ПРОГРАММИРОВАНИЕ, 2023, № 3, с. 56–64

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.925.83

ЭФФЕКТИВНАЯ ТЕХНОЛОГИЯ МОДЕЛИРОВАНИЯ В РЕАЛЬНОМ ВРЕМЕНИ ПОВЕРХНОСТИ ПОЛЯ ВЫСОТ НА КОНВЕЙЕРЕ ТРАССИРОВКИ ЛУЧЕЙ

© 2023 г. П. Ю. Тимохин[a,*] (ORCID: 0000-0002-0718-1436),
М. В. Михайлюк[a,] (ORCID: 0000-0002-7793-080X)**

[a]ФГУ "ФНЦ Научно-исследовательский институт системных исследований РАН",
Россия, 117218, Москва, Нахимовский просп., 36, к. 1

**E-mail: webpismo@yahoo.de*

***E-mail: mix@niisi.ras.ru*

Поступила в редакцию 09.01.2023 г.
После доработки 16.01.2023 г.
Принята к публикации 20.01.2023 г.

В данной статье, на примере поверхности поля высот, предлагается эффективная технология моделирования в реальном времени сложных процедурных объектов на конвейере трассировки лучей (RT-конвейере). Предлагаемая технология не перегружает стадию I-шейдера (шейдера пересечения), а распределяет вычислительную нагрузку между I-шейдером и AH-шейдером (шейдером любого подтвержденного пересечения). Ключевыми нововведениями в технологии являются ранняя отбраковка на стадии I-шейдера ограничивающих параллелепипедов (AABB), отобранных аппаратным блоком RT-конвейера, и концепция "прозрачного AABB", позволяющая перенести затратное вычисление пересечения луча с процедурным объектом на более позднюю стадию AH-шейдера. Также в работе описан ряд модификаций, сокращающих объем таких вычислений. Предложенная технология была реализована в программном комплексе на языках C++, GLSL и с помощью API Vulkan. Была исследована производительность разработанного решения при различных условиях трассировки лучей на задаче моделирования поверхности детализированного поля высот Пьюджет-Саунд. Полученные результаты подтвердили эффективность разработанной технологии и возможность ее применения в системах виртуального окружения, видеотренажерных комплексах, научной визуализации и др.

DOI: 10.31857/S0132347423030068, **EDN:** DENOFM

1. ВВЕДЕНИЕ

Аппаратное ускорение трассировки лучей, появившееся в серийных видеокартах NVidia, открыло новую эру в области высокореалистичной компьютерной графики реального времени. В настоящее время выпускается уже третье поколение графических процессоров с гибридной параллельной архитектурой RTX [1], которая включает в себя наряду с универсальными вычислительными ядрами (CUDA-ядрами) специализированные ядра нового типа – *ядра трассировки лучей* (RT-ядра). В отличие от CUDA-ядер, RT-ядра предназначены для эффективного решения узкого круга задач, обеспечивающих трассировку лучей (генерация лучей, расчет пересечения луча с треугольником и др.). Недавние исследования [2–5] показывают, что, несмотря на относительно небольшое количество, RT-ядра обладают высоким потенциалом для ускорения трассировки лучей в сложных трехмерных виртуальных сценах.

Одним из актуальных направлений применения аппаратно-ускоренной трассировки лучей является моделирование и визуализация в масштабе реального времени (со скоростью не менее 25 кадров в секунду) гладких поверхностей, основанных на *процедурных примитивах*. Такие примитивы отличаются от традиционных графических примитивов тем, что их геометрия не задается в явном виде, а синтезируется из точек пересечений с лучами, вычисляемых с помощью пользовательских процедур. Простым примером является сфера, пересечение луча с которой определяется аналитически. Сложнее дело обстоит с расчетом пересечений лучей с поверхностями, которые описаны с помощью сеточных функций, в частности, с *полями высот* [6]. В данной работе предлагается эффективная технология решения этой задачи на RT-ядрах в реальном времени для детализированных сеток высот (4K × 4K и выше), востребованных в современных системах виртуального окружения и симуляторах [7, 8]. Для про-

граммной реализации решения были использованы язык C++, шейдерный язык GLSL и API Vulkan.

2. ПРЕДЫДУЩИЕ ИССЛЕДОВАНИЯ

К настоящему времени опубликовано большое количество работ, предлагающих различные методы моделирования поверхностей на основе сеток высот. Так или иначе, результатом таких исследований является поверхность, в которой промежуточная информация между узлами сетки высот восполняется с помощью интерполяции. Глобально методы построения интерполированных поверхностей полей высот развиваются по двум основным направлениям: полигональное моделирование и трассировка лучей.

Первое направление основано на соединении узлов сетки высот с помощью треугольных графических примитивов (*полигонов*), в результате которого формируется триангулированная (полигональная) модель поверхности поля высот. В работе [9] можно найти хороший обзор методов и алгоритмов полигонального моделирования полей высот. Преимуществом данного подхода является высокоразвитая программно-аппаратная поддержка распараллеливания обработки треугольников на графическом конвейере GPU, обеспеченная вычислительной мощностью тысяч CUDA-ядер. Ключевым ограничивающим фактором является число треугольников в формируемой модели поверхности: чем выше частота высотных данных в сетке высот, тем больше треугольников необходимо для построения ее поверхности и тем ниже скорость визуализации такой модели. В случае высокочастотных (и, как следствие, детализированных) сеток высот это приводит к необходимости разработки сложных адаптивных техник управления уровнем детализации [10]. Другой важной проблемой является интеграция таких адаптивных моделей в системы виртуального окружения, в которых требуется моделирование реалистичной светотеневой обстановки [11].

