A Comparison of two GPU Triangular Mesh Simplification Algorithms in Vulkan on Generated Convex Terrain

William Vickers-Hastings, Falmouth University, Games Academy

***Abstract*—As more games incorporate large planetary scale terrain procedural generation the importance of optimising these models is critical to increase scalability and to leave sufficient hardware power to maintain the current quality of modelled assets. This paper compares two GPU Mesh Simplification Algorithms to evaluate their effectiveness and performance on procedurally generated convex terrain meshes. By examining their execution time and geometric deviation to determine the quality of their output, they will be compared to a low sample-rate terrain generation to determine if a significant detail increase can be had at the same polygon count by using one of these algorithms. If this is true, the two algorithms will be compared to determine which algorithm is most optimal in terms of geometric error per unit of execution time for runtime terrain simplification. This paper will discuss algorithms selected for comparison, how they will be benchmarked in Vulkan.**

***Index Terms*—Mesh Simplification, Mesh Decimation, Retopology, GPU Compute, Vulkan Compute, Compute Shader, Procedural Mesh, Terrain Generation**

I. Introduction

Mesh simplification is an intensively studied field since the 1991 [1], [2]. Robust CPU based algorithms and since 2007 [3] GPU based algorithms have been freely available. At time of writing there has been a growing interest in 3d procedural terrain generation in games with titles like No Man's Sky [4], which is based on a procedurally generated universe, and others [4]–[9] promising a procedurally generated environments to explore. Optimisation of terrain meshes is challenging as detail in the mesh is lost as polygon count is reduced. Two intuitive ways to optimise terrain meshes;

- Generate a low resolution height map. This reduces generation time and increases rendering performance but has a major quality impact compared to a high-resolution height map, as a lot of detail can be lost due to increased distance between points on the height map (low sampling rate). This downside can be algorithm dependent and can even be an artistic constraint.
- Generate multiple terrains at various resolutions then use a level of detail system, to draw the high-resolution mesh as needed. This is often a workable trade off but the disadvantages of having to run the terrain generation algorithm multiple times and potentially extra overhead of slicing the terrain into tiles or chunks, which can be a limitation with some terrain generation processes, like hydraulic erosion [10].

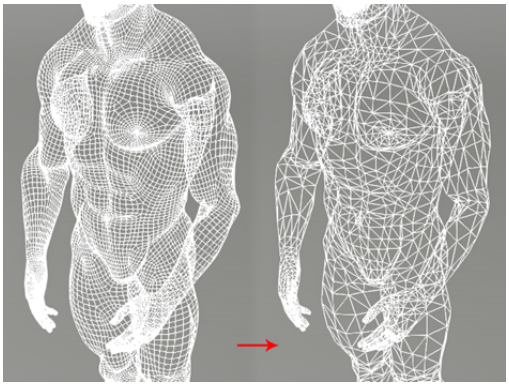


Fig. 1. Example mesh simplification of a character model from Character Creator 3 [11]

In 3D modelling, mesh simplification (also known as decimation or optimisation or retopology) is a tool for reducing the geometry in a model such as for characters, while maintaining a high level of detail.

Applying such an algorithm to a high sample-rate terrain mesh should produce a higher detail result at the same level of geometry compared to running the terrain generation at, low sample-rate with equivalent geometry count to the simplified mesh.

This paper aims to compare the following two GPU based algorithms to low resolution generation and determine which one is better for runtime convex terrain generation:

- Algorithm A, Embedded Tree Collapse, proposed by Hyunho Lee and Min-Ho Kyung in *Parallel mesh simplification using embedded tree collapsing* [12]
- Algorithm B, GPU Half Edge, proposed by Mohamed H. Mousa and Mohamed K. Hussein in *High-performance simplification of triangular surfaces using a GPU* [13]

II. Contextual Analysis

A. Mesh Simplification

To simplify a mesh is to maintain the shapes and details of the mesh with less vertices and triangles (geometry or polygon count).

Two reasons for meshes simplification:

- **Data size** Smaller meshes take up less space on the disk and in the system memory and GPU frame buffer at runtime.

- **Rendering performance** The GPU does not need to do as much work to render a smaller meshes, allowing the application to render more meshes, or render more frames in a given unit of time.

Since the early 1990s [1], [2] Many CPU based algorithms have been put forward for mesh simplification [14]–[22], with one of the most cited and widely used algorithms known as QSlim [15] proposed in the 1997.

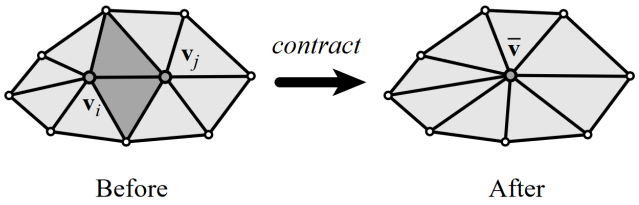


Fig. 2. Example of a single half edge contraction to simplify 10 triangles into 8, from figure 1 of [15]

QSlim removes vertices from the mesh using a process known as half-edge collapse (see figure 2).

