Моделирование вычислительных, телекоммуникационных, управляющих и социально-экономических систем

DOI: 10.18721/JCSTCS.12408
УДК 004.021

ФОРМИРОВАНИЕ ОКТОДЕРЕВА ПО ОБЛАКУ ТОЧЕК ПРИ ОГРАНИЧЕНИИ ОБЪЁМА ОПЕРАТИВНОЙ ПАМЯТИ

К.О. Беляевский

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

Лазерное сканирование – современный и активно развивающийся метод дистанционного зондирования. Результат лазерного сканирования представляет собой облако точек, которое кроме координат каждой точки может содержать цвет точки и другие атрибуты. Одной из особенностей технологии лазерного сканирования, обусловившей её популярность, является возможность получения достаточно плотного облака точек, что определяет высокую точность цифрового представления геометрии объекта сканирования. В некоторых случаях облака точек могут содержать миллиарды точек, для хранения которых необходимы сотни гигабайт. Загрузка и обработка таких колоссальных объёмов данных требует больших временных и вычислительных ресурсов. Распространенным подходом является построение октодерева для ускорения операций пространственного поиска и группировки близких в пространстве точек. Использование такого октодерева совместно с внешней памятью открывает возможность ограничения объёма потребляемой оперативной памяти. В статье представлен метод построения октодерева с использованием двухуровневой системы кеширования участков облака точек. Предложены способ организации процесса построения структуры данных и способ анализа эффективности метода с помощью вычислительного эксперимента. Приведены результаты анализа эффективности предложенного метода.

Ключевые слова: трёхмерное сканирование, облако точек, структуры данных, октодерево, потребление памяти, кеширование, измерение производительности.

Ссылка при цитировании: Беляевский К.О. Формирование октодерева по облаку точек при ограничении объёма оперативной памяти // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2019. Т. 12. № 4. С. 97–110. DOI: 10.18721/JCSTCS.12408

GENERATING OCTREES FROM THE POINT CLOUD IN CONDITIONS OF LIMITED RANDOM ACCESS MEMORY VOLUME

K.O. Beliaevskii

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

Laser scanning is a modern and actively developing remote sensing method. Laser scanning results are represented by point clouds, which can contain not only the coordinates of each point, but also the color of the points and other attributes. One

of laser scanning technology features, which caused its popularity, is the ability to obtain a sufficiently dense cloud of points. Thanks to this function, laser scanning allows you to determine the digital representation of the geometry of the scanned object with high accuracy. In some cases, point clouds can contain billions of points that require hundreds of gigabytes of data storage. Loading and processing such enormous amounts of data requires significant time and computational resources. A common approach is to build an octree to speed up the spatial search operations and group points close in space. Using such an octree together with external memory opens the possibility of limiting the amount of RAM consumed. This article presents the method of an octree generating using a two-tier cache system for caching sections of the point cloud. The method of organizing the process of data structure building is proposed, and the approach to the analysis of the proposed method efficiency on the basis of a computational experiment is given. The results of the proposed method efficiency analysis are presented.

Keywords: three-dimensional scanning, point cloud, data structures, octree, memory consumption, caching, performance measurement.

Citation: Beliaevskii K.O. Generating octrees from the point cloud in conditions of limited random access memory volume. St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunications and Control Systems, 2019, Vol. 12, No. 4, Pp. 97–110. DOI: 10.18721/JCSTCS.12408

Введение

Анализ предметной области. Процесс обработки данных лазерного сканирования состоит из двух основных этапов: 1) загрузка данных и подготовка данных к последующей обработке; 2) обработка данных с помощью конкретного алгоритма [1]. Существующие исследования сосредоточены преимущественно на решении задач, относящихся ко второму этапу, и посвящены вопросам оптимизации обработки данных [2–4].

Часто при обработке облака точек необходимо использовать информацию о пространственных отношениях между точками, например, близость (соседство) точек или попадание точек в область определенной пространственной фигуры (куба, сферы и т. п.). В силу специфики технологии лазерного сканирования, точки в полученном облаке расположены рядом в соответствии с траекторией луча сканирования. Вместе с тем, даже зная модель сканера, предсказать распределение точек не представляется возможным, т. к. их порядок в облаке может быть изменен в результате обработки каким-либо алгоритмом или при объединении нескольких облаков точек. Пример подобных облаков приведен на рис. 1 (в верхней части рисунка точки расположены участками в результате обработки, нижняя часть рисунка иллюстрирует результат сведения данных с различных сканеров; оттенки серого цвета отражают нарастание индекса точки). Точки в облаке могут быть распределены непредсказуемым образом, и операции поиска требуют полного перебора облака точек, т. е. вычислительная сложность задачи составляет O(n) [5].

Для ускорения вычислений обычно используют так называемую пространственную индексацию облака точек. Результатом подобной операции будет структура данных, описывающая объём, занимаемый облаком точек, в виде набора непересекающихся регионов, и позволяющая идентифицировать каждую точку в облаке как принадлежащую к определенному региону. Таким образом, пространственный поиск будет выполняться не во всем облаке, а в интересующем нас регионе, что позволит уменьшить сложность поиска, которая в данном случае составит O(log n) [5]. Будем называть подобную структуру данных структурой разбиения пространства.

