GPU-Based Real-Time Procedural Distribution of Vegetation on Large-Scale Virtual Terrains

Bruno Torres do Nascimento, Flavio Paulus Franzin, Cesar Tadeu Pozzer
Universidade Federal de Santa Maria
Programa de Pós-graduação em Ciência da Computação
Santa Maria, Brazil
Emails: {brunotn, ffranzin, pozzer}@inf.ufsm.br

***Abstract*—We present a novel framework for real-time procedural distribution of vegetation, capable of handling large-scale terrains. Our approach considers several natural aspects that influence the adaptability of each plant type to topographic and environmental factors displayed across the terrain, as well as interactions between different plant types. The adaptability of each plant type is modeled through a set of consistent parameters that afford full control to the user over the final results of the distribution process. The proposed architecture relies on GPU parallelization and GPU instancing to improve performance. Our framework can be used to generate the vegetation cover of a terrain at runtime or to create an initial distribution that could latter be manually edited, expediting the process of decorating large environments. The results show that our framework can achieve natural looking vegetation distributions, while maintaining the computational costs compatible with real-time applications.**

***Index Terms*—vegetation distribution; procedural; large scale; virtual terrains; real-time; GPU-based.**

I. Introduction

Vegetation distribution is a common aspect of creating natural looking virtual landscapes. Small environments are usually decorated by an artist or designer, which allows for fine control and pleasant looking results. However, this process tends to be time-consuming and can quickly become impractical as the size of the environment increases. Moreover, manual placing of vegetation can only be done in predetermined static terrains.

Modern applications, such as simulators and games, increasingly present open worlds with large-scale or even infinite procedurally generated terrains. This demands the use of techniques that can take advantage of current graphic cards in order to generate the vegetation cover and render these terrains.

Several approaches tackle this problems from different perspectives. The concept of ecosystems [2] [8] [9] [11] is commonly used to model the interaction between different types of plants and can be combined with processes that simulate plant growth [7] [8] and competition [1] [12] to attain a realistic placement.

In order to achieve visually consistent results when compared to real-world environments, our distribution process is driven by a set of *Adaptability Parameters* that models biotic (e.g. plant interaction) and abiotic (e.g. terrain features) natural factors that might influence each type of plant. This does not intend to be a realistic simulation, but rather to afford control to the user to allow the achievement of virtually any sort of distribution.

Our approach offers a novel framework for *real-time procedural distribution of vegetation* capable of handling large-scale — static or procedurally generated — terrains, by using the computational power of modern GPU.

The factors that affect the result of the distribution are evaluated on GPU in real-time and interpreted as *Maps* (Fig. 6) that represent the occurrence and magnitude of those factors on the terrain. The *Adaptability Parameters* are used in association with the *Maps* in a deterministic process that evaluates the terrain to determine the placement of every individual plant. Stochastic values are used to achieve variation in regard to type and spatial positioning of the plants distributed on the terrain.

The architecture of our framework employs a *quadtree* to partition the terrain, manage memory usage and avoid the computational cost processing large areas at once, thus allowing the handling of large-scale terrains. The *quadtree* is supported by static sized buffers and a hash system in order to efficiently store, distribute and render the vegetation.

To lessen the overhead of rendering a large amount of plant models, each type of plant is treated separately using GPU instancing to render all the instances that use the same model.

We claim the following technical contributions: We present a framework for *GPU-based procedural distribution of vegetation*, capable of efficiently handling large-scale terrains in real-time. The user is afforded control over the distribution process through a consistent set of parameters that aim to represent biotic and abiotic natural factors in order to accomplish natural looking results. The framework is scalable and can be extended to incorporate new *Adaptability Parameters* and consider other natural factors. To our knowledge, ours is the first approach to employ an architecture for GPU-based procedural distribution of vegetation that can be applied to procedurally generated terrains of virtually any size efficiently.

This paper is structured as follows: Section II explores related works. In Section III we present an overview of our approach, which is discussed more in depth in Sections IV through VI. Finally, in Sections VII and VIII we present and discuss the achieved results.

II. Related Work

Several solutions tackle the problems of vegetation placement from different perspectives. Works presented by Cordonnier et al. [8] and Ch'ng [7] consider aspects of plant growth to attain a realistic placement. Deussen et al. [9], Cordonnier

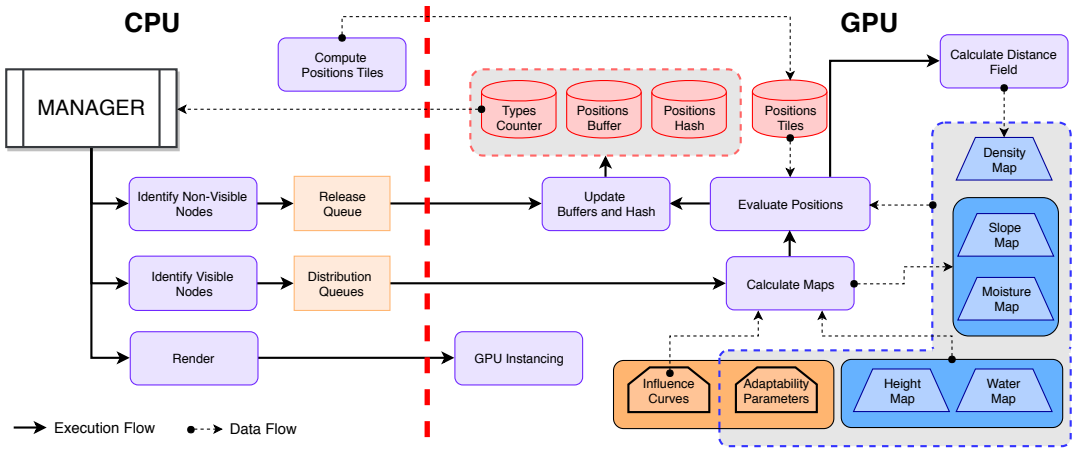


Figure 1. Overview of the architecture. Red, blue and orange elements on the GPU side represent data stored in VRAM.

et al. [8], and Beneš, Andrysco and Šťava [2] use the concept of ecosystems to model the interaction between different types of plants in the context of simulations not aimed at real-time applications.

