



Факультет информатики, математики и
компьютерных наук

Программная Инженерия

Нижний Новгород
2024

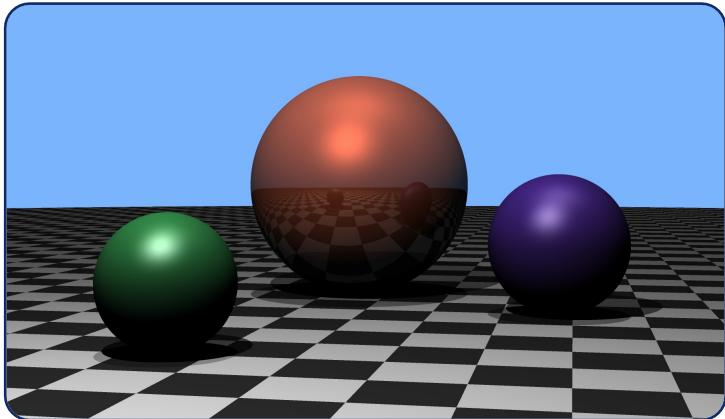
Сравнение различных стратегий выборки в рендеринге трассировкой путей

Научный руководитель:
к. ф.-м. н., старший научный сотрудник
Бычков Илья Сергеевич

Выполнил студент группы 22ПИ-1:
Канделов Дамир Русланович



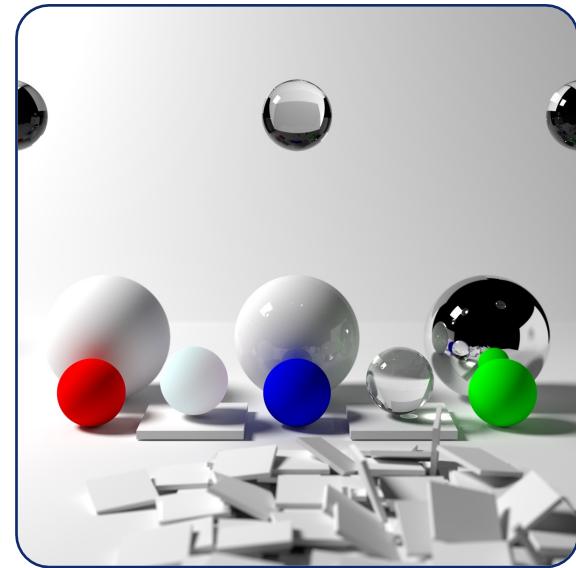
Графика и рендеринг в современном мире



Rasterization



Ray Tracing



Path Tracing

Актуальность

В современном мире среди сфер, тесно связанных с рендерингом изображений: сферы разработки игр, сферы создания визуальных эффектов (VFX) в кино, дизайна и архитектуры – существует потребность получения максимально реалистичных изображений за наименьшее время. Именно проблему ускорения процесса рендеринга мы и попробуем решить.



Цель работы

Изучение алгоритмов генерации случайных чисел и выделение алгоритмов, с помощью которых можно будет улучшить время рендеринга изображений хорошего качества.

Задачи

1. Разобраться в алгоритме работы Path Tracing
2. Узнать, где в алгоритме используются случайные числа и как они влияют на качество
3. Написать собственную программу для рендеринга с возможностью выбора метода генерации случайных чисел
4. Реализовать несколько подходов к генерации случайных чисел и имплементировать их в проекте
5. Срендерить изображения с использованием новых методов выбора случайных чисел
6. Провести комплексный анализ полученных изображений и сделать выводы о результатах использования стратегий выборки в рендеринге трассировкой путей



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, основываясь на физически верных формулах.

The rendering equation is

$$I(x, x') = g(x, x') \left[\epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right].$$

where:

- $I(x, x')$ is the related to the intensity of light passing from point x' to point x
- $g(x, x')$ is a “geometry” term
- $\epsilon(x, x')$ is related to the intensity of emitted light from x' to x
- $\rho(x, x' x'')$ is related to the intensity of light scattered from x'' to x by a patch of surface at x'

Оригинальное уравнение рендеринга, представленное в одноименной статье
«The Rendering Equation» Jim Kajiya, 1986

$$L_0(p, \omega_0) = \int_{\Omega} f(p, \omega_0, \omega_i) L_i(p, \omega_i) \cos\theta_i d\omega_i$$

Другая форма уравнения рендеринга

$L_0(p, \omega_0)$ – уходящие суммарное излучение света, отраженное в точке p в направлении ω_0

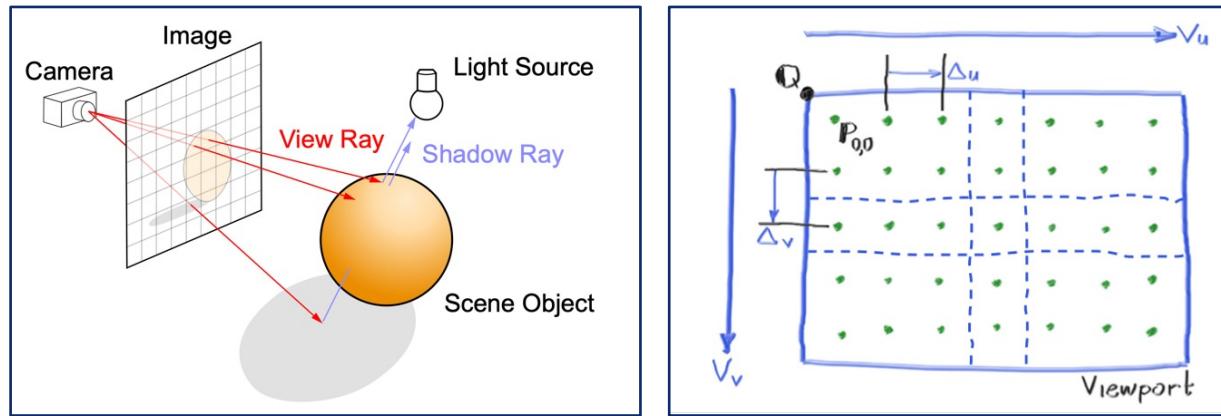
$f(p, \omega_0, \omega_i)$ – функция, которая определяет, количество излучения, которое отражается в точке p в направлении ω_0 от излучения, попадающего в точку p в направлении ω_i

