УДК 004.021

ПРИМЕНЕНИЕ ДИНАМИЧЕСКОЙ АЛЛОКАЦИИ НА ОТОБРАЖАЕМОЙ ПАМЯТИ ДЛЯ ОБРАБОТКИ БОЛЬШИХ ОБЛАКОВ ТОЧЕК В БИБЛИОТЕКЕ PCL

© 2020 К.О. Беляевский

Санкт-Петербургский Политехнический университет Петра Великого

Статья поступила в редакцию 10.02.2020

В статье рассматривается решение задачи обработки облаков точек, чей размер превышает доступные объемы оперативной памяти, при помощи библиотеки Point Cloud Library (PCL). Проблема нехватки оперативной памяти решается за счет механизма динамической аллокации на отображаемой памяти. Произведен анализ способов внедрения подобного механизма в библиотеку PCL, а также проведена оценка производительности обработки облаков точек и объема потребляемой памяти с использованием оперативной или отображаемой (внешней) памяти.
Ключевые слова: Облако точек, потребление памяти, динамическая аллокация, отображение памяти, Point Cloud Library.

ВВЕДЕНИЕ

Анализ предметной области. Современные системы лазерного сканирования позволяют производить миллионы измерений в секунду, а размеры получаемых облаков точек могут достигать нескольких сотен гигабайт, что предъявляет высокие требования к вычислительным ресурсам при обработке таких данных. Развитие систем лазерного сканирования приводит к тому, что темпы роста размеров облаков точек опережают темпы увеличения объема оперативной памяти, что делает снижение требований к вычислительным ресурсам, в частности, снижение потребления оперативной памяти актуальной задачей.

При обработке облака точек зачастую применяются специализированные программные библиотеки, предоставляющие основные структуры данных и алгоритмы, используемые при обработке. Наиболее широко используемой для обработки облаков точек является библиотека Point Cloud Library (PCL) [1]. Она содержит структуры данных для представления облаков точек и ускорения пространственного поиска, а также алгоритмы для визуализации, фильтрации, сегментации, регистрации и прочей обработке облаков точек. Недостатком библиотеки PCL является невозможность обработки облаков точек, размер которых превышает доступные объемы оперативной памяти.

Проблемы нехватки памяти при обработке облаков точек библиотекой PCL могут быть решены установкой дополнительных объемов оперативной памяти, либо увеличением размеров файла подкачки операционной системы. Однако, в первом случае повысятся требования к аппаратному обеспечению, а во втором существенно снизится скорость выполнения задач в операционной системе при заполнении оперативной памяти.

Беляевский Кирилл Олегович, аспирант.
E-mail: kirill.beliaevskii@spbpu.com

Наиболее предпочтительным способом решения проблемы обработки больших облаков точек является разработка специальных структур данных, позволяющих выполнять обработку облака точек в условиях ограниченного потребления оперативной памяти [2–5]. Однако, применение подобных структур данных и алгоритмов потребует полной переработки рассматриваемой библиотеки.

Для решения проблемы нехватки памяти в данной работе рассмотрено применение механизма отображения памяти, который позволяет выполнять отображение содержимого файла на диапазон виртуальных адресов процесса. Для управления выделением и освобождением участков памяти произвольного размера в работе используется алгоритм динамической аллокации, использующий в качестве базового блока данных отображаемую память. Механизм отображения памяти, алгоритм динамической аллокации, а также их комбинация известны и исследованы. Однако, при обработке данных лазерного сканирования такая концепция широкого применения не нашла.

Постановка задачи. Столкнувшись с данной проблемой при разработке алгоритма выделения цилиндрических объектов в облаке точек, автором была выдвинута гипотеза, что использование системы динамической аллокации на отображаемой памяти позволит сократить объемы потребления оперативной памяти без существенного падения производительности обработки и без необходимости внесения большого количества изменений в исходные коды библиотеки.

Целью данной работы является анализ способов внедрения системы динамической аллокации на отображаемой памяти в библиотеку PCL, а также оценка производительности обработки облаков точек и объема потребляемой оперативной памяти с использованием стандартной и предложенной систем аллокации данных. В качестве исследуемого алгоритма обработки облаков точек используется алгоритм выделения цилиндрических объектов, так как он разработан с использованием библиотеки PCL и таких ее компонентов и структур данных, как: облако точек, к-мерное дерево, алгоритмов кластеризации, прореживания, вычисления нормалей и векторизации.

1. АЛГОРИТМ ВЫДЕЛЕНИЯ ЦИЛИНДРИЧЕСКИХ ОБЪЕКТОВ

Алгоритм выделения цилиндрических объектов применяется для поиска в облаке точек объектов определенного класса природно-технических систем – трубопровода. Среди прочих примеров цилиндрических поверхностей природно-технических систем – труба, дерево (конус), дымовая труба (цилиндр или конус), столб (в том числе многогранный), тоннель, в некоторых случаях, кабель/провод. Выделение таких объектов требуется для:

- Проведения измерений.
- Подсчета статистики.
- Снижения визуальной нагрузки при отображении облака точек.
- Экспорта в ГИС-системы.

