Моделирование особенностей ландшафта с помощью облаков точек в системах виртуального окружения

П.Ю. Тимохин [1], М.В. Михайлюк [1]

[1] *ФГУ «ФНЦ Научно-исследовательский институт системных исследований РАН», Нахимовский пр., 36, к.1, Москва, 117218, Россия*

Аннотация

В данной работе исследуется направление моделирования и визуализации процедурных объектов в системах виртуального окружения с помощью аппаратно-ускоренной трассировки лучей. В частности, рассматривается задача моделирования объектов, заданных с помощью облаков точек, на примере элементов ландшафта с отрицательными уклонами (пещер, туннелей, утесов и др.). Предлагается подход к решению этой задачи на конвейере трассировки лучей, основанный на построении ограничивающих параллелепипедов, внутри которых точки облака сохраняют близость друг к другу. В работе описываются методы и алгоритмы формирования таких локальных групп точек и ограничивающих их параллелепипедов на основе разреженного воксельного октодерева. Был реализован программный комплекс, основанный на предложенных методах и алгоритмах, и проведена его апробация на ряде облаков точек. Апробация показала, что разработанное решение позволяет моделировать в реальном времени особенности ландшафтов, содержащие десятки миллионов точек, не тратя ресурсы унифицированных ядер GPU. Полученные решения могут быть применены в системах виртуального окружения, научной визуализации, видеосимуляторах, геоинформационных системах, и др.

Ключевые слова

Виртуальное окружение, облако точек, реальное время, конвейер трассировки лучей, GPU.

Modeling of Landscape Features by Means of Point Clouds in Virtual Environment Systems

P.Y. Timokhin [1], M.V. Mikhaylyuk [1]

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

Abstract

The paper studies the direction of modeling and visualization of procedural objects in virtual environment systems, using hardware-accelerated ray tracing. In particular, the task of modeling objects defined by point clouds is considered, exemplified on landscape elements with negative slopes (caves, tunnels, cliffs, etc.). An approach to solve this task on the ray tracing pipeline is proposed, based on constructing bounding boxes, inside which cloud points retain proximity to each other. The paper describes methods and algorithms of forming such local point groups and bounding boxes, basing on a sparse voxel octree. Based on methods and algorithms proposed, a software complex was implemented and tested on a number of point clouds. The approbation showed that developed solution allows landscape features containing tens of millions of points to be modelled in real-time without consuming the power of unified GPU cores. The solutions obtained can be applied in virtual environment systems, scientific visualization, video simulators, geoinformation systems, etc.

Keywords

Virtual environment, point cloud, real time, ray tracing pipeline, GPU.

ГрафиКон 2023: 33-я Международная конференция по компьютерной графике и машинному зрению, 19-21 сентября 2023 г., Институт проблем управления им. В.А. Трапезникова Российской академии наук, г. Москва, Россия
EMAIL: p_tim@bk.ru (П.Ю. Тимохин); mix@niisi.ras.ru (М.В. Михайлюк)
ORCID: 0000-0002-0718-1436 (П.Ю. Тимохин); 0000-0002-7793-080X (М.В. Михайлюк)

1. Введение

Появление в современных видеокартах аппаратно-ускоренной трассировки лучей открыло новые пути решений ряда актуальных задач систем виртуального окружения, связанных с визуализацией сложных 3D-сцен в масштабе реального времени (не более 40 мс на кадр). Одной из таких задач является моделирование и визуализация объектов, заданных в виде наборов несвязанных между собой точек (*облаков точек*), например, элементов ландшафта с отрицательными уклонами (пещер, шахт, лавовых туннелей, антропогенных объектов и др.). Это востребовано в области интерактивной визуализации результатов LIDAR-сканирования окружающей среды [1, 2], создания симуляторов поисково-спасательных, геологоразведочных и строительных работ, при проектировании лунных миссий [3] и др.

У моделируемых элементов ландшафта эффект непрерывной поверхности достигается за счет высокой плотности точек, ввиду чего необходимо, чтобы система виртуального окружения могла обрабатывать в реальном времени многомиллионные облака точек. В данной работе предлагается подход к решению этой задачи на конвейере трассировки лучей (RT-конвейере), основанный на формировании из исходного неупорядоченного облака точек *локальных групп*, в которых сохраняется близость точек друг к другу. Это позволяет эффективно задействовать в решении задачи новые аппаратные возможности GPU - ядра трассировки лучей (RT-ядра), и моделировать в реальном времени элементы ландшафта, состоящих из десятков миллионов точек, не тратя ресурс унифицированных вычислительных ядер GPU (CUDA-ядер), активно используемых в системах виртуального окружения. В предлагаемом решении используются языки C++ и GLSL, а доступ к RT-ядрам реализуется через API Vulkan.

2. Связанные работы

К настоящему времени опубликовано большое число работ, посвященных визуализации на GPU моделей, заданных в виде детализированных облаков точек, и эта тема продолжает активно исследоваться. Первые статьи были посвящены поиску визуальных моделей точек, эффективно обрабатываемых на многоядерных GPU и обеспечивающих визуальное качество, сопоставимое с CPU-решениями, как, например, диски, ориентированные на наблюдателя (сплэттинг) [4, 5]. С появлением в графическом конвейере GPU программируемых (шейдерных) стадий фокус исследований стал смещаться в сторону разработки решений, основанных на распараллеливании обработки облака точек на CUDA-ядрах GPU. В данном направлении можно выделить серию работ под руководством Маркуса Шутца [6-8], которые впоследствии вошли в его докторскую диссертацию [9]. В частности, в работе [8] предлагается распараллеливать обработку точек облака с помощью вычислительных шейдеров, а также показана возможность достижения реального времени визуализации при использовании только фиксированного функционала графического конвейера (растеризации графических примитивов типа GL_POINTS) за счет перестановки точек в перемешанном порядке Мортона. Вместе с освоением растущих вычислительных возможностей GPU также активное развитие получило направление разработки LOD-техник (управления уровнем детализации) [10-11] для обработки на GPU моделей, состоящих из сотен миллионов точек.

