ПРОГРАММИРОВАНИЕ, 2017, № 3, с. 83–92

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.925.3

КОМПАКТНАЯ ПО ПАМЯТИ РЕАЛИЗАЦИЯ АЛГОРИТМА METROPOLIS LIGHT TRANSPORT НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ *

© 2017 г. **В.А. Фролов**[*,**], **В.А. Галактионов**[*]

[*]*Институт прикладной математики им. М.В. Келдыша РАН*
125047 Москва, Миусская пл., д.4
[**]*Московский государственный университет им. М. В. Ломоносова*
1199991 Москва, ГСП-1, Ленинские горы, д. 1
E-mail: vova@frolov.pp.ru, vlgal@gin.keldysh.ru

Поступила в редакцию 25.11.2016

В данной работе предложены решения основных проблем, возникающих при реализации группы методов Metropolis Light Transport (MLT) на графических процессорах (GPU). Предложен новый метод реализации "прожига" ("burn in") на GPU при помощи обычного Монте–Карло, благодаря которому удаётся в значительной степени амортизировать проблему "начального смещения" ("startup bias"). Предложены методы экономии памяти, в том числе исследуется возможность применения Metropolis Light Transport с множественными предложениями переходов. Исследованы технические аспекты эффективной реализации MLT на GPU.

1. ВВЕДЕНИЕ

Можно сказать, что трассировка путей является применением "обычного метода Монте–Карло" (Ordinary Monte Carlo, OMC) для вычисления интеграла освещенности и решения уравнения рендеринга [1]. В таком Монте-Карло все выборки (или сэмплы) независимы друг от друга. Это облегчает параллельную реализацию алгоритма, но в процессе расчёта выбрасывается огромное количество информации, которую можно было бы использовать.

В свою очередь, группа методов Metropolis Light Transport — это применение Монте–Карло по схеме марковских цепей (Markov Chain Monte Carlo, MCMC) для вычисления интеграла освещенности и решения уравнения рендеринга [2], [3]. В отличие от обычного Монте–Карло MCMC использует коррелированные выборки. Это позволяет алгоритму многократно переиспользовать информацию о важных областях (областях с высокой значимостью) интегрируемой функции, автоматически помещая больше выборок в области с высокой значимостью [4].

1.1. MCMC и MIS

Трассировка путей в современных индустриальных системах расчёта освещения была бы невозможна без таких методов как выборка по значимости и многократная выборка по значимости [5]. На основе многократной выборки по значимости (Multiple Importance Sampling, MIS) были разработаны более устойчивые к выбросам алгоритмы — Bidirectional Path Tracing (BDPT) [2], Bidirectional Photon Mapping (BDPM) [6] и Vertex Connection Merging (VCM) [7]. Многократная выборка по значимости в BDPT и VCM повышает устойчивость за счёт скорости — путь от источника до камеры строится при помощи N стратегий и неудачные стратегии зануляются "MIS весом" ("MIS weight") апостериорно. Именно такой апостериорный учёт и является причиной низкой скорости, т.к. зачастую из N уже посчитанных лучей только один-два луча вносят существенный вклад (в обычном Path Tracing (PT) $N = 2$,

[*]Работа выполнена при поддержке РФФИ, гранты 16-31-60048 мол_а_дк и 16-01-00552.

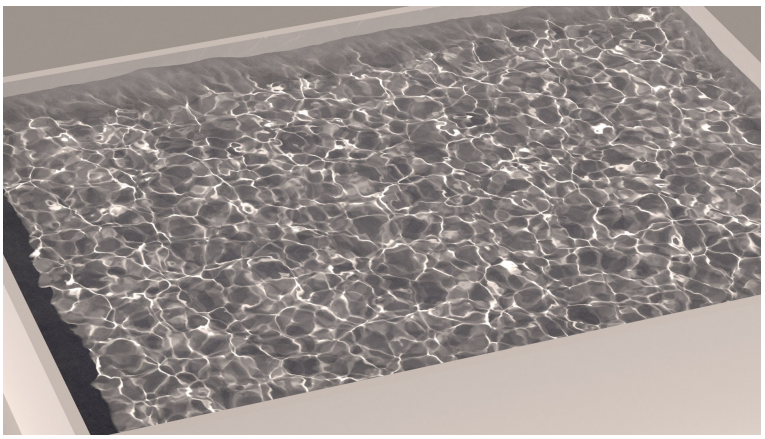

Рис 1. Расчёт водных каустиков от небольшого источника света на видеокарте HD7970 с использованием алгоритма Metropolis Light Transport; время расчёта — 1 час в разрешении 1920×1088.

поэтому условная эффективность всегда будет не ниже 50%). Аналогично BDPM выполняет достаточно дорогой сбор освещённости из фотонной карты на каждом переотражении луча, вышедшего из камеры, апостериорно учитывая вклад фотонных карт.

