ПРОГРАММИРОВАНИЕ, 2024, № 3, с. 42–53

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.925.3

МЕТОД УПОРЯДОЧИВАНИЯ ОБЛАКОВ ТОЧЕК ДЛЯ ВИЗУАЛИЗАЦИИ НА КОНВЕЙЕРЕ ТРАССИРОВКИ ЛУЧЕЙ

© 2024 г. П. Ю. Тимохин[a], *, М. В. Михайлюк[a], **

[a]ФГУ "ФНЦ Научно-исследовательский институт системных исследований РАН"
117218 Москва, Нахимовский пр-т, 36/1
**E-mail: p_tim@bk.ru*
***E-mail: mix@niisi.ras.ru*

Поступила 19.01.2024 г.
После доработки 19.01.2024 г.
Принята к публикации 24.01.2024 г.

В настоящее время активно развивается оцифровка объектов окружающей среды (растительности, рельефа, архитектурных сооружений и др.) в виде облаков точек. Интеграция таких оцифрованных объектов в системы виртуального окружения позволяет повысить качество моделируемой обстановки, однако требует эффективных методов и алгоритмов визуализации больших объемов точек в реальном времени. В данной статье исследуется решение этой задачи на современных многоядерных графических процессорах с поддержкой аппаратного ускорения трассировки лучей. Предлагается модифицированный метод разбиения исходного неупорядоченного облака точек на группы точек, визуализация которых эффективно распараллеливается на ядрах трассировки лучей. В работе описан алгоритм построения таких групп с помощью чередующихся массивов индексов точек, который работает быстрее альтернативных решений на связных списках, а также имеет меньшие накладные расходы памяти. Предложенные метод и алгоритм были реализованы в программном комплексе визуализации облаков точек и апробированы на ряде оцифрованных объектов окружающей среды. Результаты апробации подтвердили эффективность предложенных решений и возможность их применения в системах виртуального окружения, видеотренажерных и геоинформационных системах, виртуальных лабораториях и др.

Ключевые слова: виртуальное окружение, визуализация, реальное время, трассировка лучей, облако точек

DOI: 10.31857/S0132347424030054, **EDN:** QAOZYR

1. ВВЕДЕНИЕ

В настоящее время во многих областях человеческой деятельности (архитектура, промышленность, геология, археология, лесное хозяйство и др.) активно внедряется оцифровка объектов окружающей среды с помощью дистанционного 3D-сканирования [1, 2]. Результатом такой оцифровки является набор несвязанных между собой точек (*облако точек*), каждой из которых присвоено одно или несколько скалярных значений (цвет, температура, коэффициент отражения инфракрасного излучения и др.). В системах 3D-сканирования наряду со стационарными устройствами также применяется аэросъемка [3, 4], что дает возможность качественно оцифровывать архитектурные сооружения, растительность, рельеф местности и другие сложные и труднодоступные компоненты окружающей среды [5].

Одним из актуальных направлений является интеграция таких оцифрованных объектов в системы виртуального окружения [6–8]. Это позволяет повысить реалистичность и разнообразие моделируемой обстановки (например, растительности), что особенно важно для видеотренажерных комплексов, а также снизить трудоемкость создания виртуальной сцены. При этом для получения качественного представления объекта его оцифровка должна выполняться по большому числу точек (миллионам и выше), в связи с чем возникает задача визуализации таких объемов данных в реальном времени (со скоростью не менее 25 кадров в секунду).

Эффективным путем является решение описанной задачи с помощью современных многоядерных графических процессоров (GPU), поддерживающих аппаратно-ускоренную трассировку лучей [9]. В данной работе предлагается модифицированный метод разбиения исходного неупорядоченного облака точек на группы точек, обработка которых эффективно распараллеливается

на ядрах трассировки лучей. Предлагаемый метод существенно снижает время препроцессинга облака точек и накладные расходы памяти по сравнению с альтернативными решениями.

2. ПРЕДЫДУЩИЕ ИССЛЕДОВАНИЯ

Развитие методов визуализации облаков точек, как и многих других задач компьютерной графики, исторически шло по двум направлениям: растеризация и трассировка лучей. Подробный обзор работ по этим направлениям можно найти в недавнем исследовании [10]. Анализ этих работ показывает тесную связь прогресса в том или ином направлении с эволюцией графических вычислительных средств.

Первые GPU имели небольшое число ядер, которые различались по специализации (вершинные и пиксельные конвейеры) и предназначались для отображения полигональных моделей. Исследования этого периода посвящены поиску представлений облаков точек, обеспечивающих визуальную непрерывность поверхности, сопоставимую по качеству с полигональными моделями: круглые и эллиптические диски (сплэттинг) [11–13], неявные и полиномиальные поверхности [14–16]. Построение таких моделей вначале выполнялось на стороне центрального процессора (CPU), однако с ростом числа ядер и возможностей по их программированию оно стало постепенно переноситься с CPU на GPU [17, 18]. Для ускорения графических расчетов также применялись иерархические структуры данных (вложенные октодеревья [19], *k-d*-дерево [20]), построение которых выполнялось на CPU и занимало немало времени (минуты, часы).

Следующим этапом эволюции стал переход GPU к унифицированной шейдерной архитектуре, сопровождаемый появлением сотен (а затем и тысяч) программируемых вычислительных ядер общего назначения (CUDA-ядер). Это открыло путь к переносу на GPU не только всех графических построений (OpenCL [21], вычислительные шейдеры [22]), но и ускоряющих иерархических структур данных [23, 24]. В этот период активно исследуются техники управления уровнем детализации (LOD) облаков точек (адаптивный сплэттинг [25], непрерывный LOD [26]), а также влияние порядка точек в облаке на скорость визуализации [22]. Наиболее продвинутые решения представляют собой системы распараллеленных на GPU LOD-техник и иерархических структур данных, которые позволяют визуализировать в реальном времени миллионы точек [27, 28]. Общей чертой таких решений является интенсивный расход вычислительного ресурса CUDA-ядер, что ограничивает возможности их совместного использования с другими методами визуализации в системах виртуального окружения.

Появление в 2018 г. аппаратного ускорения трассировки лучей в GPU NVidia ознаменовало наступление новой эры в компьютерной графике реального времени [29]. В архитектуру GPU были введены *RT-ядра* – аппаратные ядра нового типа, предназначенные только для просчета трассировки лучей, что позволило значительно повысить ее скорость, а также разгрузить универсальные CUDA-ядра, необходимые для широкого круга методов визуализации. Для реализации распараллеливания вычислений на RT-ядрах в новую архитектуру также был добавлен специальный конвейер трассировки лучей (*RT-конвейер*), включающий в себя закрытый аппаратный блок и программируемые (шейдерные) стадии, доступные разработчикам. Наличие таких аппаратно-программных возможностей дало мощный импульс к ревизии существующих методов визуализации и разработке новых решений [30, 31].

Чтобы использовать аппаратное ускорение трассировки лучей, вся геометрия виртуальной сцены должна быть объединена в *дерево ограничивающих объемов* (Bounding Volume Hierarchy, BVH) [32]. Построение BVH-дерева выполняется автоматически драйвером видеокарты на основе данных о треугольных/процедурных примитивах сцены, которые указывает разработчик. В случае процедурных примитивов (вычисляемых в процессе визуализации) разработчик должен явно задать для них листья BVH-дерева – ограничивающие параллелепипеды (Axis-Aligned Bounding Boxes, AABBs). Построенное таким образом BVH-дерево привязывается к RT-конвейеру и в дальнейшем используется его аппаратным блоком для быстрого поиска пересечений лучей с примитивами сцены.

