

# Эффективная работа с процедурными деревьями

1 апреля 2021 г.

# Процедурная генерация в играх

На данный момент существует множество разнообразных алгоритмов для процедурной генерации в играх или для визуализации. Также есть проекты, позволяющие создавать реалистичные модели деревьев с высокой детализацией. Например, SpeedTree:



# Преимущества

- Огромные открытые миры
- Генерация по мере игры
- Внешнее разнообразие
- Уникальный игровой опыт
- Меньше работы художников, дизайнеров уровней
- Зависимость от окружения

- у человека часто получается лучше
- не подходит для точно выверенного level-дизайна

основная проблема:

**Слишком много уникальных моделей**

# Что делать?

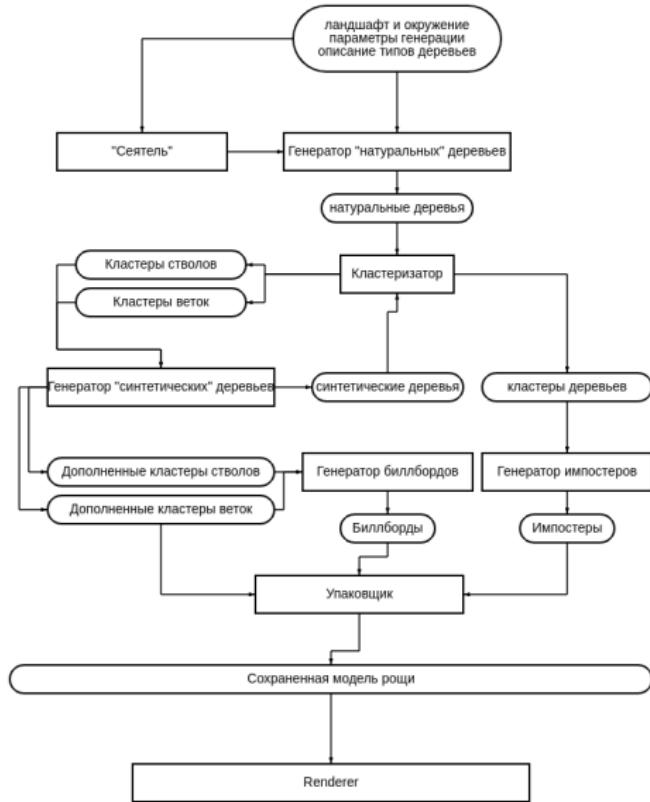
- использование заранее подготовленного набора моделей (обычно)
- создание моделей "на лету" (не всегда подходит)
- создание моделей из стандартных элементов (Minecraft)

"Сборные" деревья

или

的独特性和有效性同时

# General pipeline



# Генератор натуральных деревьев

Основная задача - создание группы деревьев с учетом переданных параметров и свойств окружающей среды.

Входные параметры:

- карта высот и "пригодности" для роста
- описание типов деревьев
- объекты на сцене - например, Bounding Boxes для геометрии уровня
- общие параметра генератора (количество деревьев, точность генерации)

# Посадить дерево

Генератор работает итеративно - на каждом шаге возможна посадка новых деревьев и рост уже существующих.

Для выбора места посадки поддерживается карта пригодности:

$$hab(x, y) = \frac{A}{1+shade(x,y)} * mask(x, y) * (B * f(h(x, y)) + BH(x, y))$$
$$f(h(x, y)) = M - grad(h)(x, y)$$

Shade - карта уровня затененности на плоскости - каждый узел и лист дерева увеличивает затененность там, где он находится.

# Вырастить дерево

Дерево представляется иерархической структурой веток, состоящих из вершин и сегментов, из вершин растут дочерние ветки

Процесс итеративный, основан на распределении питательных веществ по дереву и их конвертации в рост новых веток, удлинение или продление текущих

На каждой итерации

- считаем освещенность в каждом листе и аккумулируем в корне
- распределяем из корня питательные вещества иерархически по веткам, отдавая предпочтение тем вершинам, из которых можно что-то вырастить - пытается вырастить новые ветки или сегменты
- пересчитываем толщину веток

