Рендеринг детализированных полей высот в реальном времени с использованием аппаратного ускорения трассировки лучей

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 работе рассматривается задача визуализации в реальном времени детализированных полей высот на GPU. Предлагается эффективная двухэтапная технология рендеринга процедурной поверхности поля высот, основанная на аппаратно-ускоренной трассировке первичных лучей. Первый, предварительный, этап включает построение ускоряющей структуры данных - 2D массива параллелепипедов (AABB), ограничивающих поверхность поля высот. На втором этапе для каждого кадра визуализации реализуется построение изображения поля высот на конвейере трассировки лучей, включая генерацию лучей, определение их пересечений с AABB и поиск билинейной поверхностью поля высот внутри этих AABB. В работе описаны ключевые методы создания бесщелевого AABB-массива и отбора AABB-кандидатов с ближайшими к наблюдателю пересечениями «луч-поверхность». На основе созданной технологии разработан программный комплекс на языке C++ с использованием API Vulkan и шейдерного языка GLSL. Выполнено тестирование разработанного комплекса на ряде детализированных карт высот, включая карту лунного ударного кратера Аристарх размером 10K × 4K текселов. Полученные результаты подтвердили высокую эффективность разработанного решения и возможность его применения в системах виртуального окружения, видеотренажерных комплексах, системах научной визуализации и др.

Ключевые слова

Поле высот, рейкастинг, аппаратное ускорение, RTX, конвейер, AABB, GPU.

Real-time Rendering of Detailed Height Fields Using Hardware-based Ray Tracing Acceleration

P.Yu. 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 considers the task of real-time visualization of detailed height fields on the GPU. An efficient, two-stage technology for rendering procedural surface of height field, based on hardware-accelerated tracing of primary rays, is proposed. The first stage, preprocessing, includes constructing an accelerating data structure - a 2D array of bounding boxes (AABB) covering height field surface. At the second stage, for each visualization frame, an image of height field is constructed on the ray tracing pipeline, involving ray generation, «ray-AABB» intersection detection and searching inside the AABB for the intersection of the ray with bilinear surface of height field. The paper describes key methods for creating a gapless AABB array and extracting AABB-candidates with closest to the viewer «ray-surface» intersections. Based on the created technology, a software complex in C++ using the Vulkan API and the GLSL shading language was developed. The complex was tested on a number of detailed height maps, including 10K × 4K map of the lunar impact crater Aristarchus. The results obtained confirmed high efficiency of the

ГрафиКон 2022: 32-я Международная конференция по компьютерной графике и машинному зрению, 19-22 сентября 2022 г., Рязанский государственный радиотехнический университет им. В.Ф. Уткина, Рязань, Россия
EMAIL: webpismo@yahoo.de (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)

developed solution and the possibility of its application in virtual environment systems, simulators, scientific visualization systems, etc.

Keywords
Height field, ray casting, hardware acceleration, RTX, pipeline, AABB, GPU.

1. Введение

Несколько лет назад в компьютерной графике произошло знаковое событие, связанное с появлением в серийных графических картах новой параллельной архитектуры RTX (Ray Tracing Texel eXtreme) аппаратного ускорения трассировки лучей [1, 2]. В дополнение к существующим универсальным вычислительным ядрам (CUDA-ядрам) в графический процессор были введены специализированные вычислительные ядра нового типа (*RT-ядра*), предназначенные для эффективной генерации лучей и расчета их многократных пересечений с объектами сцены. Несмотря на то, что число RT-ядер существенно уступает числу CUDA-ядер (68 против 4352 в картах NVidia GeForce 2080Ti), уже первые тесты [3] показали, что новая архитектура обладает мощным потенциалом для решения задач трассировки лучей в реальном времени.

В данной работе исследуется применение аппаратного ускорения трассировки лучей для задачи рендеринга в реальном времени детализированных полей высот ($1K \times 1K$ и выше). Это широко востребовано в системах виртуального окружения и различного рода симуляторах для визуализации ландшафтов [4], мезоструктурных поверхностей [5], моделирования деформации [6] и др. В данной статье предлагается технология рендеринга детализированных полей высот, основанная на эффективной системе отбора *ограничивающих параллелепипедов* (Axis-Aligned Bounding Box, AABB), работающей полностью на RT-ядрах GPU. Предлагаемое решение реализовано на языке C++ с применением API Vulkan и языка GLSL программирования шейдеров.

2. Связанные работы

Исторически сложилось, что большая часть связанных исследований посвящена построению и визуализации полигональных (триангулированных) аппроксимаций полей высот (обзоры таких методов и алгоритмов можно найти в работах [4, 7]). Преимуществом подхода является высокая скорость обработки треугольников на современных тысячеядерных GPU, а основное ограничение вытекает из сути подхода: чем выше детализация поля высот, тем больше треугольников требуется для его гладкой аппроксимации. В случае детализированных полей высот это приводит к необходимости использовать сложные оптимизационные техники по динамическому понижению числа визуализируемых полигонов (адаптивная тесселяция). Другой важной проблемой является построение аккуратных динамических теней от таких адаптивных моделей. Частично это удается обойти с помощью рейкастинга теней, выполняемого напрямую по карте высот [8].

В данной работе мы исследуем второе основное направление, при котором рендеринг полей высот выполняется с помощью обратной трассировки лучей (от наблюдателя к источнику света). Один из вариантов прямой реализации данного подхода описан в работе [9], где в пространстве, разбитом на кубические ограничивающие объемы (вокселы), сначала вдоль луча ищется воксел, содержащий участок поверхности, а затем в этом вокселе выполняется расчет пересечения луча с интерполированной поверхностью. В работе [10] приведен классический алгоритм обхода вокселов вдоль трассируемого луча. Учитывая линейную сложность алгоритма, его прямая реализация приводит к серьезному падению скорости визуализации (выраженной в количестве кадров в секунду) на детализированных полях высот. Ввиду этого сформировалось три основных пути по достижению скорости визуализации реального времени (от 25 кадров в секунду и выше):

- аппроксимация пересечения луча с поверхностью (равномерная выборка значений высот вдоль луча [11], комбинация равномерной выборки и бинарного поиска [12, 13]);
- сокращение числа затратных тестов пересечений «луч-поверхность» с помощью ускоряющих структур данных (коническая карта [14], коническо-цилиндрические формы [15], мип-карты максимальных высот [16, 17]);

- распараллеливание расчетов на универсальных вычислительных ядрах GPU (с помощью фрагментных шейдеров [15-18], программно-аппаратной архитектуры CUDA [11, 19]).