Относительно недавнее введение в архитектуру GPU NVidia RT-ядер и работающего на них программируемого конвейера трассировки лучей (RT-конвейера), открыло путь к построению визуализаторов нового типа, основанных на аппаратно-ускоренной трассировке лучей. В работе [12] показана принципиальная возможность визуализации в реальном времени сцены из 2 млн. сгенерированных случайным образом точек, смоделированных сферами и кубами. Построение каждой такой модели выполняется процедурно в отдельном ограничивающем параллелепипеде (Axis-Aligned Bounding Box, AABB). Все такие AABBs также является листьями BVH-дерева (дерева ограничивающих объемов), которое строится автоматически до запуска RT-конвейера. В процессе трассировки лучей специальный аппаратный блок RT-конвейера (блок обхода BVH-дерева, далее BVH-блок) автоматически отбирает AABBs, пересекаемые лучами, и передает их на стадию шейдера обработки пересечений (Intersection Shader, далее I-шейдер), на которой выполняется реализованный разработчиком расчет пересечения луча с процедурной моделью.

Наши предыдущие исследования [13-14] показали, что перегруженный BVH-блок и «легкий» I-шейдер, как и недогруженный BVH-блок и «тяжелый» I-шейдер, приводят к нерациональному расходу вычислительного ресурса RT-ядер и, как следствие, к падению производительности всей системы визуализации. И, наоборот, лучшие результаты продемонстрировало решение, в котором вычислительная нагрузка была сбалансирована между стадиями RT-конвейера. Ввиду этого, в данной работе мы не используем отдельные AABBs для каждой модели точки облака, как в [12], а объединяем близлежащие точки в группы и строим AABBs уже для этих групп. Это позволяет распределить нагрузку между BVH-блоком и I-шейдером, а также уменьшить накладные расходы видеопамяти за счет создания меньшего числа AABBs.

3. Предлагаемый подход

Согласно рекомендациям [15] по эффективной организации аппаратно-ускоренной трассировки лучей, разработчик должен стараться распределить геометрию сцены (в том числе процедурную) между AABBs так, чтобы внутри них было как можно меньше пустого пространства. Наличие больших пустот внутри AABB приводит к систематическим избыточным срабатываниям BVH-блока RT-конвейера, что существенно замедляет просчет лучей. Ввиду того, что исходное облако точек является неупорядоченным, возникает задача построения AABBs, внутри которых точки сохраняют локальность, т.е. близость друг к другу.

Для решения этой задачи в данной работе предлагается подход, основанный на представлении объема, ограничивающего исходное облако точек, в виде *разреженного воксельного октодерева* (РВО) [16]. В отличие от обычного октодерева, РВО включает в себя только те вокселы, которые содержат части объекта (модели). В данной работе листом РВО является воксел, который содержит не более K_{max} точек (величина K_{max} является общей для всех листьев РВО и задается пользователем).

Предлагаемый подход включает три этапа. На ***первом этапе*** из исходного неупорядоченного облака точек формируется последовательность локальных групп точек, соответствующих листьям РВО (в общем случае, каждая такая группа будет содержать $k \in [1, K_{\max}]$ точек). На ***втором этапе*** выполняется построение AABBs процедурных моделей локальных групп точек. На ***третьем этапе*** выполняется визуализация процедурных моделей, входящих в AABBs. Третий этап реализуется на стадии I-шейдера во многом похожим образом, как в работе [12], с отличием в том, что внутри AABB проверяется пересечение луча не с одной процедурной моделью точки, а с k моделями, и из них выбирается ближайшее к наблюдателю. Первый и второй этапы выполняются на стадии предварительной обработки данных и являются ключевыми. Рассмотрим методы их реализации.

3.1. Метод формирования локальных групп точек

Чтобы сформировать локальные группы точек, необходим эффективный инструмент, позволяющий распределять точки облака по вокселам РВО. В качестве основы для такого инструмента мы используем функцию *std::stable_sort* устойчивой сортировки из стандартной библиотеки шаблонов C++ [17]. Данная функция сохраняет относительный порядок сравниваемых элементов с одинаковыми значениями, а также позволяет задавать собственные операторы сравнения. Оператор применяется к паре сравниваемых элементов, и, если условие внутри оператора не выполняется, то элементы меняются местами. В данной работе реализованы следующие три оператора

$$voxelSortZ : \lfloor z_1/d \rfloor < \lfloor z_2/d \rfloor , \quad (1)$$

$$voxelSortY : \lfloor y_1/d \rfloor < \lfloor y_2/d \rfloor , \quad (2)$$

$$voxelSortX : \lfloor x_1/d \rfloor < \lfloor x_2/d \rfloor , \quad (3)$$

где x_1, y_1, z_1 и x_2, y_2, z_2 - координаты позиций сравниваемых точек, d - размер воксела (узла) РВО, а $\lfloor \ \rfloor$ - целая снизу часть числа. Идея состоит в том, что если применить к исходной последовательности точек облака функцию *std::stable_sort* сначала с оператором *voxelSortZ*,

затем с *voxelSortY*, и, наконец, с *voxelSortX*, то в результате получится последовательность, в которой точки будут располагаться по группам, соответствующим вокселам РВО размера *d*.

