ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Факультет информатики, математики и компьютерных наук**

**Программа подготовки бакалавров по направлению   
09.03.04 Программная инженерия**

*Канделов Дамир Русланович*

**КУРСОВАЯ РАБОТА**

Сравнение различных стратегий выборки в рендеринге трассировкой путей

|  |  |
| --- | --- |
|  | Научный руководитель  Старший научный сотрудник  программной инженерии  факультета компьютерных наук  канд. физ. мат. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И.С. Бычков |

Нижний Новгород, 2024

**Содержание**

[1. Введение 3](#_Toc168178560)

[2. Постановка задачи 5](#_Toc168178561)

[3. Исследование предметной области 6](#_Toc168178562)

[3.1. Что такое Path tracing? 6](#_Toc168178563)

[3.2. Отличие Path tracing от Ray tracing 7](#_Toc168178564)

[3.3. Алгоритм работы Path tracing 8](#_Toc168178565)

[3.4. Случайные числа в алгоритме Path tracing 11](#_Toc168178566)

[4. Описание решения 13](#_Toc168178567)

[4.1. Реализация программы рендеринга изображений 13](#_Toc168178568)

[4.1.1. Написание базовой логики. Первые изображения. 13](#_Toc168178569)

[4.1.2. Усовершенствование программы рендеринга. 14](#_Toc168178570)

[4.2. Использование случайных чисел в 3D движке 16](#_Toc168178571)

[4.3. Генерация случайных чисел 16](#_Toc168178572)

[4.3.1. Halton sampler 17](#_Toc168178573)

[4.3.2. Halton Sampler с использованием RandomDigit скремблинга 19](#_Toc168178574)

[4.3.3. Halton Sampler с использованием Owen скремблинга 21](#_Toc168178575)

[5. Библиографический список 22](#_Toc168178576)

# Введение

В настоящее время компьютерная графика, а именно методы рендеринга изображений используются практически повсеместно. Рендеринг – это процесс генерации фотореалистичных или нереалистичных изображений по заданным 2D или 3D моделям, с помощью компьютерных программ. Он используется для создания CGI-изображений (computer-generated imagery), в киноиндустрии для создания визуальных эффектов (VFX), в разработке компьютерных игр и симуляторов для отрисовки сцен, которые пользователь будет видеть на экране, а также в сферах дизайна и архитектуры.

Существует множество алгоритмов рендеринга для создания фотореалистичных изображений, наиболее известными являются:

Rasterization (Растеризация) – это процесс перевода изображения, описанного в векторном формате, в растровое изображение. По сравнению с другими техниками рендеринга является одной из наиболее быстрых, из-за чего используется в большом количестве 3D движков, отрисовывающих изображение в реальном времени. Однако, минусом данной техники является возможность показать очень малое число оптических эффектов.

Ray tracing (Трассировка лучей) – это метод запуска «лучей» из камеры в каждый пиксель изображения, которые по мере своего движения (отражения и преломления от объектов) собирают их цвета и получает итоговый цвет пикселя. Ray tracing способен симулировать большое количество оптических эффектов, таких как отражения, преломления и дисперсию света, мягкие тени, глубину резкости, размытие в движении, каустику.

Path tracing (Трассировка путей) – этот метод похож на Ray tracing, однако он запускает большее число лучей через каждый пиксель, используя при этом методы Монте Карло, а также собирает информацию от всех источников света, которые освещают данный пиксель. Поэтому path tracing является наиболее близким к реальности по поведению глобального освещения. Если метод используется совеместно с физически корректными материалами и объектами, то в результате получаются изображения, неотличимые от реальных фотографий.

Основными потребителями алгоритмов рендеринга являются кино- и гейм-дев индустрии. Последние тенденции в обоих индустриях – создание максимально реалистичной графики. Как можно заметить, алгоритм Path tracing подходит больше всего для решения этой задачи. Однако, основным его недостатком является большая вычислительная сложность. Было замечено, что от метода выбора случайных чисел во время работы алгоритма зависит качество, а также время получения изображений хорошего качества.

Таким образом, востребованность и актуальность темы данной работы обусловлена потребностями сферы разработки игр и сферы создания визуальных эффектов в кино, нацеленных на получение реалистичной графики в своих проектах за сравнительно малое количество времени.

# Постановка задачи

В данной работе будет подробно рассмотрен алгоритм Path tracing. Для большего понимания его работы будет написан собственный 3D движок для рендеринга изображений и наглядного сравнения изменений в результатах при разных подходах.

На примере его работы мы разберем и сравним несколько наиболее популярных подходов к генерации случайных чисел.

Цель данного проекта: написать программу для рендеринга изображений с использованием алгоритма Path tracing, реализовать несколько подходов к генерации случайных чисел, добавить эту реализацию в программу для рендеринга изображений, сравнить изображения с помощью метрик и выделить методы, которые выдают наилучший результат за наименьшее время.

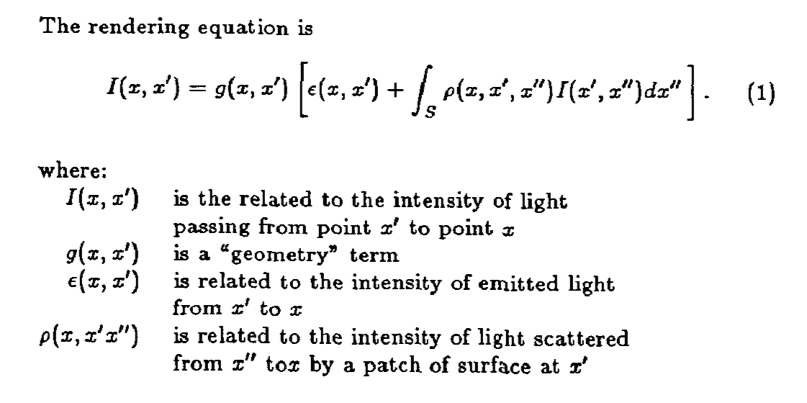
Задачами данного проекта являются:

1. Разобраться в алгоритме работы Path tracing.
2. Узнать, где в алгоритме используются случайные числа и почему метод их выбора влияет на результирующие изображения.
3. Написать собственную программу для рендеринга изображений с возможностью выбора используемого метода генерации случайных чисел.
4. Реализовать несколько подходов к генерации случайных чисел и имплементировать их в собственном проекте.
5. Сгенерировать и сравнить изображения, полученные с помощью новых методов выбора случайных чисел.

# Исследование предметной области

## Что такое Path tracing?

Path tracing – это метод рендеринга изображений по 3D сценам в компьютерной графике, который стремится воссоздать изображение максимально близкое к реальности. Данный метод реализует все визуальные эффекты, которые возможно представить на изображении.



