

Фотореалистичный рендеринг *(aka raytracing)*

Лекция 5: Оптимизация рейтрейсинга,
распараллеливание, BVH

2024

Треугольники

- В современности, основной примитив для описания трёхмерных объектов – *треугольник*

Треугольники

- В современности, основной примитив для описания трёхмерных объектов – *треугольник*
- Почему?

Треугольники

- В современности, основной примитив для описания трёхмерных объектов – *треугольник*
- Почему?
 - Простота описания

Треугольники

- В современности, основной примитив для описания трёхмерных объектов – *треугольник*
- Почему?
 - Простота описания
 - Простота рисования (важнее при растеризации)

Треугольники

- В современности, основной примитив для описания трёхмерных объектов – *треугольник*
- Почему?
 - Простота описания
 - Простота рисования (важнее при растеризации)
 - Простота интерполяции значений вершин

Треугольники

- В современности, основной примитив для описания трёхмерных объектов – *треугольник*
- Почему?
 - Простота описания
 - Простота рисования (важнее при растеризации)
 - Простота интерполяции значений вершин
 - Возможность аппроксимировать любые фигуры

Треугольники

- Как поддержать треугольник в нашем рейтрейсере?

Треугольники

- Как поддержать треугольник в нашем рейтреисере?
- Нам нужны три операции:

Треугольники

- Как поддержать треугольник в нашем рейтреисере?
- Нам нужны три операции:
 - Вычислить пересечение луча и треугольника

Треугольники

- Как поддержать треугольник в нашем рейтреисере?
- Нам нужны три операции:
 - Вычислить пересечение луча и треугольника
 - Вычислить нормаль к треугольнику

Треугольники

- Как поддержать треугольник в нашем рейтреисере?
- Нам нужны три операции:
 - Вычислить пересечение луча и треугольника
 - Вычислить нормаль к треугольнику
 - Сгенерировать случайную точку на треугольнике

Треугольники

- Точки треугольника с вершинами (A, B, C) удобно параметризовать двумя параметрами (u, v) как

$$A + u \cdot (B - A) + v \cdot (C - A)$$

Треугольники

- Точки треугольника с вершинами (A, B, C) удобно параметризовать двумя параметрами (u, v) как

$$A + u \cdot (B - A) + v \cdot (C - A)$$

- Точка принадлежит треугольнику, если $0 \leq u, 0 \leq v, u + v \leq 1$

Треугольники

- Точки треугольника с вершинами (A, B, C) удобно параметризовать двумя параметрами (u, v) как

$$A + u \cdot (B - A) + v \cdot (C - A)$$

- Точка принадлежит треугольнику, если $0 \leq u, 0 \leq v, u + v \leq 1$
- Произвольные значения (u, v) описывают точку на плоскости, в которой лежит треугольник

Треугольники

- Точки треугольника с вершинами (A, B, C) удобно параметризовать двумя параметрами (u, v) как

$$A + u \cdot (B - A) + v \cdot (C - A)$$

- Точка принадлежит треугольнику, если $0 \leq u, 0 \leq v, u + v \leq 1$
- Произвольные значения (u, v) описывают точку на плоскости, в которой лежит треугольник
- Тройка $(1 - u - v, u, v)$ – барицентрические координаты точки относительно вершин треугольника

Пересечение луча с треугольником

- Пересечение луча $O + t \cdot D$ сводится к решению уравнения

$$A + u \cdot (B - A) + v \cdot (C - A) = O + t \cdot D$$

Пересечение луча с треугольником

- Пересечение луча $O + t \cdot D$ сводится к решению уравнения

$$A + u \cdot (B - A) + v \cdot (C - A) = O + t \cdot D$$

- Немного его перепишем:

$$u \cdot (B - A) + v \cdot (C - A) - t \cdot D = O - A$$

Пересечение луча с треугольником

- Пересечение луча $O + t \cdot D$ сводится к решению уравнения

$$A + u \cdot (B - A) + v \cdot (C - A) = O + t \cdot D$$

- Немного его перепишем:

$$u \cdot (B - A) + v \cdot (C - A) - t \cdot D = O - A$$

- Это линейная система уравнений 3×3 с неизвестными (u, v, t) и матрицей со столбцами $(B - A, C - A, -D)$

Пересечение луча с треугольником

- Пересечение луча $O + t \cdot D$ сводится к решению уравнения

$$A + u \cdot (B - A) + v \cdot (C - A) = O + t \cdot D$$

- Немного его перепишем:

$$u \cdot (B - A) + v \cdot (C - A) - t \cdot D = O - A$$

- Это линейная система уравнений 3×3 с неизвестными (u, v, t) и матрицей со столбцами $(B - A, C - A, -D)$
- Решить её можно явным обращением матрицы

Пересечение луча с треугольником

- Мы получили значения (u, v, t) – они описывают точку пересечения луча с плоскостью, содержащей треугольник

Пересечение луча с треугольником

- Мы получили значения (u, v, t) – они описывают точку пересечения луча с плоскостью, содержащей треугольник
- Если не выполняется хотя бы одно из условий $0 \leq u, 0 \leq v, u + v \leq 1$, то точка не лежит в треугольнике \Rightarrow пересечения нет

Пересечение луча с треугольником

- Мы получили значения (u, v, t) – они описывают точку пересечения луча с плоскостью, содержащей треугольник
- Если не выполняется хотя бы одно из условий $0 \leq u, 0 \leq v, u + v \leq 1$, то точка не лежит в треугольнике \Rightarrow пересечения нет
- Если $t < 0$, то точка лежит сзади начала луча \Rightarrow пересечения нет

Пересечение луча с треугольником

- Мы получили значения (u, v, t) – они описывают точку пересечения луча с плоскостью, содержащей треугольник
- Если не выполняется хотя бы одно из условий $0 \leq u, 0 \leq v, u + v \leq 1$, то точка не лежит в треугольнике \Rightarrow пересечения нет
- Если $t < 0$, то точка лежит сзади начала луча \Rightarrow пересечения нет
- В противном случае пересечение есть

Пересечение луча с треугольником

- Нормаль к поверхности треугольника (A, B, C) – нормированный вектор $N = (B - A) \times (C - A)$

Пересечение луча с треугольником

- Нормаль к поверхности треугольника (A, B, C) – нормированный вектор $N = (B - A) \times (C - A)$
- Треугольник не имеет внутренности, поэтому будем считать вектор N выходящим из внешней стороны треугольника, а вектор $(-N)$ – из внутренней стороны

Пересечение луча с треугольником

- Нормаль к поверхности треугольника (A, B, C) – нормированный вектор $N = (B - A) \times (C - A)$
- Треугольник не имеет внутренности, поэтому будем считать вектор N выходящим из внешней стороны треугольника, а вектор $(-N)$ – из внутренней стороны
- Как обычно, по знаку $N \cdot D$ понимаем, снаружи луч, или внутри, и переворачиваем нормаль, если луч внутри

Семплинг поверхности треугольника

- Так как преобразование $(u, v) \mapsto P$ – аффинное, т.е. имеет константный Якобиан, нам достаточно сгенерировать равномерно точку в треугольнике $0 \leq u, 0 \leq v, u + v \leq 1$ – соответствующая точка в треугольнике (A, B, C) автоматически будет равномерно распределена

Семплинг поверхности треугольника

- Так как преобразование $(u, v) \mapsto P$ – аффинное, т.е. имеет константный Якобиан, нам достаточно сгенерировать равномерно точку в треугольнике $0 \leq u, 0 \leq v, u + v \leq 1$ – соответствующая точка в треугольнике (A, B, C) автоматически будет равномерно распределена
- Генерируем равномерные $U, V \sim U(0, 1)$