На основе данной идеи было разработано решение, позволяющее сортировать исходное облако точек по листьям РВО. Для удобства его описания введем следующие обозначения:

- структура *SPoint*, хранящая атрибуты точки облака: координаты (*x*, *y*, *z*) позиции и компоненты (*r*, *g*, *b*) цвета;
- массив *S* облака точек (структур *SPoint*), где *n* - длина массива *S* (совпадает с числом точек в облаке);
- структура *SPointGroup*, хранящая атрибуты локальной группы точек: индекс *first* первой точки группы в массиве *S* и число *size* точек в группе;
- рекурсивная функция *pointCloudSort*(*start*, *q*) сортировки подмассива массива *S*, где *start* - индекс начального элемента подмассива в массиве *S*, а *q* - длина подмассива.

Нашей задачей является получение массива *G* атрибутов локальных групп точек (структур *SPointGroup*), у всех элементов которого $m \le K_{\max}$, а также соответствующего массиву *G* отсортированного массива *S′* облака точек. Для этого создадим пустой массив *G* длиной *n* (соответствует худшему случаю, когда число локальных групп совпадает с числом точек в облаке) и выполним следующий

Алгоритм A1 формирования локальных групп точек

1. Инициализируем переменные и константы алгоритма:
   - *cntGroups* = 0; // счетчик локальных групп точек, добавленных в массив *G*.
   - ε = 1*E*-06; // погрешность машинного представления вещественных чисел.
   - $d = max(max(x_{s,max} - x_{s,min}, y_{s,max} - y_{s,min}), z_{s,max} - z_{s,min})$, // размер воксела (узла) РВО.

   где $x_{s,min}$, $y_{s,min}$, $z_{s,min}$ и $x_{s,max}$, $y_{s,max}$, $z_{s,max}$ - наименьшие и наибольшие координаты *x*, *y*, *z* из массива *S*.
2. Выполним следующую рекурсивную функцию *pointCloudSort* сортировки подмассива массива *S* с аргументами *start* = 0, *q* = *n*.

   Начало функции pointCloudSort.

   2.1. Запишем *end* = *start* + *q* - 1; // индекс конечного элемента подмассива в массиве *S*.

   2.2. Если $d \le \varepsilon$, то выполним обработку случая «слипшихся» точек:

   Цикл по *p*-му индексу из отрезка [*start*, *end*]:

   G[*cntGroups*].*first* = *p*; *G*[*cntGroups*].*size* = 1; *cntGroups* = *cntGroups* + 1;

   Конец цикла.

   Выходим из функции *pointCloudSort*.

   Конец если.

   2.3. Отсортируем подмассив [*start*, *end*] по вокселам размера *d*:

   Выполним функцию *std::stable_sort* с оператором *voxelSortZ* сравнения (1).

   Выполним функцию *std::stable_sort* с оператором *voxelSortY* сравнения (2).

   Выполним функцию *std::stable_sort* с оператором *voxelSortX* сравнения (3).

   2.4. Выделим в подмассиве [*start*, *end*] локальные группы точек и добавим их атрибуты в массив *G*:

   Создадим экземпляр *groupCandidate* структуры *SPointGroup*, хранящий атрибуты кандидата в локальные группы точек, и инициализируем его:

   groupCandidate.first = *start*; *groupCandidate.size* = 1;

   Запишем следующие переменные:

   - текущий размер d_{cur} воксела (узла) РВО: $d_{cur} = d$;
   - индекс *next* следующей точки в подмассиве [*start*, *end*]:

     next = *min*(*start* + 1, *end*);
   - тройки номеров $\{i_{cur}, j_{cur}, k_{cur}\}$ текущего и $\{i_{next}, j_{next}, k_{next}\}$ следующего вокселов вдоль осей *X*, *Y*, *Z*:

$$\{i_{cur}, j_{cur}, k_{cur}\} = \{\lfloor S[start].x/d \rfloor, \lfloor S[start].y/d \rfloor, \lfloor S[start].z/d \rfloor\};$$

$$\{i_{next}, j_{next}, k_{next}\} = \{\lfloor S[next].x/d \rfloor, \lfloor S[next].y/d \rfloor, \lfloor S[next].z/d \rfloor\};$$

   Цикл по *p*-му индексу из отрезка [*start*, *end*]:

Если $\{i_{next}, j_{next}, k_{next}\} \neq \{i_{cur}, j_{cur}, k_{cur}\}$ или p равен end, то:
Если $groupCandidate.size \leq K_{max}$, то:
$G[cntGroups] = groupCandidate$;
$cntGroups = cntGroups + 1$;
в противном случае:
$d = 0.5d$;
Выполним рекурсивную функцию *pointCloudSort* с аргументами $start = groupCandidate.first$, $q = groupCandidate.size$.
$d = d_{cur}$;
Конец если.
$groupCandidate.first = next$; $groupCandidate.size = 0$;
$\{i_{cur}, j_{cur}, k_{cur}\} = \{i_{next}, j_{next}, k_{next}\}$;
Конец если.
$groupCandidate.size = groupCandidate.size + 1$;
$next = min(next + 1, end)$;
$\{i_{next}, j_{next}, k_{next}\} = \{\lfloor S[next].x/d \rfloor, \lfloor S[next].y/d \rfloor, \lfloor S[next].z/d \rfloor\}$;
Конец цикла.
Конец функции pointCloudSort.
Конец алгоритма.

В результате выполнения алгоритма *A1* формируется искомый массив G атрибутов локальных групп точек с числом $cntGroups$ заполненных элементов ($cntGroups \leq n$), а на месте массива S формируется соответствующий массиву G массив S' отсортированных точек облака.

3.2. Метод построения AABBs процедурных моделей локальных групп точек

В данной работе точки облака моделируются процедурно в виде сфер одинакового радиуса. Если локальная группа содержит одну точку, то AABB такой группы будет куб, длина ребра которого равна удвоенному радиусу R вписанной сферы-модели точки (см. рисунок 1а), а координаты $(x_{min}, y_{min}, z_{min})$ и $(x_{max}, y_{max}, z_{max})$ диагональных вершин будут рассчитываться как

$$\begin{aligned}(x_{\min}, y_{\min}, z_{\min}) &= (S'[p].x - R, S'[p].y - R, S'[p].z - R),\\ (x_{\max}, y_{\max}, z_{\max}) &= (S'[p].x + R, S'[p].y + R, S'[p].z + R),\end{aligned} \quad (4)$$

где p - индекс точки в массиве S'. Если локальная группа содержит более одной точки, то AABB такой группы будет строиться по AABB-кубам точек группы (см. рисунок 1б).