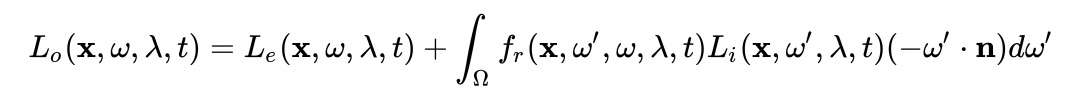
Оригинальное уравнение рендеринга, представленное в одноименной статье

«The Rendering Equation» Jim Kajiya, 1986

В 1986 году Jim Kajiya выпустил статью под названием «The Rendering Equation», которая объединяла компьютерную графику и физику. В ней было представлено уравнение, описывающие поведение света и позволяющее точно представить, как свет движется по 3D сцене. В этой же статье было и название нового метода рендеринга, который использовал данное уравнение: **Path tracing.**

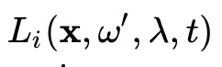
Полученное уравнение рендеринга было довольно небольшим и понятным, однако решить его не так просто из-за того, что сцены в компьютерной графике очень сложны и описываются миллионами или даже миллиардами треугольников в современном мире. Нельзя полностью решить это уравнение.

Jim Kajiya предложил решение этой проблемы: не требуется решать уравнение напрямую, его можно решить для каждого отдельного луча (например, с помощью алгоритма Ray tracing).



Другая форма уравнения рендеринга, на которой

более явно видны все составляющие

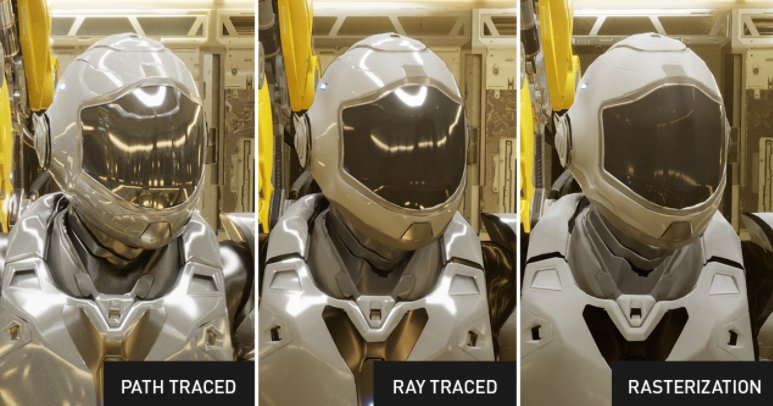
- длина волны λ

Соответственно, получение фотореалистичных изображений возможно, если решить это уравнение для достаточного количество лучей.

## Отличие Path tracing от Ray tracing

Одним из наиболее частых вопросов при знакомстве с методами рендеринга является вопрос отличия Path tracing и Ray tracing. Как уже было сказано ранее, в алгоритме Path tracing используется Ray tracing, однако следует разделять эти два метода рендеринга.

Основное отличие Path tracing от Ray tracing в том, что в трассировке путей мы следуем за лучом до тех пор, пока он не достигнет источника, чтобы получить цветовую и световую характеристику. А также, зачастую в Path tracing используется большее число лучей, получившиеся цвета которых каким-то образом образуют один цвет (обычно берется среднее значение). В результате чего мы получаем более физически-корректное и реалистичное изображение.



Сравнение Path tracing и Ray tracing [1]

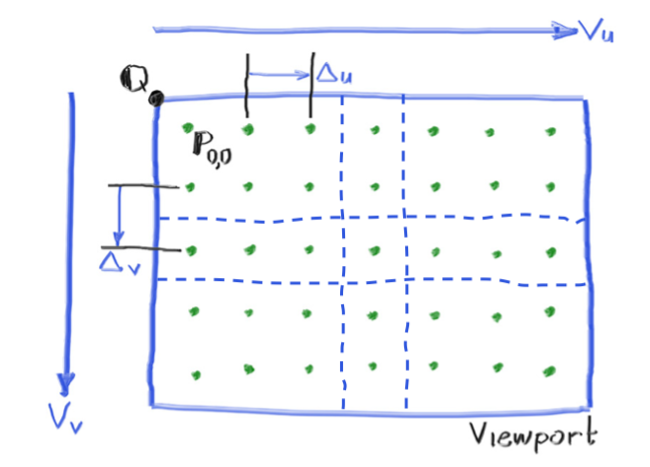
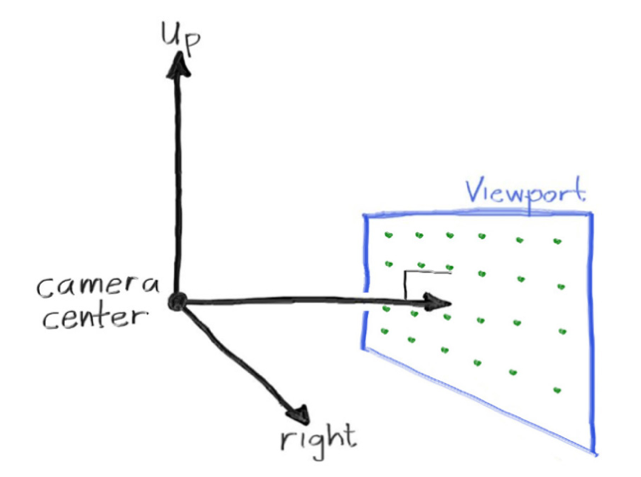
Как мы видим, в данном примере явно заметно, что Path tracing симулирует большее количество света, что гораздо ближе к реальности: в отражениях мы должны хорошо видеть все источники света.

## Алгоритм работы Path tracing

Давайте разберемся, как же это работает. Как было сказано выше, алгоритм максимально приближен к тому, как это происходит в реальном мире. Но с одним исключением.

В реальном мире свет распространяется от источника света, отражается от объектов и попадает на сетчатку глаза. Однако реализация такого поведения имеет слишком большую вычислительную сложность, так как требуется запустить лучи во все стороны от источника света (потому что свет распространяется во все стороны). Поэтому в компьютерной графике было принято решение развернуть эту последовательность действий: запускать лучи не из источников света, а из камеры (позиции, откуда пользователь смотрит на 3D сцену).

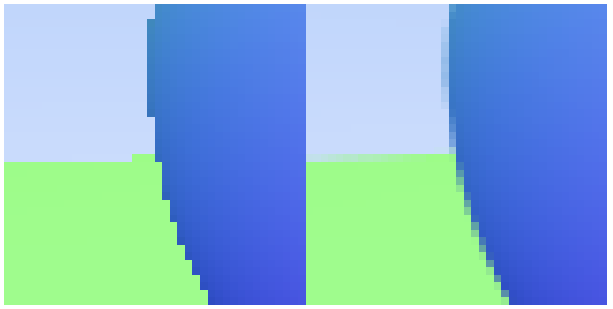
Далее, требуется понять, куда запускать лучи из камеры в пространстве.