# Вырастить дерево

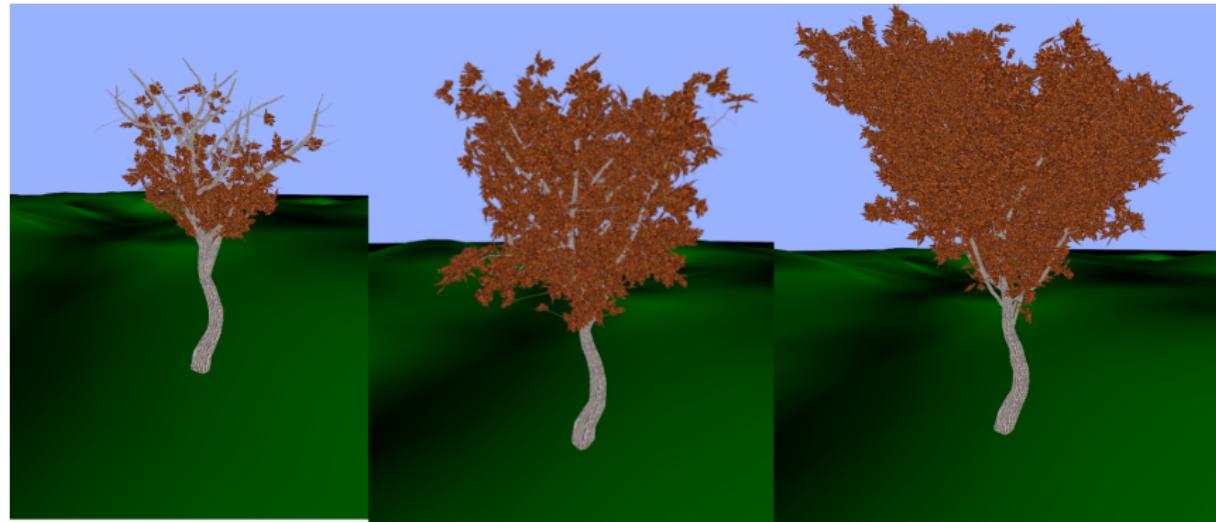
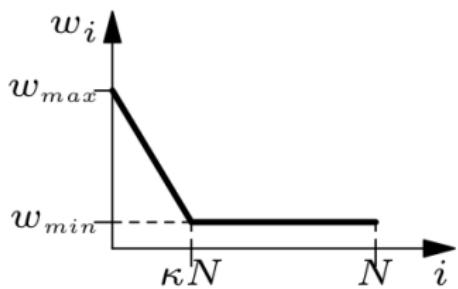
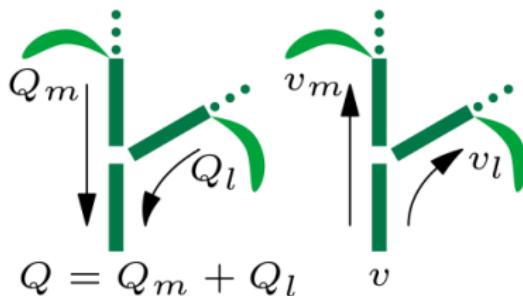
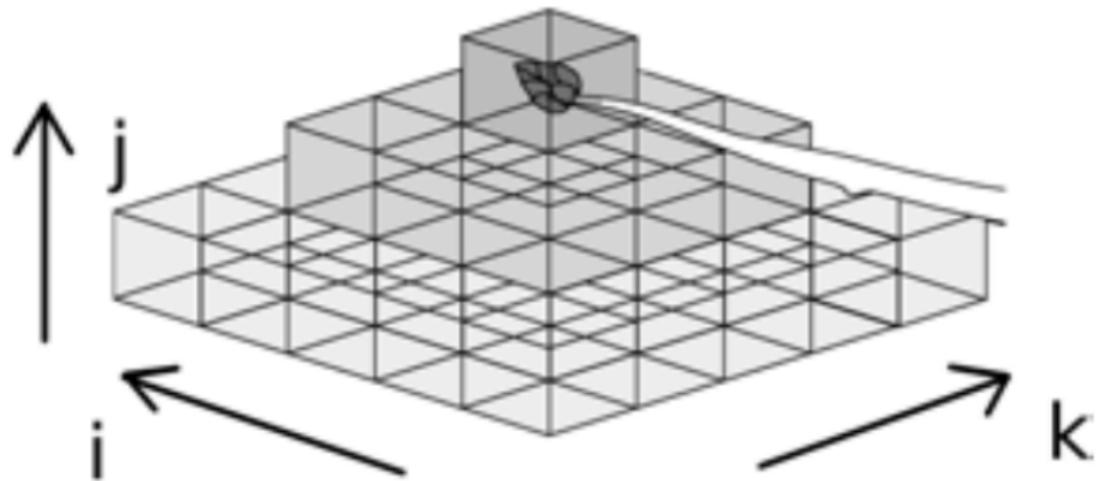


