Сравнительный анализ генераторов псевдослучайных чисел для решения задач рендеринга методом Монте-Карло

И.Ю. Сесин, В.В. Нечаев

***Аннотация* — Трассировка пути является одним из методов фотореалистичного рендеринга. Этот метод позволяет симулировать множество оптических явлений за счет применения продвинутой математической модели – интегрального уравнения рендеринга. Для расчета интегральной части этого уравнения применяется метод Монте-Карло, что на практике делает расчет этого метода очень ресурсоёмким. Ввиду чрезвычайной параллельности задачи расчета данного интеграла, этот метод рендеринга целесообразно реализовывать на графических процессорах с использованием технологии GPGPU. Однако одним из ключевых моментов использования метода Монте-Карло является использование случайных чисел для проведения вычислений статистических характеристик модели. Вынос расчетов на графический процессор очень сильно усложняет использование в данных целях истинно случайных чисел, и поэтому возникает вопрос выбора алгоритма генератора псевдослучайных чисел (ГПСЧ) для внедрения в программу, запускаемую на GPU. В этой статье произведен сравнительный анализ ряда алгоритмов ГПСЧ для использования в задаче вычисления интегральной составляющей уравнения рендеринга методом Монте-Карло при помощи технологии GPGPU. На основе результатов анализа показаны сильные и слабые стороны алгоритмов применительно к методу Монте-Карло, а так же описаны некоторые общие характеристики, которыми следует руководствоваться при выборе ГПСЧ для решения подобной задачи.**

***Ключевые слова* — генератор псевдослучайных чисел, метод Монте-Карло, трассировка пути.**

I. Введение

Под методом Монте-Карло принято понимать численный алгоритм, использующий случайные величины для решения задач посредством моделирования и дальнейшего вычисления вероятностных характеристик, исходя из полученных выборок. Численный метод Монте-Карло применяется для решения задач в самых различных областях науки. Одним из наиболее часто реализуемых приложений метода является интегрирование, а именно вычисление определенных интегралов. Интегрирование методом Монте-Карло, в основном, применяется для случаев, в которых проблематично применение иных численных методов. Одним из примеров таких случаев можно назвать вычисление по полусфере интеграла входящего потока излучения, являющееся краеугольным камнем обсчёта света в фотореалистичном рендеринге.

Естественно, что качество результатов решения задач методом Монте-Карло прямым образом зависит от случайных чисел, на основе которых вычисляются статистические параметры выборки, и далеко не всегда существует возможность воспользоваться аппаратными решениями для получения истинно случайных чисел.

Поэтому так важен рациональный выбор ГПСЧ для получения высокого качества результата решения задачи и за приемлемое время. Отметим, что неправильный выбор ГПСЧ может стоить больших временных затрат и потраченных напрасно усилий. При работе с технологией GPGPU ситуация еще сильнее осложняется – в игру вступают особенности архитектуры и необходимость адаптации программы генератора.

II. Применение ГПСЧ на примере вычисления интегральной составляющей уравнения рендеринга

Практика показывает, что, помимо выбранного алгоритма очень немалый вклад в стабильность и корректность программы вносит непосредственная реализация данного алгоритма, а также все сопутствующие участки кода, прямо и косвенно работающие с ГПСЧ.

Иначе говоря, даже имея идеальный по своим характеристикам ГПСЧ очень легко без проявления должного внимания и аккуратности снизить качество его статистических характеристик. Ввиду этого факта, целесообразно рассматривать и сравнивать ГПСЧ в контексте решения конкретной задачи, так как помимо сухих статистических характеристик существуют неявные особенности и проблемы, которые могут неожиданно усложнять реализацию и внедрение ГПСЧ в программное средство.

Трассировка пути в данном случае представляет собой отличный полигон для сравнения различных ГПСЧ ввиду целого ряда особенностей: результат работы очевидным образом визуализируется, позволяя качественно оценить числа, генерируемые ГПСЧ, а нетривиальность задачи позволяет оценить, насколько сложен тот или иной ГПСЧ во внедрении. Так же присутствуют ограничения, как по доступной памяти, так и по скорости работы алгоритма. В целом, это можно расценивать как применение ГПСЧ не к абстрактной, а к конкретной существующей задаче с достаточно строгим набором условий, что позволяет

Статья получена 19 сентября 2018.

