficiency for the considered types of structure usage.

Keywords

Spatial partitioning structures, optimization, rendering, photon maps, ray tracing.

1. Введение

В современном мире реалистичные методы визуализации широко используются для решения большого числа прикладных задач. Примерами могут служить создание различных

ГрафиКон 2023: 33-я Международная конференция по компьютерной графике и машинному зрению, 19-21 сентября 2023 г., Институт проблем управления им. В.А. Трапезникова Российской академии наук, г. Москва, Россия
EMAIL: lysykhai@ya.ru (А.И. Лысых); igorkinevitmo@gmail.com (И.Е. Кинев); ddzhdanov@mail.ru (Д.Д. Жданов)
ORCID: 0000-0002-2437-5275 (А.И. Лысых); 0000-0003-2929-1203 (И.Е. Кинев); 0000-0001-7346-8155 (Д.Д. Жданов)

анимационных фильмов, виртуальное прототипирование различного рода оптических устройств, включая устройства виртуальной и смешанной реальности, расширение наборов данных синтезированными изображениями при обучения нейронных сетей и т. п.

Для получения физически-корректных изображений часто применяются методы расчета глобальной освещенности, основанные на трассировке лучей, которые физически корректно имитируют распространение света в сцене и позволяют учитывать такие физические явления, как преломление, отражение, диффузное рассеивание и т. п.

В общем случае данные методы вычисляют интеграл яркости для наблюдаемой точки поверхности, который рекурсивно собирает освещение по всей сцене с учетом свойств поверхности, которые задаются двунаправленной функцией рассеивания (BSDF) [1,2], и таким образом решают уравнение рендеринга [3]. Так как уравнение рендеринга представляет собой бесконечную рекурсию и является бесконечно сложным, для его решения используется метод «Русской рулетки», который позволяет за конечное время выполнить оценку значения интеграла освещенности.

Для оптимизации работы методов расчёта физически-корректных изображений основанных на трассировке лучей, чаще всего используются ускоряющие структуры пространственного разбиения, позволяющие эффективно хранить и использовать исходные данные сцены или промежуточные данные расчёта [4,5,6]. Такие данные представляют собой элементы, расположенные в трёхмерном пространстве виртуальной сцены. Этими элементами могут являться геометрические объекты, источники освещения, элементы геометрии сцены, или же виртуальные фотоны, применяемые в методах фотонных карт [7,8,9].

2. Структуры пространственного разбиения

Структуры пространственного разбиения применяются для достижения двух основных целей: это уменьшение необходимой для хранения данных памяти и обеспечение ускорения доступа к данным в процессе их поиска [4,5,6]. В общем случае структура разбивает пространство сцены или область ограничивающего параллелепипеда объекта на ячейки, в которые помещаются все элементы, находящиеся внутри или пересекающие ту или иную ячейку. Таким образом, операция перебора всех элементов сцены или объекта при поиске пересечения луча, сферы или точки с этими элементами заменяется на поиск необходимых ячеек и перебор всех элементов в этих ячейках. Это заметно сокращает скорость расчёта, поскольку ускоряющие структуры пространственного разбиения устроены таким образом, чтобы обеспечивать быстрый поиск необходимых ячеек, кроме того, количество этих ячеек, как правило, значительно меньше, чем общее количество элементов.

Существует три основных типа структур пространственного разбиения: регулярные сетки, иерархические структуры и различные комбинированные решения. Два первых типа структур имеют различные свойства, касающихся адаптивности разбиения. Это позволяет использовать преимущества этих структур для решения определенных типов задач. Третий тип является компромиссным и смещает свойства структуры в ту или иную сторону, позволяя найти более эффективное решение для конкретной задачи, за счёт уменьшения влияния недостатков исходных структур.

2.1. Регулярные структуры

Одной из самых простых структур разбиения пространства является регулярная сетка [10]. Размер и разрешение такой сетки задаются заранее и остаются фиксированным до конца вычислений. Область пространства, требующая поиска в ней элементов, разбивается на заданное количество ячеек по трём осям координат. Так как размеры всех ячеек одинаковы, а положение сетки в пространстве определено, то по координатам точки, можно быстро определить соответствующий индекс ячейки.

Для хранения такой структуры в памяти используется обычная трёхмерная матрица, это позволяет получать непосредственный доступ к любой ячейке структуры по значениям координат точки пространства, напрямую связав индекс ячейки с адресом в памяти. Для того,

чтобы уменьшить объем памяти, необходимый для хранения этой структуры, такая матрица задаётся виртуально, а пустые ячейки исключаются, за счет использования хеш-таблицы. Ключом ячейки в этом случае является индекс ячейки матрицы, вычисленный по трём ее осям координат. В качестве хеш-функции как правило используется остаток от деления индекса на размер хеш-таблицы [10].

Несмотря на высокую скорость доступа к этой структуре, она содержит один существенный недостаток, заключающийся в низкой адаптивности сетки. Поскольку разрешение сетки неадаптивное в областях с большой концентрацией элементов содержится большое число элементов, поиск в этих областях пространства замедляется. Обратная ситуация возникает в областях с низкой концентрацией элементов: в этих ячейках может содержаться по одному-двум элементов, что увеличивает необходимую для хранения структуры память. Поэтому для эффективной работы такой структуры разрешение сетки необходимо тщательно подбирать для каждого отдельного случая, однако это лишь «сглаживает» недостаток, но не устраняет его. В некоторых случаях подобрать необходимое значение просто невозможно из-за переменной концентрации элементов, в результате чего объем памяти, требуемой для хранения структуры, значительно увеличиваются.

2.2. Иерархические структуры

Иерархические ускоряющие структуры представляют собой различные деревья, где каждый узел дерева является некоторым описывающим объёмом. Примерами таких структур могут являться kd-деревья, октодеревья или BVH-деревья [12, 13, 14]. Все они представляют собой иерархически связанные ограничивающие параллелепипеды, где каждый нетерминальный узел содержит определённое число потомков-параллелепипедов. Каждый тип дерева имеет специфичную организацию, что в конечном итоге влияет его глубину. Деревья, узлы которых содержат большее число потомков, имеют меньшую глубину, и наоборот.

