ASSESSING THE PERFORMANCE OF PROCEDURALLY GENERATED TERRAINS USING HOUDINI'S CLUSTERING METHOD

by

Varisht Raheja

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer Graphics Technology

West Lafayette, Indiana

May 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Carlos Morales, Chair

Department of Computer Graphics Technology

Dr. Tim E. McGraw

Department of Computer Graphics Technology

Dr. David Whittinghill

Department of Computer Graphics Technology

Approved by:

Dr. Nicoletta Adamo-Villani

Thank you to my academic adviser who guided me in this process and the committee who kept me on track with their rigorous and flexible knowledge.

ACKNOWLEDGMENTS

Thank you to my supervisor, Dr Carlos M., for providing guidance and feedback throughout this project. Thank you to the author of Terrains using Satellite Data in Houdini, David CGMK for the inspiration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

VFX	Visual Effects
3d	3-Dimensional
HDA	Houdini Digital Asset
csv	Comma Separated Value

ABSTRACT

Terrain generation is a convoluted and a popular topic in the VFX industry. Whether you are part of the film/TV or gaming industry, a terrain, is a highly nuanced feature that is usually present. Regardless of walking on a desert like terrain in the film, Blade Runner 2049 or fighting on different planets like in Avatar, 3D terrains is a major part of any digital media. The purpose of this thesis is about developing a workflow for large-scale terrains using complex data sets and utilizing this workflow to maintain a balance between the procedural content and the artistic input made especially for smaller companies which cannot afford an enhanced pipeline to deal with major technical complications. The workflow consists of two major elements, development of the tool used to optimize the workflow and the recording and maintaining of the efficiency in comparison to the older workflow.

My research findings indicate that despite the increase in overall computational abilities, one of the many issues that are still present is generating a highly advanced terrain with the added benefits of the artists and users' creative variations. Reducing the overall time to simulate and compute a highly realistic and detailed terrain is the main goal, thus this thesis will present a method to overcome the speed deficiency while keeping the details of the terrain present.

CHAPTER 1. INTRODUCTION

With the advent of better computer and an increase in computational prowess, the methods to produce and replicate terrain generation has not only improved in various parts but also has increased. Since nowadays many computers can handle dealing with complex data that arises due to the generation of structures like terrains and other 3D geometries there has been a huge demand for fast-paced generation of high quality and realistic looking terrain. Even though terrain generation is primarily found in the entertainment industry, there have been evidence of it being found in less popular areas such as, vehicle dynamics, military training, Geo-information system (GIS), flight simulation etc. One of the main advantages of using a procedural generated terrain is the noise used to develop such structures. This procedural noise aids in both immensely reducing the memory used to increase the speed and thus the time taken as well as increase the content produced. Noise here refers to the procedural generated texture that is created using an algorithm to achieve randomness and improving on the details of an existing geometry. Since early times the computer-generated imagery has always been visually "machine-like" and looked highly uniform and consistent. This is where randomness has played a vital role in bringing out a visually appealing design and features by creating a naturalistic feel and details to the virtual scenes.

However, there are still major drawbacks to such terrains, the natural realism of the terrain will affect the performance of the simulation as such high details will require extra processing steps and therefore memory. Since video games targets performance over the accuracy of the geometry so that consumers who cannot afford the top-of-the-line performance-based machines will also be able to purchase the product but in doing so reduces the amount of detail in the scene. The most popular solution to minimizing time and reducing effort would be to use a high-grade GPU render, however, despite the technological advances the implementation of such workflows and pipelines are few. Major studios like Walt Disney and Pixar, among others have created a successful GPU pipeline for the many computations and processes that occur. Similarly, despite the multitude of terrains and the techniques to create the terrains exist there has been a reduction in the overall creativity and artistic control over such terrains. Since each technique is generated using a mathematical model the procedural generation has gained importance but the amount of

variance that can be achieved has been significantly reduced. This has led to more issues as manual variance and constraints are extremely time consuming as well it is affected by the proficiency of the user themselves which has no substitute since no amount of algorithmic creation can ever produce a creative exotic art which is completely customized and modified depending on the users imagination.

1.1 Related Work

Most of the procedural generated terrains use the height-map algorithm which is based off the fractal generation algorithm or the fractal noise algorithm which uses the Perlin noise as its basis to generate a height-map. The perlin noise can be re-scaled and added onto itself to produce the fractal noise. The fractal noise makes us of the randomness and the layering of noises on top of each other to increase the complexity of the noise itself and thus the randomness. Lower the layers will lower the details of the noise and complexity. In terms of virtual terrains and other geometrical features much of the literature has been focused on how to improve the visualization process and accelerate the speed of the simulation of the terrains. This led to the ever-evolving invention of different algorithms which could handle huge datasets without a sudden drop in the computational abilities while also reducing the rendering time and boosting the overall efficiency. Thus some algorithms also make use of the distance between the camera and the actually geometry which is in closer in view to the extent that the closer the object the sharper and highly detailed that object will be in contrast to the far-away objects which will have a lower detail and so a lower resolution. Despite the origination of such algorithms the fractal technique is still the most prevalent, this is mainly due to their speed and ease of implementation as well as the randomness and irregularity of the shapes while building the terrain. The randomness generated breaks the conformity that uniformity and consistency will be more functional. In this case, the irregularity creates a more aesthetic feeling and gives a "natural feeling" and beauty which would be hindered if not for the discovery of some algorithms. Unfortunately, the expressiveness and the generality of probability also comes at a cost, inference is typically impossible. Since the very nature of randomness is to generate a value which cannot be predicted, similarly, the variation out of this randomness also cannot be. In some areas where the details are higher may not be a wanted decision but rather the decision of the algorithm and so despite an increase in the overall variation of the terrain, the control has been reduced as well.

Nevertheless, the fractal technique has a very high realistic outcome in which the terrains have realistic details and features but have a few variations of the design and a low control on the result. Even though the realism of the terrain is essential this method doesn't leave room for creative input to take place and leaves no room for attempting to represent a distinct and exotic terrain. Fractal based algorithms and techniques are highly rigid in designing and modification of terrains to an extent that users are unable to apply their own creative insights and are unable to further evolve the terrain according to their own aesthetic imaginations. The main approach is to create a terrain that is realistic versus a terrain that is aesthetically pleasing. Since creating a terrain that has not been built from scratch and which can limit the variability of the design can, furthermore, reduce the time taken for users to modify and re-structure the terrain depending on their own creativity. Apart from the perlin and simplex noise generated fractals there exists another method which is part of the random midpoint displacement method called the diamond square algorithm. This algorithm generates 4 corner values of a 2d array to fill in the inner values with midpoint displacement, the array requirements in width and height are as follows, (O'Brien, 2018):

$$2^n+1$$

Extreme efforts have been made in recent years to establish the alternative methods such as the cellular automata and tile-based approach which has shown either fast non-realistic terrain generation or slow realistic terrain generation. Many algorithms have also been developed to augment and support the fractal-based method, but while the algorithms do produce a significant level of realism, they are much slower compared to the noise(fractal) like method, for example, using a hydrology-based algorithm. These methods have serious limitations not only in the performance but also in generating autonomous generation as it involves a feature height-map generation provided by the user. Other approaches follow suit where the realism requirement has been met but the performance has dwindled, while other algorithms compromised on realism and have increased controllability and randomness of the noise such as Gabor noise, wavelet noise and random-phase noise. Interestingly, there is not a lot of literature regarding the comparisons of these algorithms. The family of the mid-point displacement method does not use multiple passes corresponding to the different octaves, but this algorithm also possesses limitations.

1. The non-locality where the h value is based upon the coordinate given by the nearest point.
2. The quality of the generated terrain.
3. The features are defined as such that this algorithm possess less control related parameters than the fractal method.

There are also difficulties presented in keeping the memory storage low as this algorithm is unable to assemble different chunks of terrain data.
As derived from the information above the Perlin and Simplex noise are by far the most popular and common means of creating a fast-paced terrain with an acceptable level of compromise of quality in terms of realism. Noise generation is the most appropriate method due to the combination of simplicity, speed and performance and the quality it provides. In conclusion, this short review of existing methods and algorithms are typically not used by themselves but are used to augment and support the previous methods to overcome the shortcomings and possible enhance the quality of terrain generation.

The aim of this dissertation is to provide a prelude into the formation of realistic terrains while not compromising on the speed and efficiency of the production and simulation of the terrain. The method here is only a commencement into the tumultuous world of procedural ism which keeps many of the previous features intact while also allowing for time optimization.

1.2 Purpose

My objectives are multi-fold. Firstly, accessing satellite data to gather data points which can be used to generate a highly realistic procedural terrain and using HoudiniFX to convert the data points into a volume-based height-field map which will be used as a representation of the terrain. Secondly, the data points which have been imported inside Houdini will go through a clustering method which can be adjusted on the user, thus defining the number of clusters the user needs to divide the data set by. Thirdly, the speed and performance of the terrains will be compared one with using the clustering and distribution method which utilizes a tool developed in python inside Houdini, against the same terrain which does not use said tool. Also, the same procedure will be done on a low level of detail in which case the amount of points will range in the lower

million versus a high level of detail in which the amount of points will be in the upper millions. The purpose of this study is to evaluate any link between the speed and performance of whether the tool being used can actually save and increase the performance of the time taken to simulate and generate a realistic terrain depending on the change of amount of points being processed.

1.3 Problem

One of the many problems with the study (terrain generation) is that the algorithms that it uses creates realistic looking maps but scaling it to fit infinite combinations is not possible. Unfortunately, even with the increase in computing power, the CPU cannot withstand huge chunks of data, so GPU needs to be used. One major problem area being faced by many production houses is how to create a credible environment in a limited amount of time and how to do it without affecting quality. Finally, there is a problem with the procedural-ism itself that a solely procedural generated terrain will lack the storage and retrieval capabilities as the data set provided would be too small while the randomness would be much too great without the aesthetic design it required for it to look good. However, these are not the only problems that have been left unheeded. Creating a realistic terrain also contains many limitations, where to develop a highly realistic terrain means gathering a huge data set which could easily run up from a few million points to a few billion points depending on the level of detail. As one moves into the high-level detail stage despite the advanced progress in the computational world, it is quite tedious to run a simulation smoothly on a few billion points. Thus, collecting and analyzing the actual speed and performance of the simulation and generation of the terrain and to what actual level of detail can they perform to would benefit not only production companies but also educational institutes who need to keep up with the advent of novel technologies.

CHAPTER 2. BACKGROUND

2.1 Introduction

Procedural content generation has become a contemporary phrase in the world of entertainment as well. Since the advent of realistic games due to the endless demands there has been a significant improvement in the overall structure and aesthetic quality of the games. Procedural terrain generation has spurred this advancement as many digitally created entertainment products do require such large creations. So, with the increase in a lot of procedural content many ways to identify and define this term became popular. One of the ways to define procedural content generation is as “any kind of automatically generated asset based on a limited set of user-defined input parameters” (Ruben M. Smelik et al.,2011). Furthermore, it can also be defined as using the amplification algorithm which is using a small set of input parameters to deliver a fully functional automated output of a large set of data. Togelius formulates a definition by an antithesis, saying that procedurally generated content does not correspond to content that is generated by users even if they make use of procedural algorithms since they must be manually parameterized. Procedural generation is an alternative to manual design which costs unnecessary time but the need for creative input on the features and the parameters can influence the generated object. Shaker et al., are more concrete and define PCG by giving examples what PCG is (e.g., a software tool to generate random dungeons without any user input) and what it is not (a map editor that lets users place items). Although the above-mentioned myriad of definitions has been documented the earlier usages of this technique was in multiple fields especially for research in computer science and biology based on the amount of influence at that time. Despite the seemingly narrowed down areas, topics like computer science possess many data structures and formalizing them is a huge effort. So, four main areas where this was used are L-systems, grammars, shape grammars and programming languages. As we can see programming seems to be an area which can never be error free despite its flexibility. Hendrikx et al. introduced the abbreviation PCG-G (procedural content generation for games) to remove this term from animated and live-action movies. However, this shows that procedural based algorithms and procedures can be applied to a variety of areas such as, urban planning, films, etc., showing that there is active research and awareness of automated procedural geometry in the industry.

Digital created terrain generation has been used for a long time and the respective algorithms and novel techniques have been created to improve the efficiency and the to decrease the computational performance. As technology was developed the load on the CPU was reduced and was passed to the GPU to bear the high-resolution geometry which accompanies from creating the terrain. Although, there have been many algorithms each of them has distinct advantages and disadvantages, some of the common ones are below:

- Low human input
- High degree of human control
- Intuitive in control
- Real time simulation and rendering
- Fast generation of a variety of terrains with quick modifications

As we can see from above there are conflicting parameters to the algorithms such as low human input and high degree of human control, this is there as to prioritize the traits of the algorithm depending upon the type and number of terrains being used. A large number of terrain will be produced faster if they are fully automated with a low amount of human input and if the terrain require a more creative input and which have to be designed very specifically to a particular type then a high degree of human control is required. In other words, compromises need to be made whether human creativeness is prioritized or not. Terrain generation also possess other attributes, representation of the terrain is important, the basic structure of the terrain will influence the overall built of the terrain, the features and the minor details. Manipulating the terrain also creates variations in the structure itself, a heightmap is the most common way of representing a terrain and any changes in the areas of a heightmap will produce a different result in the terrain. Whether larger crevices, valleys or peaks are required the heightmap can procedurally modify whatever structure as needed.