Several approaches that focus on real-time applications present great performance, but the placement result is less realistic. The *Field of Neighborhood* (FON) approach, introduced by Berger, Hildenbrandt and Grimm [5], considers a zone of influence of a plant defined by a circular area. Alsweis and Deussen [1] applied the FON approach with pregenerated tilling, considering different distributions and densities. This solution was extended by Weier et al. [15]. The use of pregenerated tiles offers a good solution for vegetation distribution at a low computational cost for real-time applications.

Looking for the improvement of plants placement, Hammes [11] considers several abiotic factors, such as height, relative height, slope and directional slope of the terrain. Ch'ng [7] complements that approach by also considering biotic factors to simulate plant growth. However, this approach is more suitable to simulations rather than real-time applications.

Procedural approaches for vegetation distribution are appealing to populate larger areas, although the obtained results do not always look believable. In this regard, Gain et al. [10] employ a procedural approach, using biotic and abiotic factors, to generate an initial distribution that is manually refined at a later stage. A similar approach is presented by Beneš et al. [3], where it is proposed a solution for distribution of vegetation within cities, simulating plant competition for resources, that also requires manual refinement. While manual placement and editing offers a finer control over the final results, this approach is not compatible with large-scale and dynamic terrains.

The concept of subdividing an ecosystem in layers allows for the modeling of dependency relations between plants. Plants on upper layers usually dominate and influence the distribution of those plants in the lower layers. Hammes [11] employs layers in a restrictive manner, where not every combination of ecosystem layers is possible. In [13], layers are used as a base for a more artistic manual-based approach, in which the vegetation areas are hand-painted and terrain features are baked based on a predetermined terrain.

Simulation-oriented approaches are usually not aimed at real-time applications and thus can consider a larger amount of factors that influence the distribution of vegetation without focus on execution time. Cordonnier et al. [8] presented the first vegetation placement solution that considers the bi-directional feedback between terrain erosion and vegetation simulation. Deussen et al. [9] proposed a system to distribute plants based on a combination of ecosystem simulation and manual placement. The quality of the results attained by simulation approaches are related to the number of factors analyzed.

Our approach combines several concepts more commonly used in simulations and other types of non-real-time applications, such as tiling, ecosystem layers and environmental factors, to deliver a solution that achieves natural-looking results while maintaining real-time compatible performance. Furthermore, we propose the use of user defined curves to evaluate environmental factors and plant placement parameters, in order to provide a higher degree of control over the results of the distribution process than what is usually observed in procedural approaches. Moreover, our approach to buffer management is designed to provide the necessary improvement in performance when handling large-scale terrains by associating a hash technique with fixed-size buffers.

III. Overview

This section briefly describes the whole process our framework follows — as defined in its architecture (Fig. 1) — to perform the distribution and rendering of the vegetation.

In the setup phase, the *Adaptability Parameters* (Sec. IV-H) for each plant type, as well as the *Influence Curves* (Sec. IV-C) are set. The parameters and curves are used to determine the plant distribution and generation of the *Maps* (Sec. IV-D), respectively. Also, at this point, the base maps (i.e., *Height* and

Water) should be set, either by the user, or procedurally generated at runtime using any suitable technique. The *Position Tiles* (Sec. IV-F) are generated using *Poisson Disk Distribution* (PDD). The *Adaptability Parameters*, *Influence Curves*, base maps and *Position Tiles* are then loaded into the VRAM. The set of plant types for each *Layer* (Sec. IV-B) of the ecosystem is defined. The quadtree (Sec. V-A) is initialized and associated with the layers.

Once the system is set up and initialized, the process of monitoring the quadtree to identify new visible nodes begins. This process happens every frame. The visible nodes are put in a *Distribution Queue* before being dispatched to the GPU to calculate the positions for the plants. Once a predefined number of frames has passed, or percentage of the *Positions Buffer* is filled, non-visible nodes are identified and put in a *Release Queue* to be later removed. Released nodes are put back in the *Node Pool* (Sec. V-B). Visible and non-visible nodes are put in *Priority Queues* (Sec. V-C) before being dispatched to GPU to avoid overhead.

After the visible nodes are identified, the distribution process (Sec. IV) begins with the generation of the Slope and Moisture maps of the terrain area associated with each node, based on the inputted maps and *Influence Curves*. Next, a pregenerated *Position Tile* is selected. Then, every position in that tile is evaluated, along with the Height, Water, Slope, Moisture and Density Maps and the *Adaptability Parameters* to determine which plant type — if any — will be placed. The *Positions Buffer* and *Positions Hash* are updated (Sec. V-D) and the Density Map of the node is generated by calculating a *Distance Field* (Sec. IV-G) on the valid positions and merged with the Density Map of the parent node.

The drawing step uses the information on the *Types Counter* and *Positions Buffer* to efficiently render multiple instances of the same plant type using GPU instancing. The *Types Counter* stores how many positions of each plant type there are in the *Positions Buffer* and is the only data to be transferred form VRAM to RAM.

IV. Vegetation Distribution

The vegetation cover is procedurally generated for every new node in the quadtree in real-time. This process is performed on GPU and, after completed, the positions of the plants are stored in the *Positions Buffer* for future usage.

The distribution is processed per layer (Sec. IV-B), starting in the top layer (layer 1). Our approach ensures no collisions between plants by evaluating positions pregenerated using PDD (Sec. IV-F) and verifying the Density Map of upper layers. Every plant type is distributed based on a set of parameters (Sec. IV-H), which will determine the density and variation of the vegetation cover.

A. Ecosystems

Our approach uses a straightforward concept of ecosystem, similar to that defined by Hammes [11]. The set of all the types of plants in the ecosystem is divided and each subset is assigned to a layer based on size and occupation radius.

Each plant type defined within the same layer of an ecosystem has an associated *Predominance Value* that controls the prevalence of that type relative to the others (Fig. 2). The sum of all of these values in a layer should be 1. During the process of distributing the vegetation of each layer, the *Predominance Values* are used to determine which type of plant will contend for a position on the terrain.

