Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Benjamin Benjaminsen

Procedural terrain modelling with geometric multi-attribute features

Master's thesis in Cybernetics and Robotics
Supervisor: Sverre Hendseth
December 2021

NTNU
Norwegian University of
Science and Technology

Benjamin Benjaminsen

Procedural terrain modelling with geometric multi-attribute features

Master's thesis in Cybernetics and Robotics
Supervisor: Sverre Hendseth
December 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Problem description

The generation of realistic landscapes is a hard and long sought after achievement in Procedural Content Genration (PCG).

While the use of PCG today is applied to the generation of a wide verity of content, the generation of landscapes and environments is possibly one of the most commonly well known due to its visible impact, and gameplay impact on several popular games.

The topic of terrain generation has also received a significant amount of attention from academic research over multiple decades, as well as there has been created numerous tools either with the capabilities of, or for the specialized purpose of authoring terrains with the help of procedural techniques.

Still after several decades of attention, efficient, detailed and realistic modelling of real world terrain is a hard problem where there is no clear universal technique that will satisfy all criteria for all cases. A usual approach to terrain generation is the use of determinant noise and simulation of natural processes, in many cases this is also combined with artistic modification to achieve desired results.

In this work the student shall approach the problem of procedural generation of terrains based on discrete terrain features instead of noise. The work is an extension of a previous work on the topic and is aimed at improving previously pointed out aspects of the previous work, and the student shall:

1. Explore existing research and methods relevant to solving the problems
2. Formulate and consider possible approaches for solving these problems
3. Make design and implementation for these approaches to solving the problems

Abstract

This work presents the designs and implementation created for further improving a method of generating digital terrains based on procedurally modelling key landscape features.

From the preceding work, a flexible framework was created for procedural generation driven by geometric skeletons made from a collection of individual shapes representing such key features of a landscape as mountains, rivers and lakes. The method however had issues with being excessively slow running and creating terrains with little detail and key features shaped like straight lines.

This work improves upon and extends the previous work, addressing the efficiency of creating the surface geometry from a skeleton, and creating more detail by the use of fractal shapes to describe features. The method is also extended to generate further values needed for a comprehensive description of a generated terrain beyond the elevation.

The implementation also makes it simple to experiment with most of the implemented functionality by graphical configuration in the WorldSynth terrain authoring software.

Sammendrag

Dette prosjektet tar for seg utforming og implementasjon for å forbedre en metode for generering av digitale landskap basert på prosedyrisk modellering av viktige trekk ved et landskap.

Fra det tidligere arbeidet, har det vært laget et fleksibelt rammeverk for prosedyrisk generering bassert på geometriske skjelett dannet fra en samling former som representerer viktige og formende trekk i et landskap, slik som fjell, elver og innsjøer. Men metoden viste seg å være urimelig treg å anvende, samt at resultatet var et landskap med lite detaljer og tydelige rette linjer.

Dette prosjektet forbedrer og bygger videre på det tidlige arbeidet, ved å forbedre effektiviteten til genereringen av overflategeometri fra en skjelett struktur, og innfører bruk av fraktale former for å skape mer detaljer. Arbeidet utvides også for å generere ytterligere verdier for en mer utfyllende beskrivelse av et terreng i tilleg til høyde.

Implementasjonen gjør det også enkelt å eksperimentere med det meste av den implementerte funksjonaliteten ved bruk av grafisk konfigurering i WorldSynth, et program spesialisert for tilvirkning av digitale landskap.

Table of Contents

List of Figures

Acronyms

DAG Directed Acyclic Graph. 16

GIS Geographic Information System. 3

LOD Level Of Detail. 8–10, 63, 78, 79, 81, 82

MBR Minimum Bounding Rectangle. 35, 39, 56

MPD MidPoint Displacement. 10–12, 16, 33, 60–63, 65–72, 74–77, 79, 81, 82, 84, 87–90

PCG Procedural Content Genration. i

PDS Poisson Disk Sampling. 16

SDF Signed Distance Field. 7, 8, 16, 22, 29, 39, 81

VDR View Dependent Rendering. 9, 79

WS WorldSynth. 12–16, 41, 42, 50–52, 65, 69, 70, 75, 94

Chapter 1

Introduction

1.1 Problem description

The problem taken on in this work is a continuation of my previous work, with the title "Feature-modeled procedural terrain generation with adaptive height evaluation" [1].

The focus of this continuation, is on accommodating further detail in the resulting output, while also improving the efficiency of the method. This is so that the method may be reasonable to use for generating a believable terrain during the run time of an interactive application, like for example a game, in which the generation of the final output needs to keep up with with a player traversing the generated terrain.

1.2 Motivation

Since environments are an important part of modern interactive media like for example games, the creation of such environments is an important topic. Due to the scale and complexity of many environment, it may quickly becomes unreasonably time consuming and expensive to have a human creating all parts of an environment, down to the finest details. Therefore the artistic creation of environments, whether they are imitations of natural environments, like lush landscapes, or environments created by human development, like dense modern cities, the process of digitally creating them will generally rely on reuse of assets or procedurally generated assets.

The procedural generation of realistic terrains is a hard problem to solve as there are many natural processes that together happen to shape a landscape over millions and billions of years. When terrain assets are created for a game or a film production and the terrain is expected to stay the same after release, it is possible for environment artists to manually design terrain with fine control, and use simulations of natural processes like erosion to greatly imitate real world landscapes that cohere to an artistic vision, but this approach is not possible if the environment is expected to instead be uniquely generated on a user's system as it is being explored.

For this fully procedural generation of environments, the problem of creating believable results becomes significantly harder as a computer will not have an artistic vision to its creation, and the simulation of natural processes quickly become demanding to apply at run time, when expecting to deliver a seamless experience to the user. And this is even before considering how results of these processes should be propagated to avoid conflicting boundary values if one is to generate infinite world put together by separately generated sections at different times.

One of the most recognized problems relating to generating realistic landscapes, especially when the landscape is large or even infinite, is the generation of rivers flowing in a consistent downwards direction.

Currently used approaches to procedural generation of terrains are often based on the usage of coherent noises like perlin, open simplex or worley. But while these noises are easy to use, and multiple octaves can produce results that have some similarities to terrains, the results lack the general structure to terrain that has been shaped by natural processes like erosion, that form continuous pathways for water to flow from higher elevations to lower elevations. Instead these methods instead tends to produce a series of local minimums where water would collect as lakes without any given drain.

Some times rivers are imitated by the use of noise, but such rivers may often be aptly named as long and narrow oceans that have infiltrated the landscape, or more simply very weird shaped lakes, as they remain flat and without direction, usually at a global elevation in all cases, and often do not even connect to another body of water, instead often ending abruptly if not even going in loops.

In my previous work I explored generating terrains based on geometric features, with deterministic evaluation of elevation at any point using weighted averages. The work showed the method applied to generate procedural landscapes that include terrain features like rivers, lakes and mountain ridges, but these show up as distinctly unnatural straight lines while nature generally shows fractal properties in its forms, and they are therefor not satisfactory for final use. Even more detrimental to actual usage is the evaluation time to produce the elevation from a skeleton layout, which grows with the total number of features in the skeleton layout instead of the density, causing scaling of the method to have a quadratic time complexity.

And lastly a terrain is more than just an elevated plane void of further context, a terrain is generally composed of different types of rocks, dirt, vegetation and other details. Much of such details also have ties to the key features of the landscape, and it is therefor also assumed to be reasonable to model these as a part of the features in the skeleton layout.

From these considerations, it is therefor desirable to further develop the method from the previous work into a more mature state, capable of efficiently creating more detailed terrains.

Chapter 2

Background

2.1 Spatial data structures

When an application needs to store and access large amounts of data, it is often desirable to access subsets of the data that meets specific condition or that is in some ways related or similar. The optimization of solving these kinds of problems for different applications are achieved trough the use of data structures and algorithms. The efficient organization and searchability of large amounts of data is an important topic in the field of database systems, where large amounts of data needs to be accessed, manipulated and queried quickly.

In relation to the topic of this work where features are in the form geometric primitives placed across a multidimensional space, more specifically a plane, there exists a lot of similarities to maps digital storage of map data, and there is also encountered some problems with similarities to problems that are relevant in Geographic Information System (GIS) applications like searching for data that overlaps with a point or multidimensional range.

In the pursuit of fast and efficiency storage and querying spatial-temporal data, significant research has gone into the development storage structures and database systems that can handle data with multidimensional indexes. It is therefore reasonable to look for relevant storage structure and practices with relevance from GIS application.

Some data structures for spatial data are:

- Spatial hashing
- Quadtree
- PM-quadtree
- MX-CIF quadtree
- KD-tree

- R-tree
- Bounding Volume Hierarchy
- AABB Tree

As with other data structures, these different structures have their own advantages and use cases they are better suited for, and some like the R-tree has a whole family of sub types that have been created over the years in pursuit of better versions for both generalized and specialized problems [2].

Out of these data structures, the R-tree, or more specifically the R* tree version of the structure, is one of the most generally utilized and flexible structures.

2.1.1 R-tree

The R-tree is an indexing structure for handling non-zero sized data in a multi-dimensional space [3]. The indexing is done through a height-balanced tree structure where each entry in a node of the tree has an associated bounding rectangle. For leaf-nodes, the entries in the node are the indexed data entries, and the bounding rectangles are the minimum bounding rectangles for these data entries. For non leaf nodes, the entries are other nodes, either leaf nodes or non leaf nodes, and the bounding rectangles are the minimum bounding rectangle enclosing all bounding rectangles of the child node entries.

The R-tree is a height-balanced structure with all leaf nodes at the same depth in the tree structure, and all indexed data entries are found in the leaf nodes. Each node in the tree has a capacity for M entries, and has a minimum fill rate of $M/2$ entries. This holds true for both leaf nodes and non-leaf nodes.

Querying

An R-tree may be queried for data that intersects a point or an are that defines a spatial query. When querying the R-tree, the search will starting from the root node, descend the tree along the entries of the root node and subsequent entries of these nodes, while the entries of the root node are not leaf nodes, when the bounding rectangle of the entry intersects with the query point or area, until all relevant nodes have been visited and data entries with intersecting bounding rectangles have been retrieved. This search of the tree structure may need to visit anything between no sub entries in a node, to all entries of the node depending on the entry bound intersecting with the spatial query, and there is thus no good way to guarantee a good worst case performance, as an area query intersecting all of the indexed entries of the R-tree is possible. In practical use where the query is of limited size relative to the coverage of the indexed entries with a uniform distribution, the R-tree offers an efficient search performance for the query.

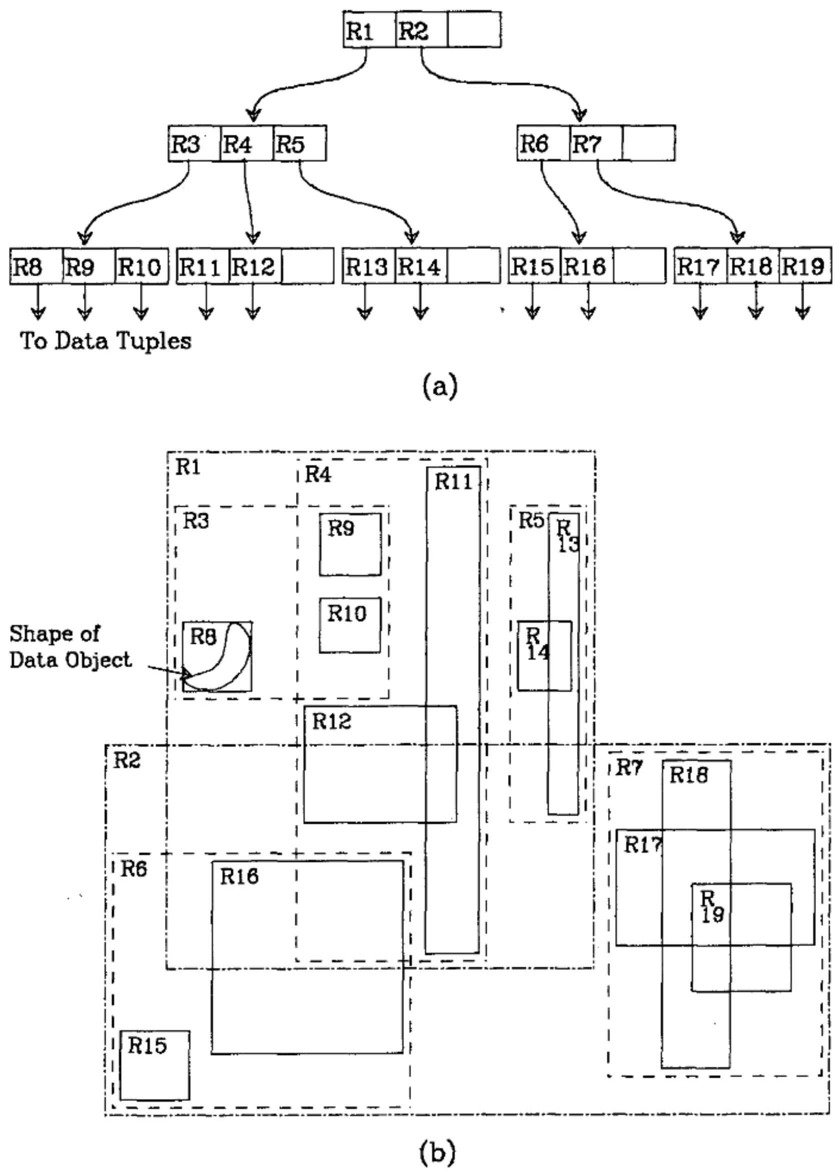


Figure 2.1: The structure of an R-tree, both tree structure and the layout of the bounding rectangles.

Source: R-trees: A dynamic index structure for spatial searching [3]

Insertion

When a new data entry is inserted into the R-tree, it is wanted to insert the new entry where the entry such that it is grouped together with entries the generally same area. Therefore the insertion of a new entry first chooses the best fit leaf node to insert the new entry into. This leaf node is chosen by traversing the tree down the most fit child node for each level of the tree, until a leaf node is found.

What is the most fit node for the entry is determined in an order of choosing a node with the least need of area enlargement of the bounding rectangle, with ties resolved by choosing the node with the smallest area.

When a leaf node has been selected for insertion, the new entry is inserted into the leaf node. If the insertion causes the leaf node to be overfilled by inserting the entry into a node already M entries such that there become $M + 1$ entries, the leaf node is split and the previous entries along with the new entry are distributed equally between two leaf nodes in such a way to minimize the overlap of these two nodes.

After the new entry is inserted, and any split done, the changes to the tree is propagated upwards the tree from the leaf node, adjusting the bounding rectangles of the node and propagating any node split upwards. If the node split is propagated to the root node such that the root node splits, the tree is grown taller.

Removal

When deleting an entry from the R-tree, the tree is searched for the leaf node containing the entry, and if it is found, it is removed. Trying to remove entries that are not indexed by the R-tree will not find any leaf node containing the entry, and therefor not modify the R-tree.

After an entry is removed from the a leaf node, the tree may possibly be condensed if this causes the leaf node to have too few entries. This causes the entries in the node to be relocated among other nodes by reinsertion to the parent node, this distributing the entries among the other leaf nodes, if the removal of the leaf node causes the parent node to have too few entries as well, the change may possibly propagate up to the root nod, in which case it is left with only one entry, the tree is shortened. The removal of the entry also causes the bounding rectangle to update for the parent node, and propagates up the tree.

R* tree

The R* tree is a variation of the R-tree optimized for making a tree structure with better properties related to covered and overlapping area of non-leaf nodes, as well as making more square nodes, thus creating smaller and more separated non-leaf nodes [4]. This improvement of the non-leaf nodes makes a tree structure more generally efficient to query, as there are fewer possible branching paths in the tree that needs to be traversed when performing a spatial query. The R* tree also only has a slightly higher implementation cost than the original R-tree.

The optimization is achieved by the means of tree changes to the insertion behaviour of the R-tree.

1) During insertion the traversal down the tree in search of a leaf node for insertion only takes takes into account the change and size of the node area, such that this is minimized in the original R-tree, the R* tree here improves the insertion by also a taking into account margin and overlap in this search for a leaf node to insert into.

2) The R* tree also changes the node splitting operation when the chosen leaf node is already full. The R* version for splitting is performed based on goodness values determined from area-, margin- and overlap-values.

3) As the order of insertion of entries into the R-tree influences the shape of the tree such that entries inserted early in the growth of the tree have introduces undesired rectangular shapes of non-leaf nodes which are less efficient than rectangular nodes. While a very local reorganization is performed during a node split, the R* tree also addresses this by reorganizing the tree on a larger scale by forced reinsertion. When a leaf node chosen during for insertion overflows with the addition of a new entry, a portion of the entries in the leaf node is attempted reinserted instead of splitting the node. The reinsertion of entries is only attempted once during an insertion, and if the reinsertion further causes leaf nodes to overflow, these subsequently overflowing nodes are split as normal.

2.2 Signed Distance Fields

Signed Distance Field (SDF) and its usage is based on the creation of functions that can determine the distance between some point and a shape, whether that shape is 2D or 3D. Since SDF is signed, the distance function can also determine whether the point is inside the shape to the edge or shell of the shape.

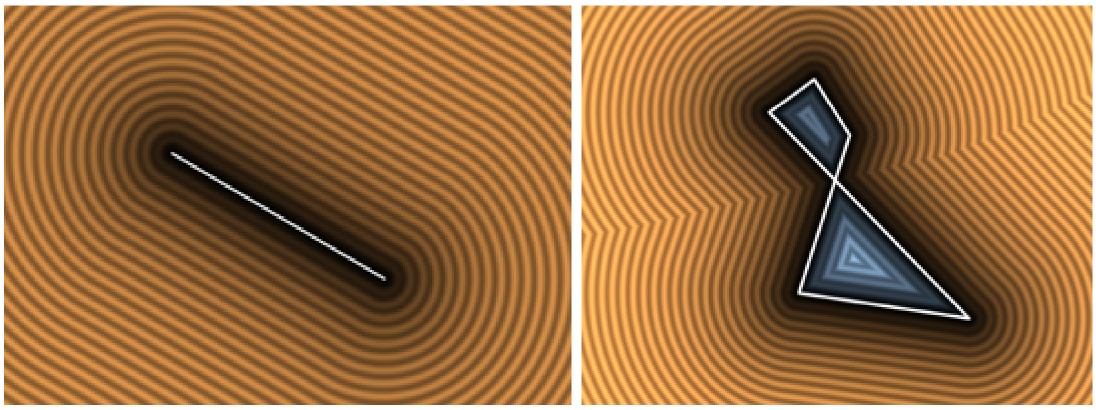


Figure 2.2: Illustration of the SDF of a line segment and a polygon.

Source: 2D distance functions [5]

The usage of distance functions has a use in rendering 3D graphics by a process of ray marching where surfaces are implicitly given by the distance functions, and this way of rendering has shown interesting properties with rendering of procedural and fractal surfaces [6] [7].

While a wide variety of shapes can be created as primitive shapes with SDF, more variations and complex shapes may also be created by additional alterations or combinations of such primitives [5] [8].

2.3 Level Of Detail

In the world of interactive 3D applications, such as games, 3D moidelling applications and others, there exists the issue of scenes consisting of 3D geometry becoming more complex than a computer is able to render within reasonable time for an such interactive applications [9]. This lag manifests itself in the form of reduced frame rate of the output picture, and if the complexity of the scene becomes too high for the hardware to render it within reasonable time, the user will experience this as an issue during use of the application.

A case where the amount of detail in a scene may easily become a problem, is when a scene becomes large with lots of details distributed all over the scene. In that case, there may be significant more detail present far away from a viewpoint than what can actually experienced in the rendered results, while the level of detail may be what is desired when the viewpoint is close, thus the details of geometry that is far away from the viewpoint is less important than the closer geometry. The rendering of all details thus makes sense when the details are close to the viewpoint, but when the details are far away, they do not contribute appreciably to the result and the geometry could reasonably be be less detailed for these less important geometries.

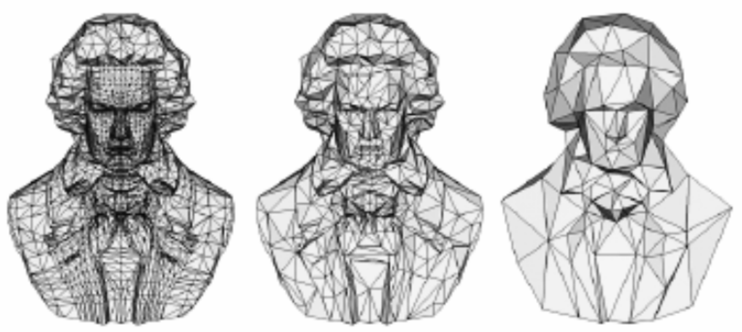


Figure 2.3: An example of a 3D model with three levels of Level Of Detail (LOD), from a fully detailed version to a lowest detail version used when the model is far away and the detail is of little importance.

Source: A Review on Level of Detail [9]

To address the problem described, where a high level of detail is wanted in a scene, while also having efficient rendering, games use LOD. LOD enables the same geometry in a scene to appear with differing amounts of detail depending on the distance the geometry is from the viewpoint, and thus ho much of the detail may be shown in the rendered results. This woks such that the geometry is less complex, with less details, and faster to render when it is further away from the viewpoint, and thus smaller in the rendered picture, while geometry closer to the viewpoint has more complexity ans higher detail that may be appreciated only when the viewpoint is close to the geometry. Moreover, in addition to just have the geometry in the scene be static, the geometry is changed as the viewpoint moves closer or further away from the given geometry, such that complex geometry becomes less detailed as the extra detail is not important, and visa versa. The geometry may even change between several levels of complexity for different distances to the viewpoint such that the amount of detail in a scene is approximately the same as what can actually be seen in the rendered result.

For LOD to work as desired, the geometry desired to be included in a scene need to be prepared for inclusion with the use of LOD. This is usually done by creating several version the geometry with varying amount of detail complexity, how many variation will depend and the variations may be prepared through various means of tools and using automated or manual processes, or a combination. These prepared versions of the geometry then may be used at the run time of the application where they will be changed between as described.

2.3.1 View Dependent Rendering and Virtualized Geometry

The traditional way of going about LOD, as described above, is to change the geometry of objects in a scene between variation of the same object with differing amounts of detail in the geometry, depending on the distance from the viewpoint. This though requires the different version of the geometry to be prepared in advance for the same object. And the geometry for an object is also completely replaced for a detail level.

Since the general goal with applying LOD is to render the geometry with a detail level fitting for how much of the detail may be displayed, and the geometry of an an object is traditionally completely replaced for different levels of detail, there comes an issue when the viewpoint becomes such that it sees a relatively large range between the closes details and the furthest details seen of the object, as the closes part of the object may need significantly more detail than the furthest part of the object would need. With the object having a detail level appropriate for the closeup view, the furthest viewable part of the same object may have significantly more detail than is necessary in such cases, and it would therefor be desirable to have differing levels of detail across the object.

With the introduction of the Unreal Engine 5, Epic Games introduced their virtualized geometry system, which they call Nanite [10]. Nanite is an implemented solution to view dependent LOD, where the density of detail across an object is adapted to the detail that can be viewed in a rendered frame, and thus the detail density of the mesh may differ across the object. This build on, and is in large part an implementation of View Dependent Rendering (VDR) for polygon meshes [11] [12].

VDR as a research topic is not especially new, and stems back several decades

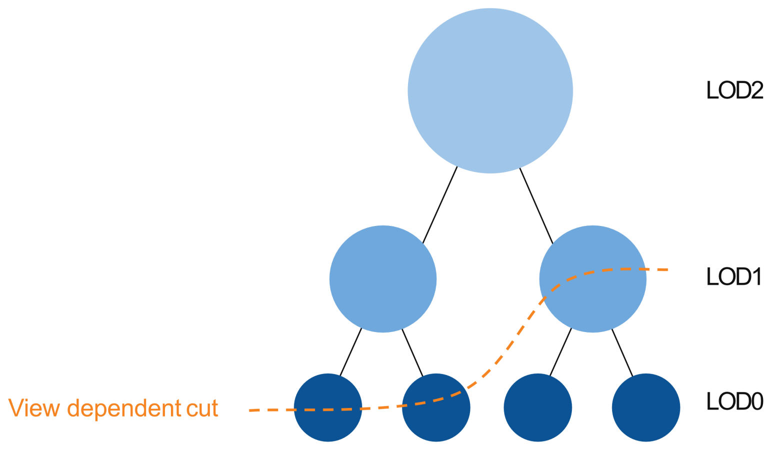


Figure 2.4: LOD with view dependent cutting in the detail levels used for different parts of a mesh allows a model to feature different detail densities that are individually chosen based on how much detail can be seen by the camera.