И.Ю. Сесин, МИРЭА (e-mail: isesin@protonmail.com).

В.В. Нечаев, к.т.н. профессор, зав. лаб. МИРЭА (e-mail: nechaev@mirea.ru).

выявить и оценить неочевидные преимущества или недостатки тех или иных ГПСЧ.

Результирующее изображение весьма чувствительно к качеству используемых случайных чисел, но не на столько, чтобы отвергать все ГПСЧ, не являющиеся криптографическими. При этом может сложиться ситуация, в которой быстрый, но менее качественный ГПСЧ будет сходиться за меньшее количество времени, нежели медленный, но качественный ГПСЧ, что делает выбор еще более неоднозначным.

Итак, в задаче рендеринга методом трассировки пути ключевым моментом является численное решение уравнения рендеринга [1]:

$$L_O(x,\omega,\lambda,t) = L_e(x,\omega,\lambda,t) + \int_\Omega f_r(x,\omega',\omega,\lambda,t)L_i(x,\omega',\lambda,t)(-\omega' \cdot n)d\omega', (1)$$

где:

λ – длина волны света,

t – время,

$L_O(x,\omega,\lambda,t)$ – количество излучения с длиной волны λ, выпущенного из точки x вдоль направления ω в момент времени t,

$\int_\Omega f_r(x,\omega',\omega,\lambda,t)L_i(x,\omega',\lambda,t)(-\omega' \cdot n)d\omega'$ – интеграл по полусфере входящего излучения,

$f_r(x,\omega',\omega,\lambda,t)$ – двулучевая функция отражательной способности (ДФОС), описывающая характеристики процесса отражения света от поверхности,

$L_i(x,\omega',\lambda,t)$ – сумма входящего излучения, зависящая от длины волны λ излучения, выпущенного в точку x по направлению ω' в момент времени t,

$-\omega' \cdot n$ – скалярное произведение направления $-\omega'$ и нормали к поверхности n, характеризующее поглощение излучения при отражении.

В основе принципа работы трассировки пути лежит использование метода Монте-Карло для вычисления интеграла входящего излучения по полусфере.

На практике, в программной реализации трассировки пути, при помощи ГПСЧ определяют:

– два числа, на основе которых вычисляется направление следующего участка пути в пределах полусферы[2];

– число, определяющее шанс луча быть рассеянным, отраженным или преломленным;

– в ряде реализаций включается еще одно число, определяющее шанс луча быть исключенным из дальнейших вычислений (т.н. «оптимизация методом русской рулетки»).

III. Характеристики ГПСЧ

Ранее автором был предложен перечень требований [3], предъявляемых к характеристикам ГПСЧ. В этот перечень вошли следующие требования:

– ГПСЧ должен иметь высокое быстродействие;

– размер внутреннего состояния ГПСЧ должен быть небольшим, желательно кратным размеру регистра;

– ГПСЧ должен генерировать равномерно распределенные числа, желательно, чтобы это свойство соблюдалось на малых выборках;

– требуется очень быстрый выход из вырожденных состояний, их полное отсутствие, или же возможность применения смягчающих мер.

Однако данный список основных требований к характеристикам не включает целый ряд побочных характеристик, а также взаимосвязь и взаимодействие скоростных и статистических характеристик.

В то же время провести оценку некоторых характеристик ГПСЧ без их непосредственного внедрения затруднительно, что значительно усложняет процесс выбора. Кроме того, скорость работы внедрённого в программное средство ГПСЧ может отличаться от скорости работы ГПСЧ в других случаях. Скорость работы кода может меняться по целому ряду причин. Например, она может расти в результате оптимизации компилятора, использующего особенности кода, вызывающего ГПСЧ, или серьезно упасть из-за аппаратных узких мест в организации оперативной памяти. В силу данных обстоятельств, представляется целесообразным рассмотрение скорости работы ГПСЧ в контексте непосредственно существующей задачи, вместо замеров скорости алгоритма отдельно от всего.

