The Visual Computer (2020) 36:2263–2272
https://doi.org/10.1007/s00371-020-01920-7

ORIGINAL ARTICLE

AutoBiomes: procedural generation of multi-biome landscapes

Roland Fischer[1] · Philipp Dittmann[1] · René Weller[1] · Gabriel Zachmann[1]

Published online: 24 July 2020

Abstract
Advances in computer technology and increasing usage of computer graphics in a broad field of applications lead to rapidly rising demands regarding size and detail of virtual landscapes. Manually creating huge, realistic looking terrains and populating them densely with assets is an expensive and laborious task. In consequence, (semi-)automatic procedural terrain generation is a popular method to reduce the amount of manual work. However, such methods are usually highly specialized for certain terrain types and especially the procedural generation of landscapes composed of different biomes is a scarcely explored topic. We present a novel system, called AutoBiomes, which is capable of efficiently creating vast terrains with plausible biome distributions and therefore different spatial characteristics. The main idea is to combine several synthetic procedural terrain generation techniques with digital elevation models (DEMs) and a simplified climate simulation. Moreover, we include an easy-to-use asset placement component which creates complex multi-object distributions. Our system relies on a pipeline approach with a major focus on usability. Our results show that our system allows the fast creation of realistic looking terrains.

Keywords Procedural content generation · Terrain generation · Virtual worlds · Biomes · Climate simulation · Digital elevation models

1 Introduction

The ever-rising demand for bigger and more complex virtual 3D worlds poses a challenge for designers to create and fill them with life. There is a broad range of applications for huge and realistic 3D landscapes, e.g., computer games, movies and simulations. With the rising accessibility of head-mounted displays (HMDs), there is also an increasing opportunity to explore these worlds in virtual reality in a more immersive environment. Generating these worlds manually is a laborious and expensive task [1]; therefore, extensive research was done in the field of procedural terrain generation (PTG). Yet it remains an important topic, as there is still much potential for improvement. Numerous algorithms for PTG have been proposed which can be roughly categorized into three types: synthetic, physics-based and example-based approaches [7,9]. Each of these approaches comes with its own strengths and weaknesses. Most of the currently used methods and terrain generators follow one of the mentioned approaches and emphasize only on a single, very specific use case. Hence, they are hardly capable of satisfying a broader set of requirements [20]. In consequence, it remains a challenge to create a system with a reasonable compromise of the four most essential but mutually contradictory requirements: realism, performance, usability and flexibility.

Two other significant factors of creating plausible, detailed 3D worlds received not much attention in the past: the distribution of assets and the generation of landscapes as a combination of different biomes. However, with the rising dimensions of 3D worlds the interest in landscapes with various characteristics is growing. The procedural distribution of assets faces similar challenges in balancing the requirements as the terrain generation itself and is equally important to create a convincing environment with an organic feel.

Our main contribution is the design and implementation of a PTG system which combines the three main approaches, synthetic, physics-based as well as example-based PTG, and unites the respective advantages to an effective and well-balanced terrain generator. The goal is to generate realistic

✉ Roland Fischer
rfischer@cs.uni-bremen.de

Philipp Dittmann
dittmann@cs.uni-bremen.de

René Weller
weller@informatik.uni-bremen.de

Gabriel Zachmann
zach@cs.uni-bremen.de

[1] University of Bremen, Bremen, Germany

terrains while keeping simultaneously computation times low and still considering usability and flexibility.

Our focus is not restricted to the generation of huge terrains but covers specifically terrains composed of different biomes, which is a relatively sparsely explored topic with additional challenges. As part of our PTG architecture, we propose an effective biome- and rule-based local-to-global model to populate the terrain with assets. This component is a vital step to produce a comprehensive solution for creating convincing 3D landscapes.

Finally, our system is implemented in the Unreal Engine and designed to be used completely from within the editor. Optionally, the exported heightmaps can be used in external applications.

2 Related work

Procedural generation is used since the 1980s and numerous different methods were developed. Noise-based methods belong to the synthetic approach of PTG, one of the oldest and most widely used techniques. Examples of well-known noise functions are Perlin noise by Perlin [17] and his improved version named Simplex noise [18]. More complex results can be achieved by combining multiple instances of noise with different frequencies, called fractal noise. Terrain generation using noise is very popular, because it is easy compared to other approaches and the computational effort low. Drawbacks are the inherently unintuitive way to adjust noise parameters and consequently, the difficulty to create genuinely realistic looking terrain, as described in [10].

On the other hand, physics-based procedural generation methods have their focus on creating realistic results at the expense of lower computation speed. Very common are erosion algorithms which try to create the terrain by simulating the natural erosion processes. In 1989, Musgrave et al. [15] proposed models for thermal and hydraulic erosion simulation which became the basis for a lot of the subsequent research on this topic. Jákó [12] adapted and improved previous work and presented a faster implementation using the GPU. Another approach is the simulation of fluid dynamics. Most of its techniques either are grid-based, called Eulerian, or particle-based, called Lagrangian. The leading concept for the latter ones is smoothed particle hydrodynamics (SPH), which was well summarized by Ihmsen et al. [11]. Well known is also the work of Jos Stam, who eventually presented a convincing real-time fluid solver [21] which combined both approaches.