Основным преимуществом таких структур является адаптивность разбиения пространства. Способ их формирования позволяет равномерно распределить элементы, подлежащие поиску, по ячейкам-узлам структуры. При создании структуры задаётся предельное число элементов на узел. Далее корневой узел, представляющий собой ограничивающий параллелепипед объекта или сцены, формирует внутри себя потомков, каждый из которых повторяет операцию рекурсивно. Если дерево сбалансировано, то достигается примерно одинаковая скорость доступа к данным. Как правило деревья не сбалансированы, и это снижает скорость доступа к элементам в областях их высокой концентрации. Однако адаптивность разбиения повышается, что в итоге ускоряет процесс доступа к элементам в областях их высокой концентрации.

В сравнении с регулярной сеткой такие структуры имеют меньшую скорость доступа к ячейкам, поскольку элементы, требующие поиска, как правило содержатся только в конечных узлах, не имеющих потомков. Для поиска этих элементов требуется произвести спуск по дереву и обойти все связанные узлы от корня до необходимого узла. Для ускорения поиска в этом случае прибегают к уменьшению глубины дерева за счёт увеличения числа потомков, однако это ведёт к увеличению числа операций, необходимых для определения нужного потомка в каждом узле. Низкая скорость поиска узлов в таких структурах является их главным недостатком. Необходимость перебирать все узлы на пути от корня до искомого, накладывается на ограничение объема кэш памяти, что снижает эффективность поиска, поскольку все элементы дерева расположены в памяти хаотично, и для каждого перехода от узла к узлу требуется запрашивать новые данные. Данный негативный процесс особенно заметен при параллельной работе с деревьями на многоядерных компьютерах.

2.3. Комбинированное решение

В данной работе проводится исследование комбинированной ускоряющей структуры, основанной на kd-дереве и хеш-таблицах. Эта структура представляет собой дерево хеш-таблиц, построенное на основе ячеек, полученных путём разбиения пространства kd-деревом [15]. Комбинированный подход позволяет получить все преимущества исходных структур, снизив

влияние их недостатков. В первую очередь эта структура обеспечивает сравнимый со скоростью регулярной структуры доступ к ячейкам при поиске. Это достигается за счёт низкой глубины хеш-дерева и быстрого доступа к ячейкам через хеш-индекс. Кроме этого, сохраняется адаптивность разбиения kd-дерева, поскольку в качестве конечных ячеек этой структуры используются конечные ячейки kd-дерева. На рисунке 1 представлена схема структуры.

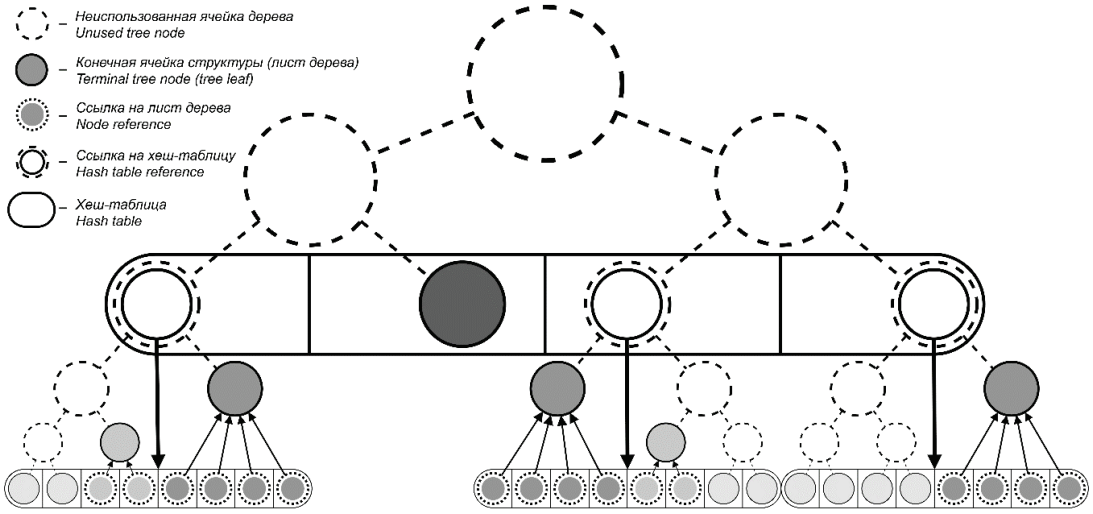

Рисунок 1 – Схема комбинированной структуры

В алгоритме построения такой структуры можно выделить два этапа: формирование kd-дерева и дальнейшее построение хеш-дерева на его основе. После выполнения построения конечной структуры исходное дерево можно удалить для уменьшения необходимой памяти, или оставить связанным со структурой, для дальнейшей её модификации при расширении дерева, чтобы избежать полного перестроения структуры.

Первым этапом происходит построение исходного kd-дерева. В этом случае можно использовать как сбалансированное, так и несбалансированное дерево. Поскольку в конечном счёте для поиска элементов исходное дерево использоваться не будет, разумно использовать несбалансированное дерево, чтобы увеличить скорость формирования структуры избежав сортировки элементов. При формировании дерева сохраняются характеристики, необходимые для поиска оптимальных уровней расположения хеш-таблиц: количество конечных ячеек, количество ячеек, содержащих элементы и общее количество ячеек. Критерием конечного узла kd-дерева является содержание в нём не более восьми элементов.

Вторым этапом происходит формирование хеш-таблиц на оптимальных уровнях начиная с таблицы-корня. В оригинальной статье [15] предлагается считать оптимальными уровни с наибольшим количеством конечных ячеек, ограничивая максимальную глубину, между соседними хеш-таблицами шестнадцатью уровнями, основываясь на данных статьи. Определение таких уровней происходит перед построением каждой следующей таблицы для текущего поддерева. Конечный уровень kd-дерева также определяется в качестве оптимального. Крайним размером хеш-таблицы было определено значение в 512 ячеек. Для таблиц, построенных на поддеревьях меньшей глубины, размеры хэш-таблицы уменьшаются, что позволяет компенсировать уменьшение количества узлов на уровне kd-дерева.