1.1. Этапы алгоритма выделения цилиндрических объектов

За основу рассматриваемого алгоритма взят алгоритм, описанный автором в работе [6]. В процессе доработки алгоритма был удален этап вычисления принципиальных направлений, и добавлен этап кластеризации, что позволило повысить точность и скорость работы. Этапы обработки реализованы в виде последовательности алгоритмов, которые вместе образуют конвейер обработки облака точек. Входными данными для каждого алгоритма является набор параметров, а также облако точек в формате PCL, основанное на стандартной реализации динамического массива, и расположенное в оперативной памяти. Рассмотрим основные этапы обработки с точки зрения используемых структур данных и компонентов библиотеки PCL.

Загрузка и представление облака точек. Облако точек на вторичной системе хранения данных представлено форматом LAS [7], который широко используется для хранения облаков точек. Загрузка облака точек осуществляется путем последовательного считывания участков такого файла, их обработкой и добавлением в структуру данных облака точек, представленную библиотекой PCL. Облако точек в библиотеке PCL представлено классом *PointCloud*, который использует для хранения точек облака реализацию динамического массива из стандартной библиотеки C++ (std::vector [8]). Хранение элементов такого массива в оперативной памяти осуществляется в виде неразрывного блока данных, что позволяет обеспечить независимость скорости выборки по индексу от объема блока данных.

Прореживание облака точек. Для прореживания облака точек используется алгоритм *VoxelGrid* [9]. Поверх исходного облака строится трехмерная сетка вокселей, после чего для каждого вокселя все точки, находящиеся внутри, заменяются их центроидом. В результате получается разряженное облако точек, которое не потеряло своих геометрических свойств. Данный алгоритм позволяет значительно ускорить дальнейшую обработку облака точек.

Расчет нормалей. Нормаль к поверхности, описываемой облаком точек, является атрибутом точки облака, и используется для таких операций, как кластеризация и поиск цилиндрических поверхностей. Для расчета нормалей используется алгоритм *Normal Estimation* [10,11] из библиотеки PCL, который в свою очередь для ускорения операций пространственного поиска использует K-мерное дерево *KdTree* [12].

Удаление малоинформативных точек. Обычно облака точек сильно зашумлены, поэтому необходимо провести процесс фильтрации для того чтобы избавиться от малоинформативных точек, которые снижают качество получаемого результата и увеличивают время обработки. Для решения этой задачи используется два алгоритма:

- Алгоритм фильтрации выбросов.
- Алгоритм удаления слишком плоских или слишком шумных поверхностей.

Для удаления точек-выбросов используется двухпроходный алгоритм статистической фильтрации точек *Statistical Outlier Removal* [11,13], предоставленный библиотекой PCL. На первом проходе для каждой точки вычисляется средняя дистанция до k ближайших соседей, k является настраиваемым значением. Далее, вычисляется среднее и стандартное отклонение для средних дистанций всех точек, при помощи данных параметров рассчитываются границы дистанции по формуле *mean* + *stddev_mult* * *stddev*, где *stddev_mult* является настраиваемым значением. На втором проходе все точки классифицируются, как внутренние или внешние, в зависимости от значения их средней дистанции до соседей. Все внешние точки считаются малоинформативными.

Алгоритм удаления слишком шумных или слишком плоских сегментов также предназначен для удаления малоинформативных точек. Его применение основывается на понимании

того, что искомые цилиндрические поверхности имеют определенный радиус кривизны, а значит слишком плоские или наоборот слишком зашумленные сегменты облака точек скорее всего не будут принадлежать искомому цилиндру. Для каждой точки берется k соседей, где k — настраиваемый параметр, и по ним методом наименьших квадратов строится описывающая плоскость. После этого рассчитывается среднеквадратическое отклонение точки от плоскости. Точки с отклонением, не удовлетворяющим заданному критерию, исключаются из выборки.

Кластеризация. Для сокращения объема вычислений при поиске цилиндрических объектов выполняется разбиение отфильтрованного облака точек на отдельные кластеры при помощи алгоритма *Euclidean Cluster Extraction* [14,15], представленного в библиотеке PCL.

Поиск цилиндрических поверхностей. Поиск цилиндрических поверхностей производится при помощи алгоритма RANSAC (RANdom Sample Consensus) [16,17]. Алгоритм основан на сборе статистики о входных данных. Из общего набора входных точек случайным образом выбирается некоторое подмножество фиксированного размера, которое аппроксимируется геометрическим примитивом. Общее количество точек входного набора, оказавшихся вблизи полученного примитива, запоминается. Этот процесс повторяется несколько раз. Геометрический примитив, вблизи которого оказалось наибольшее число точек с высокой вероятностью является наилучшей аппроксимацией всего множества входных точек [18].

Алгоритм поиска цилиндрических поверхностей также поддерживает поиск примитивов с заданными характеристиками (радиус, направление и т.д.). В качестве входных параметров алгоритм принимает список принципиальных направлений, облако точек для поиска, коэффициент вклада нормалей при поиске цилиндра, количество итераций алгоритма RANSAC, коэффициент вклада расстояний при поиске цилиндра, радиус искомых труб (минимальный и максимальный), угол максимального отклонения от принципиального направления, количество итераций поиска, минимальное количество точек, которые могут принадлежать цилиндру. На выходе алгоритма можно получить список цилиндров и облако точек без цилиндрических поверхностей.

