

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б07-мм

Реализация библиотеки с поддержкой октодеревьев для SLAM

КИСЕЛЁВ Михаил Николаевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
Старший преподаватель кафедры СП, к.т.н. Ю. В. Литвинов

Консультант:
Инженер-исследователь Mobile Robotics Lab, Skoltech А. В. Корнилова

Санкт-Петербург
2024

Оглавление

1. Введение	3
2. Постановка задачи	5
3. Обзор	6
3.1. Требования к библиотеке	6
3.2. Существующие библиотеки для работы с октодеревьями	9
3.3. Алгоритм RANSAC для выделения плоскостей и филь-	
трации точек	10
3.4. Метрики и наборы данных	12
3.5. Вывод	12
4. Библиотека для работы с октодеревьями	14
4.1. Архитектура	14
4.2. Реализация	20
5. Экспериментальное сравнение	23
5.1. Условия экспериментов	23
5.2. Цель и исследовательские вопросы	23
5.3. Исследование работы библиотеки	24
5.4. Выводы	28
6. Заключение	30
Список литературы	31
Приложение А	36

1. Введение

Автономные системы, такие как промышленные роботы и беспилотные транспортные средства, быстро развиваются и становятся неотъемлемой частью современной жизни. Ключевым элементом работы таких устройств являются SLAM-алгоритмы (Simultaneous Localization and Mapping), которые в режиме реального времени обеспечивают возможность определения местоположения и создания карты окружающей среды. SLAM-алгоритмы (рис. 1) состоят из двух взаимосвязанных частей: фронтенда, задача которого состоит в выделении ориентиров из исходных данных с датчиков автономной системы, и бэкенда, производящего построение и обновление карты на основе этих ориентиров, а также оценку местоположения системы относительно них.



Рис. 1: Принцип работы SLAM-алгоритмов [39].

Одним из распространенных типов исходных данных в автономных системах являются облака точек, полученные с помощью LiDAR-сенсора (Light Detection and Ranging) — устройства, генерирующего облака точек с помощью лазерного сканирования окружающей его области. Среди примитивов, извлекаемых (сегментируемых) из облаков точек, часто используются плоскости, поскольку они распространены в средах, созданных человеком, а также для них существует множество подходов к сегментированию, например, способы, основанные на кластеризации, разрастании регионов и делении пространства на воксели — трехмерные пиксели, объединяющие пространство в объемную сетку.

Метод, основанный на разбиении пространства на воксели, выделяется более качественными результатами [1], а также меньшими вычис-

лительными затратами по сравнению с подходами, основанными на кластеризации и разрастании регионов [7, 23]. Кроме того, отладка с визуализацией в таком способе более проста для интерпретации, поскольку каждый воксель может быть анализирован независимо, что упрощает выявление и исправление ошибок.

В рамках данного подхода к сегментированию находят широкое применение октодеревья — деревья с восемью потомками у каждого узла, где точки хранятся в листьях дерева [19, 35]. При таком подходе октодеревья обеспечивают возможность динамического добавления и обработки новых данных в реальном времени. Помимо этого, воксельный подход к решению SLAM-задач с использованием октодеревьев позволяет эффективно адаптироваться к различным масштабам окружающей среды и детализированности рассматриваемых наблюдений. Это достигается за счет способности изменять разрешение вокселей и глубину дерева в зависимости от специфики сцены.

Однако, несмотря на преимущества воксельного подхода, на текущий момент отсутствует универсальное и модульное решение для задач воксельного SLAM, что ограничивает возможности разработчиков. Лаборатория мобильной робототехники в Сколтехе ведёт разработку собственного инструмента, ориентированного на модульность и расширяемость. Одной из ключевых составляющих, необходимых для реализации такого инструмента, является библиотека для работы с октодеревьями. На данный момент не существует специализированного для воксельного SLAM решения, которое бы удовлетворяло требованиям, необходимым для реализации инструмента, разрабатываемого в Лаборатории.

Целью данной работы является разработка библиотеки для работы с октодеревьями в контексте воксельных SLAM-алгоритмов.

2. Постановка задачи

Целью работы является реализация библиотеки для работы с октодеревьями в рамках воксельных SLAM-алгоритмов.

Для достижения цели были поставлены следующие задачи.

1. Сформировать список функциональных требований к библиотеке.
2. Провести обзор существующих решений, реализующих работу с октодеревьями, и провести их анализ относительно необходимой функциональности, выполнить обзор метрик и наборов данных для измерения производительности решения.
3. Спроектировать и реализовать библиотеку для работы с октодеревьями в соответствии с функциональными требованиями.
4. Провести экспериментальное исследование производительности разработанного решения.

3. Обзор

В данном разделе сформированы требования к библиотеке, а также проведен обзор существующих решений, реализующих работу с октодеревьями и облаками точек, а также рассмотрены наборы данных и метрики, используемые для сравнения их производительности.

3.1. Требования к библиотеке

Для того чтобы составить список требований, рассмотрим базовый принцип работы воксельного SLAM-алгоритма с применением октодеревьев (рис. 2).

Подход к SLAM-задаче в рамках воксельного метода с использованием октодеревьев заключается в делении пространства с облаком точек на воксели одинакового размера, в каждом из которых создается октодерево. Каждое октодерево впоследствии подвержено рекурсивному делению на более мелкие воксели (октететы), где каждое очередное разделение осуществляется, если облако точек внутри данного вокселя удовлетворяет заданному пользователем критерию [33, 27, 17, 7]. Такой механизм позволяет обрабатывать одни части сцены более детально, в то время как другие могут оставаться более обобщенными. Кроме того, подобный способ обладает гибкостью относительно используемого критерия деления, что позволяет адаптировать алгоритм под различные сценарии использования.

Далее производится отдельная обработка поделенных вокселей с помощью функции, которая раздельно преобразует облака точек внутри каждого вокселя, что может быть использовано для устранения шумов в облаке. В частности, одним из популярных способов устранения шумов из облаков точек является применение алгоритма RANSAC [8]. Использование этого алгоритма широко распространено в SLAM-задачах из-за его способности надежно выделять ориентиры из зашумленных данных [31, 4, 3].

Принцип работы алгоритма RANSAC [8] для выделения плоскостей из облака точек и фильтрации шумов описан ниже.

1. Генерация и оценка N случайных гипотез, каждая из которых рассчитывается следующим образом:
 - (а) случайный выбор M точек из облака;
 - (б) вычисление плоскости, приближающей выбранные точки с наименьшей среднеквадратичной ошибкой, с помощью метода главных компонент;
 - (с) нахождение количества «внутренних» точек — таких, расстояние от которых до плоскости не превышает заданный порог.
2. Выбор из полученных гипотез той, которой соответствует наибольшее значение количества внутренних точек.
3. Нахождение внутренних точек для выбранной плоскости.

В конце работы фронтенда производится ассоциация вокселей в рамках нескольких наблюдений SLAM-системы и возвращение полученных облаков точек, соответствующих им вокселей и полученных ассоциаций пользователю, а далее с помощью бэкенда производится оценка карты и положения системы в пространстве.

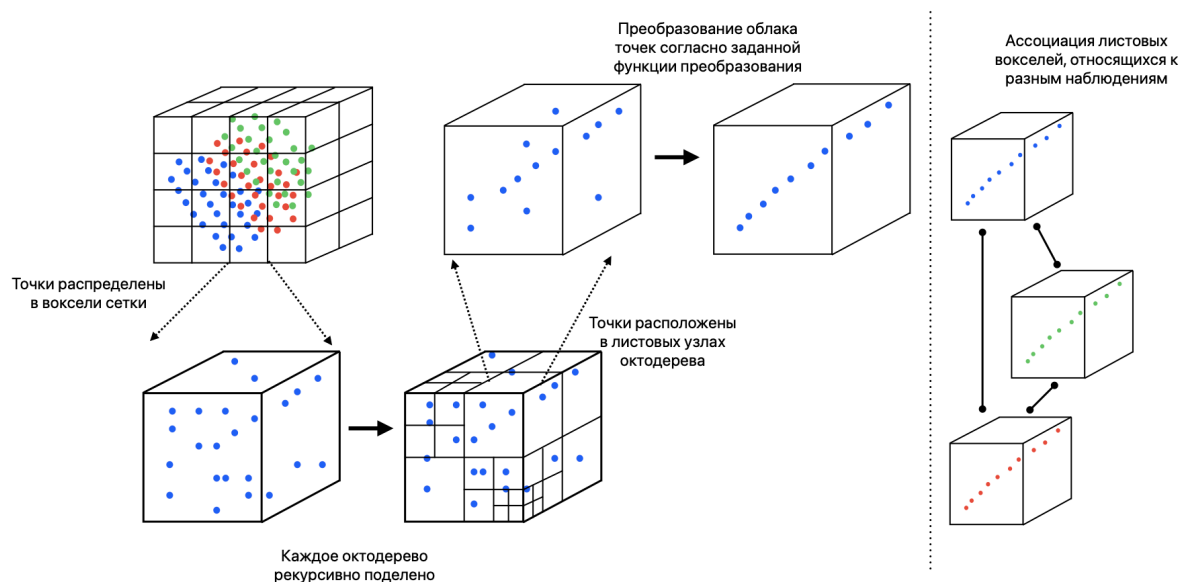


Рис. 2: Принцип работы воксельного SLAM с применением октодерев.