Всё ячейки пространства, покрытые хеш-таблицей, формируют регулярную сетку, для этого необходимо рассчитать и сохранить размер ячеек и разрешение сетки для каждой таблицы. Это позволяет осуществлять поиск ячеек по индексу. После создания таблицы происходит формирование её ячеек, соответствующих узлам kd-дерева, которые находятся на уровне или выше хеш-таблицы, ограничиваясь ближайшей таблицей-предком. Для этого происходит поиск ближайшего правого листа у каждого последующего листа, начиная с первого левого, при этом максимальная глубина листа ограничивается текущим уровнем kd-дерева, на котором строится хеш-таблица. При формировании ячеек рассматриваются четыре основных типа узлов kd-дерева:

- Пустой лист дерева — в этом случае формирование ячейки не происходит;
- Узел не является листом — по адресу рассчитанного хеш-индекса происходит формирование хеш-таблицы на следующем оптимальном уровне, в ячейку записывается ссылка на эту хеш-таблицу;
- Непустой лист на уровне хеш-таблицы — по адресу рассчитанного хеш-индекса формируется ячейка со ссылкой на данные ячейки kd-дерева;
- Непустой лист на уровне выше хеш-таблицы — происходит расчёт размера узла в регулярной решетке хеш-таблицы, для каждой ячейки сетки формируется ячейка в хеш-таблице со ссылкой на данные ячейки kd-дерева;

Каждая ячейка хеш-таблицы хранит индекс в регулярной сетке таблицы для однозначной идентификации во время поиска, а также ссылку на следующую хеш-таблицу или на данные ячейки kd-дерева. Для ускорения разрешения коллизий предлагается использовать SIMD операции, позволяющие проводить проверку сразу нескольких ячеек.

На практике предложенная структура ограничена двумя уровнями: первого оптимального уровня, полученного в результате анализа дерева, и конечного уровня, покрывающего остаточное kd-дерево. Такая глубина дерева позволяет сократить число обращений к памяти на порядок и значительно ускорить скорость поиска ячеек. Однако на формирование такой структуры уходит значительно большее время, чем на формирование kd-дерева. Поэтому она будет эффективна для задач, не требующих частого динамического обновления этой структуры, когда выигрыш по времени использования структуры будет превышать затраты на её формирование. Эта особенность может быть компенсирована динамическим алгоритмом построения структуры, позволяющим обновлять её в случае расширения исходного kd-дерева.

3. Оптимизация структуры

Ускоряющая структура предполагает две основные операции: построение структуры и поиск элементов в этой структуре. Время, затрачиваемое на построение структуры, является суммой времени построения kd-дерева и времени построения хеш-таблиц на этом дереве. Оптимизацию структуры можно осуществлять как путём ускорения поиска ячеек, так и путём ускорения её построения. Важно отметить, что различные задачи, требующие применения ускоряющей структуры, имеют различные паттерны её использования. Можно выделить несколько вариантов использования ускоряющей структуры:

- Статический — количество операций поиска в структуре значительно превышает количество элементов поиска. В этом случае требуется достичь минимального времени поиска элементов, за счёт небольшой потери в скорости построения;
- Сбалансированный — количество операций поиска в структуре сходно с количеством элементов поиска. В этом случае требуется соблюдать баланс между временем поиска и временем построения;
- Динамический — количество элементов поиска в структуре значительно превышает количество операций поиска. В этом случае требуется достичь минимального времени построения структуры за счёт незначительной потери в скорости поиска;

Поскольку предлагаемая ускоряющая структура является комбинированным решением, она позволяет производить настройку положения уровней хэш-таблиц и оптимизировать структуру для статического, сбалансированного или динамического варианта ее использования. Для эффективной работы структуры во всех случаях её применения, требуется оптимизировать алгоритмы построения и поиска ячеек, а также подобрать оптимальные параметры структуры.

3.1. Алгоритмы построения и поиска

В данной работе kd-дерево формируется методом деления пространственной области в виде параллелепипеда на две одинаковые подобласти плоскостями параллельными граням параллелепипеда. В результате формируется псевдорегулярная сетка, разрешение которой можно свести к размеру наименьшего параллелепипеда. Хеш-таблицы в этом случае также

будут представляться в виде регулярных сеток, соответствующих уровням kd-дерева, на которых расположены таблицы. Разрешения сетки хеш-таблиц предлагается считать относительными таблицы-предка и сохранять в объектах хеш-таблиц. Таким образом разрешение хеш-таблицы будет являться разрешением сетки поддерева, покрываемого этой хеш-таблицей, построенного от узла-корня, который является листом для предшествующей хеш-таблицы. Разрешение сетки хеш-таблицы в этом случае может быть рассчитано по формуле:

$$TableResolution = Resolution[HashLevel]/Resolution[RootLevel],$$

где *TableResoltion* — разрешение сетки максимального уровня поддерева, которое построено от узла-корня, являющегося листом для таблицы-предка, *Resolution* — массив разрешений для каждого уровня дерева, *HashLevel* — уровень хеш-таблицы в kd-дереве, *RootLevel* — уровень корня поддерева, покрываемого хеш-таблицей в kd-дереве.

Значения разрешений сетки для каждого уровня kd-дерева будет рассчитываться при его построении и сохраняться в массиве разрешений, содержащем значения разрешений для каждого уровня дерева. Это позволит производить быстрый расчёт размера и положения ячейки в регулярной сетке хеш-таблицы для узла kd-дерева, превышающего уровень хеш-таблицы. Размер узла в ячейках регулярной сетки хеш-таблицы будет вычисляться по формуле:

$$NodeSize = Resolution[HashLevel]/Resolution[LeafLevel],$$

где *NodeSize* — размер узла, превышающего уровень хеш-таблицы в её регулярной сетке, *Resolution* — массив разрешений сетки для каждого уровня kd-дерева, *HashLevel* — уровень хеш-таблицы в kd-дереве, *LeafLevel* — уровень обрабатываемого узла в kd-дереве.

Расчёт относительных разрешений сеток хеш-таблиц позволяет равномерно заполнить ячейки хеш-таблицы, уменьшив количество коллизий. В случае хранения абсолютных разрешений вычисление индекса ячеек таблицы затрудняется, поскольку для равномерного заполнения хеш-таблицы необходимо производить пересчёт хеш-индекса, иначе ячейки помещаются только в часть сетки хеш-таблицы, соответствующую части сетки kd-дерева, покрываемой хеш-таблицей, и число коллизий значительно возрастает.