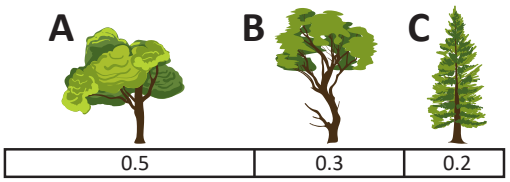


Figure 2. Set of plant types of a layer with its associated Predominance Values. In this example, plant A would contend for 50% of the positions distributed on its layer.

B. Layers

The approach presented in [13] considers layers, which are associated with plant types and other elements, to generate the vegetation cover. However, part of this solution requires manual work, impracticable to use in large or infinity scale. We adapt this concept, integrating it with a quadtree. This approach offers flexibility, since it is straightforward to add or remove layers.

The interaction (e.g., competition for soil nutrients) between plants on the same layer is not considered. However, plants on upper layers can affect the distribution of those on lower layers, since the layers are evaluated from top to bottom. This enables the reproduction of some forms of behaviours. For example, plant types that require more sunlight tend to grow on less dense regions, while other types might thrive in denser environments, due to factors such as moisture retention and soil composition.

In our solution, the layers are associated with a height in the quadtree. We regard three *Layers* (Fig. 3), differentiable by plant height h, described as follow:

- *Layer 1 (L1):* large trees — $h > 6$ m.
- *Layer 2 (L2):* small trees, shrubs — 1.5 m $\geq h \leq$ 6 m.
- *Layer 3 (L3):* ground plants, herbs — $h <$ 1.5 m.

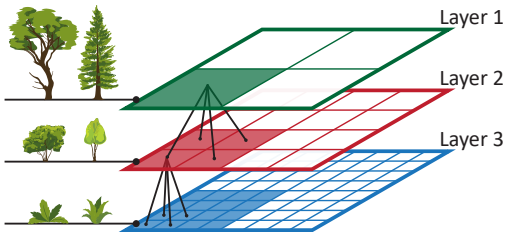


Figure 3. Example of layers defined for an ecosystem. Larger plants occupy the top layer.

C. Influence Curves

In order to allow a fine-tuning of the influence of a terrain feature, our approach uses curves that can attenuate or amplify specific ranges of values. The curves are defined along the x axis, with $x \in [0, 1]$, and are unbounded in the y axis.

There are two sets of *Influence Curves* used in this work in different processes. The first is used to control the generation of the *Moisture Map* (Fig. 4). The second is associated with each type of plant and defines its *Adaptability Parameters* (Sec. IV-H) used in the distribution of the vegetation. In any case, curves are always evaluated using values from maps.

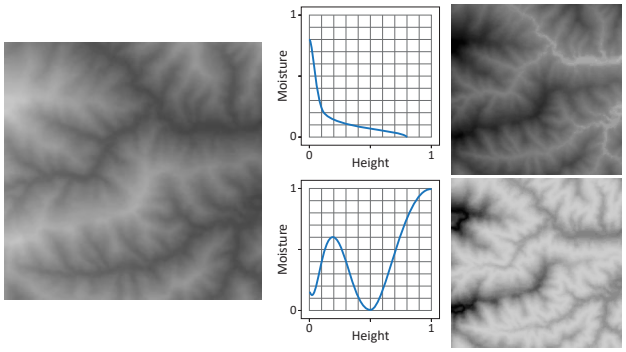


Figure 4. Examples of Moisture Maps (right) generated from the same section of a Height Map (left) using two different Influence Curves (center). Other maps (Slope, Relative Height, Water Spread) were not considered in this example.

To implement all the calculations in the GPU, the curves have to be discretized by sampling q equidistant values along the x axis. The sampled values are stored in buffers — one for each curve — in the VRAM, where they can be accessed directly. The amount of samples (q) determines the precision of the curve in the calculations. The experiments presented in this work used 128 samples. No interpolation is done between sample values. Fig. 5 illustrates how a curve is discretized and its buffer filled with the sampled values.

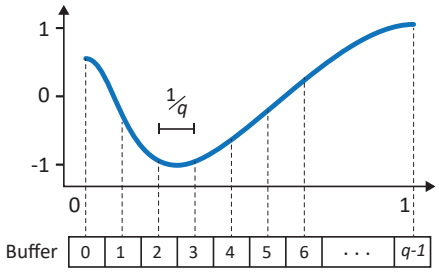


Figure 5. Filling a buffer with q samples from an *Influence Curve*. The numbers inside the buffer represent indexes, not the sampled values.

During the map generation process, each *Influence Curve* has an associated weight that can be used to efficiently modify the global intensity of the influence of the curve without the need to modify the curve itself or perform the resampling and updating of the buffers.

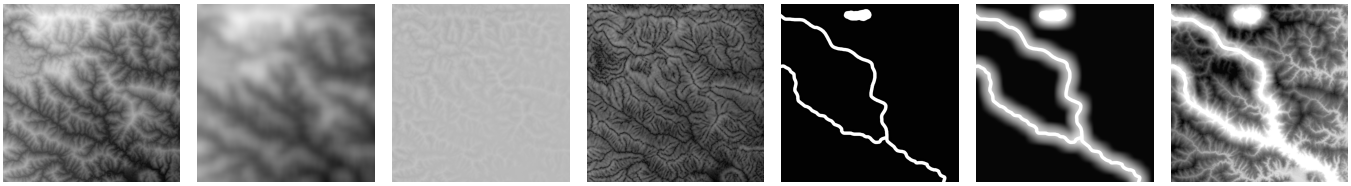
D. Maps

Topographic and environmental features of the terrain are either inputted (e.g. height and water bodies) or calculated (e.g. slope, moisture) based on other features. They are depicted as maps that represent their occurrence on the terrain and are used to evaluate the probability of a plant being placed at a position. Fig. 6 shows an example of all the maps used in the moisture calculations and the procedural distribution of vegetation, except the Density Map.

Every map is stored in VRAM as a single channel 2D texture with values normalized in the range between 0 and 1. The Height and Water Maps are transferred to the VRAM only once as soon as they are loaded. Every other map is calculated directly on GPU and is never transferred to the main memory.