В противовес методам, основанным на многократной выборке по значимости, группа методов Metropolis Light Transport (MLT) генерирует выборки не пропорционально какой-либо одной части подынтегральной функции (что делает каждая из стратегий BDPT), а пропорционально всему интегралу освещённости как многомерной функции, спроецированной на множество пикселей изображения: $F(x, y, r_0, r_1, ...r_n) \xrightarrow{project} F(x, y)$. MLT автоматически помещает больше выборок в более сложные участки функции F, значительно уменьшая дисперсию при расчёте сложных эффектов (например, водные каустики на рис. 1).

Важно отметить при этом, что и многократная выборка по значимости и MCMC на практике могут и должны использоваться вместе. Однако это можно делать как минимум несколькими способами, например, используя просто PT, BDPT[5] или Multiplexed Metropolis Light Transport (MMLT) [8].

1.2. PT vs BDPT

В нашей работе мы исследовали MLT поверх однонаправленной трассировки путей (аналогично работе [3]) в сочетании с многократной выборкой по значимости по нескольким причинам. Во-первых, MLT для двунаправленной трассировки путей (BDPT) увеличивает число путей, в которых существенный вклад будет только от одной стратегии, что в свою очередь снижает эффективность самого BDPT [8]. Во-вторых, BDPT достаточно затратен (по памяти) и сложно-реализуем на GPU, т.к. необходимо не только хранить вершины для соединений (Light Vertex Cache) [9], но и определённым образом хранить и накапливать MIS веса (Recursive MIS [10]).

Наконец, алгоритм Метрополиса обычно применяется в ситуациях, когда трудно или невозможно подобрать хорошую стратегию генерации выборок. BDPT, с другой стороны, уже умеет генерировать хорошие стратегии для большинства пикселей. Применение к нему MLT не даёт столь существенного преимущества как применение MLT к однонаправленной трассировке путей. Следовательно, исследование собственно свойств MLT будет менее показательным.

Алгоритм MMLT [8], с другой стороны, представляет собой применение MCMC к упрощённому варианту двунаправленной трассировки путей, в котором соединяются только конечные вершины путей из источника и камеры (в отличие от BDPT, где вершины соединяются по правилу “каждая с каждой”). С точки зрения практики этот алгоритм является наиболее интересным выбором, поскольку в отличие от однонаправленного MLT из работы [3] он хорошо работает для поверхностей с микрорельефом и сцен с большим количеством мелких деталей. Тем не менее, поскольку целью данной работы было ре-

шение базовых проблем, возникающих при реализации MLT на графических процессорах, микрорельеф и геометрически сложные сцены были "вынесены нами за скобки" и, делая выбор между MMLT и однонаправленным MLT из работы [3], мы выбрали последнее.

1.3. *Терминология*

Терминология для MCMC в компьютерной графике и традиционной вычислительной математике немного отличается. Если в математике говорят, что MCMC "предлагает переходы" ("make proposals") [4], которые затем принимает или отвергает, то в компьютерной графике переходы путей из одного состояния в другое называют "мутациями" [2], которые принимаются либо отвергаются. Вместо устоявшегося в последние годы термина "мутация" мы будем использовать традиционный термин "переход" или "предложение перехода", поскольку он точнее описывает то, что происходит в марковском процессе. Более того, в отличие от термина "мутация", термин "переход" не наводит на мысли о родстве между MCMC и генетическим алгоритмом (ничего общего между которыми безусловно нет [2]).

2. ПРЕДЫДУЩИЕ РАБОТЫ

2.1. *MLT на GPU*

В работе [11] исследовалась эффективность различных алгоритмов интегрирования освещённости на GPU – PT, BDPT и BDPT + MLT. Акцент сделан на реализацию MLT поверх фреймворка двунаправленной трассировки путей (BDPT). Каждый поток в работе [11] считает свою независимую от других потоков марковскую цепь, используя схему ленивых вычислений при построении путей для новых переходов. Важным моментом является то, что вероятность предложения "большого перехода" (large step) в работе [11] была очень большой — 50%. Такое значение объясняется тем, что при параллельной реализации MLT даёт очень большое "начальное смещение" и, чтобы оно было не столь заметно, автор [11] предлагает учитывать больше вклада от обычного BDPT, считая "большие шаги" как отдельный метод и смешивая результаты MLT и BDPT при помощи многократной выборки по значимости аналогично работе [3]. Такое решение опирается на предположение о том, что BDPT генерирует хорошие выборки в большинстве случаев (что может быть неверно, т.к. зависит от сцены), и в значительной мере просто "забивает" MLT обычной двунаправленной трассировкой путей.

Для накопления цвета в гистограмме изображения в работе [11] были использованы атомарные операции с плавающей точкой. Такие операции существуют далеко не на всех GPU и отсутствуют в OpenCL.