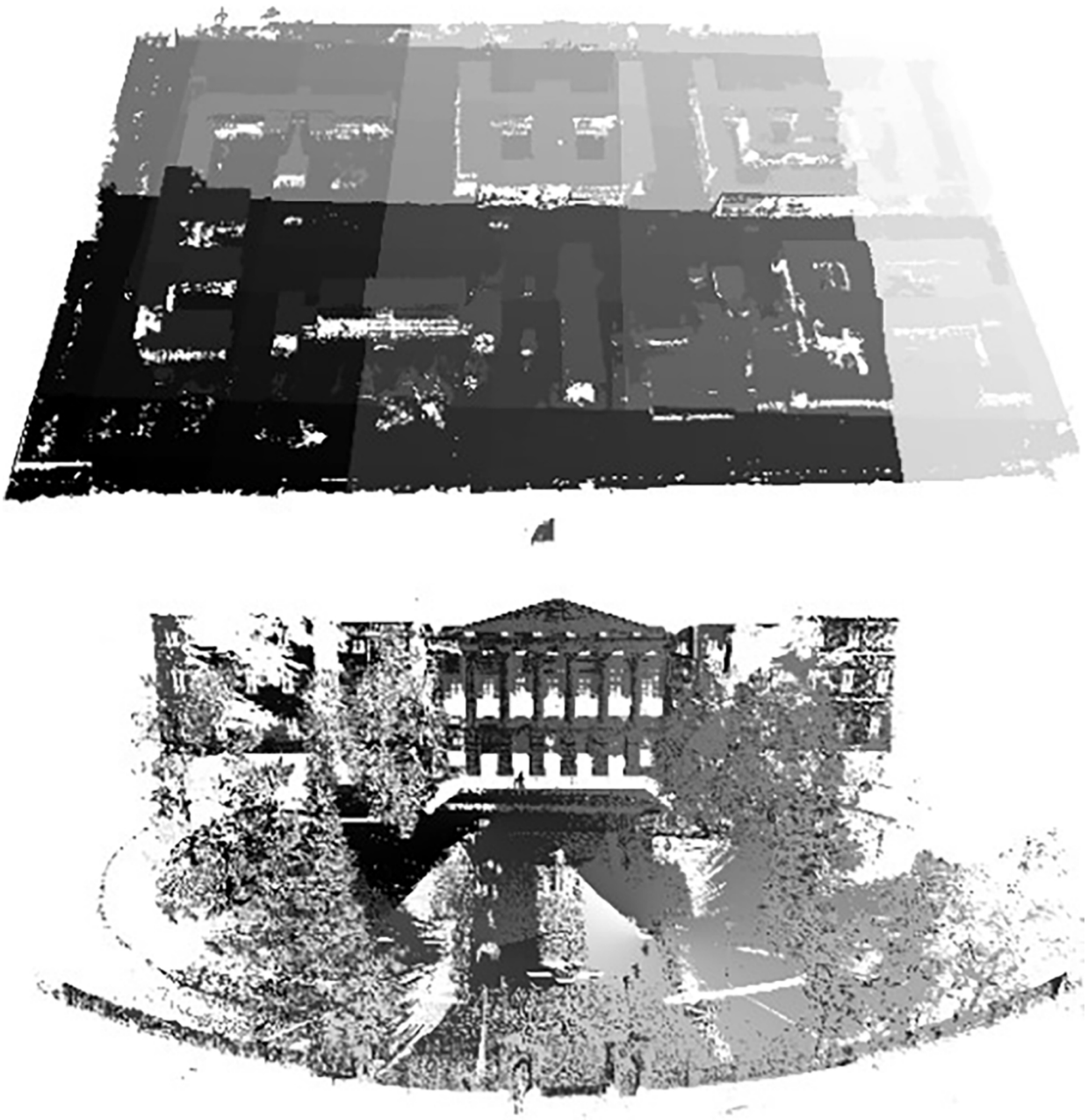

Рис. 1. Визуализация исходных индексов точек в облаке
Fig. 1. Visualization of the input indices of the point clouds

Одна из ключевых проблем при работе с облаками точек – это противоречие между большим объёмом информации, который необходимо обработать, и ограниченностью вычислительных ресурсов [6]. Размеры облаков точек нередко превышают доступные объёмы оперативной памяти вычислительных устройств, что влечет за собой необходимость обрабатывать облако точек по частям, а это, в свою очередь, существенно усложняет процесс обработки облака точек.

Постановка задачи. В данной статье рассматривается гипотеза о том, что совместное использование структуры разбиения пространства и системы кеширования блоков позволит значительно сократить затраты оперативной памяти при обработке облака точек. В результате подобного объединения появится возможность выпол-

нять кеширование по пространственному признаку с возможностью выгрузки неиспользуемых блоков на жесткий диск.

В качестве структуры разбиения пространства в настоящей статье выбраны октарные деревья [7], ввиду наличия у них ярко выраженного разбиения облака точек на блоки. Подобное свойство позволяет производить кеширование отдельных блоков, что является более эффективным подходом, чем кеширование отдельных точек. При наличии такой системы появляется возможность загружать в оперативную память ограниченный набор точек, полностью загружать облако точек не требуется.

Октарные деревья

Октарные деревья широко используются при обработке облаков точек, полученных методом лазерного сканирования, и находят применение в таких областях, как распознавание объектов [8–10], сегментация [11, 12], выделение примитивов [13], реконструкция поверхности [14], расчет нормалей [15] и пр.

Структура октарного дерева приведена на рис. 2. Узел октодерева хранит в себе информацию о восьми потомках, а также вспомогательную информацию, которая может использоваться в процессе построения или обработки облака точек (например, координаты узла, его размер, уровень и т. п.). Листья октарного дерева хранят в себе информацию о точках, которые попадают в ячейку конкретного листа на данном уровне разбиения. Листья также могут содержать вспомогательную информацию.

Формирование октодерева производится при помощи рекурсивного добавления точек, начиная с корневого узла. На этапе построения при добавлении точек в любой узел производится проверка на удовлетворение критерия разбиения для данного узла. В качестве критерия разбиения было выбрано превышение максимального количества точек в узле. Выбор максимального количества точек в узле обусловлен решаемой задачей, анализ возможных значений приведен в разделе «Результаты тестирования». При выполнении указанного критерия производится разбиение узла на восемь меньших октантов, точки родительского узла распределяются между ними по пространственному признаку. Критерием остановки служит превышение максимальной глубины дерева. Максимально допустимая глубина дерева составляет 21 уровень, что обусловлено способом идентификации узла при помощи 64-битного целочисленного значения (используется по 3 бита на уровень). Количество узлов на максимальном уровне разбиения составляет 8^{21}.