В данной статье исследуется ***второе направление***, при котором узлы сетки высот соединяются с помощью процедурных примитивов (*интерполянтов* [12]), а поверхность поля высот моделируется на основе поиска пересечений этих примитивов с лучами, испущенными из положения наблюдателя (обратная трассировка лучей [13]). Такой подход позволяет синтезировать высококачественные изображения, на которых поверхность поля высот интерполирована с попиксельной точностью. Главным недостатком является высокая вычислительная сложность, обусловленная необходимостью поиска точек, в которых лучи пересекают процедурные примитивы [14]. Для решения этой задачи были предложены различные пути: аппроксимация пересечения “луч-примитив” [15–18]; разработка ускоряющих структур данных [19–22], в том числе комбинированных с растеризацией [23]; распараллеливание лучей на CUDA-ядрах [24–26] и др. Общим ограничением предложенных решений является упорядоченный характер обработки пересечений вдоль луча, будь то бинарный поиск или проход по квадродереву. Это приводит к тому, что при появлении отдельных лучей с большим числом пересечений тормозится просчет всего изображения, даже с учетом распараллеливания обработки таких лучей на CUDA-ядрах.

С приходом новой архитектуры RTX появилась возможность распараллеливания трассировки лучей на выделенных ядрах GPU, RT-ядрах, экономя вычислительный потенциал универсальных CUDA-ядер. Аппаратно-ускоренный функционал, зашитый в RT-ядрах, позволяет существенно сократить время просчета лучей и, как следствие, повысить скорость синтеза изображений. Чтобы использовать возможности новой архитектуры RTX, виртуальная сцена должна быть представлена в виде специального *дерева ограничивающих объемов* (Bounding Volume Hierarchy, BVH), к листьям которого привязаны имеющиеся в сцене примитивы (поддерживаются и треугольные, и процедурные примитивы). Взаимодействие лучей с BVH-деревом осуществляется параллельно, согласно *конвейеру трассировки лучей* (RT-конвейеру) [27], который включает в себя ряд программируемых стадий (*шейдеров*): генерация луча (Ray Generation Shader, RG-шейдер), потенциальное пересечение (Intersection Shader, I-шейдер), любое подтвержденное пересечение (Any-Hit Shader, AH-шейдер), ближайшее подтвержденное пересечение (Closest Hit Shader, CH-шейдер) и промах (Miss Shader, M-шейдер). Суть работы RT-конвейера состоит в (a) определении пересекаемых лучом листьев BVH-дерева, (b) вычислении в них точек пересечения с примитивами и (c) выборе из этих точек ближайшей к началу луча. Этапы (a) и (c) выполняются RT-конвейером аппаратно (автоматически), а реализация этапа (b) зависит от типа примитива. Для полигональных моделей (треугольных примитивов) это также выполняется аппаратно, что существенно упрощает реализацию процесса трассировки. Если же в трассировке участвует процедурный примитив (как в данной работе), то RT-конвейер автоматически определяет только пересечение луча с *ограничивающим параллелепипедом* (Axis-Aligned Bounding Box, AABB) этого примитива, а задача реализации пересечения луча с процедурным примитивом (внутри AABB) возлагается на разработчика.

В настоящее время уже имеется ряд публикаций, в которых приведены примеры реализации на RT-конвейере пересечений лучей с различны-

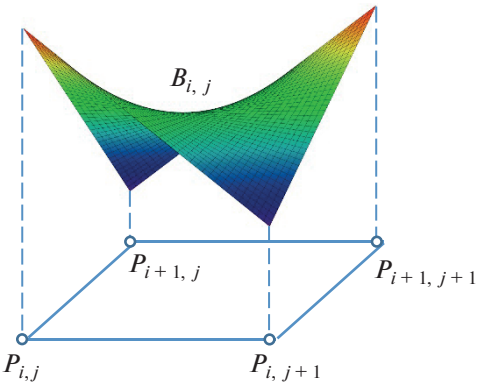

Рис. 1. Пример билинейного патча ($P_{i,j}$, ..., $P_{i+1,j+1}$ – узлы (i,j)-й ячейки сетки высот).

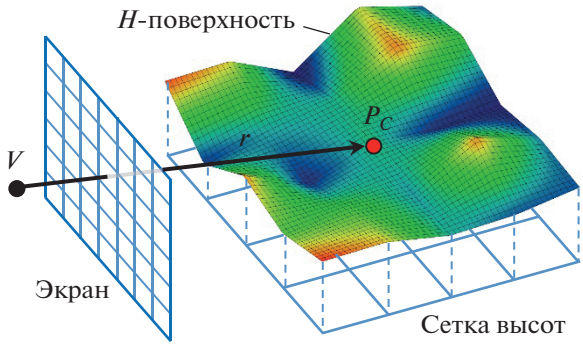

Рис. 2. Пример H-поверхности для сетки высот размера 4 × 4 ячейки.

ми процедурными примитивами [28–30]. Характерной чертой таких работ является реализация расчета пересечений на стадии I-шейдера. В данной работе, на примере задачи моделирования поверхности поля высот, предлагается модифицированная технология, при которой вычисление пересечений луча с процедурным объектом распределяется между стадиями I-шейдера и AH-шейдера. По сравнению с нашей первой реализацией (на стадии I-шейдера) [5] это позволило улучшить параллелизм решения и получить заметный выигрыш в производительности. В разделе 3 описана предлагаемая модифицированная технология, а в разделе 4 мы исследуем производительность созданного решения при различных условиях.