1.2. Потребление ресурсов при обработке в оперативной памяти

Рассмотрим процесс обработки облака точек алгоритмом в оперативной памяти. Для обработки используем облако точек под названием *factory.las*, содержащее 42 млн точек, и занимающее 1.44 Гб дискового пространства. На рисунке 1 представлена визуализация облака точек и его раскраска по плотности. Расчет плотности выполнялся путем подсчета количества соседей в радиусе 5 сантиметров для каждой точки из облака.

Рассмотрим этапы обработки алгоритма на примере предложенного облака точек (Рис. 2).

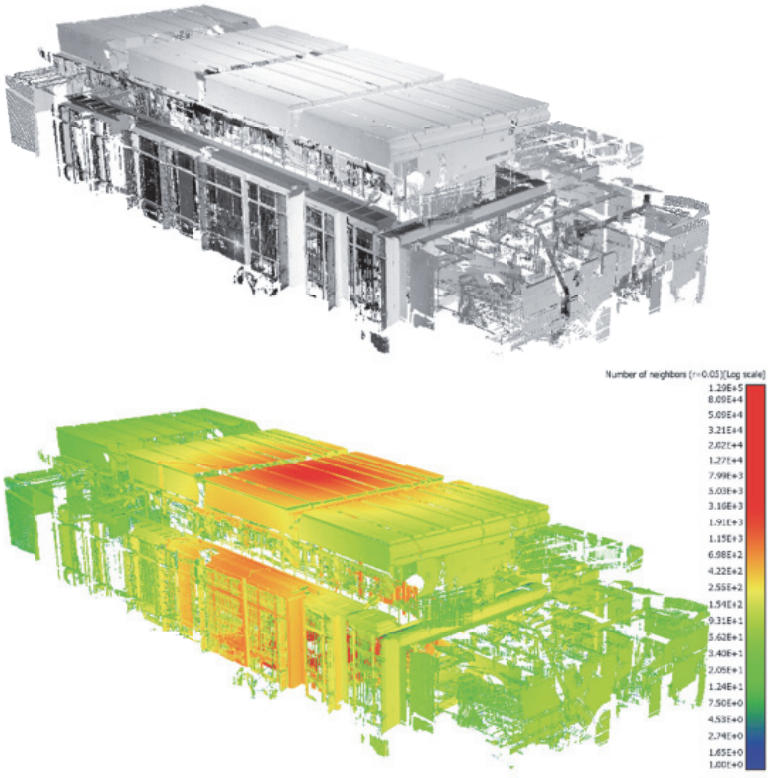

Рис. 1. Исходное облако точек (сверху) и его раскраска по плотности (снизу)

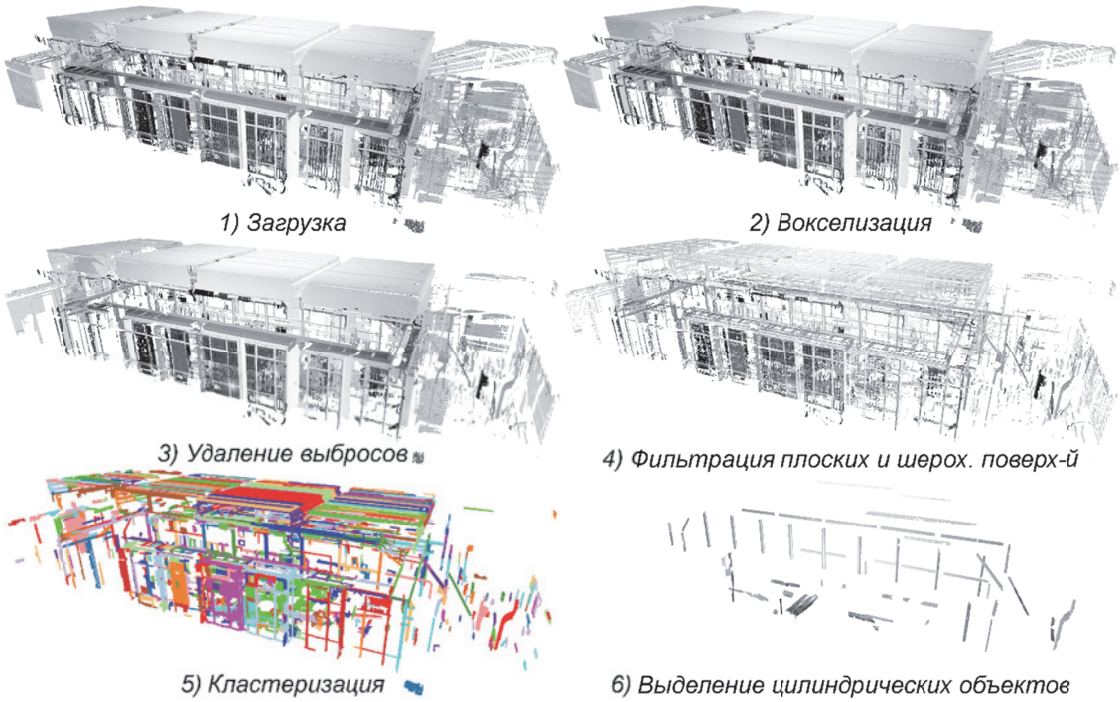

Рис. 2. Поэтапная визуализация обрабатываемого облака точек

Оценим потребляемую оперативную память в процессе обработки облака точек. Для снятия объема потребления оперативной памяти выполним в отдельном потоке опрос системного счетчика потребляемой памяти процесса с интервалом в 30 мс, а результаты выведем в виде графика (Рис. 3).