Семплинг поверхности треугольника

- Так как преобразование $(u, v) \mapsto P$ – аффинное, т.е. имеет константный Якобиан, нам достаточно сгенерировать равномерно точку в треугольнике $0 \leq u, 0 \leq v, u + v \leq 1$ – соответствующая точка в треугольнике (A, B, C) автоматически будет равномерно распределена
- Генерируем равномерные $U, V \sim U(0, 1)$
- Если $U + V > 1$, отражаем их через центр квадрата $[0, 1] \times [0, 1]$ преобразованием $U \mapsto 1 - U, V \mapsto 1 - V$

Семплинг поверхности треугольника

- Так как преобразование $(u, v) \mapsto P$ – аффинное, т.е. имеет константный Якобиан, нам достаточно сгенерировать равномерно точку в треугольнике $0 \leq u, 0 \leq v, u + v \leq 1$ – соответствующая точка в треугольнике (A, B, C) автоматически будет равномерно распределена
- Генерируем равномерные $U, V \sim U(0, 1)$
- Если $U + V > 1$, отражаем их через центр квадрата $[0, 1] \times [0, 1]$ преобразованием $U \mapsto 1 - U, V \mapsto 1 - V$
- Плотность полученной точки на треугольнике равна $1/S$, где $S = \frac{1}{2} \|(B - A) \times (C - A)\|$ – площадь треугольника

Об оптимизации рендеринга

- Мы видели, что Монте-Карло интегрирование занимает очень много времени, если мы хотим получить качественную картинку

Об оптимизации рендеринга

- Мы видели, что Монте-Карло интегрирование занимает очень много времени, если мы хотим получить качественную картинку
- Хочется научиться оптимизировать процесс

Об оптимизации рендеринга

- Мы видели, что Монте-Карло интегрирование занимает очень много времени, если мы хотим получить качественную картинку
- Хочется научиться оптимизировать процесс
- Оптимизировать = получить картинку более высокого качества за то же время

Об оптимизации рендеринга

- Мы видели, что Монте-Карло интегрирование занимает очень много времени, если мы хотим получить качественную картинку
- Хочется научиться оптимизировать процесс
- Оптимизировать = получить картинку более высокого качества за то же время
- Два варианта:

Об оптимизации рендеринга

- Мы видели, что Монте-Карло интегрирование занимает очень много времени, если мы хотим получить качественную картинку
- Хочется научиться оптимизировать процесс
- Оптимизировать = получить картинку более высокого качества за то же время
- Два варианта:
 - Увеличивать качество каждого луча (прошлая лекция)

Об оптимизации рендеринга

- Мы видели, что Монте-Карло интегрирование занимает очень много времени, если мы хотим получить качественную картинку
- Хочется научиться оптимизировать процесс
- Оптимизировать = получить картинку более высокого качества за то же время
- Два варианта:
 - Увеличивать качество каждого луча (прошлая лекция)
 - Ускорять вычисление каждого луча (эта лекция)

Оптимизация – это сложно

На производительность CPU влияют:

Оптимизация – это сложно

На производительность CPU влияют:

- Общая загруженность системы

Оптимизация – это сложно

На производительность CPU влияют:

- Общая загруженность системы
- Количество и паттерн доступов к памяти (cache-friendliness)

Оптимизация – это сложно

На производительность CPU влияют:

- Общая загруженность системы
- Количество и паттерн доступов к памяти (cache-friendliness)
- Как функции программы лежат в памяти (опять кэш)

Оптимизация – это сложно

На производительность CPU влияют:

- Общая загруженность системы
- Количество и паттерн доступов к памяти (cache-friendliness)
- Как функции программы лежат в памяти (опять кэш)
- Branch prediction

Оптимизация – это сложно

На производительность CPU влияют:

- Общая загруженность системы
- Количество и паттерн доступов к памяти (cache-friendliness)
- Как функции программы лежат в памяти (опять кэш)
- Branch prediction
- Оптимизации компилятора

Оптимизация – это сложно

На производительность CPU влияют:

- Общая загруженность системы
- Количество и паттерн доступов к памяти (cache-friendliness)
- Как функции программы лежат в памяти (опять кэш)
- Branch prediction
- Оптимизации компилятора
- Throttling

Оптимизация – это сложно

На производительность CPU влияют:

- Общая загруженность системы
- Количество и паттерн доступов к памяти (cache-friendliness)
- Как функции программы лежат в памяти (опять кэш)
- Branch prediction
- Оптимизации компилятора
- Throttling
- Многое другое

Хорошие новости

- Рейтинг – очень предсказуемый алгоритм: мы буквально огромное количество раз выполняем одну и ту же операцию

Хорошие новости

- Рейтинг – очень предсказуемый алгоритм: мы буквально огромное количество раз выполняем одну и ту же операцию
- Как оптимизировать:

Хорошие новости

- Рейтинг – очень предсказуемый алгоритм: мы буквально огромное количество раз выполняем одну и ту же операцию
- Как оптимизировать:
 - Точечно оптимизировать конкретные участки кода

Хорошие новости

- Рейтрейсинг – очень предсказуемый алгоритм: мы буквально огромное количество раз выполняем одну и ту же операцию
- Как оптимизировать:
 - Точечно оптимизировать конкретные участки кода
 - Ускорять алгоритмически

Хорошие новости

- Рейтинг – очень предсказуемый алгоритм: мы буквально огромное количество раз выполняем одну и ту же операцию
- Как оптимизировать:
 - Точечно оптимизировать конкретные участки кода
 - Ускорять алгоритмически
 - Распараллеливать

Точечные оптимизации

- Более эффективная реализация конкретных операций

Точечные оптимизации

- Более эффективная реализация конкретных операций
 - Быстрое вращение вектора кватернионом
blog.molecular-matters.com/2013/05/24/a-faster-quaternion-vector-multiplication

Точечные оптимизации

- Более эффективная реализация конкретных операций
 - Быстрое вращение вектора кватернионом
blog.molecular-matters.com/2013/05/24/a-faster-quaternion-vector-multiplication
- Кеширование значений

Точечные оптимизации

- Более эффективная реализация конкретных операций
 - Быстрое вращение вектора кватернионом
blog.molecular-matters.com/2013/05/24/a-faster-quaternion-vector-multiplication
- Кеширование значений
 - Нормаль и площадь треугольника

Точечные оптимизации

- Более эффективная реализация конкретных операций
 - Быстрое вращение вектора кватернионом
blog.molecular-matters.com/2013/05/24/a-faster-quaternion-vector-multiplication
- Кеширование значений
 - Нормаль и площадь треугольника
 - Вектор $\left(\frac{1}{D_X}, \frac{1}{D_Y}, \frac{1}{D_Z}\right)$, использующийся при пересечения луча и параллелепипеда

Точечные оптимизации

- Более эффективная реализация конкретных операций
 - Быстрое вращение вектора кватернионом
blog.molecular-matters.com/2013/05/24/a-faster-quaternion-vector-multiplication
- Кеширование значений
 - Нормаль и площадь треугольника
 - Вектор $\left(\frac{1}{D_X}, \frac{1}{D_Y}, \frac{1}{D_Z}\right)$, использующийся при пересечения луча и параллелепипеда
- Избегание бесполезных операций

Точечные оптимизации

- Более эффективная реализация конкретных операций
 - Быстрое вращение вектора кватернионом
blog.molecular-matters.com/2013/05/24/a-faster-quaternion-vector-multiplication
- Кеширование значений
 - Нормаль и площадь треугольника
 - Вектор $\left(\frac{1}{D_X}, \frac{1}{D_Y}, \frac{1}{D_Z}\right)$, использующийся при пересечения луча и параллелепипеда
- Избегание бесполезных операций
 - Не посылаем рекурсивно луч, если его вклад потом домножится на 0 (например, при importance sampling'е источника света с внутренней стороны поверхности объекта)

Точечные оптимизации

- Компактное и непрерывное расположение данных в памяти

Точечные оптимизации

- Компактное и непрерывное расположение данных в памяти
 - Меньше указателей и pointer chasing'a