3. ПРЕДЛАГАЕМАЯ ТЕХНОЛОГИЯ

В данной работе мы будем рассматривать задачу синтеза с помощью трассировки лучей *кусочно-билинейной модели поверхности поля высот*. Пусть имеется сетка высот размера $m \times n$ ячеек. Обозначим через $B_{i,j}$ билинейную поверхность (патч), соответствующую (i,j)-й ячейке сетки высот (см. рис. 1), а через H – кусочно-билинейную поверхность, составленную из смежных $B_{i,j}$-х патчей (далее H-поверхность). Проведем луч **r** из позиции наблюдателя V через центр произвольного пиксела экрана. Нашей задачей будет реализация на RT-конвейере трассировки луча **r** до пересечения с H-поверхностью (см. рис. 2).

Технология трассировки луча **r** на RT-конвейере состоит из четырех этапов (см. рис. 3). На *этапе I* осуществляется генерация луча **r**, включающая задание его точки испускания и вектора направления (стадия RG-шейдера). На *этапе II* выполняется цикл обхода BVH-дерева вдоль луча **r**, целью которого является поиск ближайшей к наблюдателю точки P_C пересечения луча **r** с H-поверхностью. На *этапе III*, если точка P_C найдена, то вычисляется цвет луча в этой точке (стадия CH-шейдера), в противном случае, считается, что луч **r** “промахнулся”, и вычисляется цвет фона (стадия M-шейдера). На *этапе IV* осуществляется возврат к стадии RG-шейдера, где выполняется запись результирующего цвета луча **r** в буфер изображения. Реализации этапов I, III и IV являются во многом типовыми, их описание можно найти, например, в материалах [31]. В данной работе мы рассмотрим более подробно реализацию этапа II.

В работе [31] описан базовый подход к реализации этапа II на примере простых процедурных объектов типа сферы и параллелепипеда. В рамках данного подхода для моделируемого объекта в виртуальной сцене создается отдельный AABB, а также разрабатывается процедура вычисления внутри этого AABB ближайшей к наблюдателю точки пересечения луча с объектом. Разработанная процедура выполняется на стадии I-шейдера, после того как аппаратный блок обхода BVH-дерева обнаруживает пересечение луча с AABB (см. рис. 3). Для простых объектов, таких как сфера или параллелепипед, выполнение процедуры занимает очень короткое время, которое практически не влияет на работу RT-конвейера. В случае же H-поверхности время выполнения процедуры нахождения точки P_C будет зависеть от числа проверок потенциальных пересечений луча **r** с патчами H-поверхности (см. рис. 2). Чем выше детализация сетки высот, тем больше будет таких проверок, что приводит к перегруженности I-шейдера, торможению цикла обхода BVH-дерева (согласно спецификации [32] I-шейдер в один момент времени работает с одним лучом и одним AABB) и, как следствие, падению производительности RT-конвейера.

Для решения описанной проблемы в данной работе предлагается ряд модификаций этапа II, направленных на более эффективное использование вычислительного ресурса RT-ядер (особен-

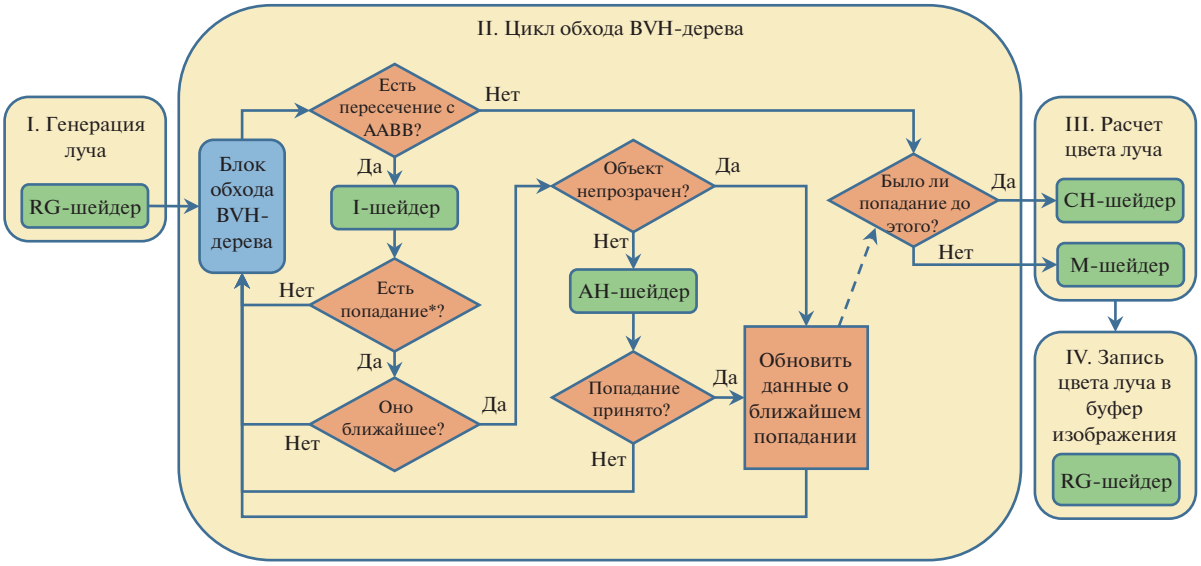

Рис. 3. Схема расширенной трассировки луча **r** на RT-конвейере (*попадание – подтвержденное пересечение).

но аппаратного блока обхода BVH-дерева) и сокращение числа проверок пересечений луча с H-поверхностью.

Во-первых, мы предлагаем создавать не один AABB для всей H-поверхности, а двумерный массив AABB, где каждый AABB охватывает свой блок $2^k \times 2^k$ патчей (см. рис. 4). Особенности построения такого массива описаны в нашей работе [5]. В процессе трассировки аппаратный блок обхода BVH-дерева проверяет все AABB, которые лежат на пути луча **r** (порядок проверки AABB произволен [32]), и определяет из них те, которые пересекаются с лучом **r**. Отобранные AABB аппаратный блок передает на стадию I-шейдера (см. рис. 3), а остальные исключает из цикла обхода BVH-дерева. Таким образом задача поиска точки P_C вдоль всего луча **r** сводится к поиску точки P_C только на участках луча **r**, проходящих через отобранные AABB, благодаря чему уменьшается число проверок потенциальных пересечений луча с патчами H-поверхности.