Поскольку воксельные SLAM-алгоритмы основаны на работе с трехмерным пространством, визуализация становится неотъемлемым инструментом для анализа и отслеживания процесса построения карты и локализации.

Учитывая принцип работы воксельных SLAM-алгоритмов на основе октодеревьев, был составлен список требований к библиотеке.

- Реализация пространственной сетки с конфигурируемым разрешением разбиения, где каждому вокселю соответствует октодерево.
- Листовые узлы октодерева должны служить контейнерами для точек, обеспечивая эффективное хранение данных.
- Поддержка возможности рекурсивного деления октодерева на октеты на основе выполнения заданного условия.
- Поддержка возможности хранения облаков, соответствующих множеству наблюдений, в одной пространственной сетке.
- Поддержка возможности одинакового деления всех октодеревьев в заданном вокселе на основе одного или нескольких облаков точек, соответствующих заданным наблюдениям. Одинаковая структура октодеревьев для разных облаков точек необходима для ассоциации листовых узлов, которые тоже должны иметь одинаковое положение в пространстве и длину ребра.
- Поддержка возможности модификации облака точек внутри вокселя с использованием трансформирующей функции.
- Визуализация пространственной сетки и находящихся в ней октодеревьев.
- Поддержка алгоритма RANSAC для выделения плоскостей из точек вокселя и выделения тех точек, которые расположены не дальше заданного расстояния от плоскости. Также необходима поддержка параллельной обработки множества вокселей.

3.2. Существующие библиотеки для работы с октодеревьями

В данном разделе будет проведен обзор существующих библиотек и их особенностей. В таблице 1 указаны рассмотренные решения и их соответствие поставленным требованиям.

В Open3D [36] и PointCloudLibrary [29] реализована структура данных «октодерево» с возможностью использовать его как контейнер для облаков точек, однако эти библиотеки не разрабатывались для SLAM-задач и не реализуют требуемые особенности функциональности, как то: модификация точек внутри заданного узла дерева, рекурсивное деление дерева на основе заданного условия, синхронизация схемы деления октодеревьев, имеющих одинаковое расположение в пространстве. Поскольку эти библиотеки не разрабатывались для описанных задач, их адаптация представляется неэффективной и потребует существенных усилий.

Авторы OctoMap [22], MID-Fusion [18] и OctNet [27] исследуют применение октодеревьев для воксельных SLAM-алгоритмов, и в этих решениях реализована ассоциация вокселей, однако октодеревья в данных подходах использованы в качестве инструмента представления пространства, а не как контейнер для точек, и поэтому их реализации не предоставляют некоторых ключевых аспектов функциональности, описанных в 3.1. Также эти реализации не нацелены на работу с несколькими октодеревьями с разделением по наблюдениям. Для уменьшения сложности доступа к листьям дерева по сравнению со стандартным октодеревом неограниченной глубины авторы OctNet [27] используют предложенную Миллером и др. [20] гибридную структуру данных Grid-Octree (см. рис. 3), не храня всю информацию о наблюдении в одном дереве, а разделяя ее между несколькими деревьями одного размера.

Кроме того, есть библиотеки октодеревьев, которые не нацелены на работу со SLAM и не реализуют необходимую функциональность: pyoctree [13], octrees [6], Sparse Voxel Octree Raycasting [9].

Проведенный анализ выявил, что ни одно из рассмотренных реше-

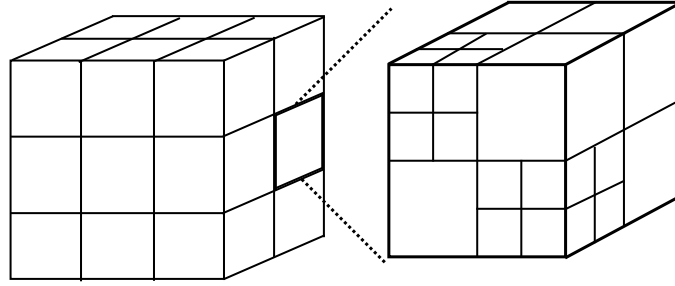


Рис. 3: Структура данных Grid-Octree, предложенная Миллером и др. [20] Пространство поделено на воксели, в каждом из которых находится октодереве.

ний не соответствует всем установленным критериям. Таким образом, для эффективной работы с октодеревьями требуется разработка нового инструмента.

3.3. Алгоритм RANSAC для выделения плоскостей и фильтрации точек

Структура алгоритма RANSAC позволяет подвергнуть его параллелизации: гипотезы для одного вокселя могут обрабатываться параллельно, так как они не связаны друг с другом. Кроме того, поскольку в рамках библиотеки алгоритм RANSAC обрабатывает каждый воксель отдельно, существует еще одна возможность для параллелизации: разные воксели могут обрабатываться параллельно. Поскольку число гипотез обычно много меньше числа вокселей, которые необходимо обрабатывать, особое внимание необходимо уделить именно параллельной обработке разных вокселей.

3.3.1. Библиотеки, реализующие алгоритм RANSAC

В данном разделе представлен обзор существующих решений, включающих реализацию алгоритма RANSAC.

Библиотеки Open3D [36], CGAL [34], RansacLib [30], ROS [28], SkLearn [32], RTL [3] включают в себя реализацию алгоритма RANSAC,

однако она приспособлена только для обработки с использованием CPU. Некоторые из этих решений реализуют параллельную обработку разных гипотез, однако не предоставляют возможности параллельной обработки разных облаков точек.

В библиотеках PCL [29] и Kornia [15] реализация RANSAC поддерживает запуск на GPU, однако не предоставляет возможности параллельного запуска на нескольких различных облаках точек, из-за чего обработка множества вокселей параллельно с помощью таких библиотек невозможна. Кроме того, в библиотеке Kornia алгоритм используется для нахождения гомографий¹, а не для сегментации примитивов [15].

Ни одно из рассмотренных решений не предоставляет возможности параллельной обработки нескольких облаков точек, что ограничивает эффективность этих решений при обработке большого числа вокселей.

	<i>Open3D</i>	<i>PCL</i>	<i>OctoMap</i>	<i>MID-Fusion</i>	<i>OctoNet</i>	<i>pyoctree</i>	<i>octrees</i>	<i>SVOR</i>
Контейнер для облака точек	-	+	-	-	-	-	+	-
Рекурсивное деление по критерию	-	+	+	+	+	-	+	-
Модификация облака точек в октодереве	-	-	-	-	-	-	-	-
Пространственная сетка	+	-	-	-	+	-	+	-
Синхронизация множества наблюдений	-	-	-	-	-	-	-	-
Визуализация	+	+	+	+	+	-	-	-

Таблица 1: Существующие решения для работы с октодеревьями и их соответствие поставленным критериям.

¹Гомография — отношение, которое связывает два изображения одной поверхности в пространстве. Это отношение характеризуется поворотом и смещением камеры в пространстве [5].

3.4. Метрики и наборы данных

Авторы OctoMap и OctNet измеряют память, используемую для обработки разных наборов данных [22], а также время обработки облаков точек при разной максимальной глубине октодеревьев. В статье, посвященной алгоритму MID-Fusion [18], упоминаются измерения времени и памяти, однако результаты этих экспериментов не предоставляются.

Поскольку основная задача библиотеки — это обработка больших массивов точек, в качестве исследуемых метрик были выбраны **время работы** библиотеки и **объем используемой памяти** при обработке разных наборов облаков точек, а также разных параметров запуска, как разрешение пространственной сетки и максимальная глубина октодерева.

Среди наборов данных, состоящих из последовательностей облаков точек, для измерения качества работы и производительности решений, относящихся к обработке облаков точек в контексте SLAM, часто используются перечисленные ниже наборы данных.

- **KITTI** [10]. Этот набор данных представляет собой данные нескольких сенсоров: LiDAR, RGB-D камеры. Набор содержит 20 сцен, для части из которых также предоставляются данные о положении устройства, полученные с помощью GPS-сенсора.
- **Hilti** [12]. Эта группа наборов данных представляет собой данные с RGB-D камер и LiDAR-сенсора. Для части наборов данных также предоставляются ground-truth данные о траектории устройства.

3.5. Вывод

Несмотря на то, что октодерево часто используется в воксельных SLAM-алгоритмах, существующие библиотеки не полностью реализуют необходимую функциональность, ввиду чего их использование в рамках разрабатываемого в Лаборатории решения невозможно. Кроме того, существующие реализации алгоритма RANSAC не предоставляют

возможности параллельной обработки множества различных исходных данных, что ограничивает их эффективность при обработке большого числа вокселей.

4. Библиотека для работы с октодеревьями

В данном разделе будет представлена архитектура, а также нюансы реализации разрабатываемой библиотеки.

4.1. Архитектура

Исходя из требований, описанных в секции 3.1, была спроектирована библиотека, реализующая работу с октодеревьями.