С появлением архитектуры RTX перед исследователями открылся новый эффективный путь, основанный на распараллеливании трассировки лучей на специализированных RT-ядрах GPU. В частности, архитектура RTX включает в себя программируемый *конвейер трассировки лучей* (RT-конвейер), в котором аппаратно реализован обход дерева ограничивающих объемов (Bounding Volume Hierarchy, BVH), являющегося входным форматом описания виртуальной сцены, а также алгоритм расчета пересечения луча с треугольником. Кроме полигональных моделей, листьями BVH-дерева могут быть и так называемые *процедурные примитивы*, т.е. геометрия которых вычисляется в процессе визуализации (в нашей задаче это поверхность поля высот). Для таких случаев в RT-конвейере предусмотрена специальная программируемая стадия («Пересечение» на рисунке 1), на которой разработчик может реализовать расчет пересечения луча со своим сложным процедурным объектом. Несмотря на малое число примеров и высокую трудоемкость разработки [20], данная область в настоящее время активно исследуется [21, 22].

В данной работе предлагается эффективная технология рендеринга детализированных полей высот с помощью аппаратно-ускоренной трассировки *первичных* лучей (до ближайшего пересечения с поверхностью поля высот). В технологии поверхность поля высот ограничивается двумерным массивом AABB, который добавляется в общее BVH-дерево сцены. Построение BVH-дерева выполняется на GPU и, даже в случае высокодетализированных полей высот, занимает секунды, в отличие, например, от конических карт [14], построение которых может занимать часы. Обход BVH-дерева аппаратно распараллелен на RT-конвейере, что является ключевым преимуществом по сравнению с подходами [16, 17], в которых проход по дереву для каждого луча реализуется последовательно во фрагментном шейдере. Другим важным преимуществом предлагаемого решения является расчет пересечения «луч-билинейная поверхность» с помощью адаптированной реализации продвинутого геометрического подхода GARP [23]. Данный подход работает в разы быстрее известного аналитического решения [24] (используемого, в частности, в [16]), которое имеет сильное ветвление и требует реализацию через double-числа для корректной работы.

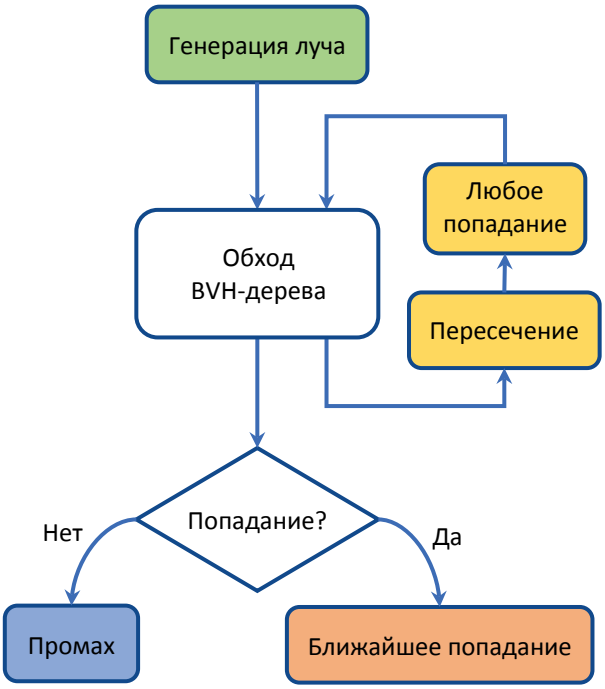

Рисунок 1 – Конвейер трассировки лучей

Для реализации аппаратно-ускоренной трассировки лучей из трех доступных в настоящее время API (NVidia OptiX, Microsoft DirectX12 и Vulkan) в качестве базового было выбрано API Vulkan [25]. Это открытый кроссплатформенный стандарт для высокопроизводительной 2D, 3D графики и вычислений, который разрабатывает тот же промышленный консорциум (Khronos Group), что и традиционное API OpenGL. Ключевыми преимуществами Vulkan перед другими API являются его нацеленность на снижение задержек и накладных расходов при обработке графических команд, обеспечение прямого доступа к GPU для полного контроля над его работой, а также эффективная поддержка многопоточности и уменьшение нагрузки на CPU.

3. Предлагаемая технология рендеринга детализированного поля высот

Как отмечалось выше, предлагаемая технология основана на распараллеливании трассировки первичных лучей на RT-конвейере (см. рисунок 1). Данный конвейер включает в себя

центральный аппаратно-реализованный *блок обхода BVH-дерева* сцены вдоль луча, а также ряд взаимосвязанных с ним программируемых стадий:

- *Ray Generation Shader (RG-шейдер*) - генерация луча;
- *Intersection Shader (I-шейдер*) - пересечение луча с примитивом;
- *Any-Hit Shader (AH-шейдер*) - любое попадание луча в примитив;
- *Closest Hit Shader* (*CH-шейдер*) - ближайшее попадание луча в примитив;
- *Miss Shader* (*M-шейдер*) - промах луча.

Опишем кратко принцип работы предлагаемой технологии. Вначале мы переводим исходное детализированное поле высот в формат, понятный RT-конвейеру. Для этого мы строим на основе поля высот двумерный массив AABB и добавляем его в BVH-дерево сцены. Далее на RT-конвейере запускается процесс построения изображения поля высот. Для этого на стадии RG-шейдера мы генерируем лучи, проходящие через центры пикселов области вывода (см. рисунок 2), и отправляем их в аппаратный блок обхода BVH-дерева. Для каждого сгенерированного луча блок обхода ищет AABB, пересекаемые этим лучом (в общем случае их может быть несколько). Из найденных AABB мы отбираем те, внутри которых луч пересекает билинейную поверхность поля высот (в данной работе мы будем называть их *AABB-кандидаты*). Это реализуется на стадии I-шейдера. Если внутри AABB не найдено пересечение луча с поверхностью поля высот, то такой AABB мы исключаем из дальнейшей обработки. Далее специальный аппаратный блок RT-конвейера сравнивает между собой точки пересечения у всех отобранных AABB-кандидатов и выбирает из них ближайшую к наблюдателю. Для этой точки вызывается стадия CH-шейдера, на которой мы вычисляем цвет луча в этой точке. Ситуация, когда вдоль луча не найдено ни одного AABB-кандидата, считается *промахом*, и для такого луча вызывается стадия M-шейдера, на которой лучу присваивается цвет фона. Полученный цвет луча (из CH-шейдера или M-шейдера) возвращается на стадию RG-шейдера, где мы записываем его в соответствующий пиксел выходного изображения.