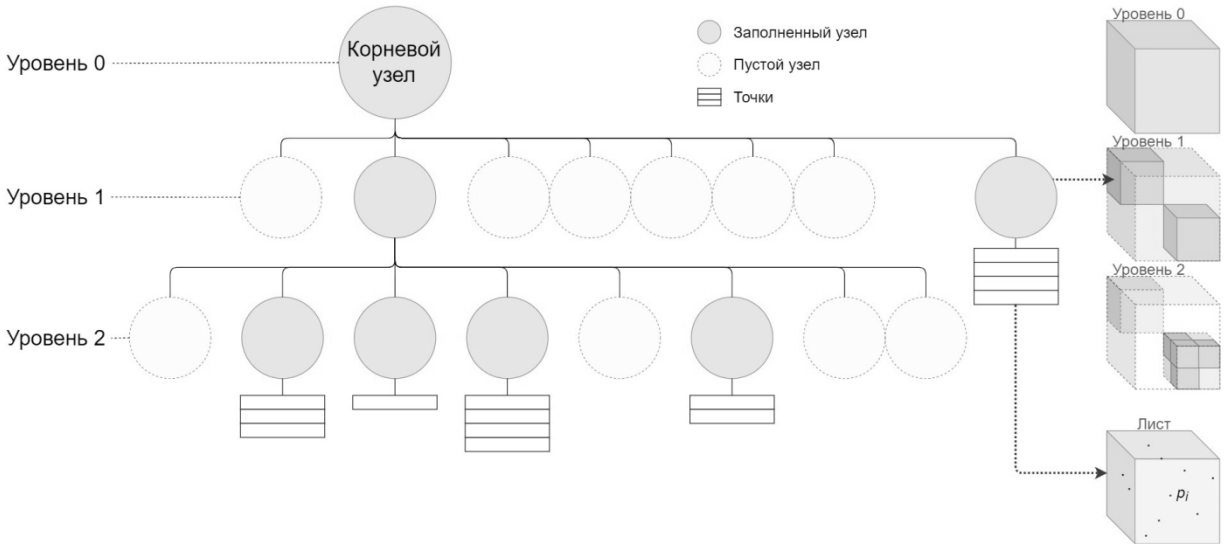

Рис. 2. Структура октарного дерева
Fig. 2. The octree structure

В большинстве опубликованных работ построение октарного дерева выполняется в оперативной памяти. Это означает, что узлы, листья и принадлежащие им точки также будут храниться в оперативной памяти. Такое решение существенно ускоряет построение дерева, однако делает невозможным обработку облаков точек, не помещающихся в оперативную память.

Использование системы кеширования в процессе формирования октарного дерева

Для разрешения указанного выше противоречия предлагается модифицировать октарное дерево для работы с блоками точек, которые могут находиться как в оперативной памяти, так и на жестком диске. Для того чтобы не усложнять реализацию октарного дерева, предлагается оригинальная система кеширования облаков точек, назначением которой будет управление размещением блоков точек. Подобная система позволит создавать/удалять, а также подгружать/выгружать блоки точек, поддерживая при этом заданный уровень потребления оперативной памяти за счет выгрузки неиспользуемых блоков.

Предлагаемая система кеширования должна обеспечивать работу с блоками точек в условиях ограничений по объёму доступной оперативной памяти. На практике это означает, что при превышении определенного лимита потребляемой памяти, часть блоков должна быть выгружена на жесткий диск. Стоит отметить, что при подобном подходе к ограничению потребляемой памяти лимитируется только та память, в которой располагаются точки из облака точек. Иерархия октодерева, а также прочие данные программы, не относящиеся к облаку точек, не подлежат ограничению.

Для реализации подобного поведения достаточно двухуровневой системы кеширования: L_1 – оперативная память; L_2 – жесткий диск.

При этом в оперативной памяти остаются только актуальные блоки, а устаревшие вытесняются на уровень L_2, т. е. сохраняются на жестком диске. Для вытеснения блоков можно использовать различные политики: First In First Out (FIFO), Least Recently Used (LRU), Least Frequently Used (LFU) и прочие [16]. В нашем случае хорошо подходит политика LRU, т. к. она, с одной стороны, отличается простотой реализации (обычно представлена двусвязным списком), а, с другой стороны, удовлетворяет требованиям по удалению устаревших блоков. Стоит отметить, что эффективность выбранной политики вытеснения может варьироваться в зависимости от вида операции, производимой над облаком точек.

Для обеспечения взаимодействия с октодеревом система кеширования должна поддерживать следующие операции: создание блока точек; добавление точек к блоку точек; выгрузка блока точек; удаление блока точек; финализация блока точек.

Для ускорения построения октодерева была организована параллельная обработка данных. Так, например, файловые операции, необходимые для работы кеша на уровне L_2, выполняются в отдельном потоке. Управляющие команды поступают при помощи асинхронной очереди сообщений. Ниже приведены описания таких операций.

1. Сохранение блока. При выгрузке блока из L_1 в очередь поступает команда на сохранение данного блока в соответствующий файл, при этом основной поток не тратит время на ожидание завершения операции.

2. Загрузка блока. В случае загрузки в очередь поступает команда на загрузку файла, после чего выполняется ожидание завершения выполнения команды.

3. Финализация блока. В процессе активной работы с блоками точек они хранятся в виде отдельных файлов (в связи с возможностью изменения их размеров). По завершении работы с блоком он присоединяется к общему файлу (данная операция была названа *финализацией*). Большое количество операций с файлами небольшого размера существенно снижает производительность [17] системы, поэтому подобную реализацию в общем случае не рекомендуется использовать. В то же время в данной работе используется алгоритм построения октарного дерева, позволяющий в большинстве случаев сразу присоединить блок к финальному файлу.

Процесс построения октарного дерева также реализуется при помощи системы кеширования: при добавлении точек в узел происходит обращение к соответствующей ячейке кеша с целью добавления новых точек. Стоит отметить отличия предложенного алгоритма накопления точек в узле от классической реализации октодерева [18]: точки после разбиения не спускаются ниже по иерархии дерева, а остаются в узле. Последующие точки пропускаются через такой узел насквозь. Такой подход позволяет существенно снизить количество файловых операций за счет двух факторов: во-первых, точно известно, что записей в данный блок не будет, и можно присоединить его к финальному файлу; во-вторых, в процессе построения практически отсутствуют операции чтения и чтения/перезаписи данных (т. к. больше нет необходимости при разбиении выполнять для накопленного блока точек цепочку операций считывание-разбиение-запись). Иначе говоря, при подобном подходе разрешается хранение блоков точек в ветвях дерева, что положительно влияет на скорость построения, однако может снизить производительность операций поиска.

Пример алгоритма построения октодерева на следующей странице в листинге.

Предложенный агоритм имеет следующие преимущества:

- отсутствие необходимости подгружать весь блок данных в L_1 при добавлении новых точек;
- возможность финализации завершенных блоков;
- возможность асинхронной работы с файлами.

Таким образом, при построении октодерева данные подгружаются только в случае финализации узла.

Методика анализа характеристик предложенного способа формирования октодерева

Для проведения анализа эффективности предложенного способа построения октодерева разработан набор тестов, представленный в табл. 1. Программная реализация тестов и октодерева выполнена на языке C++, т. к. этот язык позволяет производить низкоуровневые операции с памятью, что необходимо при работе с большими объёмами данных.

Выбор тестов обусловлен необходимостью подтверждения работы в условиях имеющихся ограничений по объёму оперативной памяти. Для анализа работоспособности системы на различных облаках точек, выбора оптимального размера узла, а также для сравнения режимов работы с кешированием и без разработан тест 1. Для доказательства ограничения потребляемой памяти, а также для демонстрации процесса формирования октарного дерева разработаны тесты 2–3. Для анализа производительности поиска по октодереву разработан тест 4.

Существует множество форматов для хранения облаков точек, например: OBJ, PLY, XYZ, PCD, LAS и многие др. [19]. В качестве входного формата облаков точек для тестирования среди них был выбран формат LAS, т. к. он широко используется и имеет открытые исходные коды [20].