Точечные оптимизации

- Компактное и непрерывное расположение данных в памяти
 - Меньше указателей и pointer chasing'a
 - Больше простых структур в непрерывных массивах

Точечные оптимизации

- Компактное и непрерывное расположение данных в памяти
 - Меньше указателей и pointer chasing'a
 - Больше простых структур в непрерывных массивах
- Branch prediction

Точечные оптимизации

- Компактное и непрерывное расположение данных в памяти
 - Меньше указателей и pointer chasing'a
 - Больше простых структур в непрерывных массивах
- Branch prediction
 - Меньше виртуальных функций и иерархий классов

Точечные оптимизации

- Компактное и непрерывное расположение данных в памяти
 - Меньше указателей и pointer chasing'a
 - Больше простых структур в непрерывных массивах
- Branch prediction
 - Меньше виртуальных функций и иерархий классов
 - Замена вычислений branchless эквивалентами

Точечные оптимизации

- Избегание лишних аллокаций памяти

Точечные оптимизации

- Избегание лишних аллокаций памяти
 - В основном коде рейтрейсера (после того, как сцена прочитана) их в идеале не должно быть вообще

Точечные оптимизации

- Избегание лишних аллокаций памяти
 - В основном коде рейтрейсера (после того, как сцена прочитана) их в идеале не должно быть вообще
- SSE-реализации математических операций

Точечные оптимизации

- Избегание лишних аллокаций памяти
 - В основном коде рейтрейсера (после того, как сцена прочитана) их в идеале не должно быть вообще
- SSE-реализации математических операций
- См. jacco.ompf2.com/2022/04/18/how-to-build-a-bvh-part-2-faster-rays

Большие сцены

- Точечные оптимизации могут дать вам весомое ускорение, вплоть до нескольких десятков процентов

Большие сцены

- Точечные оптимизации могут дать вам весомое ускорение, вплоть до нескольких десятков процентов
- Тем не менее, они никак не помогут нам с рендерингом больших сцен

Большие сцены

- Точечные оптимизации могут дать вам весомое ускорение, вплоть до нескольких десятков процентов
- Тем не менее, они никак не помогут нам с рендерингом больших сцен
- В основе вычисления цвета луча – алгоритм со сложностью $N \cdot D$, где N – количество объектов в сцене, а D – максимальная глубина

Большие сцены

- Точечные оптимизации могут дать вам весомое ускорение, вплоть до нескольких десятков процентов
- Тем не менее, они никак не помогут нам с рендерингом больших сцен
- В основе вычисления цвета луча – алгоритм со сложностью $N \cdot D$, где N – количество объектов в сцене, а D – максимальная глубина
- Зависимость от N линейная – это слишком медленно, если мы хотим рисовать сцены с миллионами треугольников!

Деревья

- В вычислительной геометрии есть множество структур данных, позволяющих ускорить запросы к геометрическим данным, использующие их локальность

Деревья

- В вычислительной геометрии есть множество структур данных, позволяющих ускорить запросы к геометрическим данным, использующие их локальность
- Почти всегда это какие-нибудь деревья: R/R+/R*-деревья, квадро/октодеревья, BSP-деревья, ...

Деревья

- В вычислительной геометрии есть множество структур данных, позволяющих ускорить запросы к геометрическим данным, использующие их локальность
- Почти всегда это какие-нибудь деревья: R/R+/R*-деревья, квадро/октодеревья, BSP-деревья, ...
- Общий термин для подобных древовидных пространственных структур данных – *bounding volume hierarchy (BVH)*

Деревья

- В вычислительной геометрии есть множество структур данных, позволяющих ускорить запросы к геометрическим данным, использующие их локальность
- Почти всегда это какие-нибудь деревья: R/R+/R*-деревья, квадро/октодеревья, BSP-деревья, ...
- Общий термин для подобных древовидных пространственных структур данных – *bounding volume hierarchy (BVH)*
- В рейтрейсинге под BVH обычно понимают разновидность R-деревьев

BVH: описание

- Каждая вершина дерева описывает некоторую область пространства, ограниченную прямоугольным параллелепипедом, параллельным осям координат (*axis-aligned bounding box, AABB*)

BVH: описание

- Каждая вершина дерева описывает некоторую область пространства, ограниченную прямоугольным параллелепипедом, параллельным осям координат (*axis-aligned bounding box, AABB*)
- Такой параллелепипед можно описать двумя точками – с минимальными и максимальными координатами по X, Y и Z соответственно

BVH: описание

- Каждая вершина дерева описывает некоторую область пространства, ограниченную прямоугольным параллелепипедом, параллельным осям координат (*axis-aligned bounding box, AABB*)
- Такой параллелепипед можно описать двумя точками – с минимальными и максимальными координатами по X, Y и Z соответственно
- У вершины могут быть дочерние вершины, AABB которых полностью содержатся в AABB самой вершины

BVH: описание

- Каждая вершина дерева описывает некоторую область пространства, ограниченную прямоугольным параллелепипедом, параллельным осям координат (*axis-aligned bounding box, AABB*)
- Такой параллелепипед можно описать двумя точками – с минимальными и максимальными координатами по X, Y и Z соответственно
- У вершины могут быть дочерние вершины, AABB которых полностью содержатся в AABB самой вершины
- У вершины может быть список объектов сцены, полностью содержащихся в AABB самой вершины

BVH: описание

- Каждая вершина дерева описывает некоторую область пространства, ограниченную прямоугольным параллелепипедом, параллельным осям координат (*axis-aligned bounding box, AABB*)
- Такой параллелепипед можно описать двумя точками – с минимальными и максимальными координатами по X, Y и Z соответственно
- У вершины могут быть дочерние вершины, AABB которых полностью содержатся в AABB самой вершины
- У вершины может быть список объектов сцены, полностью содержащихся в AABB самой вершины
- Все объекты сцены, кроме бесконечных (плоскостей), должны лежать в какой-то одной вершине дерева

BVH: описание

- Каждая вершина дерева описывает некоторую область пространства, ограниченную прямоугольным параллелепипедом, параллельным осям координат (*axis-aligned bounding box, AABB*)
- Такой параллелепипед можно описать двумя точками – с минимальными и максимальными координатами по X, Y и Z соответственно
- У вершины могут быть дочерние вершины, AABB которых полностью содержатся в AABB самой вершины
- У вершины может быть список объектов сцены, полностью содержащихся в AABB самой вершины
- Все объекты сцены, кроме бесконечных (плоскостей), должны лежать в какой-то одной вершине дерева
- **N.B.:** AABB двух дочерних вершин одной вершины могут пересекаться!

О хранении структур данных

- Кажется естественным хранить дерево в виде указателя на корневую вершину, и в вершине хранить детей тоже в виде указателей

О хранении структур данных

- Кажется естественным хранить дерево в виде указателя на корневую вершину, и в вершине хранить детей тоже в виде указателей
- Это **плохой способ** по ряду причин:

О хранении структур данных

- Кажется естественным хранить дерево в виде указателя на корневую вершину, и в вершине хранить детей тоже в виде указателей
- Это **плохой способ** по ряду причин:
 - Создание дерева использует много маленьких аллокаций памяти

О хранении структур данных

- Кажется естественным хранить дерево в виде указателя на корневую вершину, и в вершине хранить детей тоже в виде указателей
- Это **плохой способ** по ряду причин:
 - Создание дерева использует много маленьких аллокаций памяти
 - Вершины дерева будут лежать в памяти как попало, мало шансов иметь их близко в памяти

О хранении структур данных

- Кажется естественным хранить дерево в виде указателя на корневую вершину, и в вершине хранить детей тоже в виде указателей
- Это **плохой способ** по ряду причин:
 - Создание дерева использует много маленьких аллокаций памяти
 - Вершины дерева будут лежать в памяти как попало, мало шансов иметь их близко в памяти
 - Сложно поддержать циклические связи между вершинами