2.1.1 Heightmap

A height map is a scalar quantity meaning it has only a x-coordinate and a y-coordinate which corresponds to an elevation variable(*h*). “In practice, a height map is a two-dimensional rectangular grid of height values, where the axis values are spaced with regular intervals valid

over a finite domain (see Figure 1). The most common data structure to represent them is 2D arrays filled with the elevation values" (Miguel Frade et al., 24 November 2008).

$$h = f(x, y)$$

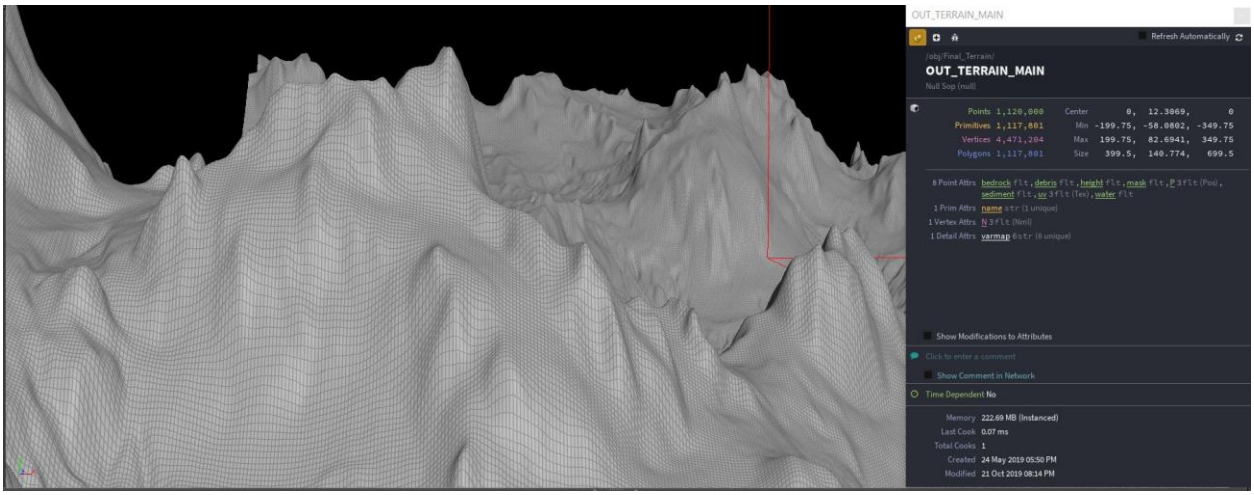


Figure 2.1 General Mock-up of a Terrain using Heightmap.

A height map has many hidden advantages that is not completely visible at the time. Rendering of height maps is much faster which can also be done in real-time. Not only that, but a highly detailed mesh can also be rendered quite quickly as the algorithm considers the distance from the camera and reduces the amount of detail as one goes further away. A hidden advantage of this technique is the collision detection which is immensely simplified as only a few triangles need to be checked for the collision to take place. Since height maps are based on noise which use a color range of 0-1, where 0 is black and 1 is white, grey scale images can also be used to construct a height map for terrain generation as shown in the figures below.

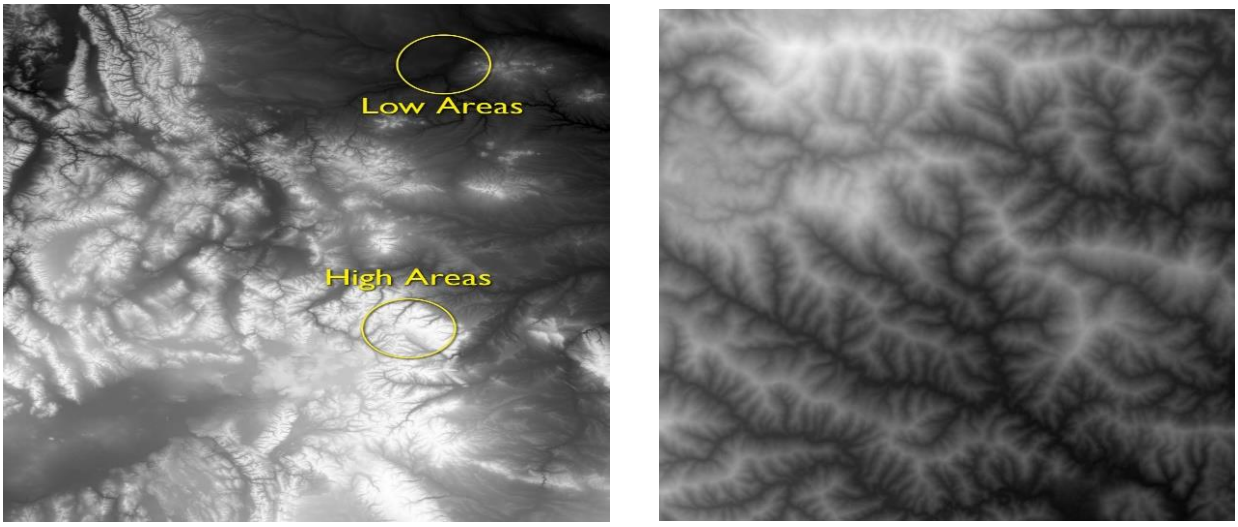


Figure 2.2 Showing the elevation in height maps according to color

Thus, many machine learning algorithms can also be applied including computer vision (CV) and image processing techniques to generate a base model for the terrain and details can be added, modified or even reconstructed using these techniques. Finally, geographical information system (GIS) can also be used by height maps to represent real world terrain but give a far more realistic look as satellite imagery is used to reconstruct the terrain. Height maps are one of the widely used techniques to generate terrains as mentioned above, but there are some limitations as well. One of the key grievances of using such a technique is that it cannot represent cave-like features. It is not possible for the algorithm to process such data due to the ineptitude of structuring data where multiple elevations(h) for the corresponding coordinates. As there is a steady increment in the details of a mesh the computational speed and power slows down to perform the necessary calculations. Thus, even height maps possess a finite amount of resolution which cannot handle the levels of detail on the terrain after a threshold. Although the algorithm is fast enough to compute the changes made in real-time a huge, complex and highly detailed mesh is quite an arduous task to uphold given the limitations of the technique. Finally, it is not possible for the technique to adapt to new scales. A height map is generally generated on a grid but cannot be accommodated to use a spherical object or a more complex object as the density of the height map will vary directly, the points will be substantially greater near the poles than the middle of the object.

2.2 Terrain Generation Techniques

In terrain generation techniques one of the most traditional technique is the measuring technique. This technique used digital elevation data to produce the elevation model required for real-world measurements to generate the terrain. These models are commonly used by remote satellite sensing techniques like satellite imagery and land surveys. The main advantage of using satellite data imagery is the high-realistic features with minimalistic human input that are generated in comparison to other techniques. However, due to the high realism factor the performance of the computer is affected and if any designer modification needs to be done the time consumption will increase dramatically to account for finding a satellite image data that would match the imagination of the designer. Another technique apart from using pre-existing data is to sculpt and design the terrain as directed by the client or to closely resemble an existing reference from another location. This technique of sculpting and designing from user's own imagination or to take inspiration from another source is the modelling technique. This is by far one of the most flexible techniques in which any kind of modifications and changes to large parts of the terrain can be made quite easily. Since the user is designing the entirety of the surface from scratch, the idea and execution are all dependent on the abilities of the user itself. Many 3d software's today provide immense control and variability when it comes to designing a terrain. However, this technique also poses a major disadvantage where the realism and the natural resolution of the terrain cannot be matched to the details of the satellite data, also, there will be an immense increase in the time consumption and effort to recreate a 3d version of any exotic and custom made design. Moreover, this is all highly dependent on the experience and the abilities of the designer themselves. This is one of the more traditional techniques to be used. Apart from this a slightly more modern approach is being used, the procedural technique, where the essential part is to save the time and effort made by the designer relatively from the traditional approach. These terrains have a strong emphasis on the technical side and are highly valued for their capabilities to possess multiple variances without having to change the design from scratch. One of the common ways to ensure realism in this technique is to simulate real world erosion techniques of the water, wind and heat. As mentioned before, although these techniques produce realistic terrains, they require highly technical knowledge in of the real-world laws. These also require high computational performance. Other procedural approaches include using the fractional Browning motion (fBm) noise has a well-defined power spectrum. Using the fast

Fourier transform (FFT) to convert the random frequencies which are calculated to convert them the frequency components into altitudes. Despite using such advanced techniques there exists no control over the features of the terrain. Also, the simulation does not seem to share the same physical laws that govern real world simulations. The fractal technique is part of the procedural generation of terrains as it allows the geometry to be self-similar, a key concept in which as the object is magnified the subsets of the object seem to have the exact same shape as the original. The smaller part seems to interpolate completely with the original parts. This feature of fractals is extremely useful in generating terrains regardless of the scale in which it is displayed. Since this technique is based of noise, every time the algorithm is run randomness is generated to create a more visual appeal to the terrain so that not all features possess the same shape. The speed and ease of use makes this one of the most popular algorithms. Several applications use this algorithm such as Terragen (which is a hybrid fractal/modelling tool) and GenSurf (a mapping tool for Quake 3 Arena video game). Due to its high popularity it is relatively easier to spot the terrain built on this algorithm and since this is an algorithm based on a mathematical model it presents less control for the designer to change certain features thus lowering variance.

Finally, proceduralism techniques has tried to bridge the gap between giving control over to the designer as well as automating certain processes to reduce effort and time consumption. These techniques heavily rely on physics and mathematical models which help establish the foundation for generating highly detailed terrains which can simulate real world behavior for wind, soil erosion, water, thermal etc. Since these are constructed upon mathematical and physics-based models, they require an extensive knowledge in the same fields to implement the knowledge in generating them. These techniques mimic real world applications and are heavy on the processing power of the computers which may cost time. The major reason proceduralism is favored is the reduction in the human effort and time since most of the computation is being done by the computer itself there are way to increase it. On the other hand, proceduralism does decrease the manual control the designer has over the terrain which decreases the creativity and originality of it. Another commonly used procedural algorithm used is the fast Fourier transform (FFT). This is done by using the fractal Brownian motion (fBm) noise known for its abilities to describe random occurrences and give rise to a wide range of interesting models. This can be best describes as, “The correlation of a random process B(t) for any time t_>0 and h>0 can be

represented as the expected value E(B) of the product of nonoverlapping increments of the fBm process where the real number H is called the Hurst index falls in the range (0, 1), and describes the roughness of the motion, with a higher value leading to a smoother motion", (Fractional Brownian Motion—An overview, October 15, 2019). This can help in calculating the frequency components and then the inverse of the FFT can be convert the frequency into amplitude as seen in the figure below.

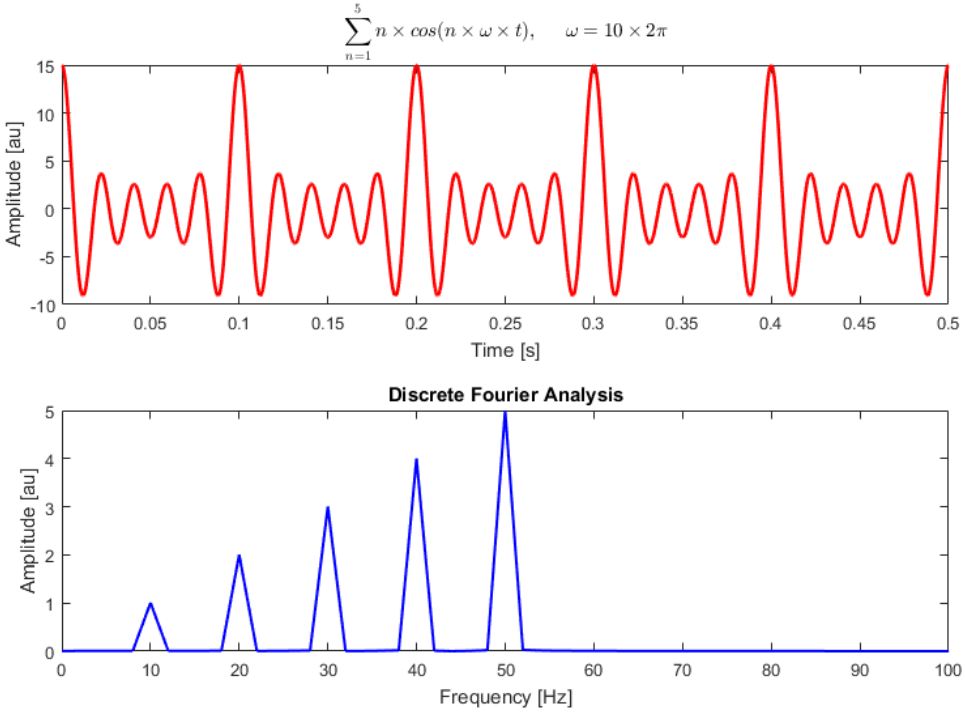


Figure 2.3 A discrete Fourier analysis of a sum of cosine waves at 10, 20, 30, 40, and 50 Hz

Heightfield data or heightmaps can be generated from both real-world scans and surveys as well as from scratch by using fictional worlds. To apply the user's own modification and creativity a tool called a "brush" is used. Strokes can be applied to the terrain structure by using an input device like a mouse and can be made interactive so the modifications can be viewed in real-time. These brushes which apply the strokes are relatively simple to use and provide maximum flexibility and control. However, much time and effort are required to create the necessary details and features that make a terrain realistic. Some of the common operations include, terrain lowering, and levelling and some brush parameters include, brush radius and effect scale. Algorithms exist which can output realistic heightmaps and can be broadly categorized in these

two classes, simulation and procedural synthesis. By imitating the geological processes, the simulation algorithms can globally transform terrains. These algorithms are an over-simplification of natural processes which exist. With enough time and continuous experimentation impressive results can be achieved, however, this is extremely time taxing and require a considerable number of iterations. In contrast, procedural synthesis algorithms which do not use the natural processes, instead they rely on the fractal techniques like semi-random patterns like rough mountains and smooth hills. Minimum effort from the user is highly effective when generating a procedural terrain and so this algorithm is used when multiple iterations do not need to be evaluated as they are many times better than the simulation algorithm. However, limitations are also several, there is a limited amount of control on the brushes as well as when the user requires more finite control over the terrains features the parameters are limited as they are used globally. Including the limited control this algorithm is also limited to only the creation of a single type of terrain.