Для такого рассмотрения и сравнения выдвигается следующий ряд показателей:

– размер внутреннего состояния генератора (измеряется в битах);

– сложность внедрения;

– время, затраченное на выполнение задачи программой;

– отсутствие визуальных артефактов.

Ряд из этих показателей затруднительно оценить численно, поэтому для них было решено использовать экспертное оценивание.

Под внутренним состоянием ГПСЧ в данной работе будет пониматься суммарный объем оперативной памяти, нужный ГПСЧ для хранения данных между вызовами.

Также, для задания исходного состояния генератора намеренно используется число с большим количеством нулевых битов. Это позволяет воспроизвести типовую проблему с передачей различных параметров (в данном случае – качественных случайных чисел для начального состояния генератора) на GPU для большого количества потоков. На практике для дифференциации между потоками есть только один способ – номер потока, который, собственно и используется для вычисления индивидуальных параметров каждого потока, таких как координаты обсчитываемого пикселя. Имея только одно число, весьма трудно превратить его в качественное, неповторяющееся исходное состояние для генератора без массивной переработки программы (что требует существенных усилий), или использования дополнительного ГПСЧ или хеширующего алгоритма (что замедлит работу программы).

Поэтому нередко в качестве исходных значений для инициализации генератора берут либо сам номер потока, либо полученное его преобразованием число.

Помимо этого, такие исходные состояния проверяют косвенный параметр качества ГПСЧ – лавинообразное изменение внутреннего состояния и быстрый выход из вырожденных случаев.

Таким образом, при правильном выборе генератора не требуется усложнять программу дополнительной

генерацией исходных значений, подходящих под какие-либо параметры – так как это не окажет существенного влияния на результаты.

IV. Сравнительный анализ ГПСЧ

Для реализации цели задачи сравнительного анализа были выбраны следующие ГПСЧ:

– линейный конгруэнтный (RANDU);

– линейный конгруэнтный (MINSTD);

– линейный конгруэнтный (glibc);

– линейный конгруэнтный (Numerical Recipes);

– регистр сдвига с линейной обратной связью (Многочлен $x^{32} + x^{22} + x^2 + x^1$ [4]);

– Xoroshiro128plus [5];

– Well Equidistributed Long-period Linear (WELL512) [6];

– ускоренная версия Hash_DRBG.

В качестве опорной точки для отсчёта при сравнении качества взята конфигурация RANDU, известная низким качеством генерируемых псевдослучайных чисел. С другой стороны, присутствует и криптографический ГПСЧ Hash_DRBG, представляющий другой конец спектра качества. Данный генератор был модифицирован с целью ускорения его работы, посредством устранения мер безопасности не влияющих на качество получаемых чисел. В качестве примера такого функционала можно привести обнуление массивов после работы с ними и переопределение начального состояния после получения определенного, достаточно большого количества случайных чисел.

Сравним размеры внутренних состояний генераторов. Важно, что размер внутреннего состояния влияет не только на период ГПСЧ но и на производительность на графических процессорах.

Результаты сравнения приведены в таблице 1.

№	Название	Размер внутреннего состояния в битах
1	Регистр сдвига с линейной обратной связью	32
2	glibc	32
3	MINSTD	32
4	Numerical Recipes	32
5	RANDU	32
6	Xoroshiro128plus	128
7	WELL512	1024
8	Hash_DRBG (упр.)	1208

Таблица 1. Сравнение размеров внутреннего состояния генераторов.

Отметим и тот факт, что в зависимости от архитектуры аппаратного обеспечения размеры структур внутренних состояний могут быть увеличены в результате выравнивания данных в памяти.