Обозначим через *SAabb* структуру атрибутов AABB, хранящую координаты $(x_{min}, y_{min}, z_{min})$ и $(x_{max}, y_{max}, z_{max})$ пары диагональных вершин AABB. Задачей рассматриваемого этапа является получение массива A атрибутов AABBs (структур *SAabb*), соответствующего массиву G атрибутов локальных групп точек и массиву S' облака точек, полученным на предыдущем этапе (см. раздел 3.1). Для этого создадим пустой массив A длиной $cntGroups$ и выполним следующий

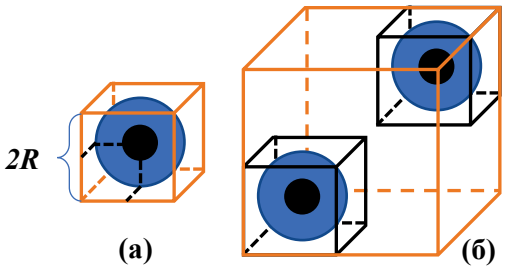

Рисунок 1 – Примеры AABBs локальных групп из (а) одной и (б) двух точек

Алгоритм A2 построения AABBs процедурных моделей локальных групп точек

Цикл по i-му индексу из отрезка $[0, cntGroups)$:
Инициализируем $A[i]$-ую структуру с помощью выражения (4) и $p = G[i].first$.
Если $G[i].size > 1$, то:
Цикл по p-му индексу из отрезка $(G[i].first, G[i].first + G[i].size)$:
Обновим значения $A[i].x_{min}$ и $A[i].x_{max}$:
$A[i].x_{min} = min(A[i].x_{min}, S'[p].x - R)$; $A[i].x_{max} = max(A[i].x_{max}, S'[p].x + R)$.

Обновим $A[i].y_{min}$, $A[i].y_{max}$ и $A[i].z_{min}$, $A[i].z_{max}$ аналогично $A[i].x_{min}$, $A[i].x_{max}$.

Конец цикла.

Конец если.

Конец цикла.

Конец алгоритма.

В результате выполнения алгоритма *A2* формируется искомый массив *A* атрибутов AABBs. Если сравнивать AABB локальной группы точек и соответствующий ей воксел-лист PBO, то, в общем случае, AABB будет находиться внутри воксел-листа. Однако, в отдельных случаях AABB может выходить за границы воксела-листа на величину до *R* (когда точка группы попадает на границу воксел-листа) и пересекаться со соседними AABBs. Наши исследования показывают, что такие малые пересечения не оказывают значительного влияния на производительность RT-конвейера.

На рисунке 2 показаны примеры разбиения реального облака точек на AABBs с помощью разработанного метода.

Рисунок 2 – Визуализация AABBs, построенных с помощью разработанного метода, при K_{max} = 256 (а) и K_{max} = 1 (б). Для визуализации была использована модель «Little cabin» автора Epic_Tree_Store [18], распространяемая по лицензии Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/)

Из рисунка 2а видно, что даже при большом значении K_{max} сохраняется узнаваемость модели, описываемой облаком точек, что подтверждает хорошую локальность сформированных групп

точек. Наиболее детальная модель ожидаемо получается при K_{max} = 1 (см. рисунок 2б), когда число AABBs (далее N_{AABBs}), получаемых в результате препроцессинга облака точек, совпадает с числом точек (далее N_{points}) в облаке точек. В таблице 1 на примере модели из рисунка 2 показано влияние параметра K_{max} на производительность BVH-блока RT-конвейера, выраженную через времена $t_{AABB,\ HD}$ и $t_{AABB,\ Full\ HD}$ синтеза изображений AABBs (в миллисекундах) при HD и Full HD разрешении экрана.

Таблица 1 – Влияние параметра K_{max} на производительность BVH-блока RT-конвейера

K_{max}	N_{AABBs}	$t_{AABB,\ HD}$	$t_{AABB,Full\ HD}$
1	5 983 843	5,13	7,80
2	4 861 771	4,44	6,46
4	3 496 588	3,53	4,83
8	2 265 157	2,66	3,44
16	1 148 595	1,67	2,04
32	704 430	1,25	1,59
64	308 233	0,73	0,98
128	187 984	0,66	0,89
256	79 195	0,66	0,82
512	47 309	0,65	0,80
1024	19 782	0,64	0,78

Как видно из таблицы 1, повышение значения K_{max} приводит к сокращению общего числа AABBs, обрабатываемых RT-конвейером, вследствие чего возрастает производительность BVH-блока, а также уменьшаются накладных расходы видеопамяти.

4. Результаты

На основе предложенных методов и алгоритмов был реализован программный комплекс, выполняющий визуализацию облаков точек с помощью аппаратно-ускоренной трассировки лучей. Комплекс разработан на языке C++ с применением API Vulkan (версия 1.3.204.1) и языка GLSL для программирования RT-конвейера.

Была проведена апробация созданного комплекса на задаче моделирования и визуализации элементов ландшафта с отрицательными уклонами (пещер, шахт, туннелей и др.) из набора данных «NASA Planetary Pits and Caves Analog Dataset» [19], полученного на основе LIDAR-съемки реальных наземных объектов, имитирующих рельеф Луны (Craters of the Moon National Monument, COTM). В таблице 2 приведены характеристики моделируемых объектов, включая суммарное время t_{prep} препроцессинга облака точек (в секундах) и отдельное время t_{group} формирования локальных групп точек (1-ый этап предложенного подхода). При расчете значений N_{AABBs}, t_{prep} и t_{group} использовалось значение K_{max} = 8, при котором скорости визуализации (в кадрах в секунду) у всех моделируемых объектов были близки к наибольшим значениям (см. рисунок 3).