Рис.: 25, 100, 1000 итераций роста

# Распределение питательных веществ



# Воксельная карта освещения



# Воксельная карта освещения

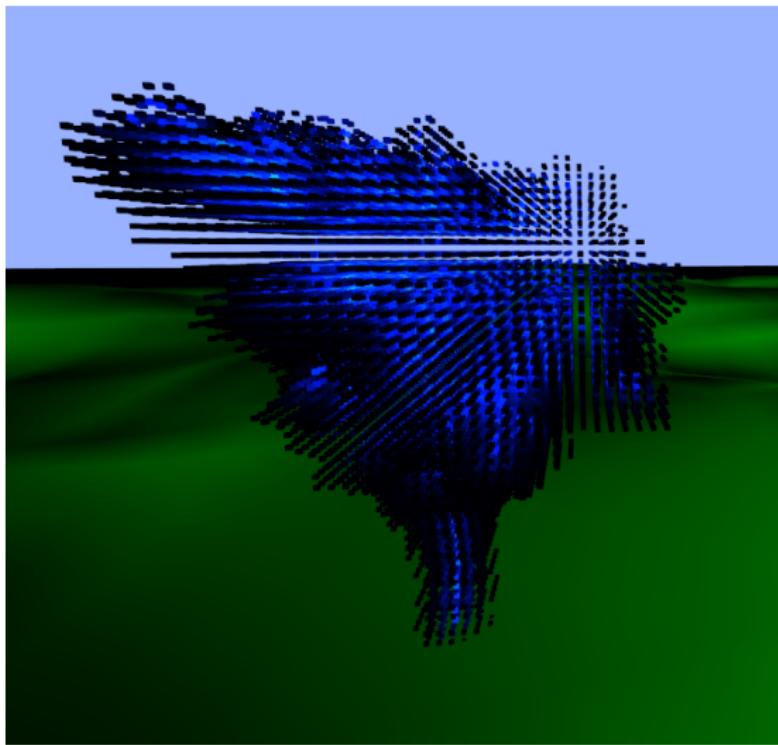


Рис.: Воксельная тень от дерева

# Направление роста

- В окрестности вершины определяется направление с максимальной освещенностью и уровень освещенности
- Для небольших веток - это минимум по соседним voxelам, для более крупных применяется трассировка конусами
- Соотношение уровня освещенности и количества питательных веществ определяет вероятность роста новой ветки
- На ее направление также влияют постоянные параметра дерева и ее положение в структуре дерева