Сравним данные генераторы на предмет сложности их внедрения. Для оценки такого трудно формализуемого параметра было использовано экспертное оценивание, а именно ранжирование методом множественных сравнений. Данные генераторы были ранжированы от наиболее легко внедряемого до наиболее проблематичного. Приводятся комментарии относительно особенностей внедрения.

1. Xoroshiro128+ – очень простой с точки зрения внедрения метод. Существуют готовые примеры внедрения, в которых код краток и не требует ручной настройки, помимо изменения типов на совместимые с OpenCL;

2. группа линейных конгруэнтных методов (RANDU, MINSTD, glibc, Numerical Recipes): различаются только константами, однако требуют определения правильных констант и модификаций исходного кода в зависимости от них. Так, например, RANDU и MINSTD не используют инкремент, что влечет удаление одной операции сложения. Так же требуется проявление внимательности при работе с константой – делителем в операции взятия остатка от деления, так как при внедрении нескольких методов легко забыть о её изменении;

3. WELL512 – существуют готовые примеры внедрения [6] для которых может потребоваться некоторая доработка для совместимости с OpenCL. Тем не менее, этот метод имеет большой размер внутреннего состояния, ввиду чего требуются особые шаги для его инициализации;

4. регистр сдвига с линейной обратной связью – требует выбора многочлена с опорой на длину периода, (которая, в свою очередь, зависит от длины внутреннего состояния) и последующую модификацию кода, требующую понимания работы алгоритма;

5. Hash_DRBG – крайне сложный для внедрения алгоритм, отличающийся от остальных алгоритмов на порядки как по сложности, так и по объёму кода. Существуют примеры внедрения этого алгоритма в продуктах с открытым исходным кодом [7, 8], но требуется очень серьезная адаптация кода для использования с технологией GPGPU. Крайне легко допустить ошибки, которые будут отрицательно сказываться на качестве получаемых чисел и которые в то же время невероятно трудно диагностировать. Даже при полном отсутствии ошибок первоначальная адаптация может занять большое количество времени (первоначальное внедрение алгоритма автором заняло около 100 часов).

Произведем сравнение времени, затраченного на выполнение задачи программой для каждого алгоритма.

Следует отметить, что для данного теста были выбраны следующие параметры программы:

– целевой размер изображения 1280 на 720 пикселей;

– 2048 итераций алгоритма на пиксель;

– 16 отражений на каждую итерацию;

– отсечение т.н. методом «русской рулетки» не применяется, для фиксации количества выборок случайных чисел в каждом тесте;

– используется максимальный размер рабочей группы (группы одновременно обрабатываемых потоков), который не приводит к перезапуску видео драйвера системой TDR [9]. На практике это означает, что каждая обработка группы пикселей не должна занимать более 2х секунд, что достигается уменьшением рабочей группы до размера, приемлемого для данной конфигурации аппаратного обеспечения.

№	Название	Среднее время выполнения
1	Numerical Recipes	48,8608 секунд
2	RANDU	48,9108 секунд
3	Регистр сдвига с линейной обратной связью	48,9716 секунд
4	Xoroshiro128plus	50,4427 секунд
5	MINSTD	50,8922 секунд
6	glibc	51,4302 секунд
7	WELL512	270,2284 секунд
8	Hash_DRBG	>20 часов

Таблица 2. Сравнение среднего времени выполнения типовой задачи генераторами.

Как можно видеть на таблице 2, усложнение метода разительно сказывается на скорости работы, и при таких скоростных характеристиках Hash_DRBG просто не в состоянии составить конкуренцию более простым методам.

Что касается визуальных артефактов, то большинство методов производят практически идентичные изображения, отличающиеся только распределением шума на изображении.

Рис. 1. Изображение – эталон.

Однако, это справедливо не для всех методов. В частности, RANDU продуцирует отчётливые артефакты на изображении, выражающиеся в диагональной «ряби», исходящей из центра координат используемой для пикселей.