Для тестирования предложенной системы было выбрано четыре облака точек. Облако точек под названием five.las выбрано для анализа поведения системы при малом количестве точек. Облако точек под названием polytech.las выбрано для исследования возможностей системы на больших облаках точек с низкой плотностью. Облако точек под названием smolny.las выбрано для исследования возможностей системы на больших облаках точек с высокой плотностью и совмещением данных с нескольких сканеров. Облако точек под названием molodezhnoe.las представляет собой наиболее сложный случай с точки зрения обработки, т. к. оно обладает сверхбольшим размером (60 Гб), а также имеет высокую плотность за счет совмещения данных сканирования. Характеристики облаков приведены в табл. 2, внешний вид облаков точек представлен на рис. 3.

```
// Пример добавления новой точки в дерево (оригинальный алгоритм оперирует блоками точек)
void InsertPoint(Node* node, Point point) {
   if(node->isFull()) // Если нельзя добавить больше точек
   {
      // Получение индекса подузла, в который попадает точка
      int childIndex = GetChildIndex(node, point);
      // Добавление точки в подузел
      InsertPoint(GetChild(node, childIndex), point);
   }
   else
   {
      // Добавление точки в узел
      AppendPoint(node, point);
      // Если узел заполнен - финализация
      if(node->isFull())
         FinalizeNode(node);
   }
}
void AppendPoint(Node* node, Point point) {
   if(*Has L1 Cache Entry*) // Если узел загружен в RAM
   {
      GetL1Entry(node)->push_back(point); // Добавление новой точки в узел
   }
   else
   {
      // Если узел не загружен в RAM - просто дописываем точку в конец файла
      // Для отправки команды используем асинхронную очередь
      // Запись будет производиться в другом потоке
      L2Queue->newCommand(AppendPointCmd(node, point));
   }
}
void FinalizeNode(Node* node) {
   // Загрузка данных в L1 и отправка на финализацию
   L2Queue->newCommand(FinalizeNodeCmd(GetL1Entry(node)));
}
```

Формирование октодерева с использованием системы кеширования
An octree formation with the usage of the caching system

Таблица 1

Список тестов

Table 1

The list of the tests

Название теста	Описание теста
Тест 1. Время построения	Анализ зависимости времени построения от размера узла и объёма кеша
Тест 2. Потребление памяти	Анализ потребления памяти (RAM, HDD) в процессе построения
Тест 3. Файловые операции	Анализ операций чтения/записи в процессе построения
Тест 4. Скорость поиска	Анализ зависимости скорости поиска точек в дереве от размера узла и объёма кеша

Таблица 2

Облака точек для тестирования

Table 2

Point clouds for testing

Название	Кол-во точек, млн	Размер, Гб	Плотность, точек/м^2
five.las	1,2	0,03	18641
polytech.las	51,6	1,8	267
smolny.las	126	4,3	23656
molodezhnoe.las	1560	53,4	55097

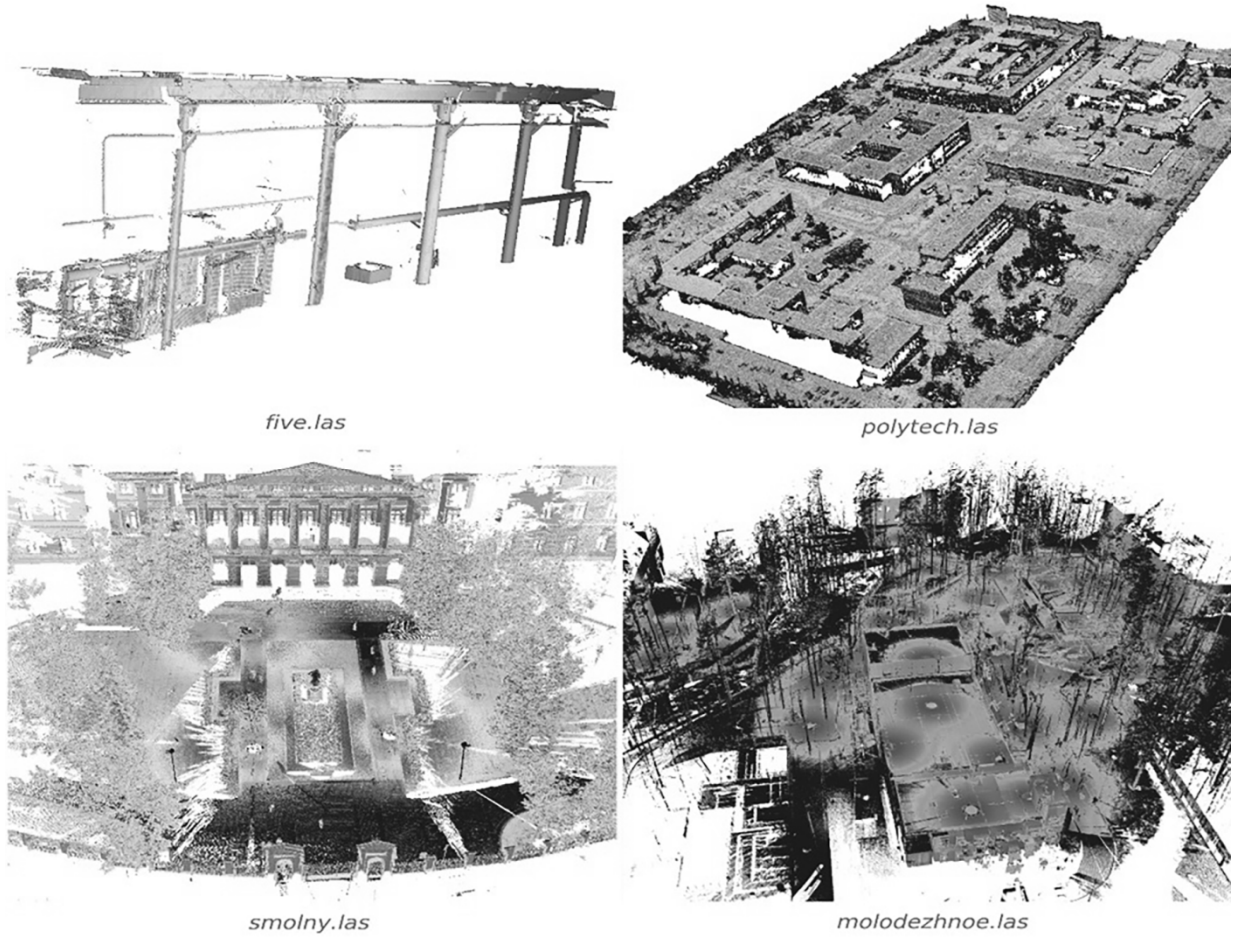

Рис. 3. Внешний вид используемых облаков точек с раскраской по плотности (шкала плотности индивидуальна для каждого облака)