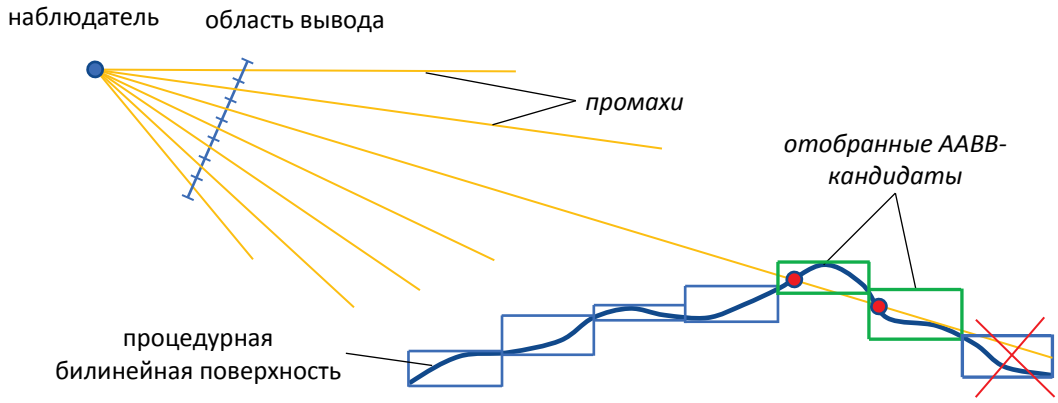

Рисунок 2 – Трассировка первичных лучей в предлагаемой технологии

Описанная технология реализуется в два этапа. На ***первом этапе*** (предварительная обработка) выполняется построение AABB-массива и его добавление в BVH-дерево сцены. На ***втором этапе*** (визуализация) выполняется построение изображения на RT-конвейере: генерация лучей, отбор AABB-кандидатов, вычисление цветов лучей и их запись в выходное изображение. Рассмотрим ключевые моменты реализации описанных этапов.

3.1. Метод построения AABB-массива

Одним из важных аспектов качественного рендеринга процедурной билинейной поверхности поля высот является то, чтобы высоты ее опорных точек в мировой системе координат (*WCS-системе*) соответствовали высотам, хранящимся в текселах карты высот (*истинным высотам*). В текстурной системе координат карты высот (*ST-системе*) значения высоты будут истинными в точках, соответствующих центрам текселов (согласно формуле билинейной интерполяции). Исходя из этого, определим «строительные» элементы, которые нам понадобятся для создания

AABB-массива. *Ячейкой* карты высот мы будем называть квадрат в ST-системе, вершинами которого являются центры 4-х соседних текселов (см. рисунок 3a). Каждой ячейке соответствует свой участок билинейной поверхности (*BL-патч*) в WCS-системе. Совокупность всех BL-патчей образует непрерывную билинейную поверхность поля высот (*BL-поверхность*).

В данной работе каждый AABB ограничивает набор BL-патчей, соответствующий *блоку ячеек* размером $2^k \times 2^k$ (см. рисунок 3b). Пусть для простоты карта высот имеет размеры $(m+1)\times(n+1)$ текселов, где m и n кратны размерам блока ячеек. Тогда для охвата всей BL-поверхности нам понадобятся $(m/2^k)\times(n/2^k)$ AABB. Обозначим через V 2D-массив координат опорных точек BL-поверхности (в WCS-системе), а через A – 2D-массив AABB (каждый AABB задается парой диагональных вершин). Алгоритм заполнения массива A состоит из следующих шагов

1) Для каждого (i_{tex}, j_{tex})-го элемента массива V:

- вычислим координаты (s,t) центра (i_{tex}, j_{tex})-го тексела карты высот (в ST-системе);
- преобразуем (s,t)-координаты в (x,y,z)-координаты опорной точки BL-поверхности;
- $V[i_{tex}][j_{tex}] = (x,y,z)$.

2) Для каждого (i_b, j_b)-го элемента массива A:

- просмотрим в массиве V координаты опорных точек, входящих в (i_b, j_b)-ый блок ячеек, и выберем из них наименьшие $(x_{\min}, y_{\min}, z_{\min})$ и наибольшие $(x_{\max}, y_{\max}, z_{\max})$ значения;
- $A[i_b][j_b] = \{(x_{\min}, y_{\min}, z_{\min}), (x_{\max}, y_{\max}, z_{\max})\}$.

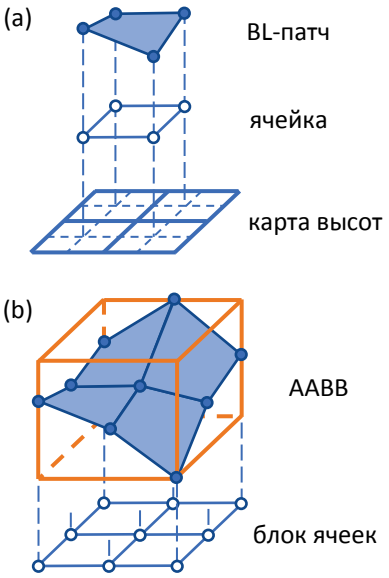

Рисунок 3 – Связь карты высот, ячейки и BL-патча (a), и построение AABB из блока ячеек (b)

В предложенном алгоритме при построении AABB используются единожды посчитанные координаты опорных точек, что позволяет избежать образования щелей (промахов лучей) между соседними AABB в процессе рендеринга BL-поверхности. Полученный AABB-массив мы загружаем в видеопамять и переводим в форматы ускоряющих структур нижнего уровня (Bottom-Level Acceleration Structure, BLAS) и верхнего уровня (Top-Level Acceleration Structure, TLAS) [25] – компоненты BVH-дерева, с которыми непосредственно работает RT-конвейер. Отметим, что согласно рекомендации [26] мы объединяем все построенные AABB в одну BLAS-структуру для повышения эффективности работы аппаратного блока обхода BVH-дерева.

3.2. Метод отбора AABB-кандидата

На этапе построения изображения поля высот на RT-конвейере ключевым моментом является отбор AABB-кандидата – AABB, внутри которого луч пересекает BL-поверхность поля высот. Процедура отбора выполняется на стадии I-шейдера сразу после того, как аппаратный блок обхода BVH-дерева находит AABB, пересекаемый сгенерированным лучом (см. рисунок 1). К сожалению, блок обхода не предоставляет данные о точках пересечения луча с AABB, а определяет лишь наличие такого пересечения. Единственной доступной информацией является индекс пересекаемого AABB (в AABB-массиве), передаваемый через встроенную переменную *gl_PrimitiveID*, и данные о луче – в RT-конвейере координаты любой точки P_{ray} на луче задаются в параметрическом виде $P_{ray} = P_{gen} + t\boldsymbol{r}$, где P_{gen} - координаты (в системе WCS) точки, из которой испускается луч (для первичных лучей это положение наблюдателя), $\boldsymbol{r}$ - единичный вектор направления луча, а t - неотрицательный параметр определяющий положение точки на луче. Используя эту информацию, а также данные из AABB-массива и карты высот, мы выполняем процедуру отбора AABB-кандидата в два шага. На ***первом шаге*** мы вычисляем параметры t_{in} и

t_{out} точек входа и выхода луча из AABB, а на ***втором шаге*** находим на отрезке $[t_{in}, t_{out}]$ значение параметра $t_{closest}$ ближайшего пересечения луча с BL-патчем. Рассмотрим эти шаги подробнее.