# Параметры

```
max_depth(4),
max_segments(20, std::vector<int>{40, 14, 14, 7, 7}),
max_branching(1),
growth_iterations(200),      You, 2 months ago • Исправил некоторые баги при создании модели деревьев.

scale(3),
seg_len_mult(4, std::vector<float>{2.25, 1.75, 1, 0.55, 0.4}),
leaf_size_mult(2.25),
base_r(1, std::vector<float>{0.9, 0.75, 0.35, 0.15, 0.12}),
r_split_save_pow(2.7, std::vector<float>{1.5, 1.7, 2.3, 3, 2}, EXPLICIT_REGENERATION, new Normal(0, 0.25)),

dir_conserv(1, std::vector<float>{2, 2, 2, 2}, REGENERATE_ON_GET, new Uniform(-0.4, 0.4), 0.1, 10),
plane_conserv(1, std::vector<float>{2, 2, 3, 2, 2}, REGENERATE_ON_GET, new Uniform(-0.4, 0.4), 0.1, 10),
spread(1, std::vector<float>{1, 3, 2, 2, 2}, REGENERATE_ON_GET, new Uniform(-0.25, 25), 0.1, 10),
phototrop(1, std::vector<float>{5, 3, 2, 0, 0}, REGENERATE_ON_GET, new Uniform(-0.2, 0.2), 0.1, 10),
gravitrop(1, std::vector<float>{5, 5, 2, 0.75, 0.25}, REGENERATE_ON_GET, new Uniform(-0.1, 0.1), 0.1, 10),
dir_random(1, std::vector<float>{1, 1, 1, 1, 1}, REGENERATE_ON_GET, new Uniform(-0.4, 0.4), 0.1, 10),

seg_dir_conserv(10, std::vector<float>{10, 10, 10, 10, 20}, REGENERATE_ON_GET, new Uniform(-0.05, 0.05), 0, 1000),
seg_plane_conserv(10, std::vector<float>{10, 10, 10, 10, 20}, REGENERATE_ON_GET, new Uniform(-0.05, 0.05), 0, 1000),
seg_spread(0, std::vector<float>{1, 1, 0.5, 0.5, 1}),
seg_phototrop(1, std::vector<float>{7, 3.5, 1.5, 0.4, 0.1}, REGENERATE_ON_GET, new Uniform(-1, 2), 0, 10),
seg_gravitrop(2, std::vector<float>{0.5, 0.2, 0.07, 0.04, 0.5}, REGENERATE_ON_GET, new Uniform(-2, 4), 0, 50),
seg_dir_random(1, std::vector<float>{1, 1, 2, 3, 5}, REGENERATE_ON_GET, new Uniform(-1, 1), 0, 10),
seg_bend(1, std::vector<float>{1, 1, 2, 3, 5}, REGENERATE_ON_GET, new Uniform(-1, 1), 0, 10),
seg_bend_pow(2, std::vector<float>{2, 2, 2, 2}, REGENERATE_ON_GET, new Uniform(-0.05, 0.05), 0, 10),

base_branch_feed(300, std::vector<float>{800, 350, 200, 40, 40}, REGENERATE_ON_GET, new Uniform(-200, 200)),
base_seg_feed(100, std::vector<float>{200, 150, 120, 40, 30}, REGENERATE_ON_GET, new Uniform(-30, 30)),
feed_distribution_min_weight(0.07),
feed_distribution_d_weight(0.05),
top_growth_bonus(0.0),

light_precision(0.25),
branch_removal(1.2, std::vector<float>{0, 0.75, 1, 1.2, 1.2}),
branch_grow_decrease_q(0.5),
segment_grow_decrease_q(0.05),

min_branching_chance(0, std::vector<float>{-0.3, 0.1, 0.5, 0.75, 0.7}),
max_branching_chance(1),
branching_power(0.5, std::vector<float>{1.2, 0.8, 0.5, 0.5, 0.4}),
r_deformation_levels(2),
r_deformation_points(0, std::vector<int>{8, 3}),
r_deformation_power(0, std::vector<float>{0, 0}, REGENERATE_ON_GET, new Normal(0, 0.025))
```