Fig. 3. Visual representation of the used point clouds with density coloring (density scale is individual for each cloud)

Конфигурация тестового стенда:

CPU: Intel® Core™ i7-7820HQ CPU @ 2.90 GHz

RAM: 2 x SODIMM DDR4 2400 MHz (2x8 GB)

SSD: Samsung SSD 850 PRO 1 TB

OS: Ubuntu 18.04

File system: Ext4

Результаты тестирования

Тест 1. Время построения дерева. На рис. 4 представлены результаты анализа

производительности метода построения октодерева при использовании системы кеширования. Для этого проведены измерения времени построения дерева для различных облаков точек в оперативной (RAM) и файловой (HDD) памяти путем изменения размера узла и размера кеша. Такой тест позволит выяснить оптимальные параметры дерева, а также продемонстрировать разницу в скорости построения дерева в оперативной памяти и с использованием жесткого диска.

На графике зависимости скорости построения дерева от максимального размера узла видно, что построение дерева значительно замедляется при использовании узлов маленького размера. Это вызвано прямой зависимостью между размером узла и глубиной дерева: чем меньше узел, тем больше глубина, на корторую спускается очередная точка из облака при построении. В случае с файловыми операциями размер узла влияет особенно сильно, т. к. приводит к значительному увеличению количества создаваемых файлов.

Как следует из графика зависимости скорости построения дерева от используемого объёма кеша, увеличение объёма кеша немного ускоряет процесс построения дерева. Тем не менее, значительного вклада это не вносит, т. к. добавляемые точки обычно разбросаны по облаку неравномерно, что понижает эффективность кеширования.

По результатам проведенного теста можно утверждать, что увеличение размера узла или объёма кеша приводит к ускорению построения октодерева. Также можно заметить, что характер изменений практически не зависит от исследуемого облака точек. Однако для выбора наилучшего значения размера узла необходимо также провести измерения скорости поиска по построенному дереву. Что касается выбора подходящего размера кеша, то рекомендуется использовать максимально допустимое для конкретной системы значение, но не превышающее размер наибольшего обрабатываемого облака точек.

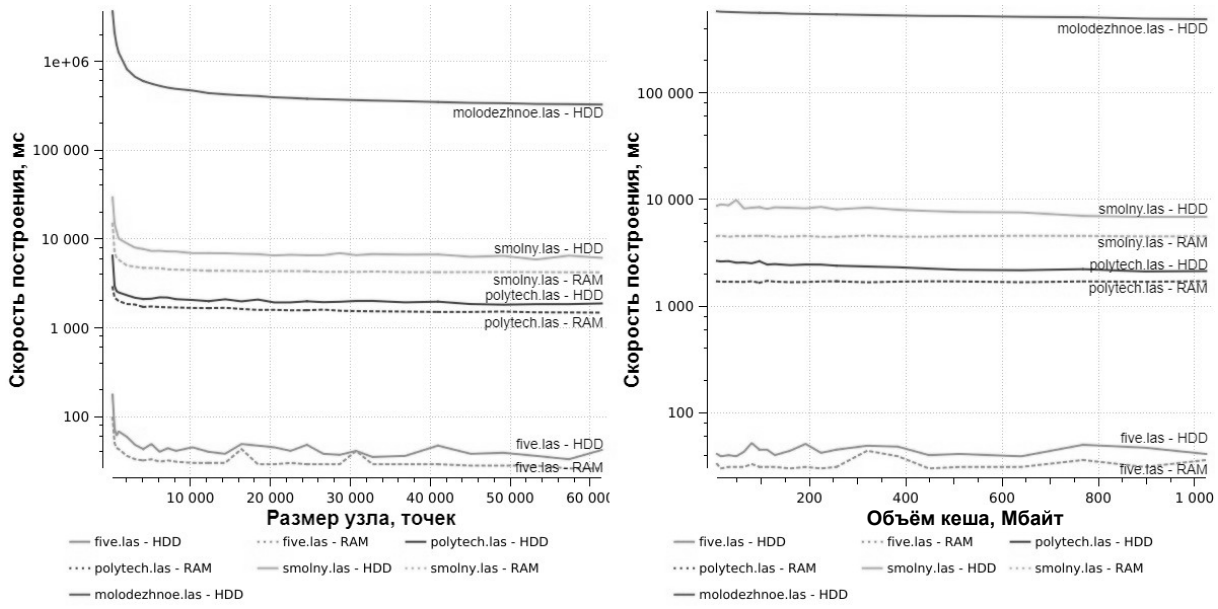

Рис. 4. Зависимость скорости построения дерева от максимального размера узла (слева) и от используемого объёма кеша (справа)

Fig. 4. Dependence of the speed of an octree formation on the maximum node size (left) and on the cache volume used (right)

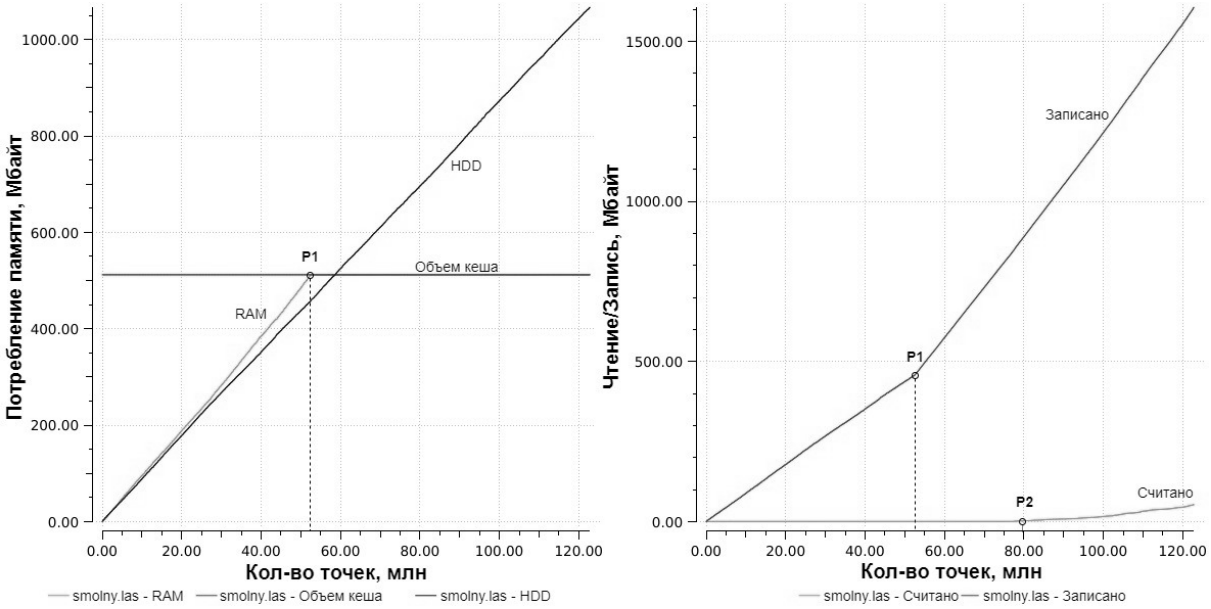

Рис. 5. Потребление памяти в процессе построения дерева (слева) и файловые операции в процессе построения дерева (справа)