$L_i(p, \omega_i)$ – световое излучение, падающее в точку p в направлении ω_i

$\cos\theta_i$ – закон косинусов Ламберта

Исследование предметной области

Алгоритм работы Path Tracing



1. Считывается сцена и задаются начальные параметры – размер изображения, расположение камеры
2. Через каждый пиксель запускаются лучи из камеры
3. При встрече с объектом луч собирает информацию о цвете пересеченного объекта и освещенности в данной точке
4. Далее луч отражается / преломляется от встречи с этим объектом и летит дальше, согласно законам физики
5. Луч продолжает свое движение до тех пор пока не дойдет до источника света, либо пока не достигнет лимита отражений
6. После того как мы полностью просимулировали движение запущенных лучей, мы высчитываем цвет данного пикселя исходя из цветов, полученных лучами

Случайные числа в алгоритме

- Выбор начальной точки на пикселе для запуска луча (2 различных дименшена – координата по Ox и по Oy)
- Выбор источника света, для которого мы рассматриваем освещенность в точке (1 дименшен – индекс источника)
- Выбор направления отражения луча от материала (3 различных дименшена – координаты Ox, Oy, Oz)
- Дополнительные дименшены для материалов, в зависимости от сложности материалов
- И другие...

Для полного решения уравнения рендеринга требуется бесконечное число дименшенов.



Исследование предметной области

Стратегии выборки в рендеринге

Следствие из методов Монте-Карло:

В зависимости от распределения случайных чисел могут получаться картинки разного качества. Чем лучше они распределены (то есть не должно быть двух точек слишком близко друг к другу, а также не должно оставаться пустых областей), тем лучше получается итоговое изображение.

Standard Sampler

В качестве стандартного генератора случайных чисел – генератора, использующегося в проектах по умолчанию, была взята встроенная в C++ генератор, на основе линейного конгруэнтного метода.

Этот генератор случайных чисел был открыт в 1969 году учеными P.A.W.Lewis, A.S.Goodman, J.M.Miller. А в 1988 был принят в качестве «минимального стандарта» Кетом Миллером и Стивеном Парком.

```
float StandardRand() {
    static std::linear_congruential_engine<uint32_t, 16807, 0, 2147483647> engine;
    static std::uniform_real_distribution<float> distribution(0, 1);
    return distribution(engine);
}
```

Реализация функции стандартного генератора случайных чисел

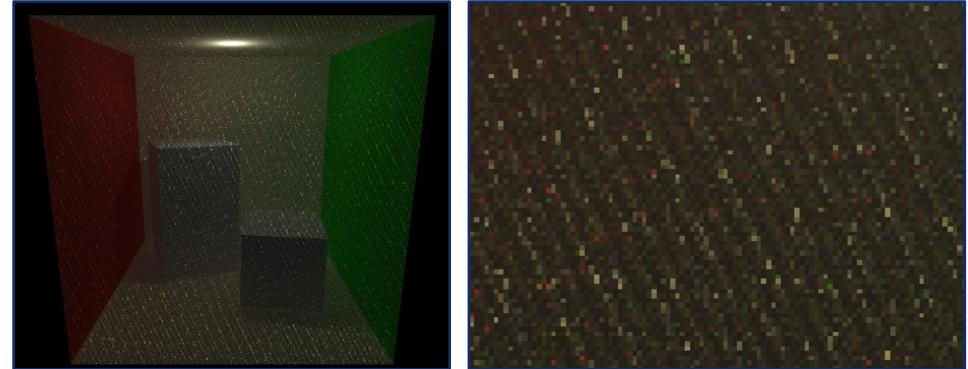
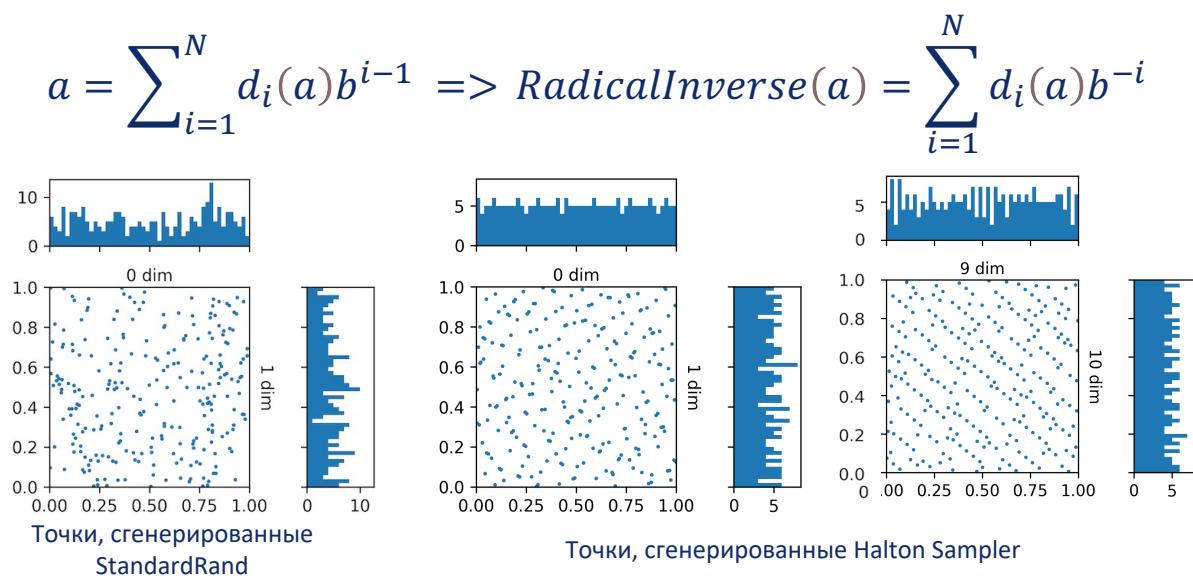