A half edge is simply a vector towards another vertex - only connected vertices can be collapsed. A vertex may have several half-edges. [15] What makes QSlim so cited is its proposal of a quadric error matrix (QEM) to rate the loss of quality removal of each vertex would have on the mesh. [15] The algorithm contracts vertices with the lowest cost until it reaches are target reduction rate. [15]

Algorithms developed since [3], [12], [13], [16], [18]–[20], [22]–[24] nearly universally use the same theory of edge contraction and QEM as a base, both CPU and GPU.

The challenge with mesh simplification is reducing geometry while maintaining a high level of detail - trying to do the same with less. A further challenge comes with simplifying in real-time (sub 16ms execution) in a video game. It can take multiple seconds [15] to simplify a mesh depending on its size, which make it impractical for real-time implementation in video games using consumer hardware even at the very high end, at time of writing.

After an extensive review, there is a lack of research on procedurally generated geometry [14], [25] and doing mesh simplification in real-time in games. [3]

Why there may be a lack of research on procedurally generated geometry could be because to reduce the geometry of a generated mesh is simple, so long as you can control the mesh size of the generation. Higher detail results should be achievable using simplification algorithms as they can put more polygons into high detail areas of the mesh at the cost to flat areas (see figure 3), instead of a uniform density found in simple heightmaps (see figure 4).

1) Quantifying Results: To quantitatively measure the quality of an algorithm, the geometric error of the simplified mesh compared to the original mesh is calculated using several methods, geometric deviation [27], [28], Hausdorff distance [29] and/or Root Mean Squared [12], [15].

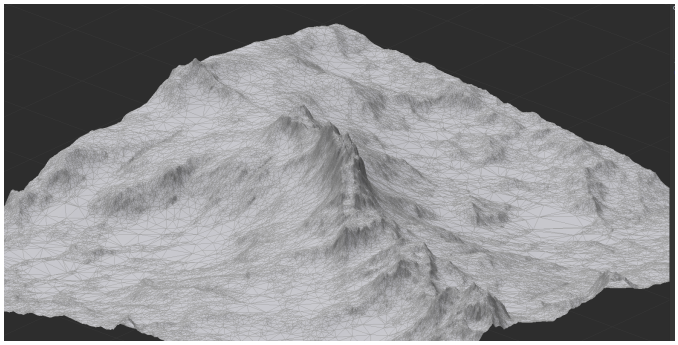


Fig. 3. Optimisation that reduce geometry in flat areas and puts the extra vertices into extreme elevation areas. Produced in World Machine [26]

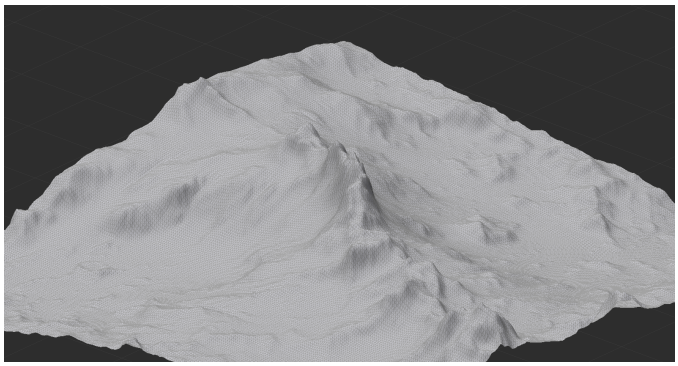


Fig. 4. Uniform grid terrain generation vertices are displaced only vertically more extreme height changes lead to larger triangles while flat areas have unnecessary geometry. Produced in World Machine [26]

Geometric Deviation is widely used [13], [22]–[24] and is the method chosen for this paper.

Geometric Deviation finds the minimum euclidean distance between each vertex on mesh A and point on Mesh B where the point is within the plane of any of Mesh Bs Triangles. This distance is the deviation for each vertex. [13], [15], [27], [28] Averaged all these distances together produces the mean geometric deviation a metric that is used to quantify the difference between two meshes. [27]

Given two meshes, meshes A and mesh B, their faces, and a point that belongs to a face of mesh A, the geometric deviation for a point and face in mesh B is defined as the minimum euclidean distance between the point in mesh A and a point in mesh B, where the point in mesh B is on the plane of one of its faces. [27]

2) GPU Algorithms: As GPUs have gotten more programmable since the late 2000s [30], GPU implementations of mesh simplification algorithms have been proposed [3], [12], [13], [23], [24], a paper from AMD in 2007 [3] was the first to take advantage of the high parallelised performance offered by GPUs on large data sets like multimillion polygon meshes and was also the first paper to put forward an implementation of mesh simplification that could be used in real-time in a game.

From an extensive review, most GPU papers build upon or directly compare to the previous paper. Some papers compare directly to results within other papers for instance

[13] compares directly to results in [24] which is an issue as there are major differences in hardware making execution time incomparable between them.