2.2.1 Digital Elevation Model

These models are extremely important and versatile in visualizing the surfaces and the terrain of environment on Earth in a 3d space. They provide rich information regarding the topography of surfaces that 2d surfaces cannot provide. A DEM (digital elevation model) is a set which defines a set of points or grid cells and is a 3-float vector (x,y,z), where the x and y-coordinate are traditional longitude and latitude measurements and the z represents the height or the elevation of the model. There are main two types of DEM's – Digital Terrain models (DTM) and the Digital Surface Model (DSM). The DTM is typically a ground only elevation model as it includes the x,y,z values of the ground only. These points do not include the vegetation part and have been removed from this particular model .The DSM on the other hand contains all elevation points regardless of the area and type of the surface, these include, ground, vegetation or even man-made structures. Since the information both contain are highly different their respective uses and applications also vary. DTM's are typically used in engineering, construction and hydrographic projects whereas the DSM's are mainly utilized in 3d visualization, infrastructure management, line-of-sight and obstruction mapping application, essentially where the height of the surface plays a vital role. Now, this has also been used in the entertainment industry. These two

techniques are frequently defined by the accuracy and high realism of the surface given the high number of points it generates to create the terrain as well as the grid cell density (spatial resolution). The horizontal accuracy is measured by the x,y RMSE or Circular Error (CE), similarly vertical accuracy is measured in either z RMSE or Linear Error (LE). To provide a small example as to the inner workings of the techniques, a 1m CE90 and a 5m LE90 mean that there is a 90 percent confidence the DEM measurement coordinates are accurate up to 5m respectively of their original position from the ground. Similarly, in DEM the method used to provide the actual resolution is defined by the elevation measurements and not spatial density like in the case of the remotely sense imagery. Raster DEM's are identified by the grid cell size, 1m or 5m cell size or by the point spacing (distance between points) like the 1m or 5m. Nowadays, the DEM's are generated using the digital sensors which are attached to an aviation machine which can fly around and scan the terrain required to formulate the points. The unmanned vehicle such as a drone can also be used to acquire the information necessary, the data collected by these sensors are used to gather the elevation measurements. The three main sensors are, optical imaging sensor (frame camera / push broom scanner), synthetic aperture radar (SAR) sensor and the laser scan (LIDAR). The Lidar data is often used by the aircraft, while optical imaging sensor can be carried by helicopters, airplanes and satellites. The SAR data cannot be carried by smaller machinery and so requires larger aircrafts or satellites. Photogrammetry is sued to create topographic maps and DEM's from aerial overlapping photographs. The process to take out the required elevation data from such aerial photography is photogrammetry. Using the ground control survey points (GCP's), a 3d visualization of the terrain can be created allowing for the direct measurement of elevation data of ground features. For complex structures like caves and dense canopies the elevation data is interpolated especially for obscure locations. This technique is widely used today but most of it is digitally automated. The main advantage of utilizing this technique is the precise capture of many small features and other parts like drainage curbs etc., to generate the 3d points which are extracted from the photographs and has a direct correlation between features in the photos and the actual surface. Using this technique all the small details can be extracted and used accordingly, ridgelines, drainage, etc., all these are extremely vital for hydrological engineering. The main weakness is the expense that will be incurred for adding the ground survey points as well as incurring the increase in time consumption. A novel enhancement on the photogrammetry technique is being employed,

instead of using the traditional method of collecting satellite tracked image data, the new approach utilizes many satellites to get the data across from many angles to capture the data in its entirety while also preserving the highest of details and features. Another active sensing data collection technology is Interferometric Synthetic Aperture Radar (IFSAR) helps in emitting the data signals from aircrafts down to the surface of the Earth, these are radar pulses which reflect off the surface containing the valuable information required to yield accurate measurements. Elevation mapping with SAR technology is also called InSar or IFSAR mapping. Moreover, even in obscure lighting conditions and where the areas have a persistent cloud coverage especially in tropical regions the SAR sensor is extremely capable of capturing and collecting the elevation data. Although, this provides coarse data set, the cost of acquiring such data is comparatively lower than other sensors.

2.2.2 Lidar

Lidar stands for Light Detection and Ranging. The need to characterize and analyze large vegetation and surfaces of the Erath require large computations and visualization such that certain tools are required that can essentially “scan” the region or ecosystem to collect all the information. Since, manually speaking one cannot go through the entire ecosystem and collect data from small features and details, remote sensing is used. This kind of sensing is digital and does not require physically being present due to the advancement in technology such as drones. These sensory as stated above scan the landscape and record all the details and information to estimate the characteristics of the terrain including vegetation or data across huge surfaces, sensors can be deployed into remote areas and can measure and collect the required information. Lidar helps in acquiring all the information required to generate any surface of which it is scanned. Multiple structures like height, vegetation, density ,etc., can be mapped over time using this sensor. Since it directly measures the height and density of vegetation of the surface it makes it extremely popular tool over large areas. Thus, using Lidar one can build a real-time map of any surface. The sensor can emit a light from a rapidly firing laser which in turn when gets reflected from the trees and branches gets converted into the information which is recorded and collected by the sensor.

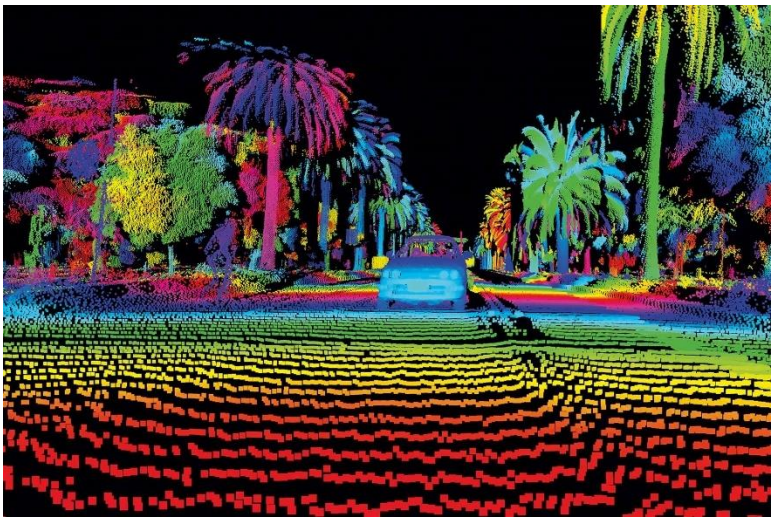


Figure 2.4 Lidar data collected

To calculate the distance travelled which is ten converted into elevation from the surface the time it takes to send a signal to when it hits back is recorded. Once the measurements have been collected, the inbuilt GPS records the position in 3d space as well calculating the light energy and Internal Measurement Unit (IMU) which provides the orientation (yaw, pitch and roll) of the airplane. For trees a photon made up of light particles moves towards the ground and hits the branches on trees which can reflect the light off and then send the information back to the sensor. This works on small like objects but if there are gaps between the surfaces then the light will pass through, some light will continue to hit the ground and so multiple reflections may be recorded. When the light returns to the sensor it creates a waveform. The amount of light being returned is the intensity. Peaks can form in the waveform, depicting that the light hit certain vegetation like leaves, branches, etc.

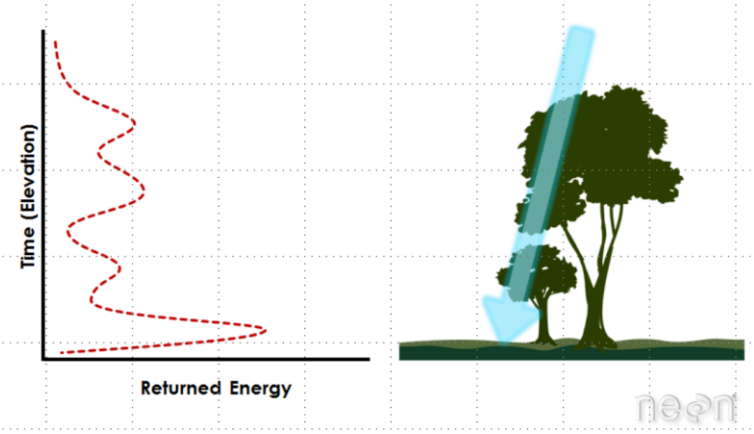


Figure 2.5 Lidar waveform returned from 2 trees and the ground. (Wasser A., 2014)

There are many different types of uses of Lidar, some of them are listed below:

1. To export high resolution data
2. Elevation height
3. Canopy cover
4. Leaf Area Index
5. Vertical Forest Structure
6. Species identification (Wasser A., 2014)

The light energy that is used to scan the surfaces are represented in the form of waveforms. These waveforms can be classified as Discrete Return Lidar System and Full Waveform Lidar System. The Discrete Return Lidar System can identify the highest point in the waveform and record the information at each location. The individual points are called returns. This system has the capability to record a range of 1 to 4 returns from each pulse. Each lidar point contains its own kind of metadata which can be determined by the attributes of said point as well as the attributed itself may vary depending on the information has been collected. All points possess the basic x,y coordinates as well as the elevation (z) attribute. Each lidar point also possess a light energy which is recorded by the sensor the amount of light energy here is represented in intensity values. Apart from the intensity, lidar data also classifies the data which essentially means representing the type of object the laser successfully scanned. So, if the light successfully scanned a tree, it could be classified as “vegetation” and the ground can be classified as

"ground". However, this is not perfect, some of the representation can overlap especially when it comes to infrastructure as there is no "building" type and this cannot be categorized above.

2.3 Modern Terrain Generation Techniques

Modern techniques have been inspired from Darwin's theory of evolution of the species where the continuous survival of the species depends on its adaptability and desire to change with the continuous times and the species are thus rewarded through their own survival. Many modern type algorithms can be divided into three main categories. The first, evolutionary algorithms, are used as a search technique and are advantageous in situations which require high flexibility and adaptability to many different scenarios. This key factor helps in a lot of different fields of study such as, science, engineering, robotics, etc. Apart from using search techniques, through the work being done in artificial life, the theories surrounding natural selection and biological evolution can be validated through this technique. Another prominent technique of modern algorithms is using the evolutionary design which utilizes aspects of design and computational abilities to enhance the design. Using automated techniques with visually appealing design aspects have increased the process and changed the norm. Art forms are very similar in generating new and creative image stop display which requires human evaluation to provide a value based on subjective design. One of the earliest forms of evolution was to incorporate the ability to use mathematical equations and formulae to define the interactivity of computer graphic elements. "Karl Sims used GP to create and evolve computer graphics by mathematical equations. The equations are used to calculate each pixel. He created several graphic art pieces including Panspermia and Primordial Dance *and* allowed visitors to interact with his interactive art system at art shows and exhibitions. His Galapagos is an L-system-based interactive evolutionary computation (IEC) system that allows visitors to create their own graphic art through their interaction." (Frade et al., 2009).

Tatsuo Unemi developed another application, Simulated Breeding Art (SBART) which was open source. This particular software uses genetic programming to develop the mathematical equations required to calculate the coordinates of each pixel. Moreover, mathematical operators can also be used as well as trigonometry functions. These works are based on nodes which are treated as variables. A system of 3 coordinates are computed using the mentioned equations by

assuming the constants are 3d vectors and the variables are 3d tuples (x,y,0) (Frade et al., 2009). Ong et al. were one the of the very first authors to propose generating procedural terrain using he genetic algorithms. Using the evolutionary design optimization technique, heightmaps were converted by applying the algorithm required for the transformation to take place. This approach takes place in 2 stages. Firstly, using a rough 2d map of a surface of the desired terrain which can be randomly generated by the designer is required. Secondly, by using the first stage it helps in removing any unwanted edges and then searches through databases to preselect heightmap data for an optimal arrangement which approximates the 2d surface designed. Both the terrain generated maps will possess variance due to the inherent randomness of the generation algorithm. A more advanced technique which automatically solves many problems regarding the structure and form in advance. Its main advantage is getting the compute to resolve the issues without any intervention. The programs establish a base and learn from the programs to evaluate the weaknesses and remove them. Since it revolves around programable applications, results cannot be guaranteed but this algorithm is useful for artificial intelligence, robotics, prediction, etc. The figure below explains the steps taken by the algorithm to achieve the results. (Frade et al., 2009) .“The primary genetic operators used to create new programs from existing ones are the following:

- crossover: the creation of a child program by combining randomly chosen parts from two selected parent programs,
- mutation: the creation of a new child program by randomly altering a randomly chosen part of a selected parent program”. (Frade et al., 2009)

1: Randomly create an initial population of programs from the available primitives
2: **repeat**
3: Execute each program and ascertain its fitness
4: Select one or two program(s) from the population with a probability based on fitness to participate in genetic operations
5: Create new individual program(s) by applying genetic operations with specified probabilities
6: **until** an acceptable solution is found or some other stopping condition is met (e.g., a maximum number of generations is reached)
7: **return** the best-so-far individual

Figure 2.6 Genetic Programming Basic Algorithm (Frade et al., 2009)

In genetic programming the data structure resembles the Heap data structure, which is similar to a tree like structure. The variables and constants act like leaves in the structure, the mathematical equations resemble the nodes called functions and the sets of functions form the system.

2.3.1 Major advantages and disadvantages of Evolutionary terrain generation

The Terrainosaurus algorithm can be used as sample terrains that will help to meet the requirements of a specific genre if an existing database of game maps exist then this algorithm would be an effective way to ensure that genre maps meet the requirements. This also allows for post-game variations where the user can change certain features of the map layout even after post game completion. This is particularly useful in multiplayer games where exploration is not a top priority but rather developing strategies by learning in-depth information regarding the maps. However, slight variations will force players to change strategies and so it is disadvantageous if enough samples are not provided. Developing the L-system is important as it is based on the fractal techniques discussed in previous sections, so this also possesses all the details and randomness the fractal technique can provide. However, the way in which the algorithm is developed the terrain being generated cannot be distinguished from regular fractal technique and do not inherit the symmetrical nature of the fractal-based techniques. The L-System has a similar limitation where if the appropriate reference terrain is not provided then the system does not produce the correct results. In certain algorithms the implementation is not as effective as their

certain features and properties which upon change causes disturbance to the player and becomes a distraction and will not have an impact on the entertainment value of a game map. Adjusting certain properties such as the feature scale require a pre-designed map and may not provide variety in strategic gameplay. In the GenTP (Genetic Programming) algorithm, the main advantage was the use of many different types of mutation which can prevent the height function from being too aggressive. However, the patterns created by this algorithm as typically unusually flat and do not provide much variance in terms of features of the terrain and are thus highly predictable, such terrains have serious limitations in their applicability to game genres. In the algorithm presented by Togelius et al. was able to generate complete maps which were made possible in many games, however, the terrain which is generated is highly basic and lacks any additional details like soft and round peaks, in a present-day height map.

2.3.2 Fractional Brownian Motion

The Brownian motion uses randomly bursts of increments over time to change the position of an object. These movement although defined to be random but are highly similar in the sense that the zoomed in paths are part of a larger path which resembles the entire structure. Thus, the fractional Brownian motion (fBm) is very similar and possess the same characteristics, in which the increments are not completed independent but there is a sort of a link to each other. Future changes in the same direction will occur only if the link between the objects is positively correlated, including the path would be much smoother. However, if it is negatively correlated the path would be randomized and the positive change would lead to a negative change. The parameter which controls the self-similarity, the fractal dimension is the Hurst Exponent (H). In terms of mathematics, this parameter allows control of the white noise to generate different characteristics and visuals. The H value is between 0 and 1, ranging from smooth fractals to rough ones, the default value is typically 0.5.