Для того чтобы связка “BVH-дерево–RT-конвейер” эффективно функционировала, необходимо перед построением BVH-дерева правильно сгруппировать примитивы сцены: их AABBs должны как можно меньше пересекаться между собой и содержать пустот [33]. Применительно к облакам точек это вызывает затруднение, так как их точки в общем случае расположены в произвольном порядке. Простым решением является создание отдельного AABB для каждой точки (ее процедурной 3D-модели, например сферы) [34], однако это переводит основную вычислительную нагрузку (поиск пересечений луча с AABBs) на аппаратный блок RT-конвейера, что

делает его “бутылочным горлышком” при увеличении размера облака точек. Кроме того, хранение AABB для каждой точки приводит к ощутимым накладным расходам видеопамяти, что также ограничивает их число, которое можно обрабатывать на видеокарте.

В целях обхода описанных ограничений в нашем предыдущем исследовании [9] мы предложили объединять точки облака в группы, соответствующие листьям разреженного воксельного октодерева (далее *октантные группы точек*, ОГТ), что дало заметный прирост скорости визуализации на RT-конвейере и позволило сократить накладные расходы видеопамяти. В настоящей работе описывается модификация данного подхода, направленная на сокращение времени препроцессинга облака точек, а именно формирования ОГТ. В разделе 3 предлагаются метод и алгоритм построения ОГТ с помощью чередующихся массивов индексов точек, которые превосходят по скорости и затратам памяти альтернативные решения, основанные на связных списках. В разделе 4 приводятся результаты апробации созданного решения на облаках точек, полученных с помощью 3D-сканирования реальных объектов.

3. МЕТОД ОКТАНТНЫХ ГРУПП ТОЧЕК

В предлагаемом методе поиск непересекающихся групп близлежащих точек осуществляется на основе обхода октодерева. Мы строим вокруг исходного облака точек ограничивающий объем, разбиваем его на 8 октантов и проверяем, сколько точек попадает в каждый октант. Если число точек в октанте превышает заданное пороговое значение $K_{\max}$, то октант снова разбивается на 8 октантов, и процедура повторяется до тех пор, пока не будут найдены все ОГТ, удовлетворяющие условию $K_{\max}$. Несмотря на кажущуюся простоту идеи, ее реализация включает в себя ряд “подводных камней”.

Во-первых, в качестве исходного ограничивающего объема мы используем не параллелепипед, а куб размера $d_{\max}$:

$$d_{\max} = \max(\max(x_{\max} - x_{\min}, y_{\max} - y_{\min}), z_{\max} - z_{\min}) + 10\varepsilon, \quad (1)$$

где $x_{\min}$, $y_{\min}$, $z_{\min}$ и $x_{\max}$, $y_{\max}$, $z_{\max}$ – наименьшие и наибольшие из координат точек облака, а ε – машинная погрешность представления вещественных чисел (в данной работе $\varepsilon = 10^{-6}$). Разбиение куба на октанты (также кубы) в результате дает *кучные* группы точек, т. е. сосредоточенные в большом количестве на небольшом пространстве. Наши эксперименты показали, что если вместо куба использовать параллелепипед, то при его разбиении будут получаться вытянутые группы точек, AABBs которых содержат больше пустот и, согласно [33], менее эффективно обрабатываются на RT-конвейере.

Во-вторых, в процессе разбиения ограничивающего куба на октанты может наступить случай, при котором размер $d_{\max}$ куба достигнет погрешности ε, и станет невозможно отделить точки друг от друга (случай “*слипшихся*” точек). В этом случае мы создаем для каждой такой точки свою отдельную ОГТ и прекращаем дальнейшее разбиение октанта. Согласно нашим исследованиям вероятность возникновения “слипшихся” точек наиболее высока при $K_{\max} = 1$ и стремительно убывает с ростом $K_{\max}$. Например, в облаке точек “Beautiful Autumn” (табл. 1) число “слипшихся” точек составляет 0.44% (от общего числа точек) при $K_{\max} = 1$, 0.003% при $K_{\max} = 2$ и 0% при $K_{\max} = 3$.

В-третьих, как показывает практика, выбор значения $K_{\max}$ зависит не только от аппаратных характеристик видеокарты, но и от сложности виртуальной сцены (размера облака точек, числа объектов в сцене и др.), ввиду чего необходимы эффективные структуры данных и алгоритмы, обеспечивающие формирование ОГТ на этапе загрузки системы визуализации. Рассмотрим это более подробно.

Введем обозначения следующих основных структур данных, используемых в данной работе:

- $\boldsymbol{S_P}$ – структура атрибутов точки облака, хранящая позицию (x, y, z) и цвет (r, g, b) точки;
- $\boldsymbol{M_{IN}}$ и $\boldsymbol{M_{OUT}}$ – входной неупорядоченный и выходной упорядоченный массивы точек облака (структур S_P), длиной n каждый;
- $\boldsymbol{S_G}$ – структура-дескриптор ОГТ вида $\{i_{FIRST}, N_{POINTS}\}$, где i_{FIRST} – индекс первой точки ОГТ из массива M_{OUT}, а N_{POINTS} – число точек в ОГТ;
- $\boldsymbol{G_{OUT}}$ – выходной массив дескрипторов октантных групп точек (структур S_G) длиной n (в худшем случае число ОГТ будет совпадать с числом точек в облаке).

Рассмотрим задачу извлечения ОГТ из массива M_{IN}, т. е. получения выходных массивов M_{OUT} и G_{OUT}. Для этого мы будем рекурсивно разбивать ограничивающий куб облака точек на 8 октантов и формировать выборки элементов из массива M_{IN} для каждого k-го номера октанта согласно выражению

$$k = 4\left(\lfloor z/d_{oct} \rfloor \% 2\right) + 2\left(\lfloor y/d_{oct} \rfloor \% 2\right) + \left(\lfloor x/d_{oct} \rfloor \% 2\right), \quad (2)$$

где x, y, z – координаты позиции точки облака (исходно нормализованы и сдвинуты в неотрицательную область); $\lfloor\ \rfloor$ – целая снизу часть числа;

% – остаток от деления; d_{oct} – размер куба-октанта, а $k \in [0, 7]$.

Для хранения и обработки таких выборок введем следующие вспомогательные структуры данных:

• $\boldsymbol{I_0}$ и $\boldsymbol{I_1}$ – пара массивов индексов точек облака из входного массива M_{IN}, длиной n каждый;

• $\boldsymbol{K}$ – байтовый массив номеров октантов точек облака из входного массива M_{IN}, длиной n.

Каждая выборка представляет собой последовательность отобранных индексов точек облака из массива M_{IN}, записанную в массиве I_0 или I_1 (зависит от четности уровня рекурсии), и задается тройкой параметров (b, u, q), где b – флаг четности уровня рекурсии (0 – четный, 1 – нечетный), u – смещение первого элемента выборки в I_b-м массиве, а q – число элементов в выборке.