Для быстрого поиска позиции ячеек предлагается хранить внутри объекта хеш-таблицы абсолютную позицию таблицы в сетке kd-дерева уровня хеш-таблицы. Значение позиции хеш-таблицы в этом случае получается из позиции узла kd-дерева, являющегося корнем поддерева, покрываемого этой хеш-таблицей, следующим образом:

$$TablePosition = RootPosition * TableResolution,$$

где *TablePosition* — абсолютная позиция хеш-таблицы в сетке kd-дерева её уровня, *RootPosition* — позиция корня поддерева, покрываемого хеш-таблицей, на его уровне сетки kd-дерева, *TableResolution* — относительное разрешение сетки хеш-таблицы.

Сохранение абсолютных позиций таблиц позволит быстро вычислять положения ячеек таблицы в её регулярной сетке для узлов kd-дерева без дополнительных расчётов и обращений. В этом случае позиция ячейки для узла kd-дерева, находящегося на уровне хеш-таблицы, будет вычисляться по следующей формуле:

$$NodePosition = LeafPosition - TablePosition,$$

где *NodePosition* — позиция ячейки в регулярной сетке хеш-таблицы, соответствующая узлу kd-дерева, *LeafPosition* — позиция обрабатываемого узла kd-дерева в сетке kd-дерева расположенной на уровне хеш-таблицы, *TablePosition* — позиция хеш-таблицы в сетке kd-дерева расположенной на уровне хеш-таблицы, или позиция корневого узла поддерева, покрываемого хеш-таблицей, спроецированная на уровень хеш-таблицы.

Для вычисления позиции узла, находящегося не уровне выше, чем уровень обрабатываемой хеш-таблицы, требуется дополнительно умножить позицию узла на относительное разрешение между уровнем узла kd-дерева и уровнем хеш-таблицы, которое соответствует размеру этого узла kd-дерева в сетке kd-дерева уровня хеш-таблицы. В таком случае позиция ячейки вычисляется по формуле:

$$NodePosition = LeafPosition * NodeSize - TablePosition,$$

где *NodePosition* — позиция ячейки в регулярной сетке хеш-таблицы, соответствующая узлу kd-дерева, *LeafPosition* — позиция обрабатываемого узла kd-дерева в сетке kd-дерева его уровня,

NodeSize — размер узла, превышающего уровень хеш-таблицы в её регулярной сетке, *TablePosition* — позиция хеш-таблицы в сетке kd-дерева уровня хеш-таблицы.

Также предлагается сохранять размер ячейки и положение сетки в глобальном масштабе сцены для каждой хеш-таблицы, это позволит дополнительно ускорить поиск ячейки в дереве для точки пространства. Вместо положения сетки для дальнейшей оптимизации трассировки эффективнее хранить координаты ограничивающего параллелепипеда. При построении хеш-таблицы этот параллелепипед копируется из узла-корня kd-дерева для этой таблицы. В этом случае позиция ячейки хеш-таблицы, принадлежащая точке пространства, будет определяться по следующей формуле:

$$NodePosition = (Point - TableBoundingBox.min)/NodeSize,$$

где *NodePosition* — позиция ячейки в сетке хеш-таблицы, соответствующая точке пространства, *Point* — глобальные координаты точки, *TableBoundingBox.min* — глобальные координата позиции сетки или глобальные координаты ограничивающего параллелепипеда, *NodeSize* — размер ячейки в масштабе глобальных координат для сетки хеш-таблицы.

3.2. Параметры структуры

Исходная структура содержит несколько изменяемых параметров, влияющих на скорость построения и поиска: размер хеш-таблицы, расстояния между уровнями хеш-таблиц, и предельная глубина поддерева, покрываемая хеш-таблицей. Алгоритм определения оптимальных уровней расположения хеш-таблиц влияет на скорость построения структуры, количество хеш-таблиц и требование по памяти. Распределение уровней по kd-дереву, их количество и положение, коррелирует со скоростью поиска ячеек в структуре.

Размер хеш-таблиц напрямую влияет на количество коллизий, возникающих в результате распределения узлов kd-дерева по ячейкам хеш-таблицы. При увеличении глубины поддерева, которое покрывает хеш-таблица, возрастает разрешение регулярной сетки этой хеш-таблицы и количество узлов kd-дерева, покрытое этой таблицей, как следствие, возрастает и количество коллизий. Для компенсации этого эффекта и расчёта оптимального размера хеш-таблиц предлагается использовать одну из следующих формул:

$$HashTableLength = k^{d/3}, \qquad HashTableLength = \left(k \cdot (d/3)\right)^3, \tag{1, 2}$$

где HashTableLength — размер хеш-таблицы, d — глубина поддерева, покрываемого хеш-таблицей, k — множитель размера.

Первый вариант определения размера хеш-таблицы показывает эффективное время поиска и построения структуры при малых значениях глубины поддерева, покрываемого хеш-таблицей, то есть большей глубине хеш-дерева. Второй вариант оказывается наиболее эффективным в поиске ячеек при значениях d близких к половине глубины kd-дерева, то есть при низкой глубине хеш-дерева. Для рассматриваемых в рамках работы значений k и d размер хеш-таблиц, вычисленный первой формулой значительно меньше размера, вычисленного второй.

В ходе исследования были проведены тесты работы структуры при различных коэффициентах k и пределах глубины поддерева хеш-таблиц d для двух вариантов вычисления размера хеш-таблиц. Выведена зависимость эффективного множителя k и предела d от глубины исходного kd-дерева D, что позволило добиться сохранения эффективности работы структуры при различных глубинах kd-дерева для большинства вариантов использования алгоритма.

Оценка работы структуры проводилась на наборе элементов, представляющем собой облако точек, сгенерированное из сетки геометрии тестовой сцены Bunny Box. В качестве элементов, подлежащих поиску, использовались точки, заданные координатами по трём осям. Сгенерированное облако состоит из областей с различной плотностью точек, что позволяет оценить работу структуры с вариативной концентрацией элементов. Количество точек задаётся при генерации облака. В ходе тестирования использовались различные значения количества точек. На рисунке 2 изображена тестовая сцена Bunny Box и пример сгенерированного облака из 500 тысяч точек.

Для оценки эффективности модифицированной структуры на центральном процессоре был реализован алгоритм построения структуры на несбалансированном kd-дереве, а также

алгоритм поиска ячеек структуры, принадлежащих заданной точке пространства. В ходе описанных далее экспериментов в однопоточном режиме была использована система со следующими характеристиками:

- Процессор AMD Ryzen 7 5700G, 3.80ГГц.
- Оперативная память GoodRAM, 32 ГБ DDR4, 3200 МГц.