The current bleeding edge GPU algorithm was proposed by Mousa & Hussein in 2021 [13], which compared only to QSlim [15] and Multiple-choice scheme [24]. However, they referenced a paper from 2016 by Lee & Kyung which proposed an algorithm using embedded tree collapsing [12], which they did not compare against.

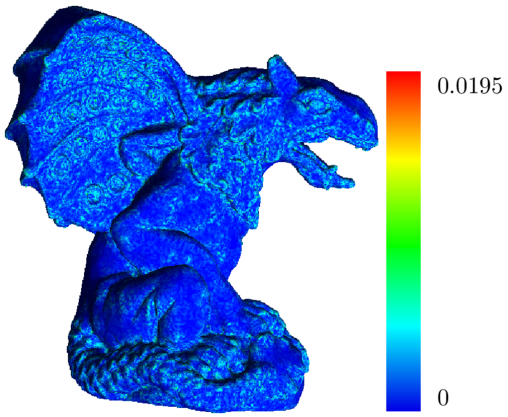


Fig. 5. The gargoyle model used in [13]

Both Mousa & Hussein and Lee & Kyung papers, used a gargoyle (see figure 5), which appears to be the same model, but the two papers used slightly variants of the model, different vertex and triangle counts. They did use directly comparable simplification (reduction) rates. The results of the simplifications look visually similar, but as Lee and Kyung used MeshLab [31] to measure Hausdorff distance [29] and Root Mean Square [12], [15] to use as their quality metrics, there is not a way to compare directly to the Mousa & Hussein algorithm, as they used geometric deviation [13], [27], [28].

The execution time for the algorithm proposed by Lee and Kyung are orders of magnitude lower than the Mousa & Hussein algorithm [12], [13] after accounting for the face Lee and Kyung used milliseconds(ms) instead of seconds. Lee and Kyung used a more powerful GPU, which likely explains this. Both algorithms were implemented using Nvidia's CUDA tool chain [32].

Given that Mousa & Hussein algorithm is more recent, it is expected that it will perform better in execution time given the same hardware, and also have a lower geometric deviation compared to Lee & Kyung's algorithm.

Based on the results in both papers, Embedded Tree collapse [12] seems competitive to [13] despite being much older, which is why this paper will compare the two.

- Algorithm A, **Embedded Tree Collapse**, proposed by Hyunho Lee and Min-Ho Kyung *Parallel mesh simplification using embedded tree collapsing* [12]
- Algorithm B, **GPU Half Edge**, proposed by Mohamed H. Mousa and Mohamed K. Hussein *High-performance simplification of triangular surfaces using a GPU* [13]

B. Embedded Tree Collapse (ETC)

This algorithm's objective is to maximise GPU utilisation by breaking data dependency in the updates of the mesh. Algorithm B [13] does not do this, instead opting to use wait commands before moving on to the next contraction step. The way ETC works, is by storing edge update information in a table and then updating the mesh data with that information in the next step of the algorithm. This allows a higher number of edges to be chosen from for contraction in each step. All of this allows each step of the algorithm to perform more contractions of the mesh compared to [13] which does only 1 per region. Unlike [13] ETC has a quality parameter, which can be tailored to the mesh its simplifying to minimise quality loss at the cost of performance. [12] [12] suggest a high value of 0.1 for real-time applications.

The operations of Embedded Tree Collapse is broken down into 6 steps. [12]

1) Calculate quadric error matrix (QEM) at every vertex. Quadric error is used to measure the detail cost to contract a vertex [15]
2) Construct a set of trees with collapsible edges, from the half-edge leaving each vertex, a tree edge must be selected without causing a cycle. This is prevented with an edge weighting system and a subsequent error checking pass.
3) Split the trees to meet the quality threshold. In the previous step the whole tree may accumulate quadric error over the quality threshold, this step splits up trees that exceed it. [12]
4) Cancel tree edges that can cause illegal mesh changes. [12] An illegal change would be the flipping of a normal vector. Fixing this is hard after the fact [12], so edges are found before contraction and discarded from the tree
5) Collapse trees to their root vertices [12] this is done to prevent topological changes to the mesh
6) Refine root vertex positions, the root vertex positions (the resultant vertices of the collapse) are refined after collapse by minimising their accumulated quadric errors using a method proposed by [15]. Repeat 1-6 until it reaches the specified vertex count. [12]

C. GPU Half Edge

GPU half edge works by splitting a mesh up into a set of regions, then simplifying each region separately in parallel and then combining their results [13] a comparatively simple algorithm by comparison to ETC.

The operations of the GPU Half Edge algorithm is broken down into 5 distinct steps. [13]

1) Build the data structure, a customised half Edge data structure [15] to work around memory allocation limits in CUDA [13], [32]
2) Calculate the edge contraction costs, the QEM for each half edge. [13].
3) Build a priority queue using a skip list (a special list type that has an average O(log n) complexity for search and insertion [33]) of these costs to choose a pair of vertices to be contracted. [13]

4) Construct a set of disjoint parts of the mesh using k-d tree space partitioning. [13] The k-d tree is used to break the mesh down into chunks for parallelisation, each kernel will work on its own region of the tree.
5) Apply the decimation process [13] During this process, one contraction per region of the k-d tree which guarantees the consistency of the neighbourhood properties among the vertices of the mesh [13] otherwise a region may rely on geometry another region optimises out. The data structures are then updated, and the mesh is decimated again until it reaches the specified vertex count.