Генерация лучей из камеры [2]

Заранее задаются параметры камеры: размер изображения, количество пикселей и т.д. На этом этапе мы создаем Viewport – матрица пикселей изображения, расположенная между камерой и сценой. На ней мы явно выделяем пиксели. Чтобы получить цвет пикселя требуется запустить луч через этот пиксель.

Здесь мы и встречаемся с первым примером использования случайных чисел: если запускать лучи только через центры пикселей (зеленые точки на рисунке), то мы можем неправильно вывести цвет для данного пикселя. Самый простой пример такого явления – рендеринг краев объектов, например шара, когда мы запускаем один луч через центр, мы получаем только один цвет. Допустим луч попал в шар, то есть, грубо говоря, функция вернет цвет шара, а луч через следующий пиксель, уже не попадет в шар и вернет цвет фона. Получается грубая граница объекта (левая картинка). Это отличается от того, как мы видим это в жизни. Здесь на помощь приходит эффект сглаживания (Antialiasing): будем запускать не один луч через центр пикселя, а несколько через разные точки внутри пикселя. Чтобы получить эти точки нам нужно взять случайные координаты внутри пикселя, затем нужно просто взять среднее значение от получившихся цветов и записать в текущий пиксель.

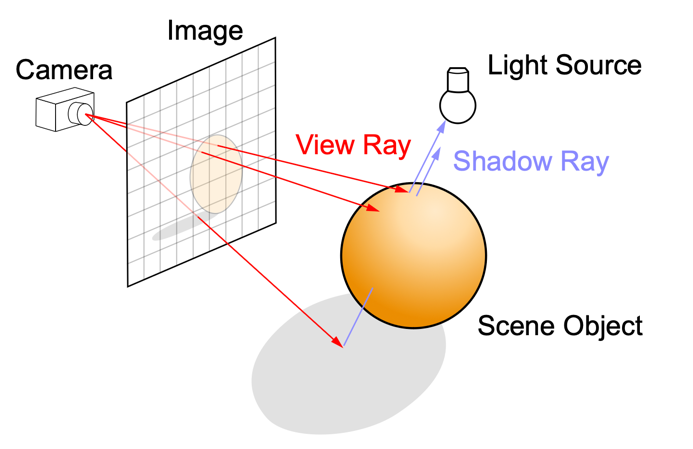


Эффект сглаживания (Antialiasing) [2]

(слева из камеры летит один луч через центры пикселей,

справа – несколько лучей, через случайные точки внутри пикселя)

После запуска луча из камеры, есть два варианта: 1. Он не достигнет никакого объекта на сцене (сцены бывают разными, необязательно они полностью окружены объектами); 2. Луч пересечет какой-то объект. Во втором варианте, в зависимости от свойств материала объекта, может произойти отражение или преломление, все это выполняется в точности, по законам физики. Во время такого пересечения луч запоминает цвет объекта, а также освещенность от источников света в данной точке, и продолжает свое движение. Далее, может быть либо достигнут предел количества отражений (его устанавливают для уменьшения времени работы алгоритма), либо достигнутым объектом станет источник света.

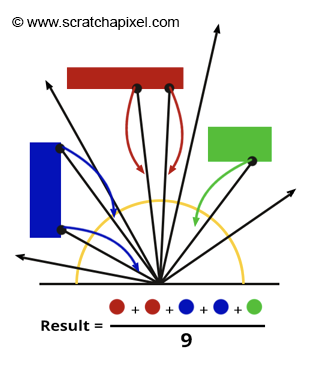


Пример лучей из камеры и получения цвета пикселя [3]

(View Ray – это лучи, которые мы запустили из камеры,

Shadow Ray – это лучи, направленные от точки пересечения к источнику света, чтобы узнать освещенность в текущей точке.)

Все полученные данные в ходе отражений и преломлений луча в дальнейшем используются в Уравнении рендеринга, и мы получаем итоговый цвет пикселя. Затем мы повторяем эту операцию, но уже с другими случайными числами: координатами точки в пикселе, через которую летит луч; направлением луча при отражении от Ламбертова материала и т.д.



Получение цвета пикселя, исходя из нескольких

запущенных лучей в разные стороны [4]

(берется среднее значение из цветов всех запущенных лучей)

В результате всех вышеописанных действий мы и получаем итоговый результат – срендеренное изображение. Теперь чуть больше углубимся в процесс использования случайных чисел.

## Случайные числа в алгоритме Path tracing

При рендеринге изображений случайные числа используются на нескольких этапах, каждое использование случайного числа в отдельном случае мы будем называть dimension (размерность):

* Выбор начальной точки для запуска луча через пиксель (два разных дименшена – координата по Ox и по Oy)
* Выбор направления отраженного луча от Ламбертова материала (два разных дименшена – координата по Ox и по Oy)

Также зачастую случайные числа используются еще в нескольких местах:

* Выбор источника света (один дименшен, отвечающий за индекс)
* Два дополнительных дименшена, отведенных под материалы (в зависимости от сложности материалов в сцене, в моем проекте используются только два из них, для выбора направления луча)

Соответственно, с этими случайными числами мы и будем работать.

# Описание решения

## Реализация программы рендеринга изображений

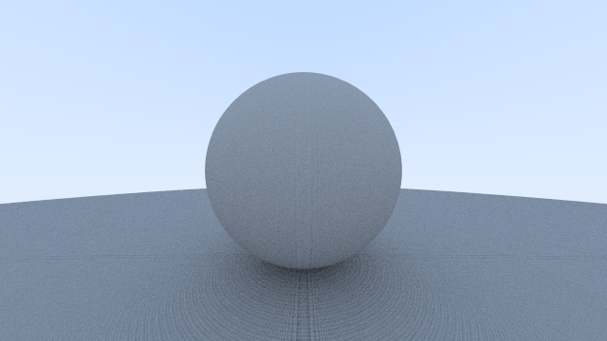
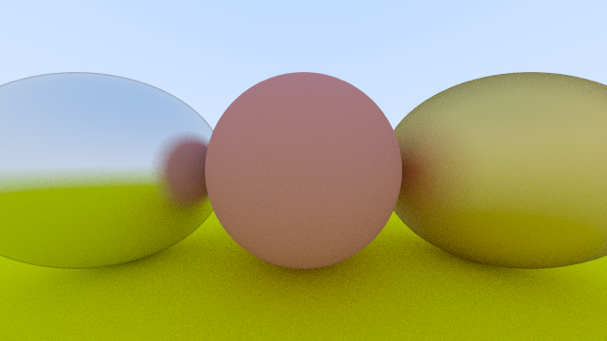
В ходе работы над курсовой работой, для лучшего понимания алгоритмов рендеринга был написан собственный 3D движок для рендеринга изображений на языке C++ с нуля.