Another concept for PTG techniques is based on using examples, e.g., images or user sketches, and synthesizing terrain according to it. DEMs are digital representations of real ground surfaces, commonly parts of the earth's topography, and can also serve as examples for PTG. Using these DEMs and texture synthesis methods, Zhou et al. [24] presented a system capable of generating realistic looking terrains if provided with appropriate and detailed data. Not long ago, generative neural networks could successfully be applied in the field of PTG. Recently, Beckham and Pal [2] and Wulff-Jensen et al. [23] trained deep convolutional generative adversarial networks (DCGANs), developed by Radford et al. [19], on DEMs to create similar looking heightmaps for terrain generation. Similarly, Guérin et al. [8] used conditional generative adversarial networks (cGANs) to create a set of task-specific synthesizers which generate terrain features based on sketches. Gatys et al. [6] also proposed an interesting technique called style transfer where convolutional neural networks (CNNs) learn to combine the artistic style of one image with the main features of arbitrary other images.

A comprehensive overview of all kinds of procedural terrain generation and modeling techniques is given by Galin et al. [4].

In the domain of generating asset distributions, two different concepts can be found. Local-to-global models are based on the individual object instances, and by constrained-based placement and simulation of interactions, the resulting distribution is determined. Global-to-local models, on the other hand, infer the position of individuals by a beforehand defined distribution. Both Deussen et al. [3] and Lane et al. [14] presented convincing individual-based simulation models to generate plant distributions. A popular distribution to sample objects from is the Poisson distribution, which ensures a minimal distance between samples. Early techniques for Poisson-disk sampling relied on the dart-throwing principle. Jones [13] introduced the combination with a spatial data structure later. Also using spatial subdivision, Gamito and Maddock [5] proposed an accurate and considerably faster algorithm.

3 Our approach

We present a PTG system which combines synthetic, physics- and example-based approaches to produce vast landscapes composed of different biomes and populated with huge amounts of assets. We chose an incremental pipeline design with a focus on high performance to ensure providing the user with quick results. The pipeline currently consists of four individual main steps where each step is customizable. Direct visualization of each step improves usability and provides a fast, iterative workflow. In case that a single step does not meet the user's desires, it can be easily repeated. Additionally, intermediate results are cached to allow reuse of finished pipeline steps. This system design guarantees the best trade-off between the partly contradictory requirements such as performance, usability, realism and flexibility.

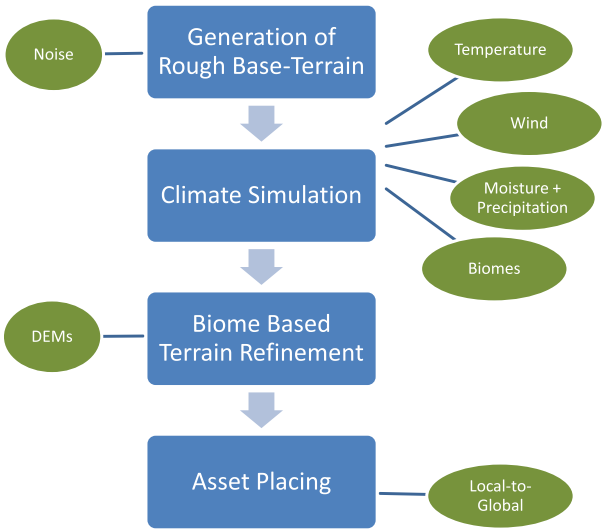

Fig. 1 The concept of our terrain-generation system as a pipeline model

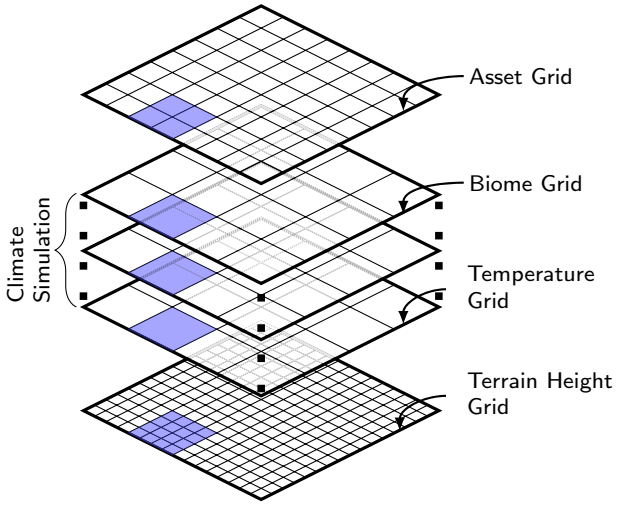

Fig. 2 The used data structures as a stack of regular grids. The layers can have different resolutions

Figure 1 illustrates the individual four steps of our sequential pipeline. The idea is to generate a coarse base terrain using noise functions first, which gets refined with biome-specific details later. To compute realistic biome distributions, we implemented a multiple-step climate simulation, which is carefully simplified to meet the performance requirements while maintaining good results. To add biome-specific terrain details, we chose an example-based approach where DEM data are combined with the previously generated base terrain. Finally, it is possible to generate asset distributions following a rule-based local-to-global model.