Вначале мы создаем *стартовую выборку* вида $\{0, 1, ..., n-1\}$, которая совпадает с порядком элементов в исходном массиве M_{IN}. На 0-уровне рекурсии мы получаем до 8-ми выборок (по числу октантов) из стартовой выборки, на 1-м уровне – до 8-ми выборок из каждой выборки 0-го уровня и т. д., пока длина выборки не перестанет превышать $K_{\max}$ или не наступит случай “слипшихся” точек (см. выше). Отметим, что сумма длин выборок $(i+1)$-го уровня, получаемых из выборки i-го уровня, равна длине этой выборки, что позволяет перезаписывать выборки $(i+1)$-го уровня на место выборки i-го уровня. Это свойство позволяет хранить выборки в массивах I_0 и I_1 фиксированного размера n и, чередуя эти массивы в качестве источника и приемника данных, продвигаться в глубь рекурсии. На основе описанной идеи был разработан следующий алгоритм.

Алгоритм извлечения октантных групп точек с помощью чередующихся массивов

1. Создадим массивы M_{OUT}, G_{OUT}, I_0, I_1 и K, а также счетчики m и g элементов в M_{OUT} и G_{OUT}.
2. Обнулим счетчики m и g, а в массив I_0 запишем стартовую выборку индексов (см. выше).
3. Вычислим размер $d_{\max}$ ограничивающего куба облака точек согласно выражению (1).
4. Выполним рекурсивную обработку стартовой выборки ($b = 0$, $u = 0$, $q = n$).

Если $d_{\max} \le \varepsilon$, то создадим ОГТ для каждой из “слипшихся” точек.

Цикл по индексу i в массиве I_b от u до $u + q - 1$

$M_{OUT}[m] = M_{IN}[I_b[i]]$; $G_{OUT}[g] = \{m, 1\}$; $m = m + 1$; $g = g + 1$;

Конец цикла.

В противном случае, если $q \le K_{\max}$, то входная выборка (b, u, q) становится ОГТ:

$G_{OUT}[g] = \{m, q\}$; $g = g + 1$;

Цикл по индексу i массива I_b от u до $u + q - 1$

$M_{OUT}[m] = M_{IN}[I_b[i]]$; $m = m + 1$;

Конец цикла.

В противном случае перейдем к обработке 8-ми выходных выборок.

Запомним текущее значение $d_{\max}$ и разделим его пополам: $d_{save} = d_{\max}$; $d_{\max} = 0.5 d_{\max}$.

Создадим массив Q длин 8-ми выходных выборок и вычислим эти длины.

Обнулим все элементы массива Q.

Цикл по индексу i в массиве I_b от u до $u + q - 1$.

Вычислим номер k октанта согласно (2) для точки $M_{IN}[I_b[i]]$ и $d_{oct} = d_{\max}$.

Запомним номер k октанта: $K[I_b[i]] = k$.

$Q[k] = Q[k] + 1$;

Конец цикла.

Создадим массив U смещений 8-ми выходных выборок и вычислим эти смещения.

Запишем $U[0] = u$, а остальные элементы массива U обнулим.

Цикл по номеру k октанта от 1 до 7

$U[k] = U[k-1] + Q[k-1]$;

Конец цикла.

Распределим точки из входной выборки (b, u, q) по 8-ми выходным выборкам.

Вычислим флаг b_{NEXT} четности следующего уровня рекурсии:

$b_{NEXT} = (b + 1)\% 2$;

Создадим массив C смещений в 8-ми выходных выборках.

Инициализируем массив C: $C[0] = U[0]$, $C[1] = U[1]$, ..., $C[7] = U[7]$.

Цикл по индексу i в массиве I_b от u до $u + q - 1$.

Получим номер k октанта: $k = K[I_b[i]]$.
$I_{b,\,NEXT}[C[k]] = I_b[i]$; $C[k] = C[k] + 1$;
Конец цикла.
Выполним обработку 8-ми выходных выборок.
Цикл по номеру k октанта от 0 до 7.
Если $Q[k] > 0$, то обработаем выборку (b_{NEXT}, $U[k]$, $Q[k]$) согласно п. 4.
Конец цикла.
Восстановим текущее значение $d_{\max}$: $d_{\max} = d_{save}$.
Конец алгоритма.

В результате выполнения приведенного алгоритма в массиве M_{OUT} формируется g октантных групп точек ($g \le n$) размера не более $K_{\max}$ каждая, а в массиве $G_{OUT} - g$ дескрипторов, позволяющих считывать эти группы из массива M_{OUT}.

Как можно заметить, при реализации предложенного алгоритма вместо массивов I_0, I_1 и K можно также использовать структуру данных типа “связный список”, например, двусвязный список *std:: list* или односвязный список *std:: forward_list* из стандартной библиотеки шаблонов C++ [35]. Преимуществом списков является константная сложность операций вставки и удаления элемента, а также возможность переставить элемент внутри списка без его пересоздания (с помощью функций *splice* и *splice_after*, см. [35]). Это позволяет сократить число шагов при обработке последнего случая в п. 4 алгоритма и, в частности, избежать отдельного шага вычисления длин выборок. Однако, элементы списка, в отличие от массива, располагаются в памяти не последовательно друг за другом, а произвольным образом, вследствие чего время перехода от элемента к элементу списка в циклах будет выше. Чтобы оценить эффект от этих факторов, в данной работе были реализованы два дополнительных варианта предложенного алгоритма, основанные на *std:: list* и *std:: forward_list*, и произведено сравнение времен извлечения ОГТ для всех трех реализаций (см. раздел “Результаты”).

4. РЕЗУЛЬТАТЫ

Предложенные метод и алгоритм были реализованы в прототипе визуализатора облаков точек, созданном в рамках нашего предыдущего исследования [9]. Данный программный комплекс разработан на языке C++ с применением языка GLSL программирования шейдеров и API Vulkan [36] и выполняет визуализацию облаков точек в реальном времени с помощью аппаратно-ускоренной трассировки лучей.

Апробация модифицированного визуализатора проводилась при разрешении HD (720p) области вывода на персональном компьютере (Intel Core i7-6800K 6x3.40 ГГц, 16 Гб RAM), оборудованном графической картой NVidia GeForce RTX 2080 (8 Гб VRAM, 46 RT-ядер, 2944 CUDA-ядер, драйвер NVidia DCH 536.40). Для апробации использовались облака точек реальных объектов окружающей среды из банка данных Sketchfab [5] (номера 1–7 в табл. 1)[1] и проекта CAVES NASA [37] (номера 8, 9).

Апробация выполнялась в два этапа. На *первом этапе* проводилось сравнение временных затрат на формирование октантных групп точек с помощью предложенного решения и альтернативных вариантов реализации (см. конец раздела 3). Замеры времени для всех трех вариантов реализации выполнялись на одном и том же оборудовании при значении $K_{\max} = 8$. Результаты замеров приведены в табл. 1.

Из табл. 1 видно, что предложенный в данной работе вариант (*a*) реализации алгоритма извлечения ОГТ работает быстрее альтернативных реализаций (*b*) и (*c*) в среднем в 1.5 и 1.9 раза. Кроме того, ввиду отсутствия необходимости хранения указателей на соседние элементы (один или два), чередующиеся массивы (*a*) требуют меньше памяти (минимум на 30%), чем связные списки (*b*) и (*c*).