Спроектированная библиотека для работы с октодеревьями содержит следующие модули:

- **internal** — внутренние классы и информация о типизации;
- **octree** — реализация класса октодерева;
- **octree_manager** — реализация класса, позволяющего работать с облаками точек, относящимися к разным наблюдениям;
- **grid** — реализация пространственной сетки.

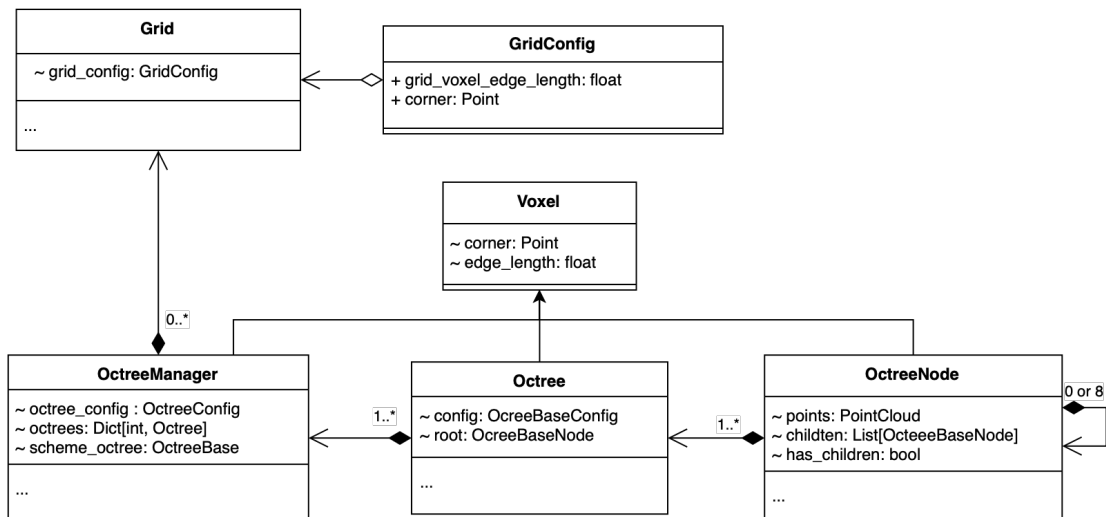


Рис. 4: Базовая структура модулей библиотеки

4.1.1. Модуль `internal`

Этот модуль предназначен для внутреннего использования. В нем содержатся интерфейс `WithID` и классы `VoxelBase`, `Voxel`. Интерфейс `WithID` обеспечивает наличие уникальных идентификаторов объектов классов, которые его наследуют. Классы `VoxelBase`, `Voxel` представляют воксель в пространстве, который определяется углом с минимальными координатами и длиной ребра. `Voxel` является наследником `VoxelBase` и отличается наличием методов для вставки и получения точек из вокселя.

4.1.2. Модуль `octree`

Модуль `octree` (рис. 5) реализует структуру данных октодерева. Предлагаемая версия использует для этого интерфейсы `OctreeBase`, `OctreeNodeBase` и классы `Octree`, `OctreeNode`. `OctreeNode` и `OctreeNodeBase` являются представлением узла октодерева. Дерево имеет рекурсивную структуру, где каждый нелистовой узел хранит восемь дочерних узлов-октетов. `Octree` и `OctreeBase` представляют само октодерево и предназначены для взаимодействия других классов со структурой данных, в то время как `OctreeNode` и `OctreeNodeBase` не предполагают взаимодействия с другими классами, кроме `Octree`. `OctreeNodeBase` и `OctreeBase` являются абстрактными классами, от которых наследованы `OctreeNode` и `Octree` соответственно. Такое разделение обусловлено требованием расширяемости библиотеки. В случае необходимости имеется возможность использования альтернативной реализации октодерева и узла октодерева, которые бы наследовали `OctreeBase`, `OctreeNodeBase`.

Возможность деления октодерева в зависимости от выполнения заданного критерия реализована с помощью метода `subdivide`, которая принимает логическую функцию от облака точек, значение которой проверяется для каждого листового узла дерева. Если функция возвращает логическую истину, листовый узел делится, а затем метод `subdivide` вызывается от новых узлов.

В рамках предварительной обработки облаков точек метод `map_leaf_points` реализует преобразование облаков для каждого листового вокселя с помощью функции, принимаемой в качестве параметра. Эта функция принимает облако точек и возвращает другое облако точек, которое присваивается данному вокселю.

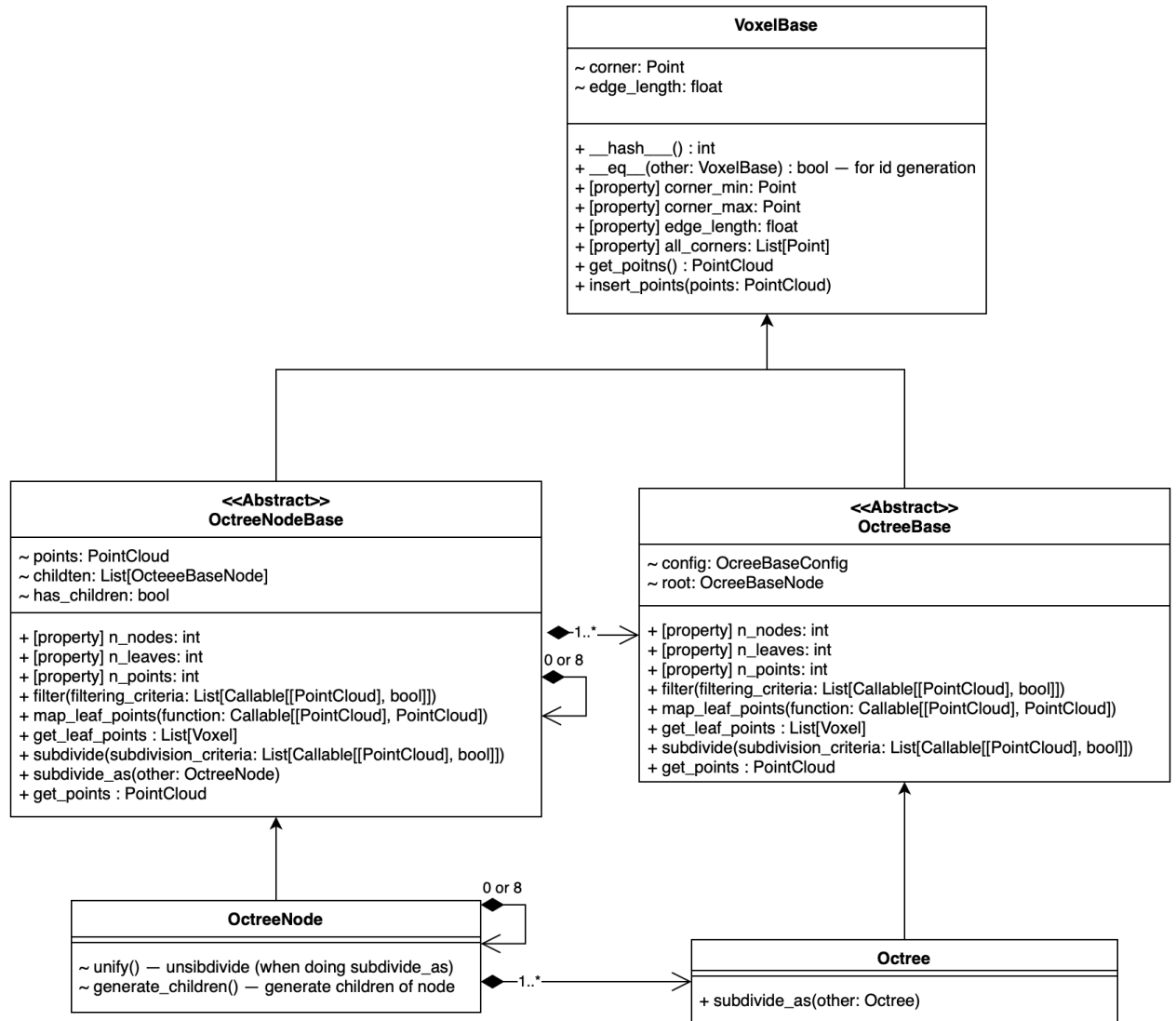


Рис. 5: Структура модуля `octree`.

4.1.3. Модуль `octree_manager`

Данный модуль содержит класс `OctreeManager` (рис. 6), отвечающий за хранение нескольких октодеревьев внутри одного вокселя, где каждое октодеревье соответствует одному наблюдению. Этот класс поддерживает инвариант, что все октодеревья, принадлежащие одной об-

ласти и относящиеся к разным наблюдениям, поделены одинаково вне зависимости от облаков, хранящихся в каждом октодереве.