Из рисунка 3 видно, как растет объем потребляемой памяти по мере считывания облака точек на этапе «Загрузка», выделяются дополнительные объемы данных и сокращается объем потребления памяти в результате прореживания на этапе «Вокселизация». На последующих этапах видны характерные пики и провалы в по-

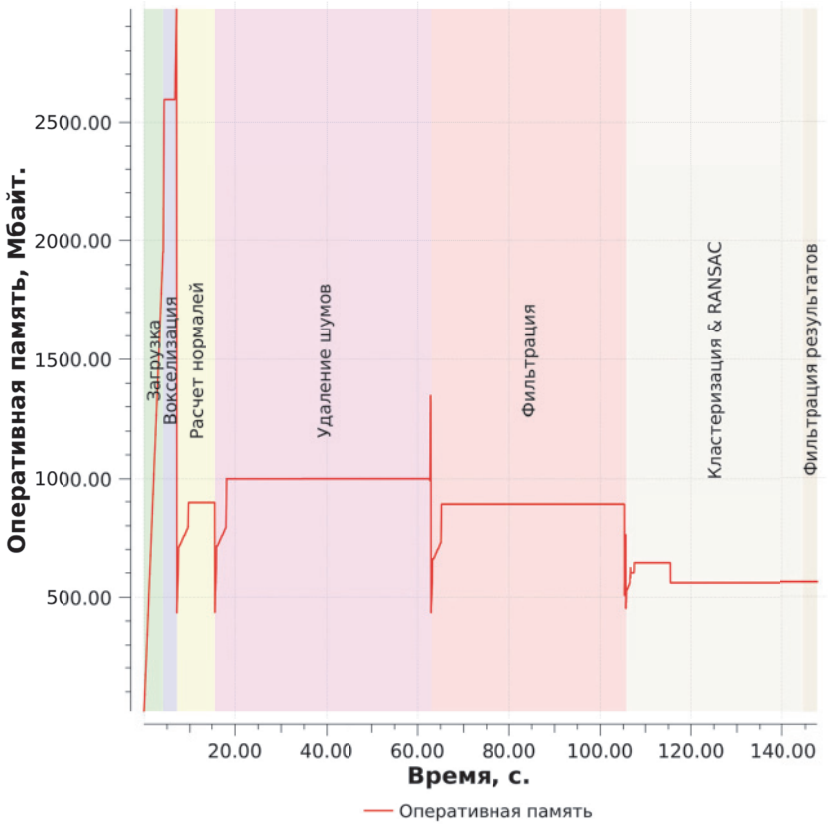

Рис. 3. Потребление оперативной памяти в процессе обработки облака точек *five.las*

треблении памяти, возникающие в результате работы алгоритмов PCL, в которых модификация данных облака точек производится в новое облако точек, после чего старое обычно удаляется.

Приведенный график иллюстрирует потребление памяти при обработке относительно небольшого облака точек, однако пиковое потребление оперативной памяти превышает размер облака точек почти в два раза. При обработке больших облаков точек, чей размер превышает объемы оперативной памяти, использование рассмотренных алгоритмов становится невозможным. Для обеспечения обработки таких облаков могут быть применены различные методики, такие как:

- Разделение облака точек на меньшие части и обработка по частям.
- Сжатие или более компактное представление данных облака точек.
- Потоковая обработка облака точек.
- Управление потребляемой памятью.

Однако, применительно к конкретной реализации алгоритмов обработки облака точек, сжатие и потоковая обработка влекут за собой необходимость широкой модификации существующих алгоритмов библиотеки PCL, разделение облака точек на меньшие части требует дополнительного этапа предобработки данных и совмещения результатов на границах участков, а встраивание механизмов управления потребляемой памятью в большинстве случаев также требует существенных изменений в программном коде. Тем не менее, управление потребляемой памятью с минимальными изменениями в исходном коде библиотеки все же возможно при помощи комбинации системы динамической аллокации и механизма отображения памяти, что и будет продемонстрировано далее.

2. ВНЕДРЕНИЕ СИСТЕМЫ ДИНАМИЧЕСКОЙ АЛЛОКАЦИИ ДАННЫХ

Целью данной работы является исследование методики, позволяющей добиться снижения потребления оперативной памяти при сохранении приемлемой производительности за счет использования механизма отображения памяти. Для обеспечения возможности обработки больших облаков точек необходимо сократить объемы данных, находящихся в оперативной памяти, с минимальными изменениями в исходном коде используемой библиотеки. Для решения поставленной задачи в статье предложено использование механизма динамической аллокации на отображаемой памяти, а также два способа внедрения такого механизма в библиотеку PCL:

1. Внедрение аллокатора в реализацию динамического массива PCL.
2. Полная подмена системного аллокатора.

