ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 267 за 2018 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Фролов В.А.

Исследование алгоритма Multiplexed Metropolis Light Transport на графических процессорах

Рекомендуемая форма библиографической ссылки: Фролов В.А. Исследование алгоритма Multiplexed Metropolis Light Transport на графических процессорах // Препринты ИПМ им. М.В.Келдыша. 2018. № 267. 47 с. doi:10.20948/prepr-2018-267
URL: http://library.keldysh.ru/preprint.asp?id=2018-267

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М. В. КЕЛДЫША
Р о с с и й с к о й а к а д е м и и н а у к

В. А. Фролов

Исследование алгоритма Multiplexed Metropolis Light Transport на графических процессорах

Москва — 2018

В. А. Фролов

Исследование алгоритма Multiplexed Metropolis Light Transport на графических процессорах.

В данной работе впервые проведено исследование GPU-реализации алгоритма Multiplexed Metropolis Light Transport (MMLT) на репрезентативном для прикладных задач фотореалистичного рендеринга наборе 3D-сцен. Выявлены неизвестные ранее особенности и проблемы алгоритма как метода интегрирования освещённости, проявляющиеся при его реализации на массивно-параллельных процессорах и не проявляющиеся при обычной реализации на многоядерных центральных процессорах. Предложены способы решения некоторых из этих проблем. Представленное в статье сравнение с зарубежным GPU аналогом Octane показывает преимущества разработанной системы на архитектурных сценах для трудновычислимых феноменов освещённости.

Ключевые слова: Metropolis Light Transport, Монте-Карло трассировка.

Frolov Vladimir

Investigation of Multiplexed Metropolis Light Transport on GPUs.

In this paper a study of Multiplexed Metropolis Light Transport (MMLT) algorithm implementation on GPUs is provided. It is carried out on a set of 3D scenes representative for photorealistic rendering applications. The previously unknown features and problems of the algorithm were discovered and some of them were solved. The comparison with GPU analogue Octane shows the advantages of the developed system in typical architectural scenes for hard sampling illumination phenomena.

Key words: Metropolis Light Transport, Markov Chain Monte Carlo.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект 16-31-60048-мол_а_дк.

Оглавление

Введение

Интегрирование освещённости методами Монте-Карло-трассировки (рис. 1) является де-факто стандартным методом точного расчёта освещённости на сегодняшний день, поскольку используется практически во всех программах рендеринга, и постепенно проникает в область интерактивной визуализации. Об этом косвенно свидетельствует вышедшая недавно технология аппаратной поддержки трассировки лучей Nvidia RTX.

Рис. 1. Изображение, полученное при помощи Монте-Карло-трассировки путей в разработанном программном обеспечении Hydra Renderer [7]. Высокий уровень реализма обеспечивается как моделированием большого количества разнообразных эффектов взаимодействия света с материалами окружающего мира, так и высокой точностью самого расчёта.

Трудновычислимые феномены освещённости, с другой стороны, представляют собой отдельную научную проблему для методов Монте-Карло трассировки, поскольку делают точный расчёт *неприемлемо долгим*. Последнее, в свою очередь, происходит из-за того, что трудновычислимые феномены освещённости приводят к трудновычислимым многомерным интегралам.

Развитие методов вычисления таких интегралов шло в основном двумя путями: (1) на основе обычного Монте-Карло (Ordinary Monte Carlo, OMC)

и (2) на основе Монте-Карло по схеме марковских цепей (Markov Chain Monte Carlo, MCMC). Обе группы методов в настоящее время успешно применяются для вычисления интеграла освещённости и решения так называемого уравнения рендеринга [27]. Однако методы на основе Markov Chain Monte Carlo показывают себя лучше именно для трудновычислимых интегралов (и не только в области рендеринга).

Multiplexed Metropolis Light Transport [21] — один из лучших MCMC-методов сам по себе, и он одновременно является основой для многих более передовых методов. Поэтому его эффективная реализация на графических процессорах представляет собой важную проблему.

1. Обзор предыдущих работ

1.1. Методы на основе ОМС. Алгоритмы на основе обыкновенного Монте-Карло, способные справляться с трудновычислимыми феноменами освещённости (такие методы называются *устойчивыми* [44]), можно условно разделить на два больших класса.

К *первому классу* относится метод **двунаправленной трассировки путей** (Bidirectional Path Tracing, BPT) [44] и его модификации. Двунаправленная трассировка путей комбинирует вклады от прямой и обратной трассировки при помощи **многократной выборки по значимости** (Multiple Importance Sampling, MIS), что позволяет построить хорошие стратегии сэмплирования (способы генерации Монте-Карло выборок) для многих феноменов освещённости. Однако метод двунаправленной трассировки путей, как и идея многократной выборки по значимости, имеет существенный недостаток — чем более устойчив алгоритм, тем он медленнее в среднем. Это происходит из-за того, что среди N стратегий построения выборок лишь некоторые вносят существенный вклад в изображение. Остальные зануляются MIS-весом апостериорно — то есть уже после того, как они были вычислены. При этом сэмплирование многими стратегиями дорого. В BPT это приводит к необходимости вычислять порядка N^2 лучей при глубине трассировки, равной N. Помимо этого, BPT требует значительного объема памяти на один поток, т. к. для вычисления MIS-весов необходимо хранить некоторую информацию о каждой вершине. Это в свою очередь затрудняет применение алгоритма на GPU [11, 13]. При этом BPT по-прежнему не позволяет строить хорошие стратегии сэмплирования для некоторых феноменов освещённости (например, Specular-Diffuse-Specular-пути и многократные glossy-отражения).

Метод **усечённой двунаправленной трассировки путей** [1, 13] (Instant Bidirectional Path Tracing, IBPT) амортизирует многие недостатки BPT и является его частным случаем, комбинируя для любой глубины N только три стратегии. IBPT увеличивает скорость за счёт незначительной потери устойчивости и не требует хранения всех вершин пути в памяти. Однако оставляет нерешённой проблему SDS-путей.

Авторы PCBPT [39] уменьшают общее число соединений в BPT за счёт того, что строят их стохастически (а не детерминированно, как в оригинальном BPT), на основе выборки по значимости при помощи оценки функции потенциальной значимости соединений. Эта функция строится во время прямой трассировки из источника света. Данный метод улучшает скорость в среднем, однако делает шум от многократных glossy-отражений более неравномерным, что визуально хуже воспринимается по сравнению с оригинальным BPT и также не решает проблемму SDS-путей.

В [15] предлагается расширение известного метода фильтрации текстур на основе дифференциалов лучей [24] на двунаправленную трассировку путей и фильтрацию BRDF. Благодаря этому методу достигается значительное повышение устойчивости для многократных glossy-отражений и отражений от материалов с микрорельефом. Тем не менее данный метод (как и рассмотренные ранее IBPT и PCBPT) всё ещё нельзя назвать эффективным для феноменов типа SDS-каустик, о которых речь пойдёт далее.

Ко *второму классу* относятся методы на основе фотонных карт [26], которые уходят от несмещённой оценки решения к смещённой (biased) либо состоятельной (consistent). Среди наиболее существенных работ следует отметить метод SPPM [22], позволяющий получать состоятельную оценку, BDPM [46] для комбинации PT + PM и его аналоги, комбинирующие фотонные карты с PT без многократной выборки по значимости [3, 25]. Фотонные карты на практике имеют два очень существенных недостатка.

1. Меньшая скорость сходимости по сравнению с методами, имеющими несмещённые оценки, — $O(\frac{1}{\sqrt[3]{N}})$ против $O(\frac{1}{\sqrt{N}})$ [29].
2. Наличие сгустков фотонов и дорогая операция сбора освещённости (density estimation).

Эти две проблемы в сумме могут уменьшить скорость на 1–2 порядка из-за необходимости вычислять двунаправленную функцию отражения (ДФО, или BRDF) на каждый фотон (притом что для глянцевых поверхностей большая часть собранных фотонов вносит нулевой вклад, т. к. фотоны падают на поверхность не с нужного направления). Хотя вторую проблему можно решить, используя контроль плотности [43] и сбор только для ламбертовской компонен-

ты освещённости [3, 4], первый недостаток имеет решающий вес. На практике он приводит к тому, что достичь «финального качества» изображения за приемлемое время фотонными картами практически невозможно.

Наконец, *промежуточное* положение между первым и вторым классом занимают методы, которые интегрируют фотонные карты в матаппарат многократной выборки по значимости в BPT — VCM [20], PEA [23], UBPT [30]. При этом многократная выборка по значимости в этих методах не решает проблему дорогой операции сбора, т. к. работает апостериорно, т. е. уже после того, как сбор освещённости был выполнен. Проблема меньшей скорости сходимости также решается не полностью, поскольку для трудновычислимых феноменов освещённости, с которыми не справляется BPT, будет наблюдаться ухудшенная сходимость фотонных карт — $O(\frac{1}{\sqrt[3]{N}})$.

При этом на практике такие методы (например, VCM, реализованный в рендер-системе Corona [41]) являются лишь незначительным улучшением простого смешивания изображений от двух методов (BPT и SPPM) по маске [2]. Это является вполне очевидным следствием механизма работы многократной выборки по значимости — сначала вычислить результат двумя разными способами, и потом скомбинировать оба с весами, которые часто вырождаются в пару (0,1) или (1,0).

Заключение по OMC методам. Несмотря на имеющиеся недостатки, OMC-методы продолжают совершенствоваться и используются в программных пакетах гораздо чаще MCMC-методов (даже для статических изображений). Основная причина — возможность быстро получать приближённое решение и видеть результат (пусть и шумный) практически сразу. Это невозможно в большинстве MCMC методов, т. к. в начале расчёта изображение для них выглядит неправильно из-за наличия так называемого «начального смещения».

Однако для достижения «финального качества» изображения при наличии трудновычислимых феноменов освещённости MCMC-методы выигрывают на порядки (что, в частности, демонстрируется в данной работе).

1.2. Методы на основе MCMC

Классификация MCMC-методов. MCMC методы в рендеринге можно условно разделить на два класса по тому, в каком пространстве они работают. К первому классу относятся методы, работающие в *мировом пространстве путей* (Path Space) — пространстве, составленном конкатенацией всех координат вершин путей в мировом пространстве (например, это Veach MLT [45], MEMLT [49], HSLT [28]). Ко второму классу можно отнести методы, работающие в так

называемом *первичном пространстве путей* (Primary Sample Space) — пространстве всех используемых (Монте-Карло выборкой) случайных чисел, многомерном единичном кубе (например, это Kelemen MLT [31] и MMLT [21]). В практических приложениях, как правило, используются методы второго класса, т. к. они более универсальны по отношению к различным феноменам освещённости. Path Space методы, напротив, более специфичны. Например, MEMLT хорошо работает для путей с SDS-вершинами, а HSLT разработан для многократных glossy-отражений.

Общая схема работы MLT. В противовес методам, основанным на многократной выборке по значимости, Metropolis Light Transport [45] генерирует выборки не пропорционально какой-либо одной части подынтегральной функции (что делает каждая из стратегий в BPT), а пропорционально всему интегралу освещённости как многомерной функции, спроецированной на множество пикселей изображения:

$$F(x, y, r_0, r_1, ...r_n) \xrightarrow{project} F(x, y). \tag{1}$$

Алгоритм Метрополиса автоматически помещает больше выборок в более значимые участки функции F, уменьшая таким образом дисперсию [12].

Multiplexed MLT (MMLT). Необходимо отметить, что многократная выборка по значимости и алгоритм Метрополиса **могут и должны использоваться вместе**. Однако это можно сделать разными способами — Veach MLT [45], Kelemen MLT [31], MMLT [21]. Ключ к успеху MMLT лежит в построении такого пространства интегрирования, в котором алгоритм Метрополиса и многократная выборка по значимости не конкурируют, а усиливают друг друга.

Для того чтобы комбинировать алгоритм Метрополиса и многократную выборку по значимости более эффективно, в MMLT [21] строится т. н. «мультиплексированное» пространство интегрирования. Это происходит путём добавления двух степеней свободы — (1) глубины трассировки d и (2) стратегии сэмплирования пути s. Далее марковская цепь статистически находит оптимальный способ построения пути в BPT, варьируя параметр s (рис. 2, формула 2). Благодаря этому алгоритм Метрополиса автоматически перераспределяет вычислительные ресурсы таким образом, что малозначимые стратегии и соединения в BPT считаются редко. Причём это происходит в том числе и с учётом функции видимости, т. к. алгоритм Метрополиса строит распределение пропорционально итоговому ответу.

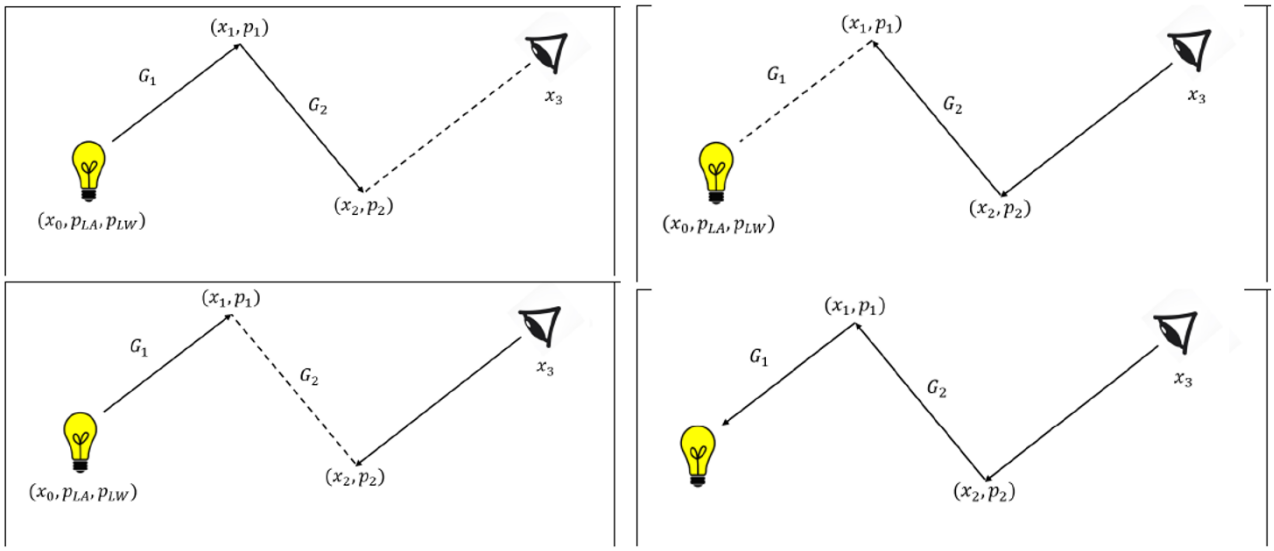


Рис. 2. Пример 4 стратегий для глубины трассировки, равной 3. Пунктирная черта изображает явное соединение вершин. Оптимимальный выбор стратегии происходит автоматически алгоритмом Метрополиса, т. к. функция вклада в MMLT построена в виде взвешенной суммы различных стратегий (формула 2)

$$C(\overline{u}) = \hat{\omega_1}\hat{C_1}^*(\overline{u}) + \hat{\omega_2}\hat{C_2}^*(\overline{u}) + \hat{\omega_3}\hat{C_3}^*(\overline{u}) + \hat{\omega_4}\hat{C_4}^*(\overline{u}). \quad (2)$$

Дальнейшее развитие MMLT. Более передовые методы, как правило, построены поверх MMLT и нацелены на его улучшение или устранение его проблем. Эти проблемы происходят из того, что «мультиплексированное» пространство интегрирования в MMLT, благодаря которому алгоритм Метрополиса и многократная выборка по значимости хорошо работают в связке, само по себе более сложно (т. е. вероятность попадания в существенную область пространства равномерно случайной выборкой в нём меньше), чем первичное пространство путей в Kelemen MLT [31] из-за априорного разбиения («мультиплексирования») путей по глубине и стратегиям. Это, в свою очередь, создаёт две проблемы: (1) низкая эффективность равномерно-случайного сэмплирования и, как следствие, низкая вероятность принятия большого шага и (2) невозможность смены стратегии сэмплирования во время маленького шага.