На *втором этапе* апробации исследовались зависимости скоростей визуализации облаков точек из табл. 1 и накладных расходов видеопамяти (числа AABBs октантных групп точек) от значения $K_{\max}$. Для этого была проведена серия экспериментов, в которой значение $K_{\max}$ увели-

[1] (1) “Beautiful Autumn”, автор Epic_Tree_Store, https://skfb.ly/VrYw; (2) “Lanyon Quoit – point cloud”, автор Penwith Landscape Partnership, https://skfb.ly/6SOGS; (3) “tree “maybe last” – point cloud”, автор Jer Bot, https://skfb.ly/6BZUP; (4) “Scotland: Dundee, The McManus (5M point cloud)”, автор Daniel Muirhead, https://skfb.ly/6SVFM; (5) “Scan Unscannable: Grass Cloud 1”, автор Epic_Tree_Store, https://skfb.ly/TAF9; (6) “Scan Unscannable: Bush Cloud”, автор Epic_Tree_Store, https://skfb.ly/THSr; (7) “Point Cloud – Plaza Mayor de Almagro, Spain”, автор Global Digital Heritage, https://skfb.ly/6RZrD. Данные модели распространяются по лицензии Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

Таблица 1. Сравнение производительности вариантов реализации алгоритма извлечения ОГТ: (*a*) чередующиеся массивы; (*b*) двусвязный список *std:: list*; (*c*) односвязный список *std:: forward_list*

№	Облако точек	Число точек	Число ОГТ	Время извлечения октантных групп точек, с		
				(*a*)	(*b*)	(*c*)
1	Beautiful Autumn	1 580 297	530 841	0.35	0.52	0.72
2	Lanyon Quoit	1 800 000	531 671	0.79	1.27	1.58
3	Stump	1 839 043	575 533	0.48	0.75	1.01
4	The McManus	5 000 000	1 610 157	2.68	4.31	4.81
5	Grass	5 133 067	1 595 571	1.13	1.63	2.17
6	Bush	5 333 200	1 891 213	1.37	1.72	2.25
7	Plaza Almagro	12 736 674	3 411 935	5.19	8.19	9.33
8	Sheepridge Site	23 882 848	7 407 168	6.28	9.72	11.46
9	King's Bowl	37 508 760	12 042 023	10.49	16.03	19.59

чивалось от 1 до 1024 (по степеням двойки) с сохранением параметров виртуальной камеры для каждого облака точек. Результаты экспериментов показаны на графиках на рис. 1. Рассмотрим их более подробно.

Из левого графика $f_{avg}(K_{\max})$ видно, что предложенный метод упорядочивания облаков точек позволяет получить прирост скорости визуализации до 47% (чем больше размер облака точек, тем больше прирост) по сравнению с референсными значениями при $K_{\max} = 1$ (AABB на точку, как в [34]). Правый график $N_{\text{AABBs}}(K_{\max})$ показывает монотонное убывание числа AABBs (и связанных с ними накладных расходов видеопамяти) с ростом $K_{\max}$. При значениях $K_{\max}$, соответствующих пикам производительности на графике $f_{avg}(K_{\max})$, число AABBs сокращается в 1.9–5.4 раз по сравнению с референсными значениями при $K_{\max} = 1$. Отметим, что даже в случае небольшого прироста производительности (см. кривую *a* на графике $f_{avg}(K_{\max})$) предложенный метод обеспечивает существенное сокращение числа AABBs.

Еще одним интересным наблюдением является то, что с ростом размеров облаков точек пики производительности на графике $f_{avg}(K_{\max})$ становятся менее ярко выраженными и смещаются в область более высоких значений $K_{\max}$. Данный факт может указывать на то, что с точки зрения

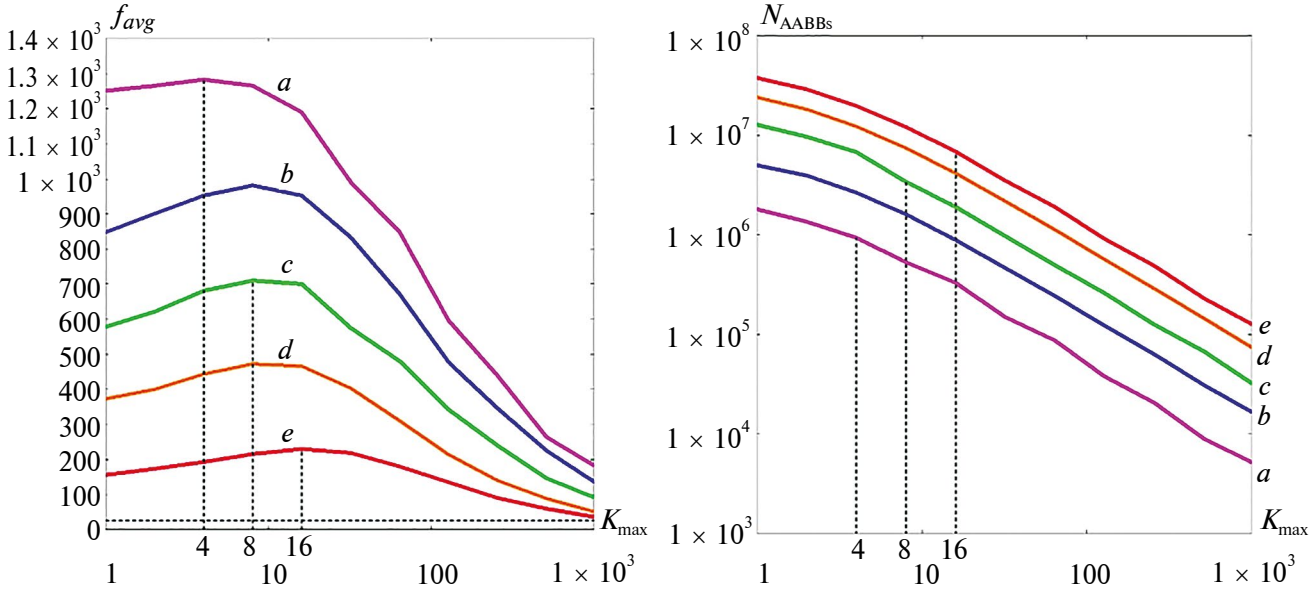
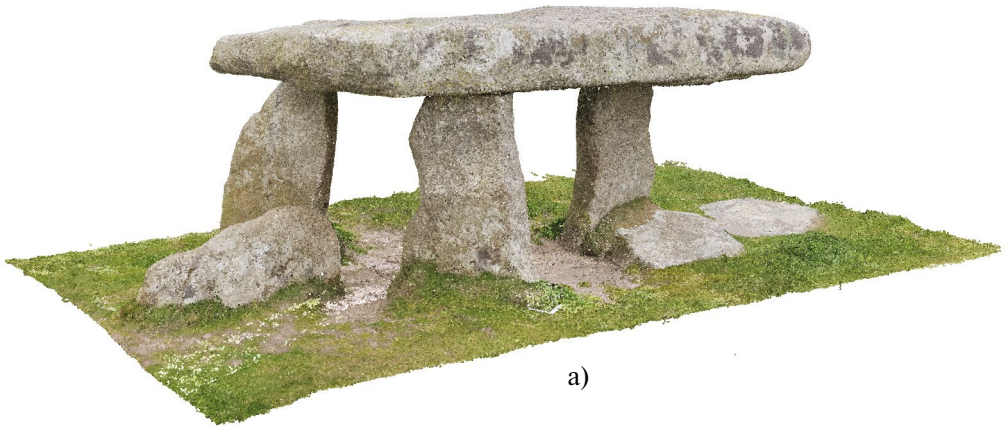
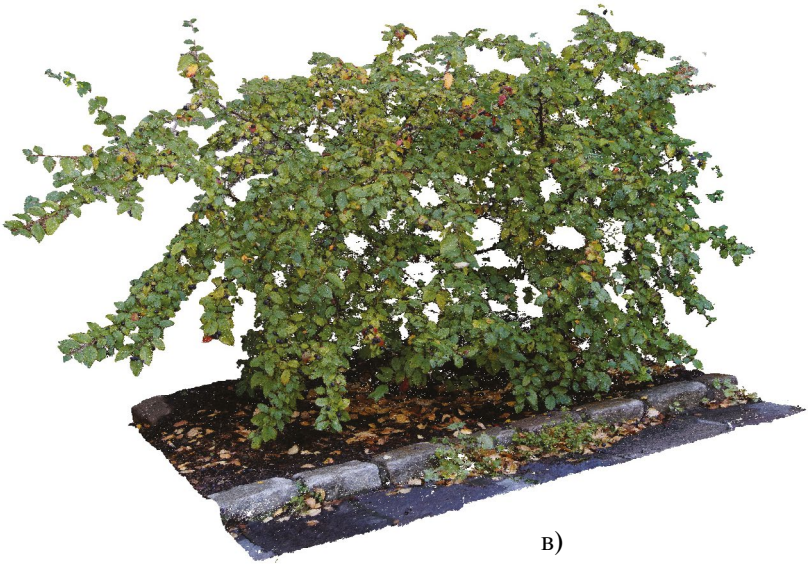


