Исследование особенностей Metropolis Light Transport на GPU *

Фролов В.А.[1,2]***, Галактионов В.А.***[1]
vfrolov@graphics.cs.msu.ru|vlgal@gin.keldysh.ru
[1]Институт прикладной математики им.М.В.Келдыша, Москва, Россия;
[2]Московский государственный университет им.М.В.Ломоносова

В данной работе исследованы неочевидные особенности, возникающие при параллельной реализации алгоритма 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»). Исследуется различные способы экономии памяти, в т.ч. применение Multiple-Proposal Metropolis Light Transport.

Ключевые слова: *Metropolis Light Transport, GPU, Markov Chain Monte Carlo, SIMD, OpenCL*

1. Введение

Можно сказать, что трассировка путей является применением «обычного 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 выбрасывает огромное количество информации, которую можно было бы использовать.

В свою очередь, 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 PT vs BDPT

В нашей работы мы исследовали однонаправленный MLT по нескольким причинам. Во-первых, MLT для двунаппрвленной трассировки путей (BDPT) увеличивает число путей, в которых существенный вклад будет только от одной стратегии, что в свою очередь снижает эффективность самого BDPT [5]. Во-вторых, BDPT достаточно затратно (по памяти) и сложно реализовыать на GPU, т.к. необходимо не только хранить вершины для соединений (Light Vertex Cache) [6], но и определённым образом хранить и накапливать MIS веса (Recursive MIS [7]). Аналолично, веса нужно вычислять и для Multiplexed MLT [5]. Наконец, алгоритм Метрополиса обычно применяется в ситуациях, когда трудно или невозможно подобрать хорошую стратегию генерации выборок. BDPT, с другой стороны, уже умеют генерировать хорошие стратегии для большинства пикселей. Применение к нему MLT не даёт столь существенного преимущества как применение MLT к однонаправленной трассировки путей. Следовательно, исследование собственно свойств MLT будет менее показательным.

2. Предыдущие работы

2.1 Metropolis Light Transport на GPU

В работе [8] 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 [8] 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 [8] была очень большой - 50%. Такое значение объясняется тем, что при параллельной реализации MLT даёт очень большое «начальное смещение» и, чтобы оно было не столь сильно заметно, автор [8] 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) и, в значительной мере просто «забивает» MLT обычной двунаправленной трассировкой путей.

2.2 Множественные предложения

Существуют 2 известных способа реализации MCMC с множественными предложениями переходов - «Kingpin» [9] и «Theater» [10], [11]. В работе [12] было предложено использовать MCMC с несколькими предложениями переходов (Multiple Try Metropolis Light Transport, MTMLT) для повышения когерентности лучей при реализации трас-

Работа выполнена при финансовой поддержке РФФИ, гранты 16-31-60048 и 16-01-00552;

сировки на CPU с использованием SIMD инструкций. Для учёта множественных переходов был использован «Kingpin» метод из работы [9]. 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 [13] была предложена иная, спекулятивно-рекурсивная схема вычисления переходов для повышения когерентности выполнения потоков на GPU и вычисления всех предложений переходов параллельно. В работе [13] также указывается на тот факт (это вторая причина), что каждое предложение в «Kingpin» вносит вклад $\frac{1}{k}$, что дополнительно будет увеличивать «начальное смещение» на GPU (мы дадим объяснение этому эффекту в следующем разделе). Идея работы [13] заключается в том, чтобы спекулятивно вычислить бинарное дерево всех возможных переходов, до последнего момента откладывая фактическое выполнение перехода. У подхода, реализованного в работе [13], 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 [8] и, таким образом уменьшить объем необходимой памяти в N раз, где N - число параллельных предложений переходов. Метод из работы [13] не позволяет воспользоваться этим потенциальным преимуществом.

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.

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 [8]:

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 3, верхний ряд).

3. Предлагаемые методы

3.1 Прожиг при помощи ОМС

В работах [2] и [3] отмечено, что один из способов снизить «начальное смещение» заключается в том, чтобы выбирать начальные точки («seeds») для MCMC не равномерно-случайно а пропорционально сэмплируемой функции $F(x)$. Однако, эта идея не была использована ранее в работах, реализующих MLT на GPU [8], [13].

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 ОМС 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 2 шаге был реализован через параллельное вычисления префиксной суммы на GPU с последующим бинарным поиском. 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 нормализа-

ции [2]. Поэтому в некотором смысле (кроме случая когда используется смешивание PT/BDPT и MLT аналогично [3]) 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 [8]) до 5-10% в нашей работе и, таким образом, повысить эффективность собственно MLT. Результат применения прожига представлен на рис. 1.