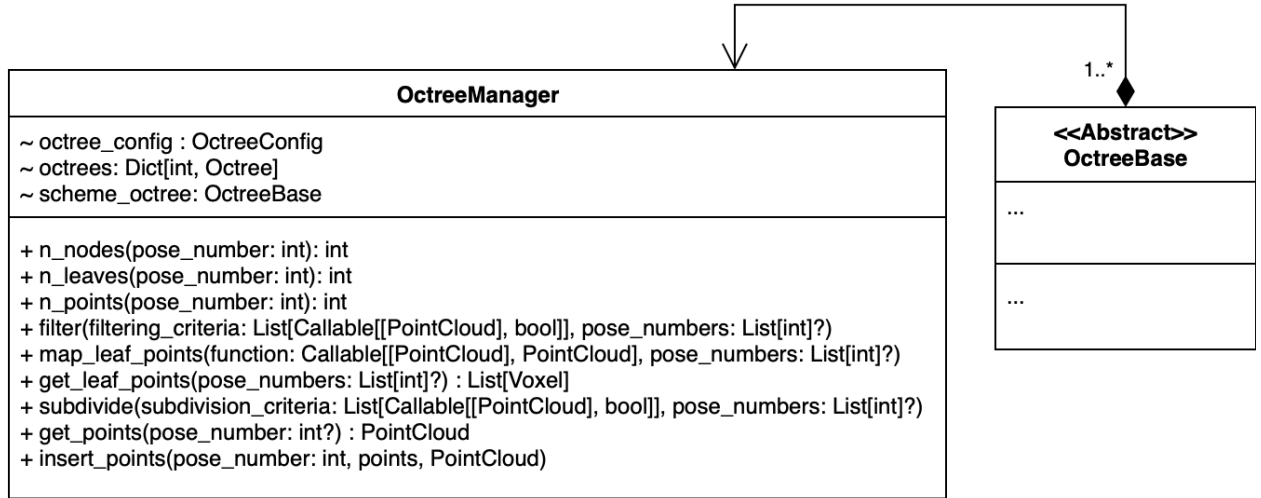


Рис. 6: Структура модуля `octree_manager`

4.1.4. Модуль `grid`

Данный модуль содержит интерфейс `GridBase` и класс `Grid` (рис. 7), которые являются представлением трехмерной сетки в гибридной архитектуре Grid-Octree [20]. Такое разделение сделано для возможности использования пользователем альтернативных реализаций пространственной сетки. В предлагаемой реализации этот модуль не хранит октодеревья непосредственно. Вместо этого для данной задачи используется промежуточный класс `OctreeManager`, который поддерживает согласованность схемы деления разных октодеревьев (рис. 8).

Подразумевается, что сетка является основным объектом, с которым взаимодействует пользователь при использовании библиотеки для обработки точек.

4.1.5. Алгоритм RANSAC

Алгоритм RANSAS прост в своей структуре и эффективен в выделении плоскостей из зашумленных облаков точек. В разрабатываемой библиотеке он применяется для получения плоскостей из облаков точек

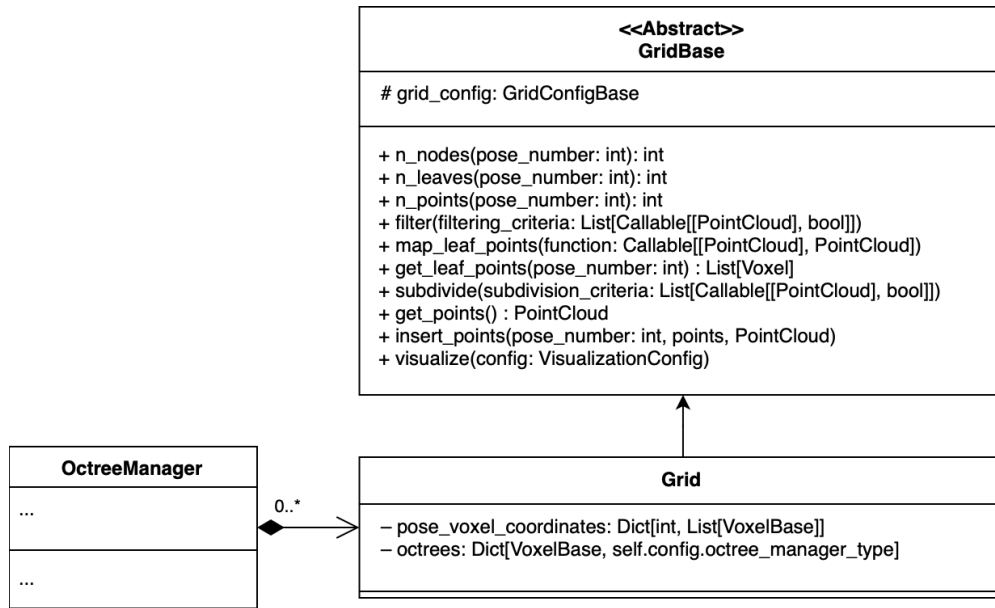


Рис. 7: Структура модуля `grid`

индивидуально для каждого вокселя. Ввиду своей структуры и особенностей его применения в библиотеке алгоритм может быть подвержен эффективной параллелизации.

Для параллельной обработки различных гипотез и вокселей используется CUDA² — архитектура для параллельных вычислений, позволяющая использовать GPU для вычислений общего назначения. Такая архитектура позволяет выполнять вычисления с использованием множества блоков и потоков (рис. 9). Один CUDA-блок представляет собой набор потоков с общей памятью и возможностью синхронизации друг с другом. CUDA-поток — это минимальная исполнительная единица в данной архитектуре, все потоки выполняются параллельно на разных ядрах GPU.

В разработанном решении один воксель обрабатывается одним CUDA-блоком, для каждого вокселя рассматривается конфигурируемое число случайных гипотез, каждая гипотеза оценивается в одном CUDA-потоке этого блока. Таким образом все воксели и гипотезы обрабатываются параллельно, и все потоки одного блока могут синхронизироваться для выбора наилучшей гипотезы (рис. 10). Обработка проис-

²<https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/> (дата обращения 16 июня 2024 г.)

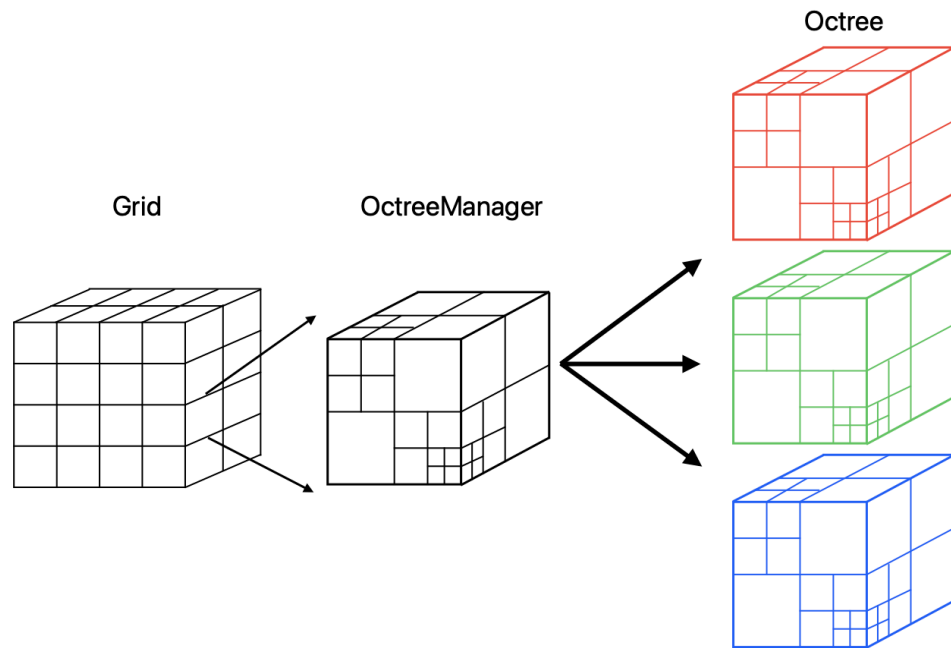


Рис. 8: Хранение множества наблюдений в одном объекте **Grid**. Внутри одного объекта **OctreeManager** все октодеревья имеют одинаковую схему деления

ходит в несколько этапов:

1. Получение точек всех вокселей каждого рассматриваемого наблюдения и конкатенация точек в один массив.
2. Определение индексов, где в полученном массиве начинается и заканчивается сегмент, соответствующий каждому отдельному вокселю, а также индексов начала данных каждого наблюдения
3. Копирование массива точек и массива индексов начала сегментов в память GPU.
4. В каждом потоке для соответствующего этому потоку сегмента точек генерируется случайная гипотеза — плоскость. Далее рассчитывается число точек, соответствующих этой гипотезе (таких, которые находятся не дальше заданного пользователем расстояния от этой плоскости).
5. Среди всех гипотез для каждого CUDA-блока выбирается та, для

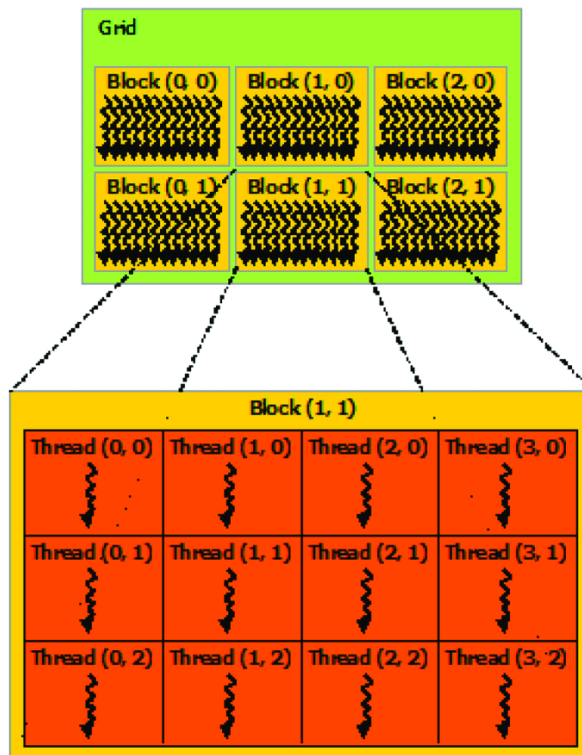


Рис. 9: Блоки и потоки архитектуры CUDA. [21]

которой число соответствующих ей точек наибольшее.