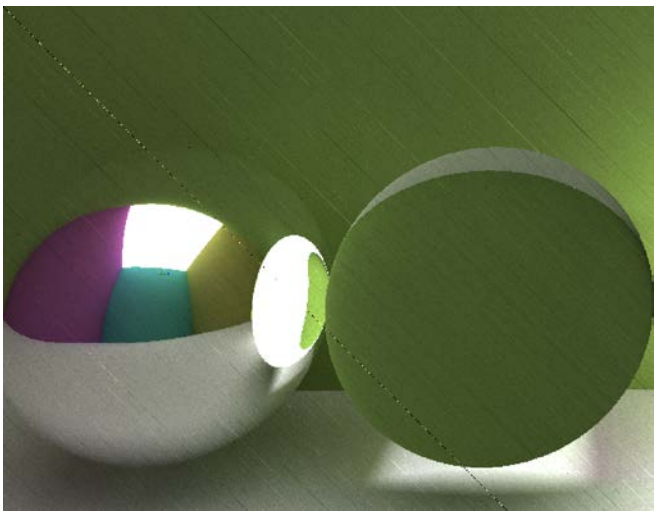

Рис. 2. Артефакты RANDU.

Это является прямым последствием того факта что для инициализации ГПСЧ используются координаты пикселя на изображении. Это, в свою очередь, показывает, что метод RANDU продуцирует предсказуемые числа, не годящиеся в качестве случайных чисел. Впрочем, данный вывод уже достаточно давно является установленным фактом. Как это ни странно, но регистр сдвига с линейной обратной связью, обладающий периодом в 4294967295 ($2^{32} - 1$) тоже начал производить артефакты.

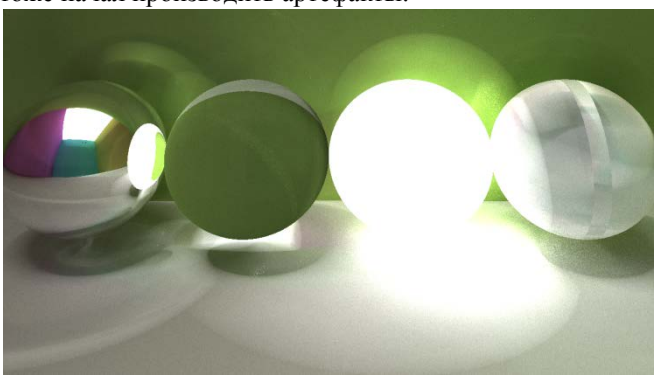

Рис. 3. Артефакты регистра сдвига с линейной обратной связью.

Примечательно, что данный генератор производит весьма нетипичные артефакты, непохожие на обычный набор из совершенно черных или белых пикселей, линий или цветовых пятен, характерных для других ГПСЧ. Вместо этого данный генератор производит достаточно цельную картинку, но при этом его результат расходится с правильным результатом. Это выражается в гораздо более резком освещении, несмотря на то, что метод трассировки пути, вообще говоря, ориентирован на получение мягкого, фотореалистичного освещения. Вероятно, это связано с однообразностью значений или проявлением закономерностей на определенных диапазонах, что, в свою очередь, приводит к неравномерному распределению направления следующего луча в пределах полусферы для некоторого количества итераций. Подобное стечение обстоятельств приводит к тому, что большое количество участков путей уходят в примерно одном направлении, создавая неравномерную выборку освещенности в данной точке, чрезмерно затемняя или освещая её, а затем этот результат частично корректируется за счет итераций, в которых лучи ушли в иные направления.

Подобные результаты очень похожи на ошибки в подсчетах вероятностных характеристик, и в реальных условиях поиск причины возникновения расхождения с правильным результатом мог бы составить существенное время, а принимая в учет серьезный период генератора, он может далеко не сразу попасть под подозрение.

В данных условиях возможность оценить результат визуально очень полезна (хоть и требуется изначальное представление о том, как выглядит результат) – выявление подобных проблем с генераторами было бы крайне затруднительно без привлечения специального математического аппарата, если бы результат работы программы был бы выражен одним или несколькими числами.