Исследование предметной области

Стратегии выборки в рендеринге

Halton Sampler

Генерация последовательности точек Хальтона с низким расхождением, которые последовательно хорошо распределены. Генерация таких точек основана на развороте битов числа (Radical Inverse) в системе счисления по основанию простого числа.



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

```
float HaltonRand(uint32_t value, const uint32_t base) {  
    const float invBase = (float) 1 / (float) base;  
    float result = 0;  
    float invBaseN = invBase;  
  
    while (value) {  
        uint32_t next = value / base;  
        uint32_t digit = value - next * base;  
        result += invBaseN * digit;  
        invBaseN *= invBase;  
        value = next;  
    }  
  
    return std::max(std::min(result, 1.f - EPSILON), 0.f);
```

Реализация Halton Sampler



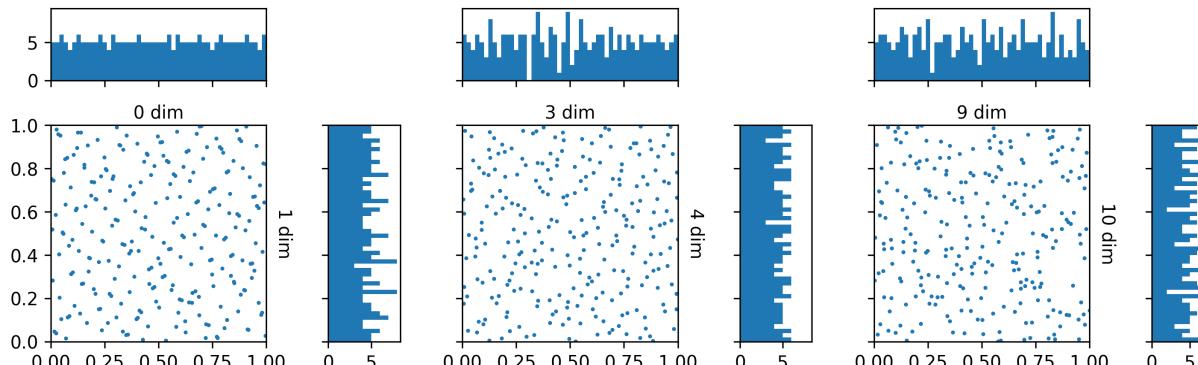
Исследование предметной области

Стратегии выборки в рендеринге

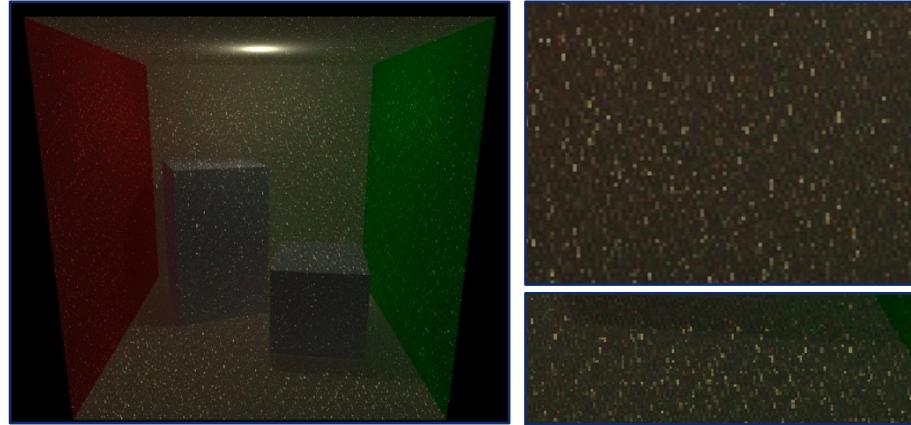
Halton Random Digit Sampler

Чтобы убрать появляющиеся паттерны зачастую используется скрембллинг, то есть перемешивание каким-то образом получающихся значений. Первым и одним из наиболее простых и понятных является RandomDigit скрембллинг.

Для него заранее генерируется массив, размер которого равен основанию текущей системы счисления. В нем, для каждой цифры из системы счисления взаимно-однозначно сопоставляется случайно выбранная цифра этой же системы счисления.