6. Для каждого CUDA-блока генерируется маска, которая фильтрует точки, не соответствующие выбранной гипотезе.
7. Полученные для каждого вокселя маски применяются к соответствующим наборам точек.

Кроме того, реализована возможность обработки конфигурируемого числа наблюдений за раз, чтобы избежать переполнения памяти GPU при обработке большого числа наблюдений.

4.2. Реализация

Библиотека реализована с использованием языка Python. Выбор этого языка обусловлен читаемостью кода, а также простотой работы со сложными типами данных. В качестве основного инструмента для хранения и обработки облаков точек используется библиотека NumPy [2],

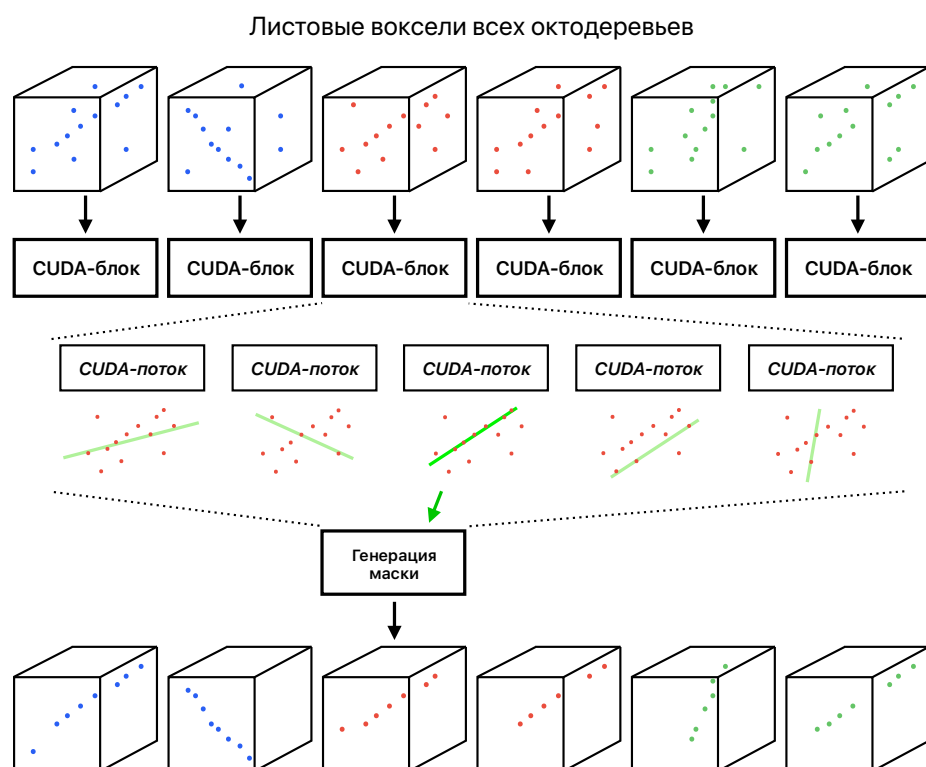


Рис. 10: Принцип работы алгоритма RANSAC в рамках библиотеки

это обусловлено ее высокой производительностью при операциях с многомерными массивами данных. Кроме того, данная библиотека предоставляет эффективные инструменты для работы с облаками точек.

Визуализация реализована с использованием библиотеки k3d [14]. Основное отличие этой библиотеки от аналогов заключается в возможности сохранения html файла с интерактивной визуализацией, что обеспечивает гибкость в предоставлении результатов исследования сцены. Этот формат файла обеспечивает взаимодействие с интерактивной визуализацией непосредственно в веб-браузере, позволяя пользователям легко и интуитивно взаимодействовать с ней, а также облегчая возможность ее передачи, поскольку просмотр визуализации не требует установки специализированного программного обеспечения.

Алгоритм RANSAC был реализован с помощью библиотеки Numba [16]. Эта библиотека позволяет реализовывать ядра (функции, запускаемые на GPU на множестве потоков с целью добиться высокой производительности) на языке Python с дальнейшей их компиляцией

в набор инструкций для GPU. Кроме того эта библиотека частично поддерживает функциональность библиотеки NumPy для реализации CUDA-ядер.

В репозитории настроен конвейер CI с помощью GitHub Actions [11] в виде проверки unit-тестов с помощью PyTest [38] и проверки форматирования с использованием линтеров flake8 [25] и black [37]. Кроме того, настроена публикация pip-пакета в репозиторий PyPI [26]³ с помощью GitHub Actions [11] и Poetry [24].

³<https://pypi.org/project/octreelib/> (дата обращения 16 июня 2024 г.)

5. Экспериментальное сравнение

Для оценки производительности и эффективности разработанной библиотеки для обработки октодеревьев было проведено экспериментальное исследование ее работы. В данном разделе будет представлено экспериментальное исследование производительности разработанного решения.

5.1. Условия экспериментов

Исследование проводилось с использованием наборов данных, указанных в разделе 3.4. Эксперименты проводились с использованием машины со следующей конфигурацией: Intel i5-8300H (8) @ 4.000GHz, 8 GiB RAM, Nvidia GeForce GTX 1050 Mobile 4GiB VRAM, Ubuntu 22.04.

5.2. Цель и исследовательские вопросы

Цель данного экспериментального исследования заключается в оценке производительности библиотеки и объема используемых ей ресурсов во время различных этапов работы при обработке объемных наборов данных. Кроме того, исследуется влияние на производительность конфигурируемых параметров, как то: разрешение сетки и максимальная глубина октодерева. Для достижения цели были поставлены перечисленные ниже исследовательские вопросы.

RQ1. Каково время выполнения различных этапов работы библиотеки на рассматриваемых наборах данных? Какова затрачиваемая память во время обработки наборов данных? Эта информация покажет, насколько эффективно и оптимизировано решение относительно времени выполнения и затрачиваемой памяти.

RQ2. Каково влияние разрешения сетки и максимальной возможной глубины октодерева на производительность библиотеки и используемый объем памяти? Эта информация поможет определить оптимальные значения этих параметров для конкретных задач и обеспечит лучшую производительность библиотеки в дальнейшем ее использовании.

RQ3. Каково время выполнения алгоритма RANSAC с использованием реализации на CUDA по сравнению с использованием реализации CPU на примере Open3D и SkLearn? Эта информация позволит сравнить эффективность работы разработанного решения с существующими часто применяющимися решениями.

5.3. Исследование работы библиотеки

В таблице 3 для KITTI и таблице A.1 (Приложение A) для Hilti указано среднее время обработки данных облаков точек на этапе **деления октодеревьев** в пространственной сетке после изначальной вставки одного облака точек. В таблице 4 для KITTI и таблице A.2 (Приложение A) для Hilti указано среднее время **распределения облаков точек**, относящихся к 9, 17, 33, 65 наблюдениям, в листовые узлы поделённых октодеревьев. Данные указаны в зависимости от наименьшего размера облака точек в листе октодерева и изначального размера вокселя в сетке. Полученные результаты указывают на то, что процесс анализа становится наиболее времязатратным при обработке сцен, требующих большого начального размера вокселя в сочетании с небольшим размером листового узла октодерева.

В таблице 2 указана память, используемая во время обработки разного количества облаков точек без учёта памяти, используемой на хранение самих точек. Объем памяти, используемый библиотекой во время работы существенно превышает объем памяти, занимаемый исходными данными. Поскольку SLAM-задачи нередко предполагают обработку больших объемов данных, обработка большого числа наблюдений может оказаться ресурсоемкой с точки зрения требуемого объема оперативной памяти.