Отдельно следует упомянуть Hash_DRBG. В правильной реализации алгоритма артефактов очевидным образом возникать не должно, однако, внесение ошибок в процессе адаптации алгоритма чрезвычайно легко, и практически неминуемо ведет к падению качества результирующего изображения, как показано на рисунке 4.

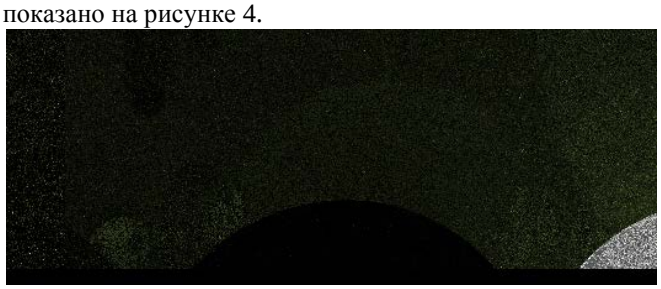

Рис. 4. Артефакты, вызванные ошибками во внедрении Hash_DRBG.

Но в процессе работы с этим алгоритмом тестировать работоспособность визуально по качеству картинки весьма и весьма накладно – вычисление одного полного изображения с приведенными характеристиками в приведенных примерах заняло бы у автора более 20 часов.

В случае работы с данным алгоритмом и подобными ему по сложности и времени работы представляется более целесообразным, если не необходимым, тестирование на полное соответствие сгенерированных чисел, полученных существующими образцами кода на CPU и разрабатываемой для GPU программы. При этом, естественно, следует использовать одно и то же исходное состояние.

Принимая во внимание результаты сравнения, можно сделать ряд выводов касательно применения ГПСЧ, основанных на алгоритмах реализации по методу Монте-Карло. В первую очередь стоит отметить, что криптографические ГПСЧ малоприменимы для подобных задач, по крайней мере, без серьезной переработки программы в целом. Они крайне медленны, сложны в реализации и невероятно чувствительны к ошибкам, которые могут быть легко допущены, учитывая большой объём кода и объёмы изменений необходимых для адаптации кода.

С другой стороны, даже относительно простые линейные конгруэнтные генераторы показали хороший результат. Наиболее быстрые и качественные их вариации (как например вариация, предложенная в книге из серии Numerical Recipes [10]), могут быть предпочтительным выбором в рамках данной задачи. Но это вовсе не означает, что все простые генераторы являются панацеей для обсчета задач методом Монте-Карло. Регистр сдвига с линейной обратной связью, несмотря на внушительный период, производил визуальные артефакты, являющиеся очевидным признаком проблем с генерируемыми числами.

Занимательно, что генератор Xoroshiro128+, имеющий, по сравнению с представленными линейными конгруэнтными методами, четырехкратный размер внутреннего состояния и большее количество операций в алгоритме, сумел обогнать в тестах по скорости сразу два линейных конгруэнтных метода. В целом, Xoroshiro128+ показал себя, как простой во внедрении, быстрый и мощный ГПСЧ.

Более сложный генератор WELL512, подобен Hash_DRBG в миниатюре – сильно проигрывает в скорости. Ни качество случайных чисел, ни значительный период не могут это скомпенсировать этот недостаток и вывести его на конкурентоспособный уровень.

В качестве итоговых выводов, касающихся методики выбора ГПСЧ для применения с методом Монте-Карло, можно привести следующие:

– в первую очередь, следует руководствоваться скоростью алгоритма. Качественного результата можно добиться подъёмом количества итераций, что суммарно выйдет быстрее и качественнее, чем генерация более качественных чисел более сложными генераторами;

– даже простые ГПСЧ, вроде линейных конгруэнтных генераторов, при хорошо подобранных константах дают весьма неплохие числа;

– не каждый простой ГПСЧ подойдет для решения обозначенной в работе задачи, предварительно следует проверять их качество;

– криптографические генераторы (даже с ускорениями и отключением функций, направленных на безопасность) слишком медленные, а их выигрыш по качеству ничтожен по сравнению с проигрышем по скорости;

– даже в тех случаях, когда алгоритм выглядит сложнее, не факт, что он медленнее, чем более простой, ибо вовлечено гораздо большее количество факторов, нежели простое количество операций.