Точки, сгенерированные Halton RandomDigit Sampler



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

```
float HaltonRandomDigitScrambling(uint32_t value,
                                    const uint32_t base,
                                    const std::vector<uint32_t> &permutation) {
    const float invBase = (float) 1 / (float) base;
    float result = 0;
    float invBaseN = invBase;
    while (value) {
        uint32_t next = value / base;
        uint32_t digit = (value - next * base);

        digit = permutation[digit];

        result += invBaseN * digit;
        invBaseN *= invBase;
        value = next;
    }
    return std::max(std::min(result, 1.f - EPSILON), 0.f);
}
```

Реализация Halton RandomDigit Sampler

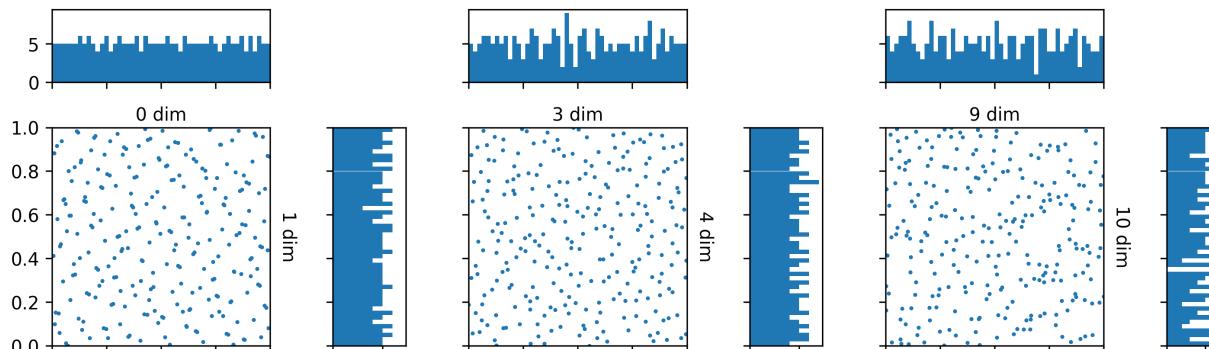


Исследование предметной области

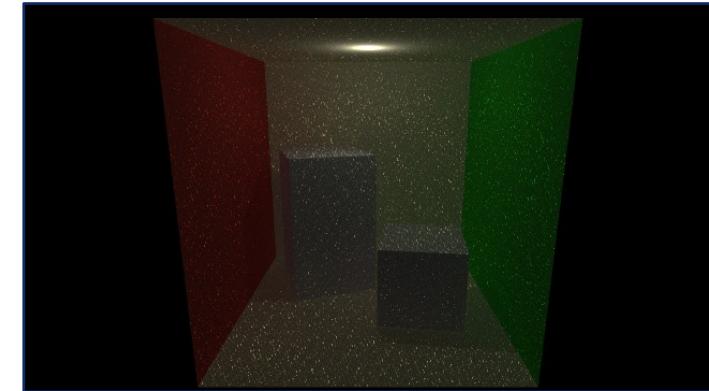
Стратегии выборки в рендеринге

Halton Owen Sampler

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 предыдущий результат.



Точки, сгенерированные Halton RandomDigit Sampler



Сцена срендеренная с использованием Halton Owen алгоритма

```
float HaltonOwenScrambling(uint32_t value,
                            const uint32_t base,
                            const std::vector<uint32_t> &permutation,
                            uint32_t hash) {
    const float invBase = (float) 1 / (float) base;
    uint32_t result = 0;
    float invBaseN = 1;
    while (value) {
        uint32_t next = value / base;
        uint32_t digit = value - next * base;

        uint32_t digit_hash = MixBits(hash ^ result);
        digit = permutation[(digit + digit_hash) % permutation.size()];

        result = result * base + digit;
        invBaseN *= invBase;
        value = next;
    }
    return std::max(std::min(result * invBaseN, 1.f - EPSILON), 0.f);
}
```

Реализация Halton Owen Sampler



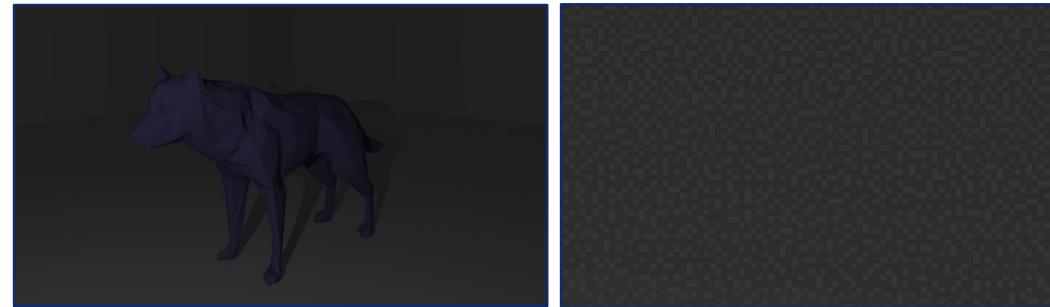
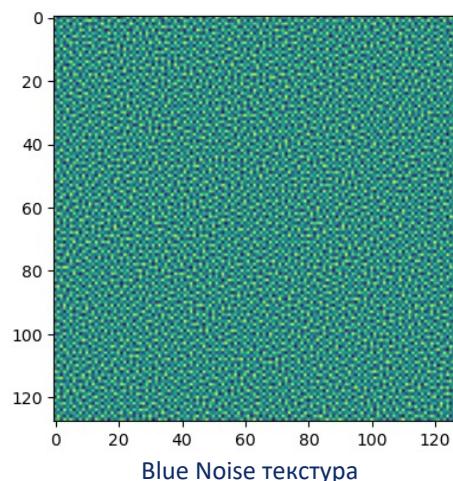
Исследование предметной области

Стратегии выборки в рендеринге

Blue Noise Sampler

Заключительным алгоритмом выбора случайных чисел стал Blue Noise sampler. Основная идея алгоритма – заполнить текстуру хорошо распределенными точками, сгенерировав некоторые начальные случайные точки и заполнив остальные точки, основываясь на том, что следующая точка должна находиться как можно дальше от всех существующих. В данном методе подход к генерации случайных чисел отличается от предыдущих: мы заранее генерируем некоторую текстуру с хорошим паттерном, а затем используем ее.