3.2.1. Вычисление параметров t_{in} и t_{out} точек входа и выхода луча

Данный шаг реализуется с помощью расширенной версии Slabs-теста [27], основанного на вычислении вдоль луча трех интервалов расстояний до пар плоскостей, определяющих грани AABB. Если луч пересекает AABB, то эти интервалы перекрываются, а точками входа и выхода луча будут границы отрезка, образуемого перекрытием (см. рисунок 4).

В референсной реализации Slabs-теста [28] полагается, что наблюдатель находится вне AABB, и возвращается только параметр t_{in} точки входа луча. Наша расширенная версия возвращает оба параметра t_{in} и t_{out}, а также учитывает случай нахождения наблюдателя внутри AABB (при этом точкой входа в AABB считается положение наблюдателя, т.е. $t_{in} = 0$). Такая ситуация может возникнуть, например, когда наблюдатель располагается на поверхности ландшафта или пролетает около какой-либо возвышенности. Отдельно отметим случай, когда $|t_{in} - t_{out}| \le \varepsilon$, где ε - погрешность машинного представления действительных чисел, т.е. луч пересекает AABB в одной точке. Это случается редко, однако без должной обработки может приводить к точечным артефактам на изображении. В таких случаях мы досрочно завершаем процедуру отбора AABB-кандидата, предварительно проверяя, не совпадает ли высота h_{in} точки входа в AABB с BL-поверхностью поля высот. Для этого мы вычисляем координаты точки входа в ST-системе карты высот и по ним извлекаем из карты высот (с помощью билинейной фильтрации) значение h_{map} высоты BL-поверхности. Если $|h_{in} - h_{map}| \le \varepsilon$, то мы возвращаем $t_{closest} = t_{in}$, в противном случае - $t_{closest} = -1$.

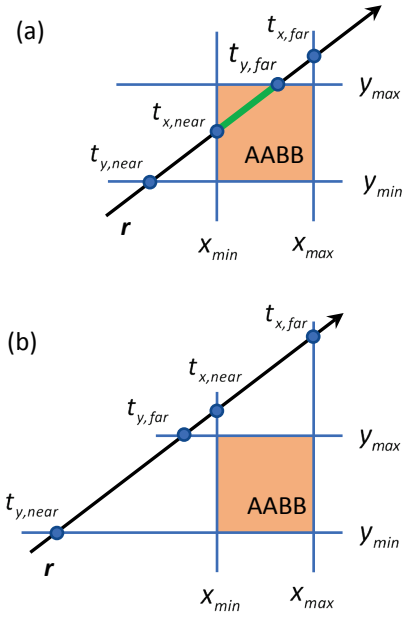

Рисунок 4 – Пример Slabs-теста для 2D-случая: (a) попадание луча ***r*** в AABB (интервалы $[t_{x,near}, t_{x,far}]$ и $[t_{y,near}, t_{y,far}]$ перекрываются), $t_{in} = t_{x,near}$, $t_{out} = t_{y,far}$; (b) промах луча ***r*** (интервалы не перекрываются)

3.2.2. Нахождение параметра $t_{closest}$ ближайшего пересечения

Данный шаг реализуется следующим образом. Мы продвигаемся вдоль луча от точки t_{in} к точке t_{out}, проверяя пересечения луча с AABB BL-патчей, лежащих на его трассе (см. рисунок 5). Если луч пересекает AABB BL-патча, то мы вычисляем параметр $t_{closest}$ пересечения луча с этим BL-патчем. Если $t_{closest} > 0$ (BL-патч пересечен), то мы возвращаем его и завершаем процедуру отбора AABB-кандидата. В противном случае, выполняется проверка AABB следующего по лучу BL-патча, и так далее, пока не будет обработан крайний AABB. Если, в конечном счете, BL-патч так и не был пересечен, то мы возвращаем значение $t_{closest} = -1$.

В описанном методе нахождения параметра $t_{closest}$ координаты вершин AABB BL-патчей мы вычисляем, используя данные из карты высот и текстурные координаты вершин соответствующих ячеек (см. рисунок 3). Проверка пересечения луча с AABB BL-патча выполняется с помощью описанного в разделе 3.2.1 расширенного Slabs-теста, а вычисление значения параметра $t_{closest}$ пересечения луча с BL-патчем - с помощью адаптированной для нашей задачи реализации подхода GARP [23]. В частности, в референсной реализации [23] для проверки валидности вычисленного значения параметра $t_{closest}$ используется встроенная функция

rtPotentialIntersection, входящая в API NVidia OptiX, которая отсутствует в API Vulkan. В нашей адаптированной реализации вместо этой функции используется проверка $t'_{in} < t_{closest} < t'_{out}$, где t'_{in}, t'_{out} - параметры точек входа и выхода луча из AABB BL-патча, вычисленные с помощью расширенного Slabs-теста. Кроме этого, в нашем варианте получаемые на выходе текстурные координаты точки пересечения преобразуются из локальной ST-системы BL-патча в глобальную ST-систему карты высот, что позволяет реализовать эффективную визуализацию сверхбольших текстур [29].

Важной частью описанного метода является реализация перехода к AABB следующего по лучу BL-патча, которая сводится к вычислению номеров строки и столбца следующей (по трассе луча) ячейки карты высот (см. рисунок 5). Для решения этой задачи используется разработанный алгоритм устойчивого продвижения вдоль трассы луча [8]. Его суть в следующем. Чтобы перейти из точки P_{start} ячейки с номерами (i_{start}, j_{start}) к следующей по лучу $\boldsymbol{u}$ ячейке, мы определяем, какой *внутренний угол* ячейки пересекает луч $\boldsymbol{u}$, и вычисляем параметры t_S и t_T пересечений с прямыми s_{corner}, t_{corner}, образующими этот угол (см. рисунок 6). Номер n_{corner} внутреннего угла вычисляется по формуле

$$n_{corner} = b_0 + 2b_1,$$

где флаг $b_0 = (|u_s| \geq 0)$, $b_1 = (|u_t| \geq 0)$, а u_s, u_t - координаты вектора $\boldsymbol{u}$ по осям S и T. Из точки P_{start} мы переходим в точку

$$P_{next} = P_{start} + \min(t_S, t_T)\boldsymbol{u}$$

и ячейку с номерами

$$i_{next} = i_{start} + sign(u_t)b_0, \quad j_{next} = j_{start} + sign(u_s)b_1,$$

где $b_0 = (t_T \leq t_S)$, $b_1 = (t_S \leq t_T)$, а $sign(x)$ - встроенная функция, которая возвращает 1 при $x > 0$, 0 при x равном 0, и -1 при $x < 0$. Отметим, что случаи нулевых значений u_s и u_t обрабатываются отдельно. В описанном способе, благодаря использованию внутреннего угла, продвижение вдоль луча будет осуществляться, даже, если вычисленная точка P_{next} окажется вне (i_{next}, j_{next})-ой ячейки (из-за погрешности машинного представления действительных чисел).