The advantage of this approach is that we can use the different PTG styles in the individual pipeline steps and concatenate them in such a way that brings out the respective strength, which results in a better trade-off between the requirements. The synthetic noise functions are able to quickly generate a general terrain and are highly adaptable. The biome distribution is then computed using our physically based climate simulation resulting in realistic looking results while being easily adjustable by transparent parameters. Highly realistic biome-specific terrain features and details finally are quickly added by overlaying DEM images, which is an example-based approach. The individual steps of our pipeline and the chosen methods are described in more detail later.

For compatibility reasons with external applications, e.g., modeling tools or 3D rendering engines, we decided to represent our terrains by heightmaps instead of voxels. Moreover, we use different resolutions for the individual steps of our pipeline, for example, the climate-simulation requires a less detailed grid (see Fig. 2). The final heightmap can be exported as a set of tiles in a standard file format (grayscale images). It can also be used directly in the Unreal Engine 4 (UE) which enables us to use build-in techniques like level of detail (LOD), instancing and level streaming. In the following, we will detail the steps of our pipeline.

3.1 Base terrain

To generate the base terrain, we decided to employ synthetic PTG methods, specifically, noise functions. In this first step, we only generate a rough terrain and such methods offer the most flexibility and widest range of possible terrains while also being very fast. Moreover, they are not limited in size or resolution. Physically and example-based methods would be more restrictive, e.g., have more constraints between parameters or need specific example images, and the potential benefits of greater realism and more details are not relevant as we refine the terrain in later steps. The drawback of noise functions, the need for tedious fine-tuning to get realistic looking results, does not apply because only the high-level terrain has to be generated.

We create the rough terrain by relying on common noise functions, more precisely, multiple octaves of simplex noise (using [16]) as this is well suited to generate a general fractal terrain. This method is fast, scalable, not too complex regarding usability and sufficient as a coherent, coarse basis. The noise parameters, as well as the number of octaves, can be set by the user. However, replacing or adding other noise functions for more diverse base terrains would be an easy modification. A user definable threshold marks the sea level to distinguish between land and water bodies (see Fig. 3a).

3.2 Climate simulation

For the next pipeline step, the computation of the biome distribution, we use a physics-based approach in contrast to

other fully synthetic methods that often rely on noise. We have developed a climate simulation which allows the generation of realistic, or at least plausible, distributions with a couple of easy-to-understand parameters. By comparison, noise functions would, in our mind, entail more fine-tuning or result in less realistic terrains and sketch-based methods would require more manual work which we want to avoid. However, sophisticated simulations are more computationally expensive, which is why we disregard some effects to simplify the system and focus on reasonable approximations. The goal of our climate simulation is to add physically plausible realism to the terrain while still being moderately fast to compute.

Following our pipeline-based design, the climate simulation is composed of multiple, sequential steps by itself, namely temperature, wind and precipitation computation, and lastly the biome classification. In detail, our climate simulation works as follows:

- The first step in our climate simulation is the temperature computation. We provide two different interpolation methods: a bilinear interpolation and a sine-based alternative. The former provides more flexibility for the user, while the latter is more suited to model one-dimensional gradients resembling the behavior observable on the earth between the equator and the poles. Both modes account for a height-based temperature falloff to simulate the temperature decline which occurs with increasing height, and are easily adjustable with a few parameters.
- The next step is the simulation of the prevailing wind to distribute the later generated moisture over the terrain. In order to keep the performance reasonable high, we use an iterative approach to calculate the wind directions instead of applying a computationally expensive fluid dynamics solver. Our method is a simplified version of the semi-Lagrangian scheme [21]. We dropped the diffusion process and the pressure calculations as we handle these separately in a later pipeline step. Therefore, we only consider external forces and self-advection to simulate the wind and compute its directions in a vector field. For these two components, we developed a less computational expensive algorithm.
  The basic idea is to specify initial values for the four corners which act as the external forces. An iterative approach distributes the wind directions on a vector field: in each iteration, the new wind direction for each cell is computed by combining it with its adjacent cell in the wind direction and adding a little random deviation to simulate micro disturbances. Finally, we additionally consider the closest corner to model the persistence of the external forces. This delivers a plausible smoothing or cancellation behavior along the dynamically moving fringes between the main wind currents.
- In the third step, we use the wind and temperature data to compute a precipitation distribution for the terrain. Again, we decided to use an iterative simulation-based approach. Basically, cells marked as water represent moisture sources. The evaporation is modeled as a temperature-dependent function; in fact, it can be chosen between an exponential and a linear version. The wind currents are responsible for distributing the moisture. Most of the moisture gets transported to the neighboring cell in the direction of the wind, but some shares also are transferred to the two cells adjacent to the neighbor and source. The actual distribution depends on the wind's direction and the previous moisture amount of all affected cells. With this algorithm, it is possible to model some form of dispersion and equalization. The amount of precipitation occurring depends on the local moisture and temperature and is modeled as a two-step process. First, the precipitation arising during moisture transport is computed. By using the previously computed temperature values and calculating the difference between the target and source, we can also simulate natural phenomenons like rain shadows. Finally, additional precipitation is computed for moisture-holding cells to simulate other, more local causes. Again, exponential or linear formulas can be used. Although we provide reasonable standard values, the system can be modified by a set of user parameters steering the formulas and therefore the results.
- The last step of the climate simulation is the classification of the resulting biomes according to the computed properties, in particular, the temperature and precipitation. For this purpose, we use a slightly modified and discretized Whittaker diagram [22] as a lookup table. For each pair of temperature and precipitation values, a specific biome ID is assigned according to the lookup table. In principle, other classification systems are possible as the lookup table can be freely changed or replaced by the user.