В результате у нас получается такая текстура, можно явно увидеть заметный и характерный для нашего алгоритма паттерн. В данном случае он играет положительную роль, так как, есть теории и статьи на тему того, что фоторецепторы в наших глазах лучше воспринимают и дорисовывают синий шум, что происходит из-за строения нашего глаза. Именно поэтому такой паттерн более хорошо согласуется с нашим восприятием и, если мы увидим его на картинке, то визуально результаты будут лучше, чем у других алгоритмов.



1. Изображение, сгенерированное с применением Blue Noise Sampler
2. Более крупно явно видимый паттерн, присутствующий на полу (добавлена яркость)

```
float BlueNoiseRand(const SamplerState &currentState, uint32_t Dim) {  
    size_t y = currentState.seed / IMAGE_WIDTH; // В currentState.seed хранится  
    size_t x = currentState.seed % IMAGE_WIDTH; // линейный индекс пикселя  
    size_t row = (y + (uint32_t) Dim * 100937  
                 + currentState.sampleIdx * 1091  
                 + currentState.depth * 133337) % blue_noise_texture.size();  
    size_t column = (x + (uint32_t) Dim * 99523  
                   + currentState.sampleIdx * 2399  
                   + currentState.depth * 2549) % blue_noise_texture[0].size();  
    return blue_noise_texture[row][column];  
}
```

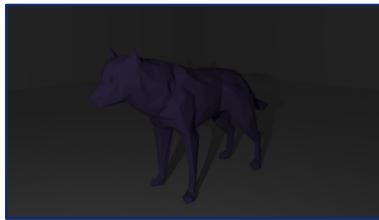
Реализация Blue Noise Sampler



Сравнение алгоритмов. Результаты

Визуальное сравнение

Оригинал



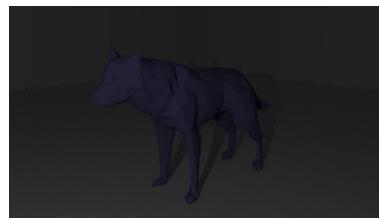
Standard Sampler



Halton Sampler



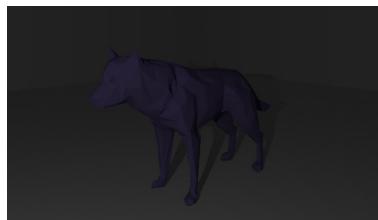
Halton RandomDigit
Sampler



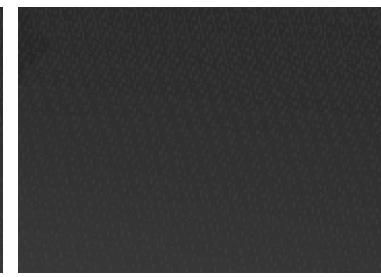
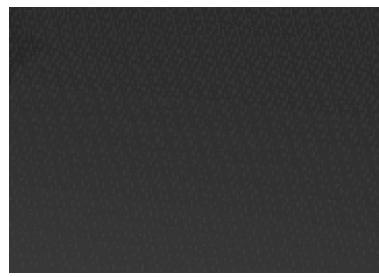
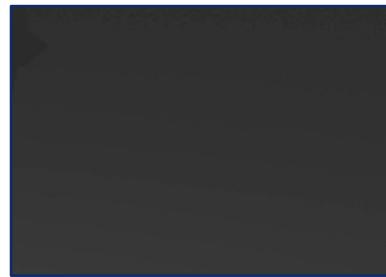
Halton Owen
Sampler



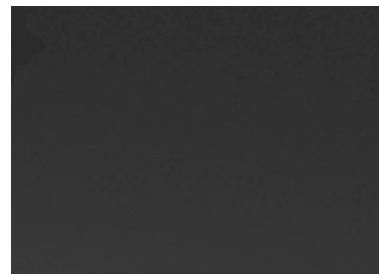
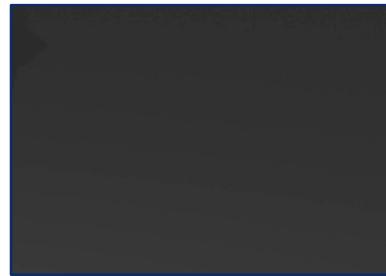
Blue Noise
Sampler