The maps defined in this work are the following:

- *Height Map:* is the main map used for rendering the terrain and for calculating of the other maps that contribute to the procedural distribution of the vegetation. It has a general influence on the moisture level and it is used to generate the Mean Height, Relative Height, Slope, and Moisture Maps.
- *Mean Height Map:* represents the weighted mean of the height values of an area adjacent to a point. The distance (radius of the area) considered is parameterized. The weight of each height value used in the calculation of a point decreases linearly as the distance to the point increases. It is used to generate the Relative Height Map.
- *Relative Height Map:* represents the local variation of the terrain height compared to its surroundings. This map is calculated by subtracting the Mean Height Map from the Height Map. Values below of 0.5 represent depressions on the terrain, and values above represent elevations. Naturally, depressions indicate areas with higher moisture.
- *Slope Map:* represents the local variation of the terrain height along the x and y directions. The slope value is calculated as defined in [11, eq. (3)], with the difference that the distance considered is parameterized. This allows the attainment of more significant slope values, specially in flatter terrains. In general, higher distances produce higher slope values.
- *Water Map:* contains the distribution of the water bodies (e.g., rivers, lakes). Water bodies act as obstacles, preventing the placement of any vegetation on the areas they cover. However, they also contribute heavily with the moisture level of the terrain on its proximity, therefore influencing the local distribution of vegetation.
- *Water Spread Map:* represents the moisture that spreads (horizontally and vertically) through the soil as a result of infiltration from water bodies.
- *Moisture Map:* represents the final value of the soil moisture. It is generated through the compilation of the Height, Relative Height, Slope, Water, and Water Spread Maps.
- *Density Map:* each node in the quadtree has an associated map representing the density of the vegetation cover.



(a) Height (b) Mean Height (c) Relative Height (d) Slope (e) Water (f) Water Spread (g) Moisture

Figure 6. All the maps compiled to generate the Moisture Map. The maps (a), (d), (e) and (g) influence directly the vegetation distribution.

This map is computed as a distance field, showing the combined influence of each plant in a given layer and all the upper layers.

E. Moisture Map Compilation

The calculation of the Moisture Map is more complex since it involves most of the other maps as well as its respective *Influence Curves* and associated weights.

The *Base Moisture* value (BM_{xy}) is defined in (1) by evaluating the height value at the coordinate (x, y) ($Hmap_{xy}$) on the *Influence Curve* of the height ($Curve_h$) and its associated weight ($Weight_h$).

$$BM_{xy} = eval(Curve_h, Hmap_{xy}) \times Weight_h \quad (1)$$

The *Slope Influence* value ($Slope_{xy}$) and *Relative Height Influence* value (RH_{xy}) are calculated in (2) and (3) respectively.

$$Slope_{xy} = eval(Curve_{slope}, Smap_{xy}) \times Weight_{slope} \quad (2)$$

$$RH_{xy} = eval(Curve_{rh}, RHmap_{xy}) \times Weight_{rh} \quad (3)$$

In (4), the value of the *Relative Moisture* (RM_{xy}) is calculated based on the topographic characteristics of the terrain represented in the Relative Height and Slope Maps. This value acts as an indicator of regions where naturally occurs accumulation or decline in soil moisture.

$$RM_{xy} = RH_{xy} - Slope_{xy} + 1 \quad (4)$$

The *Water Spread* (WS_{xy}) calculated in (5) evaluates the influence of the Relative Height ($RHmap_{xy}$) over the vertical water spread ($Curve_{wsVert}$) from a water body ($WSmap_{xy}$). The value of the Water Map ($Wmap_{xy}$) is added to ensure that the *Water Spread* value does not get attenuated in the coordinates where there is a water body.

$$WS_{xy} = Wmap_{xy} + WSmap_{xy} \times eval(Curve_{wsVert}, RHmap_{xy}) \quad (5)$$

The final value of the *Moisture Map* ($Mmap_{xy}$) is calculated in (6) by compiling the values obtained from the other maps. The value of ω, with $\omega \in [0, 1]$, is used to attenuate the direct contribution of the *Relative Height* (RH_{xy}).

$$Mmap_{xy} = saturate((BM_{xy} + WS_{xy}) \times RM_{xy} + (RH_{xy} \times \omega)) + WS_{xy} \quad (6)$$

F. Position Tiles

To ensure a collision-free distribution, while avoiding a large amount of calculations, we choose to pregenerate several sets of positions as tiles. Each tile is generated using PDD [1] [6] to guarantee a minimum distance between positions. Fig. 7 shows the contrast among different techniques used to generate distributions.

(a) Uniform Random (b) Jittered Grid (c) PPD

Figure 7. Comparison between distributions.

During the distribution process of a quadtree node, a random tile is selected to have its positions evaluated.

G. Distance Field

As discussed, the positions tiles generated using PDD ensure a minimum distance between all plants in the same layer. However, plants on different layers might still overlap. A naive solution to this problem would be to compare, at runtime, all the positions of each layer to check for collisions, which would be very expensive ($O(n^2)$).

To avoid this problem, we generate a Distance Field of the positions distributed in a node of a layer. The values of the Distance Field are calculate in (7), where φ is the influence of nearest plant, δ is the euclidean distance to that plant, τ is the *trunk radius* and *ZOI* is the *Zone of Influence* of the plant.

$$\varphi = 1 - saturate\left(\frac{\delta - \tau}{ZOI - \tau}\right) \quad (7)$$

The resulting Distance Field is used to generate the Density Map (Fig. 8). This approach reduces the cost to detect overlapping between positions in different layers to $O(n)$.

The Density Map can be evaluated in a similar manner as the FON, which is considered the *ZOI* of the plant canopy is used only to narrow the plant growth. Our approach, however, allows the Density Map to both boost or narrow plant growth.