D. Convex Terrain Generation

The type of terrain generation being used to assess the algorithms is similar to height-map generation [34] but directly displaces vertices in their local up direction around a spherical like body resembling a planet. The resultant mesh is convex.

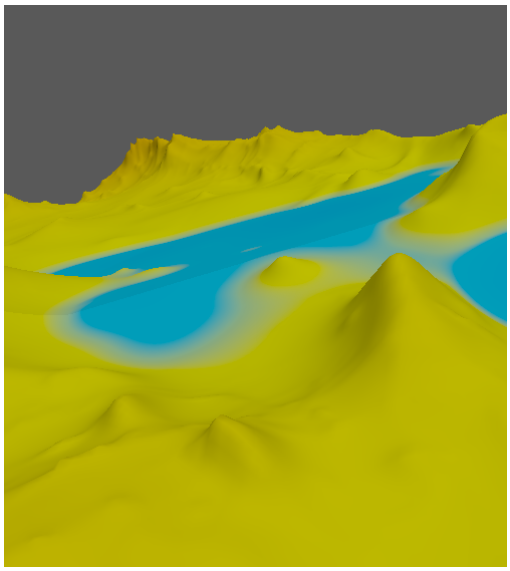


Fig. 6. Example terrain output from the computing artefact

The terrain generator uses simplex-noise [35] in multiple layers with various settings based off work by Sebastian Lague [36], [37], combined with the gradient-trick for erosion proposed by inigo quilez [38]. This generator was created by the author in a custom Vulkan [39] framework in C# with SDL3 [40]

III. Scientific Questions & Hypotheses

Both CPU and GPU simplification algorithms are designed to work on arbitrary meshes pre-modelled in software like Blender [41], or Maya [42].

As there is also a lack of research on simplification algorithms on procedural terrain geometry, this paper will compare these two algorithms using a simplex-based terrain generation algorithm which produces a convex terrain mesh.

This paper will also compare the simplification algorithms to low-sample rate terrain geometry. The algorithms will be implemented using Vulkan [39] instead of CUDA [32] as

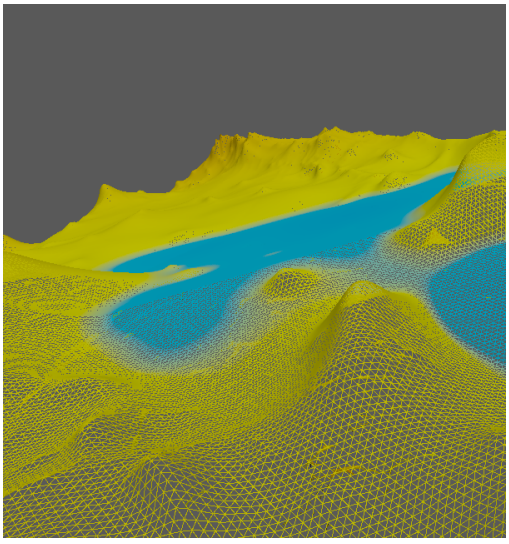


Fig. 7. Example terrain output (wireframe) from the computing artefact

the terrain generation artefact is written in a custom Vulkan application, and Vulkan is GPU agnostic where CUDA is propriety to Nvidia GPUs.

This then all leads to the research question:

From a high sample generated convex terrain mesh, do both mesh simplification algorithms A and B produce a low mean geometric deviation than a low-sample rate terrain generation pass, and if so, which simplification algorithm A or B produces the lowest mean geometric deviation for the least compute time.

Hypotheses:

- Algorithms A and B have a lower geometric deviation than a low sample rate generated mesh
- Algorithm B executes in less time than algorithm A
- Algorithm B has a lower mean geometric deviation than algorithm A
- Algorithm B has a lower mean geometric deviation per unit of execution time than algorithm A

As the purpose of this paper is to find which of the two algorithms is better, should hypotheses 2-4 be proven false, this information is still useful.

IV. Methodology

A. Computing Artefact

1) Requirements: To test the hypotheses a computing artefact that implements and can measure the performance of algorithm A and B must be created. As well as this it must be able to generate a convex hull terrain mesh and multiple resolutions and with repeatable but randomizable results.

An existing terrain generating artefact will be used, created by the author. The data recording and algorithm implementation will be added into framework.[1] The variables being measured about each algorithm are the execution time of the algorithm and the quality of the result, expressed as mean geometric deviation.

[1]Link to repository

As this experiment is to determine which algorithm is better with generated terrain meshes, multiple terrain meshes will be required to verify the null hypotheses (they are the same) is false. The existing framework can do this by randomly generating a seed which terrain generation is based off, the seed used to generate the mesh should be recorded in-case of a validation needing to be performed.