Figure 3 shows the results of the temperature (b), wind (c), moisture (d), precipitation (e) and terrain refinement (f) steps.

3.3 Terrain refinement

To complete the terrain generation, the rough base terrain is enriched with more realistic details based on the biome distribution provided by the climate simulation. We decided to use an example-based approach, in particular, DEMs, to obtain realistic biome-specific terrain details because of the vast pool of freely available DEM data which can be exploited. The DEMs serve as examples which can be blended onto the base terrain. The advantage is that the DEMs inherently provide realistic biome-specific terrain features and details.

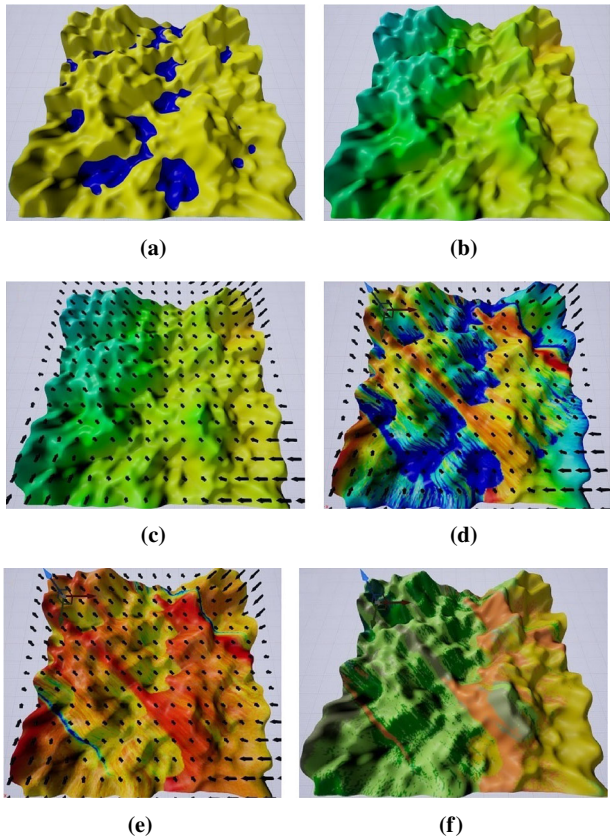

Fig. 3 Intermediate results of the first five pipeline steps. **a**: Base terrain with land in yellow and water in blue. **b**: Temperature; warmer colors denoting higher values, e.g., cold regions are blue. **c**: Wind depicted by the arrow orientations. **d**: Moisture; colder colors denoting higher values, e.g., high moisture values in blue regions. **e**: Precipitation; colder colors again denoting higher values. **f**: Biomes; the colors denote different biomes, e.g., orange depicts a hot desert, and light green depicts grassland

To get such realistic details, other methods would need a lot more user tuning and manual work, e.g., crafting specific multi-layered noise functions for each biome type, or complex computations in case of physically based methods.

Another aspect which has to be considered for multi-biome terrains is, that especially organic, natural looking biome transitions are essential. Therefore, we further customize the previously computed biome borders. The basic idea is to initially use user-adjustable, simplex-based fractal noise to distort the borders at a more granular level. For this purpose, we allocate a higher-resolution biome grid. Compared to more sophisticated techniques from the field of texture synthesis, this is a simple and fast-to-compute method which guarantees a result with consistent quality. Depending on the input data, other methods may occasionally result in technically correct but visually unsatisfactory results like straight biome borders (e.g., graph cut).

In a second step, we compute a biome-based DEM weighting using a convolution kernel to blend the adjacent biomes and their corresponding DEMs: each DEM weight equals the area of the corresponding biome inside the kernel boundaries proportional to the whole kernel region. The strength of the resulting blend and the required computation time depend on the size of the kernel, which can be set by the user. The final DEM value can be easily calculated as a weighted sum over the occurring DEMs. For simplicity, we assume a one-to-one relationship between the DEM texels and the terrain heightmap. Finally, we combine the generated biome-specific detail layer with the base terrain by using a weighted sum of the two heightmaps.

3.4 Asset placement

In the final step of our pipeline, we populate the biomes by placing assets. We have developed an iterative, rule-based local-to-global model, that, in contrast to global-to-local models, enables the creation of emergent distributions. Additional advantages are that the model can easily be modified or extended by further constraints and the individual assets, through the defined rules, inherently consider the biome transitions. We also considered using a global-to-local model in combination with real plant distribution data, but such data are hardly available for all kinds of biomes.

Our system is designed to use pre-modeled meshes, which allows for arbitrary generation methods to be used. However, the mesh generation itself, in a modeling sense, is not part of this work. We provide a basic database of pre-defined assets that can be easily extended by the user. Each asset is associated with a set of properties, e.g., clustering probability, shadow tolerance or repelling distance. The placement is done iteratively via the dart-throwing principle where a random position is sampled and checked for the assets constraints. Our sampling approach is generally based on Poisson-disk sampling, where all the points are guaranteed to maintain minimal distances between each other. However, we extended this basic approach to cover also more complex multi-object distributions with bilateral constraints. Yet our approach is very flexible through the easy-to-understand parameters which steer the placement. We divide the assets into a few main classes, e.g. organic- and inorganic, with corresponding relevant parameters, which helps to improve the usability. Additionally, assets are partitioned into size categories which are processed iteratively such that smaller assets consider the previously placed bigger ones. With this technique, we achieve more plausible mixed distributions and environments. Generally, depending on the parameters, it is possible to model clustered, random or uniform distribution and anything in between.