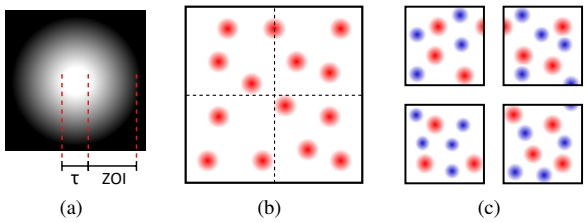


Figure 8. Distance Field definition (a). Density Map of layer (c) is generated by applying a Distance Field on the positions (blue dots) and combining the result with the Density Map of the upper layer (b).

H. Adaptability Parameters

The *Adaptability Parameters* of a plant determine its distribution, considering the direct influence of the terrain and interaction between plants in the same ecosystem. Each parameter is defined using a curve. For every plant placed on a specific position on the terrain, each parameter is evaluated along with the corresponding Map in order to determine the probability of that plant existing on that position (Fig. 12).

This work defines four *Adaptability Parameters*:

- *Height:* plants show a natural sensibility to variations in terrain height, with certain types of plants appearing in specific height ranges and displaying a denser distribution only in restrict sections of these ranges. As the height increases, factors such as temperature and oxygen saturation change, making the general conditions harsher and decreasing the size and density of plants [11].
- *Slope:* the rooting capability of some types of plants is affected by the terrain slope, causing plants to prefer, in general, flatter regions. Areas with steeper slopes tend to be covered by fewer types of plants, which display greater tolerance to steeper inclinations, since in those areas the competition is reduced as plants with lower tolerance cease to appear. Very steep slopes might be off the acceptable range of any type of plant in a ecosystem, leaving the ground exposed with no vegetation cover [11].
- *Moisture:* terrain moisture tends to be the most influential parameter regarding plant density. Most types of plants rely heavily on water to survive. In general, areas with higher height and/or distance from water bodies have less moisture in the soil. In those areas, plant density tends to decrease and types of plants better adapted to dryer climate, such as small trees and bushes, dominate the landscape.
- *Interaction:* it is common that different types of plants share adaptability parameters with similar values, causing those plants to appear in areas that satisfy those parameters. Is those areas, the existence of each plant might be influenced by the presence of the other plants already distributed in the proximity. The *Interaction* parameter aims to reproduce that influence as a combination of several factors — positive or negative —, such as competition for soil nutrients, light absorption, chemical changes in the soil. The degree of that influence is arbitrary and it is up to the user to define a curve that better represent her interpretation of the synergy and competitiveness between specific types of plants, or that allow the achievement of the intended final result. The *Interaction* is evaluated with the combined Density Map of the upper layers.

I. Evaluation

After a *Position Tile* is selected during the distribution process, each position in the set is evaluated on GPU, as delineated in the algorithm presented in Fig. 9, to determine which type of plant is to be placed in that position on the terrain.

```
1:  procedure EVALUATEPOSITION(x, y)
2:      P ← 1
3:      P ← P × (1 − Wmap_xy)
4:      Plant ← getPlant()
5:      Curves ← getCurves(Plant)
6:      l ← index of the current layer
7:      if l > 1 then
8:          P ← P × eval(Curves_interact, DensityMap_xy^(l−1))
9:          P ← P × (1 − ⌊DensityMap_xy^(l−1)⌋)
10:     end if
11:     P ← P × eval(Curves_height, Hmap_xy)
12:     P ← P × eval(Curves_slope, Smap_xy)
13:     P ← P × eval(Curves_moisture, Mmap_xy)
14:     threshold ← getThreshold()
15:     if P ≥ threshold then
16:         update the Position Buffers
17:         DensityMap_xy^l ← 1
18:     else
19:         DensityMap_xy^l ← 0
20:     end if
21: end procedure
```

Figure 9. Algorithm for evaluating a position where a plant might be placed. The coordinates (x, y) inputted in the procedure refer to a specific pixel on the maps.

Initially, the *Probability* (P) is declared and set to the maximum value (line 2). Then, each relevant map is evaluated and P results as the product of the influences of these maps.

In line 3, the complement of Water Map is used to ensure that no plant will be placed inside a water body. It is relevant to note that this process can be extended by defining more maps that represent other types of obstacles (e.g., buildings, roads) and evaluating them in the same manner.

The type of plant that will content for the current position is determined stochastically in line 4. This is dependent on the overall chance each type of plant has to be selected, as defined in the Section IV-A. In line 5, we get the curves of the *Adaptability Parameters* associated with that type.

If the current layer (l) is not the top layer (layer 1), then, in line 8, the Density Map of the upper layer ($l-1$) is evaluated with the *Interaction* parameter. Line 9 ensures that will be no overlapping between the trunk area of plants of different layers

(the collision between plants on the same layer is avoided by using the PDD).

Lines 11 to 13 evaluate, respectively, the Height, Slope and Moisture maps with their associated *Adaptability Parameters*. If the calculated P reaches the $threshold$ determined stochastically in line 14, then the buffers are updated to incorporate the new plant instance and the Density Map of the current layer (l) is set to mark the current position as occupied.

After all the positions in the tile are evaluated, the Distance Field is calculated over Density Map of the current layer and the result is merged with the Density Map of the upper layer (if any), so it can reflect the accumulated density.

V. Architecture

The architecture of our framework is designed to efficiently handle dynamic terrains by minimizing data transfers between RAM and VRAM and performing the vegetation distribution solely in the GPU, while leaving the CPU responsible only for processing quadtree-related operations.

Our architecture employs a quadtree to represent the terrain as nodes and partition the vegetation distribution process in layers. The memory allocation overload is mitigated by using a *Node Pool* to manage the memory usage and to allow the reuse of allocated memory after a node is released.

The GPU overload is managed by using *Priority Queues*, limiting the amount of dispatches per frame and ensuring a steady frame rate. Furthermore, fixed-size buffers are defined and managed in the VRAM for improved performance during the distribution and rendering of plants.

A. Quadtree

The quadtree is a well established and efficient data structure for managing virtual terrains. Our framework uses a quadtree to subdivide the terrain, avoiding the overload of processing large areas at once. This approach also favors the use of GPU parallelization, allowing the vegetation distribution process to be performed in multiples nodes simultaneously.