О хранении структур данных

- Кажется естественным хранить дерево в виде указателя на корневую вершину, и в вершине хранить детей тоже в виде указателей
- Это **плохой способ** по ряду причин:
 - Создание дерева использует много маленьких аллокаций памяти
 - Вершины дерева будут лежать в памяти как попало, мало шансов иметь их близко в памяти
 - Сложно поддержать циклические связи между вершинами
 - Удаление дерева – медленная рекурсивная операция

О хранении структур данных

- Кажется естественным хранить дерево в виде указателя на корневую вершину, и в вершине хранить детей тоже в виде указателей
- Это **плохой способ** по ряду причин:
 - Создание дерева использует много маленьких аллокаций памяти
 - Вершины дерева будут лежать в памяти как попало, мало шансов иметь их близко в памяти
 - Сложно поддержать циклические связи между вершинами
 - Удаление дерева – медленная рекурсивная операция
 - Копирование дерева (deep copy) – медленная рекурсивная операция

О хранении структур данных

- Кажется естественным хранить дерево в виде указателя на корневую вершину, и в вершине хранить детей тоже в виде указателей
- Это **плохой способ** по ряду причин:
 - Создание дерева использует много маленьких аллокаций памяти
 - Вершины дерева будут лежать в памяти как попало, мало шансов иметь их близко в памяти
 - Сложно поддержать циклические связи между вершинами
 - Удаление дерева – медленная рекурсивная операция
 - Копирование дерева (deep copy) – медленная рекурсивная операция
 - Сериализация дерева – медленная рекурсивная операция

О хранении структур данных

- Есть способ лучше:

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)
 - Использовать индексы этого массива как ссылки между вершинами

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)
 - Использовать индексы этого массива как ссылки между вершинами
- При таком способе:

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)
 - Использовать индексы этого массива как ссылки между вершинами
- При таком способе:
 - Создание дерева использует минимум аллокаций памяти

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)
 - Использовать индексы этого массива как ссылки между вершинами
- При таком способе:
 - Создание дерева использует минимум аллокаций памяти
 - Вершины дерева будут лежать в памяти непрерывно

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)
 - Использовать индексы этого массива как ссылки между вершинами
- При таком способе:
 - Создание дерева использует минимум аллокаций памяти
 - Вершины дерева будут лежать в памяти непрерывно
 - Циклические связи между вершинами тривиальны – просто храним индексы

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)
 - Использовать индексы этого массива как ссылки между вершинами
- При таком способе:
 - Создание дерева использует минимум аллокаций памяти
 - Вершины дерева будут лежать в памяти непрерывно
 - Циклические связи между вершинами тривиальны – просто храним индексы
 - Удаление дерева сводится к одной dealлокации массива вершин

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)
 - Использовать индексы этого массива как ссылки между вершинами
- При таком способе:
 - Создание дерева использует минимум аллокаций памяти
 - Вершины дерева будут лежать в памяти непрерывно
 - Циклические связи между вершинами тривиальны – просто храним индексы
 - Удаление дерева сводится к одной dealлокации массива вершин
 - Копирование дерева (deep copy) работает автоматически (скопированные индексы будут ссылаться на скопированный массив вершин)

О хранении структур данных

- Есть способ лучше:
 - Хранить все вершины дерева как структуры в непрерывном массиве (`std::vector`)
 - Использовать индексы этого массива как ссылки между вершинами
- При таком способе:
 - Создание дерева использует минимум аллокаций памяти
 - Вершины дерева будут лежать в памяти непрерывно
 - Циклические связи между вершинами тривиальны – просто храним индексы
 - Удаление дерева сводится к одной dealлокации массива вершин
 - Копирование дерева (deep copy) работает автоматически (скопированные индексы будут ссылаться на скопированный массив вершин)
 - Сериализация дерева сводится к побайтовой сериализации массива вершин

BVH: хранение объектов сцены

- Не нужно хранить объекты сцены прямо в вершинах, это сделает вершины дерева тяжёлыми и неэффективными

BVH: хранение объектов сцены

- Не нужно хранить объекты сцены прямо в вершинах, это сделает вершины дерева тяжёлыми и неэффективными
- При построении дерева можно отсортировать массив объектов сцены так, чтобы объекты, лежащие в конкретной вершине, хранились непрерывно в массиве всех объектов сцены

BVH: хранение объектов сцены

- Не нужно хранить объекты сцены прямо в вершинах, это сделает вершины дерева тяжёлыми и неэффективными
- При построении дерева можно отсортировать массив объектов сцены так, чтобы объекты, лежащие в конкретной вершине, хранились непрерывно в массиве всех объектов сцены
- Тогда в вершине дерева можно запомнить только индекс первого объекта сцены, лежащего в этой вершине, и количество таких объектов

BVH: описание в коде

```
struct Node
{
    vec3 aabb_min;
    vec3 aabb_max;
    uint32_t left_child;
    uint32_t right_child;
    uint32_t first_primitive_id;
    uint32_t primitive_count;
};

struct BVH
{
    vector<Node> nodes;
    uint32_t root;
};

struct Scene
{
    vector<Primitive> primitives;
    BVH bvh;
};
```

Data-oriented design

- *DOD (data-oriented design)* (иногда *DDD, data-driven design*) – методология, призванная структурировать код и алгоритмы вокруг потока данных вашей программы

Data-oriented design

- *DOD (data-oriented design)* (иногда *DDD, data-driven design*) – методология, призванная структурировать код и алгоритмы вокруг потока данных вашей программы
- Часто противопоставляется ООП по историческим причинам: 15 лет назад все всё писали на иерархиях классов, теперь всё пишут на непрерывных массивах

Data-oriented design

- DOD (*data-oriented design*) (иногда *DDD*, *data-driven design*) – методология, призванная структурировать код и алгоритмы вокруг потока данных вашей программы
- Часто противопоставляется ООП по историческим причинам: 15 лет назад все всё писали на иерархиях классов, теперь всё пишут на непрерывных массивах
- DOD – очень полезная концепция, но, как и с любым другим инструментом, нужно понимать, где и зачем её применять

BVH: обход

- Хотим найти пересечение луча со сценой

BVH: обход

- Хотим найти пересечение луча со сценой
- Рекурсивно спускаемся по дереву от корневой вершины

BVH: обход

- Хотим найти пересечение луча со сценой
- Рекурсивно спускаемся по дереву от корневой вершины
- Если луч не пересекает AABB вершины, игнорируем эту вершину

BVH: обход

- Хотим найти пересечение луча со сценой
- Рекурсивно спускаемся по дереву от корневой вершины
- Если луч не пересекает AABB вершины, игнорируем эту вершину
- В противном случае, пересекаем луч с объектами сцены в этой вершине (если они есть) и спускаемся в дочерние вершины (если они есть)

BVH: построение

- Можно выделить рекурсивный метод

```
BVH::build_node(vector<Primitive> & primitives,  
                uint32_t first, uint32_t count)
```

BVH: построение

- Можно выделить рекурсивный метод

```
BVH::build_node(vector<Primitive> & primitives,  
                uint32_t first, uint32_t count)
```

- Тогда построение дерева сводится к вызову

```
bvh.root = bvh.build_node(primitives, 0, primitives.size())
```

BVH: построение

- Можно выделить рекурсивный метод

```
BVH::build_node(vector<Primitive> & primitives,  
                uint32_t first, uint32_t count)
```

- Тогда построение дерева сводится к вызову

```
bvh.root = bvh.build_node(primitives, 0, primitives.size())
```

- **N.B.:** Вместо индексов может быть удобно использовать итераторы

BVH: построение

- При построении вершины дерева, надо определиться

BVH: построение

- При построении вершины дерева, надо определиться
 - 1. Хотим ли мы её разбивать на дочерние вершины