Source: Nanite - A Deep Dive [10]

already, with especially use in rendering of parametric surfaces, where it has been used for adaptive tessellation of the NURBS surface [13] [14].

2.4 Midpoint Displacement

MidPoint Displacement (MPD) is a method for producing natural irregular objects such as stones, terrain, trees or other such that are not characterizes by smooth macroscopic features [15]. Rather such objects are of a fractal nature, where the details of the object shrinks, but still remains complex even as one looks closer.

The method of MPD works by dividing primitive constructs such as lines, triangles or squares, into multiple new and similar primitives contained within the previous primitive. A random offset is then added to the newly created descriptive part of the geometry resulting from the division. This operation is illustrated in Figure 2.5.

As an increasing number of iterations of this division are performed, the amount of added displacement is lessened with each iteration, such that the random displacement produces new detail with a detail level relative to the scale of the primitive being divided. Results from such an iterative application of MPD is shown in Figure 2.6.

This is useful for producing fractal outlines and surfaces such as coastlines and terrain surfaces. A method of producing terrain surfaces by the use of MPD, is to iterative

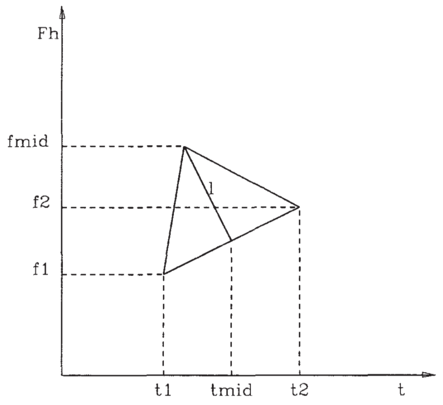


Figure 2.5: MPD performed on line between (t1, f1) and (t2, f2). The new point (tmin, fmid) is displaced a randomized distance *l* in normal direction from the midpoint of the line [15].

Source: Computer Rendering of Stochastic Models [15]

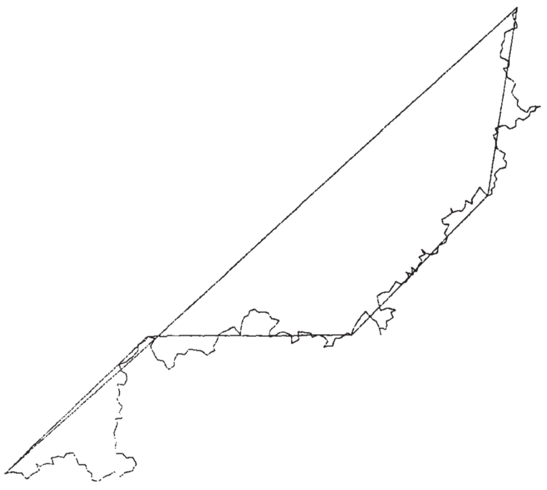


Figure 2.6: Fractal polyline with 2, 5, and 257 points created by multiple iterations of midpoint displacement [15].

Source: Computer Rendering of Stochastic Models [15]

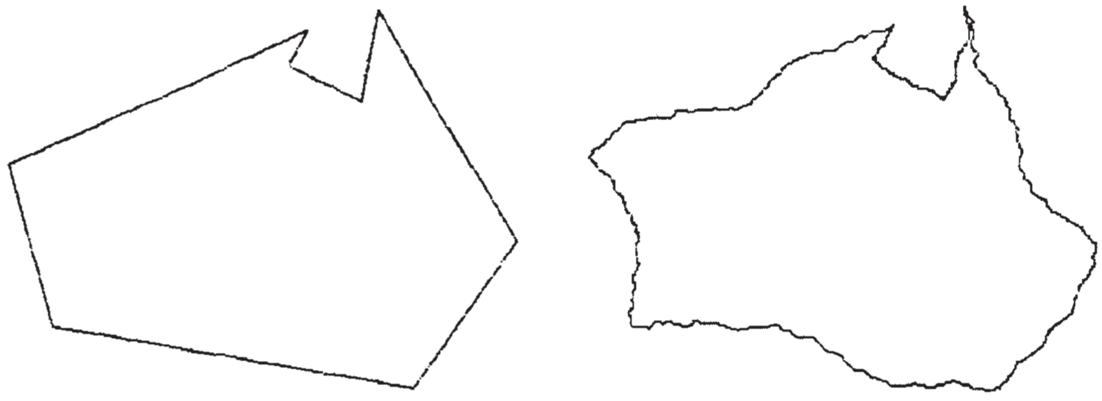


Figure 2.7: A low amount of samples along a digitized model of the Australian coastline with procedural details created by the use of MPD.

Source: Computer Rendering of Stochastic Models [15]

2.5 WorldSynth

The following section introducing WorldSynth, is an excerpt taken from: Feature-modeled procedural terrain generation with adaptive height evaluation [1].

In this work of this report I will be using WorldSynth as an underlying framework for the implementation of my generator [16].

WorldSynth (WS) is an open source project that functions as a framework, engine and editor, for procedural generation of environments. WS provides an open and flexible platform inspired by industry standard tools for terrain creation, but while such other existing tools have their primary focus on providing tools used to create large realistic terrains during the development process of a game or for other media, WS aims at expanding the application of the existing workflows from these tools, to also cover usage in-engine for generation of terrains in the run time of games, as well as standalone use by environment artists creating premade assets, and it also applies the existing workflows to handling a wider variety of environment data like volumetric data.

While originally created for the purpose of the creation of a player modification to the game Minecraft [17], to introduce a highly configurable terrain generator in replacements of the game's own included terrain generator, WS itself is game-agnostic, and any platform specific compatibility necessities for any game are provided through the use of addons to WS.

WS is a project created and maintained by the author of this report, and has been in development for several years (since middle of 2017) before the work covered by this report was started.

The work covered in this report does not, and is not intended to cover any aspects of core developments to WS, but there is a possible that aspects of the work covered in this report may be a cause and reason for some minor changes to WS during and after the duration of this project, whether such relations are conscious or unconscious.

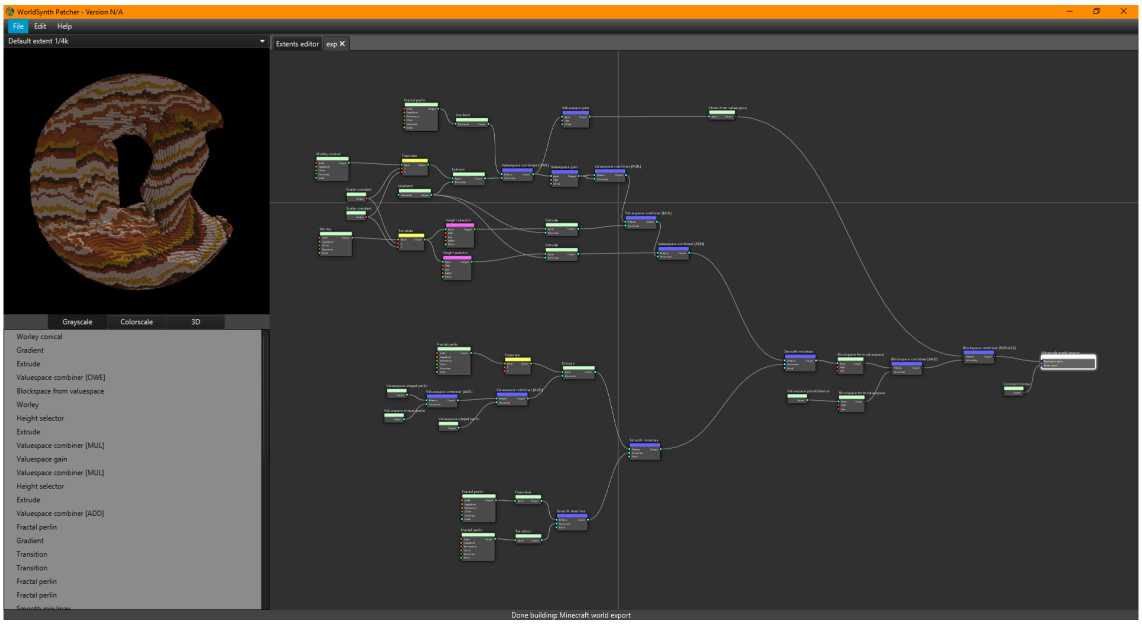


Figure 2.8: Screenshot of the WorldSynth patch editor, here shown with an example file for a volumetric voxel terrain.

This work has during its progression been developed using development versions of WS, so some of the screenshots taken within WS at different times through out the report may have some visual differences as a result of development on WS going on in parallel to this work.

2.5.1 An overview of WorldSynth

WS is created to enable a modular and graphical, data flow driven approach to terrain generation. In Figure 2.8, a screenshot of the WorldSynth patch editor, called "Patcher" is shown. A "path" in WS is an arbitrary collection of modules with configured parameters and that are connected together in a graph. This can be seen in Figure 2.8 as the block diagram view to the right. Each block in this view is called a "module", and represents an algorithm that produces an output, possibly according to one or more inputs that may or may not be optional. The modules can be connected together such that the output from one module becomes the input to one or more subsequent modules in a directed acyclic graph, thus allowing for combination of multiple simple discrete algorithms into sequentially complex combinations in a graphical manner. The graph direction is read with direction of operation from left to right, where the connections to the left of a module produces inputs to the module, and connections to the right of a module consumes the outputs from the module.

Each of the lines in the graph view represents the connections and flow of produced output result from a module, to the input of subsequent modules. The data transferred between modules can for example be information about a terrain's height at different locations in the form of a heightmap, the color of the terrain in form of a colormap, or the types of ecosystems in the form of a biomemap, and WS also extends to volumetric types.

The output results from a module can be viewed by selecting the module, and an appropriate preview of the results is rendered in the preview render to the top left of the editor as shown in Figure 2.8.

2.5.2 Data types

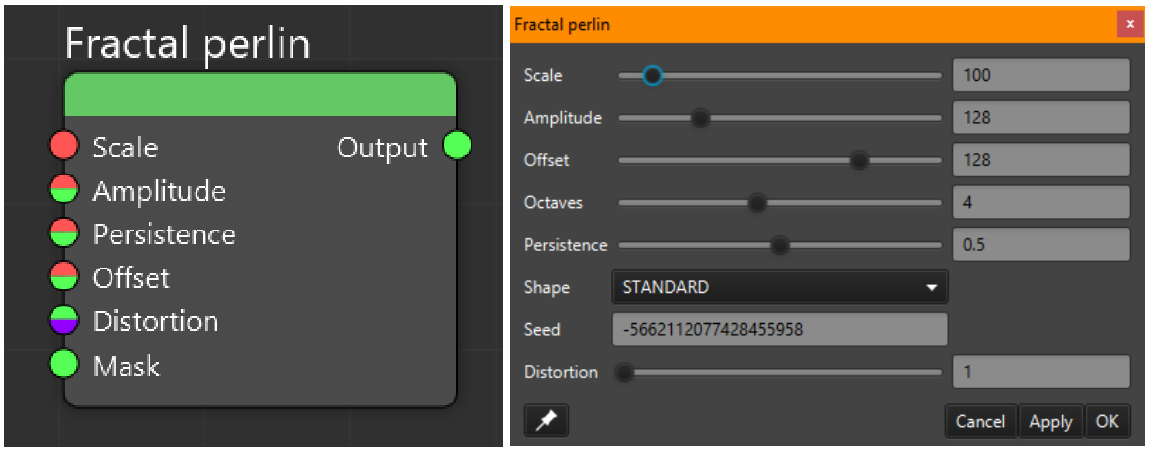
The types of information mentioned as example of outputs in the above segment are just a few examples of what types can be operated with. This information that it is moved along the graph is referred to as "data types" in WS. Each data type is used to store and different types of data used to representing a terrain or data used in its construction, and each data type implementation has an according interface to set and retrieve this data.

At the current time there are two important distinctions to used data in WS; there is globally constant data, and location based spatial data. Globally constant data is not bound to any spatial location or region, example of such are scalar values that can be used as inputs for parametrization of complex patches, with possibly algebraic combinations used as inputs to one or more modules that can accept scalar inputs. There is also spatial data that on the other hand is location based data, and this describes lactated data that needs to exist within a given location bound of up to 3 dimensions. In WS this location bound is called an for an "extent", and an examples of this type of data can be heightmaps. Heightmaps are a located data type that describes terrain height within a given two dimensional with a start X and Z coordinate, and a width and length relative to the start coordinate. Another example is the "materialspace" type that stores voxelized discrete material assignment within a three dimensional bound, and can be used to store for example a soil compositions underground.

Data types in WorldSynth are arbitrary to their purpose, and a data type for any purpose may be implemented as long as an interface for it can be created. Taking advantage of the object oriented programming and lambdas in Java, it is also possible to implement data types for functionality instead of discrete data.

All data types in WorldSynth are extensions of the AbstractDatatype class in the WorldSynth that defines the interface for WorldSynth data types. An addon can implement its own custom data types by extending this class, and a preview renderer for the datatype is implemented as an extension of the AbstractPreviewRender class and defined for the data type as a part of the data type implementation.

2.5.3 Modules



Scale 100
Amplitude 128
Offset 128
Octaves 4
Persistence 0.5
Shape STANDARD
Seed -5662112077428455958
Distortion 1

Figure 2.9: Example of a module and its parameters, shown in WS.

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

Modules in WS functions as containers for algorithms that can create, modify or convert the data of different data types. They can have an arbitrary set of inputs and outputs that can be connected up with compatible IO of other modules. Each module also also has an arbitrary set of parameters that are used to configure the module.

All modules in WorldSynth are extensions of the AbstractModule class. The AbstractModule class implements the base functionality of a module and the interfaces that need to be implemented for specific modules to define relevant IO and parameters, and for performing the module's functionality.

When building the output results of a module, the module gets a requested for an output. The module will first get a call to the **getInputRequests** method whare it will determine what inputs it need to build the requested output. This request for inputs is then returned and the individual requests are forwarded to any other modules connected the the inputs to the current module for processing.

Later the module will a call to the **buildModule** method with the same output request as well as any of the requested inputs that could be provided. According to the request for output data, and the provided input data, the implemented algorithm for the module will then generate the output if possible and return this. A more detailed description of the working of this process is found in Appendix **??**.

Implemented modules need to be registered by the module registry before they can be instanced in a patch.

2.6 Previous works

2.6.1 Feature-modeled procedural terrain generation with adaptive height evaluation

In my previous work leading up to this project, explored feature defined terrain generation for terrain formations with mountain ridges, rivers and lakes [1]. The method developed in this work allowed for the creation of low detailed terrain by procedural layout of primitive skeleton features, and the evaluation of terrain elevation of layout in the form of a function that can be used to continuously get terrain elevation in a general area where such features exist in the skeleton layout.

The method developed was based on first performing a layout of features according to an algorithm that can place the major features in the geography like mountains, lakes and river with a reasonable relationship to each other. The features represented as simple geometric primitives like lines and polygons, with corresponding elevations. And then a terrain elevation is evaluated as a weighted average if such features based on distance.

The method is broken down into several smaller discrete operations that may be treated with a high level of modularity. These operations are executed in an order that may be described by a Directed Acyclic Graph (DAG), and this order of execution of operations for how the method generated the layout, may even be graphically configured as a form of data flow programming, as it is made available in WS.

The layout is mainly based on the creation of base landmass outline in the form of a polygon, created by randomly displacing the vertices of a low-poly circle along a line through the center and the given vertex, and then applying some number of iterations of MPD to created further detail of a coastline. River may then further be grown in lands by the use of a Poisson Disk Sampling (PDS) process where newly grown points are connected with the parent to form lines. Along with the growth of rivers, there may also be generated layout for lakes as well, although the possibilities for different approaches to the creation of rivers and lakes are briefly discussed in the work itself. Finally the work creates layouts for of mountain ridges by creating a voronoi diagram from then previously laid out features, and removing edges in the diagram that are assumed to cross previous features, before a final conflict resolution is performed to clean up remaining artifacts of the mountain ridges layout. There is also a discussion relating different approaches for conflict resolutions with possible applications for different scenarios where features may be interfering by intersection, though the only approach used in implementation is removal of conflicting features based on priority.

An illustration of the configuration for skeleton layout with rivers, lakes and mountain ridges in an island outline is shown in Figure 2.10.

Further on, the elevation evaluation is as described preformed by a process of distance-weighted average of feature elevations. It is essential that all primitives describing a feature has an SDF function that can evaluate the euclidean distance to

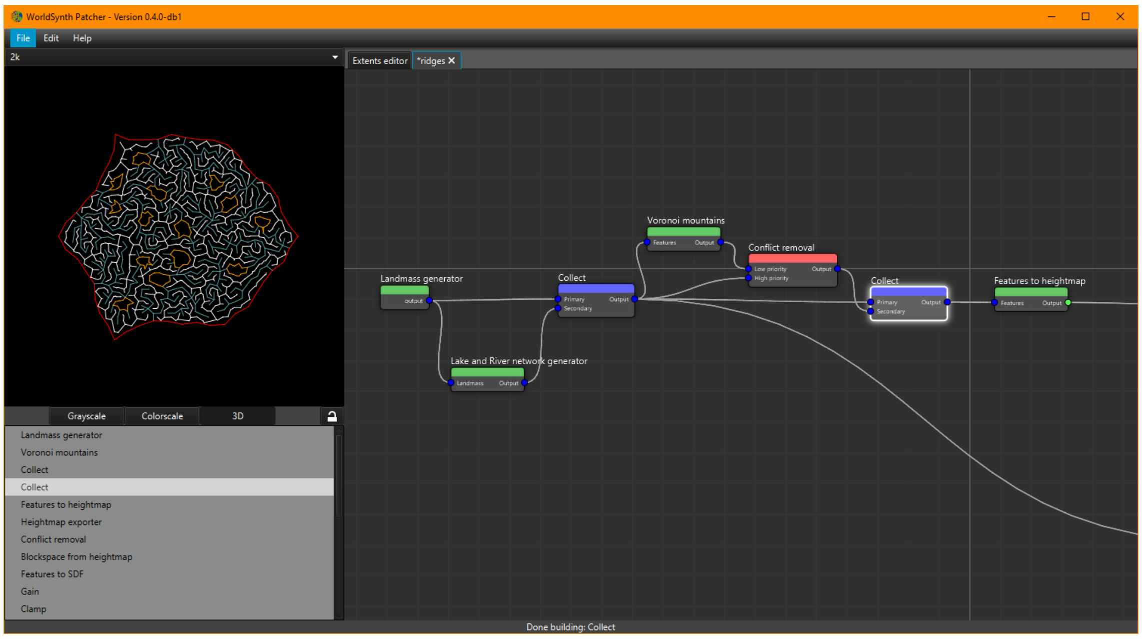


Figure 2.10: Skeleton layout for the terrain of the results from the previous work. In the layout preview the color coding is such that: red is the landmass outline, cyan are rivers, orange are lakes and white are mountain ridges.

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

the primitive from any point in the horizontal plane. Also every primitive provides a height function that can be evaluated at any point in the horizontal plane to acquire an elevation for the primitive at the closest point. This elevation of the primitive may be different, as for example for the line primitive that may have two different elevations at each end.

$$h(p) = \frac{\sum_{i=1}^{n} h(F_i, p) \times w(F_i, p)}{\sum_{i=1}^{n} w(F_i, p)} \tag{2.1}$$

$$w(F_i, p) = w_d(d(F_i, p)) \tag{2.2}$$

The weighted average is calculated according to Equation 2.1, where F is a collection of feature primitives. The h function is the height evaluation for the current feature being iterated over at a point in the plane p. The function w is the weight evaluation of the feature according to Equation 2.2, where w_d is some weight function that takes the input of the distance evaluated by function d for feature primitive F_i at point p.

The distance function $w_d(d)$ may be an assortment of possibilities, in the work, there is used three distance weight functions for w_d given in Equations 2.3 through 2.5, the resulting outputs from using each of these are shown in Figures 2.11 through 2.13.

$$w_1(d) = \frac{1}{d} \tag{2.3}$$

$$w_2(d) = max(0, \frac{1 - \frac{d}{a}}{d}) \tag{2.4}$$

$$w_3(d) = max(0, \frac{1 - (\frac{d}{a})^4}{(\frac{d}{b})^2 + 1}) \tag{2.5}$$

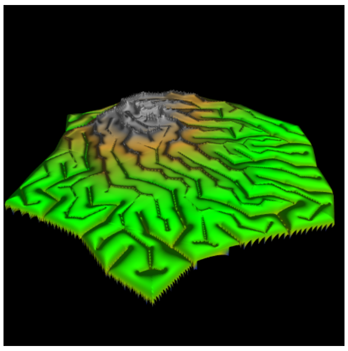


Figure 2.11: Height evaluation using distance weight function $w_1(d)$

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

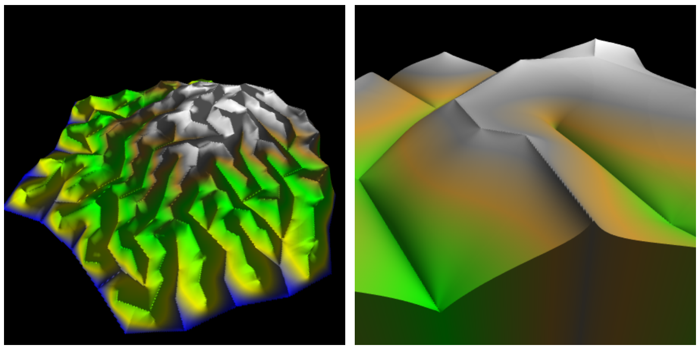


Figure 2.12: Height evaluation using distance weight function $w_2(d)$

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

The results from using each of these distance weight functions showed that the first alternative w_1 was gave poor results where the results overall tended to a global average, while the other alternatives w_2 and w_3 gave better results for merging features as these have a limited range where after the weight falls to zero, and thus has no more contribution to the resulting elevation beyond this range. In the Equations 2.4 and 2.5 for w_2 and w_3, this range is given by the value used for a. In Equation 2.5, there is also a value b that determines how fast the weight falls close to the feature, which can be used to make a feature appear wider or thinner in the resulting height evaluation.

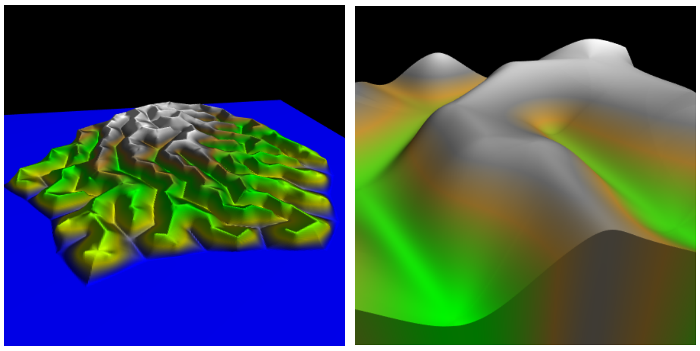


Figure 2.13: Height evaluation using distance weight function $w_3(d)$

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

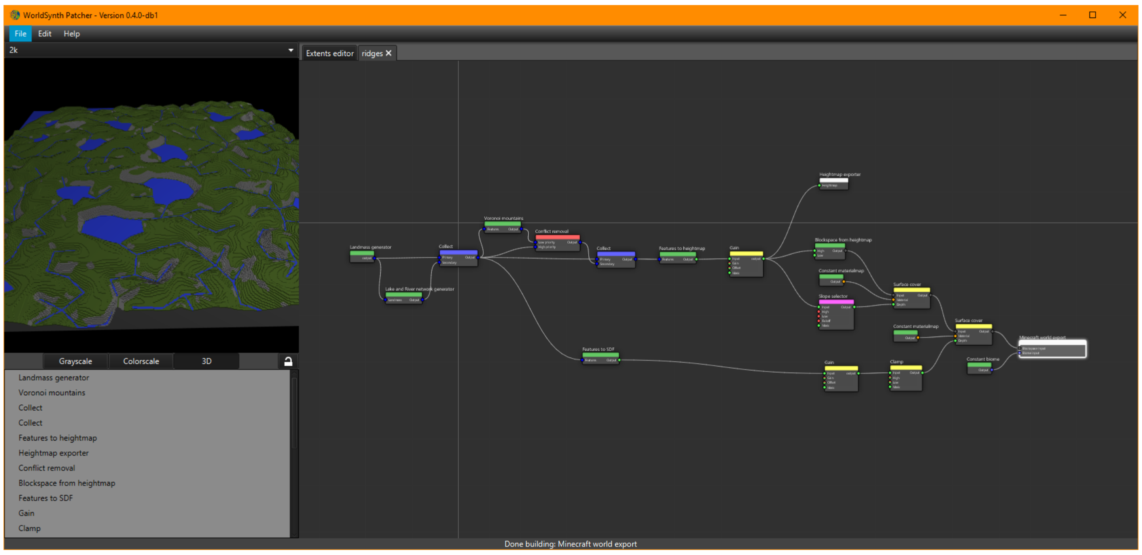


Figure 2.14: The evaluated results from the skeleton in Figure 2.10 used to construct a voxel terrain.