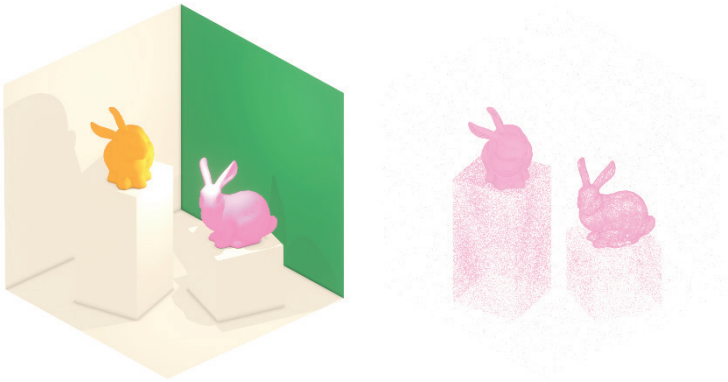

Рисунок 2 – Тестовая сцена Bunny Box и набор элементов, представляющий собой облако с вариативной концентрацией точек, сгенерированное из геометрии этой сцены

Слева на рисунке 3 представлен пример диаграммы зависимости среднего общего времени работы структуры, включающего в себя время построения хеш-таблиц и время поиска ячеек, от коэффициента k при фиксированной глубине kd-дерева $D = 34$ и оптимальном для этого случая значении глубины поддерева, покрываемого хеш-таблицей, $d = 12$. Произведено построение структуры и поиск ячеек для множества из 500 тысяч элементов. Количество операций поиска ячеек соответствует числу элементов. Использована первая формула вычисления длины хеш-таблицы. Для протестированных значений глубин дерева от 12 до 44 уровней и количестве элементов от 10^3 до 10^9 графики имеют схожий вид.

Для динамического варианта применения структуры, сохраняющего высокую скорость поиска, и имеющего наименьшее время построения, наиболее эффективным значением коэффициента является значение $k = 2$. Несмотря на то, что наименьшая скорость построения структуры достигается при значении $k = 1$, эффективность поиска при этом значительно падает, что делает нецелесообразным использование структуры с такими параметрами вместо kd-дерева. Для сбалансированного варианта применения структуры, имеющего минимальное общее время работы ускоряющей структуры, эффективным значением коэффициента является значение $k = 4$.

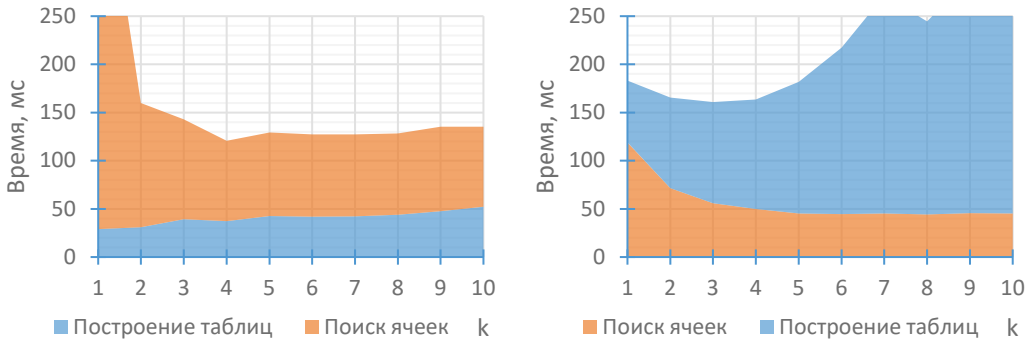

Рисунок 3 – Накопительные диаграммы зависимостей среднего времени работы структуры от множителя размера хеш-таблицы k, при значении глубины kd-дерева $D = 34$, для первой (слева) и второй (справа) формулы расчёта длины хеш-таблиц при оптимальном значении максимальной глубины поддерева хеш-таблиц $d = 12$ и $d = 14$ соответственно

Справа на рисунке 3 представлен пример диаграммы зависимости среднего общего времени работы структуры от коэффициента k для второй формулы вычисления длины хеш-таблицы при

глубине kd-дерева $D = 34$ и оптимальном для этого случая значении глубины поддерева, покрываемого хеш-таблицей, $d = 14$. Произведено построение структуры и поиск ячеек для набора данных из 500 тысяч элементов, представленных точками. Для протестированных значений глубин дерева от 12 до 44 уровней и количестве элементов от 10^3 до 10^9 графики имеют аналогичный вид. Как видно из графика, максимальной скорости поиска при незначительном замедлении построения удаётся достичь при значении $k = 4$. При уменьшении коэффициента поиск значительно замедляется. При увеличении k происходит незначительный рост скорости поиска, однако значительно падает скорость построения структуры, что может послужить причиной замедления работы структуры даже в случае её статического применения.

Таким образом подобраны формулы и оптимальные значения параметра размера хеш-таблиц для трёх типов применения структуры. Для динамического и сбалансированного формула 1 со значениями $k = 2$ и $k = 4$ соответственно, для статического формула 2 со значением $k = 4$.

Для определения уровней kd-дерева, на которых будут расположены хеш-таблицы, вместо поиска оптимальных уровней для каждой хеш-таблицы, заключающегося в определении уровней поддерева с наибольшим количеством терминальных ячеек, предлагается использовать различные подходы для каждого из типов использования структуры.

При динамическом типе использования структуры, для любых значений глубины kd-дерева D, предлагается использовать упрощённый способ размещения хеш-таблиц. Все таблицы будут расположены на одинаковом расстоянии d друг от друга. В качестве этого расстояния предлагается принять значение $d = 9$. При использовании первой формулы определения длинны таблиц с параметром $k = 2$, такой способ расположения таблиц позволяет достичь эффективной скорости построения структуры при сохранении оптимальной скорости поиска за счёт создания таблиц небольшого размера и увеличенной глубины хеш-дерева.

При статическом типе использования структуры предлагается в качестве общего оптимального уровня L_{op} принять последний уровень kd-дерева, число элементов которого соответствует следующему неравенству:

$$N_l > f \cdot N_{max}, \tag{3}$$

где f — параметр доли элементов от 0 до 1, N_l — количество элементов для обрабатываемого уровня kd-дерева, N_{max} — количество элементов уровня, содержащего наибольшее число элементов, подлежащих поиску.