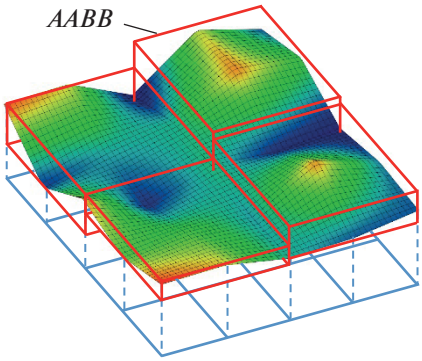

Рис. 4. Пример AABB-массива (каждый AABB охватывает блок 2 × 2 патча).

Во-вторых, для каждого AABB, отобранного аппаратным блоком, мы реализуем расширенный цикл обхода BVH-дерева (этап II на рис. 3). В отличие от базового подхода, при котором в цикле используется только стадия I-шейдера, в нашем расширенном цикле задействуются и стадия I-шейдера, и стадия AH-шейдера. Идея состоит в том, что на стадии I-шейдера мы не проводим никаких “тяжелых” расчетов, а вычисляем только точки входа и выхода луча из AABB (точнее их параметры t_{in} и t_{out} вдоль луча **r**). Это реализуется с помощью версии Slabs-теста [33, 34], модифицированной в нашей работе [5]. О вычисленном параметре t_{in} точки входа мы сообщаем RT-конвейеру в конце стадии I-шейдера с помощью оператора *reportIntersectionEXT*. Получив эту информацию, RT-конвейер автоматически сравнивает ее с вычисленным на предыдущих итерациях параметром $t_{closest}$ точки ближайшего подтвержденного пересечения (блок “Оно ближайшее?” на рис. 3), и, если $t_{in} > t_{closest}$, то RT-конвейер прерывает обработку такого AABB и переходит к следующему. Такое решение позволяет отбраковывать на раннем этапе существенную часть AABB, отобранных аппаратным блоком (см. рис. 5), не переходя непосредственно к вычислению в них точек пересечения луча **r** с блоком патчей.

В-третьих, мы предлагаем концепцию “прозрачного AABB”. Ее суть состоит в том, что изначально все AABB задаются потенциально про-

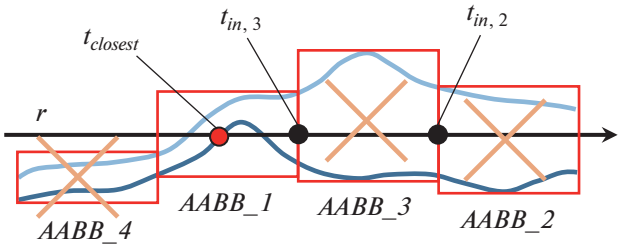

Рис. 5. Пример обработки AABB вдоль луча **r**: 1) AABB_4 не прошел отбор аппаратным блоком обхода BVH-дерева; 2) AABB_3 и AABB_2 прошли аппаратный отбор, но были отбракованы на стадии I-шейдера, так как $t_{in,3} > t_{closest}$ и $t_{in,2} > t_{closest}$; 3) AABB_1 прошел аппаратный отбор и стадию I-шейдера, и был допущен к вычислению $t_{closest}$ на стадии AH-шейдера.

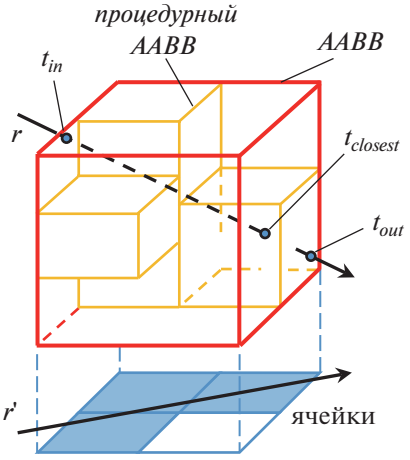

Рис. 6. Проход луча **r** через процедурные AABB внутри исходного AABB.

зрачными[1] по отношению к лучу **r**, а на стадии AH-шейдера для каждого AABB, прошедшего отбор I-шейдером, определяется, является ли он прозрачным или нет. AABB считается непрозрачным, если в нем луч ***r*** хотя бы раз пересекает блок патчей. Мы вычисляем параметр $t_{closest}$ точки ближайшего к наблюдателю пересечения и записываем значение этого параметра в атрибут луча (структуру полезной нагрузки луча) в конце стадии AH-шейдера. В этом случае RT-конвейер считает, что пересечение луча **r** с нашим процедурным объектом подтверждено (блок “Попадание принято?” на рис. 3), запоминает значение $t_{closest}$ и переходит к обработке следующего AABB. В противном случае (не найдено ни одного пересечения луча **r** с блоком патчей текущего AABB), мы сообщаем RT-конвейеру путем вызова оператора *ignoreIntersectionEXT* в конце AH-шейдера о необходимости проигнорировать текущий AABB и сразу перейти к обработке следующего AABB. Предложенная концепция “прозрачного AABB” позволяет перенести затратное вычисление пересечения луча **r** с процедурным объектом (блоком патчей) на более позднюю стадию AH-шейдера, чтобы реализовать на стадии I-шейдера описанную выше раннюю отбраковку AABB.