Это помогло понять на практике, как работает рендеринг изображений, какие техники применяются. Во время написания кода тестировались и основополагающие, базовые подходы, существующие в графике, и более продвинутые, современные, которые используются в современных 3D движках.

### Написание базовой логики. Первые изображения.

При написании собственного проекта требовалось понять, с чего начать и как дойти до результата, который бы позволил не только рендерить сцены на уровне использующихся сегодня 3D движков, но и позволял бы протестировать нашу основную гипотезу по ускорению рендеринга изображений за счет различных стратегий выбора случайных чисел.

За основу была взята серия книг Ray Tracing in One Weekend (Peter Shirley, Trevor D Black, Steve Hollasch). На первом этапе была реализована базовая логика:

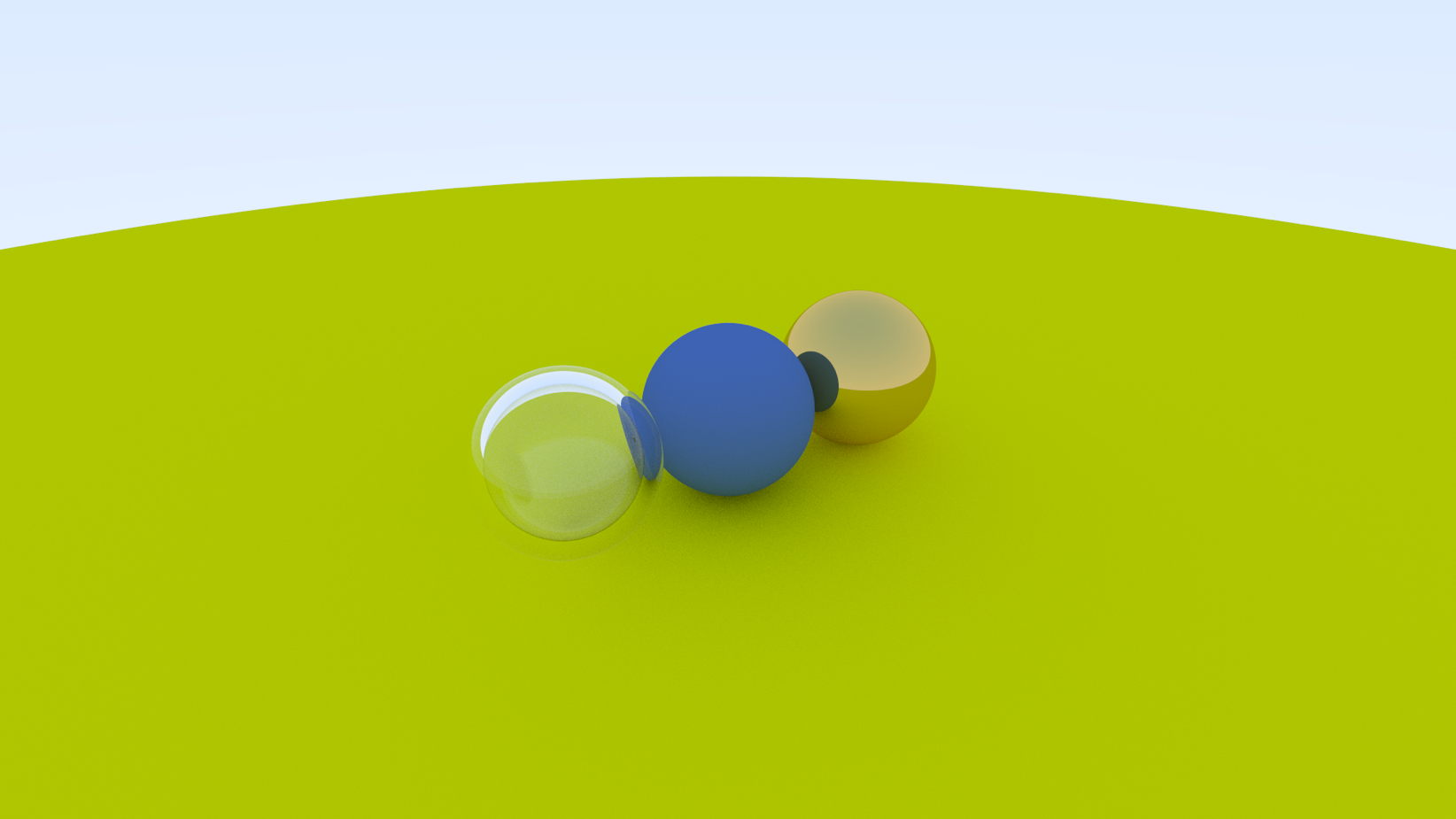
 

Первые срендеренные изображения

1. Работа лучей из камеры, аккумулирующих свет, из-за чего мы видим тени
2. Отражение с эффектом размытия – то, что характерно для металла

* отрисовка простейших геометрических форм (шар);
* создание стандартных материалов: металл, стекло, ламбертова поверхность (Lambertian surface);
* движение лучей из камеры, аккумулирующих цвет и свет от пересеченных объектов;
* пересечение лучей с объектами, преломление и отражение от различных материалов

важно сказать, что отражение происходит по разному, в зависимости от отражающих свойств материалов – для некоторых действует закон угол падения равен углу отражения (например, стекло), однако в большинстве случаев, отраженный луч отклоняется от этого угла на некоторую случайную величину (как пример, метал)

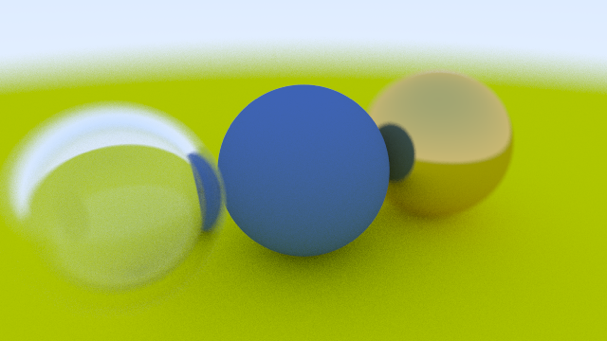
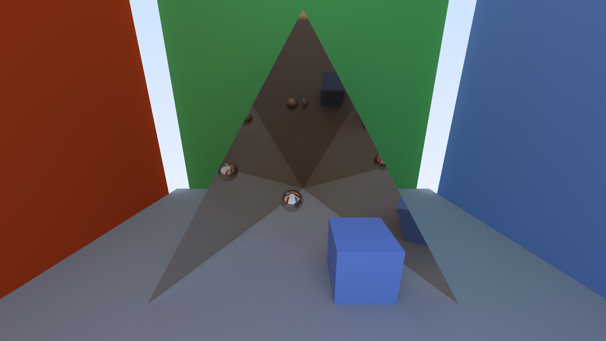