На рисунке 4 изображен пример диаграммы, показывающей распределение количества элементов на уровнях kd-дерева, построенного на тестовом наборе данных из 500 тысяч точек. Для статического типа применения структуры пример оптимального уровня L_{op}, рассчитанный для этого дерева с параметром $f = 0.25$, выделен голубым цветом.

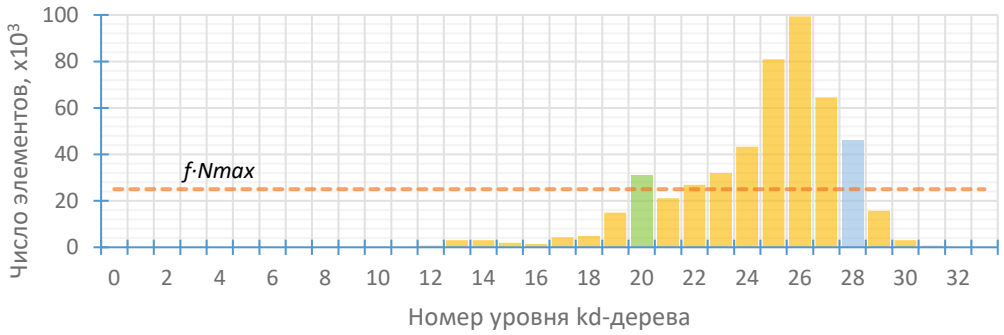

Рисунок 4 – Пример диаграммы распределения количества элементов на уровнях kd-дерева для 500 тысяч элементов, где оранжевой пунктирной линией отмечен предел оптимального уровня при $f = 0.25$, голубым цветом отмечен оптимальный уровень L_{op} для статического типа использования структуры, зелёным — для сбалансированного

Определив общий оптимальный уровень kd-дерева L_{op}, предлагается вычислять оптимальные уровни kd-дерева для расположения хеш-таблиц используя параметр количества хеш-уровней d_h до оптимального уровня L_{op} следующим образом:

$$L = L_h \cdot \frac{L_{op}}{d_h} + L_{op} \; mod \; d_h,$$

где L_h — номер уровня хеш-таблицы в хеш-дереве, L_{op} — номер оптимального уровня в kd-дереве, d_h — параметр количества хеш-уровней до оптимального уровня kd-дерева.

В этом случае все таблицы будут размещены на расстоянии L_{op}/d_h друг от друга, первый уровень хеш-таблиц будет смещён относительно L_{op}/d_h на $L_{op} \; mod \; d_h$. Расположение хеш-таблиц вычисляется таким образом, чтобы L_{op} входил в число уровней, на которых будут расположены хеш-таблицы. Изменяя параметр d_h, можно изменять глубину хеш-дерева, оказывая влияние на скорость работы структуры.

При оптимальном параметре d_h таблицы будут расположены таким образом, чтобы покрывать уровни, потенциально имеющие большее число обращений при поиске, сохраняя при этом оптимальную глубину поддерева и необходимую глубину хеш-дерева. Такой подход уменьшает затраты на построение дерева, исключая хранение в каждой ячейке дерева статистики всех уровней его поддерева, и дорогостоящий подсчёт статистики этих уровней для каждой ячейки. Высокая эффективность работы структуры получается за счёт быстрого формирования и ограничения размера хеш-таблиц малыми значениями, или за счёт уменьшения глубины хеш-дерева при сохранении низкого числа коллизий в ячейках хеш-таблицы.

При сбалансированном типе использования структуры, для обеспечения максимальной общей скорости работы, предлагается использовать аналогичный подход, однако в качестве общего оптимального L_{op} необходимо принять первый уровень, соответствующий неравенству (3). Оптимальная скорость работы в этом случае достигается за счёт увеличенной глубины хеш-дерева и меньшего, по сравнению со статическим вариантом, размера хеш-таблиц.

Для каждого из типов применения структуры была проведена оценка времени работы структуры при различных значениях параметров d_h и f на различных глубинах исходного kd-дерева. На рисунке 5 представлены диаграммы зависимости эффективности работы структуры от параметров f и d_h. В качестве оптимального параметра f для статического типа применения структуры предлагается использовать параметр $f < 0.5$, для сбалансированного типа $f > 0.5$. Прямой зависимости параметра f от глубины kd-дерева D не обнаружено. Значения, близкие к значению 0.5, показывают оптимальные результаты работы в любых случая применения. В качестве оптимального параметра d_h для обоих случаев, использующих описанный подход, предлагается принять значение $d_h = 2$. Оптимального для каждого случая времени работы структуры получается добиться за счёт различного способа определения общего оптимального уровня L_{op} и различного способа определения размера хеш-таблиц.

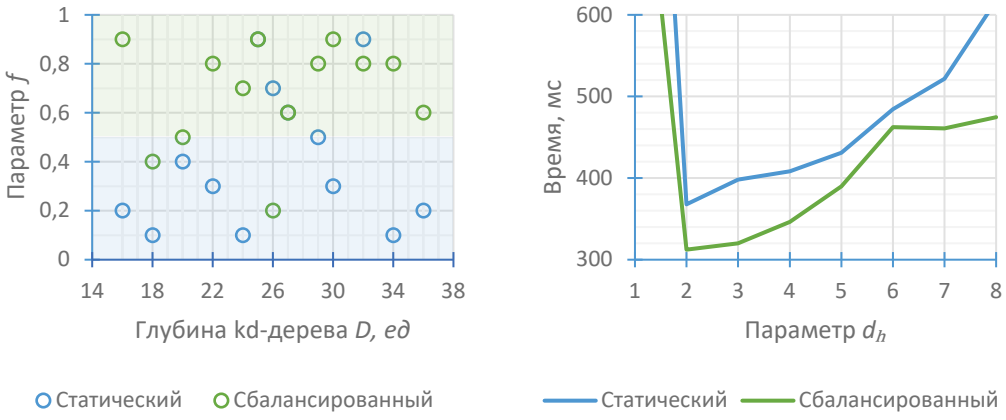

Рисунок 5 – Для двух типов применения структуры слева изображена диаграмма зависимости оптимальных значений параметра f от глубины kd-дерева D, справа — пример диаграммы зависимости общего времени работы структуры с 500 тысячью элементов, составленного из времени построения структуры и времени поиска ячеек для всех её элементов, от параметра d_h

4. Тестирование и сравнение

Был проведен сравнительный анализ модифицированного решения, использующего многоуровневые хэш-таблицы, с исходным решением, предложенным в статье [14], а также с решением, основанным на использовании несбалансированного kd-дерева. Модифицированная структура протестирована на трёх настройках параметров для трёх случаев применения структуры. Сравнение происходило в однопоточном режиме на системе, указанной в главе 3.2. Для повышения эффективности алгоритма возможна его реализация с поддержкой параллельных вычислений.