В-четвертых, мы ускоряем вычисление параметра $t_{closest}$ внутри исходного AABB с помощью *процедурных AABB*, ограничивающих патчи. Идея состоит в том, чтобы по мере продвижения от t_{in} к t_{out} вычислять пересечения луча **r** только с теми патчами, чьи процедурные AABB он пересекает (см. рис. 6). Для этого мы: 1) определяем ячейку сетки высот, которую пересекает трасса **r'** луча **r**; 2) вычисляем процедурный AABB для патча соответствующего этой ячейке (благо данные в узлах сетки высот нам известны); 3) проверяем пересечение процедурного AABB с лучом **r** с помощью модифицированной версии Slabs-теста, упомянутой выше. Если процедурный AABB пересечен лучом **r**, то мы вычисляем параметр $t_{closest}$ ближайшей к наблюдателю точки пересечения с помощью версии метода GARP [35], адаптированной для нашей задачи [5]. Если в результате вычисления мы получаем значение $t_{closest} > 0$, т.е. патч пересечен лучом **r**, то мы возвращаем это значение и прекращаем движение по лучу внутри AABB. В противном случае, мы переходим к следующей по трассе **r'** ячейке и повторяем описанную выше последовательность шагов, пока не будет вычислено положительное значение $t_{closest}$ или достигнута граница AABB (в этом случае мы присваиваем $t_{closest}$ значение -1).

В-пятых, при обходе процедурных AABB мы используем разработанный алгоритм устойчивого перехода к следующей по лучу **r'** ячейке. В классическом алгоритме обхода вокселов [36] продвижение вдоль луча реализуется на основе вычисления значений параметров t'_s и t'_t (см. рис. 7), а переход осуществляется в ячейку, соответствующую меньшей из накопленных сумм каждого из этих параметров. В нашей задаче кроме номеров (i_{next}, j_{next}) следующей ячейки необходимо также знать координаты точки P_{next} входа в эту ячейку (для вычисления $t_{closest}$ внутри процедурного AABB). Из-за погрешности машинного представления действительных чисел может возникнуть ситуация, когда точка P_{next} окажется вне границ следующей ячейки. Чтобы избежать остановки продвижения вдоль луча **r'** при таком расхождении данных, мы вычисляем номера (i_{next}, j_{next}) и координаты P_{next} следующим образом:

[1] Чтобы это реализовать, мы устанавливаем флаг прозрачности *gl_RayFlagsNoOpaqueEXT* при вызове функции *traceRayEXT* генерации луча **r** на стадии RG-шейдера. Это позволяет разблокировать стадию AH-шейдера (по умолчанию она выключена в RT-конвейере) и продолжить выполнение цикла обхода BVH-дерева после стадии I-шейдера по ветке “Нет” после блока “Объект непрозрачен?” на рис. 3.

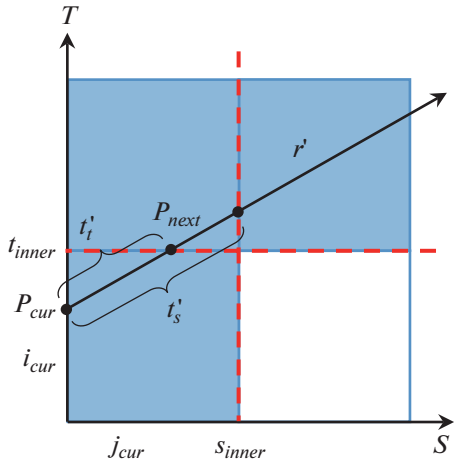

Рис. 7. Переход к следующей ячейке вдоль луча **r'**.

1) найдем номер N *внутреннего* угла текущей ячейки, который пересекает луч **r'**:

$$N = \text{int}(|r'_s| \geq 0) + 2 \times \text{int}(|r'_t| \geq 0),$$

где r'_s, r'_t – координаты вектора **r'** по осям S и T, а *int*() – операция преобразования булевского типа в целочисленный;

2) вычислим параметры t'_s и t'_t точек пересечений луча **r'** с прямыми s_{inner}, t_{inner}, образующими N-й внутренний угол (см. рис. 7);

3) запишем координаты P_{next} точки входа и номера (i_{next}, j_{next}) следующей ячейки:

$$P_{next} = P_{cur} + \min(t'_s, t'_t)\mathbf{r}',$$

$$i_{next} = i_{cur} + \text{sign}(r'_t) \cdot \text{int}(t'_t \leq t'_s),$$

$$j_{next} = j_{cur} + \text{sign}(r'_s) \cdot \text{int}(t'_s \leq t'_t),$$

где P_{cur} – координаты точки входа в текущую ячейку, а sign(x) – функция взятия знака числа x, которая возвращает 1 при $x > 0$, -1 при $x < 0$ и 0 при x равном 0 (случаи нулевых r'_s и r'_t обрабатываются отдельно).

Подводя итог, отметим, что при реализации предложенной технологии важно соблюдать “золотую середину” при выборе размера исходных блоков патчей. Если выбрать слишком маленький размер, то будет создаваться большое число AABB, и аппаратный блок обхода BVH-дерева не будет успевать их своевременно обрабатывать, а AH-шейдер будет недогружен работой. С другой стороны, при большом размере блока патчей вырастет число процедурных AABB и нагрузка на AH-шейдер, а аппаратный блок будет мало задействован. Количественное исследование этой закономерности и поиск “золотой середины” приведены в разд. 4.