Низкая эффективность равномерно-случайного сэмплирования в MMLT делает его OMC-аналог (если из MMLT убрать алгоритм Метрополиса и оставить только многократную выборку по значимости) чрезвычайно неэффективным. Это, в свою очередь, делает практически бесполезным метод смешивания OMC и MCMC, используемый в Kelemen MLT [31]. Поэтому для эффективного смешивания MCMC и OMC в работе [9] был предложен механизм смешивания в пространстве изображения.

Вторая проблема (невозможность смены стратегии сэмплирования во время маленького шага) служит источником дополнительных артефактов на изображении, поскольку отдельные участки изображения (где MIS-веса далеки от нуля и единицы) строятся разными стратегиями (и зачастую, как следствие, разными цепями Маркова). Для решения этой проблемы в работах [16, 37, 40] были разработаны гибриды MMLT с методами, работающими в пространстве путей ([45], [49], [28]). В этих работах ([16, 37, 40]) строится обратимый переход между мировым пространством путей и первичным пространством путей, за счёт чего удаётся добиться смены стратегии сэмплирования во время *маленького шага*.

Значительным шагом вперёд является метод HHMC [34]. В HHMC используется апроксимация MCMC с гамильтоновой механикой (Hamiltonian Monte Carlo, HMC) [14] для генерации анизотропного предложения перехода в MMLT. К сожалению, даже апроксимация HMC требует использования методов автоматического дифференцирования [35], что оставляет некоторые вопросы практической применимости открытыми. В частности, в [34] отмечаются случаи, когда автоматическое дифференцирование не будет корректно работать, — при наличии высокочастотных карт нормалей или карт смещений (имитация микрорельефа поверхности).

Помимо рассмотренных методов, существует ещё один популярный класс MCMC-методов для рендеринга — методы, основанные на отборе популяции сэмплов (поэтому они называются Population Monte Carlo, или PMC) [17, 32]. Основная идея этих методов состоит в том, чтобы запоминать информацию о сэмплах во времени и затем переиспользовать наиболее удачные сэмплы как стартовое состояние для небольших мутаций. Преимущество этих методов заключается в большей интерактивности (т. е. в начале расчёта изображение выглядит более правильно) по сравнению с рассмотренными выше алгоритмами и возможности вычислять в том числе прямой свет. Среди недостатков, как правило, меньшая точность в пределе. Коммерческая система на GPU Octane [38] их использует. Далее, мы проведём сравнение разработанной системы с Octane.

GPU-реализации MLT и MMLT. В настоящее время известных GPU реализаций алгоритмов рендеринга существует достаточно много. Большая часть из них исследует методы повышения когерентности лучей — пространственной близости лучей, обрабатываемых в смежных потоках. Однако Metropolis Light Transport исследуют не так много работ.

В [11] на GPU впервые был реализован метод Kelemen MLT [31] на основе традиционной двунаправленной трассировки путей (BPT). В качестве проблем в [11] отмечены высокие траты памяти (обусловленные двунаправленной

трассировкой путей и необходимостью хранить весь вектор случайных чисел на каждый поток) и высокое значение начального смещения (start-up bias). В качестве основного направления улучшения исследуются в некотором смысле технические приёмы повышения производительности — метод регенерации путей (при помощи уплотнения оставшихся активных потоков на GPU после некоторого количества переотражений) и параллельное вычисление соединений в BPT, что в сумме повышает скорость на 20–30% по сравнению с наивной реализацией.

В [42] впервые сообщается об успешной реализации MMLT на графических процессорах. К сожалению, авторы [42] в очередной раз сконцентрировались на некогерентных лучах и улучшении производительности на 30%. В [42] это достигается путём применения MCMC с множественными предложениями переходов. Предлагаемые методы — Speculative MLT (SMLT) и Rejection Chain MLT (RCMLT), позволяют вычислять несколько предложений переходов в MCMC параллельно, однако ухудшают ошибку в целом. Идея метода RCMLT состоит в том, чтобы вычислить сразу несколько предложений перехода с запасом, — на случай, — если первое предложение будет отвергнуто и можно будет сразу попробовать следующее. SMLT является развитием RCMLT. Сначала параллельно вычисляется дерево всех возможных переходов, после чего производится фактический переход в один из листов.

В [18] для Kelemen MLT впервые было предложено решение проблемы большого начального смещения на GPU при помощи отбора начальных состояний марковских цепей обыкновенным Монте-Карло. В плане оптимизаций авторы [18], в отличие от предыдущих работ, сосредоточились на уменьшении занимаемой памяти GPU и смогли достичь экономии памяти около 1 порядка по сравнению с предыдущими реализациями. Для повышения когерентности лучей в [18] используется сортировка потоков. Это даёт примерно такой же незначительный прирост в 20–30%, как и в предыдущих работах.

Таким образом, в настоящий момент большинство существующих работ сосредоточено на технических моментах реализации Metropolis Light Transport на GPU. Получаемое ускорение в работах [11, 42] хотя и заметно, но малозначимо на практике. Замеченная в [11] и частично решённая в [18] проблема начального смещения, с другой стороны, более интересна, поскольку представляет собой фундаментальное ограничение алгоритма при реализации на массивно-параллельных системах: для MCMC-методов существует большая разница между одним быстрым процессором и большим количеством медленных. При реализации MCMC на CPU в одном или нескольких потоках каждая цепь Маркова даже за очень небольшое время делает несколько миллионов шагов. При реализации MCMC на GPU мы имеем много цепей (от нескольких сотен тысяч до

миллиона), но каждая из них сделает всего лишь несколько тысяч или десятков тысяч шагов за всё время расчёта. Именно это свойство и является причиной большого начального смещения при расчёте на GPU — изображение в начале рендеринга долгое время может выглядеть неправильно, меняя яркость в отдельных участках и приходя к итоговому решению постепенно.

Из-за того что в MMLT более сложное пространство интегрирования, чем в Kelemen MLT, проблема начального смещения здесь требует дополнительного изучения, поскольку большое начальное смещение обуславливается как раз низкой вероятностью попасть случайной выборкой в высокозначимую или хотя бы ненулевую область функции вклада. Кроме того, в настоящий момент отсутствуют сравнения методов рендеринга на GPU на каком-либо более или менее существенном наборе сцен, приближенных к реальным применениям. В данной работе мы решили заполнить эти пробелы, сконцентрировавшись на алгоритмах, для которых возможна эффективная реализация на GPU — PT, IBPT и MMLT.

2. Предложенная реализация на GPU

2.1. Геометрическое ядро (трассировка лучей). Все реализации алгоритмов интегрирования освещённости требуют эффективной реализации операции пересечения луча с геометрией. Мы использовали библиотеку Embree [47] для построения BVH4-дерева. Сама трассировка лучей при этом реализована на GPU в двух вложенных while-while циклах по аналогии с работой [10] и имеет приблизительно 75%-ную производительность указанной работы. Мы не использовали технологию RTX ввиду относительно недавнего её появления. Однако напрямую это не влияет на результаты нашей работы.

2.2. Общая архитектура. В целях уменьшения регистрового давления (и увеличения таким образом занятости GPU) мы использовали паттерн программирования на GPU, называемый «разделённые ядра» [19], при котором алгоритм разбивается на некоторое число как можно более независимых друг от друга OpenCL-ядер, а данные между ядрами явно передаются через глобальную память, используя паттерн хранения SOA (Structure of Arrays). Для PT было использовано 7 различных ядер, для IBPT — 9, для MMLT — 12. При этом наиболее важно было отделить друг от друга 4 части: (1) трассировка лучей, (2) сэмплирование источников, (3) взаимодействие с материалами и вычисление MIS-весов и (4) учёт вклада в изображение (сэмплирование изображения). Такое разделение позволяет разрабатывать различные части системы в относительной независимости друг от друга. Оно же позволяет параллелить выполнение различных этапов при помощи конвейера (что будет использовано далее).

Данный подход отличается от большинства GPU-реализаций, в том числе использующих Nvidia Optix, когда весь алгоритм реализуется в единственном ядре [19, 33].

2.3. Унифицированный механизм сэмплирования изображения. Сэмплирование изображения — это процесс генерации Монте-Карло выборок и учёт итогового вклада этих выборок в изображение. При реализации алгоримов расчёта освещённости на GPU возникают две основные проблемы:

1. каждый поток вносит вклад в случайный пиксел (IBPT и MMLT);
2. изображение занимает много памяти (в больших разрешениях + MMLT удваивает объём занимаемой памяти, т. к. необходимо отдельно хранить первичное и вторичное освещение).

Эти проблемы были решены при помощи унифицированного механизма сэмплирования изображений для всех реализованных алгоритмов [5]. Основная идея этого механизма заключается в конвейеризации вычисления сэмплов и учёта их вклада таким образом, что вычисление значений сэмплов, копирование с GPU на CPU и учёт вклада на CPU выполняются параллельно как разные стадии конвейера. При этом дополнительным преимуществом данного механизма является то, что потоки отделены от пикселей (даже для алгоритма Path Tracing).

2.4. IBPT на GPU. Реализация IBPT в целом (за исключением механизма учёта вклада сэмплов в изображение, описанного выше) соответствует работе [1].

2.5. MMLT на GPU. При реализации алгоритма Multiplexed Metropolis Light Transport на графических процессорах обратим внимание на следующие особенности алгоритма.

1. MMLT использует раздельный набор цепей Маркова для разных глубин (d=3, 4, 5, ..., N). Переход между этими наборами хотя и возможен, но не рекомендуется [21].
2. Каждый путь глубиной d сэмплируется $d + 1$ стратегиями, варьируя параметр разбиения s от 0 до d включительно.
3. Источник сэмплируется только 1 раз (а не каждое переотражение, как происходит, например, в PT).

Из первого пункта автоматически следует необходимость выполнения прожига, в течение которого и будет отобрано нужное количество начальных состояний для каждой глубины d. На данном этапе также производится оценка констант нормализации для каждой глубины (в MMLT на каждую глубину d своя константа нормализации). Прожиг был реализован аналогично [18].

Далее нетрудно заметить, что основной вклад в изображение вносят пути с небольшой глубиной (от 3 до 6). Поскольку во время прожига начальные состояния отбираются пропорционально их вкладу, в итоговом наборе путей с большой глубиной будет мало. В результате наивная реализация, запускающая ядра d раз для всех потоков, будет неэффективной, т. к. большая часть потоков завершится уже после 3–4 отражений.

2.5. Сортировка по глубине. Для того чтобы решить эту проблему, в данной работе была применена сортировка отобранных начальных состояний по глубине d и последующее уменьшение количества активных потоков при каждом отражении (рис. 3). Данная схема позволила повысить производительность от 10% до 45% (таблица 1).

Сцена	Sam/s (без предл. мет.)	Sam/s (с предл. мет.)	Ускорение
scene01	21M	24M	14%
scene02	22M	26M	18%
scene03	7.9M	10M	26%
scene04	18M	20M	11%
scene05	3.1M	4.2M	45%
scene06	6.4M	7.9M	23%
scene07	6.5M	7.8M	20%

Таблица 1. Ускорение расчёта сэмплов при применении предложенной схемы уменьшения числа потоков. Протестировано на Nvidia GTX2070

При этом неожиданным результатом яляется тот факт, что при увеличении геометрической сложности сцены выигрыш от предложенного метода растёт, а не уменьшается. Это свидетельствует об имеющихся проблемах в механизме управления задачами на видеокарте, поскольку «ничего не делающие» потоки могут уменьшать производительность остальных практически на 50%. Одно из типичных объяснений в подобном случае заключается в том, что GPU не может снять блок потоков (мы использовали 256 потоков в блоке), пока все его группы warp (32 потока на GTX2070) не завершат выполнение. Однако в предложенной реализации все активные потоки находятся в начале пула потоков, а все «ничего не делающие» — в конце. Следовательно, число таких «неполных» групп должно быть равно единице. Вариант, при котором неполных групп всё же много, предусматривает случайный порядок выбора потоков из пула на GPU.

Для более плотной загрузки GPU работой в «освободившееся место»

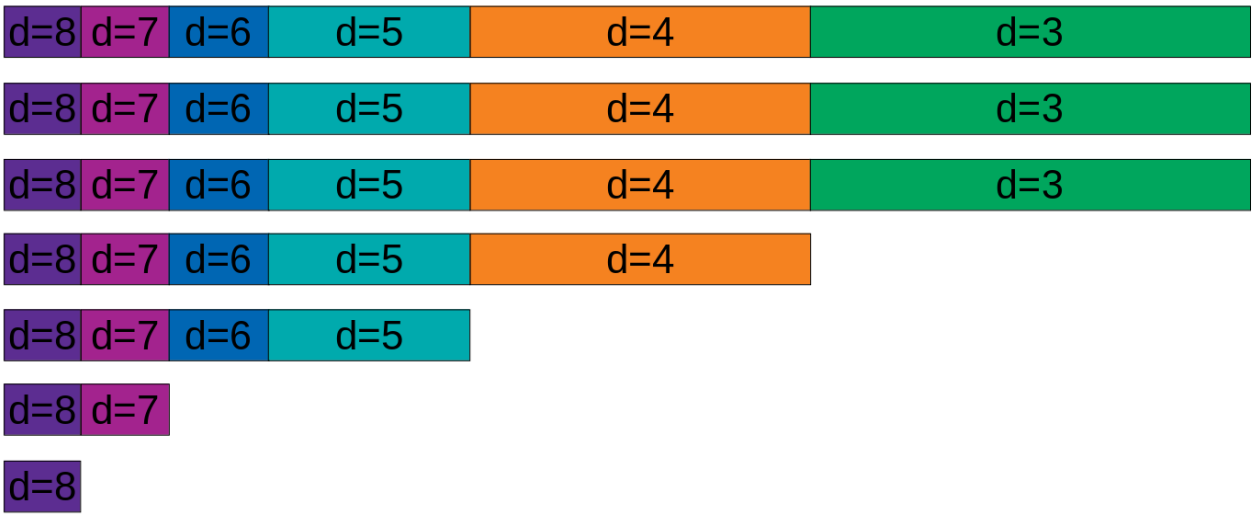


Рис. 3. Иллюстрация схемы уменьшения числа запускаемых потоков. После трёх отскоков с каждым новым отскоком количество запускаемых потоков уменьшается.