Тестовые облака точек сгенерированы на основе геометрии сцены Bunny Box для трёх значений числа точек: 500, 250 и 100 тысяч. Набор тестовых элементов содержит области с различной концентрацией точек, что позволяет оценить эффективность работы структур в практических задачах. Для каждого облака точек были построены пять ускоряющих структур и произведён поиск ячеек для каждого из элементов облака. Таким образом количество операций поиска ячеек соответствует числу элементов структуры. На рисунках 5, 6 и 7 представлены диаграммы времени построения и поиска ячеек для каждой из структур. В таблицах 1, 2 и 3 представлены результаты сравнения времени работы этих структур.

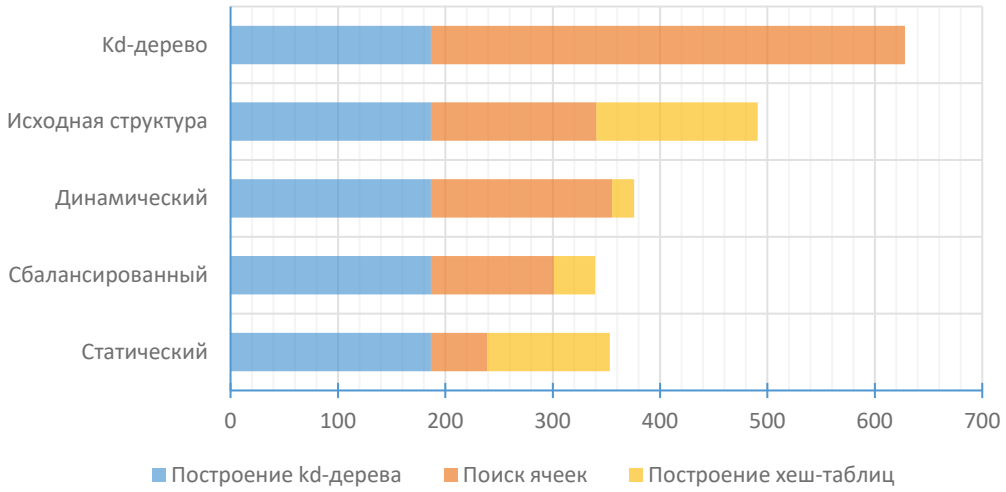

Рисунок 5 – Диаграмма сравнения времени работы модифицированной структуры для трёх её различных настроек с исходной структурой и kd-деревом для набора данных из 500 тысяч элементов, представленного в виде облака точек, сгенерированного из сцены Bunny Box

Таблица 1 – Сравнение времени работы модифицированной структуры для трёх её различных настроек с исходной структурой и kd-деревом для набора из 500 тысяч элементов, представленного в виде облака точек, сгенерированного из сцены Bunny Box

Структура / набор параметров	Построение kd-дерева, мс	Построение хеш-таблиц, мс	Поиск ячеек, мс	Общее время работы, мс
Статический	186.5	114.1	52.5	353.2
Сбалансированный		38.6	114.5	339.6
Динамический		20.6	168.6	365.8
Исходная структура		150.3	154.1	490.9
Kd-дерево		—	441.6	628.1

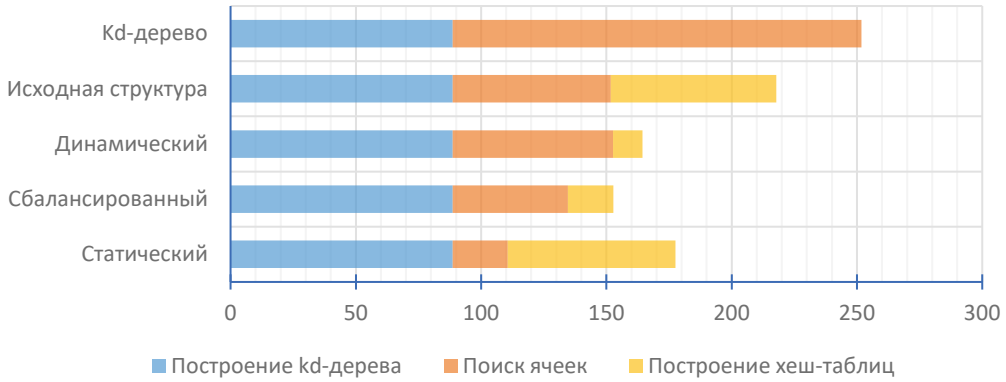

Рисунок 6 – Диаграмма сравнения времени работы модифицированной структуры для трёх её различных настроек с исходной структурой и kd-деревом для набора данных из 250 тысяч элементов, представленного в виде облака точек, сгенерированного из сцены Bunny Box

Таблица 2 – Сравнение времени работы модифицированной структуры для трёх её различных настроек с исходной структурой и kd-деревом для набора из 250 тысяч элементов, представленного в виде облака точек, сгенерированного из сцены Bunny Box

Структура / набор параметров	Построение kd-дерева, мс	Построение хеш-таблиц, мс	Поиск ячеек, мс	Общее время работы, мс
Статический		66.9	21.9	177.5
Сбалансированный		18.2	45.9	152.8
Динамический	88.6	11.7	64.1	164.3
Исходная структура		66.1	63.1	217.7
Kd-дерево		—	163.2	251.8