Apart from generating natural looking terrains other types of environments can be generated as well due to the self-similarity structures and the randomization. These shapes in the real-world follow the same principles of possessing a few big shapes that give them a basic outline and some smaller shapes which add the details, the smaller shapes also distort the contours and edges to bring out some randomization so that there is some non-uniformity. The amin advantage of using this characteristic is the ease of coding this into any object to procedurally generate the

shapes required. This also helps in establishing level of details, filtering and anti-aliasing, due to these reasons it is widely used in commercial films. The most common method of applying this technique is to use a recursive function which can implement and generate the smaller noises incrementally as the size of the noise decreases, initially, a smooth noise function is used. The pseudo code is provided below:

```
//Create a function with two variables
//Initialize a variable to add to
//Begin the loop
//Compute the noise by using a noise and pow function
//Return the value
```

The accumulation of each noise signal or waves when gets additively combined at the end of the function as well as the horizontal compression which is applied to by reducing the wavelength and amplitude is what creates the self-similarity. However, the code above is not time optimized for dealing with highly realistic and large terrains so another more popular method is used as the expensive pow function is used. The pseudocode provided below does not use that code:

```
//Keeping the major part similar, initialize more variables to replace the pow function
//Begin the loop
//Multiply back the variables with H exponent.
//Add the noise back
//Return the value
```

“In this new code implementation, not only has the frequency generation been replaced from a power-based formulation to an iterative process, but the exponential amplitude has been replaced to decay as well, the Power Law, with a geometric series driven by a "gain" factor G. One needs to convert from H to G by doing G=2-H which you can derive easily from the first version of the code.”

2.3.3 Hydrology

There have been large and various amounts of research which has been done in generating terrain, in the sections above we have discussed many different types, like modelling and

procedural techniques. These are the existing types of techniques which exist today. Procedural and physics-type techniques often lack the controllability and the artistic control that a designer may be going for if creating an exotic type of terrain. Modelling methods can be tedious and time consuming. Among all of these only the physics type techniques provide a far more realistic design than any of the others. As mentioned previously there seems to be a lot of issues regarding terrain generation, another unmentioned issue is the absence of algorithms which provide controllability to the generation of terrains. Furthermore, the scalability of these terrains also becomes an issue since the scale data seems to be fixed in the height map which is the standard data representation in many terrain-modelling systems. The height field can be then converted into a mesh. A way to quickly search and identify terrains is by searching the neighboring areas for a river network. The structure around these network areas have clearly defined structures of the terrain. Apart from this, regardless of the scale of the aforementioned data, the geological and tectonic attributes including the climate do not create any modifications to it and possess identical features despite all these factors. The terrain surface is partitioned into patches due to the rivers on the map.

2.4 GPU based terrain generation

From earlier times, traditional approaches to generation of terrains have always been limited to height maps being generated on the CPU and the process to render (is an automatic process of generating a photorealistic or non-photorealistic image from a 2D or 3D model based on mathematical models) them on the GPU. However, despite the advances in modern technology the CPU is still unable to keep with the demanding tasks that come with generation of highly detailed geometry. For a long time perlin noise has been the base for generating any terrains and similar types of geometry especially exotic features like caves, deep valleys, etc. However, as described above all of these are run on the CPU and require immense memory and computational prowess. In this section as a basic form of terrain generation is provided, tessellation shaders can also be used to generate the terrain in the OpenGL pipeline. No specific algorithm is required to generate any details of the mesh however, a few basic steps are consistent throughout the creation of most terrains. Firstly, the dimensions of the terrain are important and needs to be specified. The terrain should be generated as a flat grid made up of triangles. Secondly, a popular noise algorithm is used in the vertex shader to give the terrain an approximate shape. Thirdly,

tesselate the terrain based on distance from the camera . Fourthly, the normals need to be calculated for the primitives/polygons (triangles) and lastly, the terrain will be shaded based on the normal data and light direction. The vertex shader is the starting point for all processes in the OpenGL pipeline. The perlin noise is used in the vertex shader to generate the randomness present in the terrain which uses a Hermite curve based on input values. The height values should not change with the camera motion, to keep it consistent passing in the world coordinates to the perlin function which will then return the noise values is ensured. This shader runs parallel to each vertex so as to generate a coarse grid. The tessellation control shader is used to control the amount of subdivisions that will be applied to a primitive. Since the distance from the camera is vital, the tessellation amount is directly proportional to it. Other areas which lie outside the view of the camera can be calculated using the projection matrix and the field of view of the camera. The tessellation evaluation shader is called for all the vertices present, the barycentric coordinates are used to calculate the positions of any new vertices. These positions all lie on the same plane as the triangle. After passing through this shader a mesh is created. These vertices are assembled and are converted into primitives in the geometry shader. This is where each primitive face normals are created using the cross product of the edges. Thus, a single normal is formed on each face. After the formation of the primitives and before the pixels can be rasterized, they are colored in the fragment shader which is based on the height of the pixel from the ground. A mix function can also be used to smooth gradient colors from the ground.

Another approach to reduce the load and the computation on the CPU is to use the GPU for rendering purposes. As shown earlier the multiple algorithms have their own pros and cons, thus simply stating the conclusion that this algorithm works effectively without any data cannot be made. Similarly, even in interactive rendering of large DEM’s the effective loading of 3d graphics needs to be prioritized depending on the algorithm in use. Thus, to improve on the performance of the rendering an appropriate level-of-detail (LOD) needs to be selected for each rendered frame. However, certain LOD rendering should make use of the batched graphics primitives for rendering out the vertices instead of basing it in on the individual geometric primitives for optimization. Using the individual primitives may cause excessive load on the CPU which will consume too much time to perform any optimization. The LOD itself has characteristics which can achieve a targeted rendering quality and frame rate.

The serial processing nature of the CPU does not seem to suit the generation of complex and highly realistic terrains which is a highly parallel task as well as the height maps do not include the interesting exotic features which can be seen in caves and other valleys. Nowadays, to overcome the limitations provided by the CPU the GPU is looked at to enhance and effectively eliminate in speed deficiency that can be found. More complex terrains can be generated with high interactive frame rates due to the GPU. Features like the DirectX10, the geometry shader (GS), stream output and rendering to 3d textures, makes the generation of the terrains relatively faster especially the generation of larger blocks of a complex terrain.

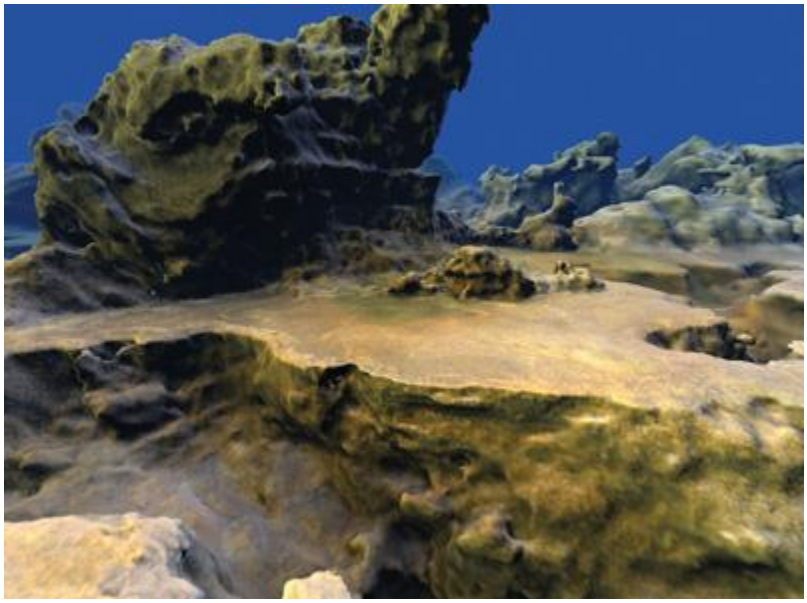


Figure 2.7 Procedural generation of terrain using the GPU

The simplest way to describe the creation of the terrain is by using the density function. In any given point in 3d space (x,y,z) the function produces a floating point number and if the said number is positive then the point in space is inside the solid terrain otherwise if its negative then the point is located in an empty space such as air or water. The location of the point if in between these two values, where the density value is 0 then it is located on the surface terrain. Along this surface is where the polygonal mesh is then created. The GPU is used to generate “blocks” of terrain at a time and then these blocks are subdivided into smaller voxels or cells. These voxels are used to generate the appropriate polygons required to generate the terrain. The marching cube

algorithms allows us to determine if an arbitrary point lies within that object and bounds within that object exist as well as generate correct density value in the eight corners of the polygon within a single voxel. The infinite world where the terrain needs to be generated is lined up with 1x1x1 size blocks, within each size of the blocks lies 32^3 voxels that potentially contain polygons. These polygons exist inside the block only if they are inside the camera frustum as these blocks have the higher priority since they are closet to the camera. 300 vertex buffers are dynamically assigned to blocks visible in the frustum and as the new bocks come into focus the newly culled blocks are evicted and reused. The polygons inside the blocks also depend on complex calculations where only after the block is generated a stream out query is put out to the GPU asking whether the polygon exists or not. If not, these blocks are considered to be empty and are put inside a list to increase efficiency and so it won't be uselessly regenerated. In every frame the same steps are followed to build a systematic and consistent terrain, the vertex buffers are sorted from front to back and then new blocks are generated while evicting the older generation blocks. Eventually, the sorted blocks are rendered in the same process to reduce the load on the GPU and to reduce the time consumption on the shaded pixels which might be occluded by other features of the terrain.

2.5 Volume based Terrain Generation

An improvement in the heightfield data structure is to use a layering technique of more than one height value per 2d terrain position. These additional layers can be used effectively to simulate different types of rock-based structures just like in the earth's surface. Each layer of these structures contains a different material type which allows for different rates of water absorption and erosion simulations. This kind of layered data structure compromises between the realism and accuracy of the terrain relative to a potentially slower but more flexible voxel data structure. The main property of an ideal data structure for realistic terrain generation is to represent boundaries as efficiently as possible. This data structure should also allow for enough flexibility to generate overhangs. Terrain algorithms with this data structure will lower the total computational processes. Simulating is not the final process for any 3d geometry thus interactive rendering is also a process which is associated with this data structure. Surface based data is used quite extensively for rendering rather than volumetric data and so a polygonal mesh is extracted,

however, mesh extraction can be quite expensive. A water layer can be modelled and added as an additional layer above the heightfield normal strata.

Due to the fact that vertically overlapping vertices are allowed, a Triangulated Irregular Network (TIN) unlike heightfields does not use regular 2d data and are irregular meshes which consists of triangle primitives which can fully represent 3d features. Unlike pixels which comprise of flat rectangles in 2d space, voxels (Volume Pixels) are similar to the representation of pixels, instead being a 3d representation. The main limitation of voxels is the storage requirements needed to contain highly detailed terrains. Since an increase in resolution of voxels will increase the overall resolution of the terrain this will increase the storage space required. Using voxels comparatively to heightfields can provide a far more precise physical simulation data due to the unique characteristics of each voxel. A densely populated layered heightfield representation is similar to using voxels. The process of editing a 2d heightfield is relatively straightforward when compared to editing a 3d voxel grid thus a technique called voxel carving was developed. The naiver-stokes equations can be used to establish the flow of water. This allows the flow field to be calculated using mathematical models and physical behavior instead of using an ad hoc attempt. A hidden advantage of using a voxel-based terrain is that it avoids any self-intersection that arises from using surface base topologies. The LOD is directly proportional to the size of the voxel. The LOD itself can be adaptively increased in certain sub-volumes of a data set. This increase in adaptivity must also be met by the renderer. A voxel is represented by a single bit where 1 represents the voxel to be “full” and 0 to be empty and so this can be stored in a 3d array. If the data set has dimensions of n^3, the memory used is also represented as n^3, the cost of this can grow quite large.

CHAPTER 3. METHODOLOGY

3.1 Introduction

In short, the main idea regarding this process was to increase computation performance and speed while reducing the time taken to simulate certain conditions. Since the GPU was not being used and since most of the simulations in Houdini run on the CPU, a technique called clustering was used to distribute the highly detailed terrain meshes into multiple parts. The imported data has a high resolution as it is making use of the satellite to gather the data, Lidar data is a form of satellite data which can produce realistic terrains, the higher the point count the more the features and thus an increase in realism. However, even computing such high data points require high computational power and so distributing them into clusters or groups of points can potentially save that time. The clustering technique is based upon the k-means clustering algorithm in which the algorithm will group similar data points together based upon position to discover underlying patterns. This algorithm is extremely popular in unsupervised machine learning techniques and is what is used to define the clusters, control on the number of clusters is also given. However, just creating clusters is not enough as that itself takes computational time so a tool was developed to use the cluster attribute as a source of export and save each cluster's positional data points to a file on the computer which can be parsed and imported back in.

This technique distributes the data into clusters and write out all the points into separate files based upon the user defined number of clusters. Since each cluster is a portion of the entire file, the details of the natural terrain have been kept as points from within the data have not been removed to decrease computational time, instead a portion of the entirety has been segregated thus reducing the computational time but keeping all the features and details. Once the particular cluster has been imported back in, the procedure to convert it into a heightfield has commenced, which will convert it into a volume-based terrain and uses Houdini's many customizable nodes to re-structure and modify the terrain since the points itself cannot be modified directly but upon conversion given the variety and considerable number of nodes to provide that extra artistic control and creativity the terrain itself can be modified. Thus, one of the major questions to navigate the labyrinth of terrain generation is providing a realistic terrain which does not

overload the performance of the computer, although there has been a substantial reduction in the overall cost of accessing and owning high-performance based computers, the underlying statement is that not everyone has said access nor can purchase those computers despite the decrease in cost. Moreover, small to medium sized companies cannot afford such advanced technology for all artists thus using the distributive system as well as those who own personal computers may not have a high-performance computer and so this technique will keep the realism intact while not overloading the machine. The simulation data will be produced by the software itself using the heightfield erosion node which will create the time taken in seconds and compare that data with the simulation time taken when using the distributive tool. The data will be recorded by Houdini's performance monitor itself, the moment the simulation starts the data will automatically start recording and will stop when the simulation stops. The performance monitor figure is provided below.