Таблица 2 – Характеристики моделируемых объектов

№	Объект (облако точек)	N_{points}	N_{AABBs}	t_{prep} (t_{group})
1	King’s Bowl (ver. cleaned)	3 740 782	1 324 484	~ 18 (14)
2	IndianTunnel (ver. full_10x)	11 620 540	4 268 314	~ 57 (54)
3	Sheepridge Site	23 882 848	8 554 258	~ 116 (109)
4	King’s Bowl (ver. orig)	37 508 760	13 775 505	~ 214 (195)

Апробация проводилась на персональном компьютере (Intel Core i7-6800K 3.40 ГГц, RAM 16 Гб DDR4) с установленной видеокартой NVidia GeForce RTX 2080 (8 Гб GDDR6, 46 RT-ядер, 2944 CUDA-ядер, драйвер NVidia DCH версии 536.40) при разрешении экрана Full HD. На рисунках 4, 5 и 6 показаны примеры кадров визуализации элементов рельефа «King’s Bowl»,

«IndianTunnel» и «Sheepridge Site», а также приведены скорости и времена визуализации этих кадров (в левом верхнем углу).

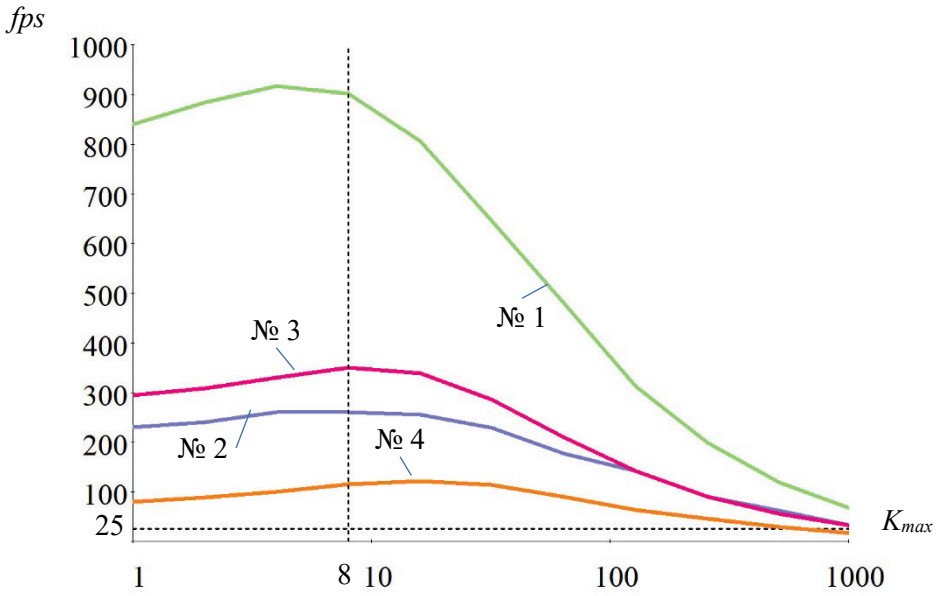

Рисунок 3 – Зависимость скорости визуализации (*fps*) объектов из таблицы 2 от параметра K_{max}

Также в ходе апробации было произведено сравнительное тестирование производительности предложенного решения и подхода, описанного в работе [12] (см. таблицу 3). Для этого было установлено значение $K_{max} = 1$, соответствующее соотношению 1:1 числа AABB и точек облака, как в работе [12], и выполнена визуализация объектов «King's Bowl (ver. cleaned)» и «Little cabin» с включенной и отключенной сортировкой массива *S*, описанной в разделе 3.1.

Таблица 3 – Сравнение времени синтеза кадров у решений, использующих аппаратно-ускоренную трассировку лучей

Объект (облако точек)	Наше решение	Подход NVidia
King's Bowl (ver. cleaned)	1,16 ms	1,80 ms
Little cabin	1,26 ms	1,48 ms

Из таблицы 3 видно, что наше решение даже в условиях перегруженности BVH-блока RT-конвейера (при $K_{max} = 1$) показывает более высокую производительность (прирост 20-50%), чем подход от NVidia.

5. Заключение

В данной работе предложен эффективный подход к моделированию на GPU в реальном времени объектов, заданных облаками из миллионов точек, с помощью аппаратно-ускоренной трассировки лучей. Ключевым преимуществом разработанного решения является алгоритм упаковки исходного неупорядоченного облака точек в группы близлежащих друг к другу точек, что дает заметный выигрыш в скорости визуализации при использовании RT-ядер GPU. Разработанные методы и алгоритмы были апробированы на задаче моделирования элементов ландшафта с отрицательными уклонами (пещер, шахт, лавовых туннелей и др.), состоящих из десятков миллионов точек. Результаты апробации подтвердили высокую эффективность предложенного решения и его применимость для построения систем виртуального окружения и научной визуализации, различных симуляторов (поисково-спасательных, геологоразведочных, строительных) и др. В качестве дальнейшей работы планируется повысить скорость препроцессинга облаков точек и развить исследование в направлении визуализации деформируемых моделей, заданных облаками точек.