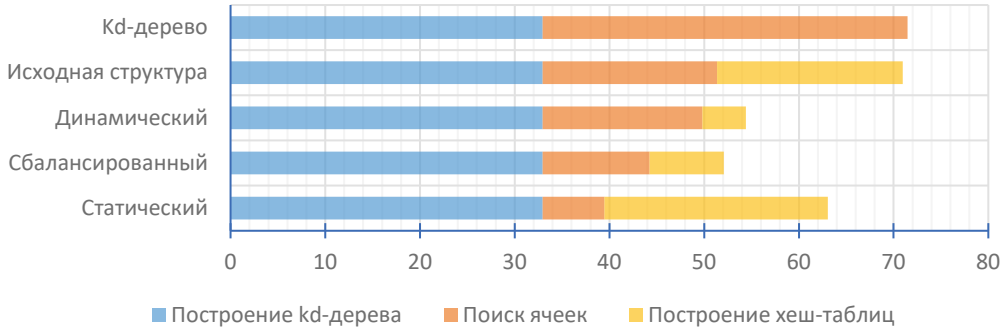

Рисунок 7 – Диаграмма сравнения времени работы модифицированной структуры для трёх её различных настроек с исходной структурой и kd-деревом для набора данных из 100 тысяч элементов, представленного в виде облака точек, сгенерированного из сцены Bunny Box

Таблица 3 – Сравнение времени работы модифицированной структуры для трёх её различных настроек с исходной структурой и kd-деревом для набора из 100 тысяч элементов, представленного в виде облака точек, сгенерированного из сцены Bunny Box

Структура / набор параметров	Построение kd-дерева, мс	Построение хеш-таблиц, мс	Поиск ячеек, мс	Общее время работы, мс
Статический		23.6	6.5	63.1
Сбалансированный		7.8	11.2	52.1
Динамический	32.9	4.5	16.8	54.4
Исходная структура		19.6	18.5	71.0
Kd-дерево		—	38.5	71.5

Анализ результатов сравнения показал, что в случае использования настройки структуры для статического применения удаётся добиться уменьшения времени поиска до восьми с половиной раз по сравнению с kd-деревом и до трёх раз по сравнению с исходным решением, сохранив исходную скорость построения хеш-таблиц. В случае использования структуры при динамической настройке удаётся уменьшить скорость построения хеш-таблиц по сравнению с исходной структурой до семи с половиной раз, сохранив исходную скорость поиска, превышающую скорость поиска в kd-дереве в три раза. При применении структуры в сбалансированной настройке удаётся получить уменьшение времени поиска до четырёх раз по сравнению с kd-деревом и до полутора раз по сравнению с исходной структурой, увеличив скорость построения хеш-таблиц до четырёх раз.

5. Заключение

В статье был представлен анализ существующих методов построения ускоряющих структур пространственного разбиения. Были рассмотрены достоинства и недостатки трёх типов ускоряющих структур: регулярных, иерархических и комбинированных. Изучены свойства и особенности комбинированной структуры, построенной на основе kd-дерева и хеш-таблиц. Выделены три типа случаев использования комбинированных ускоряющих структур, требующие оптимизации параметров этих структур. Предложены методы оптимизации алгоритмов построения и поиска ячеек исходной комбинированной структуры, позволяющие уменьшить их вычислительную сложность. Предложены два алгоритма определения размера хеш-таблиц, позволяющие получить эффективные размеры хеш-таблиц для каждого из трёх случаев использования структуры. Предложено три алгоритма определения уровней расположения хеш-таблиц на исходном kd-дереве для трёх случаев использования структуры. Проведён анализ настроечных параметров алгоритмов, обеспечивающих эффективную работу с этими ускоряющими структурами. Каждый из предложенных алгоритмов позволяет добиться оптимальной глубины хеш-дерева и размера хеш-таблиц для различных случаев применения структуры. Представленные в статье алгоритмы могут служить основой для дальнейшего улучшения методов поиска элементов и трассировки лучей с использованием комбинированных ускоряющих структур.

6. Благодарности

Работа была выполнена при финансовой поддержке Российского Научного Фонда, грант №22-11-00145.

7. Список источников

[1] Cook R.L. A reflectance model for computer graphics // ACM Transactions on Graphics (ToG). 1982. Vol. 1, № 1. 7-24.
[2] Computer Graphics: Principles and Practice, Third Edition. J.F. Hughes: Addison-Wesley Professional. 2013. 1262.
[3] Kajiya J.T. The rendering equation // ACM SIGGRAPH Computer Graphics. 1986. Vol. 20, № 4. 143-150.
[4] MacDonald J.D., Booth K.S. Heuristics for ray tracing using space subdivision // The Visual Computer. 1990. № 6. 153-166.
[5] A Survey on Bounding Volume Hierarchies for Ray Tracing / D. Meister, O. Shinji, B. Carsten, D. Michael, G. Michael, B. Jiri // Computer Graphics Forum. 2021. № 40. 683-712.
[6] Shagam J., Pfeiffer J. Dynamic Spatial Partitioning for Real-Time Visibility Determination. 2003.
[7] Jensen H.W. Global illumination using photon maps // Eurographics workshop on Rendering techniques. 1996. 21-30.

[8] Christensen P. A Practical Guide to Global Illumination Using Photon Mapping // Siggraph. 2002. № 43. 93-121.
[9] Havran V., Herzog R., Seidel H.P. Final gathering via reverse photon mapping // Computer Graphics Forum. 2005. Vol. 24, № 3. 323-332.
[10] Bentley J.L., Friedman J.H. Data structures for range searching. // ACM Computing Surveys (CSUR). 1979. Vol. 11. № 4. 397-409.
[11] Chen Q.P., Xue B., Siepmann J.I. Using the k-d tree data structure to accelerate Monte Carlo simulations // Journal of Chemical Theory and Computation. 2017. Vol. 13. № 4. 1556-1565
[12] Knoll A. A survey of octree volume rendering methods. 2006.
[13] Fabianowski B., Dingliana J. Compact BVH Storage for Ray Tracing and Photon Mapping // Eurographics Ireland Workshop. 2009. 1-8.
[14] Использование многоуровневых хэш-таблиц для ускорения процесса рендеринга / Д.Д. Жданов, А.И. Лысых, Р.Р. Халимов, И.Е. Кинев, А.Д. Жданов. // Программирование. 2023. № 3. С. 37–48.
[15] Халимов Р.Р., Жданов Д.Д., Жданов А.Д. Формирование эффективной пространственной структуры фотонных карт для ускорения процесса рендеринга // Труды Международной конференции по компьютерной графике и зрению "Графикон". 2022. № 32. С. 110-123.