As seasons have a significant influence on the terrain cover's visual appearance, each asset can be associated with up to four different meshes representing its seasonal look. The meshes then are swapped automatically according to the

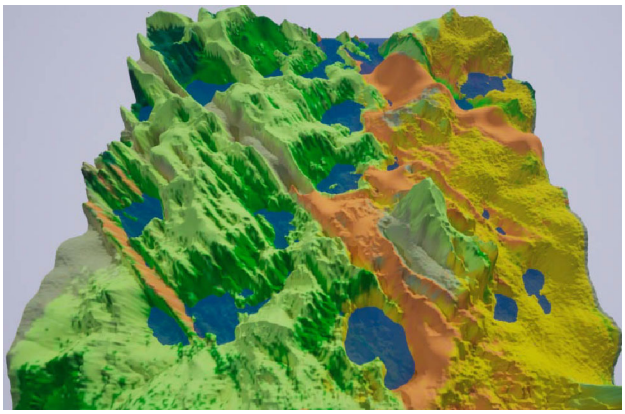

Fig. 4 A final terrain of our PTG system rendered in the Unreal Engine. The different surface characteristics caused by the distribution of multiple biomes can be seen easily. Each biome is also depicted by a different color

current season, which can be changed in real-time. Additionally, the Unreal Engine 4 provides instancing which improves the rendering performance, and a LOD system for dynamic switching between the placed assets' detail levels.

4 Results

We have implemented our terrain generation system directly in the Unreal Engine 4.20 using mainly C++ programming. It is directly accessible via the Unreal Editor which makes it very convenient for content creators.

We are not aware of quantitative measures to evaluate the quality of automatically generated terrains or biome distributions. Hence, we decided to provide mainly a qualitative evaluation of our system. We, e.g., show the influence of several of the most important parameters in the terrain generation pipeline. Additionally, we present measurements of the performance of each pipeline step in various configurations. Moreover, we are not aware of any directly comparable scientific work with the same focus—fast procedural multi-biome terrain.

First, we investigate the plausibility of the generated terrains. Figure 4 shows an example of terrain generated with our approach.

Different biomes can be easily distinguished by different surface characteristics. The distribution of the biomes is a result of correctly simulated natural phenomena like the occurrences of rain shadows. The easy accessibility of the adequate and meaningful parameters from the editor makes it easy to generate a vast variety of different terrains and asset distributions. For instance, Fig. 5a, b and c shows the influence of the temperature, the wind direction and the used base noise, while all other parameters remained constant: the resulting terrains look very different; however, they are still

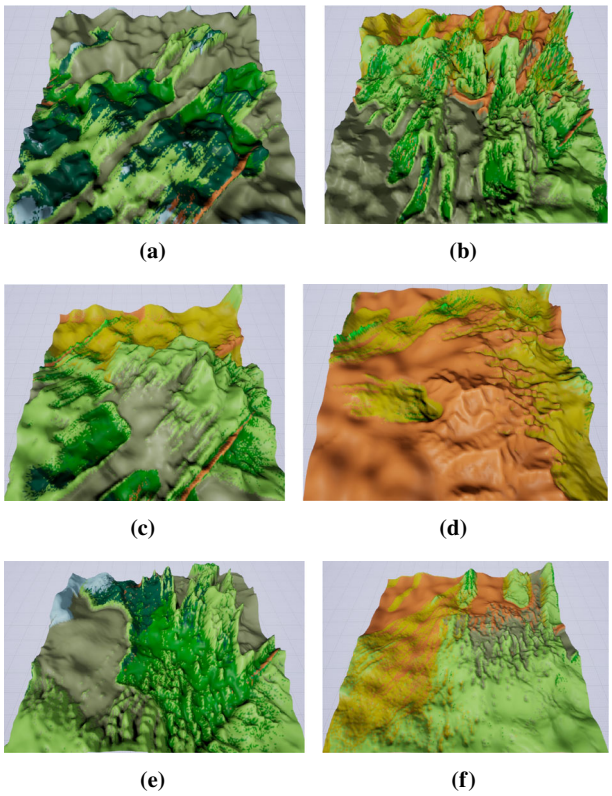

Fig. 5 A set of different terrains generated by our system, visualized on a proxy mesh with biomes depicted by different colors. In contrast to Fig. 3f, we changed the temperature **a**, the wind directions **b**, the base noise **c**, numerous parameters at once **d**, **e**, **f**. Especially the last three examples show how a wide range of different terrains with wildly varying surface characteristics and biome distributions can be generated. This is a result of the interaction of parameters like the base noise, temperature, wind direction and chosen DEMs

plausible. Figure 5d, e and f shows a variety of terrains if we change multiple parameters. The interaction of the different parameters like base noise, temperature, wind direction and chosen DEMs is responsible for the even more wide range of generated terrains. These example terrains show not only diverse occurring biomes and biome distributions but also drastically varying surface characteristics and general patterns.