(рис. 3) можно было бы вставить новые потоки, применив регенерацию путей [8, 36]. Однако это не было сделано по нескольким причинам. Во-первых, в целях сохранения простоты реализации — чтобы не перемещать данные потоков в памяти и не использовать косвенную адресацию. Во-вторых, по причине того, что регенерация путей даёт ускорение только при очень большой глубине просчёта, которая не требуется в проводимых экспериментах. Наконец, регенерация предполагает необходимость хранения данных для «ожидающих» потоков, что в случае Metropolis Light Transport является критическим недостатком — «ожидающие» потоки будут расходовать значительный объём памяти, поскольку для них нужно будет сохранять состояние марковской цепи (вектор случайных чисел).

2.5. Группировка типа мутации. Multiplexed Metropolis Light Transport реализует четыре типа предложений о переходе (4 типа мутации), используя тот факт, что путь разбит парметром s на две части (от источника и от камеры).

1. *Большой шаг.* Этот тип предложения генерирует новое полностью случайное состояние.
2. *Маленький шаг для всего пути.* Этот тип предложения добавляет небольшие пертурбации ко всем случайным числам, отвечающим за построение пути, кроме параметра разбиения s.
3. *Маленький шаг для части пути от источника.* Этот тип предложения добавляет небольшие пертурбации к случайным числам, отвечающим за путь от источника, и не изменяет построенную от камеры часть пути.
4. *Маленький шаг для части пути от камеры.* Этот тип предложения добав-

ляет небольшие пертурбации к случайным числам, отвечающим за путь от камеры, но не изменяет построенную от источника часть пути.

Такая стратегия позволяет не терять удачную конфигурацию путей при наличии микрорельефа, поскольку часть пути, дающая высокий вклад (например, от источника), будет гарантированно сохранена — даже если маленький шаг на второй части промахивается из-за наличия микрорельефа поверхности. Реализации на CPU обычно стараются не пересчитывать всю часть пути, а переиспользовать посчитанную ранее часть. Например, если была применена мутация для части пути со стороны камеры, не нужно пересчитывать часть пути от источника.

Чтобы применить ту же оптимизацию на GPU, необходимо сгруппировать типы мутаций/предложений переходов для всех потоков: *все потоки* единовременно должны выполнять *один и тот* же тип мутации. Это позволяет не запускать вычислительные ядра для части пути, которую не требуется пересчитывать. При этом на корректность решения данный способ группировки мутаций не влияет, т. к. статистически мы имеем то же самое количество предложений перехода каждого типа. Результаты представлены в таблице 2.

Сцена	Sam/s (без предл. мет.)	Sam/s (с предл. мет.)	Ускорение
scene01	24M	27M	12%
scene02	26M	29M	11%
scene03	10M	13M	30%
scene04	20M	23M	15%
scene05	4.2M	5.7M	36%
scene06	7.9M	9.2M	16%
scene07	7.8M	10M	28%

Таблица 2. Ускорение расчёта сэмплов при применении предложенного метода группировки мутаций. Протестировано на Nvidia GTX2070

2.5. Итоговое ускорение. Применение обоих типов оптимизаций, таким образом, позволяет существенно поднять производительность MMLT на графических процессорах благодаря оптимизации распределения работы на GPU (таблица 3).

Сцена	Sam/s (без предл. мет.)	Sam/s (с предл. мет.)	Ускорение
scene01	21M	27M	28%
scene02	22M	29M	32%
scene03	7.9M	13M	64%
scene04	18M	23M	27%
scene05	3.1M	5.7M	83%
scene06	6.4M	9.2M	43%
scene07	6.5M	10M	54%

Таблица 3. Итоговое ускорение при применении сортировки по глубине и группировки состояний по сравнению с наивной реализацией. Протестировано на Nvidia GTX2070

Хранение вектора случайных чисел и вычисление MIS весов. В предложенной реализации на каждый поток заводятся два дополнительных массива. Первый массив размером $d{*}2{*}sizeof(float)$ (назовём его $Array_{PDF}$) сохраняет прямые и обратные плотности вероятности в площадной мере. Данные плотности вероятности вычисляются во время разных проходов (рис. 4).

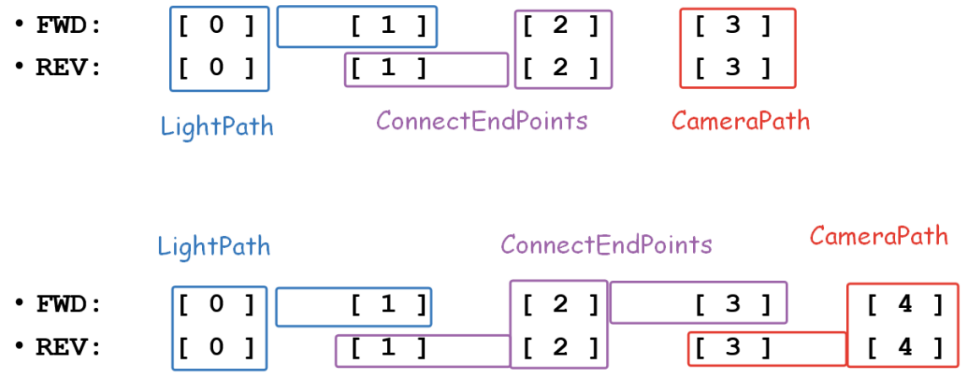


Рис. 4. Иллюстрация заполнения массива $Array_{PDF}$ во время прямой трассировки (LightPath), обратной трассировки (CameraPath) и соединения конечных вершин (ConnectEndPoints).

Второй массив сохраняет состояние марковской цепи (случайные числа, используемые при построении пути) в сжатом представлении по методу из работы [18]. Размер этого массива составляет (12 + d*6)*4 байт. Благодаря тому что источник сэмплируется только один раз, на каждое отражение удаётся компактно хранить случайные числа со следующими характеристиками: 4 числа в фиксированном представлении с точностью в 24 бита, 6 чисел в фиксированном представлении с точностью в 16 бит (последние используются для выбора

слоя в многослойных blend-материалах); 11 зарезервированных чисел используются для сэмплирования позиции на линзе/экране (4 числа) и на источнике (7 чисел). Одно число предназначено для выбора стратегии сэмплирования s.

Таким образом, с учётом обоих массивов при глубине просчёта $d = 10$ и 524K потоков MMLT требует 185 MB, что является относительно небольшим объёмом для современных GPU (от 4 GB до 8 GB).

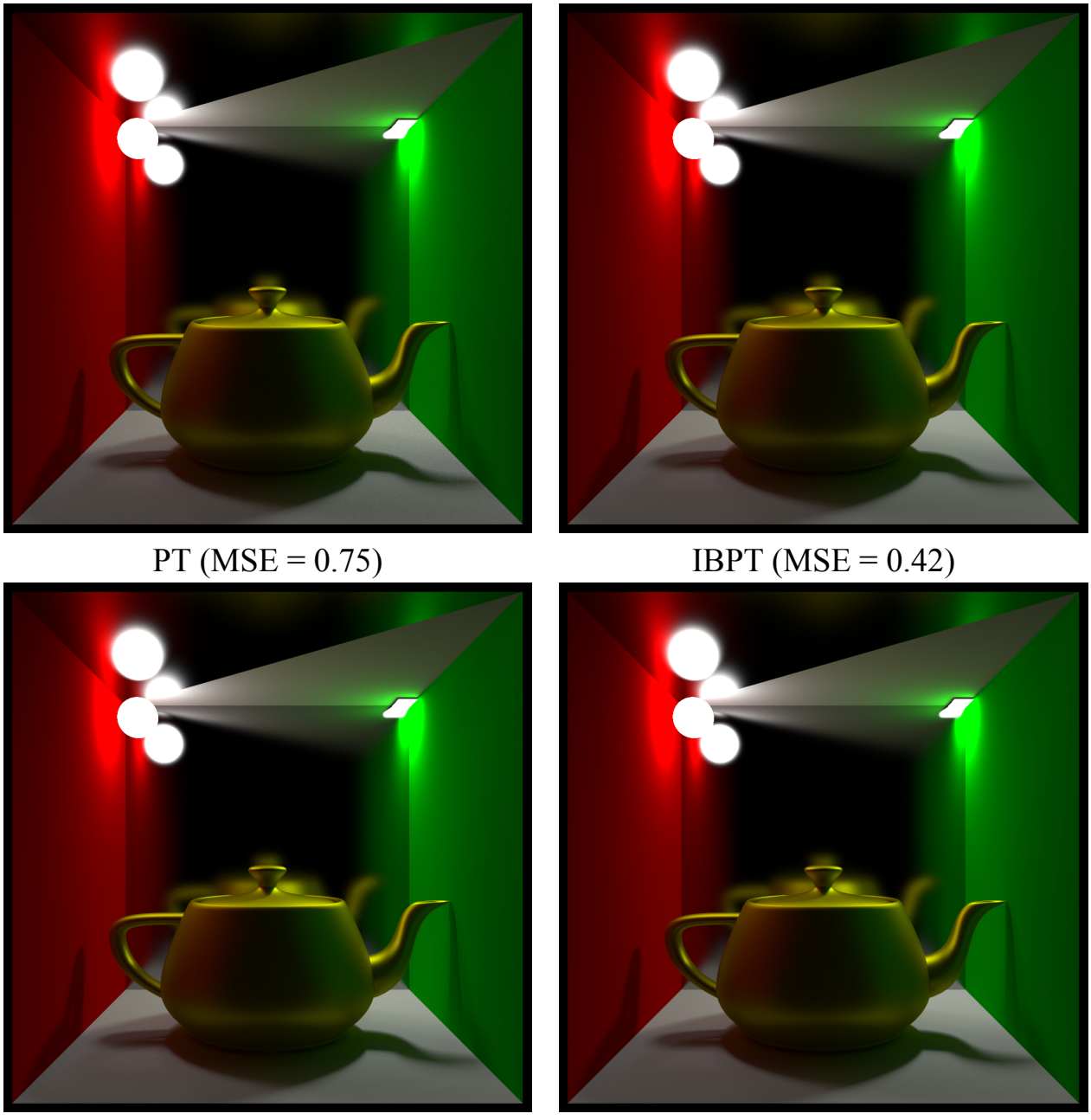
Доступ к массивам организован таким образом, чтобы соседние потоки обращались по соседним адресам в памяти. Для этого используется «транспонированная индексация»: для доступа к i-му элементу массива вычисляется смещение, равное $i * N + threadId$, где N — число потоков.

3. Методика сравнения

Объективное сравнение методов расчёта освещения — нетривиальная задача по многим причинам [6]. Прежде всего здесь имеет значение тот факт, что не все методы (и особенно их реализации) в действительности считают одно и то же. Например, некоторые реализации BPT не вычисляют SDS-пути и, таким образом, позволяют получать изображение с низким уровнем шума (хотя оно и неправильное), а в MLT на GPU всегда присутствует начальное смещение, которое, как правило, незаметно для глаза, но сильно влияет на ошибку. Кроме того, в сторонних реализациях зачастую присутствуют ошибки или неточности, которые невозможно исправить. В результате изображения, полученные разными методами, могут отличаться. Одним из возможных решений в такой ситуации (когда методы дают по какой-либо причине отличающиеся результаты) является измерение ошибки по отношению к собственному эталону [6]. Мы будем использовать данный метод наряду с визуальным сравнением изображений, получаемых разными алгоритмами. Последний способ наиболее часто используется в работах по компьютерной графике.

4. Сравнение алгоритмов

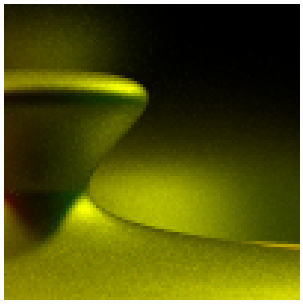
Сравнение алгоритмов проводилось на машине с 2 GPU фирмы AMD (RX580 и W9100), ОС Ubuntu Linux (18.04). Все эталоны были вычислены за 3 часа.



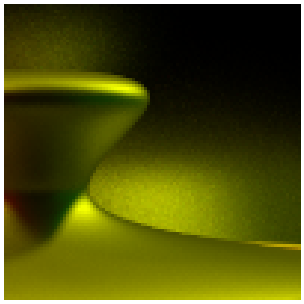
PT (MSE = 0.75) IBPT (MSE = 0.42)

MMLT (MSE = 0.52) Эталон

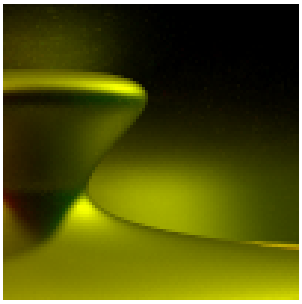
Рис. 5. Сравнение на сцене scene01; 10 минут