Figure 3.1 The performance monitor

As we can see from figure 3.1, the part which is most important is the heightfield erode node and the time taken to complete the simulation. This value will be used to compare the data which is monitored after applying the cluster distribution technique. As the color becomes more reddish in color from a green color the slower the simulation or longer the time has been taken, essentially a more reddish bar color takes longer time to simulate than the nodes with a green bar color. This also shows the processes and the time taken for all the other nodes, apart from this the node itself is highlighted with a red outline emphasizing that node is slower to "cook" or simulate relatively.

Although this procedure is not a standard form data collection and methodology, it particularly works for understanding how long and how much time is taken to complete the simulation although it does not give us a thorough explanation as to the reason it does narrow down the particular node in question and can provide details of that node and information regarding the time taken. Also, we can use the "+" button on the left-hand side and a drop-down menu with more nodes will appear which can tell us which sub-node inside this node is using most of the computational prowess. An example is provided below.

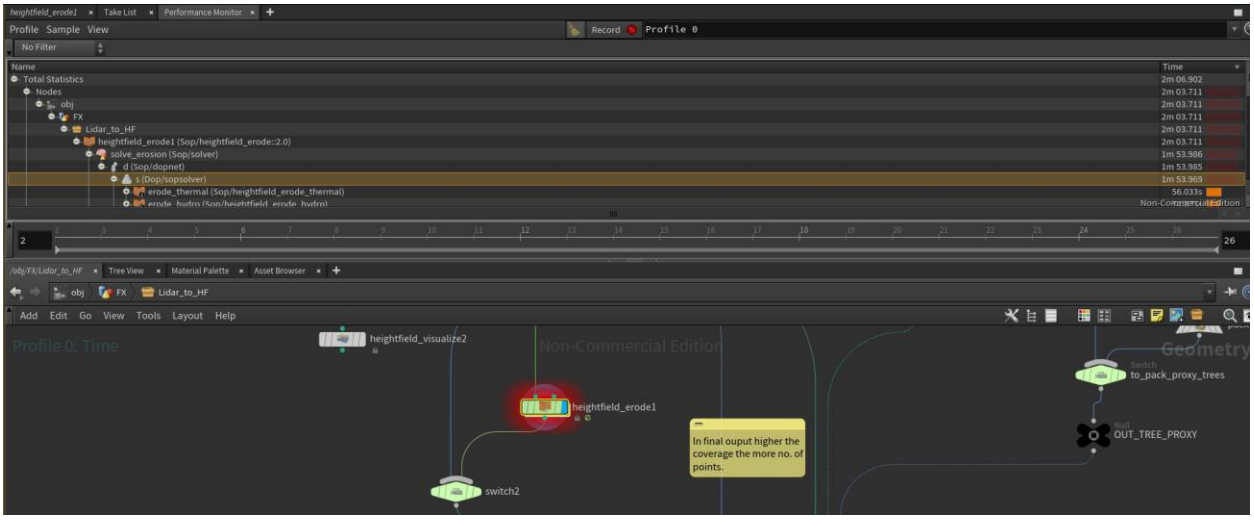


Figure 3.2 The performance monitor with the highlighted node

The information in the figure provided clearly shows that the highlighted node is the affected one in respect to the amount of time taken to simulate and the computers computational abilities. Moreover, we can dive into the node and it will show us the exact node which is taking the longest time. As shown in the figure below, the highlighted part tells us where the computational time is slowest and which part of that sub-node does that node main exists.

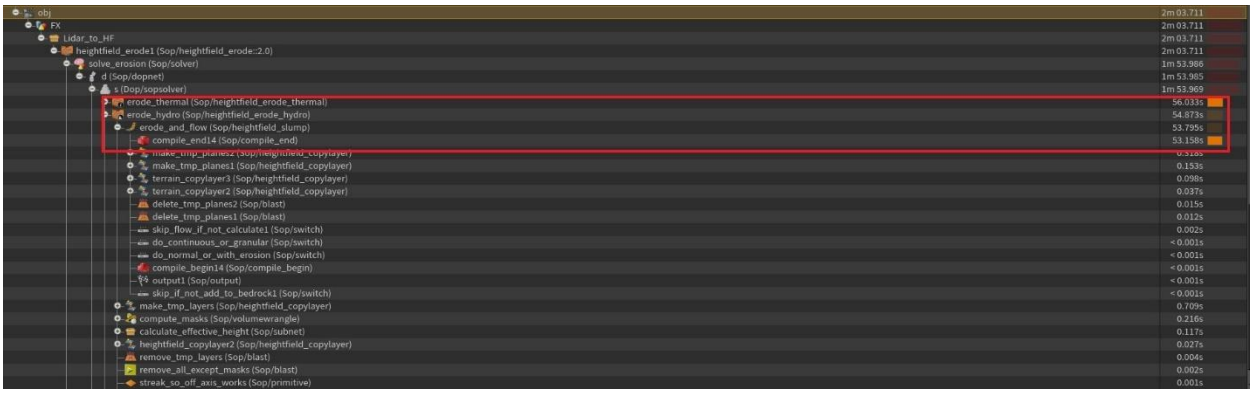


Figure 3.3 Highlighted performance monitor

Using the software's inbuilt functionality, the validity of the performance is guaranteed as the performance monitor uniquely computes the time taken for every node and displays the results of each by also using a color visualizer ranging from green to red. Since this in an inbuilt system which comes with the software itself and no tertiary application is being used the test accurately measures the parameters which it is intended to measure. Since this is a simulation in the digital world there is no real-world counterpart to compare it to, however, since the application itself is able to test the performance of each node then validity remains intact as it is not being influenced by any external factors. Apart from validity, for quantitative data the reliability requirements also need to be met. Re-testing the same simulation on the same computer specifications and with the same control variables the results were highly consistent with very minimal changes in the time taken. Since the re-test reliability is essential, there are no unaccounted parameters which exist. However, if the changes are frequent then this can be attributed to the state of the computer itself meaning, is the computer itself working at the optimal peak. Since dealing with computer generated applications, the control variables will be the specifications of the computer itself, these variables cannot change as then the results will be heavily biased towards the higher-performance computer. In this experiment the time taken to simulate although does depend on the specifications of the computer itself but also on the quality of the terrain, in other words, the amount of points generated for the heightfield map. The higher the number of points the more the detail and thus an increase in the time taken. So, the independent variable here is the number of points used to generate the terrain and the dependent variable here is the time taken to complete the simulation.

3.2 Overall Approach

The overall approach which was used here has been divided into three main stages, first the pre-processing stage, then the simulation stage and finally the processed stage. Then first stage was completely dependent in acquiring the data to make any significant contribution to this field of study. Using data acquired from a public resource which can be downloaded by anyone with an internet connection. Lidar data was used as a basis to gather the points and create the data set required to build and generate the terrain inside the 3d software (HoudiniFX). The lidar data was used, as the software stated above, possesses a node which can allow the import of such data smoothly without having to rely on an external application to first read the data and the export it out in a format which can be easily processed. However, since HoudiniFX also boasts of having a robust foundation in its cross-platform abilities, it contains a python module as well - called the HOU module. This module is extremely versatile as it allows us to do multiple modifications to the different sub-parts of a geometry namely, the points, primitives(polygons) and vertices. Using the module one can import the data into HoudiniFX. Assuming we were trying to read in the points using the python language from a text(.txt) file and the text file was like the figure provided below.

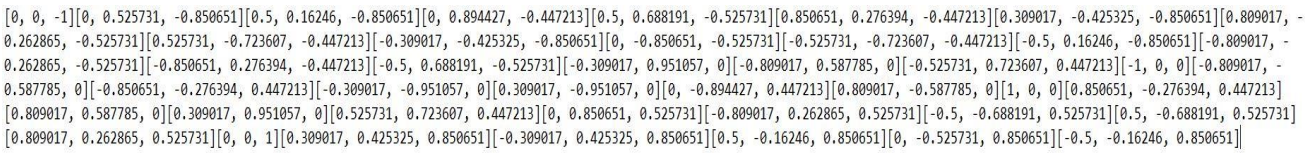


Figure 3.4 Point data from the text file

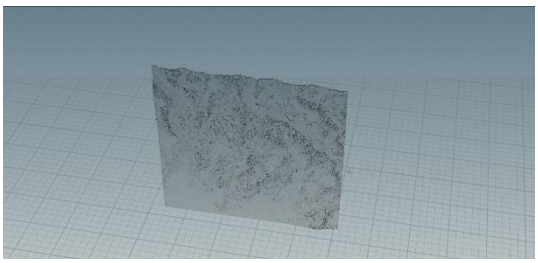
```
#The hou module calls all the built-in functions for houdini
node = hou.pwd() #This is calling the current node
geo = node.geometry() #This is calling the current geometry(points,primitives,etc.)
file = open("points.txt","r")
for lines in file:
    lines = lines.replace("]","]_")
    result = lines.split("_")
res = []
for x in result:
    res.append(x.split(','))
```

```
X = []
Y = []
Z = []
for item in res:
    if not item[0]:
        continue
    #print(item[1])
    points = geo.createPoint()
    X.append(item[0][1:])
    Y.append(item[1][1:])
    Z.append(item[2][1:-1])
    X = list(map(float,X))
    Y = list(map(float,Y))
    Z = list(map(float,Z))
    for i in X:
        x = i
    for j in Y:
        y = j
    for j in Z:
        z = j
    points.setPosition([x,y,z])#set position is the built-in function to set the position of
#all points in x,y,z co-ordinates
```

The major drawback of using python to read in the data as shown in the code above, is the amount of computational time it will take for pre-processing the data to a format which is readable. Since the node is far more streamlined and can read in the data far more easily the node has been used. In other words, parsing the data to a format which can be used to import the file in python will take more time than simply using a pre-existing node.

After the dataset has been imported in completely, most of the operations will be modified and calculated at the sop(surface operator) level. The lidar data typically can be distributed into two

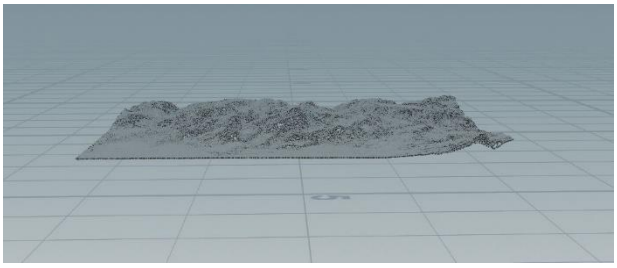
separated files which can be imported as a single file, however, for easier computation it is necessary to divide them into two smaller sized files. It is important to note that a higher data point count will result in slowing the performance of the computer, so it is imperative to start off with a smaller sized dataset and then to move into a larger size one. Once the correct size of the data set has been chosen data points went through a transformation sub-stage. The transformations applied here were done so that the data points itself when imported do not contain a "x,y,z" co-ordinate system which is used inside any 3d application including HoudiniFX, since this was the case any kind of importation which will occur will arrive in the 3d space without user control away from the (0,0,0) position. This can cause other issues like camera orientation and zooming in and out may be difficult since the camera and the data points do not lie on the same plane so to overcome this small limitation a transformation was applied using the extensive node-based library. The before and after images can be found below.



Center	643,350,	6.68596e+06,	112.935
Min	641,758,	6.68439e+06,	-20.65
Max	644,942,	6.68753e+06,	246.52
Size	3,183.31,	3,142,	267.17

(a) Raw data set (b) Raw transformation values

Figure 3.5 Image and values of raw data from Lidar



Center	0,	116.91,	0
Min	-1,570.94,	0.0300064,	-1,571
Max	1,570.94,	233.79,	1,571
Size	3,141.88,	233.76,	3,142

(a) Image after transformation (b) Data after transformation

Figure 3.6 Image and values after transformation

As we can see from the above data the initial transformation values are high (Houdini deals in meters.) thus to bring the values to a more readable stage, transformation of the values was

applied. Unfortunately, the points which are imported may have a complicated shape which makes it hard to create an infinite plane so using an expression, the bounding box of the imported data points are calculated to get the basic shape and then subtracted by a seed value to only keep the points inside the reduced bound so as to create a shape in the form of a square. The expression is given below.

min(bbox("../OUT_BOUNDS/",D_XSIZE),bbox("../OUT_BOUNDS/",D_ZSIZE))

Another potential issue was taking care of the "outlier" points, in the case of building a terrain from point positions the user needs to be careful so that the points do not lie outside the main the squarish region that was computed using the expression stated above. Since there might be a very small number of points that could be part of the data set that are not within close proximity of the minimum bounding regions of the original surface, these points have to be removed from the data set before the next phase. To deal with such situations Houdini established nodes called wrangles, an image is provided below.