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

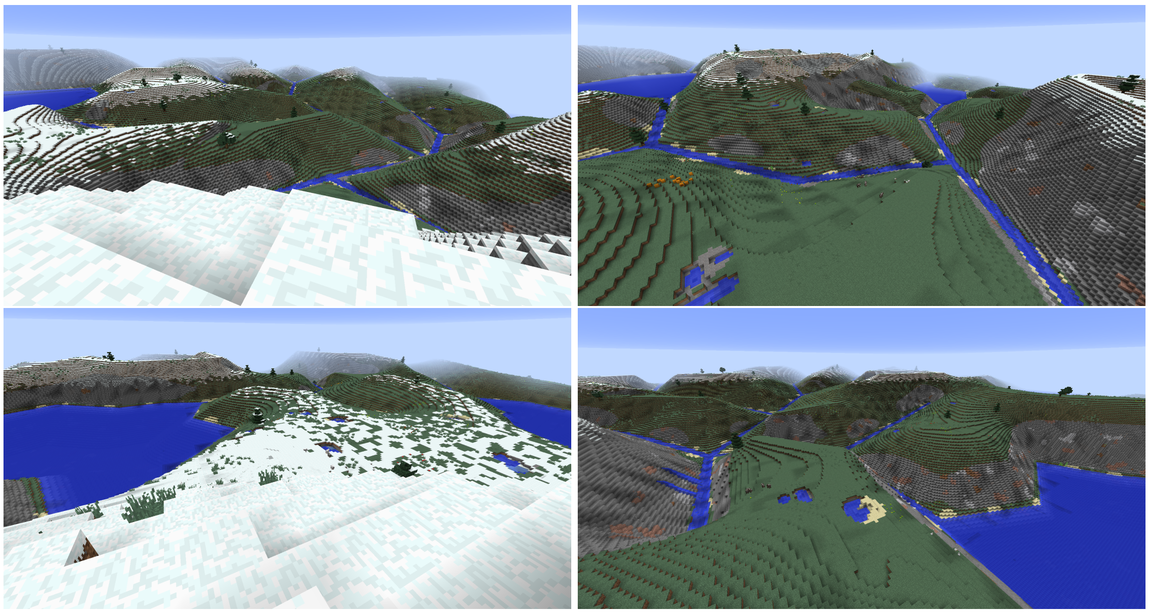


Figure 2.15: Results from the previous work.

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

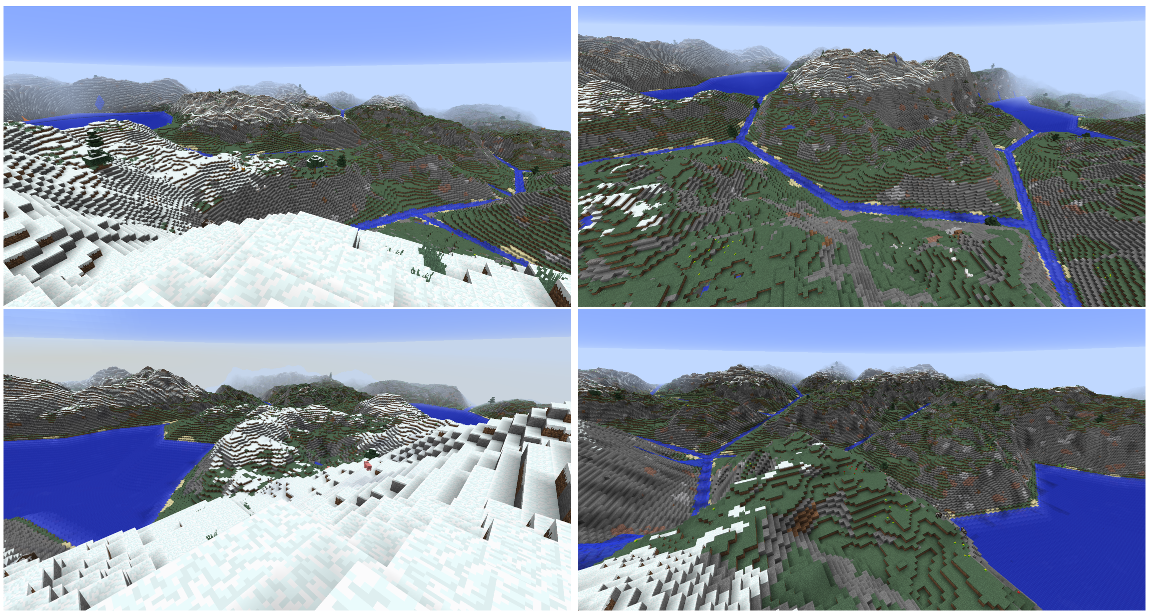


Figure 2.16: Results from the previous work with added noise for detailing.

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

Chapter 3

Specification

The work herein builds upon my previous work on modelling terrain using a procedural geometric skeleton description of features, and evaluation by merging these to produce a heightmap for a terrain [1].

Though the previous work produces the wanted results it was intended for, being heightmaps featuring a general structure for terrain including mountain ridges, rivers with a consistent downwards path, and also including lakes. The method has significant room left for improvements in relation to efficiency and the visual complexity of the results.

The goals in this project is to build further on the previous work to:

1. Improve evaluation efficiency
2. Change to accommodate non-global weighting of features
3. Add capability for generation of additional layers of terrain data
4. Generate terrain with more detailed feature geometries

These goals are set based on a desire to bring the method explored on in previous work closer to a state where it may be usable in an interactive application like for example a game, while producing visually pleasing results, and including data about additional attributes of the terrain beyond just elevation.

3.1 Evaluation

In the previous work, there was little attention given to the efficiency of the method, and the work was instead just concerned with the overall concept, in the layout of skeleton features, and the evaluation for merging feature values to produce the resulting elevation.

The evaluation method iterated over all features in the skeleton during evaluation of any location. This was in part a result from the initial design assumptions, where a distance weight function did not have a limited bound to its range of influence. But this weight function showed itself to produce undesired output results, in contrast to the the weight functions giving a bounded influence. The global range of influence also leaves little room for optimization as it demands the contribution of all features be considered.

By changing the approach to use skeleton features with limited bound to their influence, the possibility is opened to exclude features from evaluation when evaluating terrain that falls outside their bound of influence.

This exclusion of features from evaluation outside of their influence bound, may be rather better reformulated as the inclusion of features when inside their influence bound. This is a better description of the desired behaviour, and the formulation also points in the the direction of a possible solution of spatially indexing features, as that opens for a query to be performed during evaluation to acquire only the necessary features that may influence the evaluated part of the terrain.

3.2 Attributes

From the previous work, the resulting output from the method is the elevation of the terrain. There was however also shown that with additional posterior processing, one could to some extent generate additional data describing the terrain.

An example of such was shown using the SDF values for a the set of features describing rivers and lakes, where the distance was applied to determine where there should be water according to a distance threshold, and further carving out the rivers and lakes and filling them with water. But this was not an integrated part of the main method presented, and was instead performed separately as a posterior process from the main evaluation giving the elevation.

The generation of such other data to describe the terrain, should preferably be performed as a simultaneous evaluation process in a systematic manner.

3.2.1 Additional terrain data

To address the generation of further data to describe the terrain beyond elevation, the method is extended by introducing attributes representative of each data value used to describe the terrain at any point, like for example elevation, surface material and water depth. Whereas these attributes mentioned as examples should be implemented during this work, further attributes should be possible to add in a systematic manner.

Any attributes should be assignable to any feature in arbitrary combinations, such that the attributes may be used to model the contribution of a feature on the corresponding values of the resulting terrain. As such, any feature may be created to have an effect on multiple values describing the terrain, like elevation and material for a mountain ridge, or just a single value like the material for a footpath that does not alter the terrain elevation.

3.2.2 Non-global weight functions

The use of globally applied weight function to all features during evaluation in the previous work, allows for little customization of transitions between different features. As it was seen, different weight functions produced different characteristics in the results like sharp and smooth edges. In addition, this global application also puts a limit on the possibility of having features with a relatively smaller or larger influence range for merging with the terrain.

To address this, the weight functions applied during evaluation may with advantage not be a parameter of the evaluation process, with global effect, but instead be a property of the respective features. As such, individual features may be assigned unique weight functions as desired during creation of the skeleton layout, for example to create smooth valley bottoms and sharp mountain ridges, by the application of two differing weight functions for the respective features.

When considering the addition of attributes to model different data describing the terrain, the method no longer merges just elevation values though. Given the assumption that all attributes will be merged according to distance weighting in line with the previous work, it may be desirable to merge different attributes of a feature according to differing weight function. Therefore the weight functions should instead be applied as properties of the attributes of a feature than an attribute of the feature itself.

3.3 Details

The results from the previous work bear a clear characteristics of being derived from simple geometric shapes, as the results display long stretches of straight lines and little detail. This looks distinctively artificial as natural terrain exhibits surfaces and outlines with fractal properties.

The previous work showed examples attempting to address this to an extent by processing of the results. One such example made the elevation less smooth, but this did not address the straight nature of prominent features like for example the rivers, and another was the application of simulated hydraulic erosion, but this to an extent defeats the initial purpose of the work to avoid the usage of simulation.

Instead of relying on posterior processing, it is desired to create further details through the method itself. The obvious approach to creating more details is to create more features, for a more detailed skeleton layout. Besides that this should be expected to increase the number of features needed to be evaluated, and thus increasing the run time of the evaluation, it may also be reasonable to consider that this may have an impact on the shape of resulting transitions between features.

As such, this work should explore the results of applying this obvious approach, as well as explore further possible approaches that may perform better to enable increased details, taking advantage of the changes to evaluation and features.

Part I

Attributes and Evaluation

Chapter 4

Design

4.1 Attributes

4.1.1 Terrain attributes

When modelling an environment, there will usually be several aspects used in conjunction to create a more complete model, such as for example the elevation, the surface material of the terrain, and also aspects describing the biome of the endowment such as vegetation, echo systems, temperature and humidity. When the environment also contains bodies of water like rivers or lakes, we probably also want to know the depth of these water bodies, or some variation of information pairs that in composition allows the implication of water depth, or to determine other information about the environment.

In models where water is involved situations, the possibly simplest way to describe water level may be as simple as determining a global water surface elevation, and thus considering there to be water at such a height over any terrain that has a lower elevation. But this simple method is obviously not an applicable method for modelling environments with bodies of water at non-global elevations, like lakes at different elevations, and rivers flowing downwards and thus changing elevation under ways.

The description of such aspects of the terrain is essential and I will consider them to be attributes of the terrain, and it should be necessary to involve modelling of these attributes in the generation of a complete description of the terrain.

It is desirable that the method for adding such attributes to the modeling be modular in its implementation so that new attributes may be easily implemented in a structured manner. In this work I will primarily be focusing on covering aspects of the terrain that was covered with additional posterior processing in the previous work as listed below, but with with a systematic approach that is easily extended to new attributes as they are needed.

1. Elevation

2. Surface material
3. Water depth

The method from the previous work, where only elevation was taken into consideration, is intended to be possible to produce deterministic results for any single point in the continuous range of the horizontal plane where the terrain is defined on evaluation, without any dependency on results from neighbouring values to perform this evaluation. Therefore it is natural to look at the possibility of extending this approach, such that the results other aspects may be evaluated in a similar manner based on distance-weighted values.

With the introduction of further aspects that may be of a different nature than the elevation aspects of possibly different nature to how they may be described than the elevation in the previous work, as not every attribute may be best described by a single floating value like the elevation, like the surface material as an example, the modelling of attributes with different value descriptions should be explored. Here in this work I will be distinguishing the attributes into continuous and discrete attribute, but there may be more ways of modelling an attribute if it is of a more complex nature that may involve multiple values in a composite, while I expect the modelling of such attributes to be possible to build upon the foundations understanding of these two classes of attributes I describe herein as continuous and discrete attributes.

4.1.2 Continuous attributes

As a general description for continuous attributes as considered in this work, continuous attributes are used to model aspects of the terrain that have values existing on a continuous scale, such as for example the elevation, which is may change in non-discrete steps between evaluated points, ant may theoretically taking on any fractional where this is only limited by the computer's handling of this value, as in the example of the elevation where this is the precision of a floating point value.

Also the water depth attribute is a primary candidate for being a continuous attribute as this is very similar to the elevation of the terrain, but instead would be used to describe the depth of a body of water for any point being evaluated.

Building on this basic understanding of a continuous attribute to be a consciously variable value, it is also worth to mention the extension of the possibility of extending such values beyond just monotone values and the possible composite descriptions of single aspects, like for example color, where a value may be described by several sub-components values such as values for red, green and blue components, or a hue, saturation and lightness depending on desired function. While the value of a color may be more complex to describe with several components of simple primitive values, in contrast to elevation that may be described by a single such primitive value, it is still a continuous value as all the sub-components describing the value are also continuous by themselves and therefore form a continuous range with multiple dimensions.

4.1.3 Discrete attributes

As described previously, attributes can be continuous, like the terrain elevation, but some attributes of the terrain may not be best described by continuous values, or the description as a continuous value may be either too complex, or just not a necessary complexity for the usage of the evaluated results, thus is also entered the concept of discrete attributes.

Some examples of such discrete attributes of the terrain may be shown in for example surface materials, like rock and soil materials for describing the terrain surface. While such materials in reality may not be ultimately correctly described as discrete, as there may exist continuous transitions in the composition of a soil material over an area in the form of what minerals, organic materials and particle sizes it is made up of. The modelling of such may for many purposes be unnecessarily complicated, demanding and challenging for procedural terrain generation, depending on the application. And in sch cases the surface material may possibly instead be be described to satisfaction from an enumerated set of possible surface materials. It is therefore necessary with techniques for handling merging of discrete attribute types like these.

When merging of attributes based on discrete values, it is important to take into consideration what the wanted output of such a merger should be. For discrete types it may be possible that the wanted output is either a single discrete value or possibly even a collection of several values. An output with single discrete discrete value may describe the terrain such that for example a result that says "the ground surface is granite" or "the ground surface is sand". Alter naively an output based on a collection of multiple values may for example describe a material mix such that it says that "the surface is a mix of granite and sand", or even further be paired with additional weight to describe fractional mixes such as "the surface is a mix of 60% granite and 40% sand".

These different approaches to formatting an output describing the attribute may correct for different purposes, but it is worth noting here that the fractional mix example starts to show significant similarities with a continuous attribute featuring multiple dimensions of complexity, similarly to the color example described before, but with discrete materials as the dimensions in this case. This highlights how the difference between describing an attribute as either continuous or discrete, may not be entirely straight forward as to simply separate of attributes into two classes. Especially through the merging process, the way a value of an attribute describes the terrain may possibly be transformed from being single discrete materials as input, and outputting as a continuous fraction, or the other way around where the inputs to be merged are continuous values, while the output may be discretized to a closest discrete value in a set, that best describes the merged results.

4.2 Features

In the previous work, where only the terrain elevation was evaluated, the difference between a shape and a feature was not distinct, and the implementation of the shape itself was used for the representation of the feature where the shape by itself fully defined the position and elevation of the feature it described, as was all that was needed. A collection of features where as such represented simply just by a collection of shapes.

Also, what sort of a feature of the terrain, like rivers or mountain ridges, such a feature in the layout represented, was a result of its creation and usage in relation to other features in the layout. The actual mechanic of such features to be a representations of for example a rivers or a ridge lines in the final terrain, where not discrete in a way resulting from coded implementation enumerating type of features, but were instead implicitly defined by relationships emerging from layout of the features, and how grouped subsets of these features where collected used in a larger configuration.

This method of having feature types implicitly emerge without the need for being enumerated in code, where in the previous work claimed to have interesting properties, in the potential for creation of new types of features in the terrain, like for example roads, by relying primarily on the creation of layout algorithms that can create the desired layouts for such features, and without the need for further defining new behaviour for the features as representative units for the layout.

In this continued expansion of the method, this is an interesting aspect to the method to build upon, and the usage of attributes should help improve this implicit definition of feature types to be even more flexible.

A feature should as per the previous work, be described by a shape, but in addition it is herein also extended to define a set of attributes per feature.

4.2.1 Feature shape

From the previous work, shapes are simple 2D primitive geometries in the plane, that are used to define the position and form of the any feature in the terrain in a discrete manner. One of the most important aspects of such a primitive is that there needs to be some method for evaluating a distance between the primitive and any point in the plane, usually in the form of an SDF function. Additionally these primitives are also used to define the elevations of these primitives, though differences elevation should not impact the results distance elevation. Thus the shape also needs to be possible to evaluate for elevation at any point in the plane.

4.2.2 Feature attributes

As well as the previously mentioned shape the is important in defining position and form of a feature, a feature with attributes should also have attributes that define further details about the feature. For this a set of attributes is used that may contain any arbitrary number and selection of implemented attributes

These arbitrary attributes may be assigned to a feature without any constraining requirements of specific collections of attributes that must or must not be assigned in conjunction for the instance of any feature. This means that while a feature need to have defined a shape, it may not necessarily need to have assigned any attributes, though such a feature will thus not have any impact on evaluated final results, it may still have possible applications in intermediary steps of the layout procedure. In some other possible scenario, a feature may also have defined a set of attributes that involves multiple attributes of the same types, but with different values for each instance of the attribute assigned to the feature.

Attributes as assigned in to features in the layout, are during the evaluation merged into the final attribute results as was mentioned in 4.1.1. This merging is based on distance weighting of an attribute's value, therefore distance weight functions are defined per attribute, instead of the per feature itself like it was in the previous work [1]. Design of distance weight functions for this discussed in 4.2.3 and merging is further described in 4.5.2.

Mainly for the purposes shown in this work, most if not all features will define the terrain elevation attribute and thus contribute to defining the terrain elevation, and will have additional attributes like surface material and water depth where necessary.

4.2.3 Feature distance-weight

In my previous work [1], the method allowed the evaluation of terrain height height by taking a distance-weighted average height of all features present in the skeleton layout. The weight of the height contributed by any feature was determined as a function of on the distance from the evaluated point to the relevant feature in the form of the closest distance to the describing shape in the feature. Note that the distance is only calculated in the plane without taking into account the any distance in height of the feature.

In the previous work there was presented and used three different examples of weight functions that take the distance as the input value, respectively w_1 through w_3.

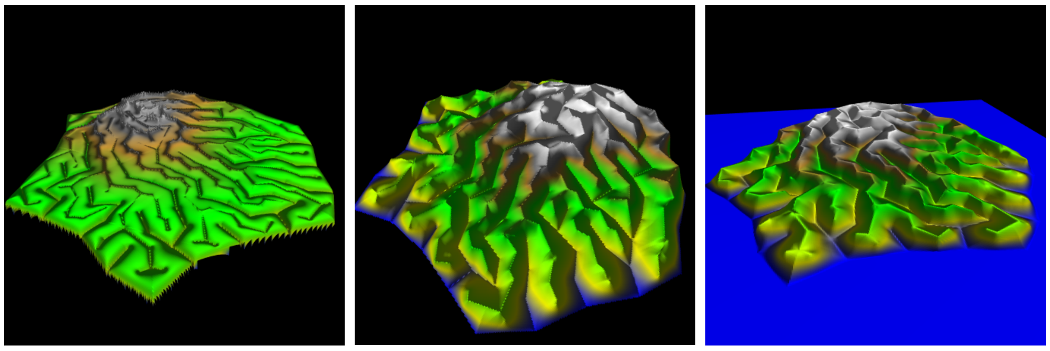
Considering these weight functions, w_1 exhibits an unbounded area of influence where the output weight tends towards zero as the input distance tends towards positive infinity, but the weight never truly becomes zero. Meanwhile w_2 and w_3 exhibits a bounded area of influence, where the weight falls to zero as $d >= a$, giving the feature now influence on the output of the weighted average beyond distance

$d = a$.

$$w_1(d) = \frac{1}{d} \tag{4.1}$$

$$w_2(d) = max(0, \frac{1 - \frac{d}{a}}{d}) \tag{4.2}$$

$$w_3(d) = max(0, \frac{1 - (\frac{d}{a})^4}{(\frac{d}{b})^2 + 1}) \tag{4.3}$$



(a) w_1 (b) w_2 (c) w_3

Figure 4.1: Example results from height evaluation with previous example distance-weight functions.

Source: Feature-modeled procedural terrain generation with adaptive height evaluation [1]

4.2.4 Unbounded distance-weight

Results produced using these three weight functions are shown in Figure 4.1, and shows that the unbounded distance-weight function w_1 had an overall tendency to have the terrain height tend towards the global average very fast between features, which is an undesired behaviour as the goal of the height evaluation is to produce fitting terrain height such that it blends surrounding features at the evaluated point.

Another issue with the unbounded distance-weight, is that by its infinite nature, the infinite bound leaves little to no opportunity for optimizing the evaluation as the complexity in the numbers of features in the skeleton layout grows, either as a result of larger layouts or more detailed layouts. With features with an infinite bound, the evaluation time must grow with an $\mathcal{O}(n)$ complexity with increasing number of features, as the influence of every feature needs to be considered for each evaluated point. And since the the features approximately uniformly spans an area of a layout, the number of features will scale quadratically with uniformly changing layout sizes, thus resulting in an evaluation time with $\mathcal{O}(n^2)$ complexity with increasing layout size.

4.2.5 Bounded distance-weight

In contrast the bounded distance-weight functions w_2 and w_3 did both show better conservation of a local average, and they also allows for optimization of the evaluation process, where features need not be evaluated when the point being evaluated is beyond the bound of the feature's influence since the weight then becomes zero. An example of this relationship between features and their area of influence is shown ion Figure 4.2.

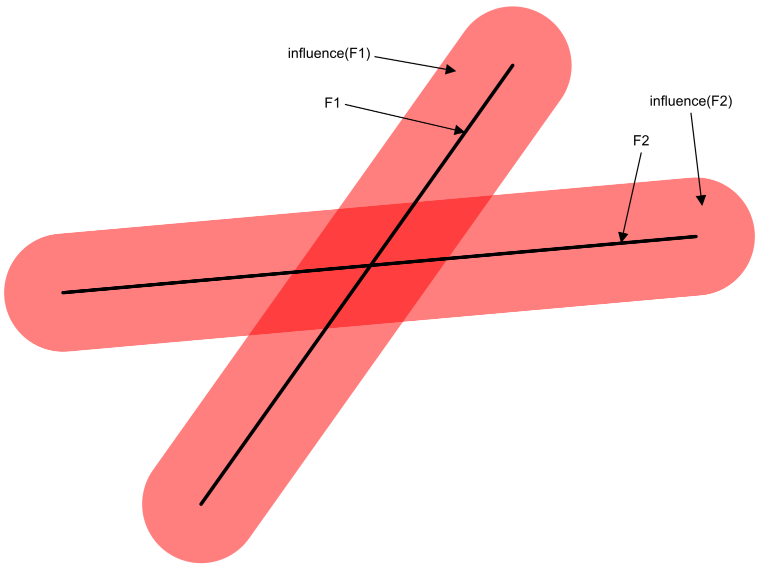


Figure 4.2: Feature F1 and F2 with a bounded weight function each has an area of influence illustrated by the red zones. For evaluation beyond these zones they do not influence the results.

As bounded distance-weight functions has showed the best promise in producing the wanted results, and also have opportunities for optimization, they will be the type used in this work. Also while the distance-weight function was global in the previous work and therefore affected the contribution of all features equally, this continued work applies the distance-weight function as a property per feature, meaning that different features can have different distance-weight functions, allowing for features with different characteristics in their blending and range for influence. An example of this could be to use previously mentioned w_2 for ridge features as this produces a sharper ridge, while w_3 could be used for river features producing a more smoother valley bottom.

$$w_1(d) = max(0, 1 - \frac{d}{a}) \tag{4.4}$$

$$w_2(d) = max(0, \frac{1 - \frac{d}{a}}{d}) \tag{4.5}$$

$$w_3(d) = max(0, \frac{1 - (\frac{d}{a})^4}{(\frac{d}{b})^2 + 1}) \tag{4.6}$$

4.3 Attributes assignment to features

As features may have arbitrary collections of attributes, it is important hat these attributes need to be possible to assign to a feature, and there needs to be an understanding of when and how these are assigned, based on a knowledge of why each specific feature needs their respectively assigned attributes.