2.2. *Множественные предложения*

Существуют два известных способа реализации MCMC с множественными предложениями переходов — "Kingpin" [12] и "Theater" [13]. В работе [15] было предложено использовать MCMC с несколькими предложениями переходов (Multiple Propolsal Metropolis Light Transport) для повышения когерентности лучей при реализации трассировки на CPU с использованием SIMD инструкций. Для учёта множественных переходов был использован "Kingpin" метод из работы [12]. Однако реализованный алгоритм в большей степени подходит для CPU чем для GPU. Причина этого в том, что "Kingpin" — двухшаговый метод. На первом шаге он генерирует k предложений $x \to y_1, ..y_k$ и выбирает из них некоторое предложение y_j пропорционально его значимости/яркости. А уже после этого он генерирует ещё $k-1$ новых предложений $y_j \to z_1, ..z_{k-1}$. Это не позволяет вычислять все предложения параллельно. Отчасти по этой причине в недавней работе [16] была предложена иная, спекулятивно-рекурсивная схема вычисления переходов для повышения когерентности выполнения потоков на GPU и вычисления всех предложений переходов параллельно. В работе [16] также указывается на тот факт (это вторая причина), что каждое предложение в "Kingpin" вносит вклад в k раз меньший, чем в обычном MLT, что дополнительно будет увеличивать "начальное смещение" на GPU (мы дадим объяснение этому эффекту в следующем разделе). Идея работы [16] заключается в том, чтобы спекулятивно вычислить бинарное дерево всех возможных переходов, до последнего момента откладывая фактическое выполнение

перехода. У подхода, реализованного в работе [16], два основных недостатка: первый — это потеря эффективности алгоритма при росте глубины дерева. Второй — сохранение всех промежуточных векторов случайных чисел (соответствующих узлам дерева) в памяти. Последнее исключительно важно, т.к. именно MCMC с множественными переходами мог бы позволить хранить вектора случайных чисел только для текущего состояния цепи, а для предложений использовать схему ленивых вычислений на лету аналогично работе [11] и, таким образом, уменьшить объем необходимой памяти в N раз, где N — число параллельных предложений переходов. Метод из работы [16] не позволяет воспользоваться этим потенциальным преимуществом.

2.3. "Начальное смещение" на GPU

Как правило, при реализации на CPU в MLT проблема "начального смещения" не заметна. Однако при реализации MLT на GPU эта проблема начинает выглядеть совершенно иначе. Поскольку MLT накапливает значения в гистограмме изображения значениями, близкими к единице, при большом количестве параллельных цепей "начальное смещение" становится огромным [11]:

На CPU 1 поток за одну условную итерацию сделает достаточно большое число шагов (например, 1 миллион). Это с высокой вероятностью позволит ему накопить нужные значения функции в областях с высокой значимостью (поскольку он в них бывает часто).

На GPU за одну условную итерацию 1 миллион потоков параллельно сделают лишь небольшое число шагов (например, 4). Если начальные позиции цепей были выбраны случайно с равномерным распределением, ни одна ячейка гистограммы не сможет накопить значение больше 4 (вернее, вероятность этого крайне мала). Это в свою очередь приводит к тому, что яркие участки изображения на начальном этапе расчёта будут недостаточно яркими, а тёмные — недостаточно тёмными. Это и есть "начальное смещение" (рис. 2, верхний ряд).

Таким образом, среди нерешённых проблем в существующих работах следует отметить: (1) — проблему большого начального смещения при выполнении большого числа потоков параллельно на GPU, и (2) — проблему значительных трат памяти для хранения случайных чисел. Именно на решении этих 2 проблем сфокусирована наша работа.

3. ПРЕДЛАГАЕМЫЕ МЕТОДЫ

3.1. "Прожиг" при помощи OMC

В работах [2] и [3] отмечено, что один из способов снизить "начальное смещение" заключается в том, чтобы выбирать начальные точки ("seed") для MCMC не равномерно-случайно, а пропорционально сэмплируемой функции $F(x)$. Однако эта идея не была использована ранее в работах, реализующих MLT на GPU [11], [16].

Традиционный метод "прожига" в MCMC с отбрасыванием некоторого количества итераций цепи Маркова в теории решает задачу: если мы будем обрывать цепи после достаточно большого числа итераций N, то мы будем получать выборки пропорционально $F(x)$: больше выборок в более значимых областях функции и меньше — в менее значимых. Но для этого нам уже нужно прогнать достаточно большое число цепей с большим числом итераций N, что эквивалентно по времени итоговому расчёту. Вместо MCMC для отбора начальных точек мы предлагаем использовать обычный Монте Карло:

1. Пусть число потоков на GPU равно M.
2. Параллельно в M потоков выполним m итераций OMC с M случайными выборками, каждый раз отбирая из них M/m выборок пропорционально их яркости.
3. Параллельно в M потоков запустим MLT с отобранными M начальными выборками.