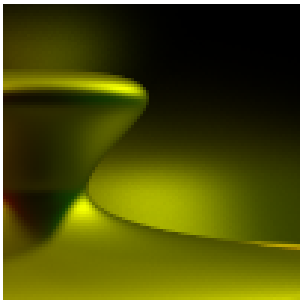
PT
MSE = 11



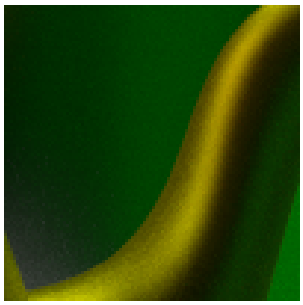
IBPT
MSE = 7.1



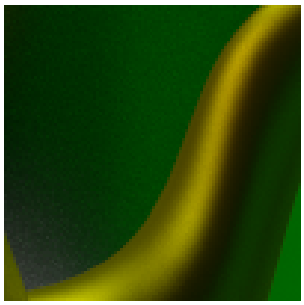
MMLT
MSE = 12



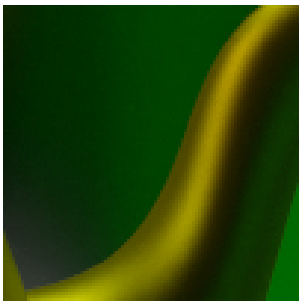
Эталон



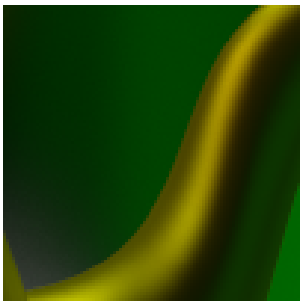
PT
MSE = 8.4



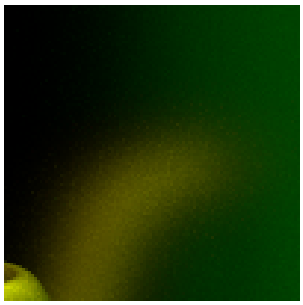
IBPT
MSE = 6.3



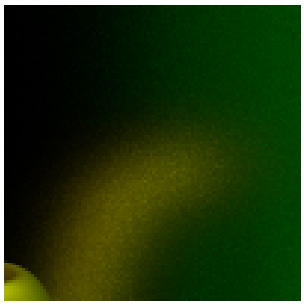
MMLT
MSE = 5.0



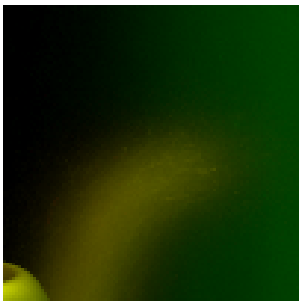
Эталон



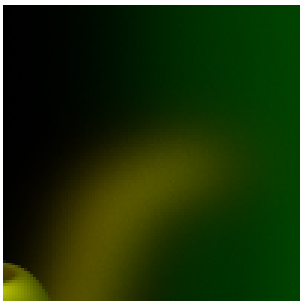
PT
MSE = 7.3



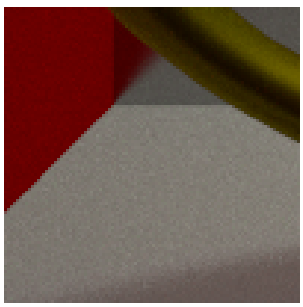
IBPT
MSE = 8.0



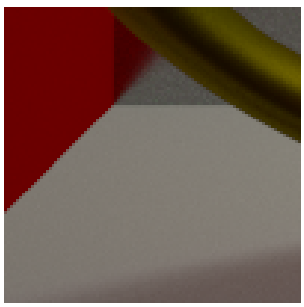
MMLT
MSE = 6.4



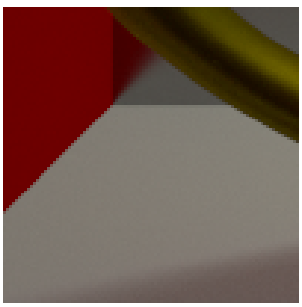
Эталон



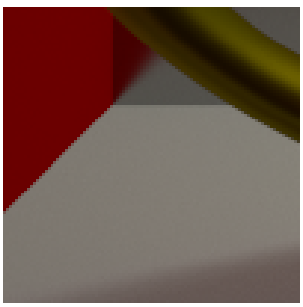
PT
MSE = 12



IBPT
MSE = 4.3

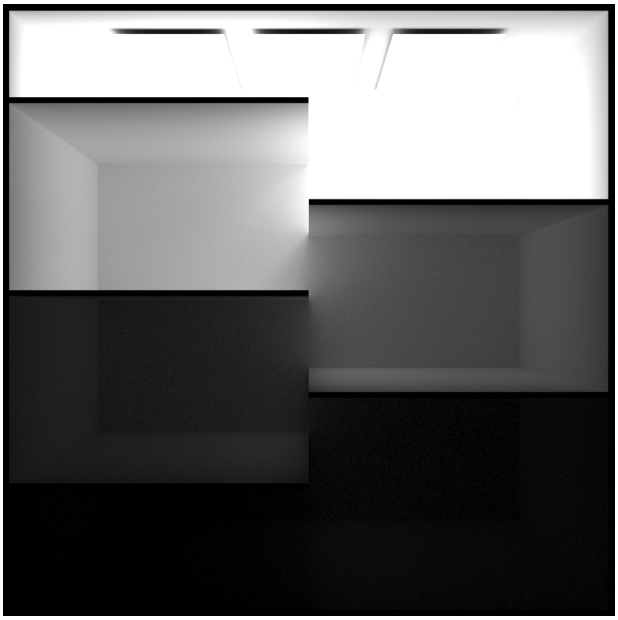


MMLT
MSE = 6.7

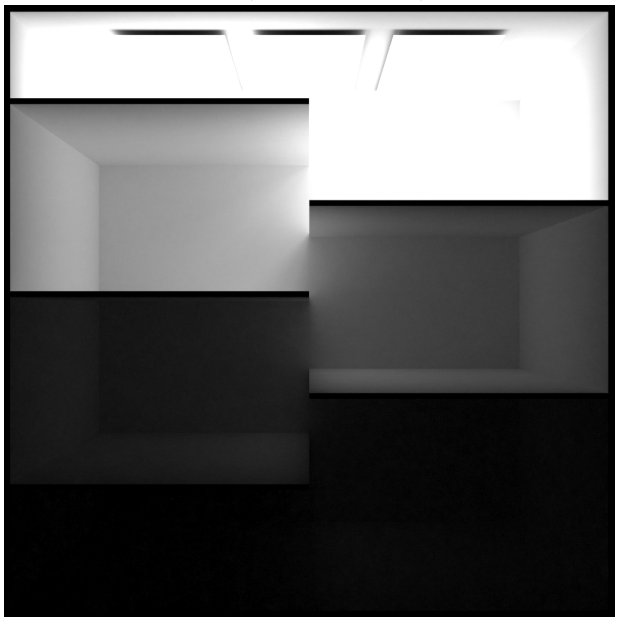


Эталон

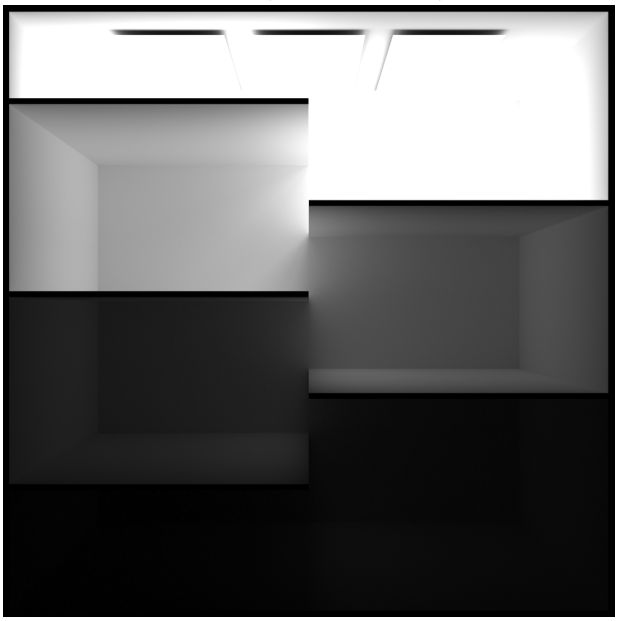
PT (MSE = 1.2)



IBPT (MSE = 1.2)

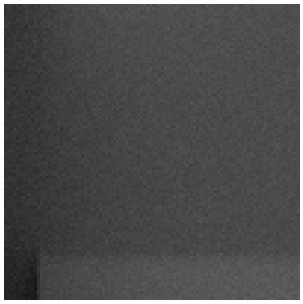


MMLT (MSE = 0.66)

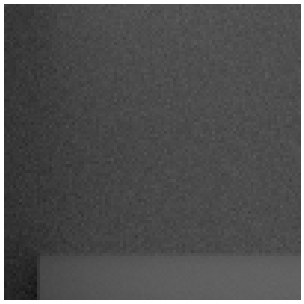


Эталон

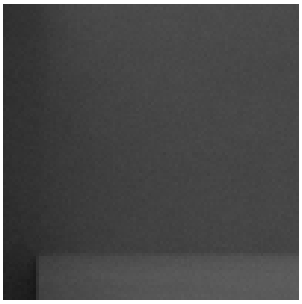
Рис. 6. Сравнение на сцене scene02; 10 минут



PT
MSE = 13

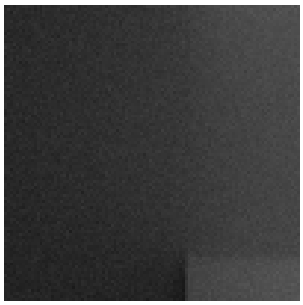


IBPT
MSE = 14

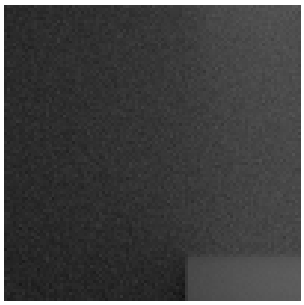


MMLT
MSE = 6.9

Эталон



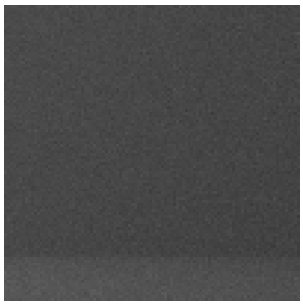
PT
MSE = 16



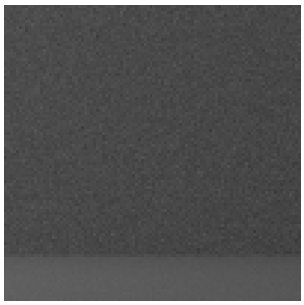
IBPT
MSE = 21

MMLT
MSE = 8.5

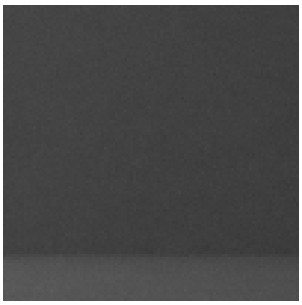
Эталон



PT
MSE = 12

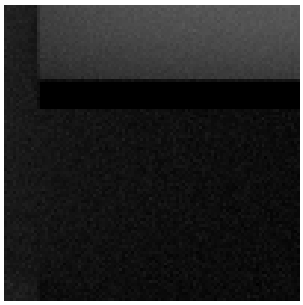


IBPT
MSE = 14

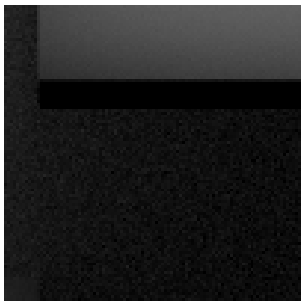


MMLT
MSE = 6.6

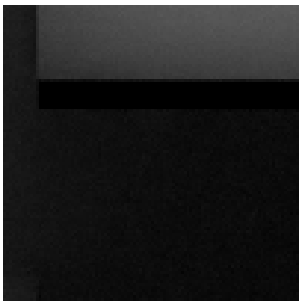
Эталон



PT
MSE = 13



IBPT
MSE = 11



MMLT
MSE = 7.6

Эталон

PT (MSE = 2.1)

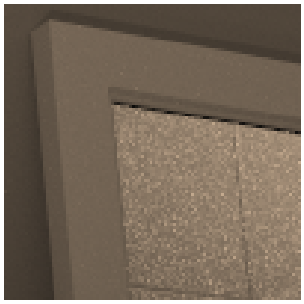
IBPT (MSE = 1.2)

MMLT (MSE = 1.7)