As described several times previously and in the previous work on this method, the nature of what a feature represents in the terrain is a result of emergent relations and definitions by configuration of the process that generates the layout, and does not involve coded enumeration in the implementation. As attributes are herein an important aspect in describing a feature, it is herein natural to expect attributes to be assigned features during the layout generation.

To perform this attributes assignment, it is reasonable to perform this assignment as part of the layout generation for specific features in the layout that should represent the same type of feature in the final terrain, such as for example rivers, as these may be expected to all share, or at the very least have very similar attributes.

The assignment of attributes during the layout process may be designed such that a layout process for a specific type of terrain feature involves the assignment of the required attributes as part of the implementation for the layout algorithm, but a further modular approach to the assignment of attributes during this layout would be to provide the set of attributes to be assigned the generated layout feature as an argument to the layout algorithm such that a layout algorithm may be reused for features with different configuration of attributes to be assigned.

This assignment of separately configured attribute sets is the method used in this work, and the modular approach later seen herein with the modular reuse of detail generation with MPD and separate configuration of attributes during the layout process where this allows the same layout algorithm to be used for different terrain features.

4.4 Layout

The layout of features is as general operation performed using the same method as in the previous work with minor changes to the implementation [1]. The changes to the the implementation of these layout methods are the inclusion configurable assignment of attributes as described in 4.3.

Additionally the layout generation herein also involves the generation of additional detail based on previously laid out features by the use of MPD, but the workings of this is further described in 8.1.

4.5 Evaluation

4.5.1 Merging features

The merging of features are performed by merging their assigned attributes. See 4.5.2.

4.5.2 Merging attributes

The task performed during the evaluation process of the method, is the merging of attribute values from close by features such that the values are best fitted to the values described by features in the skeleton structure, and that these values transition desirably in the range between the features.

In the previous work, the evaluation only considered terrain height as the value to be evaluated, but with introduction of attributes of the terrain that may be extended beyond just terrain height, merging process and resulting output need to be extendable to handle more than just this one attribute.

In the previous work, the merging of height height values from the multiple features was performed by using a WeightedAverage object. Along with a vector to the point being evaluated and the used distance-weight function, this WeightedAverage object was passed to a method part of the interface for the feature, to all features through a method provided as an interface with the feature instances. The distance-weight function was used in combination with the distance and height samples for the point to add the contribution to the WeightedAverage object. After all the features had been called, the WeightedAverage object could then be queried for the final average.

It is my suggestion for the extension that a similar process be used where a ”merger” object implementing interfaces for adding contribution and querying final merged results for all attributes be passed to the feature, since the distance-weight function is already provider per attribute, there is no need to pass a distance-weight function along with this any more. Each feature can then pass the merger object to all the registered attributes for the feature such that they can call their respective interfaces for contribution. After the merger object has been passed to all relevant features for this contribution addition, it can then be queried for the results of the various implemented attribute types.

4.5.3 Evaluation optimization

A significant part of the previous work [1] involves interactions between nearby features, while the processes that where applied featured full iteration over all existing

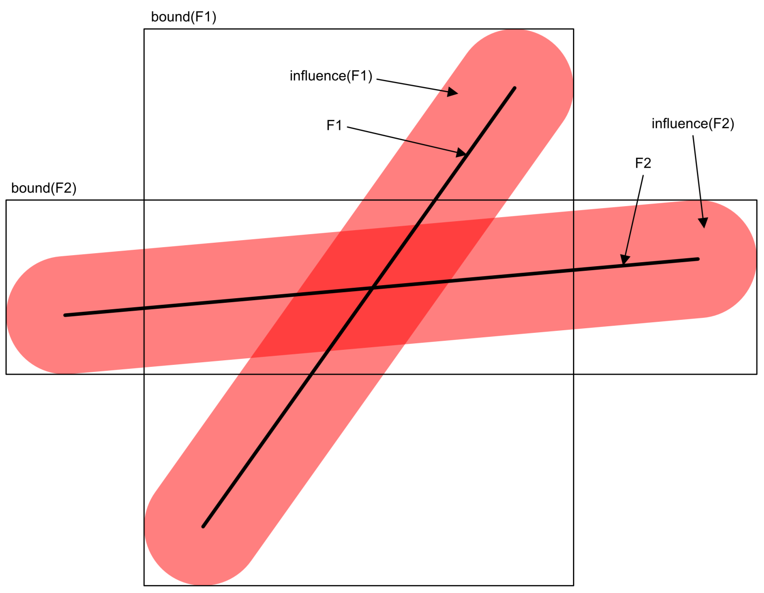


Figure 4.3: Feature F1 and F2 each has an area of influence illustrated by the red zones. A Minimum Bounding Rectangle (MBR) can easily be constructed for which a feature may be excluded from evaluation when an evaluated point does not fall within the MBR.

data independent of whether there was even any chance of interaction.

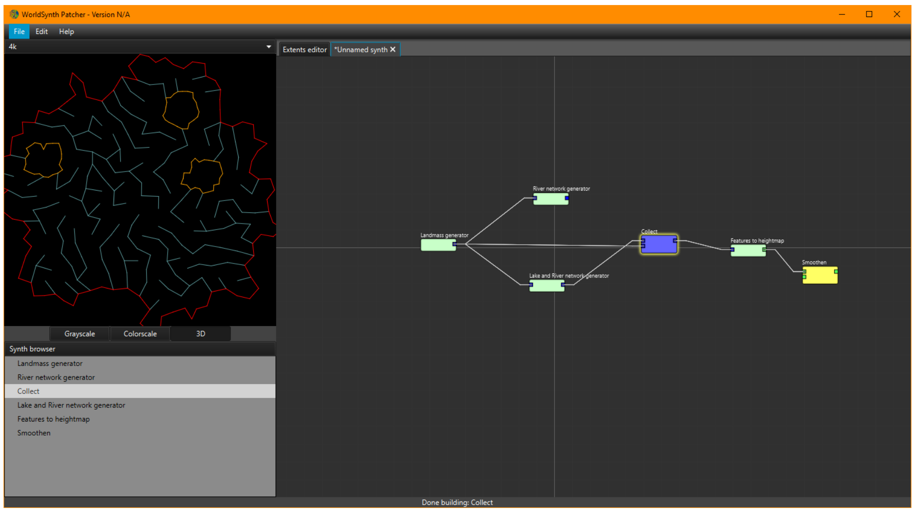


Figure 4.4: Example layout of moderate size.

Take for example the height evaluation method from the previous work that evaluates the weighted average of all features. As previously mentioned, in the previous work the features where stored in a list structure, this would not be of concern when using the w_1 weight function, as it takes an unbounded average where it is necessary to iterate over all the features. But when using w_2 and w_3, this changes as they take a bounded average where only features within a given radius of the evaluated point have any impact on the result. Features outside of this radius have no contribution on the average and iterating iterating over these features is thus just wasting time, especially important is this since this subset without contribution the may be significantly larger than the subset that actually have a contribute. It is therefore unfortunate and simply unnecessary to naively iterate over all features.

In the pursuit of improving the efficiency of the method, it is clear that efficient querying of the features to find the features close enough to the evaluated location to have an impact on the result should be of major importance.

To do this I propose to apply spatial data structures for storage of the features. Before discussing what data structure to apply to the problem, there is first a need for understanding the data that will be stored, and how it will be queried for.

In the case of what type of entries that will be stored, they will be features described by geometric primitives like for example points, lines and polygons. And in regards to querying, the collection will be queried for any feature that may have an impact on the results for a height evaluation at a point.

Chapter 5

Implementation

5.1 Initial implementation setup

The implementation builds further upon the existing implementation from the previous work [18] [1].

5.2 Features

As described earlier, features are defined by their shape and attributes. The feature is implemented in this work in the form of the `ShapedFeature` class. The shape value and list of attributes are members in this class. As well there is an annotation color value, this color value is used for coloring the feature when preview visualizing the feature layout as primitives skeleton in the editor, but the color is not important to the evaluation results and is an implementation detail to make it easier to understand a layout when previewing in the editor.

5.2.1 Shape

The shape value of a feature, as described in 4.2.1, defines the form and placement of the feature, and is an implementation of a geometric primitive. The shape defines elevation when the feature has defined the terrain height attribute.

The implementation shapes in general has two important methods that need to be implemented for evaluation purposes, these are the `sdf(p)` and the `height(p)` which respectively returns distance and elevation of the closest point of the shape at `p`, where `p` is a vector in the plane to the evaluated planar location.

For the purpose of organizing features in a spatial data structure, like the R-tree, there is also a need for determining the bound of the influence the feature may have on the evaluated results. This bound is described as a bounding rectangle that not only encapsulates the whole shape as a minimum bounding rectangle, but also has to account for the range beyond the shape that the distance-weight of an attribute allows for the influence to extend. Therefore the shape needs to implement a method that returns a bounding rectangle expanded according to an offset, this is the `getBoundGeometry(offset)` method.

5.2.2 Attribute

The attributes of a feature, as described in **??**, defines the different aspects of the terrain it has an influence on. This may be the terrain elevation, surface material, water depth, just to take the few examples from this work, or any other value that can be used for representing an aspect of interest to the final terrain.

The evaluation of these attributes require features in the layout to have assigned values for these attributes with their desired value at the feature. This is done with feature attributes.

In addition to specify the attribute value at the feature, the attribute also defines the distance weight function for use with the attribute on evaluation of the given feature with the attribute assigned.

As each feature may have assigned an arbitrary collection of such attributes, and a lot of similar features are going to have the same values for these attributes, it is prudent to minimize the memory footprint of these attributes by reusing instances. Therefore the implementation of these attributes herein is made such that a single instance of an attribute may be assigned to any number of features that should have the same attribute.

Attributes are implemented according to the interface `Attribute`. This interface defines two methods that an instance of a feature attribute needs to implement:

1. `contribute(merger, feature, p, sdf)`
   The `contribute` method is called during the merging process to let an attribute registered to a feature contribute to the merged results. The same instance of an attribute may be assigned to several features if the behaviour and configuration should be the same, this can save on unnecessary instancing of unique attributes instances in many cases.

   `merger` is an instance a merger that with the appropriate method for the given attribute to call for its contribution.

   `feature` is the feature instance the attribute contribution is called for.

   `p` is a vector in the plane pointing to the position being evaluated, this can be used for implementation of attributes with spatially dependable values.

   `sdf` is the signed distance value of `p` from the feature `feature`, this is used to determine the weight of the attribute during when contributing.

   The method returns no value.

2. `influenceDistance()`
   The `influenceDistance` method returns the influence distance of an attribute

instance. The influence distance is determined through the weight function used by the attribute, where the influence distance is the SDF distance value in `contribute` at which the weight of the attribute contributions falls to zero and remains so for any larger distance value.

The method returns the largest distance at which the attribute has any weight more than zero.

5.2.3 Bound

As has been explained in **??**, the bound of a feature is used for organizing features in a spatially indexed structure like the R-tree. As the goal of this is to make the evaluation process more efficient by not considering features that will not impact the results because of the bounded influence distance, the bound should enclose the area where the feature may provide an influence on the output result.

This is done with a MBR that encloses the feature shape, with a uniform offset of the sides in all directions equal to the maximum influence distance of any of the defined attributes.

The feature implementation provides a method `getBoundGeometry()` that returns this bounding geometry. This is done by first iterating through all the defined attributes of the feature to find the maximum influence distance among the attribute, and then the `getBoundGeometry(offset)` method of the shape is called with the maximum influence distance as the offset value. The returned returned geometry from this the bounding geometry of the feature, which outside, the given feature is guaranteed to not have an impact on any evaluation results, and may therefore be ignored outside the bound.

Algorithm 1: Feature bound

```
input : The feature $F$ with shape $F_s$ and list of attributes $F_a$ to
        determine the bounding geometry for.
output: The bounding geometry $B_f$ for the given feature.
1 begin
2     Let $offset$ be the maximum influence distance of the feature according
      to the weight functions of all the attributes, this is set to 0 to start
      with.
3     foreach attribute $a \in F_a$ do
4         Set $offset$ to be the maximum of the current value and the influence
          distance of $a$.
5     end
6     Let $B_f$ be the bounding geometry of $F_s$ with offset $offset$.
7 end
```

5.3 Implemented attributes

Back in 5.2.2 I described the general interface for implementing an attribute, but I will next describe more specifics of the implemented attributes.

5.3.1 Elevation attribute

The elevation attribute is as the name suggests used to specify the elevation of a feature.

As described in 5.2.1, the shape also defines the elevation, and there is therefore no elevation value to specify with this attribute, and the elevation value is gathered from the shape of a feature by an interface created via the shape instance passed when the `contribute` method is called.

It should though be trivial to implement a variation of the elevation attribute that neglects the shape elevation for another value, or an offset; either may be constant or using procedural noise.

The elevation attribute may also define a priority that is used when merging, in addition to the distance weight function.

The `contribute` method receives the distance of the evaluated point to the current feature and applies the weight distance function to determine the weight to contribute the value with. The value itself is the elevation at the evaluated point. This elevation is acquired by an elevation method of the passed feature, for the evaluated point. Lastly the value is merged with any other feature attributes by the contribution method for elevation of the merger. This merger instance contribution method takes the input of the elevation, weight and optionally priority if that is implemented. The details on how the elevation merging is performed are described in 5.5.3.

5.3.2 Water depth attribute

In contrast with the elevation attribute previously described, the water depth is not specified by the shape or in any other way by the feature. The he water depth is herein this implementation instead defined as a function of distance to the feature, somewhat similarly to the distance weight function, with the depth decreasing as the distance increases.

As the depth goes to zero as the distance increases, it should also be natural that the range of the weight function is coordinated with the depth function such that the weight tends to zero over the same distance as the depth goes to zero.

When evaluation of the attribute contribution is performed, the weight and depth function is evaluated for according to the passed distance value, and used as arguments in the merger contribution method for water depth.

5.3.3 Surface material attribute

The surface material attribute defines the surface material at a feature. In the implementation of this work, this value does not change depending on position or distance, so this is a constant value of the attribute alongside the distance weight function.

When evaluating the attribute for contribution through the attribute contribute method, the weight is obtained similarly to the previously described attributes using the distance value and weight function, and the material value is added to the merge using the material contribution method of the merger instance with the material and weight as arguments.

5.4 Attribute assignment within WorldSynth

In the implementation of this work, attributes has been made manipulable trough WS as a datatype that can be used in a graphical manner, allowing for creation of individual attributes and collecting them for use with generated features, also in a graphically editable way with WS.

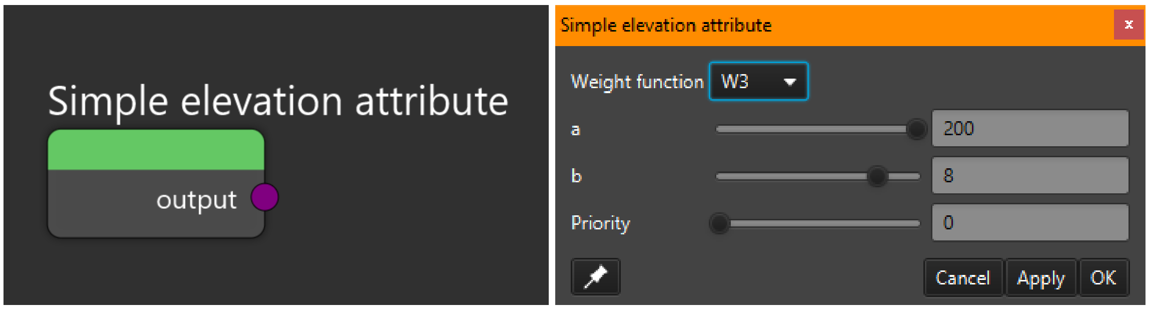
The layout generation modules from previous work has received an attributes input.

5.4.1 Attributes datatype

The attributes datatype is a simple datatype that is made with just a collection of attribute instances. The attributes are global and not located, and thus not dependent on any extent.

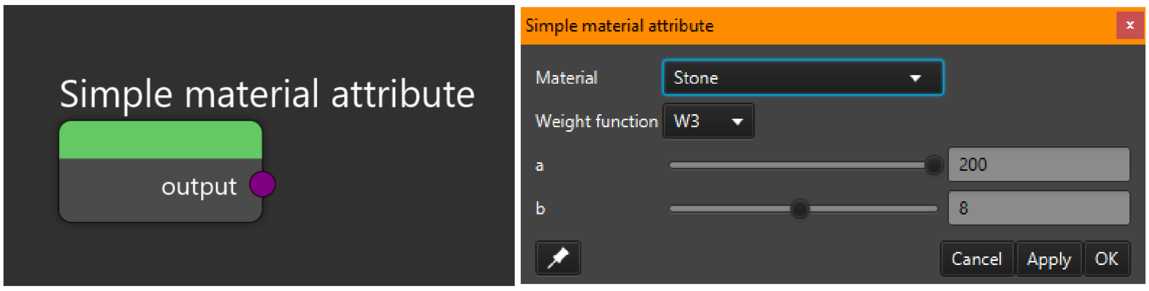
5.4.2 Attribute generators

In this implementation, attributes are generated using attribute modules. There are three modules made, one for each respectable attribute type implemented, being elevation, surface material and water depth.



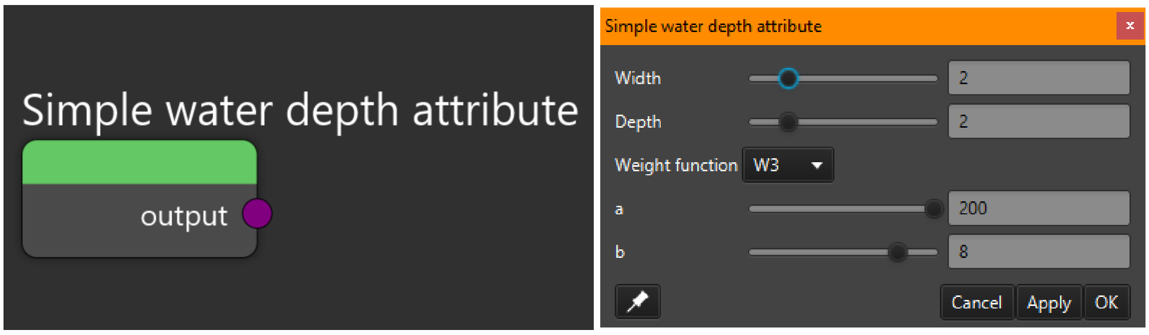
Weight function W3
a 200
b 8
Priority 0

Figure 5.1: The elevation attribute generator module and parameters shown in WS.



Material: Stone
Weight function: W3
a: 200
b: 8

Figure 5.2: The surface material attribute generator module and parameters shown in WS.



Width: 2
Depth: 2
Weight function: W3
a: 200
b: 8

Figure 5.3: The Water depth attribute generator module and parameters shown in WS.

5.4.3 Attributes collection

As with features in the previous work, there needs to be some way of combining attributes together into larger sets that can be used [1]. This is achieved with the implementation of an attributes collection module.

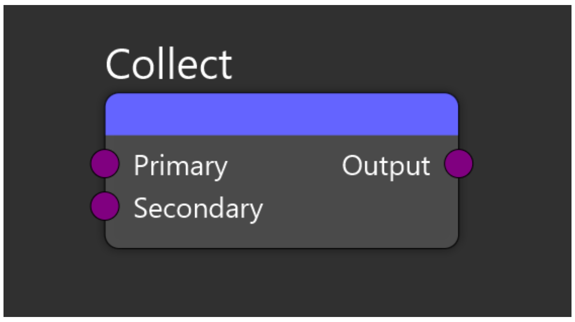


Figure 5.4: The attribute collect module and parameters shown in WS.

5.4.4 Attributes assignment

As the intention herein is to assign attributes to features at the layout time, the feature generator from the previous work needs to have the addition of attribute inputs for this purpose.

The usage of these inputs are purely extensions of the layout algorithms from the previous work to also apply the attributes provided by these inputs to the laid out features. For modules like the lake and river layout, there are provided separate inputs for usage with respectively the lake and the river features that are being laid out, since thees are to some degree different features created at the same time.

Algorithm 2: Feature layout attributes assignment

```
input  : The parameter inputs used for the relevant layout procedure.
         A collection of attributes $A$ to be assign for laid out features.
output:  A collection $F$ of features generated by the layout procedure.
1 begin
2     Let the layout of shapes be performed according to the respective layout
        procedure, collect these in a collection $S$. See the precursor work for
        details on different layout procedures [1].
3     foreach shape of a feature $S_f \in S$ do
4         Construct a feature using shape $S_f$ and attributes $A$, add this to the
            features collection $F$.
5     end
6 end
```

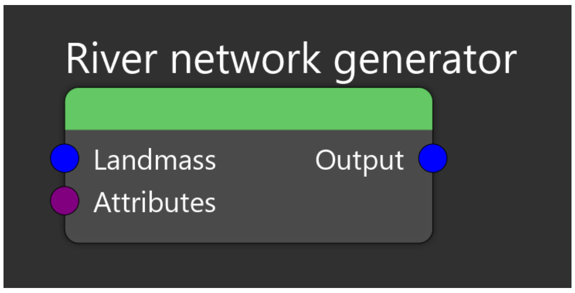


Figure 5.5: Example of an attributes input addition to the river network layout generator from the previous work.

5.5 Evaluation

The evaluation procedure performs the task of determining the results of all the features at any point of the terrain. This is done through a merging method as described in **??** where the general design of the merging process is described. The process is a sort of design pattern that can be implemented for generation of terrain with a verity of attributes, as long as it is reasonable that such determined as a result through use of some form weighted merging of values.

5.5.1 R-tree indexing

For the R-tree implementation in this work, I am using the java library "rtree2" which provides a 2D implementation of the R-tree data structure in memory, including the R* tree [19].

5.5.2 Merger

The merging process depends on the implementation of a merger, as described in ?? and referred to previously in ?? with the description of the contribute method.

The merger should implement contrition methods for all the possibly implemented attributes. These contribution methods should expect the value and weight for the attribute being merged. Subsequently the merger should also implement a method for querying the merged results for all the possibly implemented attributes.

A generic example of the interface to be implemented for these two methods are:

1. `contributeSomeAttribute(weight, <priority>, value)`
   The `contributeSomeAttribute` method is called from the `contribute` method of a corresponding attribute type, where the value and weight are determined accordingly. The method performs the task of organizing the contributed values such that the results can be queried after the end of the merging process. The exact strategy of how an attribute is merged will be an implementation detail specific to the attribute, and `contributeSomeAttribute` should be considered in combination with `getSomeAttribute`.

   `weight` is the weight with which the value should be merged into the results.

   `priority` is an optional value which may be implemented use when for merging the results.

   `value` is the value of the terrain attribute to be merged into the results.

   The method returns no value.

2. `getSomeAttribute()` The `getSomeAttribute` method is called after all features has contributed their attributes. The method is then used to queries the merger for the merged results of the given attribute. The exact strategy of how an attribute is merged will be an implementation detail specific to the attribute, and `getSomeAttribute` should be considered in combination with `contributeSomeAttribute`.

   The method returns the merged result for the attribute.

5.5.3 Elevation merging

The elevation attribute is used to describe the terrain's elevation. It is merged through creating a simple weighted average of elevation values contributed by features. This is done by first summing the elevation and weight values during the contribution stage of the evaluation. After the contribution stage is finished, the terrain elevation can be queried by dividing the elevation sum with the weight sum.

The merger contribution method for the elevation attribute follows the pattern describe in 5.5.2 with the interface `contributeElevation(weight, elevation)`, and is implemented as described in Algorithm 3.

```
Algorithm 3: Contribute elevation
  input  : $W$ is the weight with which the elevation should be merged.
           $E$ is the elevation of the elevation attribute contributing to the merge.
  output:
1 begin
2     Let $S_w$ be a weight sum, and $S_e$ be a elevation sum, both persistent to
        the merger instance and with initial value zero.
3     Add weight $W$ to the weight sum $S_w$.
4     Add elevation $E$ to the elevation sum $S_e$.
5 end
```

The merger query method for the elevation attribute follows the pattern describe in 5.5.2 with the interface `getElevation()`, and is implemented as described in Algorithm 4.

```
Algorithm 4: Get elevation
  input  :
  output:  $E$ is the elevation.
1 begin
2     Let $S_w$ be a weight sum, and $S_e$ be a elevation sum, both are the same
        as in Algorithm 3.
3     Set the merged elevation $E$ to be the weighted average $\frac{S_e}{S_w}$.
4 end
```

5.5.4 Elevation merging w/ priority

An extension of the elevation attribute is the addition of prioritized merging. This introduces an additional stage to the merging where it's possible to ensure some features of the terrain have the possibility of overriding elevation values where they occur.