Отбор выборок пропорционально яркости на втором шаге был реализован через параллельное вычисление префиксной суммы на GPU с последующим бинарным поиском [17]. Важно отметить, что использование обычного Монте–Карло в начале расчёта является необходимым условием даже без применения нашего алгоритма, поскольку при помощи обычного Монте–Карло оценивают константу нормализации [2]. Поэтому

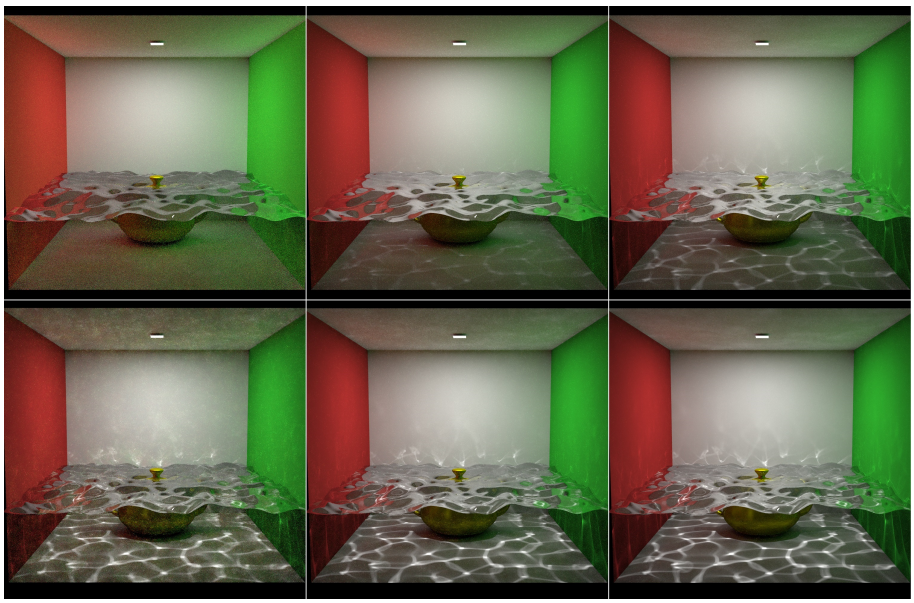

Рис 2. MLT без предложенного метода прожига (верхний ряд) и с предложенным методом (нижний ряд) за 10, 100 и 500 секунд на AMD R9200. Сцена “CornellWater”.

в некотором смысле (кроме случая когда используется смешивание PT/BDPT и MLT аналогично [3] и [11]) наш метод можно считать условно-бесплатным, т.к. описанный выше алгоритм отбора можно производить во время оценки константы нормализации. Амортизировав проблему начального смещения, мы смогли уменьшить вероятность “большого шага” (“large step”) [3] с 50% (в работе [11]) до 5–10% в нашей работе и, таким образом, повысить эффективность MLT на сложных участках. Результат применения прожига представлен на рис. 2.

3.2. Экономия памяти

Мы предлагаем использовать несколько методов экономии памяти:

1. Уменьшение количества параллельно работающих потоков в k раз. Данный способ разумно применять для GPU с небольшим числом мультипроцессоров, поскольку для загрузки их вычислениями достаточно меньше потоков. При критической нехватке памяти число потоков почти всегда можно уменьшить в 2–4 раза (рис. 3). Причём при уменьшении количества потоков дополнительно снижается “начальное смещение”, поскольку оставшиеся потоки делают шаги быстрее. Интересно отметить, что более мощные GPU, как правило, имеют бóльший объём памяти, что позволяет сохранять высокую эффективность работы на любом GPU.

2. Уменьшение битности хранимых чисел. Поскольку на практике MLT обычно хранит случайные числа в интервале от 0 до 1 (Primary Space Sample MLT [3]), нет необходимости в использовании 32 бит на 1 число с плавающей точкой. Можно использовать для хранения случайных чисел фиксированную точку и хранить только 24 бита (это следует из того факта, что при генерации нового предложения перехода будет использоваться сложение с плавающей точкой, и для чисел близких к 1 возможна максимальная точность в 24 бита). Далее, для многих событий, например, выбор номера источника света и выбор между френелевским и диффузным отражением достаточно 16 бит и даже меньше. Только сэмплирование BRDF и источника требует высокой точности. Таким образом, на каждый отскок мы хранили 4 числа с точностью в 24 бита, и ещё 4 числа с точностью в 16 бит. Итого 160 бит (5×32 бита) и 8 случайных чисел на 1 отскок.

3. Использование ленивой схемы вычисления

предложений, аналогично работе [11]. Мы расширили эту идею на Multiple Proposal MLT (см. след. раздел). Таким образом, для N потоков и k предложений мы храним N/k векторов для N/k цепей.