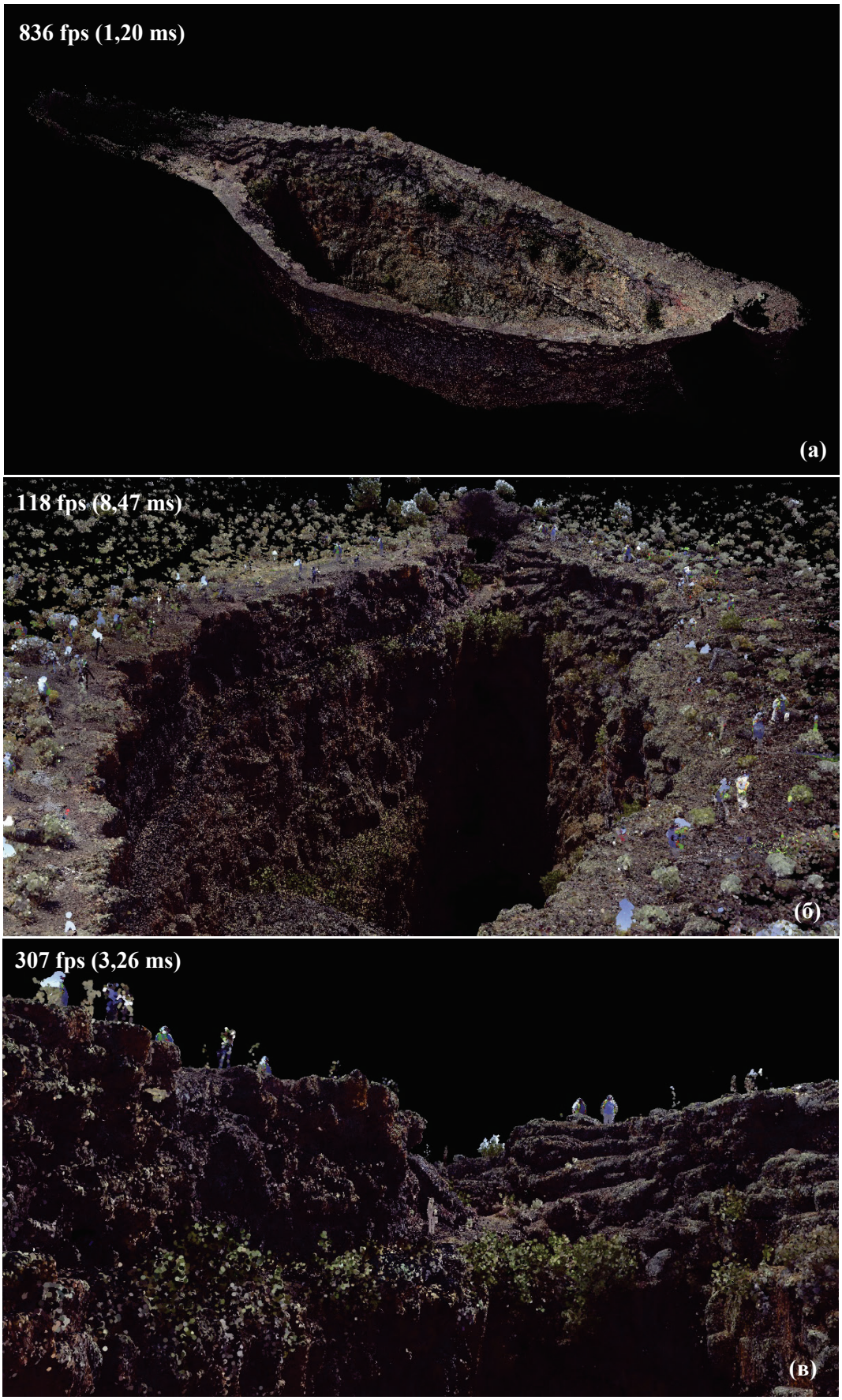

Рисунок 4 – Примеры кадров визуализации облака точек «King's Bowl» с помощью предложенного решения: (а) очищенной версии (ver. cleaned) и оригинальной версии (ver. orig) снаружи (б) и изнутри (в).