To simplify the mesh using the algorithms, they need an additional input, how much to simplify the high-resolution mesh by. This is given as a % of the original Triangle count, to maximise the resolution of the test. Each algorithm will be tested using a set of reduction levels:

Reduction of original geometry
75%
50%
25%

To compare the algorithms to a low sample rate terrain, each terrain will also be generated at a low sample rate terrain mesh which has a roughly equal polygon count to the reduction levels above.

Finally, algorithm A takes an additional input variable, a quality threshold. The lower this value is, the high quality the output mesh will be at the cost of performance. Some testing will be needed to determine a good value for this variable on a range of meshes. Alternately, this variable could be used to normalize the quality between the two algorithms and compare only the execution time.

2) Breakdown of Operations: The artefact will evaluate the algorithms through by first generating a high sample-rate terrain mesh with N triangles, 1 million for instance.

The initial sample-rate will be a broadly configurable variable. For each algorithm and for each reduction level, the artefact will run the algorithm using the high sample-rate mesh as its input to create a simplified mesh, measuring how long it takes the algorithm to create the simplified mesh.

After the simplified mesh is created, the artefact will then compare the simplified result to the high sample-rate mesh, by calculating a geometric deviation mean, with 0 meaning identical in shape to the high-resolution mesh.

Finally, the artefact will create a csv entry, recording the terrain generation seed, the starting triangle count, the reduction level, the execution time and the geometric deviation mean.

3) Development: To deliver this artefact, agile [43] development practices will be used to rapidly iterate on the application. Tasks will be kept track of using GitHub's taskboard [44]. As its iterated, the artefact will be tested using unit tests and refactored based on static code analysis [45] To ensure the geometric deviation implementation is being accurately calculated it will be compared to its source paper and source code [27], [46] To ensure Algorithm A and B are providing outputs consistent with their author's expectations, they will be tested using the same models each paper used as they are widely available.

4) Current State: One of the authors of [27] produced a C++ application [46] for measuring and visualising geometric deviation as a heat map (see figure 8) and made source code available under GNU General Public License [46].

The author has translated this program from C++ to C# and integrated it into the computing artefact.

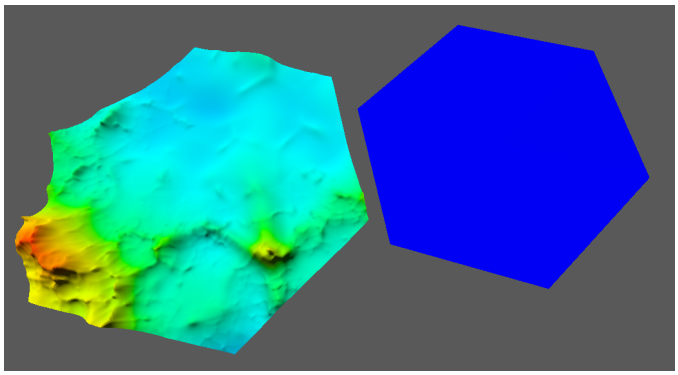


Fig. 8. Example visualisation of geometric deviation from the computing artefact. The program [46] has been ported into the computing artefact by the author.

Some preliminary tests have been run (see Appendix B) on the terrain generator comparing a high sample-rate terrain to various sample rate levels (subdivisions). Results so far show when the meshes are the same there is no deviation, and as the sample rate goes down for the left mesh the mean and max Deviations increase. All of which is inline with expectations.

B. Ethical Considerations

The ethical concerns for this project end at the usage of the algorithms being tested within the paper from an IP perspective and the IP of the libraries, frameworks and tools used to carry out the experiment and analysis.

It is acknowledged that this is a computing experiment and so consideration for sustainability (energy usage) has been made, and deemed low as the proposed experiment can be run on a desktop computer, a high-power data centre is not necessary.

The overall ethical risk is low as this project does not involve human participants, or any significant legal issues. This research cannot be weaponised.

C. Data Management and Analysis Plan

1) Sampling: **Control tests**

Each algorithm will be tested on a control generated mesh to determine its run-to-run variance in geometric deviation and execution time. If run-to-run error is high for one or both algorithms, then during the experiment, each step can be repeated and averaged together as many times as needed to ensure the results have statistical power. Additionally, Algorithm A, Embedded Tree Collapse, will be tested against itself to find an optimal value for its quality threshold. This optimal value will then be used in the comparison tests against algorithm B. Algorithm B does not need a test like this as it has no quality parameters.

Sample Size

From analysis of [12], [13] execution time, Cohen's d effect size of 9 was calculated, this puts sample size total for both algorithms at 4 for a t-test. This is far too low and speaks to

the major hardware differences between the two papers. To compensate, initially Cohen's d will be capped at 2 and beta error will be raised from 0.95 to 0.99. This makes total sample size 22, 11 for each algorithm. After data collection, it may be necessary to raise further as the hardware will be the same, or the large effect will support the hypotheses.
Due to the difference in geometric error measurement, the data sets between the papers are not comparable, making calculation of sample size less accurate. Comparing within the papers by looking at the algorithms they compared against, a large Cohen's d is produced between what they compared to. This suggests that to each other, Cohen's d will be lower, for now a value of 1 will be assumed, for a sample size of 76(38 per algorithm).
More data can be easily collected in the case Cohen's d is lower than predicted based on initial analysis.