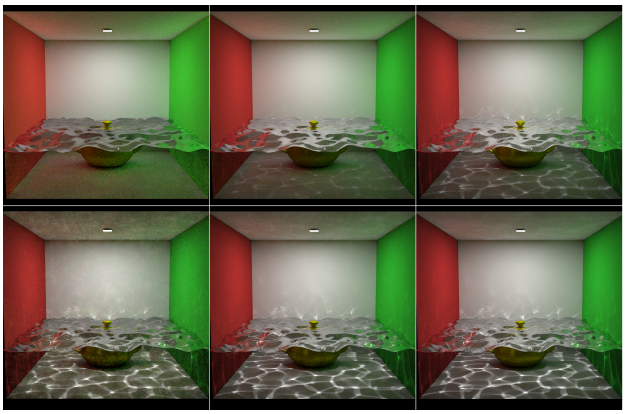

Рис. 1. MLT без предложенного метода прожига (верхний ряд) и с предложенным методом (нижний ряд) за 10, 100 и 500 секунд на AMD R9200.

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 их загрузки вычислениями достаточно меньше потоков. При критической нехватке памяти, в 2-4 раза число потоков почти всегда можно уменьшить (рис 2). Причём, при уменьшении количества потоков дополнительно снижается «начальное смещение», поскольку оставшиеся потоки делают шаги быстрее.

2. Уменьшение битности хранимых чисел. Поскольку на практике MLT обычно хранит случайные числа в интервале от 0 до 1 (Primary Space Sample MLT), нет необходимости в использовании 32 бит на 1 число с плавающей точкой. Достаточно хранить только мантису в 23 (или 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, мы хранили 4 числа с точностью в 24 бита, и ещё 4 числа с точностью в 16 бит. Итого - 160 бит (5x32 бита) и 8 случайных чисел в среднем на 1 отскок.

3. Использование ленивой схемы вычисления предложений, аналогично работе [8]. Мы расширили эту идею на Multiple Proposal MCMC (см. след. раздел). Таким образом, для N потоков и k предложений мы храним N/k векторов для N/k цепей.

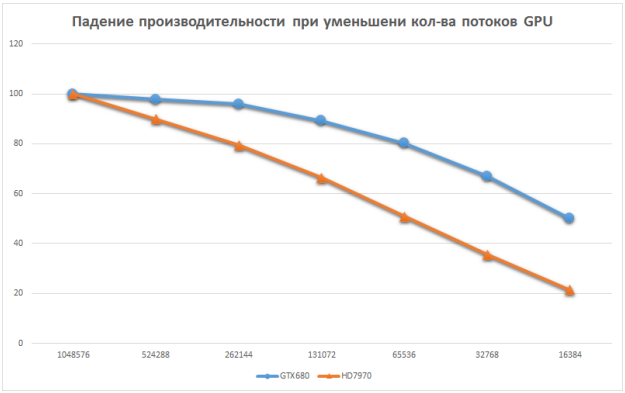

Рис. 2. Падение производительности трассировки на различных GPU при уменьшении числа потоков в процентах. Характер кривых сохранялся для всех тестовых сцен (рис 5). Для увеличения наглядности продемонстрированы только кривые с 1 сцены (CornellWater).

3.3 MLT с расщеплённым переходом

В целях экономии памяти и дополнительного уменьшения «начального смещения» 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, преложенный алгоритм преобразует начальное смещение в шум (рис.

3, 4), и может быть использован для повышения степени паралелизма - когда на нескольких GPU считается большое количество относительно-коротких цепей. Недостатком алгоритма является ухудшение сходимости (рис. 3).

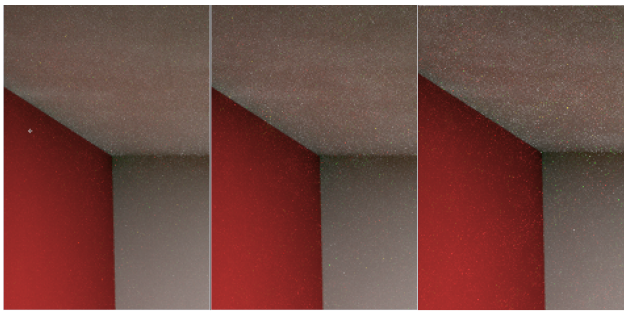

Рис. 3. Справа налево - фрагмент изображения для 1, 2 и 4 попыток за одно и то же время (10 минут). Несмотря на явно увеличивающийся уровень шума, фактическая ошибка (разница с эталоном) для всего изображения при увеличении числа попыток уменьшается (рис.4). Это происходит из-за наличия «начального смещения» в изображениях. При увеличении числа попыток смещение переходит в шум, который более заметен для глаза.