2.1. Динамическая аллокация на отображаемой памяти

Сокращение объема потребления оперативной памяти обеспечивается за счет выгрузки неактуальных данных на вторичную систему хранения. Учитывая требование к минимальным изменениям в исходном коде библиотеки, для решения задачи используется механизм отображения памяти, позволяющий выполнить отображение содержимого файла на диапазон виртуальных адресов процесса. Таким образом, доступ к содержимому файла будет обеспечиваться по прямому указателю, то есть так же, как и к данным в оперативной памяти, что позволит неявно подменить данные облака точек в оперативной памяти на данные на вторичной системе хранения.

Учитывая необходимость работы с участками памяти произвольного размера, для выделения и освобождения блоков данных на вторичной системе хранения, а также для снижения фрагментации блоков используется механизм динамической аллокации на отображаемом диапазоне виртуальных адресов, рассмотренный в статье [19]. В нем отображаемый диапазон адресов используется в качестве пула памяти, на котором производится выделение блоков данных. Выделенные и свободные участки внутри пула хранятся в виде двусвязного списка блоков, каждый из которых состоит из начального (Header) и конечного (Footer) заголовка, и блока данных между ними.

Выделение нового блока начинается с поиска наиболее подходящего свободного блока данных. В случае, если такой блок найден, в зависимости от свободного объема он либо будет помечен занятым, либо будет разделен на два блока, один из которых будет возвращен в качестве искомого. В случае, если места в пуле недостаточно для выделения нового блока, размер файла будет увеличен на необходимое значение. При освобождении блок помечается свободным, после чего для снижения фрагментации производится его объединение с соседними свободными блоками.

2.2. Внедрение в реализацию динамического массива

Внедрение механизма динамической аллокации на отображаемой памяти в реализацию динамического массива основано на возможности подмены аллокатора в используемом библиотекой PCL для хранения данных облака точек динамическом массиве *std::vector*. При помощи наследования от класса стандартной библиотеки C++ под названием *std::allocator,* и используя возможности прямого доступа к отображаемой

памяти, вызовы системного аллокатора подменяются вызовами аллокатора на отображаемой памяти.

Преимуществом такого подхода является точечное использование для определенных структур данных, а именно – облака точек. Недостатком является необходимость изменения исходных кодов библиотеки. Само изменение невелико, однако, ввиду того, что объявление динамического массива дублируется во многих местах в библиотеке, побочные изменения, которые потребуется внести будут значительными.

2.3. Подмена системного аллокатора

Полная подмена системного аллокатора позволяет обеспечить использование отображаемой памяти без внесения изменений в исходные коды библиотеки. В таком случает используется возможность переопределения __malloc_hook и __free_hook [20] для Linux-систем, которая позволила использовать в функциях malloc и free свою реализацию аллокатора. При этом, ограничить применение строго для определенных структур данных не представляется возможным. Применение аллокатора было ограничено только потоком исполнения и размерами выделяемого блока данных: для блоков, чей размер меньше 1 мегабайта, вызывается системный аллокатор.

Недостатком подобной глобальной подмены аллокатора является невозможность точечного применения для определенных структур данных, а также меньшая стабильность системы: в процессе работы алгоритма были зафиксированы ошибки при работе с памятью в пользовательском интерфейсе тестового приложения. Для обеспечения стабильной работы потребовалась изоляция процесса вычислений в отдельном потоке. Разнесение временных промежутков исполнения также помогло справиться с проблемой.

3. ЭКСПЕРИМЕНТАЛЬНЫЙ АНАЛИЗ

В данном разделе будет произведено тестирование механизма динамической аллокации на отображаемой памяти применительно к алгоритму выделения цилиндрических объектов.

3.1. Методика тестирования

Для тестирования использовано два облака точек: описанное ранее *factory.las* для сравнения с результатами, полученными при обработке в оперативной памяти, и облако точек *molodezhnoe.las*, для демонстрации обработки облака точек, чей размер превышает доступные объемы оперативной памяти. Облако точек *molodezhnoe.las* является результатом совмещения данных сканирования с множества точек, содержит 1.5 млрд точек и занимает 54 Гб дискового пространства (Рис. 4).