This is done by creating a series of prioritized average merges and linearly interpolating in steps from the lowest priority average merge to the highest.

The merger contribution method for the height attribute with priority follows the pattern describe in 5.5.2 with the interface `contributeElevation(weight, priority, elevation)`, and is implemented as described in Algorithm 5.

The merger query method for the elevation attribute with priority follows the pattern describe in 5.5.2 with the interface `getElevation()`, and is implemented as described in Algorithm 6.

```
Algorithm 5: Contribute elevation w/ priority
input  : $W$ is the weight with which the elevation should be merged.
         $P$ is the priority with which the elevation should be merged.
         $E$ is the elevation of the elevation attribute contributing to the
         merge.
output:
1 begin
2     Let $S$ be a list of tuples where the index of the tuple equals the priority.
      The tuple is made of the values are: $S_w$ a weight sum, and $S_e$, an
      elevation sum. $S$ is persistent to the merger, and all the tuples have
      both values initialized to zero.
3     Add weight $W$ to the weight sum $S_w$ of the tuple at the index equal to
      the priority $P$.
4     Add elevation $E$ to the elevation sum $S_e$ of the tuple at the index equal
      to the priority $P$.
5 end
```

```
Algorithm 6: Get elevation w/ priority
input  :
output: $E$ is the elevation.
1 begin
2     Let $S$ be the same list of $S_w$, and $S_e$ as in Algorithm 5.
3     Set the merged height $E$ to be the weighted average $\frac{S_e}{S_w}$ of the lowest
      priority.
4     foreach tuple $(S_w, S_e) \in S$ do
5         Let $E_p$ be the weighted average $\frac{S_e}{S_w}$ of current priority.
6         Set $E$ to be the linear interpolation between $E$ and $E_p$ according to
          $S_w$ of the current priority, clamped between 0 to 1.
7     end
8 end
```

5.5.5 Water depth merging

The water depth attribute is used to describe the depth of a river or lake in the terrain. This is merged similarly to how the terrain elevation is merged in 5.5.3, through a weighted average of depth values.

As with the terrain elevation merging, this is done by first summing the depth and weight values during the contribution stage of the evaluation. After the contribution stage is finished, the water depth can be queried by dividing the depth sum with the weight sum.

The merger contribution method for the water depth attribute follows the pattern describe in 5.5.2 with the interface `contributeWaterDepth(weight, depth)`, and is implemented as described in Algorithm 7.

Algorithm 7: Contribute water depth

```
   input : $W$ is the weight with which the elevation should be merged.
           $D$ is the depth of the water depth attribute contributed to the
   merge.
   output:
1  begin
2      Let $S_w$ be a weight sum, and $S_d$ be a depth sum, both persistent to the
         merger instance and with initial value zero.
3      Add weight $W$ to the weight sum $S_w$.
4      Add depth $D$ to the depth sum $S_d$.
5  end
```

The merger query method for the height attribute follows the pattern describe in 5.5.2 with the interface `getWaterDepth()`, and is implemented as described in Algorithm 8.

Algorithm 8: Get water depth

```
   input :
   output: $D$ is the water depth.
1  begin
2      Let $S_w$ be the weight sum, and $S_d$ be the depth sum, both are the same
         as in Algorithm 7.
3      Set the merged water depth $D$ to be the weighted average $\frac{S_d}{S_w}$.
4  end
```

5.5.6 Surface material merging

The surface material attribute is used to describe the material composition of the terrain surface. The merging of the surface material attribute may depending on the usage either result in a discrete material as an output, or a material fraction of multiple materials in a mix. As the assumption for the surface material attribute in this work, is that any single surface material attribute assignment to a feature in the skeleton layout can only define one material to contribute, the contribution method for merging surface materials is the same, whether the output is a discrete material or a material fraction. The querying method though is different for discrete or fractional surface materials outputs.

The merger contribution method for the surface material attribute follows the pattern describe in 5.5.2 with the interface `contributeMaterial(weight, material)`, and is implemented as described in Algorithm 9.

Algorithm 9: Contribute material

```
input  : $W$ is the weight with which the elevation should be merged.
         $M$ is the material of the surface material attribute contributing to
         the merge.
output:
1  begin
2      Let $S$ be a map with materials as key and weight as value, persistent to
         the merger.
3      Add $W$ to the value of the map entry with key $M$, if there is no entry
         for the key yet, assume the value for the key is zero, and set the the
         value value for key $M$ to be this new sum.
4  end
```

The merger query method for the surface material attribute follows the pattern describe in 5.5.2 with the interface `getMaterial()`, but depending on wether a discrete material or a fractional material mix is wanted, the implementations are respectively described in Algorithm 10 and Algorithm 11.

Algorithm 10: Get discrete material

```
input  :
output: $M$ is the surface material.
1  begin
2      Let $S$ be a the same map with materials as key and weight as value as in
         Algorithm 9.
3      Let $W_m$ be the maximum weight encountered while iterating through $S$,
         initial value is zero.
4      foreach material key $M_k \in S$ do
5          if The weight value $S(M_k) > W_m$ then
6              Set $M$ to be $M_k$.
7              Update the maximum weight $W_m$ to be $S(M_k)$.
8          end
9      end
10 end
```

When merging discrete materials into a fractional material, the merged result consists of a list of materials and their relative portion in the mix. This is in general similar to the internal organization of the merged material data, but converted from a map to a list of tuples with their weight normalizes according to the complete material mix.

Algorithm 11: Get fractional material

```
input  :
output: $M_m$ is a material mix in the form of a list of tuples $(M, F)$ of
        material and fractional portion.
1 begin
2     Let $S$ be a the same map with materials as key and weight as value as in
      Algorithm 9.
3     Let $W_s$ be the weight sum for all material weights $W_m \in S$, this is used
      to normalize the fractional mix.
4     foreach material key $M_k \in S$ do
5         Let $W_k$ be the weight of $M_k$ given the value in $S$ with $M_k$ as key.
6         Add the tuple $(M_k, \frac{W_k}{W_s})$ to $M_m$.
7     end
8 end
```

Chapter 6

Results

6.1 Generator configurations

For the presentation of the results, I will be using two generator configurations. The first configuration focuses only on the generation of mountain ridges and rivers, while the second is a simple alteration of the configuration that also generates lakes.

6.1.1 Ridges and rivers

The generated skeleton layout for this configuration is seen in Figure 6.1, and the complete configuration within WS is seen in Figure 6.2 with annotation for groups of modules performing specific task.

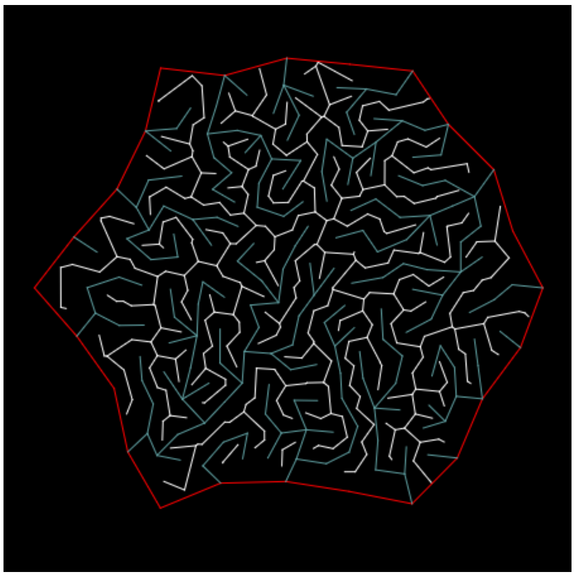


Figure 6.1: The ”Ridges and rivers” skeleton layout. The annotation colors in the skeleton layout are as follows: Red is the landmass outline, cyan is rivers and white are mountain ridges.

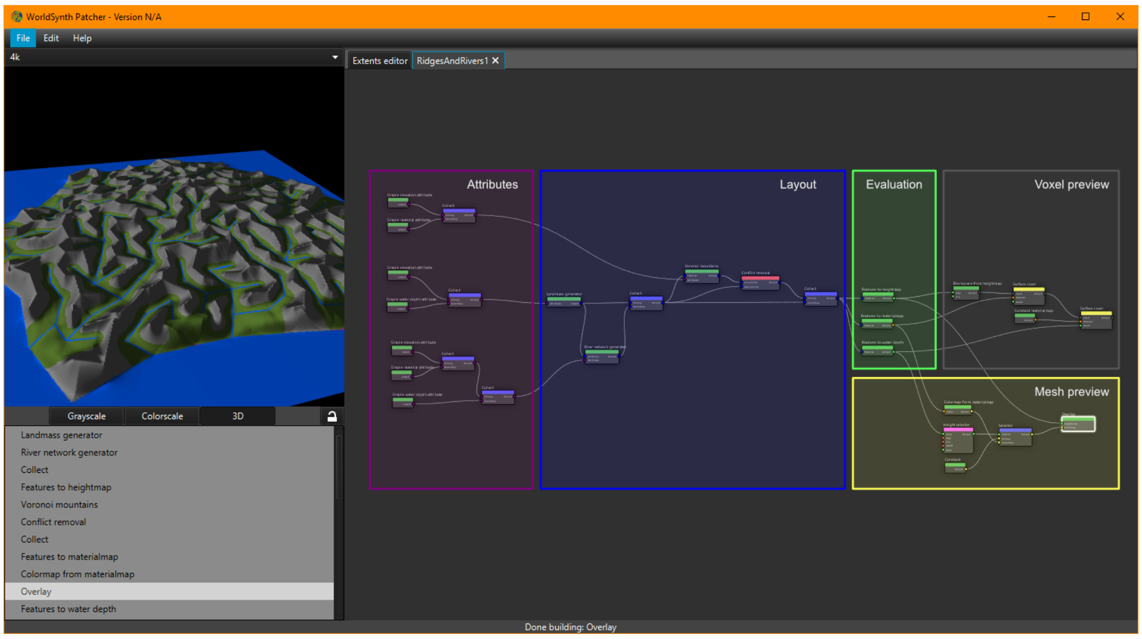


Figure 6.2: The "Ridges and rivers" WS configuration. The highlighted areas groups modules used to configure different aspects of the terrain generation, like configuring assigned feature attributes and feature layout operations.

In the configuration, there can be seen that there are three evaluation modules for the respective terrain attributes: elevation, surface material and water depth. Due to limitations of how WS works, the evaluation needs to be performed three times, once for each attribute to produce a map of. But this triple evaluation is only necessary because of WS. The evaluation process evaluates all the attributes in the same operation, each of these modules just output the results for one of the attributes at the time.

The final generated results of the configuration may either be viewed as voxels or as a colored mesh. The colored mesh is based on the evaluated elevation, with coloring of the surface materials applied as texturing. Where the water depth is more than zero,the color is set to blue to indicate water, this only indicates water and there is no geometry for the depth of the the water. The voxel results on the other hand involve the surface being carved out for the water as voxels with the appropriate material and depth for the water replaces the surface.

To the left in the configuration can be seen the attribute configurations used for the three different features in the skeleton layout. 1) The top group are the attributes used for the mountain ridges. The set of attributes created for this is contains an elevation attribute, and a surface material attribute. 2) The middle group are the attributes used for the landmass outline, and are used to define the water beyond the landmass. The attribute set for this involves an elevation attribute, and and a water depth attribute. 3) The last group to the bottom are the attributes configuration for the rivers. This attributes set involves all the current attributes, being elevation, surface material, and water depth attributes.

Collections of these attributes sets are provided as inputs to their respective feature layout generators, where they are assigned to all the features generated by their respective layout generator.

6.1.2 Ridges, rivers and lakes

The second configuration produces terrain that includes mountain ridges, rivers and lakes. This configuration only exchanges the river layout generator module, for the lake and river layout generator module from the previous work.

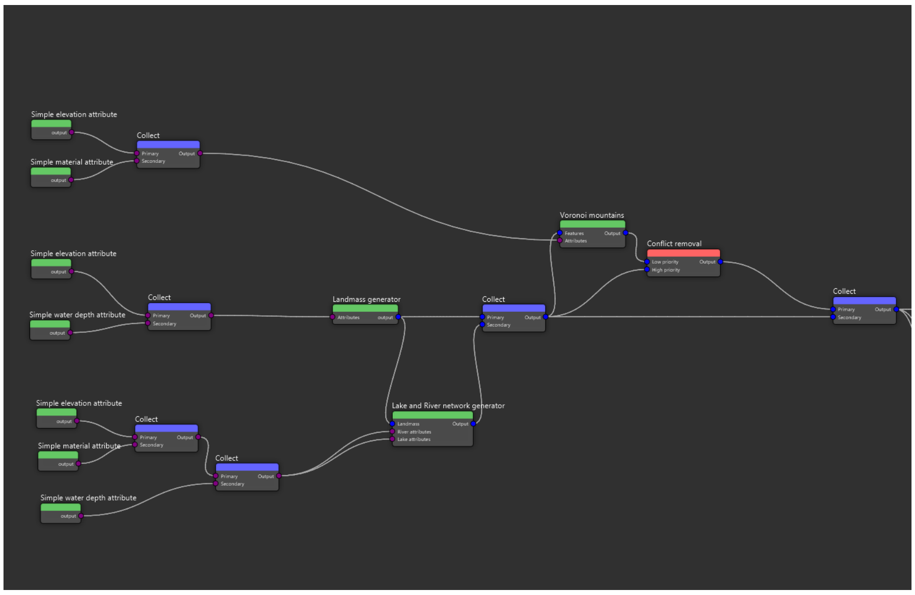


Figure 6.3: The WS configuration of used for the "Ridges, rivers and lakes" example.

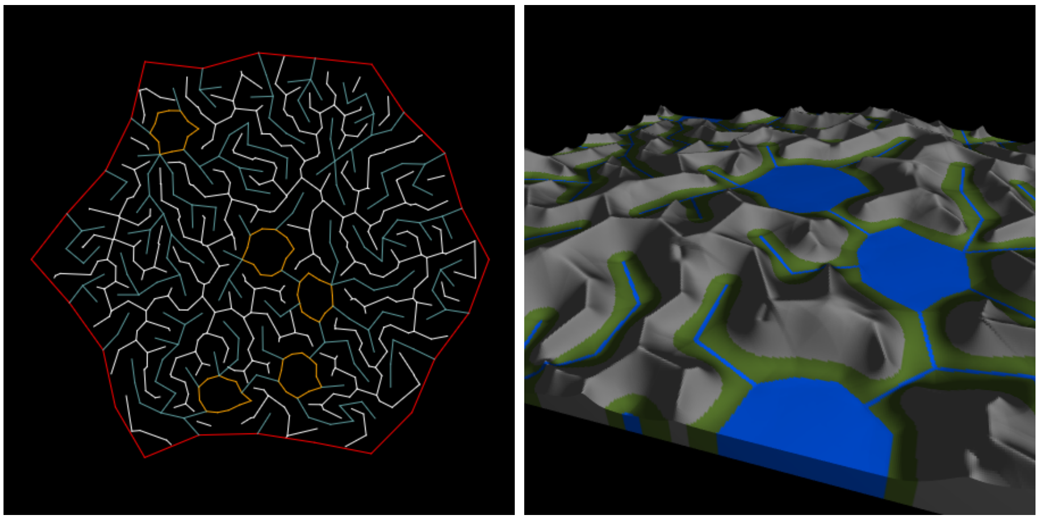


Figure 6.4: "Ridges, rivers and lakes" skeleton layout and 2k extent results. The addition of the lakes are annotated by orange color in the layout.

6.2 Improved evaluation efficiency

To show the difference the selective evaluation makes to the efficiency of evaluation, an extent of the "Ridges and rivers" configuration has been evaluated using three different methods to be compared. The methods are as following:

1) No optimization. This is the same method as the previous work, where all features are taken into account during the evaluation of a point, whether they are in range to have an influence on the terrain or not. This means the evaluation of any point in the terrain evaluates every feature in the layout for its contribution, even tough these for the most part may be zero.

2) R-Tree with querying for single points. This is the ideal way the new improvement is intended to be used. This involves each point in the terrain being evaluated is queried for the features with influence bounding rectangles that overlap with the point being evaluated. While this query may still return features that have no contribution due to the bounding box not being a perfect model for the influence bound, but rather an upper bound in each of the four directions, it should still be expected to drastically reduce the number of number of features evaluated.

3) T-Tree with grouped points. This is a hybrid of the first two methods that will evaluate a group of points in one go, utilizing an initial query that covers all these points in the group. The reasoning behind this method is that points relatively closely clustered mostly share the same features being in influence range, and that they thus will have mostly the same answers to their queries of the R-Tree. Depending on the time it takes to perform an R-Tree query in relation to the time it takes to evaluate features, a sufficiently high ratio of high query time to low feature evaluation time, may possibly have an advantage by evaluating clustered points if this is applicable, like generation of chunked terrain. The attempt of this method is therefore of interest to get an indication of whether a bottleneck in the terrain evaluation is in feature querying or feature evaluation.

These methods are applied to produce a map within a 2048x2048 centered extent, hereafter called the "2k" extent. The sampling resolution is 512x512 points, for a total sample set of 262144 evaluated points in a regular grid. The results are organized in Table 6.1.

The "Ridges and rivers" configuration also has a polygonal landmass feature, which is used to define the landmass outline and generate ocean beyond and therefor has an infinite range. The infinite range makes makes it so the feature is always include in the evaluation. Because the polygon also is relatively more intensive to evaluate than single lines, the same set of comparisons on generating the map are also performed to the configuration with the landmass feature excluded from the evaluation, and the results are organized in Table 6.2.

Since the implementation in this work is produced within the Java programming language, it is worth taking note of Java being a language that utilizes garbage collecting that may run sporadically and with varying duration. The time measurements in these results do not tack the time spent performing garbage collection

"Ridges and rivers"

Evaluation type	Features			Time	
	Layout	Evaluated		Evaluation	
	Total	Total	Avg. point	Total	Avg. point
No optimization	529	138674176	529	13574ms	0.0518ms
R-Tree single	529	4286801	16.3528	1185ms	0.0045ms
R-Tree group 16x16	529	5372160	20.4932	1356ms	0.0052ms

Table 6.1: Evaluation performance results for the "Ridges and rivers" configuration using the different evaluation methods.

"Ridges and rivers" excluding landmass

Evaluation type	Features			Time	
	Layout	Evaluated		Evaluation	
	Total	Total	Avg. point	Total	Avg. point
No optimization	528	138412032	528	13380ms	0.0510ms
R-Tree single	528	4024657	15.3528	873ms	0.0034ms
R-Tree group 16x16	528	5110016	19.4932	1042ms	0.0040ms

Table 6.2: Evaluation performance results for the "Ridges and rivers" configuration, excluding the landmass polygon from evaluation, using the different evaluation methods.

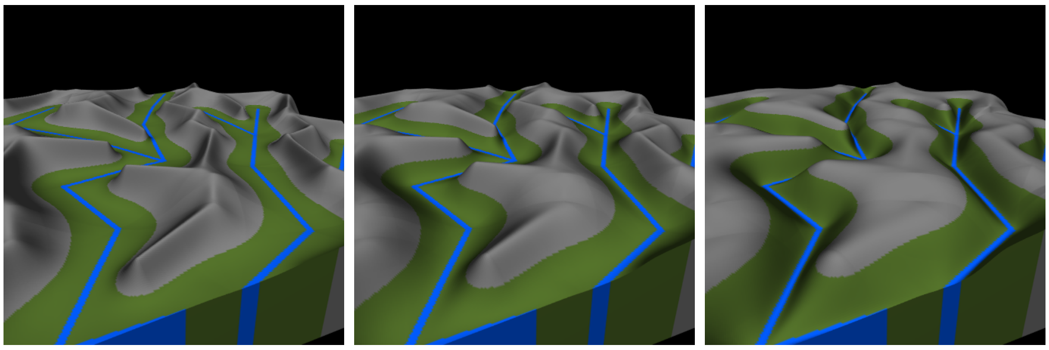
during the evaluation as it is assumed to be negligible in relation to the overall time, but this may be a possible source to some level of variation in evaluation times.

6.3 Modelling of features using customized attributes

The usage of attributes allows for the inclusion of several terrain attributes by assignment to any skeleton features. In this implementation they are limited to elevation, surface material and water depth attributes, but these may be freely assigned during configuration of the generator operation.

The attributes assigned to the features are here used to map out values describing the terrain as an inherit part of the method. Where for example rivers in the previous work was produced by selecting the the subset of features in the skeleton layout that represented rivers and using a threshold on the distance and depth based on distance to any of these, this implementation with attributes system allows instead for any evaluated point to be queried for the depth value similarly to how elevation is acquired.

The implementation herein features the possibility for simple configuration of attributes and assignment to different features during the layout of the skeleton. An example of how attributes may be used to further stylize the results is shown by changing the weight function for rivers elevation, as this may be configured to be different than the weight function for mountain ridges. This allows not just for different ranges of influence to be applied for different features, but also for changing how the weights interact with each other. This is shown by the example in Figure 6.5, where the value b of weight function w_3 is changed to different values for the rivers in relation to the ridges to produce different characteristics for the resulting terrain where these features interact.



(a) $b = 20$ (b) $b = 10$ (c) $b = 5$

Figure 6.5: Elevation attributes with different b values for distance weight function w_3 for the river features. The b value is 8 for the mountain ridge line features.

The results of this it can be seen that a higher value of b produces wider and flat U-shaped valleys in contrast to an otherwise lower value of b result in sharper V-shaped valleys.

Chapter 7

Discussion

7.1 Improved evaluation efficiency

In the implementation from the previous work, the time it took to evaluate the terrain from the skeleton layout was directly tied to the complexity of the skeleton layout in how many features it involved [1]. As scaling up to a larger layout with more features, the evaluation time would quickly increase with it.

To address this efficiency issue, features are now expected to have a bounded area of influence, and the evaluation is selective of which features to include when evaluating any point or area. A MBR enclosing the area of influence is created for all the features in an evaluated skeleton, and they are indexed in an R-Tree that may be queried for all the features with MBRs intersecting the point or area to be evaluated.

It is clear from the results that the selective evaluation dramatically decreases the number of evaluated features per point. The "Ridges and rivers" configuration used to test generates a total of 529 features, whereas the number of features used to evaluate the terrain, was on average just a little over 16 for any single point. Previously without the selective evaluation, the evaluation would have used all the 529 features.

This is a reduction to using around every $\frac{1}{31}$ features, while the corresponding evaluation time is decreases to around $\frac{1}{11}$. When looking at the same results excluding the landmass polygon, this time decreases to around $\frac{1}{15}$ because polygons are a relatively more intensive shape to evaluate than single lines.

But an ideal reduction of the evaluation time similarly the reduction in evaluated features is not achievable, as searching of the R-tree is relatively slower than iterating through a list.

With larger skeleton layouts containing more features, the selective evaluation is expected to be mostly constant for the same feature density, and thus becomes increasingly more efficient than the non-selective method given the quadratic increase in features.

The grouping of evaluated points also did not provide any advantage over evaluating single points. This indicates that the evaluation of a few extra features is more expensive than querying the R-tree for for each point.

7.2 Attributes

The introduction of attributes has has enabled the same evaluation process to produce additional values describing the terrain besides the original elevation value. The resulting values for any attribute of the terrain are available from the merger after evaluation, and are easy to use without further modifications.

The attributes serve as a flexible tool for further modelling unique classes of features that goes beyond the emergence resulting from context and intent, as was introduced in the previous work, and still this is without any need for hard coding as attributes can be freely assigned as part of generating the skeleton layout by configuration as needed.

The river feature is an example of how how assignment of differing attribute sets may alter a feature. While both the ridges and rivers features have elevation and material attributes assigned, but with differing values, the river feature in addition also has a water depth attribute. The elevation and location arguably define these features as ridges and rivers if following the conclusions from the previous work. But the usage of the water depth attribute here in this work, is also a defining property of the river feature, as without the creation of the river, the feature may better be called for a valley feature if it does not carve out the river.

Meanwhile the attribute contribution is decided by distance through the weight function. For the evaluation of a feature at any point, the distance only needs to be evaluated once, and is reused for the contribution of all assigned attributes. Thus performance impact of assigning any arbitrary set of multiple attributes, is therefor dependent on the weight function, value function and merge function of the assigned attributes.