Рис.: Список параметров генератора

# Разнообразие

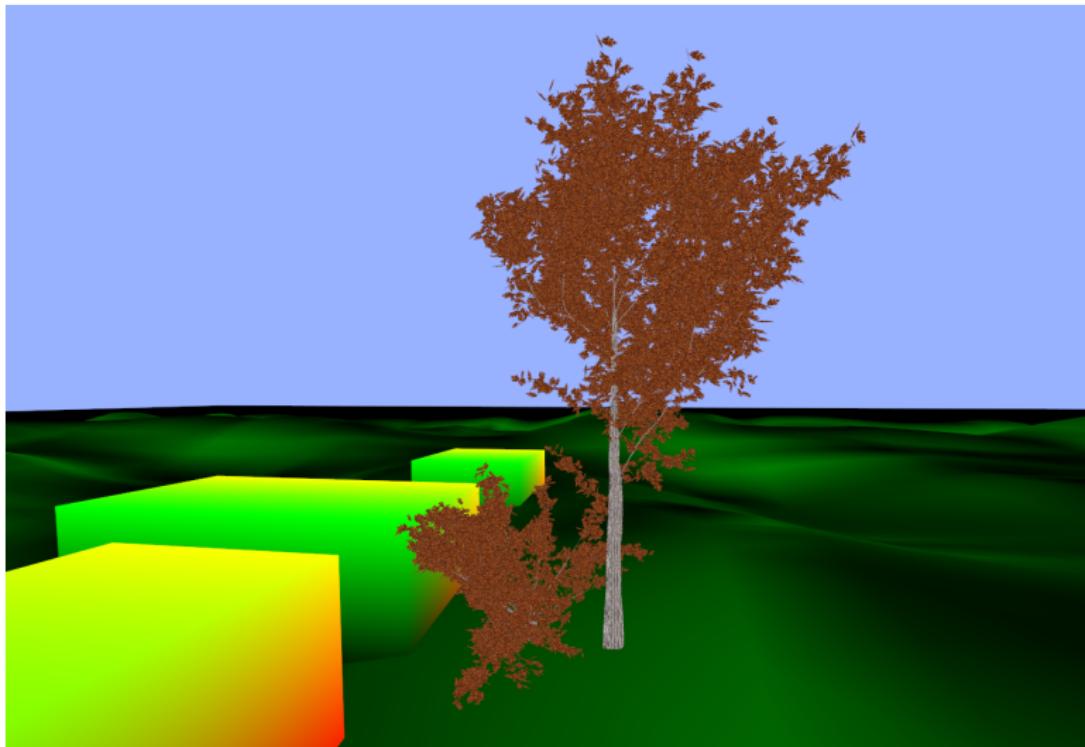


Рис.: Различные наборы параметров позволяют создавать растения разных форм и размеров

## Задача:

По множеству уникальных деревьев построить набор базовых структурных элементов, из которых, используя простые геометрические преобразования, можно получить деревья, внешне минимально отличающиеся от исходных.

# Кластеризация

Представим дерево как ствол и множество веток, растущих непосредственно из него.

Соберем стволы и ветки всех деревьев вместе и разделим на группы (кластеры) похожих.

Сохраним по одному представителю каждого кластера, а остальных выразим через него с помощью матриц трансформации.

# Кластеризация

Зная метрику расстояния между любыми двумя элементами множества, мы можем разделить его на кластеры одним из существующих общих алгоритмов

---

## Алгоритм 1.6. Быстрая агломеративная кластеризация на основе редуктивности

---

- 1: инициализировать множество кластеров  $C_1$ :  
 $t := 1; C_t = \{\{x_1\}, \dots, \{x_\ell\}\};$
  - 2: выбрать начальное значение параметра  $\delta$ ;
  - 3:  $P(\delta) := \{(U, V) \mid U, V \in C_t, R(U, V) \leq \delta\};$
  - 4: **для всех**  $t = 2, \dots, \ell$  ( $t$  – номер итерации):  
5:   **если**  $P(\delta) = \emptyset$  **то**  
6:     увеличить  $\delta$  так, чтобы  $P(\delta) \neq \emptyset$ ;  
7:     найти в  $P(\delta)$  пару ближайших кластеров:  
     $(U, V) := \arg \min_{(U, V) \in P(\delta)} R(U, V);$   
     $R_t := R(U, V);$   
8:     изъять кластеры  $U$  и  $V$ , добавить слитый кластер  $W = U \cup V$ :  
     $C_t := C_{t-1} \cup \{W\} \setminus \{U, V\};$   
9:     **для всех**  $S \in C_t$   
10:       вычислить расстояние  $R(W, S)$  по формуле Ланса-Уильямса;  
11:       **если**  $R(W, S) \leq \delta$  **то**  
12:          $P(\delta) := P(\delta) \cup \{(W, S)\};$
-

# Кластеризация

Расстояние Йорда:

$$R^y(W, S) = \frac{|S||W|}{|S|+|W|} \rho^2 \left( \sum_{w \in W} \frac{w}{|W|}, \sum_{s \in S} \frac{s}{|S|} \right); \quad \alpha_U = \frac{|S|+|U|}{|S|+|W|}, \quad \alpha_V = \frac{|S|+|V|}{|S|+|W|}, \quad \beta = \frac{-|S|}{|S|+|W|}, \quad \gamma = 0.$$

формула Ланса-Уильямса:

$$R(U \cup V, S) = \alpha_U R(U, S) + \alpha_V R(V, S) + \beta R(U, V) + \gamma |R(U, S) - R(V, S)|,$$