Fig. 5. Memory consumption in the process of an octree formation (left) and file operations in the process of an octree formation (right)

Тесты 2 и 3. Потребление памяти и файловые операции. Далее проверялась корректность работы системы кеширования в условиях ограничения потребляемой оперативной памяти. Результаты выполнения данного теста иллюстрируются графиками потребления оперативной и файловой памяти. На рис. 5 приведены графики, снятые в процессе построения октодерева, и отражающие зависимость объёма потребления L_1 (RAM) и L_2 (HDD) от количества загруженных точек (слева) и файловые операции в процессе построения октодерева (справа).

Как видно из приведенных графиков, сначала происходит заполнение кеша до значения его ёмкости (точка P1), параллельно с выгрузкой заполненных узлов на жесткий диск. Расход оперативной памяти остается в пределах заданных значений за счет использования дискового пространства. Также на рисунке можно увидеть, как растет интенсивность записи на жесткий диск после заполнения кеша (точка P1), и как возрастает количество операций считывания по мере приближения к концу процесса построения (точка P2). Последнее вызвано загрузкой недозаполненных узлов, выгруженных из кеша ранее.

Результаты данного теста позволяют сделать вывод об успешности объединения октодерева и системы кеширования, т. к. потребление памяти оставалось в пределах заданных лимитов на протяжении всего процесса обработки облака точек.

Тест 4. Скорость поиска. Если в предыдущих тестах оценивались влияние размеров узла и объёма кеша на скорость построения, то в данном тесте целью будет измерение скорости поиска в уже построенном дереве. Данный тест позволит оценить, насколько замедляется скорость поиска при больших размерах узлов, а также насколько увеличение объёмов кеша ускоряет поиск.

На рис. 6 приведен график зависимости скорости поиска от размеров узлов (слева) и график зависимости скорости поиска от объёмов кеша (справа). В процессе тестирования для каждого исследуемого параметра производится большое количество операций поиска точек, находящихся в определенном радиусе от случайной точки из облака. В качестве результата используется среднее значение времени поиска.

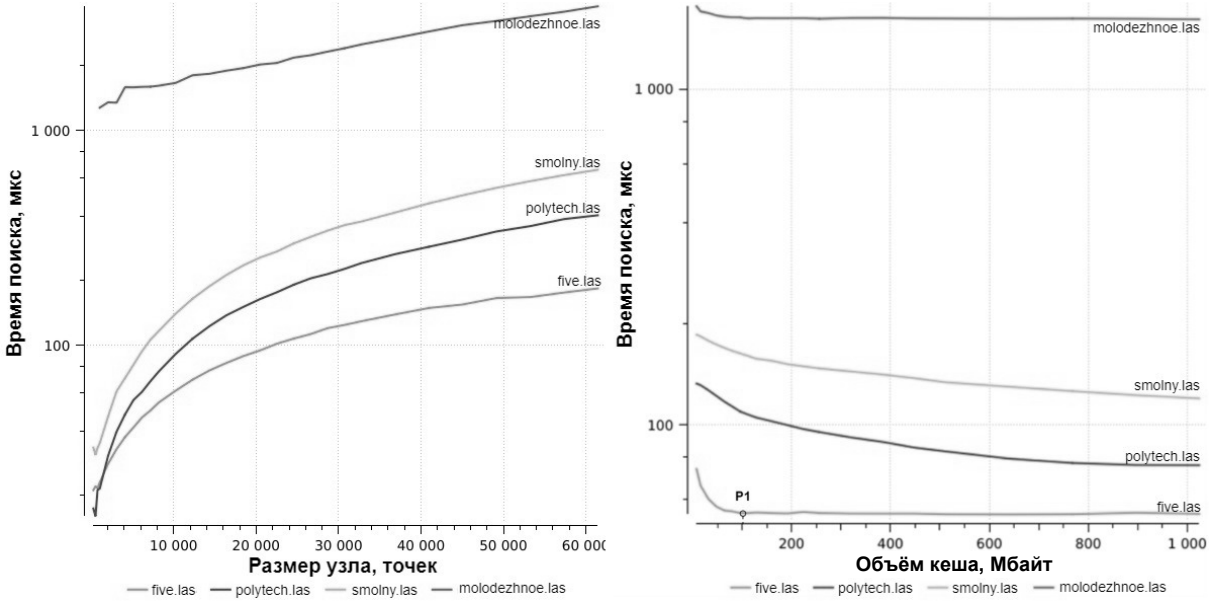

Рис. 6. Скорость поиска в зависимости от размера узла (слева) и в зависимости от объёма кеша (справа)
Fig. 6. Search speed depending on the node size (left) and depending on the cache volume (right)

Как видно из графика зависимости скорости поиска от размера узла, чем больше количество точек в узле, тем медленнее происходит поиск. Данный результат обусловлен механизмом поиска: после проверки попадания узла в радиус поиска алгоритм выполняет проверку для всех точек, входящих в узел (т. к. попадание узла не гарантирует попадание всех точек узла). Соответственно, чем больше точек в узле, тем большее количество проверок отдельных точек предстоит провести. В случае, когда размер узла невелик, большая часть точек отсекается на этапе более быстрой проверки узлов.

На графике зависимости скорости поиска от объёма кеша можно наблюдать ускорение операций поиска при увеличении объёма кеша. Стоит отметить, что оптимальный объём кеша возрастает с увеличением размера облака точек: это можно увидеть по тому, как насыщается график five.las в точке P1, а также по динамике изменения графиков polytech.las и smolny.las.

По результатам проведенного теста можно утверждать, что увеличение объёма кеша положительно влияет на скорость поиска, в то время как при увеличении объёма узла скорость поиска замедляется. Таким образом, при выборе подходящего размера узла следует также руководствоваться необходимой скоростью поиска. Нами было выбрано значение 2048 точек, как компромисс между скоростью построения и скоростью поиска. Способ выбора подходящего размера кеша остается неизменным.