3.3. *MLT с расщеплённым переходом*

В целях экономии памяти и дополнительного уменьшения "начального смещение" мы предлагаем новый алгоритм, названный нами "MLT с расщеплённым переходом" (алгоритм 1). В противоположность "Kingpin" методу за 1 итерацию мы вносим в гистограмму вклад в k раз больше ("Kingpin" вносит в k раз меньше), где k — число предложений. За счёт этого "начальное смещение" уменьшается по сравнению с обычным MLT. Рассмотрим реализацию одного шага для изображения $F(x)$ и функции значимости $I(x) = lum(F(x))$; x — точка на текущем шаге; $y1$; $y2$ — предложения перехода; $xNew$ — точка на следующем шаге:

$$y1, y2 \leftarrow makeProposals(x)$$
$$a1 = \min(1, I(y1)/I(x))$$
$$a2 = \min(1, I(y2)/I(x))$$
$$image(y1.xy) \xleftarrow{contribute} \frac{F(y1)}{I(y1)} * a1$$
$$image(y2.xy) \xleftarrow{contribute} \frac{F(y2)}{I(y2)} * a2$$
$$image(x.xy) \xleftarrow{contribute} \frac{F(x)}{I(x)} * (2 - a1 - a2)$$
$$xNew \xleftarrow{select} (x, y1, y2) \propto (2 - a1 - a2, a1, a2)$$

Алгоритм 1. MLT с расщеплённым переходом.

Кроме экономии памяти, предложенный алгоритм преобразует начальное смещение в шум (рис. 4, 5) и может быть использован для повышения степени параллелизма, когда на нескольких GPU вычисляется большое количество относительно коротких цепей. Недостатком алгоритма является ухудшение точности (рис. 5) в пределе (при длительном расчёте). Причина, по которой сходимость ухудшается, заключается в том, что предложенный алгоритм не увеличивает вероятность принятия перехода, а лишь делает сам переход "расщеплённым". Итоговая точность в алгоритме Метрополиса при этом зависит от фактического числа принятых переходов, а не от количества сделанных попыток/предложений, от чего как раз зависит время расчёта). Чтобы это исправить, необходима некоторая иная схема вычисления вероятностей принятия перехода (например, аналогично схеме "theater" в [13] и [14], что является предметом наших будущих исследований.

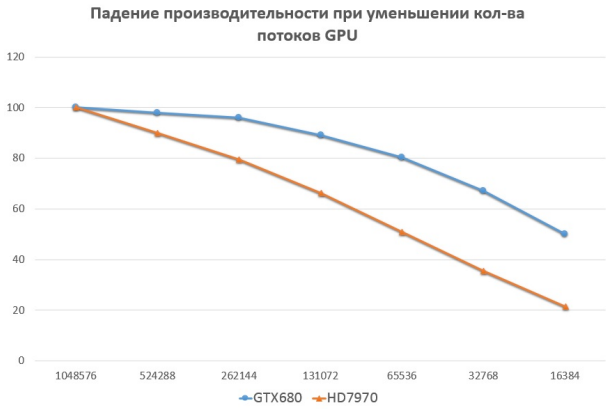

Рис. 3. Падение производительности трассировки путей на различных GPU при уменьшении числа потоков в процентах. Характер кривых сохранялся для всех тестовых сцен. Для увеличения наглядности продемонстрированы только графики для сцены CornellWater (рис. 2).

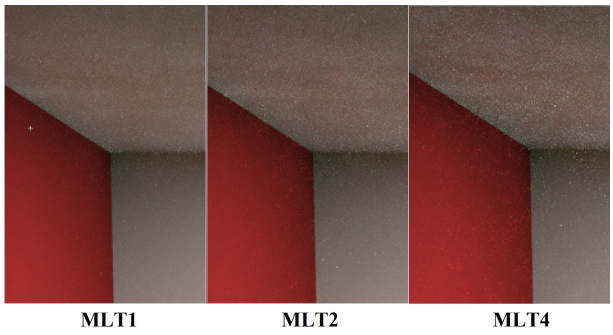

Рис 4. Справа налево — фрагмент изображения для 1, 2 и 4 попыток за одно и то же время (10 минут). Несмотря на явно увеличивающийся уровень шума, фактическая ошибка (разница с эталоном) для всего изображения при увеличении числа попыток уменьшается (рис. 5). Это происходит из-за наличия "начального смещения" в изображениях. При увеличении числа попыток смещение переходит в шум, который более заметен для глаза.

3.4. *Аккумулирование вклада*

Metropolis Light Transport работает с путями, свободно и неконтролируемо перемещающимися в пространстве экрана. Для того чтобы учесть вклад от какого-либо потока в определённый пиксел изображения, в работах [11] и [16] были

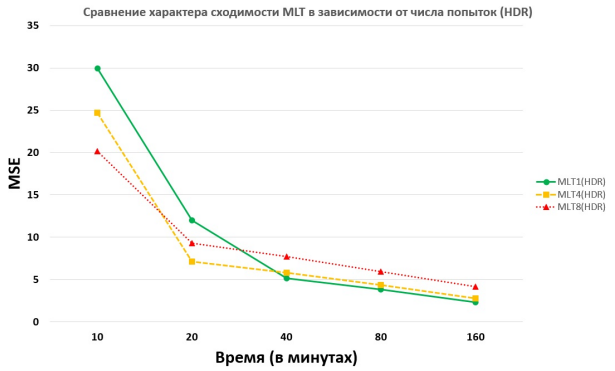

Рис 5. Зависимость ошибки (MSE) от времени для MLT с разделённым переходом и различным числом попыток. Эксперимент проведён на NV GTX670 для сцены CornellWater. 10 мин. соответствуют 1200 шагам цепей для 524K потоков. Посчитано для HDR изображений. Характер ошибки сохраняется и на других тестовых сценах. По оси X — время в минутах. По оси Y — ошибка (MSE).