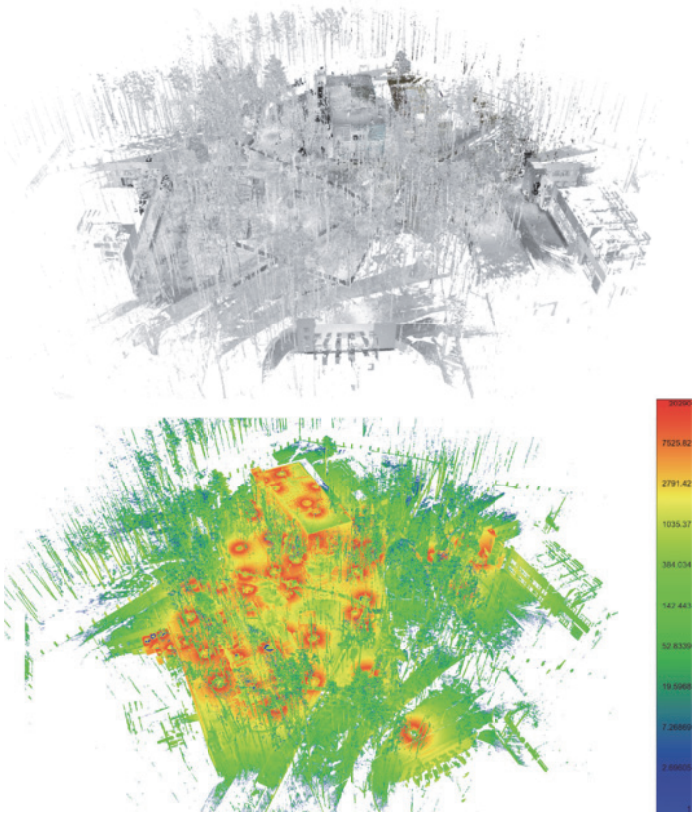

Рис. 4. Облако точек *molodezhnoe.las* (сверху) и его плотность (снизу)

Конфигурация тестового стенда:

- CPU: Intel® Core™ i7-7820HQ CPU @ 2.90 GHzRAM: 2 x SODIMM DDR4 2400 MHz (2x8 GB)
- SSD: Samsung SSD 850 PRO 1 TB
- OS: Ubuntu 18.04
- File system: Ext4

3.2. Сравнение с обработкой в оперативной памяти

Оценим потребляемую память на вторичной системе хранения при обработке облака точек *factory.las* с использованием динамической аллокации на отображаемой памяти (Рис. 5). Для сравнения на рисунке 5 (слева) приведен график обработки в оперативной памяти, полученный ранее. Снятие объема потребления файловой памяти производилось путем опроса системы аллокации с интервалом в 30 мс. Стоит отметить, что на графике приведен размер занятых блоков памяти, а общий размер файла равен пиковому потреблению в процессе обработки. Как видно из графиков, скорость обработки замедлилась незначительно (147 секунд в оперативной памяти и 156 секунд в файловой).

Рассмотрим потребление оперативной памяти в процессе обработки облака точек *factory.las* с использованием динамической аллокации на отображаемой памяти. Как видно из графика на рисунке 6, объем потребления оперативной

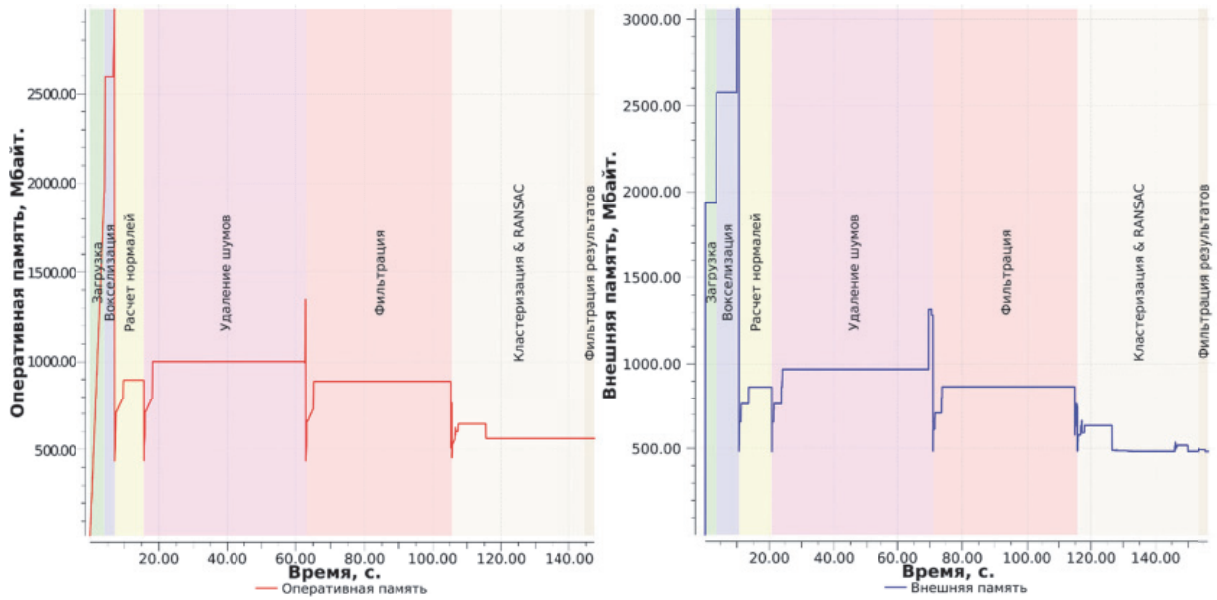

Рис. 5. Обработка облака точек *factory.las* в оперативной памяти (слева) и в файловой памяти (справа)