Further new attribute types are also reasonably simple to create, as the method provides a framework for implementing and handling additional attributes, with the existing attributes from this work serving as examples for how different attributes may be implemented.

7.3 Local weight functions

The local weight functions, defined as part of the attributes assigned to features, adds to the flexibility of uniquely modelling features by configuration. As the weight function properties of the feature may differ between attributes, a feature may contribute to differing sets of attributes depending on the position of an evaluated point relative to the feature geometry.

This is seen by example in the river feature, where the range of influence for the water depth is significantly smaller than the range of influence for the elevation and material.

When considering the impact of such weight functions differing between attributes, it is important to recognize that the bounding area the feature is indexed by is based on the largest range resulting from the weight functions of a feature's assigned attributes. Therefore the addition of one or more attributes with a shorter range than at least one other assigned attribute, does not alter the area for which the feature is evaluated.

As was seen in the results, the usage of differing weight functions between features may have an important impact on the results, as the example with the elevation attribute showed, it may be applied to produce differing characteristic for the terrain, like U- or V-shaped valleys.

Part II

Detailing of Features

Chapter 8

Design

8.1 More features

As from the previous work, the usage of straight lines as shapes of features in the skeleton layout produces visibly straight lines in the finished evaluated results [1]. As the occurrence of continuous straight lines in nature is limited, and instead tends to form fractal shapes and outlines, it is desirable to model procedural environments to also feature such fractal details.

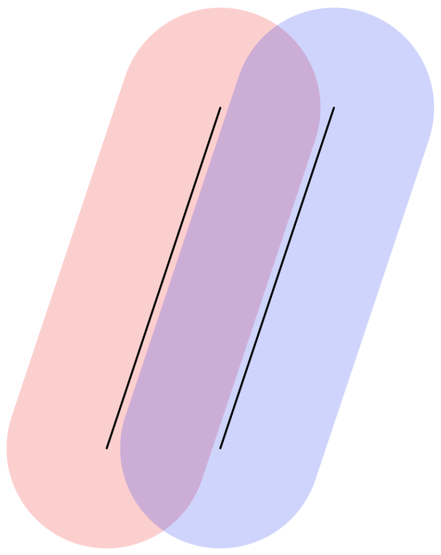
The possibly simplest and most obvious approach to creating such details, is in the feature layout process by creating a more detailed layout with more and smaller features. A method to create such features from existing features, is to use MPD on features in the skeleton layout to produce several new and smaller features replacing the originals.

For example features with a line shape that represent a river segments, may have MPD applied to them, such that the segments resulting form the subdivision are used as new river features replacing the original features, and together defines the same river paths with higher detail, see Figure 8.1a and Figure 8.1b.

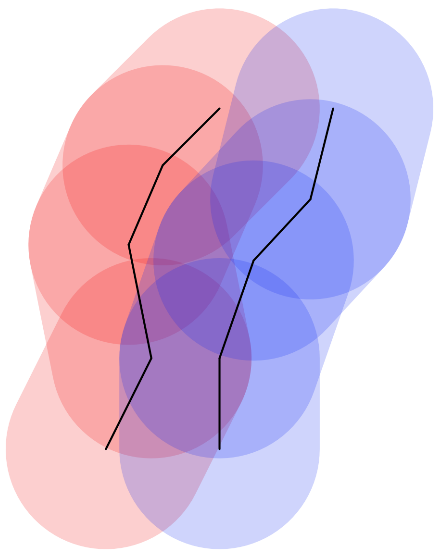
This method by itself is simple and requires no other new addition to the method other than a function that can perform MPD on a set of features. Out of the implemented shapes available, it only makes sense to perform MPD on shapes like lines and polygons, but since this approach also produces separate new features, it is not applicable as more detailed polygons can not simply be split into multiple shapes.

Though this method is simple in application at first sight, there are expectable issues that should be taken into account when applying this method for creating more features for a high detail skeleton from a low detail skeleton layout. One of them is already mentioned as it not being directly applicable to polygons.

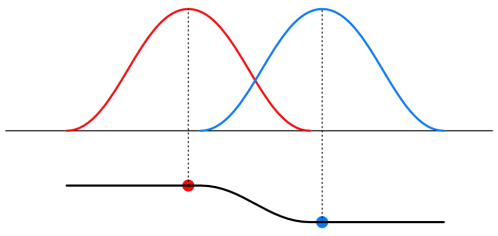
Another expected issue with this method, is that this creation of fractal details by more features, should be expected to give a quadratic increase in the number of features that needs to be evaluated. This can be seen as the number of features



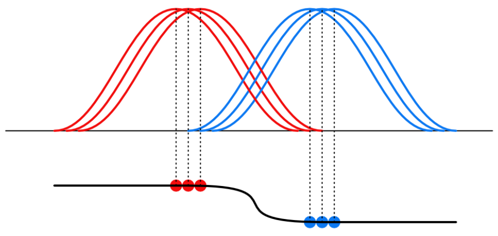
(a) Single feature with a line shape.



(b) Four smaller line features resulting from applied MPD.



(c) Weight and transition from single features making up the detail.



(d) Weight and transition from grouped similar features making up the detail.

Figure 8.1: Transition being distorted by groupings of features changing wight proportions in the transitioning area.

replacing the original layout will be increased by a factor of 2^n where n is the number of iterations of MPD applied.

With an initial assumption that the newly created features from the MPD being applied replaces the original low detail features, the weight function is kept the same for the new collection of high detail features as for the features they where created from, there may arise several possible issues that needs to be addressed. 1) This dramatic increase in new features should be expected to increase the evaluation time accordingly, as the evaluation time is directly related to the number of features that need to be evaluated to acquire the results for any point. 2) The creation of dense groups of features may affect the overall transitions results between such groups of features, where the tightly packed groups may make up proportionally increasing weight in the area close to the respectively dense groupings, causing the values of the grouping to become more pronounced in the general area, see Figure 8.1.

8.2 Overlapping features

To address some of these issues related to evaluation complexity from 8.1, a solution may be the use of both the low and high detail features, such that high detail features define details relatively locally with shorted weight distance, while the low detail features that where the starting point are still maintained with their original weighting to provide the general shape of the transition area.

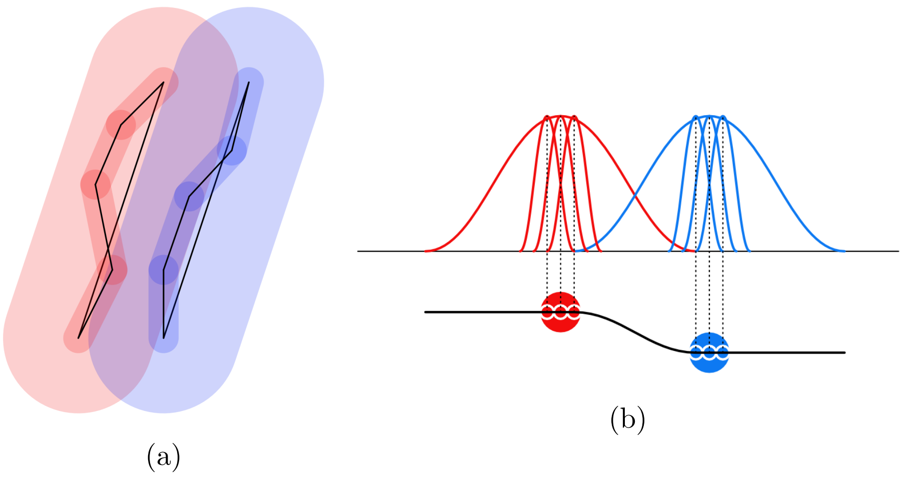


Figure 8.2: High detail features with smaller area of influence overlapping with low detail base feature may be used for adding details with a more limited impact an higher efficiency than just high detail features.

In practice this will mean that the resulting features from the applied MPD operation is given a smaller range of influence, to only impact the results within the area that is well defined already by the original low detail feature, in a sense overlapping with the original feature where more detail is created. High detail group from different features should thus not in practice have overlapping areas of influence. The area of influence for such detailed features should in practice be determined in relation to the scale of the clusters of detail features that have large overlaps, which may also create distortions of the details. The reduction of the area of influence of high density features also helps reducing the expected number of features that need to be evaluated to produce evaluated results from the skeleton model.

Also in addition to the possibility of this working with pure weighted average merging, this is also has a good potential for use with priority merging, where detail features have attributes with higher priority than the base low detail feature such that the detail takes over control of the evaluation results completely when approaching.

This approach is simple to approach and relies may relay on already existing systems with only the addition of an operation to generate new features from existing features sets by applying MPD. In addition this method may also be used make progressively higher detail features that perform as a a form of LOD for evaluation of features with a fractal nature.

One issue with the application of MPD to create more details in the layout phase, is that the evaluation time of polygons is directly dependent on how many vertexes they are made up of, and the progressive approach of applying a smaller range of influence to more detailed polygons does not help resolve this issue as the whole polygon needs to be evaluated as one unit either way. The progressive use of multiple polygons with different detail level is rather likely to instead produce problems, both by increasing the evaluation time for evaluation versions of the same polygon at different detail level, and may also be likely to produce artefacts thanks to the polygon having both an inside and outside area, and new details generated such that they fall inside the low detail polygon may cause artefacts when merging is performed, though this would already be a moot point thanks to the previously mentioned reason of performance issues evaluating detailed polygons.

8.3 Fractal shapes

Lastly, the application of MPD may be used to alter the shape of a feature rather than producing a collection of new features. This method instead makes a more detailed shape for a feature, replacing the low detail shape with a higher detail polyline or polygon than the original of the input.

This introduces the polyline shape, as the polygon shape already can support a higher detail level in contrast to a line segment shape. The polyline is made up of more than two vertices that connect together in a chain of line segments. The distance to such a polyline is calculated by taking the minimum distance of all the line segments that it is made up by.

This method may be expected to produce the most correct results as we get a true distance function for a more complex shape, but it should also be expected to be more demanding to evaluate as the distance to all smaller line segments have to be calculated to acquire the results, and thus has a similar evaluation complexity comparable to or worse than the the method in 8.1.

In contrast to the previous methods though, this method works for both lines and polygons as well.

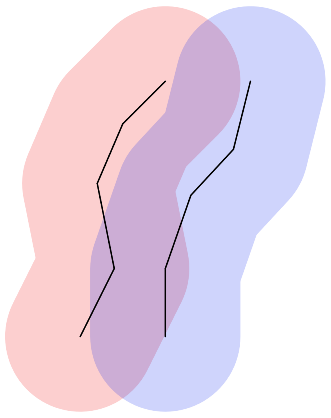
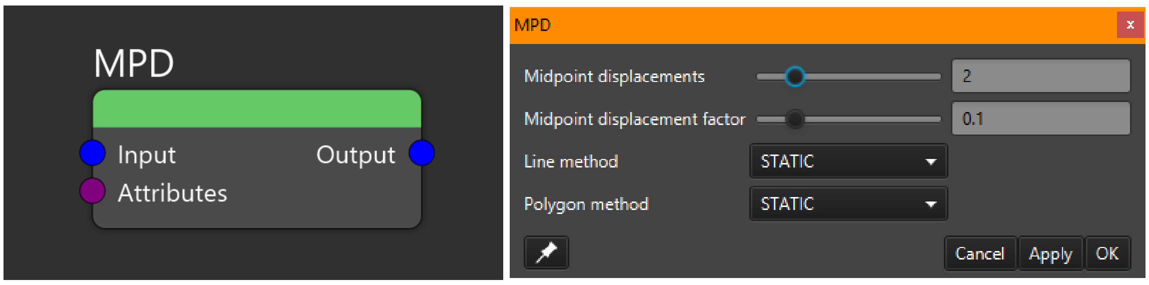


Figure 8.3: Fractal polylines are made up of multiple short line segments treated as part of the same line, instead of being split up into multiple discrete line.

Chapter 9

Implementation

9.1 MPD module



Midpoint displacements: 2
Midpoint displacement factor: 0.1
Line method: STATIC
Polygon method: STATIC

Figure 9.1: The MPD module and parameters shown in WS.

To create the additional details according to the outlined methods, it is necessary to apply MPD on existing features during the layout generation. For the implementation in WS, this is acomodated by the creation of a new module, with the purpose to MPD to existing features in the skeleton layout. This MPD module takes a collection of features as input to apply MPD on, as well as an optional collection of new attributes to be used as replacements in the output features. If no such attributes input is provided in the configuration, the attributes of the input features are carried on in their resulting features after MPD has been applied.

The module has two parameters for the MPD application, giving the iterations and displacement factor uses, as well as two more parameters for how the MPD results are used to create the output features. For respectively lines and polygons, these options are NONE, REMOVE, STATIC and SPLIT, where SPLIT is only an option for lines. The NONE option performs no MPD on shapes of the given type, and outputs a feature with the same shape as result. The REMOVE option applies no MPD and instead removes a feature with the given shape from the output. The static STATIC option creates a higher detail shape from applying MPD to the respective shape, a line result in a polyline, while a polygon results in a more detailed polygon. The SPLIT option available for lines, splits the would be polyline into individual simple line segments that form individual features.

The pseudo code for applying MPD and generating the output features is described in Algorithm 12, with details of how MPD is applied in line 4, for respectively lines and polygons, being detailed in 9.1.1 and 9.1.2.

Algorithm 12: Features MPD

```
input  : Iterations of MPD to apply $n$.
         Displacement factor $d$.
         A collection of features $F_i$ to apply MPD to.
         A collection of attributes $A$ to be assign to the new features.
output: A collection of the new features $F_o$.
1  begin
2      foreach Feature $f_i \in F_i$ do
3          if Shape $S_{fi}$ of $f_i$ is applicable for MPD then
4              Apply MPD to $S_{fi}$ according to the shape method, $n$ and $d$. This
                 may produce from none to multiple shapes in a collection $S$.
5              foreach Shape of a new feature $S_{fo} \in S$ do
6                  if Input $A$ provides no attributes then
7                      Create a new feature $F_n$ with shape $S_{fo}$ and attributes
                         copied from $f_i$, add to $F_o$.
8                  end
9                  else
10                     Create a new feature $F_n$ with shape $S_{fo}$ and attributes $A$,
                         add to $F_o$.
11                 end
12             end
13         end
14     end
15 end
```

9.1.1 Line MPD

When the line method is set to NONE or REMOVE, the returned value is the collection S either containing the input line or being empty, either way no iterations of MPD are applied for these. When the line method is set to either STATIC or SPLIT, a line shape will have MPD applied and then produce either a simple polyline or several separate line segments in the collection S.

Both methods follow the same initial steps by applying n iterations of MPD, before either a polyline or several split lines are created from the results. The pseudo code for applying MPD to the input line is described in Algorithm 13, and the subsequent operations for creating either split lines or a polyline shapes, is described in Algorithm **??** and **??**.

Algorithm 13: Line MPD

```
input : Iterations of MPD to apply $n$.
        Displacement factor $d$.
        A line shape $S_i$ to apply MPD to.
output: A list of new shapes $S_o$.
1  begin
2      Create a linked list $V$ with initial entries being the two vertices of $S_i$.
3      Hash the positions of the initial vertices and use it as seed for the
         random number generator.
4      for $i = 1$ to $n$ do
5          $j = 0$
6          while $j < V.size - 1$ do
7              Let $l$ describe a line connecting the vertex pair $(V[j], V[j+1])$.
8              Create a new vertex $v$ on the midpoint of $l$.
9              Let $l_l$ be the length of $l$.
10             Move $v$ normally to the direction of $l$ by a random value between
                 $\pm d \cdot l_l$.
11             Insert $v$ at $V[j+1]$, increasing the size of $V$.
12             $j = j + 2$
13         end
14     end
15     Creates shapes from $V$ and add to $S_o$ according to Algorithm ?? or ??.
16 end
```

Split line

The SPLIT method takes the vertices from the initial MPD application in Algorithm 13 and creates separate straight line segments between each pair of consecutive vertices. This uses the existing line shape and is a simple iterative loop of constructing lines and adding them to the shapes list.

Algorithm 14: Line MPD - Split

```
input : list of vertices $V$ from Algorithm 13.
output: The list $S_o$ containing the new lines.
1  begin
2      for $i = 1$ to $V.size - 1$ do
3          Create a line shape with the vertex pair $(V[j], V[i+1])$, add to $S_o$.
4      end
5  end
```

Polyline

Since the previous work does not already have a polyline shape implementation, this needs to be implemented. The implementation of a polyline is somewhat similar to the the implementation of a polygon, using a list of vertices that describe a series of connecting line segments, but without the last and first vertices being connected, and as such also has no inside.

The distance from the polyline shape is calculated by taking the minimum distance among all the line segments that make up the polyline.

The elevation of the polyline is calculated according to the straight line connecting the end points, since determining elevation according to the closest line segment of the polyline, will result in discontinuous transitions on the inner corner of connecting line segments.

The construction of the polyline simply applies the same list vertices created from Algorithm 13 as the vertices of the line.

Algorithm 15: Line MPD - Polyline

```
  input  : list of vertices $V$ from Algorithm 13.
  output: The list $S_o$ containing the polyline.
1 begin
2     Create a polyline shape with the vertices in $V$, add to $S_o$.
3 end
```

9.1.2 Polygon MPD

Algorithm 16: Polygon MPD

```
   input  : Iterations of MPD to apply $n$.
            Displacement factor $d$.
            A polygon shape $S_i$ to apply MPD to.
   output: A list containing a single polygon shape $S_o$.
1  begin
2      Create a linked list $V$ with initial entries being the vertices of $S_i$.
3      for $i = 1$ to $n$ do
4          $j = 0$
5          while $j < V.size$ do
6              Let $l$ describe a line connecting the vertex pair $(V[j], V[j+1])$,
                 where the $j + 1 == V.size$ overflows back to index 0.
7              Create a new vertex $v$ on the midpoint of $l$.
8              Let $l_l$ be the length of $l$.
9              Move $v$ normally to the direction of $l$ by a random value between
                 $\pm d \cdot l_l$.
10             Insert $v$ at $V[j + 1]$, increasing the size of $V$.
11             $j = j + 2$
12         end
13     end
14     Create a polygon shape with the vertices in $V$, add to $S_o$.
15 end
```

9.2 Details with more features

For this approach, there there needs to be implemented functionality for applying MPD to create new features from the existing ones. This is implemented as a new WS module that takes the input of the set of features for this to be applied to, and the attributes to apply to the set of newly created features, which in this case will have the input of the same attributes applied to the input set of features.

The method is applied in practice by inserting the MPD module from 9.1 after the initial creation of the features to enhance, in this case being the river and mountain ridge features. The attributes input is given the same attributes used for the initial features, and the new features created is replacing the original features for the collection used for evaluation.

For the creation of the original mountain features, the original river layout before MPD is used.

Figure 9.2 shows how the method is applied to the "Ridges and rivers 1" configuration from 6.1.1.

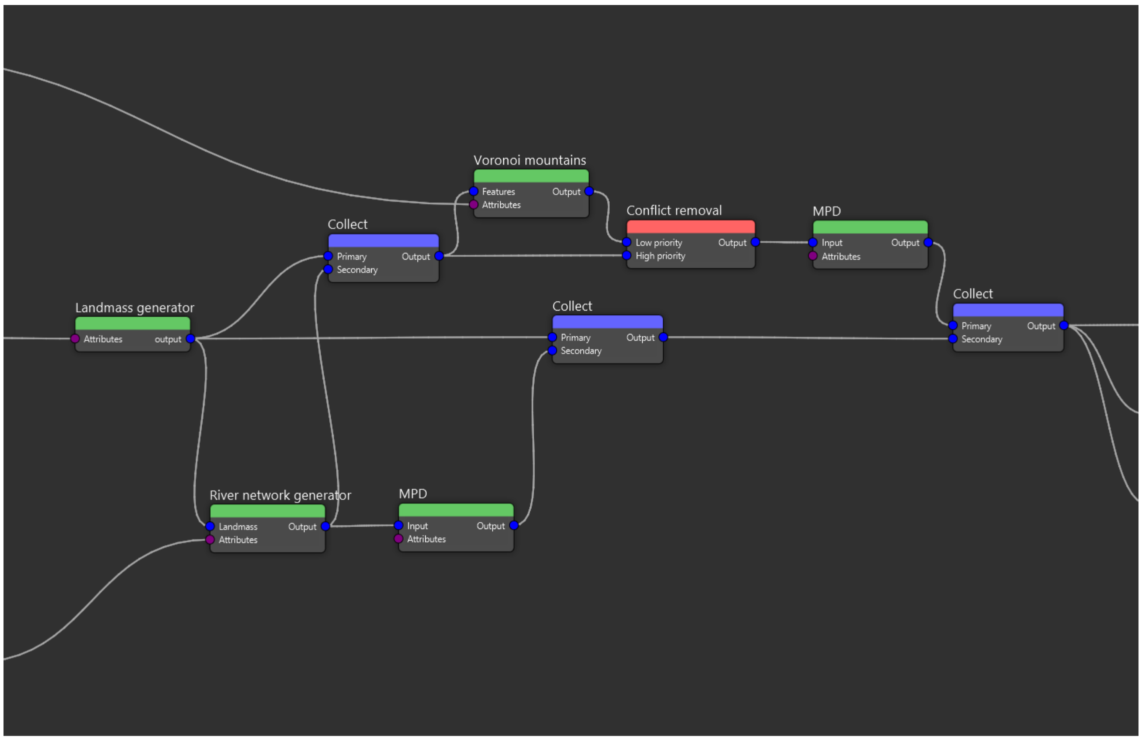
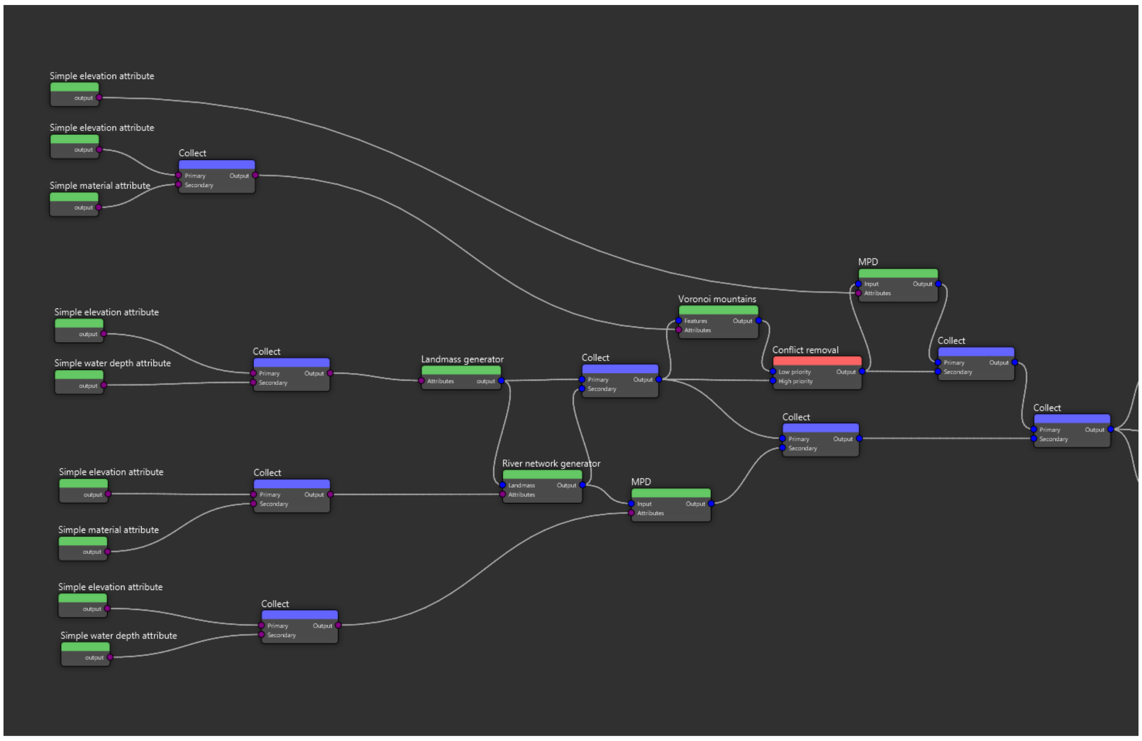


Figure 9.2: The WS layout configuration from "Ridges and rivers 1" in 6.1.1, modified to apply details with more features.

9.3 Details with overlapping features

For the approach with overlapping features, there is not needed any new modules as it instead relies on a different configuration of already implemented functionality.

Primarily this method is a further modification to the configuration described in **??**, where the output from the MPD module is combined with the input, using different sets of attributes with a shorter influence range and higher priority for the the more detailed output relative to the low detail original layout.