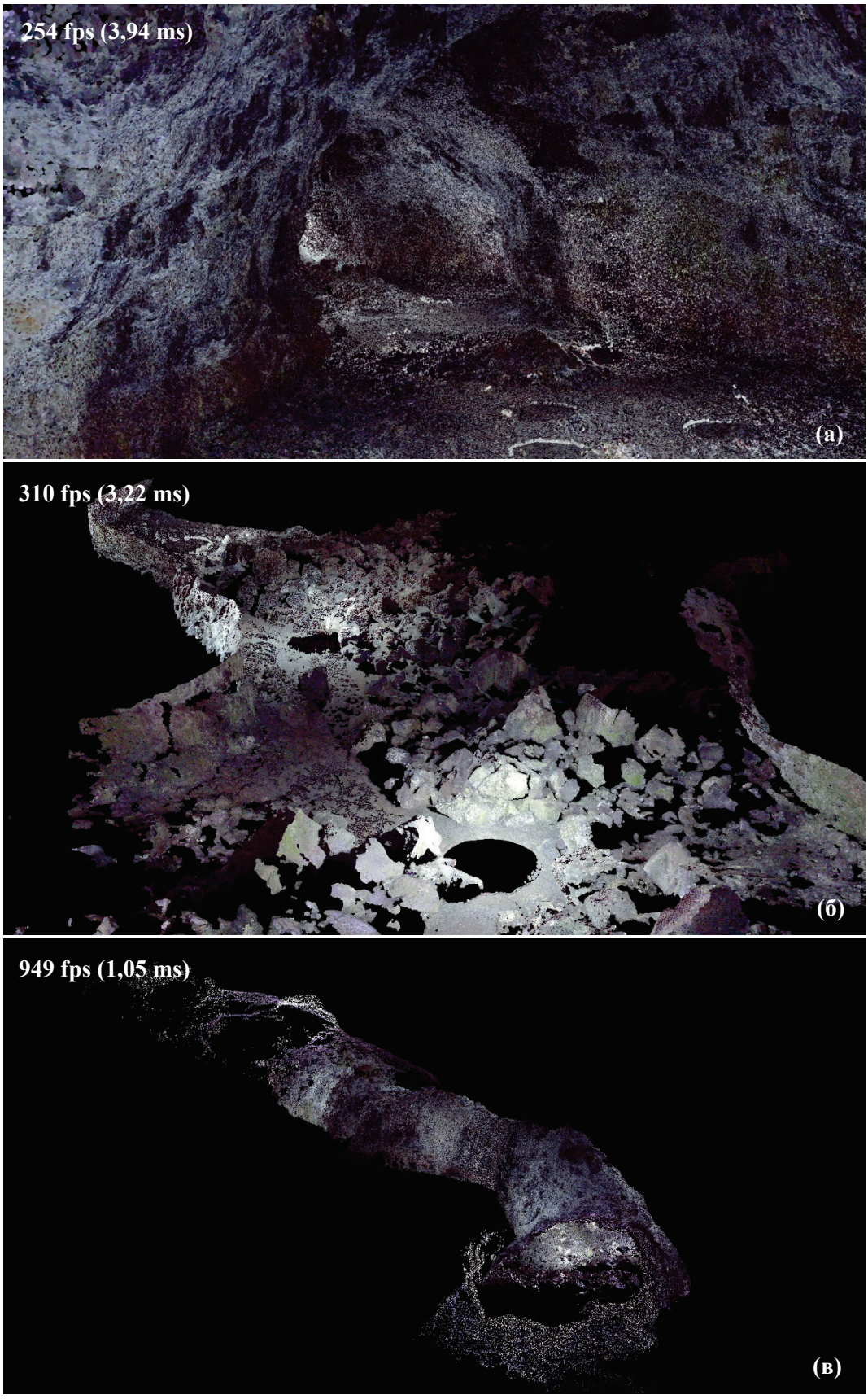

Рисунок 5 – Примеры кадров визуализации облака точек «IndianTunnel» с помощью предложенного решения: (а, б) изнутри и (в) снаружи.

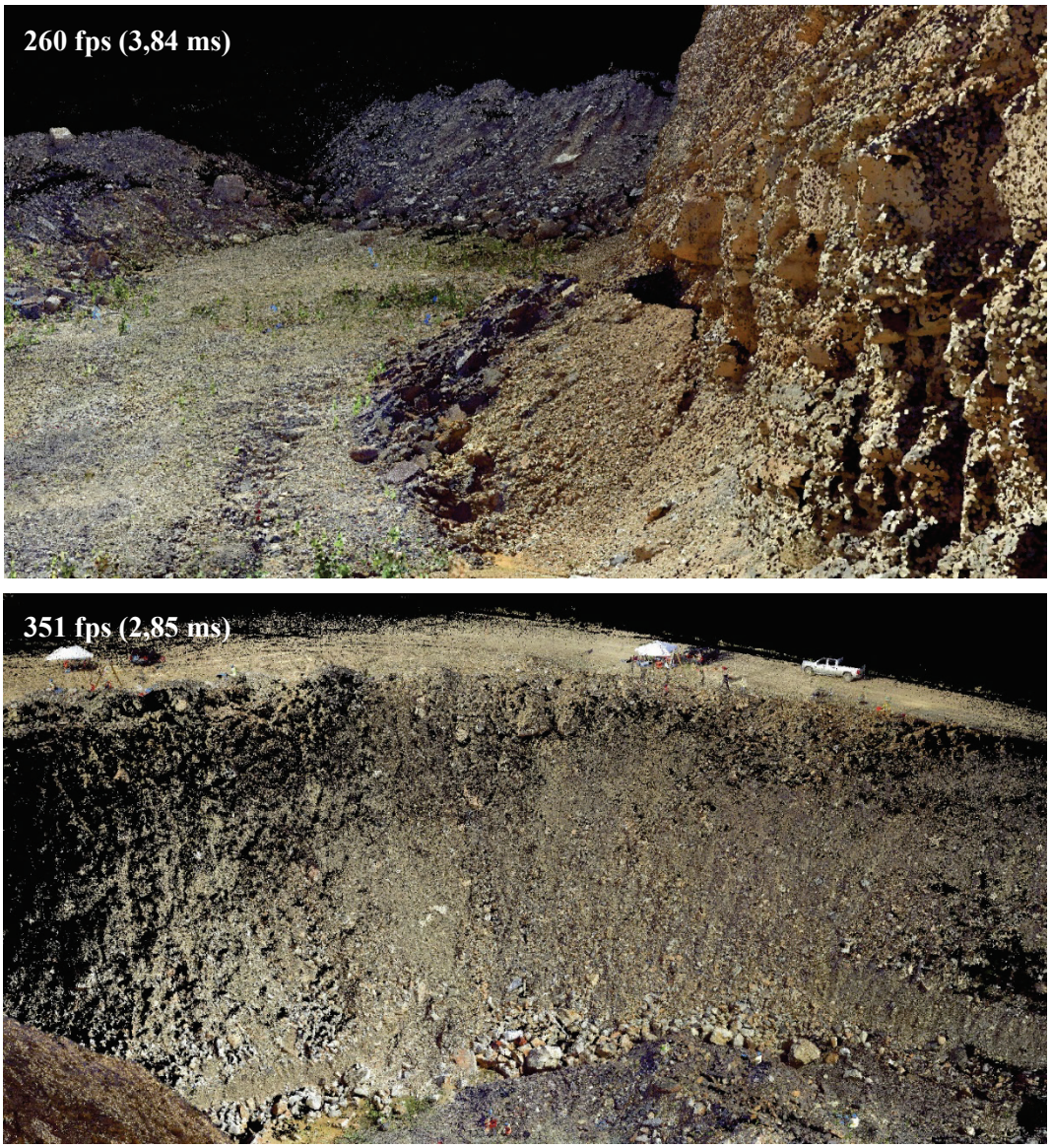

Рисунок 6 – Примеры кадров визуализации облака точек «Sheepridge Site» из различных позиций наблюдателя с помощью предложенного решения.