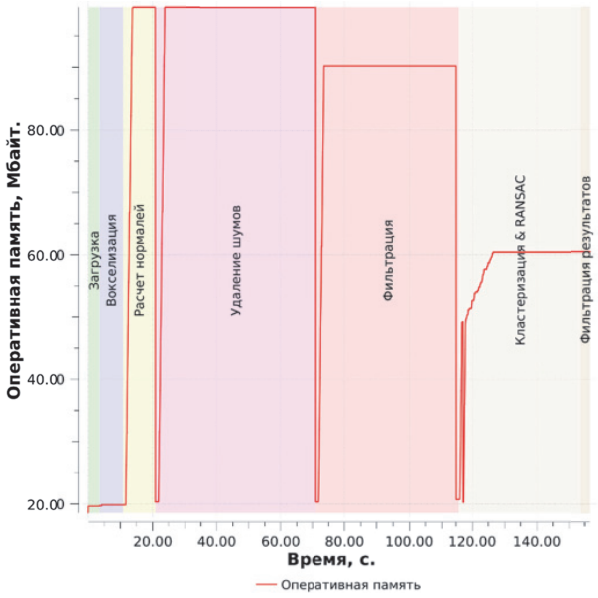

Рис. 6. Потребление оперативной памяти при обработке облака точек с использованием вторичной системы хранения

памяти в таком случае существенно меньше, чем при обработке исключительно в оперативной памяти. Оперативная память при обработке выделяется благодаря ограничению на использование аллокации во внешней памяти при размере выделяемого блока больше 1 мегабайта. Соответственно, участки меньше мегабайта будут выделяться в оперативной памяти. Пики потребления на графике соответствуют построению и использованию *K-d* деревьев, которые при построении совершают множество выделений небольших участков данных.

Проведенный эксперимент показывает снижение потребляемой оперативной памяти при применении системы динамической аллокации на отображаемой памяти, с незначительным (~6%) снижением общей скорости обработки облака точек.

3.3. Обработка больших облаков точек

Оценим потребляемую память при обработке облака точек *molodezhnoe.las,* чей размер больше установленного объема оперативной памяти. Как видно из графика на рисунке 7 (слева), пиковое потребление файловой памяти (115 Гб) и среднее потребление файловой памяти (~30 Гб) значительно превышает доступные объемы оперативной памяти (16 Гб). Потребление оперативной памяти в процессе обработки (на графике справа), как и в предыдущих экспериментах, обусловлено выделением участков данных меньше мегабайта с использованием системного аллокатора.

Проведенный эксперимент доказывает возможность применения библиотеки PCL для обработки облаков точек, больших, чем доступные объемы оперативной памяти, при помощи внедрения системы динамической аллокации на отображаемой памяти.

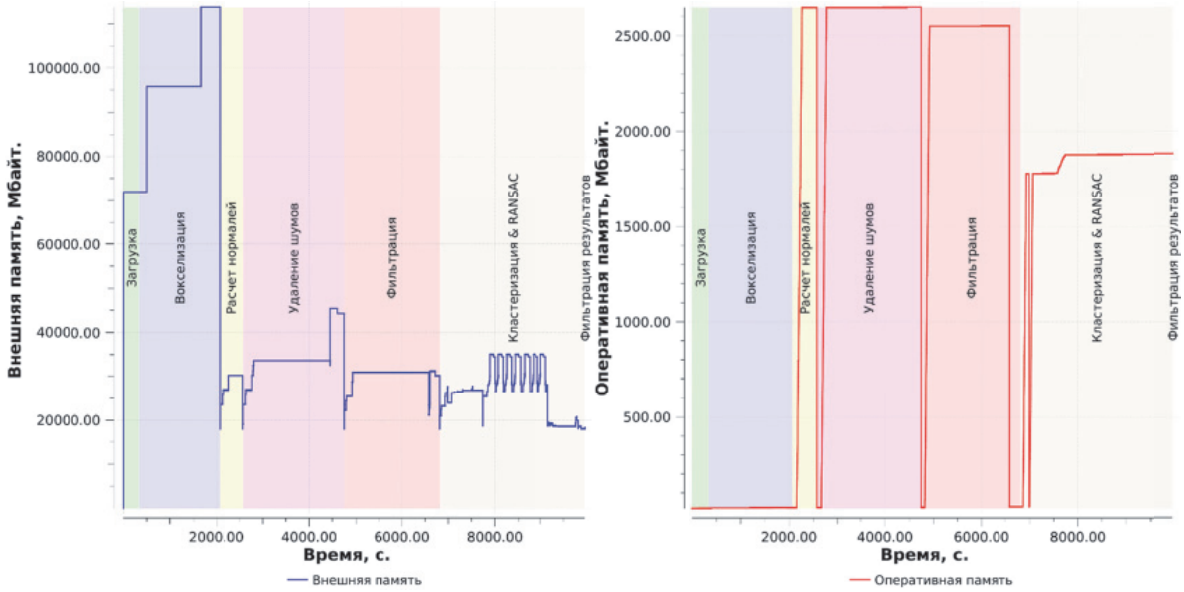

Рис. 7. Потребление файловой (слева) и оперативной (справа) памяти при обработке облака точек *molodezhnoe.las*

ЗАКЛЮЧЕНИЕ

Полученные результаты позволяют сделать вывод, что внедрение системы динамической аллокации на отображаемой памяти позволяет выполнять обработку больших облаков точек используя алгоритмы, рассчитанные на применение исключительно в оперативной памяти, без существенного снижения производительности. Однако, внедрение подобного механизма сопряжено с рядом проблем: полная подмена системного аллокатора может привести к ошибкам при работе с памятью в некоторых программных компонентах (в частности, в пользовательском интерфейсе), а частичная подмена аллокатора динамического массива требует значительных изменений в исходном коде библиотеки. Решением может стать поддержка внедрения пользовательских систем аллокации в библиотеке PCL, что позволит устанавливать собственную систему аллокации исключительно для облака точек, без изменения исходных кодов библиотеки.