Рис. 1. Влияние параметра $K_{\max}$ на среднюю частоту f_{avg} синтеза изображений, в кадрах в секунду (слева), и на число N_{AABBs} ограничивающих параллелепипедов (справа). На приведенных графиках для осей $K_{\max}$ и N_{AABBs} используется логарифмическая шкала, а кривые *a*, *b*, *c*, *d*, *e* соответствуют облакам точек № 2, 4, 7, 8, 9 из табл. 1. На левом графике пунктирная горизонтальная линия обозначает нижний порог частоты 25 кадров в секунду, соответствующей визуализации в реальном времени.

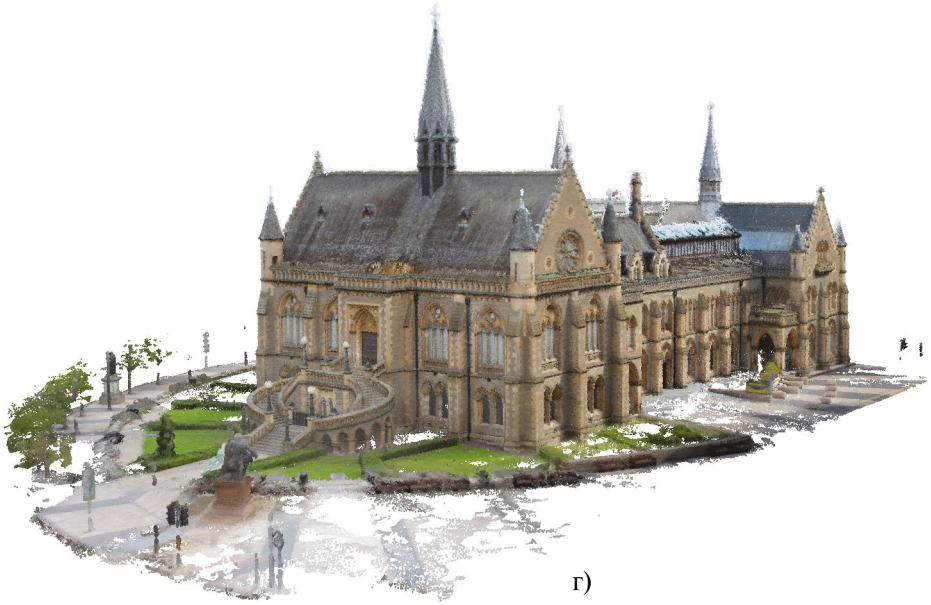


а)

б)



в)



г)

Рис. 2. Примеры кадров визуализации, полученных с помощью нашего модифицированного визуализатора облаков точек. Изображения а) – г) соответствуют облакам точек № 2, 1, 6, 4 из табл. 1.

оптимизации накладных расходов видеопамяти выгоднее объединять несколько небольших облаков точек в одно крупное, чем визуализировать каждое облако точек по отдельности. В этой связи разработка эффективной стратегии визуализации мультиобъектных сцен из облаков точек с помощью аппаратно-ускоренной трассировки лучей представляется нам интересным направлением для будущих исследований.

5. ЗАКЛЮЧЕНИЕ

Современные технологии позволяют создавать подробные цифровые 3D-модели реальных объектов окружающей среды в виде облаков точек, состоящих из миллионов элементов. Интеграция таких объектов в системы виртуального окружения позволяет существенно повысить реалистичность и разнообразие моделируемой обстановки, однако требует эффективных решений для обработки больших объемов точек в реальном времени. В данной работе предложен метод упорядочивания облаков точек, обеспечивающий прирост скорости визуализации до 47% на видеокартах с аппаратным ускорением трассировки лучей, а также эффективное снижение расходов видеопамяти на построение ускоряющих структур данных. Предложенный метод основан на объединении точек облака в октантные группы точек, которые хорошо вписываются в BVH-дерево – ускоряющую иерархическую структуру данных, на основе которой работает конвейер трассировки лучей. В работе предложен алгоритм извлечения октантных групп точек из исходного неупорядоченного облака точек с помощью чередующихся массивов индексов точек, который работает до 1.9 раз быстрее альтернативных решений, основанных на связных списках, и требует минимум на 30% меньше памяти. Разработанный алгоритм может выполняться на этапе загрузки системы визуализации, что позволяет оперативно настраивать пороговый размер $K_{\max}$ октантных групп точек для достижения пиковой производительности на имеющихся исходных данных и оборудовании. Предложенное решение было реализовано в разработанном ранее прототипе визуализатора облака точек и апробировано на ряде облаков точек реальных объектов окружающей среды (на рис. 2 показаны примеры полученных кадров визуализации таких объектов). Полученные результаты подтвердили эффективность разработанного решения и возможность его применения в системах виртуального окружения, видеотренажерных комплексах, научной визуализации, геоинформационных системах и др. В качестве будущей работы планируются исследование и разработка эффективной стратегии визуализации мультиобъектных сцен из облаков точек с помощью аппаратно-ускоренной трассировки лучей.

6. БЛАГОДАРНОСТИ

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0002 “Математическое моделирование многомасштабных динамических процессов и системы виртуального окружения”.