BVH: построение

- При построении вершины дерева, надо определиться
 - 1. Хотим ли мы её разбивать на дочерние вершины
 - 2. Если да, то как именно это сделать

BVH: построение

- При построении вершины дерева, надо определиться
 - 1. Хотим ли мы её разбивать на дочерние вершины
 - 2. Если да, то как именно это сделать
- Если `count` достаточно мал (например, не больше 4), то разбивать вершину дальше нет смысла – мы быстрее проверим пересечение с самими объектами сцены, чем разберёмся с AABB вершины

BVH: построение

- Если count больше порогового значения, надо как-то разбить объекты на две группы

BVH: построение

- Если count больше порогового значения, надо как-то разбить объекты на две группы
- Простой вариант – разбить вдоль середины самого длинного ребра AABB вершины:

BVH: построение

- Если count больше порогового значения, надо как-то разбить объекты на две группы
- Простой вариант – разбить вдоль середины самого длинного ребра AABB вершины:
 - Вычисляем, какое из рёбер (X, Y или Z) самое длинное

BVH: построение

- Если `count` больше порогового значения, надо как-то разбить объекты на две группы
- Простой вариант – разбить вдоль середины самого длинного ребра AABB вершины:
 - Вычисляем, какое из рёбер (`X`, `Y` или `Z`) самое длинное
 - Разбиваем объекты `[first, first + count)` на две группы (в C++ это `std::partition`)

BVH: построение

- Если `count` больше порогового значения, надо как-то разбить объекты на две группы
- Простой вариант – разбить вдоль середины самого длинного ребра AABB вершины:
 - Вычисляем, какое из рёбер (`X`, `Y` или `Z`) самое длинное
 - Разбиваем объекты `[first, first + count)` на две группы (в C++ это `std::partition`)
 - `[first, first + middle)` – те, кто левее центра ребра (вдоль выбранной оси)
 - `[first + middle, first + count)` – те, кто правее

BVH: построение

- Если `count` больше порогового значения, надо как-то разбить объекты на две группы
- Простой вариант – разбить вдоль середины самого длинного ребра AABB вершины:
 - Вычисляем, какое из рёбер (`X`, `Y` или `Z`) самое длинное
 - Разбиваем объекты `[first, first + count)` на две группы (в C++ это `std::partition`)
 - `[first, first + middle)` – те, кто левее центра ребра (вдоль выбранной оси)
 - `[first + middle, first + count)` – те, кто правее
 - Рекурсивно вызываем создание двух дочерних вершин на двух подмножествах объектов сцены

BVH: построение

- При рекурсивном разбиении нужно аккуратно обработать патологические случаи

BVH: построение

- При рекурсивном разбиении нужно аккуратно обработать патологические случаи
- Если в одну из дочерних вершин попадает 0 объектов сцены, разбиение нужно отменить, и положить все объекты в текущую вершину

BVH: построение

- При рекурсивном разбиении нужно аккуратно обработать патологические случаи
- Если в одну из дочерних вершин попадает 0 объектов сцены, разбиение нужно отменить, и положить все объекты в текущую вершину
- Иначе может произойти бесконечная рекурсия

Ускорение обхода: early-out

- Запоминаем расстояние до ближайшего уже найденного пересечения

Ускорение обхода: early-out

- Запоминаем расстояние до ближайшего уже найденного пересечения
- При посещении вершины, если луч пересекает её AABB, вычисляем расстояние до ближайшего пересечения AABB вершины

Ускорение обхода: early-out

- Запоминаем расстояние до ближайшего уже найденного пересечения
- При посещении вершины, если луч пересекает её AABB, вычисляем расстояние до ближайшего пересечения AABB вершины
- Если расстояние до AABB вершины больше, чем ближайшее уже найденное пересечение с каким-то объектом, то вершину можно пропустить

Ускорение обхода: ordered traversal

- При посещении дочерних вершин какой-либо вершины, мы можем выбрать оптимальный порядок их посещения

Ускорение обхода: ordered traversal

- При посещении дочерних вершин какой-либо вершины, мы можем выбрать оптимальный порядок их посещения
- Например, если вершина была разбита по оси X, и при этом X-компоненте вектора направления луча отрицательна, то имеет смысл сначала посетить правую вершину, и только потом левую

Ускорение обхода: ordered traversal

- При посещении дочерних вершин какой-либо вершины, мы можем выбрать оптимальный порядок их посещения
- Например, если вершина была разбита по оси X, и при этом X-компонента вектора направления луча отрицательна, то имеет смысл сначала посетить правую вершину, и только потом левую
- Для этого в вершинах нужно помнить, по какой оси было разбиение

BVH: улучшение качества дерева

- Уже разбиение по центру самого длинного ребра даст хорошую производительность на типичных сценах, но мы можем сделать лучше

BVH: улучшение качества дерева

- Уже разбиение по центру самого длинного ребра даст хорошую производительность на типичных сценах, но мы можем сделать лучше
- Пусть, мы придумали, как разбить конкретную вершину на две

BVH: улучшение качества дерева

- Уже разбиение по центру самого длинного ребра даст хорошую производительность на типичных сценах, но мы можем сделать лучше
- Пусть, мы придумали, как разбить конкретную вершину на две
- Как оценить качество такого разбиения?

BVH: улучшение качества дерева

- Пусть есть случайный луч, пересекающий родительскую вершину

BVH: улучшение качества дерева

- Пусть есть случайный луч, пересекающий родительскую вершину
- Пусть $p_{1,2}$ – вероятность, что такой луч попадёт в первую (вторую) дочернюю вершину

BVH: улучшение качества дерева

- Пусть есть случайный луч, пересекающий родительскую вершину
- Пусть $p_{1,2}$ – вероятность, что такой луч попадёт в первую (вторую) дочернюю вершину
- Пусть $N_{1,2}$ – количество объектов в первой (второй) дочерней вершиной

BVH: улучшение качества дерева

- Пусть есть случайный луч, пересекающий родительскую вершину
- Пусть $p_{1,2}$ – вероятность, что такой луч попадёт в первую (вторую) дочернюю вершину
- Пусть $N_{1,2}$ – количество объектов в первой (второй) дочерней вершиной
- Тогда в среднем нам придётся обработать $p_1N_1 + p_2N_2$ объектов

- Можно показать, что $p_i = \frac{S_i}{S}$, где S_i – площадь поверхности AABB дочерней вершины, а S – площадь поверхности AABB родительской вершины

- Можно показать, что $p_i = \frac{S_i}{S}$, где S_i – площадь поверхности AABB дочерней вершины, а S – площадь поверхности AABB родительской вершины
- Тогда мы можем использовать величину $S \cdot N$ как стоимость вершины

- Можно показать, что $p_i = \frac{S_i}{S}$, где S_i – площадь поверхности AABB дочерней вершины, а S – площадь поверхности AABB родительской вершины
- Тогда мы можем использовать величину $S \cdot N$ как стоимость вершины
- Разбиение вершины на дочерние имеет смысл, если стоимость вершины больше, чем сумма стоимости дочерних вершин

- Можно показать, что $p_i = \frac{S_i}{S}$, где S_i – площадь поверхности AABB дочерней вершины, а S – площадь поверхности AABB родительской вершины
- Тогда мы можем использовать величину $S \cdot N$ как стоимость вершины
- Разбиение вершины на дочерние имеет смысл, если стоимость вершины больше, чем сумма стоимости дочерних вершин
- Этот подход называется *surface area heuristic (SAH)*

- Рассмотрим идеальный случай: разбиение по центру самого длинного ребра даёт две непересекающихся дочерних вершины, не пересекающих плоскость разбиения

- Рассмотрим идеальный случай: разбиение по центру самого длинного ребра даёт две непересекающихся дочерних вершины, не пересекающих плоскость разбиения
- Тогда площадь поверхности AABB дочерних вершин $S_{1,2} \leq \frac{2}{3}S$