СПИСОК ЛИТЕРАТУРЫ

1. *Rusu R.B., Cousins S.* 3D is here: Point Cloud Library (PCL) // Proceedings - IEEE International Conference on Robotics and Automation. 2011.
2. *Pajarola R.* Stream-processing points // Proceedings of the IEEE Visualization Conference. 2005.
3. *Richter R., Discher S., Döllner J.* Out-of-core visualization of classified 3D point clouds // Lecture Notes in Geoinformation and Cartography. 2015.

4. *Wenzel K. et al*. An out-of-core octree for massive point cloud processing // rs.tudelft.nl. 2011.
5. *Wand M. et al*. Processing and interactive editing of huge point clouds from 3D scanners // Comput. Graph. 2008.
6. *Bolsunovskaya M. et al*. Experimental sample of a software module for processing of a cloud of laser scanning points for natural-technical systems development // Proceedings of the 33rd International Business Information Management Association Conference, IBIMA 2019: Education Excellence and Innovation Management through Vision 2020. 2019.
7. The American Society for Photogrammetry & Remote Sensing. LAS SPECIFICATION VERSION 1.4 – R13 15 July 2013 // American Society for Photogrammetry & Remote Sensing. 2013.
8. Vector - cppreference.com [Electronic resource]. 2019. URL: https://en.cppreference.com/w/cpp/container/vector (accessed: 15.10.2019).
9. *Orts-Escolano S. et al*. Point cloud data filtering and downsampling using growing neural gas // Proceedings of the International Joint Conference on Neural Networks. 2013.
10. Estimating Surface Normals in a PointCloud [Electronic resource]. 2019. URL: http://pointclouds.org/documentation/tutorials/normal_estimation.php (accessed: 22.10.2019).
11. *Hsieh C.-T.* An efficient development of 3D surface registration by Point Cloud Library (PCL) // 2012 International Symposium on Intelligent Signal Processing and Communications Systems. 2012. P. 729–734.
12. How to use a KdTree to search [Electronic resource]. 2019. URL: http://pointclouds.org/documentation/tutorials/kdtree_search.php (accessed: 22.12.2019).
13. Removing outliers using a StatisticalOutlierRemoval filter [Electronic resource]. 2019. URL: http://pointclouds.org/documentation/tutorials/statistical_outlier.php (accessed: 22.12.2019).
14. Euclidean Cluster Extraction [Electronic resource]. 2019. URL: http://pointclouds.org/documentation/tutorials/cluster_extraction.php (accessed: 22.12.2019).
15. *Liu Y., Zhong R.* Buildings and terrain of urban area point cloud segmentation based on PCL // IOP Conference Series: Earth and Environmental Science. 2014. Vol. 17, № 1. P. 12238.
16. *Bolles R.C., Fischler M.A.* A RANSAC-Based Approach to Model Fitting and Its Application to Finding Cylinders in Range Data. // IJCAI. 1981. Vol. 1981. P. 637–643.
17. How to use Random Sample Consensus model [Electronic resource]. 2019. URL: http://pointclouds.org/documentation/tutorials/random_sample_consensus.php (accessed: 22.12.2019).
18. *Дегтярева А., Вежневец В.* Line fitting, или методы аппроксимации набора точек прямой // Компьютерная графика и мультимедиа. Выпуск. 2003. № 1. P. 3.
19. *Беляевский К.О., Болсуновская М.В.* Использование механизма отображения памяти при формировании октодерева облака точек // Неделя науки СПбПУ. 2019. Vol. 1. P. 126–128.
20. MALLOC_HOOK(3) - Linux Programmer's Manual [Electronic resource]. 2019. URL: http://man7.org/linux/man-pages/man3/malloc_hook.3.html (accessed: 15.10.2019).

USING DYNAMIC ALLOCATION ON MAPPED MEMORY TO PROCESS LARGE POINT CLOUDS WITH THE PCL LIBRARY

© 2020 K.O. Beliaevskii

Peter the Great St. Petersburg Polytechnic University

The article proposes a solution to the problem of processing point clouds whose size exceeds available RAM by using Point Cloud Library (PCL). The problem of lack of RAM is solved using the dynamic allocation mechanism on the mapped memory. The methods of introducing such a mechanism into the PCL library are analyzed, and the performance of processing point clouds and the amount of memory consumed using RAM or mapped (external) memory is evaluated.
Keywords: Point cloud, memory consumption, dynamic allocation, memory mapping, Point Cloud Library.

Kirill Beliaevskii, Graduate Student.
E-mail: kirill.beliaevskii@spbpu.com