Стандартные материалы: стекло, Lambertian material, металл

### Усовершенствование программы рендеринга.

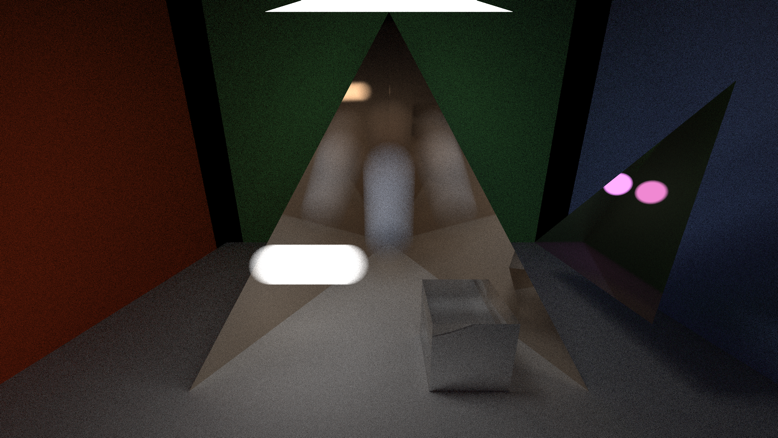
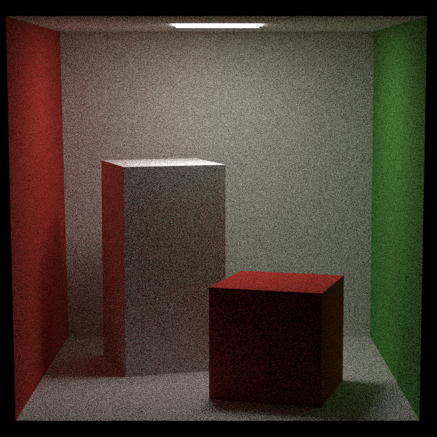
На втором этапе стояла задача:

* добавить новые геометрические формы – треугольники, чтобы была возможность отрисовывать более сложные сцены;
* реализовать эффекты, которые используются в современных 3D движках: глубина резкости, размытие в движении;

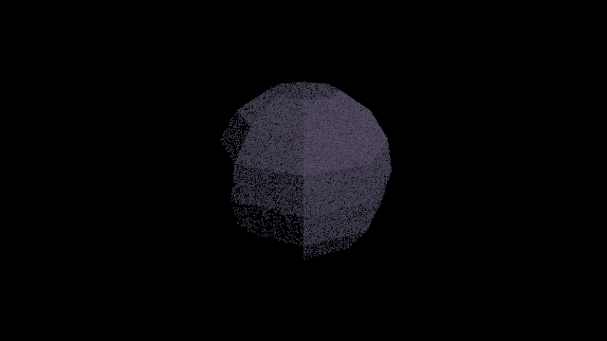
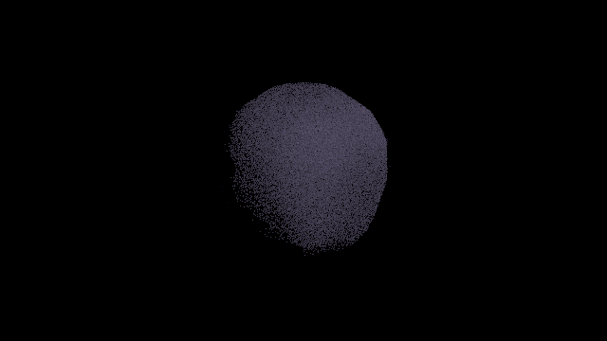
1. Эффект глубины резкости
2. Добавление в сцену треугольников

* добавить возможность загрузки сцен и материалов из файлов (в данном случае, использовались форматы .obj / .mtl с применением библиотеки tinyobjloader);
* добавить материалы со свечением, источники света;

1. Добавление источников света, а также размытия в движении
2. Сцена и материалы, загруженные из файла

* добавить поддержку и обработку нормалей у вершин треугольников (они используются для рендеринга закругленных форм);
* обновить метод получения освещенности, в зависимости от дистанции от источника света до объекта;

1. Шар из треугольников без нормалей в вершинах
2. Шар из тех же треугольников с поддержкой нормалей

* изменить использующуюся цветовую модель на более современную и более подходящую для нашей задачи модель Фонга (Phong model)

Эта модель гораздо более правильная с точки зрения физики, а так как мы хотим добиться изображений максимально приближенных к реальности, то использовать ее было просто необходимо. Также, в .mtl файлах материалы задаются именно в таком формате.

После того, как все вышеперечисленные пункты были сделаны, у нас появилась единственная функция, которая отвечает за всю генерацию случайных чисел в нашем движке.

## Использование случайных чисел в 3D движке

Во время написания кода 3D движка, были явно замечены места, где используются случайные числа в нашем коде, они уже были подробно описаны ранее, поэтому здесь только их краткое описание:

* Начальная точка в пикселе, через которую пройдет луч, а также выбор направления, в котором полетит отраженный от объекта луч – дименшены 0 и 1;
* Материалы – выбор направления при случайном отражении от поверхности (например, от Ламбертовой поверхности) – дименшены 5, 6;
* Время запуска луча – требуется для создания эффекта размытия в движении, чтобы однозначно понимать, в каком месте находится объект в данный момент времени – дименшен 9.

## Генерация случайных чисел

Среди различных подходов к генерации случайных чисел были выбраны: Halton sampler, Scrambled Halton sampler (с использованием RandomDigit и Owen скрэмблинга) и BlueNoiseSampler. Также для проведения сравнения, использовался встроенный генератор случайных чисел из C++, так как по умолчанию в проектах чаще всего используется именно он.

Как было написано выше: наша цель - сгенерировать хорошо распределенный рандомизированный набор точек, то есть набор точек, в котором почти отсутствуют точки, находящиеся очень близко друг к другу, и при этом нет регионов на координатной плоскости без точек.