Placing hundreds of thousands of assets with cross-class dependencies to populate a huge terrain is easy with our system (see Fig. 6, where the generated terrain is populated with around 200,000 assets).

By changing the asset-placement parameters, the user can directly influence the distribution and density of the assets while maintaining high realism (see Fig. 7a, b and c).

The real-time-adjustable season of the year has a significant impact on the landscape's appearance, as can be seen in Fig. 8 and enables the visualization of an even wider range of environments and adds an additional layer of realism to the user if switched dynamically.

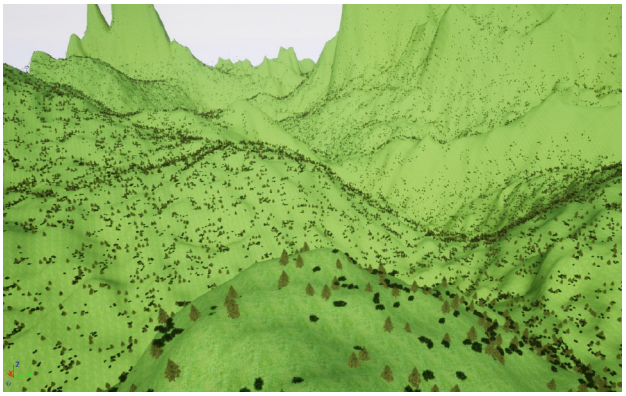

Fig. 6 The same final terrain as in Fig. 4 with procedurally distributed assets using our asset placement component. Around 200,000 instances were spawned in total

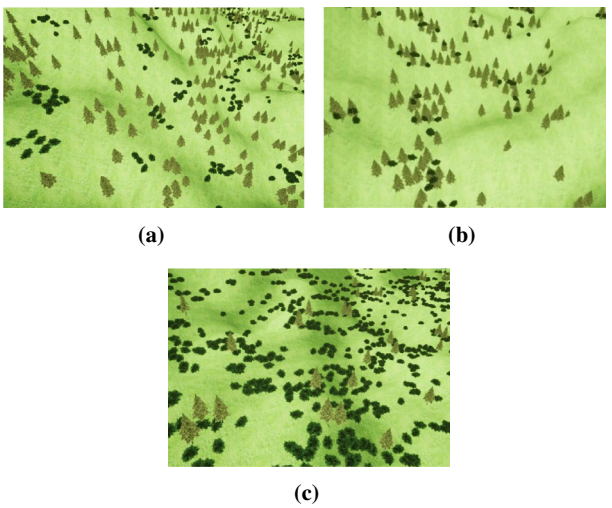

Fig. 7 Three different asset distributions generated by our system. **a**: Tight clusters of shrubs in open spaces between trees. **b**: Shrubs growing exclusively in shadowed areas within dense tree clusters. **c**: Dense, clumped distribution of shrubs around loosely grouped trees

Fig. 8 A terrain with its cover at different seasons: summer on the left **a**, winter on the right **b**. Seasons and the corresponding meshes can be automatically switched by our system

Additionally, we have investigated the performance of our terrain generator. All timings were done on a Windows 10 PC with Intel Core i7-7800X processor, NVIDIA GeForce RTX 2070 graphics card and 16 GB system memory. The running time of the pipeline depends mainly on the resolution of the particular grids. The expected running time is $\mathcal{O}(n^3)$ with n

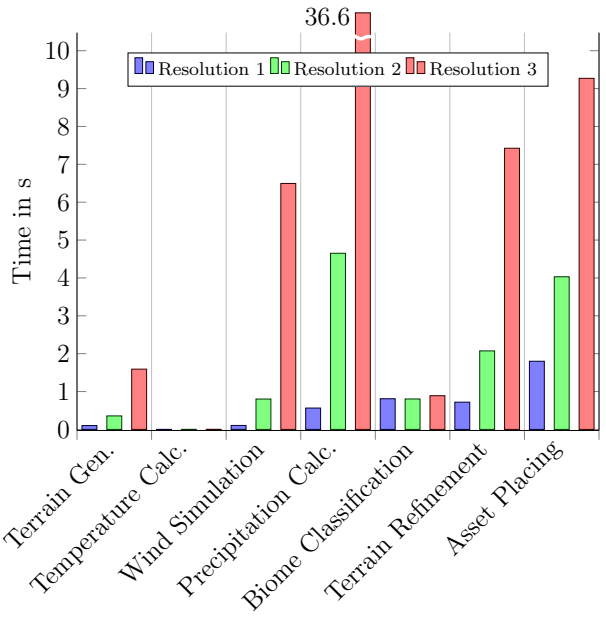

Fig. 9 Computation times of the different pipeline steps: For the first pipeline step, the cell resolutions (per axis) were set to 1024, 2048 and 4096, respectively. For the distribution of assets, the resolutions were set to 30, 60 and 120 assets per cell, while for all other steps, the resolutions are 128, 256 and 512 cells per axis

denoting the number of cells per axis. This is dominated by the simulation of wind and precipitation with an expected running time of $\mathcal{O}(n^3)$. The other steps are expected to have a running time of $\mathcal{O}(n^2)$. However, the biome classification is actually bound by the constant DEM loading times and the asset placement is nearly linear in the number of assets. The theoretical memory consumption is $\mathcal{O}(n^2)$. In practice, it is dominated by the number of asset instances.