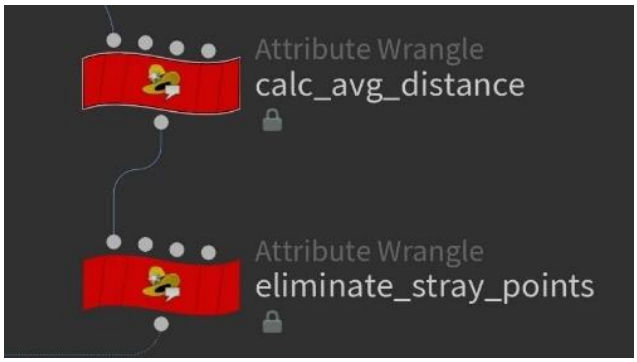


Figure 3.7 Wrangle nodes

These wrangles are essentially small code blocks where C based language can be written, inside Houdini this is called VEX(Vector Expressions). First, we calculate the average distance between the current point and the points closest to it. Then the distance between each of these points are calculated.

```
//Calculating the total number of points
int total_pts = npoints(0);
float total_dist = 0;
//Sampling a random number of points to iterate through
int sample_pts = int(total_pts*chf("Threshold"));
for (int i=0;i<sample_pts;i++){
    int random_pt = int(rand(i)*total_pts);
    vector pos = point(0,"P",random_pt);
    /*If one is using the same input as the near points is then it will
    use the earlier point position as a look up.
    The list will comprise of our points as well as the nearest point to out point.
    */
    int neighs[] = nearpoints(0,pos,chf("Max_Distance"),chi("Number_of_Points"));

    //If we find the neaighbour points then calc the avg distance.
    if(len(neighs)==2){
        //Calc the position of our neighbour.
        vector neigh_pos = point(0,"P",neighs[1]);
        //Measuring the distance between our point position and the neighbours.
        float dist = distance(neigh_pos,pos);
        total_dist += dist;
    }
}
f@avg_dist = total_dist/sample_pts;
```

Once the average distance has been calculated the second part of the code is where the removing of the points needs to happen for the points to be deleted.

```
float avg_dist = detail(0,"avg_dist");
//Keeping a good margin of error
float thresh_dist = chf("Margin") * avg_dist;
```

```
//Using an array to gather all the near positional points based upon a threshold value
int neighs[] = nearpoints(0,@P,thresh_dist,chi("Number_of_Points"));
if(len(neighs)<2){
   //this is a function inside of houdini to remove points according to the point number.
   removepoint(0,@ptnum);
}
```

3.2.1 Houdini's Clustering Method

There exist many types of nodes in the sop level (surface operator level) which is also known as the geometry level. One of the higher-level nodes which exists is called the clustering point node. This node can cluster a set of points based on their position or any given(created) vector attributes. These attributes could even be based on forces or color. The basic functionality of this node is to create a set of colored points or clusters indicating the point number(integer) or to the closest cluster they may belong to. Thus, all the points which have the same point position will be allocated to the same cluster. The algorithm which creates these clusters is based on the k-means algorithm. This algorithm is very susceptible and sensitive to change especially by the randomly chosen initial cluster center values for a small set of clusters. This essentially means if there are two clusters A and B, of points which are not in close proximity to each other there is a possibility that all the points belong to cluster A and none belong to cluster B as the random states of each cluster are based upon a seed value which is an inherent parameter of this node and provides the randomization. To change this the seed value parameter until the appropriate clusters are formed is a way to solve this potential weakness. The development of a faster computation system depended on the usage of this clustering technique which served as the backbone of the method. Dividing the clusters into 4 groups(clusters) initially served the purpose of avoiding the potential weakness stated above and in doing so created equal parts of the cluster. Since the position of the points were static and not animated the computational time was saved even further as in case of a terrain the user will not change the initial position of the points. After the creation of the cluster, the tool was created in python to increase the cross-platform capabilities. The Houdini-based python code would write out the positions of each clustered point into any type of file on the number of clusters defined by the user. For example, if the user had set 5 to be the number of clusters then the code will write out the positions of each point

related to every cluster. This basically helped in diving up a large data set according to the performance of the computer as well as preserving the details of the terrain as the total point count was not decreased instead it was only split to accommodate the overheating of the CPU. However, the main advantage lied in not writing out these points but also reading them back in, since a single large data set was split into user-defined smaller data sets(text files) the user can choose which cluster to read back in if there were an odd number of clusters. Running all the simulations and generating the terrain on a single cluster preserved the details for that section of the terrain but also increased computational capabilities. Given below is a low-resolution terrain with four clusters as well as the code used to write out the points to a text file.

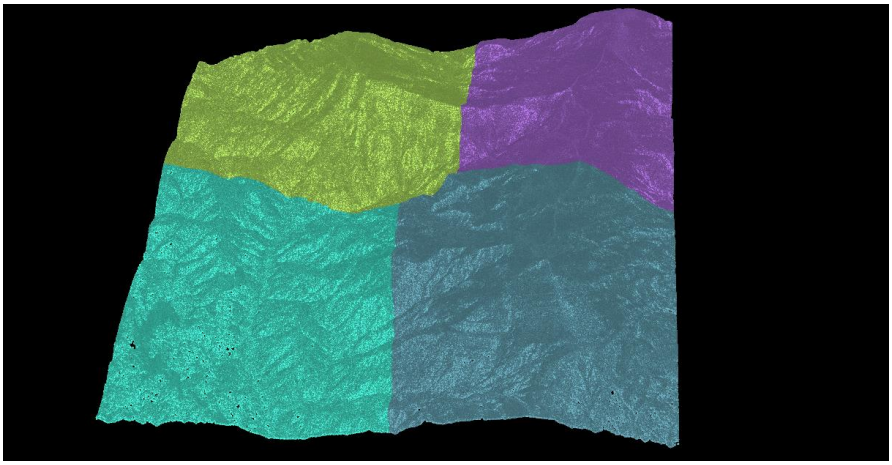


Figure 3.8 Terrain with cluster

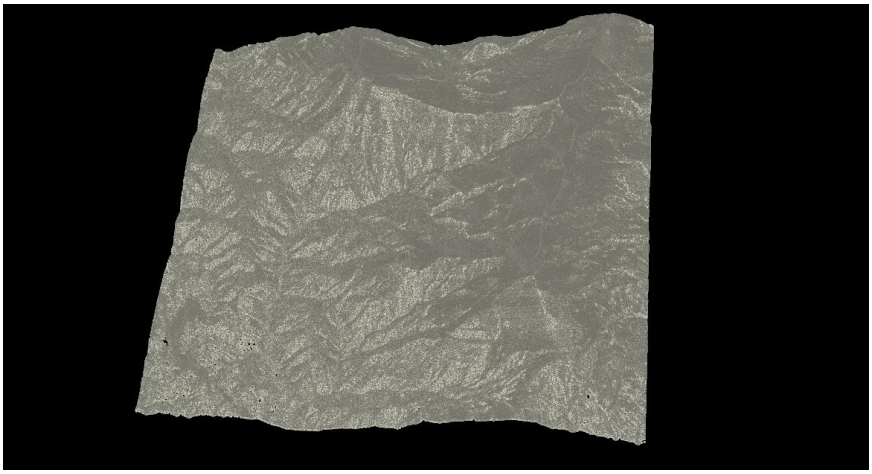


Figure 3.9 Terrain without cluster

The code below was used to call in and parse all the points so that they may be created at the same position in world space from when they were written.

```
node = hou.pwd()
geo = node.geometry()
# Add code to modify contents of geo.
# Use drop down menu to select examples.
file_eval = hou.evalParm("filename") //This is done to directly call the file name from when the points were being written
with open(file_eval,"r") as file:
    data = file.read()
    for line in data.splitlines():
        if not line:
            continue
        line = line[1:-1].split(",")
        x = float(line[0])
        y = float(line[1])
        z = float(line[2])
        finalPos = geo.createPoint()
        finalPos.setPosition((x,y,z))
```

To further enhance and make the pipeline development far more efficient I have also added an external drop-down list of the types of interface the user would like to write the cluster data with namely, the in-text file format or csv file format. Since these two formats are commonly used and are freely available for any software. Simply choosing and running the script allows the user to write all the points from all clusters to be saved in that file type. Furthermore, this is not limited to only the writing part but also the reading part where the python script can automatically read in the correct file type and extension provided. This is done using an hscript (Houdini script) expression to determine whether the value is true or not, if the value is true, i.e., 1, read in the csv file else read in the text file.

`ifs(chs("../writing_out_a_cluster2/output"),"$HIP/new_text_"+ch("cluster_points")+".csv","$HIP/new_text_"+ch("cluster_points")+".txt")`

The above expression essentially states that if the value of the node in the first bracket right after "ifs" is equal to 1 then run the first statement, which tells it to read in that particular cluster file which the user has selected marked with the ".csv" extension, otherwise read in the same cluster file which the user wants marked with the ".txt" extension.
Not only this but all these files are also being saved in the same directory that the Houdini(hip) file has been saved. Since the initial project was saved in a folder using the expression everything gets saved there. However, the above-mentioned expression is only a tiny part of the process as that simply switches between a true and false value of the filet type. The actual reading in the data is handled by the python script inside the software which can read in the data fairly easily since its reading in each file separately depending on the amount of clusters present instead of reading an entire file with all the points generated in a single file. Although the data is separated out, pre-processing the data is equally important, since the data cannot be imported in if one directly uses a node as the file type does not support it, even though, if one uses an extension type which is supported by the software even then the points are unbale to be read in, the particular node in question is a simple file import node but it throws an error. This was tested on a smaller data set of points stemming from a very simple geometry like a grid. This was dome to test the effectiveness and to check if the code works, as well as the result and output is correct. The process to create such a testing environment is easy if you have the software. The process has been provided below:

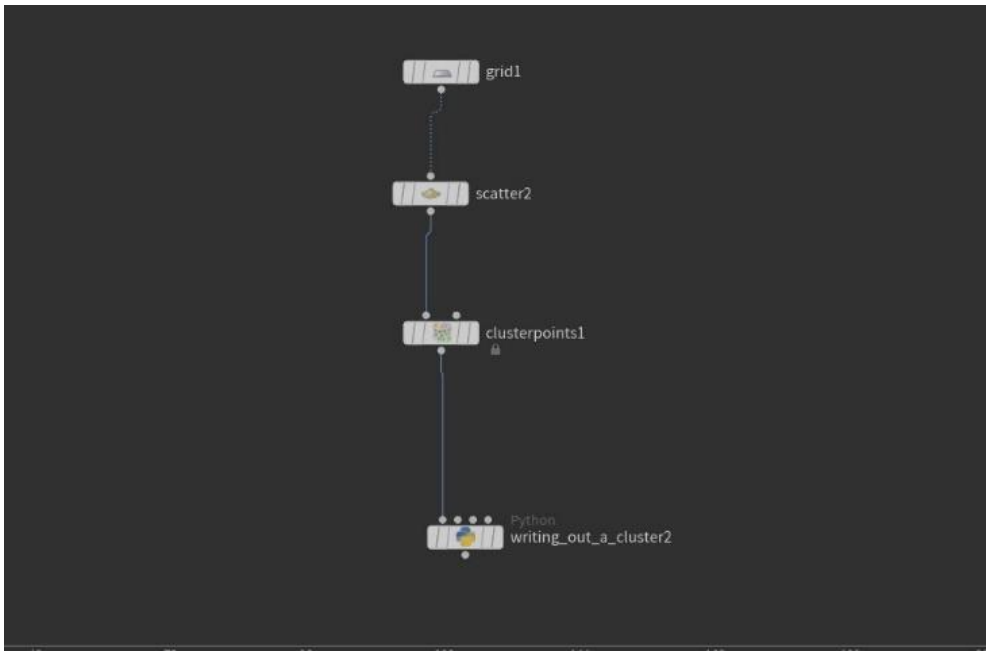
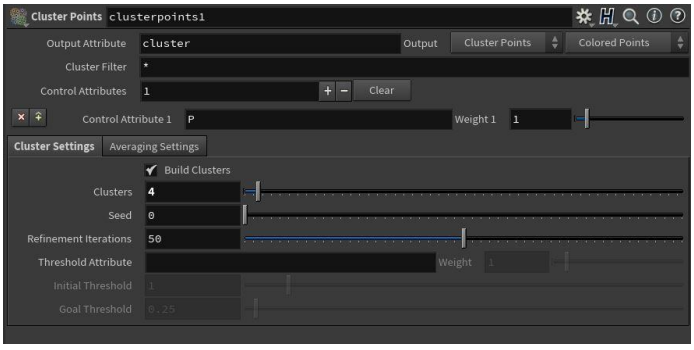
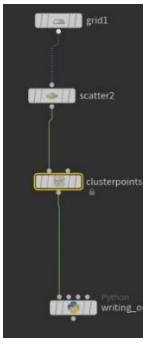


Figure 3.10 Displaying the test process of the creating the cluster.

In the image shown above the process is similar to the process done while generating the terrain, where the first node (grid1) is replaced by any geometry this can be triangles, quadrilaterals or any kind of complex shape which the user may desire. The next step is converting the mesh (geometry) into points, this is done using the scatter node. One can simply skip this test if the points have already been generated. The next step is important as this is where the number of clusters can be defined, once you have selected the node a secondary window will open where the total number of parameters can be modified as shown in the image below:





(a) Parameters window (b) Selected node

Figure 3.11 The images show the parameters of the highlighted node.

The next step would be to write out the data that has been generated by using the number of clusters as a trigger. Simply running the code will not be able to write out the data since there exists a secondary parameter called the type of extension which has been provided to not be limited to only one type of data file. This can be done using Houdini's parameter interface, which is an inbuilt option, this allows us to change, modify, add and delete different parameters depending on the needs and desires of the user, since here we are adding the parameter to change the type of extension we will need to make sure to add this parameter inside the node which is running the script. The parameter interface will modify that node, which is selected, thus it is easier to select the node first then apply the change in the interface. The image of the interface is provided below:

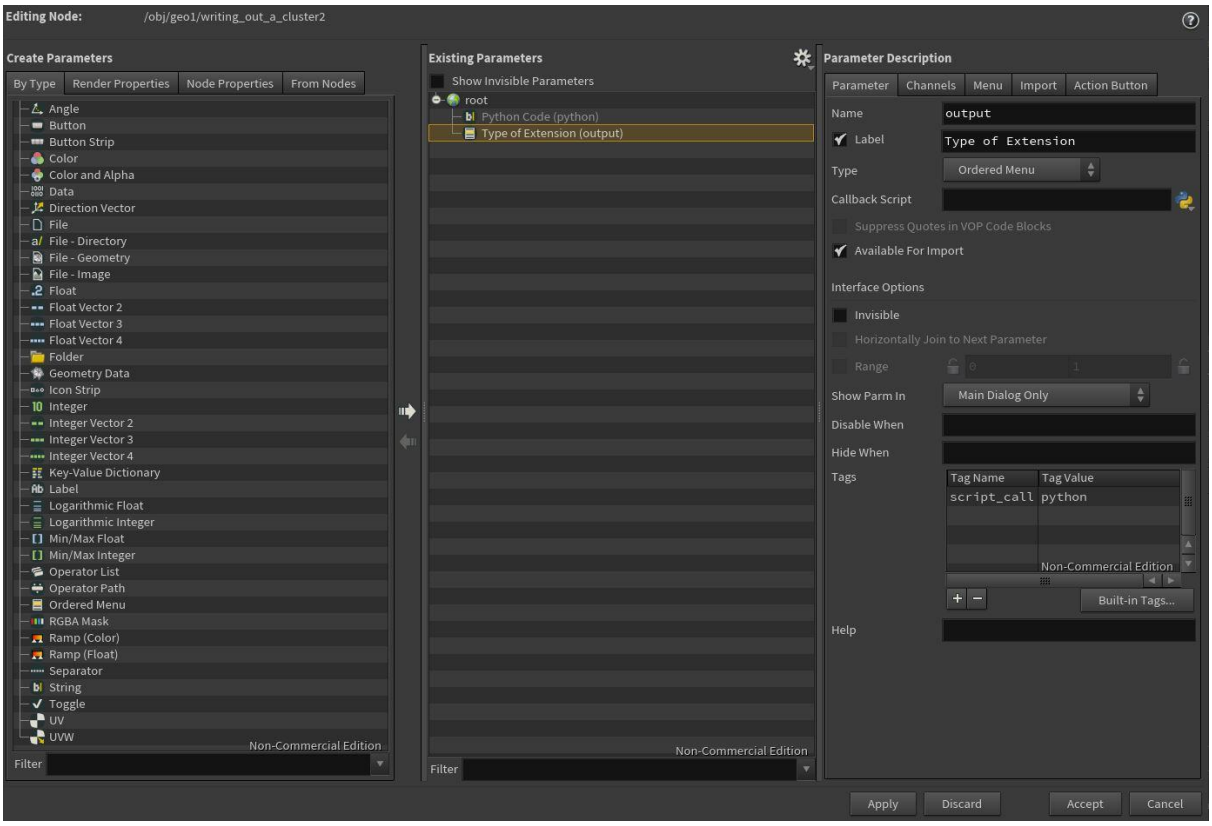


Figure 3.12 Selected changes to the parameter interface

Once the selected parameter has been applied, this parameter can be used in the python script as a variable to control a variety of objects. In this case it is being used to provide control for the user to switch between the text file extension and a csv file extension. By making the parameter a

drop-down menu button it also makes the interface easier to understand and control. Once the cluster has been written out, the data in the file needs to be read in, however since we are providing control for the number of the cluster and also since the software is unaware of which file type is being imported that parameter needs to be created as well. Since this is a procedural creation of the terrain the exported output will always be inside the project of the and in the same directory as the software file itself, thus it will always be read from there without any errors. Houdini has its own functions for reading in the data which was used as well as making sure the data which was created had been divided into three sections (3 float vector), where the x-co-ordinate had all the points of the cluster's x-co-ordinate position and this process was repeated for the y-co-ordinate and z-co-ordinate. Finally, once the respective point positions matched the data which was exported out the positions had to be set suing the same co-ordinate system.