- Рассмотрим идеальный случай: разбиение по центру самого длинного ребра даёт две непересекающихся дочерних вершины, не пересекающих плоскость разбиения
- Тогда площадь поверхности AABB дочерних вершин $S_{1,2} \leq \frac{2}{3}S$
- И суммарная стоимость дочерних вершин:
$$S_1N_1 + S_2N_2 \leq \frac{2}{3}S\frac{1}{2}N_1 + \frac{2}{3}S\frac{1}{2}(N - N_1) = \frac{2}{3}SN$$

- Рассмотрим идеальный случай: разбиение по центру самого длинного ребра даёт две непересекающихся дочерних вершины, не пересекающих плоскость разбиения
- Тогда площадь поверхности AABB дочерних вершин $S_{1,2} \leq \frac{2}{3}S$
- И суммарная стоимость дочерних вершин:
$$S_1N_1 + S_2N_2 \leq \frac{2}{3}S\frac{1}{2}N_1 + \frac{2}{3}S\frac{1}{2}(N - N_1) = \frac{2}{3}SN$$
- \implies Стоимость разбиения меньше стоимости родительской вершины

- Мы научились измерять качество конкретного разбиения

- Мы научились измерять качество конкретного разбиения
- Как найти идеальное разбиение?

- Мы научились измерять качество конкретного разбиения
- Как найти идеальное разбиение? Перебором!

- Мы научились измерять качество конкретного разбиения
- Как найти идеальное разбиение? Перебором!
- Переберём все 3 возможных оси разбиения

- Мы научились измерять качество конкретного разбиения
- Как найти идеальное разбиение? Перебором!
- Переберём все 3 возможных оси разбиения
- Отсортируем объекты данной вершины в порядке возрастания i -ой координаты центра объекта (или его AABB), где i – выбранная ось разбиение

- Мы научились измерять качество конкретного разбиения
- Как найти идеальное разбиение? Перебором!
- Переберём все 3 возможных оси разбиения
- Отсортируем объекты данной вершины в порядке возрастания i -ой координаты центра объекта (или его AABB), где i – выбранная ось разбиение
- Для каждой оси у нас есть $N - 1$ вариантов разбиения (первые K объектов идут в левую вершину, оставшиеся $N - K$ – в правую)

- Мы научились измерять качество конкретного разбиения
- Как найти идеальное разбиение? Перебором!
- Переберём все 3 возможных оси разбиения
- Отсортируем объекты данной вершины в порядке возрастания i -ой координаты центра объекта (или его AABB), где i – выбранная ось разбиение
- Для каждой оси у нас есть $N - 1$ вариантов разбиения (первые K объектов идут в левую вершину, оставшиеся $N - K$ – в правую)
- Вычислим стоимость разбиения для всех этих вариантов и возьмём минимальный

- Мы научились измерять качество конкретного разбиения
- Как найти идеальное разбиение? Перебором!
- Переберём все 3 возможных оси разбиения
- Отсортируем объекты данной вершины в порядке возрастания i -ой координаты центра объекта (или его AABB), где i – выбранная ось разбиение
- Для каждой оси у нас есть $N - 1$ вариантов разбиения (первые K объектов идут в левую вершину, оставшиеся $N - K$ – в правую)
- Вычислим стоимость разбиения для всех этих вариантов и возьмём минимальный
- Если стоимость минимального разбиения всё ещё выше стоимости самой вершины, не производим разбиение

- Для вычисления стоимости всех $N - 1$ разбиений нам нужно знать площади поверхности соответствующих AABB

- Для вычисления стоимости всех $N - 1$ разбиений нам нужно знать площади поверхности соответствующих AABB
- AABB всех объектов с первого до К-ого можно вычислить за один проход для всех K, как префиксную сумму:

- Для вычисления стоимости всех $N - 1$ разбиений нам нужно знать площади поверхности соответствующих AABB
- AABB всех объектов с первого до К-ого можно вычислить за один проход для всех K, как префиксную сумму:
 - `aabb_left[0] = empty`

- Для вычисления стоимости всех $N - 1$ разбиений нам нужно знать площади поверхности соответствующих AABB
- AABB всех объектов с первого до К-ого можно вычислить за один проход для всех K, как префиксную сумму:
 - `aabb_left[0] = empty`
 - `aabb_left[i] = union(aabb_left[i-1], object[i-1].aabb)`

- Для вычисления стоимости всех $N - 1$ разбиений нам нужно знать площади поверхности соответствующих AABB
- AABB всех объектов с первого до К-ого можно вычислить за один проход для всех K, как префиксную сумму:
 - `aabb_left[0] = empty`
 - `aabb_left[i] = union(aabb_left[i-1], object[i-1].aabb)`
- Аналогично можно вычислить AABB всех объектов от K-ого до последнего за один проход в обратном порядке

BVH: binned SAH

- Итоговая стоимость разбиения вершины $O(n \cdot \log n)$ (сортировка) и построения дерева $O(n \cdot \log^2 n)$

BVH: binned SAH

- Итоговая стоимость разбиения вершины $O(n \cdot \log n)$ (сортировка) и построения дерева $O(n \cdot \log^2 n)$
- Построение BVH работает в целом достаточно быстро (порядка 100ms на 100k вершин), но хочется быстрее

BVH: binned SAH

- Итоговая стоимость разбиения вершины $O(n \cdot \log n)$ (сортировка) и построения дерева $O(n \cdot \log^2 n)$
- Построение BVH работает в целом достаточно быстро (порядка 100ms на 100k вершин), но хочется быстрее
- Binning-эвристика: вместо проверки всех возможных разбиений, проверим фиксированное количество (напр. 8 или 16) разбиений с фиксированным шагом

BVH: binned SAH

- Итоговая стоимость разбиения вершины $O(n \cdot \log n)$ (сортировка) и построения дерева $O(n \cdot \log^2 n)$
- Построение BVH работает в целом достаточно быстро (порядка 100ms на 100k вершин), но хочется быстрее
- Binning-эвристика: вместо проверки всех возможных разбиений, проверим фиксированное количество (напр. 8 или 16) разбиений с фиксированным шагом
- Большая серия статей про построение BVH для рейтрейсинга: jacco.ompf2.com/2022/04/13/how-to-build-a-bvh-part-1-basics

BVH: применение

- При вычислении пересечения луча со сценой, проходим отдельно по всем бесконечным объектам, а для конечных объектов обходим BVH

BVH: применение

- При вычислении пересечения луча со сценой, проходим отдельно по всем бесконечным объектам, а для конечных объектов обходим BVH
- При multiple importance sampling'е источников света, для вычисления правильной вероятности направления, нам нужно найти все источники света, пересекающие луч из текущей точки в этом направлении

BVH: применение

- При вычислении пересечения луча со сценой, проходим отдельно по всем бесконечным объектам, а для конечных объектов обходим BVH
- При multiple importance sampling'е источников света, для вычисления правильной вероятности направления, нам нужно найти все источники света, пересекающие луч из текущей точки в этом направлении
- Лучше для источников света завести отдельную, независимую BVH (и **не делать** early-out при её обходе, так как нам нужны **все** пересечения, а не ближайшее)

Распараллеливание

- Чтобы распараллелить рендеринг на несколько потоков, нужно сначала разбить его на независимые подзадачи

Распараллеливание

- Чтобы распараллелить рендеринг на несколько потоков, нужно сначала разбить его на независимые подзадачи
- Можно выбрать разную гранулярность подзадач:

Распараллеливание

- Чтобы распараллелить рендеринг на несколько потоков, нужно сначала разбить его на независимые подзадачи
- Можно выбрать разную гранулярность подзадач:
 - Подзадача – большой кусок изображения