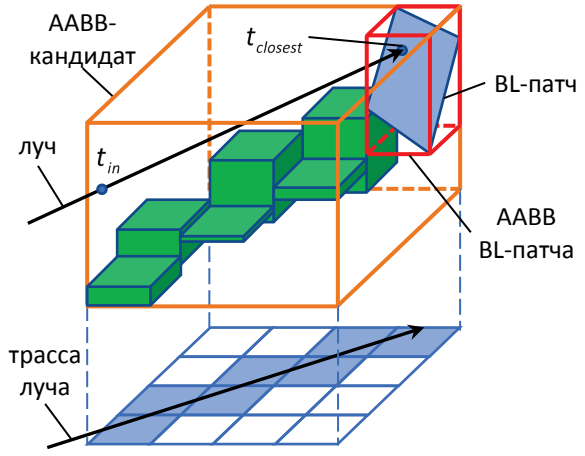

Рисунок 5 – Поиск ближайшего пересечения луча с BL-патчем внутри AABB-кандидата

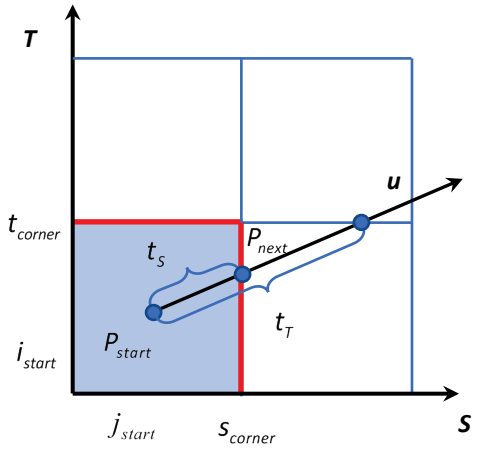

Рисунок 6 – Схема перехода к следующей по лучу $\boldsymbol{u}$ ячейке

После завершения описанной процедуры отбора AABB-кандидата мы выполняем проверку вычисленного значения $t_{closest}$. Если $t_{closest} > 0$, то мы вызываем встроенную функцию *reportIntersectionEXT*($t_{closest}$, *0)*, которая сообщает RT-конвейеру о наличии пересечения луча с нашим процедурным примитивом (полем высот). В этом случае обработка AABB-кандидата на RT-конвейере продолжается, как описано в начале раздела 3. В противном случае функция *reportIntersectionEXT* не вызывается, что приводит к автоматическому исключению данного AABB-кандидата из дальнейшей обработки на RT-конвейере (см. рисунок 2).

4. Результаты

На основе предложенной технологии был разработан программный комплекс - трассировщик первичных лучей (рейкастер) детализированных полей высот. Рейкастер работает полностью на RT-конвейере через API Vulkan v. 1.3.204.1 (драйвер NVidia DCH 512.59). Для сравнения скорости и качества рендеринга в рейкастере также были реализованы два метода трассировки полей высот: с равномерной выборкой вдоль луча (далее РВ-метод) [11] и с классическим аналитическим расчетом пересечения «луч-билинейный патч» (далее КА-метод) [24], используемом в [16]. Все три реализации были протестированы на референсных картах высот Пьюджет-Саунд (1K×1K, 2K×2K и 4K×4K) и Великий Каньон (4K×2K) [30], а также на высокодетализированной карте высот лунного ударного кратера Аристарх (10K×4K, см. рисунок 7) [31].