- все результаты вычисления индивидуальных расстояний кэшируются
- есть предел на индивидуальную дистанцию между двумя элементами кластера
- функция расстояния завершает работу, если предельное расстояние достигнуто

# Метрика расстояния ("похожести")

Все ветки нормализуются - трансформируются так, чтобы их Bounding box был единичным кубом

Метрика расстояния:

$$M(B_1, B_2) = \min_{\alpha \in [0, 2\pi]} M_0(B_1, \text{rotate}(B_2, \vec{x}, \alpha))$$

$$M_0 = a * M_{struct} * (1 - b * R_{diff}) + (1 - a) * M_{light}$$

$M_{struct}$  - метрика структурной "похожести"

$M_{light}$  - метрика "похожести" затенения

$R_{diff}$  - показатель разницы в толщине веток

# Метрика структурной "похожести"

- Сопоставляем вершины сравниваем веток друг с другом
- Начинаем с вершин собственно самих веток, сопоставлению подлежат вершины, находящиеся на расстоянии не более  $\delta * len$ , где  $\delta$  - заданная константа
  - $len$  - длина кратчайшей из сравниваемых веток
- Рекурсивно проводим процедуру сопоставления для дочерних веток, растущих из сопоставленных вершин

Формально получаем множество пар:

$$MJ = \{(j_1, j_2), j_1 \in B_1, j_2 \in B_2\}$$

Тогда:

$$M_{struct} = \frac{1}{W_s} * \frac{\sum_{(j_1, j_2) \in MJ} (sw(j_1.level) + sw(j_2.level))}{\sum_{j_1 \in B_1} sw(j_1.level) + \sum_{j_2 \in B_2} sw(j_2.level)}$$

$sw(n)$  - константный массив весов для возможных уровней вершин

$W_s$  - нормировочный множитель

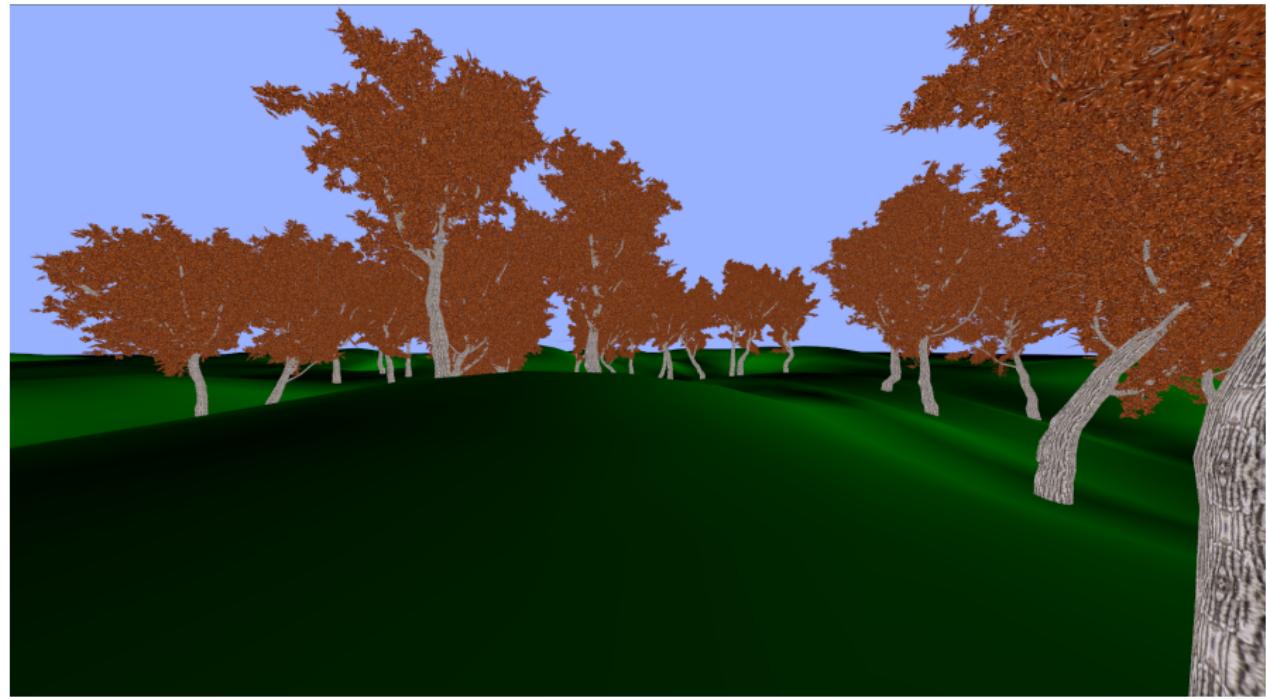
# Метрика структурной "похожести"