Simple elevation attribute
Simple elevation attribute
Collect
Simple material attribute
MPD
Voronoi mountains
Simple elevation attribute
Simple water depth attribute
Collect
Landmass generator
Collect
Conflict removal
Collect
Collect
Collect
Simple elevation attribute
Collect
River network generator
MPD
Simple material attribute
Simple elevation attribute
Collect
Simple water depth attribute

Figure 9.3: The WS layout configuration from the "Ridges and rivers 1" in 6.1.1, modified to apply details with overlapping features.

9.4 Details with fractal shapes

The application of this replaces the shapes of existing features with more detailed shapes resulting from applying MPD. The attributes of the features are carried over in the more detailed features without modification. In the practical implementation created in WS, the configuration is the same as for the method with more features from 9.2, seen in Figure 9.2, but with the MPD module application method parameter for lines and polygons set to the STATIC option. The configurations are otherwise exactly the same.

Chapter 10

Results

10.1 Method comparisons

To compare the differences between the three methods for adding details, the methods are applied to the "Ridges and rivers" configuration, each used to produce a map within the "2k" extent. The sampling resolution is 512x512 points, for a total sample set of 262144 evaluated points in a regular grid.

The amount of details to create is adjusted using the MPD modules, and there are used 3 iterations of MPD for the creation of the given results.

Both the visual and performance differences are compared for the three methods, along with the base configuration without detailing for reference. The resulting maps can be seen in Figure 10.1, and the performance results in Table 10.1.

Detailing method	Features			Time	
	Layout	Evaluated		Evaluation	
	Total	Total	Avg. point	Total	Avg. point
Reference base	529	4286801	16.3528	1185ms	0.0045ms
More features	4225	22370076	85.3351	4386ms	0.0167ms
Overlapping features	4753	5933890	22.6360	2206ms	0.0084ms
Fractal shapes	529	4313839	16.4560	1850ms	0.0071ms

Table 10.1: Evaluation performance for "Ridges and rivers" with 3 iterations of MPD.

Firstly When comparing these visual results of these methods, it is clearly visible in Figure 10.1c, that the the overlapping features method has an unfortunate tendency to create undesired artefacts in the form of double ridges and flat spots at the tops. The rivers in contrast does not bear these same artefacts of duality, or at the very least not visibly so.

The methods using more features or fractal shapes, seen in Figure 10.1b and 10.1d in contrast produce results that have a visibly similar profiles to the elevation transitions between the ridges and the rivers as the reference, but with the features following the desired, more detailed fractal path.

Comparing these two methods further, the method using more features generates a more jagged edge along the top of the ridges compared to the smooth elevation changes when using the fractal shapes method. Meanwhile the method using fractal shapes generate more crease details in the transition areas between the ridges and rivers, these creases following the path of these features being merged, where the method splitting these paths up into smaller pieces create a smoothed out transition.

Considering the evaluation performance of these two methods as seen in Table 10.1, it is clear that the method using more features is the least efficient approach, needing around twice the time to be evaluated than the two other approaches.

The method using fractal shapes gives results that are visually comparable or even better than the results when using more features, while also being the fastest one in these result.

10.1.1 "Ridges and rivers" with fractal shapes

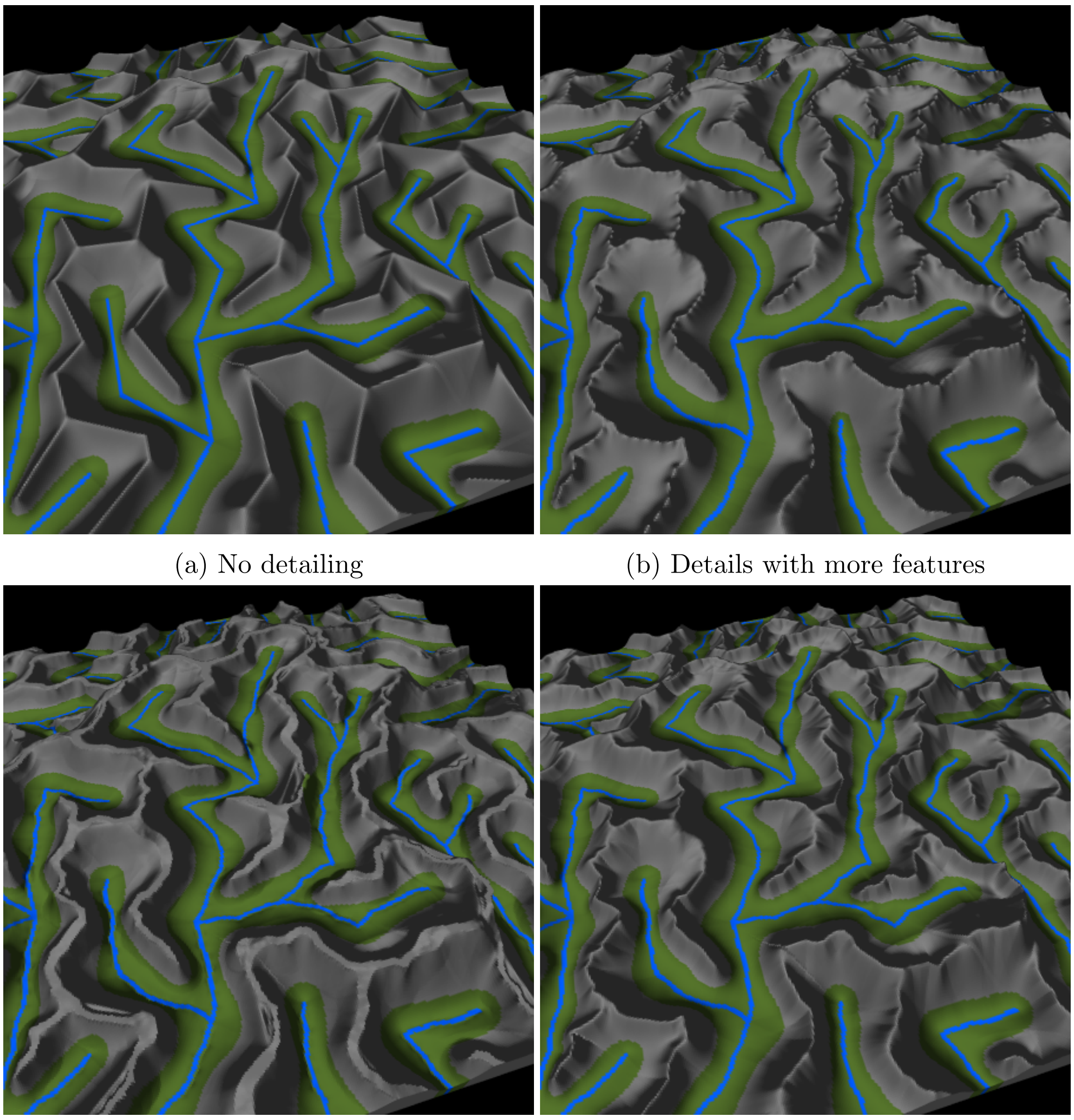
Since the method using fractal shapes has shown itself to be a favorable approach by brief comparison to the other methods, it is of interest to understand how the performance is impacted by creating an increasing amount of detail by applying further iterations of MPD.

To determine this, the configuration is kept the same, except for the iterations of MPD being applied by the MPD modules being changed, and the performance results are organized in Table 10.2, and the times plotted in Figure 10.2.

It is clear from the results that the evaluation time increases is quadratic, approaching a doubling for each increment in MPD iterations being applied. It can also be observed that the number of features evaluated slightly increases with more details.

Evaluation type	Features			Time	
	Layout	Evaluated		Evaluation	
	Total	Total	Avg. point	Total	Avg. point
Reference base	529	4286801	16.3528	1185ms	0.0045ms
MPD 1	529	4300907	16.4067	1321ms	0.0050ms
MPD 2	529	4307768	16.4328	1511ms	0.0058ms
MPD 4	529	4316499	16.4661	2549ms	0.0097ms
MPD 6	529	4318931	16.4754	6565ms	0.0250ms
MPD 8	529	4319200	16.4764	23353ms	0.0891ms

Table 10.2: Evaluation performance for "Ridges and rivers" using static evaluation of fractal shapes for details.


(a) No detailing (b) Details with more features

(c) Details with overlapping features (d) Details with fractal shapes

Figure 10.1: Results of the different approaches to adding details.

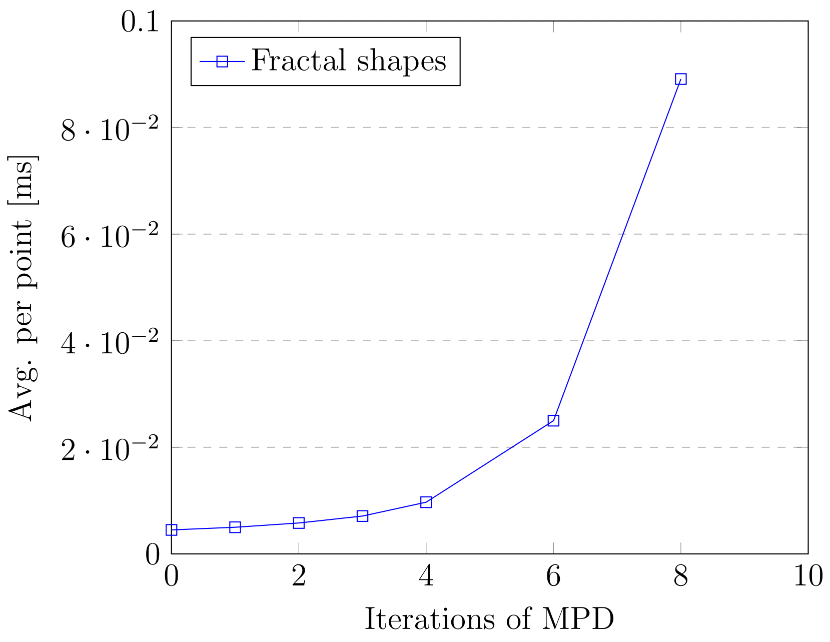


Figure 10.2: Evaluation times per point for "Ridges and rivers" with static fractal shapes.

10.1.2 "Ridges, rivers and lakes" with fractal shapes

The previous evaluation for different numbers of MPD iterations are also repeated for the "Ridges, rivers and lakes" configuration.

The MPD modules are similarly added to the configuration "Ridges, rivers and lakes" configuration as described 9.4.

In addition, there is added an added an additional elevation attribute to to the river and lake features. This new elevation attribute has a short influence range and higher priority, and is added to address the elevation of the water in lakes not being flat in the initial configuration, as the elevation was merged with surrounding mountains when approaching the lake edges. The finished configuration for the layout is seen in Figure 10.3.

Evaluation type	Features			Time	
	Layout	Evaluated		Evaluation	
	Total	Total	Avg. point	Total	Avg. point
Reference base	551	4200864	16.0250	1332ms	0.0051ms
MPD 1	551	4213833	16.0745	1502ms	0.0057ms
MPD 2	551	4223485	16.1113	1744ms	0.0067ms
MPD 4	551	4232776	16.1468	3296ms	0.0126ms
MPD 6	551	4237016	16.1629	9439ms	0.0360ms
MPD 8	551	4237493	16.1648	34735ms	0.1325ms

Table 10.3: Evaluation performance for "Ridges, rivers and lakes" using static evaluation of fractal shapes for details.

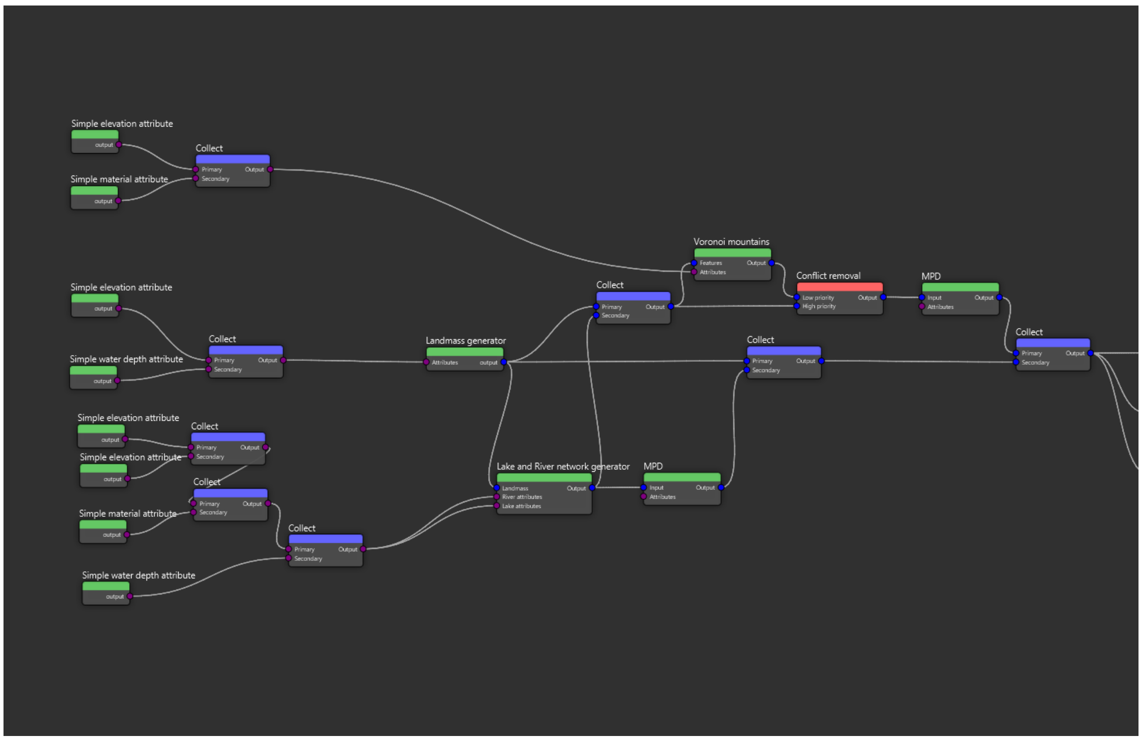
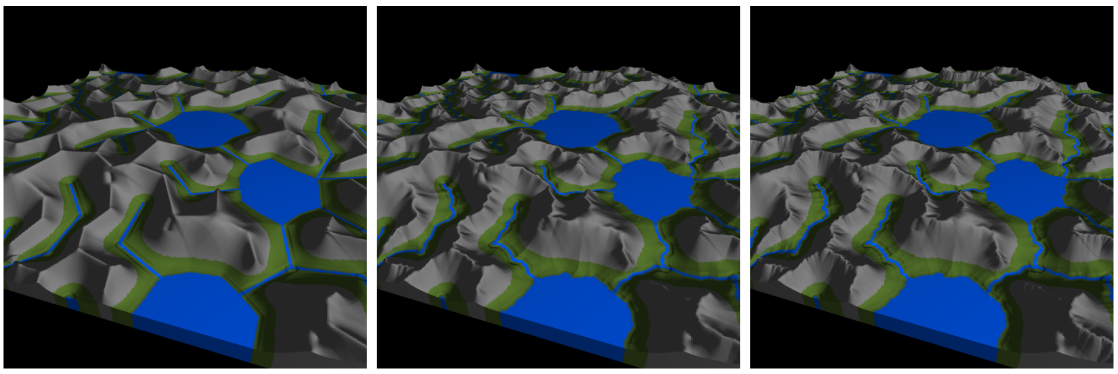

Simple elevation attribute
Simple material attribute
Collect
Simple elevation attribute
Simple water depth attribute
Collect
Landmass generator
Simple elevation attribute
Simple elevation attribute
Collect
Collect
Simple material attribute
Collect
Simple water depth attribute
Lake and River network generator
MPD
Collect
Voronoi mountains
Collect
Conflict removal
MPD
Collect

Figure 10.3: The WS layout configuration for ”Ridges, rivers and lakes” with fractal shapes.



(a) MPD 0 (Reference) (b) MPD 3 (c) MPD 6

Figure 10.4: Evaluation results for ”Ridges, rivers and lakes” with fractal shapes.

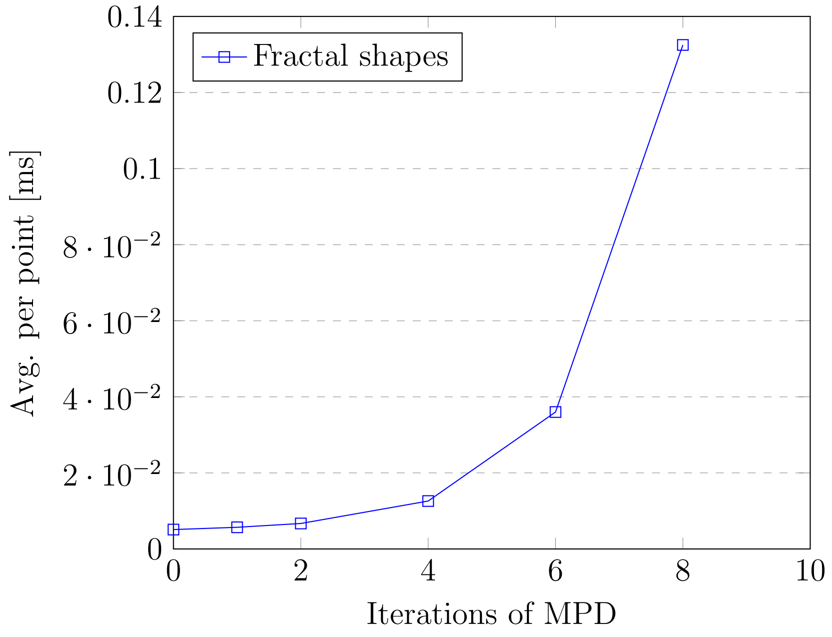


Figure 10.5: Evaluation times per point for ”Ridges, rivers and lakes” with static fractal shapes.

Comparing the performance results from this, the same pattern of quadratic growth to the evaluation time is observer, but with relatively longer evaluation times.

Chapter 11

Discussion

With the addition of the MPD module, it has been made possible to create more detailed layout by further refining an existing layout. The module has been used in three different method to generate more details.

Out of these three methods, the approach with using fractal shapes was shown to produce the best results the fastest, but the reasons why this might be, gives further insight on the different methods how the advantages of the methods may change on application.

11.1 More features

I will firstly address the simplest, and to some extent naive approach of applying MPD to the features and splitting them into multiple new features with the same attributes. While this method was shown to create visually pleasing results for the most part, with the exception of the jaggedness of the ridges that was not intended as part of the model, the evaluation time is the worst of the group. Looking at the number of feature evaluated, the method evaluated roughly 5 times more features per point than the original reference, meanwhile the time is only $\frac{1}{4}$ as slow.

The difference in the ratios here are likely to be explained in the overhead querying the R-tree, as well as also in part the relatively higher workload of evaluating the included landmass polygon feature, which was shown in the results of Part I.

The jaggedness of the ridges may be explained by the height average not perfectly conforming to the height of the features modelling the ridges. As the max weight per feature is a value of one, the weight of two ridge features sum up to a value of two where tow adjacent ridge features connect. This gives the elevation value of the ridge a higher weight where ridge feature connects than anywhere else. As the weight of closely adjacent ridge features fall off towards the middle of any given ridge feature, the elevation may be expected to rise and fall along the feature with peaks at the connections causing the observed jaggedness. This behaviour of a slight rise in the elevation of ridges where features connect is also possible to observe in the reference configuration without detailing on closer inspection.

In addition it is worth consideration that the approach of splitting polylines into individual line segments is not applicable to polygons. With polygons being connected to form an encloses area on the inside, it is not reasonable to be splitting polygon shapes as can be done for lines, and the approach is therefor limited to application on lines only.

11.2 Overlapping features

The overlapping features method approaches the problem of adding details with a mindset inspired by LOD, where more detailed geometry is applied when something is observed from a closer perspective, where the translation to this implementation is the usage of more detailed features representative of the same original feature when evaluating the terrain closer to the feature.

The method attempts to achieve this by utilizing the priory merging of elevations to use multiple smaller features providing more detail, to override the original low detail feature when evaluating points close by.

The method shows the issue of creating features with a duality for the ridges, as the same ridge may appear twice, with low and high detail level. But while the river works similarly, this behaviour is not clearly observed for the river.

This is due to the elevation not changing nearly as fast close to the river as the elevation changes close to the ridge lines. In addition, the displacement factor used for the ridges is twice that of the rivers, causing the high detail ridges to deviate more from the low detail features than the rivers do.

The method is reliant on the detailing features representing values somewhat close to the values resulting from the low detail features, as the high detail features are functions to provide relatively local overrides, and do not merge with other features that otherwise merge with the low detail features. Therefor the displacement factor for the ridge lines are in this case way to high to represent local details of the ridge lines as the details fall far outside of the low detail ridge line.

The flat area on the top of the high detail ridges is due to the features overlapping to produce weight sums higher than one further away from the feature, and thus completely overriding the low detail merge for a region around the features, thus not blending as desired with the low detail merge.

While the method has interesting properties that imitate a LOD system, and has the potential to be applied to create several further levels of details in theory, the practical results from merging the features do not produce the desired results using the current method.

The method may possibly be improved by modifying the priority merge, such that the linear interpolation between priority levels is performed according to a weight maximum per level rather than the weight sum. This would be intended to avoid the expansion the full override beyond the location of the features, and thus avoid the flat observed in the presented results.

11.3 Fractal shapes

The approach of using fractal shapes was in the results shown to be the comparably better solution, both visually and performance wise, at least for the given detail level.

While the method under the hood produces as much geometry to evaluate the distance of as the first method, where the geometry is split up into multiple individual features, and evaluates the distance as many times, this the increase in distance evaluations has clearly a significantly smaller impact on the evaluation time as the overhead of the rest of the process, and it should be expected to outperform the first method for any level of detailing.

Considering the method with different numbers of MPD iterations applied, it is clear though that the evaluation time approaches an quadratic growth with further iterations applied. The first few iterations do not experience this quadratic growth to begin with because of the overhead of the rest of the process, like searching the R-tree and merging the values for the attributes. But at higher detail, the distance evaluations become the relatively dominant usage of the time, and the quadratic nature of the method becomes a problem when wanting further details.

The same patterns are seen when considering a configuration that increases the details of polygonal features, just with polygons being even more demanding to evaluate to start with, which is amplified accordingly with the quadratic growth when creating further details.

One advantage of the method i that it applies to polygons as well as it does to lines, which neither of the two previous methods can accommodate likewise.

The quadratic growth in evaluation time is the main disadvantage of the method, as it can quickly make it prohibitively expensive to create further details beyond 3 to 4 iterations of MPD, where the quadratic growth start to be dominant in increasing the evaluation time.

Given that a possibly significant part of a shape may be relatively distant, relative to the closest part of the shape, there are a lot of details to a shape being evaluation that is far from even possibly affecting the results. Considering this it would be desirable to ignore such details that are relatively distant in favour of the closer parts of the geometry, with similarly to how LOD systems based on VDR may adapt to use differing detail levels within the same geometry based on distance. This possibility of this is further explored as the topic of Part III.

Part III

Dynamic Shapes

Chapter 12

Design

The results from adding further details in Part II showed that the the of creating more detailed feature shapes held the most potential for adding details, both in terms of the visual results and the efficiency. However the time taken by distance evaluations of increasingly complex shapes created by an increasing number of MPD iterations grows quadratic. Depending on the desired output detail level, the expense of the evaluation time that may quickly become prohibitively large, especially considering usage at run time in a user interactive application.

To address this, two different possibilities to approach to mitigate the quadratic growth has been considered that may apply to fractal shapes generated with MPD. Both these take a dynamic approach to evaluating the distance. One of these is a progressive approach to LOD, and the other is a progressive optimization of the distance evaluation.

For the implementation, only the later of these is applied, but the former approach may also have possible use cases.

12.1 Distance LOD

This first method is based on how LOD is used to limit the amount of detail rendered of an object depending on the distance, as less details are visible from a distance. When considering any shape with an SDF function, the distance function will approach that of a circle with the shape inscribed over a large enough distance. Given this it is clear that a the details in the original shape is decreasingly propagated by the distance function the further away one views it.

Given that this also will be the case for a fractal shape, it may be expectantly reasonable replace the shape by decreasingly less detailed approximations as the distance increases as the smaller details are lost or become indistinguishable.

The concept of this method is therefore to introduce these details progressively by partial replacement of the shape as the evaluated point gets closer, as evaluation from far distances does not appreciably display the smaller details. This has some essential similarities to how the detail creation approach overlapping features was intended to work, as the smaller details only start taking effect when within a set bound. And also similarly to the aforementioned approach to detailing, these increasingly more detailed parts introduced for the shape, must to be smoothly blend to override their previous detail level to avoid discontinuities in the distance function that would cause discontinuous artefacts in output results.