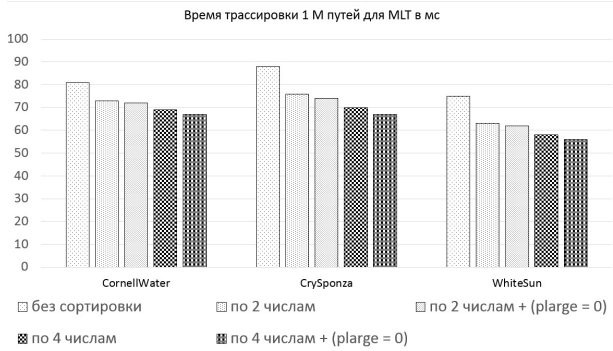

Рис 6. Время в милисекунд. трассировки путей с сортировкой (включая время сортировки) и без. По 2 числам — сортировка только по экранным координатам. По 4 числам — с учётом 2 дополнительных случайных чисел, используемых для генерации направления отражения на первом отскоке. Надпись “(plarge = 0)” означает время при вероятности большого шага равной нулю вместо 10%, используемой нами по умолчанию. Соответсвующие сцены изображены на рисунках 11, 12, 13.

использованы атомарные операции с плавающей точкой. Однако, поскольку в OpenCL отсутствует поддержка атомарных операций с плавающей точкой, мы реализовали и исследовали производительность двух альтернативных методов аккумулирования вклада путей в гистограмму итогового изображения.

Первый алгоритм (будем далее называть его “метод 1”) имитирует поддержку атомарных операций при помощи сериализации запросов в память и операции “atomic_cmpxchg” с целыми числами (доступной в OpenCL 1.1) [18]. Второй, предложенный нами (будем далее называть его “метод 2”), сортирует все пути по экранным координатам при помощи битонической сортировки (колонка “bitonic_sort”) и в отдельном ядре собирает вклад с отсортированных путей для каждого пиксела при помощи бинарного поиска (колонка “gather”). Сравнение производительности методов представлено в таблице 1.

GPU	метод1	метод2	bitonic_sort	gather
GTX680	6 мс	8 мс	5 мс	3 мс
GTX980	4.1 мс	5.8 мс	3.6 мс	2.2 мс
HD7970	4.6 мс	5.4 мс	3.5 мс	1.9 мс
R9200	4.5 мс	4.1 мс	2.6 мс	1.5 мс

Таблица 1. Сравнение времени, потраченного на аккумулирование цвета в пространстве экрана для 524 тысяч путей при помощи атомарных операций и их имитации (метод 1) и предложенным алгоритмом (метод 2, состоящий из 2 этапов — bitonic_sort и gather). Для видеокарт NV (две первые строчки) были использованы аппаратные атомарные операции, доступные в CUDA, для видеокарт AMD (две последние строчки) — сериализация запросов в память [18]. Разрешение изображения — 1024×1024. Сцена — CornellWater (рис. 2).

Таким образом, можно заключить, что оба метода на практике одинаково подходят для учёта вклада путей в изображение, т.к. имеют сравнимое время работы много меньшее времени трассировки путей (около 100 мс). Однако предложенный нами метод, хотя и медленнее в среднем, не использует атомарные операции даже с целыми числами и, в отличие от первого метода, может быть реализован на OpenCL 1.0. Предложенный алгоритм дополнительно сортирует потоки, что является полезным побочным эффектом и будет использовано нами далее.

Отсортированные пути обладают повышенной когерентностью, что уменьшает расхождения на ветвлениях при выполнение кода на GPU (рис. 6). Причём сортировка путей на текущем шаге позволяет сохранить пространственно-близкие группы лучей и на следующем шаге, поскольку небольшой шаг (при генерации предложения перехода) в первичном пространстве

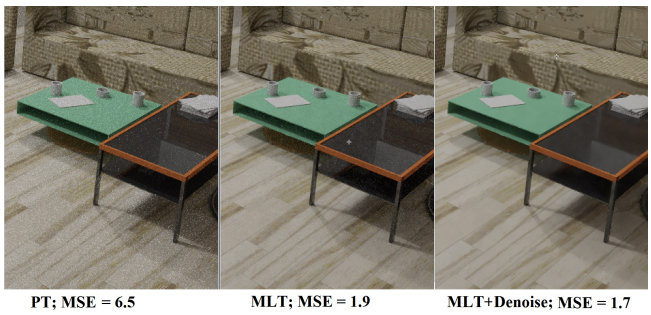

Рис. 7. Сравнение на сцене “comp1” (рис. 14). Время рендера — 10 минут на GTX680.