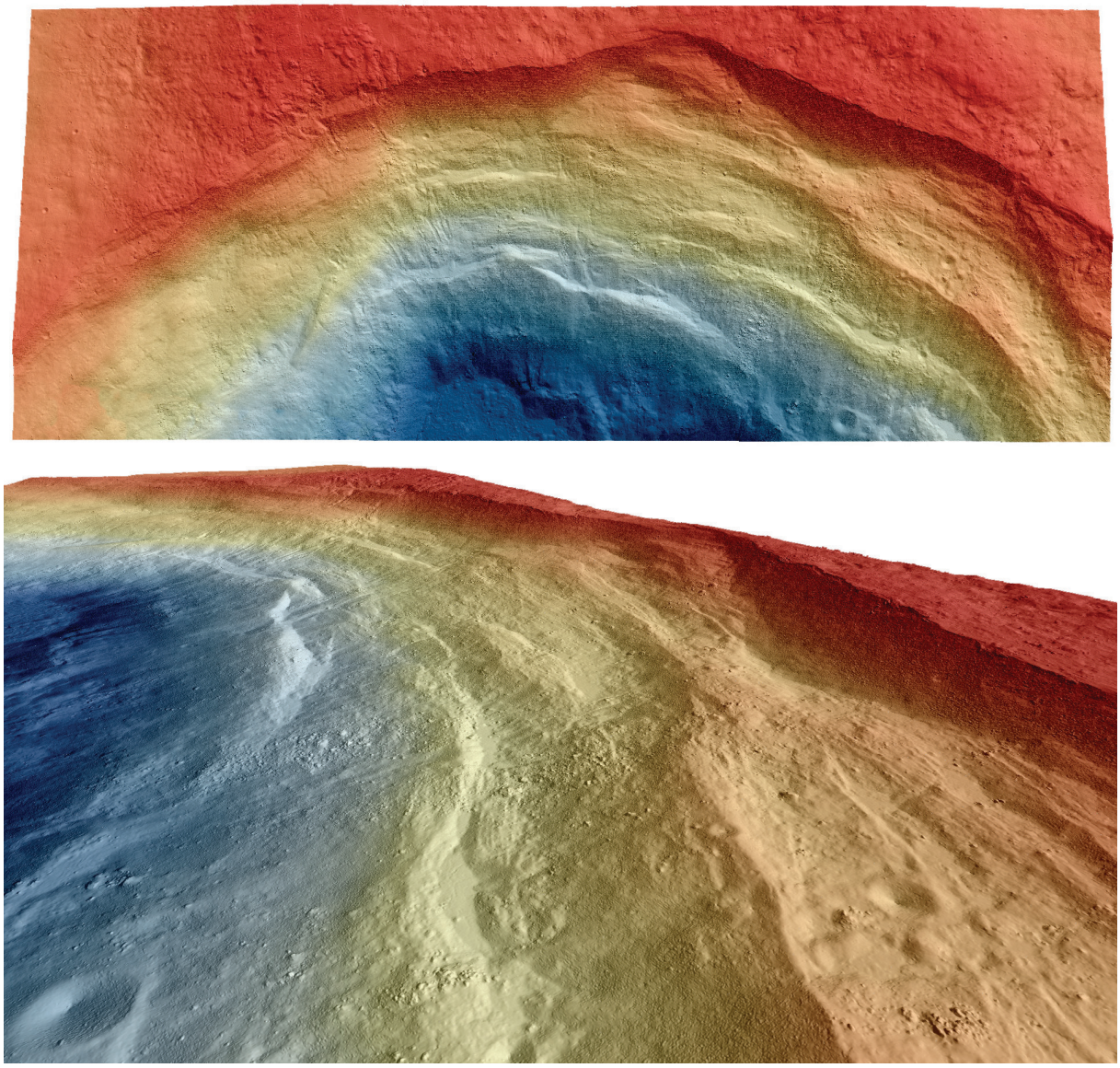

Рисунок 7 – Поле высот лунного ударного кратера Аристарх (рендеринг выполнен с помощью нашей технологии): (сверху) общий вид кратера; (внизу) мелкие детали внутри кратера

Все тесты запускались на персональном компьютере, оборудованным процессором Intel Core i7-6800K 3.40 ГГц, оперативной памятью 16 Гб DDR4 и графической картой NVidia GeForce RTX 2080 (выделенная видеопамять 8 Гб GDDR6, 46 RT-ядер и 2944 ядер CUDA). Тесты

производительности проводились при разрешении области вывода 1920×1080 в условиях наблюдения, как показано на рисунке 8.

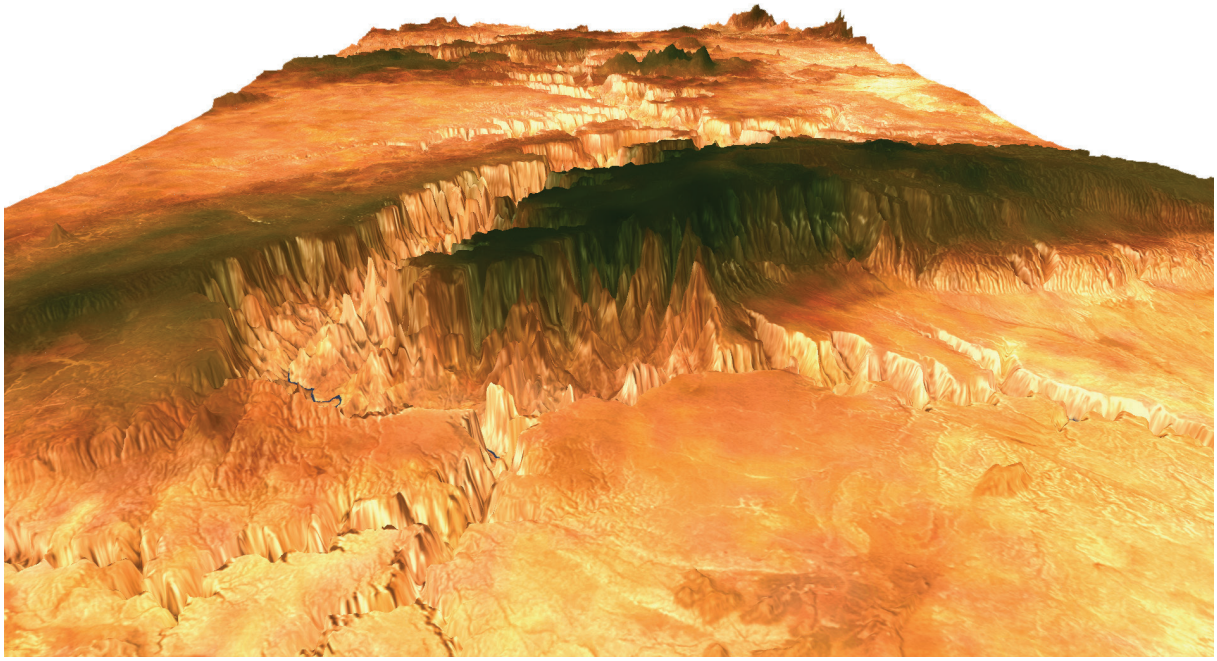

Рисунок 8 – Поле высот Великого Каньона (рендеринг выполнен с помощью нашей технологии)

Для каждой из трех реализаций мы провели серии экспериментов, в которых изменяли размер блока ячеек от 1×1 до 1024×1024. В таблице 1 приведены размеры блоков (в ячейках), при которых показатели производительности были близки к наибольшим. Эти размеры были использованы для сравнения скоростей работы трех тестируемых реализаций (см. таблицу 2).

Таблица 1 – Размеры блоков ячеек

Карта высот		Наша технология	PB-метод	KA-метод
Пьюджет-Саунд	1K×1K	32	64	2
	2K×2K	32	64	2
	4K×4K	16	32	2
Великий Каньон	4K×2K	16	16	2
Аристарх	10K×4K	16	32	4

Таблица 2 – Средние скорости визуализации (в кадрах в секунду)

Карта высот		Наша технология	PB-метод	KA-метод
Пьюджет-Саунд	1K×1K	590	422	48
	2K×2K	393	280	31
	4K×4K	270	186	26
Великий Каньон	4K×2K	497	384	55
Аристарх	10K×4K	415	366	55

В отличие от других сравниваемых методов, качество и скорость синтеза изображений, получаемых с помощью PB-метода, зависят от числа выборок на ячейку вдоль луча (по сути, от частоты дискретизации функции высоты). На рисунке 9a показаны артефакты в виде градаций цвета и неровности кромок горного пика при 4-х выборках на ячейку. При увеличении числа выборок до 16-ти (это значение использовалось для замера скорости PB-метода в таблице 2) качество изображения становится лучше (см. рис. 9b), но все равно уступает классическому KA-методу (см. рис. 9c). При этом из таблицы 2 видно, что KA-метод имеет крайне высокие

вычислительные затраты, даже при реализации на конвейере трассировки лучей. Из всех трех сравниваемых решений наша технология показала наиболее высокие показатели производительности и обеспечила рендеринг полей высот (см. рисунок 9d), по качеству не уступающий классическому КА-методу.