The method is illustrated Figure 12.1, where it can be seen how the red dot being evaluated sees a progressively more detailed shape as it closes in on the shape.

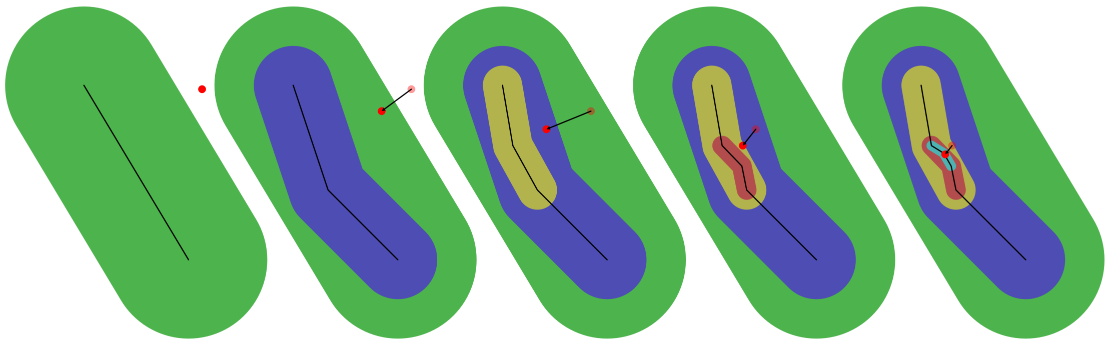


Figure 12.1: Distance LOD with progressively partial application of MPD evaluation. As the red point being evaluated approaches a line segment, the line is subdivided and the distance is transitioned to give the distance to the new detail.

12.2 Distance evaluation optimization

The second method is a progressive optimization of the distance evaluation for fractal shapes generated by MPD. The resulting distance found by this method will always be descriptive of the distance to the fully detailed shape, no matter how far away the distance is evaluated from. As such the distance is always the true distance and not an approximation like the former method provided.

This progressive optimization incrementally evaluated the shape for each stage of MPD, and selectively only progresses with further subdividing of lines that by an estimate present the possibility of becoming the closest after subdividing, this estimate is based on the maximum possible displacement a line segment can be given after subdivision, and any line that given the most favorable case cannot end up at least as close as the currently closest line at a stage is not subdivided. This progressive subdivision is then carried out until the most detailed version level is reached, at which point the minimum distance is recorded.

For polygons, the point is also either inside or outside the polygon. The evaluation of a normal polygon determines this by observing how many sides of the polygon a line from the evaluated point stretching in the left direction would be intersecting, or alternatively formulated, the number of sides in the polygon that has the evaluated point falls in the right projected shadow of. If the number is even, it gives that the point is outside, while an odd number gives that it is inside.

The progressive approach uses this same method, and observes this relation with sides of the polygon when they either are not subdivided, whether that be because they cannot become the closest, or because they cannot be subdivided because the final level of subdivision has been reached.

Chapter 13

Implementation

As mentioned, the implementation only includes the second method of distance evaluation optimization.

The method is implemented such that it is made available as a method option for lines and polygons in the MPD module, the option is called DYNAMIC. This option creates the fractal shape similarly to the STATIC option, but the shape is evaluated using the optimized method.

The implementation sets up a queue of active line segments to be considered for subdivision. For each iteration of MPD applied, all the previously queued line segments are evaluated for distance, and their distances queued up as well as minimum stored. Given the an assumption of all these line segments being optimally aligned symmetrically to evaluated point, the maximum possible displacement is subtracted from each and if it falls below the current minimum, indicating an estimate of the line possibly creating closer geometry by subdivision, the line segment is subdivided into the subsequent two line segments that are queued up. If the final detail level is reached, the subdivision check is ignored and the distances are instead compared for their minimum.

The pseudo code for the implemented evaluations are shown in Algorithm 17 for polylines, and Algorithm 18 for polygons.

Algorithm 17: Polyline dynamic distance evaluation

```
input  : A list $V$ of vertices giving the polyline.
         Position $p$ to find the distance at.
output: Distance $D$ from $p$ to the polyline.
1  begin
2      $D = \infty$
3      Create list $L_t$ for vertex tuples representing lines.
4      Create list $L_d$ for distance values.
5      Add the initial line as vertex tuple $(V.fist, V.last)$ to $L_t$.
6      while $L_t.size > 0$ do
7          $d = \infty$
8          foreach $l_t \in L_f$ do
9              Let $dd$ be the distance from $p$ to the line described by $l_t$.
10             $L_d.add(dd)$
11             $d = min(d, dd)$
12         end
13         foreach $l_t \in L_t$ and $l_d \in L_d$ when entering the loop do
14             Remove $l_t$ and $l_d$ from their respective lists.
15             if Vertices in $l_t$ are not adjacent in $V$ then
16                 $D = min(D, l_d)$
17                 Continue to next loop iteration.
18             end
19             Let $l_{md}$ be the maximum possible displacement when splitting $l_t$.
20             Let $l_{mpd} = l_d - l_{md}$ estimate the minimum possible distance after
                splitting $l_t$.
21             if $l_{mpd} <= min(D, d)$ then
22                 Split $l_t$ into two new line tuples and add them to $L_t$.
23             end
24         end
25     end
26 end
```

Algorithm 18: Polygon dynamic distance evaluation

```
input  : A list $V$ of vertices giving the polygon.
         Position $p$ to find the distance at.
output: Distance $D$ from $p$ to the polyline.
begin
    $D = \infty$
    $s = 1$
    Create list $L_t$ for vertex tuples representing lines.
    Create list $L_d$ for distance values.
    Create list $L_s$ for distance signs.
    Add the initial polygon sides as vertex tuples to $L_t$.
    while $L_t.size > 0$ do
        $d = \infty$
        foreach $l_t \in L_t$ do
            Let $dd$ be the distance from $p$ to the line $l_t$, also let $dd$ be
              negative if $p$ is in the right projected shadow of $l_t$.
            $L_s.add(sign(dd))$
            $L_d.add(abs(dd))$
            $d = min(d, abs(dd))$
        end
        foreach $l_t \in L_t$, $l_d \in L_d$ and $l_s \in L_s$ when entering the loop do
            Remove $l_t$, $l_d$ and $l_s$ from their respective lists.
            if Vertices in $l_t$ are not adjacent in $V$ then
                $D = min(D, l_d)$
                $s = s \cdot l_s$
                Continue to next loop iteration.
            end
            Let $l_{md}$ be the maximum possible displacement when splitting $l_t$.
            Let $l_{mpd} = l_d - l_{md}$ estimate the minimum possible distance after
              splitting $l_t$.
            if $l_{mpd} <= min(D, d)$ then
                Split $l_t$ into two new line tuples and add them to $L_t$.
            end
            else
                $s = s \cdot l_s$
            end
        end
    end
end
```

Chapter 14

Results

Since dynamically evaluating the distance to polylines and polygons produce the same results as statically evaluating the distances, the resulting output from the generation are the same as previously, with only the aim of improving the evaluation performance.

The implementations are also the same as previously, only with the method parameters in the MPD modules being set to DYNAMIC instead of the previous STATIC.

Because of this, the results here only present the resulting evaluation performances.

14.1 Adaptive calculation of polyline distances

Results for the "Ridges and rivers" configuration is evaluated similarly as in 10.1.1 for different iterations of MPD being applied, but with the MPD modules method parameters set to DYNAMIC.

Evaluation type	Features			Time	
	Layout	Evaluated		Evaluation	
	Total	Total	Avg. point	Total	Avg. point
Reference base	529	4286801	16.3528	1185ms	0.0045ms
MPD 1	529	4300907	16.4067	1732ms	0.0066ms
MPD 2	529	4307768	16.4328	2172ms	0.0083ms
MPD 4	529	4316499	16.4661	3405ms	0.0130ms
MPD 6	529	4318931	16.4754	4913ms	0.0187ms
MPD 8	529	4319200	16.4764	6655ms	0.0254ms
MPD 10	529	4319480	16.4775	8775ms	0.0335ms

Table 14.1: Evaluation performance for "Ridges and rivers" using dynamic evaluation of fractal shapes.

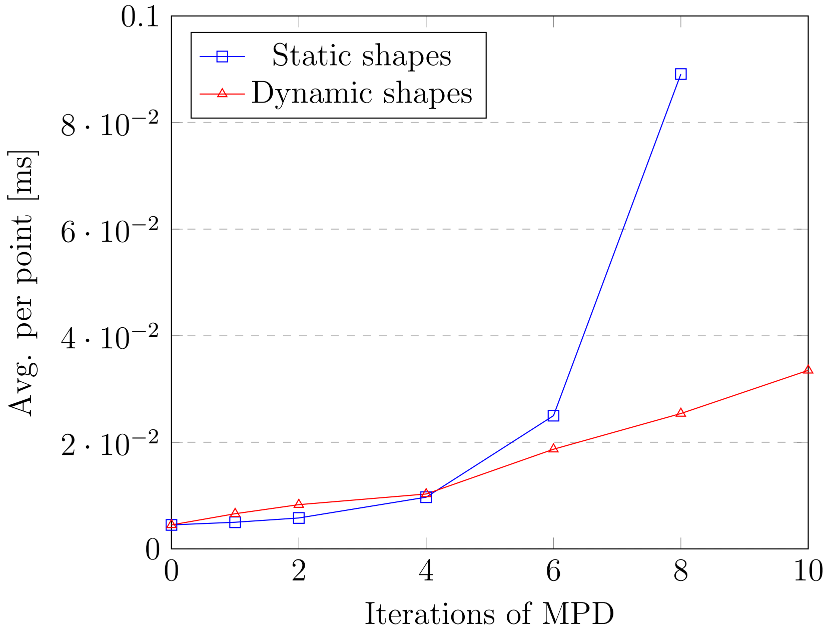


Figure 14.1: Evaluation times per point for "Ridges and rivers" with dynamic fractal shapes.

14.2 Adaptive calculation with polygon distances

Results for the "Ridges, rivers and lakes" configuration is evaluated similarly as in 10.1.2 for different iterations of MPD being applied, but with the MPD modules method parameters set to DYNAMIC.

Evaluation type	Features			Time	
	Layout	Evaluated		Evaluation	
	Total	Total	Avg. point	Total	Avg. point
Reference base	551	4200864	16.0250	1332ms	0.0051ms
MPD 1	551	4213833	16.0745	1964ms	0.0075ms
MPD 2	551	4223485	16.1113	2502ms	0.0095ms
MPD 4	551	4232776	16.1468	3884ms	0.0148ms
MPD 6	551	4237016	16.1629	5541ms	0.0211ms
MPD 8	551	4237493	16.1648	7401ms	0.0282ms
MPD 10	551	4237777	16.1658	9462ms	0.0361ms

Table 14.2: Evaluation performance for "Ridges, rivers and lakes" using dynamic evaluation of fractal shapes.

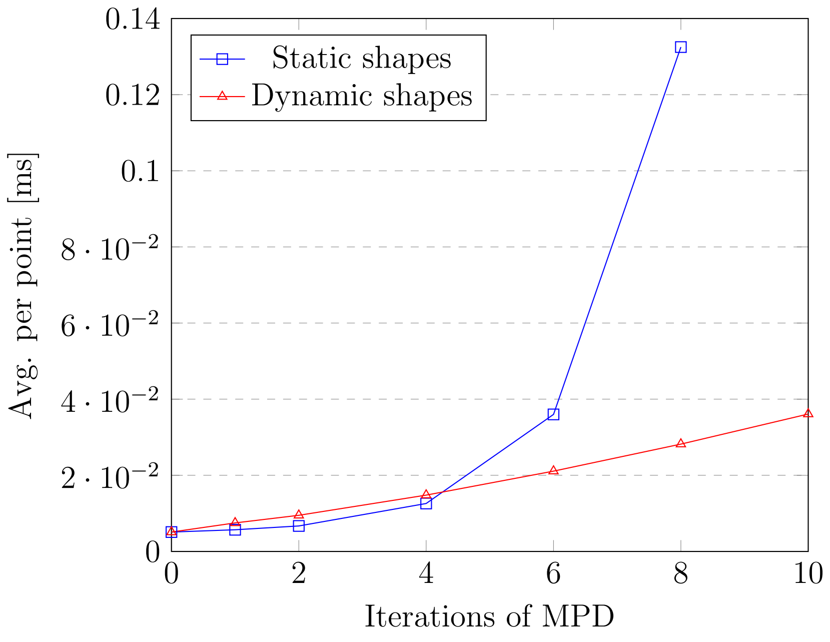


Figure 14.2: Evaluation times per point for "Ridges, rivers and lakes" with dynamic fractal shapes.

Chapter 15

Discussion

In Part II, it was shown that the approach of using fractal shapes to create details was the most favorable out of the methods attempted. But it also had the problem of creating an quadratic growth in the evaluation time as the detail level was increased.

By adapting to the detail level of a fractal shape on the base of relative proximity during distance evaluation, it was considered that the evaluation time for features with fractal shapes could be reduced.

The implementation of a method to progressively adapts to increasing details created by MPD during distance evaluation, has here been shown to in practice be capable of reducing the time complexity of the distance evaluation from quadratic growth, to linear growth when increasing the number pf MPD iterations.

The implemented method also evaluates the true distance to the fractal shapes, similarly to the the results in the previous part, and thus produces the same final results.

Part IV

Concluding remarks

Chapter 16

Discussion

With the finished results presented for all parts, it is now possible to consider the full picture of what has been achieved. The goal of the project was to bring the method the method presented in my previous work to a more competent state for reel world usage in interactive applications such as games.

The method as of the time of entering this project was intolerably slow compared to the expected possibility, especially if considering larger layouts than the presented ones used in the results. It was also lacking in detail when observing the final output, as it did not imitate any of the fractal properties of real world terrains.

The method as it functions now after the applied changes is now considerably faster, as it is more efficient with the selective evaluation of features, excluding all features that are guaranteed to not have an impact for any given location in the resulting terrain. It is important to take notice that this inclusion is not a guarantee for only feature with an impact being evaluated though. The selection is based on the minimum bounding rectangle of the area of influence for any given feature, which should not be expected to be a perfect representation, and it will therefor contain parts in the bounded area where the feature contribution becomes zero. The amount if this non utilized area will depend on the feature shape and influence range.

Whether the new run times are acceptable for usage in an interactive application, will depend on the constraints set for the application. The method is still significantly slower than using noise will be in most cases. I consider it unreasonable to draw any general conclusions on this topic as was is acceptable is a consideration that needs to be done on a per use basis for real use cases. But I will give some brief thoughts on the usage of the method to generate terrain with the game Minecraft as an example case [17].

In Minecraft the world is generated in chunks with a correctional area of 16x16 units of blocks. The generation of this requires a 16x16 samples heightmap to generate the chunk. Given that the method without extra details needs an average of around 0.0045ms per sample, this results in 1.1520ms per chunk needed to apply this method method. If the time spent on other aspects of generating the chunk is ignored for now, this would give up to 868 chunks per second, or over a 29 chunks square. This is in the general case, a larger area than the player can view with normal settings. As such, while the method may be relatively slower than using noise for the same purpose, it is not unreasonable that it can be acceptable.

If considering the application with more details, the evaluation time per sample is 0.0148ms for the "Ridges, rivers and lakes" configuration with 4 iterations of MPD applied using dynamic evaluation of fractal shapes. Making the same assumptions of currently ignoring time spent on other parts of the chunk generation, this results in 3.7888ms spent per chunk, for a total of 263 chunks per second, or over a 16x16 square of chunks. While this is slower, it may possibly still be acceptable if the player is traversing the world at slow pace. If the time is to slow, it may also be possible to reduce the sample rate and perform linear interpolation over small distances if a minor loss of details can be acceptable.

While these examples avoid looking at the total time of actually generating a full chunk, from earlier experience with modifying certain operations of chunk generation with comparable impact on the time spent per chunk, the addition of one or possibly even a few milliseconds per chunk may not be very noticeable to a player, and does not break the experience.

As such, it may considering the time and given similar constraints, be considered reasonable to use the method in an interactive application with the implemented improvements.

Chapter 17

Further work

Further work could approach several different possibilities of the project, some are more fundamental about the design of the, and some other are related to the implementation.

One possibility of interest, it the combination of this method with persistent noise or other input to modulate or even generate the values for various attributes assigned to features. This could be used to compliment, or outright replace the currently constant values assigned to a feature. This could allow the resulting outputs to take on certain characteristics and textures, such as a mountain feature may have its elevation modulated by a noise value to create rougher surfaces, while the river feature creating the valley below may not be modulated as such and result in a smooth surface, and these characteristics would be blended together as the elevation is merged.

Another possibility that could be of interest to the implementation, is creating new attributes within the graphical editor of WS, for example by making available a set of modules for different types of values that can be used to create unique attributes defined by a tag name, were attributes with similar tag and value type are merged together. Likewise the possibility for a user to create their own unique weight functions in the editor could be useful, instead the current reliance the available hard coded weight functions.

An important part modelling further details are also the creation of methods of generation feature layouts. A topic of interest here would be the generation of for example roads in combination with mountains and rivers, possibility even having such roads connecting areas defined to generate suitable terrain for settlements.

Considering the topic of settlements, the usage of the method to generate the terrain of a settled area, whether a city or suburban area may be a possible case study for usability. Such an environment would may have completely different rules to generation of features, and a collection of other feature classes than has been considered in the work up until now, such as different infrastructure aspects of the environment and plots of land for purposed for buildings or agriculture.

Since the method is based on merging values using weighted SDF evaluation, the method may reasonably also extended to generate three dimensional. While generating a volume is significantly more demanding than a plane, the use for generating cave systems is one interesting possible use case for such an extension.

Another possibility that relates to further approaches to layout generation is the possibility creating manual feature layouts,and to further use such manual layouts as seed values for generating features by modifying, extending or coexisting with such manual features.

Chapter 18

Conclusion

In this work I have improved the method of feature-based procedural generation presented in my previous work ”Feature-modeled procedural terrain generation with adaptive height evaluation” [1].

Where the previous work was capable of producing a terrain including mountains, rivers and lakes, such that water could flow in a consistent downwards direction, the method was slow, only accommodated the generation of elevation values, and the results looked artificial with a lack of details.

In this work, the evaluation performance has been improved by spatially indexing features, such that they can be selectively used for evaluation only when relevant, and this has been shown to significantly reduce the evaluation times. Instead of evaluation time relying on the number of features in a layout, as it was previously, it now instead relies on the density and configuration of features in the layout. Whether the improved evaluation efficiency is enough to make the method applicable for run time generation in interactive applications, like for example games, may still depend on the constraints and expectations for the application. The method is still significantly slower than the simplest forms of noise based generation, but the evaluation of arbitrarily large skeleton layouts can now be considered to be relatively consistent independently of the size of the skeleton layout, as it is also the general case for noise based generation.

The work presented has also shown the method being extended to handle further values describing the terrain through attributes, like surface materials for texturing, and water depth for carving out rivers and lakes filled with water, as well as the elevation. The use of attributes has also been shown to be an important tool in the modelling of various types of features by configuration, further developing the approach introduced in the previous work of feature classes emerging from intent when configuring a generator, rather than by hard coded implementations.

How attributes are merged is also in part flexible to the needs of an implementation. Including continuous and discrete attributes, as well as the possibility of merging attributes values with priority which allows for values to be overridden by higher priority values. The attributes system may enable building of complex models with feature interactions, and an example of this concept has even been shown in the usage to implement a simple level of detail system for the evaluation of elevation.

As with the introduction of attributes themselves, the decision of making the weight function a property of the individual attributes to be assigned features, has become a useful tool in configuring unique classes of features that can provide different characteristics to the resulting terrain.

To address the problem of the previous work producing low detail results, there has been presented and compared three different approaches for adding further detail. Out of these the most promising method of using fractal shapes has been further developed to be more efficient, by a progressive optimisation to determine distance, it may evaluates complex shapes with the level of detail changing dynamically across parts of the shape according to relative proximity.

The combinations of these improvements has made the method capable of producing terrains with the desired logical structure to bodies of water, but now more efficiently and featuring fractal details to the paths and outlines taken by features. With the flexible options to configuring features, and the possibility to design configurations with a graphical tool like WS, it may be possible for a user given the access to appropriated layout generators can to create skeleton layouts with complex interaction between features configured to behave as unique classes of features.

The implementation is also well on the way to becoming a greatly powerful addition to the WorldSynth library of functionality, and it is my intention to further develop it further as a public addon to WorldSynth. With future work including including such as further options to specify wight functions, modulation of feature elevations using noise, as well as further attributes and layout options, possibility even manual pipeline to creating full or partial layouts, it has potential become a systematic tool for use in both artistic terrain authoring and run time terrain generation.

Bibliography

[1] B. Benjaminsen, 'Feature-modeled procedural terrain generation with adaptive height evaluation', 2021.

[2] L. Balasubramanian and M. Sugumaran, 'A state-of-art in r-tree variants for spatial indexing', *International Journal of Computer Applications*, vol. 42, no. 20, pp. 35–41, 2012.

[3] A. Guttman, 'R-trees: A dynamic index structure for spatial searching', in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984, pp. 47–57.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, 'The r*-tree: An efficient and robust access method for points and rectangles', in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, 1990, pp. 322–331.

[5] I. Quilez. (). '2d distance functions', [Online]. Available: https://www.iquilezles.org/www/articles/distfunctions2d/distfunctions2d.htm (visited on 15th Feb. 2021).

[6] P. Jiarathanakul, 'Ray marching distance fields in real-time on webgl', Citeseer, Tech. Rep.

[7] I. Quilez. (2008). 'Raymarching distance fields', [Online]. Available: https://iquilezles.org/www/articles/raymarchingdf/raymarchingdf.htm (visited on 2nd Jun. 2021).

[8] ——, (). 'Distance functions', [Online]. Available: https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm (visited on 15th Feb. 2021).

[9] T. K. Heok and D. Daman, 'A review on level of detail', in *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.*, IEEE, 2004, pp. 70–75.

[10] B. Karis, R. Stubbe and G. Wihlida. (10th Aug. 2021). 'Nanite, a deep dive', Epic Games, [Online]. Available: http://advances.realtimerendering.com/s2021/Karis_Nanite_SIGGRAPH_Advances_2021_final.pdf (visited on 11th Aug. 2021).

[11] S.-E. Yoon, B. Salomon, R. Gayle and D. Manocha, 'Quick-VDR: Interactive View-dependent Rendering of Massive Models', *IEEE Visualization*, pp. 131–138, 2004.

[12] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio and R. Scopigno, 'Batched multi triangulation', in *VIS 05. IEEE Visualization, 2005.*, IEEE, 2005, pp. 207–214.

[13] C. Eisenacher, Q. Meyer and C. Loop, ‘Real-time view-dependent rendering of parametric surfaces’, in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 2009, pp. 137–143.

[14] A. Rockwood, K. Heaton and T. Davis, ‘Real-time rendering of trimmed surfaces’, in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, 1989, pp. 107–116.

[15] A. Fournier, D. Fussell and L. Carpenter, ‘Computer rendering of stochastic models’, *Communications of the ACM*, vol. 25, no. 6, pp. 371–384, 1982.

[16] B. Benjaminsen. (2021). ‘Worldsynth’, [Online]. Available: https://www.worldsynth.net (visited on 9th Mar. 2021).

[17] M. AB. (). ‘Minecraft’, [Online]. Available: https://www.minecraft.net/en-us (visited on 25th Sep. 2021).

[18] B. Benjaminsen. (2021). ‘Feature-modeled procedural terrain generation with adaptive height evaluation - implementation’, [Online]. Available: https://gitlab.com/booleanbyte/specialization4551.

[19] D. Moten. (2019). ‘Rtree2 0.9-rc1, java library’, [Online]. Available: https://github.com/davidmoten/rtree2 (visited on 25th Sep. 2021).

Appendix

A Project source compiled version

The source code and compiled version project can be acquired at:
https://gitlab.com/booleanbyte/project4900

The implementation is an addon for WorldSynth version 0.4.X. WorldSynth can be acquired from the download section of the WorldSynth website:
https://www.worldsynth.net

To use the project implementation, acquire the compiled project and WorldSynth software as described above. After unzipping the WorldSynth download, put the compiled project version ”Project4900.jar” file in the ”addon” folder of the WorldSynth download. The modules implemented in this work will then be available after launching the application.

The WorldSynth example patches used in the results are also provided with along with the source and compiled addon of the project, these files are found in the project example folder.

NTNU

Norwegian University of
Science and Technology