Для исследования эффективности работы реализации алгоритма RANSAC было проведено по 10 запусков на наборе данных KITTI для следующих параметров:

- Число рассматриваемых наблюдений: 100;
- Длина ребра вокселя пространственной сетки (в метрах): 64;

9 облаков (0.67 КБ)		Наименьший размер вокселя					
		2	4	8	16	32	64
Изначальный размер вокселя	2	40.89					
	4	42.84	30.84				
	8	45.47	32.19	27.53			
	16	46.71	33.23	28.13	26.24		
	32	47.46	33.92	28.65	26.44	25.80	
	64	47.90	34.32	29.01	26.68	25.96	25.64
17 облаков (1.17 КБ)		Наименьший размер вокселя					
		2	4	8	16	32	64
Изначальный размер вокселя	2	74.32					
	4	78.35	57.00				
	8	83.01	59.54	51.17			
	16	85.32	61.47	52.15	48.98		
	32	86.50	62.59	53.00	49.33	48.18	
	64	87.23	63.26	53.52	49.69	48.46	47.89
33 облака (2.17 КБ)		Наименьший размер вокселя					
		2	4	8	16	32	64
Изначальный размер вокселя	2	141.19					
	4	146.32	110.35				
	8	154.36	113.82	99.74			
	16	159.38	117.10	101.39	95.70		
	32	162.57	119.20	102.77	96.50	94.14	
	64	166.16	121.46	104.13	97.26	94.67	93.58
65 облаков (4.20 КБ)		Наименьший размер вокселя					
		2	4	8	16	32	64
Изначальный размер вокселя	2	264.56					
	4	265.33	211.47				
	8	276.28	215.47	193.13			
	16	284.41	220.91	194.46	185.97		
	32	289.63	224.44	196.96	187.62	183.22	
	64	296.82	228.95	199.71	189.68	183.87	182.28

Таблица 2: Объем памяти (в **мегабайтах**), затрачиваемой на обработку **9, 17, 33, 65** наблюдений **KITTI** (сверху вниз) в зависимости от начального размера вокселя сетки (в метрах) и наименьшего возможного размера облака точек в итоговом узле октодерева (в метрах) в сравнении с объемом памяти, занимаемой исключительно облаком точек.

KITTI		Наименьший размер вокселя					
		2	4	8	16	32	64
Изначальный размер вокселя	2	0.063 (0.000)					
	4	0.378 (0.042)	0.023 (0.000)				
	8	0.616 (0.040)	0.258 (0.007)	0.011 (0.001)			
	16	0.860 (0.044)	0.504 (0.036)	0.230 (0.001)	0.005 (0.000)		
	32	1.061 (0.028)	0.722 (0.029)	0.464 (0.028)	0.222 (0.001)	0.004 (0.000)	
	64	1.281 (0.007)	0.941 (0.030)	0.676 (0.003)	0.441 (0.002)	0.222 (0.001)	0.003 (0.000)

Таблица 3: Среднее значение и стандартное отклонение времени (в секундах), затрачиваемого на деление октодеревьев для облаков точек набора данных **KITTI**.

- Наименьший возможный размер облака точек при делении узлов октодерева (в метрах): 2, 4, 8, 16, 32, 64.

Для сравнения были использованы реализации Open3D и SkLearn, поскольку в этих библиотеках доступны высокоуровневые интерфейсы для языка программирования Python.

На рис. 11 изображен график, показывающий **время выполнения алгоритма RANSAC** с использованием Open3D на CPU и с использованием новой реализации на GPU. При малом наименьшем размере вокселя общее число листовых вокселей высокое, а значит для обработки необходимо значительное количество запусков алгоритма. При рассмотрении 71 000 листовых вокселей реализация на GPU занимает в ≈ 11.7 раз меньше времени, чем реализация Open3D, и в ≈ 52.3 раз меньше, чем реализация SkLearn. При рассмотрении 1 000 вокселей, время на обработку с использованием новой реализации в ≈ 4.6 и ≈ 3.4 раз меньше времени обработки с помощью Open3D и SkLearn соответственно. На рис. 12 изображен график, показывающий занимаемый объем памяти GPU при обработке различного числа облаков точек с использованием реализации RANSAC на GPU. Объем памяти, используемый

KITTI 9 облаков		Наименьший размер облака в вокселе					
		2	4	8	16	32	64
Изначальный размер вокселя	2	1.575 (0.011)					
	4	2.504 (0.012)	1.225 (0.045)				
	8	3.427 (0.034)	2.084 (0.044)	1.128 (0.030)			
	16	4.284 (0.018)	2.937 (0.047)	1.988 (0.035)	1.081 (0.009)		
	32	5.155 (0.038)	3.831 (0.044)	2.851 (0.029)	1.952 (0.028)	1.069 (0.006)	
	64	6.058 (0.036)	4.702 (0.063)	3.716 (0.038)	2.807 (0.038)	1.931 (0.009)	1.062 (0.009)
KITTI 17 облаков		Наименьший размер облака в вокселе					
		2	4	8	16	32	64
Изначальный размер вокселя	2	3.100 (0.046)					
	4	4.851 (0.039)	2.443 (0.046)				
	8	6.722 (0.026)	4.146 (0.050)	2.245 (0.052)			
	16	8.473 (0.051)	5.780 (0.023)	3.917 (0.042)	2.133 (0.010)		
	32	10.280 (0.080)	7.635 (0.046)	5.682 (0.048)	3.887 (0.047)	2.094 (0.014)	
	64	12.066 (0.087)	9.390 (0.059)	7.462 (0.100)	5.603 (0.040)	3.850 (0.014)	2.073 (0.005)
KITTI 33 облака		Наименьший размер облака в вокселе					
		2	4	8	16	32	64
Изначальный размер вокселя	2	6.139 (0.059)					
	4	9.482 (0.064)	4.892 (0.054)				
	8	13.002 (0.092)	8.094 (0.086)	4.510 (0.063)			
	16	16.651 (0.094)	11.597 (0.074)	7.894 (0.051)	4.341 (0.045)		
	32	20.229 (0.151)	15.168 (0.053)	11.455 (0.069)	7.886 (0.036)	4.218 (0.014)	
	64	24.019 (0.046)	18.860 (0.089)	15.066 (0.058)	11.477 (0.066)	7.872 (0.033)	4.195 (0.018)
KITTI 65 облаков		Наименьший размер облака в вокселе					
		2	4	8	16	32	64
Изначальный размер вокселя	2	11.744 (0.123)					
	4	17.463 (0.170)	9.578 (0.043)				
	8	23.959 (0.129)	15.585 (0.087)	8.822 (0.064)			
	16	30.868 (0.153)	22.188 (0.076)	15.293 (0.048)	8.530 (0.055)		
	32	38.027 (0.149)	29.359 (0.148)	22.277 (0.057)	15.499 (0.093)	8.340 (0.040)	
	64	45.450 (0.149)	36.711 (0.112)	29.555 (0.104)	22.707 (0.107)	15.205 (0.078)	8.270 (0.033)

Таблица 4: Среднее значение и стандартное отклонение времени (в секундах), затрачиваемого на вставку и распределение облаков точек в листовые воксели поделенных октодеревьев для **9, 17, 33, 65** облаков набора данных **KITTI**.

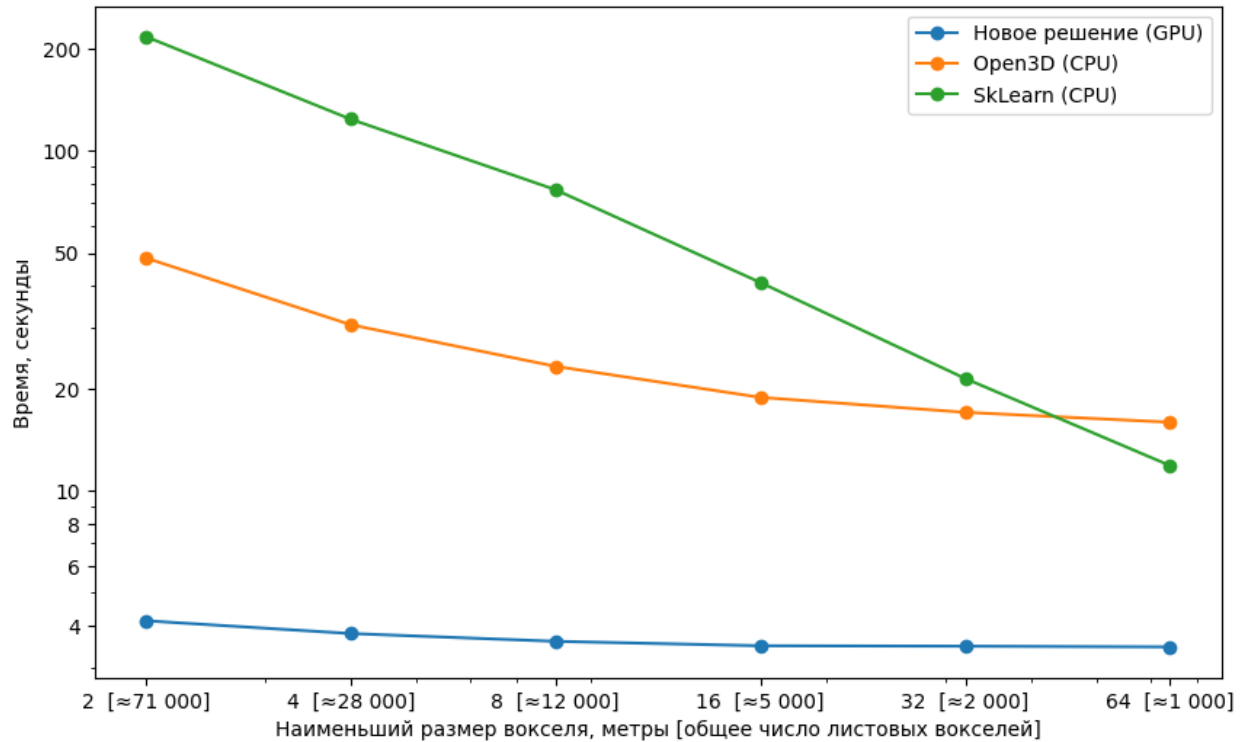


Рис. 11: Время выполнения RANSAC на GPU и CPU (с помощью библиотек Open3D и SkLearn) в зависимости от наименьшего возможного размера вокселя. В квадратных скобках указано общее число рассматриваемых вокселей, соответствующее данному наименьшему размеру вокселя.