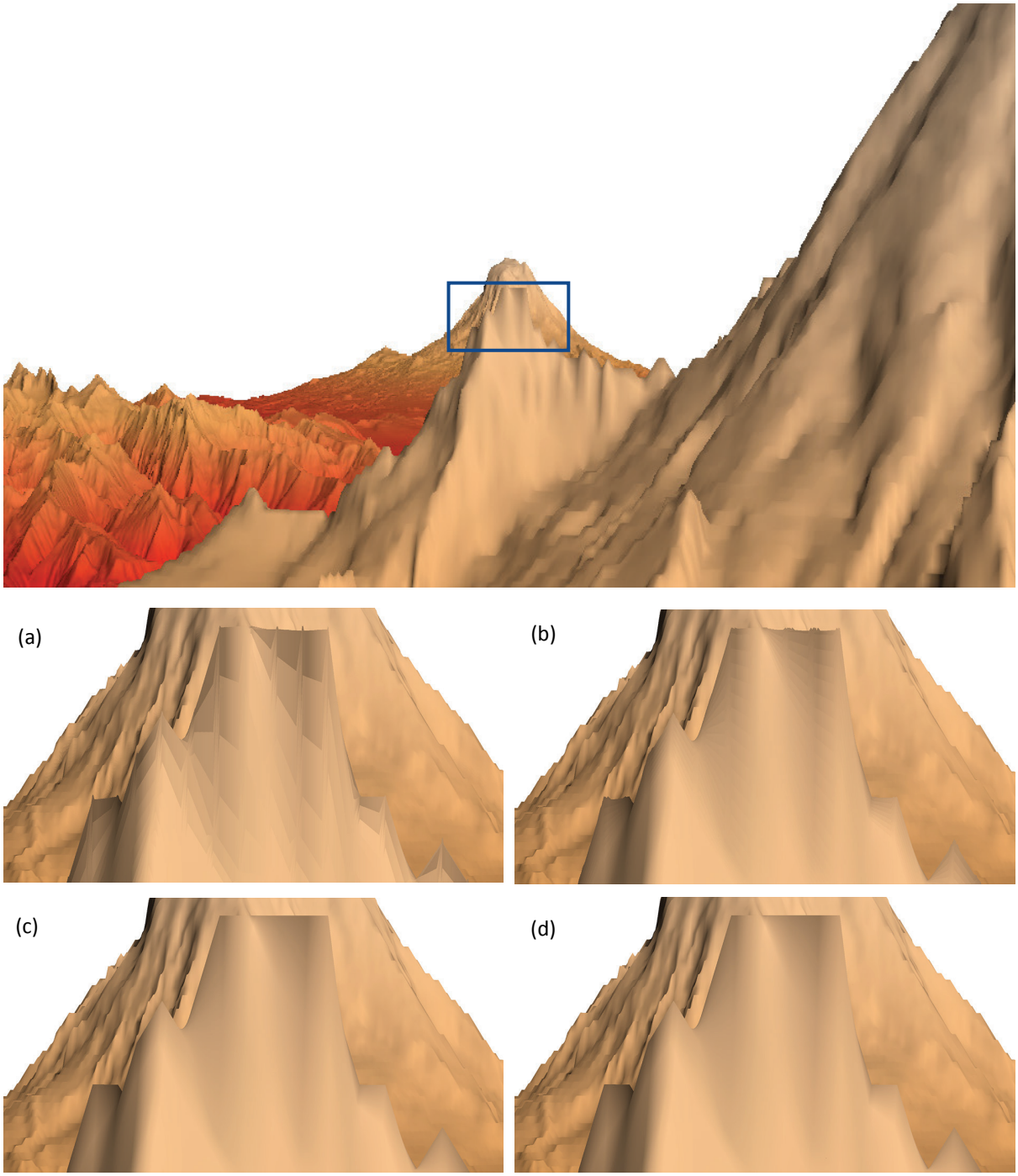

Рисунок 9 – Поле высот Пьюджет Саунд (4K×4K), рендеринг увеличенной области: (a) РВ-метод (4 выборки на ячейку); (b) РВ-метод (16 выборок на ячейку); (c) КА-метод; (d) наша технология

5. Заключение

В данной работе предложена эффективная технология рендеринга на GPU в реальном времени детализированных полей высот, основанная на новой архитектуре RTX аппаратного ускорения трассировки лучей. Технология работает на RT-ядрах - вычислительных ядрах GPU

нового типа, доступ к которым реализуется через RT-конвейер - программируемый конвейер трассировки лучей. В исследовании предложен метод построения ускоряющей структуры данных в формате, понятном RT-конвейеру, основанный на представлении поверхности поля высот в виде 2D-массива ограничивающих параллелепипедов (AABB). Для этого был введен ключевой «строительный» элемент - блок ячеек карты высот, охватываемый AABB. Кроме этого, в методе описан ряд решений, позволяющих избежать образования щелей (промахов лучей) между смежными AABB в процессе рендеринга поля высот.

Ядром предложенной технологии рендеринга является разработанная система отбора AABB-кандидатов на обладание ближайшим к наблюдателю пересечением «луч-поверхность поля высот». Ключевым преимуществом разработанной системы является то, что отбор AABB-кандидатов вдоль трассируемых лучей выполняется параллельно и независимо друг от друга на RT-ядрах GPU. В нашей системе реализована оригинальная связка расширенного Slabs-теста [27] и адаптированного алгоритма GARP [23], которая показала лучшие результаты по скорости и качеству визуализации в нашем сравнительном тестировании.

Полученные результаты подтвердили высокую эффективность разработанной технологии и ее применимость для построения систем виртуального окружения, научной визуализации, видеотренажеров и др. В качестве дальнейшей работы планируется развить технологию для рендеринга 3D-особенностей ландшафта (пещер, туннелей и т.п.).

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. 0580-2022-0012”.

7. Список источников

[1] NVIDIA Turing GPU Architecture Whitepaper // NVIDIA Corporation. 2018. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf (дата обращения 31.05.2022).

[2] Light Transport in Realistic Rendering: State-of-the-Art Simulation Methods / V.A. Frolov, A.G. Voloboy, S.V. Ershov, V.A. Galaktionov // Programming and Computer Software. 2021. Vol. 47. No. 4. P. 298–326.

[3] Examination of the Nvidia RTX / V.V. Sanzharov, A.I. Gorbonosov, V.A. Frolov, A.G. Voloboy // GraphiCon 2019: proceedings of the 29th International Conference on Computer Graphics and Vision / CEUR Workshop Proceedings, 2019. Vol. 2485. P. 7–12.