To further reduce overhead, our approach uses Frustum Culling and *View Distances* (Fig. 10) to ignore nodes outside the View Frustum and limit the amount of distributions based on distance.

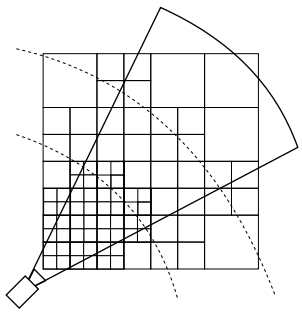


Figure 10. Quadtree refinement based on the frustum and view distances.

B. Node Pool

In the setup phase, the Node Pool is filled with a set of preallocated nodes. All these nodes are marked as not used and will be occupied as the quadtree expands. Once a used node is released, it returns to the Node Pool. If all the nodes are used, the Node pool generates more nodes, on demand.

When the Node Pool is empty, or after predefined number of frames, the quadtree searches for non-visible nodes to be released.

C. Priority Queues

Our solution employs three Priority Queues to manage the amount of GPU dispatches. A *Distribution Queue*, which is subdivided in *Normal* and *High* priority queues, and a *Release Queue*.

The *Distribution Queue* is used for quadtree nodes that require vegetation distribution. Nodes in the *High* priority queue are dispatched before those in the *Normal* priority queue. Nodes are assigned to each queue based on their distance to the camera and the camera's height. For example, when the camera is at a high altitude, top layer nodes are assigned a higher priority, since plants in that layer will have a higher impact on the final image.

The *Release Queue* is used for quadtree nodes that are marked to be released. When a node is released, the Positions Buffer, Positions Hash and Types Counter are updated to remove the plants that existed in the area of the terrain associated with that node. After released, the node returns to the Node Pool.

Priority Queues mitigate the problem mentioned in [11], where sudden changes in camera direction might cause a frame rate drop or gaps in the vegetation cover due to the overhead of processing a large area.

D. Buffers

The buffers used in our architecture are specifically designed to enable the handling of large-scale terrains in a dynamic context, where the all the vegetation is procedurally distributed. All buffer are fixed-size to avoid the overhead of resizing. While this approach might lead to sections of allocated memory being unused in some situations, the trade-off between memory usage and allocation cost is necessary to maintain performance.

To take advantage of the static nature of the buffers, our architecture adopts an approach based on *Spatial Hash* devised to index linear memory [14].

Our architecture employs the following buffers:

- *Plants Pool:* stores the model of each plant type. This buffer resides in the main memory.
- *Positions Buffer:* this 2D static buffer stores in VRAM the positions of all the instances of plants placed on the terrain. Each row in this buffer contains the positions of single type o plant, so all the instances of that plant can be efficiently rendered using GPU instancing. When a node is released and the positions are removed, this buffer

is rearranged to maintain all the valid positions at the beginning of each row.

- *Types Counter:* stores the amount of valid positions in each row of the Positions Buffer. This buffer resides in VRAM and is read back to RAM to be used in the draw calls.
- *Positions Hash:* this buffer contains the *start index* and *count* of each plant type for every node in the quadtree. The *start index* refers to the first index in the Positions Buffer where are stored the positions of a plant type, the *count* is the amount of plants of that type in a given node. For example, in Fig. 11, the *Node 1* contains two plants of type *P1* (A and B), one of type *P2* (C) and one of type *P3* (D). Looking at the portion of the Positions Hash associated with the *Node 1*, the first value indicates where in the Positions Buffer the first plant of type *P1* is stored (index 0, in this case) and the second value represents the amount (2 instances) of plants of type P1 that are associated with the *Node 1*. So we can determine that, starting at index 0, 2 positions in the Positions Buffer store plants (A and B) associated with the *Node 1*.

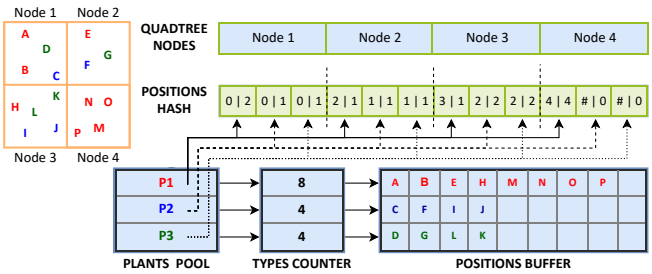


Figure 11. Example of how each buffer stores positions and references plants on the terrain. The *Positions hash* store the initial position and the amount of positions for one node.

This approach is essential to maintain performance when handling large-scale terrains, since it allows the framework to efficiently access and update the section of the Positions Buffer associated with any node in the quadtree.

VI. Rendering

One important challenge when rendering large-scale natural landscapes is the amount of plant instances in the scene. Considering that each plant type usually is represented by the same model (i.e., same mesh), it becomes advantageous an approach that uses GPU instancing to allow the dispatching of multiple instances that use the same mesh to the GPU in a single draw call, increasing the performance drastically [16]. In this context, our framework employs GPU instancing to render all the instances of each plant type stored in the Positions Buffer using fewer draw calls.

It is relevant to note that our framework does not apply any form of Level of Detail (LOD) regarding the meshes of the plants, thus, rendering a large amount of highly detailed meshes can still become overwhelming. This problem can be lessened by applying a LOD technique.

VII. Results

All the experiments and measurements where performed on an Intel Core i7-4790 3.5 GHz processor, with 24 GB of DDR3 RAM and a NVIDIA GeForce GTX 1070 graphics card, with 8 GB DDR5 VRAM.

Table I presents the times to generate each map at different resolutions. The Height and Water maps are inputted rather than generated. Also, since the Density Map is generated in separated steps, its times are not shown in the table, but computed with the times presented in Table II. Further looking into Table I, one can observe that the Mean Height and Water Spread maps are the most time consuming tasks. Their times are directly proportional to the distance used for their computation, in this case, 32 pixels. The distance used for the Slope Map was 12 pixels.