Распараллеливание

- Чтобы распараллелить рендеринг на несколько потоков, нужно сначала разбить его на независимые подзадачи
- Можно выбрать разную гранулярность подзадач:
 - Подзадача – большой кусок изображения
 - Подзадача – одна строка изображения

Распараллеливание

- Чтобы распараллелить рендеринг на несколько потоков, нужно сначала разбить его на независимые подзадачи
- Можно выбрать разную гранулярность подзадач:
 - Подзадача – большой кусок изображения
 - Подзадача – одна строка изображения
 - Подзадача – маленький (8x8) тайл изображения

Распараллеливание

- Чтобы распараллелить рендеринг на несколько потоков, нужно сначала разбить его на независимые подзадачи
- Можно выбрать разную гранулярность подзадач:
 - Подзадача – большой кусок изображения
 - Подзадача – одна строка изображения
 - Подзадача – маленький (8×8) тайл изображения
 - Подзадача – один пиксель

Распараллеливание

- Чтобы распараллелить рендеринг на несколько потоков, нужно сначала разбить его на независимые подзадачи
- Можно выбрать разную гранулярность подзадач:
 - Подзадача – большой кусок изображения
 - Подзадача – одна строка изображения
 - Подзадача – маленький (8×8) тайл изображения
 - Подзадача – один пиксель
 - Подзадача – один семпл в пикселе

Распараллеливание

- Чтобы распараллелить рендеринг на несколько потоков, нужно сначала разбить его на независимые подзадачи
- Можно выбрать разную гранулярность подзадач:
 - Подзадача – большой кусок изображения
 - Подзадача – одна строка изображения
 - Подзадача – маленький (8×8) тайл изображения
 - Подзадача – один пиксель
 - Подзадача – один семпл в пикселе
 - Подзадача – один луч

Распараллеливание: гранулярность

- Подбирать гранулярность нужно из нескольких соображений

Распараллеливание: гранулярность

- Подбирать гранулярность нужно из нескольких соображений
- Нужно максимизировать равномерность нагрузки на разные подзадачи (чтобы не простаивали потоки, справившиеся раньше других)

Распараллеливание: гранулярность

- Подбирать гранулярность нужно из нескольких соображений
- Нужно максимизировать равномерность нагрузки на разные подзадачи (чтобы не простоявали потоки, справившиеся раньше других)
 - \Rightarrow Разбиение на большие куски или даже строки изображения нам не подходит: в одном куске может оказаться намного больше объектов, чем в другом

Распараллеливание: гранулярность

- Подбирать гранулярность нужно из нескольких соображений
- Нужно максимизировать равномерность нагрузки на разные подзадачи (чтобы не простаивали потоки, справившиеся раньше других)
 - \Rightarrow Разбиение на большие куски или даже строки изображения нам не подходит: в одном куске может оказаться намного больше объектов, чем в другом
- Нужно минимизировать необходимость синхронизации

Распараллеливание: гранулярность

- Подбирать гранулярность нужно из нескольких соображений
- Нужно максимизировать равномерность нагрузки на разные подзадачи (чтобы не простаивали потоки, справившиеся раньше других)
 - \Rightarrow Разбиение на большие куски или даже строки изображения нам не подходит: в одном куске может оказаться намного больше объектов, чем в другом
- Нужно минимизировать необходимость синхронизации
 - \Rightarrow Разбиение на отдельные лучи не подходит: лучу придётся ждать рекурсивно вызванный луч

Распараллеливание: гранулярность

- Подбирать гранулярность нужно из нескольких соображений
- Нужно максимизировать равномерность нагрузки на разные подзадачи (чтобы не простаивали потоки, справившиеся раньше других)
 - \Rightarrow Разбиение на большие куски или даже строки изображения нам не подходит: в одном куске может оказаться намного больше объектов, чем в другом
- Нужно минимизировать необходимость синхронизации
 - \Rightarrow Разбиение на отдельные лучи не подходит: лучу придётся ждать рекурсивно вызванный луч
- Нужно максимизировать локальность данных

Распараллеливание: гранулярность

- Подбирать гранулярность нужно из нескольких соображений
- Нужно максимизировать равномерность нагрузки на разные подзадачи (чтобы не простоявали потоки, справившиеся раньше других)
 - \Rightarrow Разбиение на большие куски или даже строки изображения нам не подходит: в одном куске может оказаться намного больше объектов, чем в другом
- Нужно минимизировать необходимость синхронизации
 - \Rightarrow Разбиение на отдельные лучи не подходит: лучу придётся ждать рекурсивно вызванный луч
- Нужно максимизировать локальность данных
 - \Rightarrow Разбиение на отдельные семплы не подходит: все семплы одного пикселя будут обращаться +/- к одним и тем же данным, лучше их сгруппировать в одну подзадачу

Распараллеливание: тайлы

- Стандартный выбор при рейтрейсинге – гранулярность на уровне пикселей или небольших (8x8) тайлов пикселей

Распараллеливание: тайлы

- Стандартный выбор при рейтрейсинге – гранулярность на уровне пикселей или небольших (8x8) тайлов пикселей
- Пиксели тайла объединяются в одну подзадачу из тех же соображений, что и семплы одного пикселя

Распараллеливание: тайлы

- Стандартный выбор при рейтрейсинге – гранулярность на уровне пикселей или небольших (8x8) тайлов пикселей
- Пиксели тайла объединяются в одну подзадачу из тех же соображений, что и семплы одного пикселя
- Выигрыш от тайлов становится заметен на очень больших сценах (десятки-сотни миллионов вершин), когда сцена уже не влезает ни в какие кеши CPU

Распараллеливание: реализация

- Нужно придумать уникальное описание конкретной подзадачи (напр. ID пикселя), и завести синхронизированную очередь всех подзадач

Распараллеливание: реализация

- Нужно придумать уникальное описание конкретной подзадачи (напр. ID пикселя), и завести синхронизированную очередь всех подзадач
- Нужно создать N потоков (где N – количество ядер процессора, напр. `std::thread::hardware_concurrency`), которые будут по одной доставать подзадачи из очереди и выполнять их

Распараллеливание: реализация

- Нужно придумать уникальное описание конкретной подзадачи (напр. ID пикселя), и завести синхронизированную очередь всех подзадач
- Нужно создать N потоков (где N – количество ядер процессора, напр. `std::thread::hardware_concurrency`), которые будут по одной доставать подзадачи из очереди и выполнять их
- В C++ это можно легко сделать с помощью OpenMP (нужно будет добавить его в CMakeLists) директивой
`#pragma omp parallel for`

Распараллеливание: pitfalls

- Нужно убедиться, что все подзадачи независимы, и не перезатирают данные друг друга (напр. разные потоки не пишут в один и тот же пиксель)

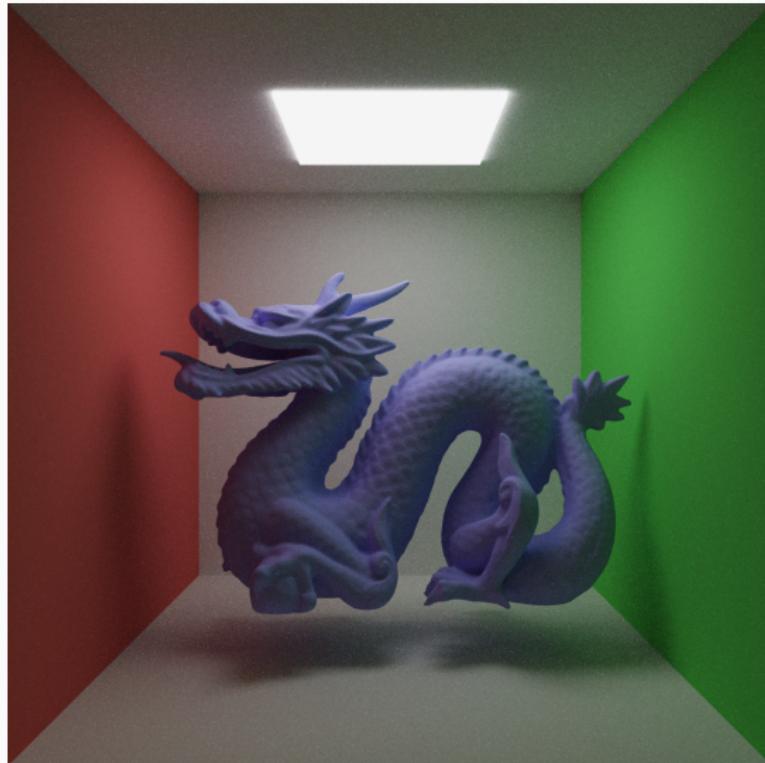
Распараллеливание: pitfalls

- Нужно убедиться, что все подзадачи независимы, и не перезатирают данные друг друга (напр. разные потоки не пишут в один и тот же пиксель)
- Нужно убедиться в потокобезопасности сцены и BVH (их непросто сделать потоконебезопасными, – например, неаккуратно что-нибудь кешируя)