$$R_{diff} = \frac{1}{W_r} * \left( \sum_{(j_1, j_2) \in MJ} \int_0^{2\pi} \frac{|j_1.r(\alpha) - j_2.r(\alpha)|}{|j_1.r(\alpha) + j_2.r(\alpha)|} d\alpha \right)$$

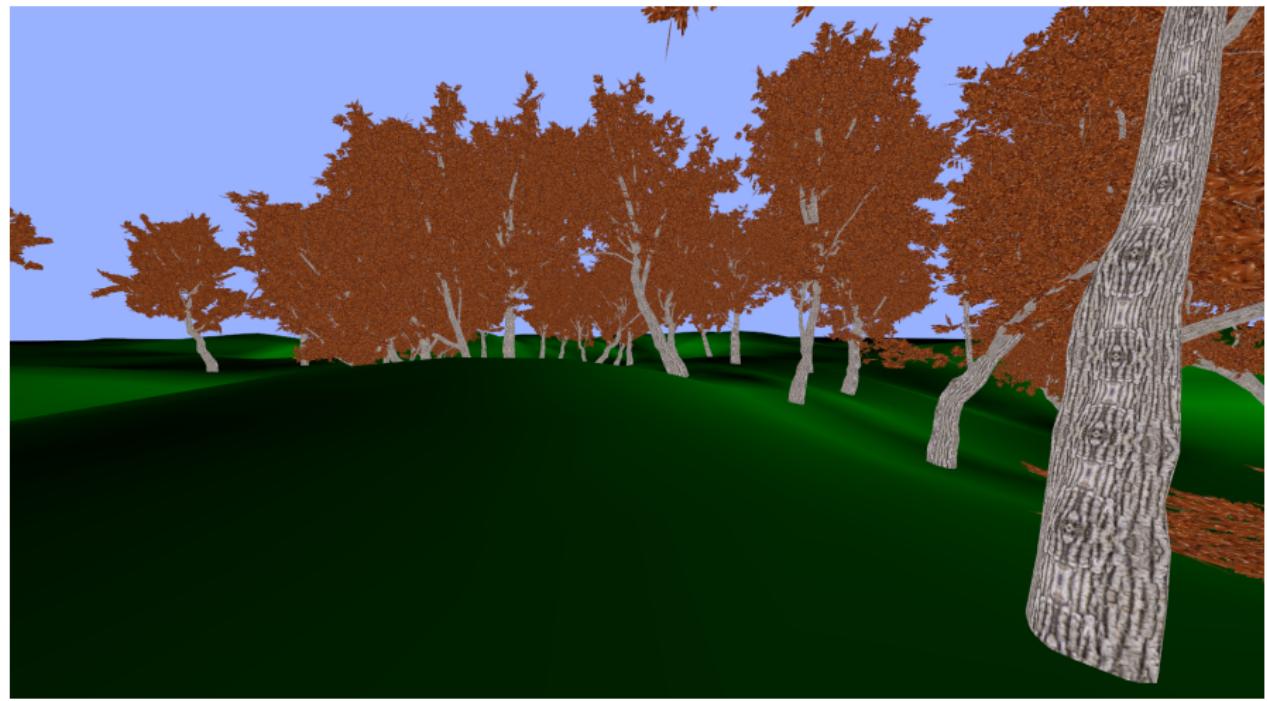
$W_r$  - нормировочный множитель

$$M_{light} = \iiint_V \frac{|V_1(x, y, z) - V_2(x, y, z)|}{V_1(x, y, z) + V_2(x, y, z)} dx dy dz$$

# Без кластеризации



# С кластеризацией



## Производительность

Кластеризация 100 деревьев занимает 10-15 минут

Сложность  $O(n^2)$  от количества веток!