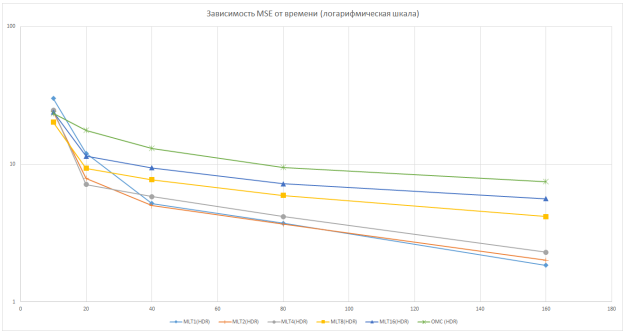

Рис. 4. Зависимость ошибки (MSE) от времени для MLT с разделённым переходом с различным числом попыток. Эксперимент проведён на GTX670 для сцены CornellWater (рис. 3). 10 мин. соответствуют 1200 шагам цепей для 524K потоков. Посчитано для HDR изображений. Характер ошибки сохраняется и на других тестовых сценах (рис. 5).

4. Выводы

Таким образом, по сравнению с существующими реализациями MLT на GPU, предлагаемый метод «прожига» впервые позволил амортизировать проблему «начального смещения» и получить приемлимую оценку изображения в начале расчёта на GPU (рис 1), 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

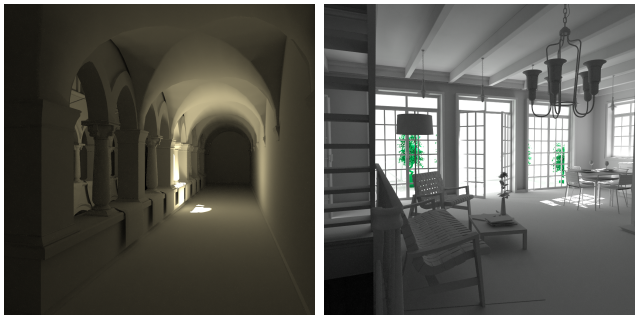

Рис. 5. Дополнительные тестовые сцены.

случайных чисел (128x32 бит на 1 поток) занял 128 MB.

Литература

[1] *Kajiya, James T.* The rendering equation // SIGGRAPH '86 Proceedings of the 13th annual conference on Computer graphics and interactive techniques. Pages 143-150. 1986.

[2] *Eric Veach, Leonidas J. Guibas.* Metropolis Light Transport // SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques. Pages 65-76.

[3] *Csaba Kelemen, László Szirmay-Kalos, György Antal and Ferenc Csonka.* A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm // EUROGRAPHICS 2002

[4] *Charles J. Geyer.* Introduction to Markov Chain Monte Carlo // A famous book chapter

[5] *Toshiya Hachisuka, Anton S. Kaplanyan, Carsten Dachsbacher.* Multiplexed Metropolis Light Transport // Proceedings of ACM SIGGRAPH 2014

[6] *Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, and Philipp Slusallek.* Progressive Light Transport Simulation on the GPU: Survey and Improvements. // ACM Trans. Graph. (2014)

[7] *Dietger van Antwerpen.* Unbiased physically based rendering on the GPU // PhD thesis. 2010.

[8] *Dietger van Antwerpen.* Improving SIMD Efficiency for Parallel Monte Carlo Light Transport on the GPU // Proceedings of High Performance Graphics. 2011.

[9] *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).

[10] *Hakon Tjelmeland* Using all Metropolis Hastings proposals to estimate mean values // Statistics No. 4, Trondheim, Norwway (2004).

[11] *Kristian Stormark* Multiple Proposal Strategies for Markov Chain Monte Carlo // Master of Science in Physics and Mathematics, 2006

[12] *B. Segovia, J.C. Iehl, B. Péroche* Coherent Metropolis Light Transport with Multiple-Try Mutations // Soumis à Eurographics Symposium on Rendering 2007.

[13] *Martin Schmidt, Oleg Lobachev, Michael Guthe* Coherent Metropolis Light Transport on the GPU using Speculative Mutations // WSCG 2016