СПИСОК ЛИТЕРАТУРЫ

1. *Guo M., Sun M., Pan D., Wang G., Zhou Y., Yan B., Fu Z.* High-precision deformation analysis of yingxian wooden pagoda based on UAV image and terrestrial LiDAR point cloud // Heritage Science. 2023. V. 11. P. 1–18. https://doi.org/10.1186/s40494-022-00833-z
2. *Adamopoulos E., Papadopoulou E.-E., Mpia M., Deligianni E.-O., Papadopoulou G., Athanasoulis D., Konioti M., Koutsoumpou M., Anagnostopoulos C.N.* 3D Survey and Monitoring of Ongoing Archaeological Excavations via Terrestrial and Drone LIDAR // ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2023. V. X-M-1-2023. P. 3–10. https://doi.org/10.5194/isprs-annals-X-M-1-2023-3-2023
3. *Weiser H., Schäfer J., Winiwarter L., Krašovec N., Fassnacht F.E., Höfle B.* Individual tree point clouds and tree measurements from multi-platform laser scanning in German forests // Earth System Science Data. 2022. V. 14. № 7. P. 2989–3012. https://doi.org/10.5194/essd-14-2989-2022
4. *Risbøl O., Gustavsen L.* LiDAR from drones employed for mapping archaeology – Potential, benefits and challenges // Archaeological Prospection. 2018. V. 25. P. 329–338. https://doi.org/10.1002/arp.1712
5. Sketchfab – The leading platform for 3D & AR on the web. 2023. https://sketchfab.com/
6. *Casado-Coscolla A., Sanchez-Belenguer C., Wolfart E., Sequeira V.* Rendering massive indoor point clouds in virtual reality // Virtual Reality. 2023. V. 27. P. 1859–1874. https://doi.org/10.1007/s10055-023-00766-3
7. *Kharroubi A., Hajji R., Billen R., Poux F.* Classification and Integration of Massive 3D Points Clouds in a Virtual Reality (VR) Environment // The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2019. V. XLII-2/W17. P. 165–171. https://doi.org/10.5194/isprs-archives-XLII-2-W17-165-2019
8. *Discher S., Masopust L., Schulz S., Richter R., Döllner J.* A Point-Based and Image-Based Multi-Pass Render-

ing Technique for Visualizing Massive 3D Point Clouds in VR Environments // Journal of WSCG. 2018. V. 26. № 2. P. 76–84.
https://doi.org/10.24132/JWSCG.2018.26.2.2

9. *Тимохин П.Ю., Михайлюк М.В.* Моделирование особенностей ландшафта с помощью облаков точек в системах виртуального окружения // Труды 33-й Международной конференции по компьютерной графике и машинному зрению (GraphiCon 2023). 2023. С. 157–168.
https://doi.org/10.20948/graphicon-2023-157-168
10. *Kivi P.E.J., Mäkitalo M.J., Žádník J., Ikkala J., Vadakital V.K.M., Jääskeläinen P.O.* Real-Time Rendering of Point Clouds With Photorealistic Effects: A Survey // IEEE Access. 2022. V. 10. P. 13151–13173.
https://doi.org/10.1109/ACCESS.2022.3146768
11. *Kobbelt L., Botsch M.* A survey of point-based techniques in computer graphics // Computers & Graphics. 2004. V. 28. № 6. P. 801–814.
https://doi.org/10.1016/j.cag.2004.08.009
12. *Botsch M., Hornung A., Zwicker M., Kobbelt L.* High-Quality Surface Splatting on Today's GPUs // Proc. Eurographics / IEEE VGTC Symposium Point-Based Graphics. 2005. P. 17–24.
https://doi.org/10.2312/SPBG/SPBG05/017-024
13. *Linsen L., Müller K., Rosenthal P.* Splat-based Ray Tracing of Point Clouds // Journal of WSCG. 2007. V. 15. P. 51–58.
https://dspace5.zcu.cz/bitstream/11025/1426/1/Linsen.pdf
14. *Wald I., Seidel H.-P.* Interactive ray tracing of point-based models // Proceedings Eurographics / IEEE VGTC Symposium Point-Based Graphics (Jun. 2005). 2005. P. 9–16.
https://doi.org/10.1145/1187112.1187176
15. *Adamson A., Alexa M.* Ray tracing point set surfaces // Proceedings of the Shape Modeling International (SMI '03). 2003. P. 272–279.
https://doi.org/10.1109/SMI.2003.1199627
16. *Hubo E., Mertens T., Haber T., Bekaert P.* Self-similarity based compression of point set surfaces with application to ray tracing // Computers & Graphics. 2008. V. 32. № 2. P. 221–234.
https://doi.org/10.1016/j.cag.2008.01.012
17. *Tejada E., Gois J.P., Nonato L.G., Castelo A., Ertl T.* Hardware-accelerated Extraction and Rendering of Point Set Surfaces // Proceedings of the 8th Joint Eurographics – IEEE VGTC Symposium on Visualization (EuroVis '06). 2006. P. 21–28.
https://doi.org/10.2312/VisSym/EuroVis06/021-028
18. *Zhang Y., Pajarola R.* Deferred blending: Image composition for single-pass point rendering // Computers & Graphics. 2007. V. 31. № 2. P. 175–189.
https://doi.org/10.1016/j.cag.2006.11.012
19. *Wimmer M., Scheiblauer C.* Instant points: Fast rendering of unprocessed point clouds // Eurographics Symposium on Point-Based Graphics (eds. Botsch M., Chen B., Pauly M., Zwicker M.). Geneva, Switzerland: The Eurographics Association. 2006. P. 129–136.
https://doi.org/10.2312/SPBG/SPBG06/129-136
20. *Hubo E., Mertens T., Haber T., Bekaert P.* The Quantized kd-Tree: Efficient Ray Tracing of Compressed Point Clouds // 2006 IEEE Symposium on Interactive Ray Tracing, Salt Lake City, UT, USA. 2006. P. 105–113.
https://doi.org/10.1109/RT.2006.280221
21. *Günther C., Kanzok T., Linsen L., Rosenthal P.* A GPGPU-based Pipeline for Accelerated Rendering of Point Clouds // Journal of WSCG. 2013. V. 21. № 2. P. 153–162.
https://dspace5.zcu.cz/bitstream/11025/6978/1/Gunther.pdf
22. *Schütz M., Kerbl B., Wimmer M.* Rendering Point Clouds with Compute Shaders and Vertex Order Optimization // Computer Graphics Forum. 2021. V. 40. № 4. P. 115–126.
https://doi.org/10.1111/cgf.14345
23. *Kashyap S., Goradia R., Chaudhuri P., Chandran S.* Implicit Surface Octrees For Ray Tracing Point Models // ICVGIP'10: proceedings of the 7th Indian Conference on Computer Vision, Graphics and Image Processing (December 2010). 2010. P. 227–234.
https://doi.org/10.1145/1924559.1924590
24. *Karras T.* Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees // Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics. Eurographics Association. 2012. P. 33–37.
https://doi.org/10.2312/EGGH/HPG12/033-037
25. *Kim H.-J., Cengiz Öztireli A., Gross M., Choi S.-M.* Adaptive surface splatting for facial rendering // Computer Animation Virtual Worlds. 2012. V. 23. № 3–4. P. 363–373.
https://doi.org/10.1002/cav.1463
26. *Schütz M., Krösl K., Wimmer M.* Real-Time Continuous Level of Detail Rendering of Point Clouds // IEEE VR2019: the 26th IEEE Conference on Virtual Reality and 3D User Interfaces (VR), Osaka, Japan. IEEE. 2019. P. 103–110.
https://doi.org/10.1109/VR.2019.8798284
27. *Kulik A., Kunert A., Beck S., Matthes C.-F., Schollmeyer A., Kreskowski A., Fröhlich B., Cobb S., D'Cruz M.* Virtual Valcamonica: Collaborative Exploration of Prehistoric Petroglyphs and Their Surrounding Environment in Multi-User Virtual Reality // Presence: Teleoperators and Virtual Environments. 2017. V. 26. № 3. P. 297–321.
https://doi.org/10.1162/pres_a_00297
28. *Schütz M., Herzberger L., Wimmer M.* SimLOD: Simultaneous LOD Generation and Rendering // ArXiv, abs/2310.03567. 2023. P. 1–12.
https://doi.org/10.48550/arXiv.2310.03567
29. NVIDIA Turing GPU Architecture // NVIDIA Corporation. 2018.
https://images.nvidia.com/aem-dam/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