SPP = 1



SPP = 8





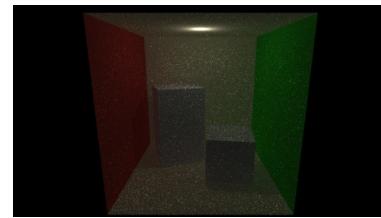
Сравнение алгоритмов. Результаты

Визуальное сравнение

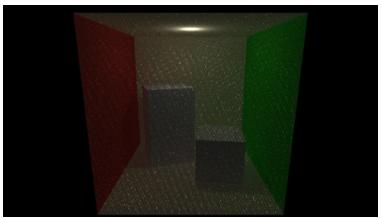
Оригинал



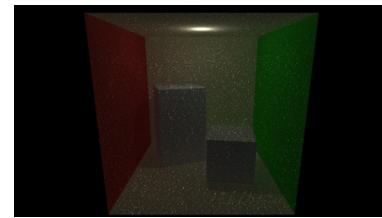
Standard Sampler



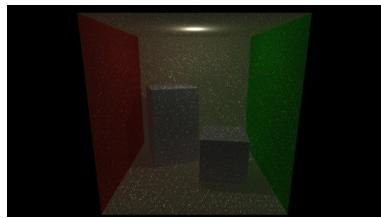
Halton Sampler



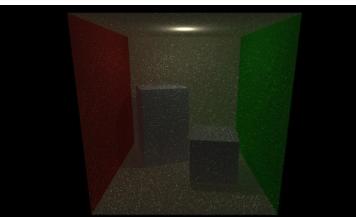
Halton RandomDigit
Sampler



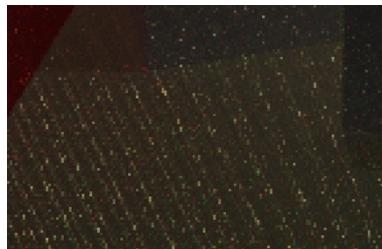
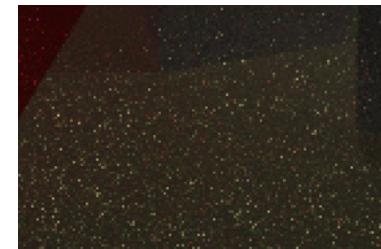
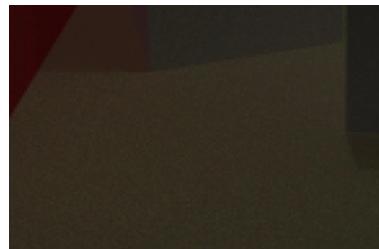
Halton Owen
Sampler



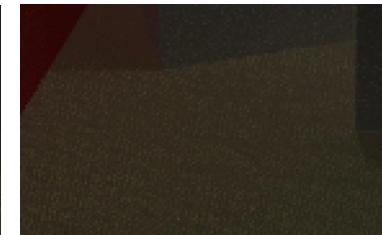
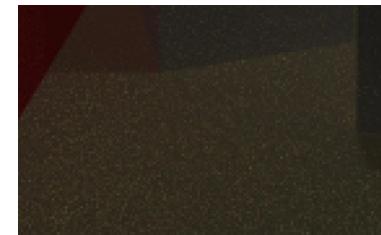
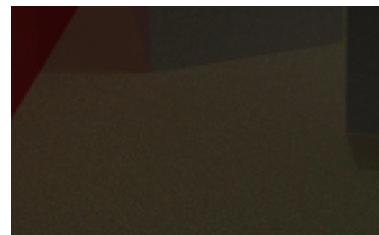
Blue Noise
Sampler



SPP = 1



SPP = 8





Сравнение алгоритмов. Результаты

Метрика MSE

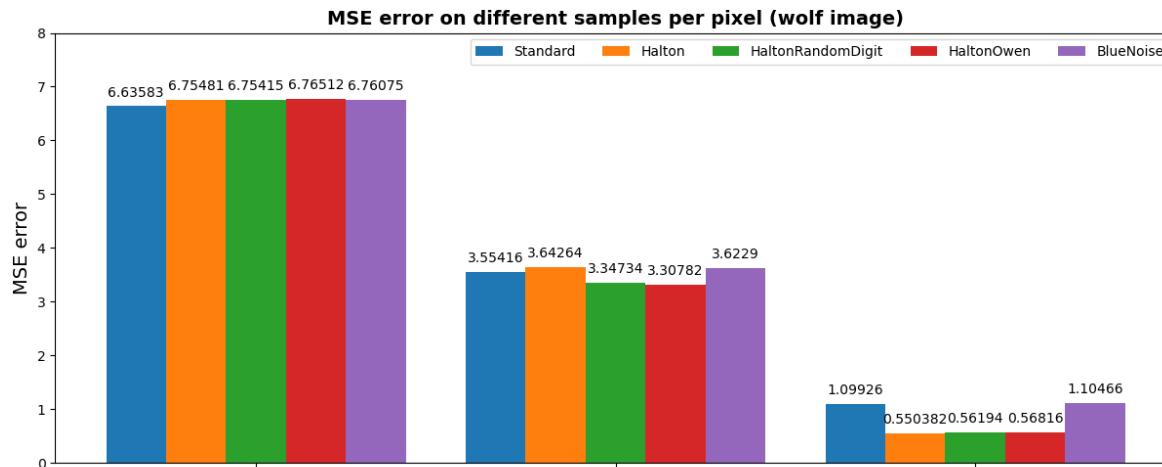


График сравнения метрики MSE на сцене «Волк»