Эталон

Рис. 7. Сравнение на сцене scene03; 30 минут

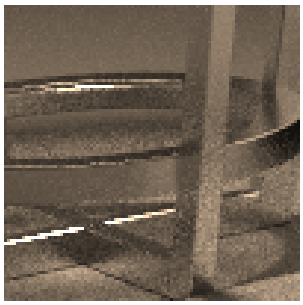
PT
MSE = 28



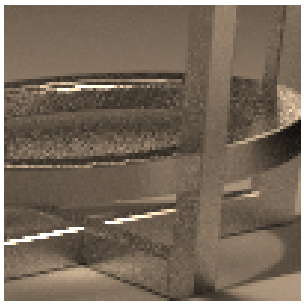
IBPT
MSE = 22

MMLT
MSE = 7

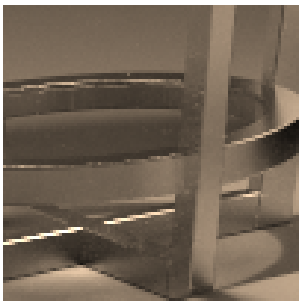
Эталон



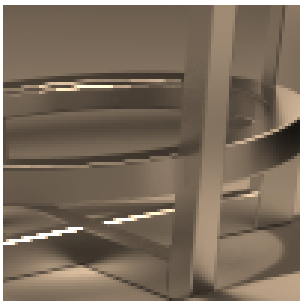
PT
MSE = 29



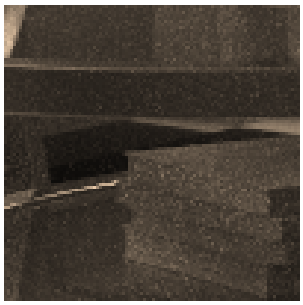
IBPT
MSE = 24



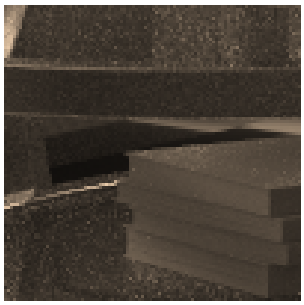
MMLT
MSE = 61



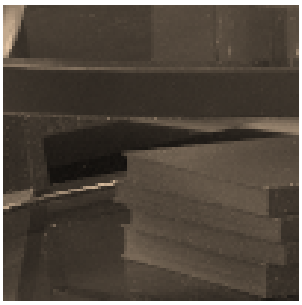
Эталон



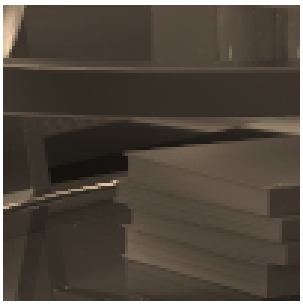
PT
MSE = 24



IBPT
MSE = 22

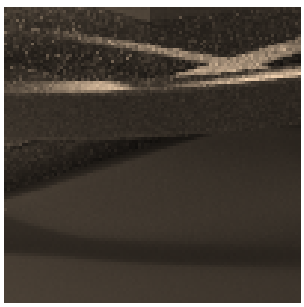


MMLT
MSE = 15



Эталон

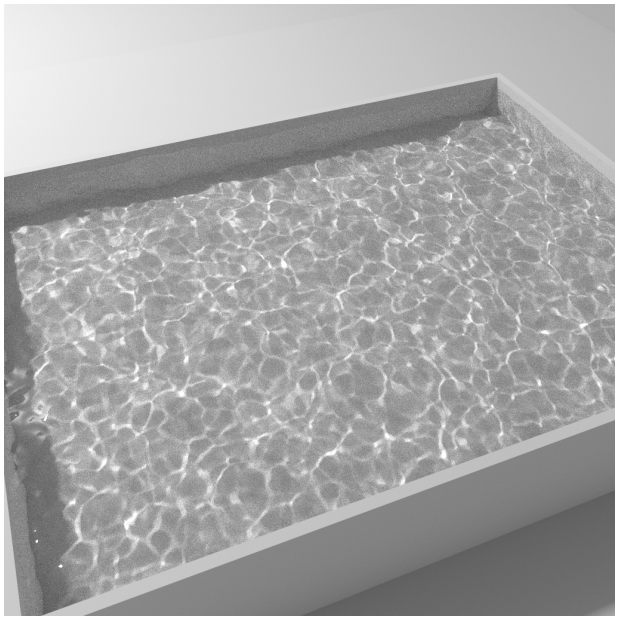
PT
MSE = 26



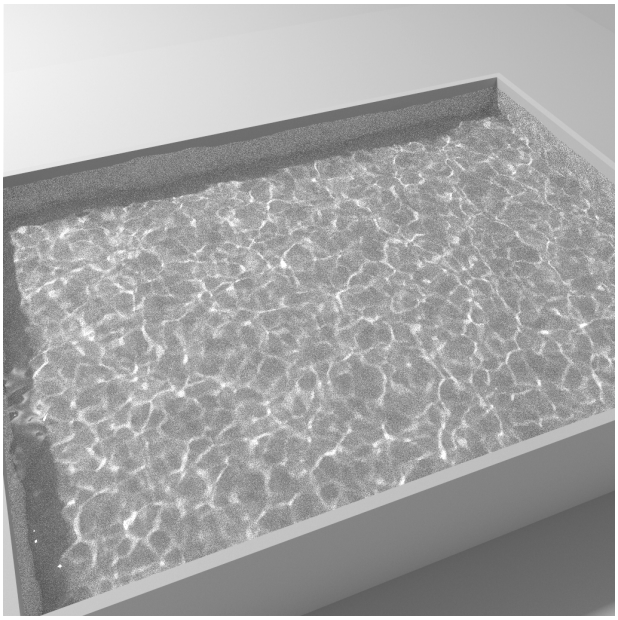
IBPT
MSE = 22

MMLT
MSE = 56

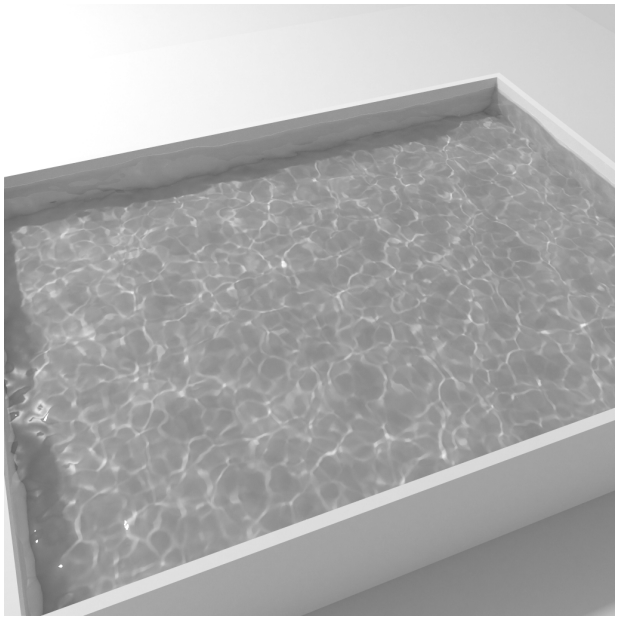
Эталон



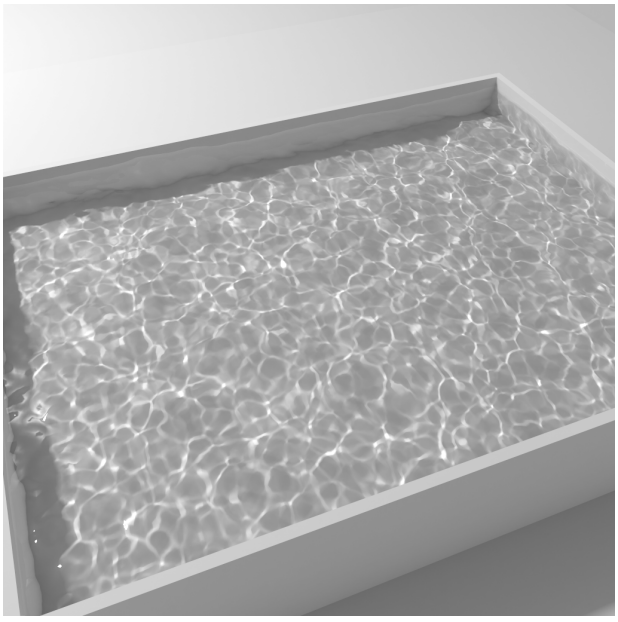
PT (MSE = 3.5)



IBPT (MSE = 5.6)

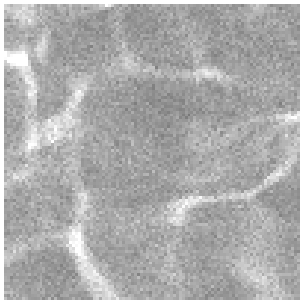


MMLT (MSE = 6.9)

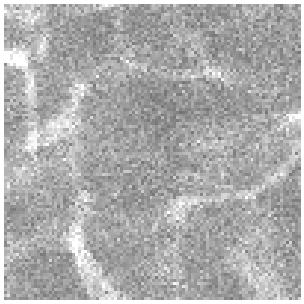


Эталон

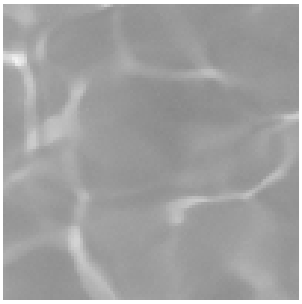
Рис. 8. Сравнение на сцене scene04; 30 минут



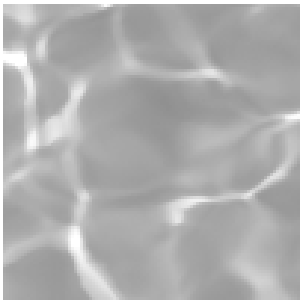
PT
MSE = 45



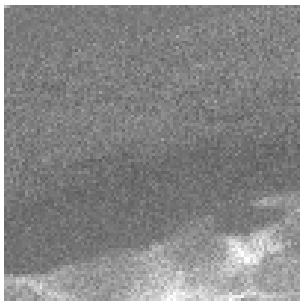
IBPT
MSE = 75



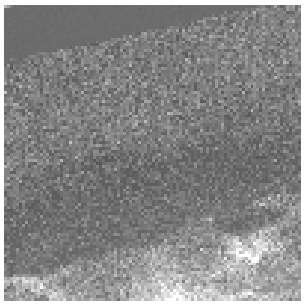
MMLT
MSE = 75



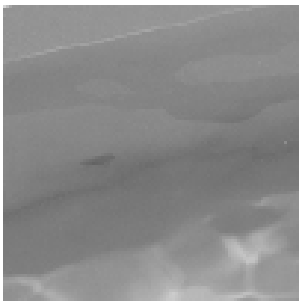
Эталон



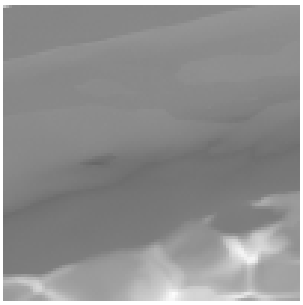
PT
MSE = 40



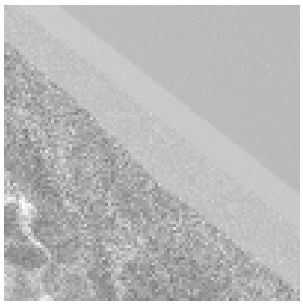
IBPT
MSE = 64



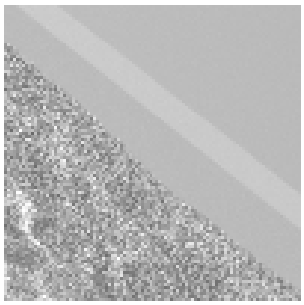
MMLT
MSE = 44



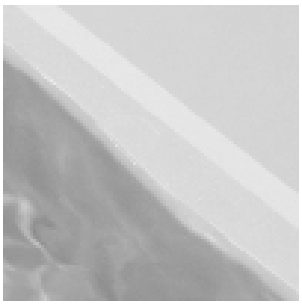
Эталон



PT
MSE = 37

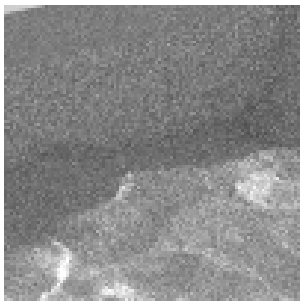


IBPT
MSE = 60

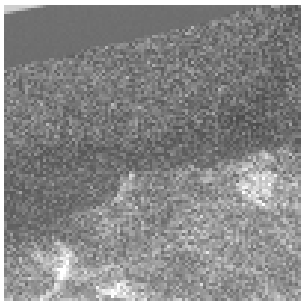


MMLT
MSE = 71

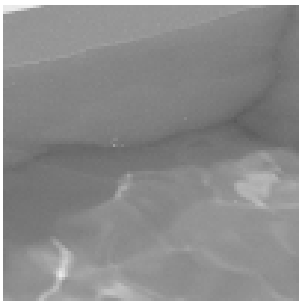
Эталон



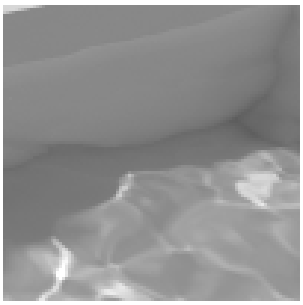
PT
MSE = 42



IBPT
MSE = 66



MMLT
MSE = 46



Эталон

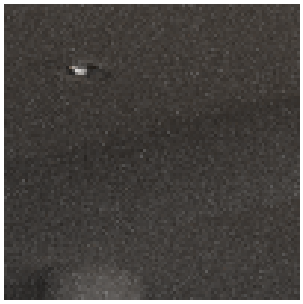
PT (MSE = 2.1)

IBPT (MSE = 2.85)

MMLT (MSE = 2.1)

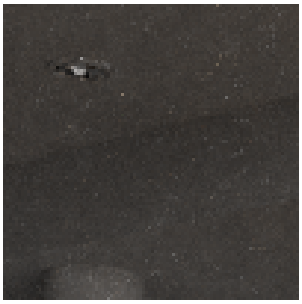
Эталон

Рис. 9. Сравнение на сцене scene05; 30 минут



PT
MSE = 24

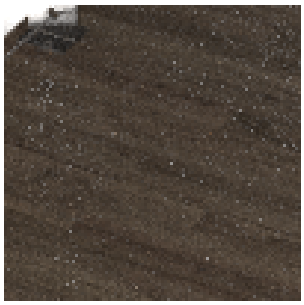
IBPT
MSE = 19



MMLT
MSE = 16

Эталон

PT
MSE = 19



IBPT
MSE = 33

MMLT
MSE = 19

Эталон

PT
MSE = 21

IBPT
MSE = 19

MMLT
MSE = 17

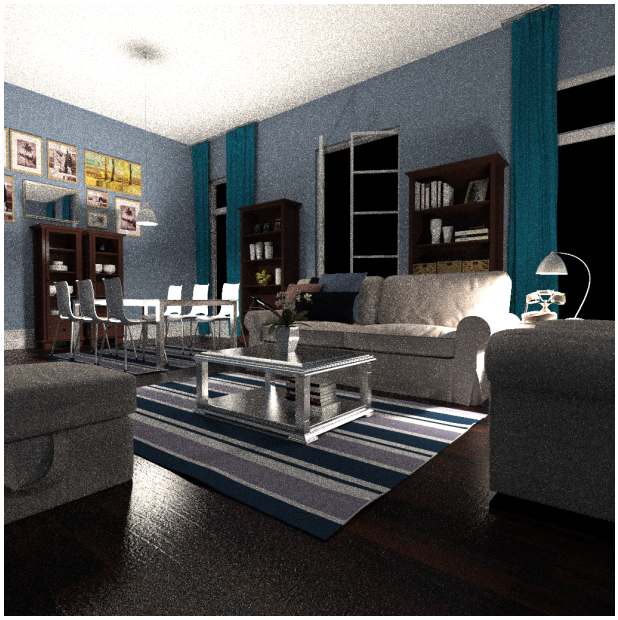
Эталон

PT
MSE = 34

IBPT
MSE = 30

MMLT
MSE = 15

Эталон



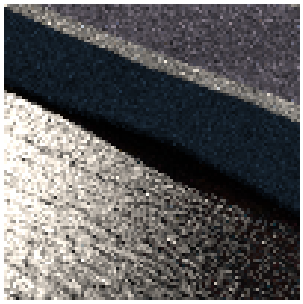
PT (MSE = 2.0)

IBPT (MSE = 2.1)

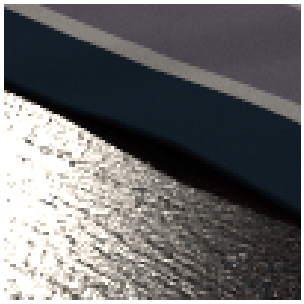
MMLT (MSE = 2.1)

Эталон

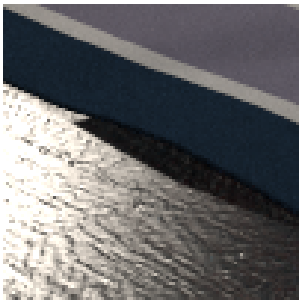
Рис. 10. Сравнение на сцене scene06; 60 минут



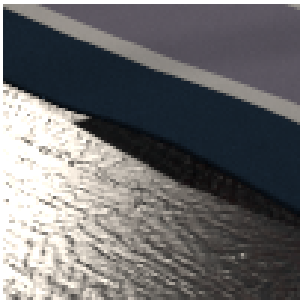
PT
MSE = 79



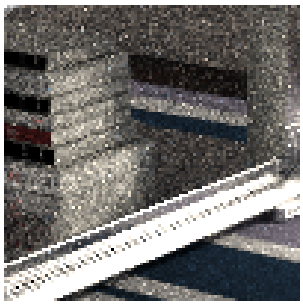
IBPT
MSE = 34



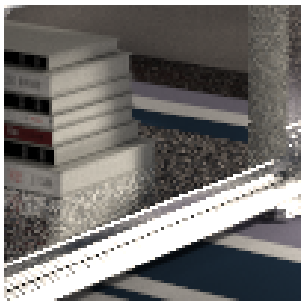
MMLT
MSE = 15



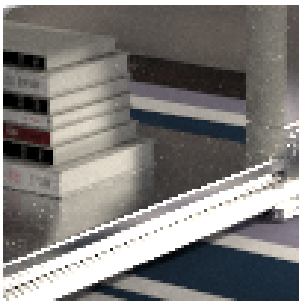
Эталон



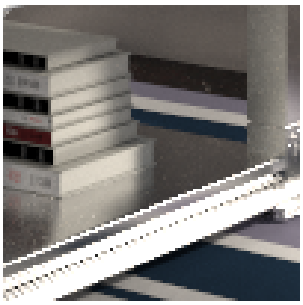
PT
MSE = 73



IBPT
MSE = 29

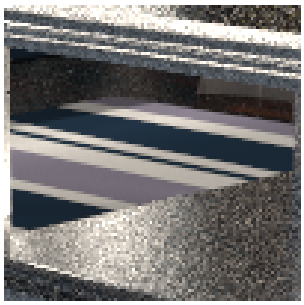


MMLT
MSE = 21

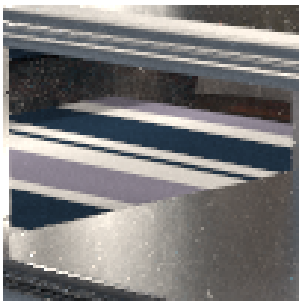


Эталон

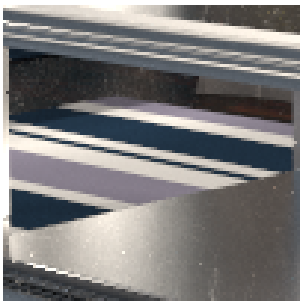
PT
MSE = 88



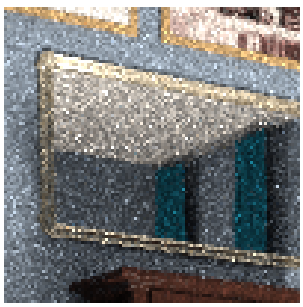
IBPT
MSE = 54



MMLT
MSE = 24



Эталон



PT
MSE = 82

IBPT
MSE = 74

MMLT
MSE = 17

Эталон

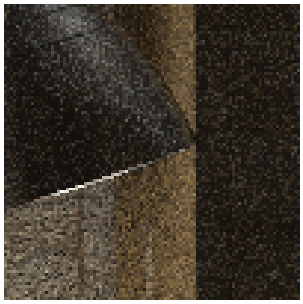
PT (MSE = 4.5)

IBPT (MSE = 1.7)

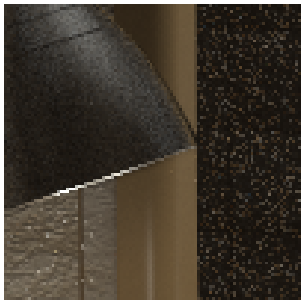
MMLT (MSE = 1.2)