4. РЕЗУЛЬТАТЫ

Предложенная технология была реализована в виде программного комплекса, осуществляющего моделирование и визуализацию поверхностей детализированных полей высот в реальном времени. Данный комплекс написан на языке C++, трассировка лучей реализована на RT-конвейере с помощью языка GLSL программирования шейдеров и API Vulkan (версия 1.3.204.1). С помощью разработанного комплекса было выполнено моделирование поверхностей на основе трех вариантов эталонной 16-битной сетки высот Пьюджет-Саунд [37, 38]: 4K × 4K, 8K × 8K и 16K × 16K[2]. В качестве аппаратной базы использовался персональный компьютер (Intel Core i7-6800K 3.40 ГГц, RAM 16 Гб DDR4), оборудованный видеокартой NVidia GeForce RTX 2080 (VRAM 8 Гб GDDR6, 2944 CUDA-ядер, 46 RT-ядер, драйвер NVidia DCH 512.59). Моделирование выполнялось при разрешении области вывода Full HD в условиях наблюдения, соответствующих средней сложности (см. рис. 8a) и высокой сложности (см. рис. 8b) трассировки лучей.

Для каждого из трех вариантов сетки высот мы провели серии экспериментов, в которых изменяли размер блока патчей от 4 × 4 до 1024 × 1024. После каждого изменения размера была измерена средняя скорость визуализации (в кадрах в секунду) в условиях средней и высокой сложности трассировки лучей, изображенных на рис. 8. Результаты замеров приведены на графиках рис. 9.

Проведенные эксперименты подтвердили наши теоретические предположения (см. конец разд. 3), касающиеся закономерности влияния размера блока патчей на производительность предлагаемого решения. Из графиков можно видеть, что “золотой серединой” будет выбор размера d блока патчей из диапазона [16, 32]. Именно при этих значениях d показатели производительности являются близкими к наибольшим, как при средней, так и при высокой сложности трассировки лучей. Сравнение полученных результатов с нашей первой реализацией (только на I-шейдере) [5] показало, что предложенное распределение вычислений между I-шейдером и AH-шейдером обеспечило прирост скорости визуализации до 20% (сравнение проводилось на тех же сетках высот, аппаратной базе и при тех же условиях наблюдения).

5. ЗАКЛЮЧЕНИЕ

В данной работе рассмотрена задача синтеза в реальном времени кусочно-билинейной модели

[2] Выражаем благодарность профессору Джону Реппи из Чикагского университета (факультет компьютерных наук) за помощь в предоставлении детализированной версии сетки высот Пьюджет-Саунд [38].

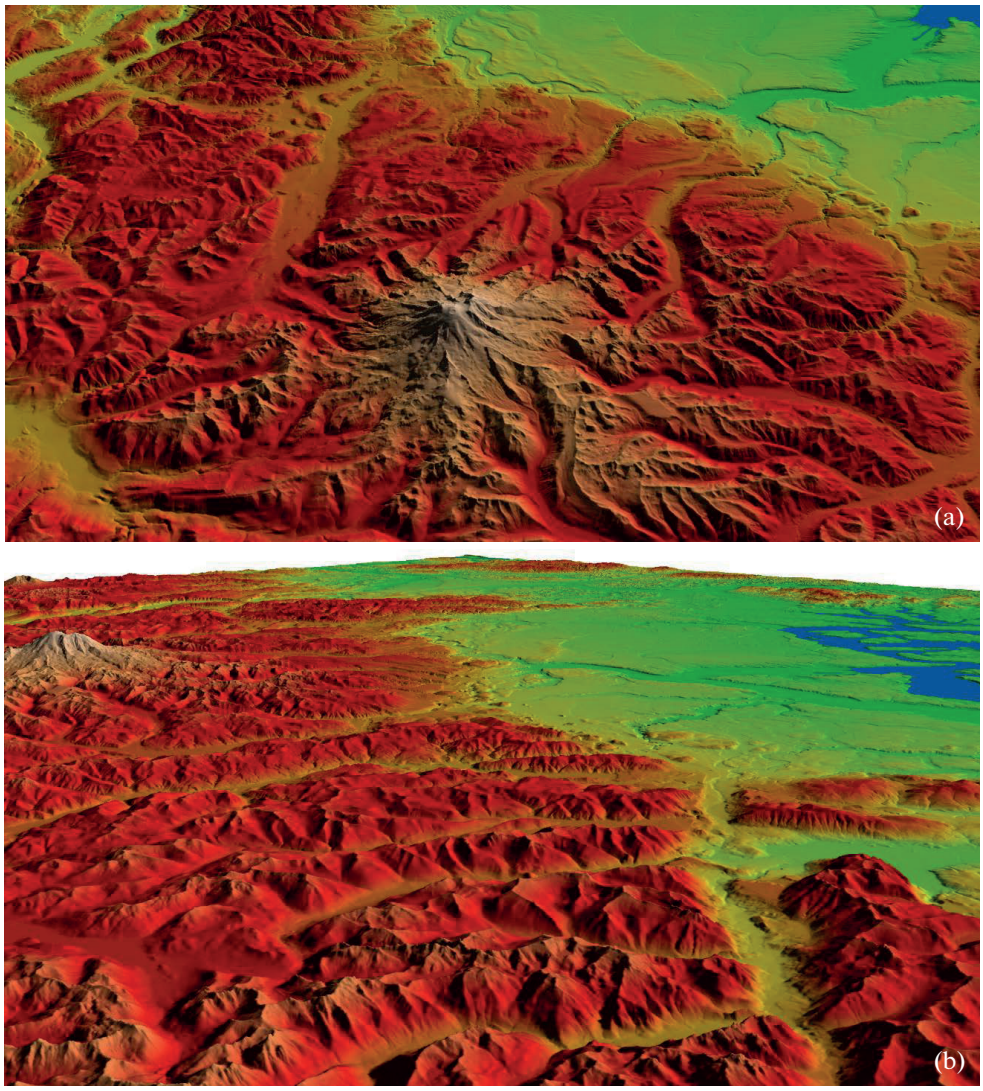

Рис. 8. Моделирование поверхности поля высот Пьюджет-Саунд с помощью предложенной технологии в условиях (a) средней сложности и (b) высокой сложности трассировки лучей.