реализацией существенно превышает объем памяти, занимаемый исходными данными. Таким образом, при обработке большого числа облаков, возможности параллелизации в этом случае будут ограничены памятью GPU.

5.4. Выводы

На основании результатов экспериментального исследования ответим на поставленные вопросы.

RQ1. Результаты эксперимента показывают, что время обработки данных растет с увеличением начального размера вокселя и уменьшением минимального размера вокселя.

RQ2. Полученные в результате эксперимента данные указывают на то, что память, требуемая для обработки данных увеличивается при

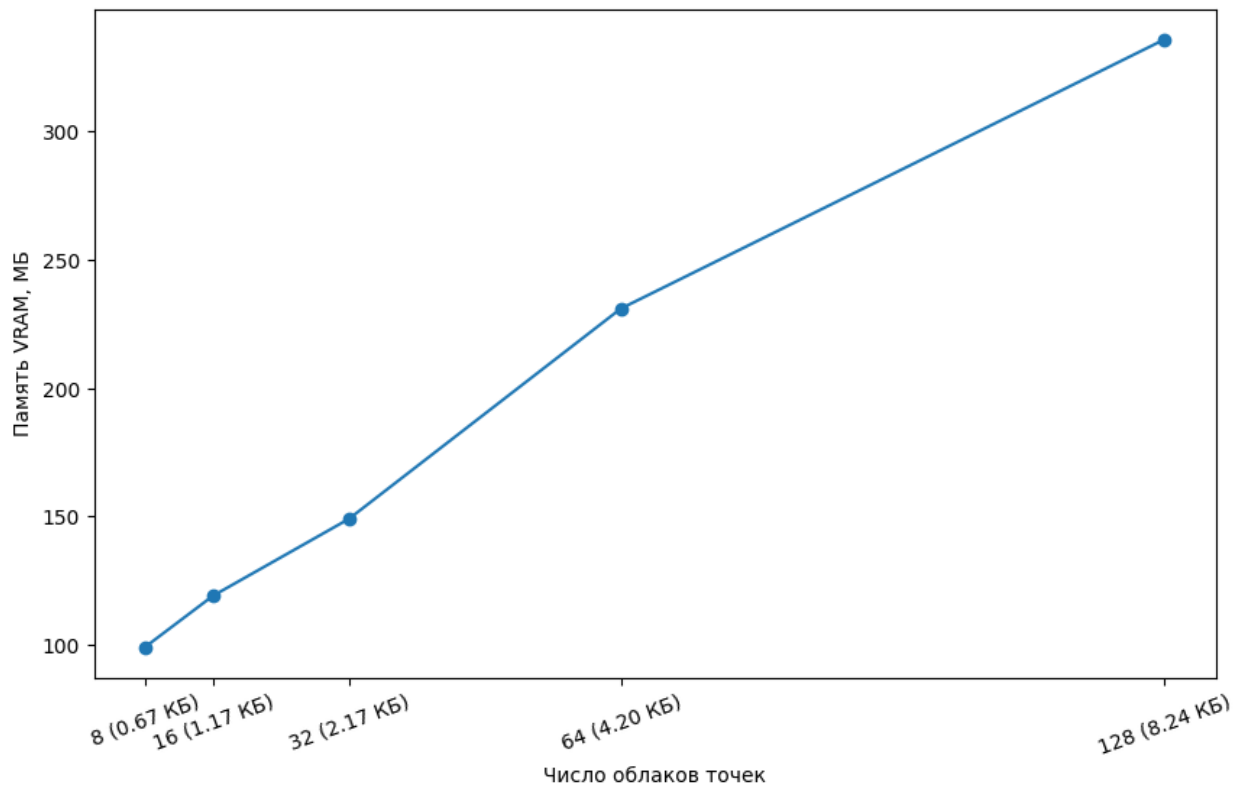


Рис. 12: Объем памяти VRAM, затрачиваемый на обработку различного числа облаков точек с использованием новой реализации RANSAC.

большем изначальном размере вокселя и меньшем минимальном возможном размере вокселя. Объем используемой памяти растет линейно, что может повлечь трудности при использовании библиотеки с большим числом облаков точек.

RQ3. Результаты эксперимента показывают, что реализация на GPU с параллельной обработкой множества вокселей обеспечивает существенное сокращение времени обработки по сравнению с реализациями на CPU. Наиболее сильно различие проявляется при большом числе рассматриваемых вокселей. Требуемая для обработки память также растет линейно, что при обработке большого числа облаков может ограничить возможности параллелизации.

6. Заключение

Цель работы была достигнута, были выполнены следующие задачи.

- Сформирован список функциональных требований к библиотеке и проведен анализ существующих решений относительно необходимой функциональности.
- Проведен обзор существующих решений, реализующих работу с октодеревьями, сформирован список наборов данных для измерения производительности решения.
- Спроектирована и реализована библиотека для работы с октодеревьями в соответствии с функциональными требованиями.
- Проведено экспериментальное исследование производительности разработанного решения.

Реализация библиотеки доступна в репозитории на GitHub: <https://github.com/prime-slam/octreelib>⁴.

Pip-пакет доступен в репозитории PyPI: <https://pypi.org/project/octreelib/>⁵.

⁴Дата обращения: 16 июня 2024 г.

⁵Дата обращения: 16 июня 2024 г.

Список литературы

- [1] Aijazi Ahmad Kamal, Checchin Paul, Trassoudaine Laurent. Segmentation Based Classification of 3D Urban Point Clouds: A Super-Voxel Based Approach with Evaluation // [Remote Sensing](#). — 2013. — Vol. 5, no. 4. — P. 1624–1650. — URL: <https://www.mdpi.com/2072-4292/5/4/1624>.
- [2] Array programming with NumPy / Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt et al. // [Nature](#). — 2020. — sep. — Vol. 585, no. 7825. — P. 357–362. — URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [3] Choi Sunglok, Kim Taemin, Yu Wonpil. [Performance evaluation of RANSAC family](#). — Vol. 24. — 2009. — 01.
- [4] Chum Ondřej, Matas Jiří. Optimal randomized RANSAC // [IEEE Transactions on Pattern Analysis and Machine Intelligence](#). — 2008. — Vol. 30, no. 8. — P. 1472–1482.
- [5] Computing homography with RANSAC algorithm: a novel method of registration / Xiaowei Li, Yue Liu, Yongtian Wang, Dayuan Yan // [Electronic Imaging and Multimedia Technology IV](#) / SPIE. — Vol. 5637. — 2005. — P. 109–112.
- [6] Cranch James. octrees. — <https://github.com/jcranch/octrees>. — 2013.
- [7] Efficient and Probabilistic Adaptive Voxel Mapping for Accurate Online 3D SLAM / Chongjian Yuan, Wei Xu, Xiyuan Liu et al. // [CoRR](#). — 2021. — Vol. abs/2109.07082. — arXiv : [2109.07082](#).
- [8] Fischler Martin A., Bolles Robert C. [Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography](#) // [Readings in Computer Vision](#) / Ed. by Martin A. Fischler, Oscar Firschein. —

- San Francisco (CA) : Morgan Kaufmann, 1987. — P. 726–740. — URL: <https://www.sciencedirect.com/science/article/pii/B9780080515816500702>.
- [9] Forstmann Sven. Sparse Voxel Octree Raycasting. — <https://github.com/sp4cerat/Sparse-Voxel-Octree-Raycasting>. — 2015.
 - [10] Geiger Andreas, Lenz Philip, Urtasun Raquel. [Are we ready for autonomous driving? The KITTI vision benchmark suite](#) // 2012 IEEE Conference on Computer Vision and Pattern Recognition. — 2012. — P. 3354–3361.
 - [11] GitHub. GitHubActions // Site. — URL: <https://github.com/features/actions>.
 - [12] Hilti-Oxford Dataset: A Millimeter-Accurate Benchmark for Simultaneous Localization and Mapping / Lintong Zhang, Michael Helmberger, Lanke Frank Tarimo Fu et al. // [IEEE Robotics and Automation Letters](#). — 2023. — Vol. 8, no. 1. — P. 408–415.
 - [13] Hogg Michael. pyoctree. — <https://github.com/mhogg/pyoctree>. — 2018.
 - [14] Trzęsio Artur, Kostur Marcin, Gandor Tomasz, Mattone Thomas. K3D-Jupyter. — <https://k3d-jupyter.org>. — 2015.
 - [15] Kornia: an Open Source Differentiable Computer Vision Library for PyTorch / E. Riba, D. Mishkin, D. Ponsa et al. // Winter Conference on Applications of Computer Vision. — 2020. — URL: <https://arxiv.org/pdf/1910.02190.pdf>.
 - [16] Lam Siu Kwan, Pitrou Antoine, Seibert Stanley. [Numba: a LLVM-based Python JIT compiler](#) // Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. — LLVM '15. — New York, NY, USA : Association for Computing Machinery, 2015. — 6 p. — URL: <https://doi.org/10.1145/2833157.2833162>.