### Halton sampler

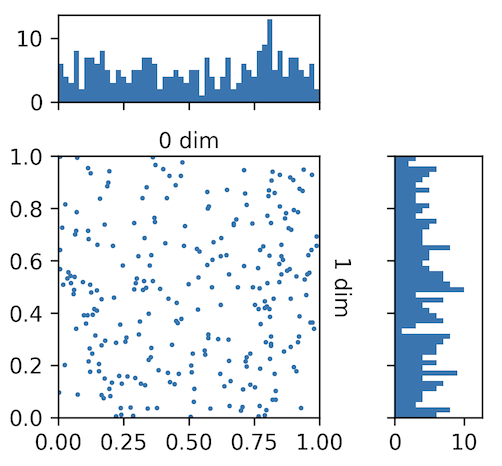
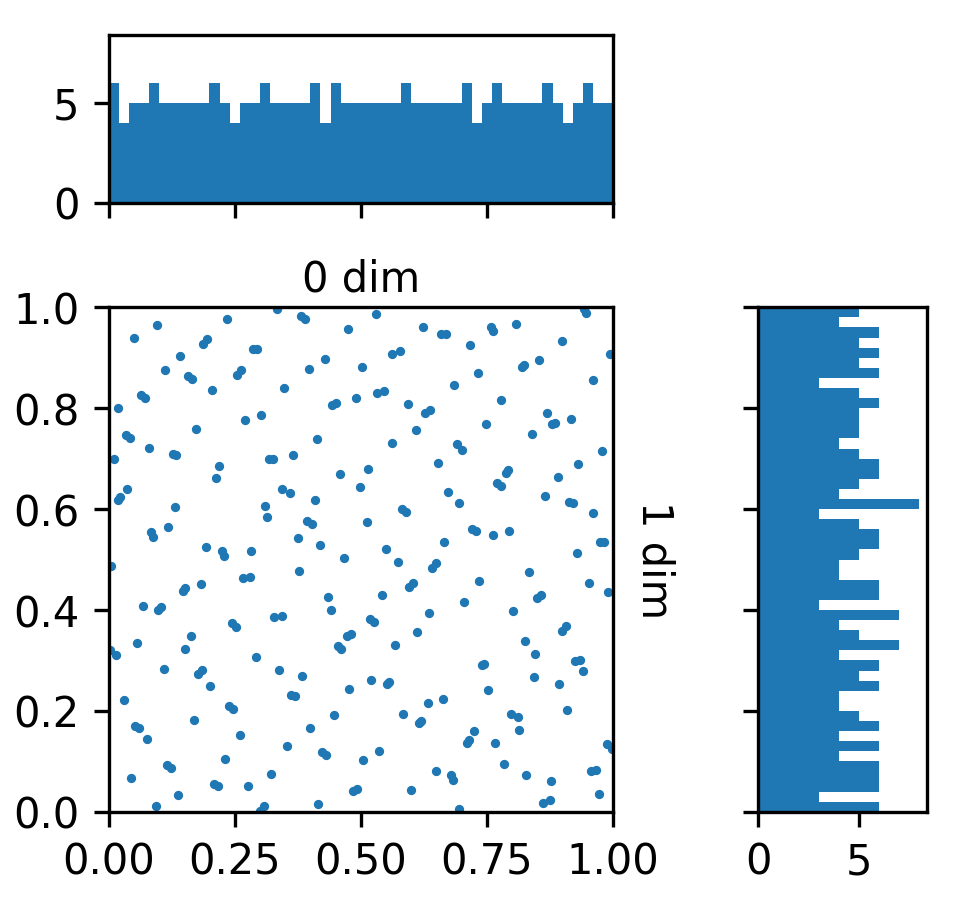
Первый алгоритм, который нам подходит - генератор случайных чисел Хальтона. Он генерирует последовательность точек с низким расхождением, которые последовательно хорошо распределены. Генерация таких точек основана на развороте битов числа (radical inverse) в системе счисления по основанию некоторого заранее заданного простого числа. Для разных дименшенов используются разные простые числа. В нашем случае для первых 32 дименшенов мы использовали первые 32 простых числа (2, 3, 5, 7, … 131).

Подробнее о самом алгоритме: Radical inverse – это разворот битов числа, то есть, если у нас было число , где di – i-ый разряд числа (цифра от нуля до максимальной цифры в некоторой системе счисления b), то после разворота битов у нас получится

Более формально:

Как можно заметить после переворота нашего числа мы получаем новое число, которое лежит в полуинтервале [0; 1).

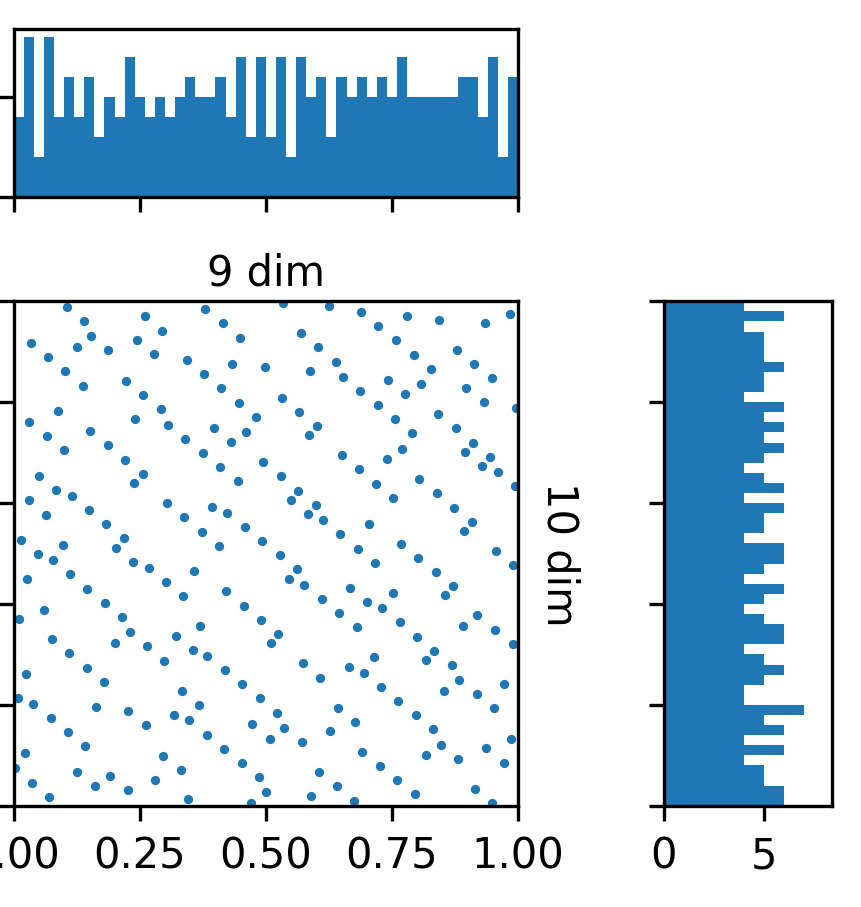
Чтобы получить последовательность Хальтона с низким расхождением нам требуется посчитать RadicalInverse от последовательности натуральных чисел.