The sample size may be increased because of the control tests. If there is a high degree of run-to-run variance between one or both algorithms, more data will be needed to average out this variance to be sure of statistical certainty of the results.

Experiment
To acquire data to answer each hypothesis, the artefact will start by generating a high sample-rate terrain mesh at a specific polygon count (100k triangles). The seed to produce this mesh will be recorded. Each algorithm will then simplify the high-resolution mesh to a range of steps 75%, 50% and 25% of the original polygon count.

The same will be done by the generation algorithm, making it regenerated the mesh at 75%, 50% and 25% of the original polygon count.

Time milliseconds it takes to simplify from the original to the reduction step will be recorded.

After a simplified mesh has been created, the result will be compared to the original to determine the mean geometric deviation of the simplified relative to the original mesh.

The mean geometric deviation of each algorithm will also be compared to running the generation algorithm at a low sample rate roughly equivalent in polygon count to the current reduction rate.

For example data collection format see Appendix A

2) Data Analysis: Each hypothesis will be tested using a T test.

```
library(readr)
dat <- read_csv("H:Stats\tests_data.csv")
view(dat)
summary(dat)
t.test(x~y)
```

x and y will refer to the mean geometric deviation or to the execution time.
Hypotheses 4 needs to have a value derived from the execution time and mean geometric deviation, this metric is execution time per mean deviation. This will be calculated by dividing the geometric deviation by the execution time.

- To prove the first null hypotheses false, both algorithms will have to show a quality change to a lower resolution terrain generation - either positive or negative.
- To prove that null hypotheses 2 and 3 are false one of the algorithms should prove to be faster to execute than the other and one of the algorithms should have a lower geometric deviation than the other.
- To prove that null hypotheses 4 is false there should be a difference between the algorithm's execution time per mean deviation.

REFERENCES

[1] M. J. DeHaemer and M. J. Zyda, "Simplification of objects rendered by polygonal approximations," vol. 15, no. 2, pp. 175–184. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0097849391900710

[2] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of triangle meshes," vol. 26, no. 2, pp. 65–70. [Online]. Available: https://dl.acm.org/doi/10.1145/142920.134010

[3] C. DeCoro and N. Tatarchuk, "Real-time mesh simplification using the GPU," in *Proceedings of the 2007 symposium on Interactive 3D graphics and games*. ACM, pp. 161–166. [Online]. Available: https://dl.acm.org/doi/10.1145/1230100.1230128

[4] Hello Games. About - no man's sky. [Online]. Available: https://www.nomanssky.com/about/

[5] Luke Plunkett. Islanders looks like the cutest lil' city-building game. [Online]. Available: https://kotaku.com/islanders-looks-like-the-cutest-lil-city-building-game-1833246254

[6] Astro Techies. Procedural generation — astroneer modding 1.0 documentation. [Online]. Available: https://astroneermodding.readthedocs.io/en/latest/guides/proceduralGeneration.html

[7] Iron Gate Studio. Valheim. [Online]. Available: https://www.valheimgame.com/#about

[8] S. Parkin, "The video game that maps the galaxy," section: annals of technology. [Online]. Available: https://www.newyorker.com/tech/annals-of-technology/the-video-game-that-maps-the-galaxy

[9] Mojang. What is minecraft? build, discover realms & more. [Online]. Available: https://www.minecraft.net/en-us/about-minecraft

[10] X. Mei, P. Decaudin, and B.-G. Hu, "Fast hydraulic erosion simulation and visualization on GPU," in *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pp. 47–56, ISSN: 1550-4085. [Online]. Available: https://ieeexplore.ieee.org/document/4392715/?arnumber=4392715

[11] Reallusion. Character creator 3 online manual - creating game base character from creation base characters. [Online]. Available: https://manual.reallusion.com/Character_Creator_3/ENU/3/Content/Character_Creator_3/3.1/05_Character/Creating_Game_Base_from_Creation_Base.htm

[12] H. Lee and M.-H. Kyung, "Parallel mesh simplification using embedded tree collapsing," vol. 32, no. 6, pp. 967–976, number: 6-8 Num Pages: 967-976 Place: Heidelberg, Netherlands Publisher: Springer Nature B.V. [Online]. Available: https://www.proquest.com/docview/2917892398/abstract/9F096AB078FB4964PQ/1

[13] M. H. Mousa and M. K. Hussein, "High-performance simplification of triangular surfaces using a GPU," vol. 16, no. 8, p. e0255832, number: 8. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0255832

[14] P. K. Agarwal and P. K. Desikan, "An efficient algorithm for terrain simplification," in *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '97. Society for Industrial and Applied Mathematics, pp. 139–147.