Эталон

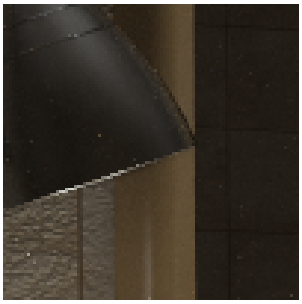
Рис. 11. Сравнение на сцене scene07; 60 минут



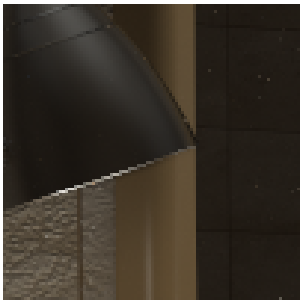
PT
MSE = 48



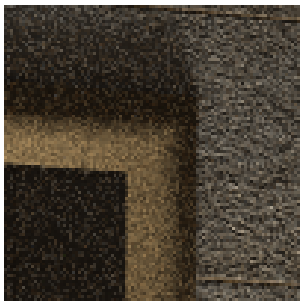
IBPT
MSE = 33



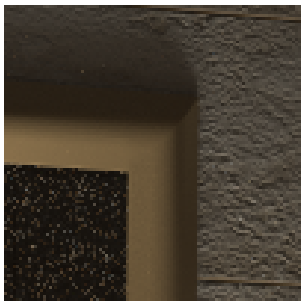
MMLT
MSE = 12



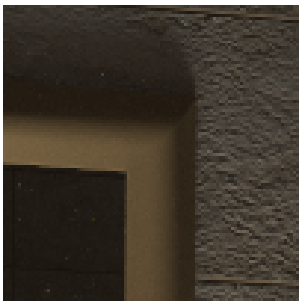
Эталон



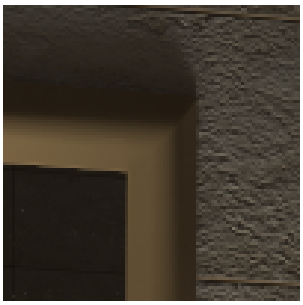
PT
MSE = 46



IBPT
MSE = 21



MMLT
MSE = 7

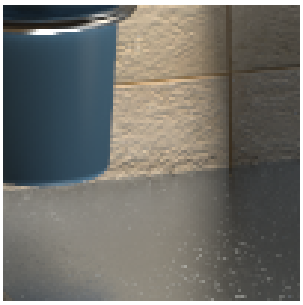


Эталон

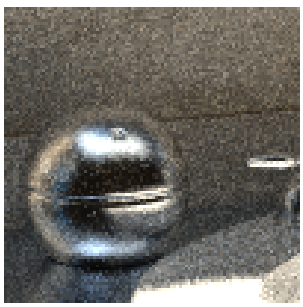
PT
MSE = 45

IBPT
MSE = 16

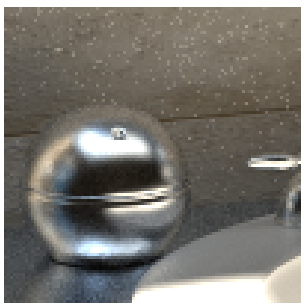
MMLT
MSE = 18



Эталон

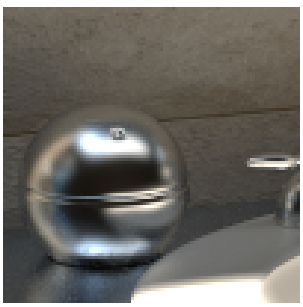


PT
MSE = 45



IBPT
MSE = 16

MMLT
MSE = 23



Эталон

5. Сравнение с Octane

Система Octane [38] была выбрана по двум основным причинам. Во-первых, Octane активно позиционируется разработчиком как самая быстрая в мире рендер-система. Во-вторых, в настоящий момент Octane — единственная в мире рендер-система на GPU, в которой реализован и используется на практике MCMC-метод интегрирования освещённости — (PMC) [17, 32].

Сравнение с рендер-системой Octane (версия 4.00–6.10) проводилось на машине с двумя GPU фирмы Nvidia (RTX2070 и GTX1070), ОС Windows 7. Эталоны были получены за 3 часа и использовались для оценки ошибки. Изображения эталонов для данного сравнения в статье не представлены. Прежде всего было проведено сравнение скорости трассировки лучей (оценивая количество сэмплов в секунду). К счастью, это легко сделать, т. к. обе сопоставляемые программы позволяют фиксировать количество Монте-Карло сэмплов в PT и глубину просчёта. Измерив время, за которое это число сэмплов было сделано, можно оценить скорость (таблица 4).

Сцена	Sam/s (Octane)	Sam/s (Our)	Выигрыш / Проигрыш
scene03	21	12	-75%
scene04	18	19	+5%
scene06	9.9	8.7	-13%
scene07	12	10	-20%

Таблица 4. Сравнение скорости трассировки лучей в системе Octane и в разработанной системе при фиксированной глубине просчёта в PT, равной 8 отскоков. В таблице отображено количество Монте-Карло сэмплов в секунду. Количество обрабатываемых лучей в секунду нетрудно оценить, если умножить это число на 16. Протестировано на Nvidia GTX2070 . Для этого сравнения в Octane денойзинг и тон-маппинг на GPU были отключены, чтобы оценить чистую производительность геометрического ядра системы.

PT (Octane, MSE = 0.99)

PT (Ours, MSE = 1.07)

PMC (Octane, MSE = 1.08)

MMLT (Ours, MSE = 1.06)

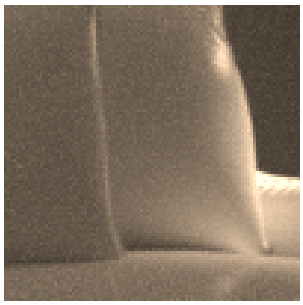
Рис. 12. Сравнение разработанного решения с Octane (scene03); 30 минут.

PT (Octane)
MSE = 16

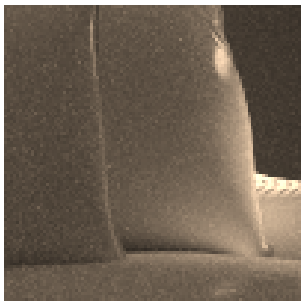
PT (Ours)
MSE = 14

PMC (Octane)
MSE = 17

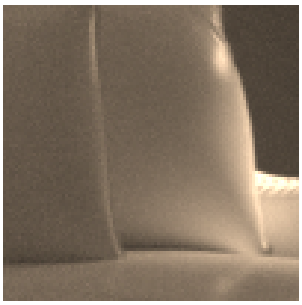
MMLT (Ours)
MSE = 38



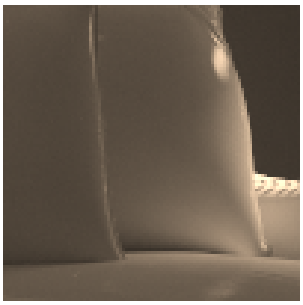
PT (Octane)
MSE = 10



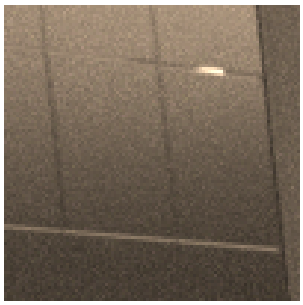
PT (Ours)
MSE = 11



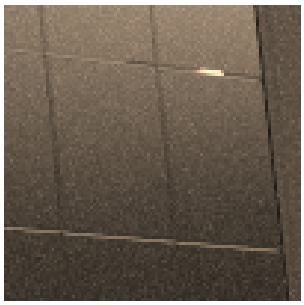
PMC (Octane)
MSE = 12



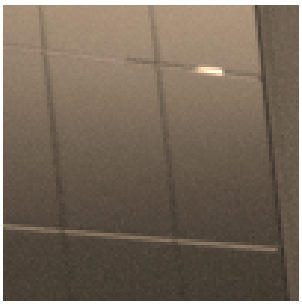
MMLT (Ours)
MSE = 6



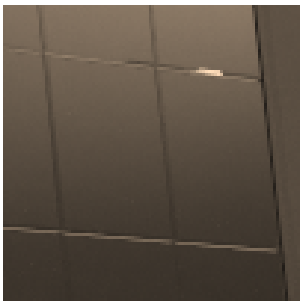
PT (Octane)
MSE = 12



PT (Ours)
MSE = 13

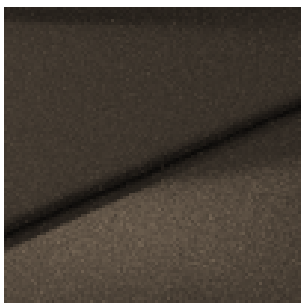


PMC (Octane)
MSE = 11



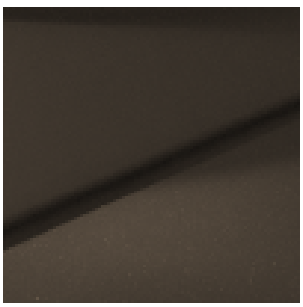
MMLT (Ours)
MSE = 5

PT (Octane)
MSE = 9

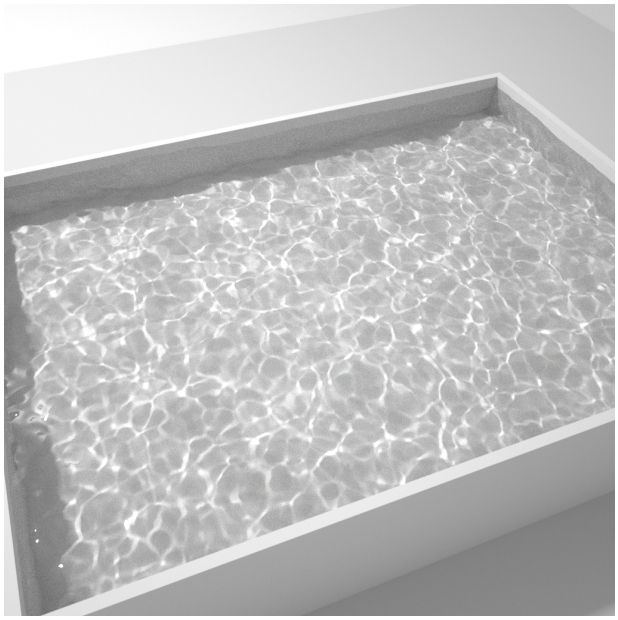


PT (Ours)
MSE = 9

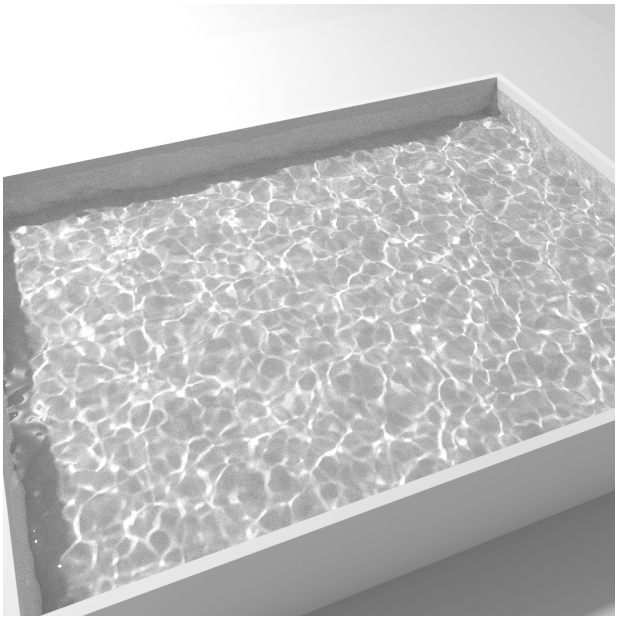
PMC (Octane)
MSE = 10



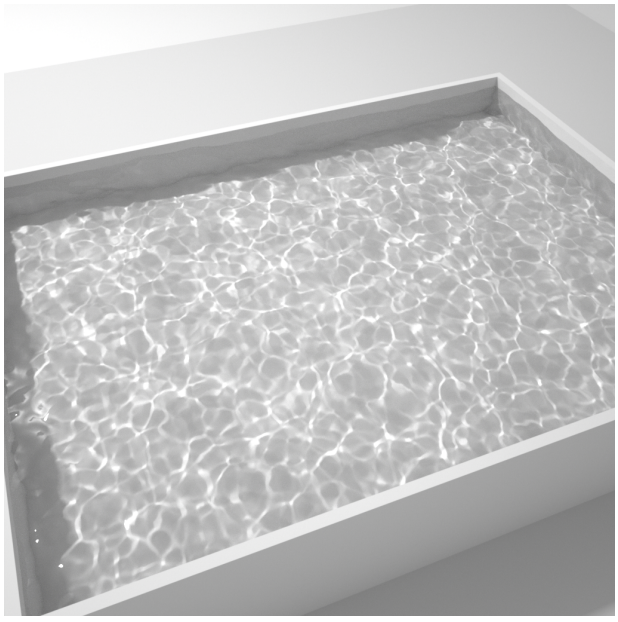
MMLT (Ours)
MSE = 8



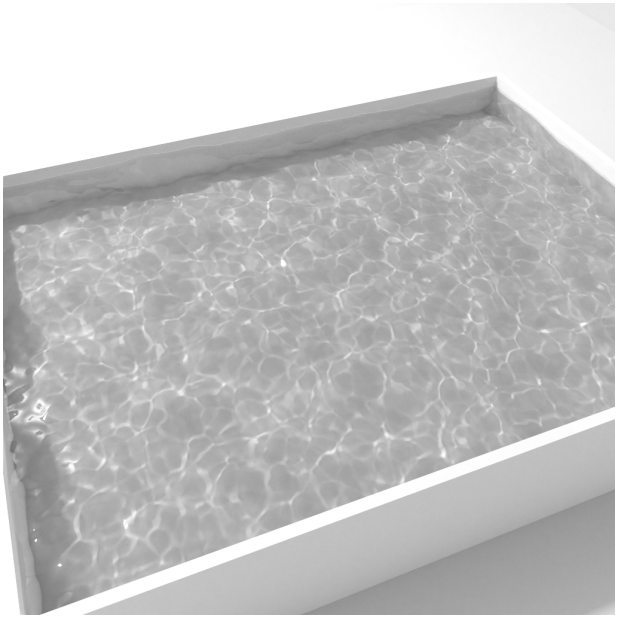
PT (Octane, MSE = 1.9)



PT (Ours, MSE = 2.5)

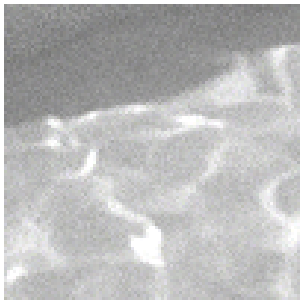


PMC (Octane, MSE = 1.08)

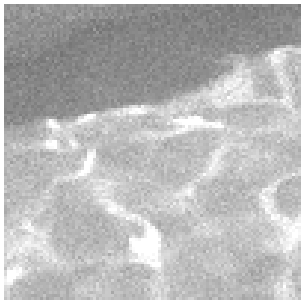


MMLT (Ours, MSE = 1.06)

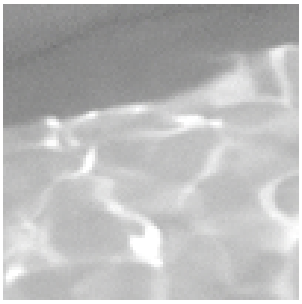
Рис. 13. Сравнение разработанного решения с Octane (scene04); 30 минут.



