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

3.1 Engine, Language & Code

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.

3.2 General

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. Cells then can transform or divide into other cell types, a process which forms complex organisms in the end. 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.

Plant cells always have an "attachment parent", not in the sense of being a predecessor in terms of cell division, but in being directly "below". 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.

Accordingly, each cell needs to be represented by a single mesh - which imposes a problem, as Unreal usually allows around 1200 draw calls per frame if 60 fps are supposed to be hit.Thus, UE projects can easily be CPU-bottlenecked by draw calls, even though the amount of rendered polygons isn't exhausting the capabilities of the graphics card.

So geometry instancing was used, in the form of Instanced Static Meshes (CITE), which is one of the UE variants of this practice. Instanced Static Meshes render the same 3D model multiple times per draw call, improving the rendering speed by several orders of magnitude. On the downside, the difference between instances is limited to scale, size and rotation, and the system is noticeably less flexible than Spline Meshes (CITE), which would commonly be used to represent linearly arranged meshes.

3.3 Plant Grow As Evenly Spread Self-Similarity

If we look at plant structures so different as rose blossoms (CITE), dandelion seed heads or tree branching (CITE), it turns out that many of those follow a somewhat even spread, and the shapes of the various arrangements are describable by different degrees of freedom. In general, this freedom is radially symmetric in grow direction, so a full description may be given by two angles α_{min} and α_{max}, defining the freedom of the distribution to deviate from the grow direction. The objects growing out of the origin then take positions such that the minimal distance between two objects is minimized - an even spread.

For punctiform origins, like those used in this project, the two angles define two circles on a sphere, and the directional possibilities are the set of vectors pointing from the origin to a point on the sphere between the two circles. Thus, a dandelion-like structure corresponds to a minimum deviation of $\alpha_{\text{min}} = 0$ (i.e. directions can be arbitrary close to the grow direction) and a maximum deviation of $\alpha_{\text{max}} \approx 90°$. For "flat" layouts, like those of the petals of the rose in (CITE), is $\alpha_{\text{max}} - \alpha_{\text{min}} \approx 0$ (or small, if there are different layers of petals).

If the number of objects that grow out of a given origin becomes small, such as in the tree in (CITE), this principle is less obvious on first glance, but it remains a special case of $\alpha_{\text{max}} - \alpha_{\text{min}} \approx 0$. In the case of trees, the "objects" that grow out of the origins are actually branches, and can branch, i.e. form new origins, in itself. This is the self-similarity in the title of this chapter, and this is why L-Systems can simulate tree-shapes as well, and why the golden ratio, being an expression of self-similarity, can be found in countless different contexts in biology.

The practical implementation of these principles obviously imposes difficulties, which were only solved by approximation in this project. For example, the distance of objects growing out of origins was approximated by the "distance" of grow directions only, and the algorithm used to generate these directions only yields semi-satisfactory results for $\alpha_{\text{min}} \neq \alpha_{\text{max}}$. More variety in basic forms was cut in scope in favor of factors of interacting grow - see chapter 3.4.

Maximales Limit

3.4 Influencing Thingies

The following behaviors are simulated:

- Plants in water are allowed to have more cells than those without access to it

- Wind / storm coming from a direction, destroying parts of plants that carry to mush weight by the wind
- There is a cone-shaped light source, leading plants to grow in the direction of the light as they divide faster on the faced site, have smaller leaves and to have more allowed cells in general
- The diameter of cells grows when a higher ”weight” rest on them, i.e. when they have many ”attachment-descendants”
- The grow direction can have a positive or negative correlation with gravity, leading to trees that always grow upward, and trees that ”bend over”, even off the stage
- Cells divisions cannot result in children that collide with external objects or other cells of the plant. Picture (CITE) shows a model plant with collision-check enabled (left) and disabled
- Certain cells - in this case the roots - grow on the ground and up to walls when they hit them (CITE)

As hinted above, the numerous collision features of Unreal were used extensively. On the one hand directly to check for overlapping collision shapes (e.g. for the water mechanics) or in the form of ”Raycasts/Line Traces” that determine where a point that travels on a given line segment hits something. This was, for example, used for the light and wind mechanics. One possible influence of light is shown in Picture (CITE). The light comes from the left, and the source is very close to the tree on the left. This tree has noticeably more leafs and children than the one further in the background, even though the ”genetics” / properties of the plants are the same. Additionally, it can be clearly seen how the tree divided more on the site facing the light source, and the image also shows nicely the vastly different possible diameters of trunk cells depending on the number of children they have, and the effect of the trees bending upwards as they have a positive correlation with gravity.

3.5 Paramteter / gewähltes teilungsverfahren

All branches divide out of leafs - leafs divide after some amount of time, and instantly (i.e. on the next iteration of recalculating cell models) if they are hit by a certain amount of light raycasts

The base structure of the division was the following: This system was inspired by (CITE)

PROPERTIES RANDOMED

There was a system implemented to random the parameters of the system - how many children dark light etc

4 Results

For closeness to reality and performance reasons, the number of cells is limited Man sieht: wasser BILD man sieht: licht BILD

For Leafs this is actually a property - some randomed trees have this behavior, others do not - it turned out that trees with very dense leafs are very hard to model with leaf collision enabled.

Interessanter Unterschied der Bäume im Wasser und der Bäume im Licht

Einfluss von Licht auf Bäume, zwei Varianten

Blätter Downscaling

Gesamtmap gerandomed

Große Bäume die über die Seite wachsen/Wurzeln, die das tun???

5 Ausblick

Mehr Interaktionen Merh Ausblick -> Nur minimalen Anteil der Vielfalt der Ausgangsformen verwendet Ganzes Ökosystem

The work done in this project was merely a starting point.

In two directions - at the beginning It resembles a basis to simulate whole ecosystems in an interactable manner, and could be extended towards etc , or later even moving organisms (i.e. "animals") that move around and consume parts of

nutrient circulation Tree dead, by age, illness or similar things, and decomposers, that feed on the leftovers and recycle the nutrietients for the ecosystem.

On the other hand, the range of possibilities describable in the terms of chapter 3.3 could only be scratched at the surface. Especially

6 Used Variables

1. $\mathcal{M}_P$ Mindset of the Player P
2. $\mathbb{S}$ Seifert-Space
3. ꞇ (Lambe) Game Feature Space TODO DOES LAMBE CONTAIN FEATURE COMBINATIONS THAT HAVE NO CORRESPONDING Z?????
4. ɕ(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. ɣ$(\widehat{T})$ (Roomen of T hat) Short for ɣ$(\{\widehat{T}\}) =$ ɣ$(\{T_1, T_2, T_3...T_n\})$ [1]

[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

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]

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