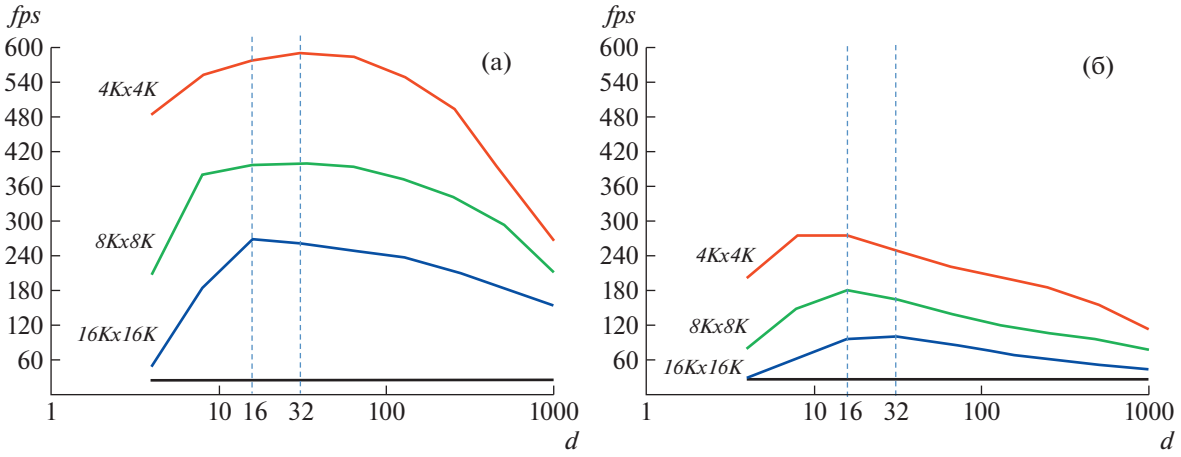

Рис. 9. Зависимость средней скорости визуализации от размера d блока патчей (логарифмическая шкала) для трех вариантов детализации сетки высот Пьюджет-Саунд при (a) средней и (b) высокой сложности трассировки лучей (нижняя черная линия означает порог в 25 кадров/секунду).

поверхности поля высот на конвейере трассировки лучей. Проблема состоит в том, что использование для рассматриваемой задачи базового подхода [31], применяемого для моделирования простых процедурных объектов типа сферы и параллелепипеда, приводит к перегруженности стадии I-шейдера и падению производительности RT-конвейера. Для решения этой проблемы была предложена эффективная технология, основанная на представлении моделируемой поверхности в виде блоков билинейных патчей и распределении обработки этих блоков (ограничивающих их AABB) между стадиями I-шейдера и AH-шейдера RT-конвейера. Ключевым нововведением такой обработки является ранняя отбраковка на стадии I-шейдера существенной части AABB, отобранных аппаратным блоком RT-конвейера, без непосредственного вычисления точек пересечения луча с процедурной поверхностью. Предложенная технология также включает в себя ряд модификаций, направленных на более эффективное использование RT-ядер и уменьшение количества затратных тестов пересечений типа “луч-билинейный патч”. В работе была исследована производительность созданного решения в условиях средней и высокой сложности трассировки лучей на эталонной сетке высот Пьюджет-Саунд в трех вариантах детализации. Проведенные исследования подтвердили эффективность предложенной технологии и возможность ее применения в системах виртуального окружения, научной визуализации, видеотренажерах и др.

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”.

СПИСОК ЛИТЕРАТУРЫ

1. NVIDIA Ada GPU Architecture // NVIDIA Corporation. 2022. https://images.nvidia.com/aem-dam/Solutions/ geforce/ada/nvidia-ada-gpu-architecture.pdf
2. *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
3. *Salmon J., McIntosh-Smith S.* Exploiting Hardware-Accelerated Ray Tracing for Monte Carlo Particle Transport with OpenMC // 2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS). 2019. P. 19–29. https://doi.org/10.1109/PMBS49563.2019.00008
4. *Komarov E.A., Zhdanov D.D., Zhdanov A.D.* Caustic Illuminance Calculation with DirectX Raytracing // Programming and Computer Software. 2022. V. 48. № 3. P. 172–180. https://doi.org/10.1134/S0361768822030069
5. *Тимохин П.Ю., Михайлюк М.В.* Рендеринг детализированных полей высот в реальном времени с использованием аппаратного ускорения трассировки лучей // GraphiCon 2022: труды 32-й Международной конференции по компьютерной графике и машинному зрению (Рязань, 19–22 сентября 2022 г.). 2022. С. 124–135. https://doi.org/10.20948/graphicon-2022-124-135.
6. *Han J.* Introduction to Computer Graphics with OpenGL ES. 1st. ed. Boca Raton, CRC Press, 2018.
7. *Михайлюк М.В., Мальцев А.В., Тимохин П.Ю., Страшнов Е.В., Крючков Б.И., Усов В.М.* Система виртуального окружения VirSim для имитационно-тренажерных комплексов подготовки космонавтов // Пилотируемые полеты в космос. 2020. № 4 (37). С. 72–95. https://doi.org/10.34131/MSF.20.4.72-95
8. *Hongxiang R., Yicheng J., Liling C.* Real-time Rendering of Ocean in Marine Simulator // 2008 Asia Simulation Conference – 7th International Conference on System Simulation and Scientific Computing. 2008. P. 1133–1136. https://doi.org/10.1109/ASC-ICSC.2008.4675536.
9. *Pajarola R., Gobbetti E.* Survey of semi-regular multiresolution models for interactive terrain rendering // The Visual Computer. 2007. V. 23. № 8. P. 583–605. https://doi.org/10.1007/s00371-007-0163-2
10. *Li S., Zheng C., Wang R., Huo Y., Zheng W., Lin H., Bao H.* Multi-resolution terrain rendering using summed-area tables // Computers & Graphics. 2021. V. 95. P. 130–140. https://doi.org/10.1016/j.cag.2021.02.003
11. *Timokhin P.Yu., Mikhaylyuk M.V.* Computer Modeling and Visualization of Accurate Terrain Shadows in Virtual Environment System // Scientific Visualization. 2022. V. 14. № 2. P. 77–87. https://doi.org/10.26583/sv.14.2.07
12. *Cornel D., Horváth Z., Waser J.* An Attempt of Adaptive Heightfield Rendering with Complex Interpolants Using Ray Casting // Technical Report. arXiv:2201.10887v1. 2022. P. 1–9. https://doi.org/10.48550/arXiv.2201.10887.
13. *Frolov V.A., Voloboy A.G., Ershov S.V., Galaktionov V.A.* Light Transport in Realistic Rendering: State-of-the-Art Simulation Methods // Programming and Computer Software. 2021. V. 47. № 4. P. 298–326. https://doi.org/10.1134/S0361768821040034
14. *Parker S., Shirley P., Livnat Y., Hansen C., Sloan P.-P.* Interactive Ray Tracing for Isosurface Rendering // VIZ’98: proceedings of the IEEE Visualization 98. 1998. P. 233–238. https://doi.org/10.1109/VISUAL.1998.745713
15. *Brawley Z., Tatarchuk N.* Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing // ShaderX3: Advanced Rendering with DirectX and OpenGL. 1st. ed. Charles River Media. 2004. P. 135–154.
16. *Tatarchuk N.* Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows // ACM Symposium on Interactive 3D Graphics and Games (I3D '06).