3.3 Converting points to heightfield

At this point the terrain is still being as displayed as points and needs to be converted into a heightfield to visualize the terrain. A wrangle node and a heightfield node was used to convert the points to generate the terrain as a volume, the data being created is a heightfield data, but the resulting visualization is a volume. The "height" attribute is used to in accordance to the near point of the position of the same point being calculated. As mentioned before the ground points and the vegetation points have been segregated so they need to be rejoined together to display the terrain as whole, however, the separated vegetation points also have to be taken into account and need to be accounted for in the entire terrain and so a ray node is used to a group of points which will be based on "minimum distance", this states that the points which are closest to the surface of the points of the secondary (collision) geometry. Once the points on the collision geometry have been selected and placed in the group, a node is used to generate a mask around the terrain to visualize the areas where the vegetation is required as shown in the image below. The area of the vegetation has been shown in red and is procedurally generated so if there is any change in the resolution or the number of points (point group) in the area regarding the vegetation then it will be automatically updated.

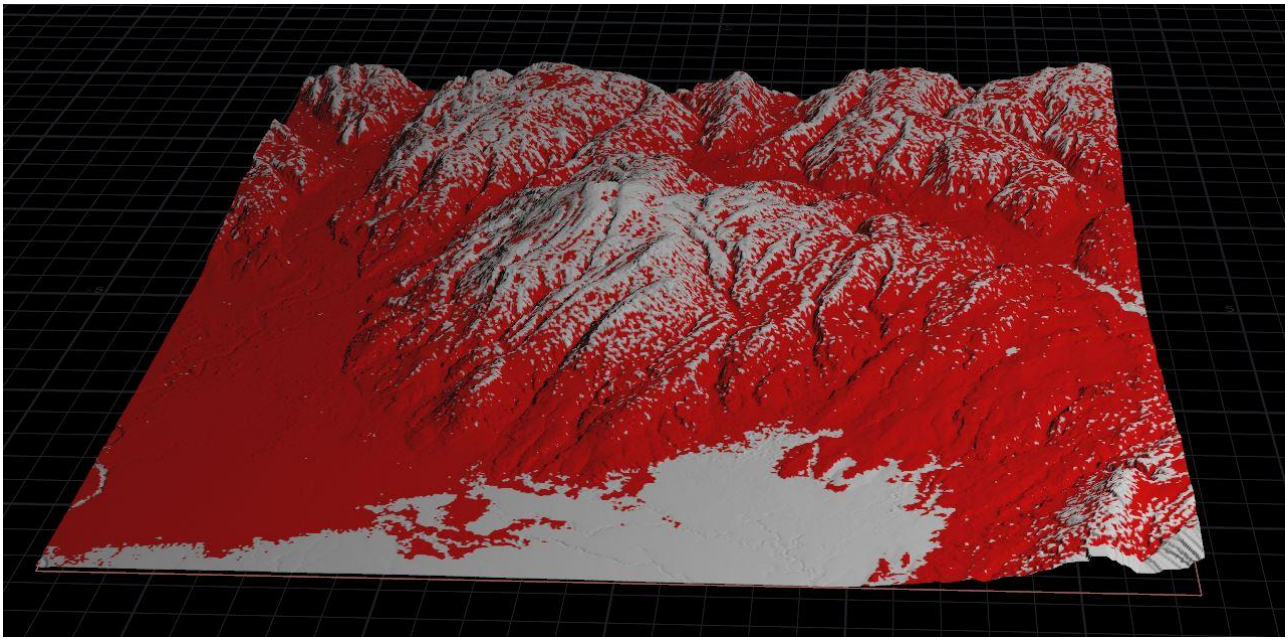


Figure 3.13 Procedurally generated vegetation

Once the vegetation area has been established in the terrain this part will be later used to copy a user defined amount of flora to provide a more realistic and detailed terrain set. As shown in figure 3.8, the terrain seems to possess a large amount of details and makes a substantial use of the noise that the heightfield algorithm produces to create the randomness that is necessary. However, this also creates potential issues, where even the edges have been affected by the same amount of noise. In terms to create an infinity plane or generating a terrain that stretches for infinity especially in a gaming environment where the same game requires exploration, a large terrain will be required. However, having a non-flat surface will cause issues as there will be no smooth interpolation between the edges of multiple terrains. Thus, turning the borders to a more flattened surface will be beneficial from a gaming standpoint, the solution to this weakness is straightforward. A secondary object or even a curve can be used to draw a boundary around the initial terrain to define how much of the noise around the borders needs to be kept. The area which is not required to be kept is placed in a mask using the heightfield mask node, another process is to create masks upon masks so that the terrain doesn’t look too uniform and natural, since this is being created for the VFX and Gaming industry it needs to look visually appealing as well. The blur to the edge adds that "appeal" factor so a heightfield blur node is used as well. Below is the basic approach in flattening the borders.

(a) Curve mask without blur (b) Curve mask with blur

Figure 3.14 Representing the mask created by a curve

Using the curve, however, breaks the proceduralism process as the curve is user defined and does not use any expressions to generate as each different user may have a different idea as to how the mask should be presented as. Another, more procedural approach can also be used to generate the mask using another geometry or mesh such as a sphere or box. The sphere will then act as a container to keep the part which requires the mask and remove the unnecessary aspects. Since one is using a sphere an already in-built expression can be used to get the maximum and minimum bounds of the terrain to always constrict it to that size only. Thus, once those values have been computed and applied, we can always take that final output and divide the number a margin value to decrease the bounds since the mask needs to be smaller than the size of the actual terrain. Thus, this method works very well with the clustering method as since the cluster will be smaller than the larger terrain, instead of using a one-shot procedure like the curve, a more modern approach to the sphere is used to auto-update the information of the minimum and maximum bounds. Although, at this juncture the mask has been created once needs to remap this mask attribute (layer) to the original terrain to bring out the area defined by the sphere. A node called heightfield layer can automatically remap the layer as well as tweaking some parameters

may be able to provide a blend between the parts of high detail and the borders. The figure below shows the terrain after sphere masking.

There is notable difference between the figure above and figure 3.6 in terms of border flattening. Using the sphere mask has significantly reduced the noise in the borders as well as flattened the edges to provide a more comfortable interpolation between multiple terrain when creating an infinite terrain. From the perspective of an artist, having multiple control parameters is extremely important. Thus, in this application both sphere mask technique and a curve technique has been used to cover all the basis required. To increase artistic control in the generation process of this terrain, the curve mask technique has been implemented and left as an option. Interestingly, another weakness of the application had been revealed, since the curve is a node that requires an input as well as the sphere requires an input, only of these inputs could be used at a time, essentially either the curve or the sphere mask could be used separately but not together. So, the user would have to switch between the two inputs manually depending on the requirements and

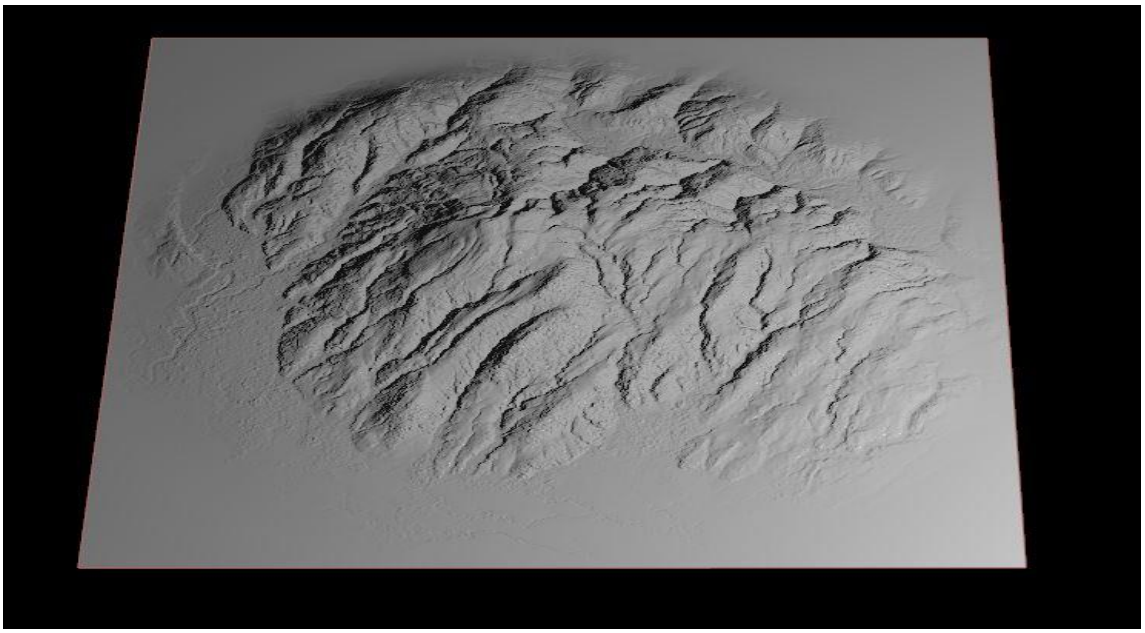


Figure 3.15 Complete terrain model with procedural sphere mask technique

the needs to visualize the details of each. To combat such a situation where the user could procedurally switch between the inputs depending on if the node input connection existed or not, meaning if the input connection of the curve exists the value becomes true and the true value

switches the masking technique to a curve input otherwise when the value is false it retains the sphere mask technique. Not only does this provide the additional artistic control of using the curve but also initiates a smoother workflow while also saving time to manually switch the input value. Finally, this has created a highly realistic terrain with artistic control from many parameters and expressions which has been used. The final stage of the application is left and from where the data regarding the simulation time will be monitored. The heightfield erosion node is responsible for creating and generating erosions in the heightfield. This simulation is important as it helps in creating a realistic look of the terrain and filtering and smoothing out the high values of the noise, as there might be some features in the terrain which are look good but are too exaggerated and therefore inaccurate in the real world so the erosion helps in removing those excess values and creates a more pleasant overall look.

3.4 Data Collection

As mentioned above the data collection is handled by the performance monitor which can pinpoint the exact node which takes the longest time to simulate and process. This plays a vital role as using this product of the software, the data can be acquired quite easily and without using an external application to process the results which also avoid any kind of bias. The product can apart from just telling the time taken also makes use of a color-coding visualizer tool which can easily depict which nodes are taking longest to process and "cook". Using this tool, calculating the amount of time taken to simulate the erosion and the generation of a complete terrain while also keeping the same specifications of the computer as well as keeping the same number of points to reduce variability and bias in the experimentation process. The comparison will take place between the distributive clustering method and saving the data erupting from that against the data coming from when the technique is not used. Also, increasing the number of points and running the same comparison can also help establish a link between the resolution of the terrain and the time taken for simulation while using the tool, against not using it. So, specifically how much time is taken to simulate the same resolution of the terrain when using a different method to process the same number of points. Thus, as observed the independent variable is the number of points given while importing the terrain and the dependent variable is the amount of time taken to process and simulate the points in generating the terrain. Here we predict the correlation between the variables especially between the number of points (resolution) and the time taken

while using the distributive method. A specific testable hypothesis is required to address the research question. The null hypothesis being the use of the distributive method does not decrease the total time taken to simulate regardless of the number of points. The alternative hypothesis being the distributive method does decrease the total time taken to simulate regardless of the number of points.

3.5 Data Analysis

In this section, the analysis portion of the data will be discussed. Initially, upon collecting the raw data the analysis was almost negligible as the difference between the two methods was quite large. However, simply providing the raw data to let others draw inference was not the best option, since the desire is to provide a significant difference between the two values a viable statistical method needs to be used. Most of the data being used here is quantitative data, the independent variable being the number of points and the dependent variable being time. A simple t-test would be enough to reject the null hypothesis. The t-test is used to measure a significant difference between two group's average and whether that difference reflects appropriately in the population. To acquire the correct method of analysing, the number of points has been changed and the time taken before and after the clustering method have been recorded. This test has been chosen due to its simplicity in being able to significantly differentiate the results and reflect it appropriately to the population, since the data is purely based on values and are quantitative this test seems most appropriate.

CHAPTER 4. RESULTS

The results of the experiment are as expected, however there were some interesting findings as well. To quickly iterate what has already been mentioned before, a t-test is being conducted to provide significant different results which can state the statistical correct results. In this experiment the control variables are several but the most important are the specifications of the computer which are running the simulation. Since an increase in computing performance can influence the runtime of the entire simulation the same specifications of the computer have been used. Apart from this, the resolution of the volume has also been kept same as well the total number of frames the simulation needs to run for. The burden of proof here has been kept on the alternative hypothesis as this needs to qualify as being superior.