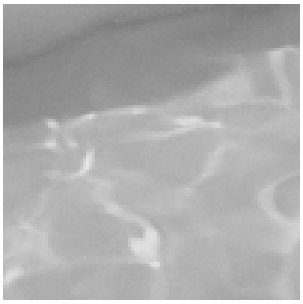
PT (Octane)
MSE = 21



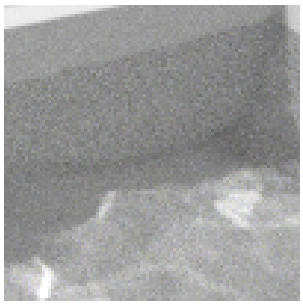
PT (Ours)
MSE = 32



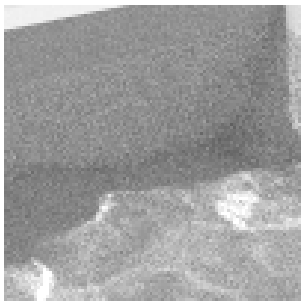
PMC (Octane)
MSE = 0.5



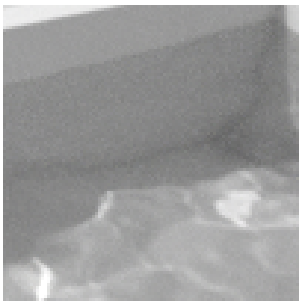
MMLT (Ours)
MSE = 6.5



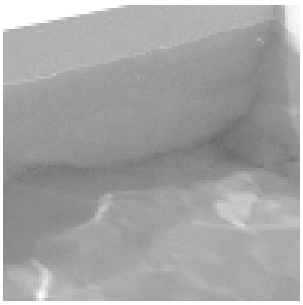
PT (Octane)
MSE = 21



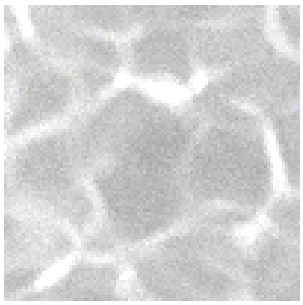
PT (Ours)
MSE = 31



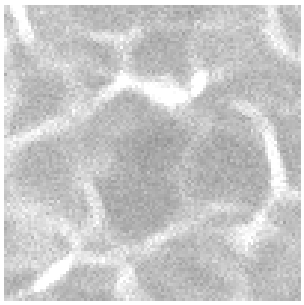
PMC (Octane)
MSE = 8.4



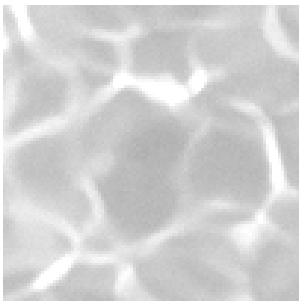
MMLT (Ours)
MSE = 65



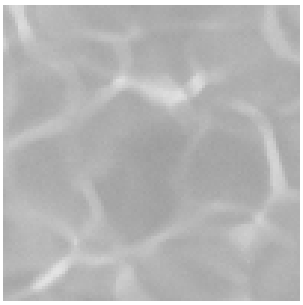
PT (Octane)
MSE = 22



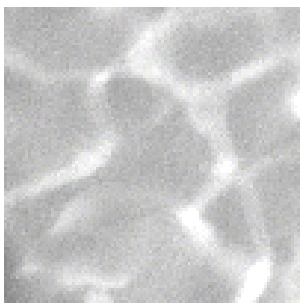
PT (Ours)
MSE = 33



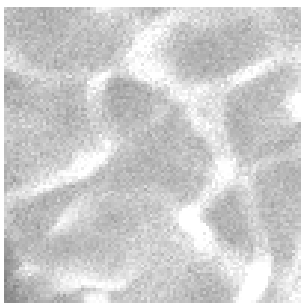
PMC (Octane)
MSE = 8.2



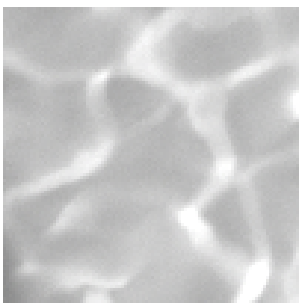
MMLT (Ours)
MSE = 83



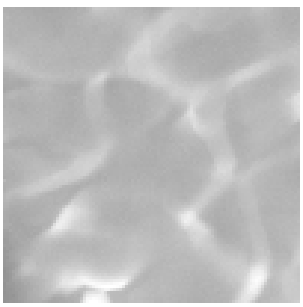
PT (Octane)
MSE = 21



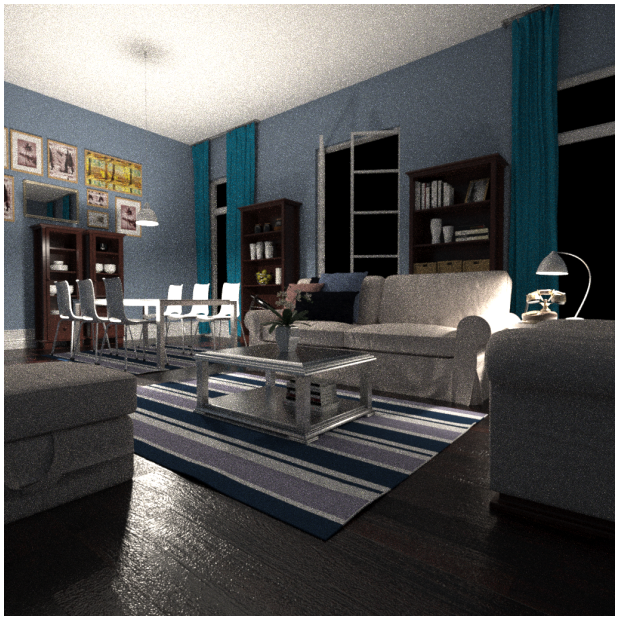
PT (Ours)
MSE = 34



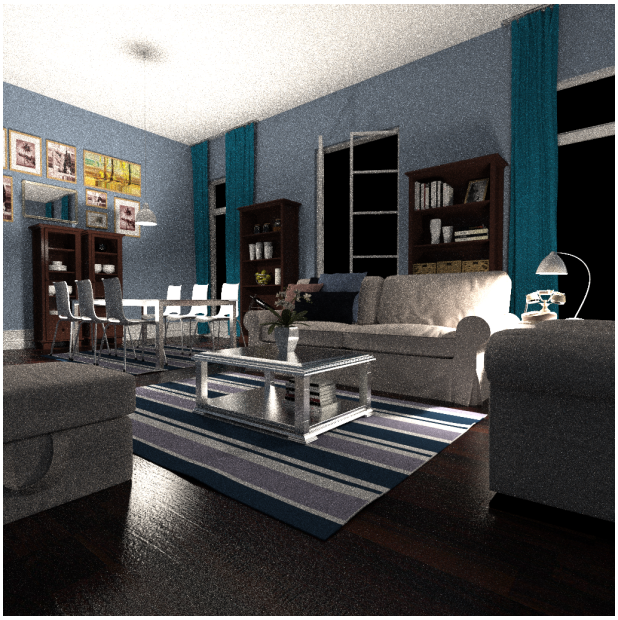
PMC (Octane)
MSE = 7.4



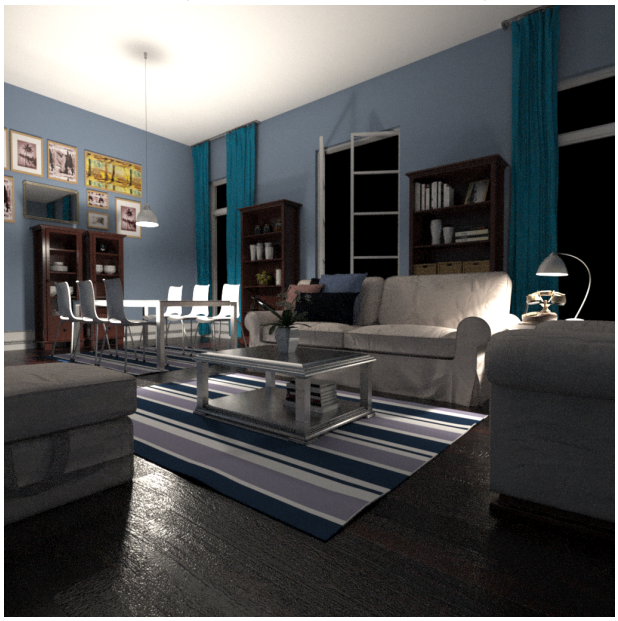
MMLT (Ours)
MSE = 84



PT (Octane, MSE = 4.5)



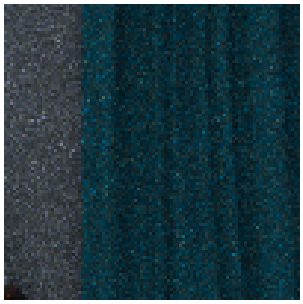
PT (Ours, MSE = 6.6)



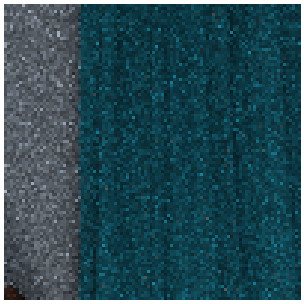
PMC (Octane, MSE = 1.8)

MMLT (Ours, MSE = 1.5)

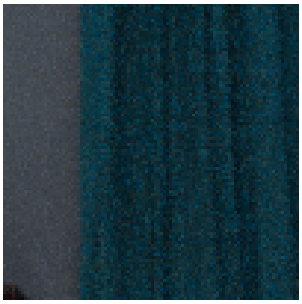
Рис. 14. Сравнение разработанного решения с Octane (scene06); 30 минут.



PT (Octane)
MSE = 35

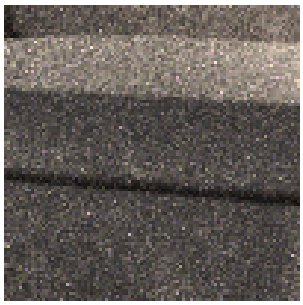


PT (Ours)
MSE = 56

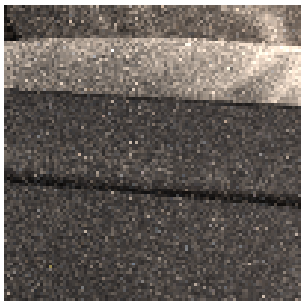


PMC (Octane)
MSE = 23

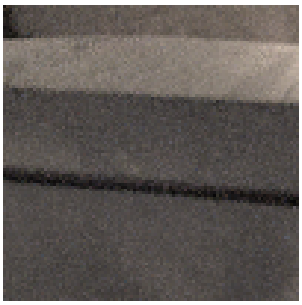
MMLT (Ours)
MSE = 12



PT (Octane)
MSE = 61

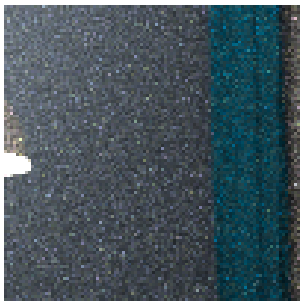


PT (Ours)
MSE = 73

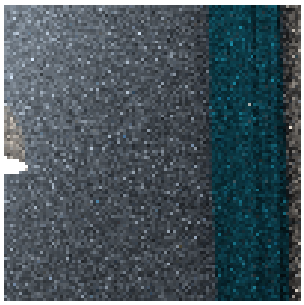


PMC (Octane)
MSE = 18

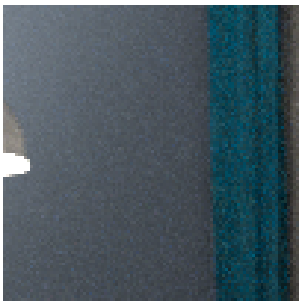
MMLT (Ours)
MSE = 14



PT (Octane)
MSE = 50



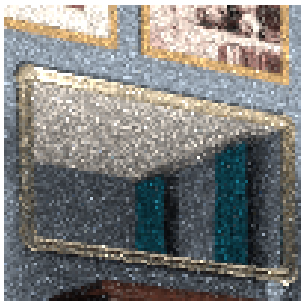
PT (Ours)
MSE = 76



PMC (Octane)
MSE = 18

MMLT (Ours)
MSE = 12

PT (Octane)
MSE = 49



PT (Ours)
MSE = 87

PMC (Octane)
MSE = 18

MMLT (Ours)
MSE = 21

PT (Octane, MSE = 3.43)

PT (Ours, MSE = 3.34)

PMC (Octane, MSE = 1.4)

MMLT (Ours, MSE = 1.0)

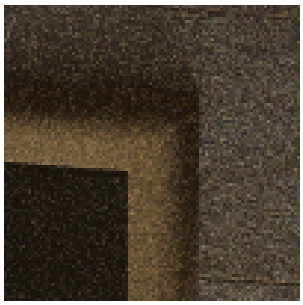
Рис. 15. Сравнение разработанного решения с Octane (scene07); 30 минут.

PT (Octane)
MSE = 35

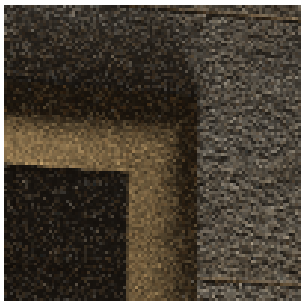
PT (Ours)
MSE = 38

PMC (Octane)
MSE = 17

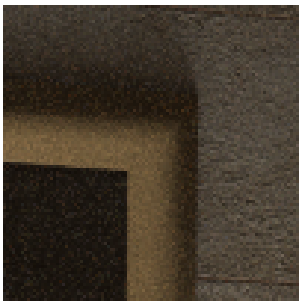
MMLT (Ours)
MSE = 15



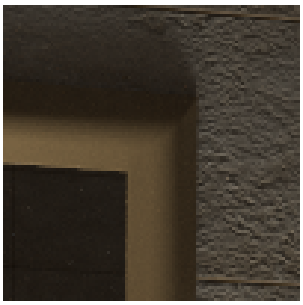
PT (Octane)
MSE = 32



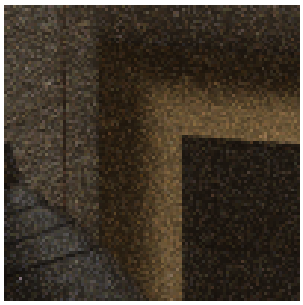
PT (Ours)
MSE = 34



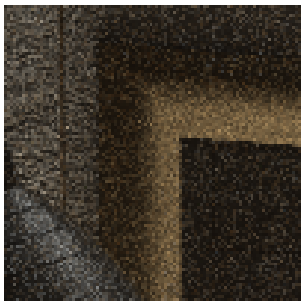
PMC (Octane)
MSE = 16



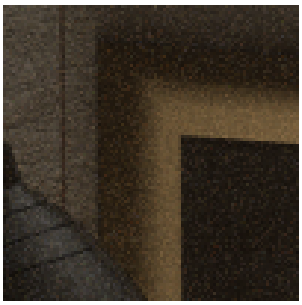
MMLT (Ours)
MSE = 6.8



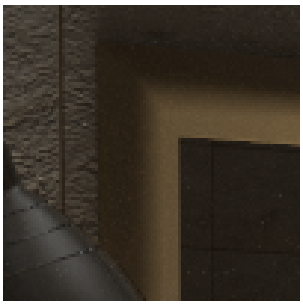
PT (Octane)
MSE = 30



PT (Ours)
MSE = 33



PMC (Octane)
MSE = 17



MMLT (Ours)
MSE = 7.8

PT (Octane)
MSE = 43

PT (Ours)
MSE = 35

PMC (Octane)
MSE = 13

MMLT (Ours)
MSE = 6.6

6. Анализ и выводы

MMLT на GPU. Применение Multiplexed Metropolis Light Transport на GPU позволяет на практике добиться «финального качества» изображения за гарантированное время (1–2 часа в зависимости от аппаратной конфигурации) для сцен, содержащих трудновычислимые феномены освещённости. В таком изображении отсутствует регулярный шум, заметный для человеческого глаза (остаточный импульсный шум тривиально удаляется медианным фильтром с порогом применения фильтрации).