Распараллеливание: pitfalls

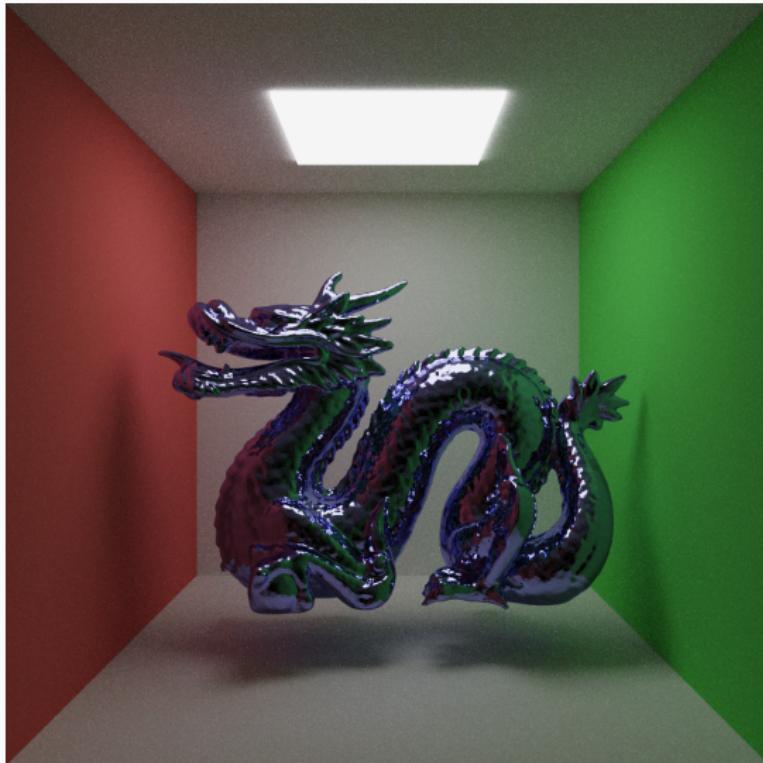
- Нужно убедиться, что все подзадачи независимы, и не перезатирают данные друг друга (напр. разные потоки не пишут в один и тот же пиксель)
- Нужно убедиться в потокобезопасности сцены и BVH (их непросто сделать потоконебезопасными, – например, неаккуратно что-нибудь кешируя)
- Нужно убедиться, что разные потоки используют разные и по-разному проинициализированные генераторы случайных чисел (можно их проинициализировать, используя ID потока или ID подзадачи)

Оптимизации: результат



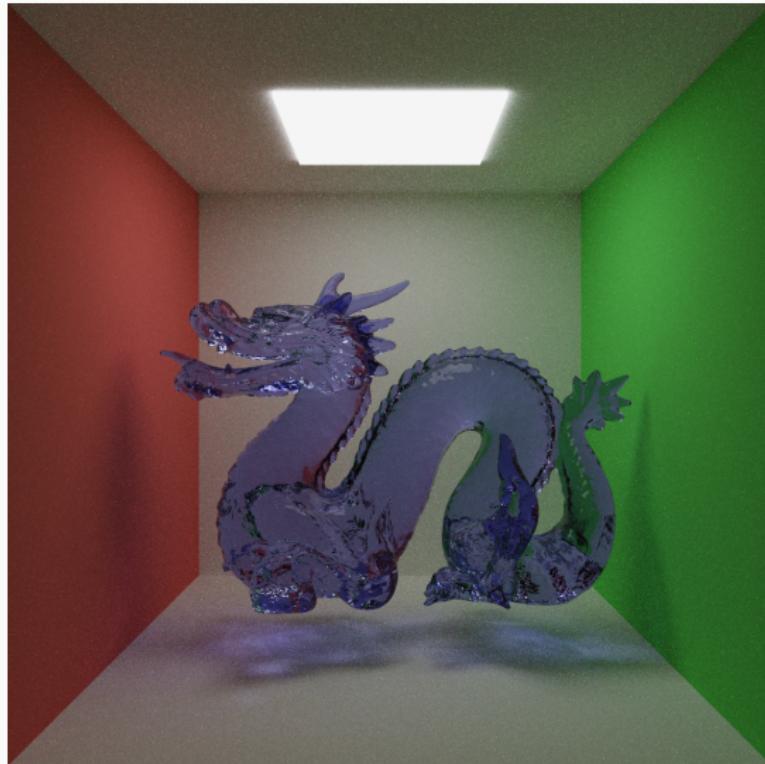
100k faces, 1024x1024, 512 spp, 180 s

Оптимизации: результат



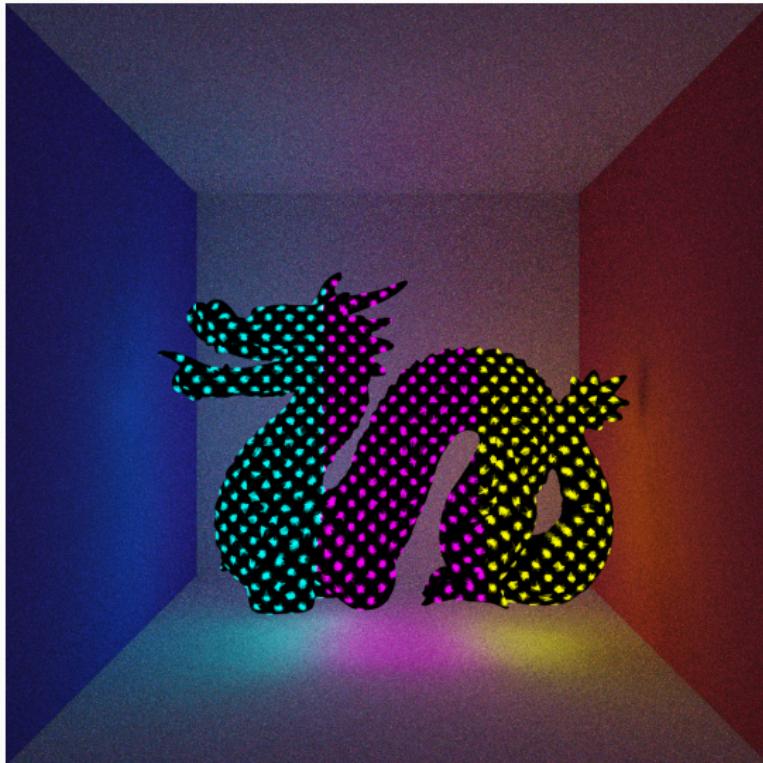
100k faces, 1024x1024, 512 spp, 160 s

Оптимизации: результат



100k faces, 1024x1024, 512 spp, 300 s

Оптимизации: результат



100k faces, 1024x1024, 512 spp, 1000 s