[4] Mikhaylyuk M.V., Timokhin P.Y., Maltsev A.V. A Method of Earth Terrain Tessellation on the GPU for Space Simulators // Programming and Computer Software. 2017. Vol. 43. No. 4. P. 243–249.

[5] Santos P., Toledo de R., Gattass M. Solid Height-map Sets: modeling and visualization // SPM ‘08: Proceedings of the 2008 ACM symposium on Solid and physical modeling, 2008. P. 359–365.

[6] Gilardi M., Watten P. L., Newbury P. Drift-Diffusion Based Real-Time Dynamic Terrain Deformation // Eurographics Proceedings, 2016.

[7] Lee E.-S., Shin B.-S. Hardware-Based Adaptive Terrain Mesh Using Temporal Coherence for Real-Time Landscape Visualization // Sustainability. 2019. Vol. 11, No. 7. P. 1–18.

[8] Timokhin P., Mikhaylyuk M. Reliable GPU-Based Methods and Algorithms of Implementation Dynamic Relief Shadows in Virtual Environment Systems // GraphiCon 2021 (Nizhny Novgorod, Russia, September 27-30, 2021): proceedings of the 31th International Conference on Computer Graphics and Vision / CEUR Workshop Proceedings, 2021. Vol. 3027. P. 83–94.

[9] Interactive Ray Tracing for Isosurface Rendering / S. Parker, P. Shirley, Y. Livnat, C. Hansen, P.-P. Sloan // VIZ’98: proceedings of the IEEE Visualization 98, 1998. P. 233–238.

[10] Amanatides J., Woo A. A Fast Voxel Traversal Algorithm for Ray Tracing // Eurographics ’87: proceedings of the 8th European Computer Graphics Conference and Exhibition, Amsterdam, 1987. P. 3–10.

[11] Aslandere T., Flatken M., Gerndt A. A Real-Time Physically Based Algorithm for Hard Shadows on Dynamic Height-Fields // Proceedings of 12. Workshop der GI-Fachgruppe on Virtuelle und Erweiterte Realität, Aachen Verlag, Bonn, 2015. P. 101–112.
[12] Policarpo F., Oliveira M.M., Comba J.L.D. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces // I3D ‘05: proceedings of the 2005 Symposium on Interactive 3D Graphics and Games, 2005. P. 155–162.
[13] Ammann L., Génevaux O., Dischler J.-M. Hybrid Rendering of Dynamic Heightfields using Ray-Casting and Mesh Rasterization // GI ’10: proceedings of Graphics Interface Conference 2010, Canadian Information Processing Society, CAN, 2010. P. 161–168.
[14] Policarpo F., Oliveira M.M. Relaxed Cone Stepping for Relief Mapping // GPU Gems 3, Addison-Wesley Professional. 2007. P. 409–428.
[15] Baboud L., Eisemann E., Seidel H.-P. Precomputed Safety Shapes for Efficient and Accurate Height-Field Rendering // IEEE Transactions on Visualization and Computer Graphics. 2012. Vol. 18, No. 11. P. 1811–1823.
[16] Tevs A., Ihrke I., Seidel H.-P. Maximum Mipmaps for Fast, Accurate, and Scalable Dynamic Height Field Rendering // I3D ‘08: proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, 2008, P. 183–190.
[17] Dick C., Krüger J.H., Westermann R. GPU Ray-Casting for Scalable Terrain Rendering // Eurographics ’09, 2009. P. 43–50.
[18] Silvestre A., Pereira J., Costa V. A Real-Time Terrain Ray-Tracing Engine // 2018 International Conference on Graphics and Interaction (ICGI), 2018. P. 1–8.
[19] A Flexible Architecture for Ray Tracing Terrain Heightfields / S. Dübel, L. Middendorf, C. Haubelt, H. Schumann // Proceedings of the International Summerschool on Visual Computing, Rostock, Germany, 2015. P. 3–22.
[20] An Auto-Programming Approach to Vulkan / V. Frolov, V. Sanzharov, V. Galaktionov, A. Scherbakov // GraphiCon 2021 (Nizhny Novgorod, Russia, September 27-30, 2021): proceedings of the 31st International Conference on Computer Graphics and Vision / CEUR Workshop Proceedings, 2021. Vol. 3027. P. 150–165.
[21] Brüll F. Fast Transparency and Billboard Ray Tracing with RTX Hardware: master thesis / Clausthal University of Technology. 2020. 81 p. DOI: 10.13140/RG.2.2.14692.19842.
[22] Proceduray - A light-weight engine for procedural primitive ray tracing / V. Silva, T. Novello, H. Lopes, L. Velho // arXiv preprint arXiv:2012.10357, 2020. P. 1–29. URL: https://arxiv.org/pdf/2012.10357.pdf (дата обращения 31.05.2022).
[23] Reshetov A. Cool Patches: A Geometric Approach to Ray/Bilinear Patch Intersections // Ray Tracing Gems. 2019. P. 95–109.
[24] Ramsey S.D., Potter K., Hansen C. Ray Bilinear Patch Intersections // Journal of Graphics Tools. 2004. Vol. 9. No 3. P. 41–47.
[25] Vulkan 1.3 Specification // The Khronos Vulkan Working Group. 2022. URL: https://www.khronos.org/registry/vulkan/specs/1.3-extensions/pdf/vkspec.pdf (дата обращения 31.05.2022).
[26] 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 31.05.2022).
[27] A Ray-Box Intersection Algorithm and Efficient Dynamic Voxel Rendering / A. Majercik, C. Crassin, P. Shirley, M. McGuire // Journal of Computer Graphics Techniques. 2018. Vol. 7. No. 3. P. 66–82.
[28] NVIDIA Vulkan Ray Tracing Tutorials. URL: https://github.com/nvpro-samples/vk_raytracing_tutorial_NV/tree/master/ray_tracing_intersection/shaders/raytrace.rint (дата обращения 31.05.2022).
[29] Тимохин П.Ю., Михайлюк М.В. Сверхбольшие текстуры для высоко реалистичной визуализации виртуальных ландшафтов // Информационные технологии и вычислительные системы. 2013. №3. С. 46–54.
[30] Large Geometric Models Archive (Puget Sound, Grand Canyon), Georgia Institute of Technology, URL: https://www.cc.gatech.edu/projects/large_models/ (дата обращения 31.05.2022).
[31] Lunar Reconnaissance Orbiter Camera. Aristarchus Crater West Rim DTM. 2020. URL: https://wms.lroc.asu.edu/lroc/view_rdr/NAC_DTM_ARISTARCH10 (дата обращения 31.05.2022).