Сравнение с распространенными октодеревьями. Сравнение исследуемой системы с распространенными библиотеками и программами, предназначенными для обработки и визуализации облаков точек, доказало, что предложенная система не уступает по производительности. Для сравнения были выбраны следующие программные компоненты:

Point Cloud Library (PCL) [21] – библиотека с открытым исходным кодом, широко использующаяся для обработки облаков точек;

CloudCompare [22] – программное обеспечение с открытым исходным кодом, предназначенное для загрузки, обработки и визуализации облаков точек.

Сравнение производилось путем измерения времени, необходимого на загрузку облака точек и построения октарного дерева. Для загрузки использовались облака точек, приведенные в табл. 3. В качестве параметров нашего дерева использовались: размер узла – 2048 точек, объём кеша – 512 Мбайт. Данные параметры выбраны, как наиболее подходящие с точки зрения компромисса между скоростью построения и поиска. Результаты сравнения приведены в табл. 3.

Таблица 3

Сравнение предложенной реализации с аналогами

Table 3

Comparison of the proposed implementation with analogues

Название облака	Размер облака	Предложенная реализация		PCL		Cloud Compare	
		CPU	RAM	CPU	RAM	CPU	RAM
five.las	33 Мбайт	60 мс	33,4 Мбайт	122 мс	66 Мбайт	435 мс	114 Мбайт
polytech.las	1,8 Гбайт	2,6 с	517 Мбайт	4,6 с	1,8 Гбайт	18,2 с	1,6 Гбайт
smolny.las	4,3 Гбайт	11,6 с	586 Мбайт	12,5 с	4,3 Гбайт	58,5 с	3,6 Гбайт
molodezhnoe.las	53,4 Гбайт	14,9 мин	986 Мбайт	–	–	–	–

В результате сравнения выявлено, что предложенная оригинальная реализация октодерева значительно превосходит по скорости построения дерева остальные программные компоненты. Помимо существующих методов оптимизации в предложенной системе, подобные различия могут быть вызваны различиями в размерах узлов, политиках вставки, разрешениях дерева и прочих скрытых параметрах. При этом по времени построения дерева данная реализация, при условии кеширования на жесткий диск, как минимум не уступает приведенным аналогам, работающим в оперативной памяти.

Стоит также отметить потребление оперативной памяти: несмотря на то, что общее потребление памяти выше установленного лимита (в основном за счет данных иерархии октодерева, содержащих информацию о миллионах узлов, и не учитывающихся при вытеснении), общее потребление памяти значительно сократилось по сравнению с аналогами.

Заключение

Рассмотрена гипотеза о том, что совместное использование структуры разбиения пространства и системы кеширования блоков позволит значительно сократить затраты оперативной памяти при обработке облака точек.

Для подтверждения гипотезы представлен метод построения октодерева с использованием двухуровневой системы кеширования участков облака точек в условиях ограничений на использование оперативной памяти, способ организации процесса построения октодерева, способ анализа эффективности метода вычислительным экспериментом.

Преимущества предложенного способа формирования октодерева показаны с использованием сравнительного анализа зависимости скорости построения и поиска по дереву от используемого размера узла дерева и объёма кеша. Сравнение с существующими программными решениями показало, что предложенный способ формирования октодерева позволяет значительно сократить объём потребляемой оперативной памяти, а также не уступает по производительности построения деревьев аналогам, работающим в оперативной памяти, что подтверждает выдвинутую гипотезу.

Экспериментальное исследование позволило выявить следующие зависимости: увеличение размеров узла положительно влияет на скорость построения дерева; снижение размеров узла положительно влияет на скорость поиска по дереву; увеличение размеров кеша положительно влияет как на скорость построения, так и на скорость поиска по дереву.

Полученные результаты могут использоваться в алгоритмах обработки больших облаков точек при нехватке оперативной памяти. Стоит отметить, что ввиду зависимости скорости построения и скорости поиска от размеров узла, данный подход покажет наибольшую производительность при использовании в алгоритмах, принимающих на вход крупные блоки точек (например, для визуализации).

СПИСОК ЛИТЕРАТУРЫ

1. **Bi Z.M., Wang L.** Advances in 3D data acquisition and processing for industrial applications // Robotics and Computer-Integrated Manufacturing. 2010. Vol. 26. Pp. 403–413.

2. **Bellekens B., Spruyt V., Berkvens R., Weyn M.** A survey of rigid 3D point cloud registration algorithms // AMBIENT 2014: the 4th Internat. Conf. on Ambient Computing, Applications, Services and Technologies. 2014. Pp. 8–13.

3. **Nguyen A., Le B.** 3D point cloud segmentation: A survey // The 6th IEEE Conf. on Robotics, Automation and Mechatronics. 2013. Pp. 225–230.

4. **Grilli E., Menna F., Remondino F.** A review of point clouds segmentation and classification algorithms // The Internat. Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences. 2017. Vol. 42. P. 339.

5. **Savage J.E.** Models of computation. Exploring the power of computing Addison-Wesley. Reading, MA. 1998.

6. **Liu X., Meng W., Guo J., Zhang X.** A survey on processing of large-scale 3D point cloud // Internat. Conf. on Technologies for E-Learning and Digital Entertainment. 2016. Pp. 267–279.

7. **Elseberg J., Borrmann D., Nьchter A.** One billion points in the cloud – An octree for efficient processing of 3D laser scans // ISPRS Journal of Photogrammetry and Remote Sensing. 2013. Vol. 76. Pp. 76–88.

8. **Wang J., Lindenbergh R. Menenti, M.** SigVox – A 3D feature matching algorithm for automatic street object recognition in mobile laser scanning point clouds // ISPRS Journal of Photogrammetry and Remote Sensing. 2017. Vol. 128. Pp. 111–129.

9. **Sveier A., Kleppe A.L., Tingelstad L., Egeland O.** Object detection in point clouds using conformal geometric algebra // Advances in Applied Clifford Algebras. 2017. Vol. 27. № 3. Pp. 1961–1976.

10. **Wang L., Xu Y., Li Y.** A Voxel-based 3D building detection algorithm for airborne LiDAR point clouds // J. of the Indian Society of Remote Sensing. 2019. Vol. 47. № 2. Pp. 349–358.

11. **Zhong L., Cheng L., Xu H., Wu Y., Chen Y., Li M.** Segmentation of individual trees from TLS and MLS Data // IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing. 2017. Vol. 10. № 2. Pp. 774–787.

12. **Xu Y., Tuttas S., Hoegner L., Stilla U.** Voxel-based segmentation of 3D point clouds from construction sites using a probabilistic connectivity model // Pattern Recognition Letters. 2018. Vol. 102. Pp. 67–74.