TABLE I
Map Generation Times

Maps	Time (ms) at different resolutions (pixels)				
	32x32	64x64	128x128	256x256	512x512
Mean Height	0.028	0.030	0.058	0.204	0.733
Relative Height	0.006	0.006	0.008	0.012	0.024
Slope	0.007	0.007	0.007	0.012	0.020
Water Spread	0.028	0.031	0.058	0.206	0.737
Moisture	0.007	0.007	0.008	0.015	0.039
Total	0.076	0.079	0.137	0.449	1.553

The effects of the Adaptability Parameters can be observed in Fig. 12. This image demonstrates how two types (red and green) of plants follow the associated curves that represent their adaptability to the terrain. Each row exemplifies the effect that a single factor (height, slope, moisture, and density) has on the results (right column) of the distribution. The Maps on the left column represent the intensity of each factor across the terrain. Defining each parameter using a curve allows the user to compose virtually any sort of distribution that is based on the terrain features and avoids the occurrence of sharp transitions.

The measurements presented in Table II represent the times to distribute the vegetation cover on the entire View Frustum, using different configurations, . Visualization Distances are used to determine the refinement of the quadtree and to select the nodes of each layer (L1, L2, L3) on which the vegetation will be distributed. Forty different types of plants were assigned to each layer. The Node Size affects the refinement and, thus, the amount of nodes that will be distributed. The amount of positions evaluated is related to the radius used to generate the Position Tiles in the setup phase. The Map Resolution was defined to maintain a scale of 2 m/pixel. Most of the CPU and GPU processing occur in parallel.

It is important to underline that these measurements represent the worst case scenario, where all the View Frustum is generated at once. This situation should only occur at the beginning of the execution or when the camera suddenly turns more than the Field of View. Rendering times were

TABLE II
VEGETATION DISTRIBUTION TIMES

View Distance (km)			Node Size (m)			Map Resolution (pixels)	Nodes Distributed			Total Positions Evaluated (± 5%)	Total Time (ms)		Average Time (ms) per Node	
L1	L2	L3	L1	L2	L3		L1	L2	L3		CPU	GPU	CPU	GPU
10	5	3	256	128	64	32x32	418	1088	1421	1,700,000	200.680	366.991	0.069	0.125
			512	256	128	64x64	111	281	366		68.320	128.965	0.090	0.170
			1024	512	256	128x128	31	75	97		29.170	63.084	0.144	0.311
5	3	1	256	128	64	32x32	281	366	98	850,000	66.980	101.749	0.090	0.137
			512	256	128	64x64	75	97	28		22.720	45.717	0.114	0.229
			1024	512	256	128x128	22	28	8		6.530	12.870	0.113	0.222
2	1	0.5	256	128	64	32x32	41	28	9	100,000	7.470	15.880	0.096	0.204
			512	256	128	64x64	13	8	3		2.460	9.389	0.103	0.391
			1024	512	256	128x128	4	3	1		1.060	4.012	0.133	0.502

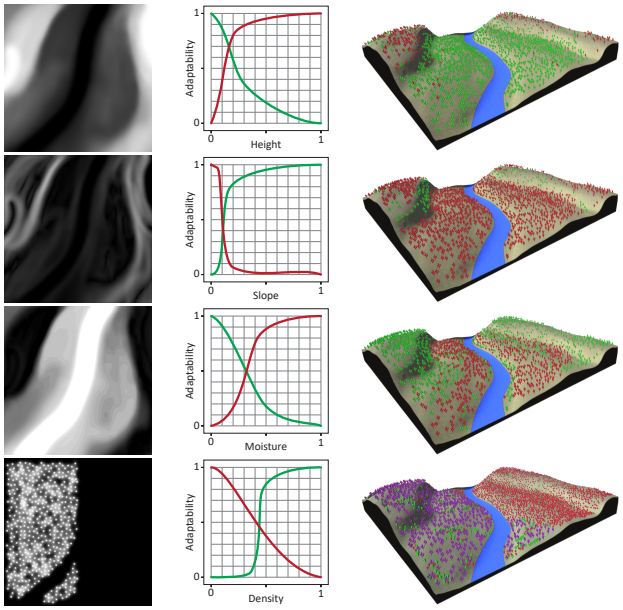


Figure 12. Effects of the *Adaptability Parameters* on the vegetation distribution. The Maps (left) are evaluated along the associated curves (center) to determine the final placement of each plant (right). Each row shows the result of a distribution using only one parameter. From top to bottom, *Height*, *Slope*, *Moisture*, and *Interaction* parameters. Each plant type is represented to a color (red or green), and its adaptability is determined by the curve with the same color. The *Density Map* on the bottom row is generated based on the positions of the purple plants.

Figure 13. Representation of the ecosystem layers. Each picture shows the combination of plants of different layers. Only the layer L3 in the top image, layers L3 and L2 on the middle, and all layers combined in the bottom image.

not considering in this measurements, since they are closely related to the mesh resolution of the plants.

The vegetation distributed on each layer can be observed in Fig. 13. Dense vegetation covers are generated without collisions between plants. Layers are also used to establish a relation between plants. For instance, the ground plants observed on top image have a disposition to grow in denser regions and, thus, appear in parts of the terrain covered by other plants, as seen in the bottom image.

Fig. 14 shows a different angle of the same landscape presented in Fig. 13, where one can observe the distribution of the pine trees, highlighted in the image. In that example, the pine trees belong in top layer (L1), and their Adaptability Parameters — represented by the colored curves — are defined to ensure that they can survive in dry and high regions.