[15] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., pp. 209–216. [Online]. Available: https://dl.acm.org/doi/10.1145/258734.258849

[16] P. S. Heckbert and M. Garland, "Optimal triangulation and quadric-based surface simplification," vol. 14, no. 1, pp. 49–65, number: 1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925772199000309

[17] Martin Franc. Methods for polygonal mesh simplication. [Online]. Available: https://naos-be.zcu.cz/server/api/core/bitstreams/76283893-f8fe-4029-9623-db852ab58410/content

[18] J. Wu and L. Kobbelt, "Fast mesh decimation by multiple-choice techniques," p. 8.

[19] Y. Zhang, H. Wang, H. Zhou, and J. Li, "Finite triangular surface mesh simplification with geometrical feature recognition," vol. 223, pp. 2627–2636, number: C11 Num Pages: 10 Place: London, United States Publisher: SAGE PUBLICATIONS, INC. [Online]. Available: https://www.proquest.com/docview/234906835/abstract/46B768EDE96F4884PQ/1

[20] Y. P. Wang, J. W. Dang, J. Y. Yang, and S. Wang, "Research on triangular mesh simplification algorithm of virtual object model," vol. 271-272, p. 1410, ISBN: 9783037855782 Num Pages: 1410 Place: Zurich, Switzerland Publisher: Trans Tech Publications Ltd. [Online]. Available: https://www.proquest.com/docview/1442631574/abstract/CF3716F398F94964PQ/1

[21] J.-L. Peyrot, F. Payan, and M. Antonini, "Aliasing-free simplification of surface meshes," in *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 4677–4681, ISSN: 2381-8549. [Online]. Available: https://ieeexplore.ieee.org/document/7025948/?arnumber=7025948

[22] D. Salinas, F. Lafarge, and P. Alliez, "Structure-aware mesh decimation," vol. 34, no. 6, pp. 211–227, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12531. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12531

[23] A. Papageorgiou and N. Platis, "Triangular mesh simplification on the GPU," vol. 31, no. 2, pp. 235–244, number: 2 Num Pages: 235-244 Place: Heidelberg, Netherlands Publisher: Springer Nature B.V. [Online]. Available: https://www.proquest.com/docview/2918055268/abstract/55C4489877454FB4PQ/1

[24] N. Koh, W. Zhang, J. Zheng, and Y. Cai, "GPU-based multiple-choice scheme for mesh simplification," in *Proceedings of Computer Graphics International 2018*. ACM, pp. 195–200. [Online]. Available: https://dl.acm.org/doi/10.1145/3208159.3208195

[25] Stiaan Uyttersprot, "Mesh compression and procedural content generation: Finding common ground for an improved 3d representation," Masters Disseration. [Online]. Available: https://libstore.ugent.be/fulltxt/RUG01/002/495/091/RUG01-002495091_2018_0001_AC.pdf

[26] World Machine Software, LLC. World machine: The leading 3d terrain generation software. [Online]. Available: https://www.world-machine.com/

[27] M. Roy, S. Foufou, and F. Truchetet, "Generic attribute deviation metric for assessing mesh simplification algorithm quality," in *Proceedings. International Conference on Image Processing*, ser. Proceedings. International Conference on Image Processing. [Online]. Available: https://hal.science/hal-01938350

[28] O. Matias van Kaick and H. Pedrini, "A comparative evaluation of metrics for fast mesh simplification," vol. 25, no. 2, pp. 197–210, number: 2 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2006.00935.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2006.00935.x

[29] M. Guthe, P. Borodin, and R. Klein, "Fast and accurate hausdorff distance calculation between meshes," accepted: 2013-03-06T13:41:38Z Publisher: Václav Skala - UNION Agency. [Online]. Available: http://dspace5.zcu.cz/handle/11025/1446

[30] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," vol. 96, no. 5, pp. 879–899, conference Name: Proceedings of the IEEE. [Online]. Available: https://ieeexplore.ieee.org/document/4490127/?arnumber=4490127

[31] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an open-source mesh processing tool," artwork Size: 8 pages ISBN: 9783905673685 Pages: 8 pages Publication Title: Eurographics Italian Chapter Conference. [Online]. Available: http://diglib.eg.org/handle/10.2312/LocalChapterEvents.ItalChap.ItalianChapConf2008.129-136

[32] Nvidia. CUDA toolkit - free tools and training. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[33] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," vol. 33, no. 6, pp. 668–676. [Online]. Available: https://dl.acm.org/doi/10.1145/78973.78977

[34] M. Frade, F. F. De Vega, and C. Cotta, "Genetic terrain programming an aesthetic approach to terrain generation," in *1st Annual International Conferences on Computer Games, Multimedia and Allied Technology (CGAT 2008)*. Global Science & Technology Forum (GSTF). [Online]. Available: https://www.dropbox.com/s/92io6p2p49y0rsd/CGAT2008P43.pdf?dl=0

[35] K. Perlin, "Chapter 2: Noise hardware." [Online]. Available: https://userpages.cs.umbc.edu/olano/s2002c36/ch02.pdf