30. *Sanzharov V.V., Frolov V.A., Galaktionov V.A.* Survey of Nvidia RTX Technology // Programming and Computer Software. 2020. V. 46. № 4. P. 297–304. https://doi.org/10.1134/S0361768820030068
31. *Timokhin P.Y., Mikhaylyuk M.V.* An Efficient Technology of Real-time Modeling of Height Field Surface on the Ray Tracing Pipeline // Programming and Computer Software. 2023. V. 49. № 3. P. 178–186. https://doi.org/10.1134/S0361768823030064
32. *Rusch M., Bickford N., Subtil N.* Introduction to Vulkan Ray Tracing // Ray Tracing Gems II. NVIDIA. 2021. P. 213–255. https://doi.org/10.1007/978-1-4842-7185-8_16
33. *Sjoholm J.* Best Practices for Using NVIDIA RTX Ray Tracing (Updated) // NVIDIA Technical Blog. Jul 25, 2022. https://developer.nvidia.com/blog/best-practices-for-using-nvidia-rtx-ray-tracing-updated/
34. *Lefrançois M.-K.* Intersection Shader // NVIDIA Vulkan Ray Tracing Tutorials. 2020–2023. https://github.com/nvpro-samples/vk_raytracing_tutorial_KHR/tree/master/ray_tracing_intersection
35. C++ reference. Containers library. Sequence containers. 2023. https://en.cppreference.com/w/cpp/container
36. Vulkan 1.3.275 – A Specification (with all ratified extensions) // The Khronos Vulkan Working Group. 2024. https://registry.khronos.org/vulkan/specs/1.3-khr-extensions/pdf/vkspec.pdf
37. *Wong U., Whittaker W., Jones H., Whittaker R.* NASA Planetary Pits and Caves Analog Dataset. 2014. https://ti.arc.nasa.gov/dataset/caves/

A METHOD TO ORDER POINT CLOUDS FOR VISUALIZATION ON THE RAY TRACING PIPELINE

© 2024 P. Y. Timokhin[a], M. V. Mikhaylyuk[a]

[a] *Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences" Nakhimovskii pr. 36/1, Moscow, 117218 Russia*

Currently, the digitization of environment objects (vegetation, terrain, architectural structures, etc.) in the form of point clouds is actively developing. The integration of such digitized objects into virtual environment systems allows the quality of the modeled environment to be improved, but requires efficient methods and algorithms for real-time visualization of large point volumes. In this paper the solution of this task on modern multicore GPUs with support of hardware-accelerated ray tracing is researched. A modified method is proposed where the original unordered point cloud is split up into point groups which visualization is effectively parallelized on ray tracing cores. The paper describes an algorithm for constructing such groups using swapping arrays of point indices, which works faster than alternative solutions based on linked lists, and also has lower memory overhead. The proposed method and algorithm were implemented in the point cloud visualization software complex and approbated on a number of digitized environment objects. The results of the approbation confirmed the efficiency of proposed solutions as well as their applicability for virtual environment systems, video simulators and geoinformation systems, virtual laboratories, etc.

Keywords: virtual environment, visualization, real time, ray tracing, point cloud

REFERENCES

1. *Guo M., Sun M., Pan D., Wang G., Zhou Y., Yan B., Fu Z.* High-precision deformation analysis of yingxian wooden pagoda based on UAV image and terrestrial LiDAR point cloud // Heritage Science. 2023. V. 11. P. 1–18. https://doi.org/10.1186/s40494-022-00833-z
2. *Adamopoulos E., Papadopoulou E.-E., Mpia M., Deligianni E.-O., Papadopoulou G., Athanasoulis D., Konioti M., Koutsoumpou M., Anagnostopoulos C.N.* 3D Survey and Monitoring of Ongoing Archaeological Excavations via Terrestrial and Drone LIDAR // ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2023. V. X-M-1-2023. P. 3–10. https://doi.org/10.5194/isprs-annals-X-M-1-2023-3-2023
3. *Weiser H., Schäfer J., Winiwarter L., Krašovec N., Fassnacht F.E., Höfle B.* Individual tree point clouds and tree measurements from multi-platform laser scanning in German forests // Earth System Science Data. 2022. V. 14. № 7. P. 2989–3012. https://doi.org/10.5194/essd-14-2989-2022
4. *Risbøl O., Gustavsen L.* LiDAR from drones employed for mapping archaeology – Potential, benefits and challenges // Archaeological Prospection. 2018. V. 25. P. 329–338. https://doi.org/10.1002/arp.1712
5. Sketchfab – The leading platform for 3D & AR on the web. 2023. https://sketchfab.com/
6. *Casado-Coscolla A., Sanchez-Belenguer C., Wolfart E., Sequeira V.* Rendering massive indoor point clouds in virtual reality // Virtual Reality. 2023. V. 27. P. 1859–1874. https://doi.org/10.1007/s10055-023-00766-3
7. *Kharroubi A., Hajji R., Billen R., Poux F.* Classification and Integration of Massive 3D Points Clouds in a Virtual

Reality (VR) Environment // The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2019. V. XLII-2/W17. P. 165–171. https://doi.org/10.5194/isprs-archives-XLII-2-W17-165-2019