# Синтетические деревья

Большинство деревьев растет в похожих условиях

Деревья одного типа похожи друг на друга

Кластеризация медленная

При росте числа деревьев количество кластеров тоже растет

Решение:

Давайте собирать новые деревья из уже имеющихся частей!

# Синтетические деревья

- 1) Создаем относительно небольшую группу деревьев генератором
- 2) Проводим кластеризацию
- 3) Собираем статистику о параметрах кластеризованных деревьев
- 4) На основании нее синтезируем новые матрицы трансформации для существующих кластеров, которые формируют новые, "синтетические" деревья

# Сбор статистики

```
struct TransformStat
{
    Normal *phi;
    Normal *psi;
    Normal *r;
    Normal *rot_angle;
};

You, 12 days ago | 1 author (You)
struct BranchStat
{
    DiscreteGeneral *typeStat = nullptr;
    TransformStat transformStat;
    bool valid = false;
};

You, 12 days ago | 1 author (You)
struct RootStat
{
    std::vector<DiscreteGeneral *> branchExistanceStat;
    BranchStat selfBranchStat;
    DiscreteGeneral *childBranchesClusterStat;
    int child_branches_count;
};

You, 18 days ago | 1 author (You)
struct FullStat
{
    DiscreteGeneral *rootClusterStat;
    std::vector<RootStat> rootStats;
    std::vector<BranchStat> branchStats;
    FullStat() {};
    ~FullStat();
};
```



# Сбор статистики

Геометрическое положение ветки задается положением ее конечной вершины в сферической СК с центром в начальной вершине и углом поворота вокруг своей оси

Считаем, что параметры эти распределены нормально, а параметры нормального распределения для каждого кластера (= типа) веток находим из статистики

# Создание синтетических деревьев

- Выбираем место посадки по карте пригодности
- Выбираем тип ствола и его геометрические параметры
- Для каждой вершины ствола выбираем, сколько дочерних веток будет из него расти
- Делаем  $N$  попыток выбора дочерней ветки:

Выбираем тип и геом. параметры

Считаем суммарную освещенность ветки

- Выбираем вариант с наибольшим освещением и добавляем ее в воксельную карту освещения
- Формируем матрицы трансформации для ствола и веток

# Подготовка моделей

- кластеризация деревьев целиком
- создание импостеров
- создание биллбордов
- создание 3D-моделей

# Level of Detail

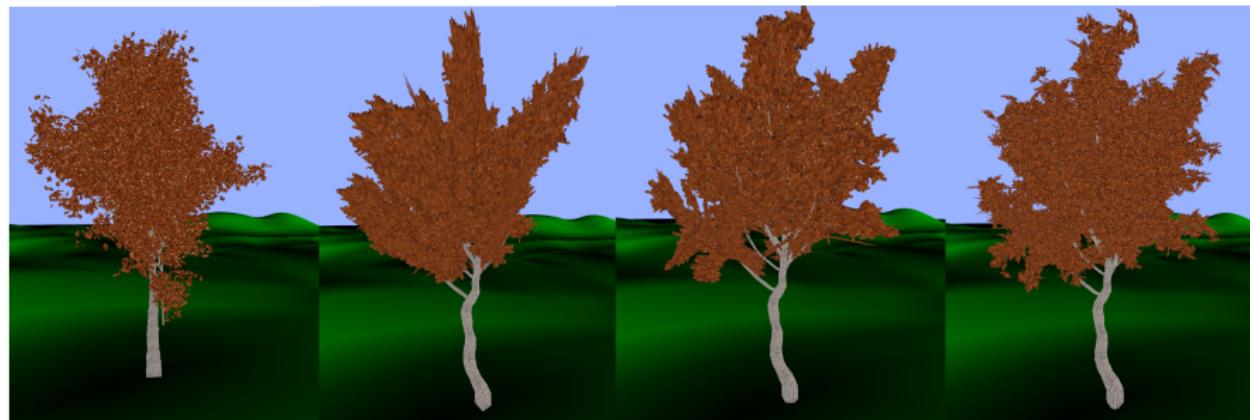


Рис.: 4 уровня детализации

- Эмпирическая метрика оценки визуального качества деревьев
- Автоподбор параметров генератора
- Сбор и использование глобальной статистики
- Реализация кластеризации на GPU
- и многое другое...

Конец

Спасибо за внимание!