[36] Sebastian Lague. [unity] procedural planets (e07: ocean depth). [Online]. Available: https://www.youtube.com/watch?v=OULxvDLojic

[37] S. Lague. SebLague/procedural-planets. Original-date: 2018-08-16T15:20:06Z. [Online]. Available: https://github.com/SebLague/Procedural-Planets

[38] I. Quilez. Inigo quilez. [Online]. Available: https://iquilezles.org/articles/morenoise/

[39] Vulkan. Home | vulkan | cross platform 3d graphics. [Online]. Available: https://www.vulkan.org/

[40] SDL Community. SDL3/FrontPage. [Online]. Available: https://wiki.libsdl.org/SDL3/FrontPage

[41] Blender Foundation. blender.org - home of the blender project - free and open 3d creation software. [Online]. Available: https://www.blender.org/

[42] Autodesk. Autodesk maya | get prices & buy official maya 2025 | autodesk. [Online]. Available: https://www.autodesk.com/uk/products/maya/overview

[43] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. [Online]. Available: https://agilemanifesto.org/

[44] GitHub. About projects. [Online]. Available: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects

[45] gewarren. Code analysis in .NET. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/overview

[46] Michaël Roy. MeshDev - mesh comparison software. [Online]. Available: https://meshdev.sourceforge.net/

APPENDIX A
EXAMPLE DATA FOR A SINGLE TEST

Seed	Algorithm	Triangle Count	Reduction Level	Runtime	Mean Geometric Deviation
1349123	Terrain Gen	15m	75%	300ms	0.0001
1349123	Terrain Gen	15m	50%	30ms	0.001
1349123	Terrain Gen	15m	25%	3ms	0.01
1349123	Algorithm A	15m	75%	600ms	0.0001
1349123	Algorithm A	15m	50%	60ms	0.001
1349123	Algorithm A	15m	25%	6ms	0.01
1349123	Algorithm B	15m	75%	1000ms	0.0001
1349123	Algorithm B	15m	50%	100ms	0.001
1349123	Algorithm B	15m	25%	10ms	0.01

APPENDIX B
PRELIMINARY TESTING OF GEOMETRIC DEVIATION

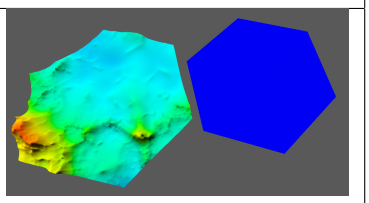
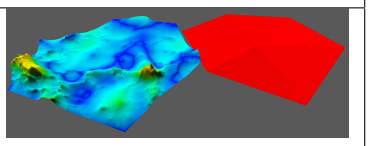
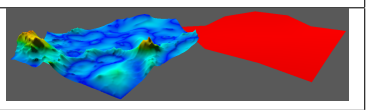
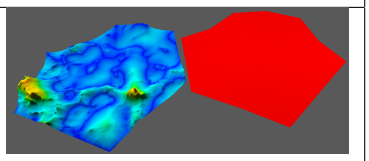
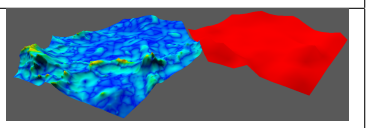
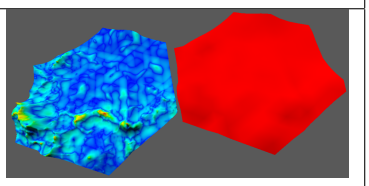
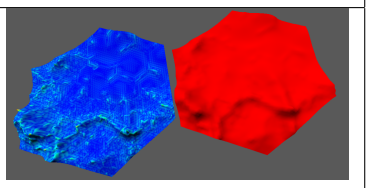
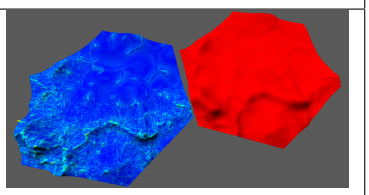
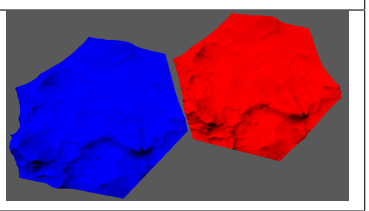
Left Subdivisions	Right Subdivisions	Mean Deviation	Max Deviation	Heatmap
7	0 (flat)	0.0351	0.1022	
7	0	0.0071	0.0415	
7	1	0.0042	0.0401	
7	2	0.0042	0.0401	
7	3	0.0010	0.0105	
7	4	0.0010	0.0105	
7	5	0.0001	0.0058	
7	6	0.0001	0.0058	
7	7	0	0	

Fig. 9. Mean & max geometric deviation results with heatmap visualisations of geometric deviation comparing high subdivision count terrain generation to the same terrain generator at various subdivision steps (0-7). The first result uses the raw shape with no subdivision or terrain generation.
These results also unveiled a bug with the subdivision algorithm that cause odd subdivision count to round up to the nearest even number.