The null hypothesis (H_0) is, there is no difference in the time taken to simulate the terrain after applying the distributive method regardless of the number of points.
The alternative hypothesis being, there is a difference in the time taken to simulate the terrain after applying the distributive method regardless of the number of points.
As mentioned before there is a burden of proof on the alternative hypothesis so the following figures show the evidence required to reject the null hypothesis.

Table 4.1 Data collected by the performance monitor

Number of Points	Before the clustering method (Time in seconds)	After the clustering method (Time in seconds)
95,040	34.5	30.8
188,756	38.4	35.8
581,342	40.5	32.8
1,279,268	167	51
1,613,319	75	45
2,910,555	109	54
3,931,178	157	85
4,824,538	179	162
5,523,809	224	94
7,653,255	272	130
8,511,389	321	140
9,574,711	332	124
10,137,846	381	171
11,198,292	409	158

From the table it can be seen that, the number of points keep changing as this is the independent variable, keeping all the control variables same, there is definitely change in the time taken to simulate. However, to determine if the changes are actually significant or not the t-test analysis was run. According to the analysis given in the table below, the Alpha value used here was 0.05 which is the default. The p-value for the one tail is less than the alpha value which can be stated as $p < 0.05$, one tailed, this means that the there is a significant difference between the two groups and the null hypothesis can be rejected. However, for the two tailed tests, where the p-value < 0.05, two tailed is more than the alpha value and so the null hypothesis not rejected. This is further proved by the fact that the t-statistic is greater than the t-critical which suggests that the null hypothesis can be rejected.

The t-test here has been run on equal variances as the data being collected and processes is based on simulated data and not observational data in which the amount of time taken does not change so substantially and so the variance does not change as dramatically either.

Table 4.2 T-test Results

t-Test: Two-Sample Assuming Equal Variances

	Without the clustering method (Time in seconds)	*With the clustering method (Time in seconds)*
Mean	195.6714286	93.81428571
Variance	16941.13451	2770.649011
Observations	14	14
Pooled Variance	9855.891758	
Hypothesized Mean Difference	0	
df	26	
t Stat	2.714516884	
P(T<=t) one-tail	0.005814598	
t Critical one-tail	1.70561792	
P(T<=t) two-tail	0.011629196	
t Critical two-tail	2.055529439	

The study is to evaluate any link between the speed and performance of whether the tool being used can save and increase the performance of the time taken to simulate and generate a realistic terrain depending on the change of amount of points being processed. Since the null hypothesis was rejected based upon the statistical analysis run and from the results in the previous slide. There is a significant difference in the time taken when running the simulation before the distributive method versus after utilizing the method. This means that there is a definite increase in the performance in the computer along with the speed of the simulation. This also allows for using the time saved to create multiple variations and apply any modifications to the terrain as directed making it more art-directable. Since the time has been significantly reduced this allows for multiple iterations and even re-testing without losing excess time. Moreover, the natural resolution of the terrain has been preserved. To bring all the analysis to a logical conclusion a graph was used to visualize the change. The figure is provided below.

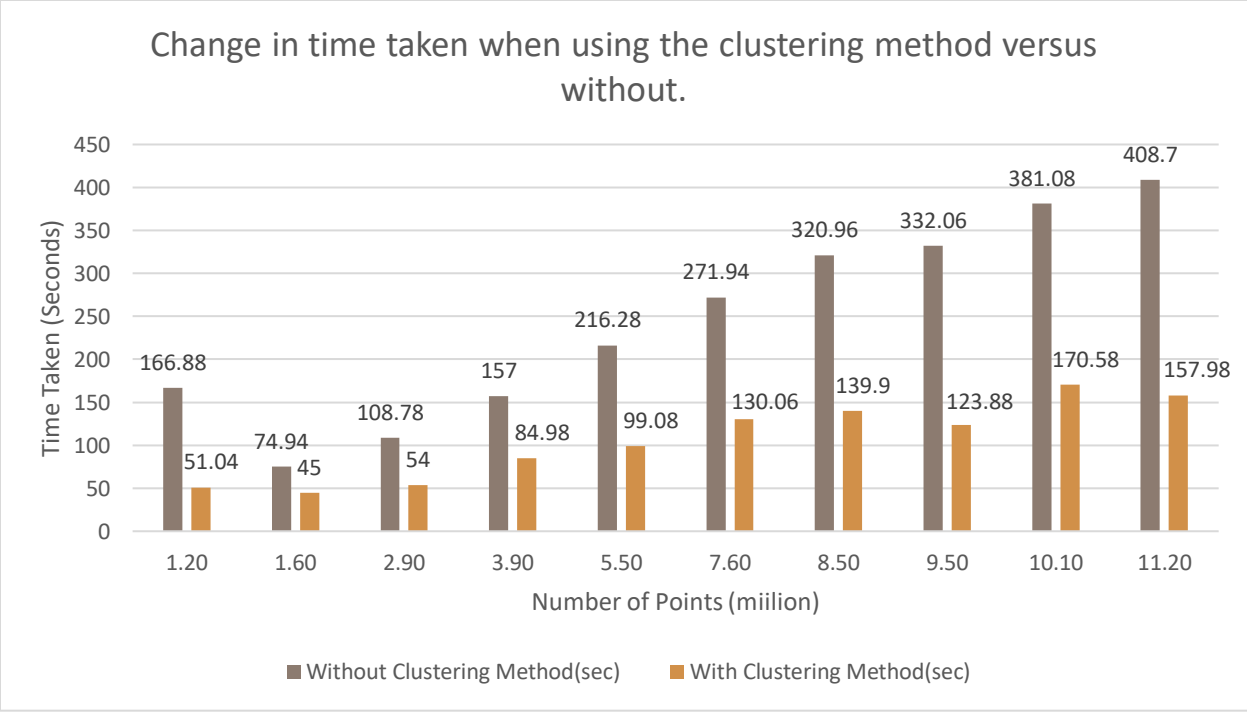


Figure 4.1 Time change in the use of clustering on the terrain.

As seen in the figure above, not all observations have been used as this was done to avoid any overlapping between the bars. So, 10 random values were chosen from table 1 to display the time taken to simulate the terrain with and without the clustering method.

CHAPTER 5. CONCLUSION

In the above-mentioned experiment, the time taken has significantly reduced and there has been an overall positive change in the computation process in distributing the terrains into clusters. As speed and time efficiency are essential parts of any simulation, time optimization is extremely vital in defining and creating any algorithm or novel method. In this case, the terrain was pushed to be as realistic as possible, meaning, the resolution of the terrain was increased to the limit of the computer's capabilities. Terrain generation and proceduralism itself have many problems, creating a credible environment in a limited amount of time and how to do it without affecting quality. Procedural generated terrains will lack the storage and retrieval capabilities as the data set provided would be too small while the randomness would be much too great, without the aesthetic design required for it to look good. Thus, collecting and analyzing the actual speed and performance of the simulation and generation of the terrain and to what actual level of detail can they perform to would benefit not only production companies but also educational institutes who need to keep up with the advent of novel technologies.

Manipulating the terrain also creates variations in the structure itself, a heightmap is the most common way of representing a terrain and any changes in the areas of a heightmap will produce a different result in them. Whether larger crevices, valleys or peaks are required, the heightmap can procedurally modify whatever structure as needed. The heightmap is a scalar quantity meaning it has only a x-coordinate and a y-coordinate which corresponds to an elevation variable(h). Since height maps are based on noise which use a color range of 0-1, where 0 is black and 1 is white, grey scale images can also be used to construct them. However, heightmaps cannot represent cave-like features. It is generally generated on a grid but cannot be accommodated to use a spherical object or a more complex object as the density of the height map will vary directly, the points will be substantially greater near the poles than the middle of the object. The terrain generation are mainly divided into 3 techniques, measuring, modelling and procedural technique. Lidar data was used to collect the raw data points and import them inside the application. GPU based terrain generation is also possible which might reduce the load on the memory of the CPU and reduce the time taken to simulate.

The study has a major limitation. Since the computer being used did not possess a large enough memory to accept huge chunks of data, a magnanimous amount of data points could not be tested on. Another limitation is in the development of the python tool itself; the tool can only be used on points only and will only export the “cluster” attribute. The prospects are extremely intriguing especially in further enhancing the resolution and distributing the simulation. An application or tool developed in python which can run a custom-made k-means algorithm will be developed, which will automatically segregate the points into clusters thus removing the need for any nodes. Perhaps using the GPU to run the clustering and exporting of the points can be developed to reduce the load on the CPU as well as building the tool to export and save any custom attribute created to a file locally.

REFERENCES

[1] A Complete Guide to LiDAR: Light Detection and Ranging. (2015, August 23). *GIS Geography*. https://gisgeography.com/lidar-light-detection-and-ranging/

[2] Frade, M., Fernandez de Vega, F., & Cotta, C. (2009). *Breeding Terrains with Genetic Terrain Programming: The Evolution of Terrain Generators* [Research Article]. International Journal of Computer Games Technology; Hindawi. https://doi.org/10.1155/2009/125714

[3] O'Brien, N. (2018, August 14). *Diamond-Square Algorithm Explanation and C++ Implementation*. Medium. https://medium.com/@nickobrien/diamond-square-algorithm-explanation-and-c-implementation-5efa891e486f

[4] Wasser A., L. (2014, October 23). *The Basics of LiDAR - Light Detection and Ranging—Remote Sensing | NSF NEON | Open Data to Understand our Ecosystems*. https://www.neonscience.org/lidar-basics

[5] Miller, G. S. P. (1986). The definition and rendering of terrain maps. *ACM SIGGRAPH Computer Graphics*, *20*(4), 39–48. https://doi.org/10.1145/15886.15890

[6] Sharma, B. (2019, January 30). What is LiDAR technology and how does it work? *Geospatial World*. https://www.geospatialworld.net/blogs/what-is-lidar-technology-and-how-does-it-work/

[7] Rusek, M., Jusiak, R., & Karwowski, W. (2018, September) *Algorithms for Random Maps Generation and Their Implementation as a Python Library: International Conference, ICCVG 2018, Warsaw, Poland, September 17—19, 2018, Proceedings*. ResearchGate. Retrieved April 13, 2020, from https://www.researchgate.net/publication/327641270_Algorithms_for_Random_Maps_Generation_and_Their_Implementation_as_a_Python_Library_International_Conference_ICCVG_2018_Warsaw_Poland_September_17_-_19_2018_Proceedings

[8] O'Neil, S. (2001, March 2). *A Real-Time Procedural Universe, Part One: Generating Planetary Bodies.* Retrieved April 13, 2020, from https://www.gamasutra.com/view/feature/131507/a_realtime_procedural_universe_.php

[9] Freiknecht, J., & Effelsberg, W. (2017). A Survey on the Procedural Generation of Virtual Worlds. *Multimodal Technologies and Interaction, 1*(4), 27. https://doi.org/10.3390/mti1040027

[10] Bösch, J., Goswami, P., & Pajarola, R. (2009). *RASTeR: Simple and Efficient Terrain Rendering on the GPU*. 8.

[11] Guérin, E., Digne, J., Galin, E., & Peytavie, A. (2016). Sparse representation of terrains for procedural modeling. *Computer Graphics Forum, 35*(2), 177–187. https://doi.org/10.1111/cgf.12821

[12] Miller, G. S. P. (1986). *The Definition and Rendering of Terrain Maps*. *20*(4), 11.

[13] *Chapter 1. Generating Complex Procedural Terrains Using the GPU*. NVIDIA Developer. Retrieved April 13, 2020, from https://developer.nvidia.com/gpugems/gpugems3/part-i-geometry/chapter-1-generating-complex-procedural-terrains-using-gpu

[14] Thomas, T. (2012, December). *Tijutv/GPU-Terrain-Generation*. GitHub. Retrieved April 13, 2020, from https://github.com/tijutv/GPU-Terrain-Generation

[15] Hoang Anh, N., Sourin, A., & Aswani, P. (2006, December). *Physically based hydraulic erosion simulation on graphics processing unit*. ResearchGate. Retrieved April 13, 2020, from https://www.researchgate.net/publication/220979051_Physically_based_hydraulic_erosion_simulation_on_graphics_processing_unit

[16] *A survey of procedural terrain generation techniques using evolutionary algorithms*. (2012, May). ResearchGate. Retrieved April 13, 2020, from https://www.researchgate.net/publication/260366081_A_survey_of_procedural_terrain_generation_techniques_using_evolutionary_algorithms

[17] de Carpentier, G. J. P., & Bidarra, R. (2009). Interactive GPU-based procedural heightfield brushes. *Proceedings of the 4th International Conference on Foundations of Digital Games - FDG '09*, 55. https://doi.org/10.1145/1536513.1536532

[18] P. Wilson, J., & C. Gallant, J. (2000, January). Digital Terrain Analysis. *Current Biology*, *7*(3), R126. https://doi.org/10.1016/S0960-9822(97)70976-X

[19] Zhou, H., Sun, J., Turk, G., & Rehg, J. M. (2007). Terrain Synthesis from Digital Elevation Models. *IEEE Transactions on Visualization and Computer Graphics*, *13*(4), 834–848. https://doi.org/10.1109/TVCG.2007.1027

[20] Santamaría-Ibirika, A., Cantero, X., Salazar, M., Devesa, J., Santos, I., Huerta, S., & Bringas, P. G. (2014). Procedural approach to volumetric terrain generation. *The Visual Computer*, *30*(9), 997–1007. https://doi.org/10.1007/s00371-013-0909-y

[21] Miller, G. S. P. (1986). *The Definition and Rendering of Terrain Maps*. *20*, 39–48.

[22] Thorimbert, Y., & Chopard, B. (2018). *Polynomial methods for fast Procedural Terrain Generation*.

[23] Ritchie, D. (2016). *PROBABILISTIC PROGRAMMING FOR PROCEDURAL MODELING AND DESIGN*. Standford Univeristy.

[24] Kahoun, M. (2013). *Realtime library for procedural generation and rendering of terrains*. Charles University.

[25] Genevaux, J.-D., Galin, E., Guérin, E., Peytavie, A., & Benes, B. (2013). Terrain Generation Using Procedural Models Based on Hydrology. *ACM*, *32*(4), 143:1-10.

[26] Greeff, G. (2009). *Interactive voxel terrain design using procedural techniques*. Stellenbosch University.