Figure 9 shows the time needed for calculating the individual pipeline steps with respect to the grid resolutions. All times were measured by performing several test runs using the Unreal Engine profiling tool and taking the median time over all test runs.

The computation time of the individual pipeline steps differs significantly, from a few milliseconds for the temperature calculation up to 36.6 seconds for the precipitation calculation in the most expensive configuration. However, the computations last in the majority of cases less than ten seconds. The overall most time-consuming steps are the precipitation calculation and terrain refinement, as expected, but also the asset placement. The main factor affecting the needed computation time is the respective grid resolution, but for some pipeline steps, additional parameters also have a great influence. For example, during the refinement of the terrain, the blending part takes the most amount of time, and therefore, the adjustable blending kernel size has a significant impact on the performance of this step. Figure 10 shows how different kernel sizes affect the performance for the terrain refinement.

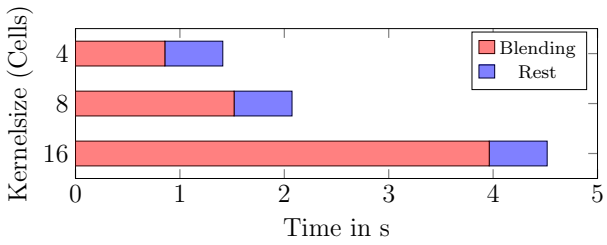

Fig. 10 Computation time of the terrain refinement step with different sizes for the blending kernel (size per axis). The resolution was fixed to 2048 cells per axis. The middle bar corresponds to the terrain-refinement result with resolution two in Fig. 9

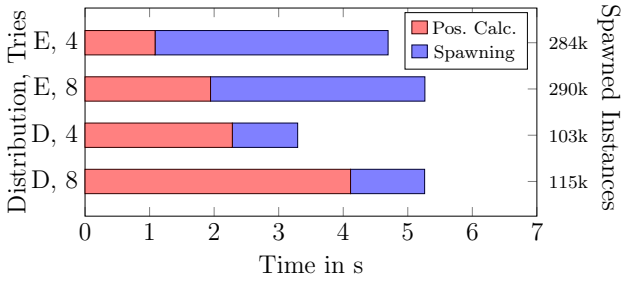

Fig. 11 Calculation time of the asset placement step, partitioned in the phases of position calculation and actual spawning in the UE, with different distribution rules and a varying amount of positioning tries per instance. D stands for a difficult to fulfill rule-set, E for an easy one. Also the resulting number of spawned assets is stated; the optimal amount would have been 300k, where k stands for thousand

In the case of distributing the assets, it is noteworthy that the computation time is highly dependent on the combination of the strictness of the placement rules and the maximum number of iterations per individual placement. Our tests show that roughly 100-300 assets can be placed in a couple of seconds. The time to spawn the assets in Unreal is included in the placement time and, in general, takes up a considerable amount of it, which is a bit surprising. Figure 11 illustrates the interaction between these parameters and the composition between the time needed for the positioning and the instance spawning. A higher number of positional tries lead to both, more instances being actually placed and a higher computational time, as expected. However, with stricter placement rules which are harder to fulfill, i.e. increasing the minimum distance between instances, the overall time can actually decrease even if the time needed for calculating the positions increases relatively. This is due to fewer valid positions being found and therefore fewer instances being spawned which is a fairly costly process.

However, even for large grid sizes, our terrain generation requires less than a minute in almost all cases. These fairly low computation times meet our expectations and enable quick iterations. Implementing multi-threading could improve the performance even further as the most algorithms are prone to parallelization.

5 Conclusion and future work

We have presented a pipeline-based system for procedurally generating multi-biome landscapes. Our pipeline model is easy to use and flexible on both, local and global scale. Our system can help level designers and other users with generating and quickly iterating over vast and yet visually plausible multi-biome terrains. This process includes the automatic but still user adjustable population of the terrain with huge amounts of pre-defined assets following complex distributions. Utilizing a carefully simplified climate simulation was a central element in the success. It is not only crucial for creating the biomes themselves and their realistic distribution but is also the basis for other landscape aspects, e.g., our DEM-based terrain refinement and also the asset placement rely on the specific biomes. Our results demonstrated that the generation is reasonably fast, while the terrains are visually plausible.

The modular pipeline approach is an excellent base for future work. There are many possibilities to extend our system or improve existing parts even further. The most promising additions, in our opinion, would be to implement a simulation step for river and erosion generation and the introduction of different geological layers and soil types. Also encasing the system in a meta-iteration to alternate between biome distribution and terrain generation could be interesting for producing results which are even more realistic. Regarding improvements, multi-threading and a more complex wind simulation would be the most important ones. Finally, we see much potential in investigating the use of neural networks, e.g., style transfer or DCGANs, for terrain generation, specifically, generating DEMs and combining them with the base terrain.

Acknowledgements Open Access funding provided by Projekt DEAL.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest. The authors received no specific funding for this work.