13. **Xu Y., Tuttas S., Hoegner L., Stilla U.** Geometric primitive extraction from point clouds of construction sites using VGS // IEEE Geoscience and Remote Sensing Letters. 2017. Vol. 14. № 3. Pp. 424–428.

14. **Tang Y., Feng J.** Multi-scale surface reconstruction based on a curvature-adaptive signed distance field // Computers and Graphics (Pergamon). 2018. Vol. 70. Pp. 28–38.

15. **Zhao R., Pang M., Liu C., Zhang Y.** Robust normal estimation for 3D LiDAR point clouds in urban environments // Sensors (Switzerland). Vol. 19. No. 5.

16. **Farooqui M.Z., Shoaib M., Khan M.Z.** A comprehensive survey of page replacement algorithms // IJARCET. 2014.

17. **Fu S., He L., Huang C., Liao X.K., Li K.** Performance optimization for managing massive numbers of small files in distributed file systems // IEEE Transactions on Parallel and Distributed Systems. 2015. Vol. 26. Pp. 3433–3448.

18. **Samet H.** An overview of quadtrees, octrees, and related hierarchical data structures // Theoretical Foundations of Computer Graphics and CAD. 1988. Pp. 51–68.

19. **Vo A.V., Laefer D.F., Bertolotto M.** Airborne laser scanning data storage and indexing: state-of-the-art review // Internat. J. of Remote Sensing. 2016. Vol. 37. Pp. 6187–6204.

20. Photogrammetry & remote sensing A.S.P.R.S. LAS specification version 1.4--R13 // URL: https://www.asprs.org/a/society/committees/standards/LAS_1_4_r13.pdf (Дата обращения: 08.10.2019).

21. **Rusu R.B., Cousins S.** Point cloud library (pcl) // 2011 IEEE Internat. Conf. on Robotics and Automation. 2011. Pp. 1–4.

22. **Girardeau-Montaut D.** Cloud compare 3D point cloud and mesh processing software // Open Source Project. 2015.

Статья поступила в редакцию 10.11.2019.

REFERENCES

1. **Bi Z.M., Wang L.** Advances in 3D data acquisition and processing for industrial applications. *Robotics and Computer-Integrated Manufacturing*, 2010, Vol. 26, Pp. 403–413.

2. **Bellekens B., Spruyt V., Berkvens R., Weyn M.** A survey of rigid 3D point cloud registration algorithms. *AMBIENT 2014: the Fourth International Conference on Ambient Computing,*

Applications, Services and Technologies, 2014, Pp. 8–13.

3. **Nguyen A., Le B.** 3D point cloud segmentation: A survey. *The 6th IEEE Conference on Robotics, Automation and Mechatronics*, 2013, Pp. 225–230.

4. **Grilli E., Menna F., Remondino F.** A review of point clouds segmentation and classification algorithms. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017, Vol. 42, P. 339.

5. **Savage J.E.** *Models of computation. Exploring the power of computing Addison-Wesley*. Reading, MA, 1998.

6. **Liu X., Meng W., Guo J., Zhang X.** A survey on processing of large-scale 3D point cloud. *International Conference on Technologies for E-Learning and Digital Entertainment*, 2016, Pp. 267–279.

7. **Elseberg J., Borrmann D., Nьchter A.** One billion points in the cloud – An octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2013, Vol. 76, Pp. 76–88.

8. **Wang J., Lindenbergh R. Menenti, M.** SigVox – A 3D feature matching algorithm for automatic street object recognition in mobile laser scanning point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2017, Vol. 128, Pp. 111–129.

9. **Sveier A., Kleppe A.L., Tingelstad L., Egeland O.** Object detection in point clouds using conformal geometric algebra. *Advances in Applied Clifford Algebras*, 2017, Vol. 27, No. 3, Pp. 1961–1976.

10. **Wang L., Xu Y., Li Y.** A Voxel-based 3D building detection algorithm for airborne LiDAR point clouds. *Journal of the Indian Society of Remote Sensing*, 2019, Vol. 47, No. 2, Pp. 349–358.

11. **Zhong L., Cheng L., Xu H., Wu Y., Chen Y., Li M.** Segmentation of individual trees from TLS and MLS Data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2017, Vol. 10, No. 2, Pp. 774–787.

12. **Xu Y., Tuttas S., Hoegner L., Stilla U.** Voxel-based segmentation of 3D point clouds from construction sites using a probabilistic connectivity model. *Pattern Recognition Letters*, 2018, Vol. 102, Pp. 67–74.

13. **Xu Y., Tuttas S., Hoegner L., Stilla U.** Geometric primitive extraction from point clouds of construction sites using VGS. *IEEE Geoscience and Remote Sensing Letters*, 2017, Vol. 14, No. 3, Pp. 424–428.

14. **Tang Y., Feng J.** Multi-scale surface reconstruction based on a curvature-adaptive signed distance field. *Computers and Graphics (Pergamon)*, 2018, Vol. 70, Pp. 28–38.

15. **Zhao R., Pang M., Liu C., Zhang Y.** Robust normal estimation for 3D LiDAR point clouds in urban environments. *Sensors* (Switzerland), Vol. 19, No. 5.

16. **Farooqui M.Z., Shoaib M., Khan M.Z.** A comprehensive survey of page replacement algorithms, *IJARCET*, 2014.

17. **Fu S., He L., Huang C., Liao X.K., Li K.** Performance optimization for managing massive numbers of small files in distributed file systems. *IEEE Transactions on Parallel and Distributed Systems*, 2015, Vol. 26, Pp. 3433–3448.

18. **Samet H.** An overview of quadtrees, octrees, and related hierarchical data structures. *Theoretical Foundations of Computer Graphics and* CAD, 1988, Pp. 51–68.

19. **Vo A.V., Laefer D.F., Bertolotto M.** Airborne laser scanning data storage and indexing: state-of-the-art review. *International Journal of Remote Sensing*, 2016, Vol. 37, Pp. 6187–6204.

20. Photogrammetry & Remote Sensing A.S.P.R.S. LAS Specification Version 1.4--R13. *Available: https://www.asprs.org/a/society/committees/standards/LAS_1_4_r13.pdf* (Accessed: 08.10.2019).

21. **Rusu R.B., Cousins S.** Point cloud library (pcl). *2011 IEEE International Conference on Robotics and Automation*, 2011, Pp. 1–4.

22. **Girardeau-Montaut D.** *Cloud compare 3D point cloud and mesh processing software. Open Source Project*, 2015.

Received 10.11.2019.

СВЕДЕНИЯ ОБ АВТОРАХ / THE AUTHORS

БЕЛЯЕВСКИЙ Кирилл Олегович
BELIAEVSKII Kirill O.
E-mail: kirill.beliaevskii@spbpu.com