This work introduces a way to generate ”plants”, especially tree-esque structures, depending on the simulated light and other environmental influences. The system is designed to model the interactions of a medium to large size of entities, less focused on creating single organisms to export and use in unrelated contexts and more on on the generation of whole ”forests”, i.e. many interacting individuals, representing a base to simulate full ecosystems with even more complex and entangled behavior in a semi-realistic fashion. Represents a base for simulated ecosystems, It is somewhat game-focused, as The goal was less to introduce a Goal of this work

1 Introduction

Generating content procedurally by algorithms instead of and in addition to manual creation is an important developing area for software design, in the gaming industry and beyond - easily evident by the widespread use of 3D-Art programs like Speedtree and Substance Designer in the gaming industry, and the outstanding commercial success of games like Minecraft, which draws an important part of it's appeal from it's randomly generated and distinct worldmaps.

Obviously, different use cases require vastly different approaches. The most important distinction one needs to draw is probably the one between robust solutions that need to produce decent results every run (most end user gaming applications fall under this category), and solutions that are meant to be heavily edited and parameterized and can have some ”bad” (whatever that means in the respective context) outcomes. Usually diverse possible results are a key incentive to use procedural generation in the first place, and necessarily the degrees of freedom for the generation system become limited with each additional constraint - especially if those are tight, i.e. fixed in code, and not soft, i.e. subject to manual assessment and editing.

Our method is an approach designed to be fully robust, limiting how much ”worst case” results can deviate from desired outputs, while still allowing the system to do as many unexpected things as possible. How we tried to tackle this problem can be found in chapter XX. The goal was to recreate behavior found in real-life biological grow, namely by modeling rules that govern the development of biological organisms, and less by working with arbitrary constraints that lead to creations that look similar to the originals but because of different reasons than those.

2 Related Work

Different approaches have been taken on rendering plants,

One example would be SpeedTree, which can generate very realistically looking results, and gets used by 3D Artists for this purpose a lot, but achieves this by doing Lindenmayer-Systems work by mapping states of a generated object to -the

3 Implementation

The implementation was done in Unreal Engine 4 (UE4 or UE in short), a framework developed by Epic Games. Originally meant to be used as a basis for video games, it soon found applications in many different fields after it became available to the general public in 2014. It offers a strong framework for efficient rendering, collision detection, physics and alike (i.e. it solves many fundamental but hard 3D problems) and is completely free of charge for use cases likes us. Especially the collision detection systems were used extensively in this project, and UE's ample debugging and performance profiling tools proofed very helpful at various points in development, too.

Most of the code was done in C++, as especially the generation of the organisms is very performance-critical. Blueprints, the visual scripting language of Unreal, was only used occasionally for prototyping and non-heavy-lifting work, e.g. keyboard input handling and variable/property setup.

The design idea at the very fundamentals was to implement grow logic on the basis of cells, and understand organisms as a collection of connected cells that interact with each other, but are otherwise mostly independent. This is, in a kind, a derivation from Lindenmayer-Systems and indeed would it be possible to encode all cell states in such a way that it would fulfill the respective definition. Yet the resulting strings would be very long, and interactions based on positioning in a 3D space are far from the way L-Systems usually work, so it makes a lot of practical sense to drop the formal constraints of thinking in those limiting terms.

Darstellung von Pflanzen mit Winkel und gleichmäßiger Verteilung

Plant cells always have a "attachment parent", not in the sense of being a predecessor in terms of cell division, but in being the cell Accordingly, there is a root cell for all plants, and all other cells are "children" or "children of children" and so on of the root cell. Constructed that way, the plant can generate it's visuals recursively, where every cell first draws itself and then it's child or children. This way, position changes (e.g. cell divisions) can easily impact positioning of cells at the treetop. Different cell types may have different properties (size, whether the cell is influenced by gravity, etc.). This cell type properties are stored on a per-organism basis. i.e. if there is one leaf cell type, all leaf cells will have the same "genetics". On the other hand, interactions and environmental influences (collisions, if the cell gets hit by light) are calculated for each cell separately. Based on this environmental influences, including passing time, is also decided how a cell divides. Each cell type has divide definitions that determine whether the cell divides into horizontally (creating a fork) or vertically (prolonging the branch) arranged children.

TODO erklärung warum man so baumstrukturen beschreiben kann, die noch viel diverser sind als das was wir hier benutzen.

TODO Wind, wasser, licht, gewicht, Kollision und Selbstkollision

4 Results

5 Used Variables

1. $\mathcal{M}_P$ Mindset of the Player P
2. $\mathbb{S}$ Seifert-Space
3. $\Tlambe$ (Lambe) Game Feature Space TODO DOES LAMBE CONTAIN FEATURE COMBINATIONS THAT HAVE NO CORRESPONDING Z?????
4. $\Tsilme(f)$ (Silme of f) Structuredness of the function f
5. T A partial strategy (Transformation)
6. $\widehat{T} = T_1 \cdot T_2 \cdot T_3 \cdot ...T_n$ The Transformation that equals the concatenation of "all" (depending on the context) Transformations (e.g. is $\widehat{T}_P$ the concatenation of all Transformations of Player P
7. $\Troomen(\widehat{T})$ (Roomen of T hat) Short for $\Troomen(\{\widehat{T}\}) = \Troomen(\{T_1, T_2, T_3...T_n\})$ [1]
8. $\mathbb{G} := G|\forall G' : G \notin G'$ Allgame. All Games except itself are part of and representable in $\mathbb{G}$. We can view it as the representation of "the Universe". $\mathbb{G}$ has no Playerslots [2]

[1] Note that this is a game-theoretical equivalency as $\widehat{T} = T_1 \cdot T_2 \cdot T_3 \cdot ...T_n$ and not just the same thing

[2] At least according to the prevailing opinion. Elon Musk certainly doesn't agree.