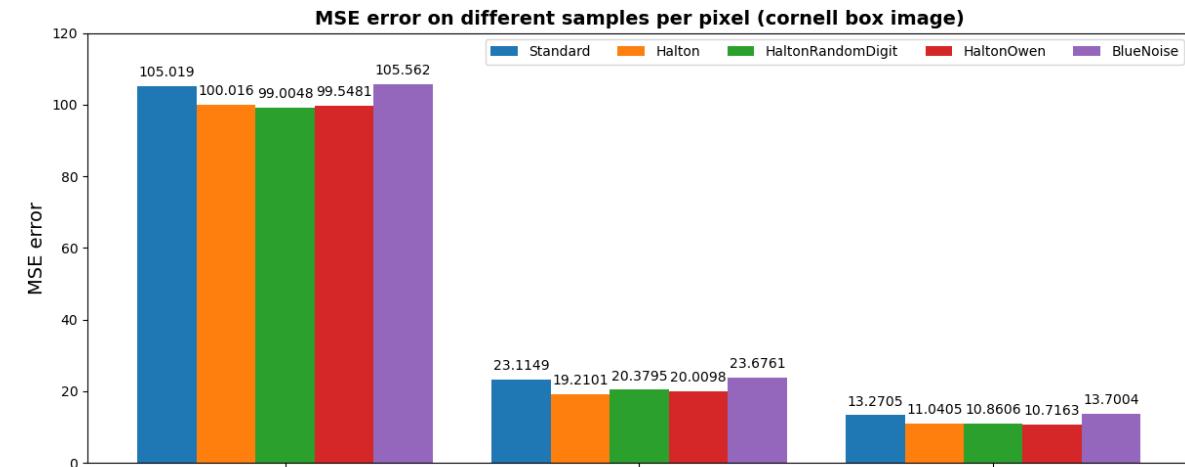


График сравнения метрики MSE на сцене «Cornell Box»



Сравнение алгоритмов. Результаты

Метрика MSE

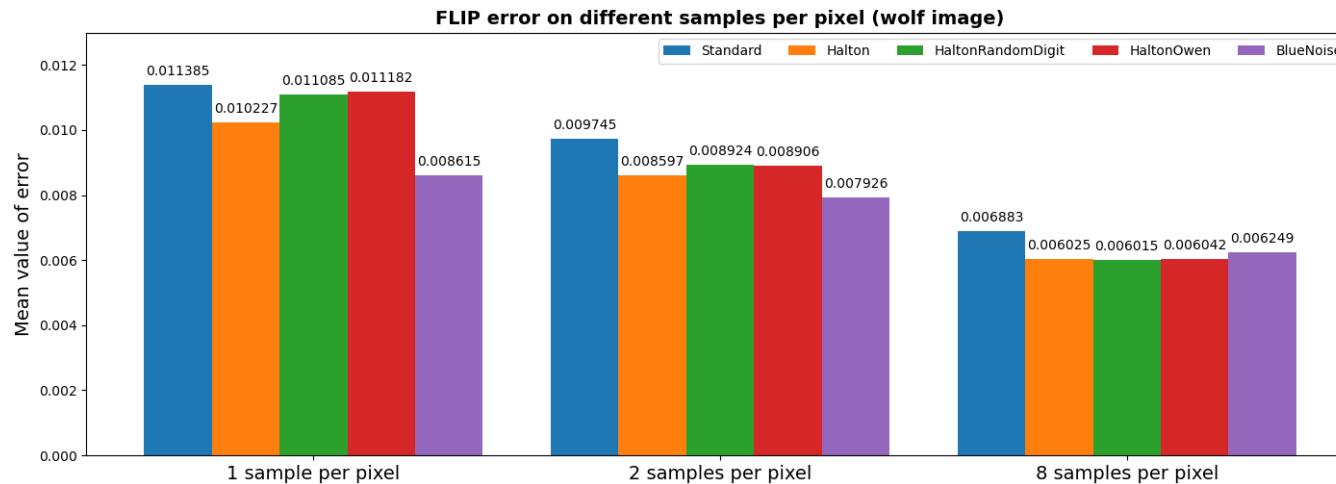


График сравнения метрики FLIP
на сцене «Волк»

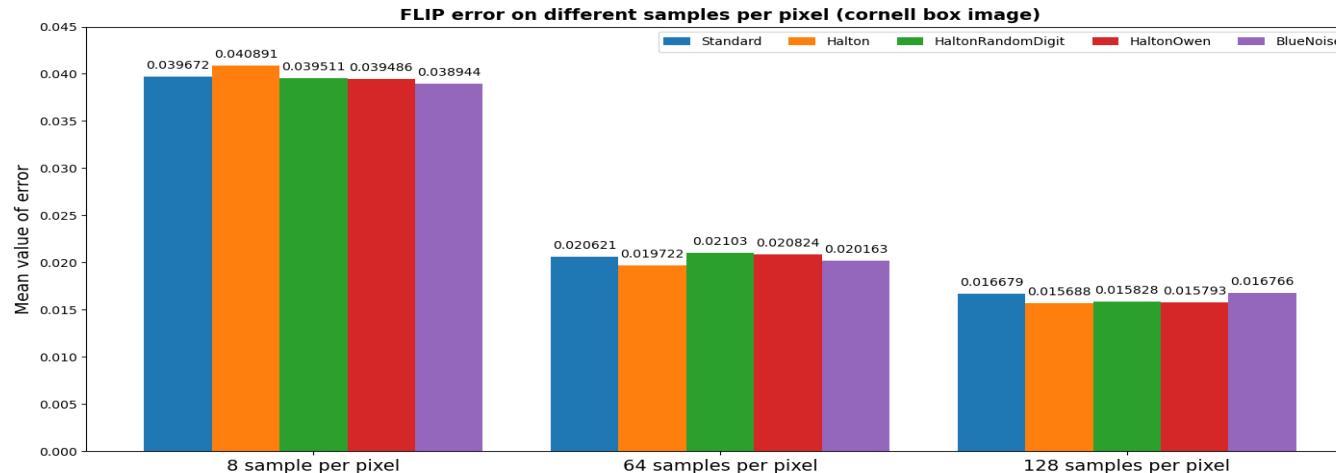


График сравнения метрики FLIP
на сцене «Cornell Box»