V. Заключение

В статье изложены результаты проведённого сравнительного анализа ряда алгоритмов генерации псевдослучайных чисел, используемых для решения задачи вычисления интеграла методом Монте-Карло.

Выполнено сравнение по ряду параметров, которые позволяют оценить качество генераторов в рамках реальной задачи, а именно простоту внедрения, скорость работы программы с внедренным алгоритмом и влияние качества сгенерированных чисел на результат.

На основе результатов проведённого анализа делается ряд выводов относительно сильных и слабых сторон алгоритмов и принципа подбора генераторов в целом.

Дальнейшее развитие работы подразумевает исследование применимости алгоритмов для других задач, включение в рассмотрение иных параметров, а также дальнейшую стратификацию алгоритмов по предложенным показателям и параметрам.

Библиография

[1] Kajiya J. T. The rendering equation// Proceedings of the 13th annual conference on Computer graphics and interactive techniques, ACM. 1986. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.1402&rep=rep1&type=pdf (Дата обращения: 28.07.2018).

[2] Philip Dutré. Global Illumination Compendium// Computer Graphics, Department of Computer Science, Katholieke Universiteit Leuven, September 29, 2003. URL: https://people.cs.kuleuven.be/~philip.dutre/GI/TotalCompendium.pdf (Дата обращения: 28.07.2018).

[3] Сесин, И.Ю. Выбор генератора псевдослучайных чисел для использования в рендериге методом трассировки пути [Текст] / Сесин И.Ю., Нечаев В.В. — INJOIT, v. 5, № 8 (2017)
[4] Peter Alfke, Application Note // XAPP 052 July 7,1996 (Version 1.1) URL: http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf (Дата обращения: 28.07.2018).
[5] David Blackman, Sebastiano Vigna, Xoroshiro128plus C implementation. URL: http://vigna.di.unimi.it/xorshift/xoroshiro128plus.c (Дата обращения: 28.07.2018).
[6] Chris Lomont, Random Number generation, 2008. URL: http://lomont.org/Math/Papers/2008/Lomont_PRNG_2008.pdf
[7] Stephan Mueller, Deterministic Random Bits Generator implementation. URL: https://github.com/torvalds/linux/blob/master/crypto/drbg.c (Дата обращения: 28.07.2018).
[8] Mozilla Foundation, Deterministic Random Bits Generator implementation. URL: https://dxr.mozilla.org/nss/source/nss/lib/freebl в первую очередь, /drbg.c (Дата обращения: 28.07.2018).
[9] Timeout detection in the Windows Display Driver Model URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/display/timeout-detection-and-recovery (Дата обращения: 28.07.2018).
[10] Numerical Recipes in C. The Art of Scientific Computing, 2nd Edition, 1992, ISBN 0-521-43108-5.

Comparison of pseudorandom number generators for use with Monte Carlo-based rendering

I.Y. Sesin, V.V. Nechaev

Abstract — **Path tracing is one of unbiased rendering methods. Said method relies on integral rendering equation to simulate wide variety of optical phenomena. In path tracing, integral part of rendering equation is calculated using Monte Carlo method, which makes the method very resource-intensive. Taking in account that the rendering method is perfectly parallel, it is appropriate to take advantage of GPGPU methods to leverage GPU processing power for the calculations. However, Monte Carlo method requires random numbers to operate, and truly random numbers are difficult to obtain in bulk, especially on GPU. This leads to the crux of the problem: which pseudorandom number generator algorithm would be preferable to use on GPU with Monte Carlo method?**

This paper focuses on comparison between several PRNGs (pseudorandom number generators) being used with Monte Carlo method to solve integral rendering equation on GPU. The comparison is used to show advantages and disadvantages to using these PRNGs with Monte Carlo-based rendering methods, and draws some conclusions regarding picking correct tool for the job.

Keywords — **pseudorandom number generator, Monte Carlo method, path tracing.**