6. Благодарности

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН “Проведение фундаментальных научных исследований (47 ГП)” по теме № FNEF-2022-0012 “Системы виртуального окружения: технологии, методы и алгоритмы математического моделирования и визуализации”.

7. Список источников

[1] Progressive Real-Time Rendering of One Billion Points Without Hierarchical Acceleration Structures / M. Schütz, G. Mandlburger, J. Otepka, M. Wimmer // Eurographics 2020. Vol. 39. No. 2. 2020. P. 51–64. DOI: 10.1111/cgf.13911.

[2] Rendering massive indoor point clouds in virtual reality / A. Casado-Coscolla, C. Sanchez-Belenguer, E. Wolfart, V. Sequeira // Virtual Reality. 2023. DOI: 10.1007/s10055-023-00766-3.

[3] Крючков Б.И., Куликов И.Н., Бурдин Б.В. Исследование и использование лавовых туннелей в перспективных лунных миссиях // Пилотируемые полеты в космос. 2022. № 1(42). С. 98–118. DOI: 10.34131/MSF.22.1.98-118.
[4] Botsch M., Kobbelt L.P. High-quality point-based rendering on modern GPUs // 11th Pacific Conference on Computer Graphics and Applications (08-10 October 2003), 2003. P. 335–343. DOI: 10.1109/PCCGA.2003.1238275.
[5] Linsen L., Müller K., Rosenthal P. Splat-based Ray Tracing of Point Clouds // Journal of WSCG. Vol. 15. 2007. P. 51–58. URL: https://dspace5.zcu.cz/bitstream/11025/1426/1/Linsen.pdf. (дата обращения 05.07.2023).
[6] Schütz M., Wimmer M. Progressive Real-Time Rendering of Unprocessed Point Clouds // SIGGRAPH'18 Posters (August 12-16, 2018, Vancouver, BC, Canada). ACM, New York, NY, USA, P. 1–2. DOI: 10.1145/3230744.3230816.
[7] Schütz M., Krösl K., Wimmer M. Real-Time Continuous Level of Detail Rendering of Point Clouds // IEEE VR 2019: the 26th IEEE Conference on Virtual Reality and 3D User Interfaces (VR), Osaka, Japan. P. 103–110. IEEE. 2019. DOI: 10.1109/VR.2019.8798284.
[8] Schütz M., Kerbl B., Wimmer M. Rendering Point Clouds with Compute Shaders and Vertex Order Optimization // arXiv preprint arXiv:2104.07526v1, 2021. P. 1–18. URL: https://arxiv.org/pdf/2104.07526.pdf. (дата обращения 05.07.2023). DOI: 10.48550/arXiv.2104.07526.
[9] Schütz M. Interactive Exploration of Point Clouds: dissertation for the degree of the Doctor of Technical Sciences: registration number 00825723 / TU Wien. 2021. 106 pages.
[10] Futterlieb J., Teutsch C., Berndt D. Smooth Visualization of Large Point Clouds // IADIS International Journal on Computer Science and Information Systems. 2016. Vol. 11. No. 2. P. 146–158. URL: https://iadisportal.org/ijcsis/papers/2016190211.pdf. (дата обращения 05.07.2023).
[11] GPU-Accelerated LOD Generation for Point Clouds / M. Schütz, B. Kerbl, P. Klaus, M. Wimmer // arXiv preprint arXiv:2302.14801v1, 2023. P. 1–11. URL: https://arxiv.org/pdf/2302.14801.pdf. (дата обращения 05.07.2023). DOI: 10.48550/arXiv.2302.14801.
[12] Lefrançois M.-K. Intersection Shader // NVIDIA DesignWorks Samples. 2020-2023. URL: https://github.com/nvpro-samples/vk_raytracing_tutorial_KHR/tree/master/ray_tracing_intersection. (дата обращения 05.07.2023).
[13] Тимохин П.Ю., Михайлюк М.В. Рендеринг детализированных полей высот в реальном времени с использованием аппаратного ускорения трассировки лучей // ГрафиКон 2022: 32-я Международная конференция по компьютерной графике и машинному зрению (Рязань, 19-22 сентября 2022 г.). – М.: Институт прикладной математики им. М.В. Келдыша РАН, 2022. С. 124–135. DOI: 10.20948/graphicon-2022-124-135.
[14] Тимохин П.Ю., Михайлюк М.В. Эффективная технология моделирования в реальном времени поверхности поля высот на конвейере трассировки лучей // Программирование. 2023. № 3. С. 56–64. DOI: 10.31857/S0132347423030068, EDN: DENOFM.
[15] Sjoholm J. Best Practices: Using NVIDIA RTX Ray Tracing // NVIDIA Developer Technical Blog. 2020. URL: https://developer.nvidia.com/blog/best-practices-using-nvidia-rtx-ray-tracing/. (дата обращения 05.07.2023).
[16] Laine S., Karras T. Efficient Sparse Voxel Octrees // IEEE Transactions on Visualization and Computer Graphics. 2010. Vol. 17. P. 1048–1059. DOI: 10.1145/1730804.1730814.
[17] Standard Template Library: Algorithms: std::stable_sort // Standard C++ Library reference. 2000-2023. URL: https://cplusplus.com/reference/algorithm/stable_sort/. (дата обращения 05.07.2023).
[18] 3D Model «Little cabin» by Epic_Tree_Store (licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/)) // Sketchfab. URL: https://skfb.ly/TGrH. (дата обращения 05.07.2023).
[19] NASA Planetary Pits and Caves Analog Dataset / U. Wong, W. Whittaker, H. Jones, R. Whittaker. December, 2014. URL: https://ti.arc.nasa.gov/dataset/caves/. (дата обращения 05.07.2023).