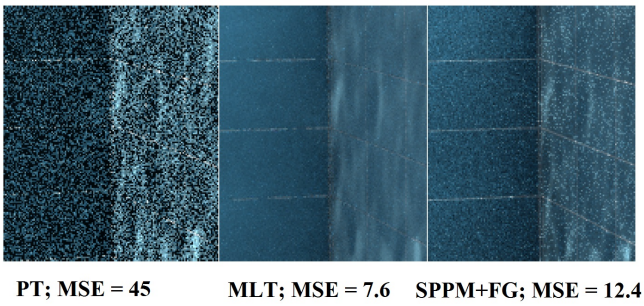

Рис. 8. Сравнение на сцене “comp2” (рис. 15). Время рендера — 10 минут на GTX680.

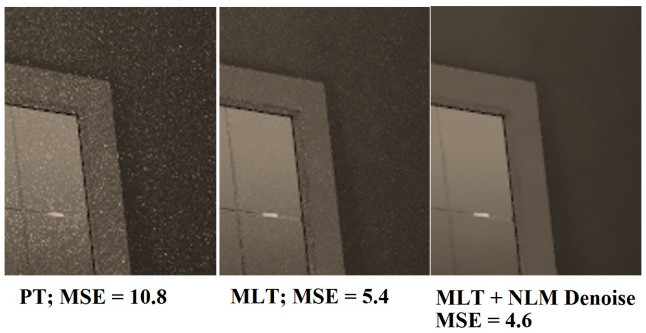

Рис. 9. Сравнение на сцене “comp3” (рис. 16). Время рендера — 10 минут на GTX680.

путей приводит, как правило, к небольшим изменениям путей в мировом пространстве. Для построения пространственного индекса (по которому производилась сортировка) мы использовали коды Мортона [19] для 16 битных представлений чисел с фиксированной точкой. Два 16-битных числа, отвечающие за экранные координаты, и два 16-битных числа, отвечающие за генерацию отражённого луча на первом отскоке. Все четыре 16-битных числа при этом были получены из 24-битных представлений (которые были использованы нами для хранения чисел в памяти) путём сдвига вправо.

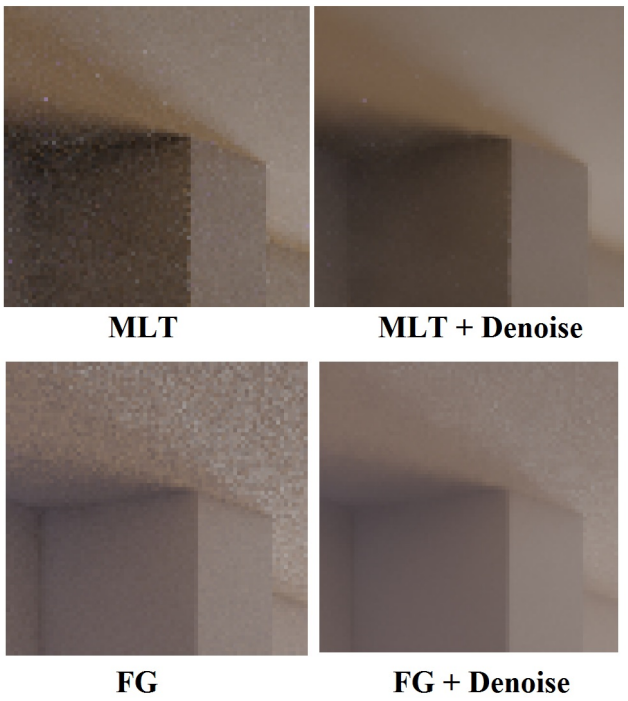

Рис. 10. Сравнение денойзинга для MLT и FG.

4. РЕЗУЛЬТАТЫ

Мы сравнили разработанную нами реализацию алгоритма MLT на OpenCL с трассировкой путей (PT) и стохастическими прогрессивными фотонныим картами в сочетании с финальным сбором (SPPM + FG) на GPU, реализованными на CUDA в рендер-системе Hydra Renderer [20]. Наша реализация Metropolis Light Transport на GPU выигрывает по точности (т.е. квадратичная ошибка “MSE” при сравнении с эталоном становится меньше) при одинаковом времени рендера от 2 до 5 раз (рис. 7–9). Следует отметить при этом, что для методов PT и комбинации SPPM+FG, обозначенный выигрыш в точности равносилен выигрышу по скорости от 4 до 25 раз.

Мы заметили также, что изображения, полученные при помощи Metropolis Light Transport, значительно лучше поддаются денойзингу (например, на основе Non Local Means фильтра), чем изображения, рассчитанные при помощи PT или комбинации SPPM+FG (рис. 10).