2006. P. 63–69.
https://doi.org/10.1145/1111411.1111423.

17. *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. https://doi.org/10.1145/1053427.1053453.

18. *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. 2010. P. 161–168. https://dl.acm.org/doi/10.5555/1839214.1839243.

19. *Policarpo F., Oliveira M.M.* Relaxed Cone Stepping for Relief Mapping // GPU Gems 3. Addison-Wesley Professional. 2007. P. 409–428. https://developer.nvidia.com/gpugems/gpugems3/part-iii-rendering/chapter-18-relaxed-cone-stepping-relief-mapping.

20. *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. V. 18. № 11. P. 1811–1823. https://doi.org/10.1109/TVCG.2011.281

21. *Tevs A., Ihrke I., Seidel H.-P.* Maximum Mipmaps for Fast, Accurate, and Scalable Dynamic Height Field Rendering // Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games (I3D'08). New York. 2008. P. 183–190. https://doi.org/10.1145/1342250.1342279.

22. *Dick C., Krüger J.H., Westermann R.* GPU Ray-Casting for Scalable Terrain Rendering // Eurographics '09, 2009. P. 43–50. https://doi.org/10.2312/ega.20091007

23. *Lee E.-S., Lee J.-H., Shin B.-S.* A bimodal empty space skipping of ray casting for terrain data // The Journal of Supercomputing. 2016. V. 72. № 7. P. 2579–2593. https://doi.org/10.1007/s11227-015-1522-9

24. *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. https://elib.dlr.de/101497/.

25. *Dübel S., Middendorf L., Haubelt C., Schumann H.* A Flexible Architecture for Ray Tracing Terrain Heightfields // Proceedings of the International Summerschool on Visual Computing, Rostock, Germany, 2015. P. 3–22.

26. *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. https://doi.org/10.1109/ITCGI.2018.8602735.

27. *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

28. *Thonat T., Beaune F., Sun X., Carr N., Boubekeur T.* Tessellation-free displacement mapping for ray tracing // ACM Transactions on Graphics. 2021. Vol. 40. No. 6. Article 282. P. 1–16. https://doi.org/10.1145/3478513.3480535.

29. *Silva V., Novello T., Lopes H., Velho L.* Real-Time Rendering of Complex Fractals // Ray Tracing Gems II. NVIDIA. 2021. P. 529–544. https://doi.org/10.1007/978-1-4842-7185-8_33

30. *Brüll F.* Fast Transparency and Billboard Ray Tracing with RTX Hardware // Master thesis. Clausthal University of Technology. 2020. 81 p. https://doi.org/10.13140/RG.2.2.14692.19842.

31. NVIDIA Vulkan Ray Tracing Tutorials. Intersection Shader – Tutorial. 2020–2022. https://github.com/nvpro-samples/vk_raytracing_tutorial_KHR/tree/master/ray_tracing_intersection.

32. Vulkan 1.3.238 – A Specification (with all registered Vulkan extensions) // The Khronos Vulkan Working Group. 2022. https://www.khronos.org/registry/vulkan/specs/1.3-extensions/pdf/vkspec.pdf.

33. *Majercik A., Crassin C., Shirley P., McGuire M.* A Ray-Box Intersection Algorithm and Efficient Dynamic Voxel Rendering // Journal of Computer Graphics Techniques. 2018. V. 7. № 3. P. 66–82. ISSN 2331-7418. https://www.jcgt.org/published/0007/03/04/paper-lowres.pdf.

34. NVIDIA Vulkan Ray Tracing Tutorials. https://github.com/nvpro-samples/vk_raytracing_tutorial_NV/tree/master/ray_tracing_intersection/shaders/raytrace.rint.

35. *Reshetov A.* Cool Patches: A Geometric Approach to Ray/Bilinear Patch Intersections // Ray Tracing Gems. 2019. P. 95–109.

36. *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.

37. Large Geometric Models Archive // Georgia Institute of Technology. https://www.cc.gatech.edu/projects/large_models/.

38. Puget Sound test map // The University of Chicago. https://www.classes.cs.uchicago.edu/archive/2015/fall/23700-1/final-project/puget-sound/index.html.