Заключение

Результаты курсовой работы

Были выбраны и реализованы на языке C++ 4 метода генерации случайных чисел: Halton Sampler, Halton RandomDigit Sampler, Halton Owen Sampler, Blue Noise Sampler. Они были добавлены в программу рендеринга изображений и с их помощью были срендерены изображения, необходимые для сравнения

Были выделены алгоритмы, которые показывают наилучшие результаты за наименьшее время: Blue Noise Sampler, Halton RandomDigit Sampler

Была написана собственная программа для рендеринга изображений на C++ с нуля, с возможностью дополнять ее новыми функциями, а также продолжать работать над ней, как над серьезным проектом

Подробное изучение существующих подходов к рендерингу изображений, как они устроены изнутри, а также методы их оптимизации и дальнейшего развития

Было проведено исследование и поиск алгоритмов для генерации случайных чисел, которые могли бы поспособствовать уменьшению времени рендеринга изображений в хорошем качестве

Было проведено сравнение и анализ написанных методов генерации случайных чисел с использованием современных метрик, использующихся в настоящих проектах по 3D графике



Факультет информатики, математики
и компьютерных наук

Сравнение различных стратегий выборки
в рендеринге трассировкой путей

Канделов Дамир Русланович
Группа: 22ПИ-1

16

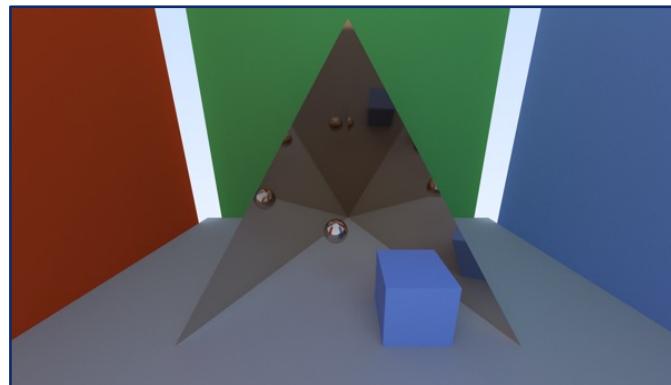
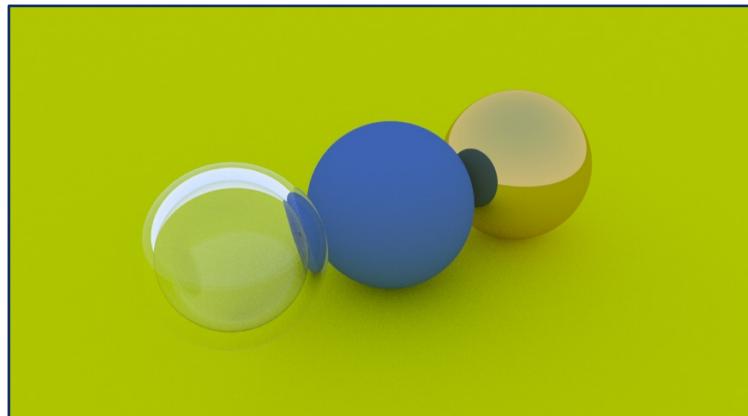
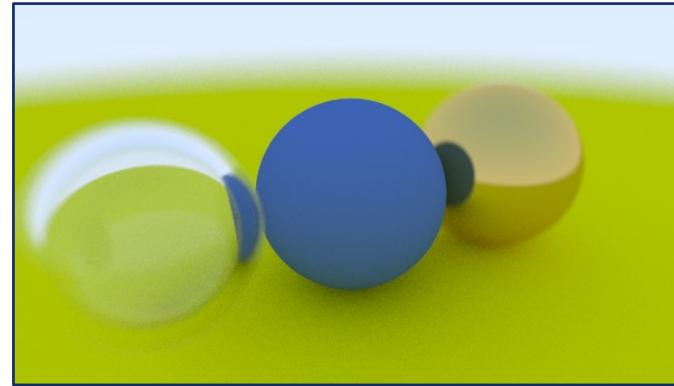
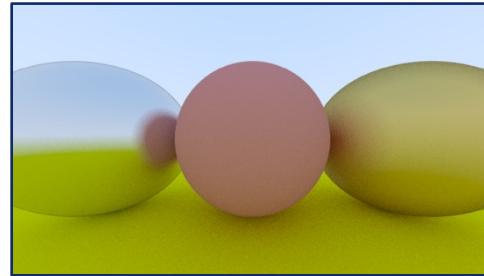
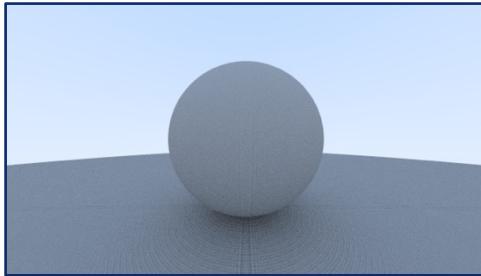
Спасибо за внимание!





Приложение 1

Собственная программа для рендеринга изображения





Приложение 2

Собственная программа для рендеринга изображения