5. ВЫВОДЫ

Таким образом, по сравнению с существующими реализациями MLT на GPU предлагаемый

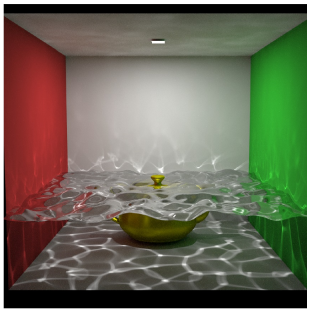

Рис 11. Сцена “CornellWater”

Рис 12. Сцена “CrySponza”

Рис 13. Сцена “WhiteSun”

Рис 14. Сцена “comp1”

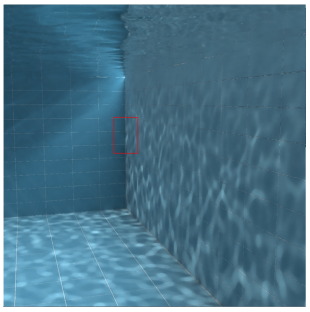

Рис 15. Сцена “comp2”

Рис 16. Сцена “comp3”

метод “прожига” впервые позволил амортизировать проблему “начального смещения” и получить приемлемую оценку изображения в начале расчёта на GPU (рис. 2), а предлагаемые методы экономии памяти позволяют нам на практике сократить объем необходимой памяти от 4 до 16 раз. Для глубины трассировки в 24 отскока и 256K потоков вектор случайных чисел (128×32 бит на 1 поток) занял 128 MB.

Предложенный метод аккумулирования вклада в изображение на основе сортировки потоков в качестве полезного побочного эффекта повышает производительность расчёта на 25–30% (с учётом времени самой сортировки) за счет группировки пространственно-близких путей.

СПИСОК ЛИТЕРАТУРЫ

1. *Kajiya J.T.* The rendering equation. SIGGRAPH ’86 Proceedings of the 13th annual conference on Computer graphics and interactive techniques. 1986, pp. 143–150.

2. *Veach E., Leonidas J.G.* Metropolis Light Transport. SIGGRAPH’97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques. pp. 65–76.

3. *Kelemen C., Szirmay-Kalos L., Antal G., Csonka F.* A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm. EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel. 2002, v. 21, no. 3.

4. *Charles J.G.* Introduction to Markov Chain Monte Carlo. A chapter of the book “Handbook of Markov Chain Monte Carlo”. Edited by S. Brooks, A. Gelman, G.L. Jones, X.-L. Meng.

5. *Veach E.* Robust monte carlo methods for light transport simulation. Ph.D. dissertation, Stanford University, December 1997.

6. *Vorba J.* Bidirectional Photon Mapping. Charles University, Prague, 2011.

7. *Georgiev I., Krivanek J., Davidovic T., Slusallek P.* Light Transport Simulation with Vertex Connection and Merging. ACM Trans. Graph. (SIGGRAPH Asia) 2012, v. 31, no. 6, article № 192.

8. *Hachisuka T., Kaplanyan A.S., Dachsbacher C.* Multiplexed Metropolis Light Transport. ACM Transactions on Graphics (TOG) — Proceedings of ACM SIGGRAPH 2014.

9. *Davidovic T., Krivanek J., Hasan M., Slusallek P.* Progressive Light Transport Simulation on the GPU: Survey and Improvements. ACM Trans. Graph., 2014, v. 33, no. 3, article № 29.

10. *van Antwerpen D.* Unbiased physically based rendering on the GPU. PhD thesis. 2010.

11. *van Antwerpen D.* Improving SIMD Efficiency for Parallel Monte Carlo Light Transport on the GPU.

HPG '11 Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics.

12. *Liu J.S., Liang F., Wong W.H.* The Multiple-Try Method and Local Optimization in Metropolis Sampling. J. Am. Statist. Ass., 95:121:134 (2000).

13. *Tjelmeland H.*. Using all Metropolis Hastings proposals to estimate mean values. Statistics No. 4, Department of Mathematical Sciences, Norwegian University of Science and Technology, Trondheim, Norwway (2004).

14. *Stormark K.*. Multiple Proposal Strategies for Markov Chain Monte Carlo. Master of Science in Physics and Mathematics, 2006

15. *Segovia B., Iehl J.C., Peroche B.* Coherent Metropolis Light Transport with Multiple-Try Mutations. Soumis a Eurographics Symposium on Rendering 2007.

16. *Schmidt M., Lobachev O., Guthe M.* Coherent Metropolis Light Transport on the GPU using Speculative Mutations. Journal of WSCG. 2016, v. 24, no. 1, pp. 1–8.

17. *Harris M., Sengupta S., Owens J.D.* Parallel Prefix Sum (Scan) with CUDA. GPU Gems 3. Second printing, December 2007.

18. *Suhorukov I.* OpenCL 1.1: Atomic operations on floating point values. 2011. http://suhorukov.blogspot.ru/2011/12/opencl-11-atomic-operations-on-floating.html

19. *Morton G.M.* A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing, Technical Report, Ottawa, Canada: IBM Ltd. 1966.

20. Hydra Renderer. Плагин для рендера реалистичных изображений в 3D Studio Max при помощи GPU. http://www.raytracing.ru/