- [17] Limberger Frederico A., Oliveira Manuel M. Real-time detection of planar regions in unorganized point clouds // [Pattern Recognition](#). — 2015. — Vol. 48, no. 6. — P. 2043–2053. — URL: <https://www.sciencedirect.com/science/article/pii/S0031320315000072>.
- [18] Xu Binbin, Li Wenbin, Tzoumanikas Dimos et al. MID-Fusion: Octree-based Object-Level Multi-Instance Dynamic SLAM. — 2019. — 1812.07976.
- [19] Meagher Donald. Geometric modeling using octree encoding // [Computer graphics and image processing](#). — 1982. — Vol. 19, no. 2. — P. 129–147.
- [20] Miller Andrew Edmund, Jain Vishal, Mundy Joseph L. [Real-time rendering and dynamic updating of 3-d volumetric data](#) // [Proceedings of the 4th Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-4](#). — ACM International Conference Proceeding Series. — 2011. — apr. — 4th Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-4 ; Conference date: 05-03-2011 Through 05-03-2011.
- [21] NVIDIA, Vingelmann Péter, Fitzek Frank H.P. CUDA, release: 10.2.89. — 2020. — URL: <https://developer.nvidia.com/cuda-toolkit>.
- [22] OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees / Armin Hornung, Kai M. Wurm, Maren Bennewitz et al. // [Autonomous Robots](#). — 2013. — Software available at <https://octomap.github.io>. URL: <https://octomap.github.io>.
- [23] Octree-based region growing for point cloud segmentation / Anh-Vu Vo, Linh Truong-Hong, Debra F. Laefer, Michela Bertolotto // [ISPRS Journal of Photogrammetry and Remote Sensing](#). — 2015. — Vol. 104. — P. 88–100. — URL: <https://www.sciencedirect.com/science/article/pii/S0924271615000283>.

- [24] Poetry. — Site. — 2018. — URL: <https://python-poetry.org/>.
- [25] PyCQA. flake8 // Site. — URL: <https://flake8.pycqa.org/en/latest/>.
- [26] Python Package Index - PyPI. — Python Software Foundation. — URL: <https://pypi.org/> (дата обращения: 2021-03-28).
- [27] Riegler Gernot, Ulusoy Ali Osman, Geiger Andreas. OctNet: Learning Deep 3D Representations at High Resolutions. — 2017. — 1611.05009.
- [28] Robot Operating System 2: Design, architecture, and uses in the wild / Steven Macenski, Tully Foote, Brian Gerkey et al. // [Science Robotics](#). — 2022. — Vol. 7, no. 66. — P. eabm6074. — URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.
- [29] Rusu Radu Bogdan, Cousins Steve. 3D is here: Point Cloud Library (PCL) // IEEE International Conference on Robotics and Automation (ICRA). — Shanghai, China, 2011. — May 9-13.
- [30] Sattler Torsten et al. RansacLib - A Template-based *SAC Implementation. — 2019. — URL: <https://github.com/tsattler/RansacLib>.
- [31] Schnabel Ruwen, Wahl Roland, Klein Reinhard. Efficient RANSAC for point-cloud shape detection // Computer graphics forum / Wiley Online Library. — Vol. 26. — 2007. — P. 214–226.
- [32] Scikit-learn: Machine Learning in Python / F. Pedregosa, G. Varoquaux, A. Gramfort et al. // Journal of Machine Learning Research. — 2011. — Vol. 12. — P. 2825–2830.
- [33] Sun Bo, Mordohai Philippos. Oriented Point Sampling for Plane Detection in Unorganized Point Clouds // CoRR. — 2019. — Vol. abs/1905.02553. — arXiv : [1905.02553](#).
- [34] The CGAL Project. CGAL User and Reference Manual. — 5.6.1 edition. — CGAL Editorial Board, 2024. — URL: <https://doc.cgal.org/5.6.1/Manual/packages.html>.

- [35] Wilhelms Jane, Van Gelder Allen. Octrees for faster isosurface generation // ACM Transactions on Graphics (TOG). — 1992. — Vol. 11, no. 3. — P. 201–227.
- [36] Zhou Qian-Yi, Park Jaesik, Koltun Vladlen. Open3D: A Modern Library for 3D Data Processing // arXiv:1801.09847. — 2018.
- [37] psf. Black Formatter. — GitHub. — URL: <https://github.com/psf/black>.
- [38] Krekel Holger, Oliveira Bruno, Pfannschmidt Ronny et al. pytest. — 2004. — URL: <https://github.com/pytest-dev/pytest>.
- [39] Мищенко С. А. Разработка инфраструктурной части инструмента для визуализации ориентиров в SLAM алгоритмах // Сайт кафедры Системного Программирования Математико-Механического факультета СПбГУ. — 2023.

Приложение А

Hilti		Наименьший размер вокселя					
		2	4	8	16	32	64
Изначальный размер вокселя	2	0.0033 (0.0000)					
	4	0.0316 (0.0005)	0.0015 (0.0000)				
	8	0.0663 (0.0012)	0.0350 (0.0002)	0.0006 (0.0000)			
	16	0.1002 (0.0005)	0.0690 (0.0003)	0.0344 (0.0000)	0.0004 (0.0000)		
	32	0.1382 (0.0006)	0.1061 (0.0003)	0.0713 (0.0002)	0.0366 (0.0001)	0.0003 (0.0000)	
	64	0.1771 (0.0003)	0.1450 (0.0002)	0.1095 (0.0002)	0.0752 (0.0001)	0.0003 (0.0000)	0.0003 (0.0000)

Таблица А.1: Среднее значение и стандартное отклонение времени (в секундах), затрачиваемого на деление октодеревьев для облаков точек набора данных **Hilti**.

Hilti 9 облаков		Наименьший размер облака в вокселе					
		2	4	8	16	32	64
Изначальный размер вокселя	2	0.167 (0.001)					
	4	0.278 (0.001)	0.159 (0.001)				
	8	0.420 (0.001)	0.303 (0.003)	0.157 (0.000)			
	16	0.571 (0.007)	0.445 (0.001)	0.301 (0.001)	0.155 (0.000)		
	32	0.729 (0.001)	0.602 (0.001)	0.457 (0.001)	0.311 (0.000)	0.158 (0.000)	
	64	0.900 (0.018)	0.766 (0.001)	0.620 (0.001)	0.473 (0.000)	0.159 (0.000)	0.159 (0.000)
Hilti 17 облаков		Наименьший размер облака в вокселе					
		2	4	8	16	32	64
Изначальный размер вокселя	2	0.334 (0.017)					
	4	0.554 (0.017)	0.314 (0.001)				
	8	0.833 (0.020)	0.582 (0.001)	0.308 (0.001)			
	16	1.126 (0.018)	0.874 (0.003)	0.407 (0.000)	0.308 (0.000)		
	32	1.442 (0.007)	1.186 (0.021)	0.713 (0.002)	0.312 (0.001)	0.312 (0.001)	
	64	1.758 (0.002)	1.497 (0.002)	1.029 (0.001)	0.311 (0.000)	0.311 (0.000)	0.312 (0.001)
Hilti 33 облака		Наименьший размер облака в вокселе					
		2	4	8	16	32	64
Изначальный размер вокселя	2	0.930 (0.024)					
	4	1.440 (0.028)	0.894 (0.001)				
	8	2.189 (0.022)	1.452 (0.021)	0.872 (0.001)			
	16	3.008 (0.018)	2.243 (0.024)	1.664 (0.007)	0.862 (0.001)		
	32	3.920 (0.019)	3.115 (0.003)	2.550 (0.021)	0.882 (0.001)	0.881 (0.001)	
	64	4.807 (0.018)	4.016 (0.019)	3.427 (0.002)	0.880 (0.001)	0.880 (0.002)	0.883 (0.007)
Hilti 65 облаков		Наименьший размер облака в вокселе					
		2	4	8	16	32	64
Изначальный размер вокселя	2	1.601 (0.028)					
	4	2.665 (0.033)	1.529 (0.023)				
	8	3.966 (0.047)	2.843 (0.031)	1.487 (0.002)			
	16	5.367 (0.033)	4.214 (0.040)	2.855 (0.023)	1.487 (0.007)		
	32	6.904 (0.045)	5.726 (0.036)	4.377 (0.014)	1.518 (0.007)	1.511 (0.007)	
	64	8.425 (0.037)	7.245 (0.034)	5.886 (0.022)	1.510 (0.008)	1.507 (0.001)	1.507 (0.001)

Таблица A.2: Среднее значение и стандартное отклонение времени (в секундах), затрачиваемого на вставку и распределение облаков точек в листовые воксели поделенных октодеревьев для **9, 17, 33, 65** облаков набора данных **Hilti**.