Однако, при наличии трудновычислимых феноменов освещённости с SDS-путями (типа водных каустиков, сравнение на сцене scene04) в изображении всё ещё может наблюдаться начальное смещение, которое уходит лишь при увеличении времени расчёта (до 3–4 часов). Важно отметить, что увеличение количества GPU или их мощности (т. е. увеличение количества потоков) не приводит к желаемому устранению начального смещения — оно приводит лишь к уменьшению уровня шума. Для устранения начального смещения из изображения необходимо, чтобы каждая марковская цепь сделала достаточно большое число шагов. Предложенный в [18] метод устранения начального смещения при помощи прожига амортизирует, но не решает проблему. Поскольку в MMLT пространство интегрирования стало сложнее, чем в однонаправленном MLT из работы [18], прожиг уже не помогает — отобранные цепи не успевают исследовать всё пространство за небольшое время (10–30 минут). При этом в проведённых экспериментах трёх часов было всегда достаточно, чтобы полностью устранить видимое начальное смещение в изображении.

Из этого следует неожиданный вывод. Хотя феномен возросшего «начального смещения» для MMLT на GPU не позволяет получить быстрый предпросмотр кадра, а при недостаточном времени расчёта само изображение выглядит неправильно (яркие участки получаются недостаточно яркими), зато при фиксированном времени расчёта в 1–2 часа на изображении нет видимого регулярного шума. Таким образом, MMLT на GPU не подходит для быстрого предпросмотра и не всегда лучше ОМС-методов в плане численной ошибки. Однако он позволяет гарантированно получить чистое изображение при длительном расчёте (от одного часа и более) во всех протестированных сценариях и лучше других методов подходит для синтеза высококачественных изображений сцен с трудновычислимыми феноменами освещённости.

Сравнение с Octane. На алгоритме трассировки путей (PT) предложенное решение в среднем достигает партритета с системой Octane. Проведённое прямое сравнение прозводительности трассировки лучей (таблица 4) это подтверждает. При этом на трудновычислимых феноменах освещённости предложенное

решение в 3–4 раза выигрывает у Octane (исходя из квадратичной зависимости производительности от ошибки [6]). Существенный выигрыш наблюдается даже на сцене scene03, несмотря на то что геометрическое ядро на ней проигрывает Октану 75% производительности (таблица 4).

На сцене с бассеином разработанный метод проиграл методу PMC в ошибке из-за наличия сильного начального смещения, хотя при этом визуально произвёл меньше шума.

Улучшение методов сравнения. Отдельно хотелось бы отметить, что численная оценка (особенно проводимая по всему изображению) при помощи метрики MSE слабо коррелирует с визуальной оценкой качества изображения. Тем не менее данную метрику часто используют из-за хорошо известной квадратичной зависимости времени расчёта от ошибки. Использование в сравнениях метрики SSIM [48] и других более интеллектуальных метрик является предметом будущих исследований.

Заключение

В данной работе был реализован и исследован алгоритм MMLT на графических процессорах. Предложенные оптимизации позволяют повысить производительность MMLT на GPU от 30% до 80% по сравнению с наивной реализацией (работа [42]). При этом для геометрически сложных сцен ускорение больше, чем для простых.

Реализованный алгоритм интегрирован в свободно распространяемую систему расчёта освещения с открытым исходным кодом Hydra Renderer (имеет плагин для 3ds Max) и позволяет на одном-двух графических процессорах с «финальным качеством» рассчитывать освещение трёхмерных сцен с трудновычислимыми феноменами освещённости. Наиболее близким аналогом является коммерческая рендер-система Octane [38], с которой было проведено сравнение, и показано преимущество разработанного решения на архитектурных сценах.

Благодарности

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект 16-31-60048-мол_а_дк.

Список литературы

[1] Боголепов Д. К. *Методы глобального освещения для интерактивного синтеза изображений сложных сцен на графических процессорах* // Диссерта-

ция на соискание степени кандидата технических наук. 2013. Нижний Новгород.

[2] Изотов И. *Уроки 3ds Max. Caustics CORONA. Вода в бассейне с каустикой в короне* // Видеоурок на youtube (2018). URL = https://www.youtube.com/watch?v=Nv1ULR8sMZY

[3] Жданов Д. Д., Ершов С. В., Волобой А. Г. *Адаптивный выбор глубины трассировки обратного луча в методе двунаправленной стохастической трассировки лучей* // Труды международной конференции Графикон 2015. Протвино, с. 44–49.

[4] Фролов В. А. *Методы решения проблемы глобальной освещенности на графических процессорах* // Дисс. канд наук. ИПМ им. М. В. Келдыша РАН. Москва. 2015.

[5] Фролов В. А., Галактионов В. А. *Унифицированный механизм сэмплирования изображений для современных методов интегрирования освещённости на GPU* // Новые информационные технологии в автоматизированных системах: материалы двадцать первого научно-практического семинара - М.: ИПМ им. М. В. Келдыша, 20 апреля 2018, с. 63–69.

[6] Фролов В. А., Галактионов В. А., Трофимов М. А. *Сравнение индустриальных систем расчёта глобального освещения (по состоянию на 2014 год)* // Приволжский научный журнал, № 4, 2014, с. 79–85.

[7] Фролов В. А., Санжаров В. А., Трофимов М. А., Галактионов В. А. *Hydra Renderer* // Рендер-система, разрабатываемая совместно в ИПМ им. М.В. Келдыша, МГУ имени М.В. Ломоносова и компании Ray Tracing Systems. Москва. 2017. URL = http://www.raytracing.ru/.

[8] Фролов В. А, Галактионов В. А. *Регенерация путей с низкими накладными расходами* // Программирование, 2016, № 6, с.67-74. English translation: V.A. Frolov, V.A. Galaktionov. *Low overhead path regeneration* // Programming and Computer Software, 2016, Vol. 42, № 6, pp. 382–387.

[9] Фролов. В. А. *Избирательное применение Metropolis Light Transport для трудновычислимых феноменов освещённости* // Препринты ИПМ им. М.В.Келдыша. 2017. № 116. 34 с. doi: 10.20948/prepr-2017-116

[10] Aila T., Laine S. *Understanding the Efficiency of Ray Traversal on GPUs* // High-Performance Graphics 2009.

[11] Antwerpen D. *Improving SIMD efficiency for parallel Monte Carlo light transport on the GPU* // In Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics (HPG '11), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, pp. 41–50. DOI: https://doi.org/10.1145/2018323.2018330

[12] Ashikhmin M., Premoze S., Shirley P., Smits B. *A Variance Analysis of the Metropolis Light Transport Algorithm* // Computers and Graphics (2001), pp. 287–294.

[13] Bogolepov D., Ulyanov D. *GPU-Optimized Bi-Directional Path Tracing* // WSCG'2013. Conference on Computer Graphics, Visualization and Computer Vision.

[14] Brooks S., Gelman A., Jones G., Meng X. *MCMC using Hamiltonian dynamics* // Handbook of Markov Chain Monte Carlo. May 10, 2011 by Chapman and Hall/CRC. Chapter 5. pp. 113–163.

[15] Belcour L., Yan L., Ramamoorthi R., and Nowrouzezahrai D. *Antialiasing complex global illumination effects in path-space* // ACM Trans. Graph. 36, 4, pages. DOI: https://doi.org/10.1145/3072959.3126812

[16] Bitterli B., Wenzel J., Novák J., Jarosz W. *Reversible Jump Metropolis Light Transport using Inverse Mappings* // ArXiv preprint. Submitted on 22 Apr 2017. https://arxiv.org/abs/1704.06835

[17] Cline D., Talbot J., Egbert P. *Energy redistribution path tracing* // ACM Trans. Graph. 24, 3 (July 2005), 1186–1195. DOI: https://doi.org/10.1145/1073204.1073330

[18] Frolov V. A., Galaktionov V. A. *Memory-Compact Metropolis Light Transport on GPUs* // Programming and Computer Software, 2017, Vol. 43, № 3, pp. 196–203. DOI: 10.1134/S0361768817030057

[19] Frolov V., Kharlamov A., Ignatenko A. *Biased global illumination via Irradiance Caching and Adaptive Path Tracing on GPUs* // Programming and Computer Software (2011).

[20] Georgiev I., Křivánek J., Davidovič T., Slusallek P. *Light Transport Simulation with Vertex Connection and Merging* // ACM Trans. Graph. (SIGGRAPH Asia 2012) 31, 6 (2012).

[21] Hachisuka T., Kaplanyan A.S., Dachsbache C. *Multiplexed Metropolis Light Transport* // ACM Transactions on Graphics (TOG) — Proceedings of ACM SIGGRAPH 2014.

[22] Hachisuka T., Wann Jensen H. *Stochastic progressive photon mapping* // Proceeding SIGGRAPH Asia ’09 ACM SIGGRAPH Asia 2009 papers Article No. 141.

[23] Hachisuka T., Pantaleoni J., Wann Jensen H. *A path space extension for robust light transport simulation* // Proceedings of ACM SIGGRAPH Asia 2012, Volume 31 Issue 6, November 2012 Article No. 191.

[24] Igehy H. *Tracing ray differentials* // In Proceedings of the 26th annual conference on Computer graphics and interactive techniques (SIGGRAPH ’99). 1999. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 179–186. DOI = http://dx.doi.org/10.1145/311535.311555

[25] Jendersie K. R, Brüll F., Grosch T. *Pixel Cache Light Tracing* // Vision, Modeling, and Visualization (2017).

[26] Jensen H. W. *Global Illumination using Photon Maps* // In Proc. of Eurographics Workshop on Rendering (EGWR) (1996), EGWR, Springer, pp. 21–30.

[27] Kajiya, J. T. *The rendering equation* // In Proceedings of the 13th annual conference on Computer graphics and interactive techniques (SIGGRAPH ’86), David C. Evans and Russell J. Athay (Eds.). ACM, New York, NY, USA, 143–150. DOI = http://dx.doi.org/10.1145/15922.15902

[28] Kaplanyan A. S., Hanika J., Dachsbacher C. *The Natural-constraint Representation of the Path Space for Efficient Light Transport Simulation* // ACM Transactions on Graphics (Proceedings of SIGGRAPH) 33, 4, Article 102 (July 2014), 13 pages.

[29] Kaplanyan A. S., Dachsbacher C. *Adaptive progressive photon mapping* // ACM Trans. Graph. 32, 2, Article 16 (April 2013), 13 pages. DOI: https://doi.org/10.1145/2451236.2451242

[30] Křivánek J., Georgiev I., Hachisuka T., Vévoda P., Šik M., Nowrouzezahrai D., Jarosz W. *Unifying Points, Beams, and Paths in Volumetric Light Transport Simulation* ACM Transactions on Graphics (Proceedings of SIGGRAPH), 33(4), July 2014.

[31] 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. Volume 21, 2002, Number 3.

[32] Lai Y., Fan S. H., Chenney S., Dyer C. *Photorealistic image rendering with population Monte-Carlo energy redistribution* // In Proceedings of the 18th Eurographics conference on Rendering Techniques (2007).

[33] Laine S., Karras T, Aila T. *Megakernels Considered Harmful: Wavefront Path Tracing on GPUs* // High-Performance Graphics 2013.

[34] Li T., Lehtinen J., Ramamoorthi R., Jakob W., Durand F. *Anisotropic Gaussian mutations for metropolis light transport through Hessian-Hamiltonian dynamics* // ACM Trans. Graph. 34, 6, Article 209 (October 2015), 13 pages.

[35] Leal R. J. *CppADCodeGen. Source Code Generation for Automatic Differentiation using Operator Overloading* // GitHub. 2011–2015. URL=https://github.com/joaoleal/CppADCodeGen/

[36] Novak J., Havran V., Dachsbacher C. *Path Regeneration for Interactive Path Tracing* // Proceedings of EUROGRAPHICS 2010.

[37] Otsu H., Kaplanyan A. S., Hanika J., Dachsbacher C., Hachisuka T. *Fusing state spaces for markov chain Monte Carlo rendering* // ACM Trans. Graph. 36, 4, Article 74 (July 2017), 10 pages. DOI: https://doi.org/10.1145/3072959.3073691

[38] OTOY. *Octane Render* URL = https://home.otoy.com/render/octane-render/

[39] Popov S., Ramamoorthi R., Durand F., Drettakis G. *Probabilistic Connections for Bidirectional Path Tracing* // Eurographics Symposium on Rendering 2015, Volume 34 (2015), Number 4.

[40] Pantaleoni J. *Charted metropolis light transport* // ACM Trans. Graph. 36, 4, Article 75 (July 2017), 14 pages. DOI: https://doi.org/10.1145/3072959.3073677

[41] Render Legion. *Corona Render System* URL = https://corona-renderer.com/

[42] Schmidt M., Lobachev O., Guthe M. *Coherent Metropolis Light Transport on the GPU using Speculative Mutations* // Journal of WSCG, №1б 2016.

[43] Suykens F., Willems Y. D. *Density Control for Photon Maps* In: Péroche B., Rushmeier H. (eds) Rendering Techniques 2000. Eurographics. Springer, Vienna.

[44] Veach E. *Robust monte carlo methods for light transport simulation* // (1998) Ph.D. Dissertation. Stanford University, Stanford, CA, USA. Advisor(s) Leonidas J. Guibas. URL=https://graphics.stanford.edu/papers/veach_thesis/thesis-bw.pdf

[45] Veach E., Guibas L. J. *Metropolis Light Transport* // SIGGRAPH ’97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques. Pages 65-76.

[46] Vorba J. *Bidirectional Photon Mapping* // Proceedings of CESCG 2011: The 15th Central European Seminar on Computer Graphics. Charles University, Prague, 2011.

[47] Wald I., Woop S., Benthin C., Johnson G. S., Ernst M. *Embree: a kernel framework for efficient CPU ray tracing.* ACM Trans. Graph. 33, 4, Article 143 (July 2014), 8 pages. DOI: https://doi.org/10.1145/2601097.2601199

[48] Wang Z., Bovik A. C., Sheikh H. R., Simoncelli E. P. *Image quality assessment: From error visibility to structural similarity*, IEEE Transactions on Image Processing, vol. 13, no. 4, p. 600–612, Apr. 2004.

[49] Wenzel J. *Light Transport on Path-Space Manifolds* // Ph.D. Dissertation. Cornell University (2013).