VIII. CONCLUSION AND FUTURE WORK

We have presented a framework for GPU-based procedural distribution of vegetation in real-time, capable of handling

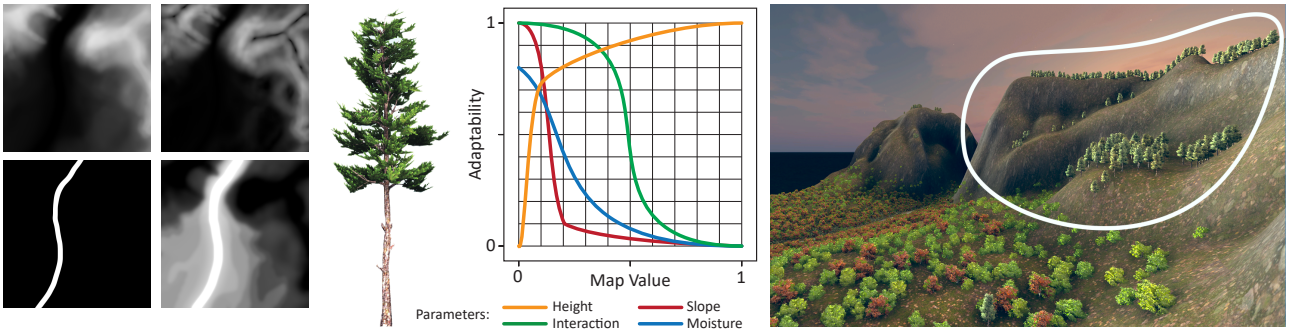


Figure 14. In this example, pine trees dominate the area of the terrain indicated by the white loop because they are better adapted to low moisture and high elevation regions. On the left are the Height, Slope, Water, and Moisture maps.

large-scale terrains, which uses the concept of ecosystems to model the interaction between different plant types, while considering biotic and abiotic factors to determine the adaptability of plants in terrains with various topographic features. This is achieved with a layered ecosystem model that contains a subset of plant types in each layer. Plant types in top layers are considered to dominate, so plants are distributed in a descending fashion. Terrain features are calculated and stored as maps that are evaluated along with Adaptability Parameters to determine the placement of each plant. This technique, associated with a dynamic quadtree, Priority Queues and pre-generated sets of distributions, proved to be an efficient way to handle vegetation distribution on large-scale terrains, allowing control of memory usage and achieving satisfactory times regarding distribution and rendering using GPU instancing.

One drawback of using a fixed buffer to store the positions of the plants is that sudden changes in the View Frustum might require the generation and distribution of new visible nodes, while plants in nodes outside the View Frustum would still be rendered, since the they were not yet removed from the Positions Buffer. This would cause a brief overhead on the GPU due to the distribution process and rendering of plants — including those out of view — occurring simultaneously. Other current limitation is the fact that the framework does not support random access to any plant position. This does not allow dynamic modifications to limited areas that would require a change in the plant models and rearrangement of the Positions Buffer. Common game events, such as woodcutting or forest fires, are good examples of that. This could be improved by using some form of spatial hashing to enable specific changes in the Positions Buffer.

Future work could include integrating a system for procedural generation of terrains and water bodies into the framework, creating a more complete solution for procedural generation of virtual environments. Furthermore, it would be interesting to incorporate maps to represent other obstacles for plants, such as roads and buildings, and also maps to consider more factors, such as sunlight exposure, temperature, and wind. One important open problem is to improve the rendering phase to better handle even larger amounts of plants instances when a very long view distance is defined. A LOD technique could be implemented reduce the overhead and improve performance.

Acknowledgment

We thank the Brazilian Army for the financial support through the SIS-ASTROS project.

References

[1] M. Alsweis and O. Deussen. *Wang-tiles for the simulation and visualization of plant competition*, Advances in Computer Graphics. Springer, Berlin, Heidelberg, 1-11, 2006.

[2] B. Beneš, N. Andrysco and O. Št'ava, *Interactive modeling of virtual ecosystems*, in Proceedings of the Fifth Eurographics conference on Natural Phenomena, pp. 9-16. Eurographics Association, 2009.

[3] B. Beneš, M. A. Massih, P. Jarvis, D. G. Aliaga and C. A. Vanegas, *Urban Ecosystem Design*, Symposium on Interactive 3D Graphics and Games, 2011.

[4] U. Berger, H. Hildenbrandt, *A new appoarch to spatially explicit modelling of forest dynamics: Spacing, ageing and neighbourhood competition of mangrove trees*, Ecological Modelling, 2000.

[5] U. Berger, H. Hildenbrandt and V. Grimm, *Towards a standard for the individual-based modeling of plant populations: self-thinning and the field-of-neighborhood approach*, Natural resource modeling, 15(1), pp. 39-54, 2002.

[6] R. Bridson, *Fast Poisson Disk Sampling in Arbitrary Dimensions*, ACM SIGGRAPH 2007 Sketches. ACM, 2007.

[7] E. Ch'ng, *Realistic placement of plants for virtual environments*, IEEE computer graphics and applications 31.4, 66-77, 2011.

[8] G. Cordonnier, E. Galin, J. Gain, B. Benes, E. Guérin, A. Peytavie and M. P. Cani, *Authoring landscapes by combining ecosystem and terrain erosion simulation*, ACM Transactions on Graphics (TOG), 134, 2017.

[9] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr and P. Prusinkiewicz, *Realistic modeling and rendering of plant ecosystems*, in Proceedings of the 25th annual conference on Computer graphics and interactive techniques pp. 275-286, ACM, 1998.

[10] J. Gain, H. Long, G. Condonnier, and M. P. Cani, *EcoBrush: Interactive Control of Visually Consistent Large-Scale Ecosystems*, Computer Graphics Forum, 2017.

[11] J. Hammes, *Modeling of ecosystems as a data source for real-time terrain rendering*, Digital Earth Moving. Springer, Berlin, Heidelberg, 98-111, 2001.

[12] B. Lane and P. Prusinkiewicz, *Generating spatial distributions for multilevel models of plant communities*, in Graphics interface, vol. 2002, pp. 69-87.

[13] J. van Muijden, *GPU-based procedural placement in Horizon Zero Dawn*, GDC, 2017.

[14] C. T. Pozzer, C. A. L. Pahins and I. Heldal, *A hash table construction algorithm for spatial hashing based on linear memory*, in Proceedings of the 11th Conference on Advances in Computer Entertainment Technology. ACM, 2014.

[15] M. Weier, A. Hinkenjann, G. Demme and P. Slusallek, *Generating and rendering large scale tiled plant populations*, Journal of Virtual Reality and Broadcasting, 10(1), 53-58, 2013.

[16] R. S. Wright Jr, N. Haemel, G. M. Sellers and B. Lipchak, *OpenGL SuperBible: comprehensive tutorial and reference*, Pearson Education, 2010.