Слева – точки, сгенерированные стандартным генератором

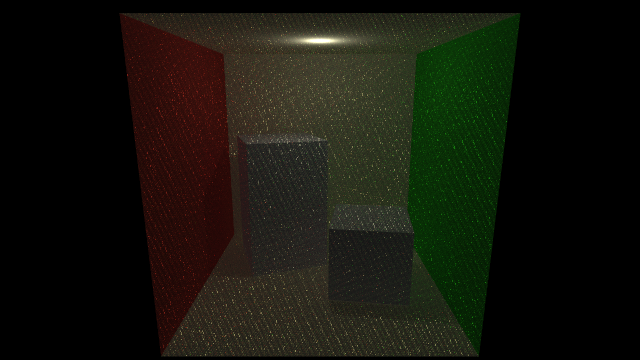
Справа – Halton Sampler (нулевой и первый дименшен)

Построим графики распределения точек в первых двух дименшенах. Явно видно, что последовательность Хальтона более хорошо покрывает пространство. То же самое показывают графики справа и сверху, которые демонстрируют распределение точек каждого отдельного дименшена. Однако тут явно заметны и минусы Halton Sampler – точки кучкуются и образуют некоторый паттерн, что очень плохо для нашего движка, так как если соседние пиксели будут с одним паттерном, то этот паттерн станет заметен и на самом изображении.

Halton Sampler для дименшенов 9 и 10

Однако, для следующих дименшенов ситуация становится еще хуже. В 9 и 10 дименшене мы получаем следующую ситуацию: точки становятся строго структурированными и появляется явно различимый паттерн.

1. Cцена, срендеренная с использованием Halton Sampler
2. Более крупно видимый паттерн на изображении

Как было сказано ранее, если при выборе случайных чисел виден некоторый паттерн, то некоторый паттерн и структурированность будет явно видна и на результирующем изображении. В качестве примера – изображение, при рендеринге которого использовался Halton Sampler. Мы получаем пиксели примерно одного цвета, которые вместе образуют целые линии на изображении.

### Halton Sampler с использованием RandomDigit скремблинга

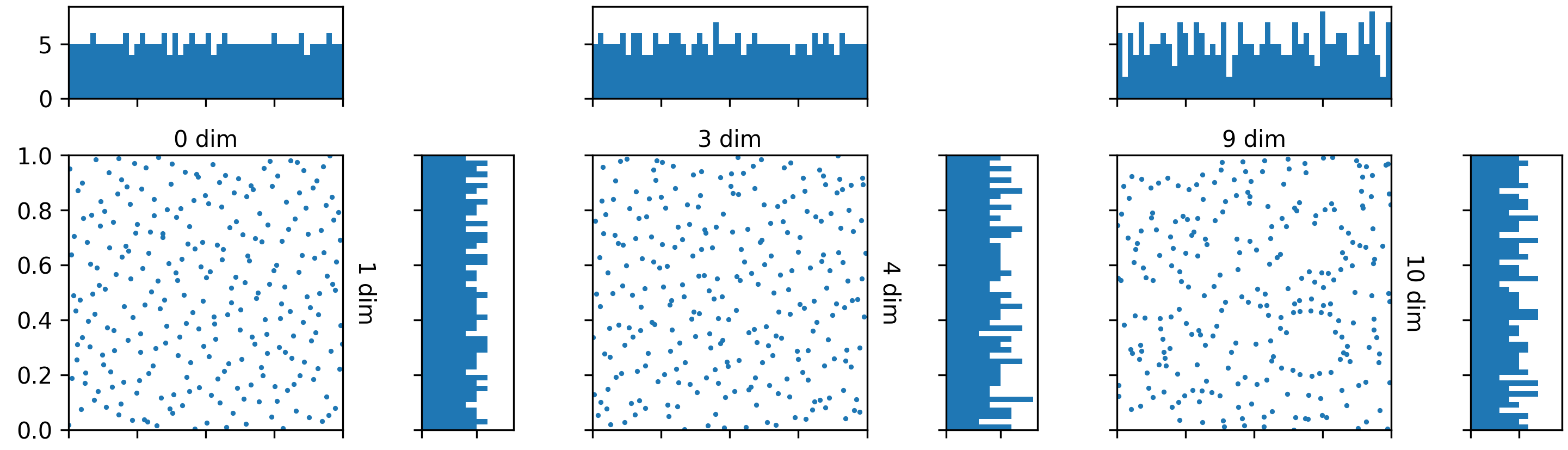
Чтобы убрать появляющиеся паттерны зачастую используется скремблинг, то есть перемешивание каким-то образом получающихся значений.

Первым и одним из наиболее простых и понятных является RandomDigit скремблинг.

Для него заранее генерируется массив, размер которого равен основанию текущей системы счисления (b). В нем, для каждой цифры из b-ичной системы счисления взаимно-однозначно сопоставляется случайно выбранная цифру этой же системы счисления. Аналогично делаем для всех дименшенов.

То есть, если мы работаем в системе счисления по основанию 5, то массив может быть, например [3, 0, 4, 1, 2]. Соответственно, когда при развороте битов мы берем некоторую цифру, то мы берем не эту цифру, а ту, которая находится в нашем массиве по индексу нашей цифры. Если мы встретили цифру 3, то возьмем вместо нее 1 из нашего массива.

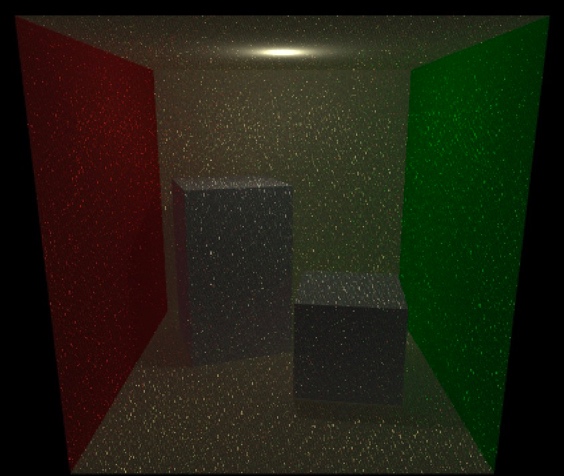
В результате, с помощью таких несложных операций (и по времени, и по дополнительной памяти), мы получаем следующие распределения:



Распределение случайных чисел по дименшенам (0-1, 3-4, 9-10)

с использованием RandomDigit скремблинга.

Как можно заметить, мы справились с задачей разбиения явно структурированных групп точек, однако все равно, в высоких дименшенах у нас не получается хорошее распределение – остаются области без точек.