References

1. Amato, A.: Procedural content generation in the game industry. In: Oliver, K., Newton, L. (eds.) Game Dynamics, pp. 15–25. Springer (2017)
2. Beckham, C., Pal, C.: A step towards procedural terrain generation with gans. arXiv:1707.03383 (2017)
3. Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., Prusinkiewicz, P.: 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)
4. Galin, E., Guérin, E., Peytavie, A., Cordonnier, G., Cani, M.P., Benes, B., Gain, J.: A review of digital terrain modeling. In: Computer Graphics Forum (2019)
5. Gamito, M.N., Maddock, S.C.: Accurate multi-dimensional poisson-disk sampling. ACM Trans. Graph. (TOG) **29**(1), 8:1–8:19 (2009)
6. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2414–2423 (2016)
7. Génevaux, J.D., Galin, É., Guérin, E., Peytavie, A., Benes, B.: Terrain generation using procedural models based on hydrology. ACM Trans. Graph. (TOG) **32**(4), 143 (2013)
8. Guérin, E., Digne, J., Galin, E., Peytavie, A., Wolf, C., Benes, B., Martinez, B.: Interactive example-based terrain authoring with conditional generative adversarial networks. ACM Trans. Graph. **36**(6), 228:1–228:13 (2017)
9. Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A.: Procedural content generation for games: a survey. ACM Trans. Multimed. Comput. Commun. Appl. (TOMM) **9**(1), 1 (2013)
10. Hyttinen, T.: Terrain synthesis using noise. Master's thesis, University of Tampere (2017)
11. Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A., Teschner, M.: Sph fluids in computer graphics. In: Lefebvre, S., Spagnuolo, M. (Eds.) Eurographics 2014—State of the Art Reports. The Eurographics Association (2014)
12. Jákó, B.: Fast hydraulic and thermal erosion on gpu. In: Eurographics 2011—Short Papers (2011)
13. Jones, T.R.: Efficient generation of poisson-disk sampling patterns. J. Graph. Tools **11**(2), 27–36 (2006)
14. Lane, B., Prusinkiewicz, P., et al.: Generating spatial distributions for multilevel models of plant communities. In: Graphics Interface 2002 Conference, pp. 69–80. Citeseer (2002)
15. Musgrave, F.K., Kolb, C.E., Mace, R.S.: The synthesis and rendering of eroded fractal terrains. In: ACM Siggraph Computer Graphics, vol. 23(2), pp. 41–50. ACM (1989)
16. Peck, J.: Fastnoise simd (2016). Software library for noise, https://github.com/Auburns/FastNoiseSIMD. Last accessed (23 May 2019)
17. Perlin, K.: An image synthesizer. SIGGRAPH Comput. Graph. **19**(3), 287–296 (1985)
18. Perlin, K.: Improving noise. In: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '02, pp. 681–682. ACM (2002)
19. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv:1511.06434 (2015)
20. Smelik, R.M., Tutenel, T., Bidarra, R., Benes, B.: A survey on procedural modelling for virtual worlds. In: Computer Graphics Forum, vol. 33(6), pp. 31–50. Wiley Online Library (2014)
21. Stam, J.: Real-time fluid dynamics for games. In: Proceedings of the Game Developer Conference (2003)
22. Whittaker, R.H.: Communities and ecosystems (1975)
23. Wulff-Jensen, A., Rant, N.N., Møller, T.N., Billeskov, J.A.: Deep convolutional generative adversarial network for procedural 3d landscape generation based on dem. In: 6th EAI International Conference on Arts and Technology, Interactivity & Game Creation, pp. 85–94 (2018)
24. Zhou, H., Sun, J., Turk, G., Rehg, J.M.: Terrain synthesis from digital elevation models. IEEE Trans. Vis. Comput. Graph. **13**(4), 834–848 (2007)

Roland Fischer received his M.Sc. degree in Computer Science in 2019 from the University of Bremen, Germany. He currently works as a research assistant and Ph.D. student at the Department for Computer Graphics and Virtual Reality at the University of Bremen. His research interests mainly include computer graphics, procedural generation, virtual reality and image processing.

Philipp Dittmann received his M.Sc. degree in Computer Science in 2016 at the University of Bremen, Germany. He currently works as a research assistant and Ph.D. student at the Department of Computer Graphics and Virtual Reality at the University of Bremen. His main research interests include geometrical computation, data compression and virtual reality.

René Weller received his Ph.D. in Computer Science in 2012 from the University of Bremen. Currently, he is a postdoctoral research assistant at the Computer Graphics Group at the University of Bremen and works in the field of computer graphics and virtual reality with specialization in geometric algorithms for collision detection, sphere packings and haptic rendering.

Gabriel Zachmann is professor for computer graphics, visual computing and virtual reality at University of Bremen, Germany, since 2012. Before that, he established and headed the computer graphics group at Clausthal University, Germany, where he was a professor with the computer science department since 2005. Prior to that, he was assistant professor with Prof. Reinhard Klein's computer graphics group at Bonn University, Germany, and head of the research group (Nachwuchsgruppe) for novel interaction methods in virtual prototyping, which was funded by the DFG within the Emmy-Noether program ("Aktionsplan Informatik"). In 2000, Dr. Zachmann received a Ph.D. in computer science, and in 1994 a Dipl.-Inform (M.Sc.), both from Darmstadt University. He worked on his Diploma thesis during a half-year visit to the National Center for Supercomputing Applications (NCSA) in Urbana-Champaign, Illinois. He began his studies of computer science at Karlsruhe University.