8. *Discher S., Masopust L., Schulz S., Richter R., Döllner J.* A Point-Based and Image-Based Multi-Pass Rendering Technique for Visualizing Massive 3D Point Clouds in VR Environments // Journal of WSCG. 2018. V. 26. № 2. P. 76–84. https://doi.org/10.24132/JWSCG.2018.26.2.2
9. *Timokhin P. Yu., Mikhaylyuk M.V.* Modeling of Landscape Features by Means of Point Clouds in Virtual Environment Systems // Proc. GraphiCon. Proceedings of the 33rd International Conference on Computer Graphics and Vision (GraphiCon 2023). Moscow, 2023. P. 157–168. https://doi.org/10.20948/graphicon-2023-157-168.
10. *Kivi P.E.J., Mäkitalo M.J., Žádník J., Ikkala J., Vadakital V.K.M., Jääskeläinen P.O.* Real-Time Rendering of Point Clouds With Photorealistic Effects: A Survey // IEEE Access. 2022. V. 10. P. 13151–13173. https://doi.org/10.1109/ACCESS.2022.3146768
11. *Kobbelt L., Botsch M.* A survey of point-based techniques in computer graphics // Computers & Graphics. 2004. V. 28. № 6. P. 801–814. https://doi.org/10.1016/j.cag.2004.08.009
12. *Botsch M., Hornung A., Zwicker M., Kobbelt L.* High-Quality Surface Splatting on Today's GPUs // Proc. Eurographics/IEEE VGTC Symposium Point-Based Graphics. 2005. P. 17–24. https://doi.org/10.2312/SPBG/SPBG05/017-024
13. *Linsen L., Müller K., Rosenthal P.* Splat-based Ray Tracing of Point Clouds // Journal of WSCG. 2007. V. 15. P. 51–58. https://dspace5.zcu.cz/bitstream/11025/1426/1/Linsen.pdf
14. *Wald I., Seidel H.-P.* Interactive ray tracing of point-based models // Proceedings Eurographics / IEEE VGTC Symposium Point-Based Graphics (Jun. 2005). 2005. P. 9–16. https://doi.org/10.1145/1187112.1187176
15. *Adamson A., Alexa M.* Ray tracing point set surfaces // Proceedings of the Shape Modeling International (SMI '03). 2003. P. 272–279. https://doi.org/10.1109/SMI.2003.1199627
16. *Hubo E., Mertens T., Haber T., Bekaert P.* Self-similarity based compression of point set surfaces with application to ray tracing // Computers & Graphics. 2008. V. 32. № 2. P. 221–234. https://doi.org/10.1016/j.cag.2008.01.012
17. *Tejada E., Gois J.P., Nonato L.G., Castelo A., Ertl T.* Hardware-accelerated Extraction and Rendering of Point Set Surfaces // Proceedings of the 8th Joint Eurographics – IEEE VGTC Symposium on Visualization (EuroVis '06). 2006. P. 21–28. https://doi.org/10.2312/VisSym/EuroVis06/021-028
18. *Zhang Y., Pajarola R.* Deferred blending: Image composition for single-pass point rendering // Computers & Graphics. 2007. V. 31. № 2. P. 175–189. https://doi.org/10.1016/j.cag.2006.11.012
19. *Wimmer M., Scheiblauer C.* Instant points: Fast rendering of unprocessed point clouds // Eurographics Symposium on Point-Based Graphics (eds. Botsch M., Chen B., Pauly M., Zwicker M.). Geneva, Switzerland: The Eurographics Association. 2006. P. 129–136. https://doi.org/10.2312/SPBG/SPBG06/129-136
20. *Hubo E., Mertens T., Haber T., Bekaert P.* The Quantized kd-Tree: Efficient Ray Tracing of Compressed Point Clouds // 2006 IEEE Symposium on Interactive Ray Tracing, Salt Lake City, UT, USA. 2006. P. 105–113. https://doi.org/10.1109/RT.2006.280221
21. *Günther C., Kanzok T., Linsen L., Rosenthal P.* A GPGPU-based Pipeline for Accelerated Rendering of Point Clouds // Journal of WSCG. 2013. V. 21. № 2. P. 153–162. https://dspace5.zcu.cz/bitstream/11025/6978/1/Gunther.pdf
22. *Schütz M., Kerbl B., Wimmer M.* Rendering Point Clouds with Compute Shaders and Vertex Order Optimization // Computer Graphics Forum. 2021. V. 40. № 4. P. 115–126. https://doi.org/10.1111/cgf.14345
23. *Kashyap S., Goradia R., Chaudhuri P., Chandran S.* Implicit Surface Octrees For Ray Tracing Point Models // ICVGIP'10: proceedings of the 7th Indian Conference on Computer Vision, Graphics and Image Processing (December 2010). 2010. P. 227–234. https://doi.org/10.1145/1924559.1924590
24. *Karras T.* Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees // Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics. Eurographics Association. 2012. P. 33–37. https://doi.org/10.2312/EGGH/HPG12/033-037
25. *Kim H.-J., Cengiz Öztireli A., Gross M., Choi S.-M.* Adaptive surface splatting for facial rendering // Computer Animation Virtual Worlds. 2012. V. 23. № 3–4. P. 363–373. https://doi.org/10.1002/cav.1463
26. *Schütz M., Krösl K., Wimmer M.* Real-Time Continuous Level of Detail Rendering of Point Clouds // IEEE VR2019: the 26th IEEE Conference on Virtual Reality and 3D User Interfaces (VR), Osaka, Japan. IEEE. 2019. P. 103–110. https://doi.org/10.1109/VR.2019.8798284
27. *Kulik A., Kunert A., Beck S., Matthes C.-F., Schollmeyer A., Kreskowski A., Fröhlich B., Cobb S., D'Cruz M.* Virtual Valcamonica: Collaborative Exploration of Prehistoric Petroglyphs and Their Surrounding Environment in Multi-User Virtual Reality // Presence: Teleoperators and Virtual Environments. 2017. V. 26. № 3. P. 297–321. https://doi.org/10.1162/pres_a_00297
28. *Schütz M., Herzberger L., Wimmer M.* SimLOD: Simultaneous LOD Generation and Rendering // ArXiv, abs/2310.03567. 2023. P. 1–12. https://doi.org/10.48550/arXiv.2310.03567

29. NVIDIA Turing GPU Architecture // NVIDIA Corporation. 2018.
https://images.nvidia.com/aem-dam/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf
30. *Sanzharov V.V., Frolov V.A., Galaktionov V.A.* Survey of Nvidia RTX Technology // Programming and Computer Software. 2020. V. 46. № 4. P. 297–304.
https://doi.org/10.1134/S0361768820030068
31. *Timokhin P.Y., Mikhaylyuk M.V.* An Efficient Technology of Real-time Modeling of Height Field Surface on the Ray Tracing Pipeline // Programming and Computer Software. 2023. V. 49. № 3. P. 178–186.
https://doi.org/10.1134/S0361768823030064
32. *Rusch M., Bickford N., Subtil N.* Introduction to Vulkan Ray Tracing // Ray Tracing Gems II. NVIDIA. 2021. P. 213–255.
https://doi.org/10.1007/978-1-4842-7185-8_16
33. *Sjoholm J.* Best Practices for Using NVIDIA RTX Ray Tracing (Updated) // NVIDIA Technical Blog. Jul 25, 2022.
https://developer.nvidia.com/blog/best-practices-for-using-nvidia-rtx-ray-tracing-updated/
34. *Lefrançois M.-K.* Intersection Shader // NVIDIA Vulkan Ray Tracing Tutorials. 2020–2023.
https://github.com/nvpro-samples/vk_raytracing_tutorial_KHR/tree/master/ray_tracing_intersection
35. C++ reference. Containers library. Sequence containers. 2023.
https://en.cppreference.com/w/cpp/container
36. Vulkan 1.3.275 – A Specification (with all ratified extensions) // The Khronos Vulkan Working Group. 2024.
https://registry.khronos.org/vulkan/specs/1.3-khr-extensions/pdf/vkspec.pdf
37. *Wong U., Whittaker W., Jones H., Whittaker R.* NASA Planetary Pits and Caves Analog Dataset. 2014.
https://ti.arc.nasa.gov/dataset/caves/