1. Сцена срендеренная с использованием Halton RandomDigit алгоритма
2. То же самое место, что и в изображении Halton Sampler



Оставшийся паттерн на срендеренном изображении

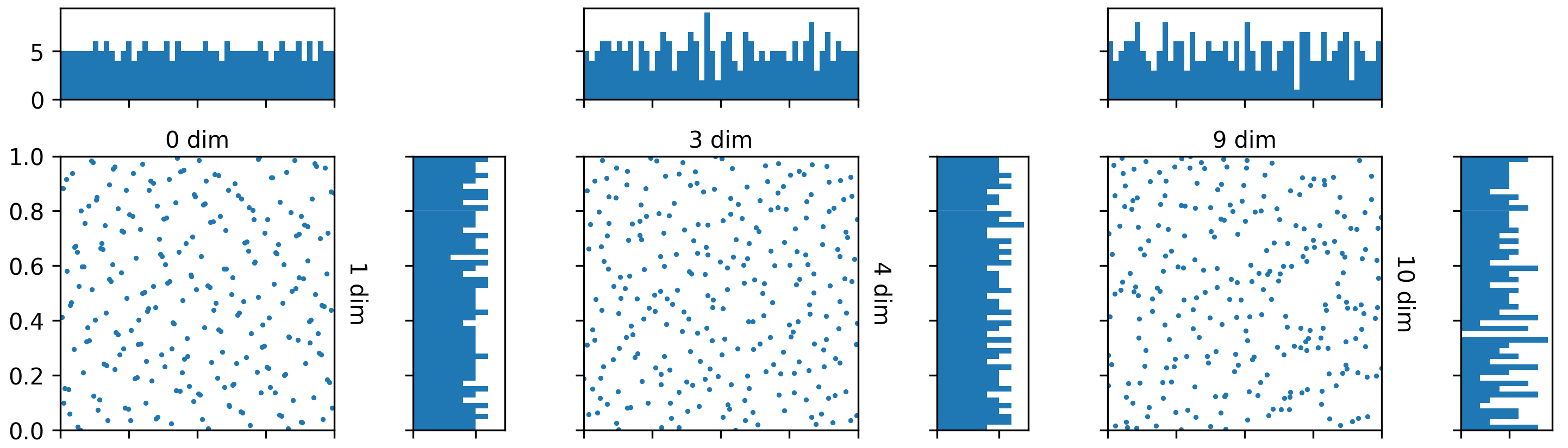
Если смотреть, на получившееся изображение, то можно заметить, что паттерн, который был в Halton Sampler мы на самом деле убрали и очень хорошо. Однако, появился другой, который менее заметен, но все равно остается – пиксели образуют менее различимые, но все еще различимые линии точек.

### Halton Sampler с использованием Owen скремблинга

Вторым более популярным и часто использующимся скремблингом является Owen скремблинг. Основная идея, в том что при получении следующего числа мы не только совершаем какую то операцию с текущим разрядом, но и смотрим и учитываем то, что у нас уже получилось до этого. Существуют разные имплементации, я использовал одну из них (из блога Psychopath Renderer), немного переделанную.

В данном алгоритме при развороте бита мы смотрим не только на текущую цифру, но и на уже развернутые биты. То есть алгоритм при развороте i-го бита следующий:

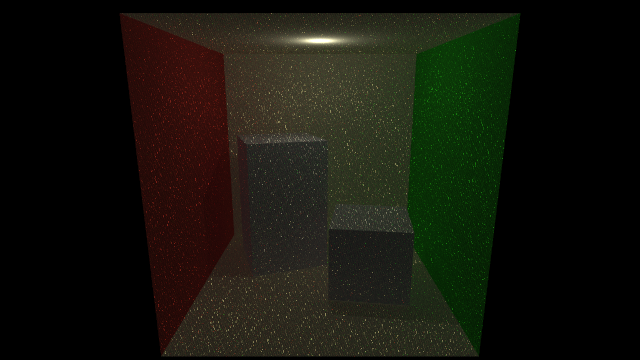
1. Посчитаем вспомогательное значение, равное XOR от заранее заданного числа (некоторая 32-битная константа, разная для каждого дименшена) и уже развернутых битов (результата на предыдущей итерации).
2. Перемешаем биты получившегося в прошлом пункте числа, используя Laine-Karras hash. Он заключается в том, что мы перемешиваем биты нашего числа с помощью операций XOR, умножения на нечетную константу, прибавления константы и XOR на наше число, умноженное на четную константу.
3. Чтобы получить новую цифру, мы обращаемся к массиву перестановок для нашей системы счисления (аналогичному тому, что используется в RandomDigit) по индексу равному сумме текущей цифры и получившемуся во втором пункте значения, все это берется по модулю от основания нашей системы счисления.



Распределение случайных чисел по дименшенам (0-1, 3-4, 9-10)

с использованием Owen скремблинга.

В результате мы получаем довольно хорошую картинку распределения, областей без точек у нас получилось меньше чем в RandomDigit скремблинге. А также у нас практически нет структурированных паттернов. Соответственно, появляется надежда на то, что и результаты у нас будут лучше.



Изображение, срендеренное с использованием Owen скремблинга

На результирующем изображении видно, что мы пришли к довольно хорошему результату.

### Blue Noise sampler

Последовательность с низким расхождением – последовательность, в которой доля элементов, попавших в произвольное множество B, пропорциональна усредненному значению некоторой меры множества B.

# Библиографический список

<https://blogs.nvidia.com/blog/what-is-path-tracing/>

<https://raytracing.github.io/books/RayTracingInOneWeekend.html>

<https://developer.nvidia.com/discover/ray-tracing#:~:text=This%20reverse%20tracing%20process%20of,light%20sources%20in%20multiple%20directions>.

<https://www.scratchapixel.com/index.html>

<https://www.techspot.com/article/2485-path-tracing-vs-ray-tracing/#:~:text=Traditional%20ray%20tracing%20involves%20calculating,off%20in%20a%20random%20direction>.

<https://dl.acm.org/doi/pdf/10.1145/15922.15902>

<https://raytracing.github.io/books/RayTracingTheNextWeek.html>

<https://raytracing.github.io/books/RayTracingTheRestOfYourLife.html>

<https://pbr-book.org/4ed/contents>

<https://jcgt.org/published/0009/03/02/>

<https://psychopath.io>

<https://www.scratchapixel.com/index.html>

<https://developer.nvidia.com/discover/ray-tracing#:~:text=Ray%20tracing%20generates%20computer%20graphics,back%20to%20the%20light%20sources>.

**Blue-noise Dithered Sampling** Iliyan Georgiev Marcos Fajardo